On the FPGA Implementation and Performance Analysis of a Digital Carrier Synchronizer

Sayed Hafizur Rahman

A Thesis

in

The Department

of

Electrical and Computer Engineering

Presented in Partial Fulfillment of the Requirements

for the Degree of Masters of Applied Science at

Concordia University

Montréal, Québec, Canada

November 2006

Your file Votre référence
ISBN: 978-0-494-28925-9
Our file Notre référence
ISBN: 978-0-494-28925-9

Abstract

On the FPGA Implementation and Performance Analysis of a Digital Carrier Synchronizer

Sayed Hafizur Rahman, M.A.Sc

Concordia University, 2006

The evolutionary growth of digital communication has an acute impact on the digital integrated circuit (IC) design industry. Nowadays instead of ASICs (Application Specific Integrated Circuits), Field programmable gate arrays (FPGAs) are often employed to implement digital communication systems due to the speed, performance, reliability and flexibility. Digital communication systems such as modulation-demodulation and M-PSK require the use of carrier synchronization in phase and frequency. This work addresses the FPGA implementation and analysis of a Digital Carrier Synchronizer (DCS), which is a phase-locked loop (PLL), realized using digital circuits. This novel methodology highlights implementation promises towards some of the critical issues associated with the design of its analog counterpart, usually known as PLL. The principle function of this DCS is heavily dependent on the Numerically Controlled Oscillator (NCO) and the Loop Filter (LF). There are various methods to

implement NCOs and LFs that are used in the architectural model of DCS. This research work examines the performance of two different NCOs and LFs realization in DCS for modem (modulator-demodulator) application using FPGA based design solutions. The methods presented are Look up Table (LUT) and Xilinx ROM based NCO in one hand, and 1st order and 2nd order based LF in the other hand. Each has its own merits and de-merits. A DCS mathematical model has been developed in order to analyze the stability of the design. Furthermore, the performance of this two implementations based on three performance metrics i.e. stability, locking-time and tracking range has been studied. From the analysis, Xilinx ROM based NCO with 2nd order LF performs better and are more suited for modem's DCS.

Acknowledgements

First and foremost, my wholehearted thanks and admire are due to the Holy One, Allah, Who has always been with me through out my entire struggle in life wiping out all my fears, for oblation His countless blessings.

I would like to express my sincere gratitude and appreciation to my supervisor Dr. Otmane Ait Mohamed for providing me the opportunity to work in this challenging but exciting field and to be part of his research group, for his expert guidance, and for his support and encouragement throughout my research and thesis process.

I would like to specially thank Asif Iqbal Ahmed of Hardware Verification Group, Concordia University, for his invaluable guidance. Without his insight and steering, needless to say, this project would have never got completed. Also, I would like to curiously thank Dr. Youcef Fouzar, Zarlink Semiconductor, Ottawa, Canada, for valuable discussion of my project.

To all my fellow researchers in Hardware Verification Group (HVG) at Concordia University, thank you for encouragement, thoughtful discussion, and productive feedback. I wish to express my sincere and heartfelt thanks to Kamran Hussain, Amer Samara, Abu Nasser Mohammed Abdullah and Haja Moinudeen who have helped me during the course of my research work. I would specially like to thank to my eldest brother, Mujibur Rahman in Texas Instrument, USA, who helped me out whenever I was in difficulties.

Last but not least I would like to thank my eldest brothers, father and the rest of my family members in Bangladesh for their constant moral support and encouragement which were invaluable in completing this thesis.

To

My Father,
Sayed Mostafizur Rahman

And

My Mother,
Late – Firoza Begum

TABLE *of* CONTENETS

List *of* Figures ..... **xi**

List *of* Tables ..... **xiii**

List *of* Acronyms ..... **xiv**

Chapter 1 Introduction **1**

1.1 Thesis Contributions *4*

1.2 Thesis organization *6*

Chapter 2 Related Work and Preliminaries **7**

2.1 Related Work *7*

2.2 Preliminaries *9*

2.2.1 Synthesis Process 9

2.2.1.1 Synthesizable vs. Non-Synthesizable RTL 11

2.2.1.2 Synthesis for FPGA 13

2.2.1.3 Xilinx Integrated Software Environment (ISE) 13

2.2.2 Functional Verification Using Simulation 16

2.2.2.1 *What is Functional Verification?* 16

2.2.2.2 *The Importance of verification* 17

2.2.2.3 *What is testbench?* 19

2.3 Summary *20*

Chapter 3 Conventional Analog Phase Locked (APLL) **21**

3.1 Introduction *21*

3.2 Phase Detector (PD) *22*

3.3 Loop Filter (LF) *24*

3.4 Voltage Controlled Oscillator (VCO) *26*

3.5 Summary *30*

Chapter 4 Mathematical Model of a Digital Carrier Synchronizer (DCS) in the Discrete Time Domain (Z-Domain) **31**

4.1 Introduction *31*

4.1.1 Mapping the Poles of a second-order system from S-domain to Z domain 32
4.2 Mathematical model of DCS Architecture 33
4.3 Analysis of the DCS 34
4.3.1 Transfer function of Loop Filter in the Z-domain 34
4.3.2 Transfer function of NCO in the Z-domain 36
4.3.3 Phase Error Response of the DCS 39
4.3.4 Stability analysis of the DCS 41
4.3.5 Tracking Range of the DCS 43
4.4 Summary 43
Chapter 5 Design and Datapath Implementation of Digital Carrier Synchronizer (DCS) 44
5.1 Introduction 44
5.2 Phase Detector 45
5.2.1 Data path of the Phase Detector 46
5.3 First Order Digital Loop Filter (LF) 48
5.3.1 Data path of the Loop Filter 49
5.4 Numerically Controlled Oscillator (NCO) 50
5.4.1 Datapath of LooK Up Table (LUT) Based NCO 51
5.5 Digital Loop Filter 53
5.5.1 Designing a IIR Low pass Filter 54
5.5.2 Designing a FIR Low pass filter 55
5.6 Xilinx System Generator 57
5.6.1 Strategy 58
5.6.1.1 Selecting an FPGA Board 58
5.6.1.2 Selecting a Digital Filter Design Method 58
5.6.1.3 Kaiser Window Method 59
5.7 CORDIC Based NCO 62
5.8 Xilinx ROM Based NCO 65
5.9 Summary 67

Chapter 6 Synthesis and Emulation **68**

6.1 Space Exploration via Synthesis Process *68*

6.1.1 Comparison of Different NCOs 68

6.1.2 DCS Synthesis Using Various Configuration of NCO and LF 71

6.2 Emulation *71*

6.2.1 Introduction 71

6.2.2 The Emulation Process 73

6.2.3 Synthesized Emulation Environment setup 73

6.2.4 Design Implementation 75

6.2.4.1 Internal Reset Generator through an FSM 76

6.2.4.2 Linear Feedback Shift Register (LFSR) 77

6.2.4.3 Digital Magnitude Comparator 78

6.2.4.4 ADC (Analog to Digital Converter) 81

6.2.4.5 Clock Divider (clk_div) 82

6.2.4.6 Display 83

6.3 Summary *83*

Chapter 7 Simulation Results and Performance Evaluation of Different DCS Implementations **84**

7.1 Introduction *84*

7.2 Simulation Environment for DCS *85*

7.2.1 Instantiations 86

7.2.2 Initial and Always Blocks 87

7.2.3 Printing Using $display During Simulation 88

7.3 Simulation Results and Discussion *89*

7.4 Tracking Range and Locking Time Analysis *93*

7.5 Summary *97*

Chapter 8 Conclusion and Future Work **98**

References **101**

List *of* Figures

Figure 1. Digital Carrier Synchronization (DCS) path 2

Figure 2. Logic Synthesis Flow from RTL to Gates 10

Figure 3. Screen shot of Xilinx Project Navigator, from the ISE Software Suite..... 14

Figure 4. A Functional Verification Path 16

Figure 5. Verification Dominates Design 18

Figure 6. Block Diagram of a Testbench 20

Figure 7. Analog Phase Locked Loop 21

Figure 8. Phase Detector as a Multiplier..... 23

Figure 9. Filtering the Phase Detector Output signal to remove 25

Figure 10. The Circuit Implementation of the first order passive Loop Filter..... 25

Figure 11. Block Diagram of Voltage Controlled Oscillator..... 27

Figure 12. Mathematical model of DCS in discrete time domain (Z-domain)..... 34

Figure 13. Block diagram of Loop Filter in Z-domain 35

Figure 14. Block diagram of NCO in Z-domain 36

Figure 15. Pole-Zero Plot for a stable DCS 42

Figure 16. Digital Carrier Synchronization Path on the FPGA platform 44

Figure 17. Datapath of the Digital Carrier Synchronization..... 45

Figure 18. Datapath of Phase Detector..... 46

Figure 19. Phase Detection State machine 48

Figure 20. Datapath of 1st Order Loop Filter..... 49

Figure 21. Look Up Table (LUT) Based NCO 51

Figure 22. Second Order Low Pass IIR Filter 54

Figure 23. Second Order Low Pass FIR Filter 56

Figure 24. CORDIC Based NCO..... 62

Figure 25. Xilinx ROM 1024X1 65
Figure 26. Xilinx ROM Based NCO with Modulo N Counter 65
Figure 27. Emulation Environment for DCS 72
Figure 28. FSM for internal Reset signal generator 77
Figure 29. An 8-bit Pseudo Random Generator 78
Figure 30. 8-Bit Digital Magnitude Comparator 79
Figure 31. ADC with a Magnitude Comparator 80
Figure 32. Datapath of Analog to Digital Converter 81
Figure 33. Simulation Environment for DCS 85
Figure 34. Stimulus Module Instantiation 86
Figure 35. DCS Using LUT based NCO and 1st Order LF (When Fs = 72 MHz) 89
Figure 36. DCS Using LUT based NCO and 1st Order LF (When Fs = 75MHz) 89
Figure 37. DCS Using Xilinx ROM based NCO and 1st Order LF (When Fs = 73 MHz) 90
Figure 38. DCS Using Xilinx ROM based NCO and 1st Order LF (When Fs =77MHz) .. 90
Figure 39. DCS Using LUT based NCO and 2nd Order LF (When Fs =85 MHz) 91
Figure 40. DCS Using LUT based NCO and 2nd Order LF (When Fs = 88 MHz) 92
Figure 41. DCS Using Xilinx ROM based NCO and 2nd Order LF(When Fs = 95 MHz) 92
Figure 42. DCS Using Xilinx ROM based NCO and 2nd Order LF(when Fs = 105 MHz)93
Figure 43. Tracking Frequency vs. Lock Time of DCS using LUT Based NCO and 1st Order LF 94
Figure 44. Tracking Frequency vs. Lock Time of DCS using Xilinx ROM based NCO and 1st Order LF 95
Figure 45. Tracking Vs. Lock Time of DCS using LUT based NCO and 2nd Order LF... 96
Figure 46. Tracking Frequency Vs. Lock Time of DCS using Xilinx ROM based NCO and 2nd Order LF 96

List *of* Tables

Table 1. Physical resources Occupation on FPGA board for FIR and IIR Filter..... 59

Table 2. Angle Value & Shift Sequences of CORDIC Based NCO 64

Table 3. Xilinx ROM1024X8 Initialization (When Fs = 50 MHz)..... 66

Table 4. Performance Evaluation Using Xilinx..... 69

Table 5. Performance Evaluation Using Synopsys 69

Table 6. Synthesis Report of Different DCS Implementations..... 71

Table 7. Summarized Emulation Results of Different DCS Implementations 83

Table 8. Summarized Analysis of Different DCS Implementations 97

List *of* Acronyms

ADC	Analog to Digital Converter
ASIC	Application Specific Integrated Circuit
APLL	Analog Phase Locked Loop
CORDIC	Coodinate Rotation Digital Computer
DAC	Digital to Analog Converter
DCS	Digital Carrier Synchronizer
DPLL	Digital Phase Locked Loop
FPGA	Filed Programmable Gate Array
FIR	Finite Impulse Response
FSM	Finite State Machine
IIR	Infinite Impulse Response
LUT	Look Up Table
LF	Loop Filter
LPF	Low Pass Filter
NCO	Numerically Controlled Oscillator
PD	Phase Detector
PM	Phase Modulation
SOC	System-On-Chip
UCF	User Constraint File
VCO	Voltage Controlled Oscillator

Chapter 1

Introduction

The continuous progress in modern digital communication systems, such as wireless, telecom, and datacom require a stable periodic signal to offer timing solutions. This stability provides the basis for synchronizing, aligning the sampling clock, restraining the clock skew or synthesizing frequencies. Phase locking, studied for more than half a century, is the principal technique to provide timing solutions. A list of tasks recognized by phase-locked loops (PLL) includes carrier synchronization, carrier recovery, clock recovery, phase modulation, phase/frequency demodulation, frequency synthesis, duty cycle correction, and jitter reduction [1]. Carrier Synchronization is an important part in coherent communication systems, especially for those employing a high bandwidth efficiency modulation schemes such as modem. Conventionally, a feedback loop called the phase-locked loop (PLL) is used to implement the carrier synchronization [2]. Digital carrier synchronizers (DCS) (Shown in Fig. 1) follow the analog phase locked loop techniques to synchronize any given signal [3]. The input signal of the DCS is sampled by the analog to digital converter (ADC) [4], which is not synchronous with the signal of the receiver's digital to analog converter (DAC) [3] [5]. There is a frequency or phase offset that exists between

the transmitter (ADC) and the receiver (DAC). In order to remove this offset a digital carrier synchronizer (DCS) is required.

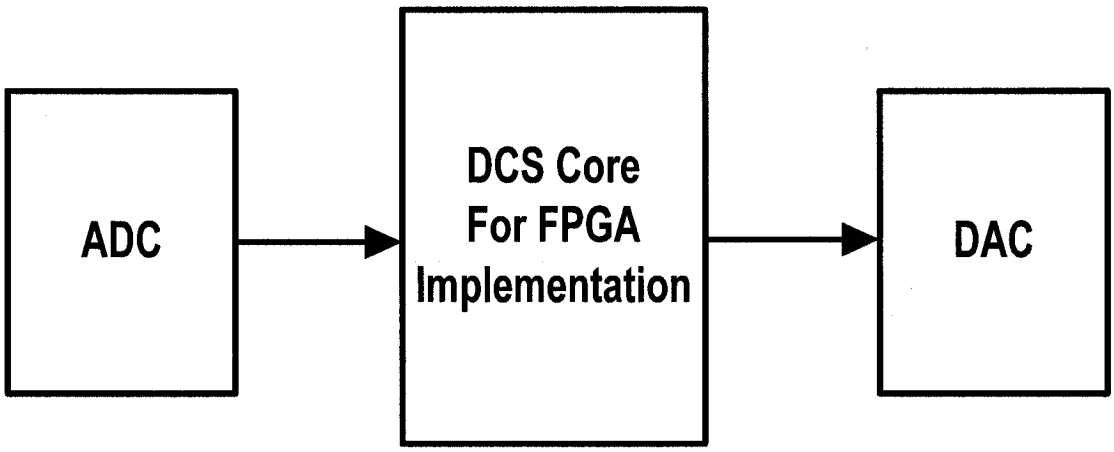

Figure 1. Digital Carrier Synchronization (DCS) path

In this research work, the carrier synchronization is considered in digital domain in the context of digital modulation application, i.e. modem, which are designed mainly using a digital approach, because of the flexibility and high performance of digital systems. The carrier synchronization in digital modems is achieved by phase-locked loops, and as the DPLL is purely digital, it can be used in these systems. In modem applications, the locking time and tracking frequency are of extreme importance. Whenever the phase-locked loop loses lock in a modem, it means that hundreds of bits of data will be lost before the modem can regain synchronization [6]. The carrier frequency of our target application, modem, is 2.048 MHz. In this research work, two distinct realizations of DCS are investigated. The first one is the LUT based NCO with 1^{st} and 2^{nd} order LF and the second one is the Xilinx ROM based NCO with 1^{st} and 2^{nd} order LF. For each design we try to obtain the optimal locking time, and a wider tracking frequency

range, when changing the frequencies from 35 MHz to 110 MHz. Furthermore, the stability of our design was analyzed using MATLAB and showed that it's stable under certain constraints, which become then our design requirements. Our implementation targets a Field Programmable Gate Arrays (FPGAs).

As the technology of a multi-million gate in new devices is advancing very fast, both FPGAs and Application Specific Integrated Circuits (ASICs) are competitively demonstrating their capabilities in very large and high-speed applications. Consecutively, choosing the right technology to implement a given design is becoming the key question for several applications. In one hand, with ASICs one can implement multi-million gates in a small area of silicon using a library of reusable hardware and software blocks as Intellectual Property (IP) cores [7]. On the other hand, FPGAs have also satisfied wonderfully the requirements of fulfilment large complex designs in their today's multi-million gate ranges. Large variety of high performance IP cores (Microprocessors, Microcontrollers, Intellectual functional logics etc.) as well as high speed memories are much more accessible in today's FPGAs [8]. These features facilitate the implementation of a large complex system designs in FPGAs. Since the FPGAs are programmable, this indeed lowers the cost of any required changes or modifications in the design for future, and considering the importance of shorter Time To Market (TTM) in industry, which is a great benefit. For our DCS implementation we chose ***Xilinx Virtex-II Pro*** FPGA family which is based on IP cores and customized modules. It has multi-gigabit transceivers and PowerPC CPU blocks. It empowers complete solutions for telecommunication,

wireless, networking, video, and DSP applications. Virtex-II Pro architectures are optimized for high performance designs in a wide range of densities. Combining a wide variety of flexible features and IP cores, the Virtex-II Pro family enhances programmable logic design capabilities and it is a powerful alternative to mask-programmed gate arrays [9]. Virtex-II Pro has some highly advanced features which include high performance Digital Clock Manager (DCM), and large storage.

1.1 Thesis Contributions

The main contributions of the thesis are as follows:

- A mathematical model of our DCS implementation was developed in order to verify the stability of the model. The mathematical model was derived by the z-transform techniques. Initially the transfer functions of each component of the DCS model (i.e. phase detector, loop filter and NCO) were derived in the Z-domain. Subsequently the transfer function of our full DCS model was developed using negative feedback loop criteria. Finally the pole-zero condition was investigated using MatLab in order to verify the stable condition of our DCS architecture. The pole-zero plots also presented which are depicting the location of the poles and zeroes in the unit circle. Details about this development are provided in Chapter 4.

- Our initial focus was started of on the improvement of the NCO. Provided that a precomposed datapath was consisted of an NCO implemented in Look Up Table based method. The NCO using CORDIC based method was

reimplemented and afterwards used Xilinx ROM based method. Details about this development are presented in Chapter 5.

- After that our goal was to enhance the loop filter configuration as it played a major role in the feed forward gain of DCS along with NCO. Originally different NCOs were integrated with first order loop filter and checked the performance. Although synthesis wise it was the perfect candidate our goal lied to implement an NCO for faster locking time and wider tracking range. Therefore a second order loop filter was opted in order to accomplish the tracking range and locking criteria. A detail explanation is described in Chapter 5.

- Our developed emulation environment was portable and was two fold: First we wanted to validate the sanity of DCS core and second, we wanted to reuse the same emulation environment for different DCS configurations.

- Our initial examination was involved with the locking time and tracking frequency range of DCS model using first order loop filter and LUT based NCO. Subsequently, a LUT based NCO was replaced with the Xilinx ROM based NCO and observed the locking time and tracking frequency range. At last, the first order loop filter was substituted with the second order loop in the DCS datapath in order to examine the locking time and tracking frequency range.

1.2 Thesis organization

This thesis work illustrates the different DCS implementations using different order of loop filters and various NCOs. The highlight of our research work is to present a novel idea of DCS using Xilinx ROM based NCO in order to carry out the better performance metrics in the context of locking time and tracking frequency range for modem applications. The rest of this thesis is organized as follows: Chapter 2 provides a detailed explanation of the synthesis procedure and the functional verification using simulation. Chapter 3 gives a brief overview of the conventional analog phase locked loop. Chapter 4 explains a mathematical model of DCS architecture with some analysis. Chapter 5 discusses the detail datapath design and implementation of different DCSs with some modifications. Chapter 6 elaborates the emulation environment for FPGAs and Chapter 7 is devoted to simulation results and discussions. Finally Chapter 8 concludes this research work.

Chapter 2

Related Work and Preliminaries

2.1 Related Work

An assortment of NCO implementation schemes, using both LUT and CORDIC algorithms, can be found in Kadam *et al.* [10]. The work revealed that CORDIC based NCO requires less space (hardware) than the LUT based implementation. Our work leads to that Xilinx ROM based NCO uses less hardware than others NCO. However, the major difference is rooted in the performance of NCO block in the framework of DCS in our research. However, the major difference is rooted in the performance of NCO block in the framework of DCS implementation in our research while in Kadam *et al.* [10]; their investigation tackles single modular implementation of NCO.

Ray *et al.* [11] is another noteworthy research that studied the use of CORDIC architecture in implementing commonly used functions into specific FPGAs. They mainly focused on applicable Digital Signal Processing applications. Once again, the difference with our research to the mentioned work is in the context of DCS.

Liang Yi & others [12] analyzed a direct digital frequency synthesizer (DDS) which is also called NCO. Their work focused only on the size of ROM lookup table and the precision of sine wave. In our work, we focus on the different performance metrics of NCO for speeding up the DCS.

A notable work to mention is the research performed by Khalid et al. [13]. They mainly focused on designing the emulator for quantum algorithms. Their work did not mention about the emulation environment nor did they mention about the complexity of the emulation environment. The principal contribution of our research to the above mentioned work is in the context of designing an emulation environment for DCS architecture. Our emulation environment provides the flexibility of integrating any data synchronizer. With a Xilinx soft IP (e.g. ADC) core, an internal reset signal generator, LFSR, Comparator and a clock divider, ours was truly a well fitting emulation environment.

Kyung-soo Oh et al. [14] as well developed an emulator environment for functional verification of a multi-media processor. They needed customized board control functions to control the specific FPGA board but on the contrary our environment can target on any Xilinx family of FPGA without any control circuitry. Last but not least we would like to cite the work done by P. Civera et al. [15]. Their work presented FPGA based circuit emulation for performing fault-injection. The key difference between this research to theirs is the application area. Our work focused more on verification of DCS circuits used on MODEMS, whereas their prescribed work completed the flow for Design-For-Testability (DFT).

In the following sections some preliminaries will be discussed which are required for our research work.

2.2 Preliminaries

2.2.1 Synthesis Process

Synthesis is a process, which enables the conversion of a behavioral-level model, in the form of RTL coding, to a gate-level model. It is important to note that the synthesis process will not always implement the same section of code in the same manner, since the tools employed for the task are powerful and have the capacity to optimize the design; attempting to provide an optimal solution in terms of timing, area, and power. In order for a design to be optimized, a compromise must be reached between the minimization of area, power, and timing requirements. The best compromise for a particular design depends on the requirements of the system that the design is to be used in. In order for the synthesis engineer to have some control over the optimization process, *constraints* can be set to define the maximum acceptable area and power, and to define the operating frequency of the module. The synthesis tool will work, iteratively, to meet these requirements; however, if they are too optimistic then timing violations may result, requiring the design of the module to be addressed, or the constraints relaxed.

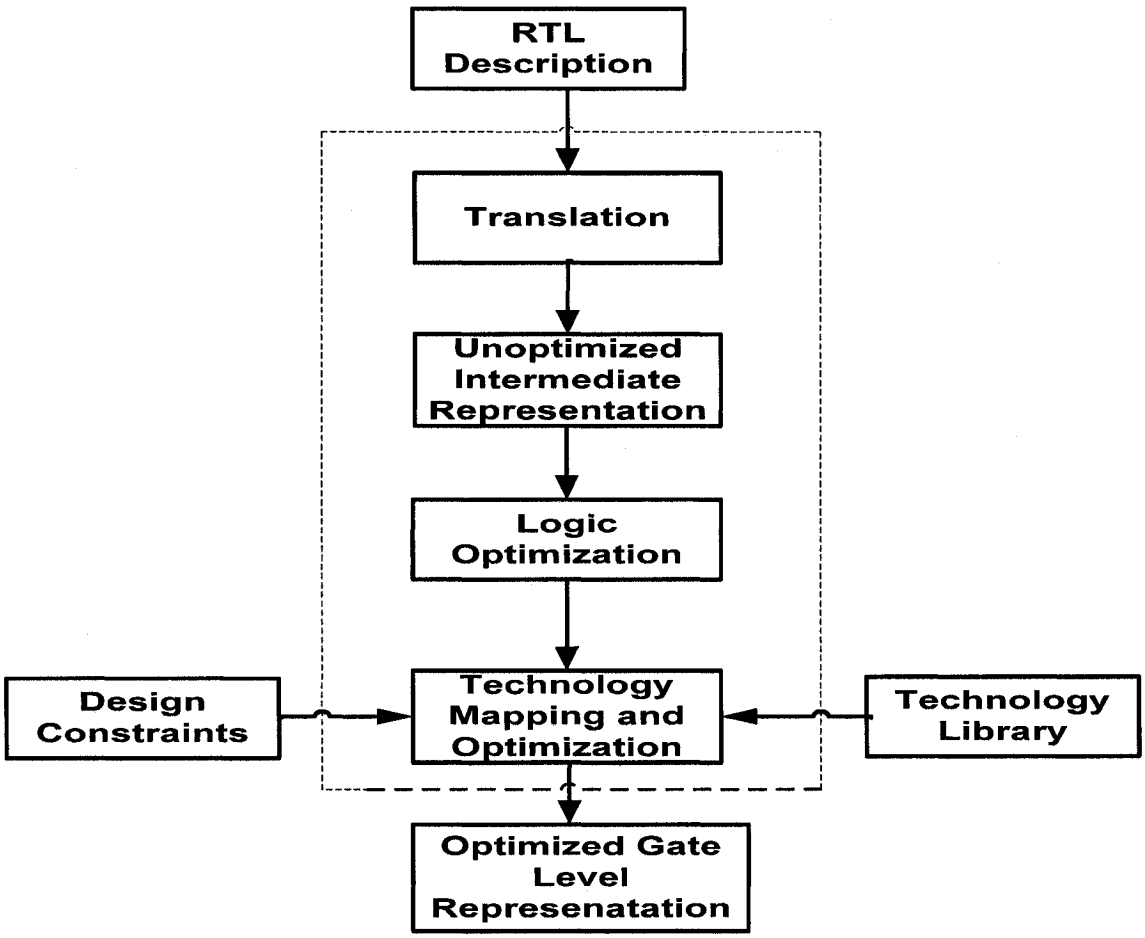

Figure 2. Logic Synthesis Flow from RTL to Gates [16]

Managing a complex datapath requires that the synthesis possess a high degree of automation. The synthesis procedure can be automated so that it is only necessary to run a single script (i.e. *dc_shell)*; nevertheless, it is useful to have a good understanding of this process throughout the development of the design. Although a variety of synthesis tools are available, to address our research need we have employed *Xilinx ISE* and *Synopsys Design Compiler* [9]. The main characteristics of these tools are they work on similar principles and involve similar stages during synthesis. Figure 2, extracted from [16], provides a good

illustration of the key stages of the synthesis process. In Figure 2, **RTL Description** represents behavioral-level Verilog coding which describing the functionality of the design. **Translation** is a process, which converts the RTL coding at the input into a representation more convenient for the tool; 'unnecessary' code such as comments will be removed at this stage. **Unoptimized Intermediate Representation** converts the output of the Translation process to a form based on the structure of the design, which is incomprehensible to the user. **Logic Optimization** employs various techniques to remove redundant logic, providing an optimized representation of the initial RTL Description, a very significant stage before the physical information and constraints are considered. **Technology Mapping and Optimization** maps cells in the gate-level representation to cells from a specified **Technology Library** and then iteratively optimizes the design in an attempt to meet the specified **Constraints**. An **Optimized Gate-Level Representation** of the design is the output from the synthesis process, along with reports on timing, area, and power of the design. A number of alterations are required in the synthesis scripts in order to define design characteristics, such as the interface, the clock periods, the top-level module name and the *Technology Library*.

