An Efficient Architecture for Closed–Loop Power–Control Access

by

Brandon C. Brown

Bachelor of Applied Science, Queen's University, 2006

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

Master's of Electrical Engineering

In the Graduate Academic Unit of Electrical and Computer Engineering

Supervisors:	Mary E. Kaye, M.Eng., Electrical and Computer Engineering
	Brent R. Petersen, Ph.D., Electrical and Computer Engineering
Examining Board:	Christopher P. Diduch, Ph.D., Electrical and Computer Engineering, Chair
	Howard Li, Ph.D., Electrical and Computer Engineering
	Kenneth B. Kent, Ph.D., Faculty of Computer Science

This thesis is accepted

Dean of Graduate Studies

THE UNIVERSITY OF NEW BRUNSWICK

September, 2009

To Mom and Dad

Abstract

In this thesis a cost-effective means to gain access to wireless communication channels is proposed and implemented. A testbed is created using mostly commercial off–the–shelf components, and the signal processing is implemented in an Altera® DE–3 development board, which includes Altera®'s Stratix III® field programmable gate array, as well as two Terasic® ADA boards which provide analog–to–digital and digital–to–analog conversions. Modulation and demodulation was performed on both the forward and reverse channels to bring the signals to an intermediate frequency such that no low pass filter was needed before the signal was acquired by the development board. Testing showed that the CDMA power–control algorithm was able to perform within acceptable limits, and that the implemented system was nearly transparent to both the mobile and the base station. To demonstrate how the system could be used, non–uniform sampling was also implemented in a variety of ways, which included a simple method of incrementally introducing noise.

Acknowledgements

First and foremost, I would like to extend my thanks and gratitude to my supervisors Brent R. Petersen and Mary E. Kaye for all the guidance and assistance they afforded me throughout my research.

I would also like to thank all of the Electrical and Computer Engineering technical staff for their help with the circuit board fabrication, mounting the final project, and the many other various technical aspects of project implementation that they assisted with.

This research was supported by the Atlantic Innovation Fund from the Atlantic Canada Opportunities Agency, and by Bell Aliant, our industrial partner.

Table of Contents

Abstract iii

Acknowledgments iv

Table of Contents v

List of Figures viii

List of Abbreviations x

1 Introduction 1
- 1.1 Background . . . 1
- 1.2 Problem . . . 2
- 1.3 Objective . . . 2
- 1.4 Contributions . . . 3

2 Overview of Architecture 5
- 2.1 Description of System . . . 5
- 2.2 Overall Requirements Analysis . . . 6
  - 2.2.1 Baseband Data Transfer Rate . . . 6
- 2.3 Architecture Design Iterations . . . 7
  - 2.3.1 Initial Designs . . . 7
  - 2.3.2 Final Idea . . . 8

3 Hardware Selection 10
- 3.1 Clock Sources . . . 10
- 3.2 Individual Part Requirements . . . 11
  - 3.2.1 Modulators and Demodulators . . . 11
  - 3.2.2 FPGA and Daughter Cards . . . 11
- 3.3 System Power Requirements . . . 12
- 3.4 Amplifiers . . . 12

4 Implementation 14
- 4.1 Selected Parts . . . 14
  - 4.1.1 FPGA Development Board . . . 14
  - 4.1.2 Modulators and Demodulators . . . 14

4.2 Bias Board . . . 15
4.3 Frequency Selection . . . 17
4.4 Clock Synchronization . . . 18
4.5 RF Isolation . . . 18

5 Testing **19**
5.1 Bias Boards . . . 19
5.2 Modulator . . . 19
5.3 Demodulator . . . 20
5.4 FPGA . . . 22
5.5 CDMA/EVDO Transparency . . . 25

6 Non-Uniform Sampling **27**
6.1 Mathematical Model . . . 28
6.2 Implementation . . . 33
6.3 Known Signal Tests . . . 34
6.3.1 Test Results . . . 35
6.3.2 Discussion . . . 37
6.4 Application to Architecture . . . 40
6.5 Noise Parameter Discovery . . . 41
6.5.1 Modification to PN Generator . . . 42
6.5.2 Length of Shift Register . . . 46
6.5.3 Relation to Theoretical Results . . . 49
6.5.4 Effect on CDMA/EVDO Traffic . . . 49

7 Summary and Future Work **51**
7.1 Summary of Completed Work . . . 51
7.2 Future Work . . . 52

Appendices **56**

A Bias Board Schematic **56**

B Detailed Wiring Diagram **57**

C Final Project Functional Design **59**

D MATLAB® Code Listing **60**
D.1 spec1.m . . . 60
D.2 pb2.m . . . 61
D.3 pb1.m . . . 61
D.4 pulsef.m . . . 62
D.5 pulser.m . . . 62
D.6 rect.m . . . 63
D.7 zero.m . . . 63
D.8 one.m . . . 63

D.9 function_feeder.m 64

E Verilog Code and Block Diagrams **66**
E.1 Code Used in Section 6.3 66
E.1.1 jitter6BitBuff.v 66
E.1.2 jitter6BitBuffEven.v 68
E.1.3 jitter6BitBuffSingle.v 70
E.1.4 jitter6BitBuffTight.v 72
E.2 Code Used in Section 6.5.1 74
E.2.1 jitterMultiPN.v 74
E.2.2 pnGenerator3bit.v 75
E.2.3 pnGenerator4bit.v 76
E.2.4 pnGenerator5bit.v 77
E.2.5 pnGenerator6bit.v 79
E.2.6 pnGenerator7bit.v 80
E.2.7 pnGenerator8bit.v 81
E.2.8 pnGenerator9bit.v 82
E.2.9 pnGenerator10bit.v 84
E.2.10 DE3.v 85
E.2.11 Block Diagram 92
E.3 Code Used in Section 6.5.2 97
E.3.1 DE3.v 97
E.3.2 jitterUniform3Regs.v 103
E.3.3 jitterUniform5Regs.v 104
E.3.4 downsample.v 105
E.3.5 jitterUniform25Regs.v 106
E.3.6 jitterUniform13Regs.v 108
E.3.7 Block Diagram 110

Vita

List of Figures

2.1 High–level proposal 5
2.2 Initial design idea. 7
2.3 Functional diagram for the final design. 8

4.1 Signal biasing board. 16

5.1 Test setup for testing the Inphase. 20
5.2 Test setup for fully testing the modulator. 20
5.3 Spectral output from the modulator. 21
5.4 FPGA test setup. 22
5.5 Resulting spectrum of a 10 MHz signal passing through the FPGA. . 23
5.6 A 10 MHz input signal directly from the clock source used without passing through the architecture. 24
5.7 Data being transmitted during an FTAP test through the system. . . 25
5.8 A successful power–control test. 26

6.1 Non-uniform sampling implementation. 33
6.2 Showing the maximum value (top line), an instantaneous reading (middle line), and the minimum value (bottom line) of a 10 MHz signal using uniform sampling. 34
6.3 A normal distribution showing how many times each register is called per period. 35
6.4 Showing the maximum value (top line), an instantaneous reading (middle line), and the minimum value (bottom line) of a 10 MHz signal using non-uniform sampling. 36
6.5 Modified distribution. 37
6.6 The resulting maximum value (top line), an instantaneous reading (middle line), and the minimum value (bottom line) of the spectrum using the modified normal distribution. 38
6.7 The nearly even distribution used for register selection. 39
6.8 The resulting maximum value (top line), an instantaneous reading (middle line), and the minimum value (bottom line) of the spectrum using the nearly even distribution. 39
6.9 Noise dominating both forward and reverse channels. 41
6.10 An Implementation of non–uniform sampling. 44
6.11 Output when equation 6.33 was implemented. 44

6.12 Output when equation 6.26 was implemented. 45
6.13 Output for 3 registers. 46
6.14 Output for 5 registers. 47
6.15 Output for 25 registers. 48
6.16 Data transfer with non–uniform sampling noise added 50

A.1 Bias board schematic. 56

B.1 Wire connections. 58

C.1 Functional design of the final project. 59

E.1 Part 1 of 4: block diagram of the Verilog implementation used. 93
E.2 Part 2 of 4: block diagram of the Verilog implementation used. 94
E.3 Part 3 of 4: block diagram of the Verilog implementation used. 95
E.4 Part 4 of 4: block diagram of the Verilog implementation used. 96
E.5 Part 1 of 2: block diagram of the Verilog implementation used. 111
E.6 Part 2 of 2: block diagram of the Verilog implementation used. 112

List of Abbreviations

ADC	Analog–to–Digital Converter
CDMA	Code Division Multiple Access
COTS	Commercial off the Shelf
DAC	Digital–to–Analog Converter
dB	DeciBels
dBm	DeciBels relative to one milliwatt
EVDO	Evolution-Data Optimized
FPGA	Field Programmable Gate Array
GPIO	General Purpose Input/Output
I	Inphase
IF	Intermediate Frequency
kbps	Kilo-bits per second
LPF	Low Pass Filter
Mbps	Mega-bits per second
Msps	Mega-samples per second
OCXO	Oven Controlled Crystal Oscillator
PCB	Printed Circuit Board
PCS	Personal Communications Services
PN	Psudorandom Number
PSU	Power Supply Unit

Q	Quadrature
RTAP	Reverse Test Application Protocol
RF	Radio Frequency
V_{DC}	Voltage Direct Current

Chapter 1

Introduction

1.1 Background

In a CDMA system, all mobile stations must be received by the base station at the same power. If one mobile station were to be stationed next to the base station and transmitting at full power, the base station would not be able to detect the other mobiles which are further away. One of the most used solutions to mitigate this *near-far* problem is power control [1].

Several types of power control are available, such as the ones simulated by Chulajata and Kwon, who showed that by combining more than one power control algorithm it was possible to increase performance [2]. However, as is shown by Sim et al. it is possible to lower the bit error rate of a moving mobile station by simply updating the power control more frequently but at the cost of using increased resources [3].

CDMA is a spread spectrum technology that originally had its background in military applications providing secure communications [4]. The standard started as IS95, and has since evolved to its current form seen today: cdma2000. EVDO was integrated with cdma2000 to provide high rate data packet services without having

to support old standards [5]. This project utilizes the CDMA/EVDO technology as a basis for communication between the mobile and the base station.

Time–varying channel conditions are also present in CDMA systems, which requires the power control algorithm to respond accordingly to ensure communication continues, as well as signal filters to remove possible multipath effects. Channel fading generally follows Rayleigh or Ricean fading distributions, and consists of flat fading or fading in the small scale [1].

1.2 Problem

The communication medium used by cellular CDMA systems for power control is a closed system; the relationship between the transmitter and receiver is such that it is not meant to have a third party join the communication or modify the channel. In order to gain access into the communication stream, a system must be set up in order to intercept the signal sent from a mobile station to the base station and vice versa. Once the signal is captured (and possibly modified), having it continue back through the original transmission medium will be required to make it appear to both devices as if the system is transparent.

1.3 Objective

The objective of the proposed research consists of designing and implementing a system in which it is possible to capture and retransmit the communication between a mobile station and a CDMA base station. The solution is implemented using as many COTS components as possible.

The motivation for developing this system is to analyze both the forward and reverse links simultaneously. Modern mobile phones are able to separate the forward and reverse links internally using customized proprietary hardware. Because of this

it is impossible to gain access or control of the phone's hardware.

There are other systems which have been designed or implemented which perform similar functions to the system proposed in this thesis, each with their own benefits and drawbacks. Niida et al. had an entire cell phone network set up with multiple base transceiver stations and multiple mobile stations [6]. An ultra wideband transceiver prototyping test bed has also been built, but requires 8 high speed ADCs to capture signals using time samples, not frequency shifting, at sampling rates of up to 8 GHz [7]. Also, the emphasis in the research was on the demodulation only. Another design by Shono et al. has a transceiver architecture with a digital signal processor at its core, and was designed for the IEEE 802.11 standard, however it did not use an explicit modulator or demodulator [8].

In order to gain access to both the forward and reverse channels, the signal will have to be demodulated, passed through a device which is controllable such as an FPGA, and then modulated back up to RF. With this system it should be possible to monitor the transmission that is being sent by the base station to the mobile station and vice–versa. It could also allow researchers to modify the captured signal to determine if that signal can be controlled externally. Also, using this system it may be possible to model the channel and possibly evaluate current power-control algorithms and phone performance.

1.4 Contributions

The architecture put forth in this research will serve as a test bed for future research on mobile communications channels. Since the forward and reverse link channels are separated and handled individually within the FPGA it is possible to apply channel conditions to only one channel at a time. This gives the ability to clearly see how a test condition on one channel can potentially affect the other channel.

The purpose of this system is to allow researchers to view the communications channel while in real world conditions and allow modifications to the channels. Using the system proposed it may also be possible to obtain data on the channel characteristics from these measurements.

After dominion over the power-control is established it would be possible to test alternative power-control algorithms in a real-world scenario and compare the results to well established algorithms. Also, because it would be possible to assert a channel model, the ability to test to see under what conditions current power-control algorithms fail becomes a possible use of this architecture.

To help save complexity and to help ensure that as little signal is lost as possible, the LPFs in the original design were removed. In order to do this the guarantee that there would be no interference with aliasing effects present after demodulation had to be ensured. After much investigation, it was found that the signal of interest in both the forward and reverse channels could be safely brought down to IF without any interference.

It was also discovered that the base station and the mobile phone would still communicate with each other when the signals were demodulated down to baseband before they were acquired by the FPGA. Because the ADCs are not guaranteed to function properly when acquiring signals below 1 MHz, it appeared that there would be a local fade in the channel, similar to the channel modeled by Mar and Chen [19].

Finally, non-uniform sampling was implemented as a low complexity method of injecting noise. Three separate approaches were taken during the implementation. The first approach was a proof of concept to show that it was possible to introduce noise using non–uniform sampling. The other two approaches investigated a simple method of incrementally introducing noise.

Chapter 2

Overview of Architecture

2.1 Description of System

The system proposed is placed in-between a base station and a CDMA–enabled phone using the PCS1900 standard and handles all communication between the two. This is shown graphically in Figure 2.1. Since the 10 MHz wide I and Q signal components are separated before they are passed to the FPGA there is a minimum requirement of four ADCs and four DACs; two of each on both the sending and receiving side. Because the ADCs have a documented lower bound on the frequency they are able to acquire, an IF of 10 MHz will be used to transmit the individual I and Q channels from the demodulator to the ADCs.

Because the FPGA is the centre of the design, the architecture is able to be broken up into two separate sides: one side that includes the base station and another side that includes the mobile. Because of this split, it is possible to use the FPGA to

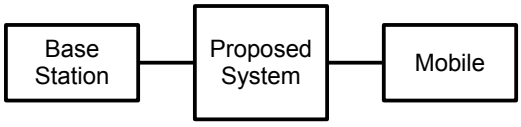

Figure 2.1: High–level proposal

control the communications path and monitor all data being passed back and forth.

For all except one test, the signals were not demodulated down to baseband but instead to an IF. This was done because of limitations of the ADCs, however, it also means that this architecture can be implemented in such a way that it does not impede on an individual user's privacy.

2.2 Overall Requirements Analysis

The major requirement of this project was to establish a communications channel through a medium that was externally controllable. External control could be done through a connection to a personal computer, or even a simple approach such as using pushbuttons and dip switches present on most FPGA evaluation boards. This transparent pass–through system would then be extensible to further assist researchers. In order to facilitate the extensibility, the system must also be reconfigurable so that new tests and algorithms can be implemented.

In order to achieve data pass–through, the system would have to appear transparent so that there is minimal data corruption and delay. The delay introduced by the system should be minimal. Because of this, the processing speed of the system must guarantee a timely response. This can be done by choosing fast components and using a parallel architecture.

Keeping the final implementation cost effective was also a requirement due to a limited budget.

2.2.1 Baseband Data Transfer Rate

Since the maximum RTAP rate for the EVDO protocol over the cdma2000 standard is 153.6 kbps for the reverse link and 2.4 Mbps for the forward link, the bus transporting the data must at a minimum support double the sum of these speeds.

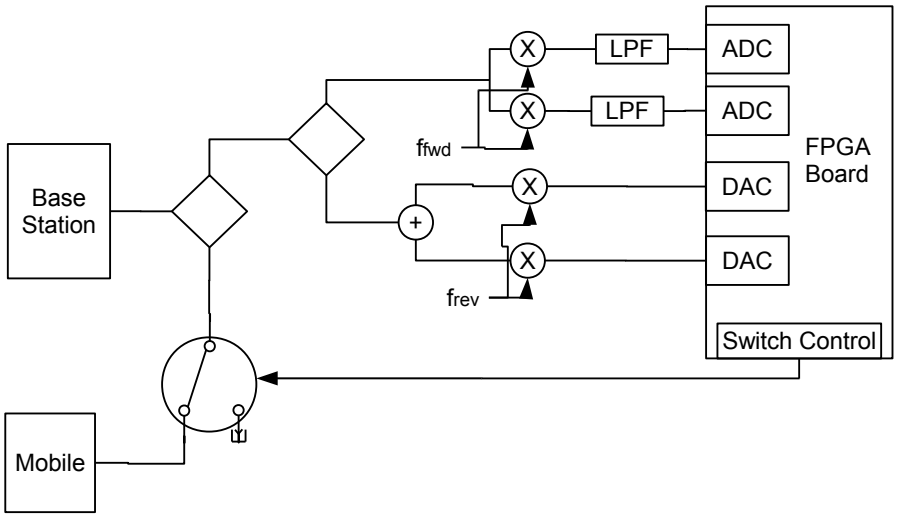

Figure 2.2: Initial design idea.

However, if the data is to be passed through at an IF, the bus has to ensure a throughput of double the IF to compensate for data at IF going in both directions. At the IF rate used, the GPIO pins provided by all the FPGAs investigated for this project were able to transfer data at the necessary speeds.

2.3 Architecture Design Iterations

The design of the overall system went through many iterations. Some changes occurred because they made the design more extensible, other changes were made because of the difficulty and cost of procuring parts. Major decisions are discussed below.

2.3.1 Initial Designs

Shown in Figure 2.2, the initial design had some major issues that needed to be resolved. Top among them is that the FPGA has to generate the data required for the test, or at least be able to play back a pre-recorded data stream for testing.

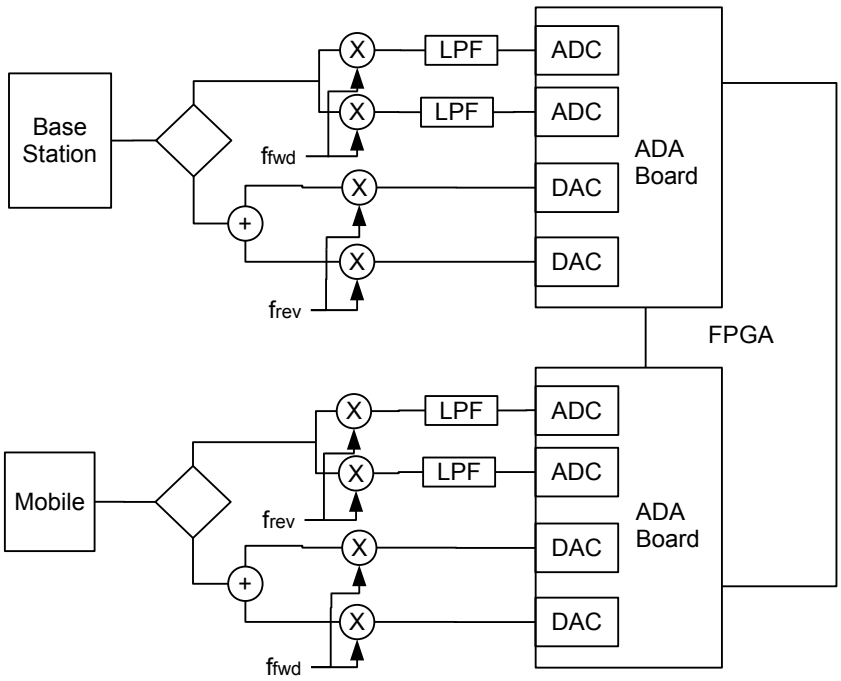

Figure 2.3: Functional diagram for the final design.

A switch was to be included so that the mobile would be taken completely out of the loop, and the entire data transfer would be supported by the architecture that was to be designed. Since implementing the EVDO standard would have been a task too large given the time constraints, and the playback of recorded data would have not guaranteed the normal responses to various channel perturbations, these ideas were eventually rejected.

Another design iteration was similar to the first, however, it required a relatively expensive and difficult–to–obtain diplexer, and still required that the entire CDMA/EVDO specification be implemented in the FPGA. Such an implementation would have increased the time to completion to an unreasonable length.

2.3.2 Final Idea

The final functional design which was built is shown in Figure 2.3. This design has the benefit of not needing the redundant diplexer, has fewer power splitter/com-

biners, and allows for both the forward and reverse links to be completely separated, as well as also having the I and Q channels separated on each link. The trade-off, however, is a slightly higher cost for the FPGA system. However, cost was kept down by choosing the Terasic® DE3 FPGA development board which had the resources to attach two ADA daughter boards. This is discussed further in Chapter 3.

Chapter 3

Hardware Selection

In order to meet the requirement of being cost effective, the majority of the system proposed was implemented using COTS components. In doing so, this removed the need to design complex PCBs and allowed the focus to be shifted to unit testing and full system tests. Although one PCB was required to be designed, it was relatively simple compared to the Altera® DE-3 [9] development board used.

One of the first major decisions made while selecting hardware was choosing between a microprocessor–based design, or an FPGA–based design. Because the I and Q channels need to be captured and processed simultaneously, having the ability to perform parallel processing became a requirement. Also, if a general purpose central processing unit were implemented, the time it would take to perform any signal processing would become an issue. Although both platforms would provide an extendable architecture, the choice to use an FPGA was ultimately made because of it's high reconfigurability to perform specific tasks.

3.1 Clock Sources

Several clock sources were investigated because of the requirement to have a highly stable reference frequency. Standard clock sources which were based on

rubidium oscillators, cesium oscillators, the GPS reference signal and the CDMA reference signal were investigated, before the decision to purchase two high stability OCXO–based signal sources was made. Unlike the other options, this provided a stable 10 MHz reference output out of the back panel which could be used to drive other parts of the project to keep everything locked together, and did not require any custom PCBs to be built to provide the high frequency clock sources required for modulation and demodulation.

3.2 Individual Part Requirements

3.2.1 Modulators and Demodulators

The Hittite® HMC97LP4 modulators and Hittite® HMC597LP4 demodulators are used in this project. The demodulators were required to bring signals from the PCS band, in the 1900 MHz range, down to an IF which could be acquired by the FPGA. The modulators are required to do the opposite, bring signals from the IF up to the appropriate PCS band. Although the PCS band is used for the purposes of this project, the modulator's output and demodulator's input have an operational range of 100-4000 MHz [10][11]. This allows the system put forward to operate over a large range, and can be used in the future to run tests in another frequency band.

3.2.2 FPGA and Daughter Cards

Most FPGA development boards investigated included ADCs and DACs pre–built on to them. However, all of the development boards investigated contained only one or two ADCs and DACs each, if any. If the board did not contain enough ADCs and DACs, the ability to network the FPGAs together would become a requirement. Another possible solution to this problem, and ultimately the solution chosen for this project, was to use a board which was extendable in such a way that daughter cards

which contained the required four ADCs and four DACs could be used. Because all of this was contained on one FPGA board, synchronization between the ADA boards was made trivial, and allowed for data to be passed through the system at rates much higher than needed. Also, not having to create a communication interface for two FPGAs simplified the final design.

Because an IF of 10 MHz was selected, the ADCs and DACs chosen would have to ensure at least a minimum of a 40 MHz sampling rate in order to satisfy the Nyquist criterion; bandpass sampling requires the sampling frequency is more than four times the width of the passband. On both sides of the FPGA the signals are broken into their I and Q components, and then passed into the FPGA. After any signal processing has taken place, the I and Q components are transmitted out of the FPGA, recombined and continue along the communications channel.

3.3 System Power Requirements

When supplied by a 5 V_{DC} source, the maximum given current requirement of the demodulator is 230 mA, and 168 mA for the modulator. Since there are two modulators and two demodulators, the total power required is 796 mA. The bias–Ts discussed in Section 4.2 are also powered by the 5 V_{DC} power supply, however their current requirement is negligible.

The amplifier is powered by a separate 12 V_{DC} PSU as indicated in the part specifications. The FPGA board included its own power supply, and only required a 120 V outlet.

3.4 Amplifiers

To help with signal demodulation, a connectorized amplifier was included in the final design in an effort to boost the signal of interest. This was done as an attempt

to utilize more of the ADCs dynamic range. Had the amplifiers been excluded, the ADCs' resolution would have been reduced. The amplifier used was a wide bandwidth, low noise Mini-Circuits® ZX60-6013E-S+. At a frequency of 2000 MHz, the gain is 15.2 dB.

Chapter 4

Implementation

4.1 Selected Parts

4.1.1 FPGA Development Board

At the centre of this system is an Altera® DE–3 development board which includes the Stratix III® FPGA, and two Terasic® ADA boards [12]. The ADA boards are where the ADCs and DACs reside. On both sides of the FPGA the signal is broken into its I and Q components, and then passed into the FPGA. After any signal processing has taken place, the signal is transmitted out of the FPGA, recombined and continues along the communications channel.

The ADCs have a maximum sampling rate of 65 MHz [13], and the DACs have a maximum update rate of 125 MHz [14]. This met the requirements with a wide margin.

4.1.2 Modulators and Demodulators

The Hittite® modulators were purchased on a connectorized board to minimize the time required for setup. The modulator used was intended to be driven by a dual–ended signal with a 1.5 V_{DC} offset on both ends. However, for this project, only a

single–ended signal was required and applied; the negative side of the terminal was not used, but instead terminated. It is important to note that without the 1.5 V_{DC} offset in the signal voltage on the positive terminal as well as the constant 1.5 V_{DC} present at the negative terminal, the modulator would not operate.

The chosen demodulator was also from Hittite®. It was selected because of implementation knowledge that already existed, expediting implementation and testing. The demodulator only required that the local oscillator be offset by 1.5 V_{DC}. This was taken account of when designing the bias board shown in Appendix A.1 and described below.

4.2 Bias Board

Because the modulators and demodulators required that certain input ports include a 1.5V_{DC} offset, a bias board was designed and constructed. The PCB in Figure 4.1 is the signal biasing board which was designed and implemented. It consists of a simple bias–T circuit. It was designed with both the modulators and demodulators in mind; only the needed sections of each board were populated. Each connectorized side of the bias board had a signal input, a signal output, and a 1.5 V_{DC} output. The signal output was the small signal input transposed +1.5 V_{DC}.

Spice simulations were run to ensure proper operation in the 1900 MHz range. The circuit diagram of the bias–T circuit can be seen in Appendix A.1. The 1.5 V_{DC} supplied to the circuit was provided by a voltage regulator that was built onto the board. The board is powered by a 5 V_{DC} power source.

