FINAL REPORT OUTLINE

ANALOG FRONT END

1. Introduction
2. System Overview
   1. Drawing
   2. Parameters (IIP3, OIP3, Gain, dynamic range, NF)
3. Enclosures
4. Analog Front End
   1. Explanation of circuit choices
   2. Circuit analysis?
   3. Impedance matching?
   4. Scattering parameters
      1. Gain
      2. Output impedance
      3. Input impedance
5. Power board
   1. Current consumption
   2. Protection
6. ADC board
   1. Sean
7. FPGA Firmware
   1. Mixer
   2. FIR filters
      1. Distributed arithmetic
8. Software Installation
9. Software Usage
10. Software Inner Workings

**FINAL REPORT**

**SDR Jove/DSP**

**Daniel Soberal**

**Sean Bredeweg**

**Phil Behnke**

**August 5, 2011**

A special note of thanks to those companies that provided us with free samples during the prototyping phase of our project.

Linear Technology

Maxim Integrated Circuits

Samtech

Coilcraft

# Abstract

The goal this project was to develop a software-defined radio receiver for amateur radio astronomers which was capable of detecting and recording Jovian radiation from the planet Jupiter and solar radiation from the sun similar to that of the radio JOVE receiver which is distributed by NASA. The receiver is intended to be used in conjunction with an antenna provided by the Grand Valley State University physics department. The high frequency radiation of interest from Jupiter, the decametric band, has a range of 3MHz – 40MHz. The receiver was designed to focus on a 1MHz bandwidth centered at 20.1MHz. A software application has been created, specifically for this receiver, to be run on either a laptop or desktop computer. This application will store and process the data streaming from the hardware and also allow the user to select between several different modes of operation.

The system is comprised of three main sections. The first of these sections is the receiver RF front end which handles filtering out the undesirable frequencies and amplifying the band of interest. The second section is the pre-processing section which handles the conversion of the analog signal to a digital signal; sending the data to the PC. The final section is the software application which handles further signal processing of the samples, and has an easy to navigate graphical user interface.

# Section 1.0 - Introduction and Background

## Section 1.1 - Radio Astronomy Background

Radio astronomy is the study of celestial objects that emit radio waves. With radio astronomy, scientists can study astronomical phenomena that are often invisible in other portions of the electromagnetic spectrum. Using radio astronomy techniques, astronomers can observe the cosmic microwave background radiation, which is considered the remnant signal of the birth of our universe.

The properties of electromagnetic radiation depend strongly on its frequency. Electromagnetic radiation is produced whenever electric charges change either the speed or direction of their movement. In a hot object like the sun, molecules are continuously vibrating or bumping into each other, sending each other off in different directions and at different speeds. Each of these collisions produces electromagnetic radiation at frequencies all across the electromagnetic spectrum.

Electromagnetic radiation is also generated as the result of free electrons or protons spiraling around magnetic field lines as show in Figure 1.1.1. The electrons and protons become energized in the magnetic field and begin to accelerate. As the electron and protons accelerate they release electromagnetic waves. It is observed that the shorter the wavelength (and higher the frequency), the more energy the radiation carries. It is this radiation which can be detected by radio telescopes.

Figure 1.1.1: Synchrotron Radiation

Magnetic Field Line

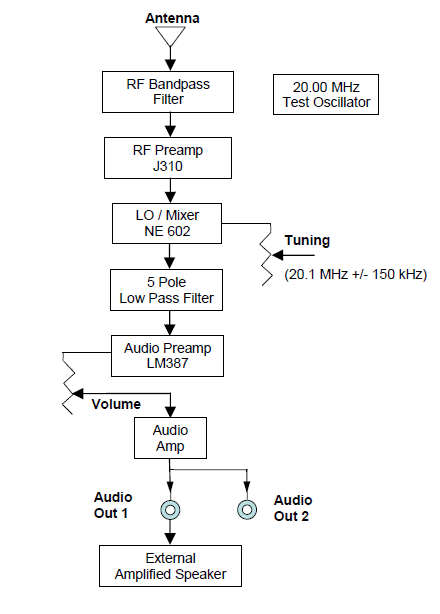
Spiraling Electrons

Synchrotron Radiation

Earth’s atmosphere acts as a barrier to much of the electromagnetic spectrum. The atmosphere absorbs most of the wavelengths shorter than ultraviolet, most of the wavelengths between infrared and microwaves, and most of the longest radio waves. This leaves only visible light, some ultraviolet, infrared, and short wave radio to penetrate the atmosphere and bring information about the universe to the surface of our planet. One of the largest frequency ranges allowed to pass through the atmosphere is referred to as the radio window. The radio window is the range of frequencies from approximately 5 MHz to over 300 GHz (wavelengths of almost 100m down to approximately 1mm). Although these wavelengths have no discernable effect on the human eye, they can be detected with an antenna. Electronic filters in a receiver can be tuned to amplify a small frequency range at a time or, using sophisticated data processing techniques, thousands of separate narrow frequency bands can be detected. With this method it is possible to find out what frequencies are present in the RF radiation and what their relative strengths are.

## Section 1.2 - Radio JOVE Background

The radio JOVE project is a collaborative effort by NASA and several universities that began in 1998, to get students and amateur radio operators interested in radio astronomy by designing a relatively inexpensive direct conversion radio receiver and antenna kit. The kits come with everything you would need to listen to RF radiation coming from Jupiter and the Sun. The receiver is powered either by battery or wall transformer, has an adjustable gain feature, and an audio jack to plug headphone into or to connect it directly to the microphone jack on a personal computer. Software is used to see waterfall plots, strip charts and record data which can be shared with other amateur radio astronomers and scientists by submitting your data online. The receiver its self is a simple design. A block diagram can be seen in figure 1.2.1. The main limitation of the radio JOVE receiver is the narrow 3.5KHz range limited by the lowpass filter and the small dynamic range of 25dB.

  
Figure 1.2.1: Radio JOVE receiver block diagram

## Section 1.3 - Software Defined Radio Background

A software-defined radio system, or SDR, is a radio communication system where components that have typically been implemented in hardware are instead implemented by means of software on a personal computer or embedded computing devices. The ideal receiver scheme would be to attach an analog-to-digital converter to an antenna. A digital signal processor would read the digitized data from the converter, and then its software would transform the stream of data from the converter to any other form the application requires. Most present day receivers use a variable-frequency oscillator, mixer, and filter to tune the desired signal to a common intermediate frequency or baseband, where it is then sampled by the analog-to-digital converter.

In some applications it is not necessary to tune the signal to an intermediate frequency and the radio frequency signal is directly sampled by the analog-to-digital converter (after amplification). Real analog-to-digital converters lack the dynamic range to pick up sub-microvolt, nanowatt-power radio signals. Therefore a low-noise amplifier must precede the conversion step and this device introduces its own problems. For example, if spurious signals are present (which is typical), these compete with the desired signals within the amplifier's dynamic range. They may introduce distortion in the desired signals, or may block them completely. The standard solution is to put band-pass filters between the antenna and the amplifier, but these reduce the radio's flexibility - which some see as the main purpose of a software defined radio. A simple block diagram of an SDR receiver is shown in Figure 1.2.

Figure 1.2: Simple Generic SDR Receiver Diagram

X100

ADC

Band-pass   
filter

Antenna

Gain Stage

Local Oscillator

DSP

Mixer

The receiver performance of this style of SDR is directly related to the dynamic range of the analog-to-digital converters used. Radio frequency signals are down converted to the audio frequency band, which is sampled by a high performance ADC. The newer software defined radios use embedded high performance ADCs that provide higher dynamic range and are more resistant to noise and RF interference. The SDR software performs all of the demodulation, filtering, and signal enhancement.

## Requirements

At the onset of this project we were asked to design a software defined radio receiver to perform the same function as the radio JOVE receiver but with improved performance and a custom software application to process the data. We were given a few target values in terms of hardware performance and a basic description of the desired functionality of the accompanying software and from these requirements we developed our system.

## System Requirements

Since this system was supposed to be an improved model of the radio JOVE receiver, we needed ensure that we did not stray too far from the basic design features of the direct conversion receiver, mainly its portability, and ability to be powered from either a wall transformer or a battery source. With these things in mind the system had to be able to be easily transported by a single person, powered from either a wall transformer or from a DC source in the range of 13V – 20V and also have circuitry to protect the receiver from overvoltage and from applying power with the incorrect polarity. It was also specified that the system could not be solely powered via USB. The system was required to be secured within a metal enclosure, with access to connectors for an antenna input, signal out, power switch, power jacks, and access to a USB port.

