

A Reconfigurable Four-Channel Transceiver Testbed with Signalling-Wavelength-Spaced Antennas

by

J. Andy Harriman

B.Sc.E., University of New Brunswick, 2004

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

Master of Science in Engineering

In the Graduate Academic Unit of Electrical Engineering

Supervisor(s):	Mary E. Kaye, B.Sc.E., M.Eng. Brent R. Petersen, B.Eng., M.A.Sc., Ph.D.
Examining Board:	Yevgen Biletskiy, M.Sc.C.S., Ph.D. Dennis Lovely, B.Sc., Ph.D., Chair Julian Meng, B.Sc.E., M.Sc.E., Ph.D.
External Examiner:	Kenneth Kent, B.Sc., M.Sc., Ph.D.

This thesis is accepted

Dean of Graduate Studies

THE UNIVERSITY OF NEW BRUNSWICK

September, 2006

© J. Andy Harriman, 2006

To Nana and Grampy.

Abstract

We propose a novel software-defined radio implementation of a four-channel transceiver testbed with signalling-wavelength-spaced antennas. The radio frequency side of the system is implemented using commercial off-the-shelf products, whereas the baseband processing portion of the system is implemented on a pair of Altera® Stratix® II EP2S180 development boards with high-speed Analog Devices analog-to-digital converter daughter boards. The Altera® field programmable gate array is used to implement digital signal processing algorithms and to affect centralized control over the entire system. A goal was to create a testbed for algorithms and radio channel measurements.

Acknowledgements

I would like to thank my supervisors Brent Petersen and Mary Kaye for their generous support and guidance throughout my research work.

I would like to thank the ECE technical staff for their help with various technical issues throughout the course of the project. Specifically, I would like to thank Bruce Miller for his routing of my seemingly never ending supply of circuit boards, Blair Allen for his technical assistance as well as creation of many purchase orders, and Kevin Hanscom for his selfless loaning of equipment and his technical assistance with coaxial cable connectorization. Special thanks to Troy Lavigne, who on numerous occasions provided excellent support and knowledge of FPGA technology and software as well as debugging ideas.

I would like to thank Ian Veach for his technical expertise and support in the electronics area and the development of various interface boards, Peter Jacobs for his guidance in the area of cabling and RS-232 communications, and Denise Burke, Karen Annett, and Shelley Cormier from the ECE office for their kindness and support.

My thanks also go out to the staff at LyrTech for their support and their selfless loan of a SignalMaster system for my evaluation.

This research was supported by the Atlantic Innovation Fund from the Atlantic Canada Opportunities Agency, and by Aliant, our industrial partner. In particular, I would like to thank David M. Brown from Aliant for all his support. We thank CMC Microsystems for providing the FPGA software development tools via their

System-on-Chip Research Network Canadian Foundation for Innovation grant.

Table of Contents

Dedication	ii
Abstract	iii
Acknowledgments	iv
Table of Contents	vi
List of Tables	x
List of Figures	xi
Abbreviations	xiii
1 Introduction	1
1.1 Background and Literature Review	1
1.2 Thesis Contribution	2
1.3 Thesis Structure	4
2 System Design Process	5
2.1 Requirements Analysis	5
2.1.1 Design Decisions	5
2.2 System Architecture Design and Hardware Selection	11

2.2.1	FPGA Development Platform	11
2.2.2	Transmit Side	15
2.2.3	Receive Side	20
2.2.4	Baseband Processing	24
2.2.5	Centralized Control	24
2.2.6	Testing and Debugging Equipment	28
3	System Implementation	30
3.1	Transmit Side	30
3.1.1	Data Generation	30
3.1.2	FPGA Interfacing to the Hittite Modulator Board	32
3.1.3	Data Rate Calculation	37
3.2	Receive Side	37
3.2.1	Clock Distribution	37
3.2.2	Sampling the Data	40
3.3	Baseband Processing	42
3.3.1	Data Recovery and Demodulation	44
3.3.2	Carrier Recovery	50
3.3.3	LMS Adaptive Filtering	52
3.4	Centralized Control	54
3.4.1	Black Box® RS-232 Control	56
3.5	System Configuration	57
4	System Development and Testing	58
4.1	Component Testing and Integration	59
4.2	Interfacing SignalTap® and MATLAB®	64
4.3	Implementation Issues	65

5	Summary and Future Work	67
5.1	Summary of Work Completed	67
5.2	Future Work	68
	Bibliography	69
	Appendices	73
A	System Specifications	73
A.1	Altera® Pinouts to AD6645 ADC Boards	73
A.2	Cabling and System Interconnections	73
A.3	Black Box® COS Configuration	77
A.4	System Test Setup	77
B	MATLAB® Simulation and Debugging Source Code	79
B.1	LMS Adaptive Filter Array	79
B.2	Constellation Plotter	88
B.3	QAM Synchronizer	90
B.4	SignalTap® to MATLAB® Example	93
B.4.1	Binary-to-Decimal Conversion Scripts	94
C	Design Source Code	98
C.1	PN Generators	98
D	Custom PCB Board Designs	101
D.1	FPGA to Hittite Signal Conditioning Circuit	101
D.2	Hittite Bias Circuit	101
D.3	Clock Signal Level Translator	101
D.4	GPIO-to-SMA Testpoint Board	108
E	Modifications to the Power Supply	110

List of Tables

1.1	Testbed Features Comparison	4
2.1	FPGA vs. Microprocessor [1]	10
3.1	Primitive Prime Generating Polynomials [2]	31
3.2	Required Gain for Radio PLL Lock	40
3.3	RS-232 Communication Parameters [3]	56
A.1	ADC Board to Altera® FPGA Board Connector J5 Pinout [2]	74
A.2	ADC Board to Altera® FPGA Board Connector J6 Pinout [2]	74
A.3	Testbed Cable Inventory	76
A.4	RS-232 Connector Wiring	77
A.5	COS DIP Switch Positions	78
E.1	Power Supply Wiring	110

List of Figures

2.1	AOR AR5000AC Wideband Radio	7
2.2	AOR AR5000AC Wideband Radio Schematic [4]	8
2.3	Hittite HMC497LP4 Direct Quadrature Modulator Evaluation Board	9
2.4	Altera® Stratix® II EP2S180 DSP Development Kit [5]	16
2.5	Raised Cosine Curves [6]	18
2.6	Aeroflex 2025 AM/FM Signal Generator	19
2.7	Simplified Transmitter Structure	20
2.8	Analog Devices AD6645/PCB Evaluation Board	21
2.9	Simplified Receiver Structure	22
2.10	Antenna Factory F02400-8 Antenna	24
2.11	Space-Time Receiver Structure	25
2.12	Black Box® SWE-854A-R2 8-Port COS	26
2.13	Central Control Structure	27
2.14	Testbed Topology	29
3.1	LRS Generator Circuit Architecture	32
3.2	FPGA to Hittite Signal Conditioning Circuit	33
3.3	FPGA to Hittite Signal Conditioning PCB	35
3.4	Hittite Bias Circuit Board	36
3.5	Clock Signal Level Translator PCB	39
3.6	System Clock Interconnection Architecture	41

3.7	Mounted Baseband Processing Hardware	43
3.8	Spectrum of the Sampled IF Data	44
3.9	Spectrum of Mixed Inphase Data	46
3.10	Spectrum of Digital Down Converted Inphase Signal	47
3.11	Implemented Demodulator Architecture	48
3.12	Demodulated I and Q Constellation	49
3.13	Demodulated I and Q Constellation with Phase Corrected	51
3.14	QAM Synchronizer Circuit [6]	52
3.15	MATLAB® Simulation Results for Carrier Recovery Loop	53
3.16	MATLAB® Simulation Results for LMS Adaptive Filter Array	55
D.1	Hittite Signal Conditioning Circuit Schematic, Page 1/2	102
D.2	Hittite Signal Conditioning Circuit Schematic, Page 2/2	103
D.3	Hittite Signal Conditioning Circuit PCB Layout	104
D.4	Hittite Signal Conditioning Circuit Simulation	105
D.5	Hittite Bias Circuit Schematic	106
D.6	Clock Level Translator Circuit Schematic	106
D.7	Clock Level Translator Circuit Layout	107
D.8	GPIO-to-SMA Testpoint Board Schematic	108
D.9	GPIO-to-SMA Testpoint Board Layout	109
E.1	Power Supply Wiring Modifications	111

List of Symbols, Nomenclature or Abbreviations

ADC	Analog-to-Digital Converter
AGC	Automatic Gain Control
BPSK	Binary Phase Shift Keying
CNSR	Communication Networks and Services Research
COS	Code Operated Switch
COTS	Commercial Off-The-Shelf
cPCI	Compact PCI
DAC	Digital-to-Analog Converter
dB	DeciBels
dB _i	DeciBels relative to isotropic
dB _m	DeciBels relative to one milliwatt
DCE	Data Communications Equipment
DR	Dynamic Range
DSP	Digital Signal Processing
DTE	Data Terminal Equipment
FFT	Fast Fourier Transform
FIR	Finite Impulse Response
FPGA	Field Programmable Gate Array

GiB	Gibibyte, 2^{30} bytes, 1073741824 bytes
GPIO	General Purpose Input/Output
GSPS	Giga-Samples Per Second
HDL	Hardware Description Language
I	Inphase
IF	Intermediate Frequency
ISI	Intersymbol Interference
ISM	Industrial, Scientific, and Medical
LO	Local Oscillator
LOS	Line of Sight
LMS	Least Mean Square
LRS	Linear Recursive Sequence
LVTTL	Low-Voltage Transistor-Transistor Logic
MiB	Mebibyte, 2^{20} bytes, 1048576 bytes
MIMO	Multiple-Input-Multiple-Output
MSym/s	Mega-Symbols Per Second
MSPS	Mega-Samples Per Second
PCB	Printed Circuit Board
PG	Processing Gain
PLL	Phase Locked Loop
PN	Pseudorandom Noise
Q	Quadrature
QAM	Quadrature Amplitude Modulation
RF	Radio Frequency
RSS	Radio Standards Specification
SDR	Software Defined Radio
SNR	Signal-to-Noise Ratio

ST	Space-Time
SWAP	Signalling Wavelength Antenna Placement
TI	Texas Instruments
V_{DC}	Volt Direct Current
VHDL	Very-High-Speed Integrated Circuit Hardware Description Language
V_{pp}	Volt Peak-to-Peak
VSWR	Voltage Standing-Wave Ratio

Chapter 1

Introduction

In the area of multiple-input-multiple-output (MIMO) antenna arrays, it is often difficult to define mathematically or model all factors in a real-world environment. Thus, the reliability of simulations are limited by the accuracy of the models used [7]. This thesis is part of a larger project called Communication Networks and Services Research (CNSR). The goal of this thesis is to develop a centrally controlled, reconfigurable, four-channel radio testbed which takes advantage of space-time techniques as well as new antenna spacing theories. This testbed will be used for experimental research in the areas of antenna system, algorithms, and radio channel estimation, to name a few. Furthermore, it will enable researchers to take advantage of the rapid prototyping capabilities of field programmable gate array (FPGA) technology, in order to better investigate their hypotheses.

1.1 Background and Literature Review

Currently, there is a great deal of academic and industrial interest in the area of MIMO radio systems. This movement stems from the throughput gains achieved through the use of multiple transmit and receive antennas, first demonstrated by Telatar [8]. Since that time, there have been many testbeds, and commercial systems

alike, which make use of MIMO technology to increase throughput and reliability [7, 9, 10]. Three testbeds in particular highlight the different types of testbeds currently in existence. The first is a four transmitter, four receiver testbed from the University of California, Los Angeles (2005) which is designed for indoor use, has a LEGO® robotic rail system to move the antennas along a fixed track, and is remotely controllable via the internet [7]. The second system was built at Virginia Polytechnic in 2005. It consists of one transmitter and one receiver, and is designed for ultra wideband communication algorithm testing with sampling rates up to 8 giga-samples per second (GSPS) [11]. The third system, developed at McGill University in 2003, supports up to twelve transmitters and twelve receivers and is built from COTS components which reside in a rack mount system.

Recent findings by Zhu et al. [12] and Yanikomeroglu et al. [13] suggest that receive antennas should be separated by at least one signalling wavelength rather than some multiple of the carrier wavelength, in near line-of-sight (LOS) environments. According to Zhu, this leads to signalling wavelength antenna placement (SWAP) gain; Yanikomeroglu and Sousa called this the chiplength [13].

1.2 Thesis Contribution

The goal of this thesis is to develop a centrally controlled, highly reconfigurable, four-channel radio testbed which takes advantage of space-time techniques as well as new antenna spacing theories. This testbed will be used to receive signals from commercial radio systems, to make radio channel measurements, and to enable researchers to take advantage of the rapid prototyping capabilities of FPGA technology, in order to better investigate their hypotheses.

The system is comprised of four transmitters and four receivers. A space-time (ST) receiver has been designed in order to distinguish amongst users and provide

diversity gain. The receive antennas are separated by one signalling wavelength, in order to take advantage of SWAP gain. We used the 2.4 GHz industrial, scientific, and medical (ISM) band for this thesis with a 10 Mbps data rate, and a symbol rate of 5 Mega-Symbols Per Second (MSym/s). Based on signalling-wavelength-spaced antennas, this corresponds to receiver inter-antenna spacing of 60 meters (with 100% excess bandwidth),

$$\begin{aligned}\lambda_g &= \frac{c}{f_g} \\ &= \frac{3 \times 10^8}{5 \times 10^6} \\ &= 60 \text{ m.}\end{aligned}\tag{1.1}$$

In order to allow for more flexibility and to avoid having to create our own radio frequency (RF) front end, we have chosen suitable high-end radio receivers with 10.7 MHz intermediate frequency (IF) outputs, automatic gain control (AGC), and phase locking capabilities to interface to the system.

The MIMO testbed, developed in this thesis, will provide researchers with a rapid prototyping platform for algorithm analysis and radio channel measurements. The use of signalling-wavelength-spaced antennas affords the capability to explore the real-world effects of SWAP gain. However, this is merely one potential application of the testbed. The functionality of the testbed allows most, if not all, parameters to be modified. The antennas are physically repositionable, while many other parameters such as the carrier frequency, baseband processing, and RF front-end configuration are under centralized software control. Furthermore, the current hardware platform supports up to eight receive channels which could be useful for future work with Aliant’s base station in the 1900 MHz range.

[Table 1.1](#) compares some of the main features of the testbed developed in this

Table 1.1: Testbed Features Comparison

Feature	UCLA	VAP	McGill	UNB
Repositionable antennas	✓			✓
SWAP Gain				✓
Central Control	✓	✓	✓	✓
FPGA Based	✓	✓	✓	✓
Portable				✓
Num. of Channels	4	1	12	4*

thesis to those of the existing testbeds outlined above.

*Note: The system has been designed for seamless expansion with the addition of more FPGA boards and ADC boards.

1.3 Thesis Structure

The remainder of this document is broken down into four chapters. [Chapter 2](#) describes the system requirements analysis, high-level design decisions, and the subsequent architecture design and hardware selection process. [Chapter 3](#) outlines the details of the system implementation, and [Chapter 4](#) describes the development process along with the testing procedures used during the implementation of the system. [Chapter 5](#) summarizes the work completed in the thesis and outlines potential future work for the project.

Chapter 2

System Design Process

2.1 Requirements Analysis

In order to begin to design a testbed for wireless channel measurements, as with any other engineering endeavour, it is crucial to first understand the requirements for the final product. In this case, we were interested in building a radio testbed which would not only meet our requirement for a receiver implementation, but which would also provide a useful tool for future researchers to receive signals from commercial systems as well as test new theories and algorithms in an easily reconfigurable real-world environment. At the onset of system design, the best frequency band for experimental communication was chosen to be the 2.4 GHz ISM band due to the fact that it is freely available to use, provided that the requirements outlined in the Canadian Radio Standards Specification (RSS) RSS-210 are satisfied [14].

2.1.1 Design Decisions

Based on the general objectives outlined above, some high-level design decisions were necessary in order to proceed with the hardware selection and system design stages. It was decided that it would be best to build the system from mostly com-

mercial off-the-shelf (COTS) components in order to make the minimize the design time and reduce cost. Moreover, a modular system constructed of COTS components would allow a much greater degree of reconfigurability and interoperability than an application specific design. By selecting reconfigurable general purpose components, with standardized interfaces to interconnect to the rest of system, the system was made capable of supporting many different applications, including those using many frequency bands.

In order to enable the system to take advantage of SWAP gain, as well as the diversity gain and interference rejection capabilities of spatially multiplexed systems, it was decided that our testbed would be implemented as a MIMO system. To take advantage of ST techniques, without reaching the point of diminishing returns in terms of the number of antenna elements versus signal-to-noise ratio (SNR), and without exceeding our budget limitations, it was decided that our MIMO architecture would consist of four transmitters and four receivers (4x4).

Based on this decision it became clear that we would require the hardware to generate, as well as receive, four channels of data simultaneously. In keeping with the desire to conserve both time and financial resources, some components of the system were selected because they were already available in the department and had been used for previous work on the CNSR project. Two items purchased by a previous graduate student for his thesis work were of particular interest; specifically, two AOR AR5000AC wideband radio receivers and a Hittite direct quadrature modulator evaluation board. The AOR radio receiver was chosen to act as our RF receiver front-end partially to avoid having to create our own RF front-end. However, the main reason for choosing to use the AOR radio was due to its excellent features. The most important features for our application being: wideband frequency coverage from 10 kHz to 2600 MHz with no blocking of the cell band frequencies, selectable AGC function before the IF output, frequency locking capabilities to an external signal,

RS-232 remote control of radio functions, and a relatively low second stage IF output at $10.7 \text{ MHz} \pm 5 \text{ MHz}$ available before any processing is done on the data inside the radio; the first IF stage is at 622 MHz [4]. The relatively low frequency IF output of the second stage is helpful because it greatly reduces the sampling rate required to digitize the received signal. Figure 2.1 shows the front panel of the AOR AR5000AC radio, while Figure 2.2 shows the internal architecture of the radio.



Figure 2.1: AOR AR5000AC Wideband Radio

The second available component, was a Hittite HMC497LP4 direct quadrature modulator evaluation board. This board, shown in Figure 2.3, performs a quadrature amplitude modulation (QAM) on a pair of input signals, inphase (I) and quadrature (Q). This board is capable of modulating (sine/cosine) incoming digital signals from

DC to 700 MHz onto a carrier sinusoid ranging from 100 to 4000 MHz with a typical gain of 5 decibels (dB) relative to one milliwatt (dBm) or approximately 3.16 mW at 2.4 GHz [15]. Due to the excellent capabilities of the Hittite evaluation board this is the choice for our RF transmitter front-end.

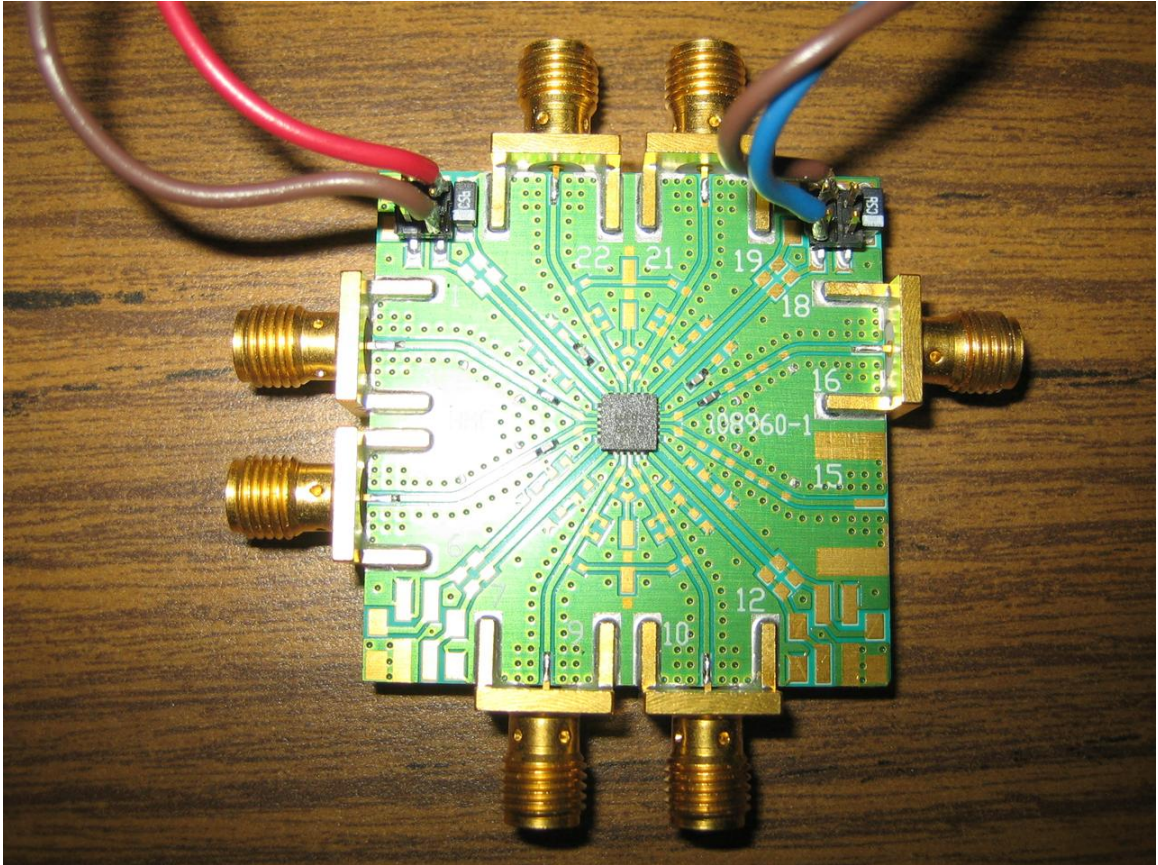


Figure 2.3: Hittite HMC497LP4 Direct Quadrature Modulator Evaluation Board

Once the selection of the RF transmit and receive front-ends was complete, the remainder of the system was designed around them. The choice of RF hardware imposed several design restrictions which were taken into account. The choice of radio receiver forced our work to be done at frequencies under 2.6 GHz, with a bandwidth no greater than 10 MHz, and analog-to-digital converters (ADCs) with sampling rates capable of digitizing the radios' IF output. Furthermore, the Hittite evaluation board

Table 2.1: FPGA vs. Microprocessor [1]

	Itanium 2	Virtex 2VP100
Technology	0.13 Micron	0.13 Micron
Clock Speed	1.6 GHz	180 MHz
Internal Memory Bandwidth	102 GB/s	7.5 TB/s
Power Consumption	130 W	15 W
Peak Performance	8 GFLOPS	38 GFLOPS
Sustained Performance	2 GFLOPS	19 GFLOPS
I/O External Memory Bandwidth	6.4 GB/s	67 GB/s

limits the baseband inputs to less than 700 MHz and the carrier frequency to less than 4000 MHz. However, due to the radio bandwidth of 10 MHz, and tunable frequency range of 10 kHz up to 2.6 GHz, both of these factors are not of any consequence. Thus, there is no effective limitation due to the modulator.

In order to maximize the reconfigurability and usefulness of the testbed, the decision was made, from the onset, that the core of the design would be FPGA-based using software defined radio (SDR) techniques to process the signals in digital form at baseband. Moreover, we wanted the generation and later processing of the signals to both take place within the FPGA, in order to have the greatest degree of both control and configurability over the system. [Table 2.1](#) shows an up-to-date comparison between an FPGA device and a microprocessor. This highlights many of the excellent features of FPGA technology for DSP applications, which include much higher memory bandwidths, and data throughput, with a significantly lower clock speed. These results are possible due to the parallel capabilities of FPGA technology as compared to the sequential operation of a microprocessor. Due to the often parallel and high speed nature of many DSP applications, an FPGA architecture was chosen for this thesis.

Another useful feature we wanted to have for our system, was the ability to remotely control all parts of the system from a central FPGA. Thus, any further components selected needed some mechanism for remotely controlling them, preferably

RS-232 due to the fact that the AOR radios have this built in, as well as the relative simplicity and widespread availability of RS-232 hardware.

To sum up the high-level design requirements, we wanted to build a 4x4 MIMO radio testbed from COTS components which is centrally controlled using FPGA technology to generate and process the signals. The system was required to be easily expandable and reconfigurable, while meeting our budget limitations. Furthermore, the system was required to take advantage of SWAP gain by separating the antennas on the scale of one signalling wavelength.

2.2 System Architecture Design and Hardware Selection

With the goals previously outlined, the system architecture design along with the associated hardware selections necessary to meet the design goals, commenced. The following subsections describe the design of each of the subsystems along with explanations regarding any additional hardware selections.

2.2.1 FPGA Development Platform

The selection of FPGA hardware involves many choices. Currently, there are two main competitors in the mainstream FPGA chip manufacturing market, Altera® and Xilinx® (listed alphabetically), with countless other companies developing FPGA development platforms based on their FPGAs. The challenge is not only to choose which FPGA chip manufacturer is best for your application, but also to try and locate a development platform which suits your needs. This can be extremely tedious because each company describes their FPGA chip features using their own proprietary architectures and their associated terminology which does not directly translate for easy comparison with that of their competitors.

Based on the requirements of this thesis, and the best efforts of the author, two systems were found to have the most potential for use in the testbed design: an Altera® Digital Signal Processing (DSP) Development Board, and a LyrTech SignalMaster based system. At the core of the LyrTech system is the SignalMaster board which contains a Xilinx® Virtex-II FPGA (up to XC2V8000), a Texas Instruments (TI) TMS320C6701 32-bit DSP which operates at 167 MHz, and up to 128 MiB of RAM to name but a few of its features. The SignalMaster board sits within a rack mount system and is connected to a central Compact PCI (cPCI) bus. Within the rack itself, there is a Pentium® 4 processor, a 20 GiB hard disk, and an Ethernet connection. The SignalMaster board itself does not have any analog inputs or outputs. To add these features additional boards are required, such as the VHS-ADC and VHS-DAC boards which respectively add up to sixteen 14-bit analog inputs (105 mega-samples per second (MSPS)) or outputs and connect to the system via the cPCI bus. On each of the VHS-ADC and VHS-DAC there is a Xilinx® Virtex-II FPGA (Virtex-4 available on VHS-ADC at time of publishing), along with some onboard SDRAM [16, 17]. Indeed the LyrTech system has an incredible amount of processing power available to the user [18].

The second potential solution is the Altera® Stratix® II EP2S180 DSP Development Board. This board is a more general purpose FPGA board which contains the largest available Stratix® II device from Altera®, 180 million logic elements. As well, the Altera® development board has a variety of external switches, LEDs, and interfaces for additional components. The board features two 12-bit 125 MSPS ADCs and two 14-bit 165 MSPS digital-to-analog converters (DACs). The Altera® FPGA development board provides 32 MiB of onboard SDRAM, a Compact Flash card slot which can be used to store user data, an Ethernet port, an RS-232 interface, and a minimum of 82 general purpose input/output (GPIO) pins which are user programmable. Furthermore, the Altera® board provides two dedicated connectors for

connecting Analog Devices ADC evaluation boards, and a dedicated connector for interfacing a TI DSP evaluation board [5].