The decision to use an external 5 V_{DC} power source was made so that a common power supply unit could be used for almost all parts on the board: both the modulators and the demodulators also require a 5 V_{DC} power supply. Since the current requirements described in Section 3.3 show that the current draw was not large,

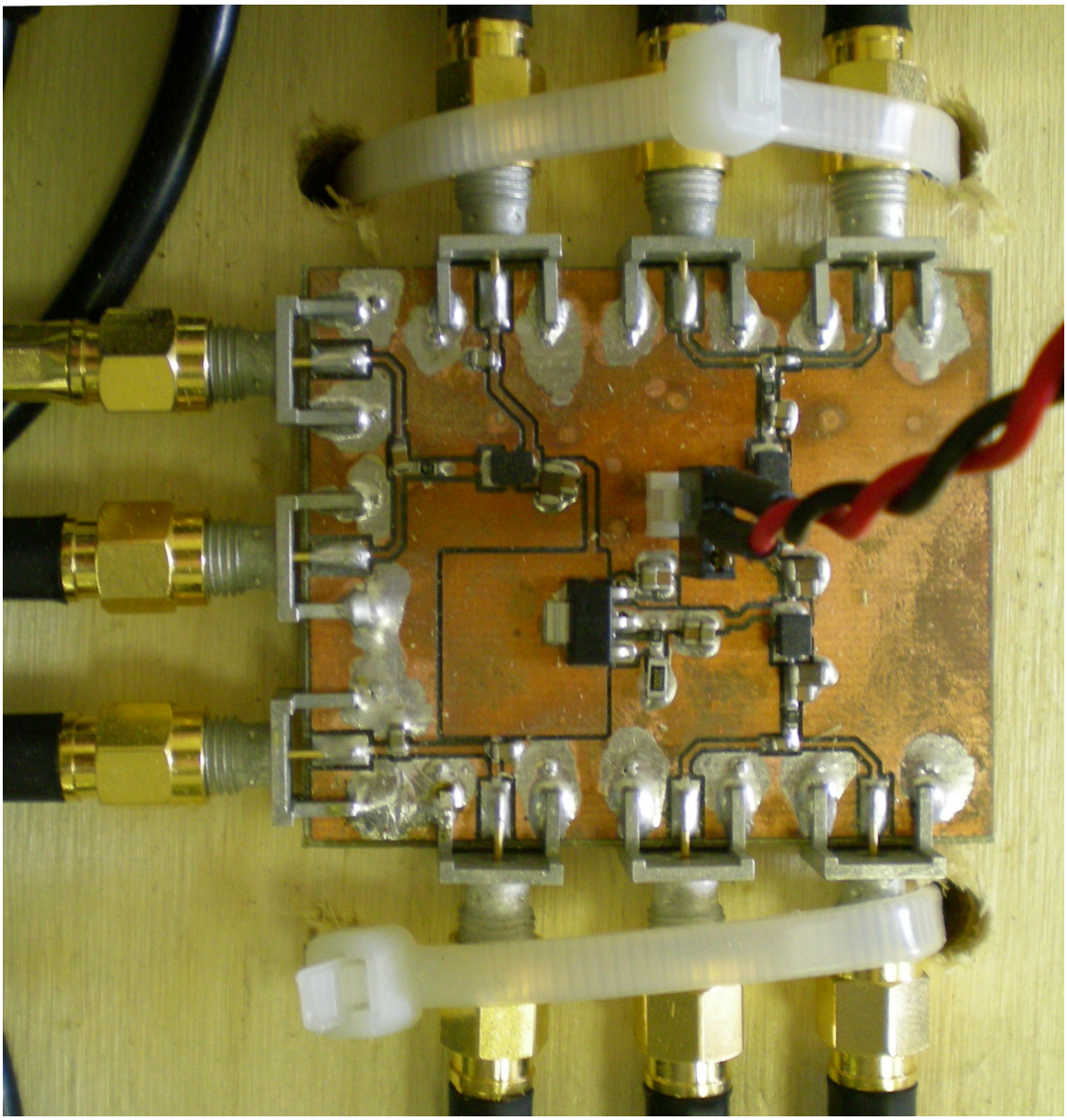

Figure 4.1: Signal biasing board.

all parts that required 5 V_{DC} were able to be powered using a single supply.

4.3 Frequency Selection

For the investigation of frequencies to be used for the local oscillator of the modulators and demodulators, a MATLAB® program was written for simulating the artifact interference caused during the modulation and demodulation phases.

The final proposed design, shown in Figure 2.3, called for a LPF to remove any signals outside of the acquisition range. Originally, the filter was to remove any of the sampling artifacts that may be caused. After modeling the acquisition in MATLAB®, it was discovered that the signal could be cleanly acquired without any sampling artifacts caused by harmonics. The final implemented design can be seen in Figure C.1.

A sampling frequency of $f_s = 62.5$ MHz was used in conjunction with the reverse channel modulation and demodulation frequency $f_r = 2007$ MHz, and the forward channel modulation and demodulation frequency $f_f = 1930$ MHz. Other frequencies that were shown to have good performance were $f_r = 1893$ MHz and $f_f = 1993$ MHz. The MATLAB® code that was used to model this behaviour is shown in Appendix D. By selecting frequencies that would ensure that the signal of interest in both the forward and reverse communication channels were untouched by sampling artifacts and removing the LPF, it enabled this architecture to be a general purpose, protocol agnostic, channel interception and modification design. With this, it becomes possible to extend the use of this project past the CDMA/EVDO research to study other standards.

4.4 Clock Synchronization

Supplying a base band signal to the Hittite® modulators and demodulators are two high–stability OCXO clock sources. To avoid any synchronization issues, the clock sources, FPGA and base station were locked to a common 10 MHz reference signal supplied by one of the OCXO clock sources.

4.5 RF Isolation

To isolate the mobile from any other corporate network that is being used, and to ensure that the implemented system did not interfere with any legitimate businesses, the mobile was placed inside a metal anechoic chamber. It was found that an empty chamber caused the mobile to not connect, and it was postulated that there were too many RF reflections on the inside of the chamber. Special absorptive RF insulation barriers were added to the chamber to reduce the interior reflections. Also, holes that were present in the chamber were discovered and were sealed using metal tape. A special cable and adapter which plugged into the back of the mobile and bypassed its antenna was purchased in an attempt to have a more managable size system. This provided limited success, as the mobile was not purchased new and the antenna bypass was suspected of having an intermittent connection. For all tests performed for this thesis, the anechoic chamber was used.

Chapter 5

Testing

Throughout the development of this architecture, testing had been considered of paramount importance. In Harriman's thesis [15] it was found that the parts selected did not always conform to their associated specification sheet. Because this project and Harriman's shared a few common parts, it was important to ensure proper testing was done at every stage.

5.1 Bias Boards

A prototype board was first commissioned, thoroughly tested, and performed as expected with no errors or unexpected behaviours present. Testing was conducted using a signal source and an oscilloscope. A signal was presented at the input port, and the oscilloscope which was connected to the output port displayed a signal identical to the input with the required V_{DC} offset.

5.2 Modulator

The initial tests of the modulator were done using the system layout in Figure 5.1. Testing both the I and Q channels were done on the setup shown in Figure 5.2.

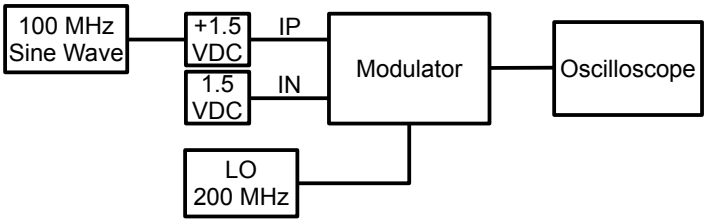

Figure 5.1: Test setup for testing the Inphase.

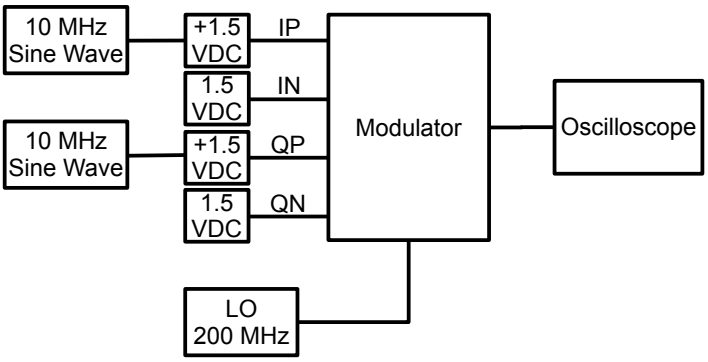

Figure 5.2: Test setup for fully testing the modulator.

A mathematical model was used to show the expected output as seen by the oscilloscope, and was extended later to include the expected output when both the I and Q channels were used.

The spectral output is shown in Figure 5.3 from the tests done using the setup in Figure 5.2, which matched expectations.

5.3 Demodulator

Testing of the demodulator required that the modulator first be successfully implemented. Once the modulator was successfully implemented, the output of the modulator was connected to the input of the demodulator. Using the same clock source for the demodulator as the modulator, it was possible to recover the original signals that were fed into the modulator. This was verified for both the I and Q

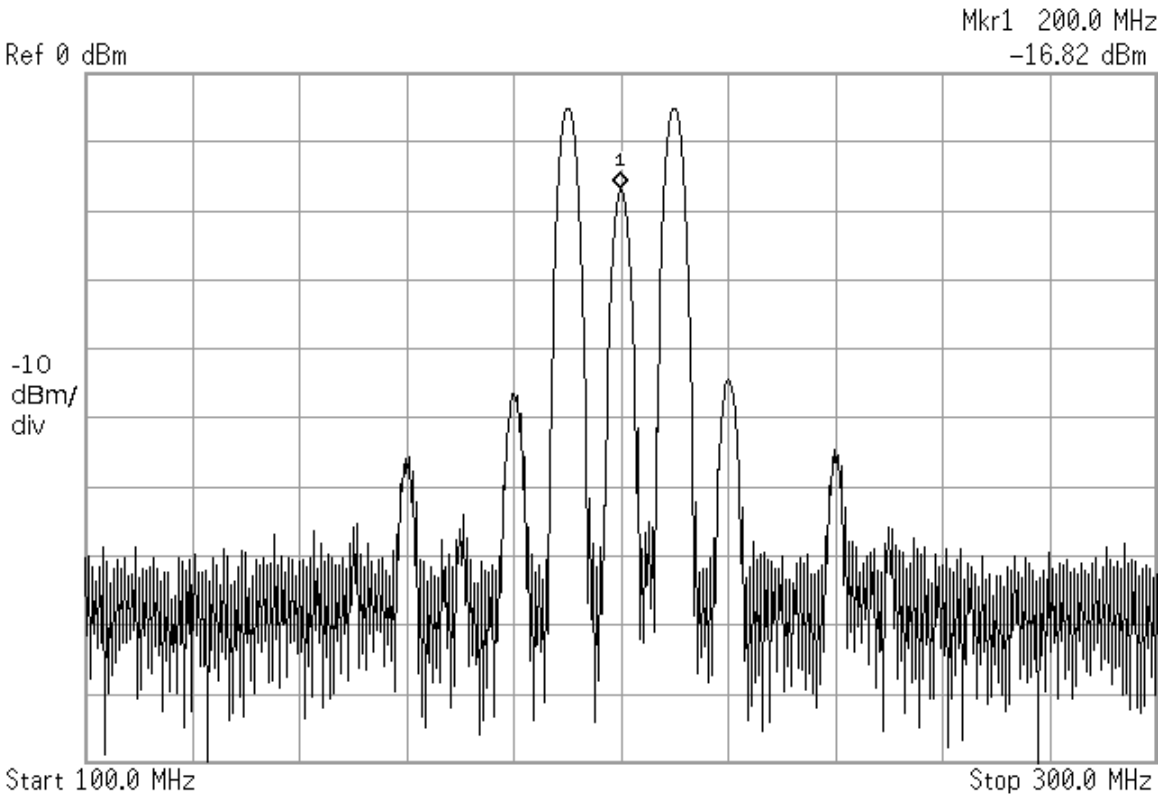

Figure 5.3: Spectral output from the modulator.

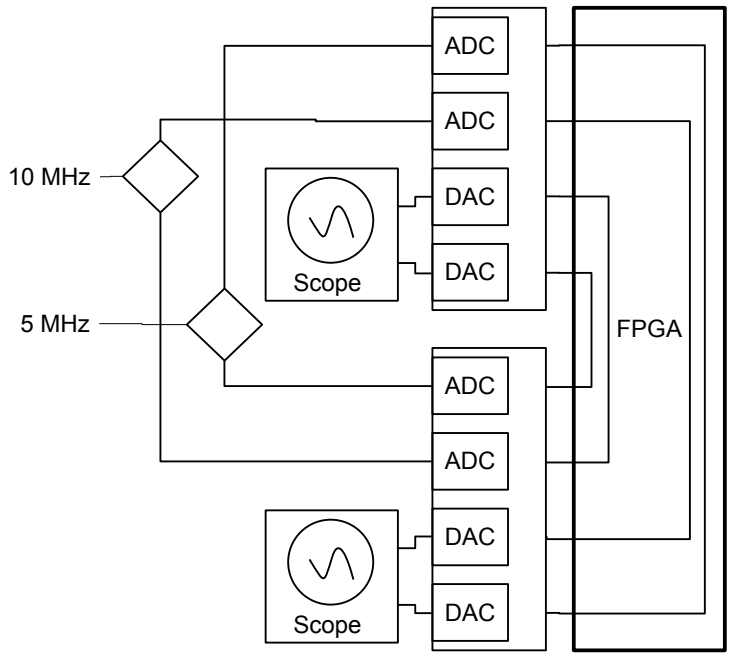

Figure 5.4: FPGA test setup.

channels independently, and finally concurrently. The power level received was not exactly matched to the power transmitted, but some loss was expected. This could be attributed to using the demodulators in single–ended mode, which was found to cause a small amount of power loss.

5.4 FPGA

Several tests were conducted on the FPGA and ADA daughter boards to ensure proper operation. Each test built on the previous one, until the test setup shown in Figure 5.4 was reached. Using this setup, it is possible to show that the signal captured on any ADC channel was able to be regenerated using the DAC channel on the opposite board. This setup closely relates to the final setup that was used in the final implementation.

A 10 MHz sinusoidal signal was fed into an ADC port on the FPGA at 0 dBm,

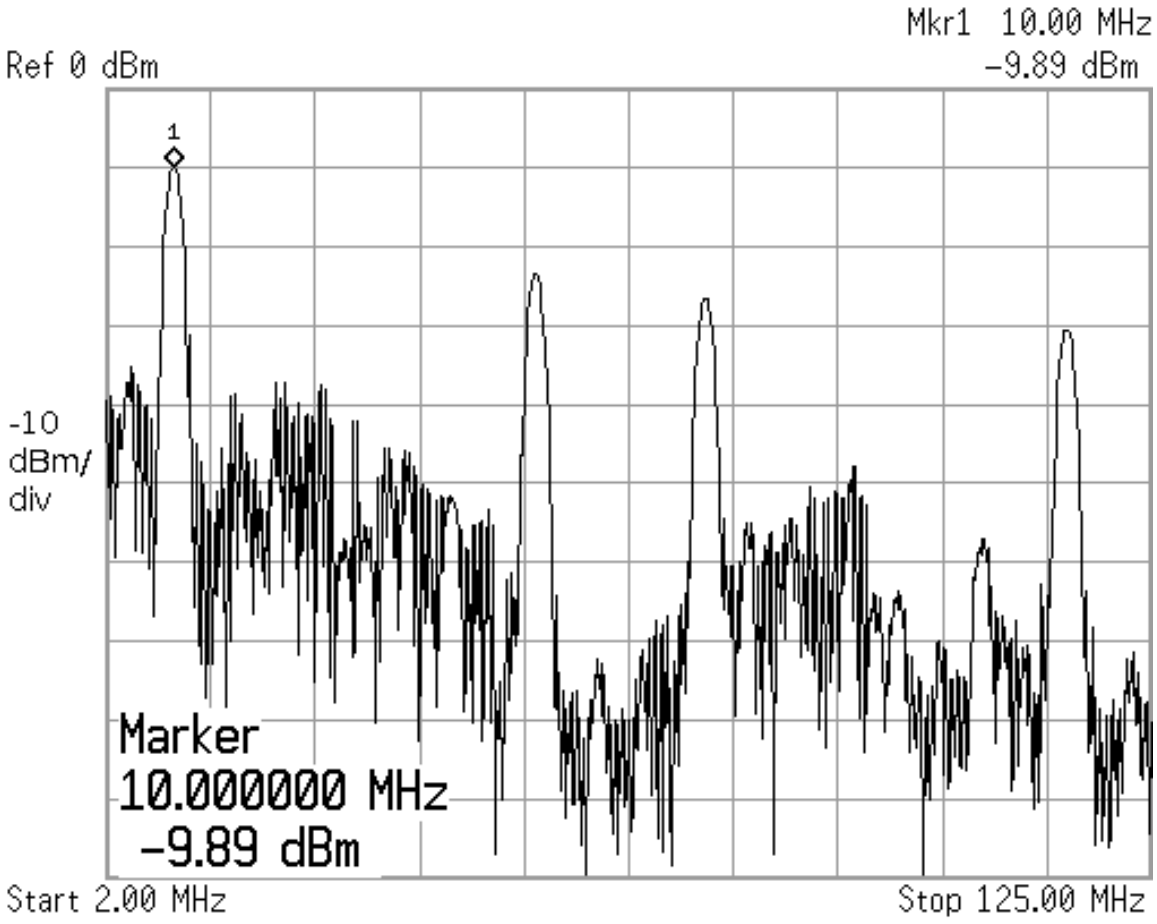

Figure 5.5: Resulting spectrum of a 10 MHz signal passing through the FPGA.

sampled and regenerated at 62.5 Msps, then fed directly into a spectrum analyzer. The resulting frequency graph shown in Figure 5.5 is a waveform capture showing the output signal is approximately 10 dBm below the input signal.

Also shown in Figure 5.5 is that the noise floor has been increased. For reference, see Figure 5.6, which shows the 10 MHz sine wave that was output from the signal source and fed directly into the spectrum analyzer. The extra spikes in Figure 5.5 that are above the expected 10 MHz signal are artifacts that occur because of the sampling performed by the ADCs.

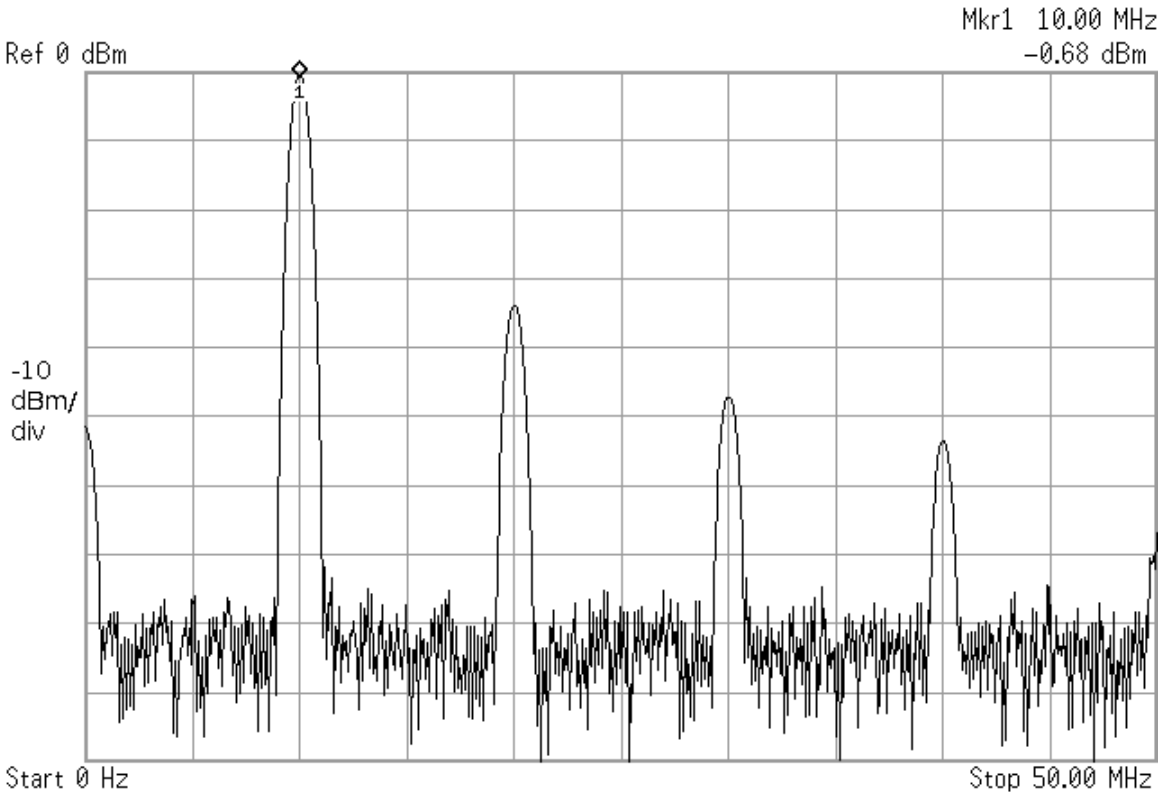

Figure 5.6: A 10 MHz input signal directly from the clock source used without passing through the architecture.

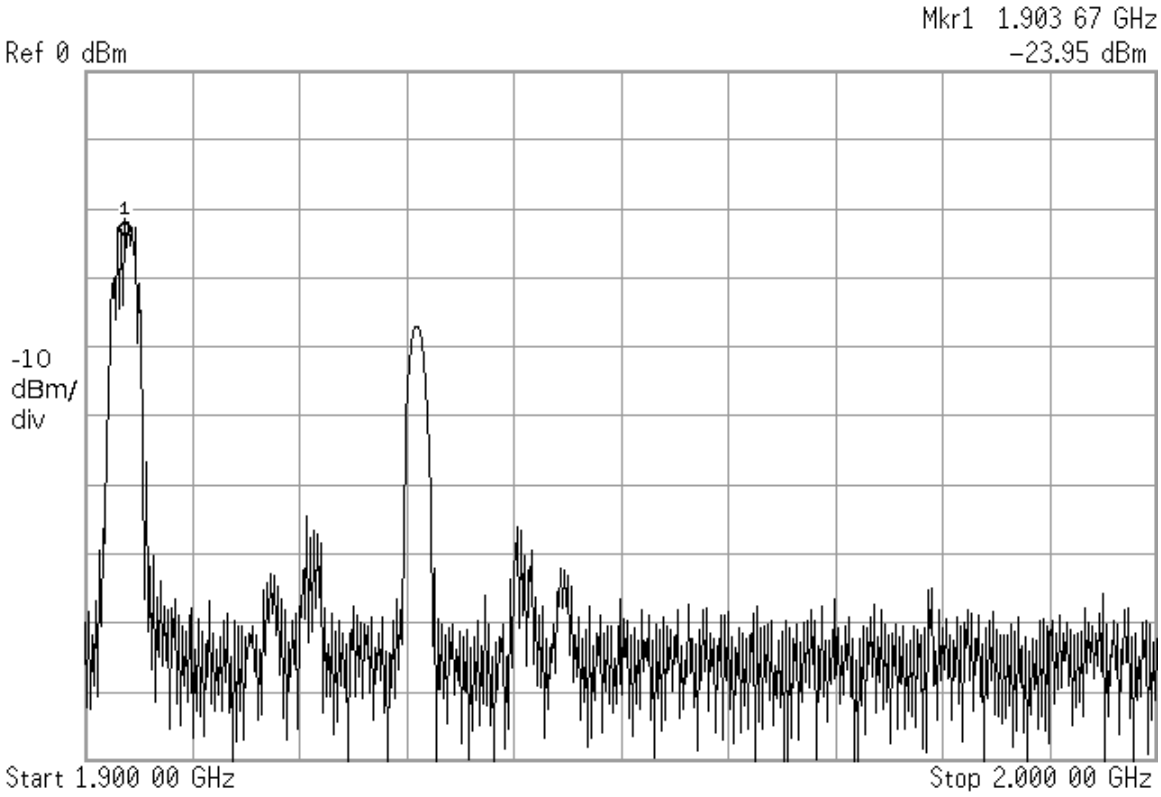

Figure 5.7: Data being transmitted during an FTAP test through the system.

5.5 CDMA/EVDO Transparency

To show that the system was able to appear transparent to both the mobile and the base station, tests were conducted showing that data was able to be transmitted and received. This was done by using a power splitter/combiner between the FPGA and the mobile phone and allowing a spectrum analyzer to observe the transactions taking place. Figure 5.7 is a spectral plot showing the data being sent from the mobile to the base station at 1903.75 MHz; marker 1 is positioned at the channel in use. The other obvious peak at 1930 MHz in Figure 5.7 is a result of the modulation that was performed to transpose the signal from IF to the required 1903.75 MHz.

Figure 5.8 shows that the system put forward is successfully able to pass through commands from the base station to the mobile regarding power–control. The top and bottom lines are power limits which move in time. The middle line

is the actual power output. The test shown was a "power–up" command: the base station signaled the mobile to increase the cell power by 20 dBm. As can be seen in the Figure, the mobile passed the test, showing that the system in place between the mobile and the base station did not adversely affect the communication between the two.

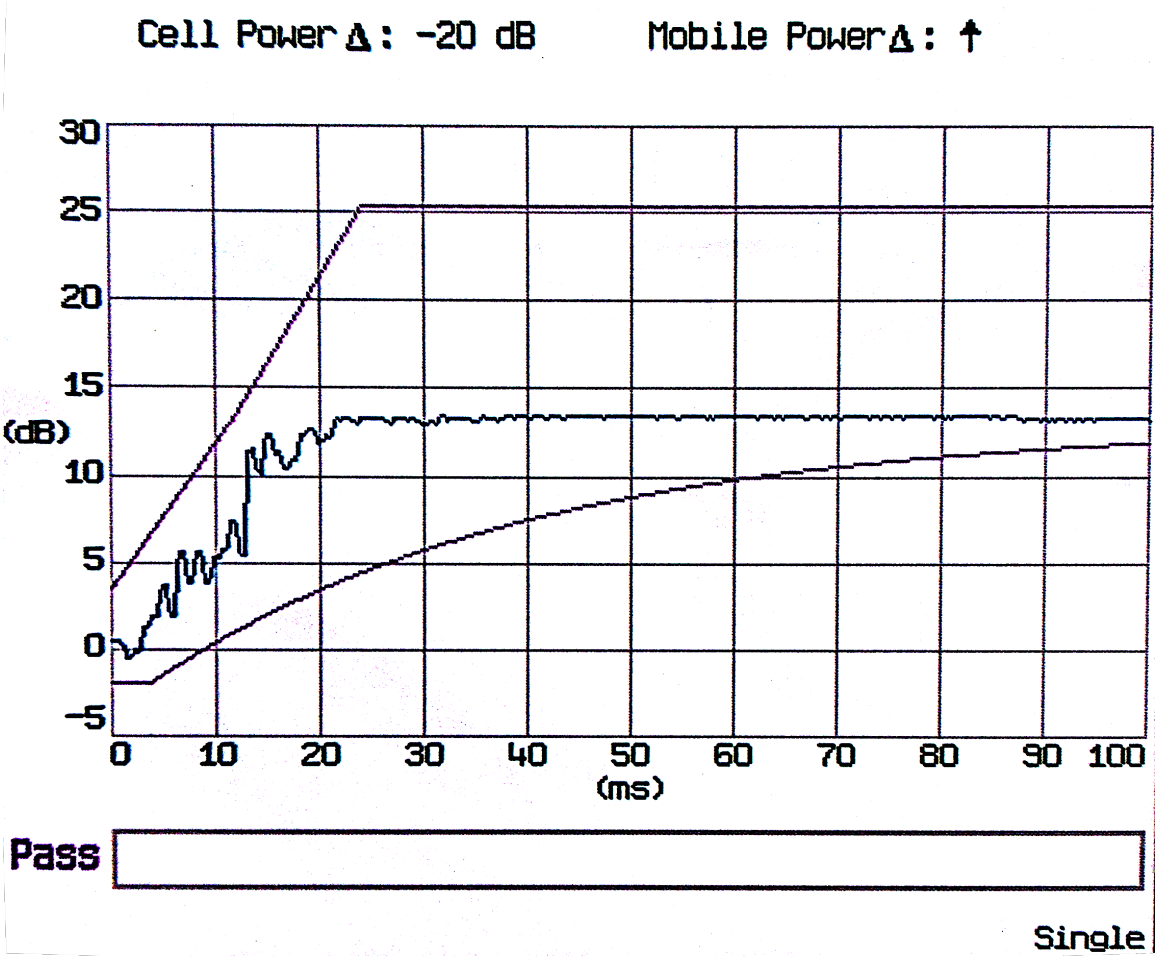

Figure 5.8: A successful power–control test.

Chapter 6

Non-Uniform Sampling

The ability to implement non-uniform sampling on this project was in no small part because of the reconfigurable nature of the architecture; having easy access to both the forward and reverse channels allowed for new manipulations within the FPGA. To implement non–uniform sampling, the uniformly sampled signal was fed from the ADC into a shift register, and a PN generator was used to select which register the output should be tied to. Because the goal was non–uniform sampling, the register selection needed to be random. The randomization was accomplished by implementing PN generator to select which register in the shift register was to be used for output. This has the effect of randomizing the sampling, and also does not guarantee sample order.

Randomization caused by implementing this style of non-uniform sampling is analogous to multipath effects—delays caused in time and noise generated by the signal not arriving in the correct order due to various reflective surfaces in the environment. In this implementation, the clock for the PN generator is much higher than the signal frequency, and the number of buffers were chosen to ensure that samples were being taken within a confined region of the signal period.