2.2.1.1 Synthesizable vs. Non-Synthesizable RTL

There are three levels of abstraction that may be used to represent the design; Behavioral, RTL (Register Transfer Level) and Structural. The behavioral code is at a higher level of abstraction. It is used primarily for translating the architectural specification, to a code that can be simulated. Behavioral coding is initially

performed to explore the authenticity and feasibility of the chosen implementation for the design. Conversely, the RTL coding actually describes and infers the structural components and their connections. This type of coding is used to describe the functionality of the design and synthesizable to produce the structural netlist, which uses the leaf cells of a library [17].

If the modules in a design contain only synthesizable statements, software can be used to transform or synthesize the design into a netlist that describes the basic components and connections to be implemented in hardware. The netlist may then be transformed into, for example, a form describing the standard cells of an integrated circuit (e.g. an ASIC) or a bitstream for a programmable logic device (e.g. FPGA). For example the following snapshot of the multiplexer code is synthesizable:

```
module mux (a, b, sel, y)
 input a, b, sel ;
 output y ;
  assign  y = sel ? a : b;
endmodule
```

The following code signifies that when sel = true, y will get the value of a, otherwise b. This characterizes the behavior of a multiplexer. On the contrary, there is also non-synthesizable verilog RTL coding style exist. For instance, let's examine the following lines of code,

```
initial begin   // beginning of the simulation
```

```
  clock = 0;
  clear_n = 1;
  cycle_count = 0;
end
always # 100 clock = ~clock; // clock period of 100 time unit
```

The above code segment is generally adopted to generate the clock in the testbench but in reality the synthesis tool will not allow to synthesize the code segment. The main reason for this is in real digital circuit we can not assume an initial value without a reset.

2.2.1.2 Synthesis for FPGA

The *Synthesis for FPGA* phase of the project includes similar stages to those involved in standard synthesis procedure, but has additional stages to convert the design to a format, which can be successfully programmed onto the FPGA.

The following section considers the use of the Xilinx Integrated Software Environment (ISE) employed in the synthesis for FPGA of the different NCOs, LFs and Digital Carrier Synchronization (DCS). Issues specific to the synthesis of the module are also addressed.

2.2.1.3 Xilinx Integrated Software Environment (ISE)

The *Xilinx Integrated Software Environment (ISE)* is a tool suite developed for the synthesis and conversion process. *Project Navigator* is a tool from this suite

which enables a design flow to be set up to automate the control of the other software tools in the suite [18].

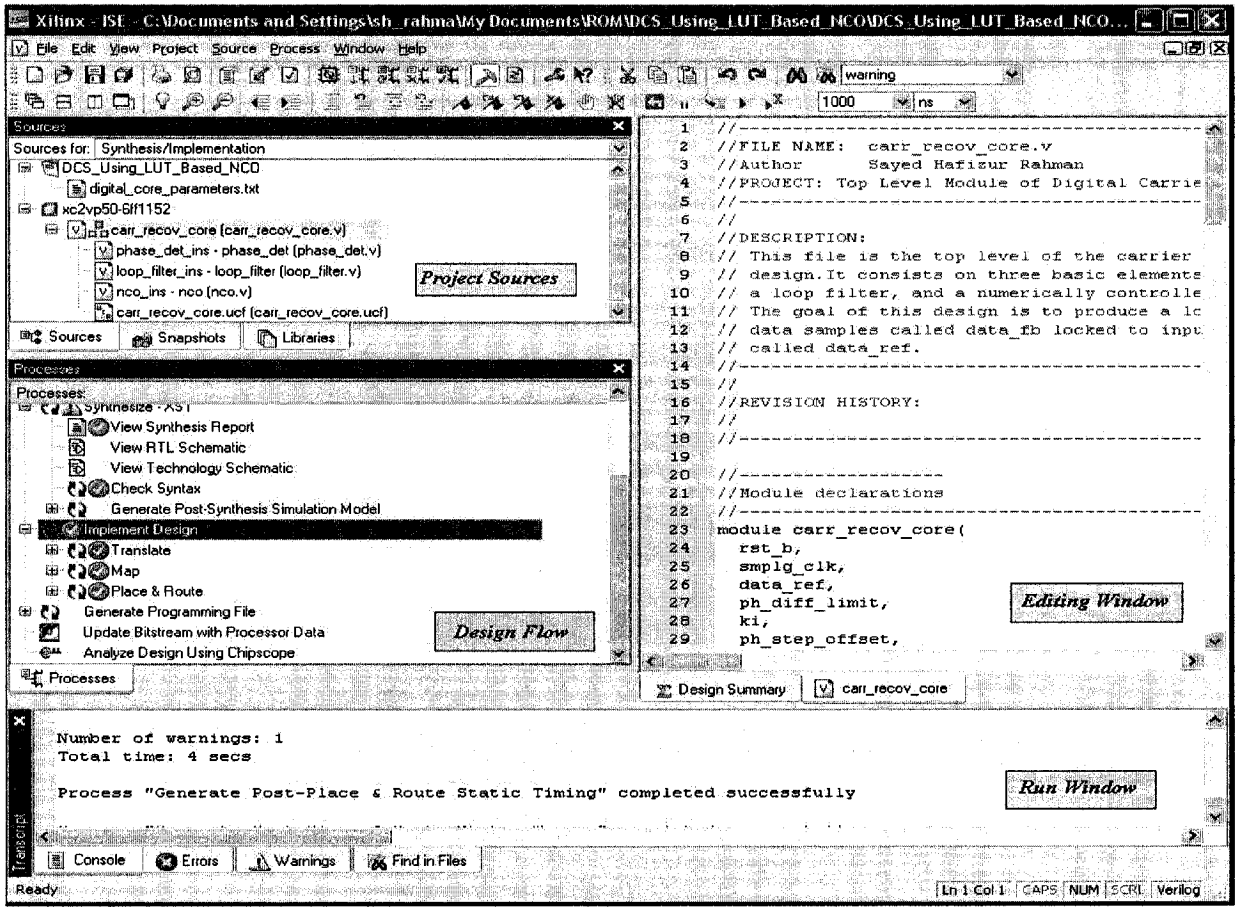

Figure 3. Screen shot of Xilinx Project Navigator, from the ISE Software Suite.

The user has control over how the design progresses through the flow, and can easily view reports from different stages of the synthesis flow. Figure 3 shows a labelled screen shot from Project Navigator. The **Project Sources** area includes links to the Verilog coding and instantiated modules, which are required at the input to the synthesis procedure, and also allows the assignment of constraint files to any of the listed Verilog files. The **Design Flow** area shows the complete

flow, user constraints and also provides indication as to the design's progress through this flow. The **Editing Window** allows for the modification of any input file, while the **Run Window** shows the report for either: the stage which is currently running or the last stage to be run.

In the Design Flow window (see Fig. 3) the options to setup the user constraint files for the design should be specified. The constraint files are:

- **Create Timing Constraints:** This constraint file enables us to setup the clock period on the FPGA global clock pin.

- **Assign Package Pins:** Assign package pins were used when we downloaded into our target device.

- **Create Area Constraints:** Since no area constraints are not specified in our design, so we didn't use this constraint for our implementation.

There are three main stages in the Synthesis for FPGA flow shown in the *Project Navigator* **Design Flow** window in Figure 3. The *Synthesis* stage of the flow is essentially the same as standard synthesis processes, converting behavioural Verilog RTL to a gate-level equivalent. The XST (Xilinx Synthesis Technology) synthesis tool is employed in the design flow for the Digital Carrier Synchronization Module. The *Implement Design* phase employs NGDBuild software to *translate* the netlist, which is output from the *Synthesis* stage, along with design constraint information, to an intermediate format. The design is then *mapped* to pins of the Xilinx FPGA in use; this is specified in *Project Navigator.*

The design then undergoes the *place and route* operation, which provides output to the bit-stream generator. The *Generate Programming File* phase is entered largely on the iMPACT tool, which is capable of generating various file formats, depending on how the download to FPGA is to be conducted. Project Navigator also enables the incorporation of the Xilinx ChipScope™ Pro Integrated Logic Analyser into the flow.

2.2.2 Functional Verification Using Simulation

2.2.2.1 *What is Functional Verification?*

The main purpose of functional verification is to ensure that a design implements intended functionality. A functional verification path is shown in Figure 4. Functional verification reconciles a design with its specification. Without functional verification, one must trust that the transformation of a specification document into RTL code was performed correctly, without misinterpretation of the specification's intent [19]. It is important to note that, unless a specification is written in a formal language with precise semantics, it is impossible to prove that a design meets the intent of

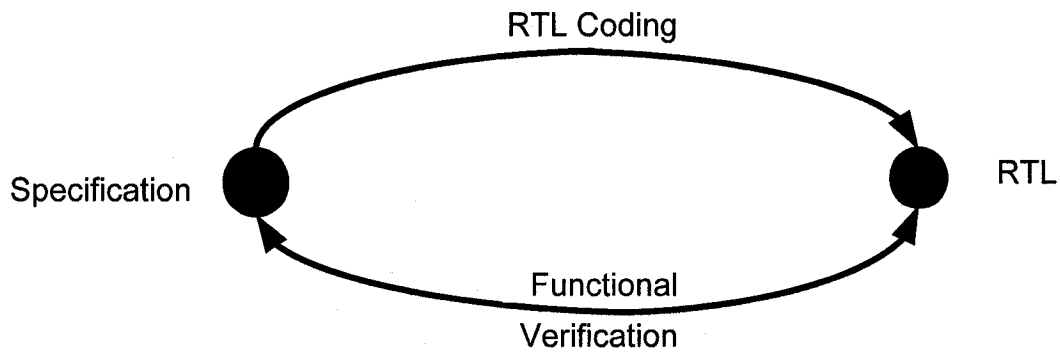

Figure 4. A Functional Verification Path [19]

its specification. Functional verification, as a process, can show that a design meets the intent of its specification [19]. Functional verification can be realized using three complementary approaches: Black-box, White-box, and Grey-box verification.

In our DCS project, we follow the simulation-based verification, which is also called Black-box verification. In a Simulation-Based Verification, the test environment has the following features:

• The testbench consisted of HDL procedures that provided stimulus data to the DUT or read data from it.

• The tests (testcases), which called the testbench procedures in sequence to apply selected input stimuli (random or directed) to the DUT and check the results, were directed towards specific features of the design.

2.2.2.2 *The Importance of verification*

Today, in the epoch of multimillion gates of ASICs, reusable Intellectual Property (IP), and System-on-a-Chip (SoC) designs, verification consumes about 70% of the design effort. The number of verification engineers is usually twice the number of RTL designers. When design projects are completed, the code that implements the testbenches makes up to 80% of the total code volume.

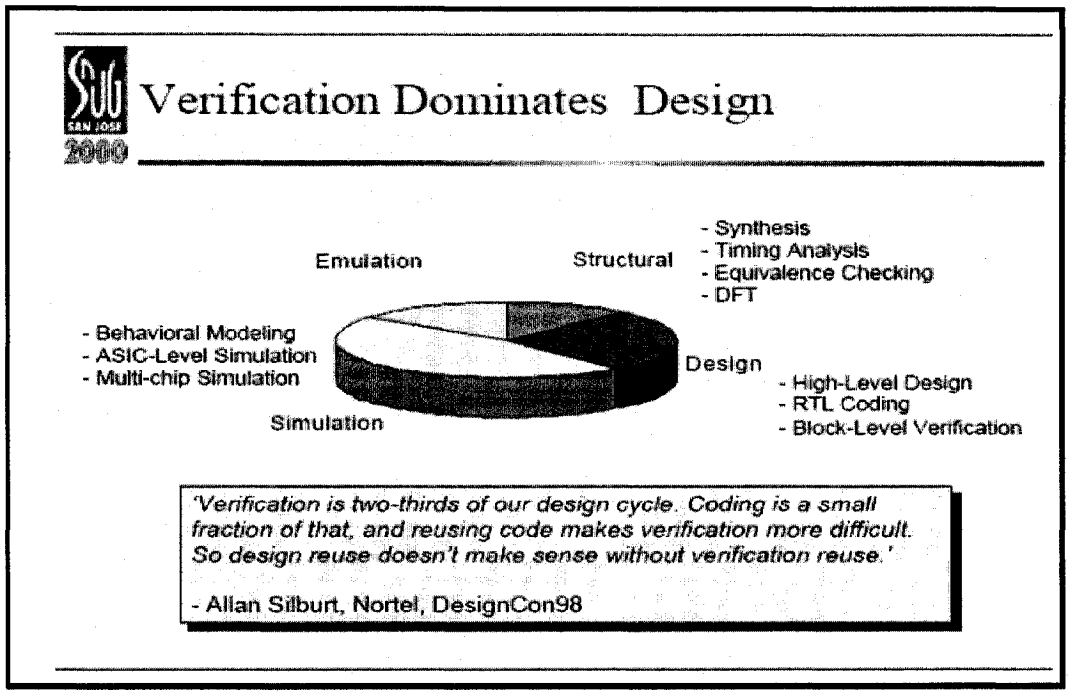

Figure 5. Verification Dominates Design [20]

Most of the case studies have shown that functional verification consumes more than 50 percent of the design-cycle time (Fig. 5) [20].

With functional verification involving such a dominant portion of the chip development process, SoC teams look for any opportunity for leverage. Some of the best leverage comes from the designers who wrote the RTL code that must be verified. Designers can no longer pass their code "over the wall" to the Verification team; they need to be involved in the verification process in order to ensure that the SoC works as intended. Therefore, the process of SoC verification is becoming a necessity for large, complex chip projects. This process encompasses a broad range of concepts, including verification friendly RTL coding standards for the designers, cross-participation in design and

Verification plan reviews, and clean, consistent interfaces at multiple levels of design abstraction.

2.2.2.3 *What is testbench?*

To simulate a design, an external apparatus called a test bench is often required to mimic the environment in which the design will reside. Among other functionality, the main purpose of a test bench, written in HDL, is to supply input waveforms to the design under test (DUT) and to monitor the response using waveform viewer such as Simvision whether the DUT produces the expected outputs.

Figure 6 shows a simple block diagram of a testbench that surrounds the DUT. A test bench is not manufactured as the design; it has far fewer coding style restrictions. Together with the perception that test benches are discarded once the design is verified, the structures of a test bench are often at the mercy of verification engineers [21]. Consequently, test benches frequently generate wrong stimuli, compare with wrong results, or miss corner cases, eventually diverting valuable engineering time to debugging the test benches instead of the design.

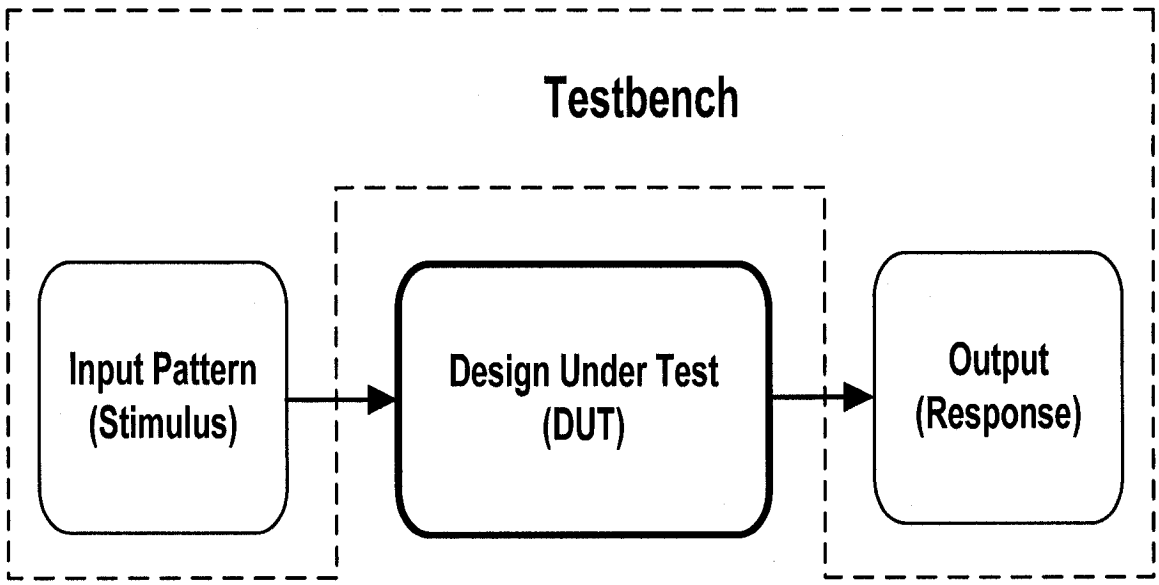

Figure 6. Block Diagram of a Testbench [19]

Furthermore, without well-organized guidelines, test benches can be a nightmare to maintain and hence are not reusable. Therefore, to have easily maintainable and reliable test benches, it is important to understand organizations and designs of test benches [21].

2.3 Summary

In this chapter, related works and also some preliminaries such as synthesis, testbench and simulation which are related to our research work are presented. In the following chapter, the analog phase locked loop (APLL) techniques and the operations of its key components will be discussed.

Chapter 3

Conventional Analog Phase Locked (APLL)

3.1 Introduction

This chapter focuses on the most common analog phase locked loop technique which enables us the thinking about the digital carrier synchronization.

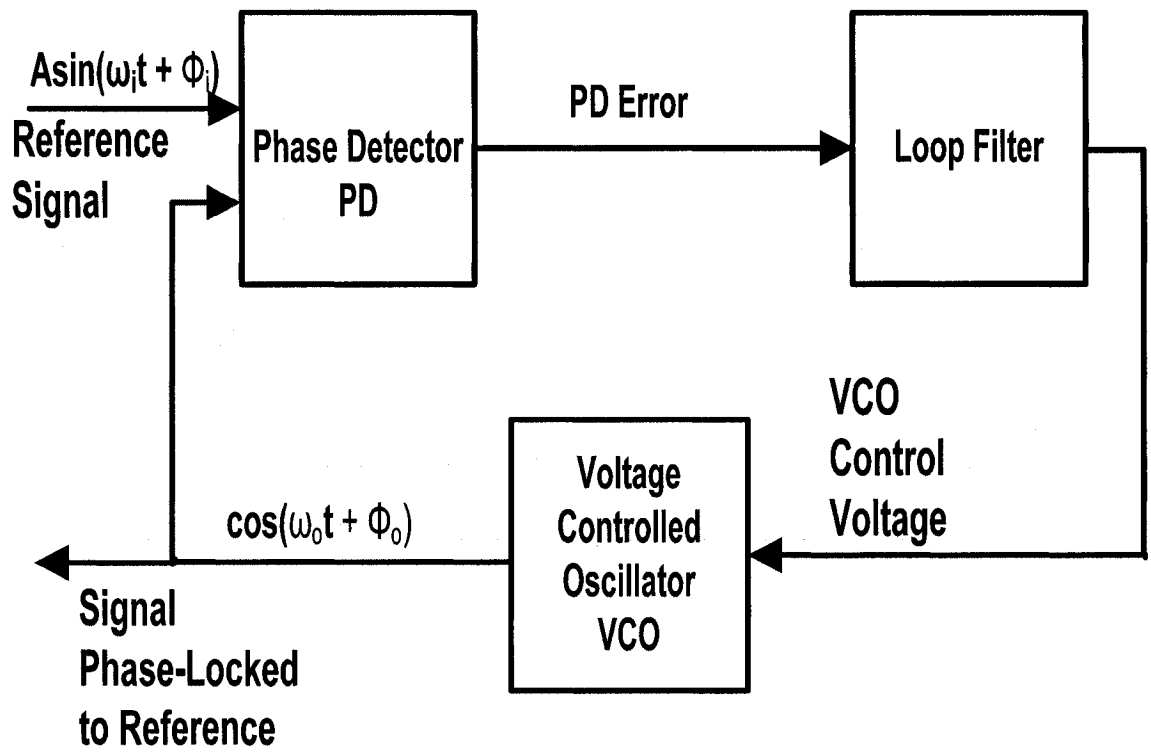

Figure 7. Analog Phase Locked Loop

An APLL is a circuit shown in Figure 7 synchronizing an output signal generated by an Oscillator, with a reference or input signal in frequency as well as in phase [22]. The synchronized Oscillator in analog PLL is a Voltage controlled Oscillator (VCO). If a phase error builds up, a control mechanism acts on the Oscillator in such a way that the phase error is again reduced to a minimum. In such a control system the phase of the output signal is actually locked to the phase of the reference signal.

In a feedback-system PLL regulates the phase φ_o of its periodic output signal, with the frequency ω_o, in a constant relationship to the phase φ_{in} of a periodic input signal, with the input frequency ω_{in}, by a feedback process [23]. The basic functional blocks of a classical analog PLL are phase detector (PD), loop filter (LF) and voltage controlled oscillator (VCO). In analog PLL all the functional blocks are in analog. In the following section we will introduce about the different components of Analog PLL.

3.2 Phase Detector (PD)

A phase detector (PD) is a circuit capable of delivering an output signal that is proportional to the phase difference between its two input signals S_1 (t) and S_2 (t). In analog PLLs, different types of phase detectors are used. In this analog PLL, we are considering an ideal multiplier phase detector shown in Figure 8 which is the first phase detector in the history of the PLL.

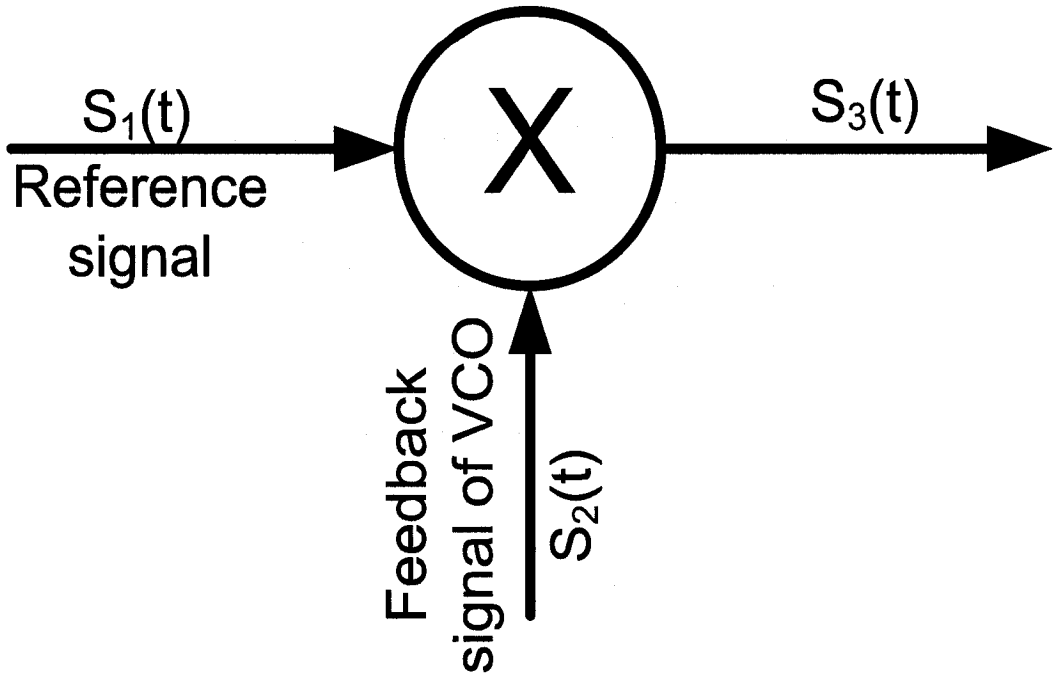

Figure 8. Phase Detector as a Multiplier

The multiplier phase detector (Figure 8) is completely used in analog PLL. Here we are considering with two sinusoid signals S_1 (t), which is the reference signal and S_2 (t), which is the feedback signal of the VCO. Both signals have same frequency but the phases are 90^0 out of phase. We have set the phase of these signals as a variable. Note that S_2 (t) is a cosine hence is 90^0 shifted from S_1 (t).

Multiplier Output, S_3 (t) = Reference Signal (t) * Feedback Signal of VCO

$$S_3(t) = S_1(t) * S_2(t)$$

Where,

$$S_1(t) = A_1 \sin[\omega t + \varphi_1(t)]$$
$$S_2(t) = A_2 \cos[\omega t + \varphi_2(t)]$$

The output of the multiplier is

$$S_3(t) = K_d A_1 A_2 \sin[\omega t + \varphi_1(t)] \cos[\omega t + \varphi_2(t)]$$

Where K_d is the gain of the multiplier. Now manipulating the above equation, we can get

$$S_3(t) = \underbrace{\frac{K_d A_1 A_2}{2} \sin[\varphi_1(t) - \varphi_2(t)]}_{First\ Part} + \underbrace{\frac{K_d A_1 A_2}{2} \sin[2\omega t + \varphi_1(t) + \varphi_2(t)]}_{Second\ Part} \qquad (1.1)$$

In the equation (1.1), we can see that the multiplier signal consists of two parts, the first one is the function of only the phase difference of two signals $S_1(t)$ & $S_2(t)$ and the second term is at a frequency which is twice the signal frequency ($the\ 2\omega t\ term$) plus the sum of the two phases [24].