As one might expect, the LyrTech system is more expensive than the Altera® evaluation board, but also contains more processing power. It has the ability to support many more transmitters and receivers and is organized in a modular fashion which is consistent with the design objectives. It also has large amounts of disk space for recording data which could be useful for real-time applications as well as post-processing of data in an academic environment. To investigate the trade-off between the added complexity of the LyrTech system and the associated learning curve, the author arranged to borrow an evaluation system from LyrTech in order to give the system a trial run. Based on the two-week trial evaluation of the LyrTech system, as well as the previous generation of the Altera® system (DSP Development Kit, Stratix Professional Edition), available in the lab, a number of important points were noted.

At the time, the added features of the LyrTech system only added to the complexity of using it, especially at a low level. The system is meant to be used with the MATLAB® Simulink® package. Development using the standard Xilinx® development environment required that the user try to control the many different interactions among the various I/O standards, buses, and devices in the system. LyrTech did provide drivers for these tasks, however a substantial amount of work would be required in order to fully understand and take advantage of the many different modules. In the case of the Altera® development board, there was only one FPGA device, albeit very large, which limited the design possibilities but which also kept the degree of complexity to a minimum. The development environment for the Altera® development board is the developers choice of either Simulink® or Quartus® II, Altera's® proprietary development environment. Through the testing experience, the author found that the Simulink® software package was indeed a very powerful tool for designing and simulating at a high level, but not without a cost. Simulink® basically acts as

a modelling environment in which the designer can create a design and simply tell the software to generate code for the design. The code created by Simulink® is more like machine code and not easily readable, which leads one to wonder how difficult it would be to uncover a bug in the design at a low level. The Simulink® tools add another layer of complexity on top of the already growing stack of new knowledge for a beginner to learn in a short period of time. Having said that, it is the opinion of the author that a user who is well acquainted with using Simulink® might have the upper hand, in terms of time spent developing, on a user who is designing at a lower level.

Based on the experience gained through the trial evaluations of the two FPGA development systems, the following conclusions were made. For a new user and for the purposes of the work in this thesis, the inherent complexity of the LyrTech system along with the need to work with Simulink®, created a learning curve above and beyond the initial time involved with learning FPGA design and hardware design language implementation. The LyrTech system provided eight ADCs and eight DACs in the default configuration, whereas the Altera® system only provided two of each. However, due to the fact that the Altera® system was available for a lower cost than the LyrTech system, coupled with its many available GPIO pins, two Altera® FPGA boards can easily be doubled up at a lower cost than a single LyrTech system. That is, two Altera® FPGA development boards can be interconnected to provide a system with two large FPGAs, four ADCs, and four DACs. The DACs on the Altera® board are faster than those on the LyrTech VHS-DAC board, however the ADCs are only 12-bits, compared to the 14-bit ADCs on the LyrTech VHS-ADC board. In any software defined radio system, the ADCs and DACs play a crucial role and are of paramount importance because they effectively govern the overall system performance [19]. Furthermore, the LyrTech system has an onboard DSP processor which could be useful for baseband algorithms and control functions. The Altera® system can counteract

both of these factors thanks to its dedicated connectors for two ADCs and a DSP. By coupling two Altera® boards, four external Analog Devices AD6645 evaluation boards, the same ADC as on the SignalMaster, and a TI DSP evaluation board, the Altera® board overcomes the most crucial hardware differences between the two systems. Moreover, the size of the LyrTech system (rack mount) severely limits its mobility, which is a crucial point given that radio channel measurements will likely have to be taken outdoors given the large antenna separations required for SWAP gain and the desire for LOS testing.

For the aforementioned reasons as well as the fact that the Altera® system meets the design requirements and was available at a lower cost than the LyrTech system, the decision was made that we would purchase the Altera® hardware for the thesis. Another factor which contributed to this decision was that several other students in the department were also working with similar Altera® hardware and the Quartus® II software package. The LyrTech system, which is Xilinx® based, is an excellent product, but in the opinion of the author would be more suitable for a dedicated non-mobile application requiring large amounts of processing power such as a base station.

Figure 2.4 illustrates the Altera® Stratix® II EPS2180 DSP Development Kit along with many of its main hardware features. Note that the connector for the TI DSP is on the bottom side of the board.

2.2.2 Transmit Side

To meet the requirements of a 4x4 MIMO array, the system must be able to produce four signals to transmit. Each transmitter represents a single user in the system, enabling our system to simulate up to four simultaneous users at a time. As mentioned earlier, the signal generation is done within the FPGA board. Note that since the Hittite direct quadrature modulator board accepts inphase and quadrature

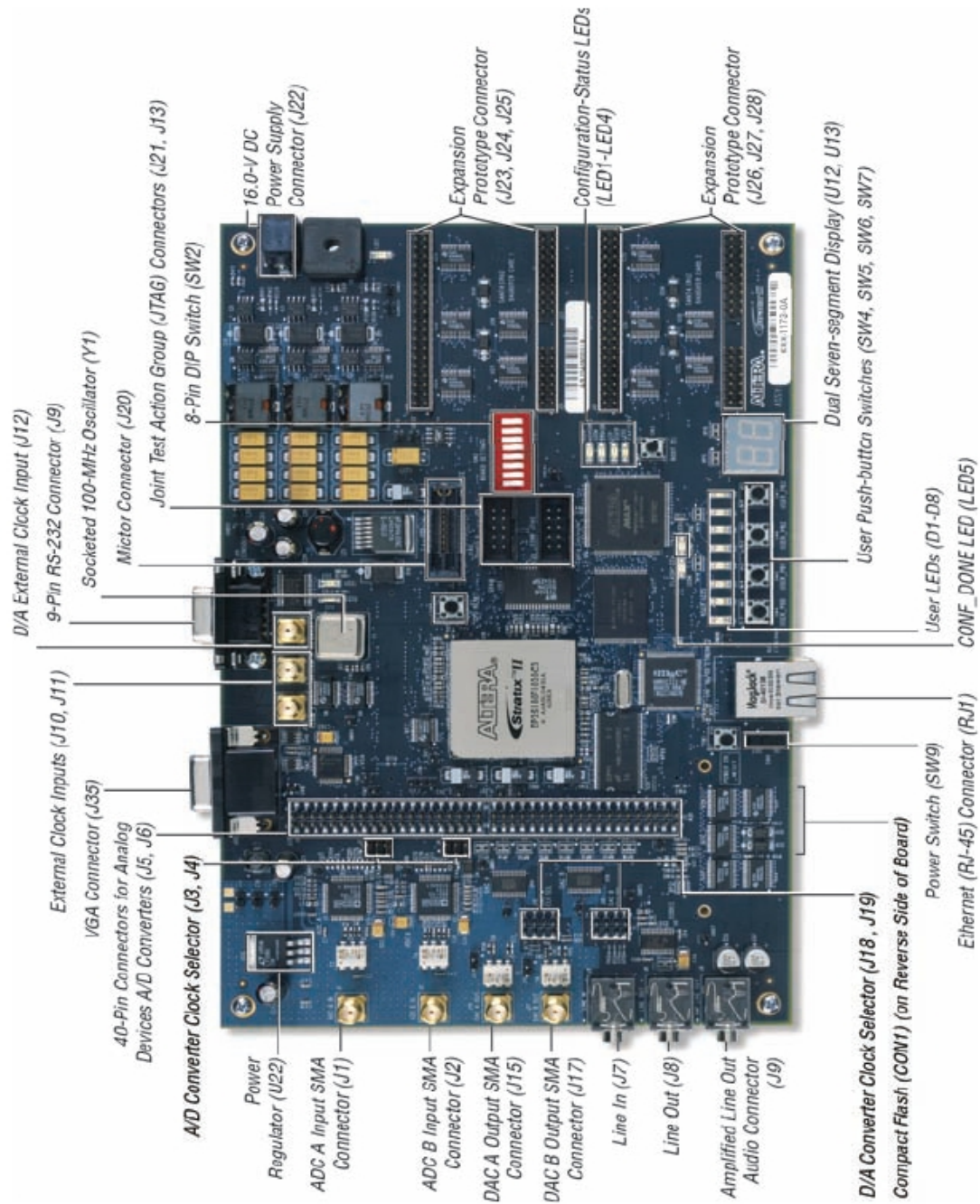


Figure 2.4: Altera® Stratix® II EP2S180 DSP Development Kit [5]

signals for each user, it is necessary to generate two signals for each of the four users; that is, four pairs of I and Q signals. For testing purposes, it is useful to know exactly what the transmitted signal was, in order to facilitate the recovery of the received signals through demodulation and equalization. To meet this need, eight linear recursive sequence (LRS) generators with different tap configurations can be used to generate a known distinct pair of I and Q signals for each user. Circuits to generate LRS sequences are relatively straightforward to implement in the FPGA, and the resulting bit streams can be output to the Hittite modulator boards using eight GPIO pins on the development board.

Ideally, the generated signals would be output from the FPGA boards to the Hittite modulator board using the onboard DACs. This would allow digital pulse shaping filters, such as a raised cosine, to be applied to the data stream inside the FPGA before transmission. However, due to the fact that we only have four DACs (two per Altera[®] board), we are forced to output the signals as purely digital waveforms in order to meet the goal of a 4x4 MIMO array. The advantage of using a rectangular pulse shape is that there is no intersymbol interference (ISI) since the pulse does not extend beyond its own interval. However, the Fourier transform of a square, or rectangular, function in time is a sinc pulse of infinite bandwidth in frequency, and as a result the rectangular pulse is not a particularly good choice for a communication system because it wastes bandwidth. A raised cosine pulse, in the frequency domain, on the other hand, has the advantage of limiting bandwidth while decaying quickly and having zeros crossing at the sampling instants [20]. To generate a raised cosine pulse, a damped sinc function is used in the time domain as shown in Equation 2.1 [6],

$$x(t) = \left(\frac{\sin \frac{\pi t}{T}}{\frac{\pi t}{T}} \right) \left(\frac{\cos \frac{\beta \pi t}{T}}{1 - \frac{4\beta^2 t^2}{T^2}} \right). \quad (2.1)$$

The variable β , $0 \leq \beta \leq 1$, is called the rolloff parameter or damping coefficient. As β increases, the time-domain response decays more rapidly, however the frequency domain raised cosine response bandwidth increases [20]. Figure 2.5 illustrates this tradeoff.

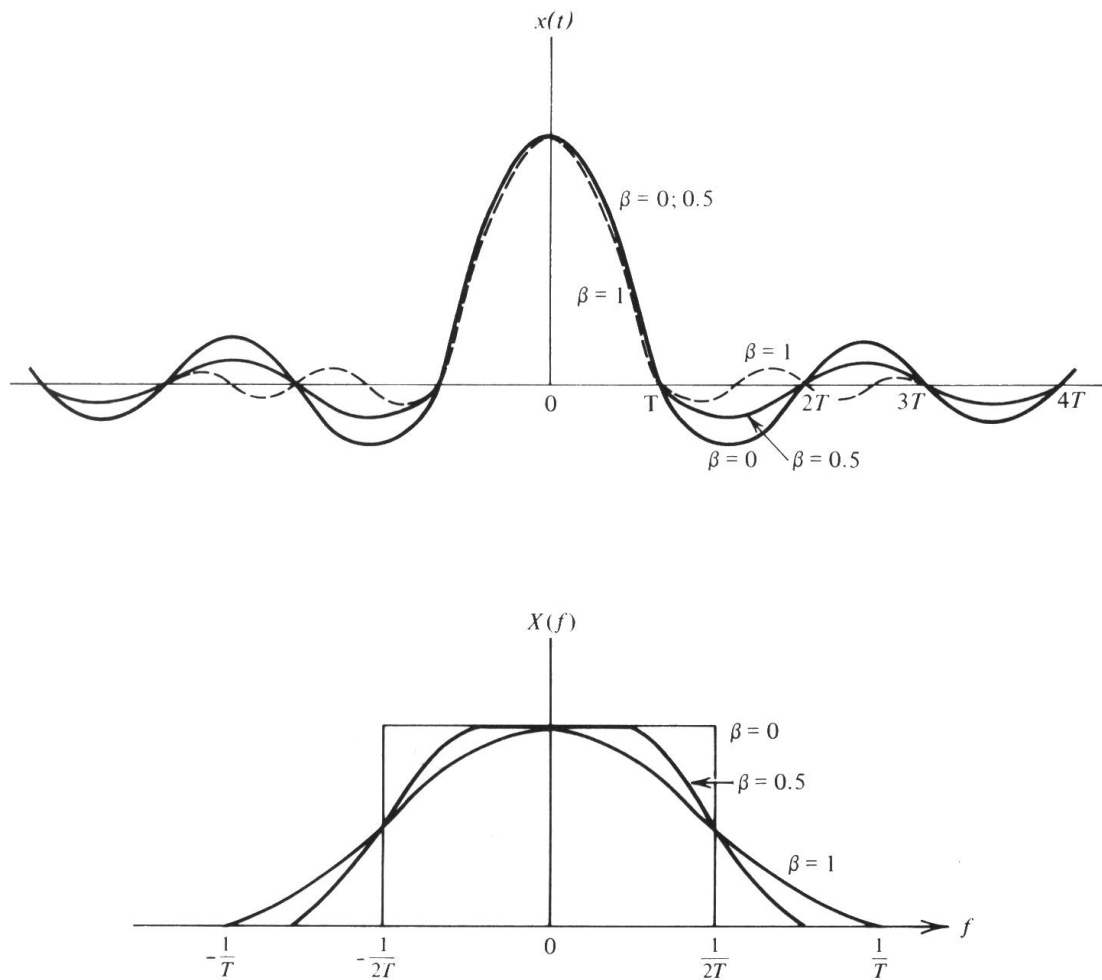


Figure 2.5: Raised Cosine Curves [6]

Due to the modular design of the system, the ability to use pulse shaping filters can be added at a later time by either adding two additional ADCs, or performing the modulation within the FPGA and then mixing it with the carrier externally. The latter would require only one DAC per user signal, both I and Q. With the current hardware we can support up to four users using this technique.

Once the digital signals have been created and output from the FPGA, all that is left is to modulate the data onto a 2.4 GHz carrier waveform and radiate it via the transmit antennas. As mentioned earlier, the Hittite HMC497LP4 direct quadrature modulator boards are used to modulate the data stream onto the carrier. To provide a carrier signal in the 2.4 GHz ISM band, such as 2.45 GHz, the Aeroflex 2025 AM/FM signal generator shown in Figure 2.6 was chosen, and purchased, based on its rich feature set and relatively low cost. As per the system requirements, it can generate signals from 9 kHz up to 2.51 GHz, covering the 2.4 GHz ISM band, and also provides RS-232 remote control functionality for all features, excluding the power button. As well, the signal generator provides a high stability 10 MHz frequency standard output which is used to synchronize the entire system [21].



Figure 2.6: Aeroflex 2025 AM/FM Signal Generator

Thus, modulation of the transmit signal can be achieved by applying the digital data streams generated within the FPGA board, along with the 2.4 GHz carrier sinusoid from the signal generator to the appropriate inputs of the Hittite modulator board. However, in order to interface the eight FPGA GPIO pins to the Hittite vector modulators, some voltage translation is required. Specifically, the FPGA board output uses the low voltage transistor-transistor logic (LVTTTL) standard 0-3.3 V [5], whereas the Hittite vector modulator requires a 1.6 Volt peak-to-peak (V_{pp}) input

with a 1.5 Volt direct current (V_{DC}) offset [15]. To achieve the required translation, a custom printed circuit board (PCB) was necessary. The design and fabrication of this board is described in the implementation section in Chapter 3. A simplified block diagram of an individual transmitter is shown in Figure 2.7. Note that since only one Hittite modulator board was available, it was necessary to purchase three additional units.

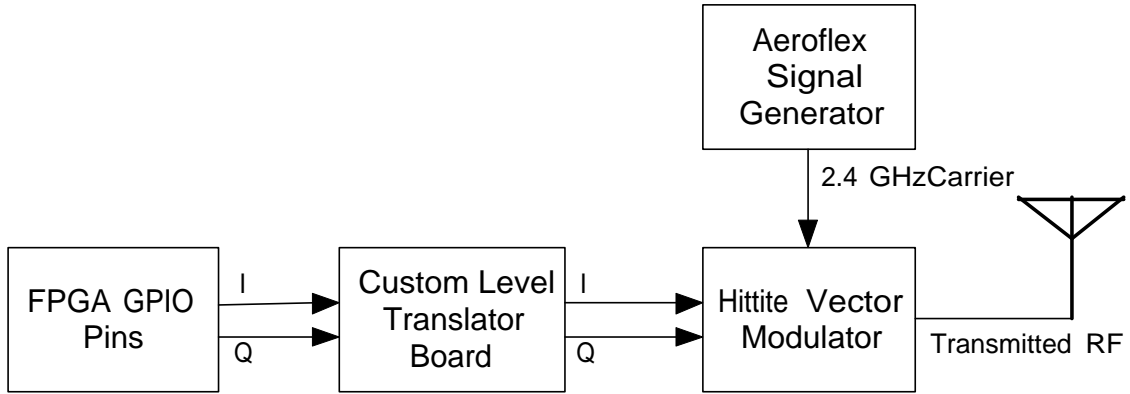


Figure 2.7: Simplified Transmitter Structure

Note that each of the I and Q signals are effectively a pair of binary phase shift keying (BPSK) signals which, when modulated, produce a 4-QAM constellation with I on the horizontal axis and Q on the vertical axis. The location of each of the transmit antennas is midway between each of the receive antennas, approximately 60 meters away from the centre of the array.

2.2.3 Receive Side

As with the transmit side, the receive side of the system consists of four identical circuits for receiving data. These receivers provide four channels of incoming RF data to be processed by the system. In order to simplify the RF front-end design of our receivers, we are using four AOR AR5000AC wide band all-mode receivers.

This required that two additional radio receivers be purchased for the thesis, to supplement the two that were already in our possession. The RF signals which arrive at each of the four receive antennas are down-converted to 10.7 MHz using the AOR radios, which provide a 10 MHz bandwidth IF output centered at 10.7 MHz.

Sampling of the IF signals is performed using four Analog Devices AD6645/PCB daughter boards, shown in [Figure 2.8](#), which have special headers allocated to them on the Altera® Stratix® II DSP Development Board. This allows two channels of incoming data to be received by each Altera® Development Board. The sampling rate of the ADCs is greater than 31.4 MHz in order to satisfy the Nyquist criterion. The conversion clock is provided by the FPGAs.



Figure 2.8: Analog Devices AD6645/PCB Evaluation Board

To avoid synchronization issues, each of the four AOR radios as well as the two Altera® FPGA boards are locked to a common 10 MHz reference. The common reference signal for the entire system comes from the Aeroflex signal generator and is provided in order to frequency lock each of the four radios as well as the FPGA circuitry. AGC is provided by the AOR radios. [Figure 2.9](#) illustrates the basic structure of each of the four receivers.

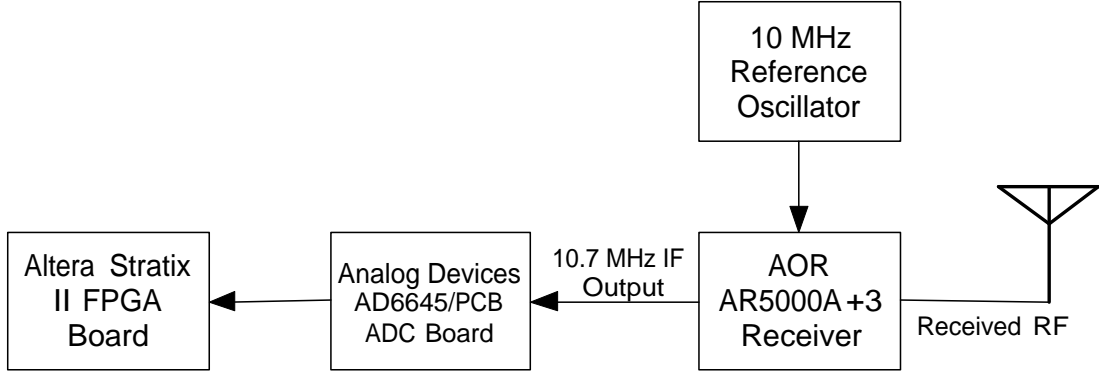


Figure 2.9: Simplified Receiver Structure

The SNR, or dynamic range (DR), of a digital receiver can be calculated as shown in [Equation 2.2](#), where N is the number of bits in the ADC. Note that f_s is the sampling frequency of our ADCs and B is the passband bandwidth, 10 MHz in this case. Ten times the log of the ratio of $f_s/2$ over B is called the processing gain (PG), in dB. From this relationship, it is clear that as f_s is increased beyond twice the passband bandwidth of $2B$, the SNR increases. This is due to the effect of spreading the quantization noise power over the bandwidth [\[22, 23\]](#).

$$SNR_{dB} = DR_{dB} = 6.02N + 1.76 + 10 \log_{10} \left(\frac{f_s}{2B} \right) \quad (2.2)$$

Ignoring the processing gain portion for the time being, the SNR for our receiver can be roughly approximated to 86.04 dB, above the quantization noise, as

shown in Equation 2.3. By oversampling the data the quantization noise is spread over a greater frequency range, resulting in a processing gain of 6.02 dB as shown in Equation 2.4. Thus, the total estimated dynamic range of the system is 92.06 dB.

$$\begin{aligned}
 SNR_{dB} &= DR_{dB} = 6.02N + 1.76 \\
 &= 6.02 \times 14 + 1.76 \\
 &= 86.04 \text{ dB}
 \end{aligned} \tag{2.3}$$

$$\begin{aligned}
 PG_{dB} &= 10 \log_{10} \left(\frac{f_s}{2B} \right) \\
 &= 10 \log_{10} \left(\frac{80}{2 \times 10} \right) \\
 &= 6.02 \text{ dB}
 \end{aligned} \tag{2.4}$$

Although this is a rough theoretical approximation of the best case scenario, our ADCs coupled with the AGC capabilities of the AOR radio receivers should provide more than enough dynamic range.

Identical antennas were chosen for both of the transmitters and receivers. For this we selected Antenna Factory FO2400-8 vertical antennas with a gain of 8 dBi relative to isotropic (dBi) and 83 MHz of bandwidth from 2.4-2.483 GHz, covering the 2.4 GHz ISM band [24]. As well, four Antenna Factory FO1710-8 vertical antennas with a gain of 8 dBi and frequency coverage from 1710-1990 MHz were purchased for receiving signals from commercial systems in the 1900 MHz range, although they are not used in this design [25]. Figure 2.10 shows the antennas which are used in this design for signalling in the ISM band, they are 0.6 m long. Note that the receive antennas are spaced 60 m apart from one another, and are connected to the central hub using coaxial cable, in order to take advantage of SWAP gain.



Figure 2.10: Antenna Factory F02400-8 Antenna

2.2.4 Baseband Processing

Once the IF signal has been sampled, processing within the FPGA can begin. A 5.7-15.7 MHz bandpass finite impulse response (FIR) filter removes any unwanted spectrum, followed by a demodulator circuit and a ST receiver, using the least mean square (LMS) adaptive algorithm to equalize the channel, reduce interference, and recover each user's transmission. In order to achieve this, the ST array design consists of four groups of four adaptive filters. Each of the four filters, within a single group, connects to a different receive antenna input, and each particular group is trained to isolate the signal of only one user in the system. The outputs of each of the four adaptive filters, in a particular group, are then combined to average the signals from the four antennas. This limits the number of users in the system to four, but provides diversity gain over the single antenna case. The ST receiver structure design which is implemented for this system is described by Paulraj et al. [26], and is illustrated in [Figure 2.11](#).

2.2.5 Centralized Control

The addition of a TI 6416 1 GHz fixed point DSP is intended to facilitate control algorithms and perhaps some low speed signal processing at baseband.

In order to achieve central control over the entire functionality of the system, the FPGAs need to be able to communicate with each of the individual COTS

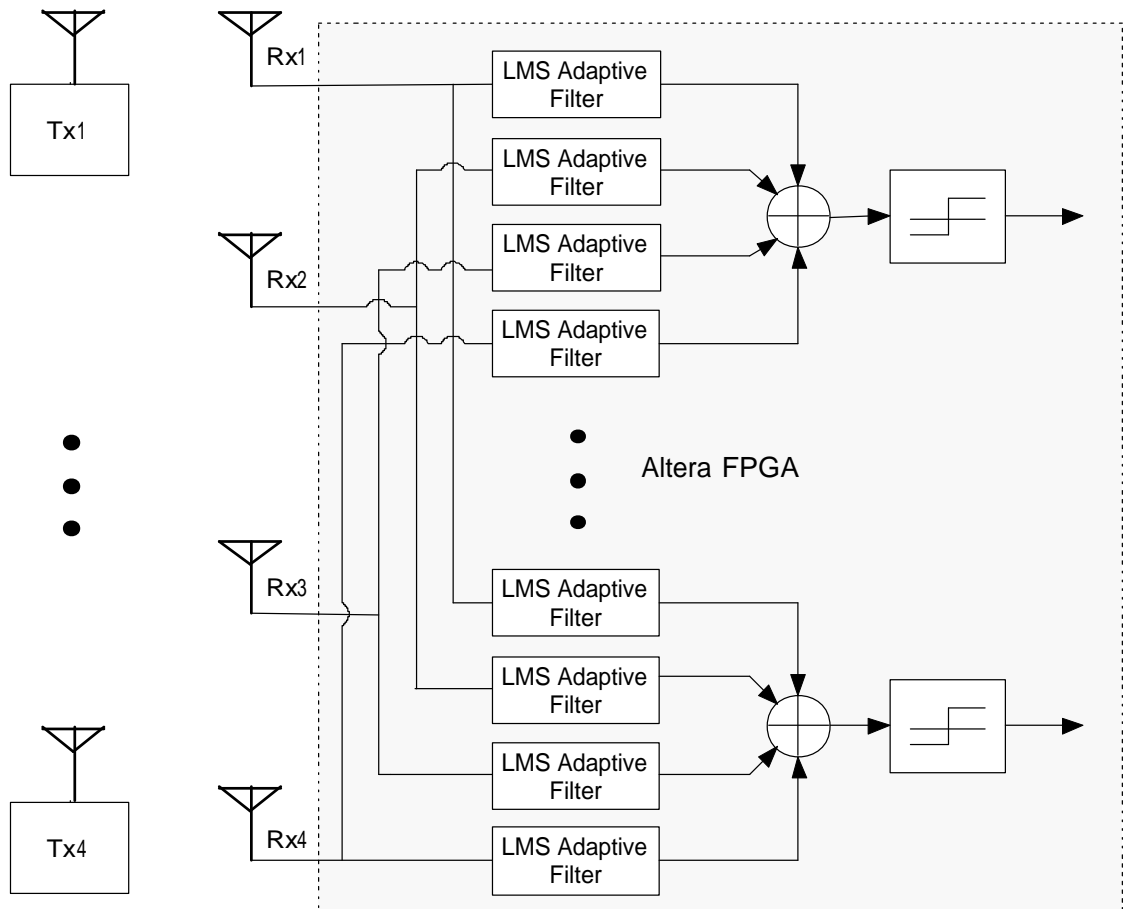


Figure 2.11: Space-Time Receiver Structure

components in the design. The first step in designing this control system was to select components which can be remotely controlled via RS-232. The Altera® FPGAs each have one RS-232 port, however there is a need to control at least five different devices simultaneously – four radios and the signal generator. To meet this need, a Black Box® SWE-854A-R2 (COS-8P) 8-port Code Operated Switch (COS), as shown in [Figure 2.12](#), was selected. This device allows one master to control up to eight slaves by controlling which slave is connected at any particular time. To control the switching between slaves, the master transmits a code word, which will not appear in the data stream, followed by a number indicating to which slave it would like to communicate. As such, it is possible to uniquely address each of the components, which enables remote control access to all the features of the radios and the signal generator. Unfortunately, the ability to broadcast the same message to multiple slaves is not available. Instead, each slave has to be addressed and signalled separately. The average time required to switch between slaves has been quoted by Black Box® technical support to be no greater than 10 ms [27]. The details regarding the configuration of the switch are provided in the implementation section.

