April 2009

Widely Tunable RF Frontend for the Universal Software Radio Peripheral: the MMP9000

Matthew Brian Murdy
Worcester Polytechnic Institute

Michael Carpenter Bruno
Worcester Polytechnic Institute

Peter J. Perreault
Worcester Polytechnic Institute

Widely Tunable RF Frontend for the Universal Software Radio Peripheral: the MMP9000

A Major Qualifying Project Report
Submitted to the Faculty of the

WORCESTER POLYTECHNIC INSTITUTE

in partial fulfillment of the requirements for the
Degree of Bachelor of Science
in Electrical and Computer Engineering
by

_____
Michael Bruno

_____
Matthew Murdy

_____
Peter Perreault

Sponsoring Organization:
NECAMSID
Project Advisors:

_____
Professor Alexander Wyglinski, Advisor

_____
Professor John McNeill, Co-Advisor

April 30, 2009

Abstract

This report presents the design and construction of a wideband transceiver in the context of an RF frontend for a software radio development platform, the Universal Software Radio Peripheral (USRP). This daughterboard is designed to operate at either full or half duplex modes over a frequency range of 100 MHz to 1.3 GHz or greater. It is fully integrated with both the USRP and GNU Radio, a free software radio development toolkit, to fully control the daughterboard via software.

Acknowledgements

The accomplishments made by this team throughout this project would not have been possible without the help of several very important people. First, we would like to thank Professor Wyglinski, our advisor. He set up our team, gave us ideas to choose from and guided us towards developing our own project. Throughout the project, he set high standards for our team, which kept us on the track to meeting our goals.

A project like this needed funding from the school and more specifically, these funds came from NECAMSID. The team would like to thank Professor McNeil for sponsoring our project and allowing our team to use his lab resources throughout our project.

The team would also like to thank Bob Boisse for soldering components on two different revisions of the board for us. Mr. Boisse soldered down several of the components that have pads underneath them, a task that would not have been too difficult for us. We would not have been able to test or debug our systems in a timely fashion without his help.

A large portion of the development for this project was facilitated through the use of evaluation boards which were supplied to the team free of charge. For this, the team would like to thank Lon Cecil from RFMD for giving the team a free RF2051 Evaluation Board, test software, and free samples of the RF2052. We would like to point out that at the time we asked for all of this, the RF2052 was not even available for sale. We would also like to thank Lauren Stieh from Minicircuits for providing us with a free evaluation board of their GVA-84+ evaluation board. Both of these boards were extremely helpful and we would not have been able to afford them on our budget without the generosity of these two people.

Finally, we would like to thank any other company that sent us samples throughout the project. Many of these companies give samples to students and we just want to point out how invaluable such generosity can be. Also, thank-you to anyone else who helped answer our questions at one point or another or provided any kind of assistance with this project.

Executive Summary

Historically, the field of radio and communications has been an ever changing and thriving field, both culturally and technologically. Improvements in radio technology have brought about their widespread use through all facets of society, for countless purposes, enabling communication between consumers, public utilities, military entities, and more. Unfortunately, one of the side effects of the Communication Age is a scarcity of the main resource enabling radio communications; the wireless spectrum. Recent developments in the field of radio communications have sought to address this problem, increasing spectral efficiency through technological developments in software defined radio (SDR) and cognitive radio.

This Major Qualifying Project seeks to extend the capabilities of software defined radio through the design and creation of a wideband RF frontend for a software defined radio development platform called the Universal Software Radio Peripheral (USRP). The RF frontend created in this project extends the frequency range available to software radio applications designed for the USRP, and is compatible with the accompanying open source software platform, GNU Radio. In addition, because all the designs for the USRP daughterboards and motherboards are available under the GPL license, the schematics, PCB designs and layout, and other information for this project will be available under the same license.

The main motivation for creating a wideband transceiver lies in the convenience of having a large frequency range available for software radio applications. Existing daughterboards available for the USRP cover smaller frequency ranges and thus severely limit possible wideband applications that can be made on this platform. The creation of a wideband transceiver is truly a step in the direction of a perfect cognitive radio; able to operate over all frequencies and change any operating parameter, such as transmission power, modulation type, or bandwidth. Design elements of this daughterboard can also be abstracted and applied to other software radio applications, with this report serving as a guide to possible software radio engineers.

Once the general idea of the project was agreed upon, a few objectives were established to make sure the design criteria were met. The first and most important objective was to create a wideband transceiver that can operate over a large range of frequencies with an accurate tunable resolution. Initial plans were to make it tunable between 100 MHz and 1.3 GHz, but after finding components with a larger available range, the final version of the transceiver is tunable between 50 MHz and 2.5 GHz. The second objective was to make the transceiver completely compatible with GNU Radio and the USRP. This includes both physical connections to the USRP and software to allow GNU Radio to control the daughterboard. As the last objective, the transceiver had to be able to operate in full or half duplex modes, again controllable through GNU Radio. To meet these objectives the MQP team had three academic terms with a possible fourth, and a budget of one thousand dollars.

Initial designs of this wideband transceiver explored the possibility of using relays and switches to control the signal path between various VCOs, mixers, or even separate daughterboards; a technique often used for wideband radio applications. This project takes a different approach by interconnecting various wideband components to save board space, reduce power consumption, and decrease complexity compared to an exhaustive system consisting of multiple RF frontends.

Once the initial design was generally determined, simulation had to be used where possible to help improve the design and ensure its feasibility. Unfortunately, there is no conceivable way to simulate the entire design, so only the amplifier and antenna switching stages could be simulated before the PCB design. The design of the low noise amplifiers, power amplifier, and antenna switches used in the design were simulated using S-Parameter data in Agilent's ADS software. Though simulating these components aided in the overall design, the values of the discrete components were difficult to determine because small changes in these component values would lead to drastic changes on the overall shape of the gain of the amplifier stage. Since the simulation did not take into account the effects of trace length and other non-idealities of the final design, the usefulness of the simulation results were brought into question. In the end, the best component values were chosen, and the design remained open to any changes in these values. The design of other daughterboards was also taken into account, with some design aspects of the transceiver similar to other daughterboards wherever possible.

Where simulations fell short in aiding in the design process, evaluation boards provided by generous contributors to this project greatly aided in verifying the overall design of the transceiver. An evaluation board for the RF2051 wideband frequency synthesizer and mixer, along with an evaluation board of the GVA-84+ power amplifier, allowed the creation of a basic prototype tunable between 300 MHz and 2.5 GHz. Experiments in transmitting a simple FM signal over frequencies in this range were successful, with the signal being received by other USRP daughterboards, commercial FM radio receivers, amateur radio receivers, and even the other mixer on the same evaluation board.

With confidence gained from a successful prototype, the first schematic layouts and PCB designs were created using Mentor Graphic's PADS software. This software was chosen because of its features such as stitching vias, impedance calculations, copper pouring, and trace shielding. The design process in this program was relatively straightforward. First, the design is created in schematic form in PADS Logic, linking each schematic component to a footprint file. This schematic is then imported into PADS Layout, where the footprint information is used to create a PCB design and Gerber files needed to fabricate the PCB. Changes to the schematic would be reflected in the PCB design, allowing easy manipulation of the overall design.

A four layer PCB was used for the transceiver, with a signal layer, ground layer, power layer, and another signal layer. Utilities in the PADS software suite allowed the design parameters, such as minimum trace width and via size, to be changed so the PCB would fit in an affordable price bracket. These features were especially helpful to keep the project within budget, leaving enough money for a PCB revision to fix complications that arose from the initial PCB design.

Throughout the entire design process, a common concern was the small package sizes of some of the components, since the VCOs and modulators/demodulators were designed to be soldered by machines. Thankfully, Bob Boisse, a staff member in the ECE department, was able to solder these components to the PCBs. All components on the PCBs were placed by hand with a fine tipped soldering iron under a microscope.

To save time and parts, the components soldered to the PCB were tested at various opportunities to make sure everything was working as planned. In addition, the transmitter side was soldered first. Due to time constraints, if the transmitter half of the board was not functional, the problem would be diagnosed, appropriate changes would be made to the PCB, and a fixed PCB would be ordered. Unfortunately, this was the case. Connecting the USRP's general purpose input/output lines to the voltage regulators caused IO across the entire motherboard to stop working. A quick email to Matt Ettus, creator of the USRP and its daughterboards, revealed that the voltage regulators used in the design could not be controlled by the IO pins; they needed to always be on. A cut trace and jumper wire solved this problem, but further debugging revealed that a PLL filter was left out of the design, ensuring that another PCB revision would be needed.

The second PCB addressed both problems faced with the first PCB, but it was still to be determined if there were any additional problems. Also, the receive side of the first PCB was not soldered or tested because the RF2052 was missing the same PLL loop filter. More problems were expected to be encountered in the second PCB revision, but both time and budget would not allow for a third PCB to be fabricated.

After soldering the transmitter side of the PCB, it was verified that the changes to the design were successful and the transmitter was operating as expected. Connecting the transmitter to the spectrum analyzer revealed that the daughterboard operated over a range of frequencies that is much larger than the initial goals, but expected given the operating frequency of the components used in the design. Unfortunately, tests on the receiver revealed what is most likely a damaged demodulator, but other receiver components were verified to be functional. Software control over the antenna path circuitry also demonstrated that the daughterboard could indeed operate in both full duplex and half duplex mode if both the transmitter and receiver were fully functional. In short, even though the receiver was not functional in time for the deadline, the other objectives were technically met and the project was, overall, a success.

Table of Contents

Table of Figures

1 Introduction

1.1 Radio History

The word *spectrum* was initially used to refer to the range of colors observed when light passed through a prism. As an increasing number of radiation types in the electromagnetic spectrum were discovered, the term spectrum began to be used to describe any type of wave. In modern times, the term has found such widespread use that it is often used to describe any type of range of related objects or values, such as the political spectrum. Some years after James Clerk Maxwell predicted the propagation of electromagnetic waves in the mid 1800s, they were put to use in communication using spark gap transmitters. This type of design, which is incredibly impractical according to today's standards, used voltages on the order of tens of thousands of volts to charge an antenna, emitting electromagnetic radiation as the spark gap discharged. Unfortunately, this type of design does not take too much care of spectral usage, and few spark gap stations could be used in the same geographic area without causing interference to each other.

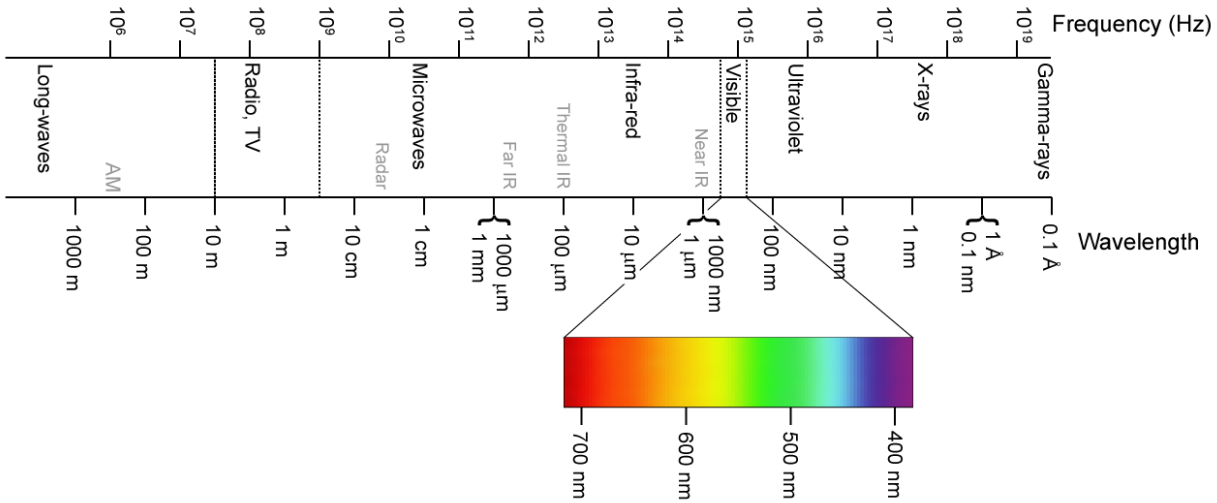


Figure 1 - The Electromagnetic Spectrum from long waves to gamma-rays. (Image released under Creative Commons Share Alike 3.0)

It was around this time that people realized the electromagnetic spectrum is a commodity. However, unlike copper, wool, or corn, the spectrum is, for the most part, invisible, location dependent, and theoretically limitless. In the United States, regulation of this commodity is the responsibility of the Federal Communications Commission (FCC). Without such regulation, interference and other problems communicating would be much more prevalent. Thankfully, we have come a long way since the spark gap transmitter, with advances in transceiver design allowing simultaneous usage of the electromagnetic spectrum between many users. Despite technical advances, the electromagnetic spectrum is still a much sought after commodity. For

example, the FCC recently ended an auction on the 700 MHz band with 101 winning bidders awarded 1091 licenses and a total of roughly 19 billion US dollars in bids. [**1**]

As usage of the electromagnetic spectrum increases steadily, the current approach to spectrum management is beginning to show signs of weakness. The FCC has designated different portions of the spectrum to different uses or services, for example, portions of the electromagnetic spectrum have been sectioned off for use in AM, FM, and TV broadcasting, amateur radio, maritime, satellite use, and many, many other uses. In the recent years, this approach has become more of a problem, since there is a limit to the number of users that can simultaneously use a portion of the spectrum at the same time and in the same geographical area. New applications are found for the spectrum, but there is a limited quantity of spectral space to serve these new uses.

One of the main problems leading to spectral crowding is the inefficient usage of the spectrum. Measurements of spectral occupancy, even in largely metropolitan areas, are incredibly low, at less than 20% (See Figure 2). Despite the low values of spectral occupancy, some frequency ranges are extremely overcrowded, such as the cell phone bands, around 850 MHz and 1800 MHz. [**2**] An obvious solution would be to allow services that need more bandwidth to take up unoccupied spectrum, but this type of operation is typically disallowed by the FCC because it causes interference.

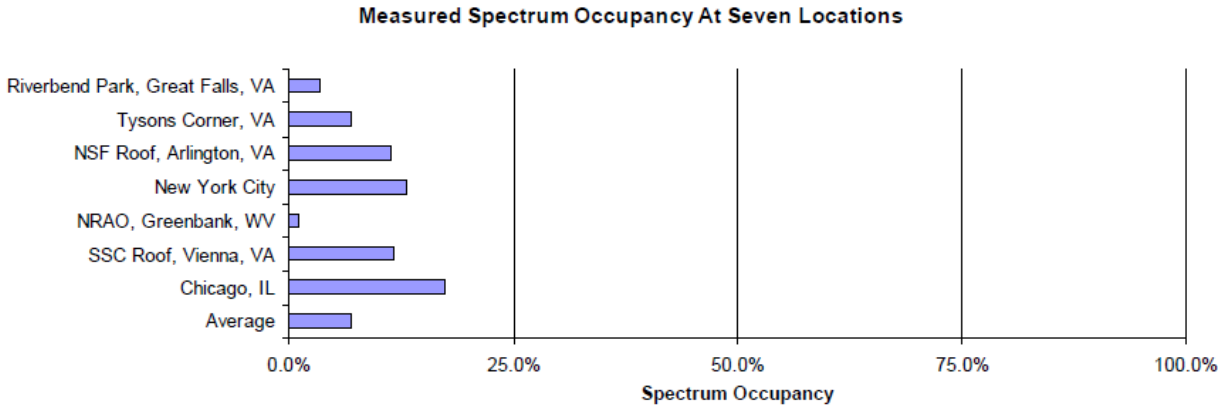


Figure 2- Spectral Occupancy Measured at Seven Locations - study done by the Shared Spectrum Company (reprinted with permission from the Shared Spectrum Company) [2**]**

Fortunately, regulating agencies such as the FCC are aware of this problem, and to offset the effects of spectral overcrowding there are a few frequency bands set aside for general use, such as the Industrial, Scientific, and Medical (ISM) band. It is in these bands that services such as Wi-Fi and Bluetooth have thrived, but there is a large amount of interference due to their widespread usage.

There are several techniques to help combat interference due to overcrowding, but in anticipation of changes in spectral laws, the next wave of radio technology seeks to make more efficient use of the electromagnetic spectrum with smarter radios. These radios, called *cognitive radios*,

monitor the wireless channel to choose the best parameters for transmission, such as modulation type or frequency. Clearly, the next step in radio history lies in effective, efficient, and intelligent use of spectral space, along with improvements in licensing and spectral management.

1.2 The Emergence of Software Defined Radio (SDR)

Software defined radio (SDR) is the technology enabling modern advances in cognitive radio. Since the invention of the term in 1991, SDR has generally referred to a radio whose functionality is at least partially controlled or implemented in software. To achieve this, RF components such as mixers and filters are moved to the software domain, where a computer or digital signal processor performs baseband signal processing. An ideal software radio would be able to replicate any waveform on any frequency, most likely by connecting digital-to-analog (DAC) and analog-to-digital converters (ADC) directly to an antenna. [**3**] The emergence of cheap high speed DACs and ADCs has made the ideal software radio concept closer and closer to a reality, while the ubiquity of personal computers and digital signal processors has caused quite a boom in the development of SDR and, since they are fundamentally related, cognitive radio.

Wireless devices that can be described as SDR have actually been around for quite some time, initially finding their niche in military applications before finding applications in the civilian market. Military programs such as SPEAKeasy sought to enable communication and interoperability between different, conflicting types of military radio. [**4**] The SPEAKeasy project was very ambitious, and the first prototype worked, but design choices of programming waveforms in low level assembly language meant that the software was not compatible with better processors as they were developed. Note that the Phase I prototype of SPEAKeasy was large enough to fit into the back of a truck. [**5**]

More recent software radio products and development testbeds take on a more modular approach, as to not have the same fate as the early prototype of the SPEAKeasy project. Commercial products, such as Vanu Inc.'s Anywave software radio, incorporate multiple cellular access standards into a simpler interface by using a software defined radio architecture. Since the cellular standards are implemented in software, they can be changed on the fly to adapt to different user needs of each cell, rather than by the costly approach of replacing RF hardware. This also means that they can easily be upgraded to match new standards. [**6**] To increase the effectiveness and improve the aging process of a software radio platform, most developers seek to use portable code for their software, reusable components that can work under different waveform configurations and generic hardware that can easily be upgraded. [**7**]

1.3 The Testbed: The Universal Software Radio Peripheral (USRP)

This project is based on the Universal Software Radio Peripheral, a research testbed and hardware development tool for software defined radio. The USRP, developed by Matt Ettus of Ettus Research LLC, has found significant popularity in cognitive radio research and general SDR studies and applications due to its relatively low cost, simple configuration, and its ability to work with GNU Radio, an open source software suite for SDR development. In addition to working with open source software, the schematics and PCB layouts for the USRP are also available online, free of charge.

The USRP is based on a motherboard/daughterboard concept. Daughterboards serve as a radio interface for the USRP motherboard, which is mainly concerned with providing a connection between the daughterboards and a PC. The motherboard connects to a PC by a USB 2.0 connection, giving it the ability to send up to 16 MHz of RF bandwidth in either direction. The motherboard contains 4 ADCs and 4 DACs, along with digital I/O lines to control connected daughterboards. Two daughterboard transceivers can be connected to the motherboard. Figure 3 and Figure 4 below show the design of the motherboard, along with an example of a daughterboard.

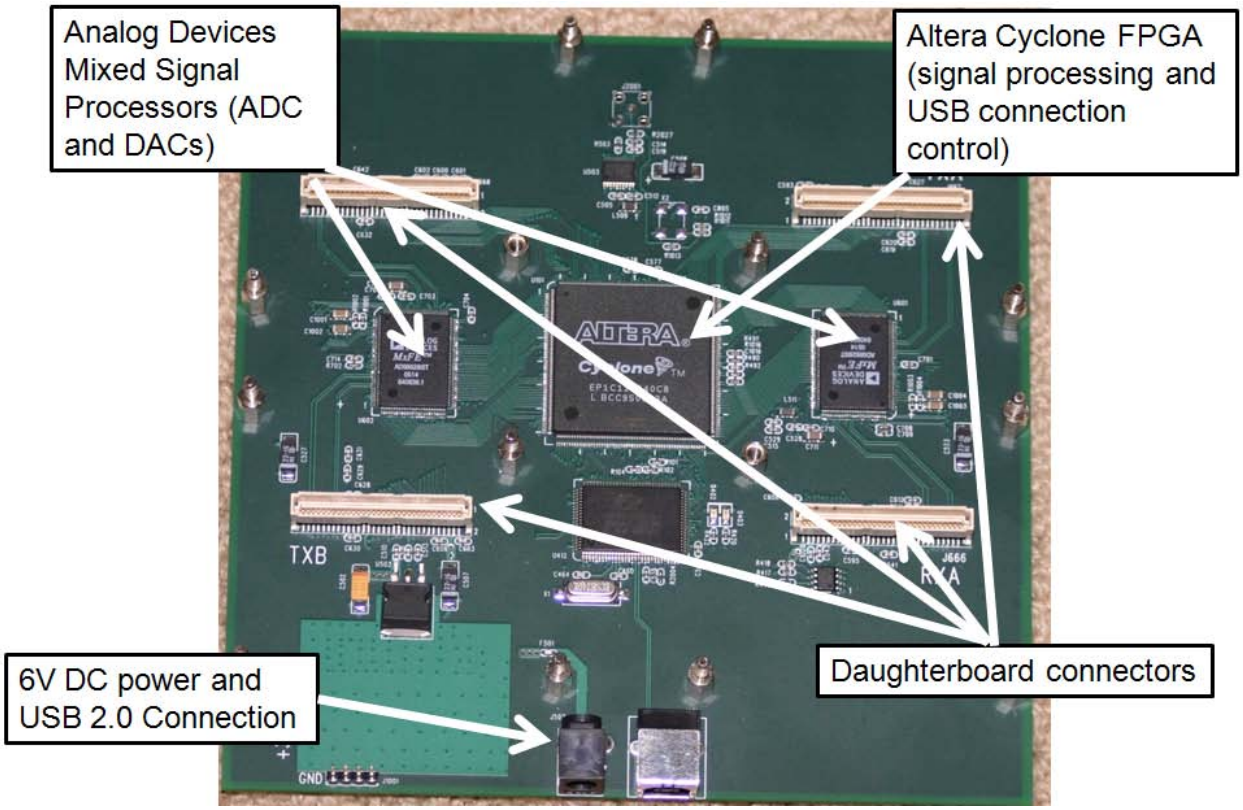


Figure 3 - The USRP motherboard components and layout, including the mixed signal processors, FPGA, connectors for daughterboards, and DC power and USB connections.

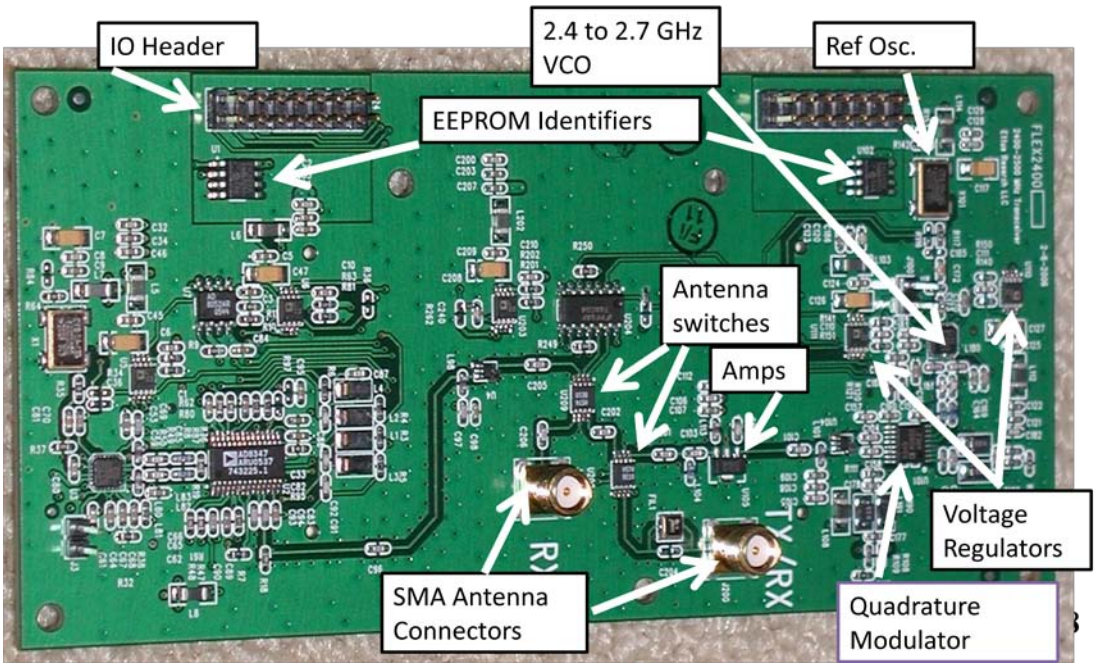


Figure 4 – An example of a USRP Daughterboard, the RFX2400. The RFX2400 board features a transmit and receive range from 2.3 to 2.9 GHz, with an output power of up to 50 mW.

1.4 The Project: A Wideband Transceiver for the USRP

This project is concerned with building a daughterboard for the USRP, specifically, a wideband transceiver that would allow the USRP to operate over a frequency range greater than what current daughterboards can offer. The following objectives were established at the beginning of the project to ensure success:

1.4.1 Objective 1: Tunable between 100 MHz and 1.3 GHz

The main objective of this project is to have a wideband transceiver that could operate in a large range of frequencies with an accurate resolution of operating frequencies. The initial range was rather arbitrarily chosen by examining a frequency chart and the acceptable frequency range for use on the discone antenna that is available in the lab. In the end, based on components that were chosen, it is expected that the transceiver will be able to operate over an even greater range, but the design will also have to take parasitic effects of high frequency signals into consideration, which will have a negative effect on the performance of the prototype.

1.4.2 Objective 2: Compatible with USRP and GNU Radio

In order to ensure correct operation, the prototype must properly interface with the USRP. This means the prototype must physically connect with the USRP and be controlled by software through GNU Radio, in order to change frequency and other parameters of the daughterboard. In addition, the prototype needs to be recognized the same way other available daughterboards are recognized by GNU Radio.

1.4.3 Objective 3: Full Duplex and Half Duplex Transceiver

The prototype needs to be able to act as a transceiver, such that it can both transmit and receive according to the needs of the user. Since the USRP has an ADC and DAC for receiving and transmitting, this functionality is a natural extension of the capabilities of most daughterboards. The mode of transceiver operation, whether half duplex or full duplex, will be controlled through software.

2 Background

This chapter will expand upon required background information needed to understand the motivation behind this project. A brief explanation of topics such as software defined radios, cognitive radio, the RF spectrum, and RF hardware is provided to aid the reader in understanding topics relevant to this project and provide more information on topics touched upon in the introduction.

2.1 Software Defined Radio

Since software defined radio is such a relatively new concept, it is difficult to find a consensus on a single definition. Essentially, it is a radio whose functionality is at least partially implemented in or controlled by software. The SDR Forum, in collaboration with IEEE, has defined it as "radio in which some or all of the physical layer functions are software defined." [**8**] To help get a better idea of what software defined radio is, it's helpful to examine its applications. Traditionally, radios were designed with a single target application in mind, such as receiving music, making a phone call, or receiving GPS data, but as technology advances, along with wireless standards and protocols, the need for multipurpose radios to support different types of information, modulation types, frequency ranges, bandwidth, and other aspects of radio design is ever increasing. [**9**] See Figure 5 for a block diagram representation of a software radio system.

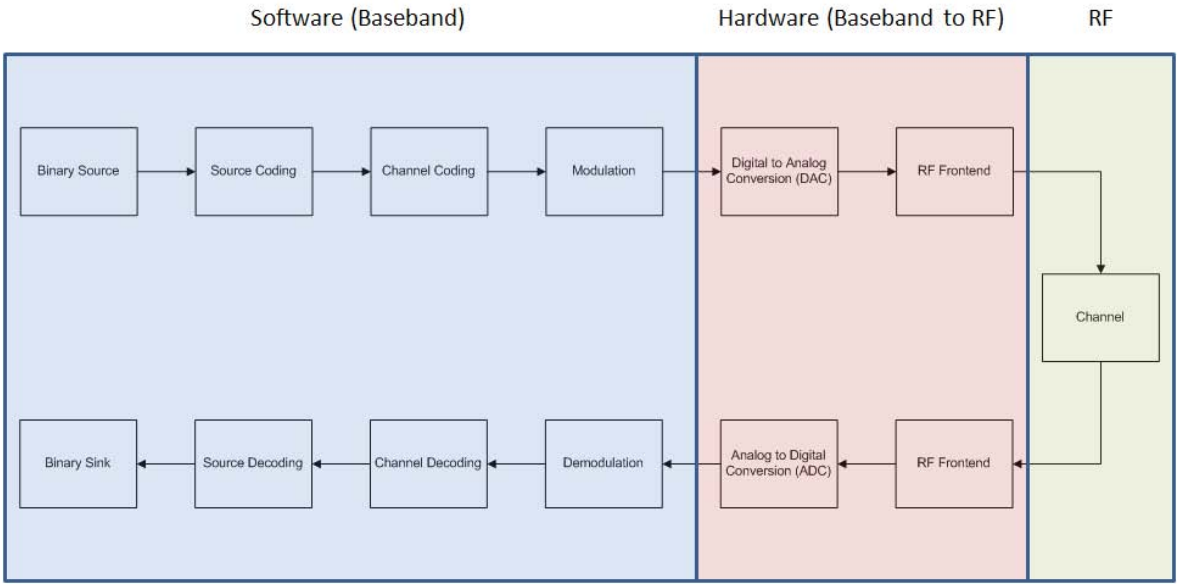


Figure 5 - Software Defined Radio block diagram of a transmitter and receiver.

As software radio concepts become more advanced, the trend is to move digital signal processing aspects closer and closer to the antenna, both to support radio flexibility and to drive down costs of expensive RF stages. Alternatively, another solution would be to design a completely flexible software controlled RF front end. Both approaches have their limitations, so current solutions lie

somewhere in between the two. Practical software radios keep signal processing in the digital domain as much as possible [**9**]. A typical approach for receivers is to digitize the signal at the intermediate frequency stage of a super heterodyne receiver, so the signal travels through the antenna, is filtered, amplified, and mixed with a local oscillator. After this stage, the signal is fed into an analog to digital converter, where the digital baseband processing is done on the sampled signal. This approach is advantageous because it is often cheaper and easier to do signal processing in the digital domain, as costs for microprocessors and FPGAs continue to drop while they increase in processing power.

The number of applications for software defined radio is nearly endless; since it can be applied anywhere there is a need for greater spectral flexibility or interoperability between radio modes. Software radio has already had success in the military, where people's lives depend on effective communications. For example, a military program called SPEAKeasy employs many aspects of software radio to provide communications interoperability between 10 different types of military radios. The SPEAKeasy radio is "an open architecture, simultaneous multichannel, multiband, multimode software programmable/re-programmable, networked and secure radio system that operates continuously and contiguously in the radio spectrum from 2 MHz to 2 GHz." [**10**]

Software radio has also seen some success in the consumer market, especially in the mobile phone sector. Vanu's Software Radio is the first wireless infrastructure solution to provide cellular base stations the ability to simultaneously operate in GSM, CDMA, and iDEN modes. Clearly, this technology has the power to vastly change how cellular networks operate, provided better roaming coverage at different cell sites and increasing interoperability between different carriers and service types. [**7**] As DAC/ADC sample rates and processing speeds continue to improve, it is only a matter of time until software defined radios break into the consumer market to provide better features and services to the consumer.

2.2 Cognitive Radio

The term cognitive radio was first coined in the year 2000 by Joseph Mitola in his doctoral thesis entitled "Cognitive Radio: An Integrated Agent Architecture for Software Defined Radio." [**11**] According to Mitola, cognitive radio technology is the "intersection of personal wireless technology and computational intelligence." Mitola also defines a cognitive radio as "a really smart radio that would be self-aware, RF-aware, user-aware, and that would include language technology and machine vision along with a lot of high-fidelity knowledge of the radio environment." [**12**] Cognitive radio clearly goes hand in hand with software defined radio; together, they can efficiently manage communications to provide greater throughput and efficient use of the RF spectrum.

Technological advances in cognitive radio have occurred due to the limited resource available in the RF spectrum. As the RF spectrum gets increasingly crowded by various users, the frequency allocation approach used by the FCC begins to show its limitations. According to the FCC's Spectrum Policy Task Force, spectrum policy is not keeping up with increasing demand on the

market. [**13**] Cognitive radio offers a solution to this problem. In order to decrease interference between different users of the spectrum, the FCC has a policy of assigning different frequency bands for different services. Unfortunately, this is not very efficient, with usage not spread evenly across the spectrum. Cognitive radio, along with software defined radio, offers the solution of spectrum sensing, where a smart radio can change a device's frequency and mode of operation to provide efficient use of the spectrum and more effective communication between users.

2.3 RF Spectrum

Methods for accessing the RF spectrum have evolved greatly over time. Since its accidental discovery by Heinrich Hertz in 1887, the overall application for the spectrum has been to relay information over distances too long or inconvenient for wires [**14**]. There has always been a need to move information between two points. However, an increasing amount of information is transmitted every day since it is becoming easier to do. With many people all transmitting at the same time, using noisy transmitters, the spectrum soon became very crowded. As an answer to this issue, a new technology was developed. By making use of a previously invented piece of technology, the bandpass filter, the FCC began dividing up the spectrum because the new filter allowed people to transmit and receive on isolated parts of the spectrum. It does this by removing all frequencies except the desired one before the signal is amplified and transmitted. The filter removes all frequencies above a certain limit and all frequencies below a certain limit, leaving a gap in the middle. This basically makes it a high pass and a low pass filter that have been stuck together. The gap in the middle is the desired frequencies, called the pass band.

The FCC has carefully divided the spectrum by application. Since there are already so many different applications for radio, the spectrum has quickly become extremely crowded. As a result, the spectrum has been divided to the point where certain parts of cannot be divided again to allow new applications to have their own frequency allocations. Figure 6 shows a small part of the spectrum. This portion includes 2.4 GHz Wi-Fi, Bluetooth, cordless phones and even microwave ovens.

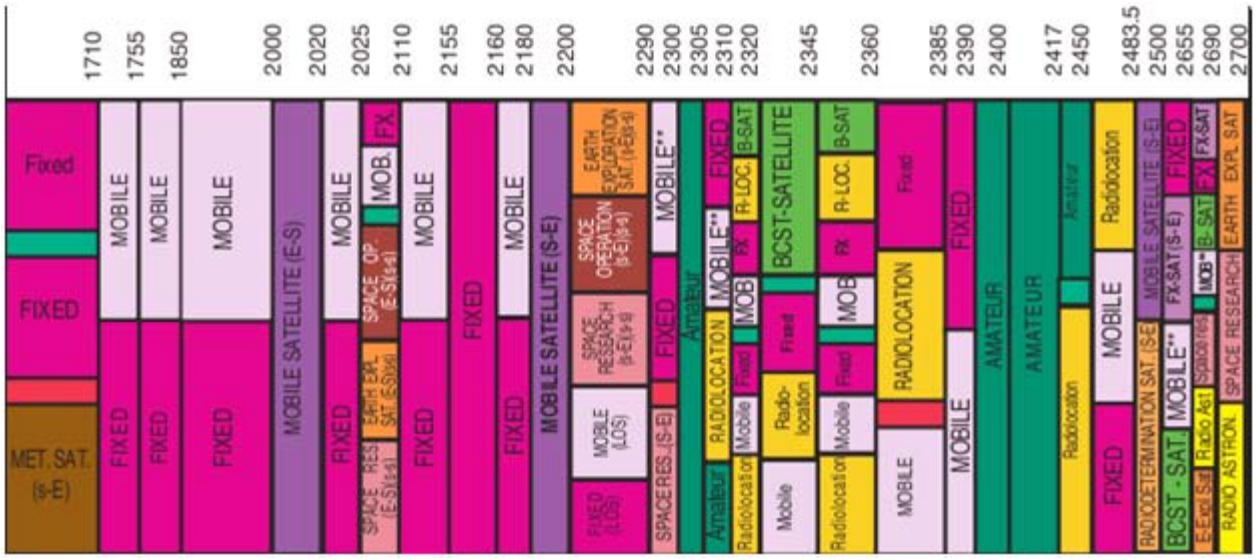


Figure 6 - 1.7 GHz to 2.7 GHz on the frequency allocation chart

The FCC is a U.S. Federal Agency, which means that their frequency allocations are defined by law. In 2005, the Federal Government authorized the FCC to fine stations up to $325,000 for misuse of their licenses [**15**]. Since there are strict rules about what can be transmitted on which frequency, the FCC has created a set of frequencies that anyone can use. Examples of services operating on these frequencies are citizens band (CB) radio, the family radio service (FRS) and the industrial, scientific and medical (ISM) bands. All of these services may be used by any person in the United States without a license. These services are very limited, both in the amount of power they are allowed to use and the number of available channels. According to Part 18 of the FCC rules, the ISM band is open for non-licensed use in the United States. However, communications devices must accept any interference they receive from other ISM equipment and may not intentionally cause such interference.

On the other hand, amateur radio is an option available to people who want to get licensed. Anyone interested is allowed to pay a fee to take a certification exam. This allows them to transmit with more power than the other services and on many more frequencies. However, the station operators are still limited in what they are allowed to transmit on the bands. [**16**].

In a different light, since cognitive radios are much less limited in what they can transmit and what band they are allow to transmit on, there are now people trying to encourage the use of cognitive radios. These are radios which are aware of their surrounds. They know which frequencies are being used and which ones are open. This allows them to make the best use of the available spectrum at a given time. The problem is that this model violates the current FCC rules by operating on unauthorized or private frequencies. Cognitive radios, along with a spectrally agile band plan would allow the general public to use any part of the RF spectrum for any purpose. The difference from the current model is that a cognitive radio would keep things

organized automatically, opposed to the fence-out model currently being employed by the FCC, where access to portions of the spectrum depends on license or ownership. [**17**]

2.4 RF Hardware

Typical RF frontend hardware is tuned for a narrow range of frequencies. The various filters and components used thus will only operate correctly over that narrow range of frequencies that they are designed to operate on. If a certain application needs to operate on several different radio frequencies, the typical approach is to use multiple RF frontends and switch between them. However, this approach is not ideal, especially for cognitive radio, because it requires an entire RF frontend for each desired narrow range of frequencies. Ideally, a cognitive radio should be able to tune itself to any frequency over a wide range, to facilitate the most efficient communications based on channel conditions [**18**][**19**].