## Hardware Requirements

The hardware specifications required that the system be able to measure and record a 1MHz bandwidth centered at 20.1MHz, split the signal into its in phase and quadrature components and transmit that data over USB to a personal computer. The system was required to be able to detect signals as small as 1μV at the center frequency of 20.1MHz.

It was required that a high performance 14bit or greater ADC be used in the design. It was desired that the system have a dynamic range of approximately 100dB and it was required that the system be able to mix down the signal of interest to audio frequency before the ADC. This was to be used strictly for testing purposes to verify proper operation of the receiver.

## Software Requirements

The software application was required to be run on a mid-range laptop running Linux. It should have four different modes of operation: real time, radio astronomy, full record, and data analysis.

In real time mode the application should plot to the screen frequency vs. time vs. signal strength and be able to process live data into audio which can be saved to the hard disk. The user should have control over which part of the 1MHz bandwidth is processed to audio.

In radio astronomy mode the application should give the user the ability to plot a live or previously recorded data by taking the integrated RMS average of signal energy over the entire 1MHz bandwidth and plotting this average over time. The user should also be able to export the averaged data in a parsable format.

The full record mode is required to save all unprocessed, digitized I and Q streams to a file. The user should be able to configure how much data the program should store. The user should be able to select whether a buffer should be used to only store a certain amount of data, or whether the program should continue to record until out of disk space. Also while in “Full Record” mode, a waterfall plot should be displayed.

The data analysis mode is required to give the user the ability to analyze previously recorded I/Q data recorded in “Full Record” mode or in “Radio Astronomy” mode.

A graphical user interface is required in order to present the user with different modes of operation, different ways of capturing the data, and options for what data should be captured or displayed. The GUI should have a file menu, toolbar, options area, and pane for displaying plots.

# Section 2.0 – System Overview

The overview of the system hardware is shown below. The power requirements will be explained in the section dealing with power. The power supply is housed in an aluminum box along with the RF front end circuitry. The enclosed system will accept either a battery input (two plug red/black input) or a barrel jack input from a switching regulator. Its other input is a 50-Ohm BNC input. The enclosed system output a 6V line through a barrel plug designed to power the analog-digital converter board, as well as the RF system output through another 50-Ohm BNC jack. The second box houses the FPGA board as well as the analog-digital converter board. This enclosure has a barrel plug input for the power supply for the analog-digital converter. It also has a 50-Ohm BNC jack for the input signal to the converter. The enclosure has a single USB output. The overall system is diagramed in Figure 2.0.1.



Figure 2.0.1 – System Overview

# Section 3.0 - Enclosures

The project itself is split between three separate enclosures. The first enclosure houses the RF frontend and the power board and in the second enclosure the ADC and FPGA. The third box contains an audio mixer board which is used for system testing to ensure the RF front end is working properly. There are several terminals and connectors on each of the enclosures.

## Section 3.1 - RF Frontend and Power Board Enclosure

The box chosen for this enclosure was a prefabricated unpainted aluminum case from Hammond Manufacturing, specifically the J-884 PL/UNPD which separates into two U-shaped pieces. The two pieces of the enclosure slide into each other and lock into place when the rivets of the top half pop into the holes of bottom half. The box can quickly be opened up by simply applying pressure to both sides of the box and removing the top half.

On the front face of the enclosure there are two banana plugs which can be used to power the system with a DC source such as a battery or benchtop power supply. The color of each plug represents the expected polarity; Red for positive and black for negative. There is also a 2.1mm DC power jack which can be used to power the system from a wall transformer. To properly power the system, a voltage of 13V – 20V is expected with the capacity to deliver 1.5A of current. Also on the front face of the enclosure just above the power terminals is a three position switch which can be used to select the power source or turn the system off entirely. The left most position of the switch selects the wall transformer supply while the right most position selects the banana plug terminals. The middle position of the neutral position where neither side is selected. On the back face of the enclosure is also another 2.1mm DC power jack which is used to power the ADC in the second enclosure.

There are also two female, panel mounted, BNC connectors on the RF frontend and power board enclosure. The BNC connector on the front panel is designated for the antenna input while the BNC connector on the back panel is used to connect the RF front end to the ADC.

The board assembly is supported and mounted to the enclosure using aluminum standoffs which electrically ground both the RF frontend and the power board to the aluminum enclosure. The aluminum case helps to shield the sensitive RF frontend from outside sources of RF radiation as well as act as an excellent heatsink. Rubber feet on the bottom of the enclosure ensure that there is clearance between the bottom of the box and what ever it may be resting on. This is particularly important if it is placed on anything metallic which may have a ground at a different reference level, as this may reduce the performance of the RF front end or even potentially damage the sensitive analog components.

## Section 3.2 - ADC and FPGA Enclosure

The enclosure chosen to house the ADC and FPGA is the 1415F produced by Hammond Manufacturing. This relatively large enclosure is made of 18 gauge steel and has two end panels which can be removed by unscrewing six screws from each panel. This particular enclosure was chosen because of the unique ADC and FPGA board assembly which requires a larger area to mount the boards.

This enclosure only has two external panel mounted connectors, where one of the connectors is for the input signal from the RF frontend and the other connector is a 2.1mm DC power jack used to power the ADC. Both the ADC and FPGA are mounted to the enclosure using nylon standoffs which ensure that neither the ADC’s nor the FPGA’s ground references are electrically connected to the case.

One of the reasons a multiple enclosure scheme was utilized was for modular testing purposes (the RF frontend and power board could easily be tested separate from the ADC and FPGA), and to keep the RF ground, Analog ground, and Digital ground separate to reduce the amount of noise fed to the digital side of the project which could drastically effect the performance of the ADC.

## 

## Section 3.3 - Audio Mixer Enclosure

The enclosure chosen for the audio mixer is a small aluminum project box purchased from RadioShack. The mixer printed circuit board is just the right size for this enclosure. There are two access holes drilled into the front face and side of the box. On the front face of the box is a 2.1mm DC power jack and on the side of the box there is an SMA connector extends out from the box. The audio mixer is only used to test the operation of the RF frontend and as such, the ADC and FPGA will not be connected to either power or signal out. Instead, the audio mixer is powered from the barrel jack on the RF frontend and the output signal from the RF frontend is connected to the mixer circuit.

Opposite the DC power jack is a 3.5mm male stereo cable. This cable can be connected to the microphone jack on the personal computer and with a single command in the terminal window, enable the audio from the stereo jack to be played through the computers internal or external speakers. Alternatively, a female to female adaptor cable may be used if headphones are the preferred method of playing the mixer audio.

# Section 4.0 – Analog RF Front End

This section deals with the analog circuitry of the RF front end. This circuit contains the RF amplifiers, IF Amplifiers, and filters required to boost the desired signal and filter out any unwanted frequency content. The overall block diagram of the system is shown in Figure 4.0.1.



Figure 4.0.1 – Cascaded System Block Diagram

## Section 4.1 – Pre-Filtering

The first section of the circuit is the pre-filter. This is a bandpass filter designed to filter out the majority of the unwanted power in the spectrum. It is a third order filter, with an approximate passband of 2 MHz. The filter was designed using the freeware version of the Elsie program from Tonne Software. The circuit itself is shown below in Figure 4.1.1.

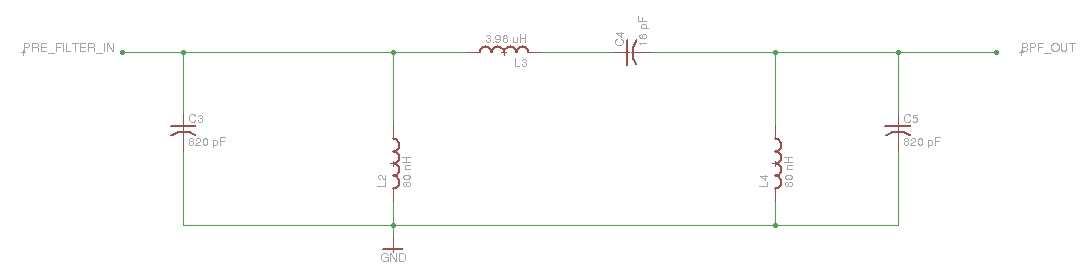


Figure 4.1.1 – Pre-Filter Circuit

The circuit was prototyped, and its scattering parameters were gathered using the network analyzer. Figure 4.1.2 shows its gain characteristics, and Figure 4.1.3 shows its output reflection coefficient (which can be used to determine the output impedance).