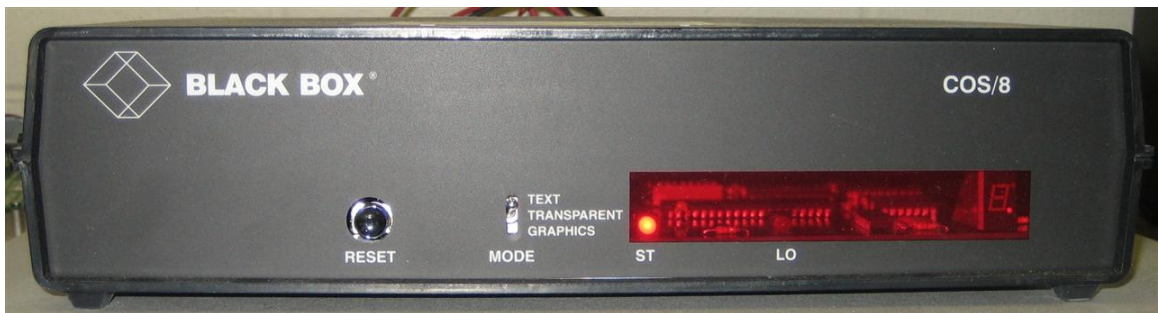


Figure 2.12: Black Box® SWE-854A-R2 8-Port COS

Since we also control the baseband transmit and receive functions directly via the FPGA, this provides complete control over the system. Advanced testing techniques such as swept frequency, for instance, can be achieved by simply sweeping

the tuned frequency of the radio receivers across the frequency spectrum of interest. Figure 2.13 illustrates the baseband side of the system and how it interacts with and controls the rest of the hardware in the system.

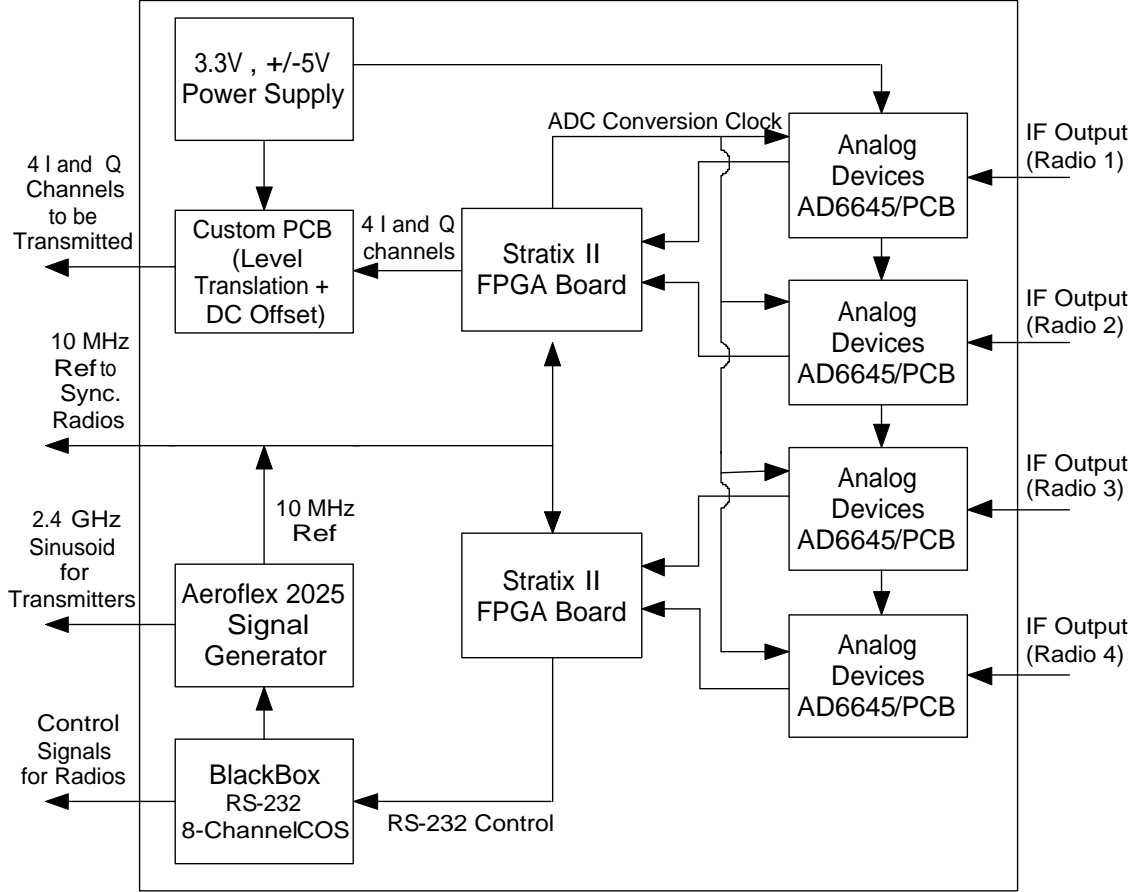


Figure 2.13: Central Control Structure

Due to the fact that our antenna array is spread over such a large area, several issues arise. Firstly, the location of each of the components becomes important. Since the incoming data rate to both the receive and transmit antennas is 2.4 GHz, there would be a significant amount of attenuation over 60 meters of coaxial cable. To avoid this problem, both the AOR radios and the Hittite vector modulator boards are situated close to their respective receive and transmit antennas. Thus, the signals

travelling through the 60 meter run of coaxial cable will be at the IF frequency of 10.7 MHz which, coupled with low loss cabling, is acceptable. The tradeoff of this decision is that the RF equipment must now be situated remotely at each of the eight antennas.

The second issue derives from the solution to the first issue, and our desire to control the array centrally. In order to synchronize all the AOR radios, a 10 MHz reference signal must be shared amongst them. Furthermore, centralized control over these radios requires that each have an RS-232 connection to the central hub. This translates to two 60 m runs of coaxial cable, plus one 60 m run of RS-232 cable per receive antenna. Combined with the I and Q, and 2.4 GHz reference signals sent to each transmit antenna, three 60 m runs of coaxial cable, the grand total comes to 1200 m of coaxial cable plus 240 m of RS-232 cabling. [Figure 2.14](#) shows the topology of the array along with the signals which are sent amongst the various components. Note that this figure is not to scale.

2.2.6 Testing and Debugging Equipment

To facilitate the testing and debugging process, a Lecroy WaveSurfer 424 200 MHz 2 GSPS oscilloscope was borrowed from another lab. As well, an AOR SDU 5600 fast Fourier transform (FFT) spectrum display unit which is designed to interface to and be controllable by the AOR AR5000AC radios, was purchased. This device can also work as a standalone unit.

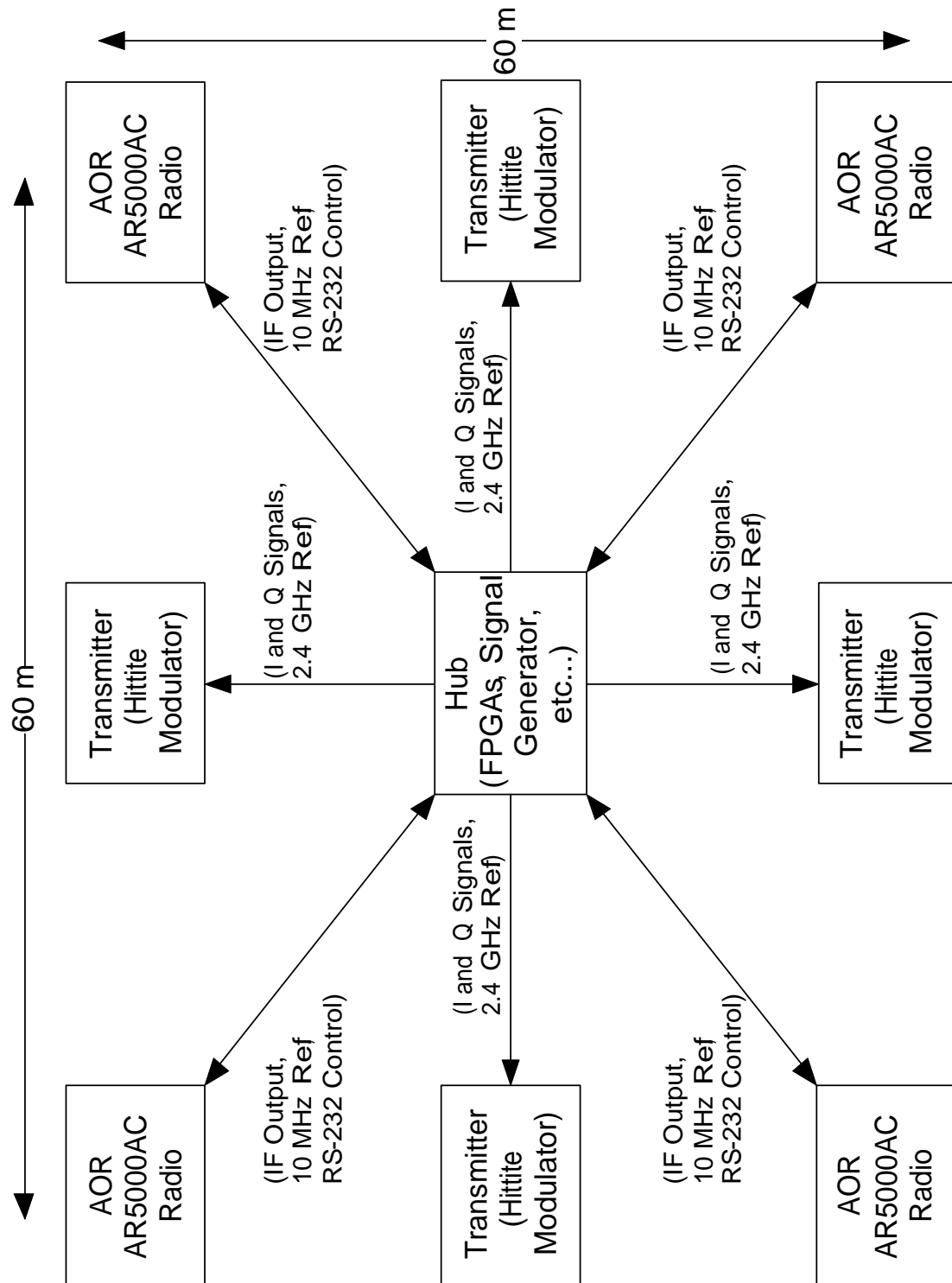


Figure 2.14: Testbed Topology

Chapter 3

System Implementation

This chapter describes the implementation specific details of the system design described in [Chapter 2](#), and as such, it is broken down in a similar fashion. The system was developed using the Verilog hardware description language (HDL) using the Altera® Quartus® II (version 5.1 service pack 1) development environment. The choice of Verilog was based solely on the fact that the author had prior experience with that language. It should be noted that this choice is not in any way a tradeoff, and that any future additions to the system can be written in either Verilog or Very-High-Speed Integrated Circuit HDL, more commonly known as VHDL.

3.1 Transmit Side

3.1.1 Data Generation

The first stage of development for the system was to be able to generate signals for transmission. As mentioned earlier, LRS generators are used to create predetermined random bit streams, otherwise known as pseudorandom noise (PN) sequences. Such circuits can be easily constructed using a shift register constructed of D flip-flops with configurable feedback taps. As implied by the name, the sequences in fact

appear to be random and have the same basic properties, but are fixed in both bit sequence and length. The bit sequence is determined by both the feedback tap connections and the length of the shift register. The bit sequence is not infinitely long however, it repeats at predefined intervals. In the ideal PN generator, the sequence is of length $2^N - 1$, where N is the number of flip-flops in the shift register. In this case, the sequence is called a maximal length sequence. A maximal length sequence can only be created by using a feedback tap polynomial which is a primitive prime [28]. The following definition holds of primitive prime polynomials, “For a prime $P(x)$ to give a maximum length sequence of length L , $P(x)$ must be a factor of $x^L + 1$ (and of no other smaller L). Such a prime is called primitive” [28]. In order to ensure that the ST receiver would have enough time to train properly, we chose to use tenth order primitive prime polynomials which result in a maximal length sequence of $2^{10} - 1 = 1023$ bits. Table 3.1 lists the primitive prime generating polynomials used in the system design, although any can be used. Note that this is not an exhaustive list. A complete table of irreducible polynomials over Galois fields GF(2), the binary case, through GF(5) up to order eleven can be found in Church’s “Tables of Irreducible Polynomials for the First Four Prime Moduli” [2].

Table 3.1: Primitive Prime Generating Polynomials [2]

Polynomial	Tap Configuration
$x^{10} + x^3 + 1$	(10,3,0)
$x^{10} + x^4 + x^3 + x + 1$	(10,4,3,1,0)
$x^{10} + x^5 + x^2 + x + 1$	(10,5,2,1,0)
$x^{10} + x^5 + x^3 + x^2 + 1$	(10,5,3,2,0)
$x^{10} + x^6 + x^5 + x^2 + 1$	(10,6,5,2,0)
$x^{10} + x^6 + x^5 + x^3 + x^2 + x + 1$	(10,6,5,3,2,1,0)
$x^{10} + x^7 + 1$	(10,7,0)
$x^{10} + x^7 + x^3 + x + 1$	(10,7,3,1,0)

Based on these generating polynomials, it is relatively straightforward to de-

velop an LRS generator circuit in Verilog. [Figure 3.1](#) shows the basic architecture of the circuit for the generating polynomial $x^{10} + x^3 + 1$. The source code for this module can be found in [Appendix C](#) and on the enclosed CD-ROM.

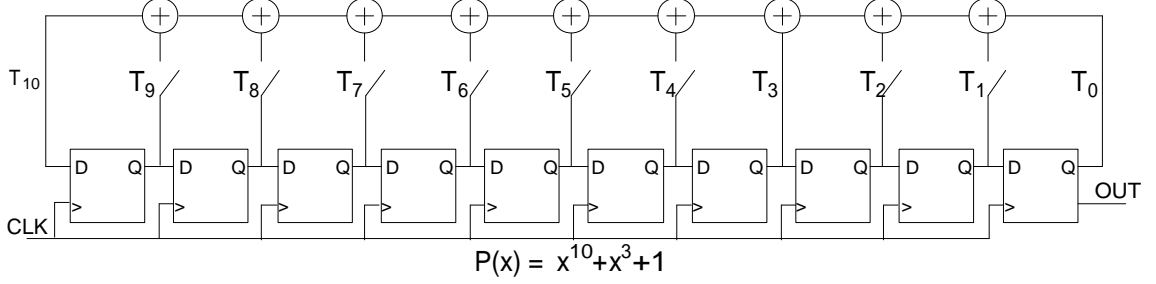


Figure 3.1: LRS Generator Circuit Architecture

3.1.2 FPGA Interfacing to the Hittite Modulator Board

Once the user signals have been generated within the FPGA, they are output using GPIO pins to the Hittite modulator board. However, in order to interface the LVTTTL, 0-3.3 V output from the FPGA, to the requirements of the Hittite modulator board of $1.5 V_{DC} + 1.6 V_{AC}$, a custom interface board was needed. Since the modulator board is situated near the transit antennas, at the far end of 91.4 m, or 300 feet, of cable, the signal conditioning circuitry does not only have to meet the level translation requirements, but also handle the drive current requirements. The design choice of 300 foot cable lengths is discussed in [Section A.2](#). Based on these requirements, a simple resistive network was selected and designed such that it would provide the necessary voltage translation and also appear as a 50Ω input looking inward at the cable. [Figure 3.2](#) illustrates the final resistive network design, along with the DC bias point simulation numbers. The circuit was simulated using PSpice®.

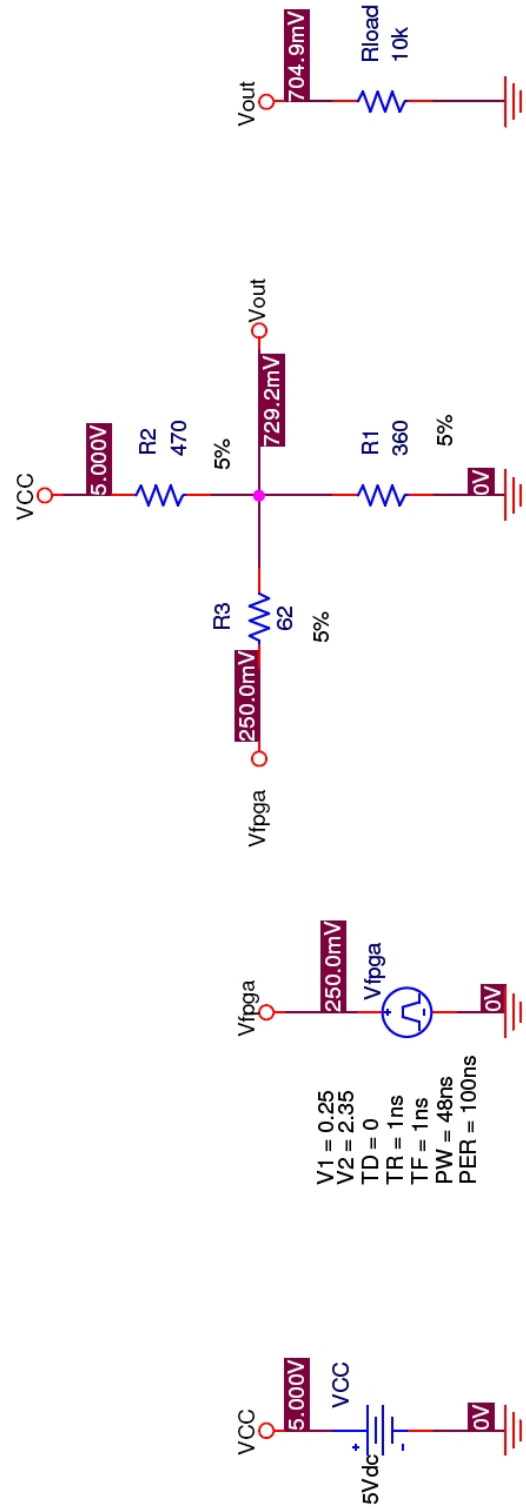


Figure 3.2: FPGA to Hittite Signal Conditioning Circuit

The input to the resistive divider circuit is driven through a TI SN74ALB16244 16-bit buffer/driver which provides up to 25 mA source and sink currents [29]. This easily meets the circuit requirements of 7.3 mA sink and 0.8 mA source (see [Appendix D](#) for details). As well, the voltage output of the buffer chip was measured, under real-world operating conditions, to be 0.3 V for logic low and 2.3 V for logic high. This was accounted for in the circuit design in order to ensure that the output met the specifications needed. The PSpice® simulation output, GPIO pins used, PCB schematic, and board layout are located in [Appendix D](#). [Figure 3.3](#) shows the finished circuit board mounted atop the GPIO header on the Altera® FPGA board. The eight SMA connectors situated at the top of the figure are the inphase and quadrature signal outputs, while the four SMA connectors on the right side of the board are the clock output signal to the external AD6645 ADC boards.

MATLAB® analysis of the Hittite modulator board, revealed that the inphase and quadrature components of the received signal were only 45 degrees out of phase rather than 90 degrees. Debugging efforts along with communications with Hittite technical support, showed that the setup of the board was incorrectly performed by previous users of the hardware. In fact, the $1.5 V_{DC}$ signal should be a common mode signal shared between both the negative and positive differential baseband I and Q inputs, as opposed to the single ended use in previous work – although this is not at all apparent from reading the board datasheet. To accommodate for this inconsistency, a second PCB board was constructed which provides a $1.5 V_{DC}$ output along with $1 \mu F$ terminating capacitors to ground for driving the I_n and Q_n negative inputs. As well, the negative RF input, RF_n , requires a large terminating capacitor based on the local oscillator (LO) frequency. In this case our LO is 2.4 GHz and as such, Hittite technical support recommended using a 100 pF terminating capacitor [30]. The bias circuit schematic can be found in [Appendix D](#). [Figure 3.4](#) shows the finished board. In order to provide the bias signals to each of the Hittite boards, four such circuits are

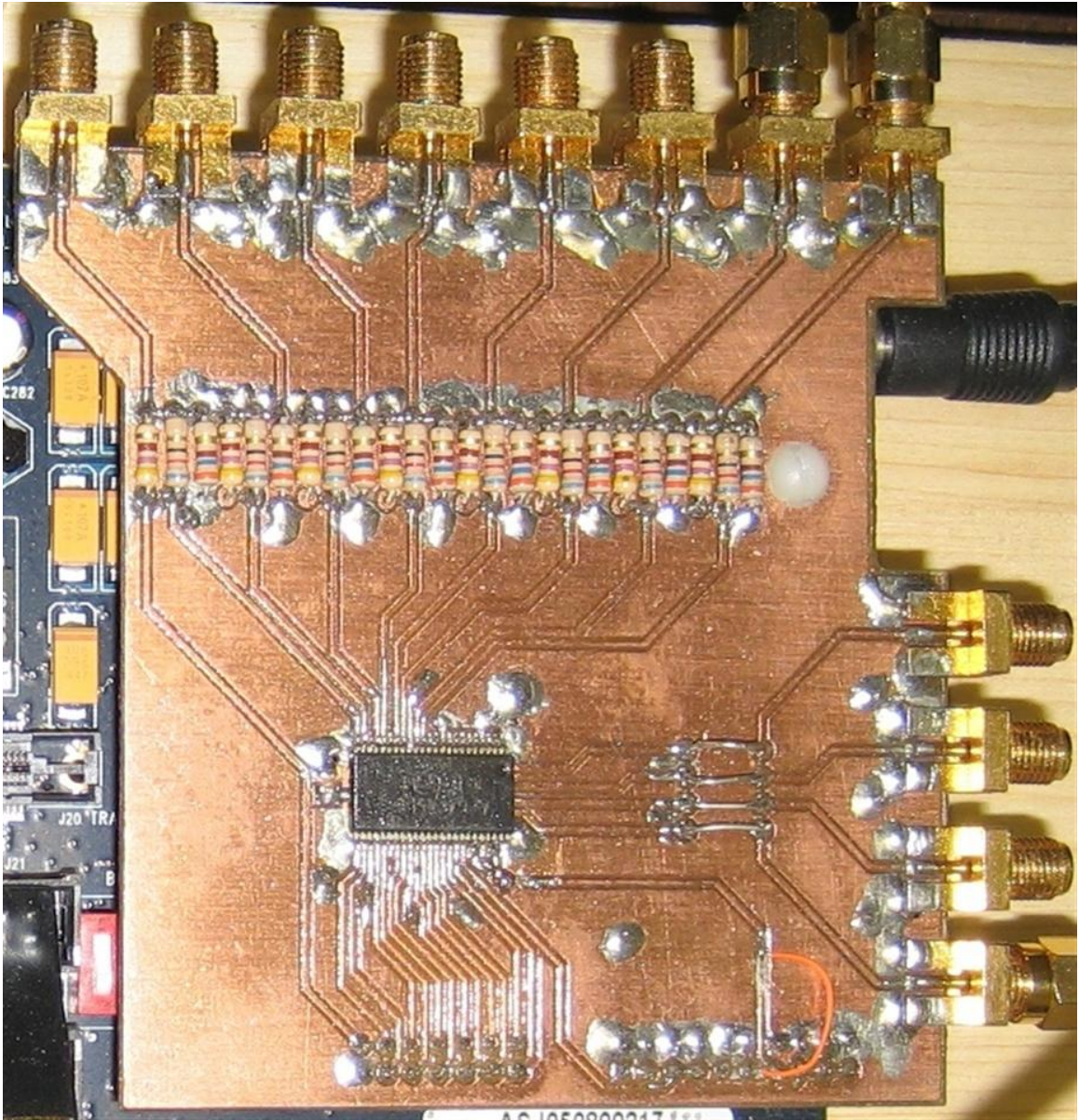


Figure 3.3: FPGA to Hittite Signal Conditioning PCB

needed in total. At the present time, only one has been completed, however they are relatively simple to replicate on a copper prototyping board. Note that the 2.4 GHz carrier signal will be sent to each of the Hittite modulator boards using coaxial cables. At this frequency there would be a great deal of attenuation of the signal. This was overcome by using a Mini – Circuits® ZX10-4A-27+ 4-way power divider and simply increasing the gain at the signal generator end.