6.1 Mathematical Model

In an effort to model the expected variance of non–uniform sampling, mathematical analysis was done. Starting with a sinusoidal signal $s(t)$ described in equation 6.1,

$$s(t) = \cos(2\pi f_1 t) \tag{6.1}$$

where f_1 is the frequency of the signal, and t represents time. For uniform sampling, all samples are taken at equal discrete intervals shown in equations 6.2 and 6.3,

$$s_n = s(t)|_{t=nT} \tag{6.2}$$

$$= \cos(2\pi f_1 nT) \tag{6.3}$$

where n is the index number, T is the time between samples, and s_n is the indexed discrete signal. However, in non–uniform sampling the discrete intervals that the samples are taken at are not equally spaced leading to the equation

$$\begin{aligned} V_n &= s(t)|_{t=nT+\Delta n} \\ &= \cos(2\pi f_1(nT + \Delta n)) \\ &= \cos(2\pi f_1 nT + 2\pi f_1 \Delta n) \end{aligned} \tag{6.4}$$

V_n is the indexed discrete non–uniformly sampled signal, and Δn is the difference in n. To demonstrate non–uniform sampling, an input signal of $y = \cos(\theta)$ was used where θ is distributed as a uniform continuous random variable in the range $0 \rightarrow \pi$. Also, Δ is distributed as a normal random variable with the range of $0 \rightarrow \sigma_n^2$, where σ_n^2 is the variance. These are described in equations 6.5 and 6.6.

$$\theta_1 \sim U(0, \pi) \tag{6.5}$$

$$\Delta \sim N(0, \sigma_n^2) \tag{6.6}$$

Given Y in equation 6.7,

$$Y = \cos(\theta) \tag{6.7}$$

equations 6.5 and 6.6 can be inserted into 6.7 to get equation 6.8.

$$Y = \cos(\theta_1 + \Delta) \tag{6.8}$$

The formula to find the variance of the non-uniformly sampled signal is shown in equation 6.9.

$$\sigma_Y^2 = E[Y^2] - E^2[Y] \tag{6.9}$$

Starting with the variance in equation 6.10, equations 6.5 and 6.6 are inserted to obtain equation 6.11, which leads to equation 6.12.

$$E_{\theta_1,\Delta}[Y^2] = \iint_{-\infty}^{+\infty} g(x) f_{x,y}(x,y) dx dy \tag{6.10}$$

$$= \iint_{-\infty}^{+\infty} \cos^2(\theta_1 + \delta) f_{\theta_1,\Delta}(\theta_1, \delta) d\theta_1 d\delta \tag{6.11}$$

$$= \iint_{-\infty}^{+\infty} \cos^2(\theta_1 + \delta) f_{\Theta_1}(\theta_1) f_{\Delta}(\delta) d\theta_1 d\delta \tag{6.12}$$

The range of equations 6.5 and 6.6 are then used when determining what range the integrals will cover and also inserted into equation 6.12 which is shown in equation 6.13, which leads to equation 6.14.

$$E[Y^2] = \int_{-\infty}^{+\infty} \int_0^{\pi} \cos^2(\theta_1 + \sigma) \left(\frac{1}{\pi}\right) \left(\frac{1}{\sqrt{2\pi}\sigma_n} e^{-\frac{\delta^2}{2\sigma_n^2}}\right) d\theta_1 d\delta \tag{6.13}$$

$$= \frac{1}{\pi} \int_{-\infty}^{+\infty} \frac{1}{\sqrt{2\pi}\sigma_n} e^{-\frac{\delta^2}{2\sigma_n^2}} \left(\int_0^{\pi} \cos(\delta + \theta_1) d\theta_1\right) d\delta \tag{6.14}$$

To solve equation 6.14, a cos integration identity is used. It is shown in equation 6.15.

$$\int \cos(a + x) dx = \frac{a + x}{2} + \frac{1}{4} \sin(2(a + x)) \tag{6.15}$$

Using equation 6.15 it is possible to continue solving equation 6.12 to get equation 6.16.

$$E[Y^2] = \int_{-\infty}^{+\infty} \frac{1}{\sqrt{2\pi}\sigma_n} e^{-\frac{\delta^2}{2\sigma_n^2}} \left(\frac{\delta + \theta_1}{2} + \frac{1}{4}(\sin(\delta + \theta_1)) \right) \Big|_{\theta_1=0}^{\pi} d\delta \tag{6.16}$$

Solving the inner part of equation 6.16 produces equation 6.17.

$$\begin{aligned} \frac{\delta + \theta_1}{2} + \frac{1}{4}(\sin(\delta + \theta_1))\Big|_{\theta_1=0}^{\pi} & \\ &= \frac{\delta + \pi}{2} + \frac{1}{4}\sin(2\delta + 2\pi) - \left[\frac{\delta}{2} + \frac{1}{4}\sin(2\delta)\right] \\ &= \frac{\pi}{2} + \frac{1}{4}\left[\sin(2\delta + 2\pi) - \sin(2\delta)\right] \\ &\because \sin(2\delta + 2\pi) = \sin(2\delta) \\ &\therefore = \frac{\pi}{2} + \frac{1}{4}\left[\sin(2\delta) - \sin(2\delta)\right] \\ &= \frac{\pi}{2} \end{aligned} \tag{6.17}$$

Inserting equation 6.17 into equation 6.16, produces equation 6.18.

$$\begin{aligned} E[Y^2] &= \frac{1}{\pi} \int_{-\infty}^{+\infty} \frac{1}{\sqrt{2\pi}\sigma_n} e^{-\frac{\delta^2}{2\sigma_n^2}} \left(\frac{\pi}{2}\right) d\delta \\ &= \frac{1}{\pi}\frac{\pi}{2}\frac{1}{\sqrt{2\pi}\sigma_n} \int_{-\infty}^{+\infty} e^{-\frac{\delta^2}{2\sigma_n^2}} d\delta \\ &= \frac{1}{2\sqrt{2\pi}\sigma_n} \left[\sqrt{\frac{\pi}{2}}\sigma_n \text{erf}\left(\frac{\delta}{2\sigma_n}\right)\right] \Big|_{\delta=-\infty}^{+\infty} \end{aligned} \tag{6.18}$$

The integration in equation 6.19 was used in equation 6.18.

$$\int e^{-\frac{\delta^2}{2\sigma_n^2}} d\sigma = \left[\sqrt{\frac{\pi}{2}}\sigma_n \text{erf}\left(\frac{\delta}{2\sigma_n}\right)\right] \tag{6.19}$$

Variable substitution is performed, shown in equation 6.20.

$$x = \frac{\delta}{2\sigma_n} \tag{6.20}$$

and is used in solving equation 6.18, which is continued in equation 6.21

$$\begin{aligned} E[Y^2] &= \frac{1}{2\sqrt{2\pi}\sigma_n}\left[\sqrt{\frac{\pi}{2}}\sigma_n\left(\lim_{x\to\infty}\operatorname{erf}(x) - \lim_{x\to\infty}\operatorname{erf}(-x)\right)\right] \\ &= \frac{1}{2\sqrt{2\pi}\sigma_n}\left[\sqrt{\frac{\pi}{2}}\sigma_n\left(1-(-1)\right)\right] \\ &= \frac{1}{2\sqrt{2\pi}\sigma_n}\left[2\sqrt{\frac{\pi}{2}}\sigma_n\right] \\ &= \frac{\frac{\sqrt{\pi}}{\sqrt{2}}}{\sqrt{2}\sqrt{\pi}} \\ &= \frac{1}{2} \end{aligned} \tag{6.21}$$

Where $\operatorname{erf}(x)$ is the error function of x. Now, solving the $E^2[Y]$ term in equation 6.9 is shown below in equation 6.22.

$$\begin{aligned} E_{\theta,\Delta} &= \iint \cos(\theta_1+\delta) f_\Theta(\theta_1) f_\Delta(\delta) d\theta_1\, d\delta \\ &= \iint \cos(\theta_1+\delta)\left(\frac{1}{\pi}\right)\left(\frac{1}{\sqrt{2\pi}\sigma_n}e^{-\frac{\delta^2}{2\sigma)n^2}}\right) d\theta_1\, d\delta \\ &= \frac{1}{\pi}\int_{-\infty}^{+\infty}\frac{1}{\sqrt{2\pi}\sigma_n}e^{-\frac{\delta^2}{2\sigma_n^2}}\int_{\theta_1=0}^{\theta_1=\pi}\cos(\theta_1+\delta)d\theta_1\, d\delta \\ &= \frac{1}{\pi}\int_{-\infty}^{+\infty}\frac{1}{\sqrt{2\pi}\sigma_n}e^{-\frac{\delta^2}{2\sigma_n^2}}\left[\sin(\theta_1+\delta)\right]\Big|_{\theta_1=0}^{\pi} d\delta \\ &= \frac{1}{\pi}\int_{-\infty}^{+\infty}\frac{1}{\sqrt{2\pi}\sigma_n}e^{-\frac{\delta^2}{2\sigma_n^2}}\left[\sin(\delta-\pi)-\sin(\delta)\right] d\delta \\ &= \frac{1}{\pi}\int_{-\infty}^{+\infty}\frac{1}{\sqrt{2\pi}\sigma_n}e^{-\frac{\delta^2}{2\sigma_n^2}}\left[-2\sin(\delta)\right] d\delta \\ &= -\frac{2}{\pi}\int_{-\infty}^{+\infty}\frac{1}{\sqrt{2\pi}\sigma_n}e^{-\frac{\delta^2}{2\sigma_n^2}}\sin(\delta)d\delta \\ &= -\frac{2}{\pi\sqrt{2\pi}\sigma_n}\left[-\frac{1}{2}e^{-\frac{\sigma_n^2}{2}}\sqrt{\frac{\pi}{2}}\sigma_n\left(\operatorname{erfi}\left(\frac{\sigma_n^2-i\delta}{\sqrt{2}\sigma_n}\right)+\operatorname{erfi}\left(\frac{\sigma_n^2+i\delta}{\sqrt{2}\sigma_n}\right)\right)\right]\Big|_{\delta=-\infty}^{+\infty} \end{aligned} \tag{6.22}$$

The $\operatorname{erfi}(x)$ used in equation 6.22 is the imaginary error function of x, and is defined in equation 6.23. As $\delta \to \infty$, the other terms inside the $\operatorname{erfi}(x)$ function in equation 6.22 do not affect the term that is tending toward infinity, they can be considered

insignificant in this case.

$$\text{erfi}(x) \equiv -i\text{erf}(ix) \tag{6.23}$$

The erfi(x) function has the following properties:

$$\begin{aligned}
\lim_{x\to\infty} \text{erfi}(x) &= \infty \\
\lim_{x\to\infty} \text{erfi}(-x) &= -\infty \\
\lim_{x\to\infty} \text{erfi}(ix) &= i \\
\lim_{x\to\infty} \text{erfi}(-ix) &= -i
\end{aligned}$$

Using the last two properties, equation 6.22 becomes

$$\begin{aligned}
E_{\theta,\Delta} &= -\frac{2}{\pi\sqrt{2\pi}\sigma_n}\left[-\frac{1}{2}e^{-\frac{\sigma_n^2}{2}}\sqrt{\frac{\pi}{2}}\sigma_n(0)\right] \\
&= 0
\end{aligned} \tag{6.24}$$

Inserting the results from equation 6.18 and equation 6.24 into the variance equation 6.9, gives equation 6.25.

$$\begin{aligned}
\sigma_Y^2 &= E_{\theta_1,\delta}[Y^2] - E_{\theta_1,\delta}^2[Y] \\
&= \frac{1}{2} - 0 \\
&= \frac{1}{2}
\end{aligned} \tag{6.25}$$

For this test case with a sinusoidal wave of unity amplitude, the variance in the readings will be half. This makes sense, as the distribution used for this mathematical model was normalized, as well as the randomness of the intervals between readings was uniformly distributed. This result means that if readings are taken using non-uniform sampling, and no effort is put in to compensate for the varying time intervals, there will be noise present. As stated, this derivation shows the variance present will be half of the signal introduced.

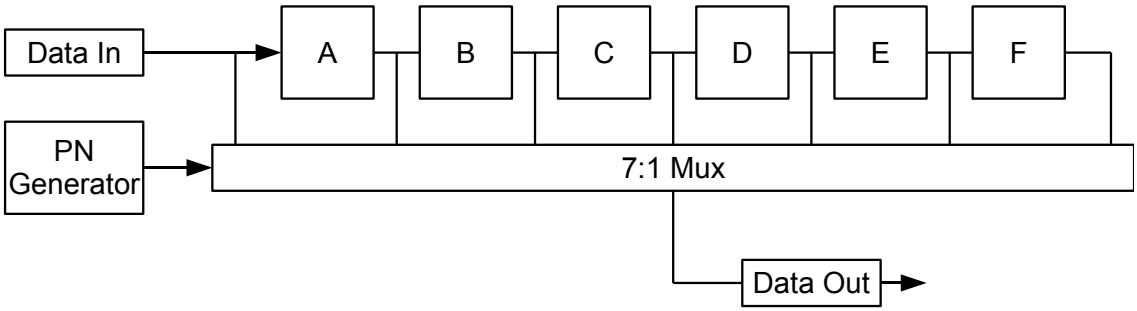

Figure 6.1: Non-uniform sampling implementation.

6.2 Implementation

The non-uniform sampling was implemented using a type of shift register system seen in Figure 6.1. The input was captured by the ADC and appeared as din. When new data was introduced, the initial value was shifted into the shift register and the new data appeared as *din*.

Using a linear recursive sequence generator with the equation $P(x) = x^5+x^2+1$ to implement a PN generator, registers were chosen at random and their outputs were connected to the module outputs. Because $P(x)$ is a primitive prime polynomial, the maximum sequence length of 31 states was achieved. The rate at which registers were chosen is controlled by the clock that the PN generator uses to advance its own internal registers. A relationship appeared between aliasing in the noise and the clock speed. When the clock was set to choose a different register at 125 MHz, there were double peaks around every 2.016 MHz. However, while running the PN generator clock at a higher frequency—at 200 MHz—there are only single peaks with less separation. A possible explanation to this phenomenon is the period that the PN sequence repeats. At 125 MHz, $P(x)$ repeated every 248 ns, while running at 200 MHz it repeated every 155 ns.

The noise introduced using this method was noted as appearing as an ordered noise; there appear to be many aliasing effects at discrete intervals.

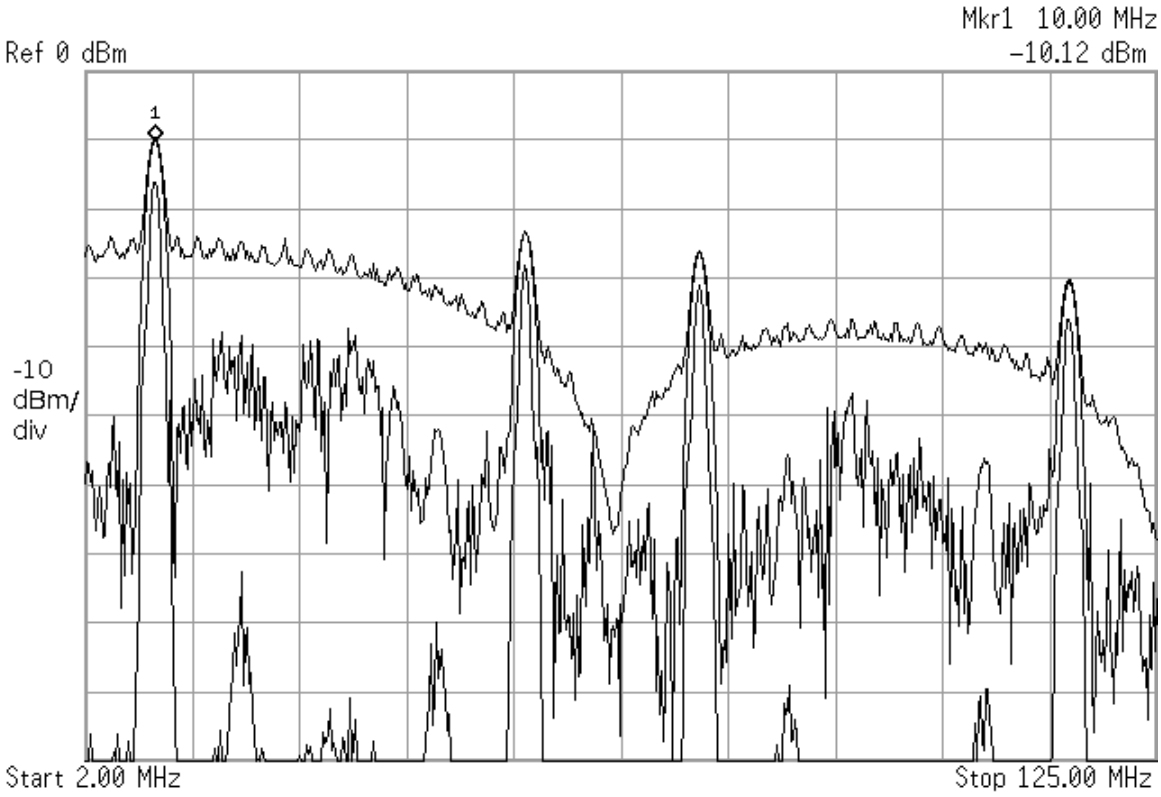

Figure 6.2: Showing the maximum value (top line), an instantaneous reading (middle line), and the minimum value (bottom line) of a 10 MHz signal using uniform sampling.

6.3 Known Signal Tests

To show the noise that is introduced, a 10 MHz sine wave was used as an input signal. From there, various non-uniform sampling parameters were used. For reference, Figure 6.2 shows the output of the 10 MHz signal that was sampled uniformly through the FPGA. The tests were performed by applying the sine wave to the ADC input, processing the signal in the FPGA, outputting the result using a DAC, which is hooked directly to a spectrum analyzer for observation.

Input signals are sampled at 62.5 MHz by the ADCs, and output at 62.5 MHz by the DACs. During the up–sampling that was performed internally in the FPGA in this test, the signal was not zero padded, but repeated by reading the ADC output register multiple times before a new reading was taken. This causes the signal to be

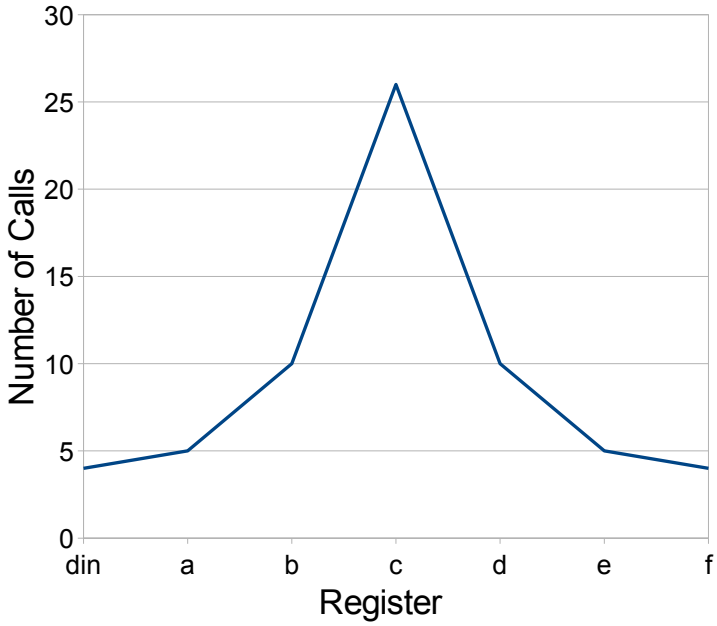

Figure 6.3: A normal distribution showing how many times each register is called per period.

repeated in the frequency domain above 62.5 MHz. Because these secondary signals are so far out of band, they do not interfere with the tests that are being performed at 10 MHz. The signal was then downsampled at the outputs by the DACs.

6.3.1 Test Results

Once non-uniform sampling was applied more noise was obviously present in the system. For the first test, a PN generator with a normal distribution was used. The module as implemented takes the input from all the registers in a normalized distribution; the centre buffer gets chosen the majority of the time, while the far end buffers get chosen significantly less. This can be seen in Figure 6.3, where din is the input register, and the following registers are lettered in alphabetical order.

As can be seen in Figure 6.4, the maximum level noise has risen approximately 5 dBm near the 10 MHz range. The minimum noise has also risen 30 dBm around 10 MHz. The instantaneous reading of the signal has also risen. Also of note is the

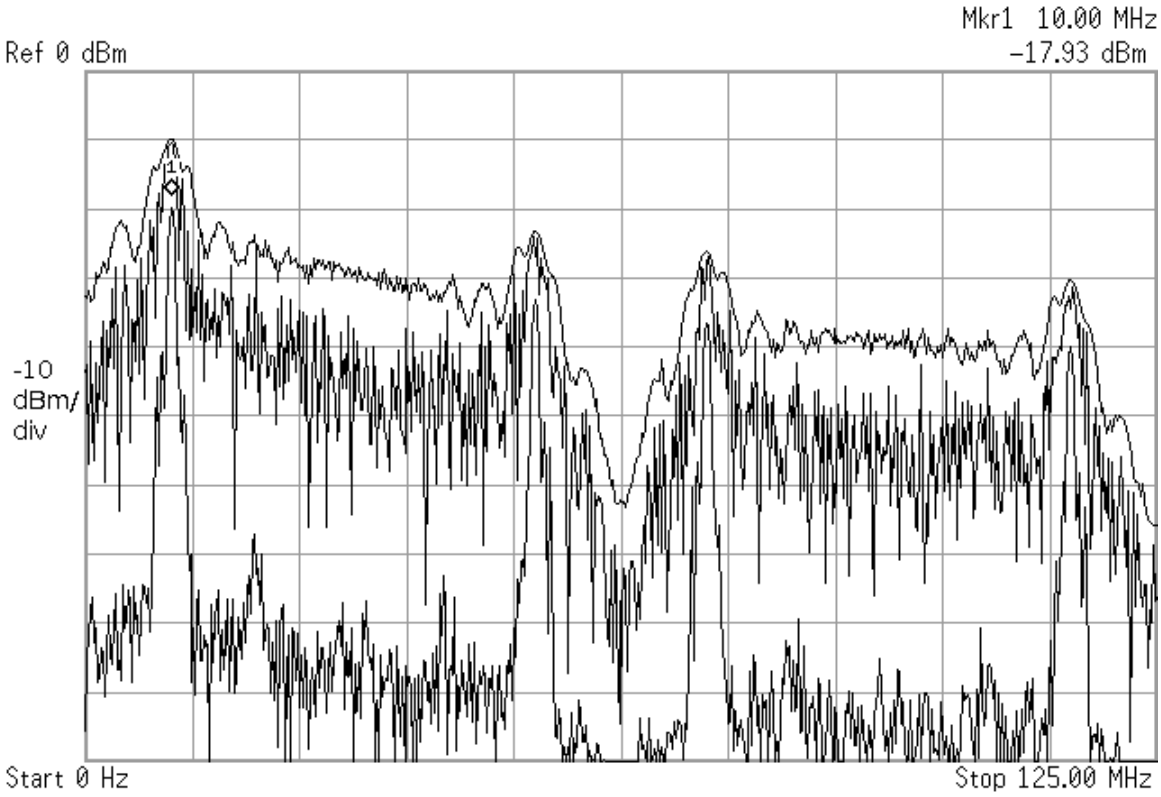

Figure 6.4: Showing the maximum value (top line), an instantaneous reading (middle line), and the minimum value (bottom line) of a 10 MHz signal using non-uniform sampling.

peak at 10 MHz is less pronounced and has slightly increased width. The 10 MHz signal is approximately -10 dBm in both the uniform and non-uniform sampled cases. The input signal used was at 0 dBm; the -10 dBm discrepancy was mentioned earlier and is consistent with results found earlier.

For the next test, the distribution was modified so that the middle register in the shift array was called more often than during the first test, and the others less. This distribution is shown in Figure 6.5

The resulting spectrum is shown in Figure 6.6. There is almost no discernable difference between the modified distribution and the initial distribution.

Finally, a nearly evenly distributed selection algorithm was used, and is shown in Figure 6.7.

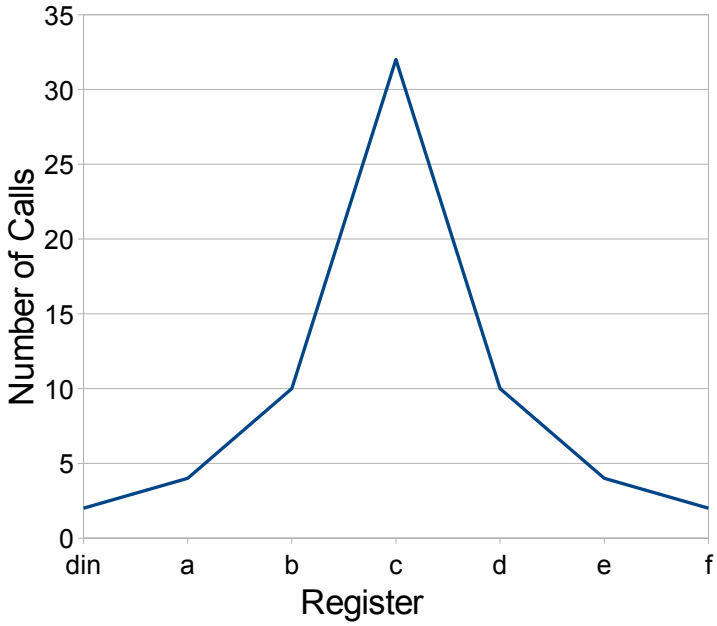

Figure 6.5: Modified distribution.

As can be seen again in Figure 6.8, there is no obvious difference between the even distribution and the previous two normalized distributions.

6.3.2 Discussion

This implementation does not guarantee order is preserved when the output is connected to another buffer in the shift register. This could be the reason why the output signals seem to all have similar amounts of noise present. As mentioned earlier, the rate at which the output connection is changed seems to have more of an effect on the output signal.

In each of the three non-uniform sampling tests performed using a 10 MHz sine wave input signal the minimum value, or noise floor, was also increased. Looking only in the first half of the spectrum shown and excluding the second hump, the noise was increased to nearly a uniform value of about -70 dBm and varied by only 10 dBm, except near the null point at 62.5 MHz.

In observing the non-uniform sampling effects on the test signal, it can be seen

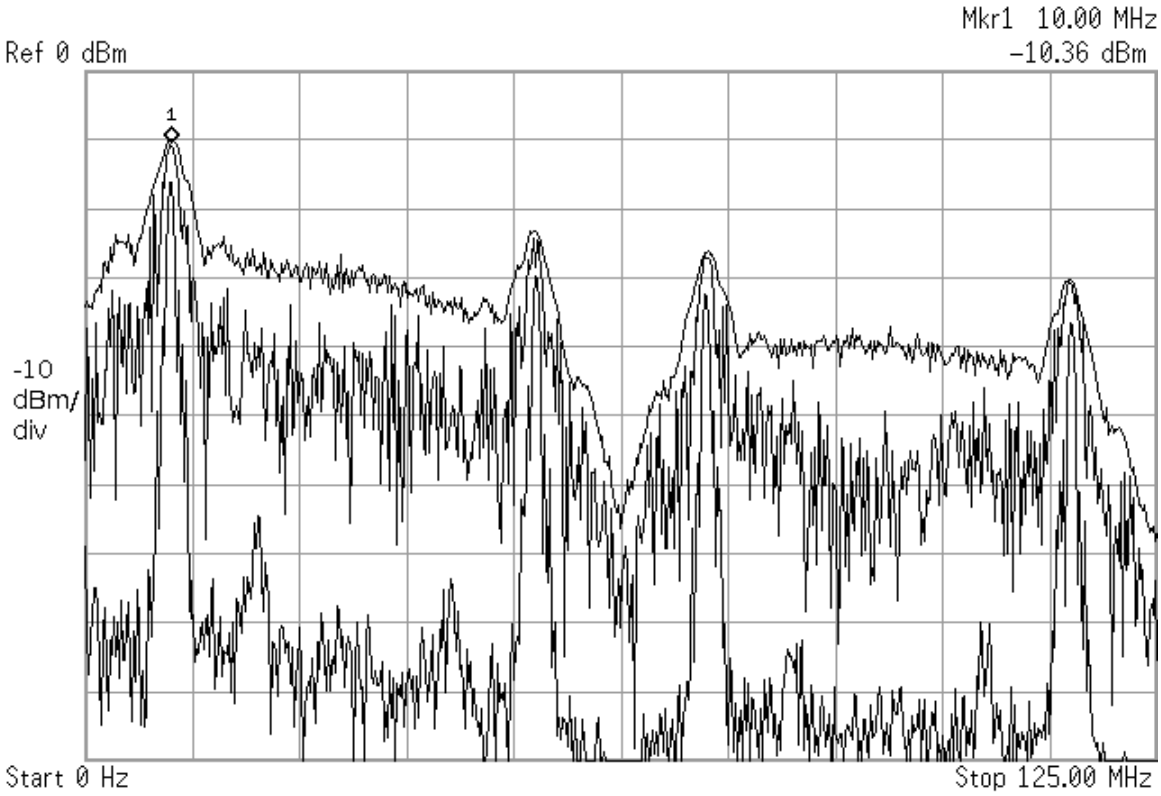

Figure 6.6: The resulting maximum value (top line), an instantaneous reading (middle line), and the minimum value (bottom line) of the spectrum using the modified normal distribution.