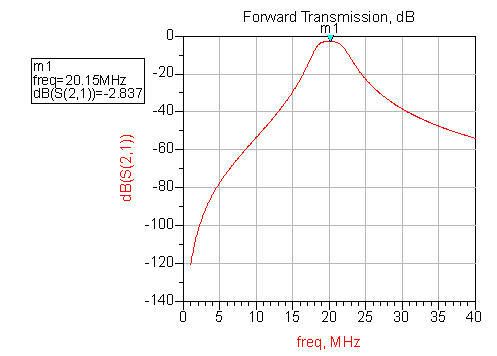


Figure 4.1.2 – Filter Gain

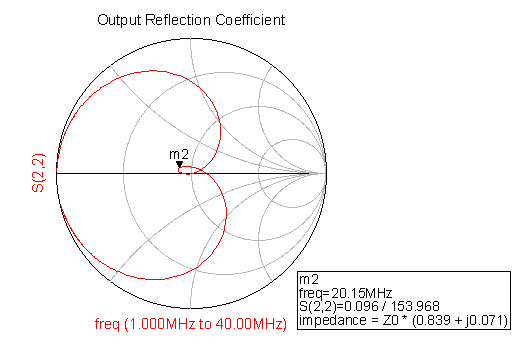


Figure 4.1.3 – Prefilter Output Reflection Coefficient

## Section 4.2 – RF Low Noise Amplification

For the low-noise amplifier stages, the MAX2611 low-noise amplifier from Maxim was used. The circuit used to bias the integrated circuit is shown in Figure 4.2.1.

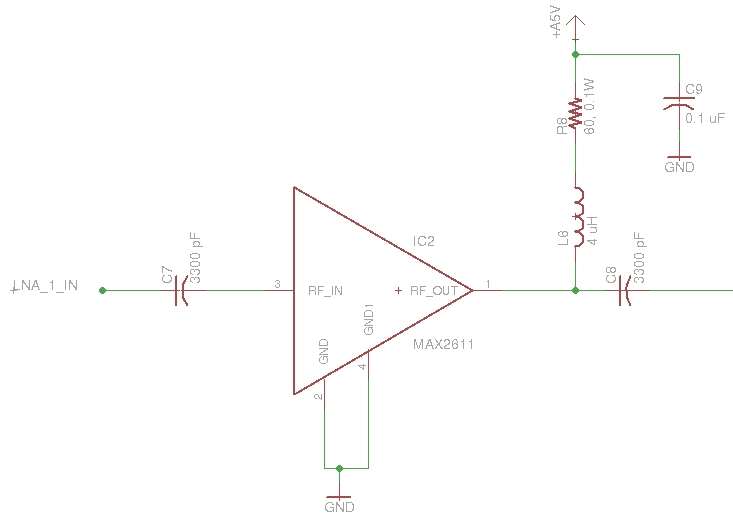


Figure 4.2.1 – Low Noise RF Amplifier

The RF choke was chosen to give 500 Ohms of reactance at the given frequency.

The capacitance of the blocking capacitors was chosen based on the absolute minimum frequency that the RF amplifier would reject (approximately 16 MHz).

The biasing resistor was chosen to drive the amplifier with a bias current of 20 mA at a nominal device voltage of 3.8 V given a 5V supply voltage.

The circuit was prototyped, and its scattering parameters were gathered using the network analyzer. Figure 4.2.2 shows its gain characteristics, and Figure 4.2.3 shows its output reflection coefficient (which can be used to determine the output impedance).

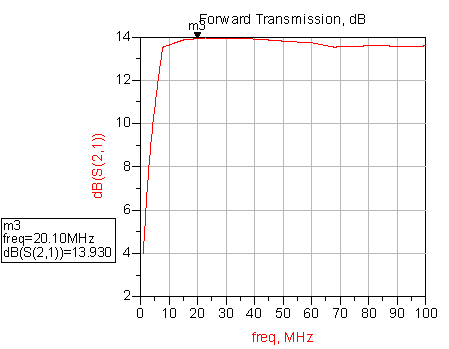


Figure 4.2.2 – MAX2611 Gain

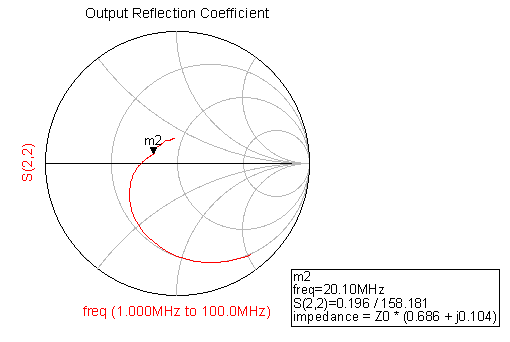


Figure 4.2.3 – MAX2611 Output Reflection Coefficient

## Section 4.3 – IF Amplification

For the intermediate frequency amplifiers, the RF2312 was used. It is an amplifier designed specifically for use after a low noise amplifier, and provides high gain with high IIP3, helping to maximize dynamic range and minimize system noise. The schematic for this amplifier is shown in Figure 4.3.1.

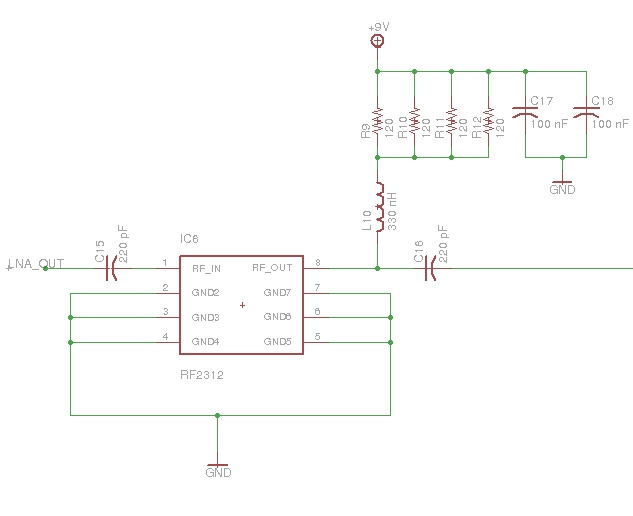


Figure 4.3.1 – IF Amplifier

The four resistors in parallel are designed to maintain the required biasing resistance of 30 Ohms while allowing for a greater current path (minimize individual power dissipation). The amplifier resistance is chosen for biasing was 30 Ohms based on the device current of 120 mA, and the nominal device voltage of 5.5V.

The RF choke inductance and input/output capacitance were chosen based on results from AppCAD (see Figure 4.3.2).

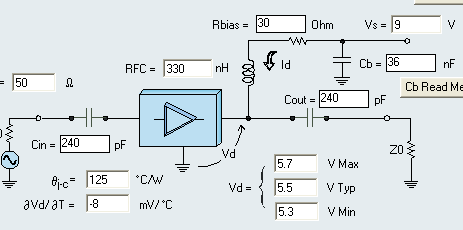


Figure 4.3.2 – AppCAD Results

## Section 4.4 – Anti-Aliasing Filter

The anti-aliasing filter is a 5th order Chebyshev coupled-resonator filter, with a bandwidth of 700 kHz. This filter was designed to prevent aliasing in the FPGA and the ADC. Since the ADC samples at 100 MHz, the anti-aliasing filter could not allow anything beyond 25 MHz inside the digital part of the system. Thus, at 25 MHz, the filter will have attenuated any signal by over 100 dB. The coupled resonator topology provide quickly attenuating narrow band filters. In addition, it allows for the use of only one inductance value for every inductor. However, these filters require extremely high Q components. Since, the required Q to maintain unity gain is not truly realizable, the true bandwidth of the system is approximately 1 MHz, and the loss can be modeled at approximately 20 MHz. So, system gain must be added after the filter in order to make up for the loss.