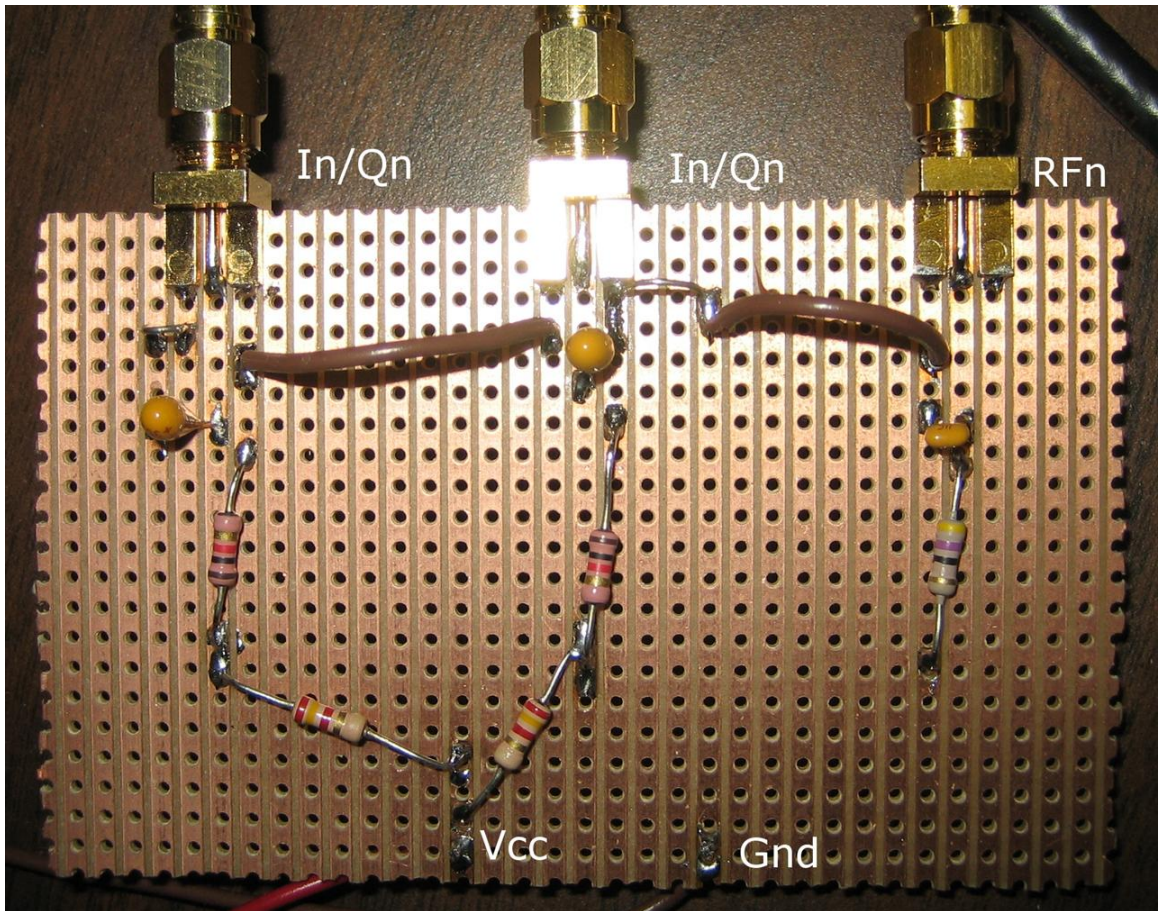


Figure 3.4: Hittite Bias Circuit Board

Testing of the Hittite board, after the addition of the bias circuit, confirmed that the problem was solved and that the inphase and quadrature signals were now being correctly modulated onto the carrier.

3.1.3 Data Rate Calculation

Recall that β represents the rolloff factor for a raised cosine response. From Equation 2.1, note that when β is one, the equation reduces down to that of a square wave [31], which is the case for this design. This leads to another important factor in communications systems – the excess bandwidth. The excess bandwidth is a measure of the amount of bandwidth, above the Nyquist bandwidth, which is being used to transmit the data. The amount of bandwidth used for transmitting data can be calculated as shown in Equation 3.1, where W_0 is the utilized bandwidth, and R_s is the symbol rate. The excess bandwidth is simply $\beta \times 100\%$ [31]

$$W_0 = (1 + \beta)R_s. \quad (3.1)$$

Given that $\beta=1$, for a square wave, and that W_0 cannot exceed 10 MHz, the symbol rate is found to be 5 MSPS. Using 4-QAM modulation, each symbol represents two bits, 2 bits/symbol, which corresponds to 10 Mbps at a symbol rate of 5 MSym/s, 5 Mbps on I and Q channels, and 100% excess bandwidth. In the ideal case where sinc functions were transmitted as data, β would be 0, which corresponds to 0% excess bandwidth. In this idealized case, the maximum throughput rate of the system would be 10 MSym/s, or 20 Mbps.

3.2 Receive Side

3.2.1 Clock Distribution

The implementation of the receive hardware is intricately related to the distribution and synchronization of clocks throughout the system. In order to simplify the signal recovery process, it was decided that a single clock source would be used to trigger the entire system – transmit, receive, and baseband processing subsystems.

To achieve this goal, the 10 MHz reference signal from the Aeroflex signal generator is divided amongst the FPGA boards and the four radios in the system. Note that the FPGA outputs clock signals, corresponding to the sample rate clock, to the ADC boards, using the four dedicated SMA outputs on the Hittite signal conditioning circuit, and thus providing user programmable sampling rates; see [Section 3.3](#) for details. The reference signal from the signal generator has a range of $\pm 1 V_{pp}$ into 50Ω [21]. By direct observation, it was found that this signal level was acceptable to drive the frequency standard inputs of the AOR radios sufficiently enough to allow the internal phase locked loop (PLL) to lock. However, in the case of the FPGA board, the clock input pin uses the LVTTLL standard which certainly cannot handle a negative input. As such, another custom PCB interface board was designed. In this case the design consists of a single chip, a TI TLV3501 high-speed comparator, which performs level translation on the incoming clock signal from the signal generator so that it can be input into the FPGA. This device basically compares the incoming signal to a reference signal, 0 V in this case, and outputs the value provided for logic high, 3.3 V in this case, if the incoming signal exceeds the reference, otherwise it outputs a logic low, 0 V in this case. For this application, the reference voltage was set to 0 V, the upper voltage reference was set to 3.3 V, and the lower reference was set to 0 V. Thus, when the $\pm 1 V_{pp}$ input clock signal exceeds 0 V, the TLV3501 outputs 3.3 V, and when the clock signal is less than 0 V, it outputs 0 V. This device was specifically chosen for its low rail-to-rail propagation delay of 4.5 ns which easily meets the needs of the 10 MHz clock signal with a corresponding period of $10 \mu s$ [32]. The comparator circuit schematic and board layout can be found in [Appendix D](#). [Figure 3.5](#) shows the finished clock signal level translator board.

As the number of radios connected to the output of the signal generator frequency standard increases, the signal becomes more and more attenuated, at a theoretical rate of 3 dB per split. As it turns out, after the FPGA and two radios are

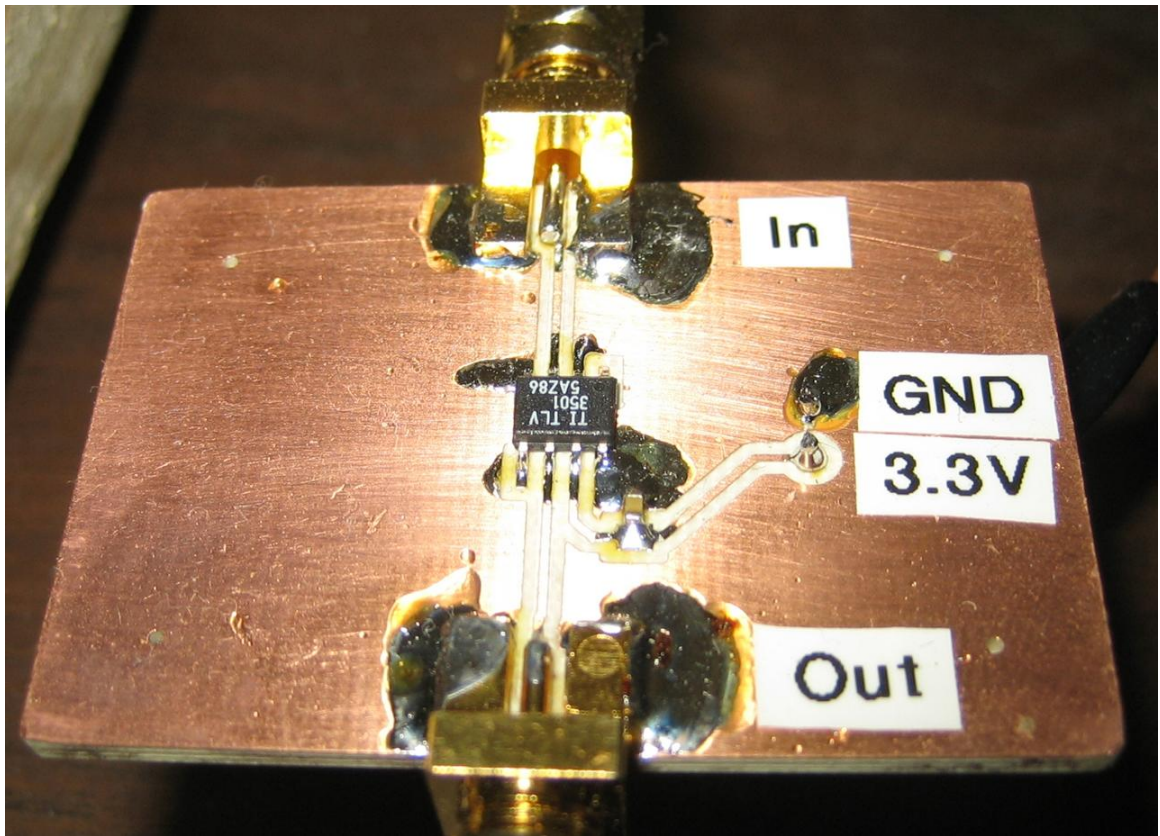


Figure 3.5: Clock Signal Level Translator PCB

connected, the signal attenuates to the point that the radios lose their PLL lock. To overcome this issue, a variable gain amplifier is required to boost the signal strength. This component has not been purchased yet, but the amplifier requirements have been evaluated by using a fixed gain amplifier which is prescaled by a variable attenuator in order to achieve the correct output gain. The attenuator is a Kay Electric model 30-0, and the amplifier is a Mini – Circuits® ZHL-6A with a typical fixed gain of 25 dB [33]. Table 3.2 lists the observed gains required as each additional radio was added to the system; the FPGA is always connected.

Table 3.2: Required Gain for Radio PLL Lock

Number of Radios	Attenuation (dB)	Gain (dB)	Effective Gain (dB)
1	25	25	0
2	20	25	5
3	17	25	8
4	15	25	10

It should be noted that the AOR radios do not have 50 Ω frequency standard inputs. Although not mentioned in the AOR documentation, they are actually high impedance which results in significant reflections when the cables are not terminated. As such, the above discussion assumes that the clock input to each radio passes through a 50 Ω feed-through terminator. For the purposes of testing this portion of the system, borrowed terminators were used. In the future, four such terminations will have to be purchased. Figure 3.6 illustrates the clock distribution scheme, external to the FPGA, for the system.

3.2.2 Sampling the Data

To meet the Nyquist criterion for sampling the incoming IF data, the sampling rate must be greater than 31.4 MHz. However, it is in our interest to have a high

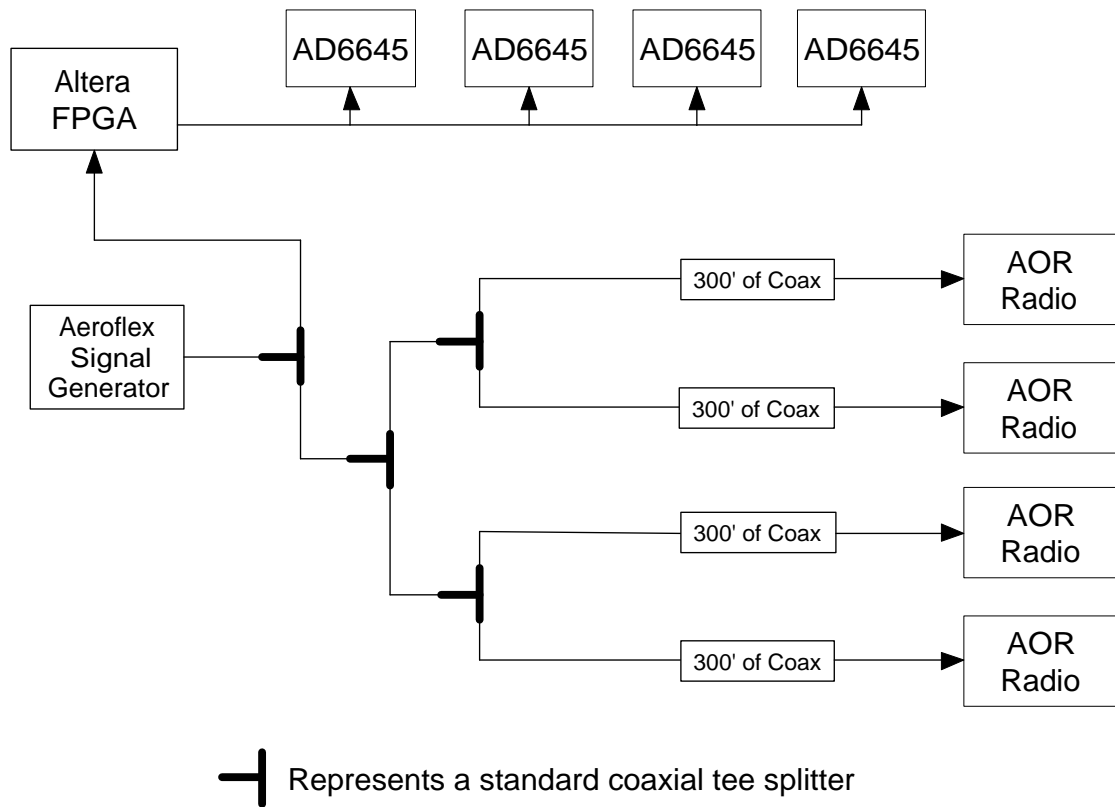


Figure 3.6: System Clock Interconnection Architecture

sampling rate in order to take advantage of processing gain and then to later decimate the signal. Based on this concept, a sampling rate of 80 MHz was selected during the system design. Note that this value is configurable and is not permanently fixed at 80 MHz.

Within the FPGA, the 10 MHz reference frequency is multiplied by eight using a PLL to generate an 80 MHz sample clock for the ADCs. The sampled data from each ADC is passed through a ribbon cable back to the FPGA – two ADCs per FPGA.

3.3 Baseband Processing

The baseband processing portion of the system comprises the two FPGAs, and four ADC boards. In order to make the system more portable and safe to use outdoors, the baseband components were mounted onto a wooden board using standoffs. [Figure 3.7](#) shows the mounted FPGA and ADC boards.

The ADC boards require +3.3 V, +5 V, -5 V, and GND power connections. The power for these boards is provided using a PC power supply which was modified for this design to operate as a standalone power supply. The power supply connects to the wooden mounting boards via the “banana connectors” located at the top of the board. The power connections are run as a bus, underneath the mounting board using a routed out groove, and then pass through holes to the top of the board where they connect to each of the ADC boards. The list of modifications made to the power supply can be found in [Appendix E](#).

The basic goal of this thesis was to build a testbed system. The baseband processing portion of this design implements the demodulation of the incoming data stream, but leaves all future additions up to the developer. That is, the system is responsible for generating the data and recovering it back on the receive side.

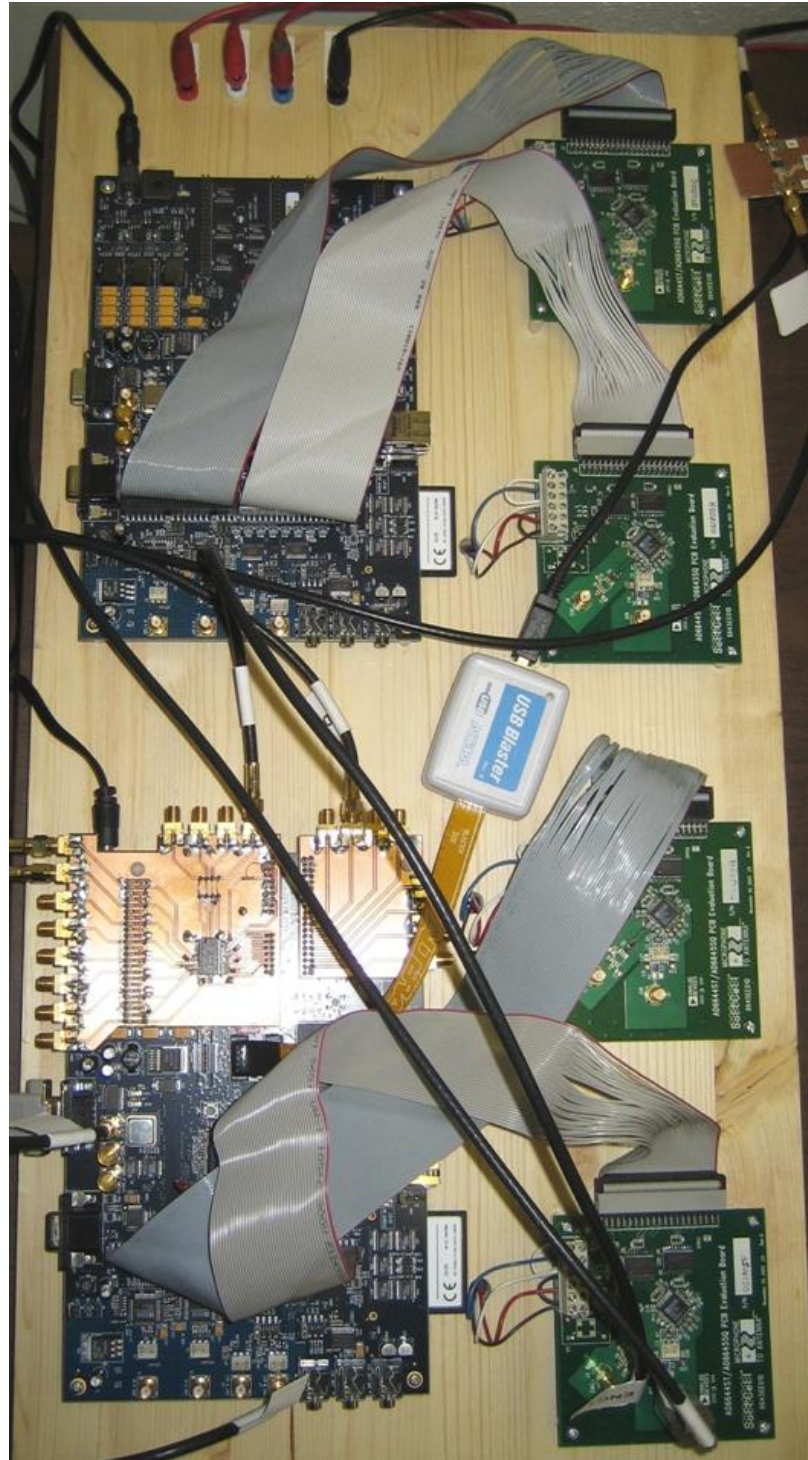


Figure 3.7: Mounted Baseband Processing Hardware

3.3.1 Data Recovery and Demodulation

The spectrum of the sampled data from the ADCs, shown in [Figure 3.8](#), shows that the recovered signal is very well contained within the passband and that there is no need to bandpass filter the data. From this figure, we can also see the effect of digitizing a data stream, in that the spectrum repeats at integer multiples of the sampling frequency. Hence the signal is situated at $0+10.7$ MHz and $80-10.7$ MHz, which is what one would expect from digitizing a 10.7 MHz signal at a sampling rate of 80 MHz. Note that frequency components at $0-10.7$ MHz and $80+10.7$ MHz are not shown.

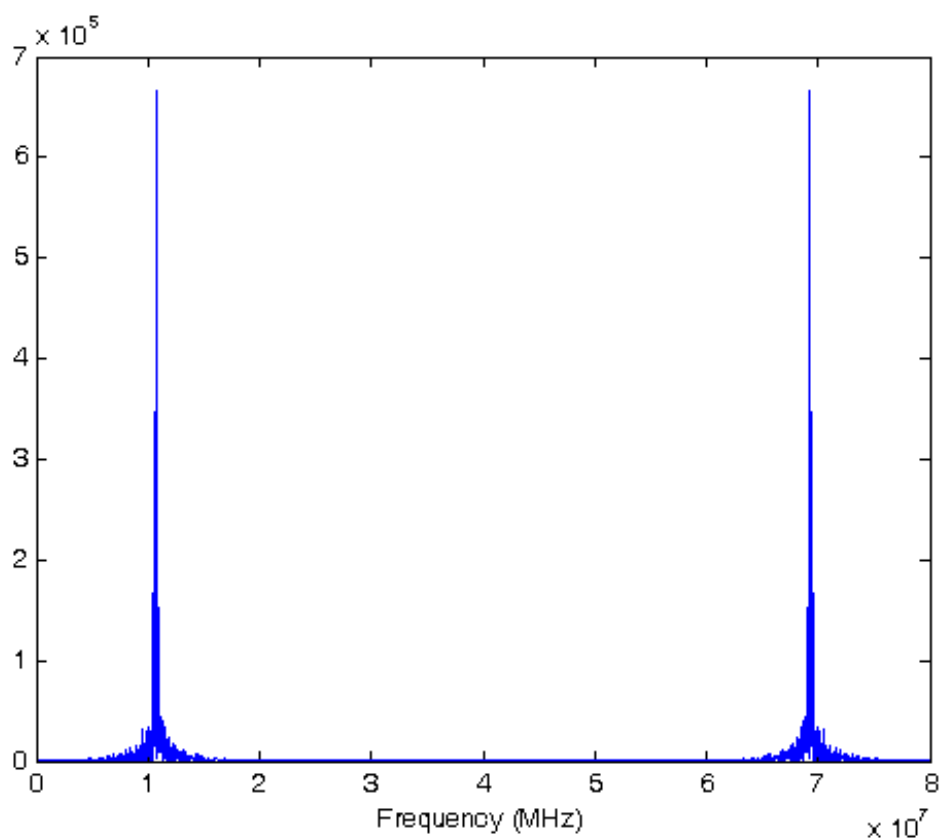


Figure 3.8: Spectrum of the Sampled IF Data

Instead, the data is passed directly on to the demodulator circuitry which attempts to undo the operations performed by the modulator in order to recover the

transmitted signal. To achieve this goal, the data stream is split into two nearly identical paths – one for the inphase portion of the signal and another for the quadrature portion. Then, the inphase stream is multiplied by a cosine wave which is matched to the carrier frequency of 10.7 MHz, while the quadrature data stream is multiplied by a sine wave of the same frequency. Based on the well known trigonometric identities,

$$\begin{aligned}\cos A \cos B &= \frac{1}{2} [\cos (A + B) + \cos (A - B)] \\ \sin A \sin B &= \frac{1}{2} [\sin (A + B) + \sin (A - B)],\end{aligned}$$

and noting that the frequencies A and B are identical between the modulator and the demodulator, we are left with the spectrum located at the sum and difference frequencies. In this case, the sum frequency is 21.4 MHz and the difference is 0 MHz, or baseband, as shown for the inphase channel in [Figure 3.9](#). The same situation applies for the quadrature data.

Now that the signals have been moved back to baseband, the double frequency component at 21.4 MHz must be removed. This is accomplished in the FPGA using a single stage digital down conversion architecture. First the signal is low pass filtered to remove the unwanted double frequency components from the spectrum. Then, the signal is downsampled, or decimated, by a factor of four in order to reduce the data rate, and thus the processing requirements for the remainder of the hardware. Note that in order to satisfy Nyquist’s criterion in the passband it is necessary to sample the data above 31.4 MHz. However, now that the signal has been returned to baseband, it only occupies 5 MHz bandwidth. By decimating the signal, the sampling frequency is effectively being changed from F_s to F_s/N , where N is the decimation factor. In this case, the new sampling frequency after decimation is 20 MHz, more than enough to fully represent the 5 MHz baseband bandwidth, while reducing the processing speed of the remainder of the hardware by 75%. However, by decimating

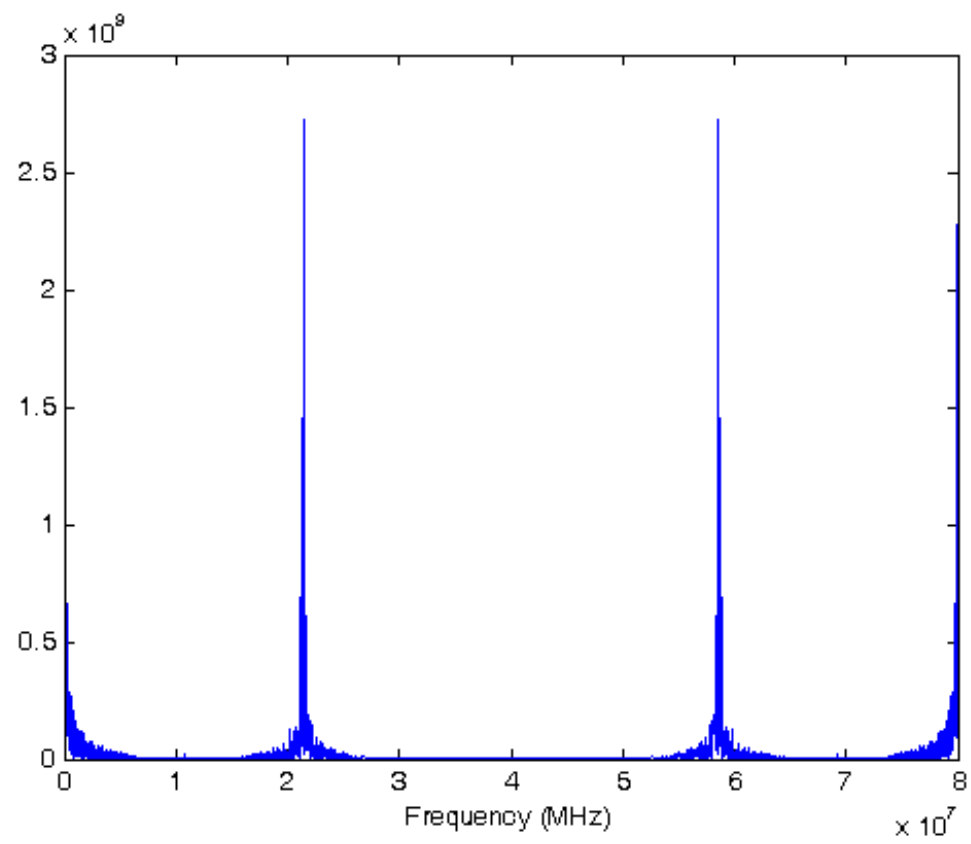


Figure 3.9: Spectrum of Mixed Inphase Data

the signal, the spectrum is essentially being compressed inward. As such, it is very important to ensure that there are no unwanted signals outside the baseband region before downsampling is performed – this is why we low pass filter first. Otherwise, the higher frequency components will end up folding back into the signal of interest [34]. Figure 3.10 depicts the spectrum of the inphase signal after low pass filtering and decimation by four. From this figure, the output SNR of the implemented demodulator circuit can be estimated to be approximately 38.33 dB.