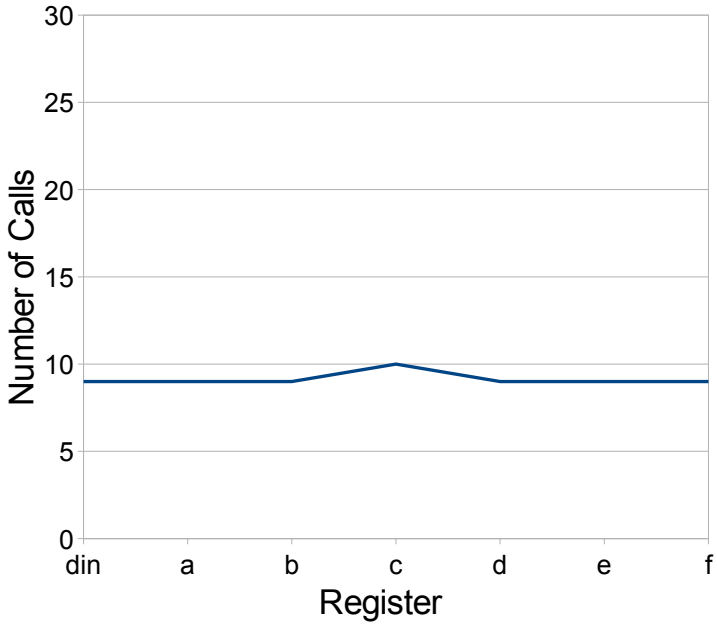

Figure 6.7: The nearly even distribution used for register selection.

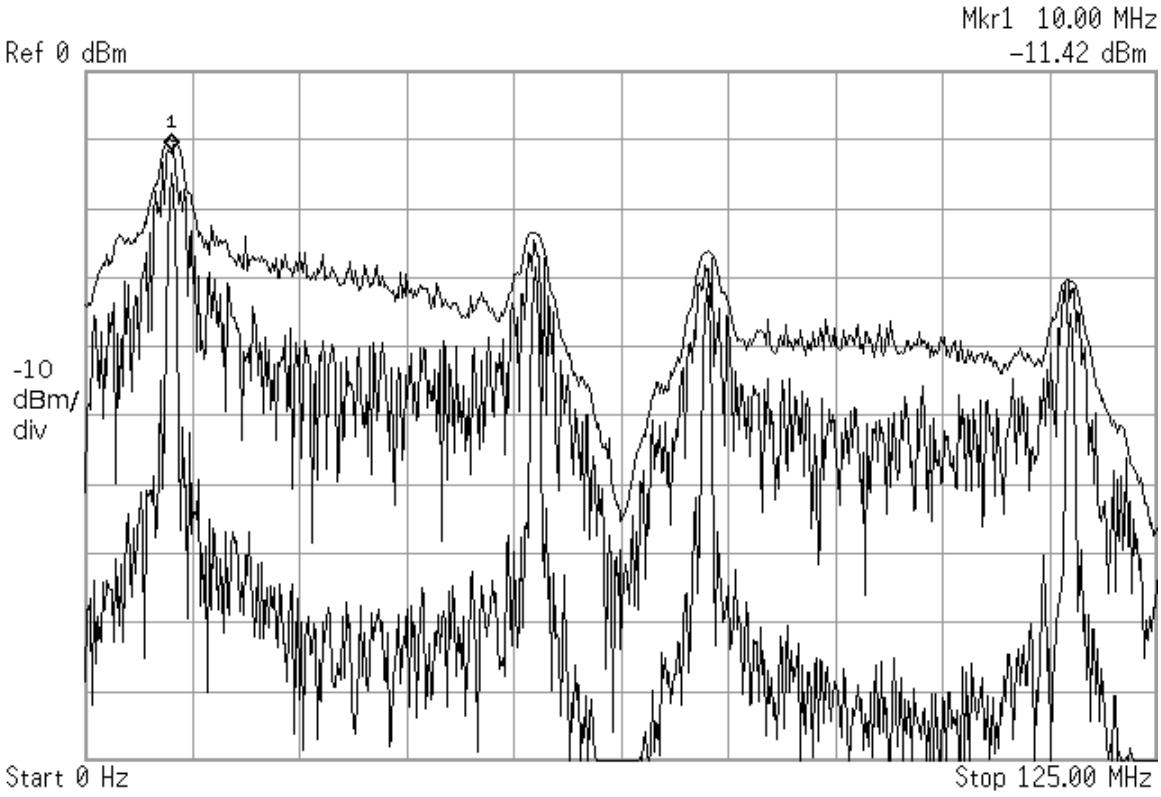

Figure 6.8: The resulting maximum value (top line), an instantaneous reading (middle line), and the minimum value (bottom line) of the spectrum using the nearly even distribution.

that the maximum value of the spectrum (the top line in the Figures 6.4, 6.6, and 6.8) had only been slightly affected. The point at 10 MHz where the signal of interest resides has been widened in frequency, showing that noise is introduced around that area.

Simply stated, if there is no signal there will be no noise added, and where there is noise, the non-uniform sampling will add noise to that signal. Much research has been done and most of it appears to be focused on acquiring a signal and reproducing that signal free of noise, but also free of any harmonics. An example of this process is described by Brueller et al. where a mathematical model and an iterative estimation process is used to describe and recreate a signal [16]. Fares et al. showed a few methods of using non-uniform sampling and regenerating the signal using uniform methods [17]. That approach, however, requires that the order in which the signal is sampled be guaranteed. This was not the case in this test; instead of trying to clean the spectrum of noise, the goal was to introduce it. The notion of using non-uniform sampling and uniform sample generation to introduce noise in a specific channel in use is what sets this test apart from other literature.

6.4 Application to Architecture

To establish a baseline, Figure 5.7 shows the communications channel of an FTAP test being performed and data being transfered. The first and highest peak shown is the channel being used, the second peak situated at approximately 1930 MHz is the frequency used by a clock source for modulation and demodulation.

During the tests using non-uniform sampling, the CDMA/EVDO system was not able to continue to transfer data, and resulted in all packets being lost, and eventually, the session between the phone and base station was also dropped. This indicates a limit to the amount of noise which can be introduced before failure. Shown

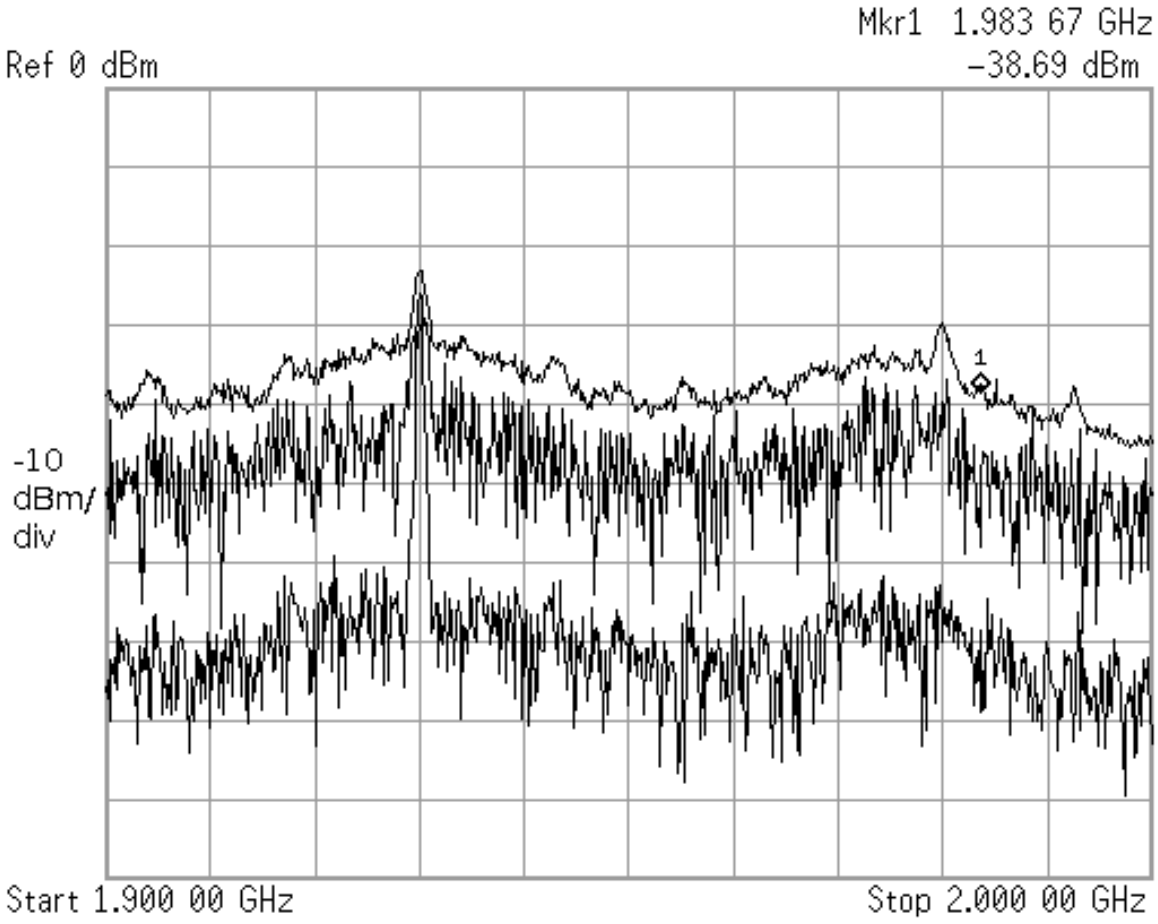

Figure 6.9: Noise dominating both forward and reverse channels.

in Figure 6.9 is the communications channel being dominated with noise during a non–uniform sampling test. The noise pattern is nearly identical for each of the 3 different non-uniform sampling scenarios used.

6.5 Noise Parameter Discovery

The previous tests implemented only non–uniform sampling with a signal generation rate equal to the signal acquisition rate. Also, the PN generator ran at a much higher rate from the rest of the system. The following is an attempt to specify which parameters could be modified to slowly introduce noise to the communication channel.

For these tests, the input signal was acquired at 62.5 MHz and the output signal was generated at 125 MHz. Up–sampling to 250 MHz and downsampling to the output frequency of 125 MHz was performed internally in the FPGA. All modifications to the signal were performed at 250 MHz.

6.5.1 Modification to PN Generator

The period of the PN generator was modified by using a higher order polynomial in the PN generator. This increased the maximum length sequence of which registers could be chosen; a smaller order polynomial used in the PN generator decreased the maximum length of the sequence.

Several PN generators were created; PN generators from 10th order down to 3rd order. The polynomials describing all PN generators used for testing are as follows [18]:

$$P(x) = x^{10} + x^3 + 1 \tag{6.26}$$

$$P(x) = x^9 + x^4 + 1 \tag{6.27}$$

$$P(x) = x^8 + x^4 + x^3 + x^2 + 1 \tag{6.28}$$

$$P(x) = x^7 + x + 1 \tag{6.29}$$

$$P(x) = x^6 + x + 1 \tag{6.30}$$

$$P(x) = x^5 + x^2 + 1 \tag{6.31}$$

$$P(x) = x^4 + x + 1 \tag{6.32}$$

$$P(x) = x^3 + x + 1 \tag{6.33}$$

Table 6.1 shows the length of each sequence. Primitive polynomials were chosen to ensure that the PN generator would have the maximum length sequence. This

Table 6.1: Length of sequences

Equation	Sequence Length
6.26	1023
6.27	511
6.28	255
6.29	127
6.30	63
6.31	31
6.32	15
6.33	7

was implemented in an effort to discover and isolate the parameter that affects the amount of noise introduced. For this a shorter shift register, consisting only of three registers, was introduced. Having all but two of the numbers supplied by the PN generator choose the middle register changed the distribution based on length of the PN sequence.

Figure 6.6 shows the basic design of the setup used for doing these tests. The incoming signal was upsampled by a factor of 4 and zero–padded, then passed through a $\mathrm{sinc}(x)$ filter which interpolated and smoothed the upsampled signal before the data was put into the shift register. As data moved through the shift register, the PN generator selected which register to attach the output to. The PN generator was running at the same frequency as the incoming signal to the shift register. Downsampling by a factor of 2 was then performed to prepare the signal for generation by the DACs.

The results from these tests show that noise is only somewhat dependent on the shape of the distribution; the noise that is generated appears in a non–random fashion. Shown in Figure 6.11 is the average spectrum observed while testing the PN generator described in equation 6.33, and the results of the longer PN generator described in equation 6.26 are shown in the averaged Figure 6.12. The noise introduced by the longer PN sequence is approximately 2 dBm higher than the shorter sequence, however, discrete peaks of noise are clearly seen.

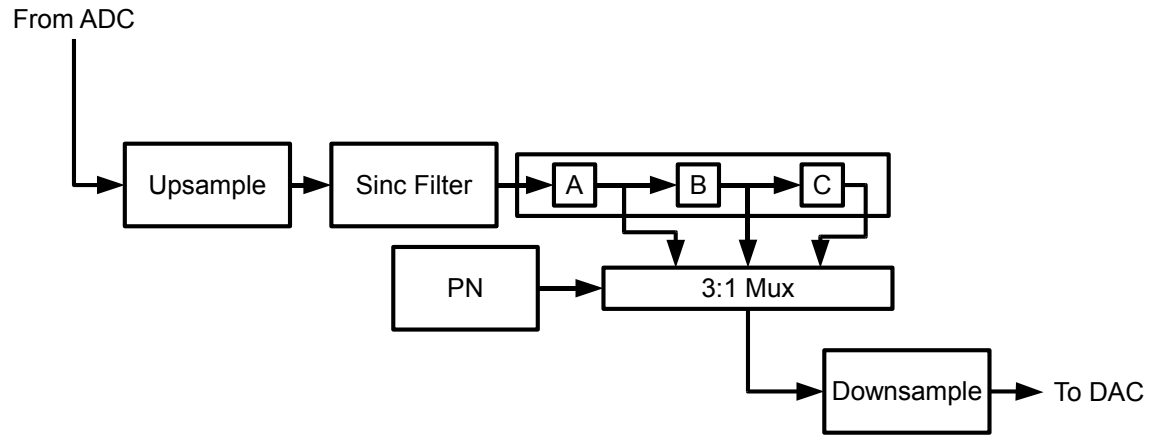

Figure 6.10: An Implementation of non–uniform sampling.

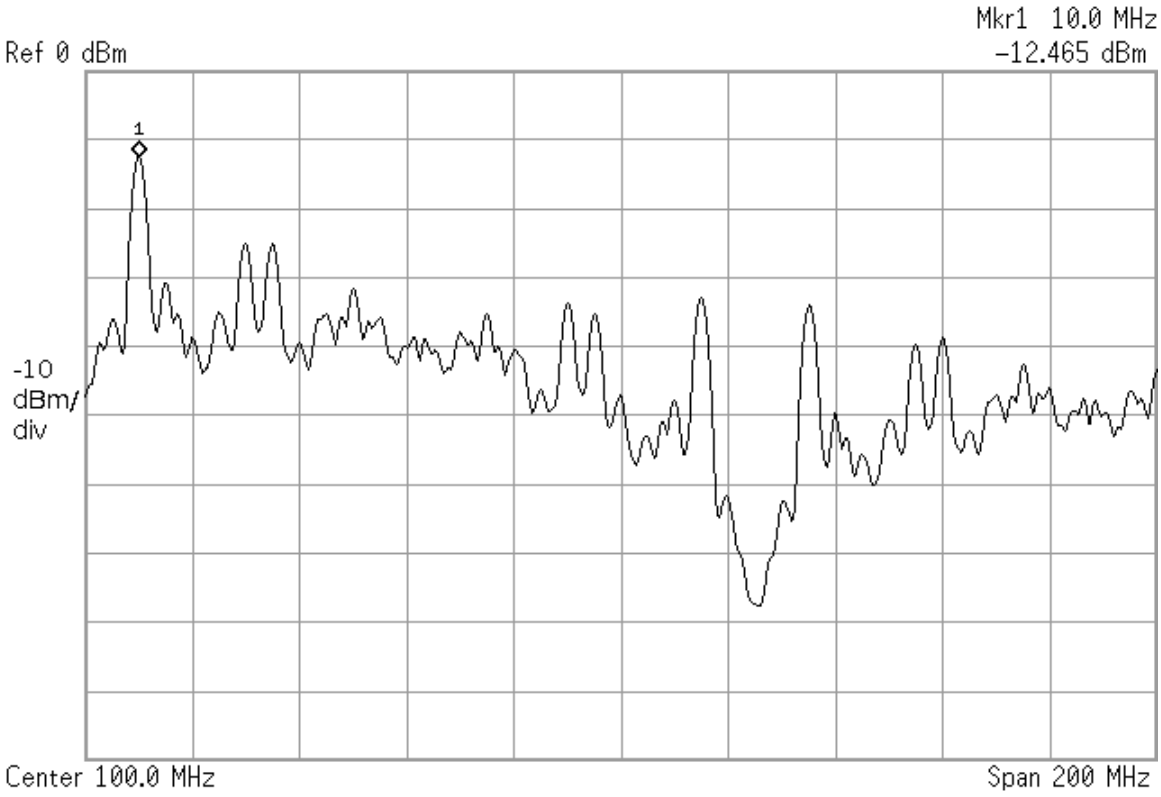

Figure 6.11: Output when equation 6.33 was implemented.

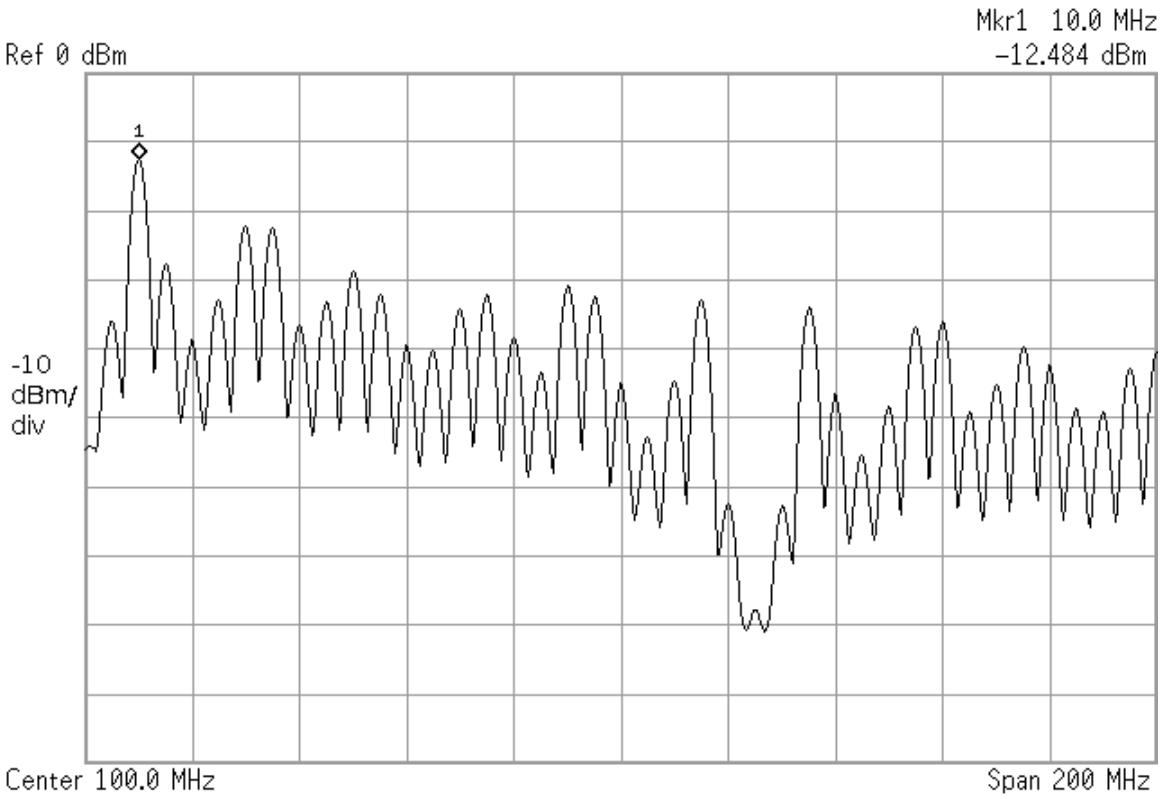

Figure 6.12: Output when equation 6.26 was implemented.

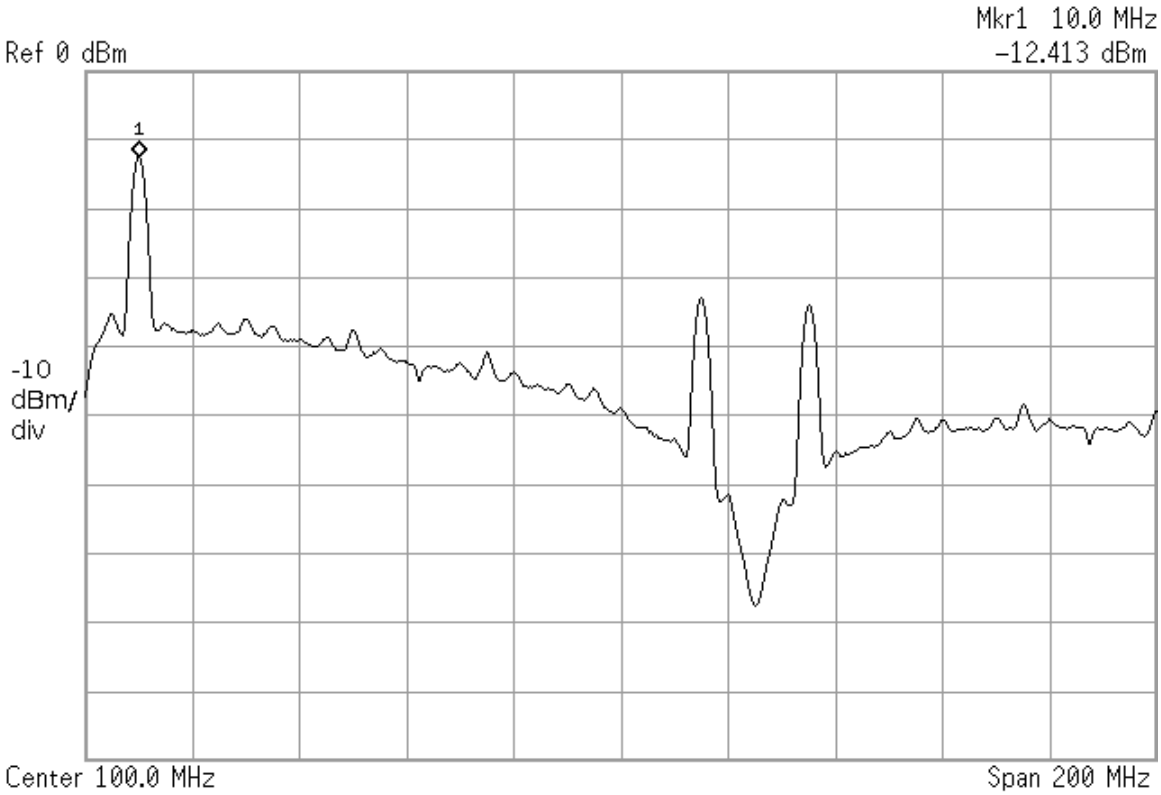

Figure 6.13: Output for 3 registers.

6.5.2 Length of Shift Register

To test the effects changing the number of shift registers would have on the output, a uniform distribution was used when determining what register to take output from. Scenarios with 3, 5, 13 and 25 registers were tested. Varying the number of registers did have a profound impact on the channel being used during communication. As the number of registers increased, more noise was introduced to the channel. Figure 6.13 shows the output signal when there were 3 shift registers used, Figure 6.14 shows the output when 5 shift registers are used, and finally, Figure 6.15 show when there were 25 shift registers being used.

As seen in Figure 6.13, the added noise in minimal. For a 10 MHz signal sampled at 62.5 MHz, approximately half a wavelength was in the shift registers. Since the values that were resampled were taken from a local area, the noise introduced

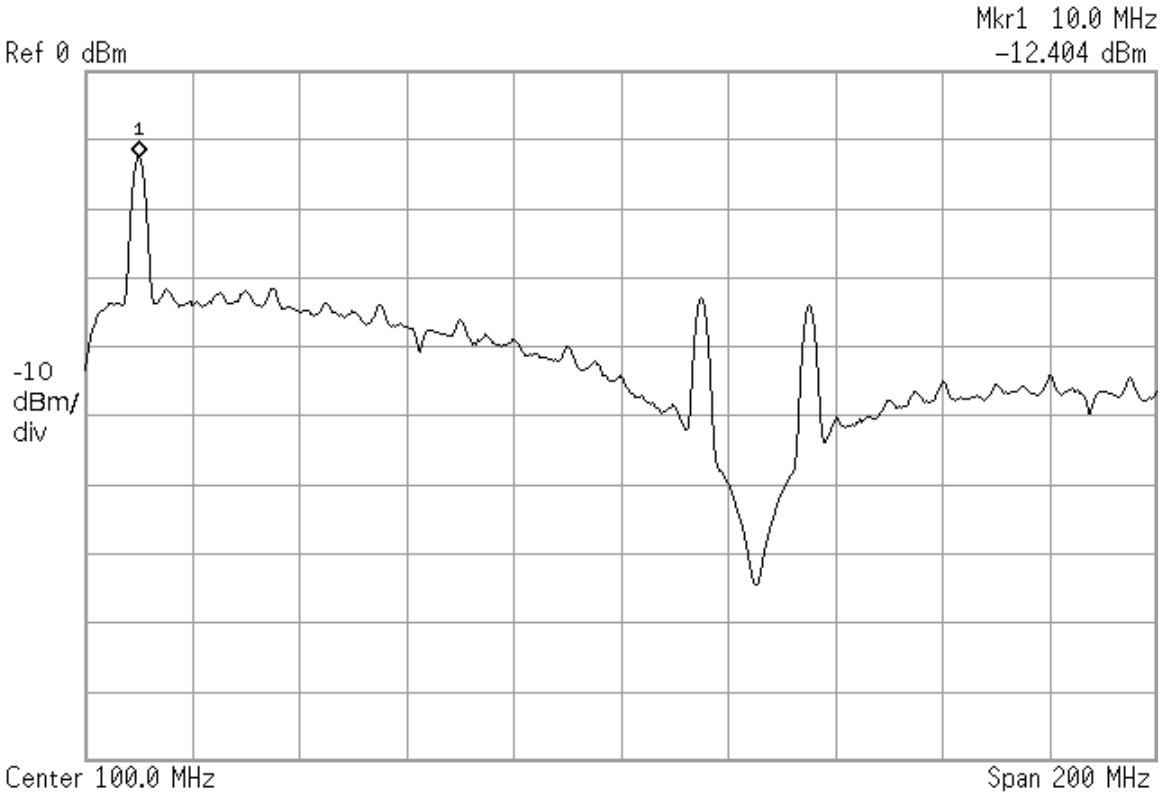

Figure 6.14: Output for 5 registers.

was minimal.

Figure 6.14 represents a non–uniform sampling system with 5 shift registers. This allows for just less than a full wavelength in the shift registers. Because of this, the values that are written to the output have the potential to come from a slot almost a full wavelength away, the noise added is greater than before.

In the extreme case of having 25 shift registers, the non–uniform sampling module has the ability to hold 4 complete periods of signal in the shift registers at any time. Because the output can be driven from anywhere in the 4 wavelengths, the noise in the output can be seen in Figure 6.15 completely suppressing the 10 MHz test signal.

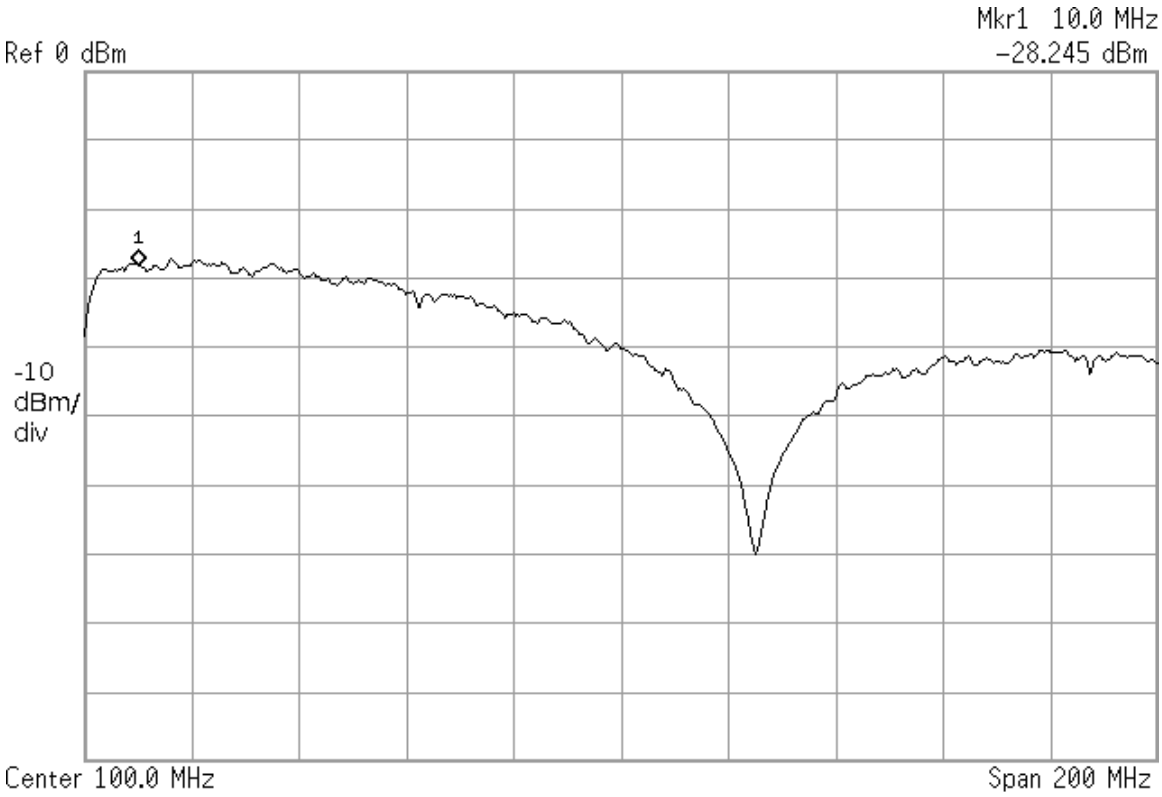

Figure 6.15: Output for 25 registers.