Most narrowband RF frontends today use a heterodyne receiver design. This converts the RF signal to an IF (intermediate frequency) signal. This is useful because this uses a process called heterodyning, which mixes the incoming signal with a local oscillator to easily step it down to a more manageable frequency. However, the problem with this design is that it actually shifts two signals to the IF signal: the desired RF signal as well as another frequency called the image frequency. Because of this it requires an image rejection filter to remove the image frequency. For a receiver that must be tunable over a wide range of frequencies, this design would require the image rejection filter to be tunable over that range. [**20**]

Another design is called a direct conversion receiver. This receiver does not require a local oscillator and mixer to help reduce the frequency of the signal. Rather than converting the RF signal to an IF signal, it converts the RF signal directly to baseband. The approach does not shift an image frequency to baseband, and thus does not require an image rejection filter. Therefore this design is more attractive for wideband RF frontend. [**20**] [**18**]

To convert the RF signal to baseband, a LO (local oscillator) is required at the frequency of the desired RF signal. If the RF frontend is to be tunable over a wide range, its LO must also be tunable over this wide range. There are a several ways to create a tunable LO. Typically a VCO and PLL are used. A VCO is an a Voltage Controlled Oscillator, which works exactly as one would think; raising the voltage increases the frequency of the oscillator and lowering the voltage decreases the frequency of the oscillator. A PLL is used to keep an oscillator from drifting out of phase by locking it to the set frequency. Numerically (sometimes called digitally) controlled oscillators also exist. While a single tunable oscillator may not cover the entire frequency range, it is possible to combine a few together followed by a programmable frequency division stage to cover the entire desired range. This is the approach taken in RFMDs RF2051 IC, which is what will be used in the design. [**18**][**19**]

3 Simulation and Prototyping

The application of simulations in the design of this project was limited because of the complexity of the overall design. Simulations of the amplifier and antenna switching stage were utilized to gain a better understanding of the design, but due to sensitivities in component values, the application of these simulations was limited. Instead, evaluation boards were used to create a prototype of the final design to better understand how various components would work in the overall system.

Once the prototype was tested and functional, the final design was laid out in PCB design software. Since every element of the design was not included in the prototype, it was expected that most of the testing and debugging would occur after completion of the PCBs, with an expected revision of the PCB design. Due to budget and time constraints, the second PCB was also the last.

3.1 Amplifier Design Simulation

Since the overall design of the project could not realistically be prototyped, for example, on a breadboard, simulation software had to be utilized wherever possible in order to ensure the design would match the specifications. Agilent's Advanced Design System (ADS) software was used for the simulation of the amplifier circuits in the design.

To simulate the amplifier design, simulation data for the integrated circuits had to be imported into ADS. The antenna switch, the low noise amplifier, and the power amplifier all had simulation data that could be imported into ADS. A design kit was included for the low noise amplifier, MGA82563, so it could be directly imported as a component in ADS. The antenna switch, HMC174M58, and the power amplifier, GVA-84+, both came with .snp files, which is essentially a text file that can be imported into ADS that contains measured s-parameter information for a range of frequencies. The antenna switch came with an .s3p file, which indicated that there are measurements for 3 ports, while the power amplifier came with a .s2p file, which indicates that there are 2 ports. After importing the files for the integrated circuits, the other discrete components in the circuit had to be added. It was a simple matter to simulate the receiver once the transmitter was simulated, since they have the same low noise amplifier.

After implementing the simulation files in ADS, other components, such as resistors and capacitors, needed to be added as well. ADS has these components in its library, but they are ideal. Regardless, substituting these components with design kit models from Coilcraft did not significantly change the results of the simulation. After all of the design was laid out in ADS, the s-parameter simulation simply had to be run to determine what is essentially the frequency response of the system, S21.

Once the frequency response was determined, the simulation results were used to tweak the values of the discrete components surrounding the ICs to achieve the best frequency response. To do this, information in the datasheets for the power amplifier, linear amplifier, and antenna

switch were taken into account, along with comparisons to the design of other daughterboards for the USRP. It seemed that the slightest change in capacitance or inductance in the surrounding circuit had a great impact on the forward voltage gain of the entire system, so the usefulness of these simulations had to be taken into account, given that the capacitive effects of PCB traces could not be simulated. Regardless of the shortcomings of the simulations, components were chosen that matched the simulation circuit as closely as possible. The following figures show the final version of the amplifier circuit, along with the simulation results.

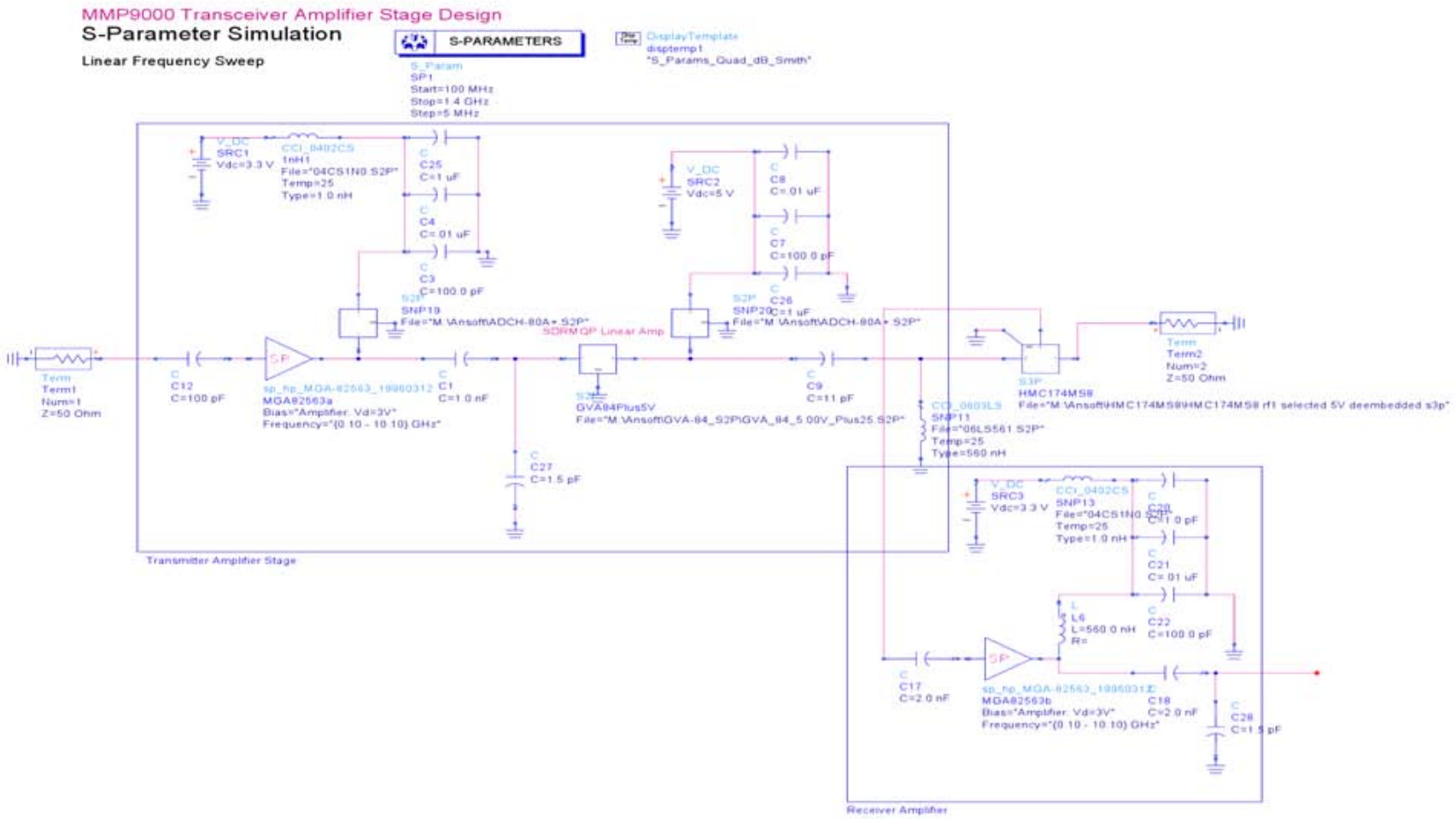


Figure 7 - MMP9000 amplifier stage simulation schematic for both transmit and receive amplifier sections of the design. The transmit side includes the MGA82563 low noise amplifier and the GVA-84+ 5 V power amplifier, while the receive side incorporates only the low noise amplifier. Also included are the DC biasing circuits to power the amplifiers and the antenna switch. The input and output of the amplifier section are both connected to 50 Ω terminators to match the impedance of the rest of the circuit.

3.2 Basic Prototype - RFMD RF2051 Evaluation Board and GVA-84+ Amplifier

RFMD was generous enough to donate an RF2051 evaluation board to the MQP. The RF2051 chip is capable of synthesizing a local oscillator frequency between 300 MHz and 2500 MHz. [**21**] Two RF mixers are also integrated into the chip that can mix an RF signal between 50 MHz and 2.5 GHz with the local oscillator. The RF2052 is almost identical to the RF2051 except that it has only one mixer. The RF2052 is what was used in the final design. Its exact function in the design is discussed later. Figure 8 shows the evaluation board used in prototype testing and experiments, while Figure 9 shows the evaluation board for the power amplifier.

Figure 8 – RF2051 evaluation board.

Figure 9 - GVA-84+ power amplifier evaluation board.

3.2.1 Experimentation with RF2051 Evaluation Board

When the RF2051 evaluation board arrived, experiments were performed to test how well it worked with receiving and transmitting RF signals over the air with the USRPs. The first test setup used the RF2052 evaluation board to transmit a signal that the RFX2400 daughterboard could receive. The second test setup used the RF2052 evaluation board to receive a signal that the RFX2400 daughterboard sent.

3.2.2 Upconversion of Basic TX Board Output from 30 MHz to 2.4 GHz

For the first setup the output from the Basic TX board was connected to an RF input on the RF2052. The associated RF output on the RF2052 was then connected to the GVA-84+ amplifier, which was in turn connected to an antenna. The local oscillator in the RF2052 was set to 2.47 GHz. GNU Radio on the transmitting computer was then programmed to transmit the Mario 2 Overworld theme in FM at 30 MHz with the Basic TX board. The receiving computer was setup to receive FM audio at 2.5 GHz with the RFX2400 board. It successfully received the

Mario 2 theme song. When the local oscillator was tuned to 2.37 GHz, the receiving computer had to be tuned to receive at 2.4 GHz, as expected, proving that the RF2052 could be used to set the transmit frequency. Due to the way mixers work, the receiving computer could also be tuned to 2.44 GHz and 2.34 GHz for the local oscillator frequencies of 2.47 GHz and 2.37 GHz respectively.

3.2.3 Downconversion of Basic RX Board Input from 2.4 GHz to 30 MHz

For the second setup an antenna was connected directly to the RF input of the RF2052 and the RF output was connected to the input of the Basic RX board. The local oscillator frequency in the RF2052 was set to 2.4 GHz. Then again the Mario 2 Overworld theme was transmitted, this time at 2.43 GHz with the RFX2400 daughterboard on one computer. It was received successfully at 30 MHz with the Basic RX board on the other.

3.2.4 Upconversion and Downconversion on the Two Evaluation Board Mixers

The final test involved mixing the signal up to a certain frequency on one mixer on the evaluation board, then mixing this higher frequency signal back down on the other mixer, to connect two USRPs over the air using the evaluation board. USRP 1 was connected to the first mixer, which mixed the signal to 900 and 960 MHz. The signal then passes through the amplifier and is transmitted over the air to an antenna connected to the second mixer input, which mixes the signal back down to baseband for reception on the second USRP's Basic RX board. This test especially demonstrates that the RF2051 and RF2052's usefulness for over the air transmission over a wide frequency range.

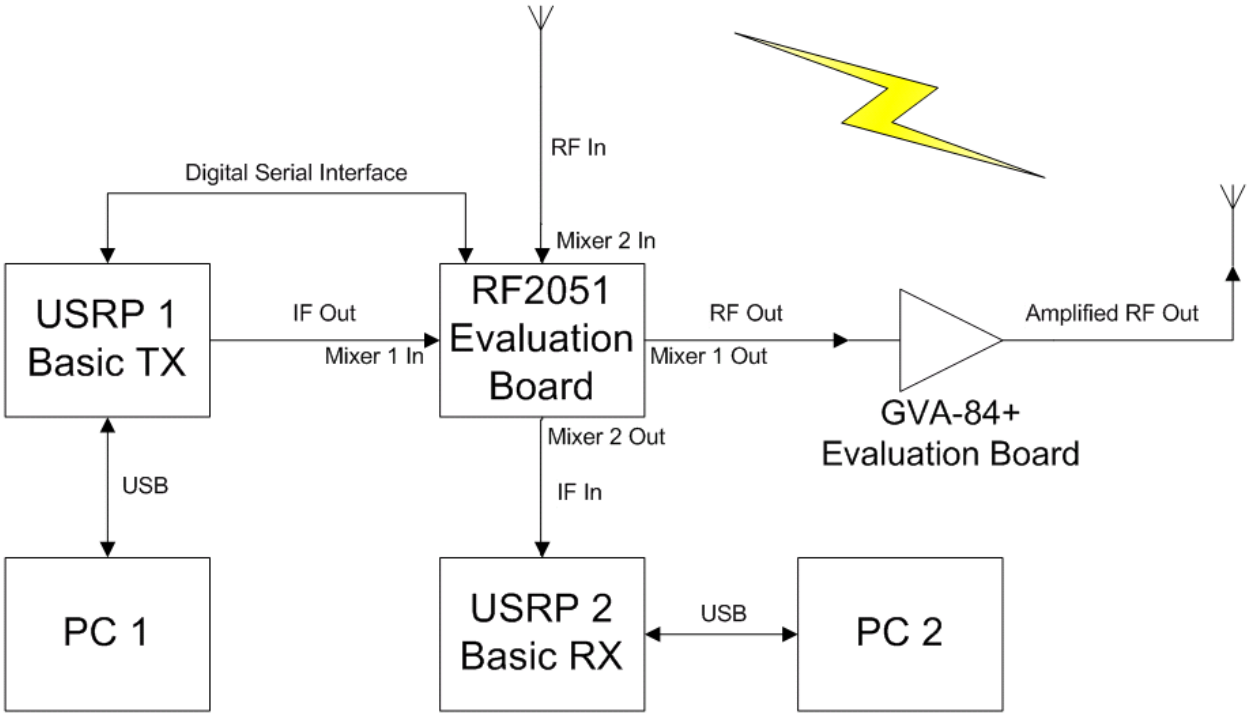


Figure 10 - Mixing a signal up and then back down using the RF2051 evaluation board.

3.2.5 Evaluation Board Test Results

Figure 11 through Figure 13 show a 30 MHz signal from the USRP being mixed with a local oscillator at 930 MHz. In Figure 11 the output is connected directly to the spectrum analyzer. The local oscillator frequency leaks through a bit and can be seen at about -45 dBm. The expected up and down converted signals at 960 MHz and 900 MHz are both seen at about -12 dBm. The local oscillator frequency is seen at about -45 dBm, while the transmitted frequencies at 930 MHz ±30 MHz are at about -12 dBm.

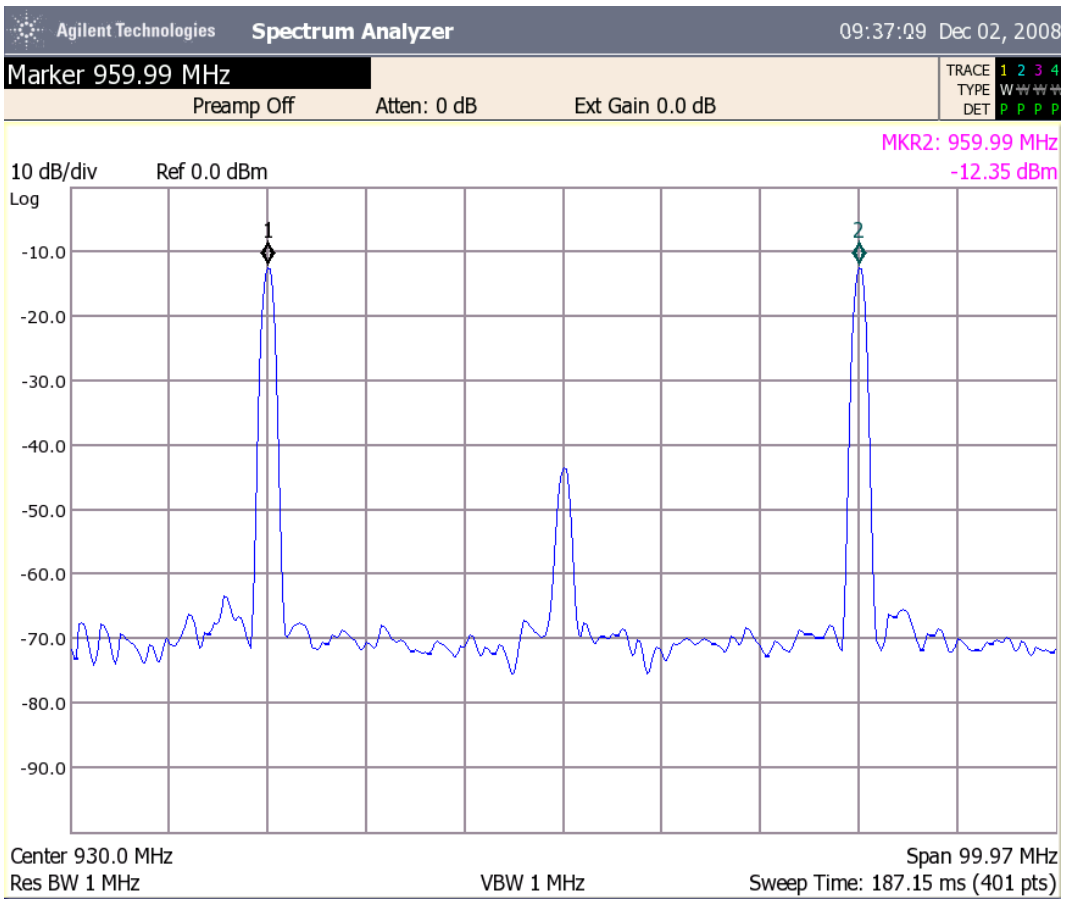


Figure 11 – Transmitted signal directly out of the RF2051 mixer.

Figure 12 illustrates the same setup as Figure 11, but this time utilizes the GVA-84+ amplifier after the evaluation board. The local oscillator frequency is seen at about -24 dBm, while the transmitted frequencies at 930 MHz ±30 MHz are at about 8 dBm. As can be seen in the figure, there is a slight amount of spectral leakage next to the two signals.

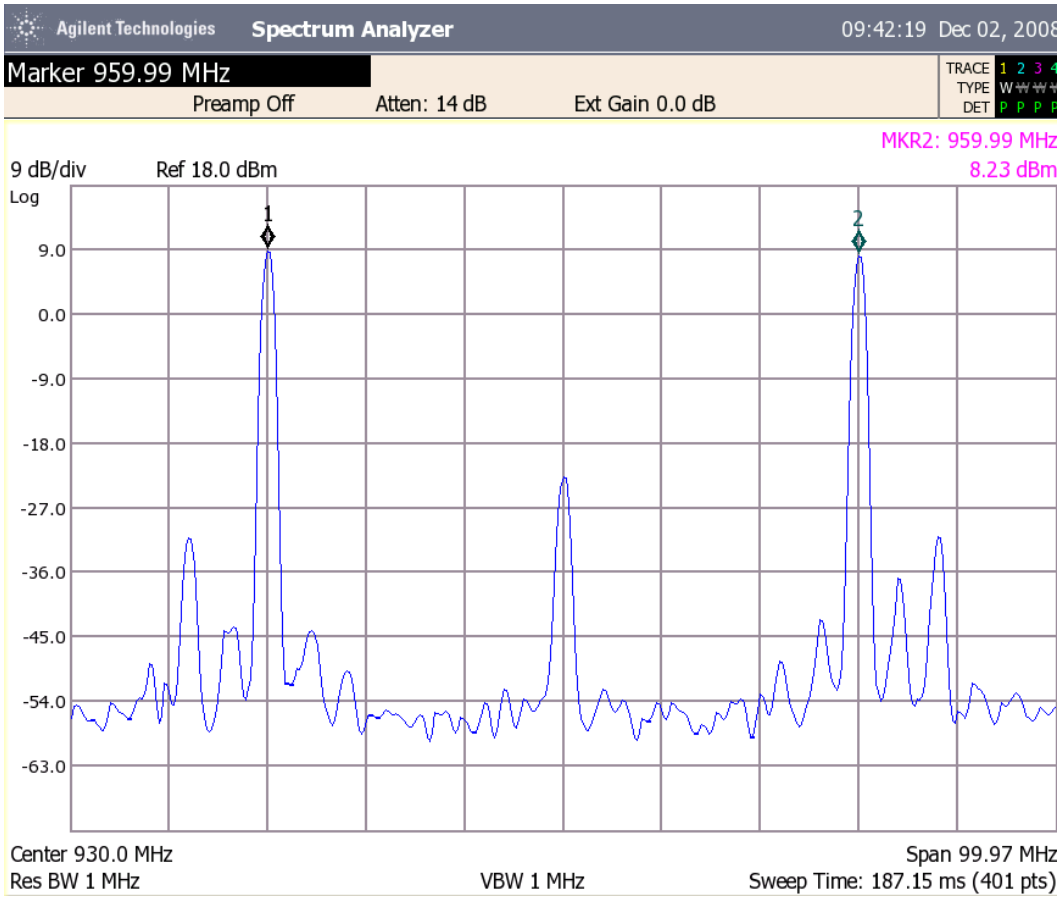


Figure 12 - Transmitted signal directly out of the GVA-84+ amplifier.

Figure 13 again shows the same signal, but after being transmitted over the air with the GVA-84+. The local oscillator frequency is not visible, while the transmitted frequencies at 930 MHz ±30 MHz are at about -42 dBm.

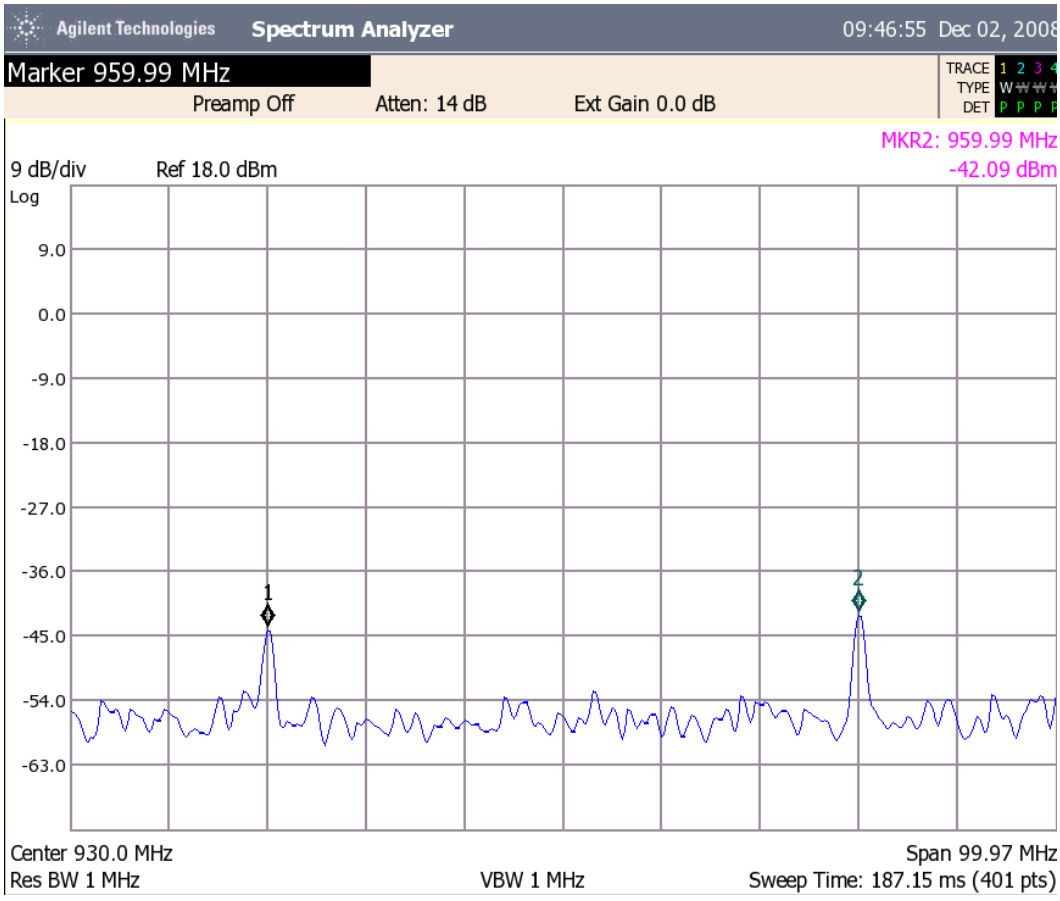


Figure 13 - Transmitted signal amplified by the GVA-84+ and transmitted over the air.

Figure 13 is significant because it shows that the desired signals at 900 MHz and 960 MHz are visible at the receiver, while the unwanted local oscillator frequency is not. This means that it should not cause any significant amount of interference at the local oscillator frequency, but successfully transmit the signal at the desired frequency. However, since there are two copies of the signal, both above and below the local oscillator frequency, the undesired one should be filtered out before the antenna.

3.3 Interfacing the RF2051 with GNU Radio

The next objective was to figure out how to program the RF2051 with GNU Radio. In the experiments documented above, setting the frequency of the local oscillator in the RF2051 was done by programming it over a USB connection to the evaluation board with software provided by RFMD. For the chip to be useful in the design, it would have to be able to be programmed by GNU Radio while connected to a USRP.

The datasheet for the RF2051 describes the serial programming interface as a “Three-wire Serial Control Interface.” Initially it was thought that this would be compatible with Serial Peripheral Interface (SPI). Since the USRP has a SPI bus available to the daughterboards which is easily programmable in GNU Radio, it was thought that it would be no problem to program the chip. Unfortunately, upon further inspection of the protocol described in the datasheet, it was discovered that it was in fact not compatible with SPI. The physical layer is closer to the I^2C protocol (in that they both only use a single data line) which the USRP also has available to the daughterboards, but it is not exactly the same. In the end it was decided that the serial interface should just be connected to the general purpose digital I/O (GPIO), of which plenty is available, and then simply bit-banged by GNU Radio in a Python program. While it is very likely possible to do this much more intelligently in either a lower level C++ module of GNU Radio or even in the FPGA itself, the Python solution worked and is fast enough.

The first thing that had to be done to begin the testing of programming the RF2051 with GNU Radio was to connect the evaluation board to the USRP. A connector was made to connect the serial interface and other digital control pins to the GPIO on the Basic TX board, and also to connect power to a power supply. Figure 4 shows the evaluation board connected to the USRP and power supply.

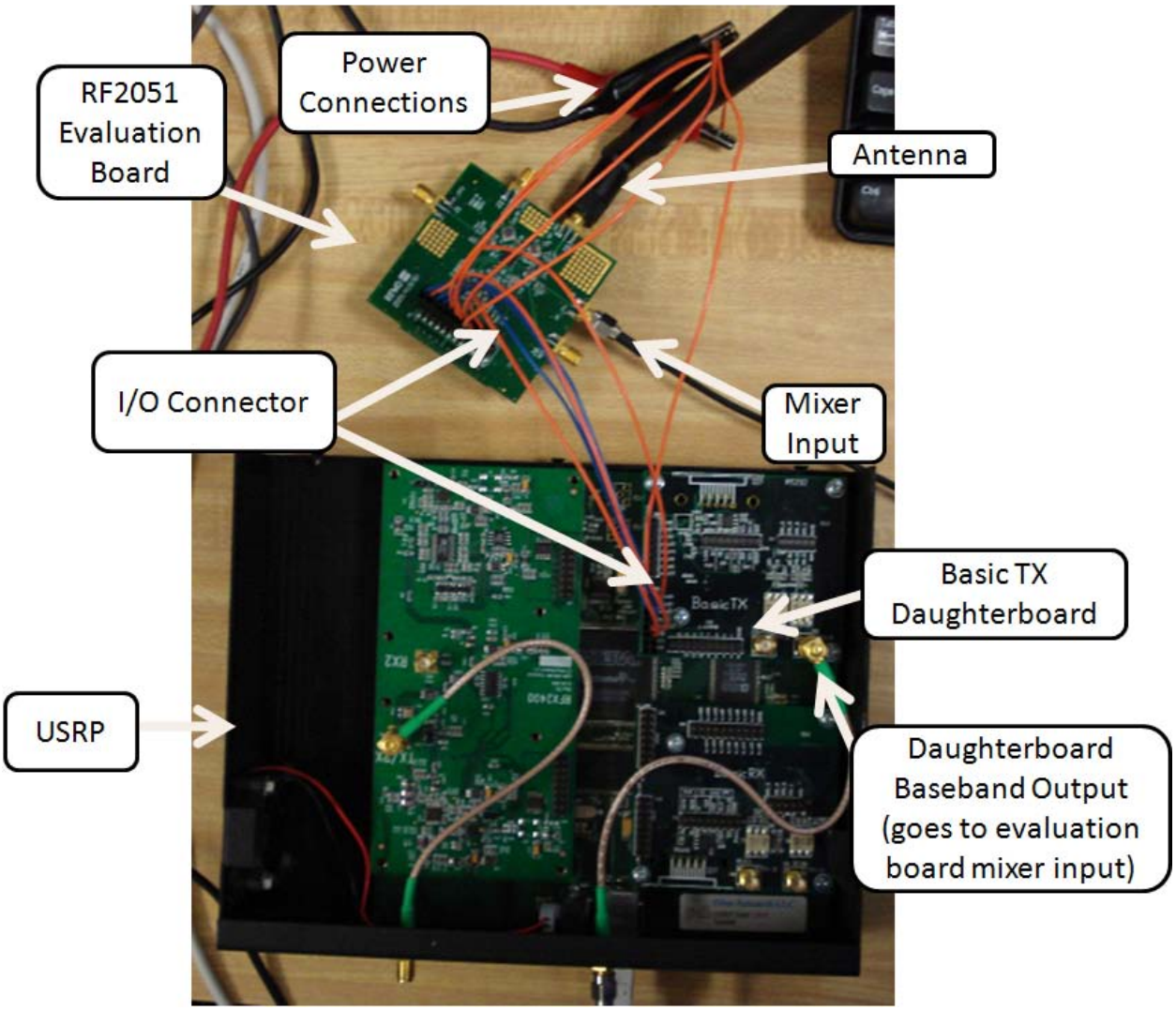


Figure 14 - RF2051 evaluation board connected to the USRP

After carefully going over the read and write timing diagrams in the RF2051 datasheet (shown below in Figure 16 and Figure 15 respectively), two Python functions were written: one to write a word to a register in the RF2051, and one to read a word from a register in the RF2051. These timing diagrams show the voltages on the three serial pins on the RF2051 chip (either high or low) over time for one read or write operation. The ENX input enables the serial interface (active low), SCLK is the clock for the data (driven by the host), and SDATA is the data line (driven by either the host or the RF2051 depending on the operation). The slightly tricky part was making sure that the read function was written correctly before testing it. This is because the data line is used as both an input and an output. The host (USRP) first writes the address of the register on the data line. After a short delay the RF2051 then writes the data in the register to the same data line. During the delay it is important that the direction USRP's serial data pin be changed from an output to an input. If both the USRP and RF2051 try to drive the line at the same time, it could potentially cause damage to the RF2051, USRP's FPGA, or both.

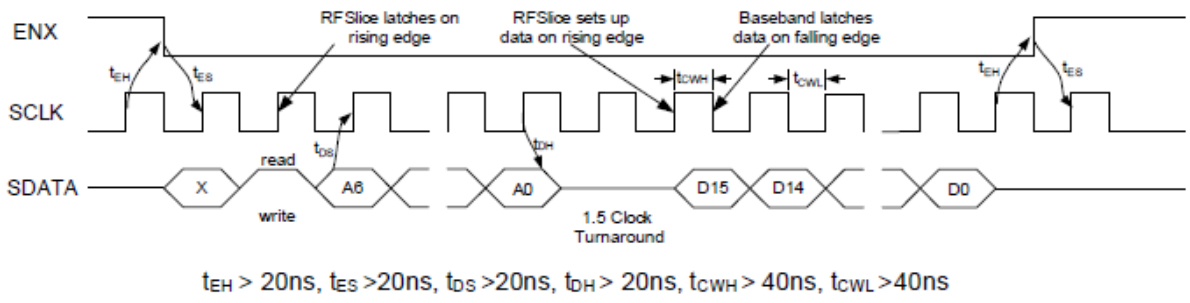


Figure 15 - RF2051 serial read timing diagram

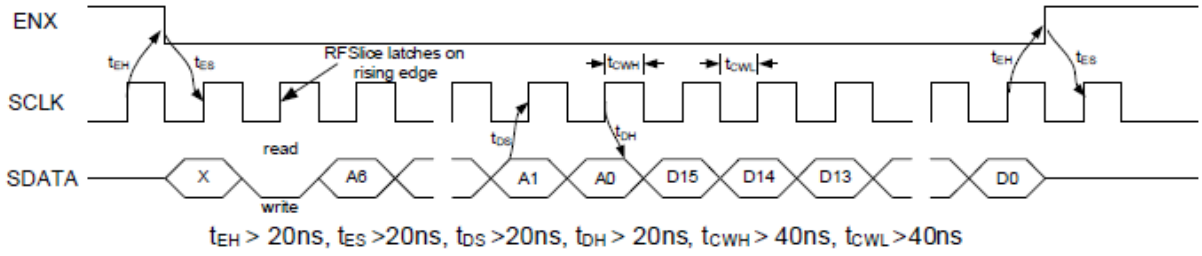


Figure 16 - RF2051 serial write timing diagram

Once it was confirmed that the read and write functions did indeed work (by successfully reading back data that was written) the next thing to do was to write a function to actually tune the local oscillator to any frequency in the range 300 MHz to 2500 MHz. The RF205x Frequency Synthesizer User Guide gave a very nice detailed example on how to program which registers to set the frequency of the local oscillator. Following this information, a function was written that could successfully tune the RF2051's local oscillator to any frequency between 300 MHz and 2500 MHz, which was confirmed by checking the local oscillator leakage on the RF output port on a spectrum analyzer. All the source code of the functions mentioned above can be found in Appendix A - Source Code.

3.4 Chapter Summary

The final design described here is the result of many different contributions. First the simulation data that was previously discussed was taken into account to verify the part selection. The parts were then put together in a schematic layout software and verified against the datasheets and what the manufacturers suggest. Finally, the design was compared against the design of similar daughterboards for the USRP to check for any large inconsistencies. From this final design, the actual assembly of all the components can begin to take place.

4 Transceiver Design

Once the requirements for the design were identified, much research had to be done before the design process could begin. For starters, the board would have to be compatible with the USRP, and the only way to determine what this required was to read through the USRP documentation and any other related resources. After doing some research into this it was discovered that the USRP hardware is entirely open source and released under the GNU General Public License (GPL). Thanks to this, all existing daughterboard schematics, parts lists, and PCB layouts are freely available on the Internet. This information by far provided the most help for the design, as it not only showed exactly how the existing daughterboards work, but also how they interface with the USRP. Of course it still took quite a bit of time and research to fully understand the schematics. Since the Wireless Innovation Laboratory already had a couple of the RFX2400 daughterboards available for experimenting with, it was decided that it would be the schematics for this board that would be analyzed. After many days of reviewing every part of the schematics for it, reading through the datasheets for all its parts, and reading up on general RF receiver and transmitter theory and design, a detailed working knowledge of its operation was gained. It was decided that it would be helpful for future reference to create a block diagram showing the operation of the RFX2400 board. This diagram appears below in Figure 17 through Figure 19.

Once this existing daughterboard design was understood, it was time to begin researching ways to increase its frequency tuning range. First the schematics for the other daughterboards in the RFX series were examined because they all operate over different small frequency ranges. It was found that they all had very similar designs, mainly only differing in some of the ICs used, which even still were ICs in the same families, just covering different frequency ranges. Since none of the ICs in any of the existing daughterboards were sufficiently wideband enough in order to operate over the desired frequency range of 100 MHz to 1.3 GHz, the search began for ICs with similar functions, namely the local oscillator and quadrature modulator and demodulator.

Almost immediately a quadrature modulator and quadrature demodulator, both from Analog Devices, were found that could cover the frequency range from 50 MHz to 2200 MHz for receiving, and 50 MHz to 2000 MHz for transmitting. They were the ADL5385 and ADL5387 respectively. However, it was a much harder task to find a solution for a local oscillator that could be digitally tunable over the 100 MHz to 1.3 GHz range. After conducting more research, it was found that the most common way to implement a tunable local oscillator is with a phase locked loop (PLL) and a voltage controlled oscillator (VCO). In fact this is what is used in the RFX series of daughterboards. Typically the two components (PLL and VCO) are integrated into a single IC. Unfortunately none were found that operated over the desired frequency range. In fact, the vast majority have to be tuned to a center frequency using discrete components which limits the tunable range to a small subset of the advertised range.

Figure 17 shows a block diagram of the initial design of the transmitter side. The two quadrature signals leave the USRP from the digital to analog converters and pass through anti-aliasing low pass filters to smooth out the DAC transitions. The quadrature modulator then will mix and modulate this signal with the local oscillator, which is controlled by the USRP. After being mixed and modulated to a higher frequency, it will then pass through a low noise amplifier and then a power amplifier. The RF signal then travels to the antenna switching block, shown in Figure 18.

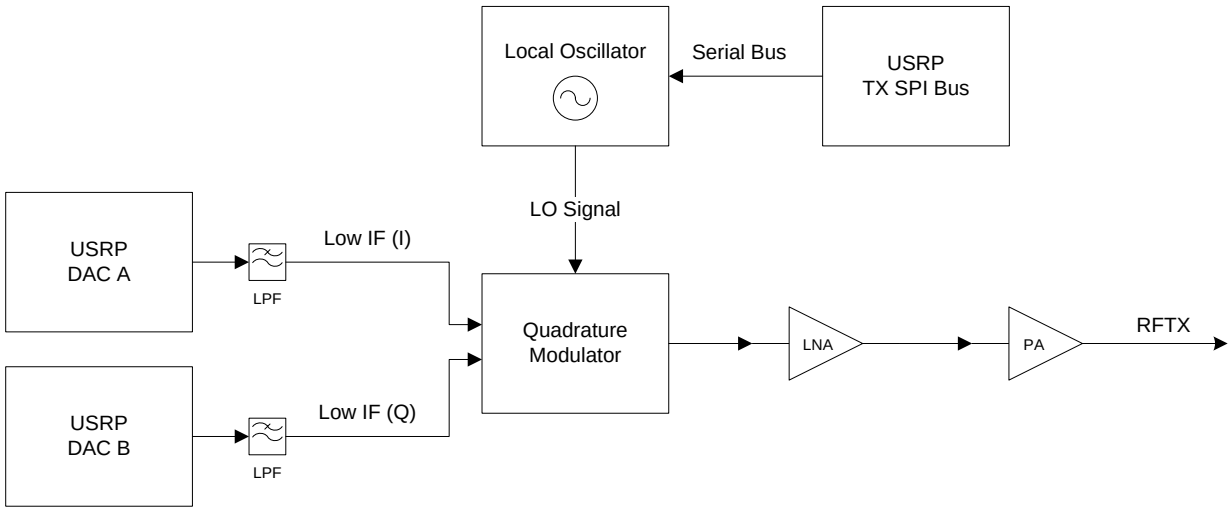


Figure 17 - Block diagram of the RFX daughterboard series transmit path

Once the signal has been modulated, it is passed to the antenna switch and antenna logic block, which are essentially two analog switches that control the antenna connections and signal path. In this stage, the software can control whether the board is operating in full duplex mode, transmitting and receiving at the same time, or half duplex mode, either transmitting or receiving.