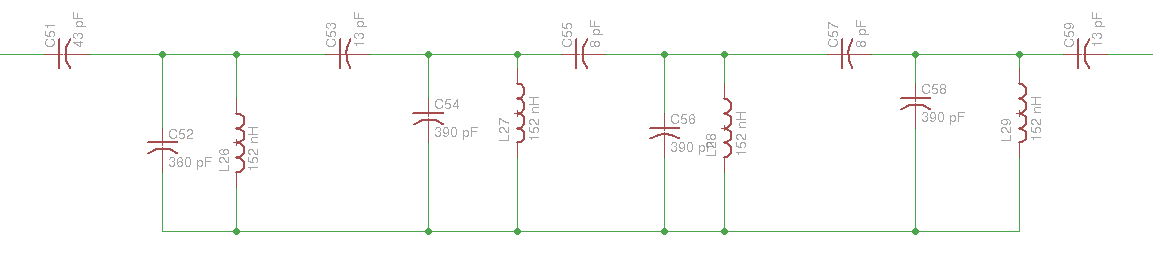


Figure 4.4.1 – Anti-Aliasing Filter

The circuit was prototyped, and its scattering parameters were gathered using the network analyzer. Figure 4.4.2 shows its gain characteristics, and Figure 4.4.3 shows its output reflection coefficient (which can be used to determine the output impedance).

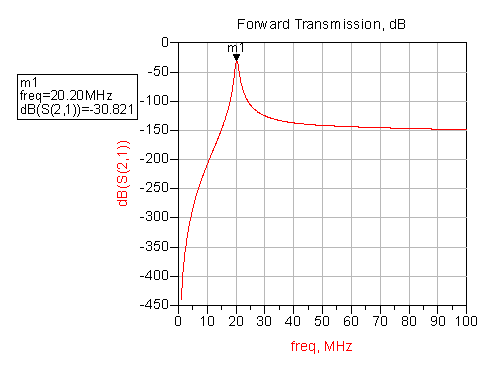


Figure 4.4.2 – Anti-Aliasing Filter Gain

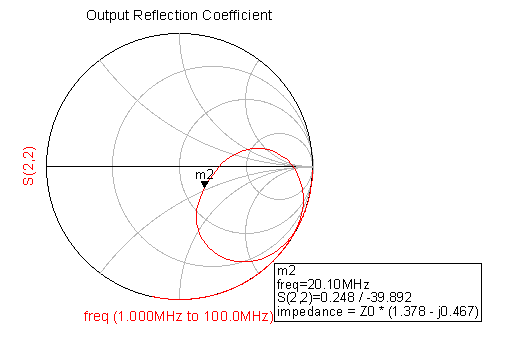


Figure 4.4.3 – Anti-Aliasing Filter Output Impedance

## Section 4.5 – Impedance Matching

If one recalls the Maximum Power Transfer Theorem, one will remember that maximum power transfer occurs in when the load and source impedances are matched. In between every block in the cascade of amplifiers and filters on the RF board, there is an impedance matching block designed to keep the overall system input and output impedance at 50 Ohms within the bandwidth of interest, thus minimizing power losses in the path that would be due to impedance mismatch. These impedance matching networks typically take the form of either low-pass of high-pass LC filters. They were designed by generating the theoretical s-parameters of the system output at the end of each successive stage, and matching the load to 50 Ohms (since the input impedance that each integrated circuit in the design wants to see is 50 Ohms) using smith charts.

## Section 4.6 – Final Results

The s-parameters of the final circuit were simulated. The gain, output reflection coefficient, and input reflection coefficient are shown as Figure 4.6.1, 4.6.2, and 4.6.3 (respectively).

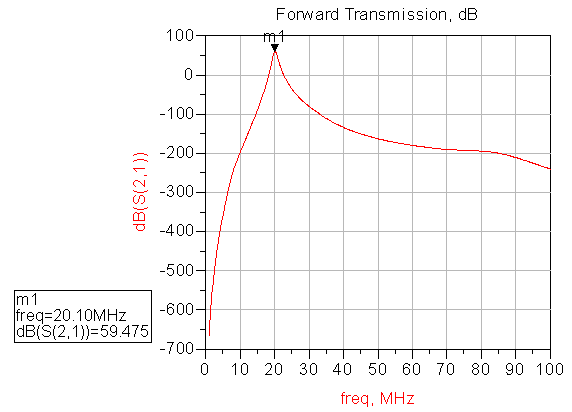


Figure 4.6.1 – System Gain

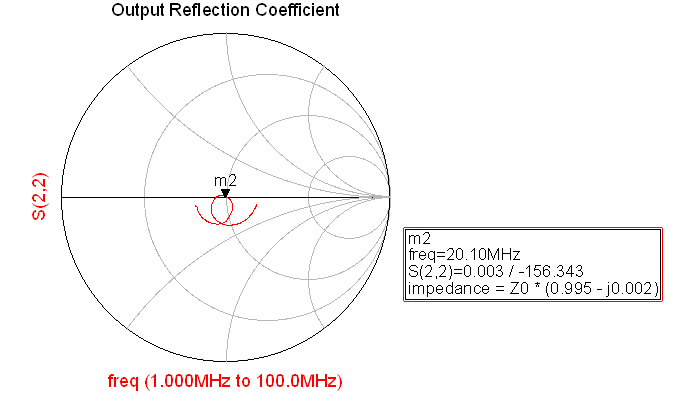


Figure 4.6.2 – System Output Reflection Coefficient

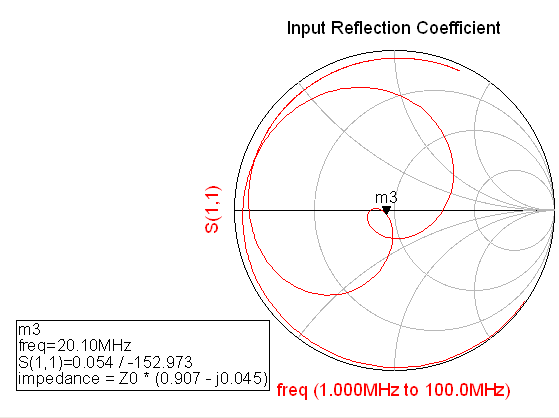


Figure 4.6.3 – System Input Reflection Coefficient

The final gain of the circuit was measured by using a system input from a function generator, and the system output was measured using the oscilloscope. A 10 mVpp 20.1 MHz sine wave was output from the function generator. With the oscilloscope input set at 50 Ohms, the measured peak-to-peak amplitude was 1.82 V with no visible distortion. Thus, the output gain can easily be calculated.

However, one must also take into account the fact that a 12 dB attenuator was present on the system output at the time. Therefore, the actual output is given by Equation 4.6.2.

The minimum discernible signal was tested by first passing a 20.1 MHz signal from a function generator (and attenuated through the channel created by two antenna) to a spectrum analyzer such that the signal level was just barely above the noise floor. According to the spectrum analyzer, the signal input was approximately -97 dBm, as shown in Figure 4.6.4.