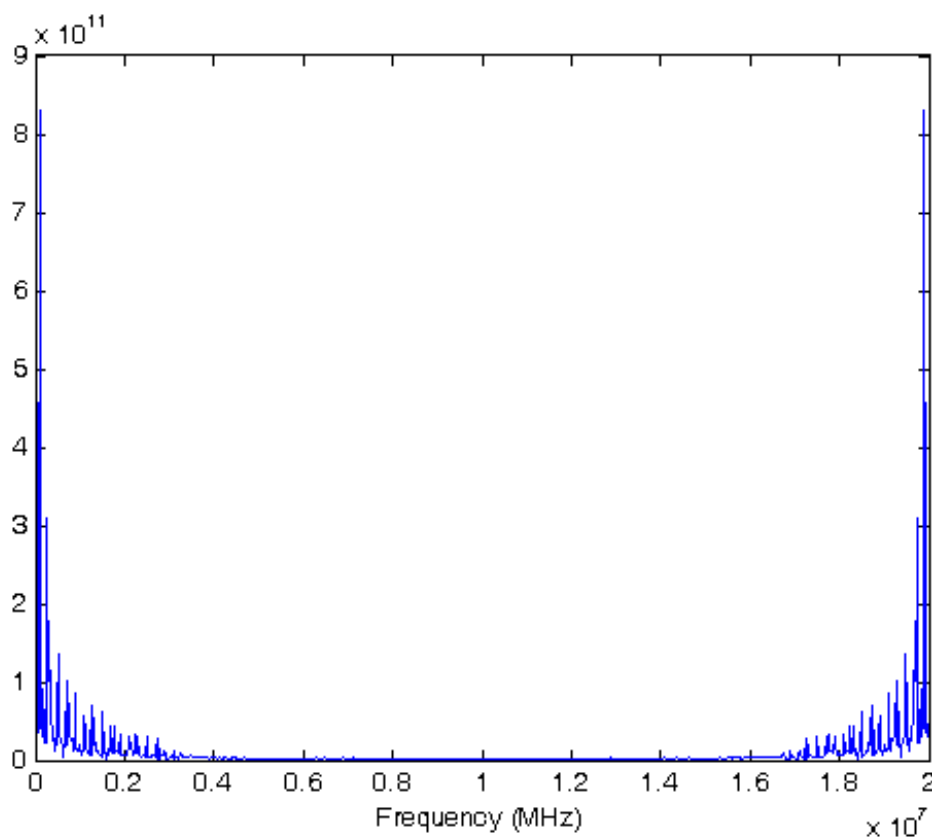


Figure 3.10: Spectrum of Digital Down Converted Inphase Signal

At this point, a simple linear decision making block is inserted in each of the data paths in order to decide whether the incoming bit is a one or a zero. Using the digitized inphase and quadrature streams the next step would be to add a symbol mapper which would return the inphase and quadrature streams back to raw

data. This however, was not done in this case because the data is only comprised of square waves. Figure 3.11 illustrates the demodulator design for this thesis. Note that this hardware has to be repeated once per incoming IF signal, a total of four times. Furthermore, it should be noted that the inphase and quadrature sides of the demodulator must be identical in order to preserve the phase relationship between the two data streams.

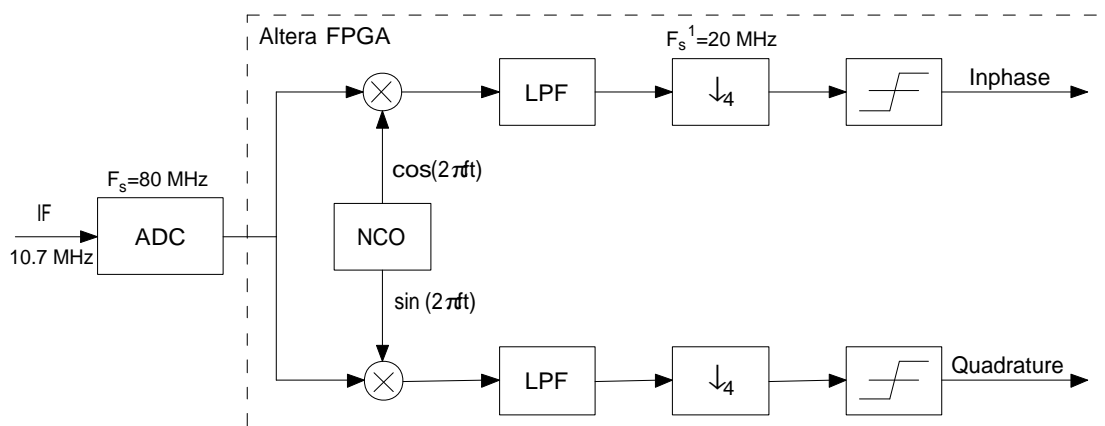


Figure 3.11: Implemented Demodulator Architecture

Unfortunately, the recovery of the content of the inphase and quadrature bit streams is very much reliant on matching the exact frequency and phase of the modulated data. Figure 3.12 shows the demodulated constellation output from the FPGA. Note that the final source code (Quartus® II project) for the hardware baseband implementation can be found on the enclosed CD-ROM along with several other versions which represent various stages of development and debugging.

From this output it can clearly be seen that there is a phase difference between the modulator and the demodulator which is causing the constellation to be rotated slightly. However, it also demonstrates that the system is completely frequency locked since the constellation is not rotating. To determine what the phase

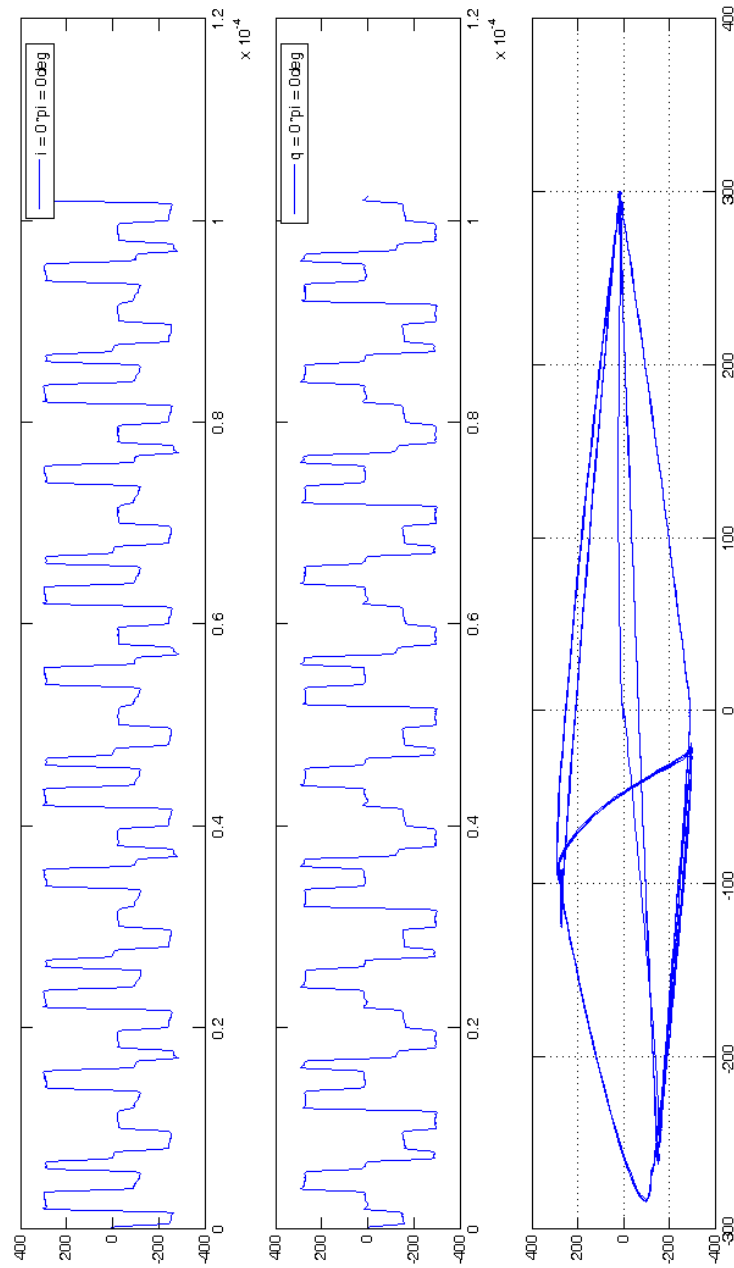


Figure 3.12: Demodulated I and Q Constellation

difference is between the modulator and demodulator, a MATLAB® script was written which gradually increments a phase offset onto the demodulator output signal; see [Appendix B](#) for code. Using this tool, the phase difference was found to be approximately 122.4 degrees. [Figure 3.13](#) shows the result of adding the 122.4 degree offset to the demodulator LO sinusoids. Note that the constellation is now upright, and the inphase and quadrature bit streams are now clearly visible – I = 100 kHz and Q = 250 kHz square waves.

3.3.2 Carrier Recovery

For the reason discussed above, most if not all real-world demodulators use a carrier recovery loop which is responsible for extracting the carrier frequency and phase, then using that information to drive the demodulator sinusoids. In so doing, the demodulator tries to get an exact copy of the modulator LO signal in order to achieve proper demodulation. A MATLAB® simulation of the QAM synchronizer circuit described in [\[6\]](#), shown below in [Figure 3.14](#), has been implemented (see [Appendix B](#) for code).

The output of the MATLAB® simulation of the synchronizer circuit is shown in [Figure 3.15](#). The square law devices are simply squaring multipliers, and the FIR filters were simulated using 256 taps and a 2 kHz passband bandwidth. Furthermore, for this system, $1/(2T) = 10.7$ MHz and $1/T = 21.4$ MHz. From the simulation results, one can clearly see the 10.7 MHz carrier signal component. Using this circuit to recover the modulator LO frequency and phase, the recovered sinusoid can then be used to demodulate the inphase and quadrature data streams.

Another possibility is to continue to use the NCO and simply calculate the phase error between its output and the recovered carrier. The resulting error could then be used to update the NCO phase modulation input. Note that a constant value to the phase increment input of the Altera® NCO megacore function produces

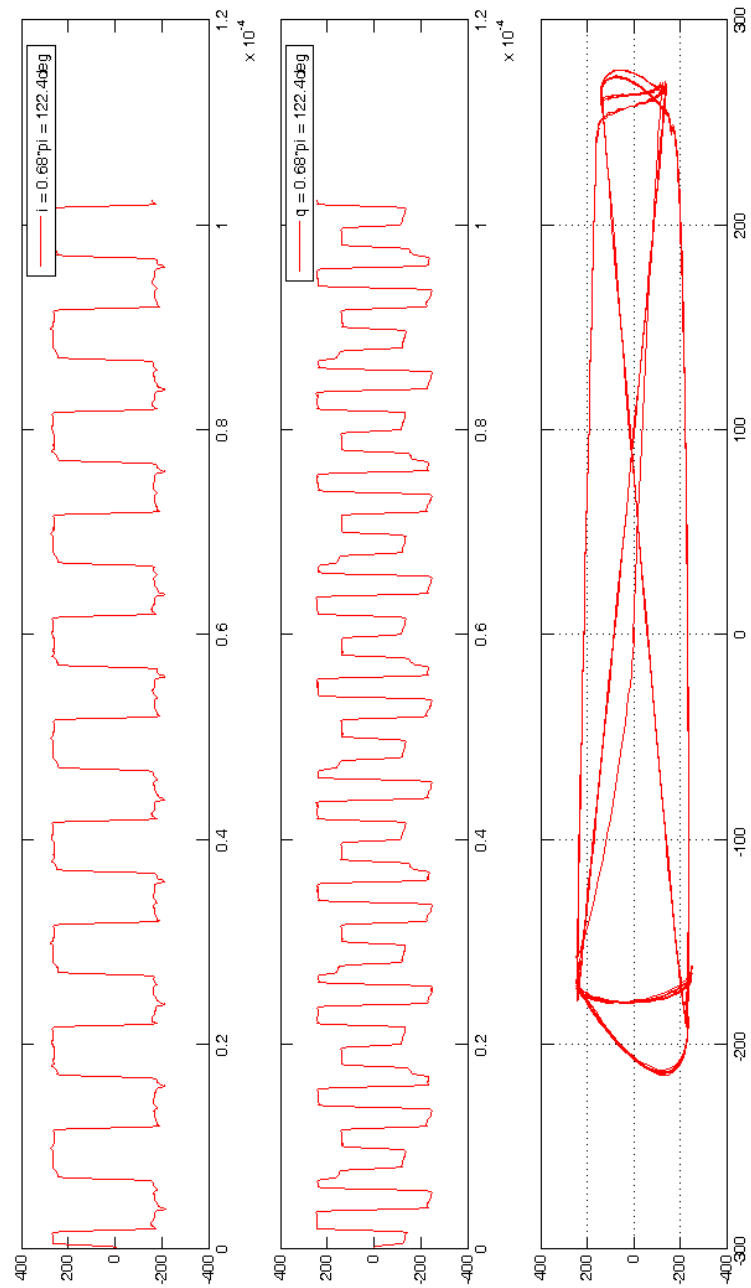


Figure 3.13: Demodulated I and Q Constellation with Phase Corrected

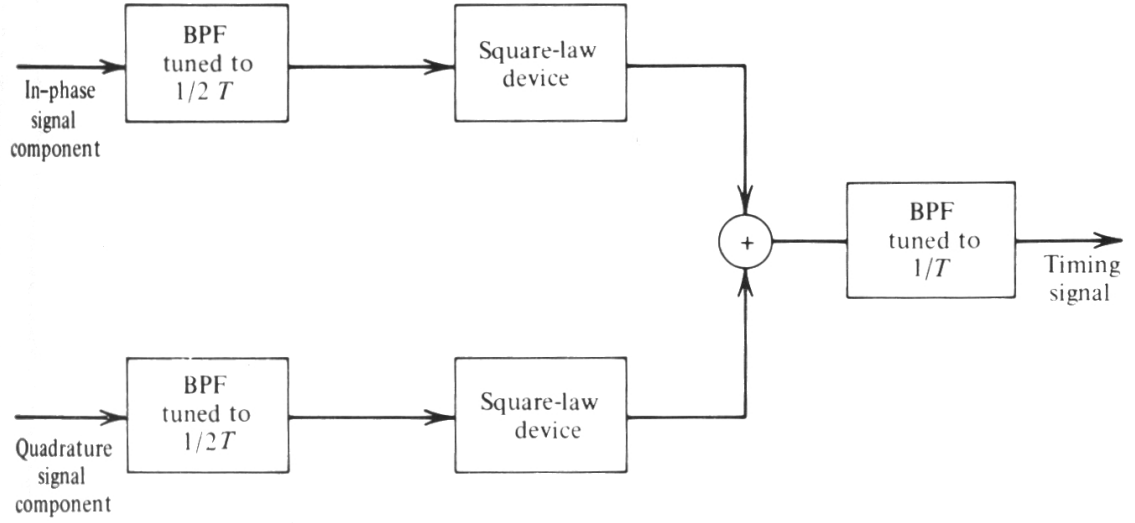


Figure 3.14: QAM Synchronizer Circuit [6]

a constant phase offset between 0 and 2π according to the relationship defined in Equation 3.2. As well, the phase increment input is unsigned binary whereas the cosine and sine outputs are signed binary. Note that none of these details regarding the NCO inputs and outputs are documented in the Altera® documentation; they were elicited through various communications with Altera® technical support staff [35]. The number of bits of resolution for the phase modulation input is N_{bits} , ϕ_{pm} is the desired phase offset in radians, and N is the required input to the phase modulation port in order to achieve this offset,

$$\phi_{pm} = \frac{2\pi N}{2^{N_{bits}}}. \quad (3.2)$$

3.3.3 LMS Adaptive Filtering

Aside from the obvious phase offset in the recovered signal, there is also some distortion and warping of the constellation. This is due to non-linearities in the system and the channel itself. In order to compensate for these non-linearities, the most common approach is to implement some form of equalizer circuit which effectively

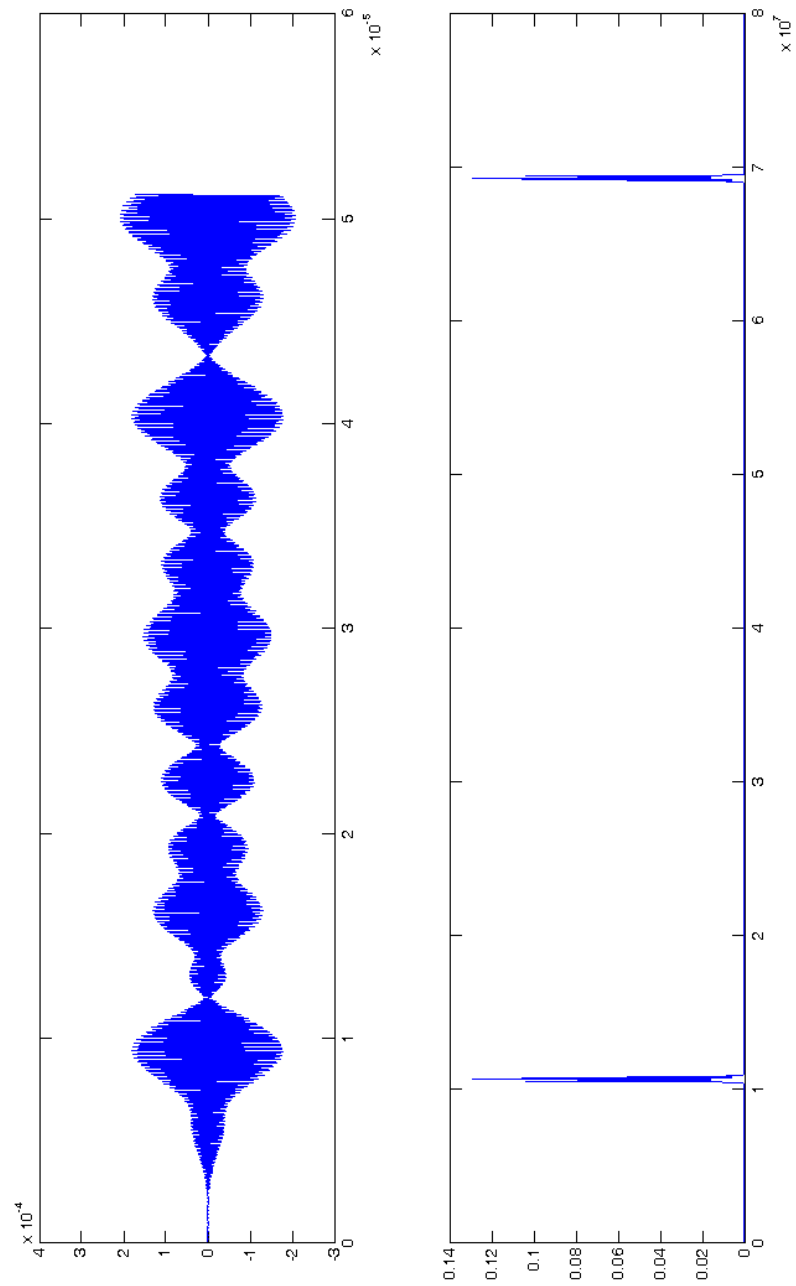


Figure 3.15: MATLAB® Simulation Results for Carrier Recovery Loop

tries to invert the channels' frequency response.

In the case of this system, a 4x4 array of LMS filters was designed not only to overcome radio channel irregularities but also to isolate and recover each individual user's transmitted signals. This is accomplished by first training the filters using a predetermined data stream for each user. As the data stream is transmitted, the error signal, e_n , is calculated based on the received signal and the known transmitted data. During this training process, the filter taps are updated based on the error and the adaptation constant, μ , as shown in Equations 3.3, 3.4, and 3.5 [36],

$$\underline{W}_{n+1} = \underline{W}_n + \mu e_n \underline{r}_n \quad (3.3)$$

$$e_n = d_n - \hat{d}_n \quad (3.4)$$

$$r_n = d_n + \eta_n. \quad (3.5)$$

Figure 3.16 shows the simulation output for the adaptive filter array. Based on this output, it is clear that the filter is stabilizing using the 1024-bit training sequence. The MATLAB® source code for the simulation can be found in Appendix B. This filter design will have to be implemented in the future in order to differentiate between the users in the system and minimize any non-linearities in the system and/or channel. However, this simulation is an excellent starting block for physically implementing the filtering design in hardware.

3.4 Centralized Control

The centralized control algorithm is mainly an application specific task. However, this design provides the underlying infrastructure in terms of cabling and interfacing all the RS-232 devices in the system.

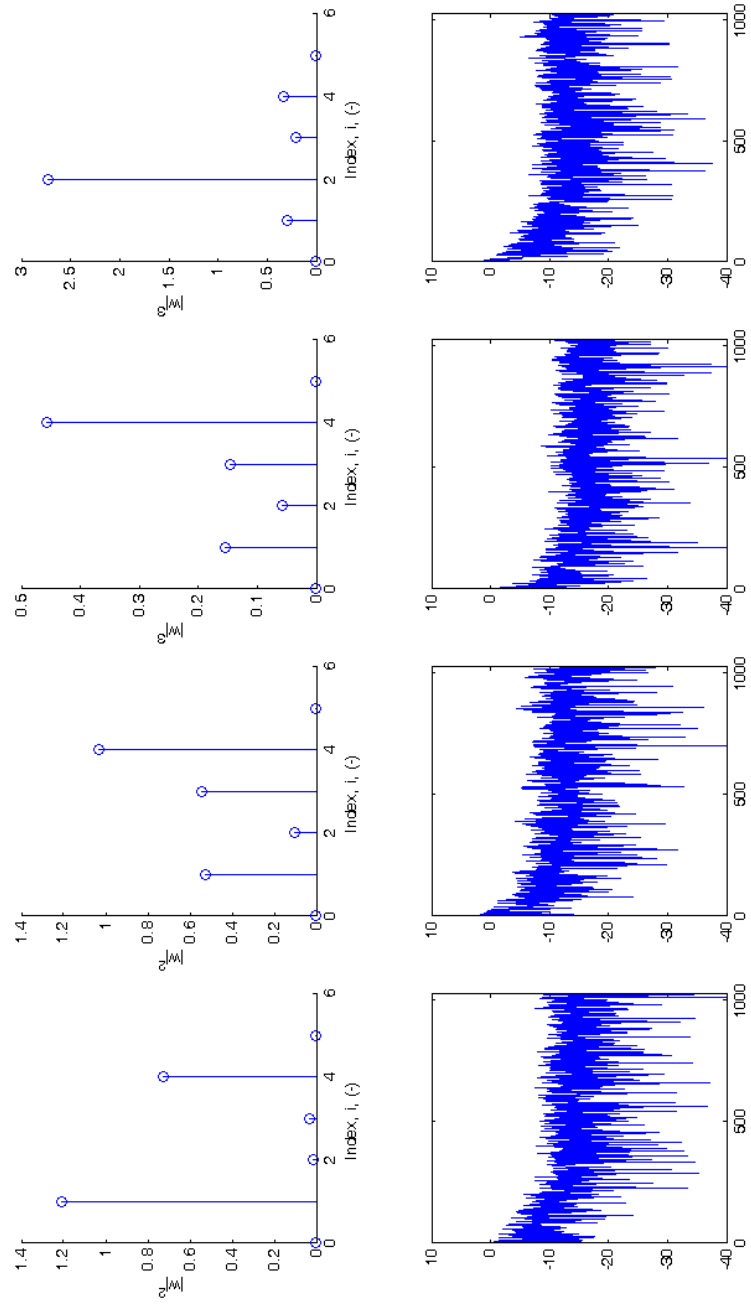


Figure 3.16: MATLAB® Simulation Results for LMS Adaptive Filter Array

3.4.1 Black Box[®] RS-232 Control

The Black Box[®] COS is responsible for interconnecting all the COTS devices in the system. The communications settings used for this system are imposed by the RS-232 configuration of the AOR radios, since they are only capable of selecting between data rates of 4800, 9600, and 19200 bps, and cannot change other parameters such as the number of data bits, number of stop bits, or the parity [4]. The Aeroflex signal generator is capable of changing all of the common RS-232 communication parameters in order to match those used by the AOR radios. [Table 3.3](#) lists the communications parameters which are required to be used for the system. This information does not appear in the AOR Radio manual, it can be found on the last page of the AR5000 RS-232C Command List manual.

Table 3.3: RS-232 Communication Parameters [3]

Parameter	Value
Data rate	9600 bps
Data bits	8
Stop bits	2
Parity	None
Flow Control	Xon

Based on these settings, the Black Box[®] COS was configured for each of the devices. The radios act as data terminal equipment (DTE), the signal generator acts as data communications equipment (DCE), and the FPGA has been selected to act as a DTE. The DIP switch settings which were established to internally configure the COS to meet the communication requirements of the system devices are listed in [Appendix A](#). The remainder of the details for configuring the system are specific to the test scenario being developed, and thus are left up to the developer. Note that all the RS-232 cables created for interconnecting the hardware are straight through cables, not null modems. Any switching between DTE and DCE is done using the

Black Box® COS.

3.5 System Configuration

A complete description of the cabling used in this design is provided in [Appendix A](#). A list of all cables created for this design as well as information on how the system should be interconnected for use in a 1x1 or 4x4 configuration can be found there. [Appendix A](#) also contains a section describing the proper configuration of the COTS hardware to achieve correct system operation.

Chapter 4

System Development and Testing

The design methodology for this thesis involved breaking the system down into small subsystems which could easily be debugged and verified. The primary simulation and debugging tool used for this thesis was MATLAB® version 7. Initially, the design for each of the subcomponents was written using MATLAB® scripts which were then subsequently simulated to verify the desired operation. Once the particular algorithm was well understood, it was implemented in the FPGA using an iterative approach. After the component was developed in Verilog and programmed into the FPGA, the Altera® SignalTap® software was used to record the system variables over a defined period of time (number of samples). Then, the recorded real-world signals were exported from SignalTap® into the MATLAB® workspace. This method allows the developer to examine the recorded variables off-line to try and determine what or where any problems might lie in the design. Furthermore, since the initial design was created in MATLAB®, the real-world variables can be compared to the simulated values to help the debugging process.

The author would absolutely recommend the use of MATLAB® for design, simulation, and debugging, coupled with SignalTap®. By first simulating the design, the developer gains a more in-depth understanding of what they are trying to implement

in hardware, which cuts down on the number of iterations required to achieve the desired goal. When dealing with FPGA designs, the time required for the software package to synthesize the hardware and generate a programmable bit file is orders of magnitude greater than the time required to compile a design in a software based environment. Thus, by using MATLAB®, a demodulator design simulation might take a couple of seconds to run, whereas when implementing the actual hardware on the FPGA, the equivalent design might take one half hour to build. Having already developed the initial design in MATLAB®, and therefore gained familiarity with it, the developer will now likely take far fewer compiles of the hardware to achieve the desired results than if they had not first simulated in MATLAB®. Since each hardware compile might take thirty minutes, for instance, a few less compiles constitutes a significant reduction in development time. Moreover, once a MATLAB® simulation exists, it acts as an excellent debugging tool for comparing the real-world results with the ideal case.