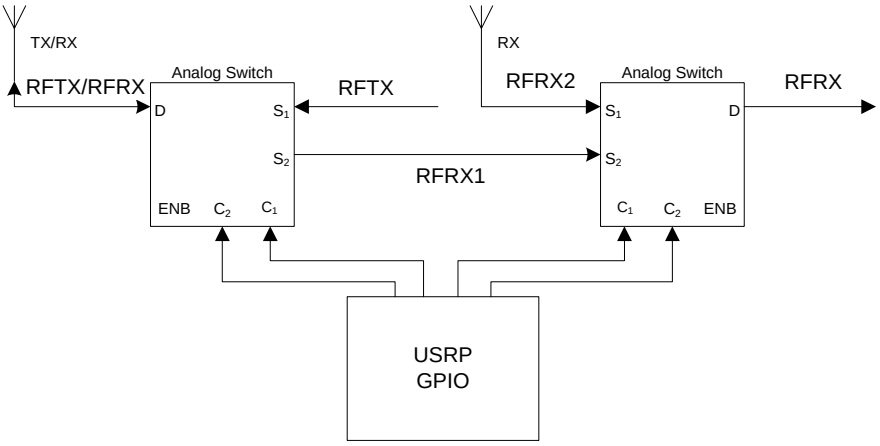


Figure 18 - Block diagram of the RFX daughterboard series transmit/receive switch configuration

The receive path is also connected to the antenna switch and logic block. Once again the USRP GPIO will control which antenna is connected to the receive blocks based on what mode the transceiver is in.

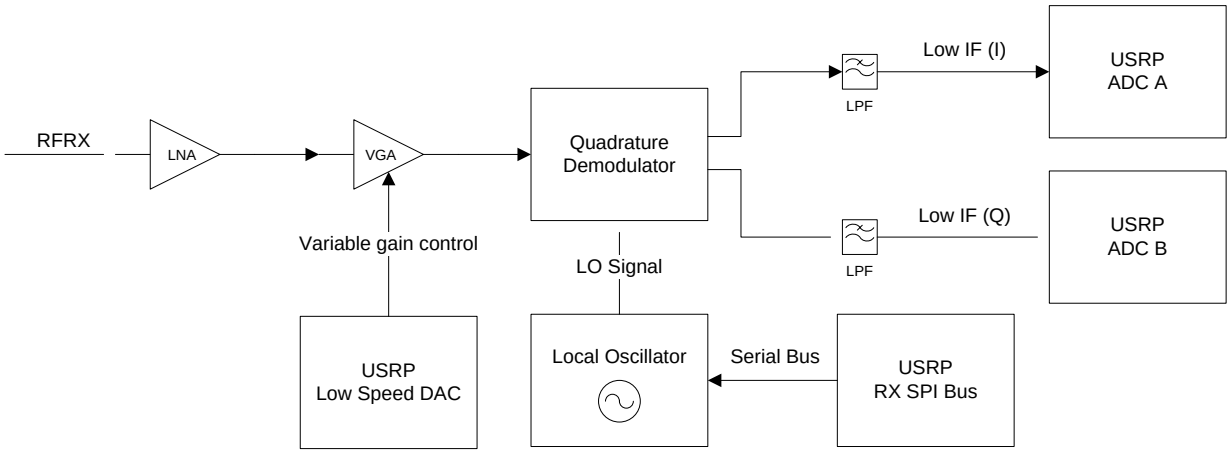


Figure 19 - Block diagram of the RFX daughterboard series receive path

Once the received signal passes through the antenna switches, it is amplified before being mixed down to the correct frequency to be sampled by the analog to digital converter on the USRP. In a similar setup as the transmitter, a local oscillator will be controlled by the USRP SPI bus, which will allow software control over the frequency that the received signal is mixed down to. After exiting the quadrature demodulator, the signal is low pass filtered to remove any noise before it enters the analog to digital converters on the USRP.

One idea to achieve the local oscillator was to combine several VCO/PLL chips and multiplex them, selecting them based on the required frequency. This approach is often used in truly wideband applications where failure or poor operation is not an option. Figure 20 illustrates this initial design idea. Unfortunately this would have required many chips and lots of extra circuitry.

In the real world this uses up precious board real estate, increases power consumption, and drives up costs. Therefore this potential design only ever existed as a vague idea, and never made it to a more complex schematic.

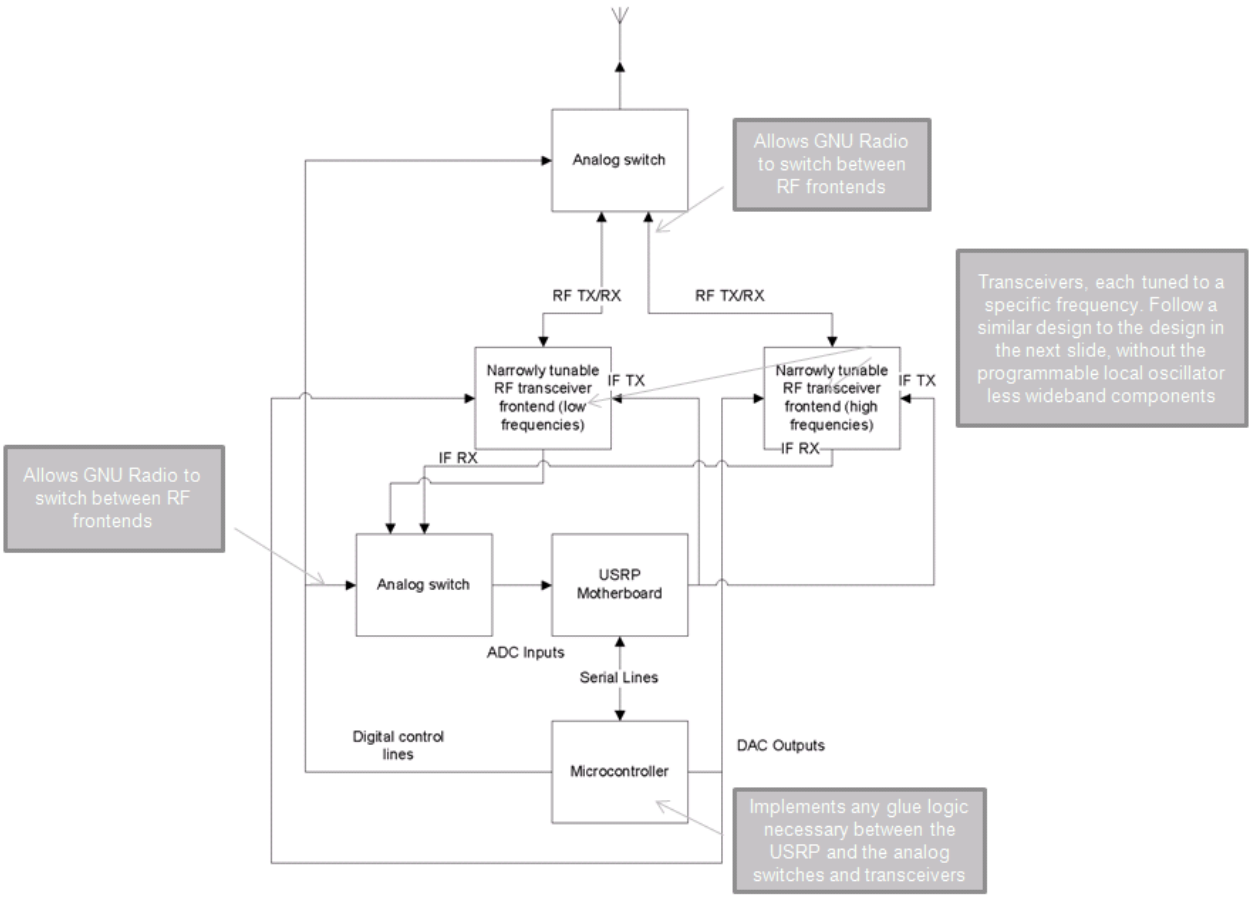


Figure 20 - Initial design idea with switchable, narrowly tunable transceiver frontends.

Eventually RFMD's RF205x series of chips was discovered. The RF2052 in particular includes a VCO/PLL local oscillator block, and one RF mixer. However it was the VCO/PLL block that was of interest, since it is digitally tunable and able to generate any frequency between 300 MHz and 2500 MHz. After reading about it, it was found that by grounding the input to the RF mixer, its output would be the local oscillator signal, which was exactly what was needed. Furthermore, both the quadrature modulator and demodulator require an input of twice the effective local oscillator frequency, meaning that by tuning the external local oscillator to 2000 MHz, the baseband or RF signal is effectively mixed with a 1000 MHz local oscillator signal. Therefore a tunable local oscillator range of 300 MHz to 2500 MHz meant that the design would be able to transmit and receive RF signals between 150 MHz and 1250 MHz, only 50 MHz off from each end of the desired frequency range.

Sometime during working out the details of the design using the RF2052 solely as the local oscillator, some potential problems came to light. The power output from the RF2052's mixer output port with the mixer input grounded could potentially exceed the maximum power input allowed on both the modulator's and demodulator's LOIN ports. While this could be fixed by using resistors to reduce the power between the chips, it would also require additional impedance matching. Another potential problem was the fact that tuning the board to transmit or receive at 1000 MHz would require tuning the RF2052 to 2000 MHz. While at first this may not seem to be a problem, since the RF2052 is easily tuned to 2000 MHz, in reality it very well could be. Designing a PCB to operate correctly at such high frequencies is very difficult, and requires very close attention to correct impedance matching of traces as well as the placement of both traces and components. Otherwise the signal can be reflected back to the source, can be greatly attenuated, and can interfere with other signal and power lines on the board. It would be rather unfortunate then if the board failed to operate at 2000 MHz, but succeeded at 1000 MHz, since the only thing that would stop it from operating at 1000 MHz would be that it did not operate at 2000 MHz.

Aside from these potential issues, the frequency range would be still slightly narrower than what had initially been decided on. While brainstorming about what to do about all of this, a new design was thought up. It required an additional PLL/VCO IC operating at a fixed frequency, or at least only a very narrow range or tunable frequencies, a new quadrature modulator/demodulator pair, and also took advantage of the mixer in the RF2051. An intermediate frequency stage would be added to the transmit and receive paths to fully utilize the capabilities of the RF2052 chip.

4.1 Final Design Overview

The final design is similar to the design of the RFX daughterboard series, but uses an additional intermediate frequency stage and the RF2052 wideband mixer and frequency synthesizer. Figure 21 through Figure 23 below show a functional block diagram of the design. Figure 24 through Figure 26 show the design with the actual components chosen for each block.

On the transmit side of the design, a low IF (intermediate frequency) signal in quadrature (Q) and in-phase (I) components is sent out of two high speed DACs on the USRP. Before anything else, these signals are put through low pass filters to remove aliases and other noise generated by the DACs. These two filtered signal components are then fed into a quadrature modulator. The quadrature modulator takes as inputs the two low IF signal I and Q components and a local oscillator (LO) signal. These inputs are mixed and combined into a single RF output at the frequency $f_{IF} + f_{LO}$. The $f_{IF} - f_{LO}$ frequency is suppressed by the quadrature modulator's circuitry.

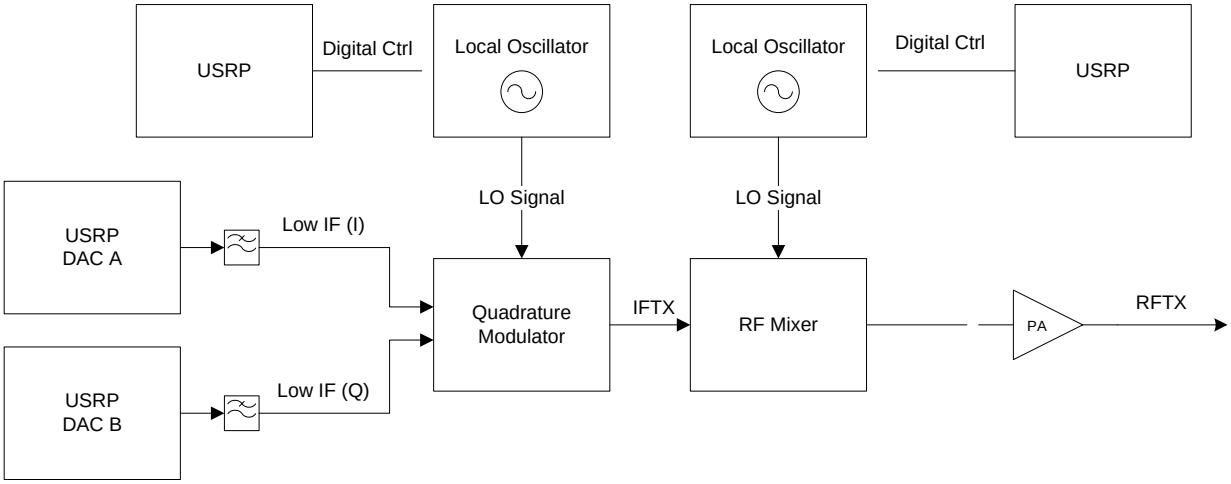


Figure 21 - Block diagram of the final design transmit path

The RF output from the quadrature modulator is used as an IF signal at around 300 MHz in the design, however the exact frequency is digitally tunable over a narrow range from about 280 MHz to 320 MHz. This signal is then fed into a wideband mixer. For this the RF2052 is used. As previously stated, this mixer is capable of mixing the IF signal with a LO frequency at anywhere between 50 MHz and 2500 MHz. The local oscillator itself, which is digitally tunable, is also contained within the RF2052 and is capable of synthesizing frequencies between 300 MHz and 2500 MHz. The output from this mixer is the signal to be transmitted at the desired RF frequency. Before it is sent to an antenna for transmission however, it is first amplified by a power amplifier.

The receive side is very similar to the transmit side, only in reverse. First the RF signal is captured by an antenna and then goes into a low-noise amplifier (LNA). The output from the LNA goes into the RF2052's mixer to mix the signal down (or up) to the IF. The mixer's output

at the IF is then fed into a variable gain amplifier (VGA) which is adjustable via the USRP's low speed DAC. This is to allow the user to adjust the gain of the receiver. After the VGA the signal is fed into a quadrature demodulator, which breaks the signal down into its I and Q components and mixes them down to the low IF signal. Before going into the USRP's high speed ADCs, these signal components are low pass filtered to prevent aliasing of higher frequencies. The design of the antenna switching blocks remains essentially the same as described earlier.

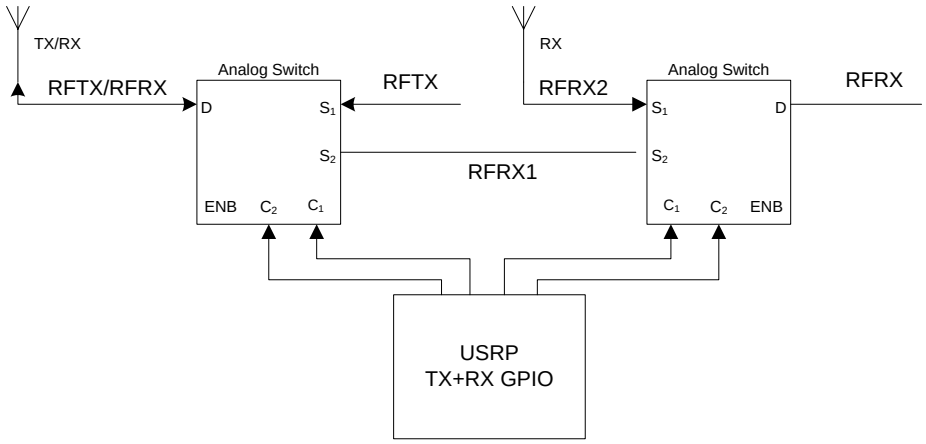


Figure 22 - Block diagram of the final design transmit/receive switch configuration

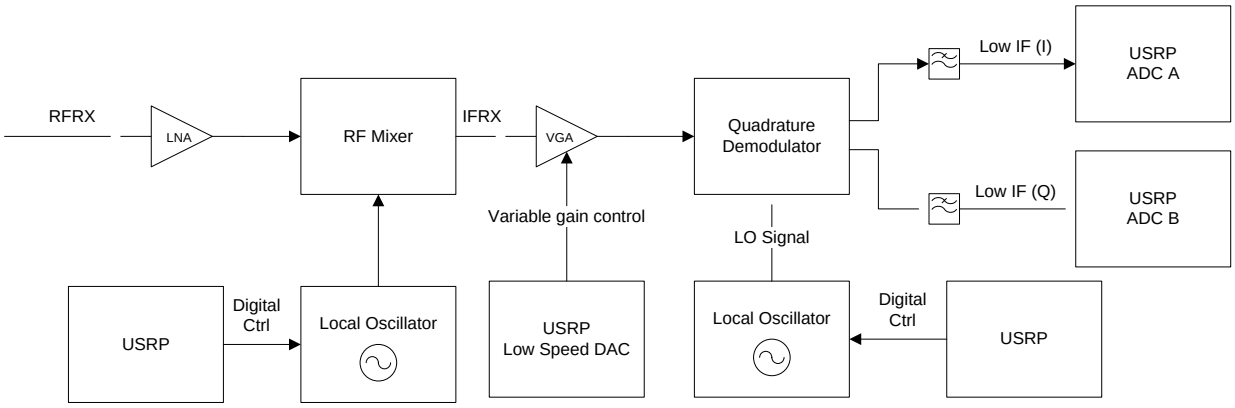


Figure 23 - Block diagram of the final design receive path

Figure 24 through Figure 26 shows the final block diagram design of the entire daughterboard, with chosen components and their functions. A complete schematic of the design can be found in Appendix B – Schematic.

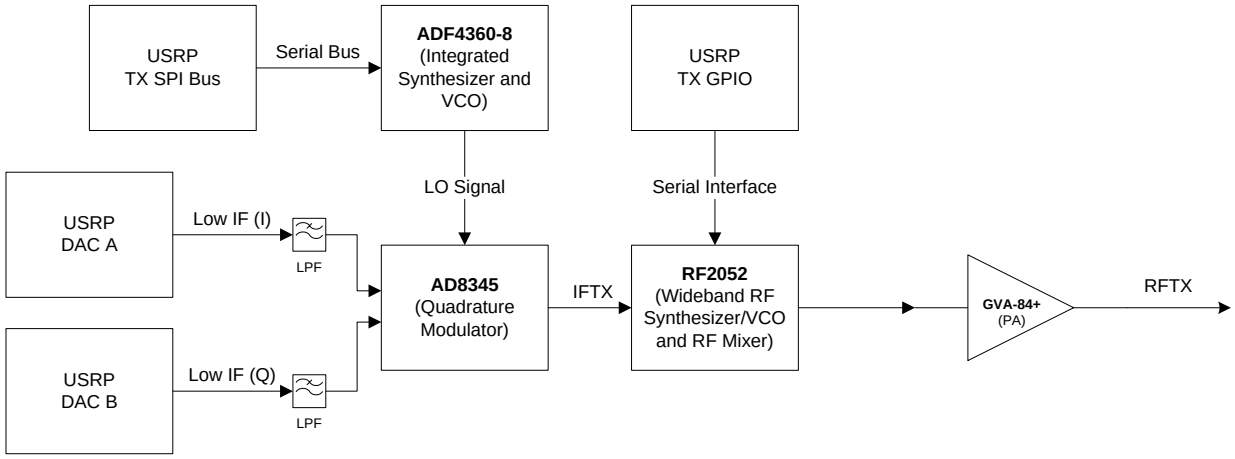


Figure 24 - Block diagram of the final design transmit path with actual components

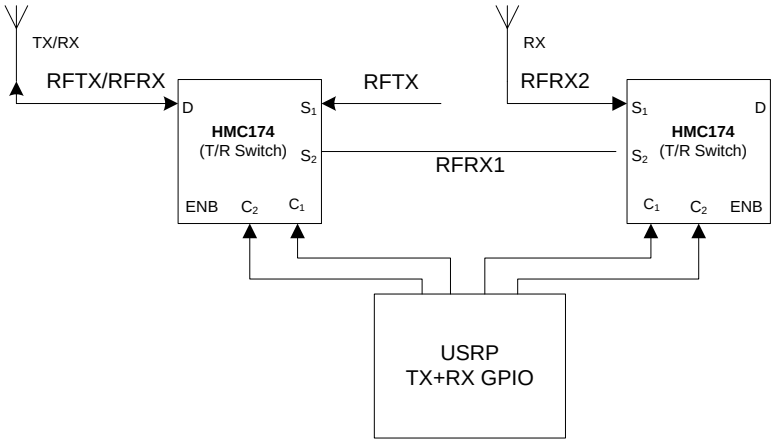


Figure 25 - Block diagram of the final design transmit/receive switch configuration with actual components

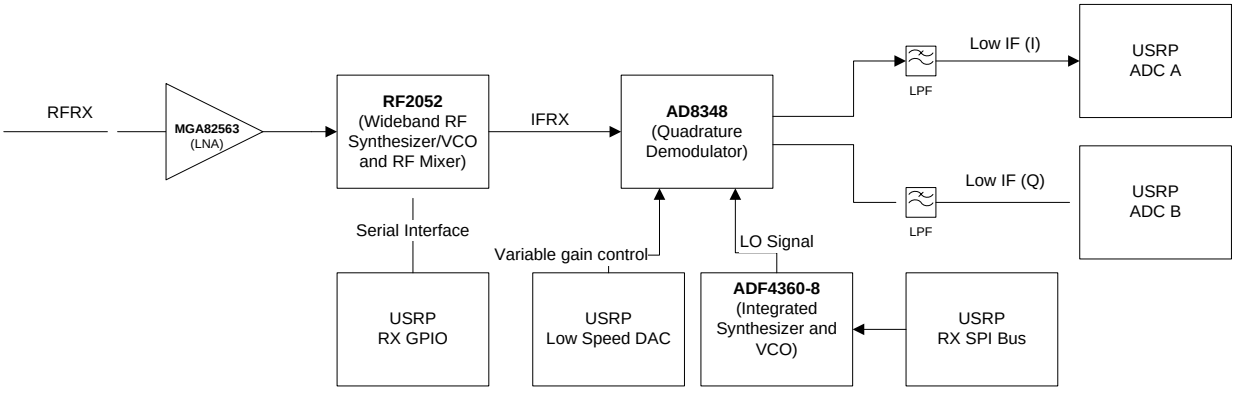


Figure 26 - Block diagram of the final design receive path with actual components

4.2 Final Design Details

This section expands upon the ideas and block diagrams presented in the previous section while going into more detail of the interconnections between various ICs and components of the final design.

4.2.1 Transmit Path

4.2.1.1 Quadrature Modulator

The quadrature modulator is the first step in the transmit path of the design. It is the circuit that upconverts the baseband signal from the USRP that is to be transmitted to a 300 MHz intermediate frequency (IF) signal. The reason a quadrature modulator is used is because the USRP provides each transmit daughterboard with both in phase (I) and quadrature (Q) components of the baseband signal. The I component is created by mixing the baseband signal from the computer with a low frequency (no more than about 30 MHz) cosine wave, and the Q component is created by mixing the baseband signal with a sine wave at the same low frequency. The quadrature modulator on the daughterboard takes these I and Q components, mixes them with a local oscillator frequency provided by another circuit (described in the next section) and combines them into a single IF signal at 300 MHz.

The heart of the quadrature modulator circuit in the design is Analog Devices' AD8345 quadrature modulator. It is capable of operating at frequencies between 140 MHz and 1000 MHz. Implementing this IC into the design was rather straightforward. The I and Q inputs are both differential inputs. This is convenient because the I and Q outputs from the USRP's DACs are also differential. To remove aliasing to higher frequencies, low pass filters were fitted between the DAC outputs and the modulator inputs. The datasheet for the USRP's DACs indicates that they output up to 18 mA full scale current (the difference between the differential output pair), and that they are 64 megasamples per second (MSPS). This means that they can output frequencies up to 32 MHz. The datasheet for the AD8345 says that the I and Q inputs should have a peak to peak voltage of 1.2 V, and have a 0.7 V bias. After referring to the schematic for the RFX400 board (which also uses the AD8345) it was found that the low pass filter shown in Figure 27 below was used. After doing a simulation on this circuit, it was determined that it had a 3dB bandwidth of about 25 MHz, which makes sense for the DACs in the USRP. This AC analysis is shown below in F2. It also converts an output current of 18 mA to 1.2 V, which agrees with the datasheets. Finally, the two 49.9 Ω resistors and the 10 Ω resistor create a DC bias of about 0.7 V on each pin, assuming an average current output of 10 mA. Since this particular low pass filter circuit seemed to agree so well with the information in the datasheets, and also since it was what was used for the RFX400 board, it is what is included in the final design.

The AD8345 is powered with 5 V on two pins, and the final circuit in the design includes two decoupling capacitors (a 1000 pF and a 0.01 uF) per pin.

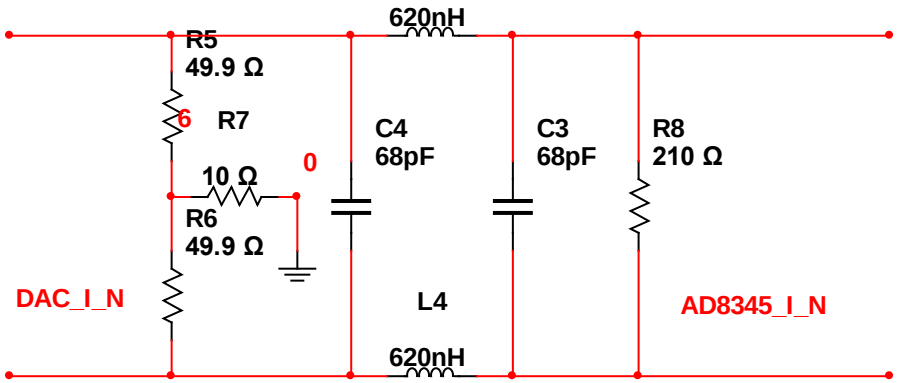


Figure 27 - 25 MHz Low pass filter between DACs and the AD8345 quadrature modulator

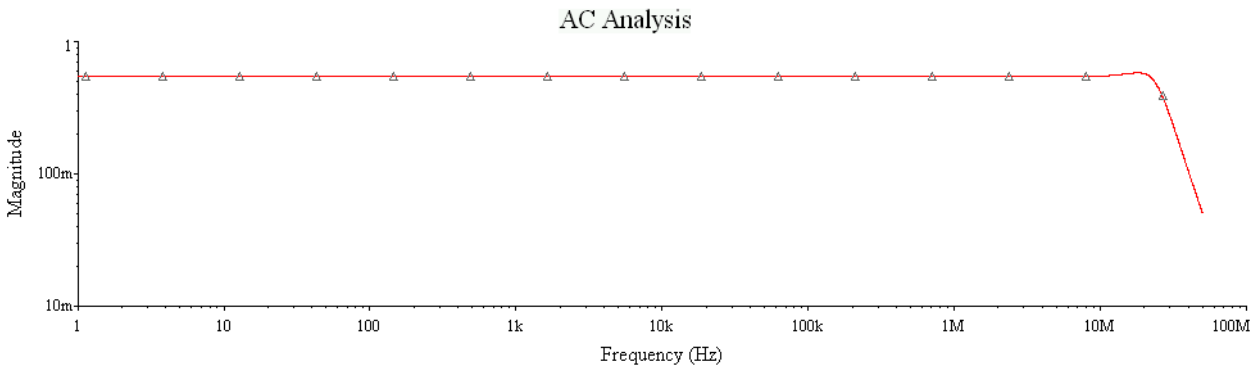


Figure 28 - AC analysis of the low pass filter between the DACs and AD8345 quadrature modulator. The 3dB bandwidth is about 25 MHz. The passband magnitude is about 600 mV, which corresponds to 1.2 Vp-p.

4.2.1.2 Local Oscillator

As mentioned in the previous section, the quadrature modulator needs a local oscillator (LO) signal to mix the baseband I and Q signals with. The central component used for this circuit is Analog Devices' ADF4360-8. This is an "Integrated Integer-N Synthesizer and VCO" with an output frequency between 65 MHz and 400 MHz. The exact output frequency is programmed over a Serial Peripheral Interface (SPI) bus by the USRP and GNU Radio. A center frequency must be chosen and set, however, by external inductors, and an external loop filter must be fitted for the desired output frequency range and reference frequency. Thus the actual output frequency range in any given design is not actually 65 MHz to 400 MHz. To help pick the values for the external inductors and the loop filter design, Analog Devices provides a free program called ADIsimPLL. Given a desired output frequency range, step interval, reference frequency, and

loop filter type, the program generates a schematic with the correct component values. Since a center frequency of 300 MHz is desired, and not much divergence from this is necessary, the software was given the range 280 MHz to 320 MHz at 1 MHz intervals using a 64 MHz reference frequency, which is taken from a crystal on the USRP motherboard. The schematic it generated is shown below in Figure 29. Unfortunately, none of the component values that it calculated are standard values. Thus the closest standard values had to be chosen.

Since the LO outputs of the ADF4360-8 and the LO inputs of the AD8345 are both differential, they can be connected together directly with no matching circuitry other than that shown below in Figure 28. The ADF4360-8 is powered with 3.3 V on three pins, and the final circuit in the design includes two decoupling capacitors (a 1000 pF and a 0.01 uF) per pin.

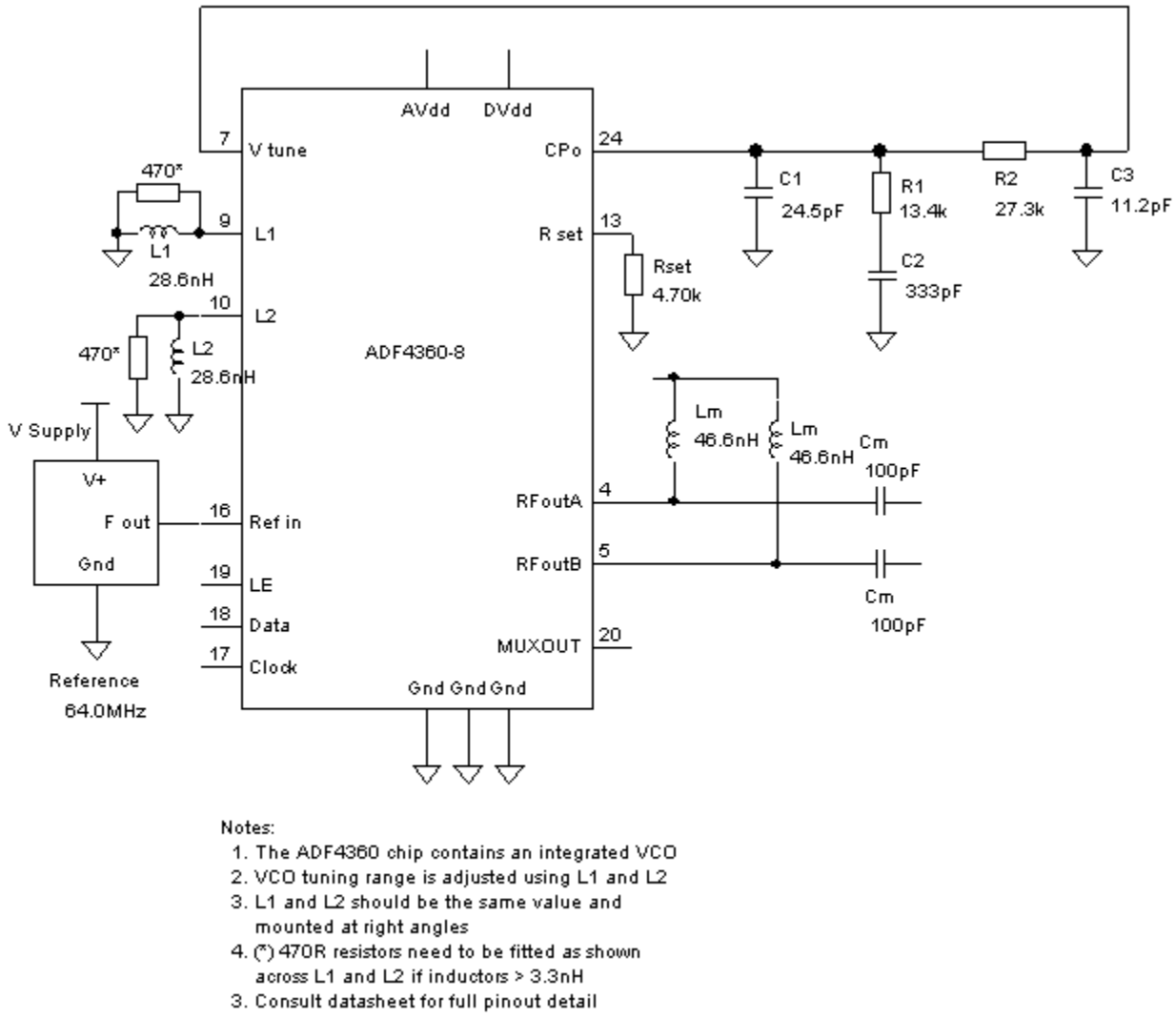


Figure 29 - Schematic for the local oscillator using the ADF4360-8 IC, tuned for 280 MHz to 320 MHz at 1 MHz intervals. Generated by ADIsimPLL.

4.2.1.3 RF Mixer

The output from the quadrature modulator stage is an RF signal at 300 MHz. The final design, however, should be able to transmit an RF signal at any desired frequency between 100 MHz and 1.3 GHz, and potentially beyond this range. To accomplish this, the 300 MHz IF signal is mixed with a LO signal that can be programmed to be any frequency between 300 MHz and 2500 MHz. For this, the RFMD RF2052 IC is used. Since the mixer in the RF2052 mixes the input signal both up and down, the 300 MHz IF signal can be mixed to any frequency between DC and 2800 MHz. However, the mixer ports are rated to operate between 50 MHz and 2500 MHz which limits the range.

The actual circuitry around the RF2052 is a little more complicated than the previous two stages. Both the input and output ports of the mixer are differential. However, the RF output from the AD8345 is single ended with an output impedance of 50 Ω, and the input to the next stage (the amplifier) is also single ended with an input impedance of 50 Ω. Therefore both the mixer input and output ports require external matching circuitry to convert them to single ended 50 Ω lines. The RFMD document "An RF205x Family Application Note Matching Circuits and Baluns" helped while figuring out how to do this. To get the best performance out of the RF2052 mixer, the matching circuits for the input and output ports should be tuned for the specific frequency ranges required for a particular application. Since the widest range possible is desired for the final design, the simplest wideband matching circuits for the output port was chosen. While its performance (in terms of output power over the desired frequency range) does not match the performance of the narrowband matching circuits, it still should show generally good performance over the desired range of 100 MHz to 1.3 GHz. In addition, it was decided to include spots on the PCB for additional resistors, capacitors, and inductors to better match the port for specific frequency ranges if desired. Initially these will not be used by either not filling them, or by filling them with 0 Ω jumpers. For the input port, since the input frequency should always be about 300 MHz, the matching circuit will be tuned to work well at 300 MHz.

The circuits chosen for the input and output mixer ports are shown below in Figure 30 and Figure 31. These were taken from RFMD's application note mentioned above, and include space for components to better match them for specific frequencies later on. They both convert the differential ports to 50 Ω single ended ports.

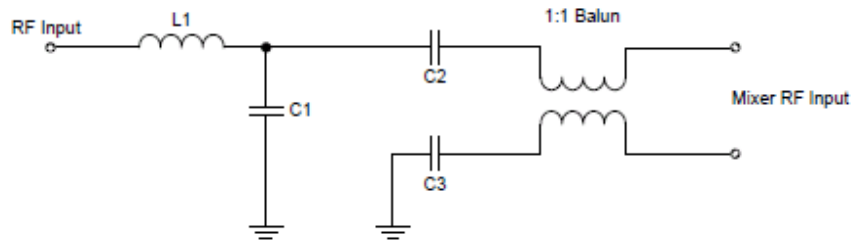


Figure 30 - Matching circuit for the RF2052's RF mixer input port. It converts the differential inputs to a 50 Ω single ended input. Initially the board will be tested with C2 and C3 equal to 1000 pF, and L1 a 0 Ω jumper, and C1 not fitted. The 1:1 Balun is an M/A-COM ETC1-1-13.

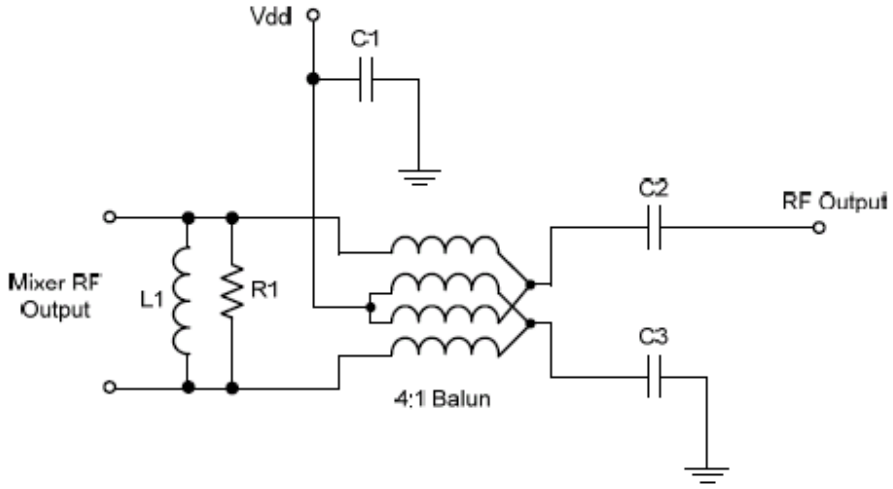


Figure 31 – Matching circuit for the RF2052's RF mixer output port. It converts the differential outputs to a 50 Ω single ended output. Initially the board will be tested with C1, C2, and C3 equal to 100 pF, and L1 and R1 not fitted. The 4:1 Balun is Minicircuits' TC4-19+.