We can use the equation (1.1) to develop the PLL by recognizing that the output signal of the multiplier is a function of the phase difference of the two input signals. We can use this useful information to synchronize the two signals. The higher frequency terms in the second part of equation (1.1) (twice the frequency) can be eliminated by filtering it out.

3.3 Loop Filter (LF)

We need a filter from getting rid of the unwanted higher frequency term which is in the second part of equation (1.1).

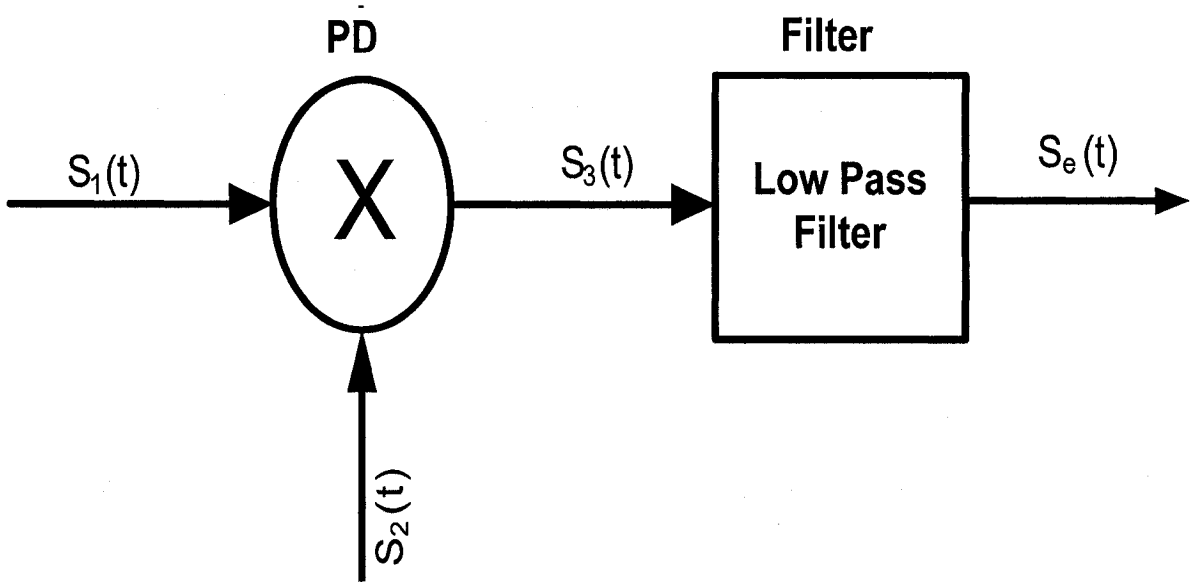

Figure 9. Filtering the Phase Detector Output signal to remove

Now we can add a loop filter shown in Figure 9, which acts as a low pass filter, at the output of the PD. In this APLL, we consider a passive first order loop filter.

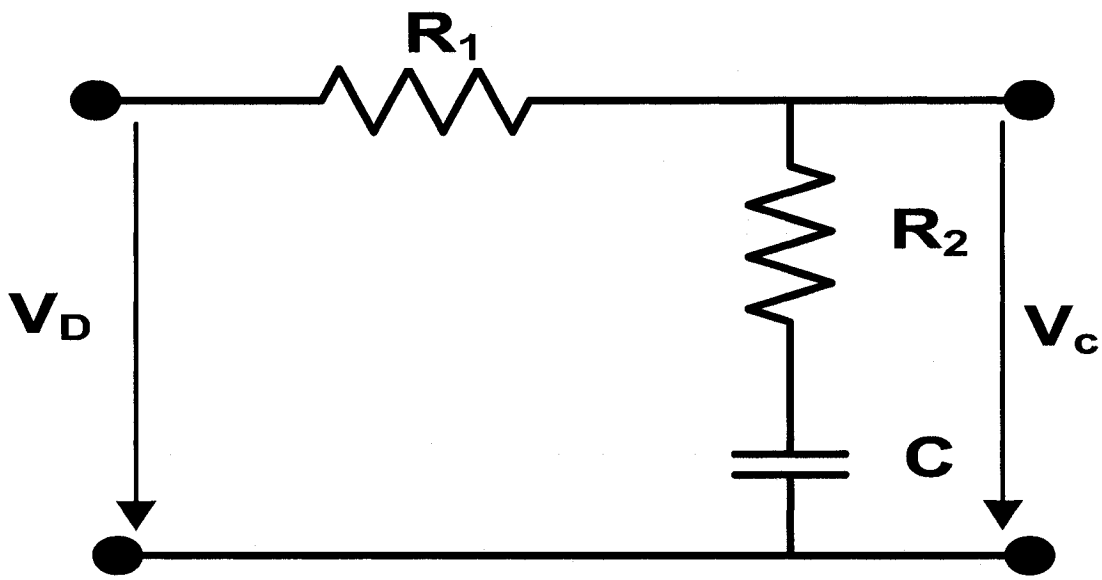

Figure 10. The Circuit Implementation of the first order passive Loop Filter

The circuit representation of the passive first order loop filter is shown in Figure 10. The output of the low pass filter (LPF) in Figure 9, as the phase difference is varied. It is called LPF because it only passes the lower frequency term and eliminates the higher frequency term. When there is a phase difference, then the signal out of the LPF is just the first part of Equation (1.1). We call this part the error signal which is also called the control signal.

$$S_e(t) = \frac{K_d A_1 A_2}{2} \sin[\varphi_1(t) - \varphi_2(t)] \qquad (1.2)$$

If phase difference is 0 degrees then we would expect the signal S_e (t) to be zero, which is the desired and the locked-state of the PLL. If the phase between the two signals (S_1 and S_2) varies from that, then, we would expect the filtered S_3 signal to change [24]. If the phase difference is not zero, i.e. S_e (t) is not zero; we need to oscillate the signal. For oscillating the signal, i.e. for making the S_e (t) signal is zero, we need an oscillator. In order for that we are using the Voltage Controlled Oscillator (VCO).

3.4 Voltage Controlled Oscillator (VCO)

Oscillators are a natural and expected part of the electronic scene. They occur in many applications and make possible circuits and subsystems that perform very useful functions. A voltage-controlled oscillator (VCO) (in Fig. 11) is an oscillator where the principle variable is a voltage. The VCO is an integral part of every phase locked loop circuit. It is an electronic oscillator, which is specially designed

to be controlled in oscillation frequency by a voltage input. The frequency of oscillation, or rate of repetition, is varied with an applied DC voltage, while modulating signals may be fed into the VCO to generate frequency modulation (FM) or phase modulation (PM).

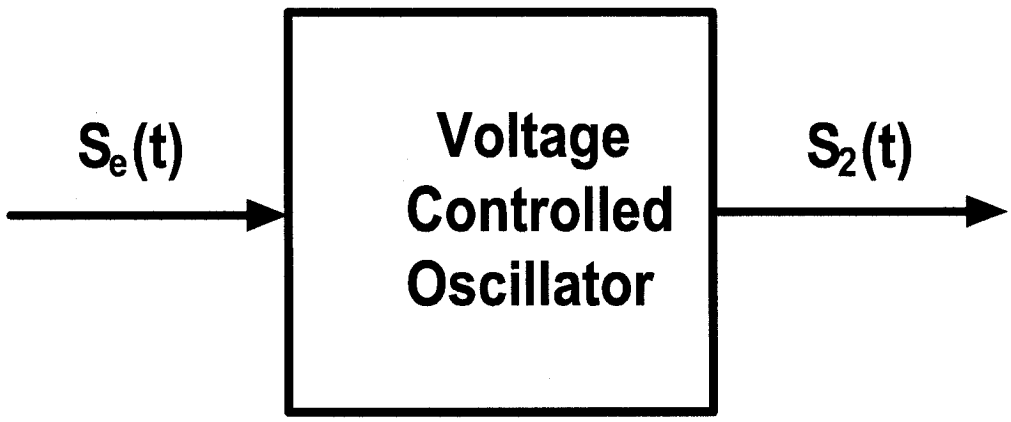

Figure 11. Block Diagram of Voltage Controlled Oscillator

The error signal provides an indication of what is happening to the input phase. We need an error signal to have zero amplitude and we can do that only by changing the phase of the signal S2 to match the phase of signal S1. VCO which produce the signal allows us to do that.

VCO in Figure 11 produces a periodic signal, the frequency of which changes based on a control signal applied externally. If the error signal is zero then, the VCO produces just its quiescent (center frequency). But if the error signal is something other than zero, then it responds by changing its operating frequency [24].

A constant of k_0 represents the sensitivity of the VCO. It represents the change in the instantaneous frequency of the VCO as a function of the error signal amplitude such that $K_0 = \frac{d\omega_0}{dv}$. The signal out of the VCO is given by, $S_2(t) = A_2 \cos(\omega_c t + \phi_2(t))$. The units of K_0 are Hertz per volts. For a given certain input voltage, it will produce a change in the output signal frequency by the following relationship.

$$\omega_{out} = \omega_c + k_0 v(t)$$

Where, ω_c is its center or operating frequency. So, if K =5000Hz/volt, then an input of 0.1 volt would produce a new output frequency of ω_c + 500 Hz.

For a periodic signal *p (t)*, if its frequency in Hz is equal to the rate of change of phase in 2π segment, or $f(t) = \frac{1}{2\pi}\frac{d\varphi_i(t)}{dt}$ and conversely, phase is the integral part of frequency over certain period of time, $\varphi_i(t) = 2\pi \int_0^t f_i(t)dt$ (1.3)

This relationship applies to all periodic signals, even those that are non-sinusoidal.

Now we can write the phase of the feed-back signal as,

$$\begin{aligned} \varphi_2(t) &= 2\pi K_0 \int_0^t S_e(t)dt \\ &= 2\pi K_0 S_e(t)t \end{aligned} \quad (1.4)$$

So as long as the error signal has non-zero amplitude, the phase of the VCO signal will keep on increasing until such time as it is decreased to zero. By substituting the equation (1.4) into equation (1.2) we get,

$$S_e(t) = \frac{K_m A_1 A_2}{2} \sin[\varphi_1(t) - \varphi_2(t)]$$

$$S_e(t) = \frac{K_m A_1 A_2}{2} \sin[\varphi_1(t) - 2\pi K_0 \int_0^t S_e(t)dt] \qquad (1.5)$$

The equation of S_e can be linearized by making the following assumption,

$$\sin(\varphi) \approx \varphi \text{ for small } \varphi$$
$$\sin[\varphi_1(t) - \varphi_2(t)] \approx [\varphi_1(t) - \varphi_2(t)]$$

Now we can rewrite equation (1.5) by removing the sine function.

$$S_e(t) = \frac{K_m A_1 A_2}{2} [\varphi_1(t) - \varphi_2(t)]$$

$$S_e(t) = \frac{K_m A_1 A_2}{2} [\varphi_1(t) - 2\pi K_0 \int_0^t S_e(t)] \qquad (1.6)$$

$$= \frac{K_m A_1 A_2}{2} [\varphi_1(t) - 2\pi K_0 S_e t]$$

In the equation (1.6), S_e is the amplitude of S_e (t) at time t.

For example, if the input signal changes by 10 degrees, this causes that the error signal to slowly increase in amplitude from 0 to 0.1. At time t, the frequency of the

signal produced by the VCO increases by K_0S_e, where S_e is the instantaneous amplitude of the error signal and time T is the sampling time.

As long as error signal is present, the phase keeps changing linearly. However, as the phase of the signal out of the VCO changes, the new difference in phase increases and the error signal amplitude decreases at the next go-around. This decreases the phase change further until the error signal amplitude has gone to zero. This is how an analog PLL works.

3.5 Summary

In this chapter, the conventional analog phase locked loop techniques and its important components are introduced. In the following chapter, the mathematical model of digital carrier synchronization (DCS) which is the digital version of analog phase locked loop will be introduced. Some important analysis of our DCS implementation will also be discussed.

Chapter 4

Mathematical Model of a Digital Carrier Synchronizer (DCS) in the Discrete Time Domain (Z-Domain)

4.1 Introduction

Before analyzing the mathematical model of a DCS system in the discrete time domain (Z-domain), we have to analyze the corresponding model in the continuous time domain (S -domain). In automatic control system theory, the transfer function of the second-order system in the S-domain (e.g. continuous time domain) can be written as,

$$H_s(S) = \frac{\omega_n^{\ 2}}{S^2 + 2\xi\omega_n S + \omega_n^{\ 2}} \qquad (4.1)$$

Where ω_n is defined as natural undamped frequency, and ξ is defined as damping ratio. This system is called a standard prototype second-order system. Based on the transfer function of a second-order prototype system, a characteristic equation of the system is defined as

$$\Delta(S) = S^2 + 2\xi\omega_n S + \omega_n^{\ 2} \qquad (4.2)$$

By solving the roots of the characteristic equation, two poles of the system, S_0 and S_1, can be derived.

$$S_0 = -\xi\omega_n + j\omega_n\sqrt{(1-\xi^2)} \qquad (4.3)$$

$$S_1 = -\xi\omega_n - j\omega_n\sqrt{(1-\xi^2)} \qquad (4.4)$$

4.1.1 Mapping the Poles of a second-order system from S-domain to Z domain

The transfer function of a second-order system in the Z-domain can be written in a general format as

$$H(Z) = \frac{N(Z)}{(Z-Z_1)(Z-Z_0)} \qquad (4.5)$$

Where Z_0 and Z_1 are two poles of the system in Z-domain. Corresponding to S-domain analysis, a characteristic equation of a discrete-time system is defined as

$$\Delta(Z) = (Z-Z_1)(Z-Z_0) = Z^2 - (Z_1+Z_0)Z + Z_1Z_0 \qquad (4.6)$$

C_1 and C_0 are defined as coefficients of the characteristic equation:

$$\begin{aligned} C_1 &= -(Z_1+Z_0) \\ C_0 &= Z_1Z_0 \end{aligned} \qquad (4.7)$$

Then the characteristic equation can be written in the simplified format

$$\Delta(Z) = Z^2 + C_1Z + C_0 \qquad (4.8)$$

By the definition of a discrete-time transformation [25], two poles of this system in the Z-domain can be mapped from the poles in S-domain as

$$Z_0 = e^{s_0 T_s}$$

$$\therefore Z_0 = e^{(-\xi\omega nT_s + j\omega nT_s\sqrt{(1-\xi^2)}} \; and$$

$$Z_1 = e^{s_1 T_s}$$

$$\therefore Z_1 = e^{(-\xi\omega_n T_s - j\omega_n T_s\sqrt{(1-\xi^2)}} \qquad (4.9)$$

Where, Ts is the sampling period of the discrete system.

With the poles mapped in the Z-domain and Equation (4.7), coefficients C_0 and C_1 of the characteristic equation (Equation 4.8) described by the parameters ξ and ω_n:

$$C_0 = e^{-2\xi\omega_n T_s}$$

$$C_1 = -2\, e^{-\xi\omega_n T_s} \cos(\omega_n T_s \sqrt{(1-\xi^2)}) \qquad (4.10)$$

4.2 Mathematical model of DCS Architecture

The mathematical model of DCS in discrete time domain (*Z-domain*) is shown in Figure 12. This DCS consists of three most important functional units:

(a) phase detector

(b) loop filter

(c) numerically controlled oscillator (NCO)

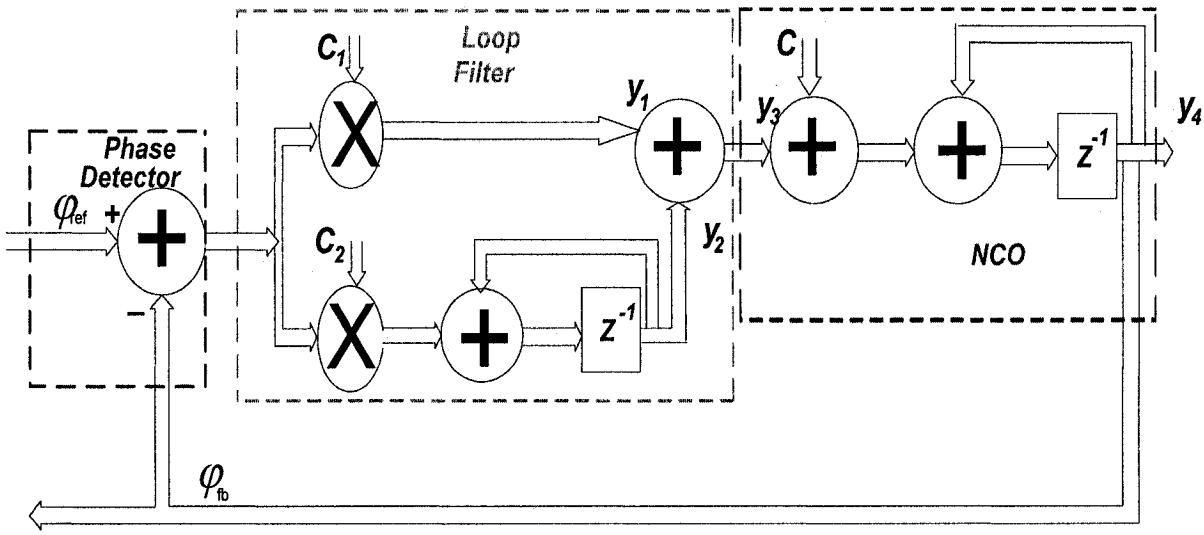

Figure 12. Mathematical model of DCS in discrete time domain (Z-domain)

Initially phase detector (PD) computes the phase error (φ_e) depending on the phase (φ_{ref}) of the reference signal, *data_ref* and the phase (φ_{fb}) of the feedback signal, *data_fb*. Then this phase error is filtered and finally, it is used to control the instantaneous output phase of the NCO (φ_{fb}). In figure 4.1, the coefficients C_1 and C_2 are used as a parameter of the digital loop filter, where $C_1 = 1$ and $C_2 = 0$, +1 or -1. In NCO, the parameter C is fixed for a constant value of 1.

The transfer functions of each component in the DCS are in the Z-transform format and analysis shown in the next section.

4.3 Analysis of the DCS

4.3.1 Transfer function of Loop Filter in the Z-domain

Figure 13 shows the loop filter in the discrete time (Z-domain) domain. The loop filter takes the phase error (φ_e) as an input and outputs y_3 depending on the

value of the coefficients C_1 and C_2. The transfer function of the loop filter is as follows:

$$y_2 = C_2\varphi_e + y_2Z^{-1}$$

$$\Rightarrow Zy_2 - y_2 = C_2Z\varphi_e$$

$$\Rightarrow y_2 = \frac{C_2Z\varphi_e}{(Z-1)} \qquad (4.11)$$

and $$y_1 = C_1\varphi_e \qquad (4.12)$$

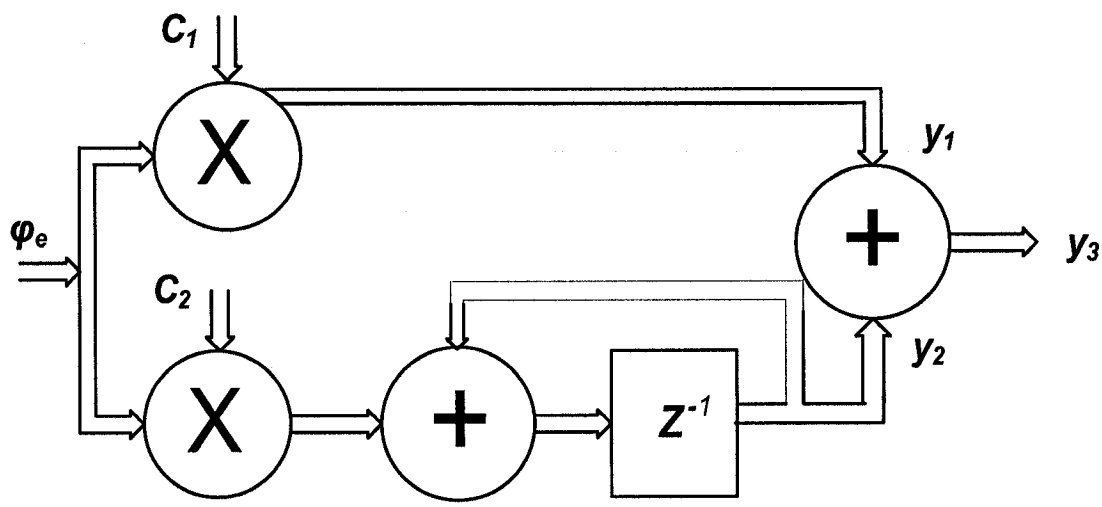

Figure 13. Block diagram of Loop Filter in Z-domain

Now adding the equation (4.11) and (4.12), we can get

$$y_3 = y_2 + y_1$$

$$\Rightarrow y_3 = \frac{C_2Z\varphi_e}{(Z-1)} + C_1\varphi_e$$

$$\Rightarrow y_3 = \frac{C_2Z\varphi_e + C_1(Z-1)\varphi_e}{(Z-1)}$$

$$\Rightarrow y_3 = H_1(Z) = \frac{\varphi_e[C_2Z + C_1(Z-1)]}{(z-1)} \qquad (4.13)$$

4.3.2 Transfer function of NCO in the Z-domain

The NCO in the discrete time (Z-domain) domain is shown in Figure 14. This NCO takes y_3 as an input which controls the instantaneous output phase of the NCO (φ_{fb}).

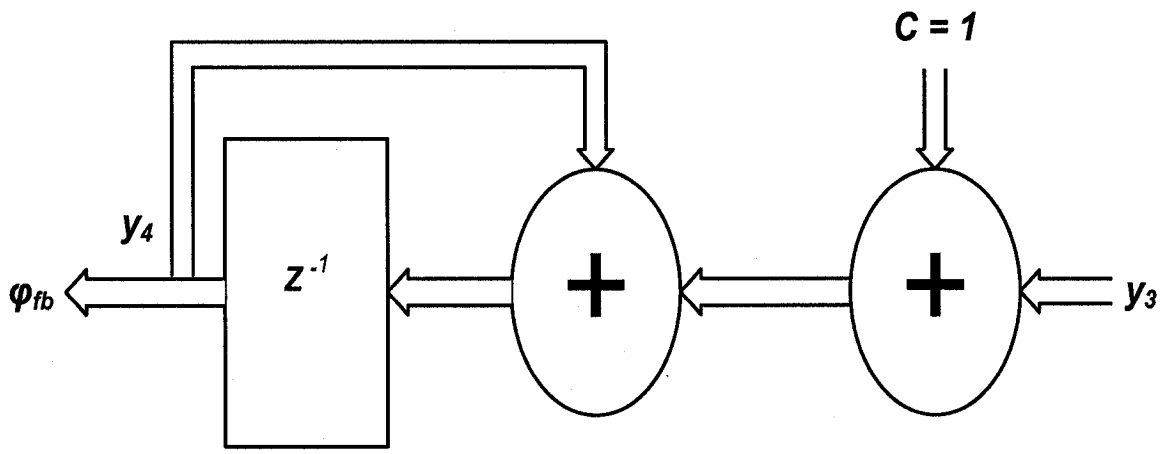

Figure 14. Block diagram of NCO in Z-domain

The transfer function of NCO in the discrete time domain is as follows:

$$y_4 = (y_4)Z^{-1} + (y_3)Z^{-1}$$

$$\Rightarrow y_4 Z = y_3 + y_4$$

$$\Rightarrow y_4 = \frac{y_3}{(Z-1)} \qquad (4.14)$$

Now combining the equation (4.13) and (4.14), we can get

$$y_4 = \frac{1}{(Z-1)} * \frac{\varphi_e[C_2 Z + C_1(Z-1)]}{(Z-1)}$$

$$\Rightarrow y_4 = \frac{1}{(Z-1)^2} * \varphi_e[C_2 Z + C_1(Z-1)]$$

So,

$$\varphi_{fb}(Z) = \frac{\varphi_e[C_2 Z + C_1(Z-1)]}{(Z-1)^2}$$

$$\Rightarrow \varphi_{fb}(Z) = H_2(Z) = \frac{\varphi_e[(C_1 + C_2)Z - C_1]}{(Z-1)^2} \qquad (4.15)$$

We know that,

$$H(Z) = \frac{\varphi_{fb}(Z)}{\varphi_{ref}(Z)} \qquad (4.16)$$

Then the closed-loop transfer function of our DCS model, H (Z), would be

$$H(Z) = \frac{\varphi_{fb}(Z)}{\varphi_{ref}(Z)} = \frac{A}{1 + \beta A} = \frac{H_1(Z) H_2(Z)}{1 + H_1(Z) H_2(Z) \beta}$$

$$\therefore H(Z) = \frac{H_1(Z) H_2(Z)}{1 + H_1(Z) H_2(Z)}$$

Since, the data from the NCO goes directly back to the phase detector, this DCS has unity feedback. Therefore, we set $\beta = 1$.