Aside from MATLAB® and SignalTap®, no other tools were used for debugging. Although the Altera® simulator tool is very useful for off-line, static types of verification, it was not suitable for dealing with the majority of the baseband processing design since the real-world signals are difficult to generate and the output difficult to verify within the simulator. That said, the Altera® simulator tool was useful for verification and debugging of some components such as the PN generators which do not require external inputs.

4.1 Component Testing and Integration

Before integrating any components into the system, they were first tested independently, starting with the FPGA board. Initially, a simple design was programmed into the FPGA to test some of the various switches and LEDS and verify that the

device could be programmed correctly using the JTAG programming cable and the Altera® Quartus® II development software. The next step was to test the ADCs and DACs on the board. This was accomplished by generating a test signal in the FPGA, outputting it via the DAC, and then feeding the output back into the ADC. The resulting signal was then examined using the SignalTap® software tool as well as an oscilloscope by outputting the sampled data via the second DAC. Both ADC and DAC channels were verified using this process. The use of both the oscilloscope and the SignalTap® software package demonstrated that these tools were in fact operating as expected. After the onboard ADCs were verified, the external Analog Devices ADCs were added to the design. Again, the ADCs were tested using a DAC to ADC loopback setup. The results were verified using the oscilloscope and SignalTap®.

The next step in the design process was to lock the FPGA internal circuitry to an external frequency standard. As such, the external 10 MHz frequency standard from the Aeroflex signal generator had to be passed into the FPGA external clock input connector. To achieve this task, a PCB board with a TI TLV3501 high-speed comparator chip was designed. This circuit performs level translation on the incoming clock signal, from the signal generator, in order to generate a 0-3.3 V square wave to input to the FPGA. This circuit was debugged and verified using an oscilloscope with the input impedance set to 50 Ω to match that of the FPGA clock input connector. Once the circuit was found to be operating as required, it was connected to the FPGA board clock input. Then, within the FPGA, a PLL was added to lock onto the external clock signal and generate a sample rate clock signal equal to eight times the input clock rate. The resulting 80 MHz was the used to drive the DAC to ADC loopback circuit, and the output was once again verified using both an oscilloscope and SignalTap®.

After the main ADC and DAC components were tested and integrated and frequency locked to the external 10 MHz frequency standard, that portion of the

design was set aside temporarily. The next step in development was to generate the signals for transmission. This was accomplished by first developing the LRS circuit within the FPGA as a standalone design unit. This unit was then tested using the Altera® simulator tool, which is part of the Quartus® software package, in order to verify that the output of the generator was in fact the correct PN sequence for the generating polynomial being used. Once the LRS circuit was debugged in software, it was then ready to be used to generate real world signals. The output of the eight LRS generators were connected to a GPIO pin, and the resulting digital waveform was observed and subsequently verified using an oscilloscope. To transmit the signals, the PN sequence outputs of the FPGA GPIO pins must be modulated onto the carrier via the Hittite modulator boards. Due to the difference in signal levels between the FPGA GPIO output pins and the expected input to the Hittite modulator board, a custom PCB board was required for this task. The board was first designed on paper, and then subsequently simulated using PSpice®. The board was then implemented, and tested using a signal generator to simulate the FPGA output along with an oscilloscope to observe the resulting input which would appear at the Hittite modulator input. The circuit board was found to provide the correct levels for the Hittite board, along with sufficient current to drive 300 feet of the coaxial cable used by the system. The signal conditioning PCB also accepted four clock inputs which it buffered in order to improve the current source/sink capabilities of the FPGA outputs. These pins are used for the sample rate clock outputs from the FPGA board to the Analog Devices ADC boards, and are not passed through the voltage translation network that the PN sequences destined for the Hittite modulator board do.

At this point, the development and testing returned to the DAC to ADC design. The signal conditioning board was mounted onto the FPGA board using an available GPIO header. The DAC to ADC design, which was previously verified,

was then modified to output the sample rate clock to the external ADC boards via the signal conditioning board. The resulting test data was recorded and analyzed in SignalTap® and observed on the oscilloscope. This verified that the clock buffer portion of the signal conditioning circuit was meeting the design requirements. The next step was to shift back to the signal generation side, and determine whether or not the signal conditioning circuit was functioning correctly for the PN sequences from the FPGA. At this point, the testing process became more difficult due to the high carrier frequency. That is, the output of the Hittite modulator board was at 2.4 GHz, and there was no equipment available in the lab which is capable of observing a signal at this frequency. Therefore, the output of the modulator board was fed into the AOR radio, tuned to the carrier frequency, and the resulting IF output was observed. To facilitate the debugging process, the sequences transmitted from the FPGA were changed to square waves of known frequency. Then, using the AOR SDU5600 spectrum display unit, the IF output of the radio could be seen, along with frequency components representative of the harmonics of the square wave.

At this point, the IF output of the radio was passed into one of the external ADC boards and recorded using SignalTap®. The recorded IF signal from the radio was imported into MATLAB® and demodulated offline using a MATLAB® script designed to recover the original signal. Based on the output of this test, it was found that the demodulated inphase and quadrature signals were only 45 degrees apart rather than 90 apart. Due to the fact that the radio operation was outside of our control, the Hittite modulator board was suspected of causing the problem. After some time spent examining the literature, and conversing with Hittite technical support, it was determined that the board was not correctly set up. To fix the problem, an additional PCB board was developed which provides the common mode voltage along with input capacitors described in [Chapter 3](#). The finished board was powered using a power supply, and output was verified using an oscilloscope. Using the

newly developed PCB board, driving the negative terminals of the Hittite modulator board, the test was repeated and the resulting sampled IF data was processed in MATLAB®. From the data collected, it was found that the PCB board driving the negative terminals of the Hittite board had solved the problem, and that the inphase and quadrature signals were now 90 degrees apart as expected. This test verified the operation of both the Hittite modulator board, and the AOR radios.

Now that the system was able to generate, transmit, and receive signals, the next stage in development was to process the received IF signal. The development of the demodulation hardware was performed in stages, as described in [Chapter 3](#). First, an NCO was implemented in the FPGA which generated a sine and cosine wave with frequency equal to the IF output of the radio. Then new design was tested independently using SignalTap® and the recorded sinusoids were observed in MATLAB®. Based on the FFT of the collected sinusoid data, it was clear that the signals were situated exactly at 10.7 MHz as desired. The NCO was then added to the overall system design, and a pair of multipliers were added to mix the incoming IF data with the sine and the cosine wave from the NCO. The output of the mixing operations was recorded using SignalTap® and analyzed using MATLAB®. The analysis found that the mixed signals were in fact situated at baseband and 21.4 MHz as predicted. The remaining task was to filter the double frequency component from the signal. To meet this need, a filter was designed using the Quartus® software, and added to the design after the mixers. Once again, the output was recorded using SignalTap® and analyzed using MATLAB®. The final analysis found that the double frequency component had indeed been removed and that the desired signal was situated at baseband. From this point onward, the analysis of the demodulated data was performed in MATLAB®.

4.2 Interfacing SignalTap[®] and MATLAB[®]

Unfortunately, at the present time, there does not exist a clean method through which data recorded using Altera's[®] SignalTap[®] hardware-in-the-loop tool can be exported into MATLAB[®]. As a workaround to the problem, a set of steps was used in order to transfer the recorded data from SignalTap[®] into the MATLAB[®] workspace. The first step is to record the data in SignalTap[®], and export it as a comma separated value (.csv) file. Then, the .csv file should be opened using Microsoft[®] Excel, and the lines of header information from the top of the file must be deleted, and the file saved as an Excel spreadsheet .xls file. In MATLAB[®], the data can be imported from the .xls file using the 'xlsread' command. Once the data has been loaded into MATLAB[®], it now exists as one large table variable where each column corresponds to a single bit of data. As such, the imported data must be broken down into the individual variables which it contains. This operation was performed using a script which acts as an import file. By examining the sequence and size of the variables recorded in SignalTap[®], or by examining the header portion of the .csv file, the column number for each bit of each variable can be determined. Using simple matrix manipulation commands, the group of bits which constitute each variable can be extracted from the imported data and converted from binary into signed and/or unsigned decimal using a converter function which was developed. The sequence of MATLAB[®] commands required, along with the converter functions and a sample import file are provided in [Appendix B](#). Note that MATLAB[®] has a built in 'csvread' function for reading in .csv files, but it does not appear to be compatible with the .csv files created by SignalTap[®].

4.3 Implementation Issues

This section very briefly mentions several issues which were encountered during development of the baseband system which could be of help to future developers using the testbed.

Firstly, it was discovered early on that the use of unsigned binary numbers leads to large DC components in the data. This is due to the inherent nature of the number system, and thus signed binary numbers which are naturally entered about zero are much better suited for working with signals within the FPGA – e.g. an 8-bit signed binary value ranges from -128 to 127, whereas an unsigned 8-bit binary value ranges from 0 to 255. The second issue, relates to the FPGA hardware itself. Altera® recommends that all unused pins be configured as tri-stated inputs [37]. However, the Quartus® II design software defaults to grounded outputs, which the author found to cause erratic programming behaviour. When programming the device, it often took three or four attempts before the programming file would be successfully written to the FPGA. This issue was resolved by ensuring that the unused pins in each new design were set to inputs tri-stated. Another issue encountered was related to the maximum clock speed for the Altera® multiplier megacore function. Although the author was unable to find a maximum frequency in the megacore documentation, it was discovered that if the clock speed is increased to more than 80 MHz, there is a substantial increase in noise at the output, which corresponds to a decrease in SNR. It is not clear whether this issue was due to an incorrect design implementation, either on my part or within the megacore itself, or an actual hardware limitation; Quartus® offered no warnings or errors to this effect. This issue was resolved by setting the multiplier clock rate to 80 MHz. The final issue occurs when creating a design with multiple processing stages. When implementing DSP designs in FPGAs, there is a strong tendency for the number of bits representing the data to rapidly, and potentially unnecessarily, increase. In order to reduce the required processing

in the FPGA, it is important to try and minimize the number of unnecessary bits which are carried through the design. It can be tedious to determine which bits are relevant, and which are not. In any case, MATLAB[®] was found to be a good tool for determining the range of the signal of interest and thus determining which bits can be eliminated.

Chapter 5

Summary and Future Work

5.1 Summary of Work Completed

This thesis provides the design and implementation of the physical layer hardware for a reconfigurable MIMO SDR based wireless testbed. The design incorporates the concept of SWAP gain as well as diversity gain, and is designed for use in investigating radio channel properties as well as the implementation of algorithms in a real-world environment. The design process along with the implementation details of each subcomponent is presented in detail.

A simple quadrature demodulator design was created within the FPGA and used to verify the correct operation of the system. A MATLAB® simulation of an adaptive LMS filter array for channel equalization and recovery of user signals was developed. As well, a carrier recovery loop was developed and simulated in MATLAB®. Both components exhibit the desired functionality and are ready for future implementation.

The overall testbed system design has been completed, the hardware has been either designed and built or selected and obtained, and the necessary interfacing hardware and software has been developed. The system is now ready for the re-

maining components to be added, after which it will be ready for use in taking field measurements for characterization of the radio channel.

5.2 Future Work

Due to the large size of this system, and the various stages of debugging that were required, some issues were discovered late which require additional hardware to be purchased or built before the system implementation is 100% complete. In terms of new hardware, the following items are required: eight power supplies, or long power cables, to power the remote transmitters and receivers, four $50\ \Omega$ feed-through terminators for the 10 MHz clock lines to the radios, and a variable amplifier or fixed amplifier with variable attenuator for driving the 10 MHz clock lines to the radios. The only components left to build are three more bias circuits for the Hittite modulators. These boards are very simple, and should be a direct reproduction of the existing board, thus no design is required.

Future FPGA development work includes implementing the LMS adaptive filter array as well as the carrier recovery circuit in hardware. As well, the implementation of an RS-232 transmitter would be required in order to provide RS-232 control of the system components. The RS-232 transmitter should be simple to create as a parallel load shift register operating at 9600 bps. From that point, the underlying testbed system would be complete, and a user could develop their own test modules on top of the design. Finally, before any real-world testing can be performed with the system radiating in the 2.4 GHz ISM band, there is a need to investigate and meet any requirements or limitations imposed by RSS-210 [14].

Bibliography

- [1] I. Bolsens, “Keynote address: Programming modern fpgas,” *Multiprocessor System on Chip*, Estes Park, Colorado, USA, August 2006.
- [2] R. Church, “Tables of irreducible polynomials for the first four prime moduli,” *The Annals of Mathematics*, vol. 36, pp. 198–209, 1935.
- [3] AOR, *AR5000 RS-232C Command List manual*. AOR, Ltd., Tokyo, Japan.
- [4] AOR, *AOR AR5000 Operating Manual*. AOR, Ltd., Tokyo, Japan.
- [5] Altera, *Stratix II EP2S180 DSP Development Board Reference Manual*. Altera Corporation, San Jose, CA, USA, Aug 2005.
- [6] J. G. Proakis, *Digital Communications*. New York, NY, USA: McGraw-Hill Book Company, 1983.
- [7] W. Zhu, D. Browne, and M. Fitz, “An open access wideband multi-antenna wireless testbed with remote control capability,” *Proceedings of the First International Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities*, pp. 72–81, Feb 23-25 2005.
- [8] I. E. Telatar, “Capacity of multi-antenna gaussian channels,” internal tech. memo, AT&T Bell Laboratories, June 1995.
- [9] J. W. Wallace, B. D. Jeffs, and M. A. Jensen, “A real-time multiple antenna element testbed for MIMO algorithm development and assessment,” *IEEE Pro-*

- ceedings of the Antennas and Propagation Society International Symposium*, vol. 2, pp. 1716–1719, June 20-25 2004.
- [10] Z. Yong and F. Zhenghe, “Smart antenna testbed for ds-cdma systems,” *Canadian Conference on Electrical and Computer Engineering*, vol. 2, pp. 1327–1332, May 13-16 2001.
 - [11] D. Agarwal, C. R. Anderson, and P. M. Athanas, “An 8 ghz ultra wideband transceiver prototyping testbed,” *IEEE Proceedings of the 16th International Workshop on Rapid System Prototyping*, pp. 121–127, 2005.
 - [12] G. Zhu, B. R. Petersen, and B. G. Colpitts, “Signalling wavelength in an antenna array for space-time wireless over LOS channels,” *IEEE Proceedings of the 3rd Annual Communications Networks Services and Research Conference (CNSR 2005)*, vol. 1, pp. 69–73, 2005.
 - [13] H. Yanikomeroglu and E. S. Sousa, “Antenna gain against interference in cdma macrodiversity systems,” *IEEE Transactions on Communications*, vol. 50, pp. 1356–1371, Aug 2002.
 - [14] Industry Canada, *Radio Standards Specification (RSS) 210*. Industry Canada, Sept 2005.
 - [15] Hittite, *HMC497LP4 Direct Quadrature Modulator Datasheet*. Hittite Microwave Corporation, Chelmsford, MA, USA.
 - [16] LyrTech, *VHS-ADC Datasheet*. LyrTech Signal Processing, Quebec City, QC, Canada, July 2005.
 - [17] LyrTech, *VHS-DAC Datasheet*. LyrTech Signal Processing, Quebec City, Quebec, Canada, July 2005.

- [18] LyrTech, *SignalMaster Datasheet*. LyrTech Signal Processing, Quebec City, Quebec, Canada, May 2005.
- [19] ICS, “Performance of 14-bit data converters for software radio applications,” Application Note AN-SR-2, Interactive Circuits & Systems Ltd., Ottawa, ON, Canada.
- [20] K. Gentile, “The care and feeding of digital pulse-shaping filters,” *RF Design Magazine*, pp. 50–61, 2002.
- [21] Aeroflex, *AM/FM Signal Generator 2025 Operating Manual*. Aeroflex International Ltd., UK, issue 10 ed., June 2005.
- [22] Maxim, “Defining and testing dynamic parameters in high-speed ADCs, part 1,” application note 728, Maxim Integrated Products, Sunnyvale, CA, USA, 2005.
- [23] Maxim, “Filter basics: Anti-aliasing,” application note 928, Maxim Integrated Products, Sunnyvale, CA, USA, 2002.
- [24] Antenna Factory, *F02400-8 Datasheet*. Antenna Factory, Inc., Schaumburg, Illinois, USA.
- [25] Antenna Factory, *F01710-8 Datasheet*. Antenna Factory, Inc., Schaumburg, Illinois, USA.
- [26] A. Paulraj, R. Nabar, and D. Gore, *Introduction to Space-Time Wireless Communications*. Cambridge, UK: Cambridge University Press, 2003.
- [27] Black Box, *Code Operated Switch 8-Port (COS-8P)*. Black Box Network Services, Lawrence, PA, USA, June 2000.
- [28] R. Tervo, “EE4253 website and course notes fall 2003,” tech. rep., University of New Brunswick, Fredericton, NB, Canada, 2003.

- [29] Texas Instruments, *SN74ALB16244 16-Bit Buffer/Driver With 3-State Outputs Datasheet*. Texas Instruments Inc., Dallas, Tx, USA, Jan 2001.
- [30] Hittite Technical Support (private communication), 2006.
- [31] C. Langton, “Intersymbol interference and raised cosine filtering,” tutorial, Loral Space Systems, Available Online: www.complextoreal.com, 2002.
- [32] Texas Instruments, *TLV3501 High-Speed Comparator Datasheet*. Texas Instruments Inc., Dallas, Tx, USA, March 2005.
- [33] Mini-Circuits, *Plug-in & Coaxial Broadband, Linear Amplifiers Datasheet*. Mini-Circuits, Brooklyn, NY, USA.
- [34] R. Mutagi, “Understanding the sampling process,” *RF Design Magazine*, pp. 38–48, 2004.
- [35] Altera Technical Support (private communication), 2006.
- [36] S. Haykin, *Adaptive Filter Theory*. Upper Saddle River, NJ, USA: Prentice Hall, fourth edition ed., 2002.
- [37] Altera, *DSP Development Kit, Stratix II Professional Edition Getting Started User Guide*. Altera Corporation, San Jose, CA, USA, Aug 2005.
- [38] Belden, *Belden 9201 Coax - RG-58/U Type Datasheet*. Belden Wire & Cable Company, Richmond, IN, USA, July 2005.
- [39] D. K. Cheng, *Field and Wave Electromagnetics*. Reading, MA, USA: Addison-Wesley Publishing Company, second edition ed., 1992.

Appendix A

System Specifications

A.1 Altera[®] Pinouts to AD6645 ADC Boards

The pinouts provided in both the Altera[®] and Analog Devices documentation for the external AD6645 ADC connections to Altera[®] FPGA interface connector are not correct. Tables [A.1](#) and [A.2](#) provide the correct interconnection information.

A.2 Cabling and System Interconnections

Two types of cabling were used for the development of this system, one coaxial and the other RS-232 cable. Due to the large antenna separations in this system, the choice of cabling was made in order to minimize attenuation. The RS-232 cabling chosen was Belden model 1422A, a stranded five pair low capacitance computer cable for RS-232. This cable, coupled with a low data rate of 9600 baud, was found to support transmission of RS-232 signals at distances of at least 300 feet. The coaxial cable chosen was Belden model 9201, an RG-58/U type cable with a nominal attenuation of 1.1 dB per 100 feet at a frequency of 10 MHz [\[38\]](#). A slight trade-off was made with the choice of the coaxial cable - the characteristic impedance of Belden 9201 is 51.5 Ω . In an effort to reduce cost, more than \$1000, a similar cable with a charac-

Table A.1: ADC Board to Altera® FPGA Board Connector J5 Pinout [2]

AD6645 ADC Board		Stratix® II FPGA Development Board			
Pin Number	Signal	Connector	Pin Number	Signal	Stratix® II Pin
33	B13	J5	33	adi_D15	P2
31	B12	J5	31	adi_D14	P1
29	B11	J5	29	adi_D13	R3
27	B10	J5	27	adi_D12	R2
25	B9	J5	25	adi_D11	M2
23	B8	J5	23	adi_D10	M1
21	B7	J5	21	adi_D9	N3
19	B6	J5	19	adi_D8	N2
17	B5	J5	17	adi_D7	L2
15	B4	J5	15	adi_D6	L1
13	B3	J5	13	adi_D5	M4
11	B2	J5	11	adi_D4	M3
9	B1	J5	9	adi_D3	N5
7	B0	J5	7	adi_D2	N4
5	GND	J5	5	adi_D1	L4
3	GND	J5	3	adi_D0	L3

Table A.2: ADC Board to Altera® FPGA Board Connector J6 Pinout [2]

AD6645 ADC Board		Stratix® II FPGA Development Board			
Pin Number	Signal	Connector	Pin Number	Signal	Stratix® II Pin
33	B13	J6	33	adi_D32	M8
31	B12	J6	31	adi_D31	M11
29	B11	J6	29	adi_D30	M10
27	B10	J6	27	adi_D29	L6
25	B9	J6	25	adi_D28	L5
23	B8	J6	23	adi_D27	K7
21	B7	J6	21	adi_D26	K6
19	B6	J6	19	adi_D25	L8
17	B5	J6	17	adi_D24	L7
15	B4	J6	15	adi_D23	L10
13	B3	J6	13	adi_D22	L9
11	B2	J6	11	adi_D21	K9
9	B1	J6	9	adi_D20	K8
7	B0	J6	7	adi_D19	J9
5	GND	J6	5	adi_D18	J8
3	GND	J6	3	adi_D17	J7

teristic impedance of 50 Ω was overlooked. In exchange for the reduction in cost, the voltage reflection coefficient, Γ becomes non-zero as shown in [Equation A.1](#),

$$\Gamma = \frac{Z_L - Z_0}{Z_L + Z_0} = \frac{50 - 51.5}{50 + 51.5} = -0.0148 \quad [39]. \quad (\text{A.1})$$

Using the reflection coefficient, the voltage standing-wave ratio (VSWR) can be calculated as,

$$\text{VSWR} = \frac{1 + |\Gamma|}{1 - |\Gamma|} = \frac{1 + 0.0148}{1 - 0.0148} = 1.03 \quad [39]. \quad (\text{A.2})$$

From this result it is clear that the cable to load impedance mismatch is negligible in this case.

To further minimize cost, the cabling was purchased in spools of 500 and 1000 feet. The cables were then cut to length and connectorized using SMA, BNC, and N-Type connectors as required in order to avoid having to purchase converters as well as to avoid any additional attenuation in the converters. After each cable was made, it was tested for both continuity and length using a time domain reflectometer. A similar procedure was performed for the RS-232 cables, where each cable was provided either a DB9 or DB25 connector as necessary, and tested using Hyperterminal. [Table A.3](#) provides a cable inventory of all cables created for the testbed. Note that the cable label column corresponds to the labels attached to the cable.

In the future, in order to connect the three additional Hittite bias PCB boards, nine more three foot coaxial cables will have to be made. As mentioned previously, the RS-232 cables are all wired straight through. [Table A.4](#) details the wiring of the RS-232 cables.

Table A.3: Testbed Cable Inventory

Cable Label	Cable Type	Qty	Length (ft)	Connectors	Purpose
IQ1-IQ8	Coaxial	8	300	SMA, SMA	I and Q signals from FPGA (PCB) to Hittite
RF2-5	Coaxial	4	300	SMA, SMA	Sig. Gen. to Hittite
IF1-4	Coaxial	4	300	SMA, BNC	AOR IF to ADC
STD4-7	Coaxial	4	300	BNC, BNC	10 MHz std. to AOR
TX1-4	Coaxial	4	8	SMA, N-Type	Hittite to T_x ant.
RX1-4	Coaxial	4	8	N-Type, N-Type	R_x ant. to radio
STD2	Coaxial	1	3	SMA, BNC	10 MHz std. to clk PCB
STD3	Coaxial	1	2	SMA, SMA	Clk PCB to FPGA
STD1	Coaxial	1	2	BNC, BNC	10 MHz std. to BNC splitter
RF1	Coaxial	1	2	N-Type, SMA	Aeroflex to SMA power divider
ENC1-4	Coaxial	4	2	SMA, SMA	FPGA (PCB) Encode to ADC
GP1	Coaxial	1	3	SMA, BNC	General purpose
GP2-5	Coaxial	4	4	SMA, SMA	General purpose
GP6-8	Coaxial	3	4	BNC, BNC	General purpose
HIT1-3	Coaxial	3	2	SMA, SMA	Hittite bias PCB to Hittite
CON1	RS-232	1	10	DB9 M, DB25 M	COS to FPGA
CON2	RS-232	1	10	DB9 F, DB25 M	COS to Aeroflex
CON3-6	RS-232	4	300	DB9 M, DB25 M	COS to AOR
Total number of cables: 53					

Table A.4: RS-232 Connector Wiring

Wire Colour	DB9 Connector	DB25 Connector	Signal
Blue	1	8	DCD
White/Orange	2	3	R _x
Orange	3	2	T _x
White/Blue	4	20	DTR
Green	5	7	GND
White/Brown	6	6	DSR
Green/White	7	4	RTS
Brown	8	5	CTS
Grey	9	22	RI
White/Grey	NC	NC	-

A.3 Black Box[®] COS Configuration

The DIP switch positions required for the RS-232 configuration used in this design are as provided in [Table A.5](#), where C denotes a closed switch, and O denotes an open switch. Note that each port of the switch must operate in the opposite mode of whatever device is connected. For example, since the FPGA acts as a DTE, the port on the COS to which it is connected must act as a DCE. Detailed information regarding the meaning of these settings can be found in the device manual, see [\[27\]](#).

A.4 System Test Setup

The design and development of this system was performed with the following settings for the radios and signal generator. The signal generator frequency was set to 2.45 GHz, approximately the middle of the ISM band, with a gain of 0 dBm and all modulation disabled. The radios were also tuned to 2.45 GHz, with the AGC enabled.