As with the ADF4360-8, the RF2052 also required a loop filter. Similar to the ADF4360-8, the RFMD also supplies software to help pick the values for the loop filter components, based on the loop bandwidth, VCO frequency, charge pump current, VCO gain, and phase detector frequency. Unfortunately it was not clear what values to use here to obtain a good loop filter for a wideband design. Fortunately, the RF2051 evaluation board was designed to be as wideband as possible, so it was decided that the same loop filter used on the evaluation board would be used in the final design, as well as the reference crystal frequency. And of course, this filter can always be tweaked later on if need be, since only the component values would have to be changed. The loop filter schematic is shown below in Figure 32.

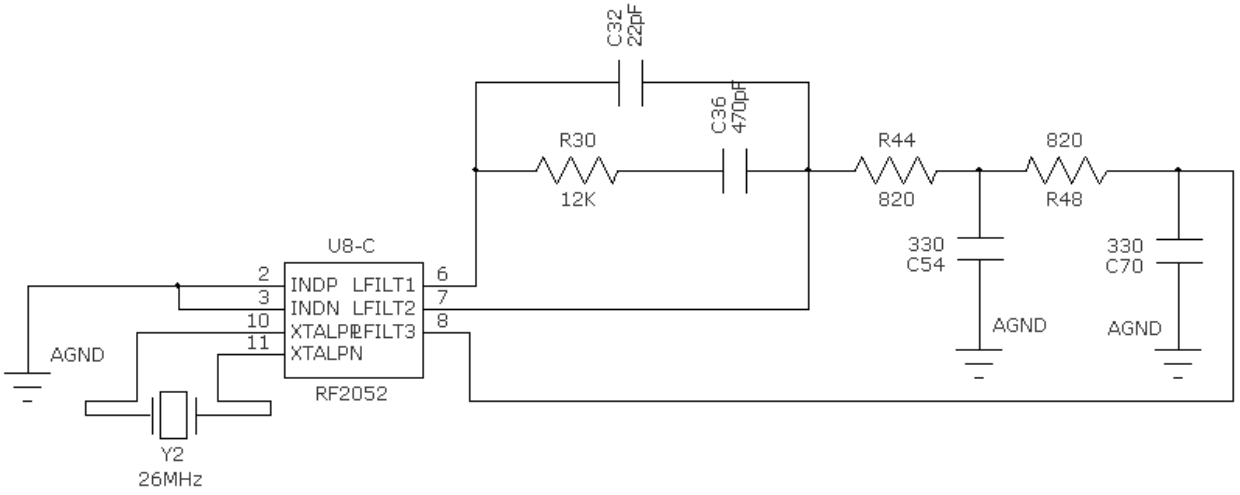


Figure 32 - The loop filter for the RF2052, configured for a wideband design. This is the same loop filter as used for the RF2051 evaluation board.

The last tricky part about the RF2052 is that its low frequency VCO, which is tunable between 1200 MHz and 1556 MHz, requires two external inductors to set its center frequency (the other two VCOs have internal inductors). According to the datasheet, these two inductors must each be about 2 nH. However, 2 nH is a very low inductance, and the combined inductance of the bondwire (IC pin, PCB pad, and solder joint) and the via to ground after the inductor is estimated to be about 1 nH, also according to the datasheet. Therefore the inductors used should only be 1 nH. Hi-Q 1 nH inductors are available from Coilcraft, however this would introduce more bondwire inductance, along with the inductances of the traces connecting the inductor to IC and ground. While these parasitic inductances are small, and normally not a problem, in this case they are large enough to increase the desired inductance by potentially up to 100%. For this reason, microstrip inductors were used, which consist only of a PCB trace going from the IC pin to a ground via. This is also the recommended method in the RF2052 datasheet. The inductance of the microstrip is calculated from its length, height above the ground plane, thickness of the copper, and the dielectric constant of the PCB material. An online microstrip calculator (http://www.technick.net/public/code/cp_dpage.php?aiocp_dp=util_pcb_imp_microstrip) was used to determine the length and width of the microstrips based on various properties of the PCB material and stackup used, which can be found in Chapter 0.

Finally, the serial interface had to be connected to the USRP. As discussed previously in Section 3.3, the interface was connected directly to free digital I/O lines. The RF2052 is powered with 3.3 V on three pins, and the final circuit in the design includes two decoupling capacitors (a 33 pF and a 0.01 uF) per pin.

4.2.1.4 Power Amplifier

The RF signal from the output of the RF mixer (RF2052) the signal that needs to be transmitted, as this is the signal that both contains the data sent from GNU Radio to the USRP, and is at the desired frequency. However before it can go an antenna, it first should be amplified if it is to travel any significant distance over the air. The amplifier chosen was Minicircuits' GVA-84+. It has a gain ranging from about 24 dB to 18 dB over the range 50 MHz to 2 GHz, and a maximum power output of 20 dBm (100 mW) at 1 dB compression. Considering that the RFX2400 board transmits 20+ mW, a maximum of 100 mW is more than enough.

The power amplifier is powered at the output through an RF choke, as shown below in Figure 33. Cblock must be chosen to have a low impedance at the lowest frequency, while the RF choke (RFC) must be chosen to have a high impedance at the lowest frequency. This is so the RF signal can pass unattenuated from the output of the amplifier to the next stage, but be blocked from passing from the output of the amplifier to V_{CC}, which in the case of the GVA-84+ is 5 V.

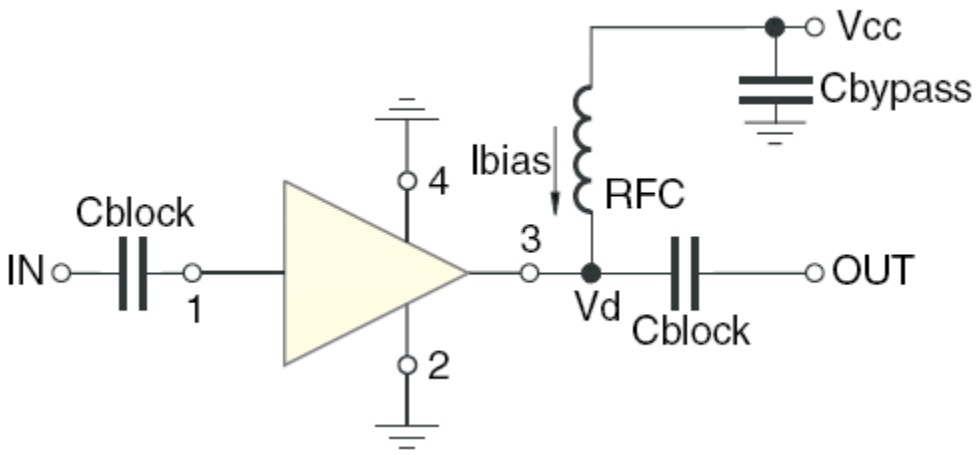


Figure 33 - Schematic for the recommended power amplifier circuit. The amplifier is powered with V_{CC} through the RF choke at the output. This DC power is blocked by Cblock, while the AC output is allowed through.

It was decided to use Minicircuits' ADCH-80+ RF choke for the RF choke, rather than a simple inductor, since it is designed specifically for this purpose. It has an impedance of 50 Ω, as does the amplifier, and is rated to operate between 50 MHz and 10 GHz. The value of Cblock was chosen to be 1 nF since the reactance of 1 nF at 50 MHz is only about 3 Ω.

4.2.1.5 Voltage Regulators

The circuits detailed in the previous sections all require either 5 V or 3.3 V. However, the USRP only provides 6 V to the motherboard (as well as 3.3 V meant specifically for the EEPROM). To generate these voltages, two voltage regulators are used. In addition, these voltage regulators are daisy chained, meaning that the 5 V regulator is powered from the 6 V, and the 3.3 V regulator is

powered from the 5 V. This was done so that the 3.3 V regulator could be placed far away from the 6 V pin on the motherboard connector, without requiring a long 6 V trace. As long as the maximum current output of the 5 V regulator is not exceeded, this approach works well.

The voltage regulator chosen was Analog Devices' ADP3336. It is a low dropout linear regulator. This was a good choice, because the low dropout voltage translates to high efficiency, and because it is linear it does not introduce high frequency noise into the power rails, as opposed to a switching regulator, which could potentially interfere with the RF operation. It also has a maximum current output of 500 mA, which is sufficient (see Section 4.3 for a power analysis). The regulator requires two external resistors to set the output voltage, and the datasheet provides values for these resistors for both a 5 V and 3.3 V output. The final 5 V regulator circuit is shown below in Figure 34, and the 3.3 V regulator circuit is shown below in Figure 35.

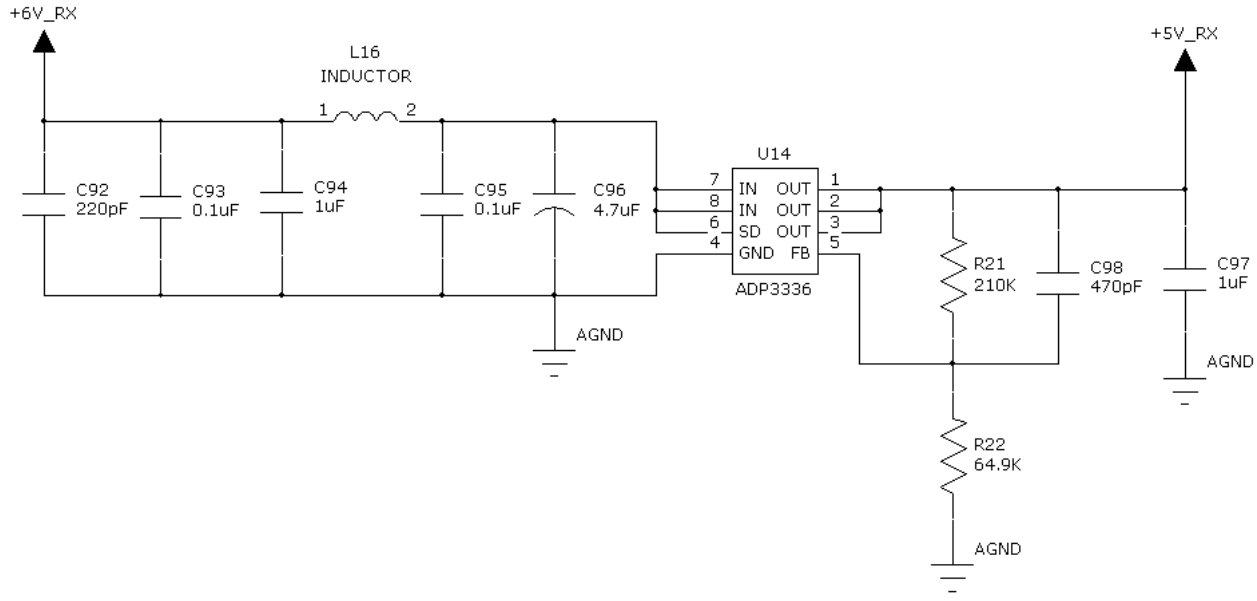


Figure 34 - Voltage regulator circuit using the ADP3336. The input voltage is 6 V and the output voltage is 5 V. R21 and R22 set the output voltage. L16 and all the capacitors clean the input and output voltages by removing any ripple present.

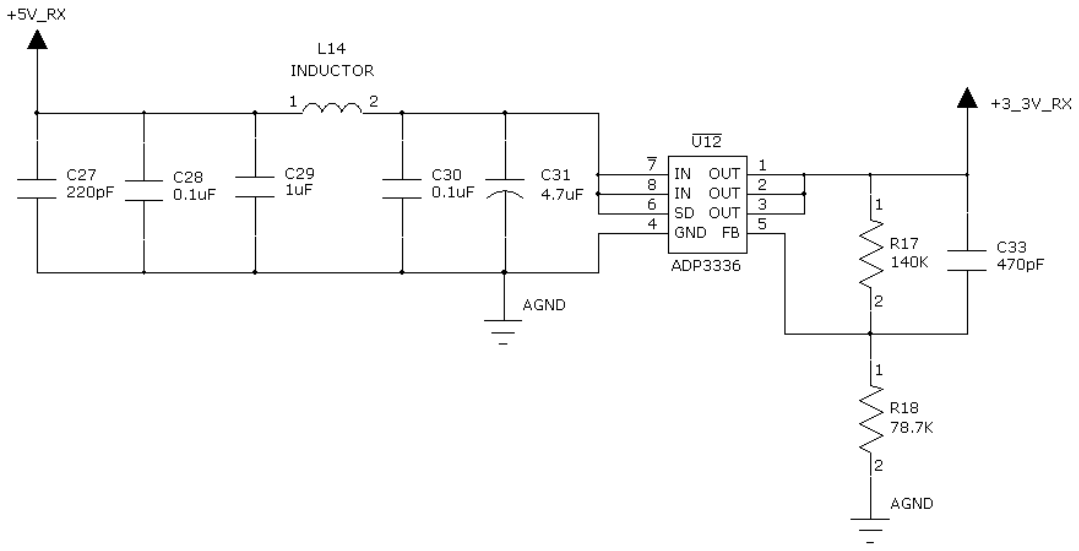


Figure 35 - Voltage regulator circuit using the ADP3336. The input voltage is 5 V and the output voltage is 3.3 V. R17 and R18 set the output voltage. L14 and all the capacitors clean the input and output voltages by removing any ripple present.

4.2.2 Receive Path

4.2.2.1 Low Noise Amplifier

The first step in the receive path is the low noise amplifier. It is responsible for amplifying the RF signal captured by the antenna while introducing as little noise as possible. The device chosen for this task was the MGA82563, which is the same amplifier used by the RFX2400 board. Its bandwidth is very wide, between 100 MHz and 6 GHz. Unfortunately this sets the lower end of the receive frequency range for the board to about 100 MHz. It has a gain ranging from about 14.5 dB to 13.5 dB over the range 100 MHz to 2 GHz, a maximum power output of 17.5 dBm (100 mW) at 1 dB compression, and a noise figure of only about 2.2 dB in a 50 Ω system.

The external circuitry required to power and interface this amplifier to the rest of the circuit is the same as that required by the GVA-84+ power amplifier detailed previously, so the circuit shown in Figure 33 was used for this amplifier as well.

4.2.2.2 RF Mixer

The next stage in the receive path is the RF mixer. The function of this stage is to convert the input RF signal frequency to an IF of about 300 MHz, either by up or down conversion. This stage is almost identical to the RF mixer stage in the transmit path detailed in section 4.2.1.3. The only difference is that the next stage (Quadrature Demodulator) has a differential input with an input impedance of about 200 Ω. Since the output port of the RF2052 is differential and should drive a load between 200 and 500 Ω, the output is connected directly to the input of the next stage with no matching circuitry.

4.2.2.3 Quadrature Demodulator

This stage in the receive path performs the exact opposite function of the Quadrature Modulator in the transmit path. It takes as input an RF signal at the IF (300 MHz) and converts it into I and Q components at a low frequency (between about 5 and 20 MHz). These two signals are fed into the high speed ADCs on the USRP motherboard, which get digitally downconverted to baseband before going to the PC.

The quadrature demodulator IC chosen for this stage was Analog Devices' AD8348, which complements the AD8345 chosen for the modulator in the transmit path. The IF input to the AD8348 is driven differentially, and as described in the previous section, the mixer output from the RF2052 is connected directly to this input without any matching circuitry. The I and Q outputs are connected to the ADC inputs on the USRP motherboard through 200 ohm resistors, since this is what is done on the RFX2400 board.

An antialiasing lowpass filter for the ADC is included, but it is connected to certain pins on the chip provided for this purpose. The datasheet gives a schematic of a 100 Ω, fourth-order elliptic low-pass filter with a 3 dB cutoff frequency of 20 MHz. Since the sampling frequency of the high speed ADCs in the USRP is 32 MSPS, a 20 MHz lowpass filter is appropriate, so it was included in the final design. Figure 36 below shows the lowpass filter from the datasheet.

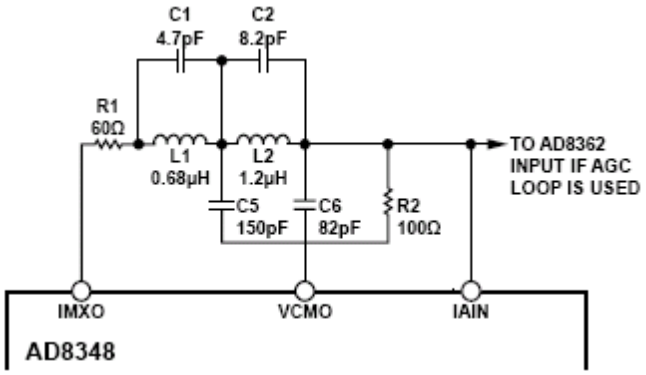


Figure 36 - Antialiasing filter used for the ADCs on the USRP. This filter is a 100 Ω, fourth-order elliptic low-pass filter with a 3 dB cutoff frequency of 20 MHz.

Figure 37 below shows the frequency response of this lowpass filter.

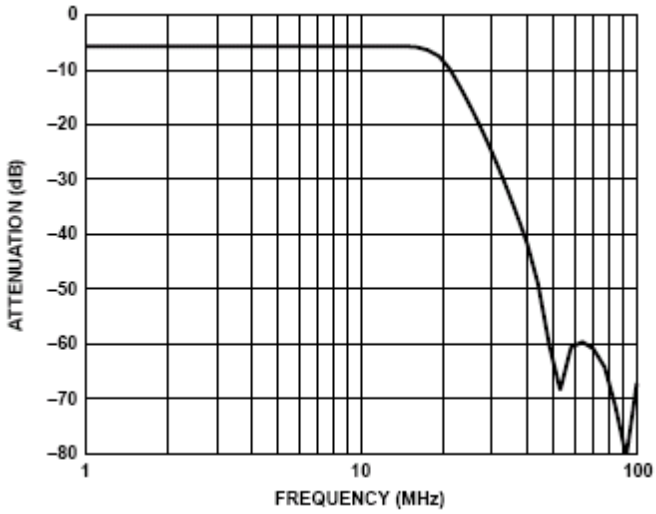


Figure 37 - Frequency response of the antialiasing filter. Note that the cutoff frequency is 20 MHz

4.2.2.4 Local Oscillator

The local oscillator used for the quadrature demodulator, as well as its configuration and integration is exactly the same as the local oscillator used for the transmit path detailed in 4.2.1.2.

4.2.2.5 Voltage Regulators

The voltage regulators used for the receive path, as well as their configuration and integration is exactly the same as for the transmit path detailed in 4.2.1.5.

4.2.3 Transmit/Receive Switches

One of the objectives for this project is to have the daughterboard be able to operate in either half-duplex or full-duplex modes. Full-duplex mode is simple, as the transmitter and receiver can be completely separated, each with its own antenna. This, however, poses problems if both the receiver and transmitter need to operate at the same frequency. Half-duplex mode not only allows both the transmitter and receiver to operate on the same frequency, but also allows them to share a single antenna.

To accomplish this goal, two transmit/receive (T/R) switches and a little bit of glue logic was used in a specific configuration. The particular T/R switch chosen was the Hittite HMC174MS8. The schematic of this configuration is shown below in Figure 38.

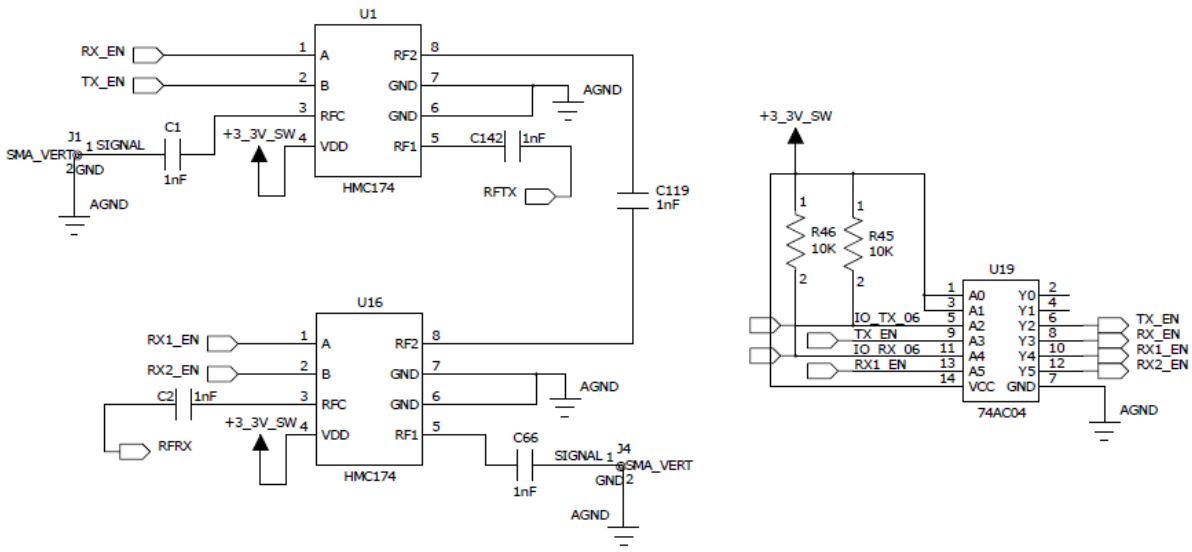


Figure 38 - T/R switch circuit allowing both half and full-duplex operation.

When B on the T/R switch is high and A is low, the RFC pin is internally connected to the RF1 pin. When B is low and A is high, RFC is internally connected to the RF2 pin. If the logic of this circuit is followed through, the behavior in the truth table shown below in Table 1 is found.

Table 1 - T/R Switch truth table for antenna functions.

IO_TX_06	IO_RX_06	Antenna J1	Antenna J4	Mode
0	0	RFTX	No connection	Half-Duplex (transmit)
0	1	RFTX	RFRX	Full Duplex
1	0	RFRX	No Connection	Half-Duplex (receive)
1	1	No Connection	RFRX	Receive only on J4

4.3 Power Analysis of Final Design

Once the major components for the final design were chosen, it was deemed necessary to do a power analysis, to ensure its power consumption would not exceed what the USRP is able to supply to its daughterboards. The power analysis was split into two parts: the transmit side and the receive side. This way it was easier to do the calculations since each side is powered by its own pair of voltage regulators. It also makes it easier to see what the power consumption will be when only one path is powered on.

First, only the components that contribute significantly to the power consumption were identified. It was decided that the components that draw current on the order of microamps would not impact the analysis in any significant way, and so were not included. After going through the datasheets and identifying each component's maximum current draw, the following components were chosen to be included in the analysis:

- ADP3336 (Adjustable Low Dropout Regulator)
- AD8345 (Quadrature Modulator)
- AD8348 (Quadrature Demodulator)
- RF2052 (Wideband RF Synthesizer/VCO and RF Mixer)
- ADF4360-8 (Integrated Synthesizer and VCO)
- GVA-84+ (Wideband Monolithic Amplifier)
- MGA-82563 (Wideband Low Noise Amplifier)

Since each path (transmit and receive) is powered by its own pair of voltage regulators, care was taken to account for this. The power hierarchy for the transmit path and receive path is shown below in Figure 39 and Figure 40 respectively.

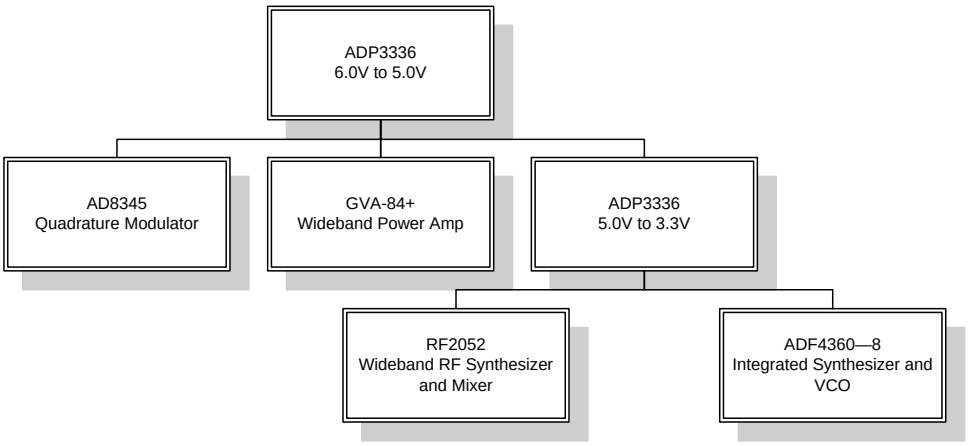


Figure 39 - Transmit path power hierarchy. The top row is the first voltage regulator powered with 6V by the USRP. The second row is powered with 5V provided by the first voltage regulator. The third row is powered with 3.3V provided by the second voltage regulator.

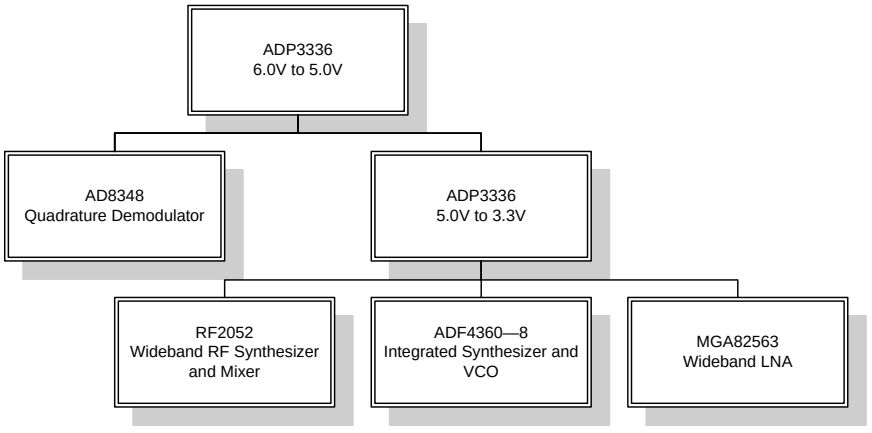


Figure 40 - Receive path power hierarchy. The top row is the first voltage regulator powered with 6V by the USRP. The second row is powered with 5V provided by the first voltage regulator. The third row is powered with 3.3V provided by the second voltage regulator.

Since the voltage regulators are not 100% efficient, the total power consumed by them plus their loads is greater than the total power consumed by their loads. Unfortunately the datasheet for the ADP3336 does not once mention its efficiency. It does, however, provide graphs of Ground Current vs. Load Current, and Dropout Voltage vs. Output Current. These two graphs are shown below in Figure 41 and Figure 42 respectively.

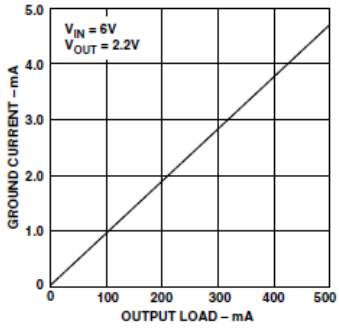


Figure 41 - Ground Current vs. Load Current

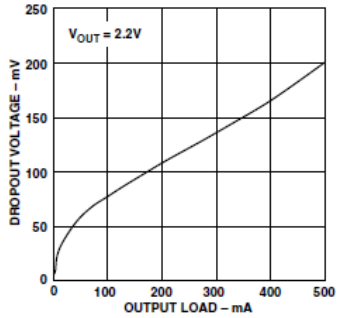


Figure 42 - Dropout Voltage vs. Load Current

These two graphs were approximated with the following equations:

$$I_G = \left(\frac{0.0046}{0.500}\right) I_L \qquad \text{Equation 1}$$

$$V_{DROP} = \left(\frac{0.150}{0.500}\right) I_L + 0.050 \qquad \text{Equation 2}$$

Note that the equation for the dropout voltage is only a valid approximation for a load current greater than about 50 mA. The power consumption of the regulator itself can be calculated with the following equation:

$$P_L = V_{DROP} \cdot I_L + V_{IN} \cdot I_G \qquad \text{Equation 3}$$

By manipulating this equation, the efficiency of the regulator is found to be

$$Eff = \frac{1}{1+\frac{(V_{DROP} \cdot I_L + V_{IN} \cdot I_G)}{V_{OUT} \cdot I_L}}$$ Equation 4

Using these equations, a spreadsheet was created using the maximum current drawn by each device according to their datasheets. This final data from the spreadsheet is shown below in Table 2. Note that the calculated current drawn and power consumed by the regulators is the sum of the regulators own consumption and its load. Thus the calculated power consumption of the top 5 V regulator for the transmit side is the entire power consumption of the transmit side, and the same applies for the receive side.

Table 2 - Power Analysis of Entire Design

	Part	Voltage (V)	Current (A)	Power (W)
TX:	ADP3336	6	0.23131553	1.387893182
	AD8345	5	0.075	0.375
	GVA-84+	5	0.13	0.65
	ADP3336	5	0.062649135	0.313245675
	RF2052	3.3	0.072	0.2376
	ADF4360-8	3.3	0.0195	0.06435
RX:	ADP3336	6	0.1617035	0.970220999
	AD8348	5	0.055	0.275
	ADP3336	5	0.132969375	0.664846875
	RF2052	3.3	0.072	0.2376
	ADF4360-8	3.3	0.0195	0.06435
	MGA82563	3.3	0.101	0.3333
Total Max:			0.39301903	2.358114181

From the analysis it can be seen that the maximum power consumption of the board in full duplex mode is 2.36 W, or 393 mA from the 6 V supply. In transmit only mode it draws 231 mA from the 6 V supply, and in receive only mode it draws 162 mA from the 6 V supply. The USRP can supply up to 1 A from its 6 V supply to the daughterboards. According to this analysis a single USRP could potentially run two of these boards simultaneously with both in full duplex mode. It is also worth noting that the maximum allowed load current of the ADP3336 regulator is 500 mA. Since the most current drawn from any of the four regulators is 231 mA, all are running within specification.

5 PCB Design

The PCB design is a key aspect for this project. This is because the high frequencies used in this project the design cannot be tested on a breadboard or on a perfboard. The PCB design needed to be very precise and follow strict guidelines to come in under budget and with the uniform impedance required by the design. The following sections cover the design of the PCB from which software was chosen and why to component placement on the actual PCB.

5.1 Tools

There are a wide variety of tools available to lay out a PCB. At the start of the PCB design process there was no specific piece of software which was clearly the best choice for this project. As a result, it made more sense to use a free layout tool over a licensed one because it allowed for the software to be installed on personal computers. This added a degree of flexibility to the project as design was not constrained to the lab computers.

PCB Artist is a free PCB layout program which is distributed by Advanced Circuits. Unlike other programs, PCB Artist has no minimum board size and is able to produce Gerber-formatted files, which are the industry standard format for PCB designs. PCB artist also has the ability to create a schematic for the design and then link it to the actual layout, thus making it much easier to check for errors in the layout.

The PCB was eventually created using PADS from Mentor Graphics. Although PCB Artist offered a wide range of features, PADS was ultimately a better choice for a couple of reasons. A large number of the parts used in this project are also used in the RFX2400 board from Matt Ettus, for which the PCB files were created using PADS. Copying footprints from that PCB layout saved a large amount of time since this meant that the footprints did not have to be recreated from scratch, and also guaranteed that the footprints would be the right size.

PADS also offers the ability to fill in the open space on the board with stitching vias to the ground plane. This is extremely important to this project for a number of reasons. Common practice when designing a PCB is to put all extremely high speed nets on internal layers and surround those layers with ground planes. This prevents those nets from picking up outside interference and also reduces their ability to couple with other nets. The problem is that internal routing is only available on production PCBs, which cost more than the entire budget for this project. As a result, it is necessary to put all the RF traces, together with all the other traces, on the outside layers to keep from going over budget. The best way to protect these traces from outside signals and coupling with each other is to fill all the empty area on the layer with copper and connect this plane back to the ground plane in as many places as possible. PADS is capable of doing both of these tasks automatically and is smart enough to make sure that all of the vias it places do not cause problems with other nets.

The time that was needed to figure out how to use these advanced features in PADS was much less than the time saved over trying to do all of this manually in PCB Artist. With the design in

PADS, it also made it much easier to compare with Matt Ettus' design to try to keep it as similar as possible, as his design was already proven to work.

5.2 Layout

The layout of the board had to be very carefully planned so that everything would fit without having long traces. In order for the board to work with the USRP, it was essential that the PMC connectors on the bottom were in the correction positions. Consequently, these connectors were the first parts placed on the board. They were then locked into place so they wouldn't accidently be moved while other parts were being placed. Also critical for the daughterboard to physically interface with the USRP are the drill holes, which also needed to be placed in exactly the same places as on Matt Ettus' board.

The PCB has 4 copper layers, 2 on the outside and 2 in the middle. All traces are on the 2 outside layers while the ground planes and multiple power planes make up the 2 internal layers. Figure 43 below is a cutaway of what the layers in the PCB look like.

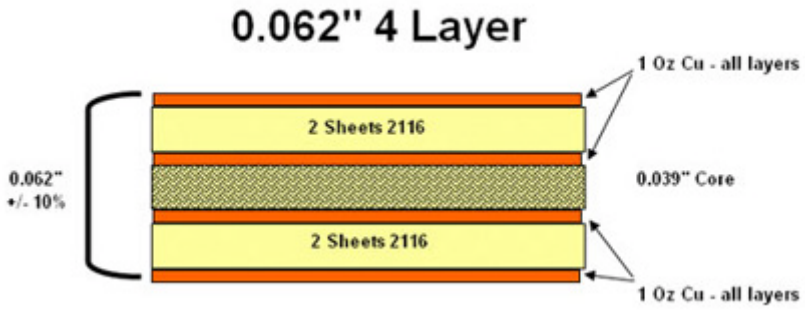


Figure 43 - Internal stack-up of PCB

Assuming the top layer to be layer 1 and the bottom layer to be layer 4, layer 2 would be the ground plane. This entire layer is a solid copper plane which is connected to the ground planes on the top and bottom layers. As mentioned above, the top and bottom layers were both flooded with copper to help reduce interference between the nets and from outside sources. This solid ground planes runs between all of that, thus keeping it all at a uniform voltage.

Layer 3 contains 5 different power planes. There is a 3.3 volt and a 5 volt power plane for each side of the board and a 3.3 volt power plane for the T/R switches in the middle of the board. The main reason for creating 5 different planes for 2 different voltage levels is that there are large areas between some of these planes. For example, there is only a very small area in the transmit side that needs 3.3 volts, the rest of the side is 5 volts. The T/R switches are located in the middle of the board, which happens to be between two areas that require 5 volts, which meant that another power plane was required. The power planes on the receive side are set up very similarly to those on the transmit side.

The PCB layout has been specifically designed to keep all RF traces as short as possible. The first step was to segregate the two major sides of the board, transmit and receive so that they would have the majority of their components on opposite end of the board and then come together in the middle near the T/R switches. Both sides of the board were placed by following the signal path on the schematics. This meant that all the components that handled the signal were place on the board first. They were placed in the order which the signal travels through them. On the transmit side, this meant starting with the baseband signal from the PMC connector and ending with the amplifier. Once all of the components that are a part of the signal path were placed and had the signal traces routed, the other components and non-signal traces were added.

The top layer of the PCB contains the majority of the traces and components on the board. This is because using vias between layers can degrade high speed signals. The bottom layer of the PCB contains some of the non-signal traces that could not be routed on the top layer, as well as some of the resistors and capacitors that could not be placed on the top layer.

5.3 Production

There are a number of factors to consider when making a multiple layer PCB under a tight budget. Before the process had even gotten very far underway, the team was aware of the $66 each deal for a 4-layer board from Advanced Circuits. There were a couple of constraints on this deal which were a problem that went unnoticed until it became time to actually order the board.

When working with high speed signals, it is important to remember that the impedance of each part of the circuit needs to be matched. When the impedance of a circuit changes part of the signal is reflected back and part is able to pass, proportional to how much the impedance changes. This reflected signal can cause a vast number of problems in a complex circuit, such as dissipating energy in traces, lowering the gain or creating noise. All of these can lead to the circuit failing, which is why it is important to keep impedance uniform throughout a signal path. In a PCB, this is done by keeping traces a uniform width and by making sure the thickness of the board material is uniform, called the dielectric constant.

Initially the board was designed to use the same dielectric properties as Matt Ettus had with his board. Advanced Circuits considers specifying dielectric properties or requiring a controlled impedance to be priced as a production board. This meant that it would end up costing around $2000 to have 5 boards made, which is way over budged. This problem was overcome by looking up what the default dielectric properties were for the boards Advanced Circuits uses for their $66 each deal. This information could then be put into a trace width calculator to figure out how wide the traces on this PCB needed to be made in order to achieve a uniform impedance of 50 ohms, which turns out to be 15 mils.

There are a few areas on the board, particularly around the RF2052 ICs that are extremely busy. As a result, it became very difficult to try to fit all the traces in on only two layers. There were a few traces that were placed on the power plane for short distances to keep them from having to

go long distances around the chips. The $66 each deal does not allow traces on internal planes, which meant that these traces, and several other traces near the ICs, had to be routed again to fit on only the outside layers.

5.4 Initial Tests

Upon arrival, the PCBs were tested using a continuity check on a multimeter. This was done to look for any shorts between the power planes and the ground plane as well as to check some of the more important nets which could damage an IC if incorrect. This check was necessary because the $66 each deal does not include electric testing on the boards. Electric testing would have guaranteed that all of the nets on all of the boards were exactly what there were supposed to be and were not shorted to any other nets.

5.5 Board Renderings

The pictures below were generated using the layer information from the PCB. They were created using GIMP and renderings of the Gerber-files exported by PADS.