The expanded format of the transfer function of *H (Z)* can be derived as follows,

$$\therefore \varphi_e(Z) = \varphi_{ref}(Z) + (-\varphi_{fb})(Z)$$

$$\Rightarrow \varphi_e(Z) = \varphi_{ref}(Z) - \frac{\varphi_e[(C_1 + C_2)Z - C_1]}{(Z-1)^2}$$

$$\Rightarrow \varphi_e(Z) = \varphi_{ref}(Z)[(Z-1)^2] * \frac{1}{(Z-1)^2 + [(C_1 + C_2)Z - C_1]}$$

$$\Rightarrow \varphi_e(Z) = \frac{(Z-1)^2}{(Z-1)^2 + [(C_1 + C_2)Z - C_1]} * \varphi_{ref}(Z) \qquad (4.17)$$

Substituting Equation (4.17) into Equation (4.15) produces

$$\varphi_{fb}(Z) = \frac{[(C_1 + C_2)Z - C_1]}{(Z-1)^2} * \frac{(Z-1)^2}{(Z-1)^2 + [(C_1 + C_2)Z - C_1} * \varphi_{ref}(Z)$$

$$\Rightarrow \varphi_{fb}(Z) = \frac{(C_1 + C_2)Z - C_1}{(Z-1)^2 + [(C_1 + C_2)Z - C_1} * \varphi_{ref}(Z)$$

$$\Rightarrow \frac{\varphi_{fb}(Z)}{\varphi_{ref}(Z)} = \frac{(C_1 + C_2)Z - C_1}{(Z-1)^2 + [(C_1 + C_2)Z - C_1}$$

$$H(Z) = \frac{\varphi_{fb}(Z)}{\varphi_{ref}(Z)} = \frac{(C_1 + C_2)Z - C_1}{(Z-1)^2 + [(C_1 + C_2)Z - C_1} \qquad (4.18)$$

$$\therefore H(Z) = \frac{(C_1 + C_2)Z - C_1}{Z^2 - 2Z - 1 + C_1Z + C_2Z - C_1}$$

$$= \frac{(C_1 + C_2)Z - C_1}{Z^2 + (C_1 + C_2 - 2)Z + (1 - C_1)}$$

$$H(Z) = \frac{(g_1 + 2)Z - (1 - g_0)}{(Z^2 + g_1Z + g_0)} \qquad (4.19)$$

where, $C_1 + C_2 - 2 = g_1$ and $1 - C_1 = g_0$ (4.20)

The coefficients C_1 and C_2 can be resolved based on Equations 4.20 and 4.10:

$$C_1 = 1 - e^{-2\xi\omega_n T_s}$$

$$C_2 = 1 + e^{-2\xi\omega_n T_s} - 2e^{-\xi\omega_n T_s} \cos(\omega_n T_s \sqrt{(1-\xi^2)})$$

4.3.3 Phase Error Response of the DCS

Due to the reference phase (φ_{ref}) i.e. input phase, the phase error response (φ_e) is

$$\varphi_e(Z) = \frac{(Z-1)^2}{(Z-1)^2 + [(C_1 + C_2)Z - C_1]} * \varphi_{ref}(Z)$$

Assume that the phase of the input signal, i.e. reference signal has a step change. In the time domain, step changing of the phase of the reference signal can be described by the step function

$$\varphi_{ref}(t) = \Delta\varphi * u(t) \qquad (4.21)$$

Where, *u(t)* is the unit step function and $\Delta\varphi$ is the constant phase value of the reference signal by which the input signal phase jump. Now applying the Z-transform to the equation (4.21) yields

$$\varphi_{ref}(Z) = \frac{\Delta\varphi * Z}{(Z-1)} \qquad (4.22)$$

The output-response function of the DCS for a phase-step input can be written as

$$\varphi_{fb}(Z) = H(Z)*\varphi_{ref}(Z)$$

$$\Rightarrow \varphi_{fb}(Z) = \frac{\Delta\varphi * Z}{(Z-1)} * \frac{(g_1+2)Z-(1-g_0)}{(Z^2+g_1Z+g_0)} \qquad (4.23)$$

Based on the Equation (4.22), a numerical analysis can be carried out by using an existing software tool such as MATLAB. By this way, the steady-state error of an implemented DCS system can be observed.

By the definition of phase error, a phase error ($Err(Z)$) function can be written as

$$Err(Z) = \varphi_{ref}(Z) - \varphi_{fb}(Z) \qquad (4.24)$$

Substituting Equation (4.16) into Equation (4.23) produces

$$Err(Z) = [1 - H(Z)]\varphi_{ref}(Z) \qquad (4.25)$$

Now substituting Equations (4.18) and (4.22) into Equation (4.25), the phase-error function can be written as

$$Err(Z) = \frac{\Delta\varphi Z(Z-1)}{(Z^2+g_1Z+g_0)} \qquad (4.26)$$

According to the Final-Value Theorem,

$$\lim_{K\to\infty} e(KT) = \lim_{z\to 1} (1 - Z^{-1}) Err(Z) \qquad (4.27)$$

The steady-state error, which is the final value of *e (KT)* in the time-domain, can be derived by using this Final-Value Theorem [26]. The condition for using the Final-Value Theorem is that the function *(1- Z^{-1}) Err (Z)* has no poles on or outside the unit circle |Z|=1 in the Z-plane.

Substituting Equation (4.18) into Equation (4.21) yields

$$\lim_{K\to\infty} e(KT) = \lim_{Z\to 1} \frac{\Delta\varphi Z(Z-1)}{(Z^2 + g_1 Z + g_0)} = 0$$

It shows that the steady-state phase error is zero for the phase step input ($\Delta\varphi$).

4.3.4 Stability analysis of the DCS

One mandatory requirement for designing DCSs is that the DCS system must be stable. Basically, the stable condition of a discrete-time system occurs when the roots of the characteristic equation are inside the unit circle |Z| = 1 in the Z-plane (Shown in Fig. 15). Normally, after a system is implemented, numerical coefficients can be substituted into the characteristic equation. By solving the characteristic equation numerically, the positions of the poles can be found to determine if the system is stable.

One of the most efficient methods for testing the stability of a discrete-time system is Jury's stability criterion [27]. This method can guide the designs of a DCS to converge to an optimized stable system quickly, without a large amount of numerical calculation and simulation.

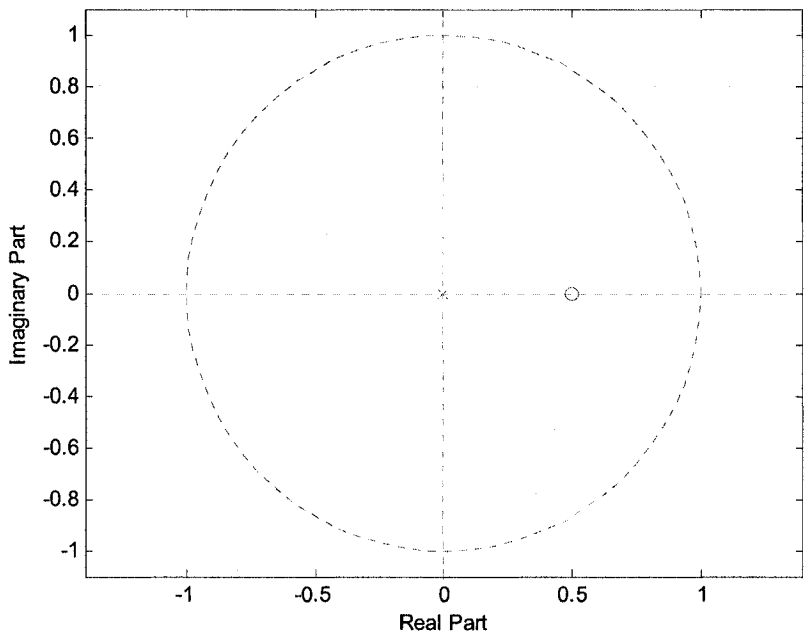

Figure 15. Pole-Zero Plot for a stable DCS

It can be applied directly to the DCS model (Shown in Fig. 12) to determine the stable condition. According to this criterion, the necessary and sufficient conditions are that the characteristic equation of DCS,

$$\Delta(Z) = a_2 Z^2 + a_1 Z + a_0 = 0$$

Should meet the following conditions in order to have no roots on or outside the unit circle:

$$\Delta(1) > 0,$$
$$\Delta(-1) > 0, \textit{and} \quad |a_0| < a_2$$

Applying these conditions to the denominator of Equation (2.11), the stable condition ranges of this DCS model can be derived as

$$0 < C_1 < 2$$
$$0 < C_2 < 4$$

4.3.5 Tracking Range of the DCS

In the previous section we showed that the 2nd order DCS is stable. Now it will be shown that this loop can also track a step input. To study the tracking, we examine the phase error (φ_e) that results from a simplified reference or input (φ_{ref}). A small phase error is usually desired and is considered to be the criterion of good tracking performance. If the error should become so large that the NCO skips cycles, the loop is considered to have lost lock, even if momentarily [28].

As the tracking range of DCS is proportional to the loop DC gain [29], we can expect a very wide tracking range. The tracking range is only limited by the half of the sampling frequency, which is $f_S/2$. The loop DC gain is given by

$$K_v = \lim_{Z \to 1} \left[\frac{C_2(Z-1)+C_1}{(Z-1)^2} \right] \to \infty \qquad (4.28)$$

4.4 Summary

In this chapter, a mathematical model of DCS architecture was derived, through which an in-depth understanding of DCS is attained. Some important parameters and stability of the DCS model are also analyzed. In the next chapter, the methodology and datapath implementation of DCS and the analysis of its important components will be presenetd.

Chapter 5

Design and Datapath Implementation of Digital Carrier Synchronizer (DCS)

5.1 Introduction

An important task for a digital communications receiver is to remove any frequency or phase offsets that might exist between the transmitter and receiver oscillators. A carrier-recovery loop is designed and implemented in order to remove this offset. A key component of a carrier-recovery loop is a phase-locked loop (PLL).

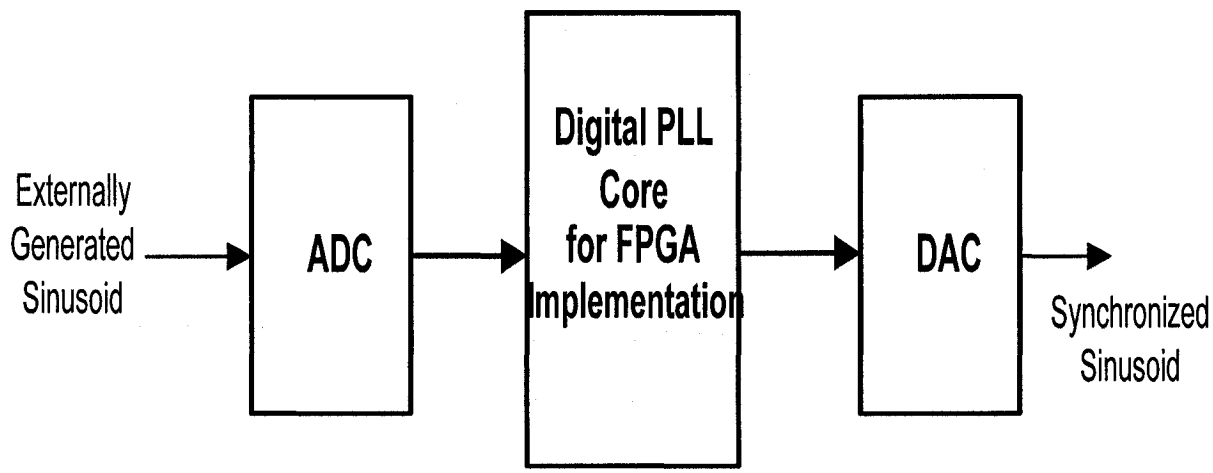

Figure 16. Digital Carrier Synchronization Path on the FPGA platform

The use of a PLL enables the receiver to adaptively track and remove frequency or phase offsets [30]. Figure 16 shows the digital carrier synchronization path.

An analog to digital carrier (ADC) receives the externally generated sinusoid signal and outputs the sample data as input to the digital PLL which is implemented in the FPGA, and a digital to analog converter (DAC) receives the locally generated data and outputs the synchronized sinusoid signal.

The goal of this digital carrier synchronization (DCS) is to produce a locally generated data samples called data_fb which is synchronized to input data samples called data_ref. Three major functional units of DCS shown in Figure 17 are A. Phase Detector (PD), B. Loop Filter (LF), and C. Numerically Controlled Oscillator (NCO).

5.2 Phase Detector

A PLL is driven by the phase error, in analog PD it was S_3 (t), signal which is

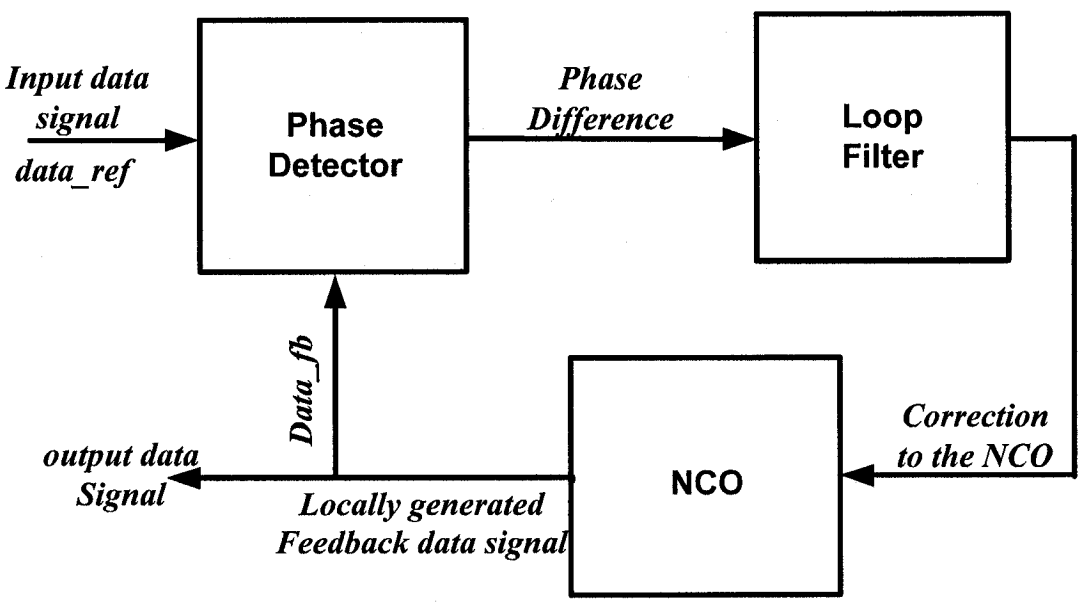

Figure 17. Datapath of the Digital Carrier Synchronization

generated by the phase detector. The phase detector generates the phase difference value on the input signal and the oscillated signal, i.e. *data_ref* (in analog it was S_1 (t) and *data_fb* (in analog it was S_2 *(t))*, with respects to their rising edges.

5.2.1 Data path of the Phase Detector

The datapath of the phase detector is shown in Figure 18. The *data_ref* signal is driven by the sampling clock, which is the system clock. In order to achieve the higher detection accuracy, a higher frequency for the detection clock may be used.

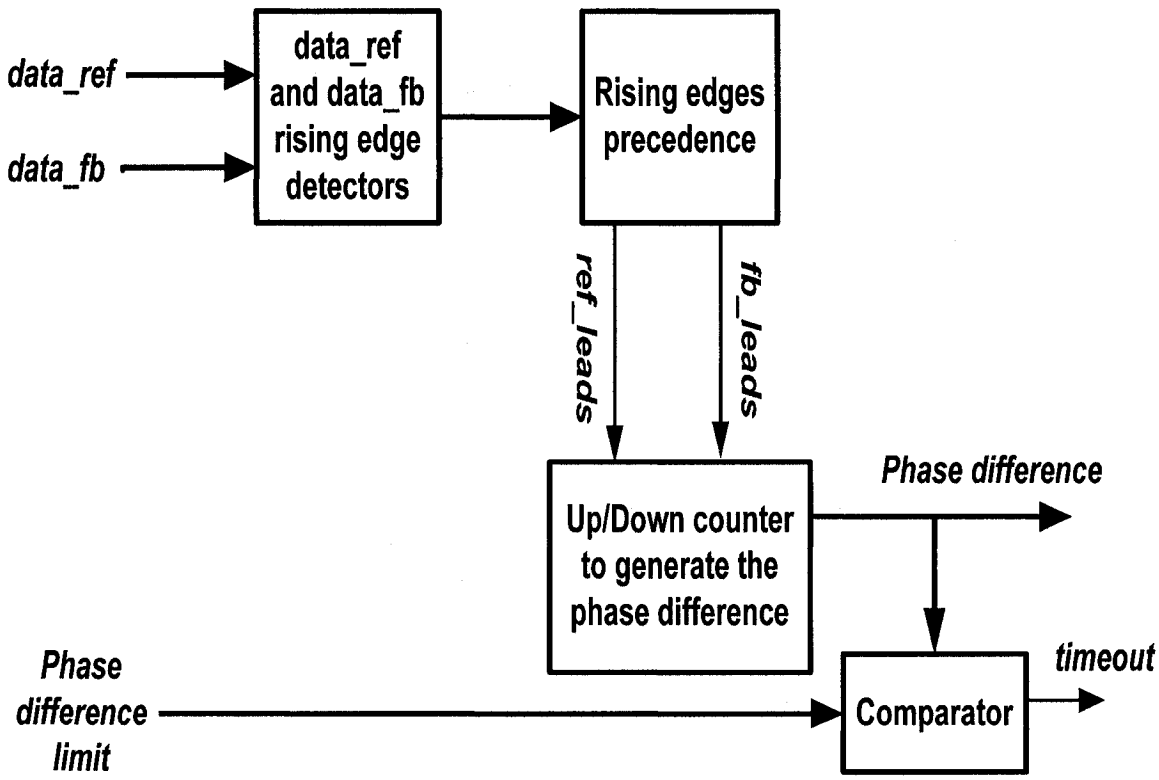

Figure 18. Datapath of Phase Detector

In Figure 18, the edge detector block detects the edges of the *data_ref* and *data_fb* signals with respect to the rising edge of the clock.

- *if (data_ref(n-1) < 0 and data_ref(n) ≥ 0), posedge_ref = 1*
- *if (data_fb(n-1) < 0 and data_fb(n) ≥ 0), posedge_fb = 1*

Where, posedge_ref indicates the rising edge of the data_ref and posedge_fb shows the rising edge of the data_fb signal. The rising edge of the data_ref and the rising edge of the data_fb are determined in one clock cycle.

The rising edges precedence block provides the precedence of the data_ref or the data_fb rising edges, which are indicated by ref_leads and fb_leads (in Fig. 18) respectively.

- if (posedge_ref =1 and data_fb < 0), ref_leads = 1
- if (posedge_fb =1 and data_ref < 0), fb_leads = 1

Where, ref_leads =1 shows that the posedge_ref comes before the posedge_fb signal, i.e. the reference signal comes before the feedback signal, and fb_leads =1 indicates that the posedge_fb occurs before the posedge_ref.

And an up/down counter accumulates the phase steps between the two edges to output the phase difference. If ref_leads = 1, the up/down counter (in Fig. 18) will increment, and if fb_leads = 1, the counter will decrement. In this way the counter provides the phase difference value at the output of the phase detector.

Finally, the timeout signal indicates that the phase difference between the two input signals data_ref and data_fb are larger than the phase limit, which is set by

the user. For reducing phase difference value (phase error), we need to filter the signal.

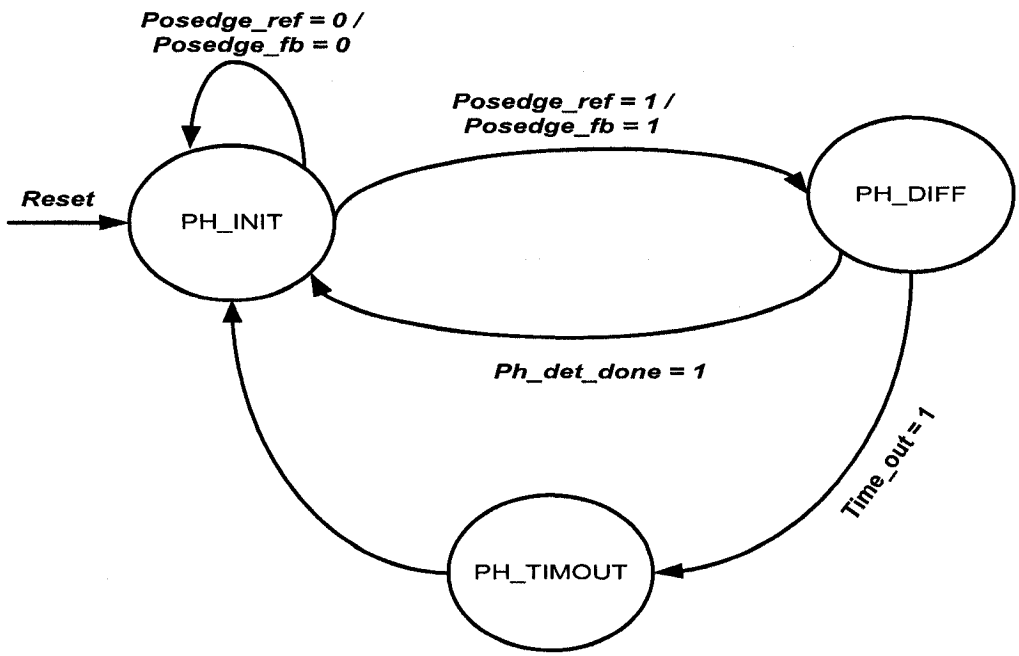

Figure 19. Phase Detection State machine

A phase detect state machine is shown in Figure 19. It has three states phase initialization, phase difference, and phase detects state.

5.3 First Order Digital Loop Filter (LF)

A first order digital Loop Filter is used to filter the phase difference or phase error (φ_e) signal, which is generated by the phase detector in order to supply the required correction to the NCO.

5.3.1 Data path of the Loop Filter

Figure 20 shows the datapath of the first order loop filter. It has two parts: the

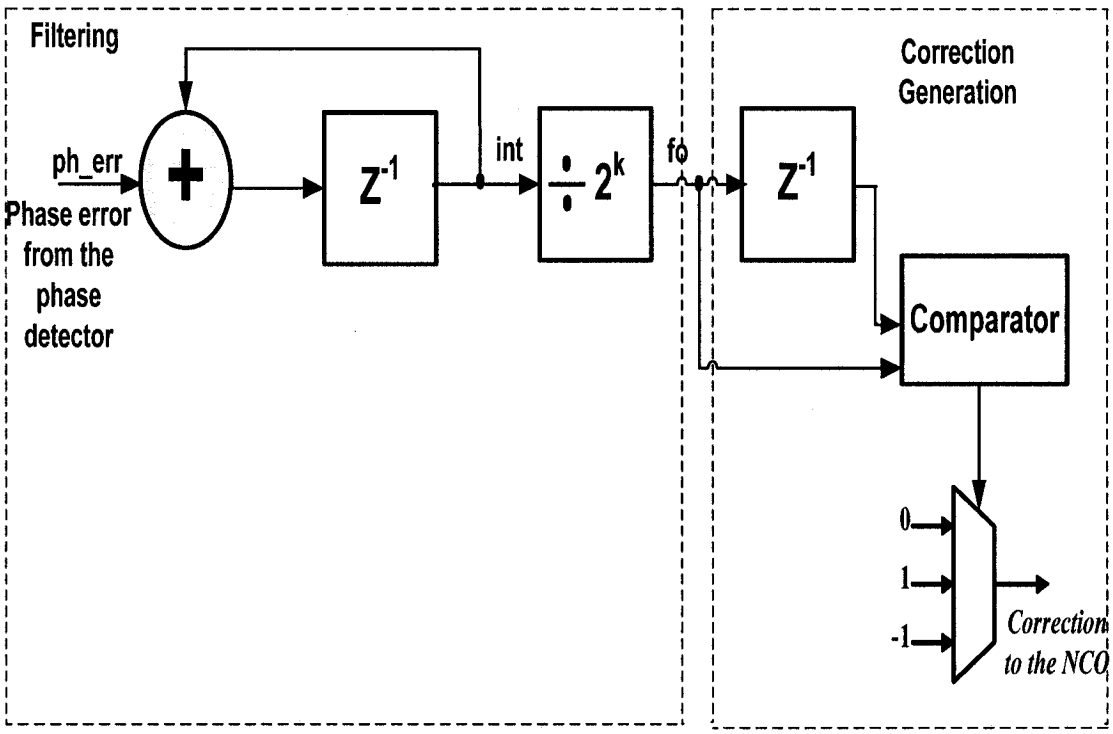

Figure 20. Datapath of 1st Order Loop Filter

filtering part and the correction generation part. The filtering part, firstly, it integrates the phase error (*ph_err*) signal, which is obtained from the output of the phase detector. Secondly, it accumulates the value and finally, it shifts the accumulated value to set the required bandwidth for the filter. If the output of the filtering part, which is the integrator value, provides the most significant bit (MSB), the filter will pass the lowest phase error frequencies. And if the output of the integrator is least significant bits, it will pass the highest phase error

frequencies. For this implementation, the filter passes the LSBs as the integrator (*int* in Fig. 20) is divided by the 2^k.

The expression of the integrator is,

$$\text{int}(n) = ph\_err(n) + ph\_err(n-1) \tag{5.1}$$

If we compare the equation (5.1) with the equation (4) of analog loop filter, we can see the same relationship. In the correction generation part, it compares the previous value of the output of the filtering part with the old value of the filtering part. This gives the trend of the phase error. The expression for the correction generation part is:

$$Correction\ to\ the\ NCO = \begin{cases} -1, & if\ fo(n) < fo(n-1) \\ 0, & if\ fo(n) = fo(n-1) \\ +1, & if\ fo(n) > fo(n-1) \end{cases}$$

Where, *fo* determines the output of the filter. In order to ensure that the output of NCO changes smoothly, the absolute correction value doesn't go over 1. Last but not the least; we also use the 2nd order loop filter in order to examine the overall performance of our DCS.

5.4 Numerically Controlled Oscillator (NCO)

Numerically Controlled Oscillator (NCO) is one of the key components of digital carrier synchronizer (DCS). In DCS, NCO allows to perform the precise

adjustment of the carrier frequency based on the output of the phase-detecting circuit. The NCO is controlled by the numerical value or the sampling index (discussed later). For generating periodic-like signal, NCO is an established method. In this research, we investigate three different types of NCO, which are used in DCS. The first one is the Look Up Table (LUT) based NCO, the second is CORDIC based NCO, and the finally Xilinx ROM based NCO.