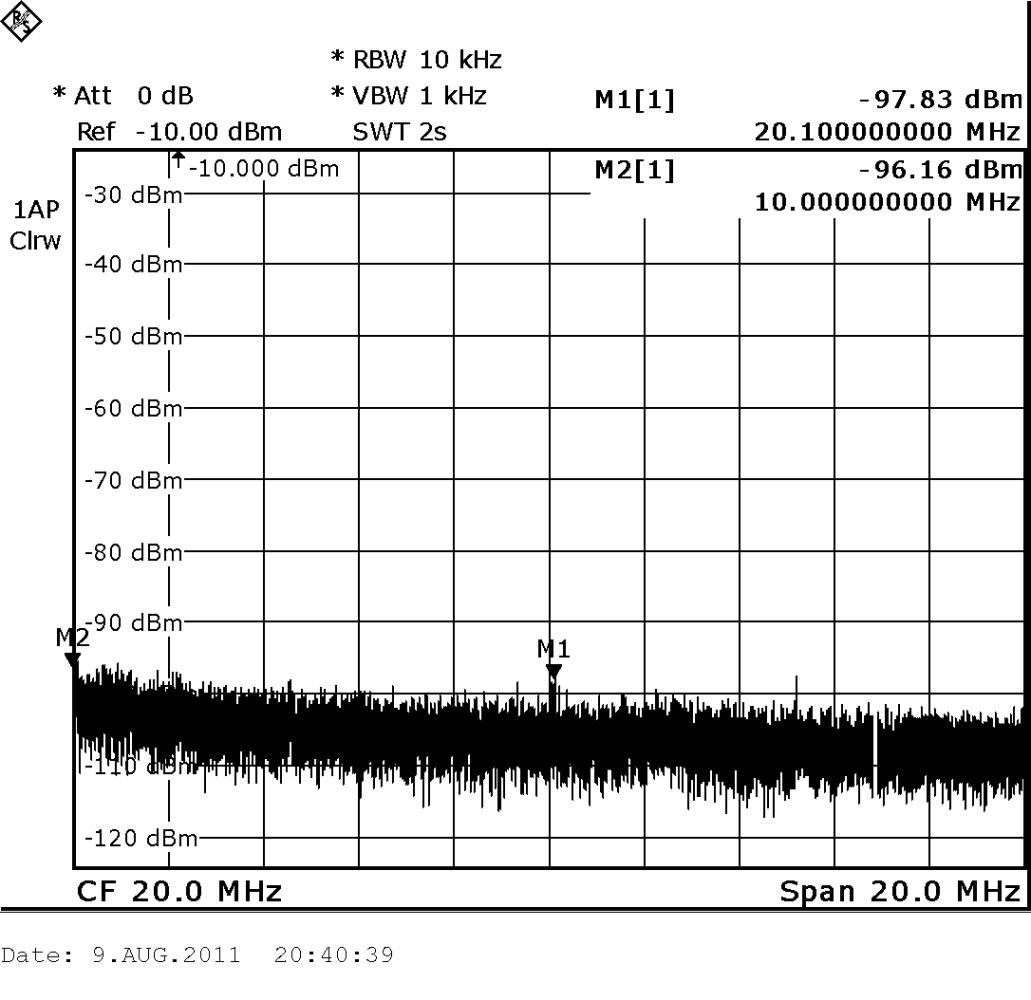


Figure 4.6.4 – MDS Input Test Signal

Next, this minimum signal was passed through the receiver itself. The output of this was observed on the spectrum analyzer, and is shown in Figure 4.6.5.



Figure 4.6.5 – Signal Level At Receiver Output

Next, the signal was attenuated until it reached the noise floor in the passband of the receiver. This is shown in Figure 4.6.6.

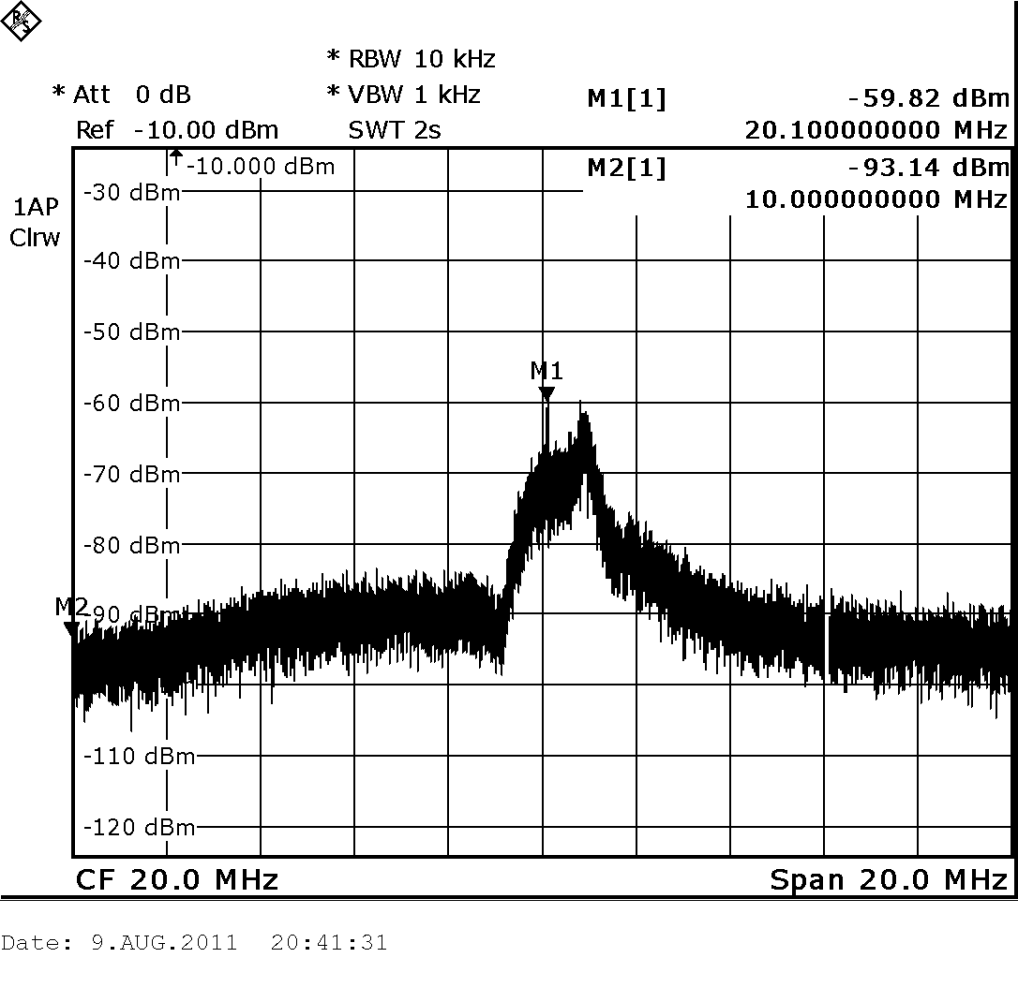


Figure 4.6.6 – Receiver Output After Attenuation

At the minimum signal level that the function generator could generate, it still was not enough to fully attenuate the signal to the noise floor in the passband of the receiver. The receiver was nonetheless removed from the path, and the antenna output was plugged directly into the spectrum analyzer. The output is shown in Figure 4.6.7.

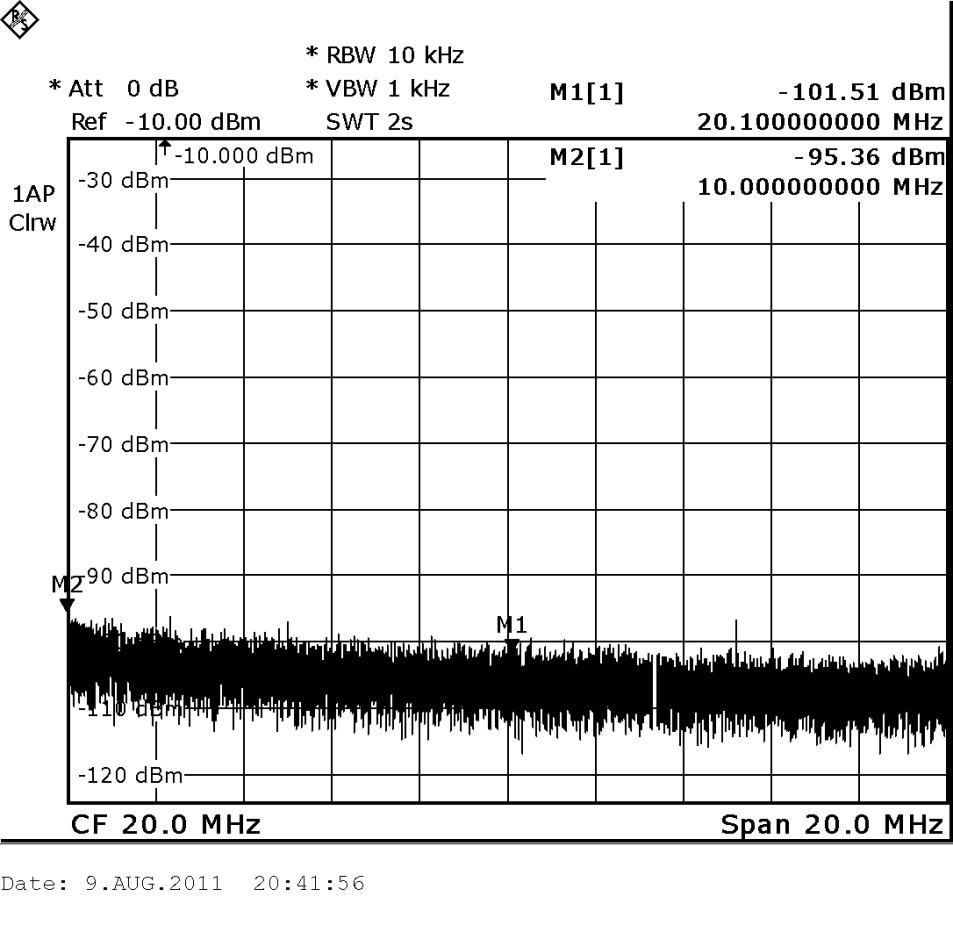


Figure 4.6.7 – Raw Signal Output

Thus, the function generator cannot even produce a signal that can bring the receiver output to its noise floor. But, as best as the spectrum analyzer can measure, the minimum discernable signal that the function generator can detect above the noise floor is -101 dBm. However, since the function generator could not provide enough signal attenuation to fully bring the received signal output to the receiver noise floor, it stands to reason that the noise floor is even further below this level, and may indeed be the projected -106 dBm.