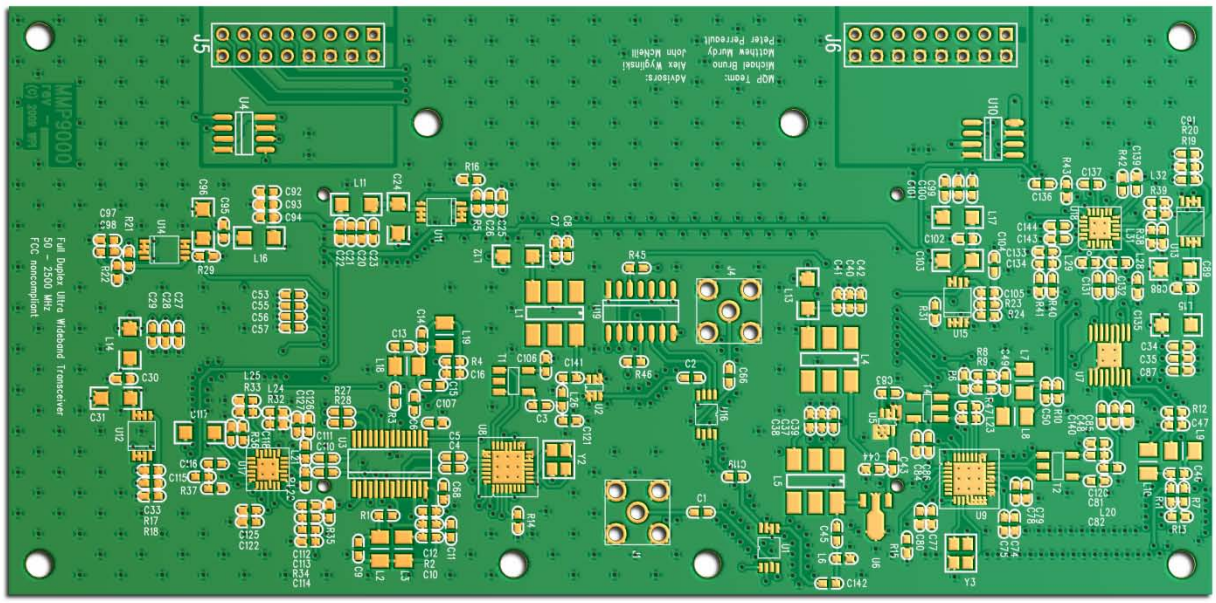


Figure 44 - The top layer of the PCB

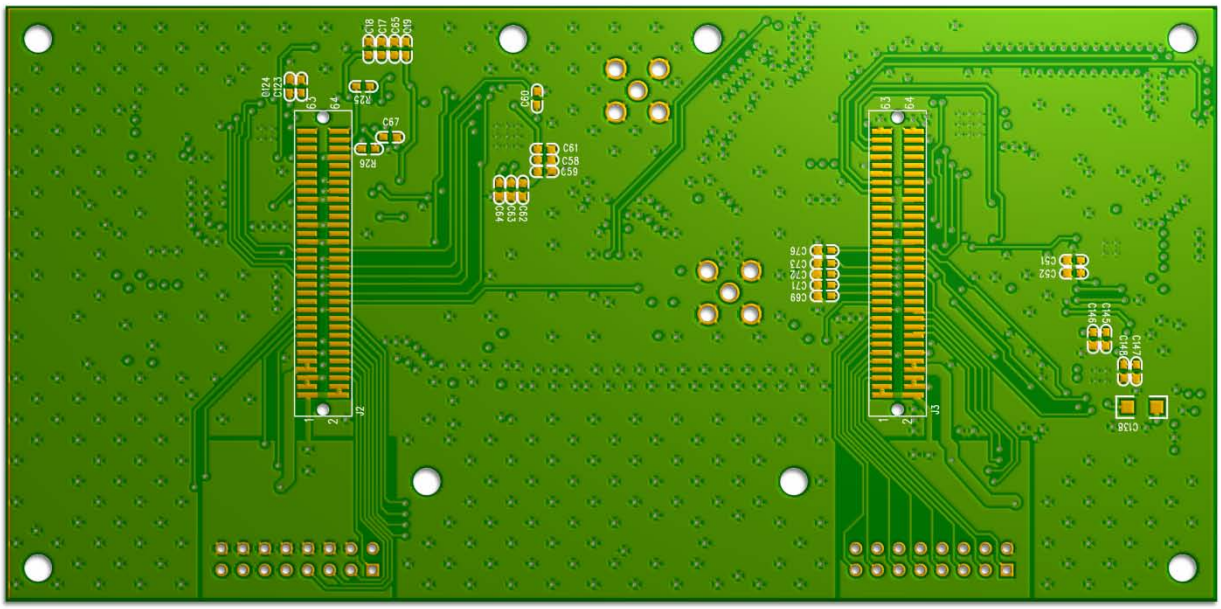


Figure 45 - The bottom layer of the PCB

5.6 Chapter Summary

PADS absolutely made the design of the PCB much easier than other freely available software suites. This is because it helped us check our layout to ensure that it could feasibly be produced. It also generates the correct industry standard files needed by the company to actually produce the boards and was able to keep the parts list on the PCB updated with any changes that were made in the schematic. The level of complexity seen in the above pictured PCB means that it cannot be produced by any equipment on campus and therefore had to be sent out to a third party company to be produced., which is why the industry standard files produced by PADS were so important. This company also guaranteed a uniform impedance, which is essential for the design to function properly.

6 Construction, Debugging, and Testing

There are several hundred surface mount components on this PCB, all of which need to be hand soldered. Normally, a board of this complexity, would be put together by machines, but that option was not available due to budget constraints. This also meant that all debugging due to damaged or incorrectly placed parts also had to be done by hand and that the testing had to be done incrementally throughout the construction. This section chronicles the process of assembling the first version and second version of the PCB for the transceiver.

6.1 Soldering and Component Placement

After the design of the PCB was complete and the boards were received, the next step towards building the transceiver was soldering down components. This was quite a technical problem in itself because of the extremely small package sizes of components on ICs required in the design. Some of the ICs had QFN type packages, such as the RF2052, with 32 pins in a 5mm by 5mm area. In addition to the small pin spacing, the pins were also partially under the IC, with a copper ground plane covering the remaining surface area under the IC. Figure 46 shows the package drawing for the RF2052 IC, taken from the data sheet. Since the ICs needed were only available in this type of package, and the team does not have experience soldering such small package sizes, expert help was needed by someone in order to fully populate the PCB. Thankfully, Bob Boisse was able to solder the ICs by hand after heating up a piece of solder under the chip to connect it to the ground pad. Once the chip was soldered in place and connected to the ground plane, he was able to heat the pins up enough so solder was wicked under them to make a good connection between the pins and the pad. Using this technique he was able to solder the RF2052 on each side of the board, along with the ADF chips.

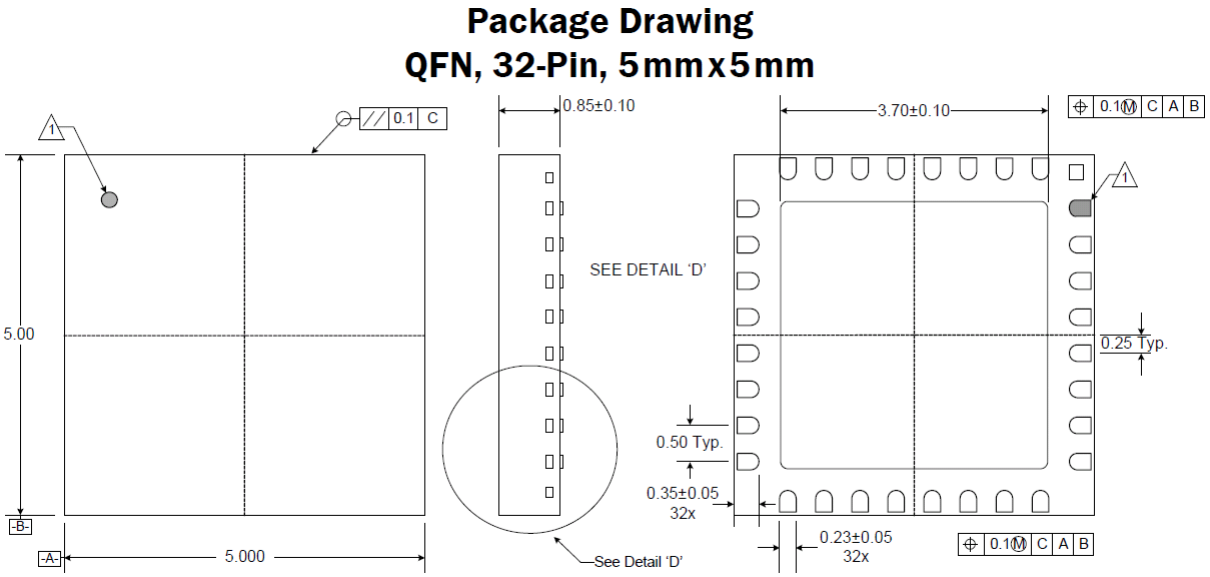


Figure 46 - QFN Package Drawing for the RF2052

After the ICs were soldered to the board, the individual components and other ICs that were not soldered down by Bob needed to be soldered as well. There were two options for soldering down the rest of the components, by using solder paste or by hand soldering them using tweezers and a microscope. Solder paste is a pasty mix of powdered metal solder and flux that can be applied to the pads using a pump. Heating the paste causes the solder to flow onto the pins and pads it is in contact with, making an electrical connection. Figure 47 shows a typical view through the lens of the soldering microscope, with 0603 and 1206 sized components. After Bob's recommendation, it was decided that the components would be placed by hand soldering them to the PCB, using tweezers, a microscope, and a fine tipped soldering iron. Since the design has over 300 components, this step was quite a task. Most of the capacitors and resistors in the design had an 0603 sized package, which initially made soldering them down difficult, but after a little practice the process became a lot easier. A good technique is to melt a little solder on the pad and then re-melt it before putting the component down, just to keep it in place when the other side is soldered. After completely soldering the opposite side, more solder can be added where the component was initially tacked down. Figure 48 shows a microscope view of solder joints for some components on the board.

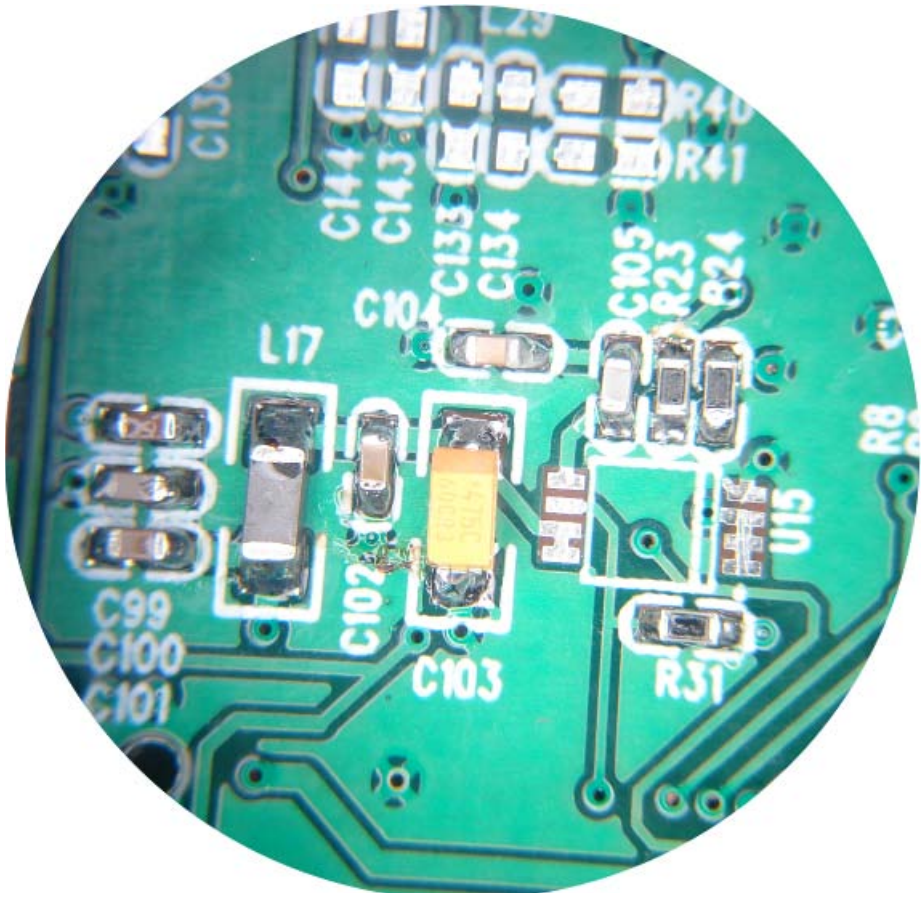


Figure 47 - Microscope view of the PCB while soldering down components.

This was the basic technique for soldering down the rest of the components to the board. Various connection tests were made throughout the progress, in order to make sure the components were placed correctly and there were no shorts. Once enough components put down to test the transmit side, the PMC connectors were soldered to the underside, so the functionality could be verified by connecting it to the USRP. In the case that it was not functional the components on the other half of the board would not have been wasted. Once the transmit side was completely soldered, it was time to test and debug the PCB by connecting it to the USRP.

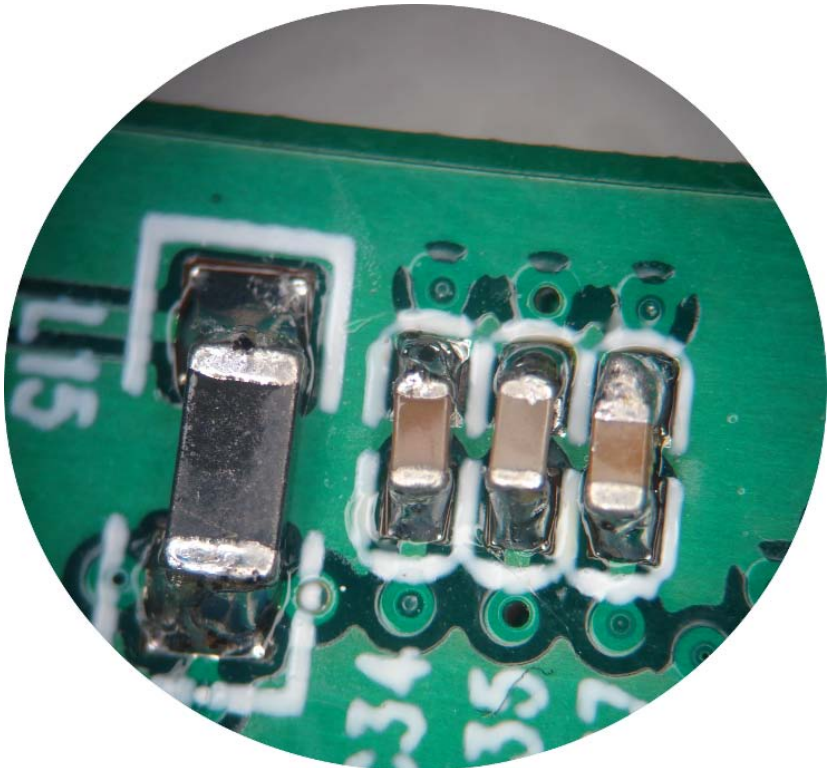


Figure 48 - Microscope view of solder joints for three capacitors and an inductor.

6.1.1 Second Soldered PCB (rev. A)

The figures below show the final soldered version of the PCB used for testing and the results section. All components shown were soldered by hand by either Bob Boisse or the MQP team. PCBs were fabricated by Advanced Circuits.

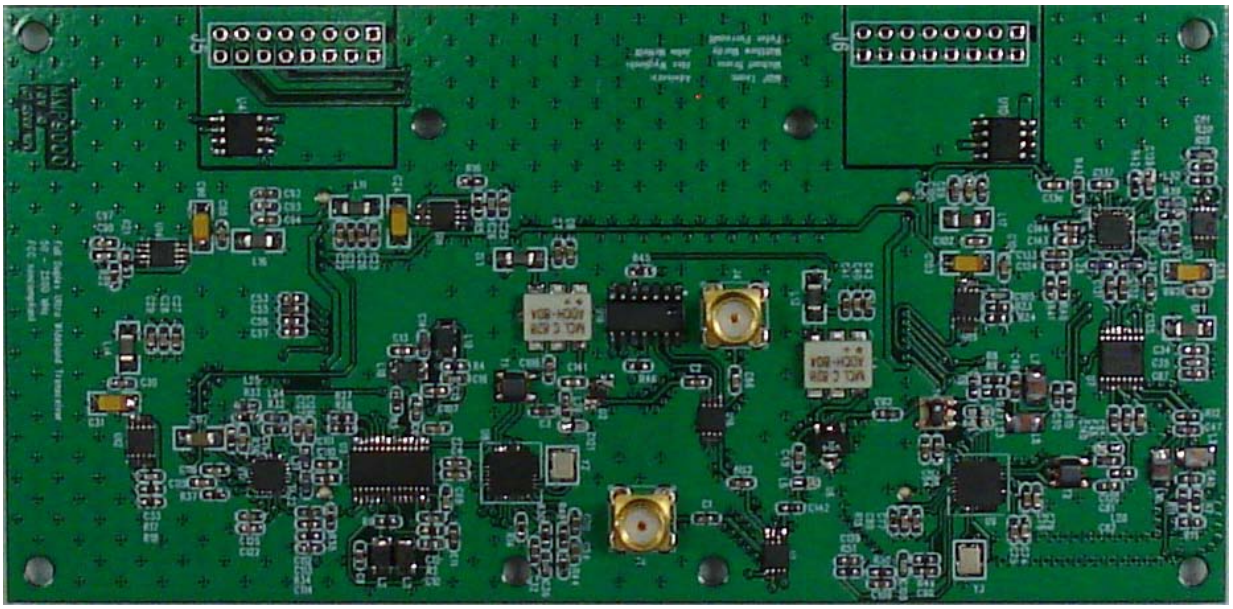


Figure 49 -The top layer of the PCB

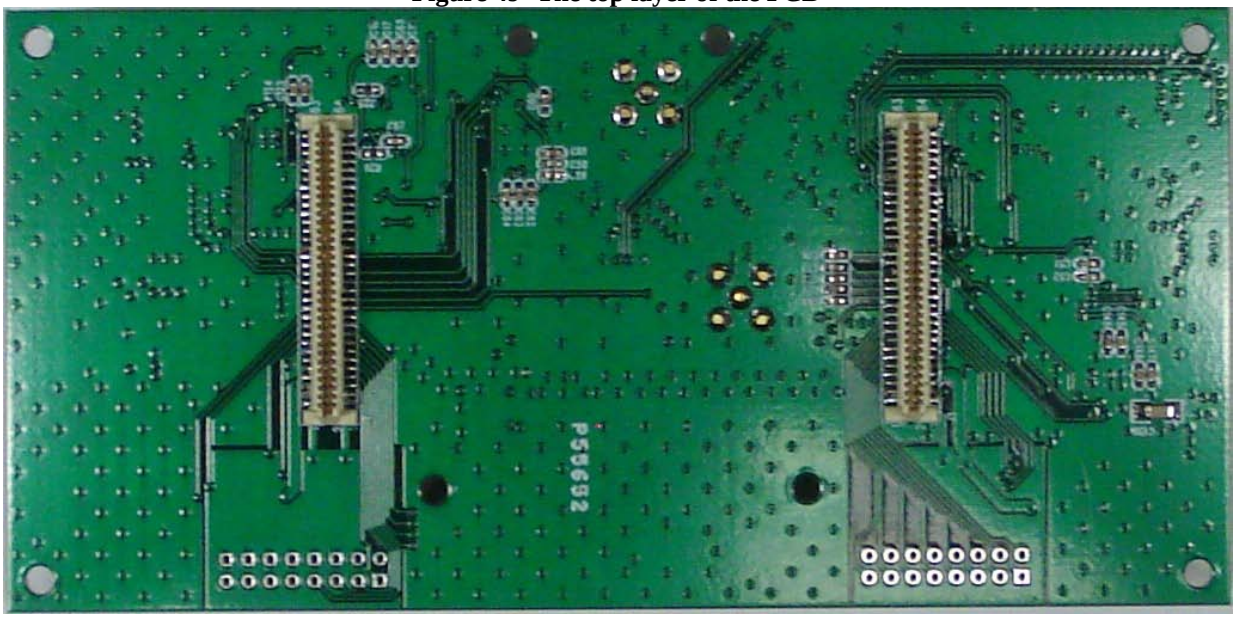


Figure 50 - The bottom layer of the PCB.

6.2 Testing and Debugging the first PCB

When all the components were placed on the transmit side of the board, it was time to debug any problems that occurred when connecting it to the USRP. The initial test of the USRP's I/O was done by plugging in the PCB when only the voltage regulators and the components associated with them were soldered down. This test was a success; the voltage regulators were able to be controlled through GNU Radio. This test also indicated the daughterboard was correctly connected to the USRP, in terms of power and I/O.

After this initial connection test, the rest of the transmit side was ready to be soldered. Pins and connections were tested with a multimeter occasionally for shorts, and all the pins on the PMC connector were tested for shorts after they were soldered. Once all the components on the transmit side were soldered to the PCB, and all connections between planes were checked for shorts, it was time to test the board by connecting it to the USRP.

The first connection of the daughterboard to the USRP had a mysterious result. The I/O on the USRP was no longer working, so there was no way the voltage regulators for the daughterboard could be turned on to power it. It was initially thought that there had to be a short somewhere on the daughterboard. Oddly enough, the problem affected both sides of the USRP; I/O would no longer work even on daughterboards connected to the other daughterboard slot. After not finding a short or any logical reason behind this behavior, it was decided that another device would need to be made to power the daughterboard outside the USRP. This device simply needed to connect a 6 V supply to the correct pins on the PMC connector, along with the enable pin on the voltage regulators.

After making this connector, it was found that the daughterboard was able to power up and supply the correct voltages as expected. This made the problem even more perplexing. It was suspected that it may have been the result of another problem, like too much noise being introduced to the amplifier stage or a missing EEPROM on the other side of the board, but none of these hypotheses turned out to affect the underlying problem. When a week of debugging went by without any results, it was decided that the best course of action would be to contact Matt Ettus, creator of the USRP and its daughterboards, to see if he has ever run into the same problem. Thankfully, his reply was prompt, asking for the schematic and the process used to debug the problem. After some short correspondences, Ettus determined that the problem was caused by the static protection diodes on the input to the ADF4360 PLL chips. Apparently, when the regulators power down, these static protection diodes on the SPI enable, clock, and data pins conduct and pull the inputs down when they are being driven high. This voltage drop and current sink made the FPGA unable to power any of the other IO on the board, thus shutting off the voltage regulators.

The simple solution to this problem was to tie the voltage regulators high, so they could never shut off as long as they were plugged into the USRP. This modification was easily made to the board by cutting the I/O trace to the enable pin on the regulators and adding a jumper to 6 V that would make the enable pin on the regulator always high. This solution worked and the daughterboard was able to be powered up and tested.

With the board able to be powered, the same basic code that was used to program the evaluation prototype was used to program the frequency on the RF2052 chip on the PCB. However, no leakage from the local oscillator could be observed on the output of the mixer. Many different frequencies were tested between the full range of frequencies between 300 and 2500 MHz, but none were observed on the output of the mixer. Thinking it may be a hardware problem, the

schematic of the evaluation board was more thoroughly reviewed, and it was found that the PLL loop, which had been omitted from the first PCB design, was actually required. Without this PLL loop, the frequency could not be synthesized in the RF2052, and the board was not functional. Due to the number and small size of the components needed to implement the PLL loop filters for both RF2052 chips on the transmit and receive sides of the PCB, it was not possible to incorporate them onto the board without doing a second revision of the PCB.

6.3 Second PCB

Since two major problems were identified in the first version of the PCB, the fixes for each were incorporated into a second version of the PCB. This included removing the digital I/O control lines to the 5V voltage regulators, and instead tying their enable pins high so they are always on. This also included adding the PLL loop filters for the two RF2052 chips on the PCB. Due to the tight placement of components around the RF2052 chips this was a somewhat difficult task, but was managed after several hours of work. The components and configuration of the PLL loop filter used are detailed in Section 4.2.1.3. Figure 51and Figure 52 highlight the differences between the two PCBs; the first version of the PCB do not have the PLL loop filter, while it was included in the second version of the PCBs.

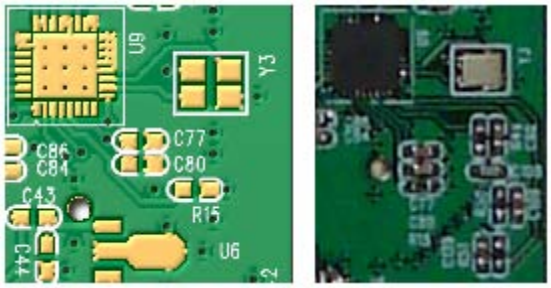


Figure 51 - Segment of the PCB with and without the PLL loop filter for the RF2052 on the transmit side on the first and second PCBs.

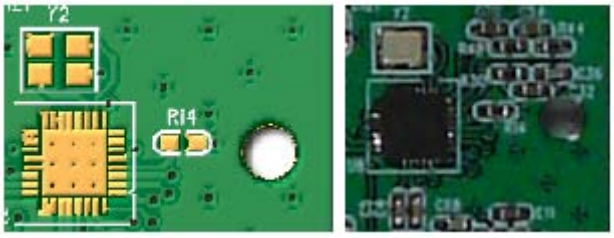


Figure 52 - Segment of the PCB with and without the PLL loop filter for the RF2052 on the receive side on the first and second PCBs.

6.4 Chapter Summary

After many long hours of looking into a microscope and two different revisions of the PCB, the board was finally fully assembled and ready to operate within the USRP. Once connected to the USRP, the team was able to start gathering results and data that will be analyzed in later sections.

7 Results

Once all the hardware design and debugging was complete, the board was connected to the USRP and tested using GNU radio. The results from the tests are documented and analyzed in the following sections. This data is the result of the simulation, design, testing and construction discussed in all the previous sections of this document. Based on the data from these tests, the overall design is evaluated according to the objectives set forth in the beginning of this paper.

7.1 Transmitter

The transmitter was successfully tested with the Mario II Overworld theme being broadcast across Atwater Kent at 94.1 MHz to a portable FM radio receiver. This test was conducted using an additional amplifier, which added the additional power that was needed to broadcast the signal across that distance. This amplifier was the GVA-84+ evaluation board connected in between the output of the board and the antenna.

This test was later verified without the additional amplifier. This was by broadcasting on open frequencies in the commercial FM radio bands to a clock radio that was less than a meter away from the USRP which had the board in it. The tests were able to show that there were no frequencies within those bands that the radio could not hear the board transmitting on.

These lower frequency tests were able to prove that the board was capable of transmitting a signal successfully, but they did not prove that the board was a wideband transceiver, capable of transmitting between 50 MHz and 2500 MHz. The next step was simply to have the board transmit to the RFX2400 in another USRP at 2.5 GHz and see if the song could be heard exactly as it had at the lower frequency. This would confirm the absolute upper end of the transmitting capabilities of the board, however, the absolute lower end still needed to be tested. The 50 MHz test was also completed using two USRPs. For this, the receiving USRP was set up to use the basic RX board, as the ADCs on the USRP motherboard are fast enough to directly decode a 50 MHz transmission.

As expected, both the 2.5 GHz and the 50 MHz transmissions were successful as the receiving boards received both transmissions. However, as part of routine procedure, the signals were also watched using the spectrum analyzer connected directly to the SMA connector on the board, which actually caught some interesting data. The waveform that was previously produced during the 94.1 MHz experiment was much more ideal than the waveform that was produced at 2.5 GHz. These waveforms are shown as spectrograms below with the 94.1 MHz transmission in Figure 53 and the 2.5 GHz transmission in Figure 54.

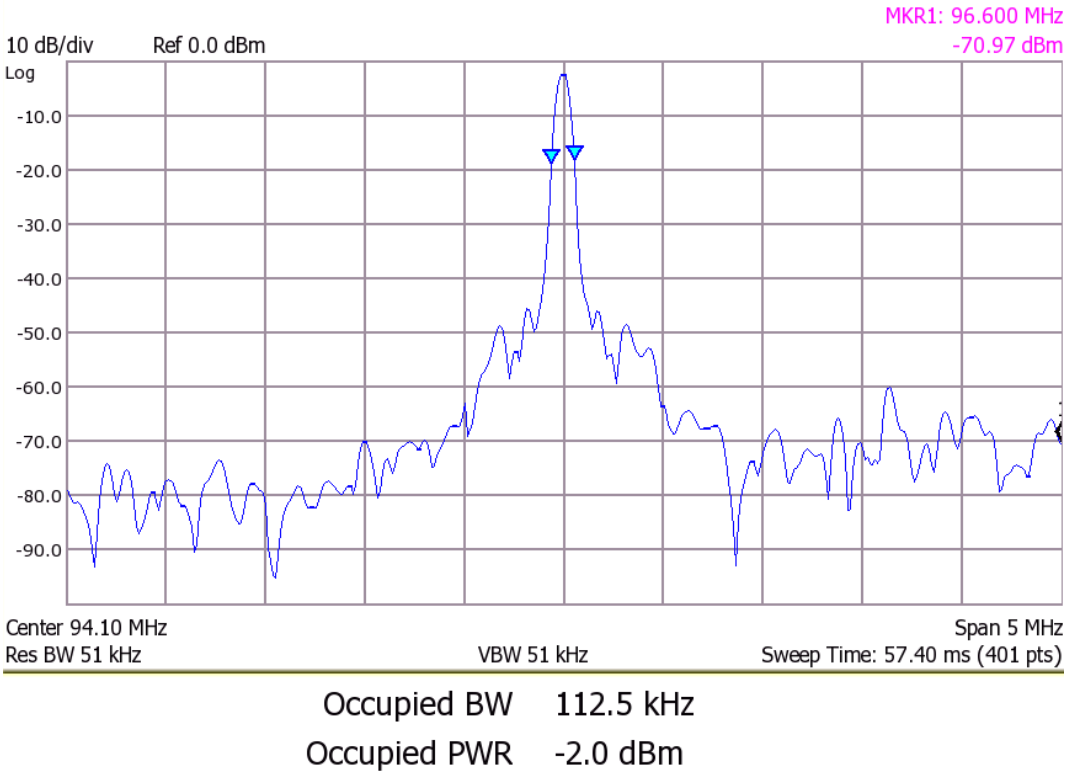


Figure 53 - Spectrogram of FM transmission test at 94.1 MHz.

Figure 53 shows a spectrogram of the transmission at 94.1 MHz, which is the frequency that was used to transmit the Mario song across the building to a portable FM radio receiver. The waveform seen here on the spectrum analyzer is very high quality since its peak is over 40 dB greater than any of the surrounding noise. The peak itself is a very clean shape, which falls off neatly at the edges of its bandwidth. The power level of the signal shown in the spectrogram in Figure 53 is -2.0 dBm, which is equal to 0.6 mW.

The 2.5 GHz transmission yielded much less favorable results, which are shown in Figure 54. It can be seen that the peak power in the waveform is -27.5 dBm, or 1.78 μW. This is roughly 354 times less power than when transmitting at 94.1 MHz. A frequency dependent loss of this magnitude is most likely due to imperfect impedance matching across the design and power losses in the PCB traces.

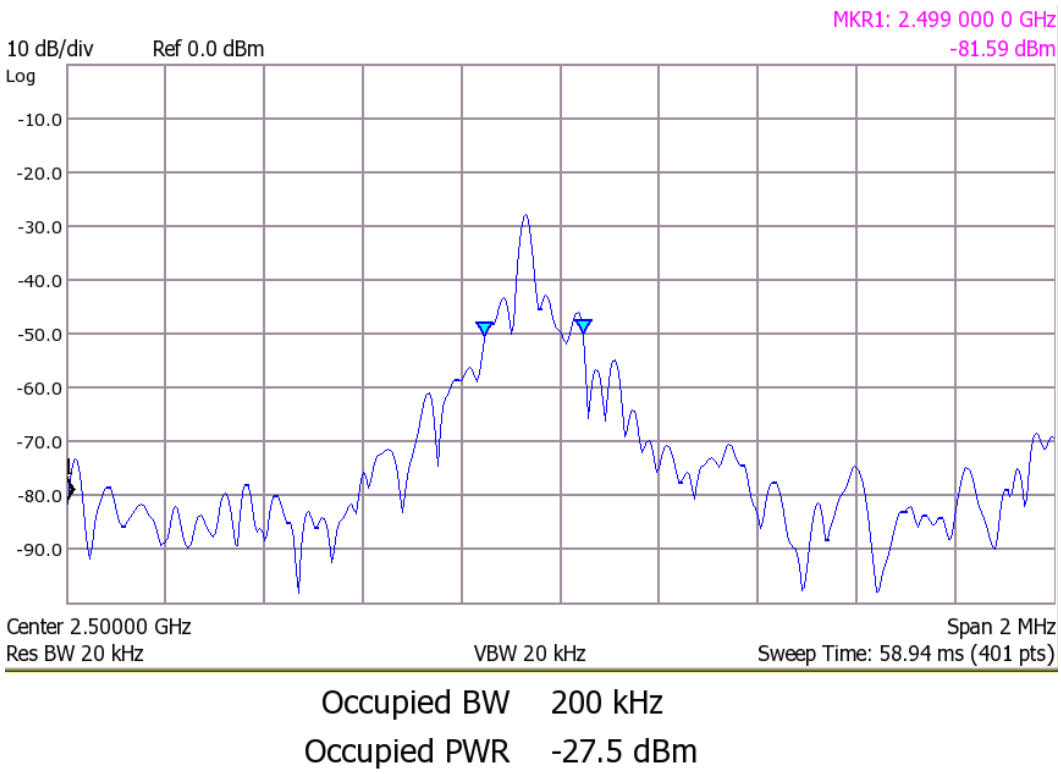


Figure 54 - Spectrogram of FM transmission test at 2.5 GHz.

When comparing Figure 53 to Figure 54, it is very important to note the overall shape and height of the peaks. In Figure 53, the peak is distinctly higher and more cleanly shaped than the peak in Figure 54. This clearly defined peak makes it much easier for the receiver to separate the desired signal from the accompanying noise by giving it a much higher signal to noise ratio (SNR).

The board was evaluated logarithmically across the entire range of frequencies that it was designed to operate across. The results from this evaluation can be seen in Table 3.

Table 3 - Transmitter experimental results

Transmit Frequency (MHz)	Image Frequency (MHz)	Power (dBm)	Next Peak (dBm)	Delta (dBm)	Image Power (dBm)	Next Peak (dBm)	RF2052 VCO
5.0	605.0	-9.3	-21.8	-12.5	-4.8	-14.0	3
6.9	606.9	-6.0	-22.0	-16.0	-4.7	-14.6	3
9.6	609.6	-4.2	-29.0	-24.8	-4.6	-15.2	3
13.3	613.3	-2.6	-28.0	-25.4	-4.5	-16.0	3
18.5	618.5	-2.3	-28.1	-25.8	-4.6	-15.6	3
25.7	625.7	-2.3	-30.0	-27.7	-5.0	-26.2	3
35.6	635.6	-2.4	-28.3	-25.9	-5.6	-23.9	3
49.4	649.4	-2.4	-28.9	-26.5	-6.0	-31.1	3
68.5	668.5	-2.3	-28.3	-26.0	-5.3	-31.6	3
94.9	694.9	-2.2	-28.0	-25.8	-5.2	-30.3	2
131.7	731.7	-2.5	-27.9	-25.4	-6.5	-30.3	2
182.6	782.6	-2.5	-21.8	-19.3	-7.2	-30.8	1
351.3	951.3	-3.1	-28.7	-25.6	-9.7	-35.1	3
487.2	1087.2	-4.5	-29.7	-25.2	-10.0	-34.7	2
675.7	75.7	-5.1	-30.8	-25.7	-2.6	-28.4	2
937.1	337.1	-9.1	-33.6	-24.5	-3.1	-28.8	3
1299.7	699.7	-12.5	-28.4	-15.9	-5.7	-19.5	1
1802.6	1202.6	-16.9	-42.6	-25.7	-13.6	-37.9	2
2500.0	1900.0	-27.6	-34.8	-7.2	-19.6	-25.5	1

The first column in Table 3 is the frequency which the transmitter was set to using GNU radio. The second column is the image frequency that is created by the RF2052 mixer and transmitted along with the desired frequency. The third column, marked as "Power" is the power level of the signal at the desired transmit frequency. The column marked "next peak" is the power level of the highest power spurious emission within 50 MHz of either side of the desired signal, which is then compared to the power of the desired signal in the delta column. An example of how this measurement was made can be seen in Figure 55.

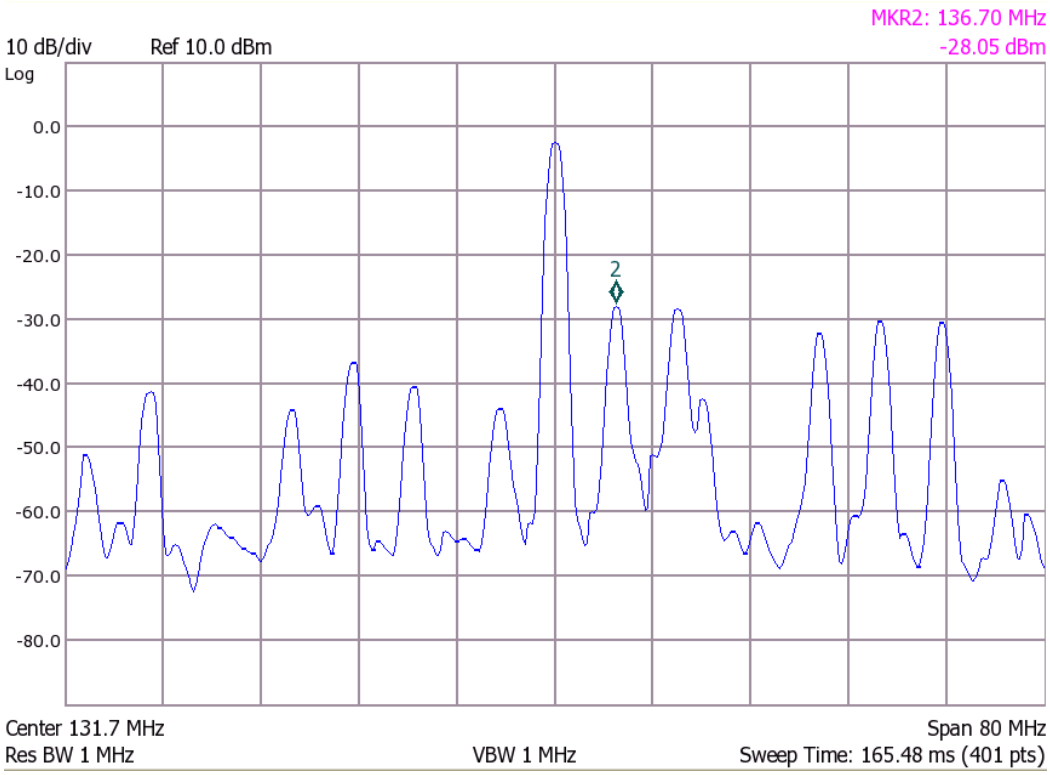


Figure 55 - Marker designates what was considered to be the next peak when making the measurements found in Table 3

Table 3 also has a column labeled "image power," which is the power of the image frequency. The next column is analogous to the first "next peak" column but for the image frequency. The final column notes which VCO in the RF2052 is used to generate the desired transmit frequency.

Plotting the data in Table 3 reveals the transmit power over the expected frequency range and is thus an effective tool to determine whether or not the transmitter met the design objectives outlined in the beginning of this report. Figure 56 shows the measured signal output power versus frequency from 5 to 2500 MHz, as measured by the spectrum analyzer connected directly to the SMA connector on the transmitter. As is evident in the image, the transmitter is effective over the range initially proposed; from 100 to 1300 MHz. Between 10 and 250 MHz the board performs extremely well, with an output power of roughly 1 mW, with the desired transmit signal around 25 dB higher than all spurious emissions around the signal. From about 350 MHz to 2500 MHz the output power decreases all the way down to about -28 dBm. Despite this decrease, the signal is still well above the greatest spurious emission in a ±50 MHz range around the signal, and thus still usable in many situations. Figure 58 shows the differences between signal power and spurious emission power over the same frequency range.