6.5.3 Relation to Theoretical Results

To relate the theoretical results discussed in Section 6.1 to the observed results, the cases of 3 and 5 shift registers are used. In the case of 3 input registers, the input variance $\sigma_i^2 = \frac{2}{3}$. Applying the result found in equation 6.25, the output variance becomes $\sigma_o^2 = \frac{1}{3}$. Moving to the case with 5 registers used in the shift register, the input variance $\sigma_i^2 = 2$. Again, applying equation 6.25, the output variance becomes $\sigma_o^2 = 1$. The difference between the two theoretical output values is an increase of a factor of 3. When this difference is calculated on a logarithmic scale, the difference is represented as 4.8 dBm.

Looking at the screen capture results from both of those cases shown in Figure 6.13 and Figure 6.14, it is seen that the difference in noise level between those two tests was approximately 5 dBm. This matches closely with the theoretical result.

6.5.4 Effect on CDMA/EVDO Traffic

Non–uniform sampling was gradually introduced into an RTAP data–transfer test using the same configuration as the pass–through tests performed in Section 5.5. For the most part, the CDMA/EVDO system was able to continue transferring with the noise introduced. Figure 6.16 shows the results of the no noise as well as the four different noise levels at 50 second intervals introduced in Section 6.5.2. The first section indicated, shows a normal data transfer, with no noise. When the minimum level of noise tested, represented by 3 registers with a uniform distribution, was suddenly introduced by toggling a switch there was no effect on the data transfer. Increasing the number of registers to 5 resulted in a delayed sudden drop in the data transfer, but the transfer immediately recovered and continued. When the number of registers was increased to 13, the transfer became intermittent as shown. Finally, when the shift register was increased to 25 registers, the transfer was immediately cut off. After testing the maximum level of noise, if the noise was not immediately

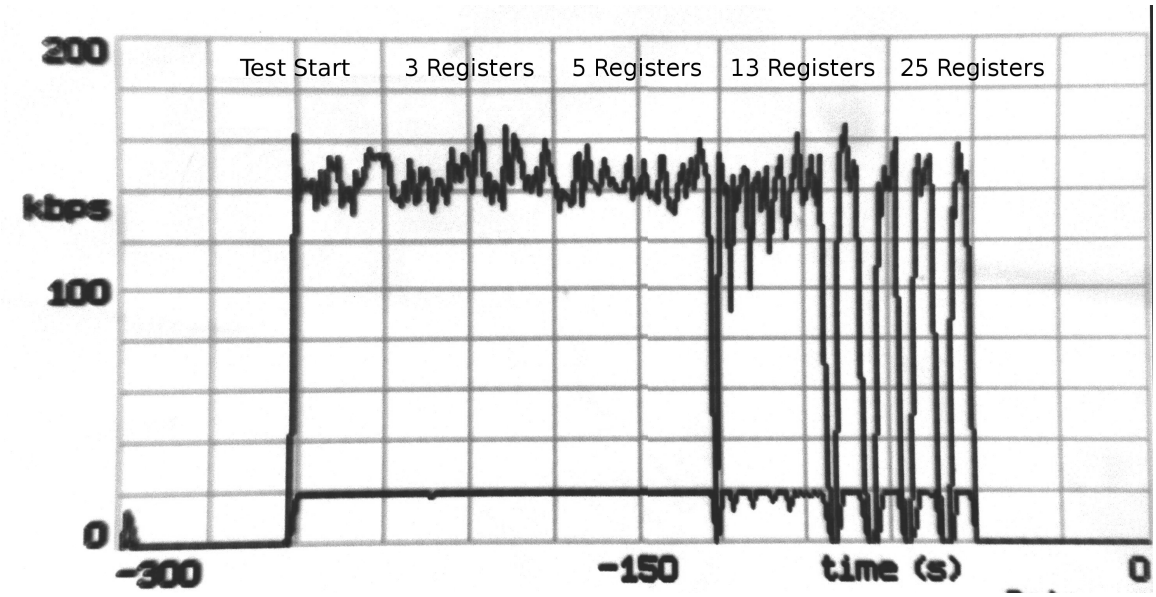

Figure 6.16: RTAP data transfer with non–uniform sampling noise added. Tests were done in 50 second intervals.

removed from the system after it was introduced, the connection would have to be reset; once the transfer had been sufficiently interrupted, the data connection was permanently lost. During power control tests, the open–loop power control test was repeated but was not able to respond within the envelope required. This was the case when any level of noise was introduced.

This test shows the resilience of the CDMA/EVDO standard against sudden increases in noise, and its ability to cope in order to continue to function. Also, an upper limit was found where the channel was dominated with noise in which no communication could take place. Because this noise was added in both the forward and reverse link, it is also possible that this is a limitation of the phone's ability to cope with the added noise.

Chapter 7

Summary and Future Work

7.1 Summary of Completed Work

This thesis provides a fully functioning cost–effective implementation of a means to access a communications medium to control the signal passing through it which is built using mostly COTS components. Tested first was the ability to receive and send a known signal, followed by receiving and sending CDMA/EVDO communications. This channel could be viewed using the SignalTap® facilities built into the FPGA, or using a splitter/combiner and a spectrum analyzer. The final implementation met all the requirements discussed in Section 2.2.

Intelligent selection of modulation and demodulation frequencies ensured that there was no interference with the removal of the LPF. By removing the LPF, the frequency range that can be used with this design is increased.

The discovery that the system would still work even when the communications were demodulated down to baseband and acquired by the FPGA in a laboratory setting was an interesting phenomenon. While the signal is being demodulated down to base band, a local fade appears near the DC point where the ADCs have a documented inability to resolve the signal. In this case, it is suspected that the ability for the base

station and mobile to still exchange data is a result of the CDMA/EVDO standard and the implementation by the manufacturers being robust enough to handle these situations.

Non–uniform sampling was implemented to show that the system could introduce noise to the point where the power control algorithm used in the CDMA/EVDO implementation would fail and that the data transfer could be incrementally affected or stopped. During this investigation, the discovery that the non–uniform sampling would only increase the noise where a signal was present under certain conditions was made. A method of slowly introducing noise was also discussed; this is a low complexity method of introducing noise.

7.2 Future Work

In depth analysis of the aliasing that appeared during the non–uniform sampling was outside the scope of this thesis, and was only briefly investigated. Upon close inspection, it appeared that the noise introduced was somewhat ordered; the noise appeared to be made up of aliasing artifacts. This endeavour was undertaken to show that it was possible to affect the power control used in the system by introducing noise, and could be further researched.

Since this architecture is protocol agnostic it would be possible to test other wireless protocols. Tests do not need to be confined to verifying the power control algorithm; this architecture makes it possible to implement any test that requires unrestricted access to the communication channel. Similar tests could be performed on a mobile network based on most modern mobile radio systems.

Bibliography

[1] T. Rappaport, *Wireless Communications Principles and Practice*. Prentice Hall Communications Engineering and Emerging Technologies Series, Upper Saddle River, NJ: Prentice Hall PTR, second ed., 2002.

[2] T. Chulajata and H. Kwon, "Combinations of power controls for cdma2000 wireless communications system," *Vehicular Technology Conference, 2000. IEEE VTS-Fall VTC 2000. 52nd*, vol. 2, pp. 638–645, 2000.

[3] M. Sim, E. Gunawan, C. Soh, and B. Soong, "Characteristics of closed loop power control algorithms for a cellular DS/CDMA system," *IEE Proceedings-Communications*, vol. 145, pp. 355–362, October 1998.

[4] L. Korowajczuk, B. de Souza Abreu Xavier, A. M. F. Filho, L. Z. Ribeiro, C. Korowajczuk, and L. A. D. Silva, *Designing cdma2000 Systems*. John Wiley and Sons, Ltd., 2004.

[5] K. Etemad, *cdma2000 Evolution: System Concepts and Design Principles*. John Wiley and Sons, Inc., 2004.

[6] S. Niida, T. Suzuki, and Y. Takeuchi, "Experimental results of outer-loop transmission power control using wideband-CDMA for IMT-2000," *Vehicular Technology Conference Proceedings, 2000. VTC 2000-Spring Tokyo. 2000 IEEE 51st*, vol. 2, pp. 775–779, 2000.

[7] D. Agarwal, C. Anderson, and P. Athanas, "An 8 ghz ultra wideband transceiver prototyping testbed," in *The 16th IEEE International Workshop on Rapid System Prototyping, 2005*, pp. 121–127, June 2005.

[8] T. Shono, Y. Shirato, H. Shiba, K. Uehara, K. Araki, and M. Umehira, "Ieee 802.11 wireless lan implemented on software defined radio with hybrid programmable architecture," *IEEE Transactions on Wireless Communications*, vol. 4, pp. 2299–2308, September 2005.

[9] Terasic, *Altera DE3 Prototyping System User Manual.* Terasic Technologies, HsinChu County, Taiwan, 2008.

[10] Hittite, *SiGe Wideband Direct Modulator RFIC Datasheet.* Hittite Microwave Corporation, Chelmsford, Massachusetts, USA, 2008.

[11] Hittite, *SiGe Wideband Direct Demodulator RFIC Datasheet.* Hittite Microwave Corporation, Chelmsford, Massachusetts, USA, 2008.

[12] Terasic, *High–Speed A/D and D/A Development Kit.* Terasic Technologies, HsinChu County, Taiwan, 2008.

[13] Analog, *14–Bit Dual Analog–to–Digital Converter Datasheet.* Analog Devices, Inc, Norwood, Massachusetts, USA, 2005.

[14] Analog, *14–Bit Dual Digital–to–Analog Converter Datasheet.* Analog Devices, Inc, Norwood, Massachusetts, USA, 2006.

[15] J. A. Harriman, "A reconfigurable four-channel transceiver testbed with signalling-wavelength-spaced antennas," Master's thesis, Dept. of Electrical and Computer Engineering, University of New Brunswick, Fredericton, NB, Canada, September 2006.

[16] N. Brueller, N. Peterfreund, and M. Porat, "On non uniform sampling of signals," in *IEEE International Symposium on Industrial Electronics Proceedings ISIE'98.*, vol. 1, pp. 249–252, July 1998.

[17] H. Fares, M. Ben-Romdhane, and C. Rebai, "Non uniform sampled signal reconstruction for software defined radio applications," in *2nd International Conference on Signals, Circuits and Systems*, pp. 1–6, November 2008.

[18] W. W. Peterson, *Error-Correcting Codes.* Cambridge, Massachusetts: M.I.T. Press, 1961.

[19] J. Mar and H.-Y. Chen, "Performance analysis of cellular cdma networks over frequency-selective fading channel," *IEEE Transactions on Vehicular Technology*, vol. 47, pp. 1234–1244, November 1998.

Appendix A

Bias Board Schematic

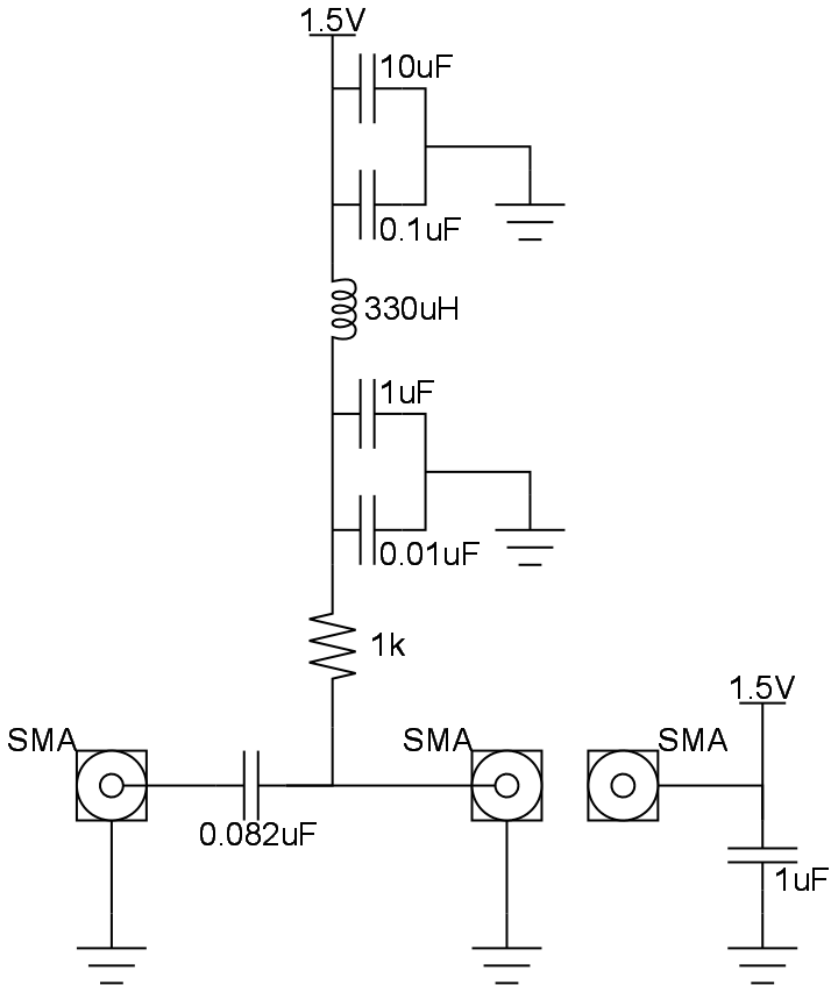

Figure A.1: Bias board schematic.

Appendix B

Detailed Wiring Diagram

A diagram showing the wired connections made in this project is seen in figure B.1.

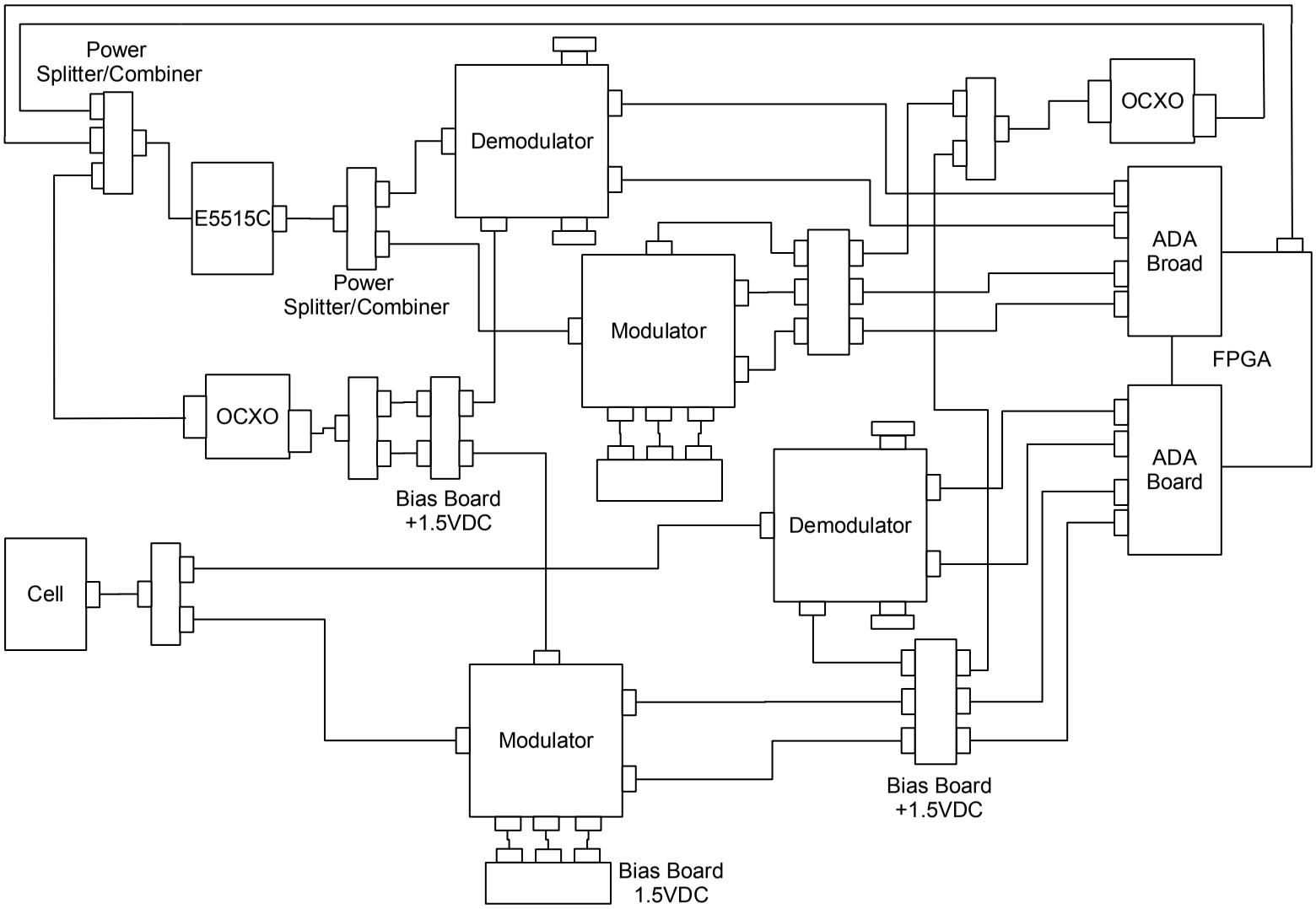

Figure B.1: Wire connections.

Appendix C

Final Project Functional Design

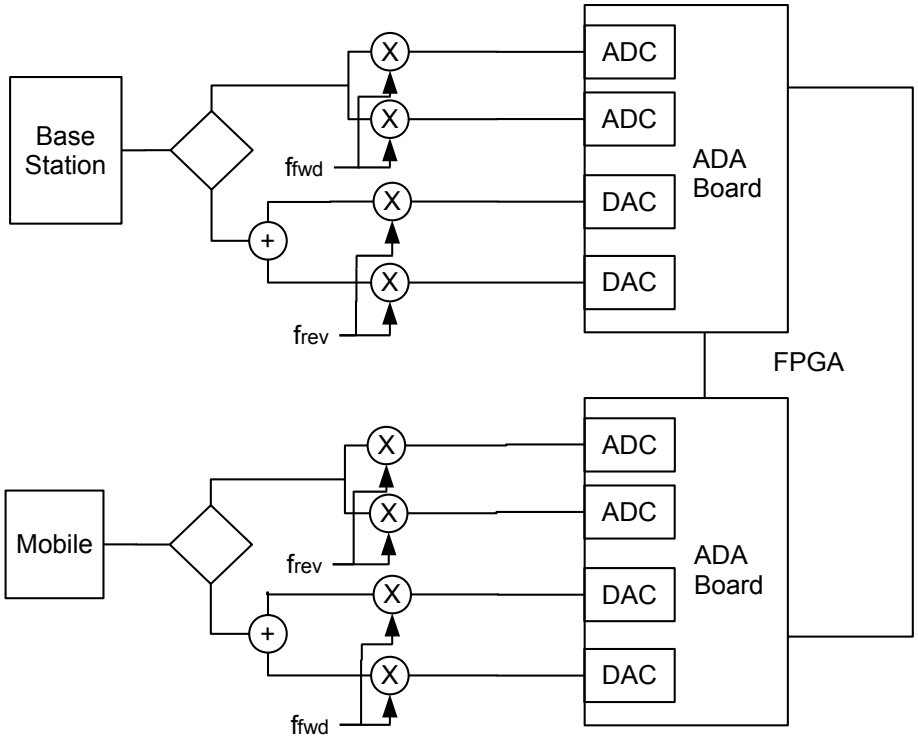

Figure C.1: Functional design of the final project.

Appendix D

MATLAB® Code Listing

D.1 spec1.m

```
clf ;
clear ;
clear functions ;
pause on;

%%%%%Forward%%%%%%%
startf = 1870;
endf = 2060;

%%%%%Reverse%%%%%%
%startf = 1900;
%endf = 1960;

for loopf = startf:1:endf;

    % All fs in MHz.

    %fset = -2000:0.5:(-1950) ;
    %fset =  1880:0.05:1910 ;
    %fset = -4500:0.5:4500 ;
    %fset = -100:0.5:100 ;
    fset = -60:0.5:60 ;

    lowpassbw = 5 ;

    fc = loopf;

    fs = 62.5;

    Nfs = fix(4100/fs) ;
```

```
31
32      y5 = 0 ;
33      for i = -Nfs:Nfs
34          y5 = y5 + pb2(fset+i*fs,fc) ;
35      end
36      plot(fset,y5) ;
37      grid ;
38      display(fc);
39      pause;
40
41  end
42
43  display(fs);
```

D.2 pb2.m

```
1   function y=pb2(f,fc)
2   %PB2    y = pb2(f,fc) is C2 PCS passband spectrum after an fc
3   %       demodulator.  f is in (MHz).
4   %
5   %   Example:
6   %       f  =  [-4500 : 0.1 : 4500 ] ;  % (MHz)
7   %       fc = 1995 ;  % (MHz)
8   %       y  =  pb2(f,fc) ;
9   %       plot(f,y) ;
10
11  yraw =   pb1(f+fc) ...
12         + pb1(f-fc)     ;
13
14  % Pass 0 to 3000 (MHz) at amplitude 1.0
15  % Drop outside 3000 (MHz) to 0.1
16  lowpassmodbw = 4000 ;
17  yhititte = 0.9 * rect(f,lowpassmodbw) + 0.1 ;
18
19  y = yhititte .* yraw ;
```

D.3 pb1.m

```
1   function y=pb1(f)
2   %PB1    y = pb1(f) is C2 PCS passband spectrum.  f is in (MHz).
3   %       1900-1905 1980-1985
4   %               R         F
5   %
6   %   Example:
7   %       f  =  [-2500 : 0.1 : 2500 ] ;  % (MHz)
8   %       y  =  pb1(f) ;
9   %       plot(f,y) ;
10
11  lowpassbw = 5 ;
12
```

```
% y = rect(f+1987.5,lowpassbw/2) + ...
%     rect(f+1902.5,lowpassbw/2) + ...
%     rect(f-1902.5,lowpassbw/2) + ...
%     rect(f-1987.5,lowpassbw/2)       ;


% Forward 1980-1985
yf = (2.5/1.75) * pulsef((-f)-1982.5) + ...
     (2.5/1.75) * pulsef(( f)-1982.5)       ;

% Reverse 1900-1905
% yr = rect(f+1902.5,lowpassbw/2) + ...
%      rect(f-1902.5,lowpassbw/2)       ;
yr = 1 * pulser((-f)-1902.5) + ...
     1 * pulser(( f)-1902.5)       ;


y = yf + yr ;
```

D.4 pulsef.m

```
function y=pulsef(f)
%BP_PULSEF  y = pulsef(f) is a pulse shape.  f is in (MHz).
%
%   Example:
%       f  =  [-10 : 0.1 : 10 ] ;  % (MHz)
%       y  =  pulsef(f) ;
%       plot(f,y) ;

lowpassbw = 5 ;

m = 0.1 ;
b = 1.5 ;

y = (m*f+b) .* rect(f,lowpassbw/2) ;
```

D.5 pulser.m

```
function y=pulser(f)
%PULSER y = pulser(f) is a pulse shape.  f is in (MHz).
%
%   Example:
%       f  =  [-10 : 0.1 : 10 ] ;  % (MHz)
%       y  =  pulser(f) ;
%       plot(f,y) ;

lowpassbw = 5 ;

m = 0.4 ;
b = 1.5 ;
```

```
13
14 y = (m*f+b) .* rect(f,lowpassbw/2) ;
```

D.6 rect.m

```
1  function y=rect(x,c)
2  %RECT   y = rect(x,c) is defined as follows (element per element):
3  %                        rect(x,c) = 1  ,  −c ≤ |x| ≤ c
4  %                                    0  ,  otherwise
5  %   x is a matrix and c is a scaler.
6  %
7  %   Example:
8  %       x  =  [−10 : 0.1 : 10 ] ;
9  %       c  =  3 ;
10 %       y  =  rect(x,c) ;
11 %       plot(x,y) ;
12
13 %   The implementation calls another function called function_feeder to
14 %   make is faster than using for loops.
15
16
17 y=function_feeder(x,'one',(−c≤x)&(x≤c),'zero');
```

D.7 zero.m

```
1  function y=zero(x)
2  %ZERO   y=zero(x) generates a matrix of zeros of the same size as x.
3  %           If x has no dimensions, then y will have no dimensions.
4  %
5  %       Usage example:
6  %               y = zero ( [ −1 0 1 2 3 4 5 6 ] )
7
8  [m,n]=size(x);
9  if m≠0,
10    y=zeros(m,n);
11 end
```

D.8 one.m

```
1  function y=one(x)
2  %ONE    y=one(x) generates a matrix of ones of the same size as x.
3  %   If x has no dimensions, then y will have no dimensions.
4  %
5  %   Usage example:
6  %       y = one ( [ −1 0 1 2 3 4 5 6 ] )
7
8  [m,n]=size(x);
9  if m≠0,
```

```
   y=ones(m,n);
end
```

D.9 function_feeder.m

```
function y=function_feeder(x,Ftrue,c1,Ffalse)
%FUNCTION_FEEDER
%   y=function_feeder(x,Ftrue,c1,Ffalse) executes the function Ftrue with
%   the input data in x when condition c1 is true, else it executes
%   the function Ffalse with the input data in x Ffalse.  Ftrue and
%   Ffalse are strings of function names.  This function is very useful
%   when Ftrue takes a long time to evaluate and Ffalse is quick; it
%   can be used so that Ftrue is not called unnecessarily.  It is also
%   useful to implement a function by partitioning the cases using matrices
%   instead of by for loops.  It seems unusual to do this, but it is
%   because matlab handles matrices more very efficiently than for loops.

%   Usage example:
%       c = 2 ;
%       x = [-4 : 4 ]
%       y = function_feeder( x, 'one', (-c≤x)&(x≤c), 'ero' )

[m,n]=size(x);
if (m≥1)&(n==1),
   x2=makerow(x);
   [m,n]=size(x2);

   i_true = find(c1);
   if length(i_true) ≠ 0,
      y(1,i_true) = feval(Ftrue,x2(1,i_true));
   end

   i_not_true = find(¬c1);
   if length(i_not_true) ≠ 0,
      y(1,i_not_true) = feval(Ffalse,x2(1,i_not_true));
   end

   y=y';
elseif (m≥1)&(n≥2),
   for i=1:m,

      i_true = find(c1(i,:));
      if length(i_true) ≠ 0,
         y(i,i_true) = feval(Ftrue,x(i,i_true));
      end

      i_not_true = find(¬c1(i,:));
      if length(i_not_true) ≠ 0,
         y(i,i_not_true) = feval(Ffalse,x(i,i_not_true));
      end

   end
```

```
48 end
```

Appendix E

Verilog Code and Block Diagrams

Code that was used in multiple tests is only listed once. Overlap is present for figures of block diagrams to facilitate easier reading. The symbol '≤' represents the Verilog symbols '<='.