5.4.1 Datapath of LooK Up Table (LUT) Based NCO

A Lookup Table (LUT) based NCO implementation is shown in Figure 21. It is

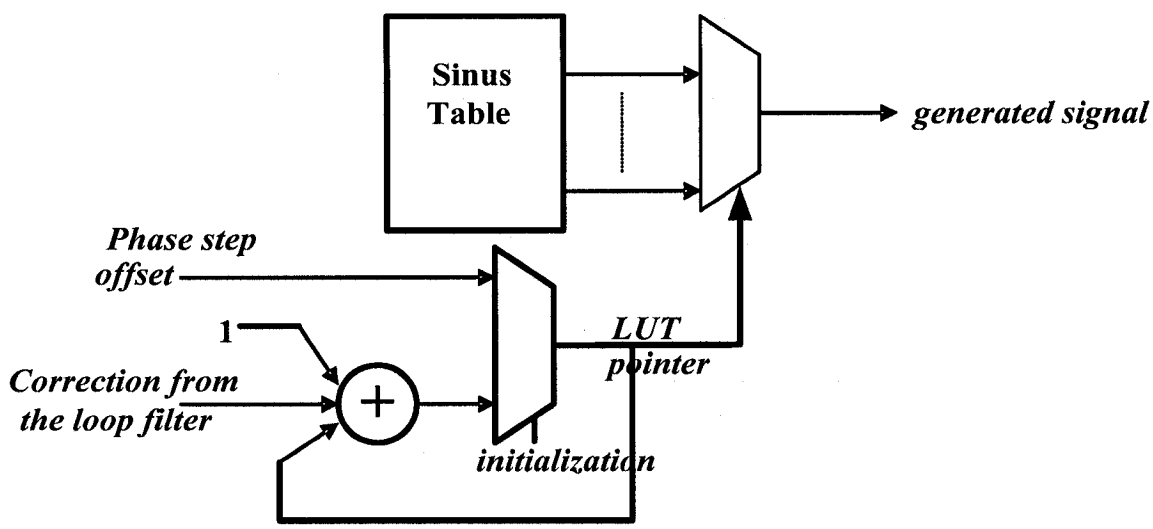

Figure 21. Look Up Table (LUT) Based NCO

one of the popular schemes to implement numerically controlled oscillator based on table-look-up. A lookup table method stores the samples of the sinusoidal signal depending on the carrier frequency, the sampling frequency and the

carrier amplitude. Afterwards those samples are being read out of the lookup table to produce the sinusoidal signal in an order determined by the total number of samples or sampling index. According to our design specification [30], the given parameters are, carrier frequency, Fc = 2.048 MHZ, the sampling frequency, Fs = 50 MHz. Since our data bus has one sign bit [MSB] and seven data bits, the carrier amplitude, A is equal to $2^7 - 1 = 127$. Hence, the maximum value of the sample index n, is equal to, (Fs / Fc), 24. Bearing this in mind, we can state that the data_fb, which is generated by the NCO block, is a function of n and can be represented by Equation (5.1).

$$data\_fb(n) = A\sin(2pi * (Fc/Fs) * n) \qquad (5.1)$$

The above equation signifies the output value of *data_fb* in terms of the sampling index n, carrier Frequency F_c, sampling frequency F_s and the amplitude A. Note that the number of sampling clock periods in one carrier period cannot exceed the value of F_s / F_c and the smallest angle value is $2*pi/(F_s/F_c)$. It is easily visible to the readers that the sample index n is the LUT pointer value which is shown in Figure 21. If the *LUT_Pointer* exceeds the value of F_s/F_c, the previous value of *data_fb* is retained with the Flip-Flop.

In the following sections, we will explain the datatapath modification of different DCS implementations that means modification of Loop Filter and NCO. The first order digital loop filter and NCO with look up table were explained in section 5.3 and 5.4 respectively. Now we will first discuss the implementation and analysis of

the second order digital loop filter with its different parameters. Then we will explain the implementation of CORDIC based and Xilinx ROM based NCO.

5.5 Digital Loop Filter

Digital filters are preferably used in the place of analog filters as they eliminate a number of problems associated with their classical analog counterparts. Digital filters belong to the class of discrete-time LTI (linear time invariant) systems, which are characterized by the properties of causality, recursibility, and stability [25]. They can be characterized in the time domain by their unit-impulse response and in the frequency domain by their transfer function. The unit-impulse response sequence of a causal LTI system can be of either finite or infinite duration, which determines whether they are classified as Finite Impulse Response (FIR) or Infinite Impulse Response (IIR) systems. FIR filters include low-pass, high-pass, band-pass, and band-stop filters [25].

A few basic steps are involved to design a digital filter. Initially, it needs to be established the desired response and based on that, a class of filters needs to be selected that approximate the desired response. The next step is to select the best member in the filter class, implement the best filter using a general purpose computer, a DSP, or a custom hardware chip, and analyze the filter performance to determine whether it satisfies all the given criteria. For the DCS, we decided to design a second order digital low pass filter using FIR and IIR.

5.5.1 Designing a IIR Low pass Filter

A low pass filter is a device that passes all frequencies below its cut-off frequency and rejects those above the cut-off. IIR filters have traditional analog counterparts (Butterworth, Chebyshev, Elliptic, and Bessel) and can be analyzed and synthesized using more familiar filter design techniques. Infinite impulse response filters get their name because their response extends for an infinite period of time. This is because they are recursive, i.e. they utilize feedback.

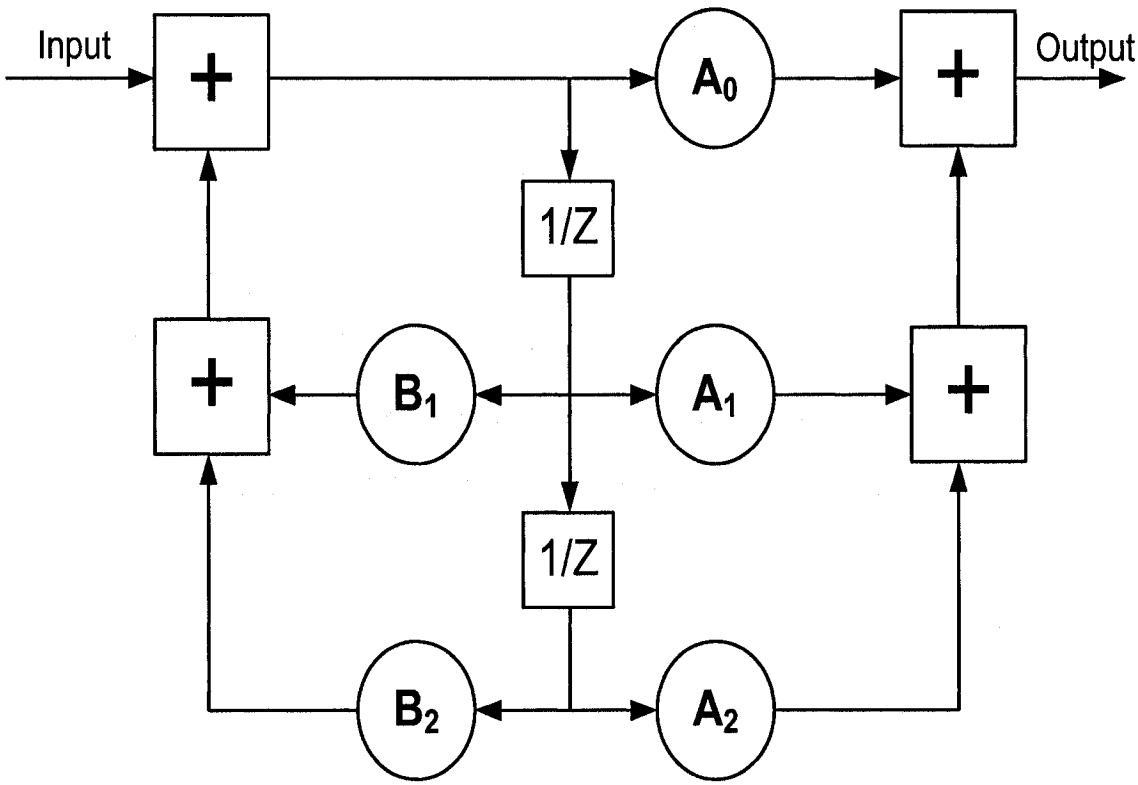

Figure 22. Second Order Low Pass IIR Filter [31]

The basic IIR filter is based on the biquadratic (biquad) structure [31]. A second order low pass IIR filter is shown in Figure 22, which is chosen for the design of

DCS. The delay elements are denoted as *1/Z* in this diagram. The *1/Z* term is sometimes written as Z^{-1}, especially in transfer function equations.

The second order digital IIR filter shown in Figure 22 uses four adders, two delays, and four multipliers. The multiplier coefficients are A0, A1, A2, B1, and B2. These coefficients are calculated during the filter design process. The transfer function of the second order IIR filter is:

$$H(Z) = \frac{Y(Z)}{X(Z)} = \frac{A_0 + A_1 Z^{-1} + A_2 Z^{-2}}{1 - B_1 Z^{-1} - B_2 Z^{-2}}$$

The feed-forward element A_0 gives the DC gain and is often unity. There is no feedback element B_0, which is replaced by a unity-valued element because the signal path through this element is forward, not backward [31].

5.5.2 Designing a FIR Low pass filter

An FIR filter comprises an array of delay elements (labelled Z^{-1}) connected in series. The input samples, X (n) are passed through these delay elements. A tap is taken after each element, and, at any sample instance, the value of the sample is multiplied by a filter coefficient. Thus a multiplier is needed for each delay element. Finally, the outputs of all the multipliers are added together to give the output. The number of taps is given by *N,* but there are *N-1* delay elements; the term *N-1* is sometimes referred to as the filter order. It is common to use an odd number of taps, which results in an even number of delay elements [31].

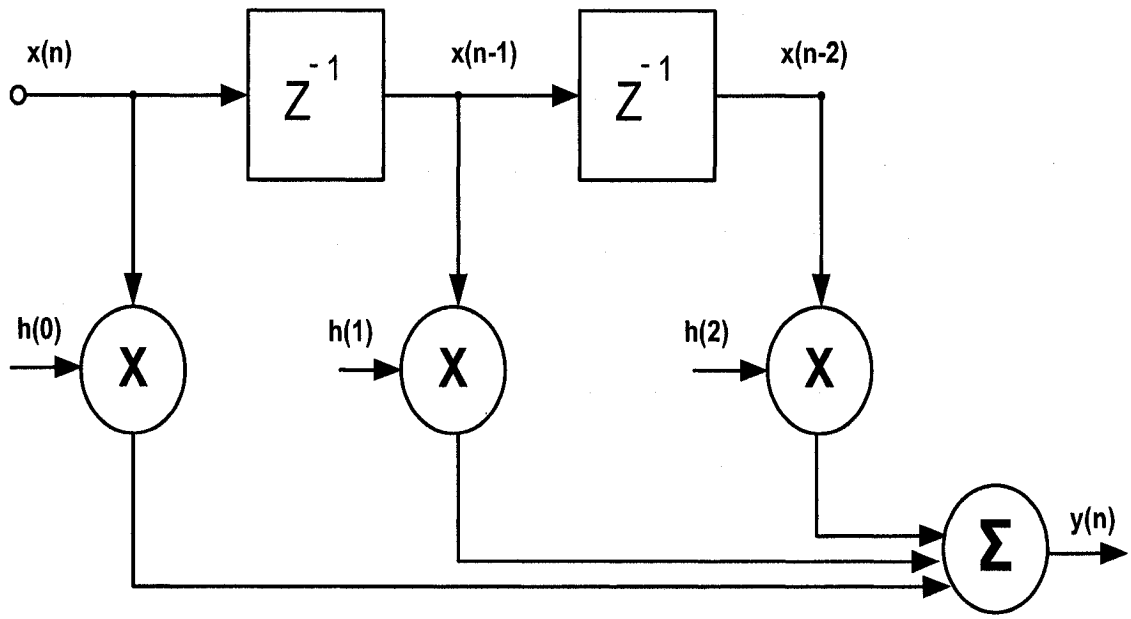

Figure 23. Second Order Low Pass FIR Filter [31]

For our design we chose 3-tap FIR filter, which has an order of 2, is given in Figure 23. The input samples of the second order FIR filter, shown in Figure 23, are x (n), x (n-1), and x (n-2), and the coefficients are h (0), h (1) and h (2). So the output becomes

$$y(n) = h(0)x(n) + h(1)x(n-1) + h(2)x(n-2) \tag{5.2}$$

and the generalized form of the N-tap FIR filter is

$$y(n) = h(k) * x(n)$$

$$\Rightarrow y(n) = \sum_{k=0}^{N-1} h(k)x(n-k) \tag{5.3}$$

By varying the weight of the coefficients and the number of filter taps, virtually any frequency response characteristic can be realized with an FIR filter. Various

methods are available for designing FIR filter, such as Equiripple, Least square, Window etc. We chose Kaiser Window method for designing the second-order FIR filter. The cut-off frequency of an FIR filter is directly proportional to the data sampling clock frequency. Using a single set of coefficients, the cut-off frequency can be doubled by doubling the sampling clock frequency. The normalized clock frequency for a digital filter is 1 Hz or 2Π *rad / s* [31].

In the following section we will analyze the designed low pass FIR and IIR filter to make sure that they met the required specifications, and using the verified coefficients to generate RTL so that the FPGA resources usage could be determined and compared.

5.6 Xilinx System Generator

System Generator is a tool from Xilinx that is used to generate HDL (Hardware Description Language) designs using high level constructs in Matlab. System Generator is capable of generating a diverse set of logic components, including mathematical, communication, and DSP components. System Generator can also generate a variety of digital filters, which is one of the primary concerns for our research.

System Generator makes use of a custom "FDATool" (Filter Analysis and Design Tool), which is an interface that allows the design of many types of digital filters to meet required filter specifications at a very high level. With this toolkit, we were able to graphically design FIR filters and interactively review filter responses. We

viewed the time waveforms and the spectra of both the input signal and the filtered output signal to show how the present filter performs on real-world signals. The FDATool panel provides its users with an interface with digital controls that allows users to adjust the desired filter specifications such as the required sampling, pass band, and stop band frequencies. In our case, initially we chose the cutoff frequency 0.042MHz and the sampling frequency 50MHz. These allowed us to design several different low-pass filters, some designed for accuracy, while others sacrificed accuracy for simplicity of design. We use System Generator to implement the entire filter design, from high level specifications down to synthesized HDL / RTL models.

5.6.1 Strategy

5.6.1.1 Selecting an FPGA Board

We designed a second-order Window FIR and Butterworth IIR low-pass filter in Matlab FDATool with 3 coefficients. The coefficients were then quantized using fixed-point arithmetic algorithm to generate the Verilog HDL code using Xilinx System Generator. Finally, we synthesized the Verilog HDL code for the FPGA board, which is VirtexII-Pro (XC2VP50-6FF1152).

5.6.1.2 Selecting a Digital Filter Design Method

We designed FIR filter using Kaiser Window method and IIR filter using Butterworth method. So that we could determine which type of filter used up the least resources on the FPGA selected. Once we had results of the resource usage for each of the types of filters designed, we picked one. In this case we

decided to select the Kaiser Window FIR filter design method for the rest of our simulations and analysis of DCS since it is one of the more commonly used filter design methods in digital signal processing. The physical resources occupations of the VirtexII-Pro PFGA board are shown in Table 1.

Table 1. Physical resources Occupation on FPGA board for FIR and IIR Filter

Device (VirtexII-Pro) XC2VP50 Package FF1152 Speed Grade -6	Report Using Xilinx ISE 8.1i	
	Second Order IIR Filter	Second Order FIR Filter
LUT	315	156
FFs	106	55
Slices	172	94
Gate Counts	1,615	3,546

5.6.1.3 Kaiser Window Method

Through the choice of the window shape and duration, we can control the properties of the resulting FIR filter. Kaiser developed a simple formalization of the window method of the digital FIR filter in 1974 [25]. The Kaiser window is defined as

$$w(n) = \begin{cases} \dfrac{Io[\beta(1-[(n-\alpha)/\alpha]^2)^{1/2}]}{Io(\beta)}, & 0 \le n \le M \\ 0, & \text{Otherwise} \end{cases} \qquad (5.4)$$

Where α = *M/2*, and *Io (.)* represents the zeroth-order modified Bessel function of the first kind. The Kaiser Window has two parameters: the length *(M + 1)* and a shape parameter β. By varying *(M + 1)* and β, the window length and shape can be adjusted to trade sidelobe amplitude for mainlobe width. If the window is tapered more, the sidelobes of the Fourier transform become smaller, but the mainlobe becomes wider. The peak approximation error δ is determined by the choice of β. Given that δ is fixed, the passband frequency is ω_p, and the stopband cut-off frequency is ω_s, so the transition region has width

$$\Delta\omega = \omega_s - \omega_p$$

for the lowpass filter approximation. Defining $A = -20 \log_{10} \delta$, Kaiser determined empirically that the value of β needed to achieve a specified value of *A* is given by

$$\beta = \begin{cases} 0.1102\,(A - 8.7), & A > 50 \\ 0.5842\,(A - 21)0.4 + 0.07886\,(A - 21), & 21 \le A \le 50 \\ 0.0, & A < 21 \end{cases} \qquad (5.5)$$

Furthermore, to achieve prescribed values of *A* and $\Delta\omega$, *M* must satisfy

$$M = \frac{A - 8}{2.285\,\Delta\omega} \qquad (5.6)$$

Equation (5.6) predicts *M* to within ± 2 over a wide range of values of $\Delta\omega$ and *A*. Thus these formulas, the Kaiser Window design method requires almost no iteration or trial and error [25].

It is straightforward to design an FIR lowpass filter to meet prescribed our specifications using the design formulas. The procedure is as follows

Step 01: Initially we have to establish the specifications. This means that we have to select the desired ω_p, ω_s and the maximum tolerable approximation error. For window design, the resulting filter will have the same peak error δ in both the passband and the stopband. For this FIR low-pass design, we choose the following specifications ω_p = 2.048 MHz, ω_s = 3.12 MHz, and δ = 0.0001.

Step 02: To determine the parameters of the Kaiser Window, we first compute

$$\Delta\omega = \omega_s - \omega_p = 3.12 - 2.048 = 1.072, \qquad A = -20\log_{10}\delta = 60$$

To obtain the required values of β and *M*, we substitute these two quantities in Equations (5.5) and (5.6).

$$\beta = 5.65,\ M = 21$$

Now in the following sections we will analysis the CORDIC and Xilinx based NCO for the implementations of DCS.

5.7 CORDIC Based NCO

A look-up table based NCO was discussed in Section 5.4. One alternate candidate for NCO design, which is well suited to FPGA implementation, is *coordinate rotation digital computer* (CORDIC) arithmetic [32]. The CORDIC algorithm is an iterative procedure that can be used to compute a diverse range of mathematical functions.

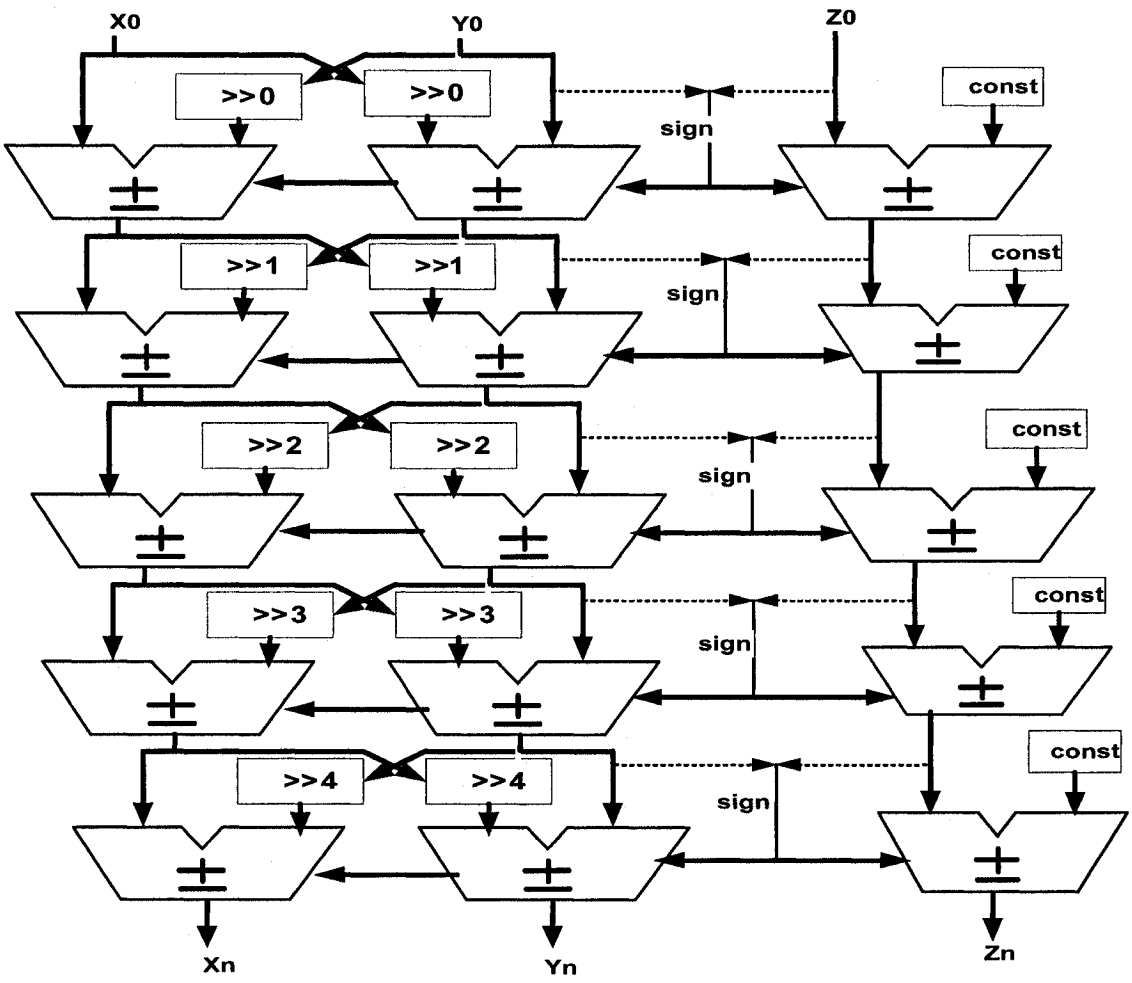

Figure 24. CORDIC Based NCO

CORDIC arctangents are computed using the *vectoring* mode as equations defined by Equation (5.7) and (5.8). The CORDIC algorithm is highly suitable for FPGA mechanization because it is dominated by additions and subtractions. These functions are very efficiently implemented by FPGA technology and require only *N*/2 logic slices for an *N*-bit adder/sub tractor.

A CORDIC based NCO implementation is shown in Figure 24. CORDIC (COordinate Rotation DIgital Computer) algorithm is an iterative method which performs the rapid coordinates rotation by arbitrary angles in digital plane. It uses only shifts and adds for performing the coordinates transformations.

The CORDIC plays an important role in digital computing. The basic idea of the CORDIC system is that the trigonometric functions of sine, cosine and phase to any desired precision can be calculated by a rotation of a vector through successive smaller angles. If a vector v with the components (x, y) is to be rotated by an angle ϕ, the new values of x' and y' are [11]:

$$\begin{aligned} x' &= x\cos(\phi) - y\sin(\phi) \\ y' &= x\sin(\phi) + y\sin(\phi) \end{aligned} \tag{5.7}$$

The equation can be re-written as:

$$\begin{aligned} x' &= \cos(\phi)[x - y\tan(\phi) \\ & s(\phi)[y + x\tan(\phi) \end{aligned} \tag{5.8}$$

The multiplication by the tangent term can be avoided, if the rotation angles $\tan(\phi)$ are restricted so that $\tan(\phi) = \pm 2^{-i}$. In digital hardware, this indicates a simple shift operation which simplifies the hardware. The arbitrary angles of rotation ϕ are decomposed into sums and differences of smaller angles ϕ_i [11]. The summary of computation which was applied for our CORDIC based NCO design is shown in Table 2.

Table 2. Angle Value & Shift Sequences of CORDIC Based NCO

Angles $\phi_i = \arctan(2^{-i})$	Shift sequence (i)
0, 15, 29, 44, 59, 74, 88, 2, 26, 41, 56, 71, 86	∞, 2, 1, 0, -1, -2, -5, 2, 1, 0, -1, -2, -4

The CORDIC based NCO (in Fig. 24) computes the sine and cosine of the input phase value by iteratively shifting the phase angle to estimate the Cartesian coordinate values for the input angle. At the end of the CORDIC iteration, the x and y coordinates for a given angle represent the sine and cosine of that angle respectively. The unrolled processor (Shown in Fig. 24) can also be converted to a bit serial design. Each adder sub tractor is replaced by a serial adder/sub tractor, separated by w bit shift registers. The shift registers are necessary to extract the sign of the y or z element before the fast bits (LSBs) reach the next adder/sub tractors [11].

5.8 Xilinx ROM Based NCO

An NCO using Xilinx ROM (Shown in Fig. 26) has been implemented by instantiating a 1024X1 ROM [33] shown in Figure 25 from Xilinx library instead of behavioral code of LUT.

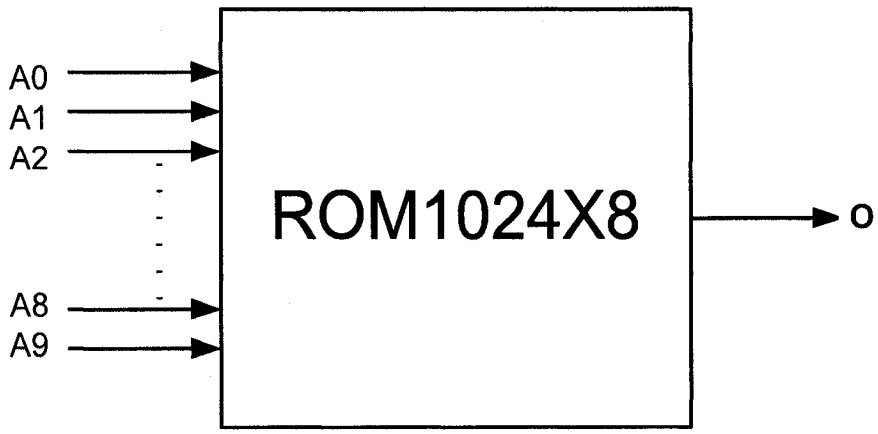

Figure 25. Xilinx ROM 1024X1 [32]

It is a 1024-word by 1-bit read-only memory. The data output, O (in Fig. 25) reflects the word selected by the 10-bit address which is A_9 to A_0.