Next, the spurious free dynamic range was measured. This was accomplished by first hooking up the receiver output to the spectrum analyzer, and forcing the function generator to output two sine waves at differing frequencies (20.1 MHz and 20.2 MHz) through the same channel (accomplished using the ‘add external input’ function on the function generator). The system output is shown in Figure 4.6.8. The amplitude of both sine waves was increased until the first spur due to intermodulation distortion could be seen.

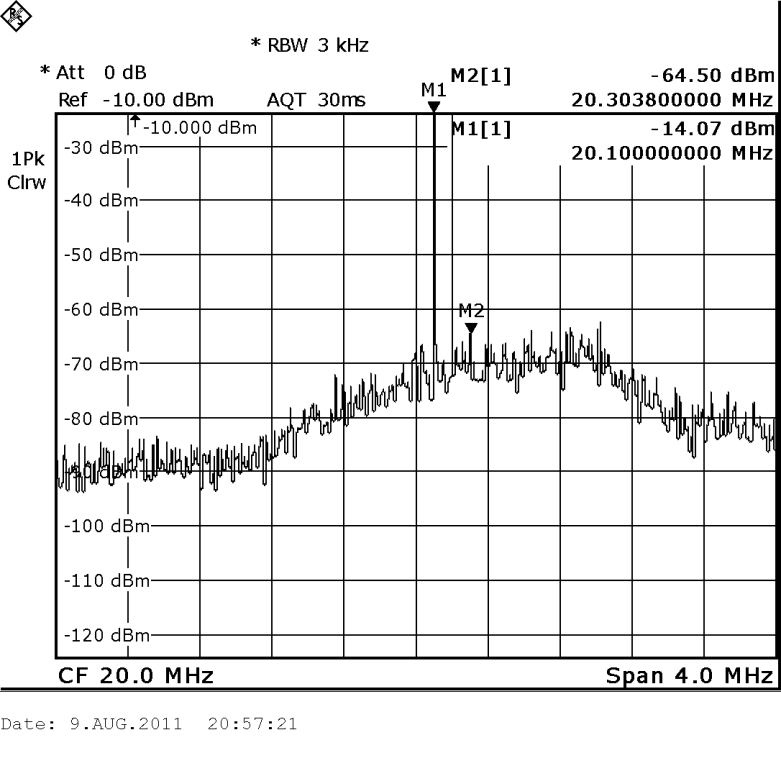


Figure 4.6.8 – IMD Spur for SFDR Test

Next, the signal amplitude was decreased until this spur disappeared. The result is shown in Figure 4.6.9.

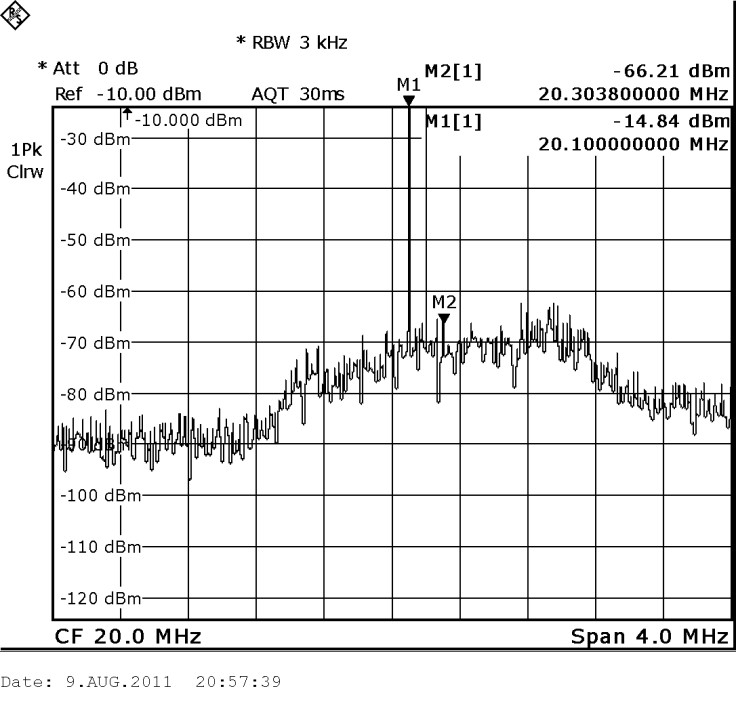


Figure 4.6.9 – Amplitude Reduction to Eliminate Spur

To determine the dynamic range, the signal level difference between the highest receiver signal and the noise floor of the receiver (stopband) at this amplitude was observed (delta marker in spectrum analyzer). This is shown in Figure 4.6.10.

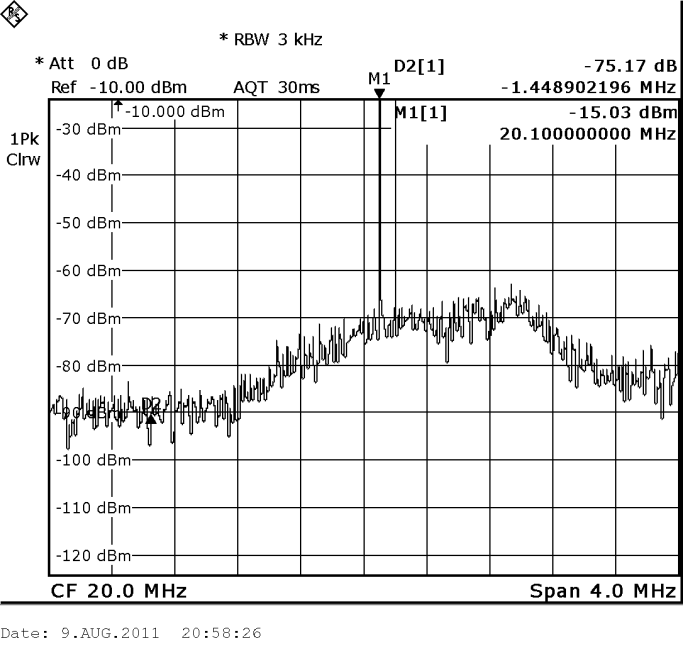


Figure 4.6.10 – Determination of SFDR

Thus, the observed SFDR is approximately 75 dBm.

## Section 4.7 – Output Power Protection

Theoretically, the results of both the simulation and the actual board tests show that the system has a possible gain of 57 dB. This is the gain necessary to increase the minimum discernable signal to a level that can be read by the ADC. However, one must also consider the maximum discernable signal level. If the dynamic range of the system is 75 dB, then the maximum discernable signal is -32 dBm. When 57 dB of gain is added to that, the result is a maximum input signal to the digital side of the system of 25 dBm. This is an input power of around 40 mW, and an input current of around 80 mA. The AD9460 does not expect an input current much beyond 35 mA. Therefore, a series PTC was placed at the output of the RF front end circuitry. This way, if the RF board ever tried to push more than 35 mA of current, the circuit will blow, but will be useable again after a power reset. For increased range, an attenuator was placed at the output of the circuit (7 dB). However, to detect the smallest signals possible, the attenuator should be removed. The circuit can still detect the minimum discernable signal without the attenuator. However, the system is more accurate with extremely low-level signals with the attenuator off.

# Section 5.0 – Power

The next board to discuss is the power board. This board was designed to supply the RF Front End board and the ADC board.

## Section 5.1 – Distribution and Protection

The expected current load of each subsystem (determined through prototype testing) is summarized in Table 5.1.1.

Table 5.1.1 – Sub-circuit Current Draw

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Circuit | Quantity | Voltage [V] | Current [mA] | Total Current [mA] |
| LNA | 3 | 5 | 18 | 54 |
| IF Amplifer | 3 | 9 | 120 | 360 |
| ADC | 1 | 6, 5, 3.3 | 612 | 660 |
|  |  |  |  | 1074 |

The current was distributed through two main paths. The first path was designed to supply the RF front end board, the second path was design to supply the ADC.