E.1 Code Used in Section 6.3

E.1.1 jitter6BitBuff.v

```
1  module jitter6BitBuff(  din,    // data in
2          dout,   // data out
3          clk,    // Clock input.
4          buff_select
5  //      a,b,c,d,e,f//,g,h,i,j,k,l,m,n,o
6      );
7
8  parameter DATAWIDTH = 14; // Number of bits on the data bus.
9
10 input [DATAWIDTH-1:0]din;
11 input clk;
12 input [5:0]buff_select;
13 output [DATAWIDTH-1:0]dout;
14 //output [DATAWIDTH-1:0]a,b,c,d,e,f;//,g,h,i,j,k,l,m,n,o;
15
16 reg [DATAWIDTH-1:0]a,b,c,d,e,f;//,g,h,i,j,k,l,m,n,o;
17 reg [DATAWIDTH-1:0]doutreg;
18
19 assign dout = doutreg;
20
21 /*************************/
22
23 always @(posedge clk) begin
24     a≤din;
25     b≤a;
26     c≤b;
27     d≤c;
```

```
28      e≤d;
29      f≤e;
30  end
31
32
33  always @(buff_select)
34      case (buff_select)
35          6'b000000: doutreg ≤din;
36          6'b000001: doutreg ≤din;
37          6'b000010: doutreg ≤din;
38          6'b000011: doutreg ≤din;
39          6'b000100: doutreg ≤f;
40          6'b000101: doutreg ≤f;
41          6'b000110: doutreg ≤f;
42          6'b000111: doutreg ≤f;
43          6'b001000: doutreg ≤a;
44          6'b001001: doutreg ≤a;
45          6'b001010: doutreg ≤a;
46          6'b001011: doutreg ≤a;
47          6'b001100: doutreg ≤a;
48          6'b001101: doutreg ≤e;
49          6'b001110: doutreg ≤e;
50          6'b001111: doutreg ≤e;
51          6'b010000: doutreg ≤e;
52          6'b010001: doutreg ≤e;
53          6'b010010: doutreg ≤d;
54          6'b010011: doutreg ≤d;
55          6'b010100: doutreg ≤d;
56          6'b010101: doutreg ≤d;
57          6'b010110: doutreg ≤d;
58          6'b010111: doutreg ≤d;
59          6'b011000: doutreg ≤d;
60          6'b011001: doutreg ≤d;
61          6'b011010: doutreg ≤d;
62          6'b011011: doutreg ≤d;
63          6'b011100: doutreg ≤b;
64          6'b011101: doutreg ≤b;
65          6'b011110: doutreg ≤b;
66          6'b011111: doutreg ≤b;
67          6'b100000: doutreg ≤b;
68          6'b100001: doutreg ≤b;
69          6'b100010: doutreg ≤b;
70          6'b100011: doutreg ≤b;
71          6'b100100: doutreg ≤b;
72          6'b100101: doutreg ≤b;
73          6'b100110: doutreg ≤c;
74          6'b100111: doutreg ≤c;
75          6'b101000: doutreg ≤c;
76          6'b101001: doutreg ≤c;
77          6'b101010: doutreg ≤c;
78          6'b101011: doutreg ≤c;
79          6'b101100: doutreg ≤c;
80          6'b101101: doutreg ≤c;
81          6'b101110: doutreg ≤c;
```

```
82          6'b101111: doutreg ≤c;
83          6'b110000: doutreg ≤c;
84          6'b110001: doutreg ≤c;
85          6'b110010: doutreg ≤c;
86          6'b110011: doutreg ≤c;
87          6'b110100: doutreg ≤c;
88          6'b110101: doutreg ≤c;
89          6'b110110: doutreg ≤c;
90          6'b110111: doutreg ≤c;
91          6'b111000: doutreg ≤c;
92          6'b111001: doutreg ≤c;
93          6'b111010: doutreg ≤c;
94          6'b111011: doutreg ≤c;
95          6'b111100: doutreg ≤c;
96          6'b111101: doutreg ≤c;
97          6'b111110: doutreg ≤c;
98          6'b111111: doutreg ≤c;
99
100         default: doutreg ≤ din;
101     endcase
102
103 endmodule
```

E.1.2 jitter6BitBuffEven.v

```
1  module jitter6BitBuffEven(  din,    // data in
2          dout,   // data out
3          clk,    // Clock input.
4          buff_select
5  //      a,b,c,d,e,f//,g,h,i,j,k,l,m,n,o
6      );
7
8  parameter DATAWIDTH = 14; // Number of bits on the data bus.
9
10 input [DATAWIDTH-1:0]din;
11 input clk;
12 input [5:0]buff_select;
13 output [DATAWIDTH-1:0]dout;
14 //output [DATAWIDTH-1:0]a,b,c,d,e,f;//,g,h,i,j,k,l,m,n,o;
15
16 reg [DATAWIDTH-1:0]a,b,c,d,e,f;//,g,h,i,j,k,l,m,n,o;
17 reg [DATAWIDTH-1:0]doutreg;
18
19 assign dout = doutreg;
20
21 /*************************/
22
23 always @(posedge clk) begin
24     a≤din;
25     b≤a;
26     c≤b;
27     d≤c;
28     e≤d;
```

```
29      f<=e;
30  end
31
32
33  always @(buff_select)
34      case (buff_select)
35          6'b000000: doutreg <=din;
36          6'b000001: doutreg <=din;
37          6'b000010: doutreg <=din;
38          6'b000011: doutreg <=din;
39          6'b000100: doutreg <=din;
40          6'b000101: doutreg <=din;
41          6'b000110: doutreg <=din;
42          6'b000111: doutreg <=din;
43          6'b001000: doutreg <=din;
44          6'b001001: doutreg <=a;
45          6'b001010: doutreg <=a;
46          6'b001011: doutreg <=a;
47          6'b001100: doutreg <=a;
48          6'b001101: doutreg <=a;
49          6'b001110: doutreg <=a;
50          6'b001111: doutreg <=a;
51          6'b010000: doutreg <=a;
52          6'b010001: doutreg <=a;
53          6'b010010: doutreg <=b;
54          6'b010011: doutreg <=b;
55          6'b010100: doutreg <=b;
56          6'b010101: doutreg <=b;
57          6'b010110: doutreg <=b;
58          6'b010111: doutreg <=b;
59          6'b011000: doutreg <=b;
60          6'b011001: doutreg <=b;
61          6'b011010: doutreg <=b;
62          6'b011011: doutreg <=c;
63          6'b011100: doutreg <=c;
64          6'b011101: doutreg <=c;
65          6'b011110: doutreg <=c;
66          6'b011111: doutreg <=c;
67          6'b100000: doutreg <=c;
68          6'b100001: doutreg <=c;
69          6'b100010: doutreg <=c;
70          6'b100011: doutreg <=c;
71          6'b100100: doutreg <=d;
72          6'b100101: doutreg <=d;
73          6'b100110: doutreg <=d;
74          6'b100111: doutreg <=d;
75          6'b101000: doutreg <=d;
76          6'b101001: doutreg <=d;
77          6'b101010: doutreg <=d;
78          6'b101011: doutreg <=d;
79          6'b101100: doutreg <=d;
80          6'b101101: doutreg <=e;
81          6'b101110: doutreg <=e;
82          6'b101111: doutreg <=e;
```

```
83          6'b110000: doutreg ≤e;
84          6'b110001: doutreg ≤e;
85          6'b110010: doutreg ≤e;
86          6'b110011: doutreg ≤e;
87          6'b110100: doutreg ≤e;
88          6'b110101: doutreg ≤e;
89          6'b110110: doutreg ≤f;
90          6'b110111: doutreg ≤f;
91          6'b111000: doutreg ≤f;
92          6'b111001: doutreg ≤f;
93          6'b111010: doutreg ≤f;
94          6'b111011: doutreg ≤f;
95          6'b111100: doutreg ≤f;
96          6'b111101: doutreg ≤f;
97          6'b111110: doutreg ≤f;
98          6'b111111: doutreg ≤c;
99
100         default: doutreg ≤ din;
101     endcase
102
103 endmodule
```

E.1.3 jitter6BitBuffSingle.v

```
1  module jitter6BitBuffSingle(    din,    // data in
2          dout,   // data out
3          clk,    // Clock input.
4          buff_select
5  //      a,b,c,d,e,f//,g,h,i,j,k,l,m,n,o
6      );
7
8  parameter DATAWIDTH = 14; // Number of bits on the data bus.
9
10 input [DATAWIDTH−1:0]din;
11 input clk;
12 input [5:0]buff_select;
13 output [DATAWIDTH−1:0]dout;
14 //output [DATAWIDTH−1:0]a,b,c,d,e,f;//,g,h,i,j,k,l,m,n,o;
15
16 reg [DATAWIDTH−1:0]a,b,c,d,e,f;//,g,h,i,j,k,l,m,n,o;
17 reg [DATAWIDTH−1:0]doutreg;
18
19 assign dout = doutreg;
20
21 /**************************/
22
23 always @(posedge clk) begin
24     a≤din;
25     b≤a;
26     c≤b;
27     d≤c;
28     e≤d;
29     f≤e;
```

```
30 end
31
32
33 always @(buff_select)
34     case (buff_select)
35         6'b000000: doutreg ≤c;
36         6'b000001: doutreg ≤c;
37         6'b000010: doutreg ≤c;
38         6'b000011: doutreg ≤c;
39         6'b000100: doutreg ≤c;
40         6'b000101: doutreg ≤c;
41         6'b000110: doutreg ≤c;
42         6'b000111: doutreg ≤c;
43         6'b001000: doutreg ≤c;
44         6'b001001: doutreg ≤c;
45         6'b001010: doutreg ≤c;
46         6'b001011: doutreg ≤c;
47         6'b001100: doutreg ≤c;
48         6'b001101: doutreg ≤c;
49         6'b001110: doutreg ≤c;
50         6'b001111: doutreg ≤c;
51         6'b010000: doutreg ≤c;
52         6'b010001: doutreg ≤c;
53         6'b010010: doutreg ≤c;
54         6'b010011: doutreg ≤c;
55         6'b010100: doutreg ≤c;
56         6'b010101: doutreg ≤c;
57         6'b010110: doutreg ≤c;
58         6'b010111: doutreg ≤c;
59         6'b011000: doutreg ≤c;
60         6'b011001: doutreg ≤c;
61         6'b011010: doutreg ≤c;
62         6'b011011: doutreg ≤c;
63         6'b011100: doutreg ≤c;
64         6'b011101: doutreg ≤c;
65         6'b011110: doutreg ≤c;
66         6'b011111: doutreg ≤c;
67         6'b100000: doutreg ≤c;
68         6'b100001: doutreg ≤c;
69         6'b100010: doutreg ≤c;
70         6'b100011: doutreg ≤c;
71         6'b100100: doutreg ≤c;
72         6'b100101: doutreg ≤c;
73         6'b100110: doutreg ≤c;
74         6'b100111: doutreg ≤c;
75         6'b101000: doutreg ≤c;
76         6'b101001: doutreg ≤c;
77         6'b101010: doutreg ≤c;
78         6'b101011: doutreg ≤c;
79         6'b101100: doutreg ≤c;
80         6'b101101: doutreg ≤c;
81         6'b101110: doutreg ≤c;
82         6'b101111: doutreg ≤c;
83         6'b110000: doutreg ≤c;
```

```
84          6'b110001: doutreg ≤c;
85          6'b110010: doutreg ≤c;
86          6'b110011: doutreg ≤c;
87          6'b110100: doutreg ≤c;
88          6'b110101: doutreg ≤c;
89          6'b110110: doutreg ≤c;
90          6'b110111: doutreg ≤c;
91          6'b111000: doutreg ≤c;
92          6'b111001: doutreg ≤c;
93          6'b111010: doutreg ≤c;
94          6'b111011: doutreg ≤c;
95          6'b111100: doutreg ≤c;
96          6'b111101: doutreg ≤c;
97          6'b111110: doutreg ≤c;
98          6'b111111: doutreg ≤c;
99
100         default: doutreg ≤ c;
101     endcase
102
103 endmodule
```

E.1.4 jitter6BitBuffTight.v

```
1  module jitter6BitBuffTight( din,    // data in
2          dout,   // data out
3          clk,    // Clock input.
4          buff_select
5  //      a,b,c,d,e,f//,g,h,i,j,k,l,m,n,o
6      );
7
8  parameter DATAWIDTH = 14; // Number of bits on the data bus.
9
10 input [DATAWIDTH-1:0]din;
11 input clk;
12 input [5:0]buff_select;
13 output [DATAWIDTH-1:0]dout;
14 //output [DATAWIDTH-1:0]a,b,c,d,e,f;//,g,h,i,j,k,l,m,n,o;
15
16 reg [DATAWIDTH-1:0]a,b,c,d,e,f;//,g,h,i,j,k,l,m,n,o;
17 reg [DATAWIDTH-1:0]doutreg;
18
19 assign dout = doutreg;
20
21 /*************************/
22
23 always @(posedge clk) begin
24     a≤din;
25     b≤a;
26     c≤b;
27     d≤c;
28     e≤d;
29     f≤e;
30 end
```

```
31
32
33 always @(buff_select)
34     case (buff_select)
35         6'b000000: doutreg ≤din;
36         6'b000001: doutreg ≤din;
37         6'b000010: doutreg ≤f;
38         6'b000011: doutreg ≤f;
39         6'b000100: doutreg ≤e;
40         6'b000101: doutreg ≤e;
41         6'b000110: doutreg ≤e;
42         6'b000111: doutreg ≤e;
43         6'b001000: doutreg ≤a;
44         6'b001001: doutreg ≤a;
45         6'b001010: doutreg ≤a;
46         6'b001011: doutreg ≤a;
47         6'b001100: doutreg ≤b;
48         6'b001101: doutreg ≤b;
49         6'b001110: doutreg ≤b;
50         6'b001111: doutreg ≤b;
51         6'b010000: doutreg ≤b;
52         6'b010001: doutreg ≤d;
53         6'b010010: doutreg ≤d;
54         6'b010011: doutreg ≤d;
55         6'b010100: doutreg ≤d;
56         6'b010101: doutreg ≤d;
57         6'b010110: doutreg ≤d;
58         6'b010111: doutreg ≤d;
59         6'b011000: doutreg ≤b;
60         6'b011001: doutreg ≤b;
61         6'b011010: doutreg ≤b;
62         6'b011011: doutreg ≤d;
63         6'b011100: doutreg ≤b;
64         6'b011101: doutreg ≤d;
65         6'b011110: doutreg ≤b;
66         6'b011111: doutreg ≤d;
67         6'b100000: doutreg ≤c;
68         6'b100001: doutreg ≤c;
69         6'b100010: doutreg ≤c;
70         6'b100011: doutreg ≤c;
71         6'b100100: doutreg ≤c;
72         6'b100101: doutreg ≤c;
73         6'b100110: doutreg ≤c;
74         6'b100111: doutreg ≤c;
75         6'b101000: doutreg ≤c;
76         6'b101001: doutreg ≤c;
77         6'b101010: doutreg ≤c;
78         6'b101011: doutreg ≤c;
79         6'b101100: doutreg ≤c;
80         6'b101101: doutreg ≤c;
81         6'b101110: doutreg ≤c;
82         6'b101111: doutreg ≤c;
83         6'b110000: doutreg ≤c;
84         6'b110001: doutreg ≤c;
```

```
85          6'b110010: doutreg <= c;
86          6'b110011: doutreg <= c;
87          6'b110100: doutreg <= c;
88          6'b110101: doutreg <= c;
89          6'b110110: doutreg <= c;
90          6'b110111: doutreg <= c;
91          6'b111000: doutreg <= c;
92          6'b111001: doutreg <= c;
93          6'b111010: doutreg <= c;
94          6'b111011: doutreg <= c;
95          6'b111100: doutreg <= c;
96          6'b111101: doutreg <= c;
97          6'b111110: doutreg <= c;
98          6'b111111: doutreg <= c;
99
100         default: doutreg <= din;
101     endcase
102
103 endmodule
```

E.2 Code Used in Section 6.5.1

E.2.1 jitterMultiPN.v

```
1  /*
2   * Author: Brandon C. Brown
3   *
4   * Date: 2009-07-25
5   *
6   * Description:
7   *     This module is purpose built to take various sizes
8   *   of PN generators, and select one of 3 buffers based on
9   *   the number.  Will chose one of 3 buffers: All but two from
10  *   the middle, and once on either side.
11  *
12  *
13  */
14
15
16 module jitterMultiPN(
17          din,    // data in
18          dout,   // data out
19          clk,    // Clock input, runs at same speed as acquisition clock.
20          buff_select,
21          a,
22          b,
23          c
24 );
25
26 parameter DATAWIDTH = 14;
27 parameter BUFFSELECTWIDTH = 10;
28
```

```
29 input   [DATAWIDTH-1:0]         din;
30 input                           clk;
31 input   [BUFFSELECTWIDTH-1:0]   buff_select;
32 output  [DATAWIDTH-1:0]         dout,a,b,c;
33
34 reg     [DATAWIDTH-1:0]         a,b,c;
35 reg     [2:0]                   dsel;
36 wire    [DATAWIDTH-1:0]         dout;
37 reg     [DATAWIDTH-1:0]         doutreg;
38
39 assign dout = doutreg;
40
41
42 always @(posedge clk) begin
43     a<=din;
44     b<=a;
45     c<=b;
46 end
47
48 // 1 and 2 chosen because they are in any buff_select sizes...
49 always @(buff_select) begin
50     case(buff_select)
51         1:          doutreg <= a;
52         2:          doutreg <= c;
53         default:    doutreg <= b;
54     endcase
55 end
56
57 endmodule
```

E.2.2 pnGenerator3bit.v

```
1  /*
2
3  Author: Brandon C. Brown
4
5  Date: 2009-05-19
6
7  General idea:
8
9  Circuit based on P(x) = x^3 + x + 1
10
11 |-------------------+<----------
12 |                   ^          |
13 |                   |          |
14 |                   |          |
15 ----->|a|---->|b|--o-->|c|---o------>dout
16
17 Seed with 1'b1
18 Period = 7 (Maximum Length Sequence)
19
20 Note:   Modulus two addition is shown
21         schematically equivalent to
```

```
22          Exclusive-OR gates.
23
24  Designed with assistance from:
25  http://www.ee.unb.ca/cgi-bin/tervo/sequence.pl
26  Site by: Richard Tervo
27
28  */
29
30
31  module pnGenerator3bit( CLK,
32                          dout,
33                          pdout
34                          );
35
36  input CLK;
37  output dout;
38  output [2:0]pdout;
39
40  reg a,b,c;
41  wire btoc, plustoa;
42
43  assign dout = c;
44  assign btoc = b;
45  assign pdout = {a,b,dout};
46
47  xor(plustoa,btoc,dout);
48
49  initial  c = 1'b1;
50
51  always @(posedge CLK) begin
52      b <= a;
53      c <= b;
54      a <= plustoa;
55  end
56
57  endmodule
```

E.2.3 pnGenerator4bit.v

```
1   /*
2
3   Author: Brandon C. Brown
4
5   Date: 2009-07-24
6
7   General idea:
8
9   Circuit based on P(x) = x^4 + x + 1
10
11  |-------------------------------+<---------
12  |                               ^         |
13  |                               |         |
14  |                               |         |
```

```
15 ------>|a|------>|b|------>|c|--o-->|d|--o-->dout
16    1        0         0         1         1
17
18
19 Seed with 1'b1
20 Period = 15 (Maximum Length Sequence)
21
22 Note:   Modulus two addition is shown
23         schematically equivalent to
24         Exclusive-OR gates.
25
26 Designed with assistance from:
27 http://www.ee.unb.ca/cgi-bin/tervo/sequence.pl
28 Site by: Richard Tervo
29
30 */
31
32
33 module pnGenerator4bit( CLK,
34                         dout,
35                         pdout
36                         );
37
38 input CLK;
39 output dout;
40 output [3:0]pdout;
41
42 reg a,b,c,d;
43 wire plustoa;
44
45 assign dout = d;
46 assign pdout = {a,b,c,d};
47
48 xor(plustoa,c,dout);
49
50 initial  d = 1'b1;
51
52 always @(posedge CLK) begin
53     b <= a;
54     c <= b;
55     d <= c;
56     a <= plustoa;
57 end
58
59 endmodule
```

E.2.4 pnGenerator5bit.v

```
1 /*
2
3 Author: Brandon C. Brown
4
5 Date: 2009-05-27
```

```
6
7  General idea:
8
9  Circuit based on P(x) = x^5 + x^2 + 1
10
11 |-------------------------------+<-----------------
12 |                               ^                 |
13 |                               |                 |
14 |                               |                 |
15 ------>|a|------>|b|------>|c|--o-->|d|------>|e|--o-->dout
16    1        0         0         1         0         1
17
18
19 Seed with 1'b1
20 Period = 31 (Maximum Length Sequence)
21
22 Note:   Modulus two addition is shown
23         schematically equivalent to
24         Exclusive-OR gates.
25
26 Designed with assistance from:
27 http://www.ee.unb.ca/cgi-bin/tervo/sequence.pl
28 Site by: Richard Tervo
29
30 */
31
32
33 module pnGenerator5bit( CLK,
34                         dout,
35                         pdout
36                         );
37
38 input CLK;
39 output dout;
40 output [4:0]pdout;
41
42 reg a,b,c,d,e;
43 wire ctod, plustoa;
44
45 assign dout = e;
46 assign ctod = c;
47 assign pdout = {a,b,c,d,e};
48
49 xor(plustoa,ctod,dout);
50
51 initial  e = 1'b1;
52
53 always @(posedge CLK) begin
54     b <= a;
55     c <= b;
56     d <= c;
57     e <= d;
58     a <= plustoa;
59 end
```

```
endmodule
```

E.2.5 pnGenerator6bit.v

```
/*

Author: Brandon C. Brown

Date: 2009-07-24

General idea:

Circuit based on P(x) = x^6 + x + 1

|-------------------------------------------------+<-------|
|                                                 |         |
|                                                 |         |
|                                                 |         |
------>|a|------>|b|------>|c|------>|d|------>|e|--o-->|f|--o-->dout
   1        0         0         0         0         1         1


Seed with 1'b1
Period = 63 (Maximum Length Sequence)

Note:   Modulus two addition is shown
        schematically equivalent to
        Exclusive-OR gates.

Designed with assistance from:
http://www.ee.unb.ca/cgi-bin/tervo/sequence.pl
Site by: Richard Tervo

*/

module pnGenerator6bit( CLK,
                        dout,
                        pdout
                        );

input CLK;
output dout;
output [5:0]pdout;

reg a,b,c,d,e,f,g;
wire plustoa;

assign dout = f;
assign pdout = {a,b,c,d,e,f};

xor(plustoa,e,dout);

```

```
49 initial  f = 1'b1;
50
51 always @(posedge CLK) begin
52     b <= a;
53     c <= b;
54     d <= c;
55     e <= d;
56     f <= e;
57     a <= plustoa;
58 end
59
60 endmodule
```

E.2.6 pnGenerator7bit.v

```
1  /*
2
3  Author: Brandon C. Brown
4
5  Date: 2009-07-24
6
7  General idea:
8
9  Circuit based on P(x) = x^7 + x + 1
10
11 |-----------------------------------------------+<-------|
12 |                                               |        |
13 |                                               |        |
14 |                                               |        |
15 --->|a|--->|b|--->|c|--->|d|--->|e|--->|f|--o-->|g|--o-->dout
16  1      0      0      0      0      0      1        1
17
18
19 Seed with 1'b1
20 Period = 127 (Maximum Length Sequence)
21
22 Note:   Modulus two addition is shown
23         schematically equivalent to
24         Exclusive-OR gates.
25
26 Designed with assistance from:
27 http://www.ee.unb.ca/cgi-bin/tervo/sequence.pl
28 Site by: Richard Tervo
29
30 */
31
32 module pnGenerator7bit( CLK,
33                         dout,
34                         pdout
35                         );
36
37 input CLK;
38 output dout;
```

```
39 output [6:0]pdout;
40
41 reg a,b,c,d,e,f,g;
42 wire plustoa;
43
44 assign dout = g;
45 assign pdout = {a,b,c,d,e,f,g};
46
47 xor(plustoa,f,dout);
48
49 initial  g = 1'b1;
50
51 always @(posedge CLK) begin
52     b <= a;
53     c <= b;
54     d <= c;
55     e <= d;
56     f <= e;
57     g <= f;
58     a <= plustoa;
59 end
60
61 endmodule
```

E.2.7 pnGenerator8bit.v

```
1  /*
2
3  Author: Brandon C. Brown
4
5  Date: 2009-07-24
6
7  General idea:
8
9  Circuit based on P(x) = x^8 + x^4 + x^3 + x^2 + 1
10
11 |-------------------------------+<------+<------+<----------------|
12 |                               |       |       |                 |
13 |                               |       |       |                 |
14 |                               |       |       |                 |
15 --->|a|--->|b|--->|c|--->|d|-o->|e|-o->|f|-o->|g|--->|h|-o->dout
16  1      0      0      0      1      1      1      0      1
17
18
19 Seed with 1'b1
20 Period = 255 (Maximum Length Sequence)
21
22 Note:   Modulus two addition is shown
23         schematically equivalent to
24         Exclusive-OR gates.
25
26 Designed with assistance from:
27 http://www.ee.unb.ca/cgi-bin/tervo/sequence.pl
```

```
28 Site by: Richard Tervo
29
30 */
31
32 module pnGenerator8bit( CLK,
33                         dout,
34                         pdout
35                         );
36
37 input CLK;
38 output dout;
39 output [7:0]pdout;
40
41 reg a,b,c,d,e,f,g,h;
42 wire plustoa;
43 wire plus2, plus1;
44
45 assign dout = h;
46 assign pdout = {a,b,c,d,e,f,g,h};
47
48 xor(plustoa,d,plus2);
49 xor(plus2,e,plus1);
50 xor(plus1,f,dout);
51
52 initial  h = 1'b1;
53
54 always @(posedge CLK) begin
55     b ≤ a;
56     c ≤ b;
57     d ≤ c;
58     e ≤ d;
59     f ≤ e;
60     g ≤ f;
61     h ≤ g;
62     a ≤ plustoa;
63 end
64
65 endmodule
```

E.2.8 pnGenerator9bit.v

```
1  /*
2
3  Author: Brandon C. Brown
4
5  Date: 2009-07-24
6
7  General idea:
8
9  Circuit based on P(x) = x^9 + x^4 + 1
10
11 |-----------------------------------+<----------------------------|
12 |                                   |                           |
```

```
13 |                                     |                               |
14 |                                     |                               |
15 ---->|a|--->|b|--->|c|--->|d|--->|e|-o->|f|--->|g|--->|h|--->|i|-o->dout
16  1      0      0      0      0      1      0      0      0      1
17
18
19 Seed with 1'b1
20 Period = 511 (Maximum Length Sequence)
21
22 Note:   Modulus two addition is shown
23         schematically equivalent to
24         Exclusive-OR gates.
25
26 Designed with assistance from:
27 http://www.ee.unb.ca/cgi-bin/tervo/sequence.pl
28 Site by: Richard Tervo
29
30 */
31
32
33 module pnGenerator9bit( CLK,
34                         dout,
35                         pdout
36                         );
37
38 input CLK;
39 output dout;
40 output [8:0]pdout;
41
42 reg a,b,c,d,e,f,g,h,i,j;
43 wire etof;
44 wire plustoa;
45
46 assign dout = i;
47 //assign etof = e;
48 assign pdout = {a,b,c,d,e,f,g,h,i};
49
50 xor(plustoa,e,dout);
51
52 initial  i = 1'b1;
53
54 always @(posedge CLK) begin
55     b ≤ a;
56     c ≤ b;
57     d ≤ c;
58     e ≤ d;
59     f ≤ e;
60     g ≤ f;
61     h ≤ g;
62     i ≤ h;
63     a ≤ plustoa;
64 end
65
66 endmodule
```

E.2.9 pnGenerator10bit.v

```
/*

Author: Brandon C. Brown

Date: 2009-07-08

General idea:

Circuit based on P(x) = x^10 + x^3 + 1

|----- ----------------------------------------+<----------------------|
|                                               |                       |
|                                               |                       |
|                                               |                       |
-->|a|-->|b|-->|c|-->|d|-->|e|-->|f|-->|g|-o->|h|-->|i|-->|j|-o->dout
 1     0     0     0     0     0     0     1     0     0     1


Seed with 1'b1
Period = 1023 (Maximum Length Sequence)

Note:   Modulus two addition is shown
        schematically equivalent to
        Exclusive-OR gates.

Designed with assistance from:
http://www.ee.unb.ca/cgi-bin/tervo/sequence.pl
Site by: Richard Tervo

*/


module pnGenerator10bit(    CLK,
                            dout,
                            pdout
                            );

input CLK;
output dout;
output [9:0]pdout;

reg a,b,c,d,e,f,g,h,i,j;
wire gtoh, plustoa;

assign dout = j;
assign gtoh = g;
assign pdout = {a,b,c,d,e,f,g,h,i,j};

xor(plustoa,gtoh,dout);

```

```
51 initial  j = 1'b1;
52
53 always @(posedge CLK) begin
54     b ≤ a;
55     c ≤ b;
56     d ≤ c;
57     e ≤ d;
58     f ≤ e;
59     g ≤ f;
60     h ≤ g;
61     i ≤ h;
62     j ≤ i;
63     a ≤ plustoa;
64 end
65
66 endmodule
```