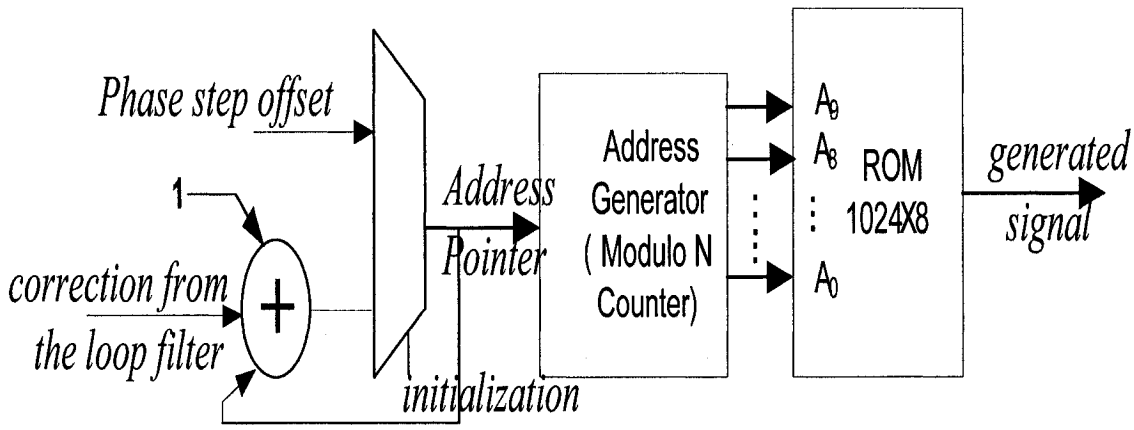

Figure 26. Xilinx ROM Based NCO with Modulo N Counter

The Xilinx ROM based NCO is shown in Figure 26. It consists of two parts: automated Address Generator (which is a Modulo *N* Counter, where $N = Fs / Fc$) and a ROM from Xilinx library.

This configuration gives us the benefit of scaling the Xilinx ROM's address size because of the automated address generation scheme via Modulo *N* counter for any *Fs / Fc.* The values of the Xilinx ROM are resolved by Equation (*5.1*). In the Xilinx ROM based NCO, we are using a large ROM, which is 1024X8, as the larger ROM length (address range), the finer the frequency resolution.

Table 3. Xilinx ROM1024X8 Initialization (When Fs = 50 MHz)

	ROM Block # (0..7)	ROM Contents
ROM1024X8 Using Xilinx ROM1024X1	ROM0(ROM1024X1)	EA317600
	ROM1(ROM1024X1)	CD1CF300
	ROM2(ROM1024X1)	4518A200
	ROM3(ROM1024X1)	000CF300
	ROM4(ROM1024X1)	88047300
	ROM5(ROM1024X1)	5059F600
	ROM6(ROM1024X1)	306EF100
	ROM7(ROM1024X1)	FF700000

As our output data bus is 8-bit wide, we designed 1024X8 ROM using Xilinx ROM1024X1. In our design, the addresses ($A_9 - A_0$) are specified by the *pointer* value and the output (*O*) is defined by the *data_fb*. The addresses generation of

the Xilinx ROM can be done using the Modulo N counter which depends on the Sampling Frequency (*Fs*) and the Carrier Frequency (*Fc*). The benefit of the Modulo N counter is the control of the frequency granularity. ROM is initialized to a known value during configuration with the INIT value parameter.

INIT value parameter is used only for the purpose of the synthesis and *defparam* parameter is for simulation [34]. The values have been calculated by using the equation (5.1). Those values are consisted of eight hexadecimal digits that are written into the ROM from the most-significant digit to the least-significant digit. An error occurs if the INIT value is not specified [34]. The initialization of a Xilinx ROM1024X8 is shown in Table 3.

5.9 Summary

This chapter focused on the methodology of different DCSs datapath implementation along with the different implementations of loop filter and NCO which increased the performance of our DCS. In the following chapter, synthesis and emulation of different DCS implementations will be discussed which target the Xilinx VirtexII Pro FPGA board.

Chapter 6

Synthesis and Emulation

Synthesis is an important stage in the development of IP, occurring after high-quality verification has taken place. The synthesis process provides quantitative information relating to the timing, area, and power requirements of designs. The following sections address the concept of synthesis in further detail and examine the synthesis phase in the development of the digital carrier synchronization (DCS) using different NCO and loop filter.

6.1 Space Exploration via Synthesis Process

The space analysis played a major factor in order to bias our verdict more towards CORDIC based implementation in the context to design a DCS Core for FPGA platform. DCS using CORDIC based NCO is appropriate when it will be used as on-chip device or as portable device where, area is more important than performance. But if we consider the other performances, we can observe that DCS using Xilinx ROM based NCO is more suitable.

6.1.1 Comparison of Different NCOs

The synthesis process results in a physical representation of the behavioral RTL, which is coded in the implementation process. This determines whether the code

can be translated to digital logic gates and whether the realized design meets the specified requirements in terms of timing, area, and power.

Table 4. Performance Evaluation Using Xilinx

Xilinx Device xc2vp50 package ff1152 speed -6	Report Using Xilinx ISE 8.1i					
	LUT Based NCO	CORDIC Based NCO	Xilinx ROM Based NCO	DCS Using LUT	DCS Using CORDIC	DCS Using Xilinx ROM
4-inputsLUT	195	84	56	411	300	275
Slices	120	45	29	239	164	152
Gate Count	1070	686	885	4113	3194	3228
Time Delay(ns)	3.818	3.148	3.478	5.471	6.141	4.892

The Verilog HDL model represented the actual design specification in order to serve the purpose of defining and verifying the behavior of the system.

Table 5. Performance Evaluation Using Synopsys

Xilinx Device xc2vp50 package ff1152 speed -6	Report Using Synopsys Design Compiler					
	LUT Based NCO	CORDIC Based NCO	Xilinx ROM Based NCO	DCS Using LUT	DCS Using CORDIC	DCS Using Xilinx ROM
Total Cell Area (μm^2)	241	84.5	117	588	431	463
Power (mW)	62.26	30.94	24.55	95.47	93.38	86.09

This procedure enabled us to compare the performance and the actual hardware implementation cost for LUT, Xilinx ROM and CORDIC (Table 4 and 5) based NCO for DCS in terms of the logic gates, or even later silicon implementations.

By means of the flexible HDL implementation as discussed above the area, power and the performance were investigated. The timing constraints were set to synthesize the design using *Fc* = 2.048MHz which was the design clock speed for synthesis. From Table 4 and Table 5, it is easily visible that Xilinx ROM based NCO implementation is the most suitable choice between the three methods when considered at the modular implementation level.

On the contrary, if a top level implementation of DCS is considered we can observe that the significance of this difference becomes much more relaxed. For example, the ratio of LUT count (Table 4) between single LUT based NCO vs. CORDIC Based NCO is 195/84 = 2.32, whereas in the DCS implementation this ratio becomes 411/300 = 1.37. On the other hand, single LUT count (Table 4) between CORDIC based NCO vs. Xilinx ROM based NCO is 84/56 = 1.5, while in the DCS implementation this ratio becomes 300/275 = 1.09. The power consumption (Table 5) also depicts the same result, 62.26/30.94 = 2.01 vs. 95.47/93.38 = 1.022 and 30.94/24.55 = 1.26 vs. 93.38/86.09 = 1.085. However, only in the gate count analysis, CORDIC based NCO proved its efficiency i.e. 1070/686 = 1.56 vs. 4113/3194 = 1.29 and 686/885 = 0.78 vs. 3194/3228 = 0.99.

6.1.2 DCS Synthesis Using Various Configuration of NCO and LF

The area usage of the FPGA implementation of different DCS implementations is displayed in Table 6. It is easily visible that the configuration involving second order LF in both scenarios occupies more gate, slices and LUT. Therefore, synthesis wise DCS using Xilinx ROM based NCO with first order LF seems to lucrative choice. Finally about synthesis results we can comment that as our DCS was not to be implemented on chip, therefore area was not a concern of our design. This was the reason behind the non-usage of Xilinx ISE's area constraint. In the latter chapter it will be discussed that our design goal was served better with DCS configuration using Xilinx ROM based NCO and second order LF, as it provided a faster locking time and better tracking frequency.

Table 6. Synthesis Report of Different DCS Implementations

Xilinx Device xc2vp50 package ff1152 speed -6	DCS using LUT based NCO		DCS using Xilinx ROM based NCO	
	1st Order LF	2nd Order LF	1st Order LF	2nd Order LF
4-input LUT	411	663	275	412
Slices	239	379	152	298
Gate Count	4113	7736	3228	6121

6.2 Emulation

6.2.1 Introduction

The aim of our emulation is to precisely model a system in an environment similar to that in which a fabricated design would operate. Since the emulation

process usually employs clocks operating at realistic frequencies, systems can operate close to full capacity. Design functionality can be verified at a much faster rate than that attainable through design simulation. The emulation of the design enabled full functionality to be verified; and the display showed only a small section of the expected output of the system. The emulation process in this case prevented the undertaking of highly-expensive fabrication of a system which did not function as expected.

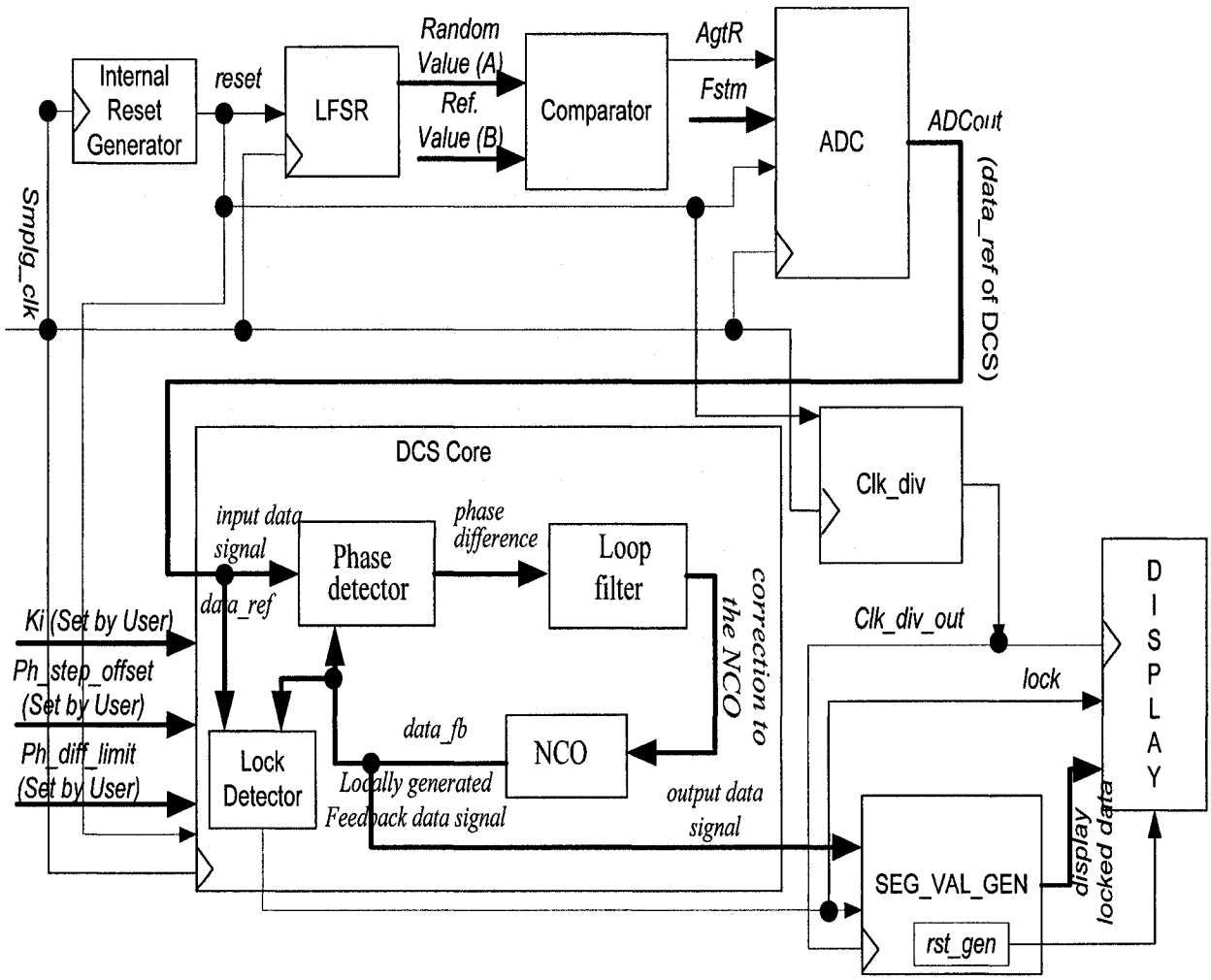

Figure 27. Emulation Environment for DCS

The benefits associated with emulation, both in terms of the potential quality of verification and in terms of the author's personal development, lead to the emulation phase in the development of the DCS Module being highlighted being prioritized over a number of other development stages. This Section considers emulation and the emulation phase of the DCS.

6.2.2 The Emulation Process

Figure 27 shows the data path of the full emulation environment. The emulation process shown in Figure 27 can be considered to be relatively opaque, in that it can be difficult to determine whether systems are operating correctly, why a system is not operating correctly, and sometimes whether a design has even been downloaded to the FPGA correctly. For this reason the strategic, methodical approach, which should be applied to all areas of digital design, is particularly crucial during this phase.

The following section addresses the emulation process, considering the setup employed in the emulation phase of the Digital Carrier Synchronization Module.

6.2.3 Synthesized Emulation Environment setup

After the development of a suitable system, for emulation with the board provided, the system's interface must be *mapped* to the physical pins of the FPGA. The required input-output pins of the Virtex-II Pro FPGA board are identified and then mapped to the equivalent interfaces in the system using a UCF (User Constraint File). The synthesis process takes at its input: RTL,

memory and instantiated components, the UCF, and information on the setup of the board and FPGA. The output of the process is a bit-file which can be downloaded to and interpreted by the FPGA.

For downloading the bit-files to the Virtex-II Pro FPGA, the most suitable method being the use of a CompactFlash™ storage device alongside the embedded SystemAce™ controller; this enables the storage of multiple bit-files and permits the selection of the design and re-loading of the FPGA without the use of the PC. Since the CompactFlash™ storage devices are not available for use, the only option for the emulation of the Carrier Synchronization project is the use of a JTAG interface on the Virtex-II Pro FPGA board alongside a MultiLINX™ 'cable'. The speed of configuration of the FPGA using the JTAG interface is poor in comparison to the use of CompactFlash™ storage devices, but this form of interface is still used and is adequate for the task being undertaken. If the output of the Reference Design is to be directly compared with the output from the Emulation System, then the use of a CompactFlash™ storage device would be a great advantage to the Carrier Synchronization project, significantly reducing the time required to switch from one design to the other. The PC is connected to the board via the MultiLINX™ cable and the JTAG interface, allowing the bit-file to be downloaded onto the FPGA. The FPGA status LED's help to show when data is being downloaded to the FPGA.

Both the Reference Design and the Emulation System are adapted to include control of the User LED's with one of the User DIP Switches. The control of these LED's is very important to the emulation process as their correct operation

provided indication that the design has been successfully downloaded and mapped to the FPGA. The manner in which the LED's are assigned for the Reference Design is made purposely discernible from that for the Emulation System, so that it is clear which design has been downloaded. The LED's can also be assigned synchronously to confirm the operation of the system clocks. The system reset of the designs is assigned to the other DIP Switch, since this allows the system to be manually reset at any point.

6.2.4 Design Implementation

The design is generated by inputting the circuit into Xilinx ISE 8.1i [17], which is done by using Verilog HDL. Once the design is implemented, the Synthesis Tools in the ISE are used to generate the bit stream files for configuring the Xilinx FPGA chip. However, it is important to perform simulations before configuring the chip, since errors often exist, even though the synthesis process is successful. The simulation process allows the debugging of both the invisible errors at the design stage and the technical problems caused by the characteristics of the FPGA, such as delays and FPGA design constraints.

A. User Constraints and Synthesis Flow

Design implementation is the process of translating, mapping, placing, routing, and generating a BIT file for the design [35]. In order to implement the designs, a UCF (User Constraints File) has to be created. The UCF is to provide timing constraints through the Constraint Editor, and pin location constraints through PACE (Pinout Area Constraints Editor). There are 3 main stages in the design implementation:

1. Translating: to merge all of the input netlists as well as design constraint information into a Xilinx database file.

2. Mapping: to map a logical design to a Xilinx FPGA.

3. Placing and routing: to place and route the FPGA, and produce output for the bit stream generator.

User constraints were set relating timing and package pin assignments.

Timing Constraint was used in order to set up the clock period of 500ns on the FPGA global clock pin. As our design clock was running at 2.048 MHz frequency.

Package Pins Assignment was used in order to download the design into our target device. LED pin was used to check the lock signal and seven segment display pins for displaying the locked data.

The main challenge in the process was to deal with the high frequency clock (2.048 MHz) in order to be visible the locked data on the seven-segment display. Therefore, the clock divider and the display module of the emulation environment enabled the locked data and the lock signal to visible through the FPGA display.

The following section discusses the different components of the emulation environment for the DCS project.

6.2.4.1 Internal Reset Generator through an FSM

The internal reset signal generator (Shown in Fig. 28) is composed of an internal Finite State machine (FSM).

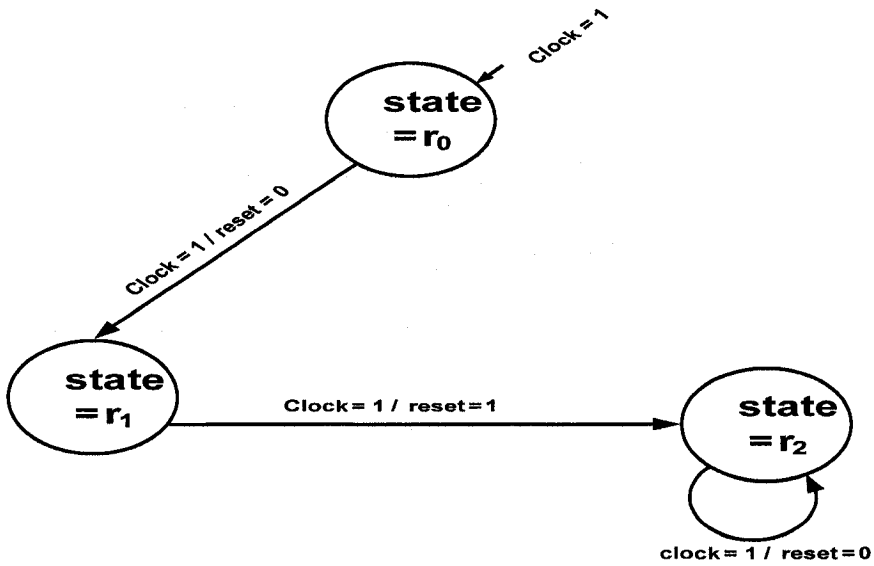

Figure 28. FSM for internal Reset signal generator

The state transition was based on the positive edge of the clock. There were three states namely: r_0, r_1 and r_2. At initial state r_0, the value of the reset signal was set to zero. The following positive clock edge, the state machine transits to r_1 where the value of the reset signal was set to 1. Afterwards, in the next rising edge of the clock, the value of the reset signal was assigned to 0. The r_2 state ensured that the reset signal was set to 0. Since the emulation environment and the DCS needed an active high reset to initialize, the scenario at state r_1 took care of it. Therefore, with the reset signal asserted low at state r_2, enabled the datapath to resume its normal operation.

6.2.4.2 Linear Feedback Shift Register (LFSR)

We have implemented an 8-bit LFSR (Shown in Fig.29), which is also known as Linear Feedback Shift Register, in order for generating the pseudo random

numbers for the digital comparator producing the reference signal *data_ref* for the DCS.

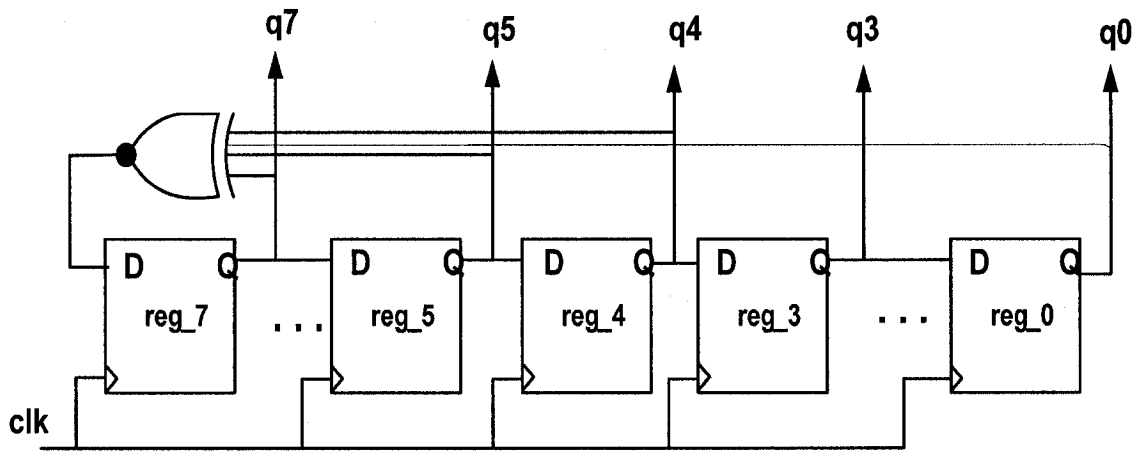

Figure 29. An 8-bit Pseudo Random Generator [36]

In theory, an n-bit LFSR can generate a (2^n-1)-bit long pseudo random sequence before repeating. In our case, the width of the LFSR is 8-bit. So, it can generate the number from 0 to 255. This 8-bit random pattern generator generates up to 90% coverage by simulation out of 2^8 = 256 before it repeats. The characteristic polynomial equation of this LFSR is:

$$X(n) = X^7 + X^5 + X^4 + X^3 + 1$$

6.2.4.3 Digital Magnitude Comparator

A digital magnitude comparator is a combinational circuit that compares two digital numbers. Figure 30 shows an 8-bit digital magnitude comparator,

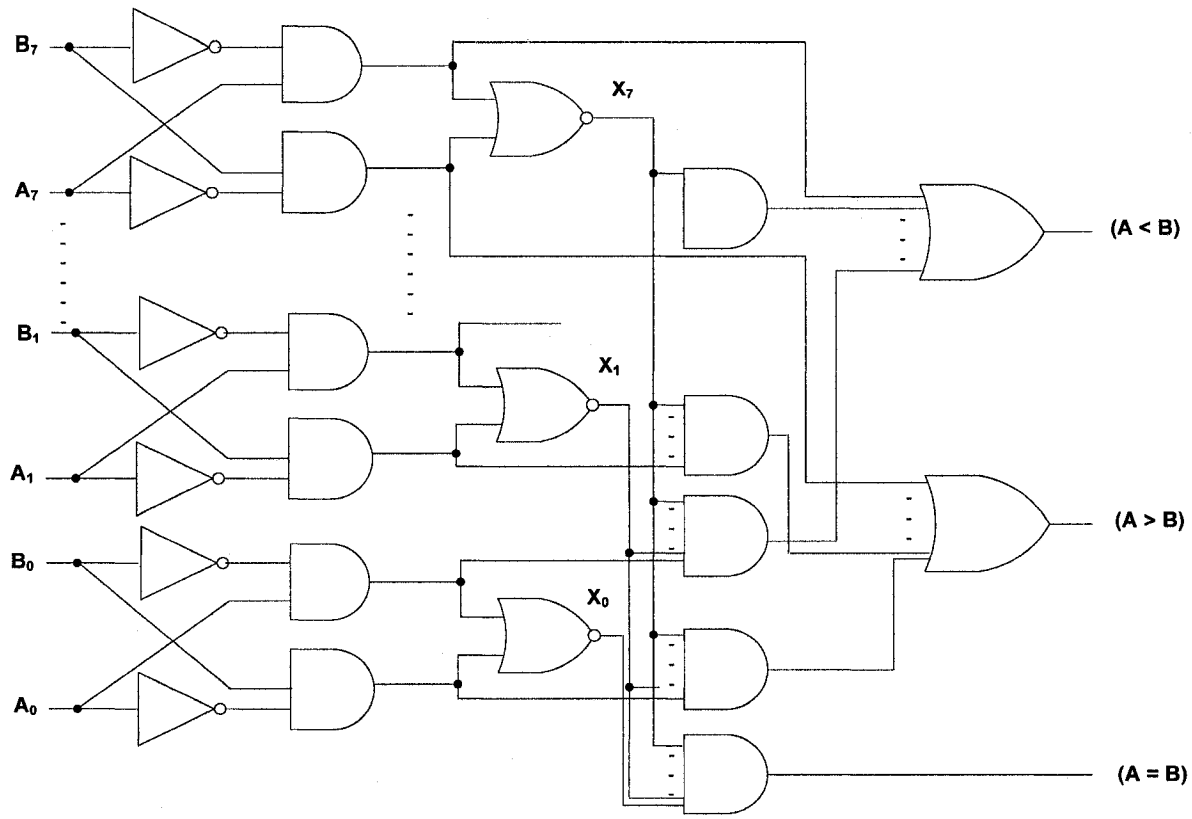

Figure 30. 8-Bit Digital Magnitude Comparator [37]

which compares the relative magnitudes of two digital numbers. We are considering two numbers, *A* and *B*, with 8-digit each. This comparator outputs any one of the following control signal, $A = B$, $A > B$ or $A < B$. In this case we are only considering the output signal when $A > B$. To determine if *A* is greater than *B*, we inspect the relative magnitudes of pairs of significant digits starting from the most significant position. If two digits are equal, we compare the next lower significant pair of digits. This comparison continues until a pair of unequal digits is reached. If the corresponding digit of *A* is 1 and of *B* is 0, we conclude that $A > B$. The sequential comparison can be expressed logically by the following Boolean function [37].

$$(A > B) = A_7 B'_7 + X_7 A_6 B'_6 + \ldots + X_7 X_6 X_5 X_4 X_3 X_2 A_1 B'_1 + X_7 X_6 X_5 X_4 X_3 X_2 X_1 A_0 B'_0$$

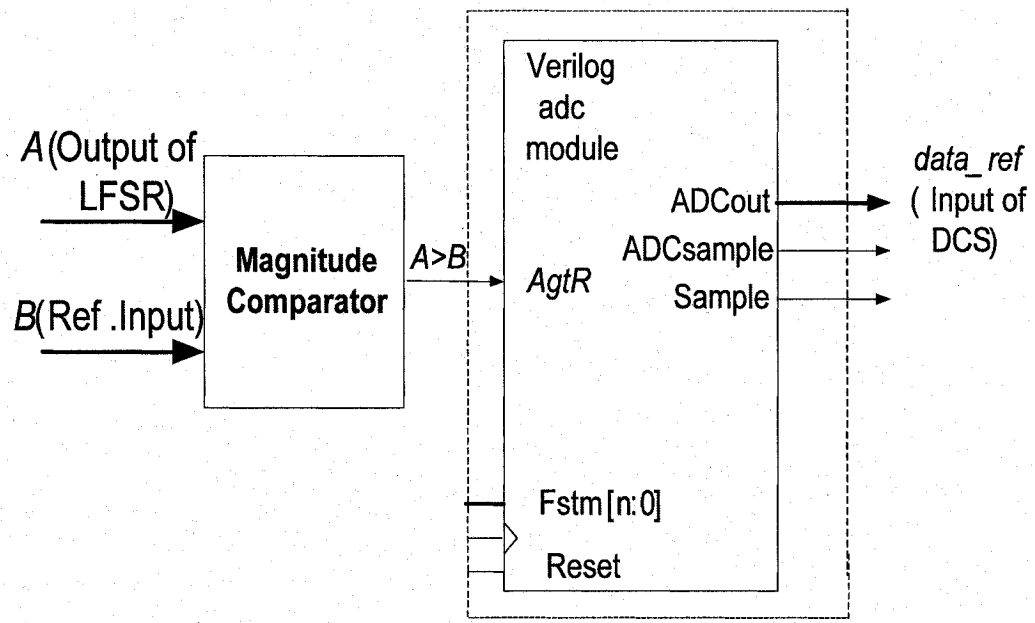

Figure 31. ADC with a Magnitude Comparator [4]

One of the inputs (e.g. *A*) of this magnitude comparator receives values from the LFSR and the other input (e.g. *B*) is fixed as a reference. We chose 8'h05 as a reference input in order to get the more number of coverage when *A* > *B*. Please note that the reference value of 8'h05 was acquired by simulating the testbench composed of LFSR and digital magnitude comparator.