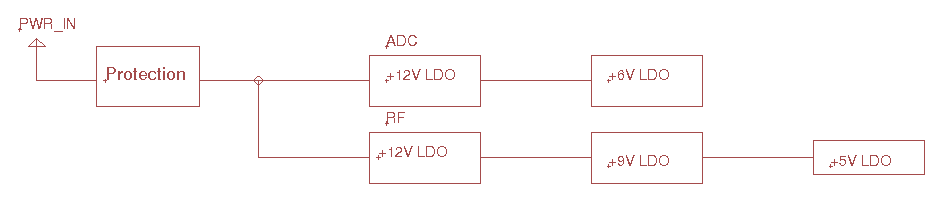


Figure 5.1.1 – Distribution Path

Both paths begin with a 12V regulator to step down the input voltage to a common 12V level. In practice, a 13.5V supply was used for all testing and initial prototyping. However, in theory, any voltage from slightly above 12V to 18V could be used as a supply. The assembly itself allows the user to attach a standard 2.1 mm barrel plug for power, and it also has banana plug jacks for battery or auxiliary power.

The main source of power-supply protection is protection from excessive current. Two parallel PTC’s are placed in series with the input supply rail. Whenever the current exceeds 1.5A, the PTCs will trip and the input will open. The power line will be restored once the system is reset.

## Section 5.2 – Regulation

The power system uses low dropout regulators exclusively for voltage regulation. Although switching regulators are generally more favorable because of their high efficiency, LDO’s were necessary in this application because of the switching noise that would contribute to the overall noise floor is switching regulators were used. LDO’s typically have much better noise performance. Standard adjustable regulators were used (see Figure 5.2.1)

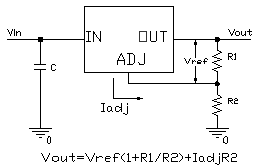


Figure 5.2.1 – Standard Adjustable LDO Circuit

## Section 5.3 – Low-Noise Augmentation

The noise performance of the LDO circuit can be further improved. The idea for this improvement came from an application note from Maxim Integrated Circuits[[1]](#footnote-1). Using a simple RC low-pass filter and a Darlington transistor, one can reduce the noise density of the LDO considerably, which will of course reduce the overall noise voltage. The filter is designed with a very low cutoff frequency. Typically, the noise voltage density for LDO’s will be very high in the kHz range, and will die off such that it is virtually nonexistent at frequencies greater than 1 MHz. The filter allows further attenuation of this low frequency noise, thus reducing the noise voltage density. For the transistor, high gain transistors are preferred, because then the base current requirement for the proper load current is decreased. Furthermore, transistors with high Early voltages should be used, since such transistors reject noise at their base. The typical configuration using the LT1086 to step down 12V to 6V is shown in Figure 5.3.1.

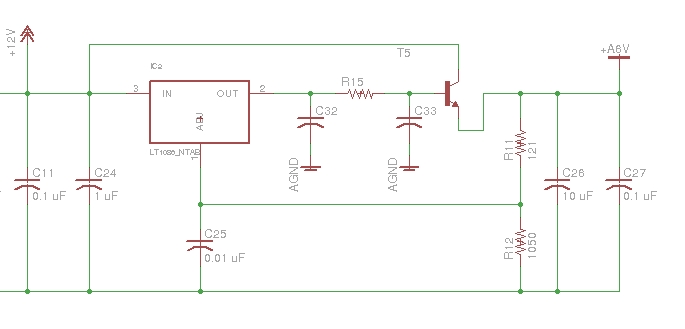


Figure 5.3.1 – Typical Configuration

In the case of the system power circuit, typically third-order pi-configuration low pass filters were used. The resistor had to be chosen to allow as much current to flow to the base of the transistor as possible, to get the full current to pass through the transistor from the input path. Thus, the capacitors needed to have fairly high capacitances to keep the cutoff frequency low. Also, the resistance values to adjust the voltage vary from those specified by the datasheet. Although the noise is lowered considerably with this circuit, the load regulation suffers. Simulation results were used to tune the voltage outputs such that they would vary within an acceptable degree within the calculated current demand variations.

# Section 6.0: Analog to Digital Converter (ADC)

The analog digital converter (ADC) is an important part of the system. The ADC has to convert the analog input signal into a digital signal that can be processed by the PC. The ADC accomplishes this by sampling the incoming continuous analog signal into a discrete digital signal. The sampling rate of the ADC must be chosen so that it is high enough to sample the sinusoidal waves that will be coming through the input. In the frequency domain, the sampled signal is the convolution of the spectrum of the original signal and the impulse train (comb function). Therefore, there will be spectral copies of the original spectrum at every integer multiple of the sampling frequency. This is illustrated in Figure 6.1



Figure 6.1: Spectrum of a Sampled Baseband Signal

If the signal is sampled too slowly, aliasing will occur. When aliasing occurs, the spectral copies of the sampled signal bleed into each other, thus making it impossible to reconstruct the signal later. The Nyquist Theorem can be used to determine how fast the signal should be sampled to avoid this. It states that the sampling frequency should be at least twice as high as the highest frequency that the signal contains.   
Often, it is sufficient to sample at exactly twice the highest frequency contained in the baseband signal. However, with sinusoids, some issues occur when this is attempted. Sine functions have the same value when they are sampled at the start of their period, halfway through their period, and at the end of their period. In order to capture a representative sample of the behavior of a sine wave, a higher sampling rate must be used. A sampling rate of 8 or 16 times the highest frequency would yield a more accurate representation.

## Section 6.1: Dynamic Range and Resolution

Dynamic range is defined as the ratio between the largest and smallest possible signals the receiver is able to detect. The dynamic range of the receiver was desired to be approximately 100dB. The dynamic range of an analog to digital converter with N-bit uniform quantization is defined by:

A 16-bit ADC would have an ideal dynamic range of 96.3dB however we cannot treat this as ideal and the goal of a 100dB of dynamic range is already out of reach. The data sheet for the AD9460 publishes a value for the effective number of bits at several different frequencies and by using this number it is possible to treat a non ideal 16-bit ADC as an ideal 12.7-bit ADC resulting in a dynamic range of 76.46dB.

The value for the effective number of bits is also very helpful in determining the resolution of an ADC. When the reference voltage is set to 3.4V, the largest value that can be input is 1.7V on a single analog input. The resolution of this ideal 12.7-bit ADC is equal to the reference voltage divided by the total number of quantizational levels. From this resolution we can calculate the minimum amount of gain required for the system to be able to measure and record microvolt level signals.

The AD9460 uses differential analog inputs which effectively double the signals peek-peek voltage. This is primarily done to remove any common mode noise present in both signal paths to improve the overall performance of the ADC. As an example, when 1µV is applied on a single analog input, the difference will result in a 2µV peek-peek signal. Since the resolution of the ADC is 510µV and an analog input of 1 µV results in a 2µV signal, a minimum gain of 256 or 48dB is required, and this will just allow the signal to be detected. By multiplying our lowest desired signal, we have also effectively reduced our upper limit by the same amount. By applying 48dB of gain we can theoretically detect a 1µV level signal but the maximum input signal per analog input has now been reduced from 1.7V to 6.64mV. In the final design, the RF front end provides the system with 57dB of gain, which amplifies a single 1µV input to 0.708mV or 1.41mV differentially and reduces the maximum input to 2.40mV single ended or 4.80mV differentially.

## Section 6.2: AD9460 Development board

For this project we selected the AD9460 development board as the ADC platform of choice. The AD9460 is a 16-bit, 105 MSPS, with a selectable 2.0 t0 4.0V p-p full scale differential input. It can also be configured for LVDS or CMOS outputs. The development board comes preconfigured to output LDVS but also implements a conversion from LVDS to TTL for the FPGA to sense the level of each bit.

The reference voltage of the ADC can easily be adjusted by populating two resistors on the board to properly bias the voltage seen by the sense pin. For our application, we wanted to utilize the entire 3.4V span. This was achieved by tying the sense pin to analog ground.

The AD9460 also offers the ability to choose the data format that is output. There are two different formats, twos compliment and offset binary. The properties of the two coding schemes can be seen in Figure 6.2. The different formats are easily selectable on the development board by adjusting the DFS jumper to either high for twos compliment or low for offset binary. For this project the DFS jumper has been placed in the low position selecting offset binary.