E.2.10 DE3.v

```
1
2  //=======================================================
3  //  This code is generated by Terasic System Builder
4  //=======================================================
5  module DE3(
6
7          ///////// CLOCK /////////
8          OSC_BA,
9          OSC_BB,
10         OSC_BC,
11         OSC_BD,
12         OSC1_50,
13         OSC2_50,
14         CLK_OUT,
15         EXT_CLK,
16
17         ///////// LED /////////
18         LEDR,
19         LEDG,
20         LEDB,
21
22         ///////// SEG7 /////////
23         HEX0,
24         HEX0_DP,
25         HEX1,
26         HEX1_DP,
27
28         ///////// BUTTON /////////
29         Button,
30
31         ///////// SW (SLIDE SWITCH) /////////
32         SW,
33
34         ///////// SDCARD /////////
```

```
35          SD_CMD,
36          SD_CLK,
37          SD_DAT,
38          SD_WPn,
39
40          ////////// HSTCC (J5 HSTC-C TOP/J6, HSTC-C BOTTOM) //////////
41          HSTCC_AD_OTRA,
42          HSTCC_PLL_OUT_ADC,
43          HSTCC_AD_OTRB,
44          HSTCC_AD_DB,
45          HSTCC_AD_DA,
46          HSTCC_ADC_OEB_B,
47          HSTCC_ADC_OEB_A,
48          HSTCC_SMA_DAC4,
49          HSTCC_PLL_OUT_DAC,
50          HSTCC_OSC_SMA_ADC4,
51          HSTCC_DA_MODE,
52          HSTCC_DA_WRTA,
53          HSTCC_DA_WRTB,
54          HSTCC_DA_DA,
55          HSTCC_DA_DB,
56          HSTCC_SDA,
57          HSTCC_SCL,
58
59          ////////// HSTCD (J7, HSTC-D TOP/J8, JSTC-D BOTTOM) //////////
60          HSTCD_AD_OTRA,
61          HSTCD_PLL_OUT_ADC,
62          HSTCD_AD_OTRB,
63          HSTCD_AD_DB,
64          HSTCD_AD_DA,
65          HSTCD_ADC_OEB_B,
66          HSTCD_ADC_OEB_A,
67          HSTCD_SMA_DAC4,
68          HSTCD_PLL_OUT_DAC,
69          HSTCD_OSC_SMA_ADC4,
70          HSTCD_DA_MODE,
71          HSTCD_DA_WRTA,
72          HSTCD_DA_WRTB,
73          HSTCD_DA_DA,
74          HSTCD_DA_DB,
75          HSTCD_SDA,
76          HSTCD_SCL,
77
78          ////////// REGULATOR //////////
79          JVC_CLK,
80          JVC_CS,
81          JVC_DATAOUT,
82          JVC_DATAIN
83
84      );
85
86  //=====================================================
87  //  PARAMETER declarations
88  //=====================================================
```

```
89
90  //=======================================================
91  //  PORT declarations
92  //=======================================================
93  ////////// CLOCK //////////
94  input                           OSC_BA;
95  input                           OSC_BB;
96  input                           OSC_BC;
97  input                           OSC_BD;
98  input                           OSC1_50;
99  input                           OSC2_50;
100 output                          CLK_OUT;
101 input                           EXT_CLK;
102
103 ////////// LED //////////
104 output      [7:0]               LEDR;
105 output      [7:0]               LEDG;
106 output      [7:0]               LEDB;
107
108 ////////// SEG7 //////////
109 output      [6:0]               HEX0;
110 output                          HEX0_DP;
111 output      [6:0]               HEX1;
112 output                          HEX1_DP;
113
114 ////////// BUTTON //////////
115 input       [3:0]               Button;
116
117 ////////// SW (SLIDE SWITCH) //////////
118 input       [3:0]               SW;
119
120 ////////// SDCARD //////////
121 inout                           SD_CMD;
122 output                          SD_CLK;
123 inout                           SD_DAT;
124 input                           SD_WPn;
125
126 / HSTCC (J5 HSTC—C TOP/J6, HSTC—C BOTTOM), connect to CELL(ADA Board) //
127 output      [1:0]               HSTCC_PLL_OUT_ADC;
128 input                           HSTCC_AD_OTRA;
129 input                           HSTCC_AD_OTRB;
130 input       [13:0]              HSTCC_AD_DB;
131 input       [13:0]              HSTCC_AD_DA;
132 output                          HSTCC_ADC_OEB_B;
133 output                          HSTCC_ADC_OEB_A;
134 output      [1:0]               HSTCC_PLL_OUT_DAC;
135 input                           HSTCC_SMA_DAC4;
136 input                           HSTCC_OSC_SMA_ADC4;
137 output                          HSTCC_DA_MODE;
138 output                          HSTCC_DA_WRTA;
139 output                          HSTCC_DA_WRTB;
140 output      [13:0]              HSTCC_DA_DA;
141 output      [13:0]              HSTCC_DA_DB;
142 inout                           HSTCC_SDA;
```

```
output                          HSTCC_SCL;

// HSTCD (J7, HSTC-D TOP/J8, JSTC-D BOTTOM), connect to BASE(ADA Board) //
output      [1:0]               HSTCD_PLL_OUT_ADC;
input                           HSTCD_AD_OTRA;
input                           HSTCD_AD_OTRB;
input       [13:0]              HSTCD_AD_DB;
input       [13:0]              HSTCD_AD_DA;
output                          HSTCD_ADC_OEB_B;
output                          HSTCD_ADC_OEB_A;
output      [1:0]               HSTCD_PLL_OUT_DAC;
input                           HSTCD_SMA_DAC4;
input                           HSTCD_OSC_SMA_ADC4;
output                          HSTCD_DA_MODE;
output                          HSTCD_DA_WRTA;
output                          HSTCD_DA_WRTB;
output      [13:0]              HSTCD_DA_DA;
output      [13:0]              HSTCD_DA_DB;
inout                           HSTCD_SDA;
output                          HSTCD_SCL;

////////// REGULATOR //////////
output                          JVC_CLK;
output                          JVC_CS;
output                          JVC_DATAOUT;
input                           JVC_DATAIN;


//=======================================================
//  REG/WIRE declarations
//=======================================================
wire        [13:0]              data_out_wire;
reg         [13:0]              dataOut;
wire                            CLK_125;
wire                            CLK_OUT;
wire        [31:0]              phaseinc;

wire                            g = 0;
wire                            v = 1;

wire        [3:0]               Button;

wire        [13:0]              cell_i;
wire        [13:0]              cell_q;
wire        [13:0]              base_i;
wire        [13:0]              base_q;

wire        [13:0]              base_i_out;
wire        [13:0]              cell_i_out;


wire                            SIGNAL_TAP_CLOCK;
wire                            CLK_65;
wire                            CLK_200;
```

```
197 wire                            CLK_500;
198 wire                            CLK_250;
199 wire                            pllLock;
200
201
202 wire        [13:0]              jitter_base_i;
203 wire        [13:0]              jitter_base_q;
204 wire        [13:0]              jitter_cell_i;
205 wire        [13:0]              jitter_cell_q;
206
207
208
209
210 //=======================================================
211 //  IO Group Voltage Configuration (Do not modify it)
212 //=======================================================
213 IOV_A3V3_B3V3_C3V3_D3V3 IOV_Instance(
214     .iCLK(OSC2_50),
215     .iRST_n(1'b1),
216     .iENABLE(1'b0),
217     .oREADY(),
218     .oERR(),
219     .oERRCODE(),
220     .oJVC_CLK(JVC_CLK),
221     .oJVC_CS(JVC_CS),
222     .oJVC_DATAOUT(JVC_DATAOUT),
223     .iJVC_DATAIN(JVC_DATAIN)
224 );
225
226 //=======================================================
227 //  Structural coding
228 //=======================================================
229
230
231 assign  HSTCC_DA_WRTB = CLK_125; //Input write signal for PORT B
232 assign  HSTCC_DA_WRTA = CLK_125; //Input write signal for PORT A
233 assign  HSTCD_DA_WRTA = CLK_125;
234 assign  HSTCD_DA_WRTB = CLK_125;
235
236
237 assign  HSTCC_DA_MODE = 1; //Mode Select. 1=dual port, 0=interleaved.
238 assign  HSTCD_DA_MODE = 1;
239
240 assign  HSTCC_PLL_OUT_DAC[0] = CLK_125; //PLL Clock to DAC_B
241 assign  HSTCC_PLL_OUT_DAC[1] = CLK_125; //PLL Clock to DAC_A
242 assign  HSTCD_PLL_OUT_DAC[0] = CLK_125;
243 assign  HSTCD_PLL_OUT_DAC[1] = CLK_125;
244
245 assign  HSTCC_PLL_OUT_ADC[0] = CLK_65; //PLL Clock to ADC_B
246 assign  HSTCC_PLL_OUT_ADC[1] = CLK_65; //PLL Clock to ADC_A
247 assign  HSTCD_PLL_OUT_ADC[0] = CLK_65;
248 assign  HSTCD_PLL_OUT_ADC[1] = CLK_65;
249
250 assign  HSTCC_ADC_OEB_B = 0;
```

```
251 assign  HSTCC_ADC_OEB_A = 0;
252 assign  HSTCD_ADC_OEB_B = 0;
253 assign  HSTCD_ADC_OEB_A = 0;
254
255 //ADC out of range indicators.
256 assign  LEDR = {EXT_CLK,v,v,v,pllLock,v,¬HSTCD_AD_OTRA,¬HSTCC_AD_OTRA};
257 assign  LEDB = {EXT_CLK,v,v,v,pllLock,v,¬HSTCD_AD_OTRB,¬HSTCC_AD_OTRB};
258 assign  LEDG = {¬SIGNAL_TAP_CLOCK,v,v,v,v,v,v,v};
259
260 decode  bside   (
261                     .bcd_input({HSTCD_AD_OTRA,g,g,HSTCD_AD_OTRB}),
262                     .seven_seg_output(HEX1)
263                 );
264
265 decode  cside   (
266                     .bcd_input({HSTCC_AD_OTRA,g,g,HSTCC_AD_OTRB}),
267                     .seven_seg_output(HEX0)
268                 );
269 assign  HEX0_DP = v;
270 assign  HEX1_DP = v;
271
272
273 /*************************************
274 * Taking input I and Q and outputting them on the other ADA Board.
275 * HSTCC = cell side,  HSTCD = base station side  */
276
277 // cell side
278
279 assign  cell_i = HSTCC_AD_DA;
280 assign  cell_q = HSTCC_AD_DB;
281 assign  HSTCC_DA_DA = base_i_out;
282
283 //base side
284 assign  base_i = HSTCD_AD_DA;
285 assign  base_q = HSTCD_AD_DB;
286 assign  HSTCD_DA_DA = cell_i_out;
287
288 /*************************************/
289
290 /************Enable both lines******************/
291
292 assign base_i_out = (SW[3]) ? jitter_base_i : base_i;
293 assign cell_i_out = (SW[3]) ? jitter_cell_i : cell_i;
294
295 /*******************************************/
296
297
298 /***********Clock Sources using OSC1_50*******/
299 assign CLK_OUT = CLK_65;
300 assign SIGNAL_TAP_CLOCK = CLK_250;
301
302 pll5clks    pll_125 (   .inclk0(OSC1_50),
303                         .areset(¬Button[2]),    //Reset the PLL
304                         .locked(pllLock),   //Vcc if locked, GND if not
```

```
305                         .c0(CLK_65),        //65MHz -> really 62.5 MHz
306                         .c1(CLK_125),   //125 MHz
307                         .c2(CLK_250),   //250 MHz
308                         .c3(CLK_200),
309                     );
310 /*****************************************/
311
312 /*********************************************/
313
314
315
316
317
318 jitterUniformDist   udist_cell(
319                             .sin(cell_i),
320                             .CLK_62(CLK_65),
321                             .CLK_500(CLK_250),
322                             .jitter_sel(SW[1:0]),
323                             .dout(jitter_cell_i)
324                         );
325 jitterUniformDist   udist_base(
326                             .sin(base_i),
327                             .CLK_62(CLK_65),
328                             .CLK_500(CLK_250),
329                             .jitter_sel(SW[1:0]),
330                             .dout(jitter_base_i)
331                         );
332
333 endmodule
```

E.2.11 Block Diagram

For readability, the block diagram on the following pages has been split into multiple sections, which includes some overlap.

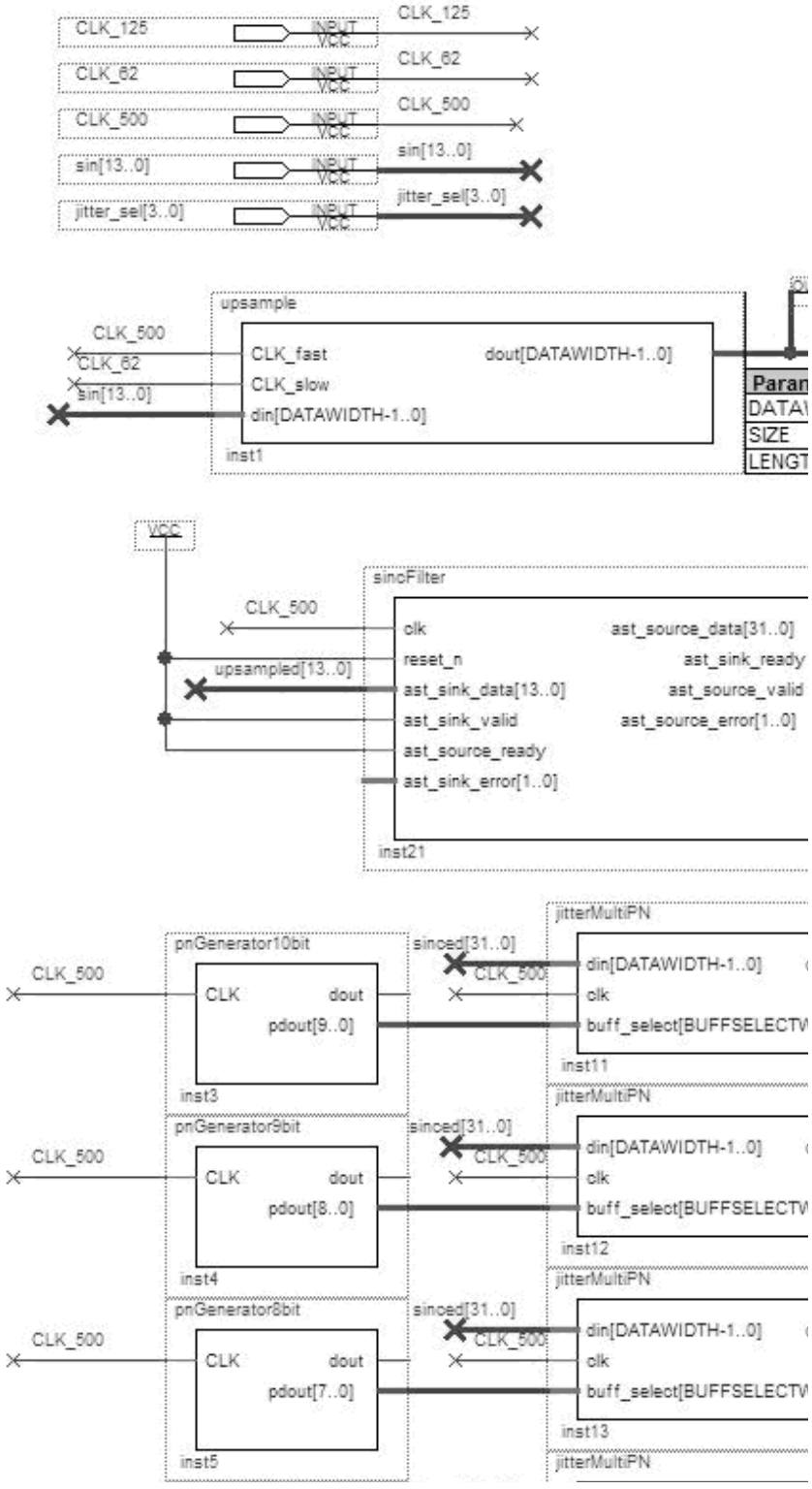

Figure E.1: Part 1 of 4: block diagram of the Verilog implementation used.

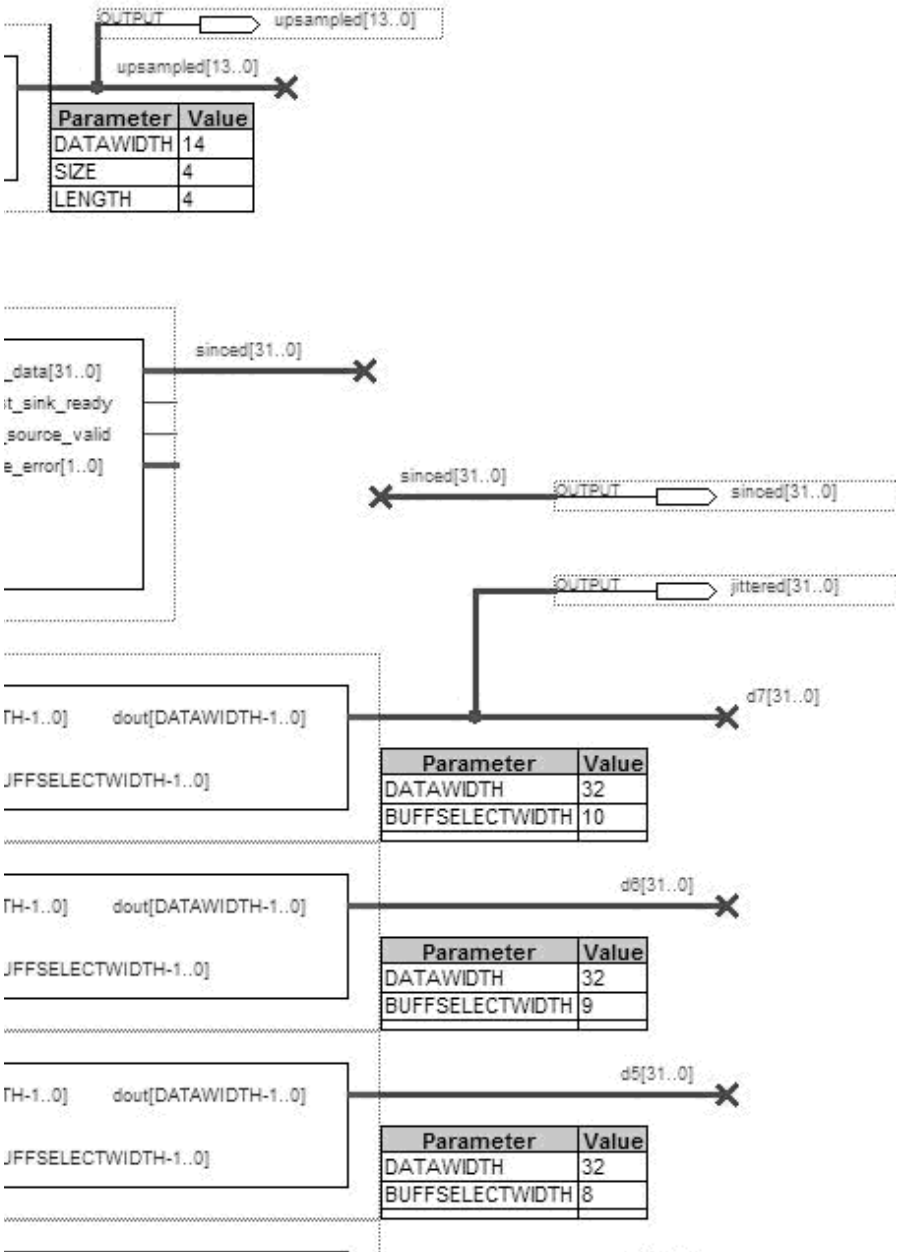

Figure E.2: Part 2 of 4: block diagram of the Verilog implementation used.

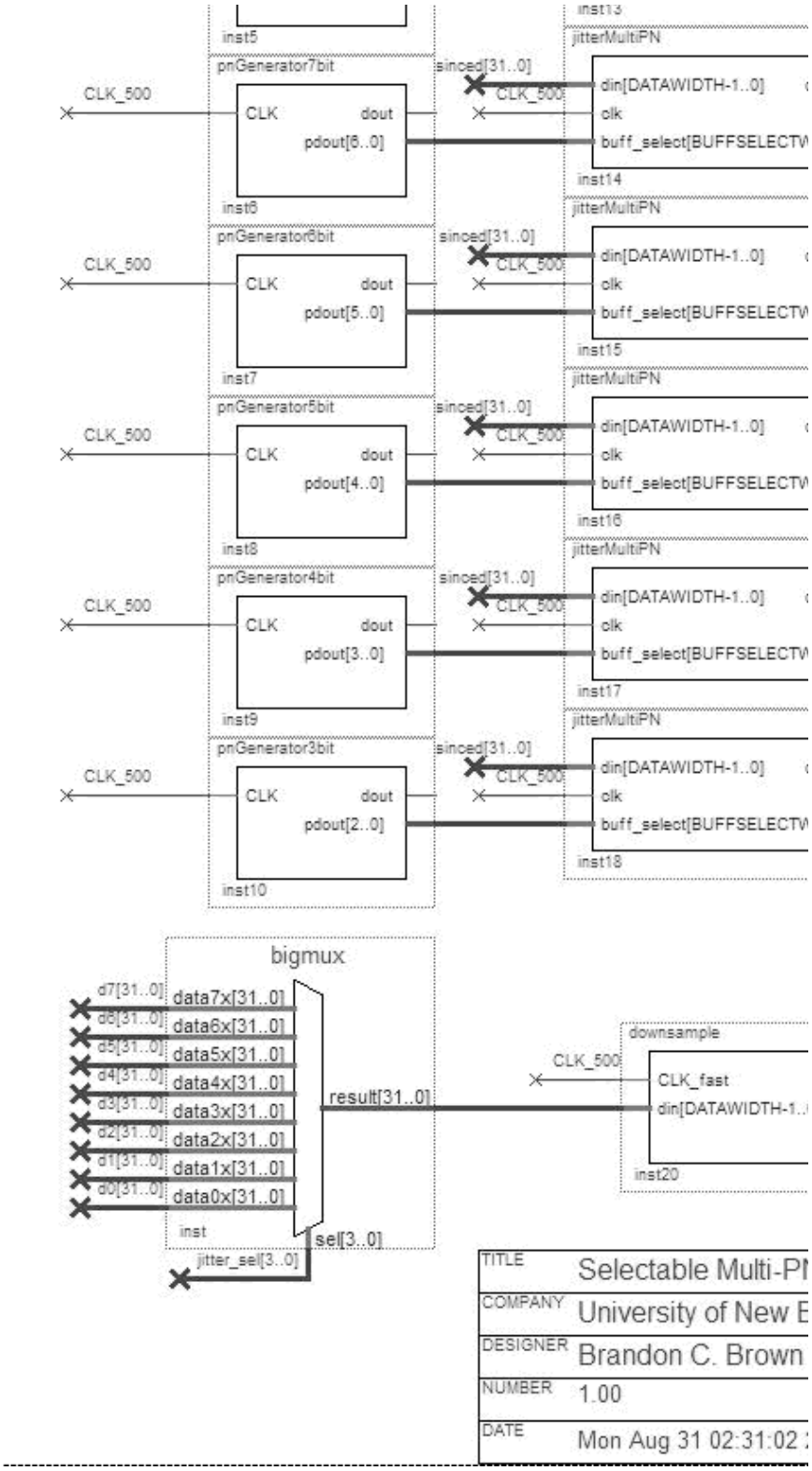

Figure E.3: Part 3 of 4: block diagram of the Verilog implementation used.

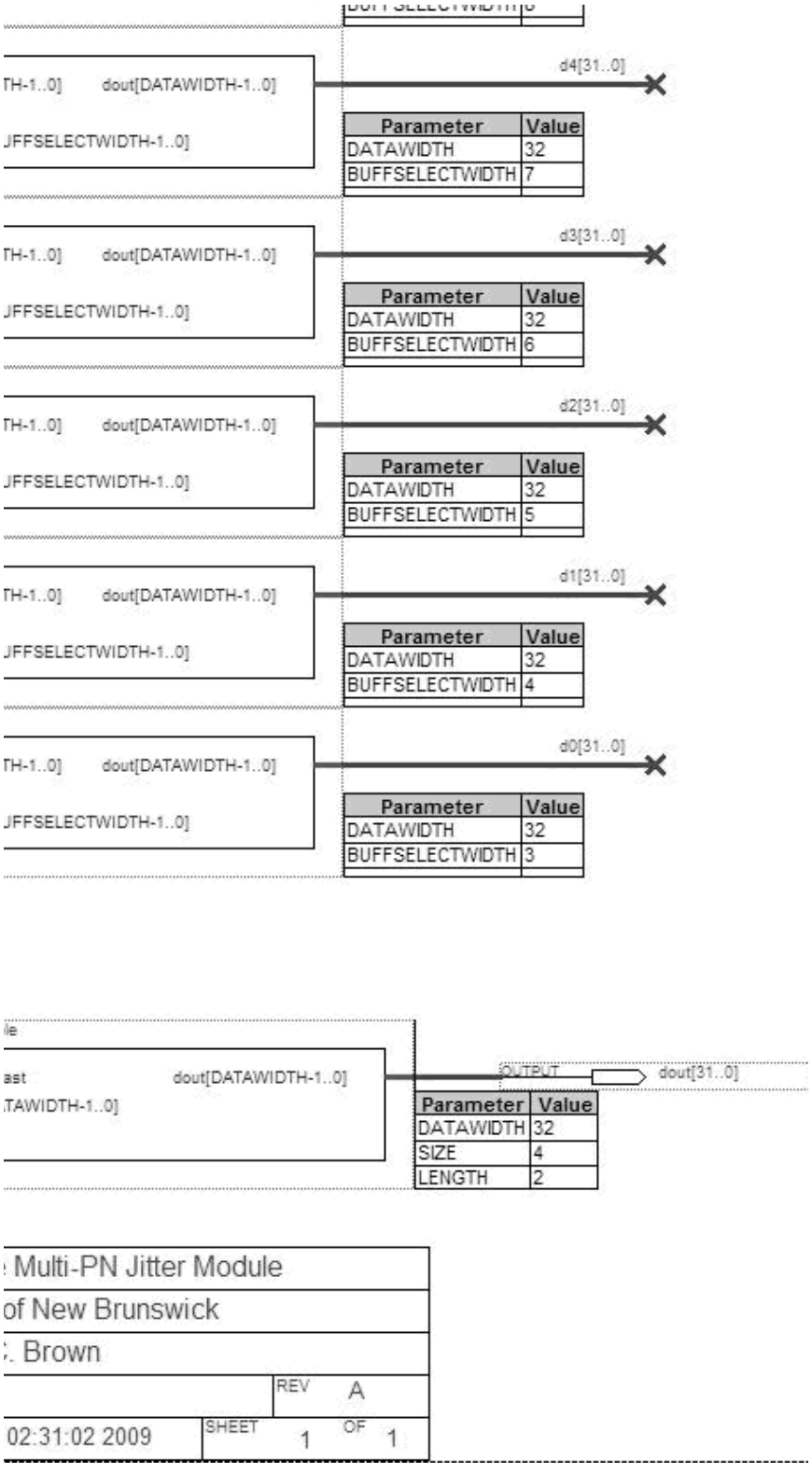

Figure E.4: Part 4 of 4: block diagram of the Verilog implementation used.