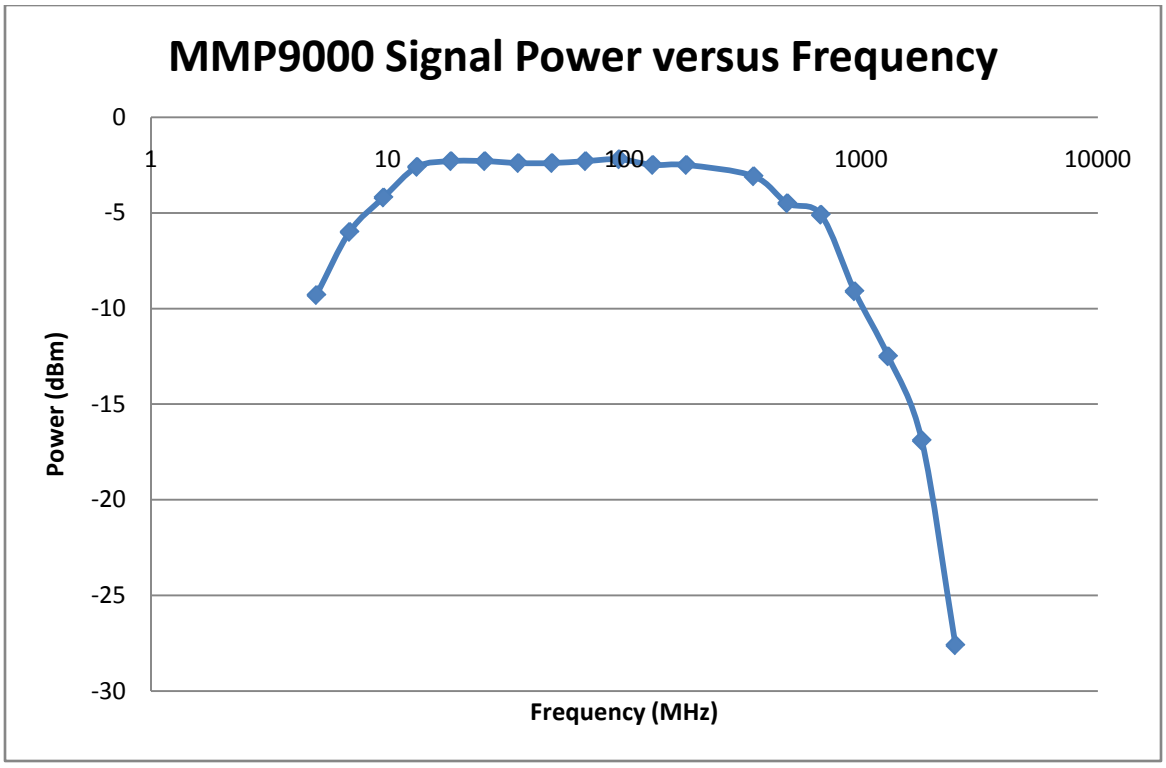


Figure 56 - Signal frequency versus power from 5 MHz to 2500 MHz.

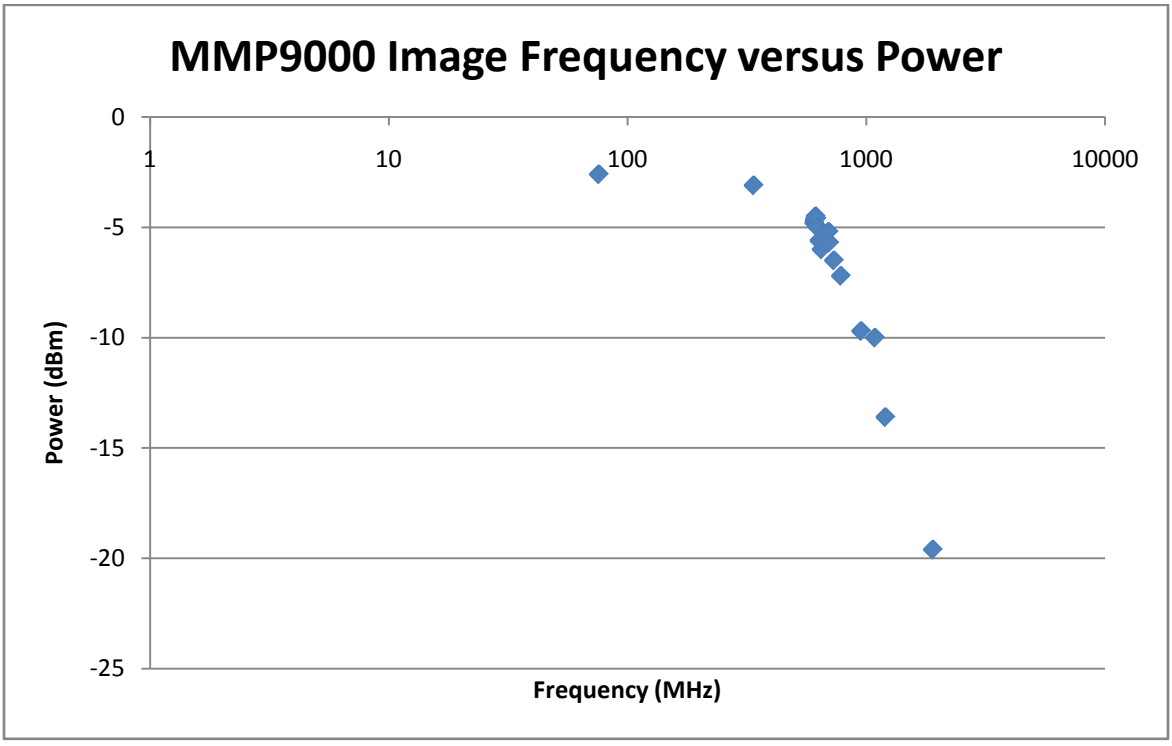


Figure 57 - Signal image frequency versus power for signals ranging from 5 to 2500 MHz.

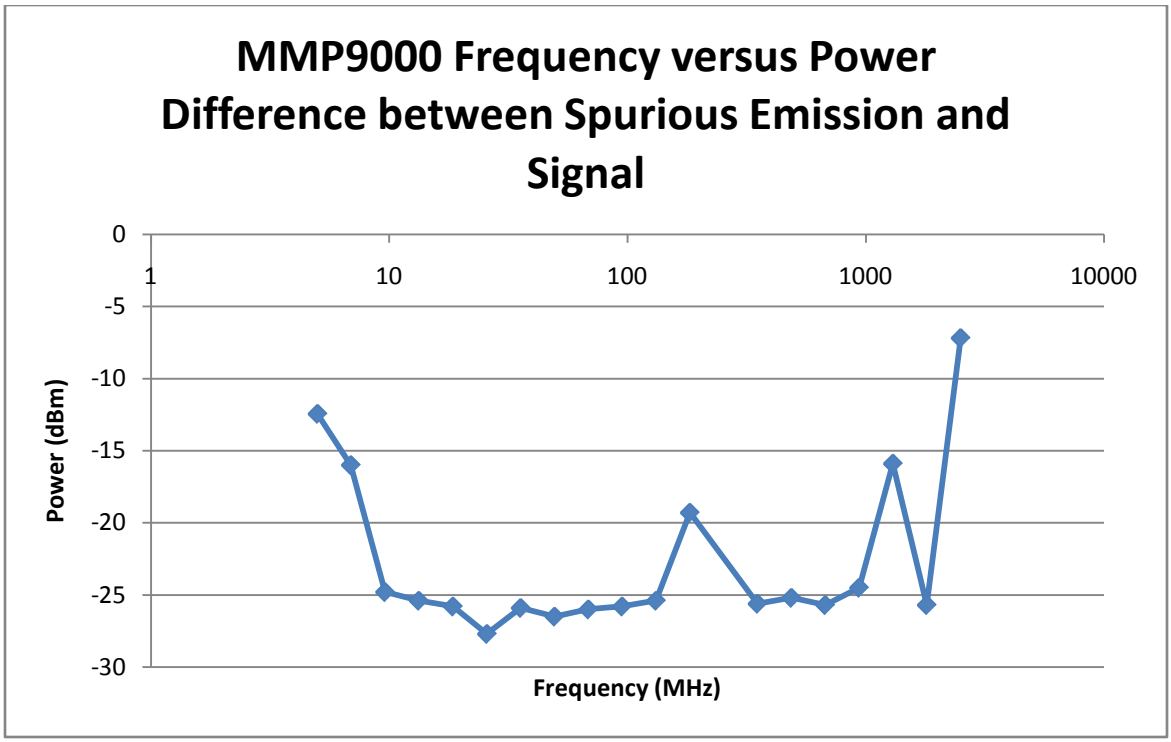


Figure 58 - Difference between signal power and spurious emission power for signals ranging from 5 MHz to 2500 MHz.

The reason for the VCO column in Table 3 is that during the testing, it was noted that a few of the frequencies produced many spurious emissions around the desired transmission frequency. These frequencies can be seen in Figure 58 as 182.6 MHz, 1299.7 MHz, and 2500 MHz. When the VCO that was used for each transmission was noted, it was found that the transmissions that had the numerous spurious emissions used VCO1 in the RF2052. To test the theory that VCO1 creates numerous spurious emissions while the other two do not, two transmissions at the same frequency were examined, one using VCO1 and one using VCO2. The frequency for the LO in the RF2052 that was chosen was 1972 MHz divided by 4, or 493 MHz. This frequency was chosen because according to the datasheet, 1972 MHz can be generated by both VCO1 and VCO2, and the divide by 4 is used to bring the frequency lower to rule out the possibility of the cause of the spurious emissions being that the frequency is too high. Figure 59 shows this transmission using VCO2, and Figure 60 shows it using VCO1.

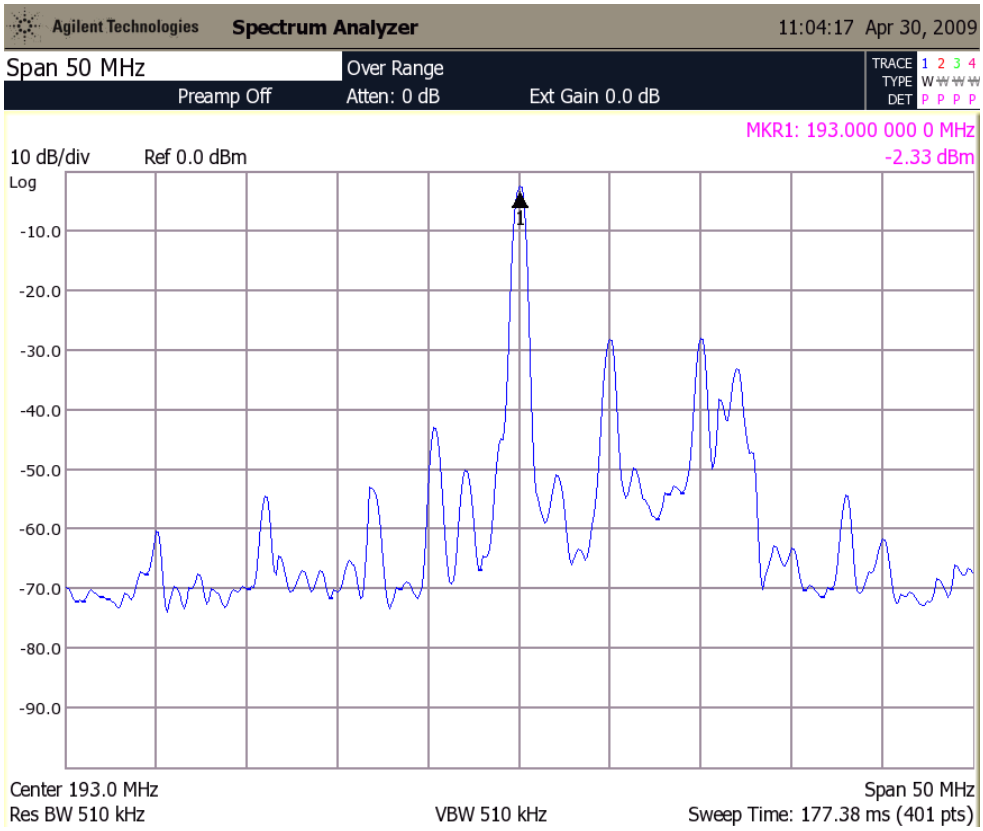


Figure 59 - Transmission with RF2052 LO frequency set to 493 MHz using VCO2.

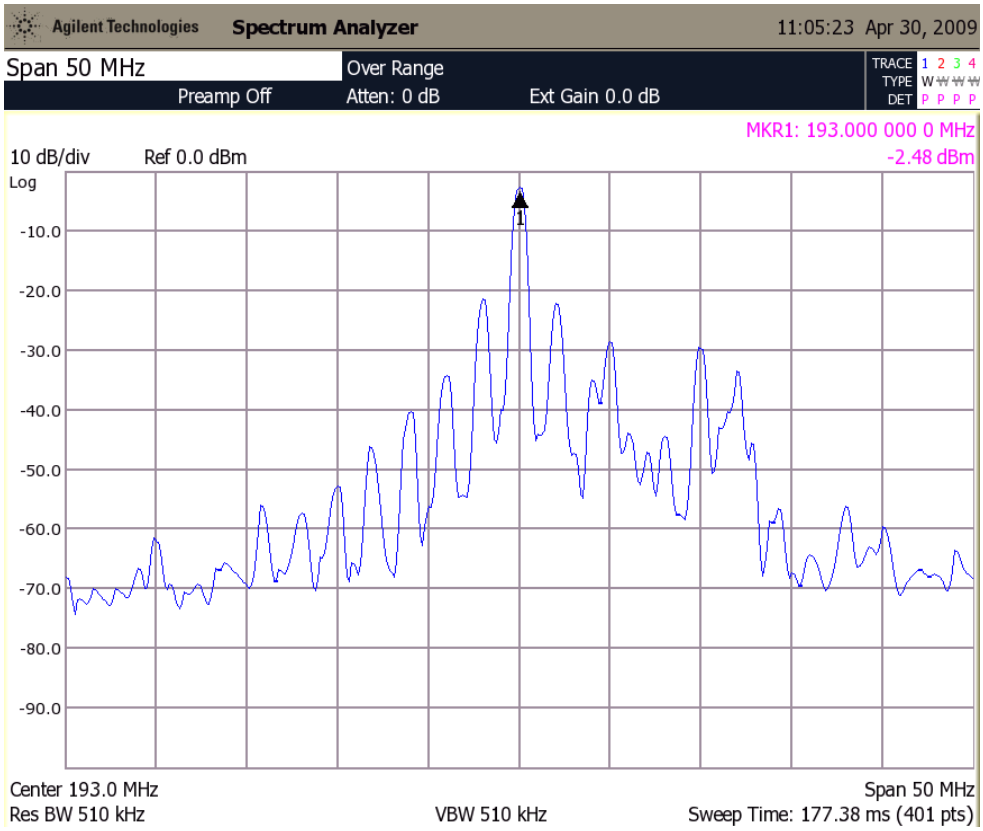


Figure 60 - Transmission with RF2052 LO frequency set to 493 MHz using VCO1.

It can be seen from these figures that while both VCOs are capable of generating the same desired frequency, and both transmissions show the desired signal at the same power level, when VCO1 is used there are many more spurious emissions surrounding the desired frequency, and they are at higher power levels as well. Initially it was thought that this behavior was solely the result of VCO1. However, after doing some more testing into the matter, it was found that VCO2 could generate the exact same spectrogram as seen in Figure 60 by increasing its mixer current above 5 mA.

Unfortunately it was never determined what really causes these many spurious emissions, why they occur with VCO2 and VCO3 with a mixer current above 5 mA and at all mixer current levels with VCO1, and if it is even actually a problem or not. Further investigation into the hardware and all the RF2052 register settings would have to be done, but due to time constraints no further investigation was possible.

7.2 Receiver

Tuning the receive side in software was almost identical to tuning the transmit side. The only difference was the addition of code to set the low speed DAC to control the variable gain amplifier (VGA) control pin on the AD8348 (quadrature demodulator). All initial tests of the receive side, however, were unsuccessful. FM transmissions were sent with a second USRP over

a coax cable using at both low frequencies using a Basic TX board and high frequencies using an RFX2400 board. Since the receive functionality did not appear to work, the output from the two local oscillators were checked using a spectrum analyzer to make sure they were functioning properly. The output from the ADF4360-8 was confirmed to be working, as the frequency of its output matched the frequency programmed to it in GNU Radio. Next, the output of the RF2052 was checked to make sure the leakage from the local oscillator was there. Again, the programmed frequency could be seen.

These results showed that while indeed both the local oscillators were functioning correctly, the signal from the other USRP was not successfully being mixed, since it was not being received in GNU Radio. It was next decided to check the output of the RF2052 while sending a signal from the other USRP to see if it was being mixed at all after the first mixer. A 100 MHz FM signal was sent from the other USRP, and the RF2052 LO frequency was programmed to 400 MHz to mix the signal to 300 MHz. The second LO (ADF4360-8) was set to 295 MHz and the USRP's digital downconverter LO was set to 5 MHz to ultimately bring the 100 MHz FM signal to DC. When this test was first run, as before, the signal was not received by GNU Radio. However, as soon as the spectrum analyzer probe was attached to the output of the RF2052 mixer to see if the signal could be seen at 300 MHz, the audio encoded in the FM signal started being received by GNU Radio! The signal, however, could not be seen at 300 MHz as expected. This led to a detailed investigation into the details of the interfacing requirements of both the RF2052 and AD8348 ICs.

It was eventually found after reading the mixer section in the datasheet in detail that the RF mixer in the RF2052 receives its DC operating current through its output port in a similar fashion to that of the amplifiers used in the design. The reason that this was missed during the design of the circuit was because the differential output of the mixer is typically converted to a single ended 50 Ω output via a center tapped balun, which happens to provide this DC current. Because the differential output of this mixer goes into a differential input on the AD8348, each with a differential impedance of 200 Ω, it appeared to be a perfect match and not require any interfacing circuitry other than AC coupling capacitors, and thus the DC current supply requirement was overlooked.

To correct this problem, both lines of the differential output of the RF2052 mixer were connected to 3.3 V through the ADCH-80A RF chokes before the AC coupling capacitors, as done with the amplifiers. A photo of this modification is shown below in Figure 61.

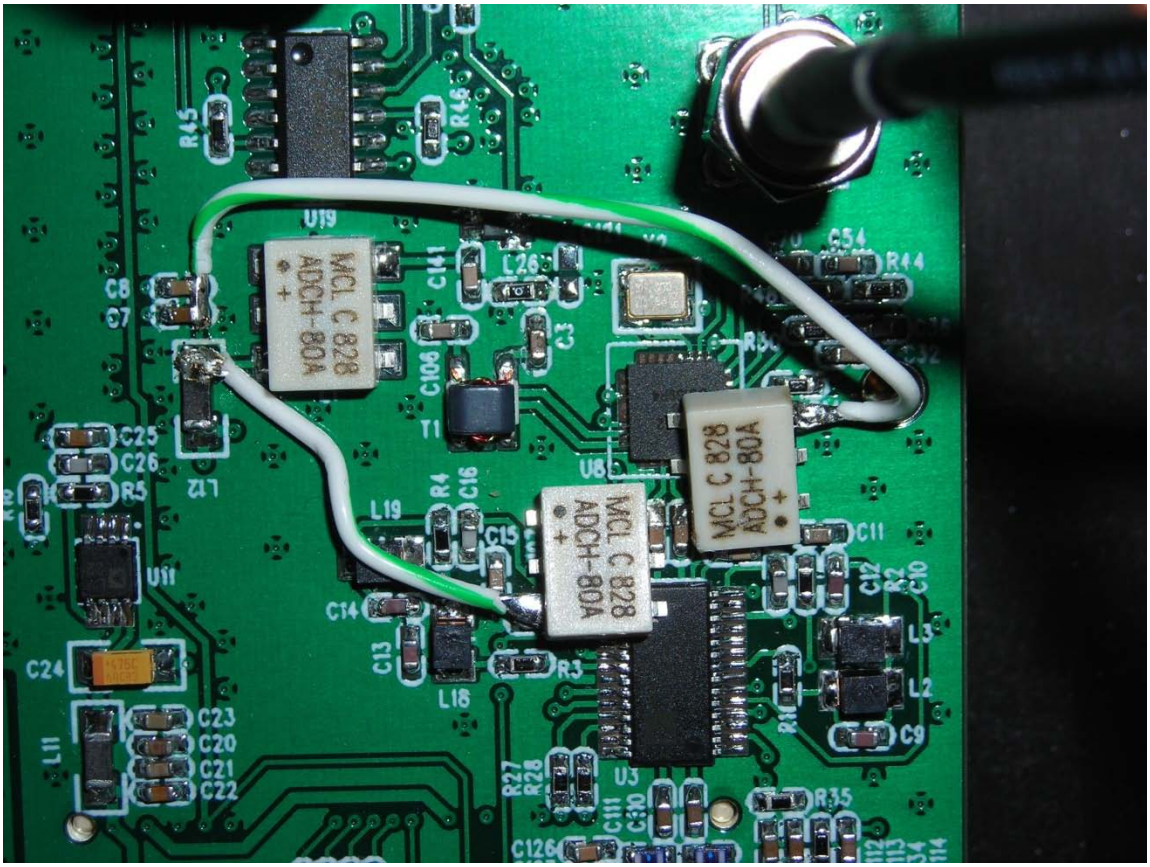


Figure 61 - Addition of RF chokes to the output of the RF2052 mixer.

The same 100 MHz transmission test described above was run again once the RF chokes were added. Figure 62 below shows the 100 MHz FM signal generated by the second USRP after the MGA−82563 LNA.

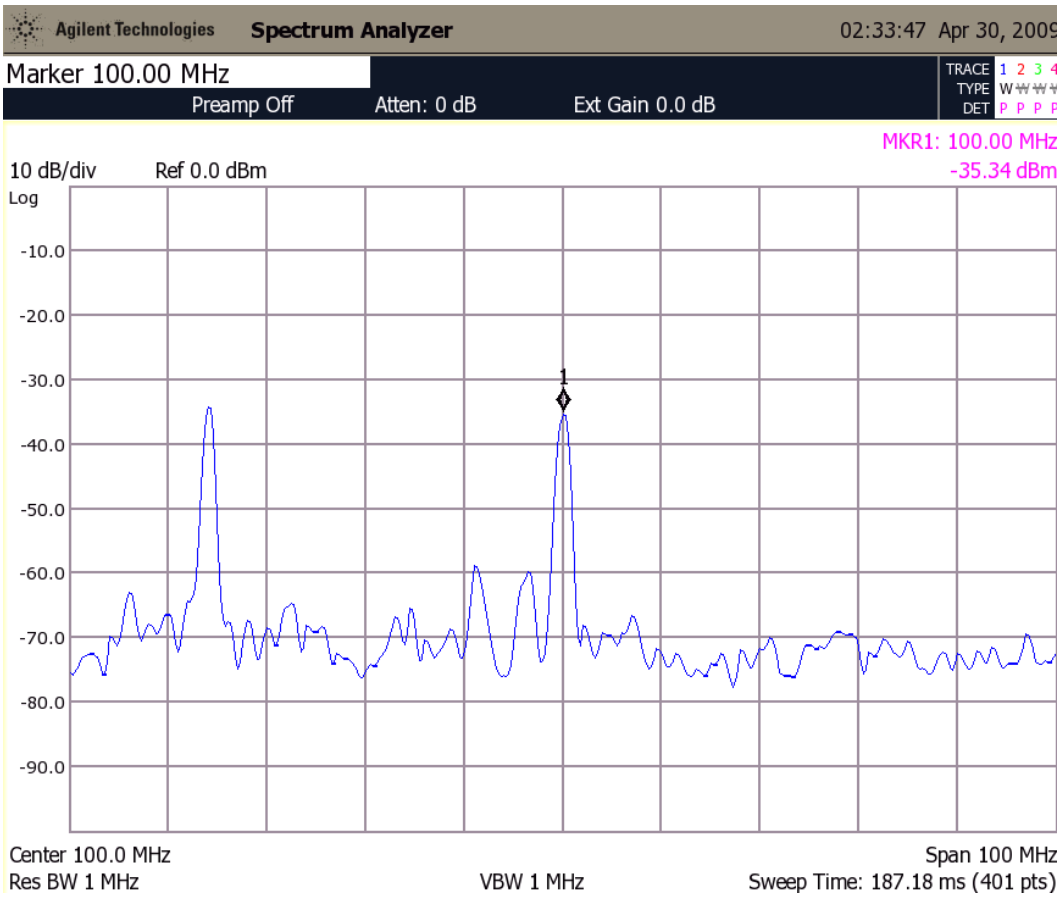


Figure 62 - 100 MHz signal from Basic TX board generated for testing the receiver.

After the LNA, the single ended 100 MHz signal is converted to a differential signal and fed into the mixer input of the RF2052. With the RF2052 LO programmed to 400 MHz, this signal is expected to be found at 300 MHz at the mixer output as described above. This time, with the RF chokes in place, that is exactly what happens, and can be seen below in Figure 63.

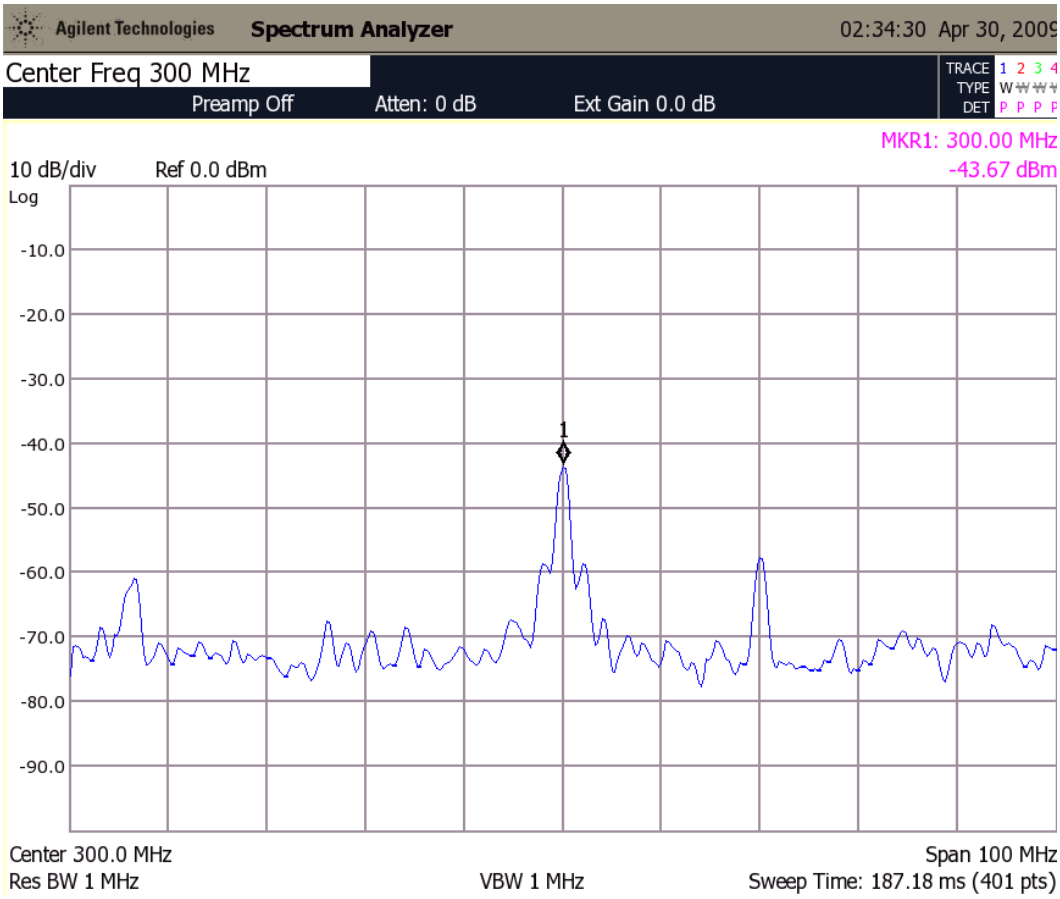


Figure 63 - 100 MHz signal mixed to 300 MHz. This shows that the LO and mixer in the receive side RF2052 are both working.

This result shows the RF2052 mixer now working. Unfortunately, the signal still would not be received by GNU Radio. The LO input to the AD8348 demodulator was again checked to make sure it was 295 MHz as programmed. Figure 64 below shows the spectrum analyzer confirming that it indeed was.

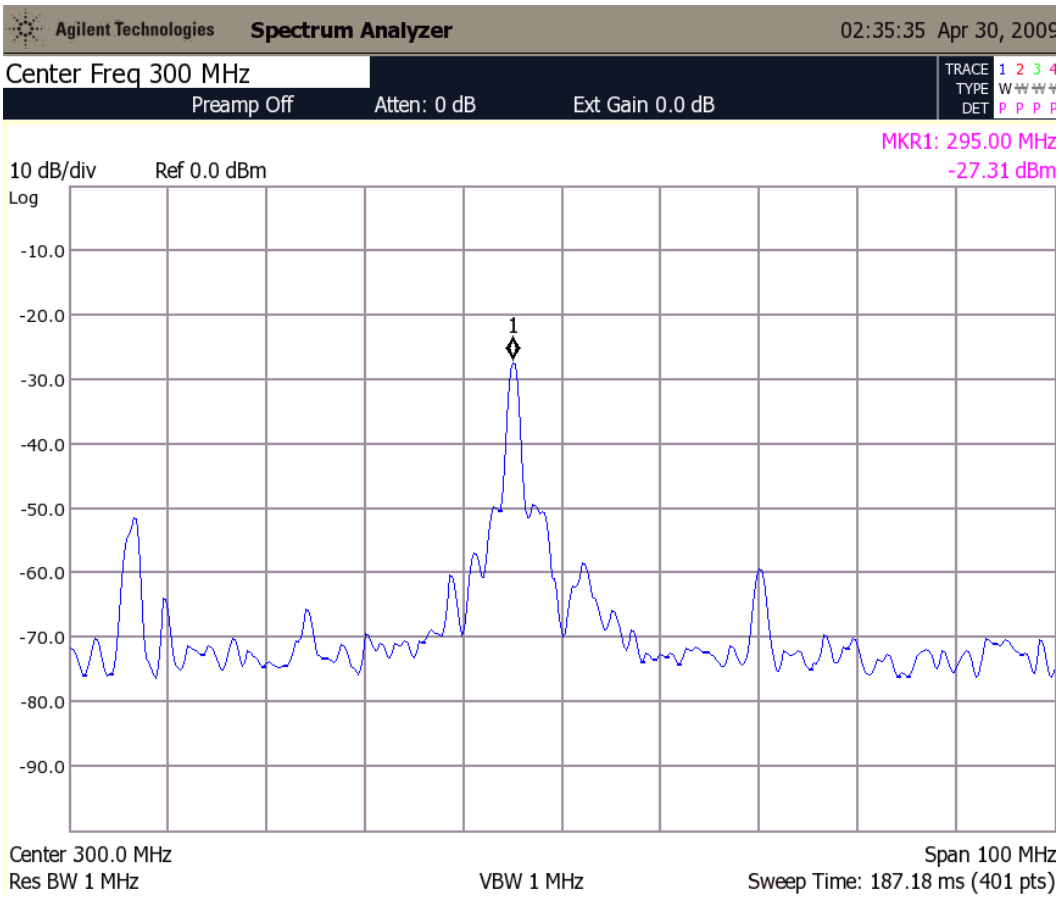


Figure 64 - Local oscillator signal for the AD8348 quadrature demodulator at 295 MHz. This shows that the ADF4360-8 frequency synthesizer is working correctly.

The IF signal of 300 MHz along with the LO signal of 295 MHz both going into the demodulator should result in a 5 MHz signal at the output. However, when the output was checked, no such signal could be found as shown below in Figure 65.

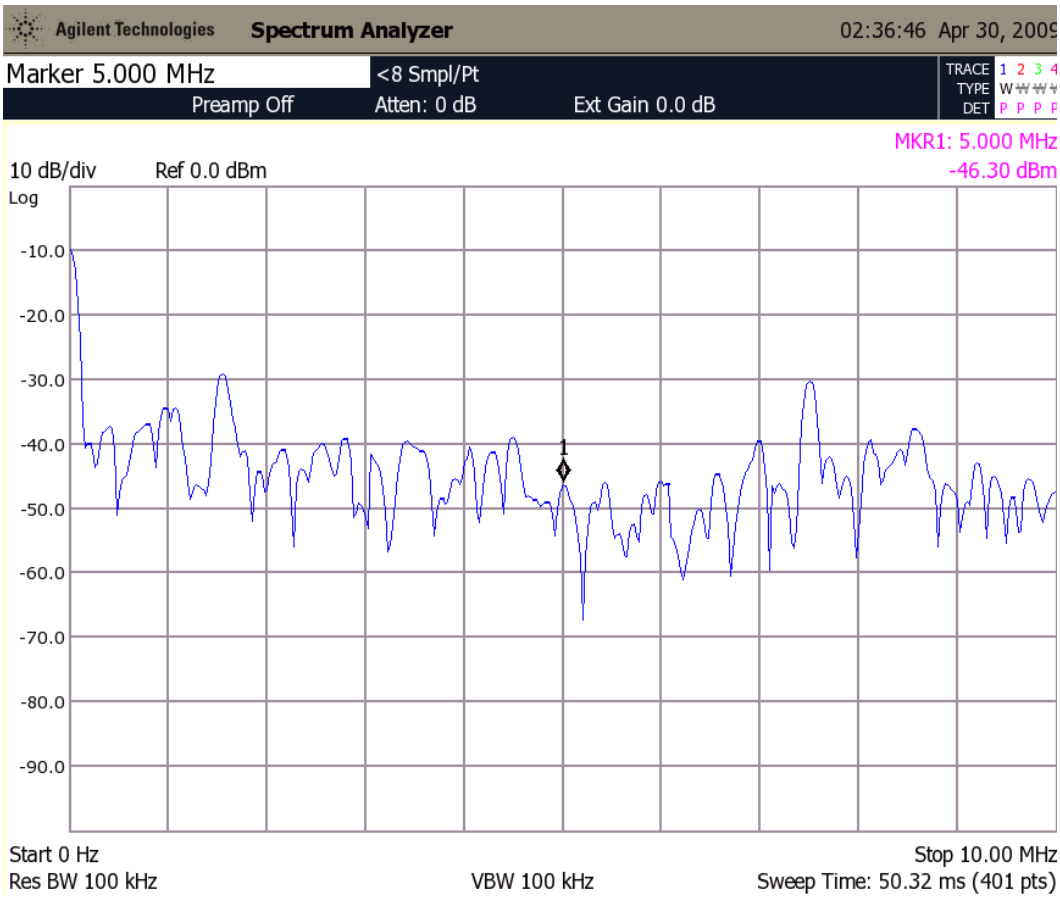


Figure 65 - Output from the AD8348 quadrature demodulator. The input signal is at 300 MHz, and the LO is at 295 MHz. The output should be a signal at 5 MHz. The fact that it is not indicates that the IC is damaged.

As the AD8348 is receiving the correct voltage of 5 V at its power pins, and its enable pins are set correctly, and because everything else appears to be correct, the most likely explanation for it not functioning correctly is that the chip itself is damaged. Unfortunately, due to time constraints it was not possible to replace the chip with another one to find out for sure, and thus further testing of the receive side was not possible.

7.3 Chapter Summary

It is clear that there are still a few minor issues with the design that need to be ironed out in a final revision of the board. The issues with the VCOs in the mixer in the transmit side can be proven to be either a faulty chip or just how the design functions after another PCB is assembled. The extra components can easily be fit into the design for future boards. Some of the possibilities for problems with the demodulator can be eliminated by simply replacing it with a new chip, in case the current one was damaged before or during assembly.

Overall, the results point to a successful design since it was able to transmit a usable signal over the entire desired range without changing any hardware on the board. The few remaining issues with the receive side will most likely be ironed out within the next week and final recommendations for a working product will be made.

8 Conclusion

Despite the progress made in testing and demonstrating the functionality of the transmitter and receiver, there are still some pressing issues that should be addressed if someone else is interested in continuing this project. These issues include the following:

- Filter design
- Fully integrate daughterboard control in GNU Radio
- Better board design, matching circuits, power output
- Standalone unit design

Since the final design of the daughterboard has no image rejection filters, there is a lot of spurious emissions on the output of the transmitter. Figure 66 shows the emissions associated with a transmitted signal of 9 MHz. Unfortunately, the frequency range of the daughterboard makes it rather difficult to design any type of filter to cover the entire range of the transmitter. One questionably feasible approach to this problem would be to design a software controlled bank of filters that would change the signal path according the transmitted signal. The complexity of this design and costs associated with creating it may outweigh the benefit of having such a filter bank, since communications can be established without such filtering, but operation of such a device at higher power outputs would be prohibited by the FCC.

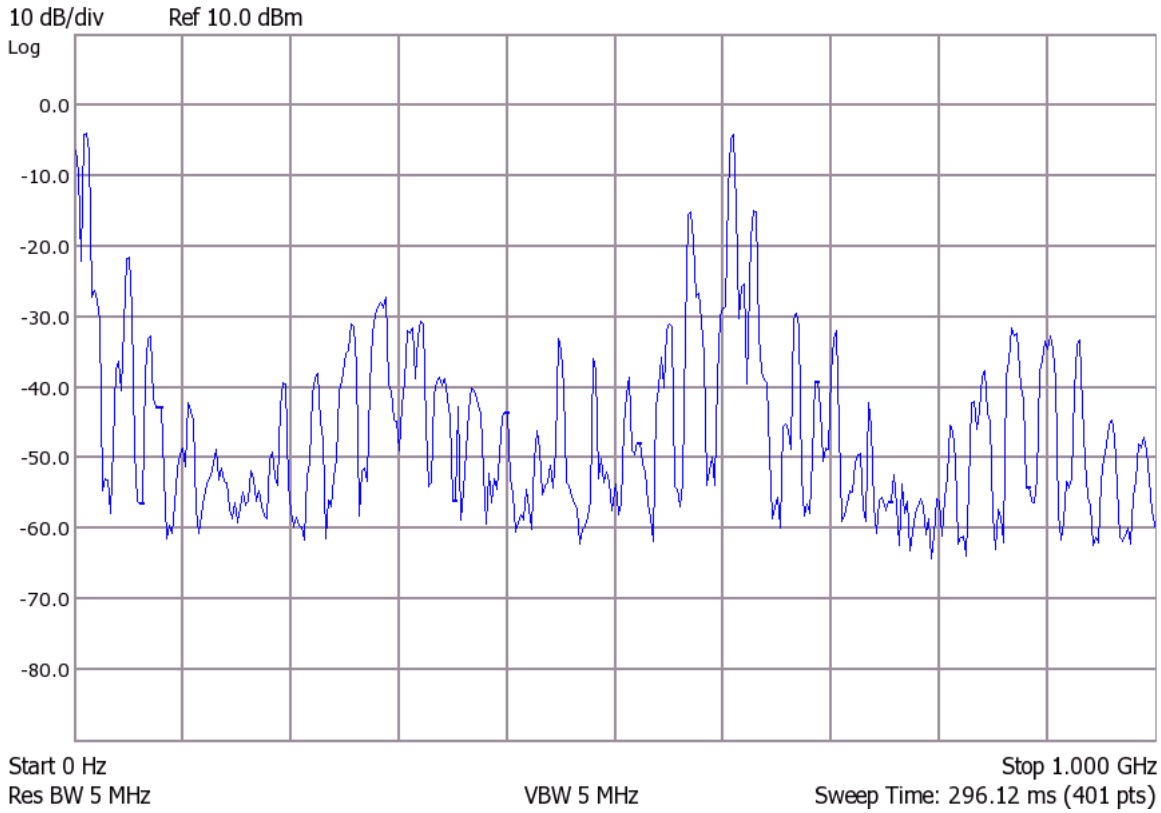


Figure 66 - Image frequency and spurious emissions on the output of the transmitter.