Table A.5: COS DIP Switch Positions

Switch	Device	COS Port Config.	Value
SW1	FPGA	DCE	A: 00000CCO B: COOCCCCC
SW2	Signal Gen.	DTE	A: 0000COOC B: OCCOCCCC
SW3	Radio	DCE	A: 00000CCO B: COOCCCCC
SW4	Radio	DCE	A: 00000CCO B: COOCCCCC
SW5	Radio	DCE	A: 00000CCO B: COOCCCCC
SW6	COS Internal Setting	N/A	OCCOOOCO
SW7	COS Internal Setting	N/A	COOCCOOO
SW8	COS Internal Setting	N/A	CCOCCCCC
SW9	COS Internal Setting	N/A	COOO
SW10	COS Internal Setting	N/A	OOOOCOOO
SW11	Radio	DCE	A: 00000CCO B: COOCCCCC
SW12	Unused	N/A	A: XXXXXXXX B: XXXXXXXX
SW13	Unused	N/A	A: XXXXXXXX B: XXXXXXXX
SW14	Unused	N/A	A: XXXXXXXX B: XXXXXXXX

Appendix B

MATLAB[®] Simulation and Debugging Source Code

B.1 LMS Adaptive Filter Array

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  
% JAH_LMS This implements an 4x4 array of LMS filters of the specified order.  
%  
% Author: J. Andy Harriman    SN: 3020639    Course: EE 6997  
% Date: Feb 27, 2006  
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  
  
clear;  
clear functions;  
clf;  
  
numBits = 2^10; % The number of bits to transmit  
filterOrder = 6; % The order of the filters  
variance = 10^(-3); % The variance of the noise  
stdDev = sqrt(variance); % The standard deviation of the noise  
delay = 1; % decoding delay  
sampling_phase = -4; % fractional phase at the sampler  
mu = 2^(-6); % LMS adaptation constant  
k = 4; %upsampling factor  
  
dn_upsampled1 = zeros(1,numBits*k);
```

```

dn_upsampled2 = zeros(1,numBits*k);
dn_upsampled3 = zeros(1,numBits*k);
dn_upsampled4 = zeros(1,numBits*k);

% Generate the random data to transmit
% Note that the data transmitted is a uniformly distributed discrete RV
dn1 = 2*(rand(1,numBits)<0.5)-1;
dn2 = 2*(rand(1,numBits)<0.5)-1;
dn3 = 2*(rand(1,numBits)<0.5)-1;
dn4 = 2*(rand(1,numBits)<0.5)-1;

%upsample the data to be transmitted by k
dn_upsampled1(1) = dn1(1);
for i = 1 : numBits-1,
    dn_upsampled1((i*k)+1) = dn1(i+1);
end
dn_upsampled2(1) = dn2(1);
for i = 1 : numBits-1,
    dn_upsampled2((i*k)+1) = dn2(i+1);
end
dn_upsampled3(1) = dn3(1);
for i = 1 : numBits-1,
    dn_upsampled3((i*k)+1) = dn3(i+1);
end
dn_upsampled4(1) = dn4(1);
for i = 1 : numBits-1,
    dn_upsampled4((i*k)+1) = dn4(i+1);
end

% generate random 4-tap fir filters the 16 radio channels
channel11 = 2*rand(1,4)-1;
channel12 = 2*rand(1,4)-1;
channel13 = 2*rand(1,4)-1;
channel14 = 2*rand(1,4)-1;
channel21 = 2*rand(1,4)-1;
channel22 = 2*rand(1,4)-1;
channel23 = 2*rand(1,4)-1;
channel24 = 2*rand(1,4)-1;
channel31 = 2*rand(1,4)-1;
channel32 = 2*rand(1,4)-1;
channel33 = 2*rand(1,4)-1;

```

```

channel34 = 2*rand(1,4)-1;
channel41 = 2*rand(1,4)-1;
channel42 = 2*rand(1,4)-1;
channel43 = 2*rand(1,4)-1;
channel44 = 2*rand(1,4)-1;

bp_test_flag_1 = 1
if ( bp_test_flag_1 == 1 ) ,
    channel11 = 2*rand(1,4)-1;
    channel12 = 0*rand(1,4)-0;
    channel13 = 0*rand(1,4)-0;
    channel14 = 0*rand(1,4)-0;
    channel21 = 0*rand(1,4)-0;
    channel22 = 2*rand(1,4)-1;
    channel23 = 0*rand(1,4)-0;
    channel24 = 0*rand(1,4)-0;
    channel31 = 0*rand(1,4)-0;
    channel32 = 0*rand(1,4)-0;
    channel33 = 2*rand(1,4)-1;
    channel34 = 0*rand(1,4)-0;
    channel41 = 0*rand(1,4)-0;
    channel42 = 0*rand(1,4)-0;
    channel43 = 0*rand(1,4)-0;
    channel44 = 2*rand(1,4)-1;
end

%Pass the data generated at each transmitter through each of its associated
%paths to the receivers. Each transmitted signal will be received by each
%receiver, therefore we have 4x4=16 radio channels.

rx11 = zeros(1,numBits*k);
rx12 = zeros(1,numBits*k);
rx13 = zeros(1,numBits*k);
rx14 = zeros(1,numBits*k);
rx21 = zeros(1,numBits*k);
rx22 = zeros(1,numBits*k);
rx23 = zeros(1,numBits*k);
rx24 = zeros(1,numBits*k);
rx31 = zeros(1,numBits*k);
rx32 = zeros(1,numBits*k);
rx33 = zeros(1,numBits*k);

```



```

rx34 = zeros(1,numBits*k);
rx41 = zeros(1,numBits*k);
rx42 = zeros(1,numBits*k);
rx43 = zeros(1,numBits*k);
rx44 = zeros(1,numBits*k);

%4 paths for transmitter 1
rx11 = jah_fir_filter(dn_upsampled1,channel11);
rx12 = jah_fir_filter(dn_upsampled1,channel12);
rx13 = jah_fir_filter(dn_upsampled1,channel13);
rx14 = jah_fir_filter(dn_upsampled1,channel14);
%4 paths for transmitter 2
rx21 = jah_fir_filter(dn_upsampled2,channel21);
rx22 = jah_fir_filter(dn_upsampled2,channel22);
rx23 = jah_fir_filter(dn_upsampled2,channel23);
rx24 = jah_fir_filter(dn_upsampled2,channel24);
%4 paths for transmitter 3
rx31 = jah_fir_filter(dn_upsampled3,channel31);
rx32 = jah_fir_filter(dn_upsampled3,channel32);
rx33 = jah_fir_filter(dn_upsampled3,channel33);
rx34 = jah_fir_filter(dn_upsampled3,channel34);
%4 paths for transmitter 4
rx41 = jah_fir_filter(dn_upsampled4,channel41);
rx42 = jah_fir_filter(dn_upsampled4,channel42);
rx43 = jah_fir_filter(dn_upsampled4,channel43);
rx44 = jah_fir_filter(dn_upsampled4,channel44);

% model the thermal noise at input to each receiver
thermal_noise1 = stdDev*randn(1,numBits*k);
thermal_noise2 = stdDev*randn(1,numBits*k);
thermal_noise3 = stdDev*randn(1,numBits*k);
thermal_noise4 = stdDev*randn(1,numBits*k);

% The received data at each receiver is the sum of each of the 4
% transmitted values which have passed through different channels...
rn1 = rx11 + rx21 + rx31 + rx41 + thermal_noise1;
rn2 = rx12 + rx22 + rx32 + rx42 + thermal_noise2;
rn3 = rx13 + rx23 + rx33 + rx43 + thermal_noise3;
rn4 = rx14 + rx24 + rx34 + rx44 + thermal_noise4;

w1 = zeros(1,filterOrder);

```

```

w2 = zeros(1,filterOrder);
w3 = zeros(1,filterOrder);
w4 = zeros(1,filterOrder);

error_log1 = zeros(1,numBits);
error_log2 = zeros(1,numBits);
error_log3 = zeros(1,numBits);
error_log4 = zeros(1,numBits);

delay = 0;
sampling_phase = 0;

%run simulation for receiver 1
for i = 1 : numBits*k,
    if(i-(filterOrder-1)+sampling_phase>=1 & i-(0-1)+sampling_phase <= length(rn1) &
        (i-1)/k-delay >= 1),
        if(mod(i-1,k) == 0),
            dn_hat = 0;
            for j = 1 : filterOrder,
                dn_hat = dn_hat + w1(j) * rn1(i-(j-1)+sampling_phase);
            end

            error = dn1((i-1)/k-delay) - dn_hat;

            error_log1((i-1)/k) = error;

            %adjust the tap coefficients
            for j = 1 : filterOrder,
                w1(j) = w1(j) + mu * error * rn1(i-(j-1)+sampling_phase);
            end
        end
    end
end

wAxisCoordinates = [0:length(w1)-1];

%receiver 1
figure(1);
subplot(241);
stem(wAxisCoordinates, abs(w1).^2, 'o');
ylabel('|w|^2');
xlabel('Index, i, (-)');

```

```

subplot(245);
se_db = 10 * log10(abs(error_log1)+eps);
plot(se_db)
axis([0 (numBits-1) -40 10]);

%run simulation for receiver 2
count = 0 ;
count2=1;
for i = 1 : numBits*k,
    if ( 1 <= (i-delay) & ((i-(filterOrder-1))>0)),

        dn_hat = 0;
        for j = 1 : filterOrder,
            dn_hat = dn_hat + w2(j) * rn2(i-(j-1));
        end

        count=count+1;
        %only need to calculate error and update w every 4 samples
        if(count==k),
            error = dn_upsampled2(i-delay*k) - dn_hat;
            %error_log(i-delay) = error;
            error_log2(count2) = error;

            %adjust the tap coefficients
            for j = 1 : filterOrder,
                w2(j) = w2(j) + mu * error * rn2(i-(j-1));
            end
            count = 0;
            count2=count2+1;
        end
    end
end

%run simulation for receiver 3
count = 0 ;
count2=1;
for i = 1 : numBits*k,
    if ( 1 <= (i-delay) & ((i-(filterOrder-1))>0)),

        dn_hat = 0;

```

```

        for j = 1 : filterOrder,
            dn_hat = dn_hat + w3(j) * rn3(i-(j-1));
        end

        count=count+1;

        %only need to calculate error and update w every 4 samples
        if(count==k),
            error = dn_upsampled3(i-delay*k) - dn_hat;
            %error_log(i-delay) = error;
            error_log3(count2) = error;

            %adjust the tap coefficients
            for j = 1 : filterOrder,
                w3(j) = w3(j) + mu * error * rn3(i-(j-1));
            end
            count = 0;
            count2=count2+1;
        end
    end
end

%run simulation for receiver 4
count = 0 ;
count2=1;
for i = 1 : numBits*k,
    if ( 1 <= (i-delay) & ((i-(filterOrder-1))>0)),

        dn_hat = 0;
        for j = 1 : filterOrder,
            dn_hat = dn_hat + w4(j) * rn4(i-(j-1));
        end

        count=count+1;

        %only need to calculate error and update w every 4 samples
        if(count==k),
            error = dn_upsampled4(i-delay*k) - dn_hat;
            %error_log(i-delay) = error;
            error_log4(count2) = error;

            %adjust the tap coefficients
            for j = 1 : filterOrder,

```

```

        w4(j) = w4(j) + mu * error * rn4(i-(j-1));
    end
    count = 0;
    count2=count2+1;
end
end
end

wAxisCoordinates = [0:length(w1)-1];

%receiver 1
figure(1);
subplot(241);
stem(wAxisCoordinates, abs(w1).^2, 'o');
ylabel('|w|^2');
xlabel('Index, i, (-)');

subplot(245);
se_db = 10 * log10(abs(error_log1)+eps);
plot(se_db)
axis([0 (numBits-1) -40 10]);

%receiver 2
subplot(242);
stem(wAxisCoordinates, abs(w2).^2, 'o');
ylabel('|w|^2');
xlabel('Index, i, (-)');

subplot(246);
se_db = 10 * log10(abs(error_log2)+eps);
plot(se_db)
axis([0 (numBits-1) -40 10]);

%receiver 3
subplot(243);
stem(wAxisCoordinates, abs(w3).^2, 'o');
ylabel('|w|^2');
xlabel('Index, i, (-)');

subplot(247);
se_db = 10 * log10(abs(error_log3)+eps);

```

```

plot(se_db)
axis([0 (numBits-1) -40 10]);

%receiver 4
subplot(244);
stem(wAxisCoordinates, abs(w4).^2, 'o');
ylabel('|w|^2');
xlabel('Index, i, (-)');

subplot(248);
se_db = 10 * log10(abs(error_log4)+eps);
plot(se_db)
axis([0 (numBits-1) -40 10]);

```

B.2 Constellation Plotter

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% JAH_CONSTELLATION_PLOTTER This script plots the demodulated inphase and
% quadrature signals, along with the corresponding 4-QAM constellation,
% based on the signal stored in data, over the range from 0 to 2*pi.
%
% Author: J. Andy Harriman
% Date: June 26, 2006
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%data = xlsread('data36.xls');
%import_data36

fi = 10.7e6;
t = time/10^12;
h2=fir1(64,0.05,'low');

%delta = 0.26; %43.2'

for i=0:0.01:2,
    yc = cos(2*pi*fi*t+i*pi);
    zc = adc_sDec .* (yc');
    zfc = conv(h2,zc);

    ys = sin(2*pi*fi*t+i*pi);
    zs = adc_sDec .* (ys');
    zfs = conv(h2,zs);

    subplot(311)
    plot(t,zfc(1:length(t)));legend(['i = ' num2str(i) '*pi = ' num2str(i*pi*180/pi) 'deg'])
    subplot(312)
    plot(t,zfs(1:length(t)));legend(['q = ' num2str(i) '*pi = ' num2str(i*pi*180/pi) 'deg'])
    subplot(313)
    plot(zfc,zfs); grid on

    if i == 0.68,

        %colors: blue = 0, red = 1
        color = 0;
```

```

for j = 1:10,
    if color == 0, %if(blue)
        subplot(311)
        plot(t,zfc(1:length(t)), 'r'); legend(['i = ' num2str(i) '*pi = '
            num2str(i*pi*180/pi) 'deg'])
        subplot(312)
        plot(t,zfs(1:length(t)), 'r'); legend(['q = ' num2str(i) '*pi = '
            num2str(i*pi*180/pi) 'deg'])
        subplot(313)
        plot(zfc,zfs, 'r'); grid on
        color = 1; %color = red
        pause(0.5)
    else
        subplot(311)
        plot(t,zfc(1:length(t)), 'b'); legend(['i = ' num2str(i) '*pi = '
            num2str(i*pi*180/pi) 'deg'])
        subplot(312)
        plot(t,zfs(1:length(t)), 'b'); legend(['q = ' num2str(i) '*pi = '
            num2str(i*pi*180/pi) 'deg'])
        subplot(313)
        plot(zfc,zfs, 'b'); grid on
        color = 0; %color = blue
        pause(0.5)
    end
    return
end
end
pause(0.05)
%pause
end

```


B.3 QAM Synchronizer

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% JAH_QAM_SYNCHRONIZER This script simulates the transmit and receive
% fuctions of a QAM transceiver. The demodulated signal is then used to
% perform carrier recovery.
%
% Author: J. Andy Harriman
% Date: June 26, 2006
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear;

nSamples = 4096; %number of samples to take
fs = 80e6; %sampling rate = 80MSPS
samplePeriod = 1/fs;
t = samplePeriod:samplePeriod:(samplePeriod*nSamples); %time in ns
f_if = 10.7e6; %IF frequency

f_i = 100e3; %100 kHz inphase signal
f_q = 250e3; %250 kHz inphase signal
i_in = square(2*pi*f_i*t);
q_in = square(2*pi*f_q*t);
%plot(t,i_in,t,q_in);legend('i input','q input');

%modulator board
%fc = 2.4e9; %2.4 GHz carrier
fc = 10.7e6;
ic = cos(2*pi*fc*t).*i_in;
qs = sin(2*pi*fc*t).*q_in;
y_tx = ic + qs;

%channel
y_rx = y_tx; %assume channel is perfect

%FPGA board
%NCO sine and cosine outputs for demodulation
phi_deg = 0;
phi_rad = phi_deg * pi/180;

yc = cos(2*pi*f_if*t + phi_rad);
```

```

ys = sin(2*pi*f_if*t + phi_rad);
%plot(t(1:100),yc(1:100),t(1:100),ys(1:100));legend('cos','sin');

zc = yc .* y_rx;
zs = ys .* y_rx;

%low pass filter the data
nTaps = 64; %64 tap filter
h1=fir1(nTaps,0.0625,'low'); % cutoff = 5 MHz
zfc = conv(h1,zc);
zfs = conv(h1,zs);
%plot(t,zfc(1:length(t)),t,zfs(1:length(t)));legend('i output','q output');
%plot(zfc,zfs);

%downsample by 4
%zfc_ds = decimate(zfc,4);
%zfs_ds = decimate(zfs,4);

%carrier recovery loop
h2 = fir1(512,[0.13374 0.13376],'bandpass'); %10.699-10.7001 MHz
i_sync = conv(h2,zc);
q_sync = conv(h2,zs);

test = conv(h2,zc);
N=length(test);
f = [0:(N-1)]*(fs/N);
%plot(f,abs(fft(test)))
%plot(test)

test=test.^2;
%plot(f,abs(fft(test)))

test2 = conv(h2,zs);
test2 = test2.^2;
%plot(f,abs(fft(test2)))

test3 = test+test2;
%plot(f,abs(fft(test3)))

h3 = fir1(512,[0.26749 0.26751],'bandpass'); %21.399-21.4001 MHz
test4 = conv(h3,test3);

```

```
N=length(test4);  
f = [0:(N-1)]*(fs/N);  
%plot(f,abs(fft(test4)))  
  
subplot(211)  
plot(t,test4(1:length(t)))  
subplot(212)  
plot(f,abs(fft(test4)))
```

B.4 SignalTap[®] to MATLAB[®] Example

The script provided below demonstrates how to import data from a SignalTap .csv file, with the header information removed, into the MATLAB[®] workspace. The scripts which are used to perform the binary to decimal conversion are provided in the following section.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% JAH_IMPORT_DATA This script extracts the binary data stored in the
% variable 'data', and converts the recovered variables back to decimal.
%
% Author: J. Andy Harriman
% Date: June 26, 2006
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Extract the data from the appropriate cells

time = data(:,1);
adc = data(:,3:16);
nco_sin = data(:,18:31);
nco_cos = data(:,33:46);
lpm_q = data(:,48:75);
lpm_i = data(:,77:104);
lpf_q = data(:,106:144);
lpf_i = data(:,146:184);

%Combine the values consisting of multiple columns (binary) into their decimal equivalent
[adc_sDec,adc_uDec] = jah_sbin2dec(adc);
[nco_sin_sDec,nco_sin_uDec] = jah_sbin2dec(nco_sin);
[nco_cos_sDec,nco_cos_uDec] = jah_sbin2dec(nco_cos);
[lpm_q_sDec,lpm_q_uDec] = jah_sbin2dec(lpm_q);
[lpm_i_sDec,lpm_i_uDec] = jah_sbin2dec(lpm_i);
[lpf_q_sDec,lpf_q_uDec] = jah_sbin2dec(lpf_q);
[lpf_i_sDec,lpf_i_uDec] = jah_sbin2dec(lpf_i);
```

B.4.1 Binary-to-Decimal Conversion Scripts

```
function [signedResult,unsignedResult] = jah_ubin2dec(ubin);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% JAH_UBIN2DEC This function returns the decimal equivalent, both a
% signed and an unsigned version, of the unsigned binary NxM array of
% numbers passed in.
%
% Parameters:
% ubin = the array of unsigned binary values
%
% Author: J. Andy Harriman
% Date: April 27, 2006
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

numBits = length(ubin(1,:));

if(numBits <=1),
    disp('Error: The number of bits must be greater than or equal to 1.')
    result = NaN;
    return
end

k = 1;
for j = 1:1:size(ubin,1),
    signedResult(k) = 0;

    if(ubin(j,1)), %MSB is set
        signedResult(k) = 0-2^(numBits);
    end

    for i = 2:1:numBits,
        signedResult(k) = signedResult(k) + ubin(j,i) * 2^(numBits-i);
    end

    k = k + 1;
end

k = 1;
for j = 1:1:size(ubin,1),
    unsignedResult(k) = 0;
```

```
    for i = 1:1:numBits,
        unsignedResult(k) = unsignedResult(k) + ubin(j,i) * 2^(numBits-i);
    end

    k = k + 1;
end
```

```

function [signedResult,unsignedResult] = jah_sbin2dec(sbin);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% JAH_SBIN2DEC This function returns the decimal equivalent, both a
% signed and an unsigned version, of the signed binary NxM array of
% numbers passed in.
%
% Parameters:
% sbin = the array of signed binary values
%
% Author: J. Andy Harriman
% Date: March 24, 2006
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

numBits = length(sbin(1,:));

if(numBits <=1),
    disp('Error: The number of bits must be greater than or equal to 1.')
    result = NaN;
    return
end

k = 1;
for j = 1:1:size(sbin,1),
    signedResult(k) = 0;

    if(sbin(j,1)), %MSB is set
        signedResult(k) = 0-2^(numBits);
    end

    for i = 1:1:numBits,
        signedResult(k) = signedResult(k) + sbin(j,i) * 2^(numBits-i);
    end

    k = k + 1;
end

for j = 1:1:size(sbin,1), %invert the MSB
    if(sbin(j,1)),
        sbin(j,1) = 0;
    else

```

```

        sbin(j,1) = 1;
    end
end

k = 1;
for j = 1:1:size(sbin,1),
    unsignedResult(k) = 0;

    for i = 1:1:numBits,
        unsignedResult(k) = unsignedResult(k) + sbin(j,i) * 2^(numBits-i);
    end

    k = k + 1;
end

```


Appendix C

Design Source Code

Due to the fact that the baseband design is primarily composed of Altera's® proprietary megacore functions, and not code developed by the author, the final system design files are included in the enclosed CD-ROM. However, the code for the PN generator module is provided below.

C.1 PN Generators

```

/*****
*
* This module implements a 10 register PN Generator Circuit.
*
* Author: J. Andy Harriman
* Date: June 26, 2006
*
*****/

module PNGenerator(clk, enable, reset, taps, dout);

input clk;
input enable;
input reset;
input [9:1] taps;
output dout;

```

```

reg dout;
reg [9:0] temp;
reg [9:0] state;

always @(posedge clk or posedge reset)
begin
if(reset)
begin
dout = 1'b0;
state = 10'b0000000001;
//dout = state[0];
end
else if(enable)
begin
dout = state[0];
temp = state;
state[8:0] = temp[9:1];
state[9] = temp[0]^(taps[1]&temp[1])^(taps[2]&temp[2])^(taps[3]&temp[3])^(taps[4]&temp[4])^(
taps[5]&temp[5])^(taps[6]&temp[6])^(taps[7]&temp[7])^(taps[8]&temp[8])^(taps[9]&temp[9]);
//dout = state[0];
end
end
endmodule

```

```

/*****
*
* This module outputs eight different tap configurations for
* the PN Generator. Each of the tap configurations produces
* a maximal length sequenc of 1023 bits.
*
* Author: J. Andy Harriman
* Date: June 26, 2006
*
*****/

module TapConfigs(taps1,taps2,taps3,taps4,taps5,taps6,taps7,taps8);

output [9:1] taps1,taps2,taps3,taps4,taps5,taps6,taps7,taps8;

/*
10-bit LRS tap positions... (this is not an exhaustive list)

1. (10,3,0)
2. (10,4,3,1,0)
3. (10,5,2,1,0)
4. (10,5,3,2,0)
5. (10,6,5,2,0)
6. (10,6,5,3,2,1,0)
7. (10,7,0)
8. (10,7,3,1,0)

The PNGenerator block assumes that 10 and 0 are already set (requirement for LRS generator circuit).
Therefore, this block specifies only tap positions 9 through 1 for each configuration.
*/

assign taps1 = 9'b000000100;
assign taps2 = 9'b000001101;
assign taps3 = 9'b000010011;
assign taps4 = 9'b000010110;
assign taps5 = 9'b000110010;
assign taps6 = 9'b000110111;
assign taps7 = 9'b001000000;
assign taps8 = 9'b001000101;

endmodule

```

Appendix D

Custom PCB Board Designs

This section provides the schematics, board layouts, and simulation results for the various PCB boards developed for this thesis.

D.1 FPGA to Hittite Signal Conditioning Circuit

Figures [D.1](#) and [D.2](#) show the schematic for the board, while [Figure D.3](#) illustrates the PCB board layout, and [Figure D.4](#) provides the PSpice® simulation results for the circuit design.

D.2 Hittite Bias Circuit

[Figure D.5](#) provides the schematic for the Hittite bias circuit design.

D.3 Clock Signal Level Translator

[Figure D.6](#) illustrates the circuit schematic for the clock signal level translator board design. The PCB board layout for this design is shown in [Figure D.7](#).