For this project a magnitude comparator replaced the analog comparator, i.e. Op-Amp since the Analog to Digital Converter (ADC) needed an analog comparator as own in Figure 31. The ADC will be discussed in detail in the latter section.

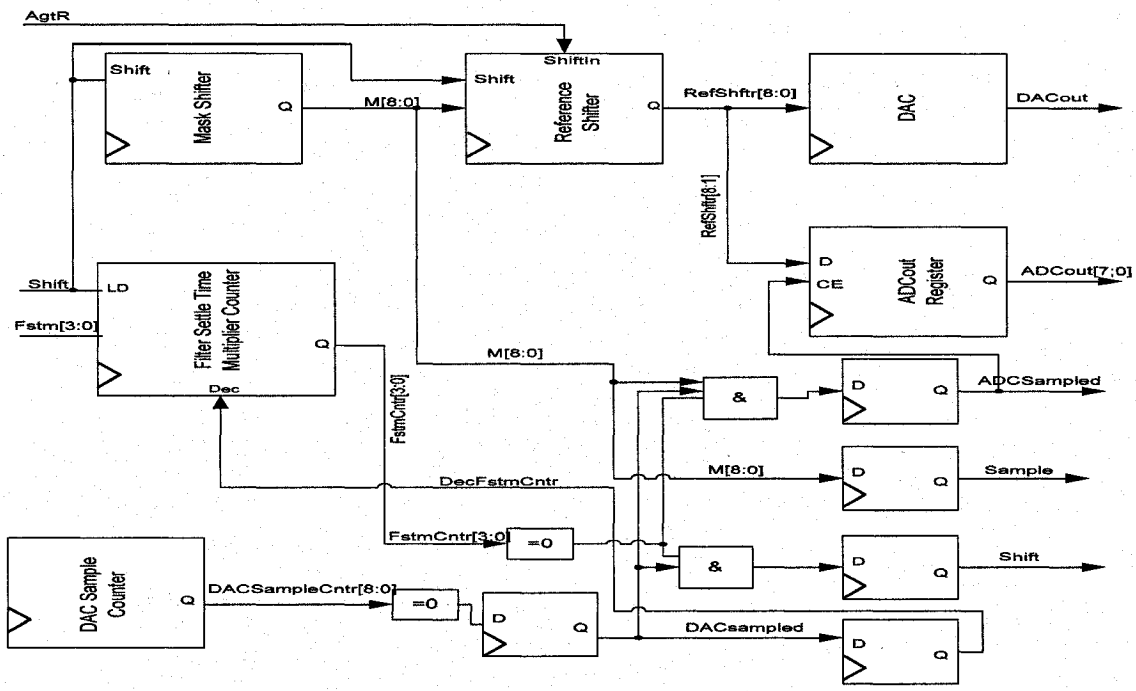

Figure 32. Datapath of Analog to Digital Converter [4]

The main goal of replacing the analog comparator with the digital magnitude comparator was to create a synthesizable emulation environment consisting of all digital signals. This magnitude comparator gives a control signal *AgtR* (which means if *A>Ref.*, value will be 1, otherwise value will be 0).

6.2.4.4 ADC (Analog to Digital Converter)

This Analog to Digital Converter (ADC) shown in Figure 32 [4] receives a control signal (*AgtR* in Fig. 31) from the digital magnitude comparator and outputs the 8-bit data, *ADCout*, which is used as a reference signal (*data_ref*) for the Digital Carrier Synchronization (DCS). This ADC is useful only on signals that are changing at a fairly low rate, i.e. *data_ref* signal in our case. Since our DCS model requires an unchanged *data_ref* for several clock cycles in order to

synchronize with *data_fb* signal, this ADC served our *data_ref* generation quite comprehensively.

This ADC, implemented in a single verilog module, was taken from the Xilinx Inc [4]. The sample rate of this ADC may be expressed as follows:

$$ADC_{SR} = f_{clk} /(2^{(MSBI+1)} * (Fstm+1) * (MSBI+1)) samples / \sec ond$$

ADC outputs *ADCout* which is used as a data reference signal for the digital carrier synchronizer (DCS). Our goal is to verify the timestamp when *data_ref* and *data_fb* (output of the NCO) are equal. This event is defined as locking time of the DCS.

6.2.4.5 Clock Divider (clk_div)

A clock divider circuit was used in the emulation environment path shown in Figure 27. It has one input clock which is running on 2.048 MHz frequency and it provides an output clock of 30 Hz frequency. The clock divider circuit was controlled by a counter which generated the output clock. The counter incremented based on the positive edge of the input clock (2.048MHz). The value of output clock was initially set to 0 by the counter. When the count value became 20h'10AAB, the output clock was asserted to high. This ensured that the output clock of the clock divider circuit had a period of 30Hz. This dividing was performed to lower the input clock frequency in order for the locked data to be visible through human eyes.

6.2.4.6 Display

The display module displayed the locked data onto the seven segment display. The seven segment display used the output clock from the clock divider as its input and displayed the locked data value. A summarized emulation results of different DCS implementations by targeting Xilinx VirtexII-Pro FPGA are shown in Table 7.

Table 7. Summarized Emulation Results of Different DCS Implementations

Xilinx Device xc2vp50 package ff1152 speed -6	DCS using LUT based NCO		DCS using Xilinx ROM based NCO	
	1st Order LF	2nd Order LF	1st Order LF	2nd Order LF
Checking Locked Signal	Locked (When Fs = 73MHz)	Locked (When Fs = 85MHz)	Locked (When Fs = 73MHz)	Locked (When Fs = 95MHz)
Locked Data (Hex)	66	D5	29	79
Phase Angle (Deg.)	53.43	19.79	18.83	72.32

6.3 Summary

This chapter focused on the synthesis results and an emulation environment of different DCS implementations. In the following chapter the performance evaluation of different DCS implementations will be discussed in the context of locking results and tracking frequencies by applying simulation technique.

Chapter 7

Simulation Results and Performance Evaluation of Different DCS Implementations

7.1 Introduction

We have followed a top-down design flow to finish the FPGA based digital carrier synchronizer. The hardware design was done in using synthesizable Verilog-HDL codes. After each individual module was compiled and tested, they were integrated and compiled together. Furthermore, the final design was targeted to an FPGA device provided by Xilinx Inc. where a VirtexII-Pro (XC2VP50-FF1152 and Speed Grade -6) device was chosen to fit the whole design. The master clock of our design is chosen to 2.048 MHz whether the board has the clock frequency of 50 MHz. Initially individual modules of the DCS were simulated using VCS from Synopsys and Active HDL from Aldec. In so doing, we wrote different testbenches for testing the operation of each individual hardware module and the whole system. We utilized the same emulation environment (i.e. generating stimulus) in order to verify the different DCS implementations. We implemented the DCS using first and second order loop filter and two different NCOs for comparing the performance of DCS.

7.2 Simulation Environment for DCS

By applying stimulus and simulating the design, the designer can be sure the correct functionality of the design is achieved. This design used a DCS and test bench to illustrate the basic elements of a Verilog simulation.

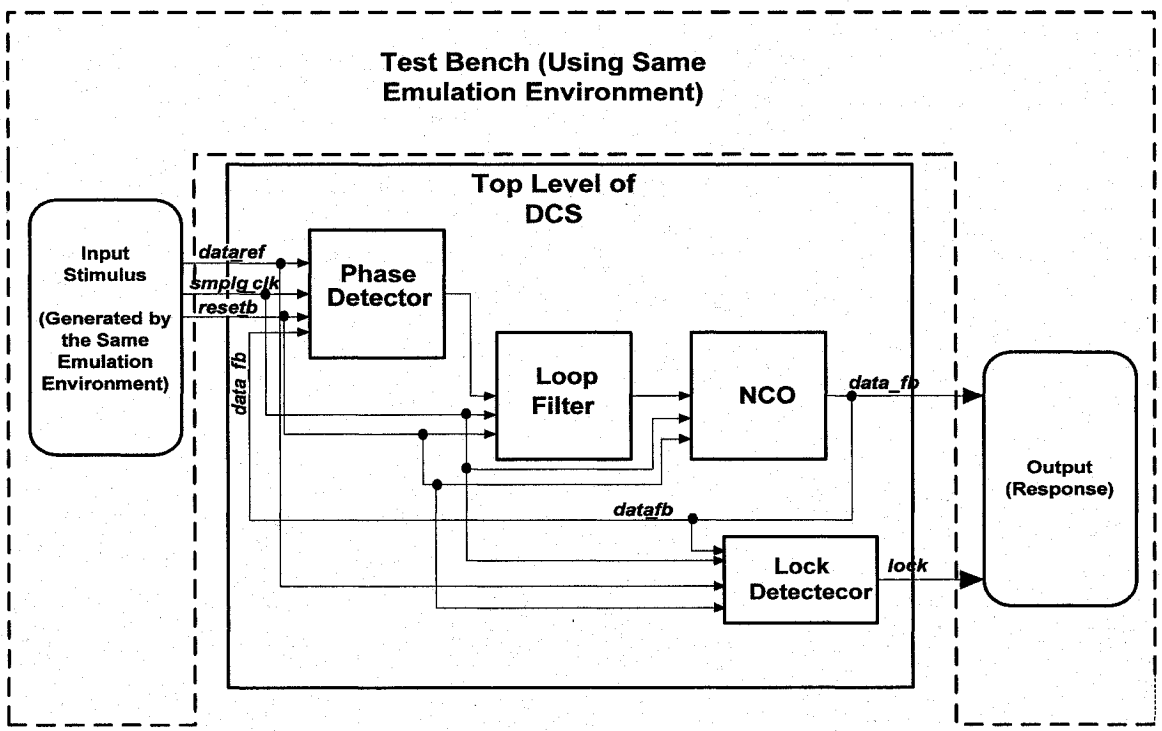

Figure 33. Simulation Environment for DCS

The DCS (which is our DUT) is instantiated into the test bench, and always and initial blocks apply the stimulus to the inputs to the design. The outputs of the design are printed to the screen, and can be captured in a waveform viewer as the simulation runs to monitor the results. We employed the same emulation environment in order to construct the testbenches (i.e. Simulation Environment) of our different DCS implementations which were our DUT (Shown in Fig. 33).

The following sections go into detail on each part of the test bench and its function.

7.2.1 Instantiations

The test bench applied stimulus to the DCS. To do this the DCS must be instantiated in the test bench, which is the equivalent to placing a component on a schematic. Stimulus generator was composed of the following components:

- Reset generator using a counter
- LFSR (Linear Feedback Shift Register) for random value generation
- Digital magnitude Comparator
- Analog to Digital Converter (ADC)

```
dcs_stimulus U1 (
                .ClkIn(smplg_clk)
                .rst_b(rst_b),
                .B(B),
                .Fstm(Fstm),
                .ADCout(ADCout),
                .ADCsampled(ADCsampled),
                .Sample(Sample)
                 )
```

Figure 34. Stimulus Module Instantiation

The wrapper was created using the above four components and was instantiated inside the testbench shown in Figure 34. *dcs_stimulus* is used to place an instance of the input stimulus in the test bench with the instance name *U1*. In

between the outer parenthesis are the signals connecting up to *U1*. The signals with a dot in front of them are the names for the signals inside the *dcs_stimulus* module, while the *wire* or *reg* they connect to in the test bench is next to the signal in parenthesis. For example, the clock to the DCS is called *ClkIn* in *dcs_stimulus*, but in the test bench *smplg_clk* is used, which now connects to *ClkIn* of *dcs_stimulus*. This type of instantiation is called "named instantiation" and allows the signals to be listed in any order, or even omitted when a module is instantiated.

7.2.2 Initial and Always Blocks

Always and initial blocks are two sequential control blocks that operate on *reg* types in Verilog simulation.

```
reg smplg_clk;

initial
begin
    $display ($time, "<<Starting the Simulation>>");
    smplg_clk = 0;       // at time 0
    #10000000 $finish; // at time 10000000 stop simulation
end

always
begin

   #250 smplg_clk= ~smplg_clk; // Every 250 ns toggle (invert)
                               // clock signal

end
```

Each *initial* and *always* block executes concurrently in every module at the start of simulation. Initial block starts executing sequentially at simulation time 0. Initial

blocks start executing sequentially at simulation time 0. Each initial and always block executes concurrently. The initial block (Shown above) starts by printing << Starting the Simulation >> to the screen, and initializes the *reg* type's *smplg_clk* to 0 at time 0. The initial block initializes the *smplg_clk* reg types at the beginning of simulation and the always block executes every 250ns starting at time index 0. Hence, the value of smplg_clk will invert from the initialized value (in initial block) of every 250ns. This causes a clock pulse to be generated on smplg_clk with a period of 500ns or a frequency of 2.048MHz which is the carrier frequency of our DCS implementation.

7.2.3 Printing Using $display During Simulation

As a simulation runs, it's important to include a printout to the screen to inform the designer on the status of the current simulation. The value a net or register holds at a certain time in the simulation may be important in debugging a function, so signals can also be printed. A snapshot of $display used for our DCS testbench shown below:

```
$display("\nData has Locked at time : %0t", $time);
$display("\ndata Ref value : %h", `data_ref);
$display("\ndata Fb value : %h", data_fb);
```

The characters found between the quotes will be printed to the screen followed by a carriage return. The locked time of DCS will be displayed when the value of *data_ref* and *data_fb* are equal. It also displays the value of *data_ref* and *data_fb*. In the following section

7.3 Simulation Results and Discussion

A snapshot of the DCS using LUT based NCO and first order LF is displayed in Figure 35. It shows the locking time of 11.735us which was the optimal locking for this implementation, when the sampling frequency was 72 MHz. It synchronized the data when the Locked data was 8'h66.

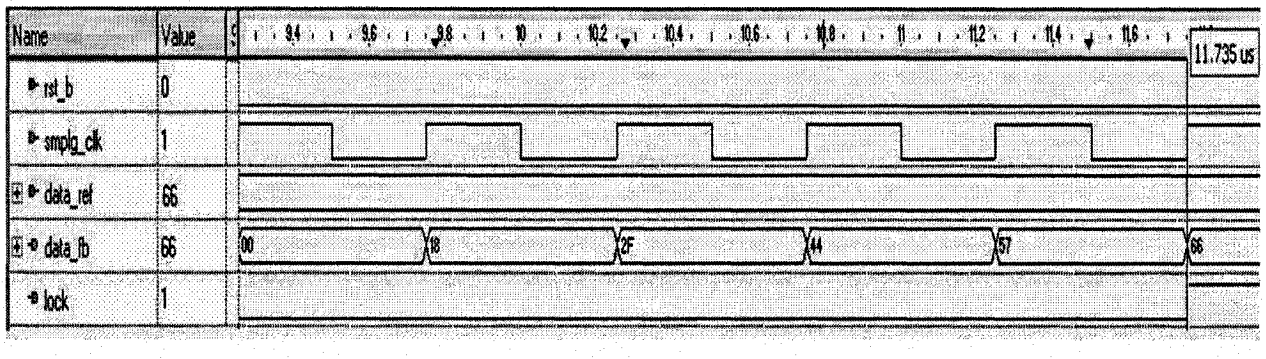

Figure 35. DCS Using LUT based NCO and 1st Order LF (When Fs = 72 MHz)

The locking time was stable until the sampling frequency changing to 75MHz. When the sampling frequency was 75 MHz, the locking time was 11.775us and the locked data was 8'h5C (Shown in Fig. 36).

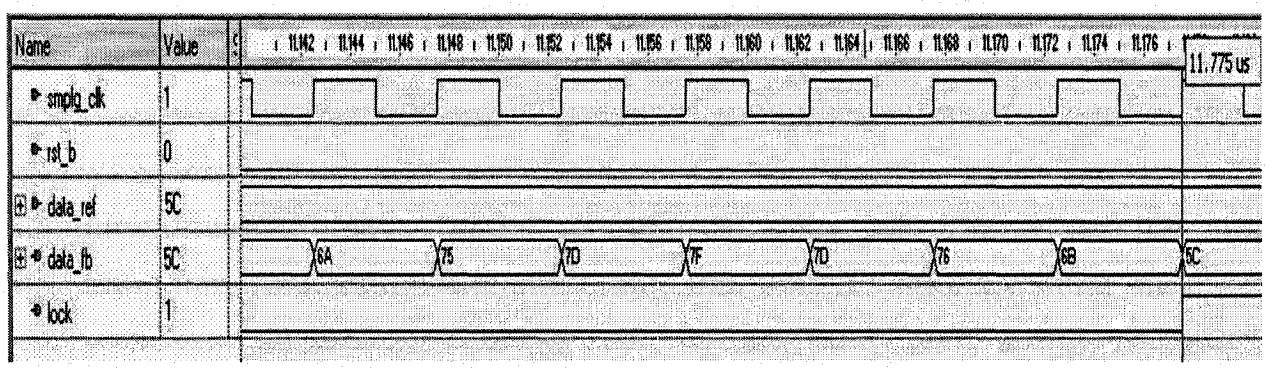

Figure 36. DCS Using LUT based NCO and 1st Order LF (When Fs = 75MHz)

Figure 37 shows the locking time of DCS using Xilinx ROM based NCO and first order LF.

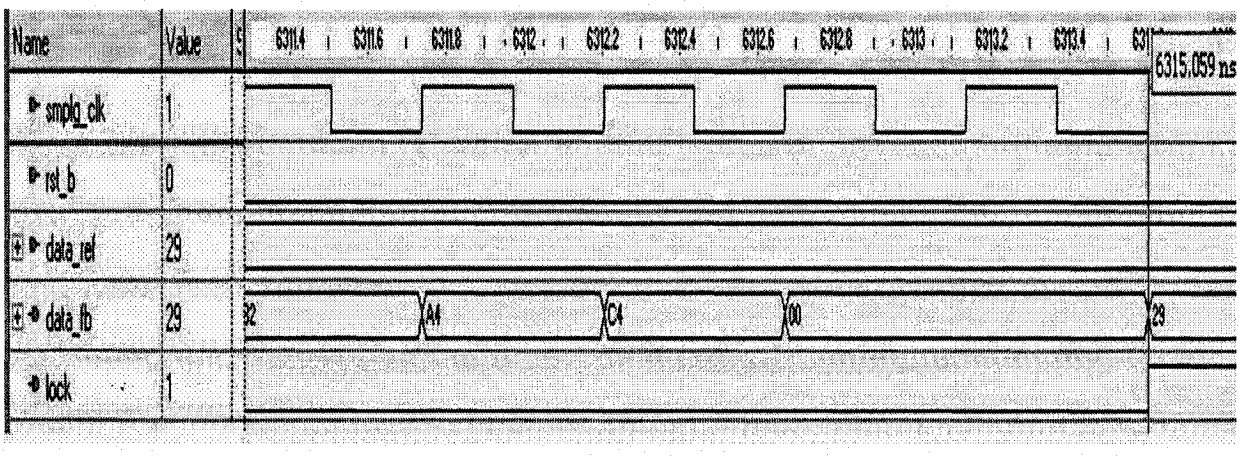

Figure 37. DCS Using Xilinx ROM based NCO and 1st Order LF (When Fs = 73 MHz)

It displayed the locking time of 6315.059ns, which is 6.315us, better than previous implementation, when the sampling frequency was 73MHz and the locked data was 8'h29.

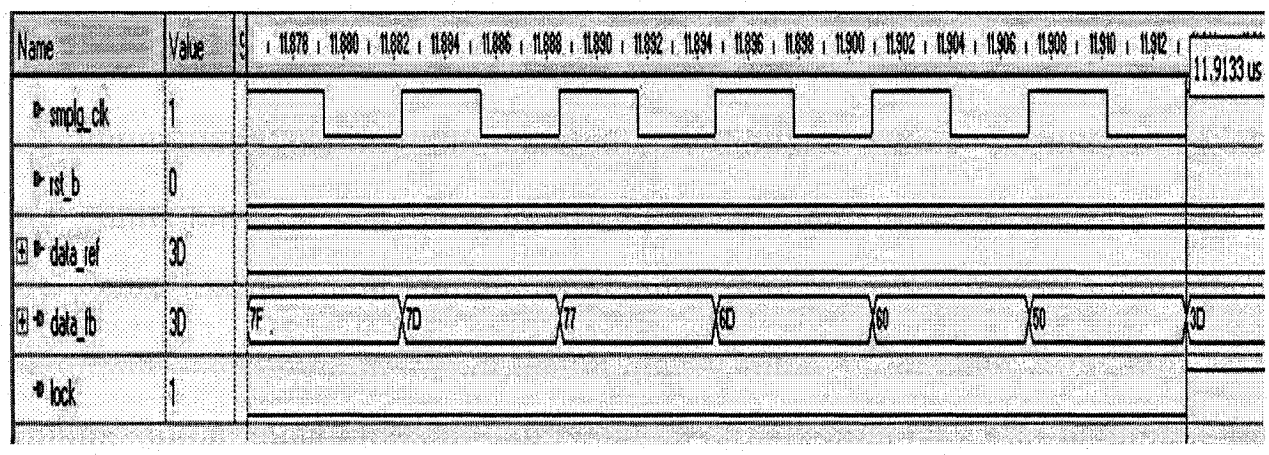

Figure 38. DCS Using Xilinx ROM based NCO and 1st Order LF (When Fs =77MHz)

When sampling frequency was 77MHz, DCS using Xilinx ROM based NCO and first order LF locked the data at 11.9133us (Shown in Fig. 38). The locking time was stable until the sampling frequency changing to 76MHz.

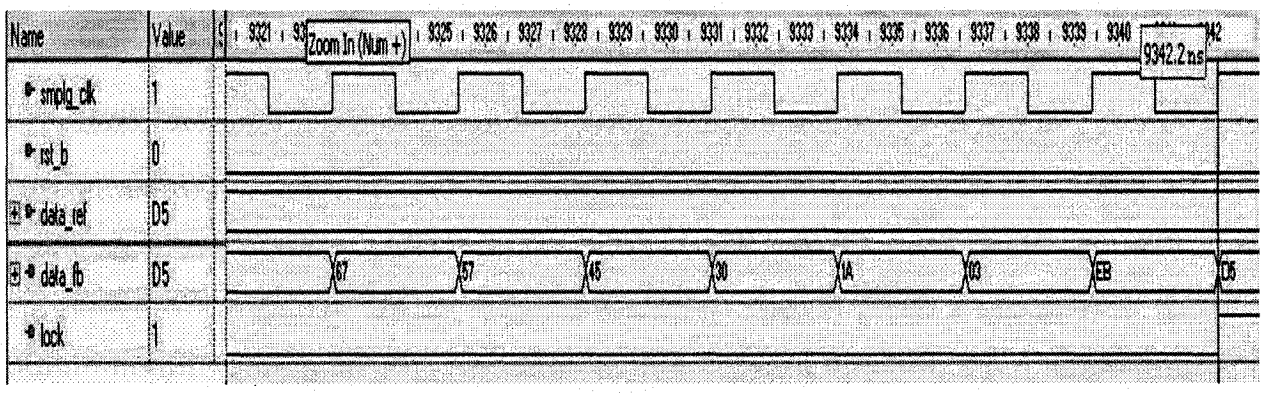

Figure 39. DCS Using LUT based NCO and 2nd Order LF (When Fs =85 MHz)

Figure 39 shows the locking time (9.342us) of DCS using LUT based NCO and second order LF which signifies better locking than DCS using LUT based NCO and first order LF but different sampling frequency (85MHz). It also shows that the locking data was 8'hD5.

The locking time of DCS using LUT based NCO and second order LF was stable until the sampling frequency changing to 88MHz. When the sampling frequency reached to 88MHz, the locking time changed to 9.4995us (Shown in Fig. 40).