The hardware design was a major focus throughout this project and as a result, there is still a lot of work that can be done to fully integrate the board into GNU Radio. A separate entry would need to be created in GNU Radio that contains all of the appropriate registers and functions to control the daughterboard. In addition, the code for the daughterboard would need to be included in future releases of GNU Radio. Unless the MMP9000 goes into production, further code incorporation is not entirely necessary.

The next point for improvement is the overall PCB design in regards to parasitic effects of the PCB traces. At higher frequencies the gain is attenuated and thus transmits at lower power. Improved design could take a serious look at trace impedance calculations and optimize the PCB for maximum efficiency.

Finally, there are many more possible applications for this design should it be made into a stand alone unit. Although this task is largely unnecessary because the transceiver was designed to operate with GNU Radio and the USRP, the components used in the design could be adapted to other applications or software radio platforms. A standalone unit could help reduce cost while still providing a large frequency range for cognitive radio applications.

The final version of the PCB could not have been possible to obtain without the careful planning and methodical work described throughout this report. This began with the initial design, and led to the need to test as many parts as possible using simulation software. From there a few changes were made to the schematics before importing them into a PCB layout program. Quite a lot of information was learned from the existing daughterboards, whose designs were taken into consideration throughout the design of this PCB.

The first version of the PCB was not without errors. After all the parts had been soldered down, a long debugging process began to try to correct the errors. This process included comparing the PCB layout against the final design schematics as well as verifying all decisions that were made in the schematics against what was recommended in the datasheets. This process eventually led to the team requesting the help of Matt Ettus, the lead designer of the USRP and its daughterboards, who was able to offer suggestions that were later incorporated into a second version of the PCB.

The second version of the PCB is what was used to obtain the data for the results section, as its transmit side works better than expected. The board was able to transmit a clearly audible FM signal across an entire building and then switch to a frequency almost 30 times higher without any kind of hardware modifications.

There were three objectives that were put forth at the beginning of this project. The first of these was to design and implement an RF frontend that is tunable between 100 MHz and 1.3 GHz. However, due to the particular component choices and topology chosen for the design, the final working frequency range for the board is between 50 MHz and 2.5 GHz, which clearly exceeds the first objective.

The second objective was to design the RF frontend described in the first objective as a daughterboard for the USRP and thus to work with GNU Radio. This meant that less hardware and software would have to be designed and implemented to test the frontend, since it would allow it to work with an existing software radio testbed. Also, the USRP is a strong platform to develop for because of its open nature. This means that all documentation for the USRP is freely available online, which made this project much easier to develop. This objective is very close to being met, since the board is a fully compatible daughterboard for the USRP, and there is already some code written for it to allow it to be controlled by GNU Radio, although it is not fully integrated into the GNU Radio software as other daughterboards are.

The third objective was to have the device capable of both full and half duplex operation. This functionality was desired because there are already daughterboards made for the USRP that have this functionality. This is possible because there are DACs and ADCs on the USRP motherboard that have been designed to transmit and receive simultaneously. The two SMA connectors and antenna switch configuration in our design give it the ability to route either the receive or transmit path to one antenna, or to route both paths to their own antennas at the same time. This allows it to switch between full and half-duplex modes. The antenna switch configuration has been confirmed to work and be controllable from within GNU Radio, and thus the third and final objective has been met.

There will always be a need for radios smaller radios with greater capabilities, suggesting that the analog hardware must become simpler while the software becomes more complex. This project sought to anticipate this trend in radio design, opening up more frequencies for use in software radio and cognitive radio applications. The goal of a wideband transceiver is to reduce the number of RF frontends that are required to cover a large range of frequencies. Having several different frontends can be very costly, since it requires more components, board real estate, and power. This project is important because it demonstrates the feasibility, versatility, and affordability of a widely tunable RF frontend.

9 Bibliography

[1] Auction 73, 700 MHz Band. (1/24/2008 - 3/18/2008) FCC Wireless Spectrum Auction. [Online]. http://wireless.fcc.gov/auctions/default.htm?job=auction_summary&id=73

[2] M. A. McHenry, D. McCloskey, D. Robertson, J. MacDonald, "Spectrum Occupancy Measurements, Chicago, Illinois, November 16-18, 2005," Shared Spectrum Company, Dec. 2005. [Online]. http://www.sharedspectrum.com/measurements/download/NSF_Chicago_2005-11_measurements_v12.pdf

[3] M. A. McHenry, D. McCloskey, G. Lane-Roberts, "Spectrum Occupancy Measurements, Location 4 of 6: Republican National Convention, New York City, N.Y., Aug 30, 2004 - Sept. 3, 2004, Revision 2," Shared Spectrum Company Report, Aug. 2004. [Online]. http://www.sharedspectrum.com/inc/content/measurements/nsf/4_NSF_NYC_Report.pdf

[4] J. Mitola, III, "Software radios: Survey, critical evaluation and future directions," *Aerospace and Electronic Systems Magazine, IEEE*, vol. 8, no. 4, pp. 25-36, Apr. 1993.

[5] R.I. Lackey, D. W. Upmal, "Speakeasy: the military software radio," *Communications Magazine, IEEE*, vol. 33, no. 5, pp. 56-61, May 1995.

[6] V. Bose, "A Software Driven Approach to SDR Design," COTS Journal, Jan. 2004. [Online]. http://www.cotsjournalonline.com/home/article.php?id=100056

[7] J. Chapin, V. Bose, "The Vanu Software Radio System," in *2002 Software Defined Radio Technical Conference*, San Diego, 2002. [Online]. http://www.vanu.com/wp-content/resources/publications/2002the_vanu_software_radio_system.pdf

[8] SDR Forum. (2007, Nov.) SDR Forum. [Online]. http://www.sdrforum.org/pages/documentLibrary/documents/SDRF-06-R-0011-V1_0_0.pdf

[9] J. Reed, *Software Radio: A Modern Approach to Radio Engineering*. Upper Saddle River, NJ, United States of America: Prentice Hall Press, 2002.

[10] R. Vidano, "SPEAKeasy II-an IPT approach to software programmable radio development," *MILCOM 97 Proceedings*, vol. 3, pp. 1212-1215, Nov. 1997.

[11] J. Mitola, III, *Cognitive Radio: An Integrated Agent Architecture for Software Defined Radio*. Kista, Sweden: Royal Institute of Technology (KTH), 2000.

[12] R. Merritt. (2006, Dec.) EE Times. [Online]. http://www.eetimes.com/disruption/interviews/mitola.jhtml

[13] P. Kolodzy, "Spectrum Policy Task Force," Federal Communications Commission, 2002.

[14] Vardalas, Mroth, Nbrewer. (2008, Aug.) Heinrich Hertz (1857-1894) - GHN. [Online]. http://www.ieeeghn.org/wiki/index.php/Heinrich_Hertz_(1857-1894)

[15] Brownback, Sen. Sam, "A bill to increase the penalties for violations by television and radio broadcasters of the prohibitions against transmission of obscene, indecent, and profane language," Apr. 2005.

[16] American Amateur Radio League. (2008, Dec.) American Amateur Radio League. [Online]. http://www.arrl.org/aarrl.html

[17] J. E. Cooley, "A day in the life of the RF spectrum," Master Thesis, Massachusetts Institute of Technology, 2005.

[18] Laskar, J. and Mukhopadhyay, R. and Hur, Y. and Lee, C.-H. and Lim, K., "Reconfigurable RFICs and modules for cognitive radio," *Silicon Monolithic Integrated Circuits in RF Systems, 2006. Digest of Papers. 2006 Topical Meeting on*, pp. 4-, Jan. 2006.

[19] Cabric, D. and Mishra, S.M. and Brodersen, R.W., "Implementation issues in spectrum sensing for cognitive radios," *Signals, Systems and Computers, 2004. Conference Record of the Thirty-Eighth Asilomar Conference on*, vol. 1, pp. 772-772Vol1, Nov. 2004.

[20] Lackey, R.I. and Upmal, D.W., "Speakeasy: the military software radio," *Communications Magazine, IEEE*, vol. 33, no. 5, pp. 56-61, May 1995.

[21] RF Micro Devices. (2009, Feb.) RFMD. [Online]. http://www.rfmd.com/pdfs/2052DS.pdf

10 Appendix A - Source Code

10.1 fmtx_mmp.py

```
#!/usr/bin/env python
#
# Copyright 2005,2007 Free Software Foundation, Inc.
#
# This file is part of GNU Radio
#
# GNU Radio is free software; you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation; either version 3, or (at your option)
# any later version.
#
# GNU Radio is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with GNU Radio; see the file COPYING.  If not, write to
# the Free Software Foundation, Inc., 51 Franklin Street,
# Boston, MA 02110-1301, USA.
#

from gnuradio import gr, eng_notation
from gnuradio import usrp
from gnuradio import audio
from gnuradio import blks2
from gnuradio.eng_option import eng_option
from optparse import OptionParser
from usrpm import usrp_dbid
import time
import math
import sys

from gnuradio.wxgui import stdgui2, fftsink2
#from gnuradio import tx_debug_gui
import wx

SPI_ENABLE_TX_A  = usrp.SPI_ENABLE_TX_A
SPI_ENABLE_RX_A  = usrp.SPI_ENABLE_RX_A
SPI_ENABLE_TX_B  = usrp.SPI_ENABLE_TX_B
SPI_ENABLE_RX_B  = usrp.SPI_ENABLE_RX_B
SPI_FMT_LSB      = usrp.SPI_FMT_LSB
SPI_FMT_MSB      = usrp.SPI_FMT_MSB
SPI_FMT_HDR_MASK = usrp.SPI_FMT_HDR_MASK
SPI_FMT_HDR_0    = usrp.SPI_FMT_HDR_0
SPI_FMT_HDR_1    = usrp.SPI_FMT_HDR_1
SPI_FMT_HDR_2    = usrp.SPI_FMT_HDR_2


RFMD_ENX_BIT = 4
RFMD_SCLK_BIT = 3
RFMD_SDATA_BIT = 2
RFMD_ENABLE_BIT = 5
```

```
RFMD_MODE_BIT = 0

TR_SWITCH_BIT = 6

RFMD_ENX = (1 << RFMD_ENX_BIT)
RFMD_SCLK = (1 << RFMD_SCLK_BIT)
RFMD_SDATA = (1 << RFMD_SDATA_BIT)
RFMD_ENABLE = (1 << RFMD_ENABLE_BIT)
RFMD_MODE = (1 << RFMD_MODE_BIT)

TR_SWITCH = (1 << TR_SWITCH_BIT)


def _write_all(u, side, R, control, N):
      """
      Write R counter latch, control latch and N counter latch to VCO.

      Adds 10ms delay between writing control and N

      @param R: 24-bit R counter latch
      @type R: int
      @param control: 24-bit control latch
      @type control: int
      @param N: 24-bit N counter latch
      @type N: int
      """
      _write_R(u, side, R)
      _write_control(u, side, control)
      time.sleep(0.010)
      _write_N(u, side, N)

def _write_control(u, side, control):
      _write_it(u, side, (control & ~0x3) | 0)

def _write_R(u, side, R):
      _write_it(u, side, (R & ~0x3) | 1)

def _write_N(u, side, N):
      _write_it(u, side, (N & ~0x3) | 2)

def _write_it(u, side, v):
      s = ''.join((chr((v >> 16) & 0xff), chr((v >>  8) & 0xff), chr(v &
0xff)))
      # fix to write to A or B, not just B
      u._write_spi(0, SPI_ENABLE_TX_B, SPI_FMT_MSB | SPI_FMT_HDR_0, s)


def set_adf4360(u, side, target_freq):

      if target_freq < 280 or target_freq > 320:
            return -1

      # R-Register Common Values
      d_BSC   = 3  # bits 21,20 Div by 8 to be safe
      d_TEST  = 0  # bit 19
      d_LDP   = 1  # bit 18
      d_ABP   = 0  # bit 17,16   3ns
```

```
      d_R_DIV = 64 # bits 15:2

      # N-Register Common Values
      d_CPG   = 0  # bit 21  CP current setting 1 is permanently used
      d_B_DIV = int(target_freq) # bits 20:8 (300/64*16)=75

      # Control Register Common Values
      d_PD    = 0  # bits 21,20   Normal operation (3 powers down, 0 powers
up)
      d_CP2   = 7  # bits 19:17   2.5mA
      d_CP1   = 7  # bits 16:14   2.5mA
      d_PL    = 0  # bits 13,12   3.5mA
      d_MTLD  = 1  # bit 11       enabled
      d_CPG   = 0  # bit 10       CP setting 1
      d_CP3S  = 0  # bit 9        Normal
      d_PDP   = 1  # bit 8        Positive
      d_MUXOUT= 1  # bits 7:5     Digital Lock Detect
      d_CR    = 0  # bit 4        Normal
      d_PC    = 1  # bits 3,2     Core power 5mA

      control = (d_PD<<20) | (d_CP2<<17) | (d_CP1<<14) | (d_PL<<12) |
(d_MTLD<<11) | (d_CPG<<10) | \
            (d_CP3S<<9) | (d_PDP<<8) | (d_MUXOUT<<5) | (d_CR<<4) | (d_PC<<2)

      N = (d_CPG<<21) | (d_B_DIV<<8)

      R = (d_BSC<<20) | (d_TEST<<19) | (d_LDP<<18) | (d_ABP<<16) |
(d_R_DIV<<2)

      _write_all(u, side, R, control, N)

      return 0

def write_rfmd_reg(u, side, addr, data, mode):
      if mode:
            u.write_io(side, RFMD_MODE, RFMD_MODE)
      else:
            u.write_io(side, 0, RFMD_MODE)

      u.write_io(side, RFMD_SCLK, RFMD_SCLK)
      u._write_oe(side, RFMD_SDATA, RFMD_SDATA) # set sdata as output
      u.write_io(side, 0, RFMD_ENX|RFMD_SCLK|RFMD_SDATA)
      u.write_io(side, RFMD_SCLK, RFMD_SCLK)
      u.write_io(side, 0, RFMD_SCLK|RFMD_SDATA) # set write
      u.write_io(side, RFMD_SCLK, RFMD_SCLK)

      for i in range(7):
            databit = ((addr>>(6-i)) & 1) << RFMD_SDATA_BIT
            u.write_io(side, databit, RFMD_SDATA|RFMD_SCLK)
            u.write_io(side, RFMD_SCLK, RFMD_SCLK)

      for i in range(16):
            databit = ((data>>(15-i)) & 1) << RFMD_SDATA_BIT
            u.write_io(side, databit, RFMD_SDATA|RFMD_SCLK)
            u.write_io(side, RFMD_SCLK, RFMD_SCLK)

      u.write_io(side, RFMD_ENX, RFMD_SCLK|RFMD_ENX)
```

```
	u._write_oe(side, 0, RFMD_SDATA) # set sdata as input
	u.write_io(side, RFMD_SCLK, RFMD_SCLK)
	u.write_io(side, 0, RFMD_SCLK)


def read_rfmd_reg(u, side, addr, mode):
	if mode:
		u.write_io(side, RFMD_MODE, RFMD_MODE)
	else:
		u.write_io(side, 0, RFMD_MODE)

	u.write_io(side, RFMD_SCLK, RFMD_SCLK)
	u._write_oe(side, RFMD_SDATA, RFMD_SDATA) # set sdata as output
	u.write_io(side, 0, RFMD_ENX|RFMD_SCLK|RFMD_SDATA)
	u.write_io(side, RFMD_SCLK, RFMD_SCLK)
	u.write_io(side, RFMD_SDATA, RFMD_SCLK|RFMD_SDATA) # set read
	u.write_io(side, RFMD_SCLK, RFMD_SCLK)

	for i in range(7):
		databit = ((addr>>(6-i)) & 1) << RFMD_SDATA_BIT
		u.write_io(side, databit, RFMD_SDATA|RFMD_SCLK)
		u.write_io(side, RFMD_SCLK, RFMD_SCLK)

	u._write_oe(side, 0, RFMD_SDATA) # set sdata as input
	u.write_io(side, 0, RFMD_SCLK)
	u.write_io(side, RFMD_SCLK, RFMD_SCLK)
	u.write_io(side, 0, RFMD_SCLK)

	data = 0x0000
	for i in range(16):
		u.write_io(side, RFMD_SCLK, RFMD_SCLK)
		data = data << 1
		data = data | ((u.read_io(side) >> RFMD_SDATA_BIT) & 1)
		u.write_io(side, 0, RFMD_SCLK)

	u.write_io(side, RFMD_SCLK, RFMD_SCLK)
	u.write_io(side, RFMD_ENX, RFMD_SCLK|RFMD_ENX)
	u.write_io(side, RFMD_SCLK, RFMD_SCLK)
	u.write_io(side, 0, RFMD_SCLK)

	return data

def set_rfmd_freq(u, side, freq):

	if freq > 2500e6:
		return -1
	elif freq >= 1200e6:
		lodiv = 0
	elif freq >= 600e6:
		lodiv = 1
	elif freq >= 200e6:
		lodiv = 2
	else:
		return -1

	vco_freq = freq*(2**lodiv)
```

```
	if vco_freq >= 1972e6:
		print "VCO 1"
		vcosel = 0
	elif vco_freq >= 1556e6:
		print "VCO 2"
		vcosel = 1
	else:
		print "VCO 3"
		vcosel = 2

	R = 1
	N = vco_freq * R / 26.0e6
	_N = int(math.floor(N))
	_NUM = int(round((N - _N) * 2**24))
	_NUM_MSB = int(math.floor(_NUM / 2**8))
	_NUM_LSB = int(_NUM - (2**8 * _NUM_MSB))

	print vco_freq
	print _N
	print _NUM_MSB
	print _NUM_LSB

	data = read_rfmd_reg(u, side, 0x10, True)
	#data = data & 0x3CFF
	#data = data | ((vcosel<<14) | (lodiv<<8))  # P2_VCOSEL = VCO2,
P2_LODIV = 2
	data = data & 0x003F
	# enable only frequency calibration
	data = data | ((vcosel<<14) | (lodiv<<8) | (0<<10) | (3<<12))  #
P2_VCOSEL = VCO2, P2_LODIV = 2
	print bin(data)
	write_rfmd_reg(u, side, 0x10, data, True)

	data = read_rfmd_reg(u, side, 0x03, True)
	data = data & 0x0FFF
	data = data | 0x1000  # CLK_DIV = 1
	print bin(data)
	write_rfmd_reg(u, side, 0x03, data, True)

	# 69.69230076923
	# _N = 69
	# _NUM = 11614996
	# _NUM_MSB = 45371
	# _NUM_LSB = 20

	# P2_NUM_MSB = 45371
	write_rfmd_reg(u, side, 0x11, _NUM_MSB, True)

	data = read_rfmd_reg(u, side, 0x12, True)
	data = data & 0x00FF
	data = data | (_NUM_LSB<<8)  # P2_NUM_LSB = 20
	print bin(data)
	write_rfmd_reg(u, side, 0x12, data, True)

	data = read_rfmd_reg(u, side, 0x13, True)
	data = data & 0x007F
	data = data | (_N<<7)  # P2_N = 69
```

```
	print bin(data)
	write_rfmd_reg(u, side, 0x13, data, True)

	return 0

def bin(x):
	return ''.join(x & (1 << i) and '1' or '0' for i in range(15,-1,-1))

def setup_mmp9000(u, side, freq):
	lo1 = 295e6
	lo2 = freq - 300e6
	if lo2 < 300e6:
		lo2 = freq+300e6

	print (300e6+lo2) / 1e6
	print (300e6-lo2) / 1e6

	# setup the digital I/O
	u._write_oe(side, RFMD_ENX|RFMD_SCLK|RFMD_ENABLE|RFMD_MODE|TR_SWITCH,
0xffff)
	u.write_io(side, RFMD_ENX,
RFMD_ENX|RFMD_SCLK|RFMD_ENABLE|RFMD_MODE|TR_SWITCH)

	#time.sleep(1)

	# set polarity negative and enable active loop filter
	data = read_rfmd_reg(u, side, 0x00, True)
	data = data & 0xfe7f
	data = data | 0x0180
	write_rfmd_reg(u, side, 0x00, data, True)

	# set mixer output current
	data = read_rfmd_reg(u, side, 0x01, True)
	data = data & 0xf8ff
	data = data | 0x0100  # 5mA
	write_rfmd_reg(u, side, 0x01, data, True)

	# set charge pump current
	data = read_rfmd_reg(u, side, 0x10, True)
	data = data & 0xffc0
	data = data | 31  # 31 is default
	write_rfmd_reg(u, side, 0x10, data, True)

	# tune the RF2052 frequency
	if set_rfmd_freq(u, side, lo2) == -1:
		return -1

	# enable the ADF4360 and RF2052
	u.write_io(side, RFMD_ENABLE, RFMD_ENABLE)


	# configure and tune the ADF4360
	if set_adf4360(u, side, lo1/1e6) == -1:
		return -1

	return 0
```

```
##################################################
# instantiate one transmit chain for each call

class pipeline(gr.hier_block2):
    def __init__(self, filename, lo_freq, audio_rate, if_rate):

        gr.hier_block2.__init__(self, "pipeline",
                                gr.io_signature(0, 0, 0),
# Input signature
                                gr.io_signature(1, 1, gr.sizeof_gr_complex))
# Output signature

        src = gr.file_source (gr.sizeof_float, filename, True)
        fmtx = blks2.nbfm_tx (audio_rate, if_rate, max_dev=5e3, tau=75e-6)

        # Local oscillator
        lo = gr.sig_source_c (if_rate,        # sample rate
                              gr.GR_SIN_WAVE, # waveform type
                              lo_freq,        #frequency
                              1.0,            # amplitude (orig 1.0)
                              0)              # DC Offset
        mixer = gr.multiply_cc ()

        self.connect (src, fmtx, (mixer, 0))
        self.connect (lo, (mixer, 1))
        self.connect (mixer, self)

class fm_tx_block(stdgui2.std_top_block):
    def __init__(self, frame, panel, vbox, argv):
        MAX_CHANNELS = 7
        stdgui2.std_top_block.__init__ (self, frame, panel, vbox, argv)

        parser = OptionParser (option_class=eng_option)
        parser.add_option("-T", "--tx-subdev-spec", type="subdev",
default=None,
                          help="select USRP Tx side A or B")
        parser.add_option("-f", "--freq", type="eng_float", default=None,
                          help="set Tx frequency to FREQ [required]",
metavar="FREQ")
        parser.add_option("-n", "--nchannels", type="int", default=4,
                          help="number of Tx channels [1,4]")
        #parser.add_option("","--debug", action="store_true", default=False,
        #                  help="Launch Tx debugger")
        (options, args) = parser.parse_args ()

        if len(args) != 0:
            parser.print_help()
            sys.exit(1)

        if options.nchannels < 1 or options.nchannels > MAX_CHANNELS:
            sys.stderr.write ("fm_tx4: nchannels out of range.  Must be in
[1,%d]\n" % MAX_CHANNELS)
            sys.exit(1)

        if options.freq is None:
```

```
            sys.stderr.write("fm_tx4: must specify frequency with -f FREQ\n")
            parser.print_help()
            sys.exit(1)

        # ---------------------------------------------------------------
        # Set up constants and parameters

        self.u = usrp.sink_c ()       # the USRP sink (consumes samples)

        self.dac_rate = self.u.dac_rate()                    # 128 MS/s
        self.usrp_interp = 400
        self.u.set_interp_rate(self.usrp_interp)
        self.usrp_rate = self.dac_rate / self.usrp_interp    # 320 kS/s
        self.sw_interp = 10
        self.audio_rate = self.usrp_rate / self.sw_interp    # 32 kS/s

        # determine the daughterboard subdevice we're using
        if options.tx_subdev_spec is None:
            options.tx_subdev_spec = usrp.pick_tx_subdevice(self.u)

        m = usrp.determine_tx_mux_value(self.u, options.tx_subdev_spec)
        #print "mux = %#04x" % (m,)
        self.u.set_mux(m)
        self.subdev = usrp.selected_subdev(self.u, options.tx_subdev_spec)

        ################ MMP9000 DB CODE ##################
        if setup_mmp9000(self.u, self.subdev.which(), options.freq) == -1:
            print "frequency out of range"
            raise SystemExit
        ###################################################

        print "Using TX d'board %s" % (self.subdev.side_and_name(),)

        self.subdev.set_gain(self.subdev.gain_range()[1])    # set max Tx
gain
        #if not self.set_freq(options.freq):
        if not self.set_freq(5e6):
            freq_range = self.subdev.freq_range()
            print "Failed to set frequency to %s.  Daughterboard supports %s
to %s" % (
                eng_notation.num_to_str(options.freq),
                eng_notation.num_to_str(freq_range[0]),
                eng_notation.num_to_str(freq_range[1]))
            raise SystemExit
        self.subdev.set_enable(True)                         # enable
transmitter

        sum = gr.add_cc ()

        # Instantiate N NBFM channels
        step = 25e3
        offset = (0 * step, 1 * step, -1 * step, 2 * step, -2 * step, 3 *
step, -3 * step)
        for i in range (options.nchannels):
            t = pipeline("audio-%d.dat" % (i % 4), offset[i],
                         self.audio_rate, self.usrp_rate)
            self.connect(t, (sum, i))
```

```
            #orig gain = 4000 / channel count
        gain = gr.multiply_const_cc (4000.0 / options.nchannels)

        # connect it all
        self.connect (sum, gain)
        self.connect (gain, self.u)

        # plot an FFT to verify we are sending what we want
        if 1:
            post_mod = fftsink2.fft_sink_c(panel, title="Post Modulation",
                                           fft_size=512,
sample_rate=self.usrp_rate,
                                           y_per_div=20, ref_level=40)
            self.connect (sum, post_mod)
            vbox.Add (post_mod.win, 1, wx.EXPAND)


        #if options.debug:
        #    self.debugger = tx_debug_gui.tx_debug_gui(self.subdev)
        #    self.debugger.Show(True)


    def set_freq(self, target_freq):
        """
        Set the center frequency we're interested in.

        @param target_freq: frequency in Hz
        @rypte: bool

        Tuning is a two step process.  First we ask the front-end to
        tune as close to the desired frequency as it can.  Then we use
        the result of that operation and our target_frequency to
        determine the value for the digital up converter.  Finally, we feed
        any residual_freq to the s/w freq translater.
        """

        r = self.u.tune(self.subdev.which(), self.subdev, target_freq)
        if r:
            print "r.baseband_freq =",
eng_notation.num_to_str(r.baseband_freq)
            print "r.dxc_freq      =", eng_notation.num_to_str(r.dxc_freq)
            print "r.residual_freq =",
eng_notation.num_to_str(r.residual_freq)
            print "r.inverted      =", r.inverted

            # Could use residual_freq in s/w freq translator
            return True

        return False

def main ():
    app = stdgui2.stdapp(fm_tx_block, "Multichannel FM Tx", nstatus=1)
    app.MainLoop ()

if __name__ == "__main__":
      main ()
```

10.2 fmrx_mmp.py

```
#!/usr/bin/env python
#
# Copyright 2005,2006,2007,2009 Free Software Foundation, Inc.
#
# This file is part of GNU Radio
#
# GNU Radio is free software; you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation; either version 3, or (at your option)
# any later version.
#
# GNU Radio is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with GNU Radio; see the file COPYING.  If not, write to
# the Free Software Foundation, Inc., 51 Franklin Street,
# Boston, MA 02110-1301, USA.
#

from gnuradio import gr, gru, eng_notation, optfir
from gnuradio import audio
from gnuradio import usrp
from gnuradio import blks2
from gnuradio.eng_option import eng_option
from gnuradio.wxgui import slider, powermate
from gnuradio.wxgui import stdgui2, fftsink2, form
from optparse import OptionParser
from usrpm import usrp_dbid
import sys
import time
import math
import wx

SPI_ENABLE_TX_A  = usrp.SPI_ENABLE_TX_A
SPI_ENABLE_RX_A  = usrp.SPI_ENABLE_RX_A
SPI_ENABLE_TX_B  = usrp.SPI_ENABLE_TX_B
SPI_ENABLE_RX_B  = usrp.SPI_ENABLE_RX_B
SPI_FMT_LSB      = usrp.SPI_FMT_LSB
SPI_FMT_MSB      = usrp.SPI_FMT_MSB
SPI_FMT_HDR_MASK = usrp.SPI_FMT_HDR_MASK
SPI_FMT_HDR_0    = usrp.SPI_FMT_HDR_0
SPI_FMT_HDR_1    = usrp.SPI_FMT_HDR_1
SPI_FMT_HDR_2    = usrp.SPI_FMT_HDR_2


RFMD_ENX_BIT = 4
RFMD_SCLK_BIT = 3
RFMD_SDATA_BIT = 2
RFMD_ENABLE_BIT = 5
RFMD_MODE_BIT = 0

TR_SWITCH_BIT = 6
```

```
RFMD_ENX = (1 << RFMD_ENX_BIT)
RFMD_SCLK = (1 << RFMD_SCLK_BIT)
RFMD_SDATA = (1 << RFMD_SDATA_BIT)
RFMD_ENABLE = (1 << RFMD_ENABLE_BIT)
RFMD_MODE = (1 << RFMD_MODE_BIT)

TR_SWITCH = (1 << TR_SWITCH_BIT)


def _write_all(u, side, R, control, N):
      """
      Write R counter latch, control latch and N counter latch to VCO.

      Adds 10ms delay between writing control and N

      @param R: 24-bit R counter latch
      @type R: int
      @param control: 24-bit control latch
      @type control: int
      @param N: 24-bit N counter latch
      @type N: int
      """
      _write_R(u, side, R)
      _write_control(u, side, control)
      time.sleep(0.010)
      _write_N(u, side, N)

def _write_control(u, side, control):
      _write_it(u, side, (control & ~0x3) | 0)

def _write_R(u, side, R):
      _write_it(u, side, (R & ~0x3) | 1)

def _write_N(u, side, N):
      _write_it(u, side, (N & ~0x3) | 2)

def _write_it(u, side, v):
      s = ''.join((chr((v >> 16) & 0xff), chr((v >>  8) & 0xff), chr(v &
0xff)))
      # fix to write to A or B, not just B
      u._write_spi(0, SPI_ENABLE_RX_B, SPI_FMT_MSB | SPI_FMT_HDR_0, s)


def set_adf4360(u, side, target_freq):

      if target_freq < 280 or target_freq > 320:
            return -1

      # R-Register Common Values
      d_BSC   = 3  # bits 21,20 Div by 8 to be safe
      d_TEST  = 0  # bit 19
      d_LDP   = 1  # bit 18
      d_ABP   = 0  # bit 17,16   3ns
      d_R_DIV = 64 # bits 15:2

      # N-Register Common Values
```

```
      d_CPG   = 0  # bit 21  CP current setting 1 is permanently used
      d_B_DIV = int(target_freq) # bits 20:8 (300/64*16)=75

      # Control Register Common Values
      d_PD    = 0  # bits 21,20   Normal operation (3 powers down, 0 powers
up)
      d_CP2   = 7  # bits 19:17   2.5mA
      d_CP1   = 7  # bits 16:14   2.5mA
      d_PL    = 0  # bits 13,12   3.5mA
      d_MTLD  = 1  # bit 11       enabled
      d_CPG   = 0  # bit 10       CP setting 1
      d_CP3S  = 0  # bit 9        Normal
      d_PDP   = 1  # bit 8        Positive
      d_MUXOUT= 1  # bits 7:5     Digital Lock Detect
      d_CR    = 0  # bit 4        Normal
      d_PC    = 1  # bits 3,2     Core power 5mA

      control = (d_PD<<20) | (d_CP2<<17) | (d_CP1<<14) | (d_PL<<12) |
(d_MTLD<<11) | (d_CPG<<10) | \
            (d_CP3S<<9) | (d_PDP<<8) | (d_MUXOUT<<5) | (d_CR<<4) | (d_PC<<2)

      N = (d_CPG<<21) | (d_B_DIV<<8)

      R = (d_BSC<<20) | (d_TEST<<19) | (d_LDP<<18) | (d_ABP<<16) |
(d_R_DIV<<2)

      _write_all(u, side, R, control, N)

      return 0

def write_rfmd_reg(u, side, addr, data, mode):
      if mode:
            u.write_io(side, RFMD_MODE, RFMD_MODE)
      else:
            u.write_io(side, 0, RFMD_MODE)

      u.write_io(side, RFMD_SCLK, RFMD_SCLK)
      u._write_oe(side, RFMD_SDATA, RFMD_SDATA) # set sdata as output
      u.write_io(side, 0, RFMD_ENX|RFMD_SCLK|RFMD_SDATA)
      u.write_io(side, RFMD_SCLK, RFMD_SCLK)
      u.write_io(side, 0, RFMD_SCLK|RFMD_SDATA) # set write
      u.write_io(side, RFMD_SCLK, RFMD_SCLK)

      for i in range(7):
            databit = ((addr>>(6-i)) & 1) << RFMD_SDATA_BIT
            u.write_io(side, databit, RFMD_SDATA|RFMD_SCLK)
            u.write_io(side, RFMD_SCLK, RFMD_SCLK)

      for i in range(16):
            databit = ((data>>(15-i)) & 1) << RFMD_SDATA_BIT
            u.write_io(side, databit, RFMD_SDATA|RFMD_SCLK)
            u.write_io(side, RFMD_SCLK, RFMD_SCLK)

      u.write_io(side, RFMD_ENX, RFMD_SCLK|RFMD_ENX)
      u._write_oe(side, 0, RFMD_SDATA) # set sdata as input
      u.write_io(side, RFMD_SCLK, RFMD_SCLK)
      u.write_io(side, 0, RFMD_SCLK)
```

```
def read_rfmd_reg(u, side, addr, mode):
      if mode:
            u.write_io(side, RFMD_MODE, RFMD_MODE)
      else:
            u.write_io(side, 0, RFMD_MODE)

      u.write_io(side, RFMD_SCLK, RFMD_SCLK)
      u._write_oe(side, RFMD_SDATA, RFMD_SDATA) # set sdata as output
      u.write_io(side, 0, RFMD_ENX|RFMD_SCLK|RFMD_SDATA)
      u.write_io(side, RFMD_SCLK, RFMD_SCLK)
      u.write_io(side, RFMD_SDATA, RFMD_SCLK|RFMD_SDATA) # set read
      u.write_io(side, RFMD_SCLK, RFMD_SCLK)

      for i in range(7):
            databit = ((addr>>(6-i)) & 1) << RFMD_SDATA_BIT
            u.write_io(side, databit, RFMD_SDATA|RFMD_SCLK)
            u.write_io(side, RFMD_SCLK, RFMD_SCLK)

      u._write_oe(side, 0, RFMD_SDATA) # set sdata as input
      u.write_io(side, 0, RFMD_SCLK)
      u.write_io(side, RFMD_SCLK, RFMD_SCLK)
      u.write_io(side, 0, RFMD_SCLK)

      data = 0x0000
      for i in range(16):
            u.write_io(side, RFMD_SCLK, RFMD_SCLK)
            data = data << 1
            data = data | ((u.read_io(side) >> RFMD_SDATA_BIT) & 1)
            u.write_io(side, 0, RFMD_SCLK)

      u.write_io(side, RFMD_SCLK, RFMD_SCLK)
      u.write_io(side, RFMD_ENX, RFMD_SCLK|RFMD_ENX)
      u.write_io(side, RFMD_SCLK, RFMD_SCLK)
      u.write_io(side, 0, RFMD_SCLK)

      return data


def set_rfmd_freq(u, side, freq):

      if freq > 2500e6:
            return -1
      elif freq >= 1200e6:
            lodiv = 0
      elif freq >= 600e6:
            lodiv = 1
      elif freq >= 200e6:
            lodiv = 2
      else:
            return -1

      vco_freq = freq*(2**lodiv)

      if vco_freq >= 1972e6:
            print "VCO 1"
```

```
            vcosel = 0
      elif vco_freq >= 1556e6:
            print "VCO 2"
            vcosel = 1
      else:
            print "VCO 3"
            vcosel = 2

      R = 1
      N = vco_freq * R / 26.0e6
      _N = int(math.floor(N))
      _NUM = int(round((N - _N) * 2**24))
      _NUM_MSB = int(math.floor(_NUM / 2**8))
      _NUM_LSB = int(_NUM - (2**8 * _NUM_MSB))

      print vco_freq
      print _N
      print _NUM_MSB
      print _NUM_LSB

      data = read_rfmd_reg(u, side, 0x10, True)
      #data = data & 0x3CFF
      #data = data | ((vcosel<<14) | (lodiv<<8))  # P2_VCOSEL = VCO2,
P2_LODIV = 2
      data = data & 0x003F
      # enable only frequency calibration
      data = data | ((vcosel<<14) | (lodiv<<8) | (0<<10) | (3<<12))  #
P2_VCOSEL = VCO2, P2_LODIV = 2
      print bin(data)
      write_rfmd_reg(u, side, 0x10, data, True)

      data = read_rfmd_reg(u, side, 0x03, True)
      data = data & 0x0FFF
      data = data | 0x1000  # CLK_DIV = 1
      print bin(data)
      write_rfmd_reg(u, side, 0x03, data, True)

      # 69.69230076923
      # _N = 69
      # _NUM = 11614996
      # _NUM_MSB = 45371
      # _NUM_LSB = 20

      # P2_NUM_MSB = 45371
      write_rfmd_reg(u, side, 0x11, _NUM_MSB, True)

      data = read_rfmd_reg(u, side, 0x12, True)
      data = data & 0x00FF
      data = data | (_NUM_LSB<<8)  # P2_NUM_LSB = 20
      print bin(data)
      write_rfmd_reg(u, side, 0x12, data, True)

      data = read_rfmd_reg(u, side, 0x13, True)
      data = data & 0x007F
      data = data | (_N<<7)  # P2_N = 69
      print bin(data)
      write_rfmd_reg(u, side, 0x13, data, True)
```

```
      return 0

def bin(x):
      return ''.join(x & (1 << i) and '1' or '0' for i in range(15,-1,-1))

def setup_mmp9000(u, side, freq):
      lo1 = 295e6
      lo2 = freq - 300e6
      if lo2 < 300e6:
            lo2 = freq+300e6

      print (300e6+lo2) / 1e6
      print (300e6-lo2) / 1e6

      # setup the digital I/O
      u._write_oe(side, RFMD_ENX|RFMD_SCLK|RFMD_ENABLE|RFMD_MODE|TR_SWITCH,
0xffff)
      u.write_io(side, RFMD_ENX,
RFMD_ENX|RFMD_SCLK|RFMD_ENABLE|RFMD_MODE|TR_SWITCH)

      #time.sleep(1)

      # set polarity negative and enable active loop filter
      data = read_rfmd_reg(u, side, 0x00, True)
      data = data & 0xfe7f
      data = data | 0x0180
      write_rfmd_reg(u, side, 0x00, data, True)

      # set mixer output current
      data = read_rfmd_reg(u, side, 0x01, True)
      data = data & 0xf8ff
      data = data | 0x0100  # 5mA
      write_rfmd_reg(u, side, 0x01, data, True)

      # set charge pump current
      data = read_rfmd_reg(u, side, 0x10, True)
      data = data & 0xffc0
      data = data | 31  # 31 is default
      write_rfmd_reg(u, side, 0x10, data, True)

      # tune the RF2052 frequency
      if set_rfmd_freq(u, side, lo2) == -1:
            return -1

      # enable the ADF4360 and RF2052
      u.write_io(side, RFMD_ENABLE, RFMD_ENABLE)


      # configure and tune the ADF4360
      if set_adf4360(u, side, lo1/1e6) == -1:
            return -1

      # set vga gain
      u.write_aux_dac(side, 0, int(300))

      return 0
```

```
def pick_subdevice(u):
    """
    The user didn't specify a subdevice on the command line.
    Try for one of these, in order: TV_RX, BASIC_RX, whatever is on side A.

    @return a subdev_spec
    """
    return usrp.pick_subdev(u, (usrp_dbid.TV_RX,
                                usrp_dbid.TV_RX_REV_2,
                          usrp_dbid.TV_RX_REV_3,
                                usrp_dbid.BASIC_RX))


class wfm_rx_block (stdgui2.std_top_block):
    def __init__(self,frame,panel,vbox,argv):
        stdgui2.std_top_block.__init__ (self,frame,panel,vbox,argv)

        parser=OptionParser(option_class=eng_option)
        parser.add_option("-R", "--rx-subdev-spec", type="subdev",
default=None,
                          help="select USRP Rx side A or B (default=A)")
        parser.add_option("-f", "--freq", type="eng_float", default=100.1e6,
                          help="set frequency to FREQ", metavar="FREQ")
        parser.add_option("-g", "--gain", type="eng_float", default=40,
                          help="set gain in dB (default is midpoint)")
        parser.add_option("-V", "--volume", type="eng_float", default=None,
                          help="set volume (default is midpoint)")
        parser.add_option("-O", "--audio-output", type="string", default="",
                          help="pcm device name.  E.g., hw:0,0 or surround51
or /dev/dsp")

        (options, args) = parser.parse_args()
        if len(args) != 0:
            parser.print_help()
            sys.exit(1)

        self.frame = frame
        self.panel = panel

        self.vol = 0
        self.state = "FREQ"
        self.freq = 0

        # build graph

        self.u = usrp.source_c()                    # usrp is data source

        adc_rate = self.u.adc_rate()                # 64 MS/s
        usrp_decim = 200
        self.u.set_decim_rate(usrp_decim)
        usrp_rate = adc_rate / usrp_decim           # 320 kS/s
        chanfilt_decim = 1
        demod_rate = usrp_rate / chanfilt_decim
        audio_decimation = 10
        audio_rate = demod_rate / audio_decimation  # 32 kHz
```

```
        if options.rx_subdev_spec is None:
            options.rx_subdev_spec = pick_subdevice(self.u)

        self.u.set_mux(usrp.determine_rx_mux_value(self.u,
options.rx_subdev_spec))
        self.subdev = usrp.selected_subdev(self.u, options.rx_subdev_spec)


        ################ MMP9000 DB CODE #################
        if setup_mmp9000(self.u, self.subdev.which(), options.freq) == -1:
            print "frequency out of range"
            raise SystemExit
        ##################################################


        print "Using RX d'board %s" % (self.subdev.side_and_name(),)
        dbid = self.subdev.dbid()
        if not (dbid == usrp_dbid.BASIC_RX or
                dbid == usrp_dbid.TV_RX or
                dbid == usrp_dbid.TV_RX_REV_2 or
                dbid == usrp_dbid.TV_RX_REV_3):
            print "This daughterboard does not cover the required frequency
range"
            print "for this application.  Please use a BasicRX or TVRX
daughterboard."
            raw_input("Press ENTER to continue anyway, or Ctrl-C to exit.")

        chan_filt_coeffs = optfir.low_pass (1,           # gain
                                            usrp_rate,   # sampling rate
                                            80e3,        # passband cutoff
                                            115e3,       # stopband cutoff
                                            0.1,         # passband ripple
                                            60)          # stopband
attenuation
        #print len(chan_filt_coeffs)
        chan_filt = gr.fir_filter_ccf (chanfilt_decim, chan_filt_coeffs)

        self.guts = blks2.wfm_rcv (demod_rate, audio_decimation)

        self.volume_control = gr.multiply_const_ff(self.vol)

        # sound card as final sink
        audio_sink = audio.sink (int (audio_rate),
                                 options.audio_output,
                                 False)  # ok_to_block

        # now wire it all together
        self.connect (self.u, chan_filt, self.guts, self.volume_control,
audio_sink)

        self._build_gui(vbox, usrp_rate, demod_rate, audio_rate)

        if options.gain is None:
            # if no gain was specified, use the mid-point in dB
            g = self.subdev.gain_range()
            options.gain = float(g[0]+g[1])/2
```