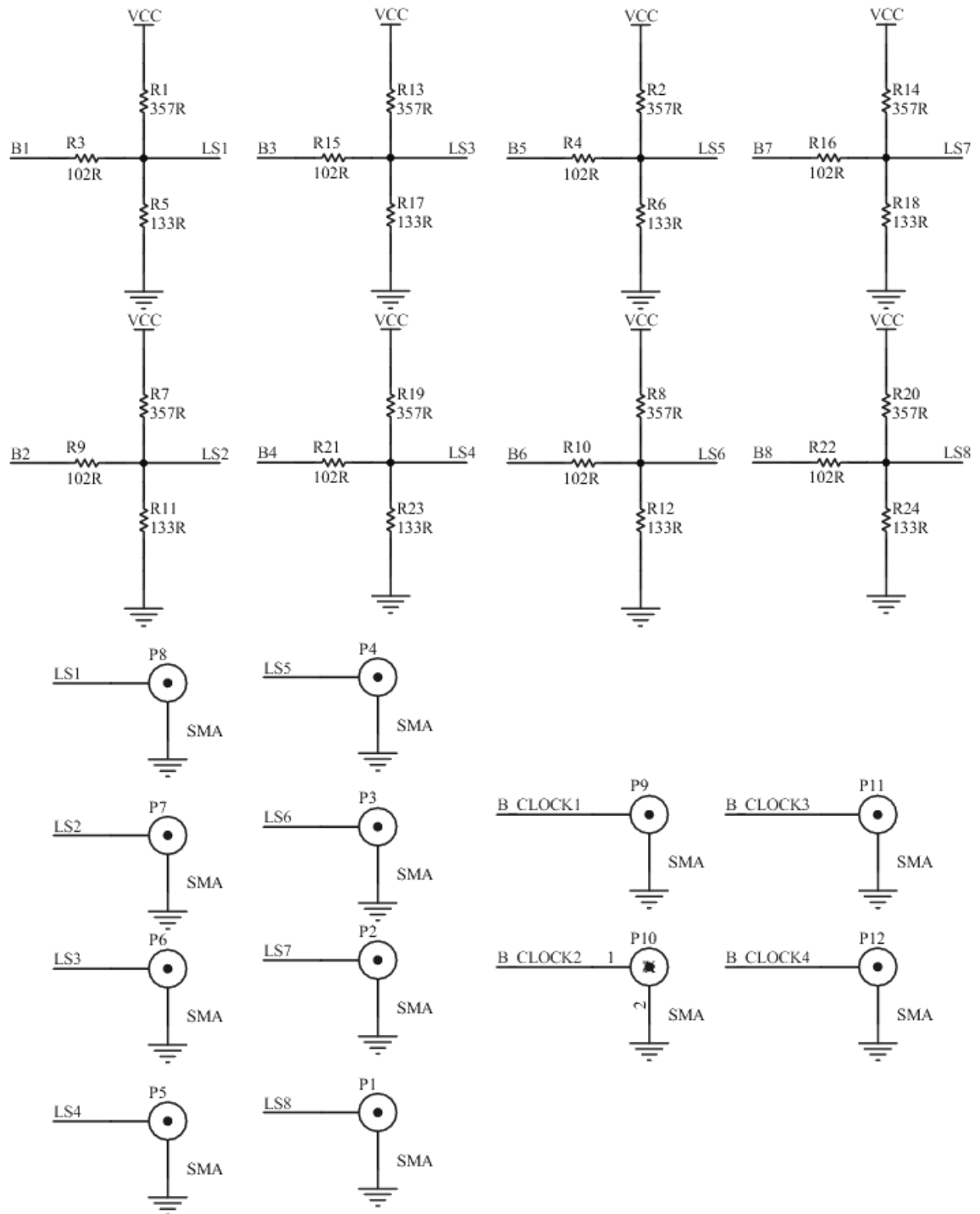


Figure D.2: Hittite Signal Conditioning Circuit Schematic, Page 2/2

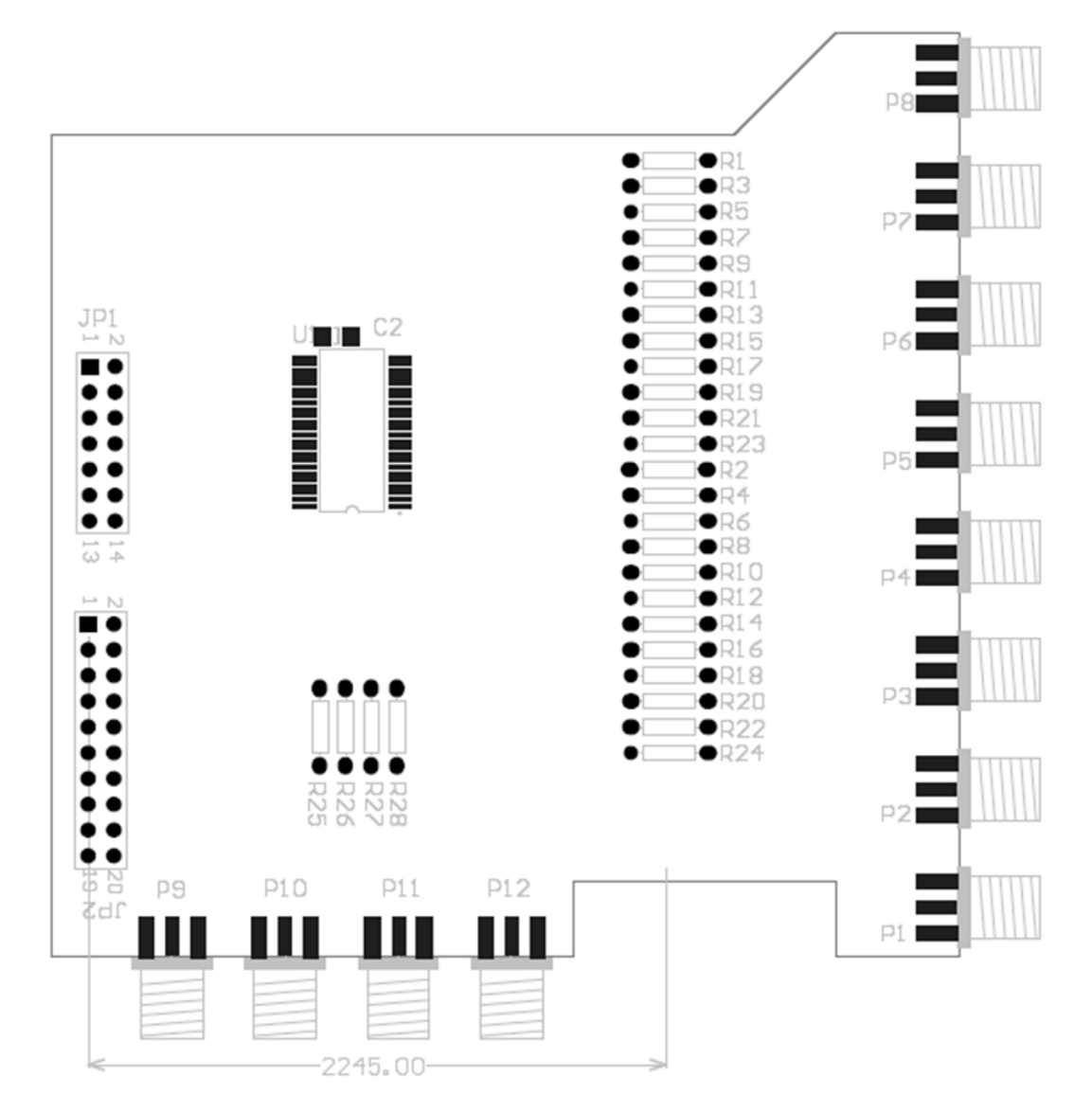


Figure D.3: Hittite Signal Conditioning Circuit PCB Layout

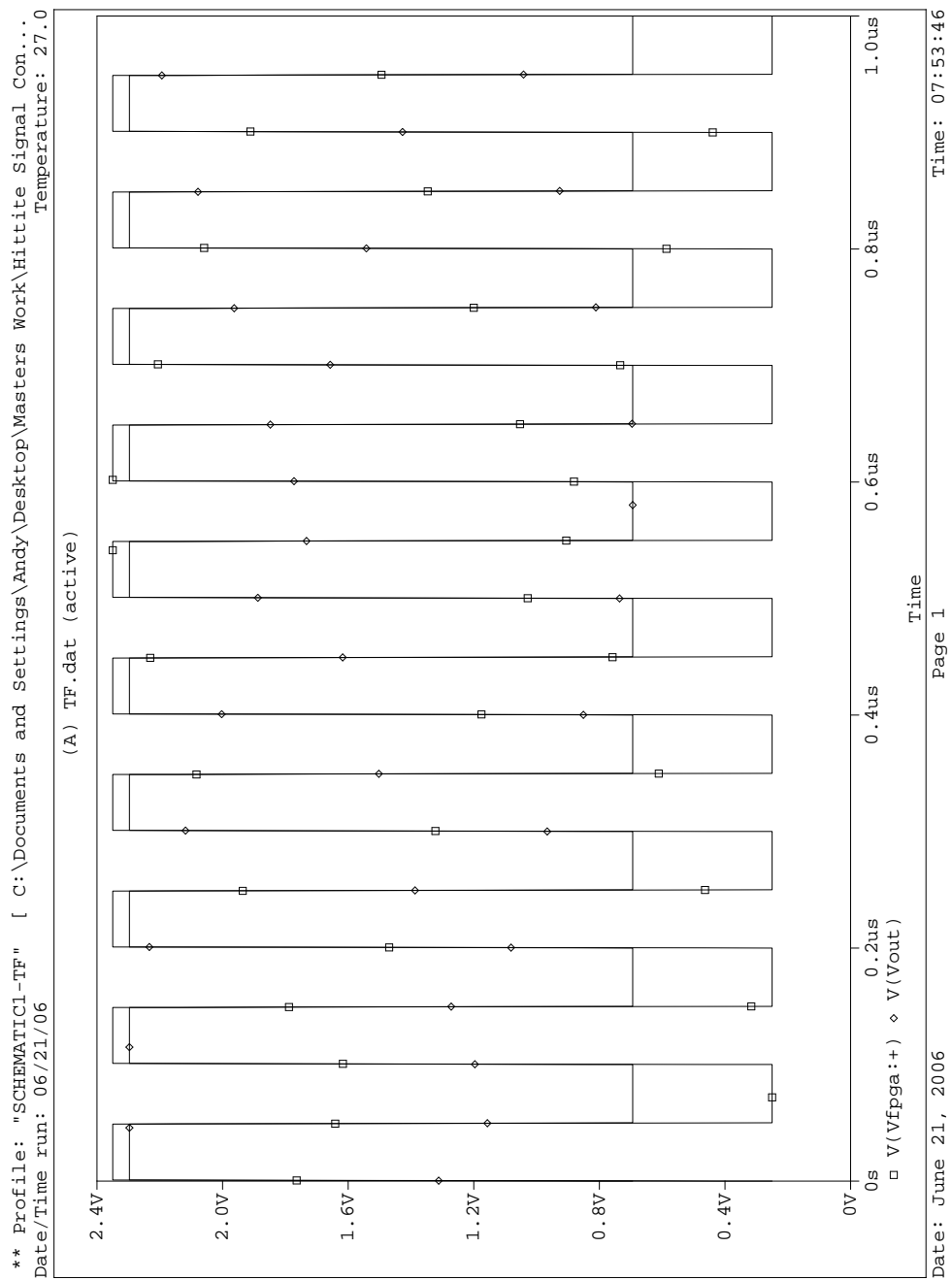


Figure D.4: Hittite Signal Conditioning Circuit Simulation

Hittite Direct Quadrature Modulator Bias Circuit

By: J. Andy Harriman, May 31st 2006

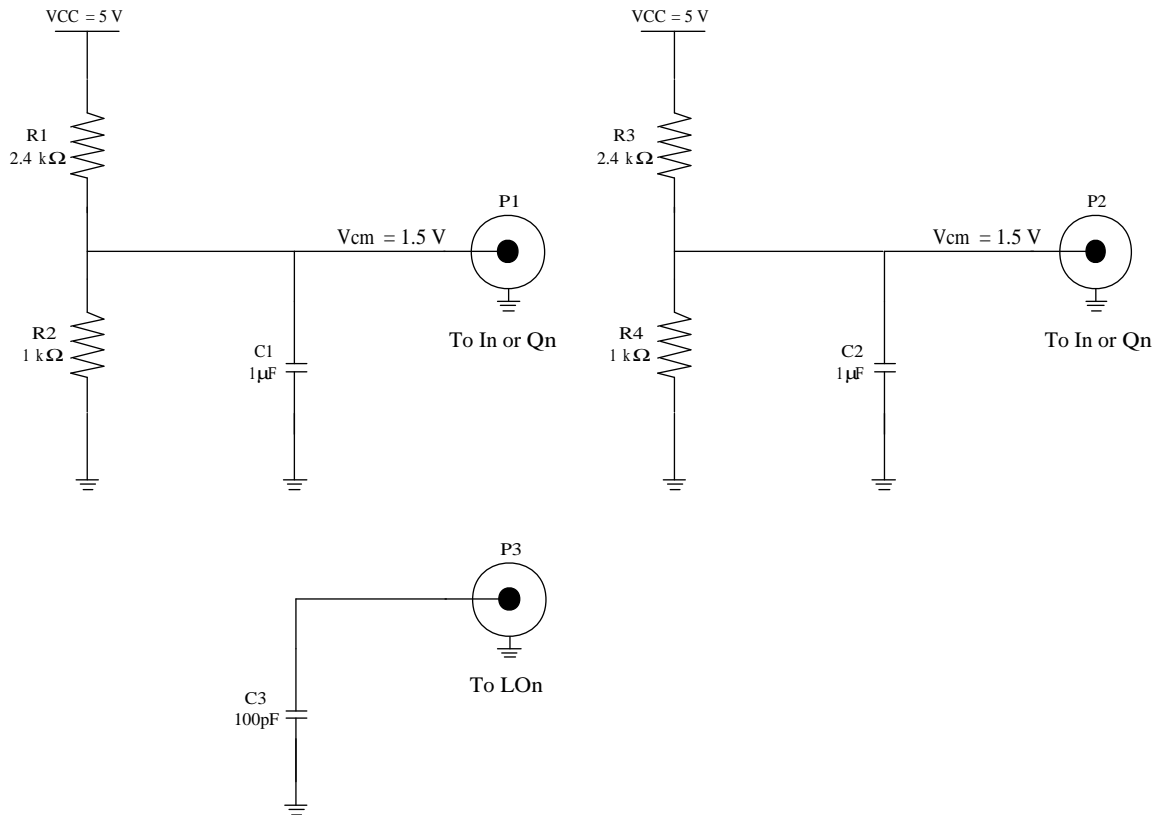


Figure D.5: Hittite Bias Circuit Schematic

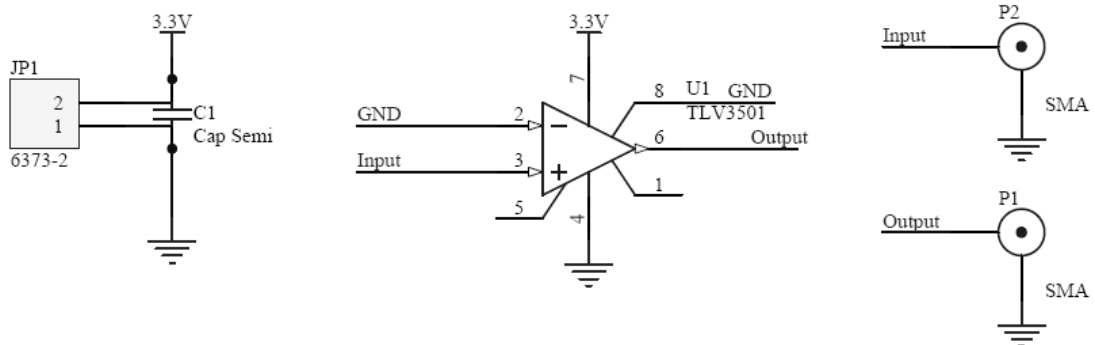


Figure D.6: Clock Level Translator Circuit Schematic

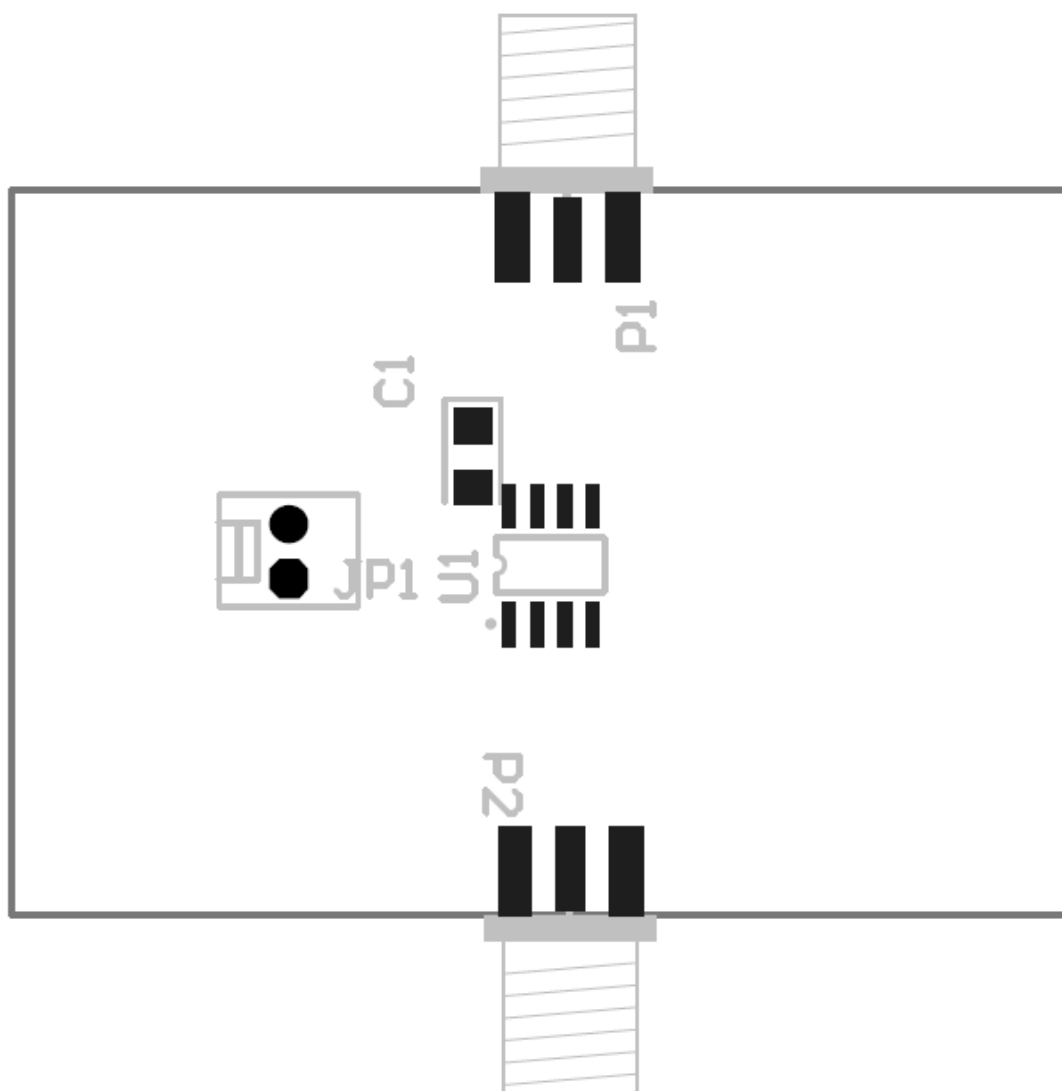


Figure D.7: Clock Level Translator Circuit Layout

D.4 GPIO-to-SMA Testpoint Board

This board was created for use as a debugging tool. It provides eight SMA testpoint connections to user programmable GPIO pins in the FPGA. The schematic and layout for this PCB board can be found in Figures D.8 and D.9 respectively.

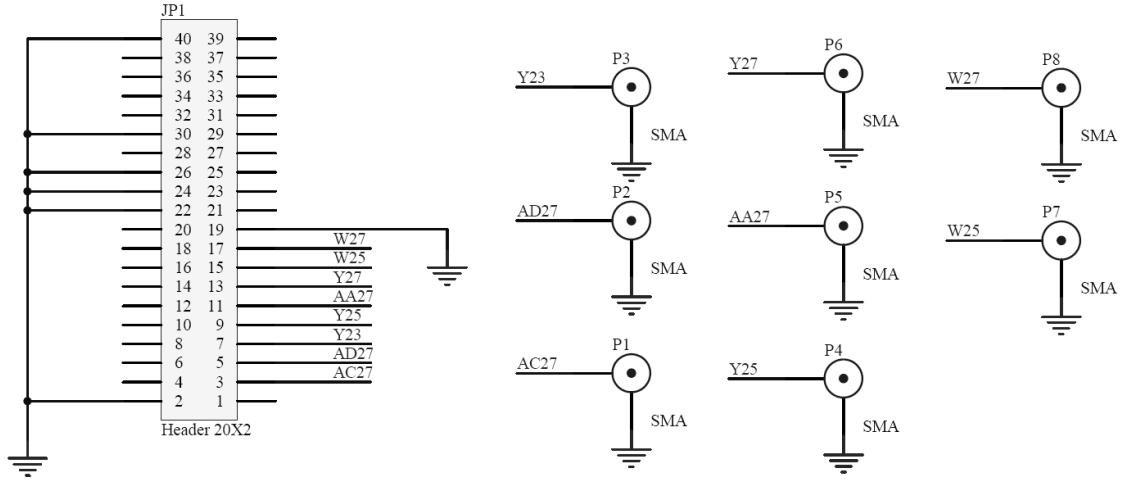


Figure D.8: GPIO-to-SMA Testpoint Board Schematic

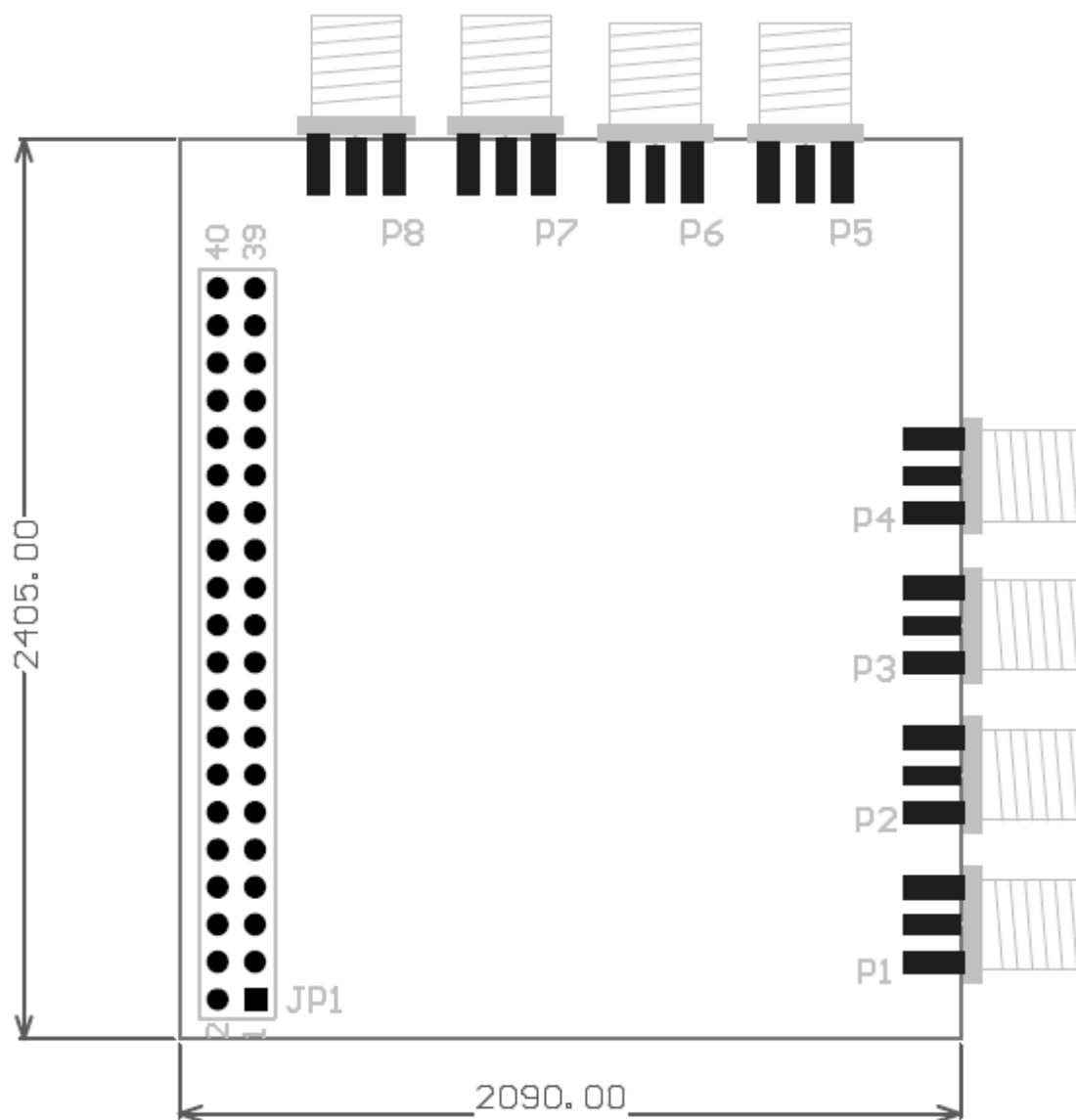


Figure D.9: GPIO-to-SMA Testpoint Board Layout

Appendix E

Modifications to the Power Supply

In order to make use of a PC power supply for general purpose applications, a dummy load must be applied in order for the supply to switch on. For the purposes of this thesis, additional features such as a power switch and status LED were added, but are not strictly required. [Table E.1](#) lists the internal wiring of most modern power supplies. This information is generally provided on the label of the power supply.

Table E.1: Power Supply Wiring

Wire Colour	Purpose
White	-5 V
Red	+5 V
Black	0 V
Yellow	+12 V
Blue	-12 V
Brown	Sense
Orange	+3.3 V
Purple	+5V Standby
Grey	Power is on
Green	Turn DC on

[Figure E.1](#) illustrates the wiring changes required to create the power supply used for this thesis. Note that the +5 V standby line is disconnected, and that the

appropriate power lines were connected to their corresponding banana connectors.

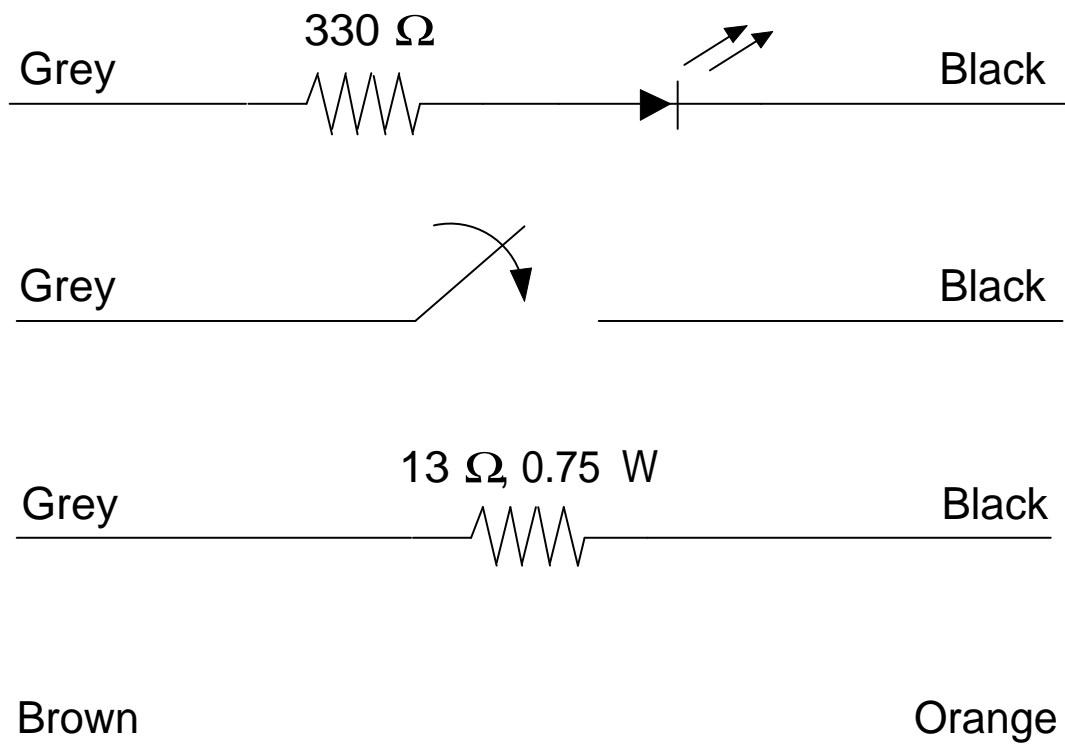


Figure E.1: Power Supply Wiring Modifications

Vita

Candidate's full name: James Andrew Harriman

University attended: University of New Brunswick, B.Sc.E., 2004

Publications:

J. Andy Harriman, Brent R. Petersen, and Mary E. Kaye, "A reconfigurable four-channel transceiver testbed with signalling-wavelength-spaced antennas under centralized FPGA control," *IEEE Proceedings of the 4th Annual Communications Networks Services and Research Conference (CNSR 2006)*, vol. 1, pp. 311-313, 2006.