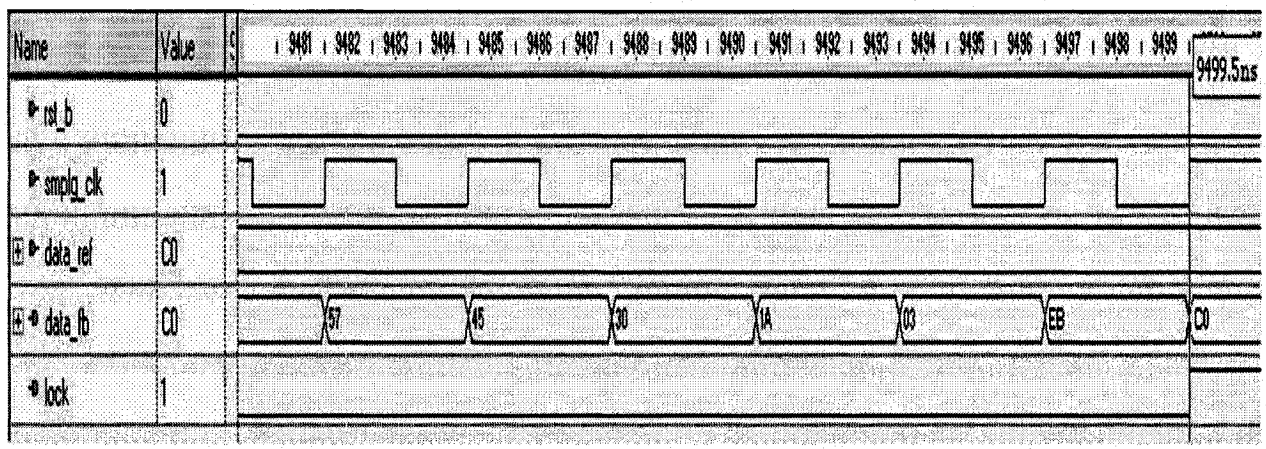

Figure 40. DCS Using LUT based NCO and 2nd Order LF (When Fs = 88 MHz)

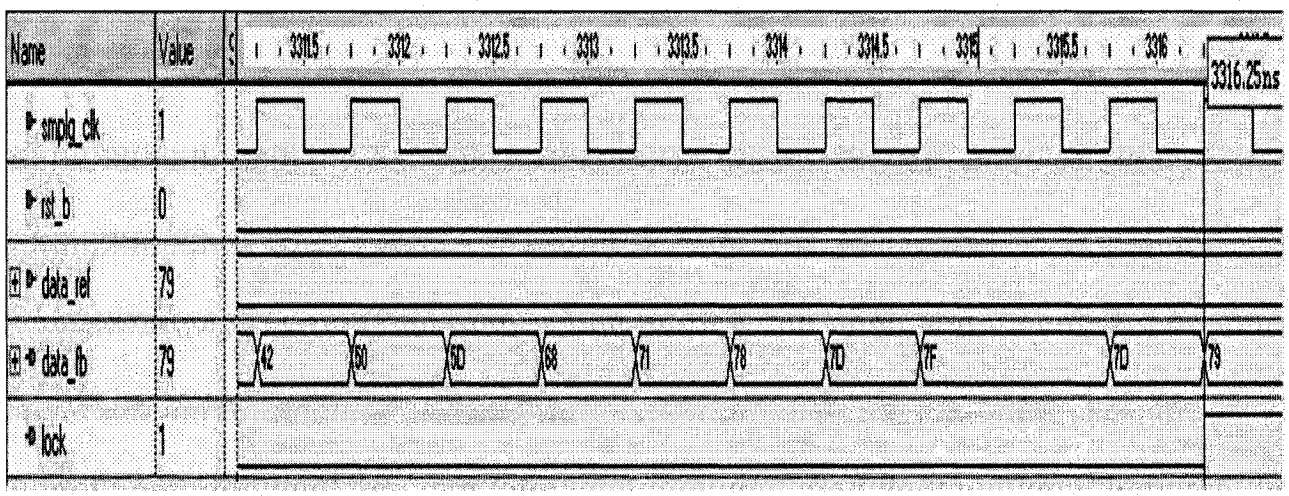

Figure 41. DCS Using Xilinx ROM based NCO and 2nd Order LF (When Fs = 95 MHz)

Figure 41 shows the locking time (3.31625us) of DCS using Xilinx ROM based NCO and second order LF, when the sampling frequency was 95MHz. It shows the optimal locking time. When it locked the data, the locked data was 8'h79.

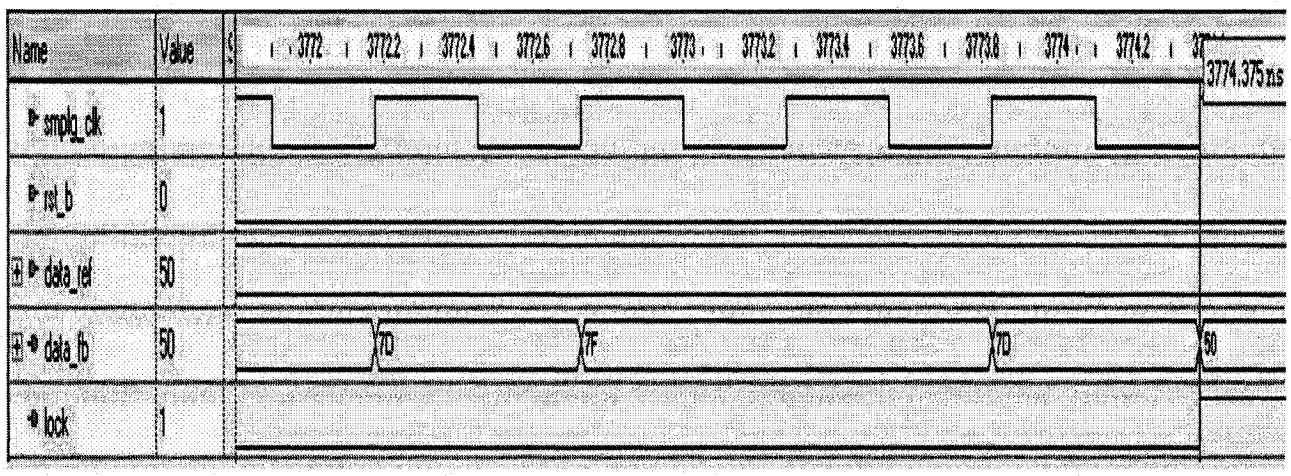

Figure 42. DCS Using Xilinx ROM based NCO and 2nd Order LF (when Fs = 105 MHz)

The locking time (3.774us) of DCS using Xilinx ROM based NCO and second order LF was changed when the sampling frequency reached to 105MHz (Shown in Fig. 42). The following section we will discuss the tracking frequency and locking time of different DCS implementations.

7.4 Tracking Range and Locking Time Analysis

The tracking range of the DCS is the range over which the loop is able to track and lock onto a carrier. A small phase error is usually desired and is considered to be the criterion of good tracking performance. If the error becomes so large that the NCO skips cycles, the loop is considered to have lost lock, even if momentarily [9].

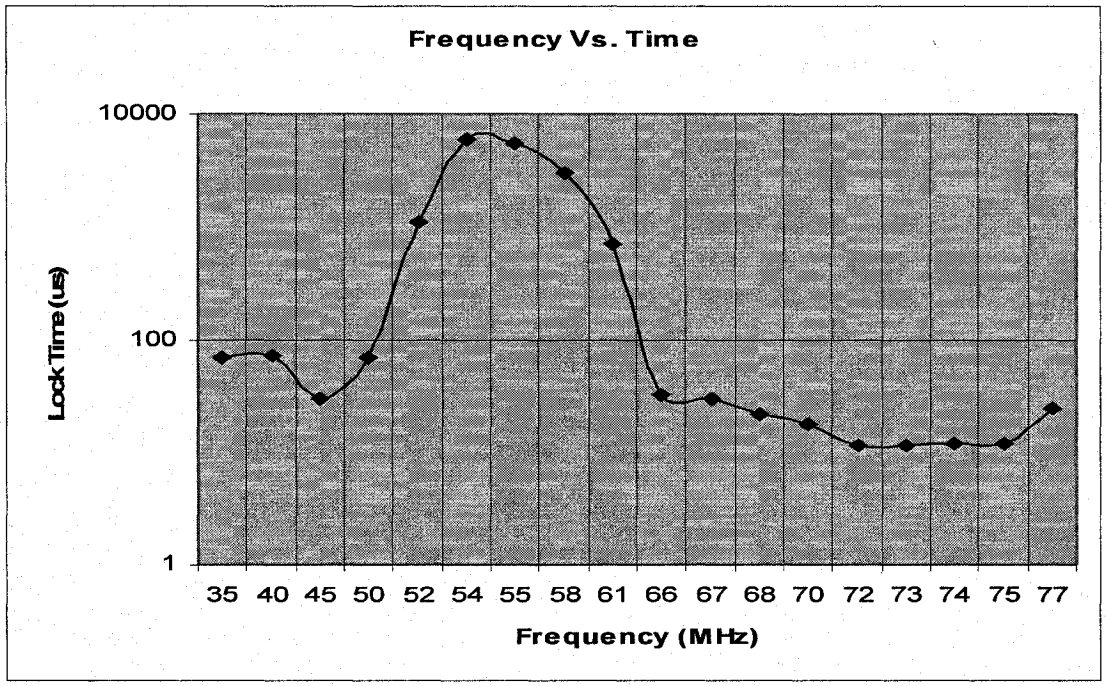

Figure 43. Tracking Frequency vs. Lock Time of DCS using LUT Based NCO and 1st Order LF

The important characteristics of DCS performance for the modem application are locking time and tracking frequency. Figure 43 - 47 shows the analysis of different DCS implementations in the context of tracking frequency and the locking time. The proposed DCS acquires lock with a reference frequency of 2.048 MHz. The tracking frequency range over which the DCS can maintain a lock is 35 – 110 MHz from the simulation results.

Simulation begins with an investigation of the stability of the closed-loop DCS configuration. The loop filter parameter values are obtained by loop stability analysis using MATLAB. Then we developed a simulation environment using a counter for internal reset generator, a LFSR for Pseudo Random Generator, and an ADC from Xilinx for providing the reference signal, *data_ref* to the DCS.

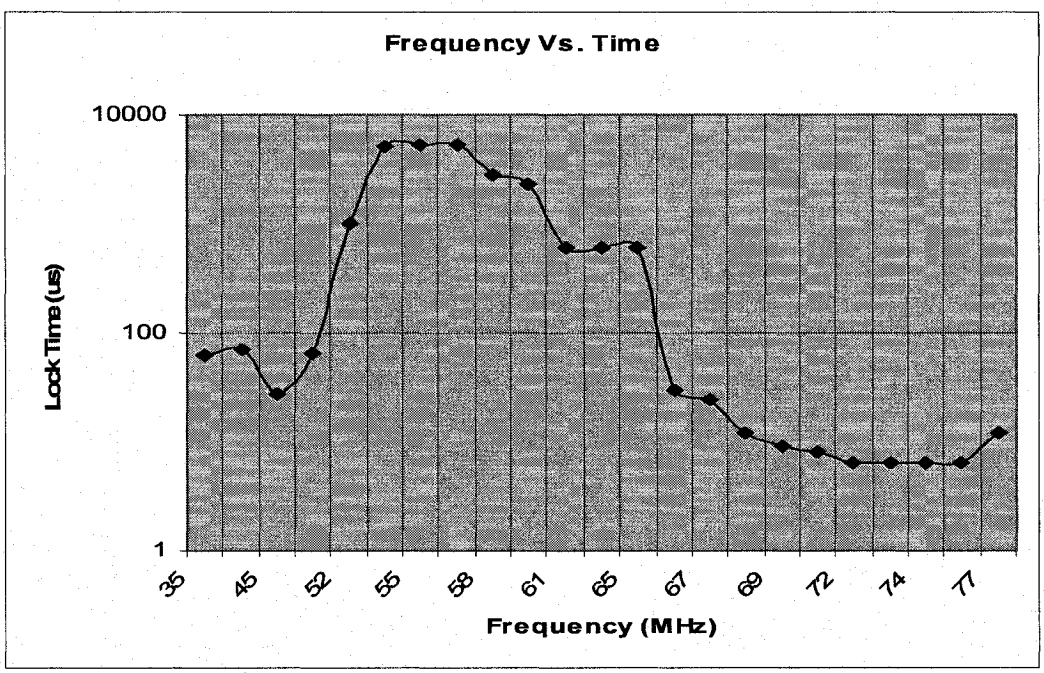

Figure 44. Tracking Frequency vs. Lock Time of DCS using Xilinx ROM based NCO and 1st Order LF

Initially, we observed that the lock time and tracking frequency of DCS with 1st order LF using LUT (Shown in Fig. 43) and Xilinx ROM based NCO (Shown in Fig. 44) and then we monitored the different locking times and tracking frequency of DCS with 2nd order LF using LUT and Xilinx ROM based NCO. Fig. 43, which is DCS with 1st order LF and LUT based NCO, shows that a steady state condition (or lock) is accomplished when the sampling frequency is about 75 MHz and the locking time is 11.735 *us*. Similarly for the DCS with first order LF and Xilinx ROM based NCO, which is shown in Figure 44, has same tracking frequency but a different locking time which is 9.325 *us*.

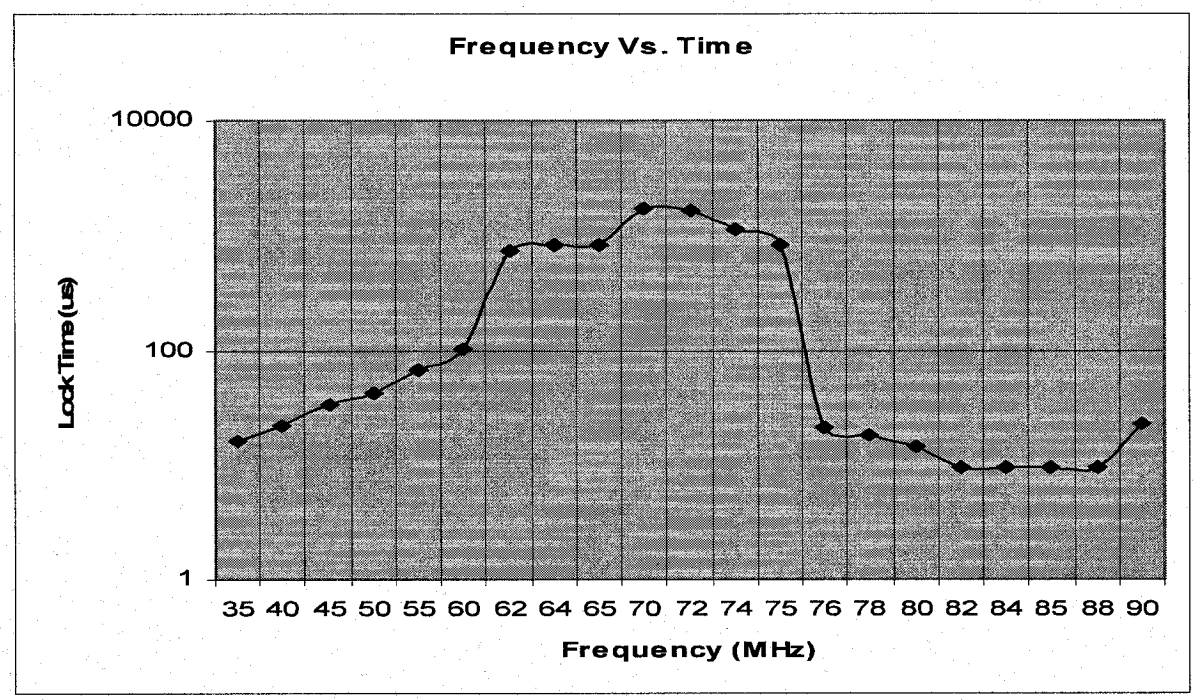

Figure 45. Tracking Vs. Lock Time of DCS using LUT based NCO and 2nd Order LF

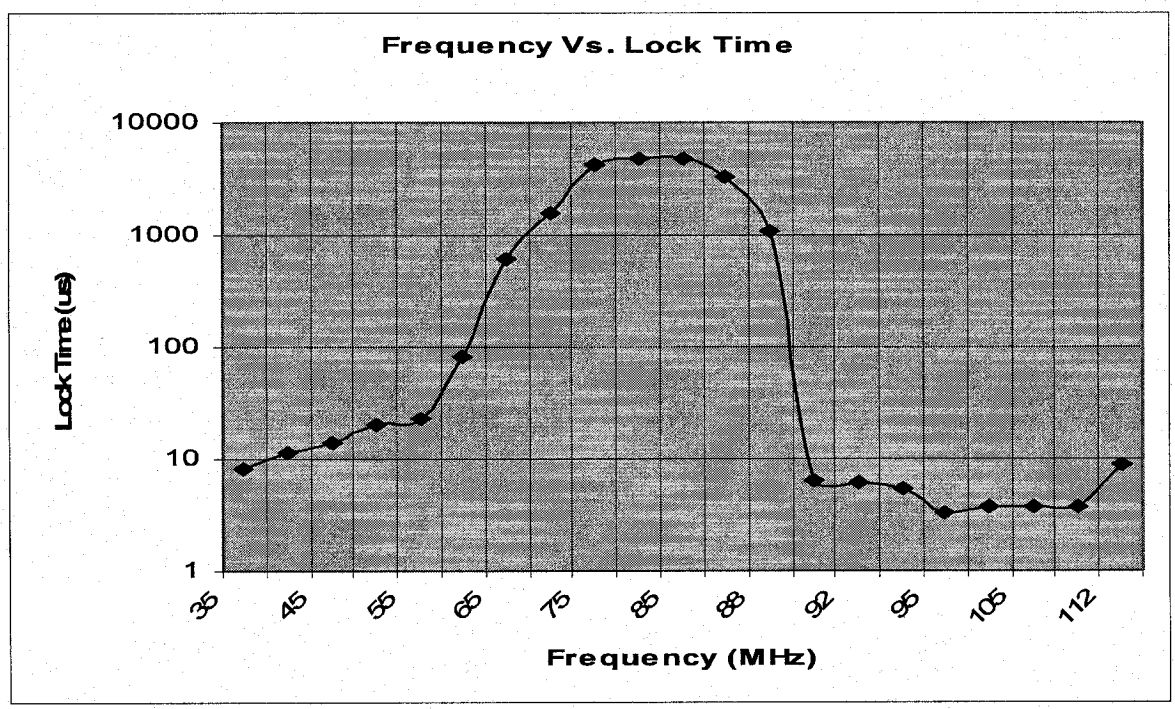

Figure 46. Tracking Frequency Vs. Lock Time of DCS using Xilinx ROM based NCO and 2nd Order LF

DCS using LUT based NCO and second order LF shown in Figure 45 has a locking time of 9.325us and a tracking frequency of 88 MHz. Figure 46 shows an optimal locking time (3.31625 *us*) and a wider tracking frequency (110 MHz) which are achieved using DCS with 2nd order LF and Xilinx ROM based NCO when the sampling frequency is about 110 MHz. The performances of different DCS implementations are summarized in Table 7.

Table 8. Summarized Analysis of Different DCS Implementations

DCS Analysis	DCS using LUT based NCO		DCS using Xilinx ROM based NCO	
	1st Order LF	2nd Order LF	1st Order LF	2nd Order LF
Stability	Stable	Stable	Stable	Stable
Locking Time (us)	11.735	9.325	6.315	3.31625
Phase Angle (Degree)	53	42	19	72
Tracking Range (MHz)	~75	~88	~75	~110

7.5 Summary

In this chapter the locking time and the tracking frequency range of our different DCS implementation using simulation based verification are investigated. Simulation results showed that DCS using Xilinx ROM based NCO and second order LF performed better. In the next and last Chapter of this thesis, the conclusion with some future works will be presented.

Chapter 8

Conclusion and Future Work

In this thesis, initially a mathematical description of the Digital Carrier Synchronizer (DCS) architecture was established which is used as a synchronizer in a modem application. One obligatory constraint for designing DCSs is that the DCS system must be stable. For this reason, afterward the stability of our DCS was verified using MatLab.

We started the design of our DCS using first order loop filter and LUT based NCO and the implementation was done using Verilog HDL. We then improved the design of our DCS architecture with first order loop filter and Xilinx ROM based NCO. Our DCS architecture was simulated using VCS7.0.1 from Synopsys and Active HDL from Aldec. These two designs were investigated and analyzed the simulation results in the context of locking time and tracking frequency. It showed that DCS using first order loop filter and Xilinx ROM based NCO provides better performance. We then again modified the datapath of DCS. We replaced the first order loop filter with the second order loop filter. The modified datapath of DCS was then simulated. The simulation results showed that DCS using Xilinx ROM based NCO and second order loop filter performed better.

The architecture was build from the ground up using digital techniques that exploit faster locking and wider tracking range. We have investigated three different implementations of DCS design using different NCOs and Loop Filters. Our goal was to analysis and compares the locking time and tracking range of the DCSs for the specific application. It has been shown that the DCS using Xilinx ROM based NCO and 2^{nd} order Loop Filter is very efficient. The locking time in the case of a 2^{nd} order LF with Xilinx ROM implementation outperformed the locking time in the case of LUT. Also 2^{nd} order LF with Xilinx ROM based implementation shows wider tracking range. Finally in this research work, we developed a flexible synthesizable emulation environment for FPGA based design solution. It should be noted that, in spite of its many benefits, our emulation system was not a replacement for verification and simulation; it merely augmented simulation. While emulation is ideal for system-level testing, simulation is a must for performing detailed, feature-level testing, which cannot be done effectively in an emulation environment. Finally we can conclude that our simulation results (Shown in Table 8) are corresponded to our emulation results (Shown in Table 7).

As future work, we consider the following research directions:

- The aspect which was not included in my research was the noise analysis. Since noise is an important parameter which affects the performance of a design mostly in non-linear fashion, is necessary to accurately measure the performance of the design. Including noise consideration provide more details about the sensitive points and parameters of a design

- Verifying the design using model checking technique.
- Performing equivalence checking between the RTL level and the gate level of our DCS models.

References

[1] R. E. Best, Phase-Locked Loops: design, simulation, and applications, Fifth Edition, New York; McGraw-Hill, 2003.

[2] Wei-Tsen Lin and Dah-Chung Chang, "The Extended Kalman Filtering Algorithm for Carrier Synchronization and the Implementation," *IEEE International Symposium on Circuits and Systems,* May, 2006.

[3] K. Gunnam et al., "New Optimizations for Carrier Synchronization in Single Carreir Systems," *IEEE International Conference on Acoustics, speech, and Signal Processing,* vol. 5, pp v/661 – v/664, March 2005.

[4] Xilinx Inc. Application Note, XAPP155, *http://direct.xilinx.com/bvdocs /appnotes /xapp155.pdf*, Version 1.1, Sept. 1999.

[5] Xilinx Inc., Application Note, XAPP154, http://www.nalanda.nitc.ac.in /industry /appnotes/xilinx/doc*uments/xapp/xapp154.pdf*, Version 1.1, Sept. 1999.

[6] Ronald R. Stephens, Phase-Locked Loops for Wireless Communications: Digital, Analog and Optical Implementations, Second Edition, New York; Kluwer Academic Publishers, 2002.

[7] Virtual Socket Interface Alliance- VSIA. Web Page: Fact Sheet.

[8] Richard Munden, ASIC and FPGA Verification: A Guide to Component Modeling (Systems on Silicon) Morgan Kaufmann, 2004.

[9] Synopsys Corporation, Synopsys Design Compiler, Version V-2004.

[10] S. Kadam, D. Sasidaran, A. Awawdeh, L. Johnson, and M. Soderstarnd, "Comparison of various numerically controlled oscillators", *The 45th Midwest Symposium on Circuits and Systems*, vol. 3, pp. 200 - 202, Aug. 2002.

[11] Ray. J. Andraka, "A survey of CORDIC algorithms for FPGA based computers", *Proc. ACM/SIGDA sixth international symposium on Field programmable gate arrays*, pp. 191 – 200, Feb. 1998.

[12] Liang Yi ,Yang Yuan, Yu Ningmei and Gao Yong, "The Application of a Novel Direct Digital Frequency Synthesizer for the IP Core Design of All Digital Three Phase SPWM Generator", The 4th International Power Electronics and Motion Control Conference, vol. 2, pp. 730-733, Aug. 2004.

[13] Khalid A. U. et al. "FPGA Emulation of Quantum Circuits," IEEE International Conference on Computer Design: VLSI in Compuerts and Processors, Oct. 2004.

[14] Kyung-Soo Oh, Sang-Yong Yoon, Soo-Ik Chae, "Emulator environment based on an FPGA prototyping board," *IEEE International Workshop on Rapid System Prototyping*, pp. 72-77, June 2000.

[15] Civera, P, Macchiarulo, L.,Rebaudengo, M., Reorda, M.S., Violante, M., "Exploiting Circuit Emulation for Fast Hardness Evaluation," *IEEE transactions on Nuclear Science*, vol. 48, pp. 2210-2216, Dec. 2001.

[16] Samir Planitkar, Verilog HDL: A Guide to Digital Design and Synthesis, Second Edition, Pearson Education Asis, 2002

[17] Himanshu Bhatnagar, Advanced ASIC Chip Synthesis: Using Synopsys' Design Compiler and Primetime, First Edition, Kluwer Academic Publishers, 1999.

[18] Xilinx Inc., Xilinx ISE 8.1i. http://toolbox.xilinx.com/docsan/xilinx8/books/manuals.pdf

[19] Janick Bergeron, Writing Testbenches: Functional Verification of HDL Models, Second Edition, Boston, Kluwer Academic Publishers, 2003.

[20] Thomas L. Anderson, "Using VCS with White-Box Verification Techniques", Synopsys Inc., SNUG San Jose 2000.

[21] William K. Lam, Sun Microsystems, Hardware Design Verification: Simulation and Formal Method-Based Approaches, First Edition, Prentice Hall PTR, 2005.

[22] William F. Egan, *Phase-Lock Basics*, First Edition, A wiley-Interscience Publication, New York, 1998.

[23] Ellinger and Jäckel, Scripts: Integrated Circuits for High-Speed Communication, http://www.ife.ee.ethz.ch/~ichsc, 2005

[24] Intuitive Guide to Principles of Communications, http://www.complextoreal.com, 2002.

[25] Alan V. Oppenheim, Ronald W. Schafer, Discrete-Time Signal Processing, Second Edition, Prentice Hall, 1999.

[26] Lokenath Debnath, Integral Transforms and Their Applications, First Edition, CRC Publishers, 1995.

[27] Benjamin C. Kuo and Farid Golnaraghi, Automatic Control Systems, Eighth Edition, A Wiley-Interscience Publication, 2002.

[28] Giovanni Bianchi, Phase-Locked Loop Synthesizer Simulation, First Edition, McGraw-Hill, 2005.

[29] Floyd M. Gardner, Phaselock Techniques, Second Edition, A Wiley-Interscience Publication, 1979

[30] Design Spec, http://www.us.design-reuse.com/articles/article5187.html.

[31] Steve Winder, Analog and Digital Filter Design, Second Edition, Newnes, 2002.

[32] J. E. Volder, "The CORDIC Trigonometric Computing Technique", *IRE Trans. On Electronic Computers,* vol. 8 no. 3, pp. 330-334, 1959.

[33] Xilinx Inc., http://toolbox.xilinx.com/docsan/xilinx7/books/data/docs/v4ldl/v4ldl0093_85.html

[34] Xilinx Inc., Answers Database.

[35] Xilinx Inc, "FPGA Design Flow Lab Manual," Xilinx University Program, 2003.

[36] Xilinx Inc., Application Note, XAPP 052, July 1996.

[37] M. Moris Mano, Digital Design, Second Edition, Parentice Hall of India Pvt. Ltd., 1998.