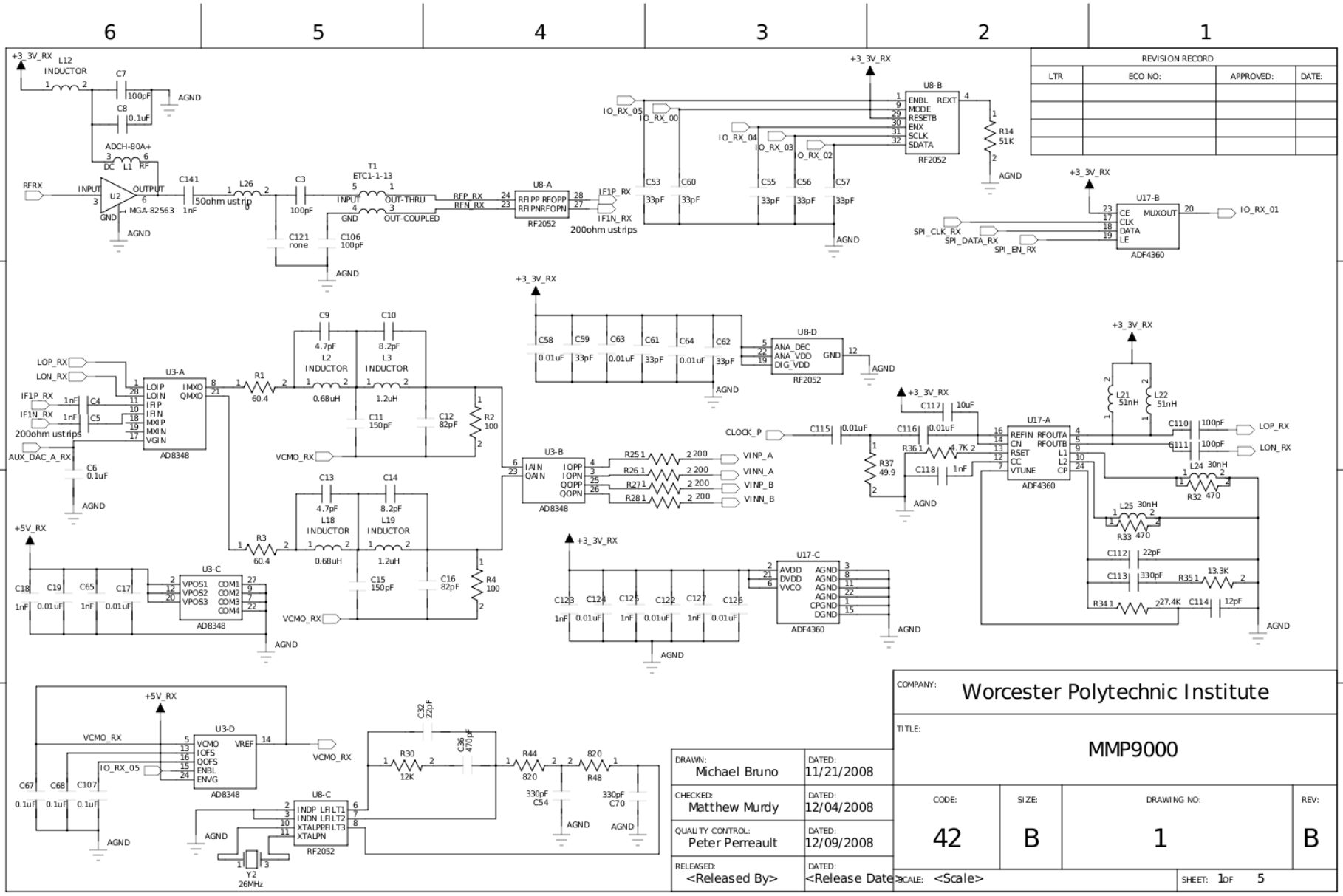
```
        if options.volume is None:
            g = self.volume_range()
            options.volume = float(g[0]+g[1])/2

        if abs(options.freq) < 1e6:
            options.freq *= 1e6

        # set initial values

        self.set_gain(options.gain)
        self.set_vol(options.volume)
        if not(self.set_freq(5e6)):
            self._set_status_msg("Failed to set initial frequency")


    def _set_status_msg(self, msg, which=0):
        self.frame.GetStatusBar().SetStatusText(msg, which)


    def _build_gui(self, vbox, usrp_rate, demod_rate, audio_rate):

        def _form_set_freq(kv):
            return self.set_freq(kv['freq'])


        if 1:
            self.src_fft = fftsink2.fft_sink_c(self.panel, title="Data from
USRP",
                                               fft_size=512,
sample_rate=usrp_rate,
                                       ref_scale=32768.0, ref_level=0,
y_divs=12)
            self.connect (self.u, self.src_fft)
            vbox.Add (self.src_fft.win, 4, wx.EXPAND)

        if 1:
            post_filt_fft = fftsink2.fft_sink_f(self.panel, title="Post
Demod",
                                                fft_size=1024,
sample_rate=usrp_rate,
                                                y_per_div=10, ref_level=0)
            self.connect (self.guts.fm_demod, post_filt_fft)
            vbox.Add (post_filt_fft.win, 4, wx.EXPAND)

        if 0:
            post_deemph_fft = fftsink2.fft_sink_f(self.panel, title="Post
Deemph",
                                                  fft_size=512,
sample_rate=audio_rate,
                                                  y_per_div=10, ref_level=-
20)
            self.connect (self.guts.deemph, post_deemph_fft)
            vbox.Add (post_deemph_fft.win, 4, wx.EXPAND)


        # control area form at bottom
        self.myform = myform = form.form()
```

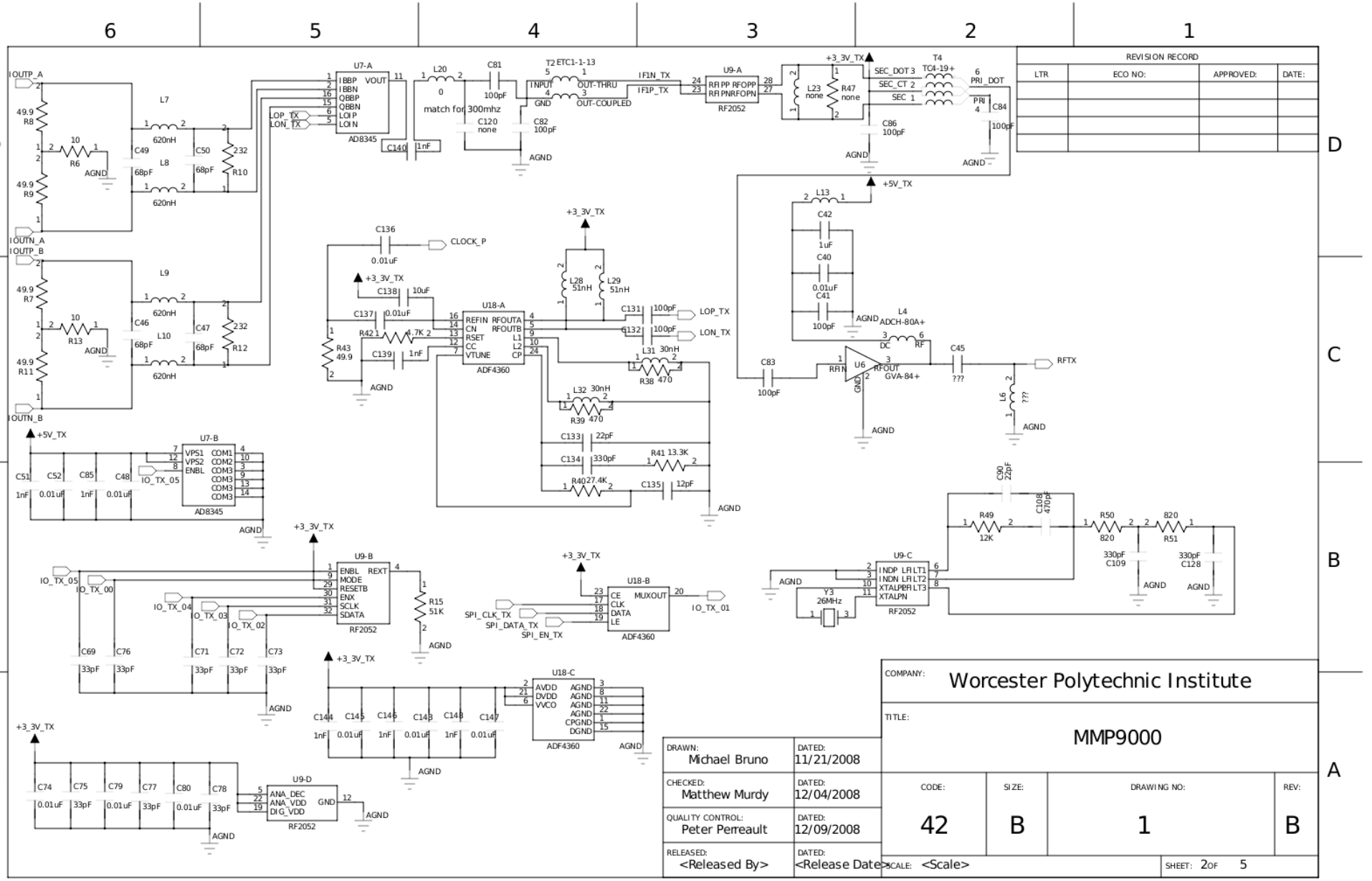
```
        hbox = wx.BoxSizer(wx.HORIZONTAL)
        hbox.Add((5,0), 0)
        myform['freq'] = form.float_field(
            parent=self.panel, sizer=hbox, label="Freq", weight=1,
            callback=myform.check_input_and_call(_form_set_freq,
self._set_status_msg))

        hbox.Add((5,0), 0)
        myform['freq_slider'] = \
            form.quantized_slider_field(parent=self.panel, sizer=hbox,
weight=3,
                                        range=(87.9e6, 108.1e6, 0.1e6),
                                        callback=self.set_freq)
        hbox.Add((5,0), 0)
        vbox.Add(hbox, 0, wx.EXPAND)

        hbox = wx.BoxSizer(wx.HORIZONTAL)
        hbox.Add((5,0), 0)

        myform['volume'] = \
            form.quantized_slider_field(parent=self.panel, sizer=hbox,
label="Volume",
                                        weight=3, range=self.volume_range(),
                                        callback=self.set_vol)
        hbox.Add((5,0), 1)

        myform['gain'] = \
            form.quantized_slider_field(parent=self.panel, sizer=hbox,
label="Gain",
                                        weight=3,
range=self.subdev.gain_range(),
                                        callback=self.set_gain)
        hbox.Add((5,0), 0)
        vbox.Add(hbox, 0, wx.EXPAND)

        try:
            self.knob = powermate.powermate(self.frame)
            self.rot = 0
            powermate.EVT_POWERMATE_ROTATE (self.frame, self.on_rotate)
            powermate.EVT_POWERMATE_BUTTON (self.frame, self.on_button)
        except:
            print "FYI: No Powermate or Contour Knob found"


    def on_rotate (self, event):
        self.rot += event.delta
        if (self.state == "FREQ"):
            if self.rot >= 3:
                self.set_freq(self.freq + .1e6)
                self.rot -= 3
            elif self.rot <=-3:
                self.set_freq(self.freq - .1e6)
                self.rot += 3
        else:
            step = self.volume_range()[2]
            if self.rot >= 3:
```

```
                self.set_vol(self.vol + step)
                self.rot -= 3
            elif self.rot <=-3:
                self.set_vol(self.vol - step)
                self.rot += 3

    def on_button (self, event):
        if event.value == 0:        # button up
            return
        self.rot = 0
        if self.state == "FREQ":
            self.state = "VOL"
        else:
            self.state = "FREQ"
        self.update_status_bar ()


    def set_vol (self, vol):
        g = self.volume_range()
        self.vol = max(g[0], min(g[1], vol))
        self.volume_control.set_k(10**(self.vol/10))
        self.myform['volume'].set_value(self.vol)
        self.update_status_bar ()

    def set_freq(self, target_freq):
        """
        Set the center frequency we're interested in.

        @param target_freq: frequency in Hz
        @rypte: bool

        Tuning is a two step process.  First we ask the front-end to
        tune as close to the desired frequency as it can.  Then we use
        the result of that operation and our target_frequency to
        determine the value for the digital down converter.
        """
        r = usrp.tune(self.u, 0, self.subdev, target_freq)

        if r:
            self.freq = target_freq
            self.myform['freq'].set_value(target_freq)         # update
displayed value
            self.myform['freq_slider'].set_value(target_freq)  # update
displayed value
            self.update_status_bar()
            self._set_status_msg("OK", 0)
            return True

        self._set_status_msg("Failed", 0)
        return False

    def set_gain(self, gain):
        self.myform['gain'].set_value(gain)     # update displayed value
        self.subdev.set_gain(gain)

    def update_status_bar (self):
        msg = "Volume:%r  Setting:%s" % (self.vol, self.state)
```

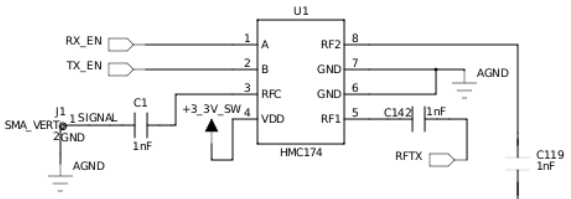
```
        self._set_status_msg(msg, 1)
        self.src_fft.set_baseband_freq(self.freq)

    def volume_range(self):
        return (-20.0, 0.0, 0.5)


if __name__ == '__main__':
    app = stdgui2.stdapp (wfm_rx_block, "USRP WFM RX")
    app.MainLoop ()
```

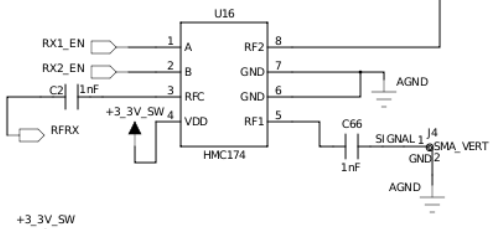
11 Appendix B – Schematic

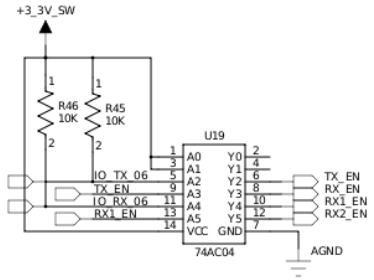

REVISION RECORD
LTR
ECO NO:
APPROVED:
DATE:
COMPANY: Worcester Polytechnic Institute
TITLE: MMP9000
DRAWN: Michael Bruno
DATED: 11/21/2008
CHECKED: Matthew Murdy
DATED: 12/04/2008
QUALITY CONTROL: Peter Perreault
DATED: 12/09/2008
RELEASED: <Released By>
DATED: <Release Date>
CODE: 42
SIZE: B
DRAWING NO: 1
REV: B
SCALE: <Scale>
SHEET: 1 OF 5

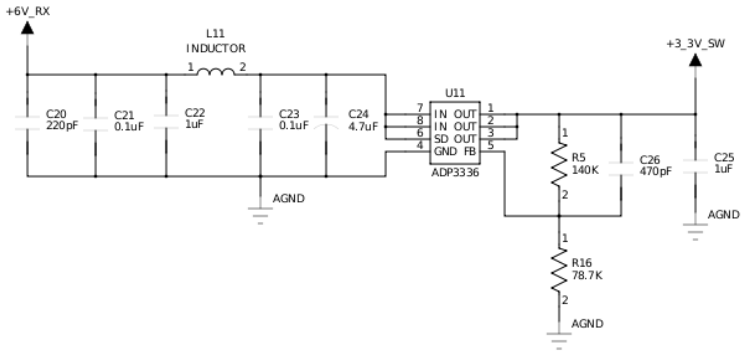

REVISION RECORD
LTR
ECO NO:
APPROVED:
DATE:
COMPANY: Worcester Polytechnic Institute
TITLE: MMP9000
DRAWN: Michael Bruno
DATED: 11/21/2008
CHECKED: Matthew Murdy
DATED: 12/04/2008
QUALITY CONTROL: Peter Perreault
DATED: 12/09/2008
RELEASED: <Released By>
DATED: <Release Date>
CODE: 42
SIZE: B
DRAWING NO: 1
REV: B
SCALE: <Scale>
SHEET: 2 OF 5

REVISION RECORD			
LTR	ECO NO:	APPROVED:	DATE:

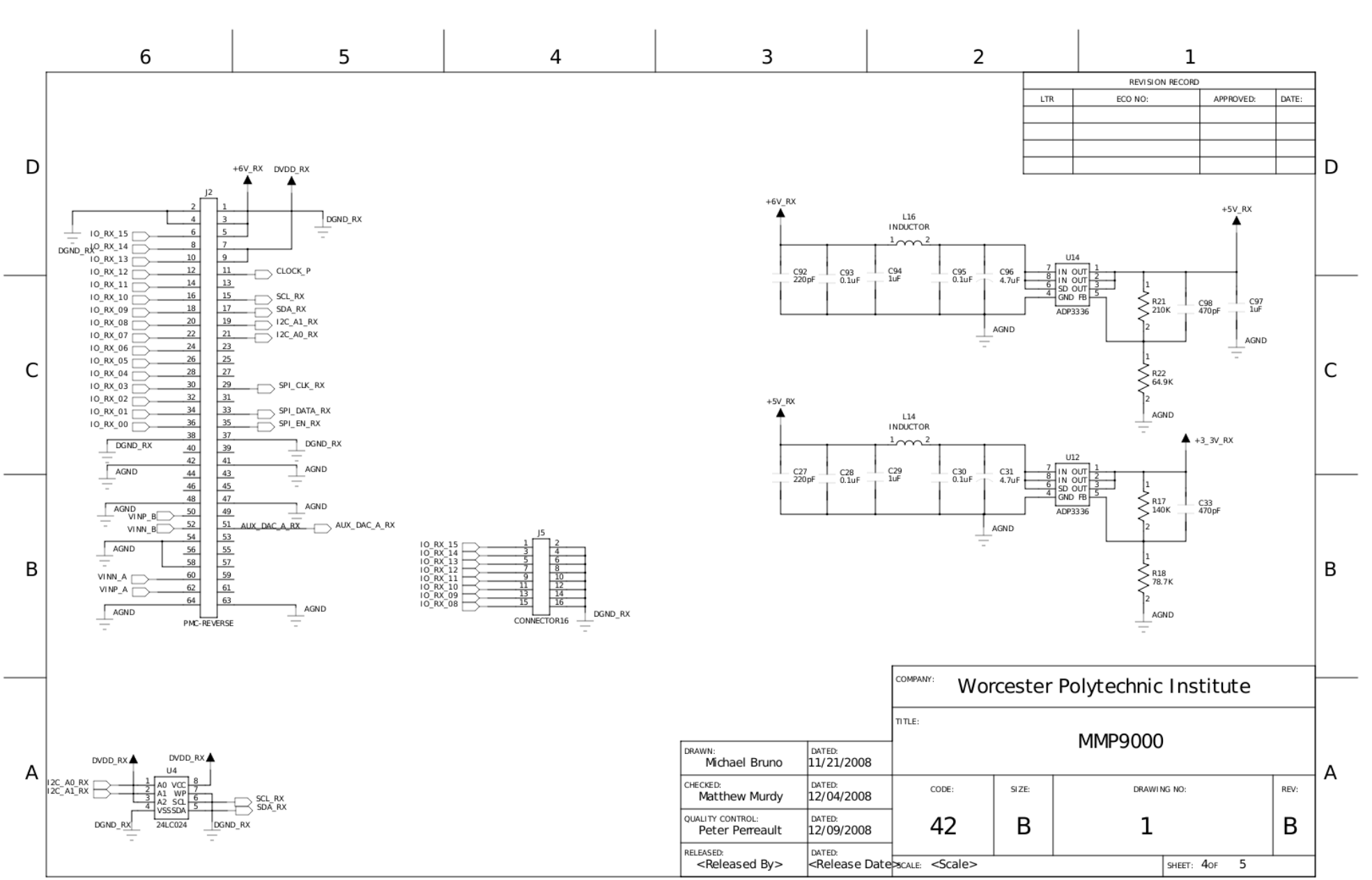


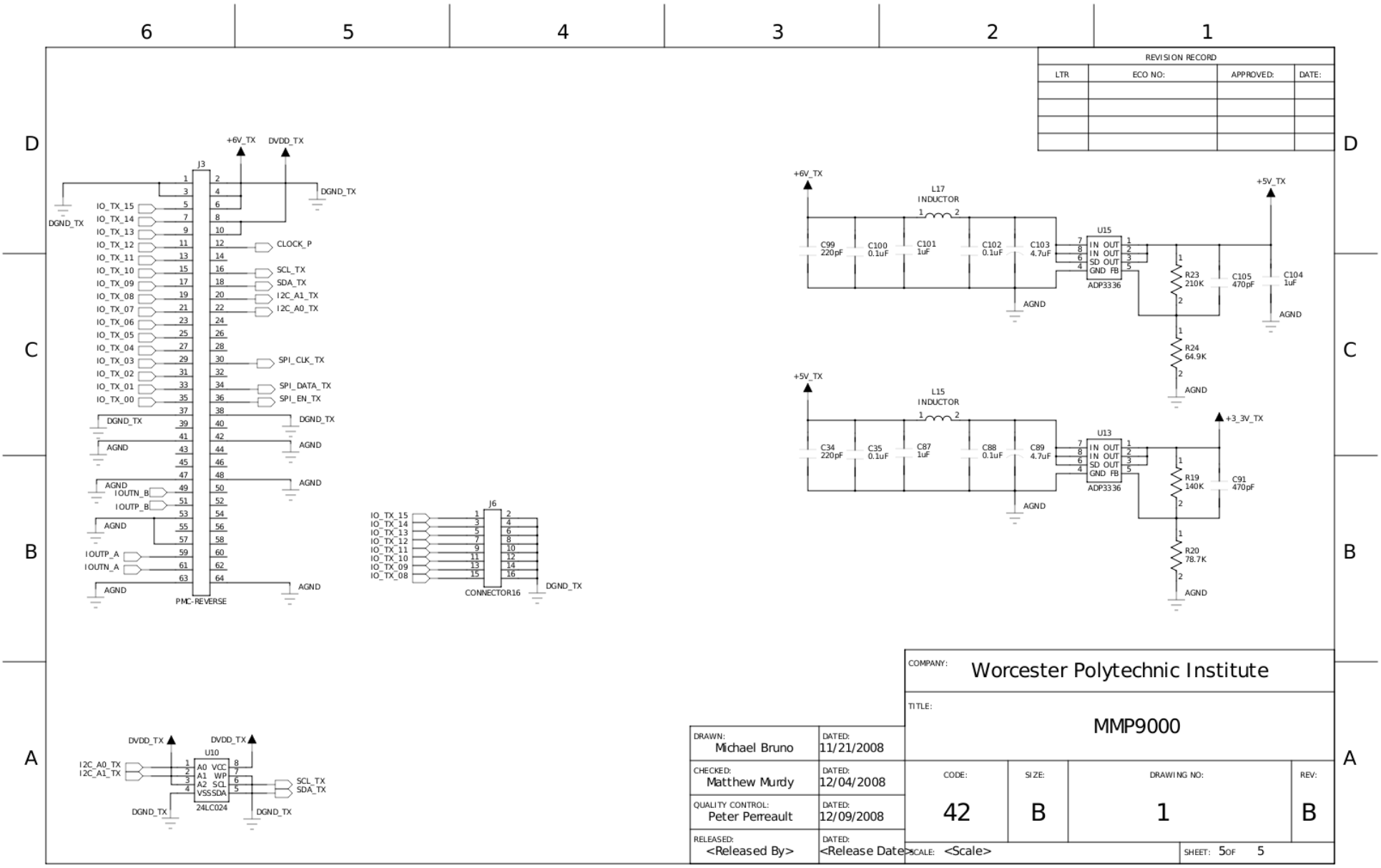






DRAWN: Michael Bruno	DATED: 11/21/2008	COMPANY: Worcester Polytechnic Institute			
CHECKED: Matthew Murdy	DATED: 12/04/2008	TITLE: MMP9000			
QUALITY CONTROL: Peter Perreault	DATED: 12/09/2008	CODE: 42	SIZE: B	DRAWING NO: 1	REV: B
RELEASED: <Released By>	DATED: <Release Date>	SCALE: <Scale>		SHEET: 3 OF 5	


REVISION RECORD
LTR
ECO NO:
APPROVED:
DATE:
+6V_RX
DVDD_RX
J2
DGND_RX
IO_RX_15
IO_RX_14
IO_RX_13
IO_RX_12
IO_RX_11
IO_RX_10
IO_RX_09
IO_RX_08
IO_RX_07
IO_RX_06
IO_RX_05
IO_RX_04
IO_RX_03
IO_RX_02
IO_RX_01
IO_RX_00
CLOCK_P
SCL_RX
SDA_RX
I2C_A1_RX
I2C_A0_RX
SPI_CLK_RX
SPI_DATA_RX
SPI_EN_RX
AGND
VINP_B
VINN_B
AUX_DAC_A_RX
VINN_A
VINP_A
PMC-REVERSE
J5
CONNECTOR16
L16
INDUCTOR
C92
220pF
C93
0.1uF
C94
1uF
C95
0.1uF
C96
4.7uF
U14
IN OUT
SD OUT
GND FB
ADP3336
R21
210K
C98
470pF
C97
1uF
+5V_RX
R22
64.9K
L14
C27
C28
C29
C30
C31
U12
R17
140K
C33
R18
78.7K
+3_3V_RX
U4
A0 VCC
A1 WP
A2 SCL
VSS SDA
24LC024
I2C_A0_RX
I2C_A1_RX
SCL_RX
SDA_RX
DRAWN:
Michael Bruno
DATED:
11/21/2008
CHECKED:
Matthew Murdy
12/04/2008
QUALITY CONTROL:
Peter Perreault
12/09/2008
RELEASED:
<Released By>
<Release Date>
COMPANY:
Worcester Polytechnic Institute
TITLE:
MMP9000
CODE:
42
SIZE:
B
DRAWING NO:
1
REV:
B
SCALE: <Scale>
SHEET: 4 OF 5


6
5
4
3
2
1
D
C
B
A
REVISION RECORD
LTR
ECO NO:
APPROVED:
DATE:
J3
PMC-REVERSE
+6V_TX
DVDD_TX
DGND_TX
AGND
IO_TX_15
IO_TX_14
IO_TX_13
IO_TX_12
IO_TX_11
IO_TX_10
IO_TX_09
IO_TX_08
IO_TX_07
IO_TX_06
IO_TX_05
IO_TX_04
IO_TX_03
IO_TX_02
IO_TX_01
IO_TX_00
CLOCK_P
SCL_TX
SDA_TX
I2C_A1_TX
I2C_A0_TX
SPI_CLK_TX
SPI_DATA_TX
SPI_EN_TX
IOUTN_B
IOUTP_B
IOUTP_A
IOUTN_A
J6
CONNECTOR16
U10
24LC024
A0 VCC
A1 WP
A2 SCL
VSS SDA
L17
INDUCTOR
C99 220pF
C100 0.1uF
C101 1uF
C102 0.1uF
C103 4.7uF
U15
ADP3336
IN OUT
SD OUT
GND FB
R23 210K
C105 470pF
C104 1uF
R24 64.9K
+5V_TX
L15
C34 220pF
C35 0.1uF
C87 1uF
C88 0.1uF
C89 4.7uF
U13
R19 140K
C91 470pF
R20 78.7K
+3_3V_TX
COMPANY:
Worcester Polytechnic Institute
TITLE:
MMP9000
DRAWN:
Michael Bruno
DATED:
11/21/2008
CHECKED:
Matthew Murdy
DATED:
12/04/2008
QUALITY CONTROL:
Peter Perreault
DATED:
12/09/2008
RELEASED:
<Released By>
DATED:
<Release Date>
CODE:
42
SIZE:
B
DRAWING NO:
1
REV:
B
SCALE:
<Scale>
SHEET:
5 OF 5

12 Appendix C – Parts List

Qty	Reference	Part Name	Cost Ea	Tot Cost
2	U4 U10	24LC024	$0.55	$1.10
1	U19	74AC04	$0.42	$0.42
1	U7	AD8345	$8.78	$8.78
1	U3	AD8348	$9.26	$9.26
2	L1 L4	ADCH-80A+	$2.75	$5.50
2	U17-18	ADF4360	$7.28	$14.56
5	U11-15	ADP3336	$2.30	$11.50
21	C17 C19 C40 C48 C52 C58 C63-64 C74 C79-80 C115-116 C122 C124 C126 C136-137 C143 C145 C147	CAP0603,0.01uF	$0.04	$0.88
15	C6 C8 C21 C23 C28 C30 C35 C67-68 C88 C93 C95 C100 C102 C107	CAP0603,0.1uF	$0.04	$0.63
13	C3 C7 C41 C81-84 C86 C106 C110-111 C131-132	CAP0603,100pF	$0.05	$0.59
2	C114 C135	CAP0603,12pF	$0.10	$0.19
2	C11 C15	CAP0603,150pF	$0.07	$0.13
21	C1-2 C4-5 C18 C51 C65-66 C85 C118-119 C123 C125 C127 C139-142 C144 C146 C148	CAP0603,1nF	$0.05	$0.99
9	C22 C25 C29 C42 C87 C94 C97 C101 C104	CAP0603,1uF	$0.32	$2.89
5	C20 C27 C34 C92 C99	CAP0603,220pF	$0.10	$0.48
4	C32 C90 C112 C133	CAP0603,22pF	$0.10	$0.38
6	C54 C70 C109 C113 C128 C134	CAP0603,330pF	$0.08	$0.45
16	C53 C55-57 C59-62 C69 C71-73 C75-78	CAP0603,33pF	$0.06	$0.99
2	C9 C13	CAP0603,4.7pF	$0.05	$0.09
7	C26 C33 C36 C91 C98 C105 C108	CAP0603,470pF	$0.06	$0.43
4	C46-47 C49-50	CAP0603,68pF	$0.10	$0.38
2	C10 C14	CAP0603,8.2pF	$0.07	$0.14
2	C12 C16	CAP0603,82pF	$0.04	$0.09
1	C45	CAP0603,???		$0.00

2	C120-121	CAP0603,none		$0.00
2	C117 C138	CAP1206,10uF	$0.48	$0.96
5	C24 C31 C89 C96 C103	CAP1206,4.7uF	$0.35	$1.74
2	J5-6	CONNECTOR16	$1.48	$2.95
2	T1-2	ETC1-1-13	$1.28	$2.56
1	U6	GVA-84+	$3.60	$3.60
2	U1 U16	HMC174	$2.87	$5.74
7	L11-17	INDUCTOR	$0.10	$0.69
2	L20 L26	INDUCTOR,0	$0.07	$0.14
2	L2 L18	INDUCTOR,0.68uH	$0.27	$0.54
2	L3 L19	INDUCTOR,1.2uH	$0.27	$0.54
4	L24-25 L31-32	INDUCTOR,30nH	$1.00	$4.00
4	L21-22 L28-29	INDUCTOR,51nH	$1.00	$4.00
4	L7-10	INDUCTOR,620nH	$1.00	$4.00
1	L6	INDUCTOR,???		$0.00
1	L23	INDUCTOR,none		$0.00
1	U2	MGA-82563	$3.20	$3.20
2	J2-3	PMC-REVERSE	$7.45	$14.90
2	R6 R13	RES0603,10	$0.07	$0.15
2	R2 R4	RES0603,100	$0.07	$0.15
2	R45-46	RES0603,10K	$0.07	$0.15
2	R30 R49	RES0603,12K	$0.07	$0.15
2	R35 R41	RES0603,13.3K	$0.07	$0.15
3	R5 R17 R19	RES0603,140K	$0.07	$0.22
4	R25-28	RES0603,200	$0.07	$0.29
2	R21 R23	RES0603,210K	$0.07	$0.15
2	R10 R12	RES0603,232	$0.07	$0.15
2	R34 R40	RES0603,27.4K	$0.07	$0.15
2	R36 R42	RES0603,4.7K	$0.07	$0.15
4	R32-33 R38-39	RES0603,470	$0.07	$0.29

6	R7-9 R11 R37 R43	RES0603,49.9	$0.07	$0.44
2	R14-15	RES0603,51K	$0.07	$0.15
2	R1 R3	RES0603,60.4	$0.07	$0.15
2	R22 R24	RES0603,64.9K	$0.07	$0.15
3	R16 R18 R20	RES0603,78.7K	$0.07	$0.22
4	R44 R48 R50-51	RES0603,820	$0.07	$0.29
1	R47	RES0603,none		$0.00
2	U8-9	RF2052	$11.37	$22.74
2	J1 J4	SMA_VERT	$9.79	$19.58
1	T4	TC4-19+	$2.09	$2.09
2	Y2,Y3	XTAL-4,26MHz	$0.92	$1.84
1	PCB		$66.00	$66.00
			Total:	$226.15