E.3 Code Used in Section 6.5.2

E.3.1 DE3.v

```
1
2  //=======================================================
3  //  This code is generated by Terasic System Builder
4  //=======================================================
5  module DE3(
6
7          ////////// CLOCK //////////
8          OSC_BA,
9          OSC_BB,
10         OSC_BC,
11         OSC_BD,
12         OSC1_50,
13         OSC2_50,
14         CLK_OUT,
15         EXT_CLK,
16
17         ////////// LED //////////
18         LEDR,
19         LEDG,
20         LEDB,
21
22         ////////// SEG7 //////////
23         HEX0,
24         HEX0_DP,
25         HEX1,
26         HEX1_DP,
27
28         ////////// BUTTON //////////
29         Button,
30
31         ////////// SW (SLIDE SWITCH) //////////
32         SW,
33
34         ////////// SDCARD //////////
35         SD_CMD,
36         SD_CLK,
37         SD_DAT,
38         SD_WPn,
39
40         ////////// HSTCC (J5 HSTC-C TOP/J6, HSTC-C BOTTOM) //////////
41         HSTCC_AD_OTRA,
42         HSTCC_PLL_OUT_ADC,
43         HSTCC_AD_OTRB,
44         HSTCC_AD_DB,
45         HSTCC_AD_DA,
46         HSTCC_ADC_OEB_B,
47         HSTCC_ADC_OEB_A,
48         HSTCC_SMA_DAC4,
```

```
49          HSTCC_PLL_OUT_DAC,
50          HSTCC_OSC_SMA_ADC4,
51          HSTCC_DA_MODE,
52          HSTCC_DA_WRTA,
53          HSTCC_DA_WRTB,
54          HSTCC_DA_DA,
55          HSTCC_DA_DB,
56          HSTCC_SDA,
57          HSTCC_SCL,
58
59          ////////// HSTCD (J7, HSTC-D TOP/J8, JSTC-D BOTTOM) //////////
60          HSTCD_AD_OTRA,
61          HSTCD_PLL_OUT_ADC,
62          HSTCD_AD_OTRB,
63          HSTCD_AD_DB,
64          HSTCD_AD_DA,
65          HSTCD_ADC_OEB_B,
66          HSTCD_ADC_OEB_A,
67          HSTCD_SMA_DAC4,
68          HSTCD_PLL_OUT_DAC,
69          HSTCD_OSC_SMA_ADC4,
70          HSTCD_DA_MODE,
71          HSTCD_DA_WRTA,
72          HSTCD_DA_WRTB,
73          HSTCD_DA_DA,
74          HSTCD_DA_DB,
75          HSTCD_SDA,
76          HSTCD_SCL,
77
78          ////////// REGULATOR //////////
79          JVC_CLK,
80          JVC_CS,
81          JVC_DATAOUT,
82          JVC_DATAIN
83
84      );
85
86  //=====================================================
87  //  PARAMETER declarations
88  //=====================================================
89
90  //=====================================================
91  //  PORT declarations
92  //=====================================================
93  ////////// CLOCK //////////
94  input                           OSC_BA;
95  input                           OSC_BB;
96  input                           OSC_BC;
97  input                           OSC_BD;
98  input                           OSC1_50;
99  input                           OSC2_50;
100 output                          CLK_OUT;
101 input                           EXT_CLK;
102
```

```
103 ////////// LED //////////
104 output      [7:0]               LEDR;
105 output      [7:0]               LEDG;
106 output      [7:0]               LEDB;
107
108 ////////// SEG7 //////////
109 output      [6:0]               HEX0;
110 output                          HEX0_DP;
111 output      [6:0]               HEX1;
112 output                          HEX1_DP;
113
114 ////////// BUTTON //////////
115 input       [3:0]               Button;
116
117 ////////// SW (SLIDE SWITCH) //////////
118 input       [3:0]               SW;
119
120 ////////// SDCARD //////////
121 inout                           SD_CMD;
122 output                          SD_CLK;
123 inout                           SD_DAT;
124 input                           SD_WPn;
125
126 / HSTCC (J5 HSTC—C TOP/J6, HSTC—C BOTTOM), connect to CELL(ADA Board) //
127 output      [1:0]               HSTCC_PLL_OUT_ADC;
128 input                           HSTCC_AD_OTRA;
129 input                           HSTCC_AD_OTRB;
130 input       [13:0]              HSTCC_AD_DB;
131 input       [13:0]              HSTCC_AD_DA;
132 output                          HSTCC_ADC_OEB_B;
133 output                          HSTCC_ADC_OEB_A;
134 output      [1:0]               HSTCC_PLL_OUT_DAC;
135 input                           HSTCC_SMA_DAC4;
136 input                           HSTCC_OSC_SMA_ADC4;
137 output                          HSTCC_DA_MODE;
138 output                          HSTCC_DA_WRTA;
139 output                          HSTCC_DA_WRTB;
140 output      [13:0]              HSTCC_DA_DA;
141 output      [13:0]              HSTCC_DA_DB;
142 inout                           HSTCC_SDA;
143 output                          HSTCC_SCL;
144
145 // HSTCD (J7, HSTC—D TOP/J8, JSTC—D BOTTOM), connect to BASE(ADA Board) //
146 output      [1:0]               HSTCD_PLL_OUT_ADC;
147 input                           HSTCD_AD_OTRA;
148 input                           HSTCD_AD_OTRB;
149 input       [13:0]              HSTCD_AD_DB;
150 input       [13:0]              HSTCD_AD_DA;
151 output                          HSTCD_ADC_OEB_B;
152 output                          HSTCD_ADC_OEB_A;
153 output      [1:0]               HSTCD_PLL_OUT_DAC;
154 input                           HSTCD_SMA_DAC4;
155 input                           HSTCD_OSC_SMA_ADC4;
156 output                          HSTCD_DA_MODE;
```

```
157 output                          HSTCD_DA_WRTA;
158 output                          HSTCD_DA_WRTB;
159 output      [13:0]              HSTCD_DA_DA;
160 output      [13:0]              HSTCD_DA_DB;
161 inout                           HSTCD_SDA;
162 output                          HSTCD_SCL;
163
164 ////////// REGULATOR //////////
165 output                          JVC_CLK;
166 output                          JVC_CS;
167 output                          JVC_DATAOUT;
168 input                           JVC_DATAIN;
169
170
171 //=======================================================
172 //  REG/WIRE declarations
173 //=======================================================
174 wire        [13:0]              data_out_wire;
175 reg         [13:0]              dataOut;
176 wire                            CLK_125;
177 wire                            CLK_OUT;
178 wire        [31:0]              phaseinc;
179
180 wire                            g = 0;
181 wire                            v = 1;
182
183 wire        [3:0]               Button;
184
185 wire        [13:0]              cell_i;
186 wire        [13:0]              cell_q;
187 wire        [13:0]              base_i;
188 wire        [13:0]              base_q;
189
190 wire        [13:0]              base_i_out;
191 wire        [13:0]              cell_i_out;
192
193
194 wire                            SIGNAL_TAP_CLOCK;
195 wire                            CLK_65;
196 wire                            CLK_200;
197 wire                            CLK_500;
198 wire                            CLK_250;
199 wire                            pllLock;
200
201
202 wire        [13:0]              jitter_base_i;
203 wire        [13:0]              jitter_base_q;
204 wire        [13:0]              jitter_cell_i;
205 wire        [13:0]              jitter_cell_q;
206
207
208
209
210 //=======================================================
```

```
211 //  IO Group Voltage Configuration (Do not modify it)
212 //=======================================================
213 IOV_A3V3_B3V3_C3V3_D3V3 IOV_Instance(
214     .iCLK(OSC2_50),
215     .iRST_n(1'b1),
216     .iENABLE(1'b0),
217     .oREADY(),
218     .oERR(),
219     .oERRCODE(),
220     .oJVC_CLK(JVC_CLK),
221     .oJVC_CS(JVC_CS),
222     .oJVC_DATAOUT(JVC_DATAOUT),
223     .iJVC_DATAIN(JVC_DATAIN)
224 );
225
226 //=======================================================
227 //  Structural coding
228 //=======================================================
229
230
231 assign  HSTCC_DA_WRTB = CLK_125; //Input write signal for PORT B
232 assign  HSTCC_DA_WRTA = CLK_125; //Input write signal for PORT A
233 assign  HSTCD_DA_WRTA = CLK_125;
234 assign  HSTCD_DA_WRTB = CLK_125;
235
236
237 assign  HSTCC_DA_MODE = 1; //Mode Select. 1=dual port, 0=interleaved.
238 assign  HSTCD_DA_MODE = 1;
239
240 assign  HSTCC_PLL_OUT_DAC[0] = CLK_125; //PLL Clock to DAC_B
241 assign  HSTCC_PLL_OUT_DAC[1] = CLK_125; //PLL Clock to DAC_A
242 assign  HSTCD_PLL_OUT_DAC[0] = CLK_125;
243 assign  HSTCD_PLL_OUT_DAC[1] = CLK_125;
244
245 assign  HSTCC_PLL_OUT_ADC[0] = CLK_65; //PLL Clock to ADC_B
246 assign  HSTCC_PLL_OUT_ADC[1] = CLK_65; //PLL Clock to ADC_A
247 assign  HSTCD_PLL_OUT_ADC[0] = CLK_65;
248 assign  HSTCD_PLL_OUT_ADC[1] = CLK_65;
249
250 assign  HSTCC_ADC_OEB_B = 0;
251 assign  HSTCC_ADC_OEB_A = 0;
252 assign  HSTCD_ADC_OEB_B = 0;
253 assign  HSTCD_ADC_OEB_A = 0;
254
255 //ADC out of range indicators.
256 assign  LEDR = {EXT_CLK,v,v,v,pllLock,v,¬HSTCD_AD_OTRA,¬HSTCC_AD_OTRA};
257 assign  LEDB = {EXT_CLK,v,v,v,pllLock,v,¬HSTCD_AD_OTRB,¬HSTCC_AD_OTRB};
258 assign  LEDG = {¬SIGNAL_TAP_CLOCK,v,v,v,v,v,v,v};
259
260 decode  bside   (
261                     .bcd_input({HSTCD_AD_OTRA,g,g,HSTCD_AD_OTRB}),
262                     .seven_seg_output(HEX1)
263                 );
264
```

```
265 decode  cside   (
266                     .bcd_input({HSTCC_AD_OTRA,g,g,HSTCC_AD_OTRB}),
267                     .seven_seg_output(HEX0)
268                 );
269 assign  HEX0_DP = v;
270 assign  HEX1_DP = v;
271
272
273 /*************************************
274 * Taking input I and Q and outputting them on the other ADA Board.
275 * HSTCC = cell side,  HSTCD = base station side  */
276
277 // cell side
278
279 assign  cell_i = HSTCC_AD_DA;
280 assign  cell_q = HSTCC_AD_DB;
281 assign  HSTCC_DA_DA = base_i_out;
282
283 //base side
284 assign  base_i = HSTCD_AD_DA;
285 assign  base_q = HSTCD_AD_DB;
286 assign  HSTCD_DA_DA = cell_i_out;
287
288 /*************************************/
289
290 /************Enable both lines******************/
291
292 assign base_i_out = (SW[3]) ? jitter_base_i : base_i;
293 assign cell_i_out = (SW[3]) ? jitter_cell_i : cell_i;
294
295 /*********************************************/
296
297
298 /***********Clock Sources using OSC1_50*******/
299 assign CLK_OUT = CLK_65;
300 assign SIGNAL_TAP_CLOCK = CLK_250;
301
302 pll5clks    pll_125 (   .inclk0(OSC1_50),
303                         .areset(¬Button[2]),    //Reset the PLL
304                         .locked(pllLock),   //Vcc if locked, GND if not
305                         .c0(CLK_65),        //65MHz -> really 62.5 MHz
306                         .c1(CLK_125),   //125 MHz
307                         .c2(CLK_250),   //250 MHz
308                         .c3(CLK_200),
309                     );
310 /*****************************************/
311
312 /*********************************************/
313
314
315
316
317
318 jitterUniformDist   udist_cell(
```

```
                                .sin(cell_i),
                                .CLK_62(CLK_65),
                                .CLK_500(CLK_250),
                                .jitter_sel(SW[1:0]),
                                .dout(jitter_cell_i)
                            );
jitterUniformDist   udist_base(
                                .sin(base_i),
                                .CLK_62(CLK_65),
                                .CLK_500(CLK_250),
                                .jitter_sel(SW[1:0]),
                                .dout(jitter_base_i)
                            );

endmodule
```

E.3.2 jitterUniform3Regs.v

```
/*
 * Author: Brandon C. Brown
 *
 * Date: 2009-07-30
 *
 * Description:
 *     This module accepts a PN Generator which has 511 unique
 *  states (1-511) and outputs a uniformly selected register.
 *  See file name for number of registers.
 *
 */

module jitterUniform3Regs(
                                din,    //Data input
                                dout,   //Data output
                                clk,    //Clock input
                                buff_select //Input from PN Generator
                            );

parameter DATAWIDTH = 14;

input   [DATAWIDTH-1:0]     din;
input                       clk;
input   [8:0]               buff_select;
output  [DATAWIDTH-1:0]     dout;

reg     [DATAWIDTH-1:0]     a,b,c;
reg     [DATAWIDTH-1:0]     doutreg;

wire    [DATAWIDTH-1:0]     dout;

assign dout = doutreg;

always @(buff_select) begin
    if(buff_select < 171) doutreg <= a;  //1-170 (170 #'s)
```

```
36      else if(buff_select < 342) doutreg <= c; //341-511 (170 #'s)
37      else doutreg <= b; //171-340 (171 #'s)
38  end
39
40  always @(posedge clk) begin
41      a<=din;
42      b<=a;
43      c<=b;
44  end
45
46  endmodule
```

E.3.3 jitterUniform5Regs.v

```
1   /*
2    * Author: Brandon C. Brown
3    *
4    * Date: 2009-07-30
5    *
6    * Description:
7    *     This module accepts a PN Generator which has 511 unique
8    *  states (1-511)  and outputs a uniformly selected register.
9    *  See file name for number of registers.
10   *
11   */
12
13  module jitterUniform5Regs(
14                              din,    //Data input
15                              dout,   //Data output
16                              clk,    //Clock input
17                              buff_select //Input from PN Generator
18                          );
19
20  parameter DATAWIDTH = 14;
21
22  input   [DATAWIDTH-1:0]     din;
23  input                       clk;
24  input   [8:0]               buff_select;
25  output  [DATAWIDTH-1:0]     dout;
26
27  reg     [DATAWIDTH-1:0]     a,b,c,d,e;
28  reg     [DATAWIDTH-1:0]     doutreg;
29
30  wire    [DATAWIDTH-1:0]     dout;
31
32  assign dout = doutreg;
33
34  always @(buff_select) begin
35      if(buff_select <= 102) doutreg <= a;
36      else if(buff_select <= 204) doutreg <= b;
37      else if(buff_select <= 307) doutreg <= c; //1 extra here
38      else if(buff_select <= 409) doutreg <= d;
39      else if(buff_select <= 511) doutreg <= e;
```

```
40      else doutreg ≤ c; //Should never get here.
41  end
42
43  always @(posedge clk) begin
44      a≤din;
45      b≤a;
46      c≤b;
47      d≤c;
48      e≤d;
49  end
50
51  endmodule
```

E.3.4 downsample.v

```
1   /*  A Down-Sampling module.
2    *
3    *  Author: Brandon C. Brown
4    *  Date: 2009-07-15
5    *  Description:
6    *      Counts the set number of clock cycles and changes output
7    *      once the counter hits the value.
8    *      Uses a counter which pulses at the LENGTH
9    *      value.  SIZE is the number of registers required
10   *      to reach LENGTH, so set accordingly.  In the
11   *      counter circuit, 4'b1 is used to add one.  Possibly
12   *      consider changing the 4 to match SIZE.
13   *
14   */
15
16  module downsample(
17
18          CLK_fast,
19  //      CLK_slow,
20          din,
21          dout
22      );
23
24  //=======================================================
25  //  PARAMETER declarations
26  //=======================================================
27  parameter DATAWIDTH = 14;
28  parameter SIZE = 4;
29  parameter LENGTH = 8;
30
31  //=======================================================
32  //  PORT declarations
33  //=======================================================
34  ////////// CLOCK //////////
35  input                           CLK_fast;
36  //input                         CLK_slow;
37  ////////// DATA ///////////
38  input       [DATAWIDTH-1:0]     din;
```

```
39 output      [DATAWIDTH-1:0]     dout;
40
41 //=======================================================
42 //  REG/WIRE declarations
43 //=======================================================
44 reg         [DATAWIDTH-1:0]     value;
45 reg         [SIZE-1:0]          count;
46 wire                            pulse;
47 wire                            g = 0;
48 wire                            v = 1;
49
50
51 //=======================================================
52 //  Structural coding
53 //=======================================================
54
55 assign dout = value;
56
57 always @(posedge pulse)
58     begin
59         value <= din;
60     end
61
62 /******
63  * Pulser - A block that creates the pulses for pulsing the output
64  *
65  */
66 assign pulse = (count == LENGTH-1);
67
68 always @(negedge CLK_fast)
69     begin
70         if(count == LENGTH-1)
71             count <= 4'b0;
72         else
73             count <= count + 4'b1;
74     end
75
76 endmodule
```

E.3.5 jitterUniform25Regs.v

```
1  /*
2   * Author: Brandon C. Brown
3   *
4   * Date: 2009-07-30
5   *
6   * Description:
7   *     This module accepts a PN Generator which has 511 unique states
8   *  (1-511) and outputs a uniformly selected register.  See file name
9   *  for number of registers.
10  *
11  */
12
```

```
13 module jitterUniform25Regs(
14                              din,    //Data input
15                              dout,   //Data output
16                              clk,    //Clock input
17                              buff_select //Input from PN Generator
18                          );
19
20 parameter DATAWIDTH = 14;
21
22 input   [DATAWIDTH-1:0]     din;
23 input                       clk;
24 input   [8:0]               buff_select;
25 output  [DATAWIDTH-1:0]     dout;
26
27 reg     [DATAWIDTH-1:0]     a,b,c,d,e,f,g,h,i,j,k,l,m,n;
28 reg     [DATAWIDTH-1:0]     o,p,q,r,s,t,u,v,w,x,y;
29 reg     [DATAWIDTH-1:0]     doutreg;
30
31 wire    [DATAWIDTH-1:0]     dout;
32
33 assign dout = doutreg;
34
35 always @(posedge clk) begin
36     a<=din;
37     b<=a;
38     c<=b;
39     d<=c;
40     e<=d;
41     f<=e;
42     g<=f;
43     h<=g;
44     i<=h;
45     j<=i;
46     k<=j;
47     l<=k;
48     m<=l;
49     n<=m;
50     o<=n;
51     p<=o;
52     q<=p;
53     r<=q;
54     s<=r;
55     t<=s;
56     u<=t;
57     v<=u;
58     w<=v;
59     x<=w;
60     y<=x;
61 end
62
63
64 always @(buff_select) begin
65     //21's
66     if (buff_select <= 21) doutreg <= h;
```

```
67      else if (buff_select ≤ 42) doutreg ≤ i;
68      else if (buff_select ≤ 63) doutreg ≤ j;
69      else if (buff_select ≤ 84) doutreg ≤ k;
70      else if (buff_select ≤ 105) doutreg ≤ l;
71      else if (buff_select ≤ 126) doutreg ≤ m;
72      else if (buff_select ≤ 147) doutreg ≤ n;
73      else if (buff_select ≤ 168) doutreg ≤ o;
74      else if (buff_select ≤ 189) doutreg ≤ p;
75      else if (buff_select ≤ 210) doutreg ≤ q;
76      else if (buff_select ≤ 231) doutreg ≤ r;
77      //20's
78      else if (buff_select ≤ 251) doutreg ≤ a;
79      else if (buff_select ≤ 271) doutreg ≤ b;
80      else if (buff_select ≤ 291) doutreg ≤ c;
81      else if (buff_select ≤ 311) doutreg ≤ d;
82      else if (buff_select ≤ 331) doutreg ≤ e;
83      else if (buff_select ≤ 351) doutreg ≤ f;
84      else if (buff_select ≤ 371) doutreg ≤ g;
85      else if (buff_select ≤ 391) doutreg ≤ s;
86      else if (buff_select ≤ 411) doutreg ≤ t;
87      else if (buff_select ≤ 431) doutreg ≤ u;
88      else if (buff_select ≤ 451) doutreg ≤ v;
89      else if (buff_select ≤ 471) doutreg ≤ w;
90      else if (buff_select ≤ 491) doutreg ≤ x;
91      else if (buff_select ≤ 511) doutreg ≤ y;
92      //Default to middle register.
93      else doutreg ≤ m;
94
95  end
96
97  endmodule
```

E.3.6 jitterUniform13Regs.v

```
1  /*
2   * Author: Brandon C. Brown
3   *
4   * Date: 2009-07-30
5   *
6   * Description:
7   *     This module accepts a PN Generator which has 511 unique states
8   * (1-511) and outputs a uniformly selected register.  See file name
9   * for number of registers.
10  *
11  */
12
13  module jitterUniform13Regs(
14                             din,    //Data input
15                             dout,   //Data output
16                             clk,    //Clock input
17                             buff_select //Input from PN Generator
18                          );
19
```

```
20 parameter DATAWIDTH = 14;
21
22 input   [DATAWIDTH-1:0]     din;
23 input                       clk;
24 input   [8:0]                buff_select;
25 output  [DATAWIDTH-1:0]     dout;
26
27 reg     [DATAWIDTH-1:0]     a,b,c,d,e,f,g,h,i,j,k,l,m;
28 reg     [DATAWIDTH-1:0]     doutreg;
29
30 wire    [DATAWIDTH-1:0]     dout;
31
32 assign dout = doutreg;
33
34 always @(posedge clk) begin
35     a<=din;
36     b<=a;
37     c<=b;
38     d<=c;
39     e<=d;
40     f<=e;
41     g<=f;
42     h<=g;
43     i<=h;
44     j<=i;
45     k<=j;
46     l<=k;
47     m<=l;
48 end
49
50 always @(buff_select) begin
51
52     if(buff_select <= 39) doutreg<= a ;
53     else if(buff_select <= 78) doutreg<= b ;
54     else if(buff_select <= 117) doutreg<= c ;
55     else if(buff_select <= 156) doutreg<= d ;
56     else if(buff_select <= 191) doutreg<= e ; //Extra reg. here
57     else if(buff_select <= 236) doutreg<= f ; //Extra reg. here
58     else if(buff_select <= 275) doutreg<= g ; //Middle!!
59     else if(buff_select <= 315) doutreg<= h ; //Extra reg. here
60     else if(buff_select <= 355) doutreg<= i ; //Extra reg. here
61     else if(buff_select <= 394) doutreg<= j ;
62     else if(buff_select <= 433) doutreg<= k ;
63     else if(buff_select <= 472) doutreg<= l ;
64     else if(buff_select <= 511) doutreg<= m ;
65     else doutreg <= g;   //should never ge here, only for completeness.
66 end
67
68 endmodule
```

E.3.7 Block Diagram

For readability, the block diagram on the following pages has been split into multiple sections, which includes some overlap.

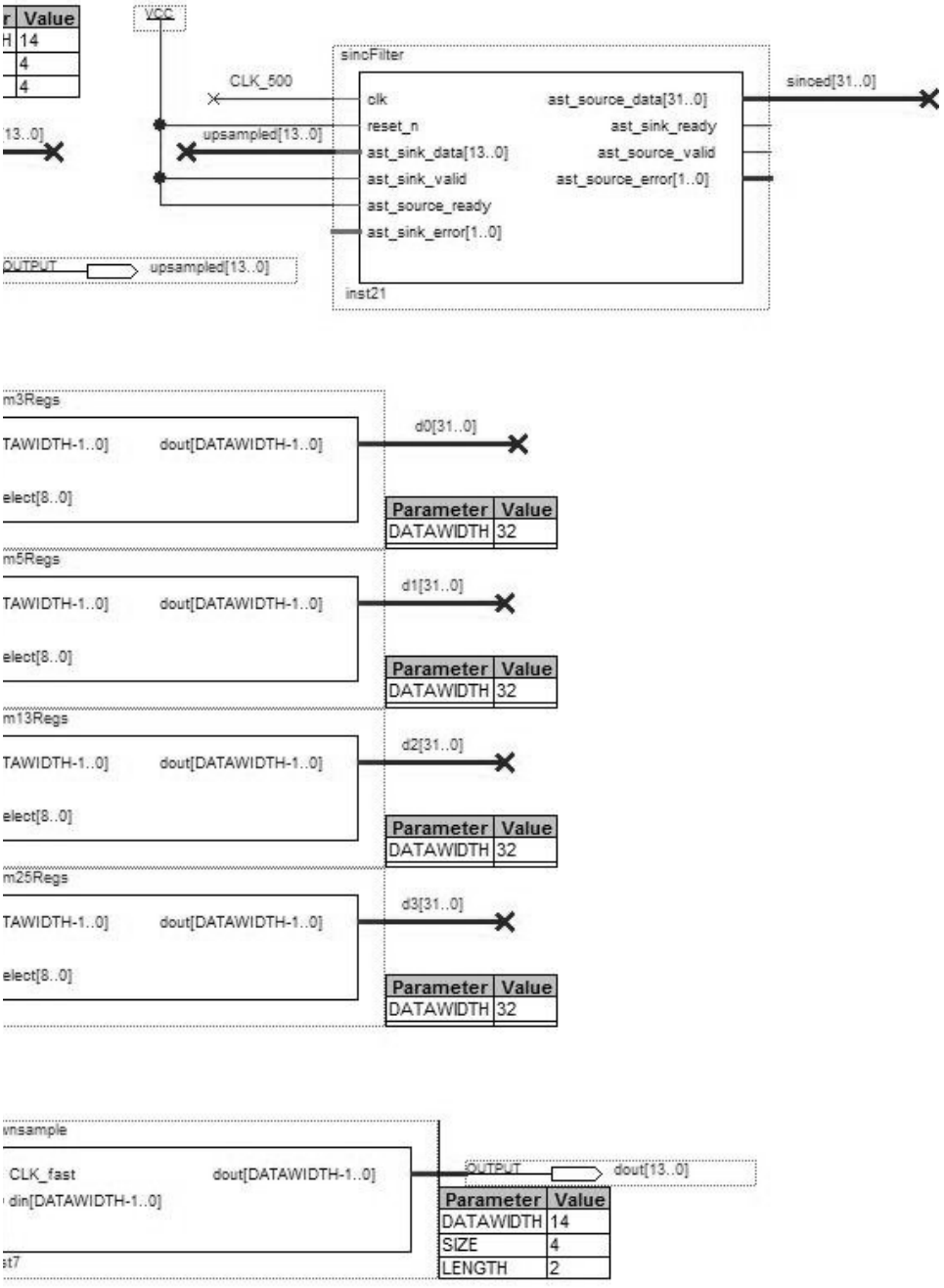

Figure E.5: Part 1 of 2: block diagram of the Verilog implementation used.

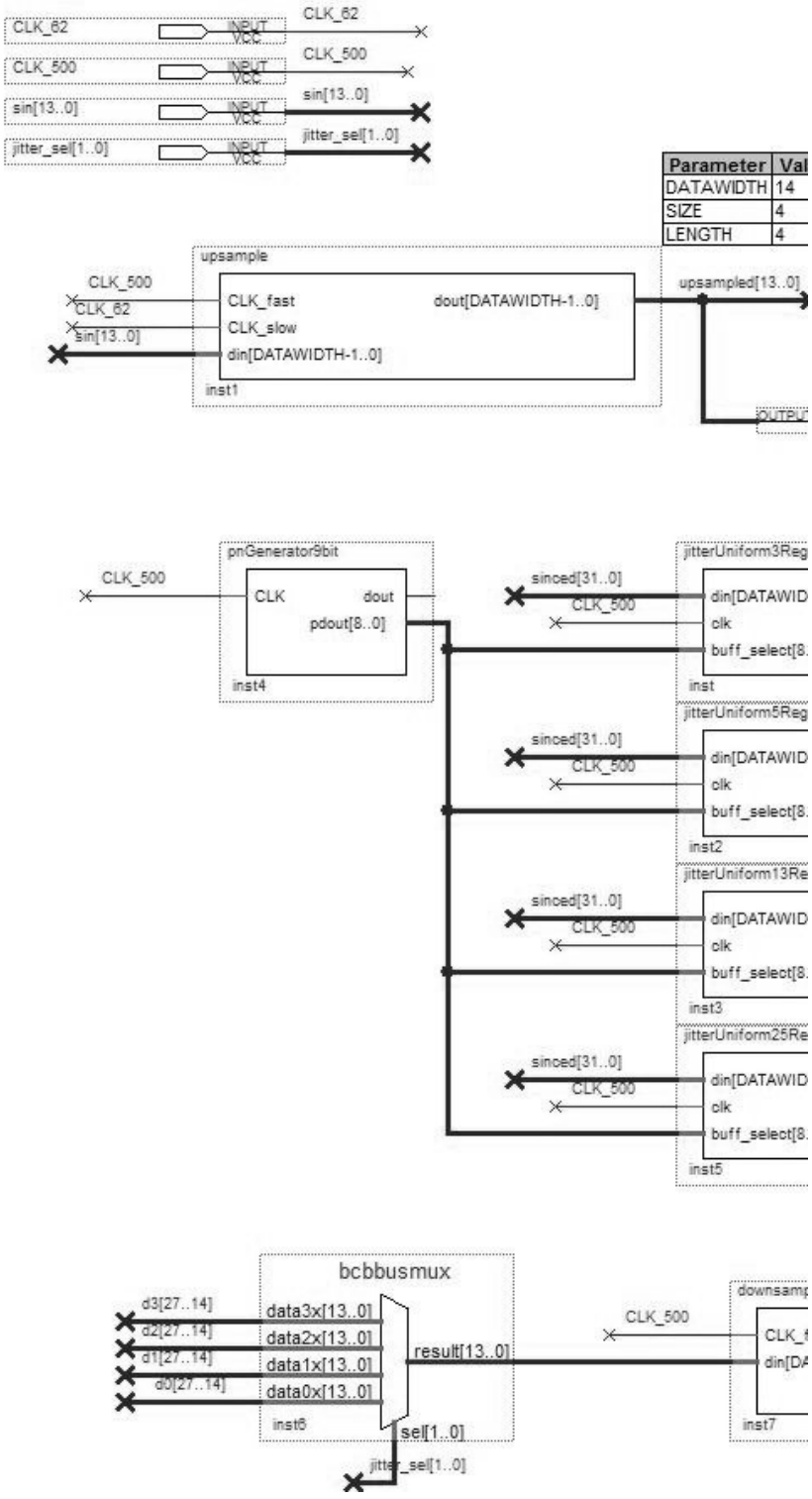

Figure E.6: Part 2 of 2: block diagram of the Verilog implementation used.

Vita

Candidate's full name: Brandon Christopher Brown

Univeristy Attended: Queen's University, B.Sc.E., 2006

Publications:

Howard Li and Brandon C. Brown, "Non–time based motion control for multiple intelligent vehicles," *Canadian Conference on Electrical and Computer Engineering 2009 (CCECE'09)*, pp. 646-649, 3-6 May 2009.