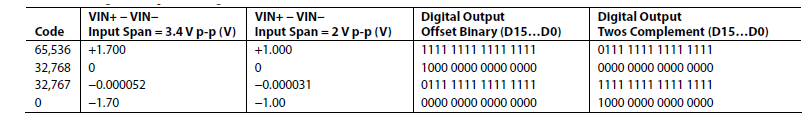


Figure 6.2: Digital output coding

## Section 6.3: Duty Cycle Stabilization

The development board also has optional duty cycle stabilization circuitry. Typical high speed ADCs use both clock edges to generate a variety of internal timing signals and, as a result, can be sensitive to the clock duty cycle. Commonly a 5% tolerance is required on the clock duty cycle to maintain dynamic performance characteristics. The AD9460 contains a clock duty cycle stabilizer (DCS) that retimes the nonsampling edge, providing an internal clock signal with an approximate 50% duty cycle. Noise and distortion performance are nearly flat for a 30% to 70% duty cycle with the DCS enabled. The DCS circuit locks to the rising edge and optimizes timing internally. This allows for a wide range of input duty cycles at the input without degrading performance. Jitter in the rising edge of the input is still of paramount concern and is not reduced by the internal stabilization circuit. The duty cycle control loop does not function for clock rates of less than 30 MHz nominally. The loop is associated with a time constant that should be considered in applications where the clock rate can change dynamically, requiring a wait time of 1.5 μs to 5 μs after a dynamic clock frequency increase or decrease before the DCS loop is relocked to the input signal. During the time that the loop is not locked, the DCS loop is bypassed, and the internal device timing is dependent on the duty cycle of the input clock signal. In such an application, it can be appropriate to disable the duty cycle stabilizer. In all other applications, enabling the DCS circuit is recommended to maximize ac performance. For this project the DCS was connected to analog ground, enabling the duty cycle stabilization.

## Section 6.4: Clocking the ADC

The AD9460 input sample clock signal must be a high quality, extremely low phase noise source to prevent degradation of performance. Maintaining 16-bit accuracy places a premium on the encode clock phase noise. SNR performance can easily degrade by 3 dB to 4 dB with 70 MHz analog input signals when using a high jitter clock source. For optimum performance, the AD9460 must be clocked differentially. The sample clock inputs are internally biased to ~1.5 V, and the input signal is usually ac-coupled into the CLK+ and CLK− pins via a transformer or capacitors. Power supplies for clock drivers were separated from the ADC output driver supplies to avoid modulating the clock signal with digital noise. The development board was designed to allow an external clock source to be mounted directly to the board in a DIP-14 package layout in the same location as the SMA connectors. A board was designed to fit the footprint of the DIP-14 package. The oscillator used was the CCHD-950-50-100 which is a 3.3V, 100MHz, CMOS oscillator with a typical RMS jitter of 0.5ps, and phase noise floor of -165dBc. This is an extremely high quality oscillator and has the performance that is desired in clocking high speed data converters. The power to the oscillator was supplied by adding a jumper connecting 5V to XTAL. A 3.3V LDO was used to regulate the 5V down to the oscillators operating voltage. In addition to this clock board, a polarized 0.1uF decoupling capacitor was soldered in place between the signal path and analog ground. The clock signal is then decoupled and split into a differential signal using a RF transformer.

## Section 6.5: SFDR Option

Under certain conditions, the SFDR performance of the AD9460 improves by adding some additional power to the core of the ADC. The SFDR control pin is a CMOS-compatible control pin to optimize the configuration of the AD9460 analog front end. Connecting SFDR to AGND optimizes SFDR performance for applications when analog input frequencies are less than 200 MHz. For applications with analog inputs are greater than 200 MHz, this pin should be connected to 3.3V for optimum SFDR performance; power dissipation from AVDD2 increases by approximately 20 mW. For this project, the jumper was positioned so that the SFDR pin is connected to analog ground.

## Section 6.6: Analog Inputs

It is strongly recommended that the analog input to the AD9460 be differential. Differential inputs improve on-chip performance because signals are processed through attenuation and gain stages. Most of the improvement is a result of differential analog stages having high rejection of even-order harmonics. There are also benefits at the PCB level. First, differential inputs have high common-mode rejection of stray signals, such as ground and power noise. Second, they provide good rejection of common-mode signals, such as local oscillator feed through. The specified noise and distortion of the AD9460 cannot be realized with a single-ended analog input. The AD9460 analog input voltage range is offset from ground by 3.5 V. The analog input voltages are limited by reference voltage selected. The voltage on each input can only have a peek voltage of half the reference voltage. The internal bias network on the input properly biases the buffer for maximum linearity and range. The method used for driving the analog inputs of the AD9460 is using an RF transformer to convert single-ended signals to differential signals. Series resistors between the output of the transformer and the AD9460 analog inputs help isolate the analog input source from switching transients caused by the internal sample-and-hold circuit. The entire signal path has been impedance matched to 50Ω to maintain the quality and power of the signal.

## Section 6.7: Powering the ADC

To power the ADC there are two options one from the RF frontend closure and the other from a wall transformer. It is recommended that when the system is being operated for an extended period of time or if there is access to a wall outlet that the ADC be powered from the provided wall wart. The power board is capable of handling the full 1.5A load but is limited on its ability to dissipate heat. The PTC’s will keep the power board from thermal runaway due to excessive current caused by self heating, but the performance of the system may be effected. For this reason it is recommended that when the receiver is not in use, disconnect the signal cable from the ADC/FPGA enclosure and disconnect the power. It is important that the ADC be powered before the antenna is connected. There is a PTC at the output of the RF frontend to limit the current seen by the analog inputs of the ADC, but if the ADC is not powered the protective biasing circuitry is not operational and a strong signal at the inputs could damage hardware.

# Section 7.0 – FPGA

The general overview of the FPGA subsystem is shown in Figure 7.0.1



Figure 7.0.1 – FPGA Block Diagram (High Level)

Basically, the system needs to down-convert the incoming 1 MHz bandwidth down to baseband. So, the system is mixed with 19.6 MHz, such that the incoming spectrum starts at DC. Then, it is low-pass filtered to remove the image component (which will exist at 39.7 MHz). Then, the signal is CIC filtered to decrease the system sampling rate to a level that can actually be transmitted over USB. A data rate of 20 MB/s was desired for the USB controller. However, the signal was translated into I and Q components, and each of those had to be transmitted a byte at a time (with a 2-byte signal sampled on each clock cycle). Therefore, signal was decimated by a factor of 20. Since the clock rate was 100 MHz, this resulted in a final sampling rate of 5 MHz. However, when the signal was time-division multiplexed for each byte of both streams (2 bytes per stream), the final sample rate of the signal entering the USB controller was 20 MHz.System Generator was used to generate an HDL netlist based on a block diagram of the system, and then Xilinx was used to synthesize and map the system onto the FPGA.

## Section 7.1 – Mixing

The first section in the system is the mixer. The mixer uses a ROM to store a sine wave with 19 bits of accuracy. The next step is to send it through a multiplier. For the purposes of this project, the embedded multipliers were used since they are not implemented in the filters.

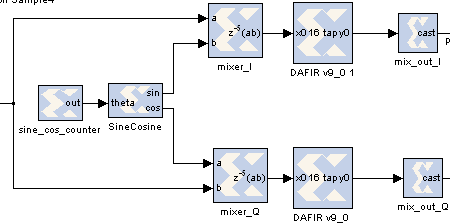


Figure 7.1.1 – Mixer Block Diagram

Next, the system was low-pass filtered. Since the image component is so high, the cutoff frequency was set at 20 MHz to relax the filter specifications. Since the cutoff frequency was not such a small fraction of the sampling frequency, the filter could achieve a good attenuation at 39 MHz while keeping the amount of resources consumed by the FPGA low enough that the rest of the system could be implemented. The filter magnitude response is shown in Figure 7.1.2.

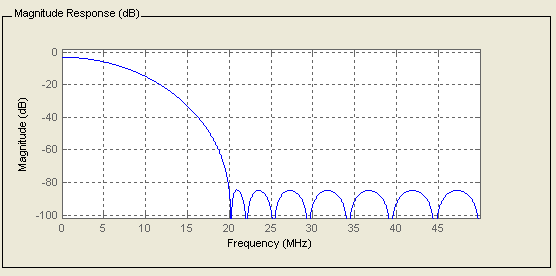


Figure 7.1.2 – Post-Mixer Low-Pass Filter (16 tap FIR, fs = 100 MHz)

The filter was implemented using a distributed-arithmetic architecture, and was structured in such a way that it was completely parallel. Since at this point in the system, the signal has not been down-sampled, this is a necessity. Finally, the filter output is recast in such a way that the normally 36 bit output becomes 16 bit with a binary point at the 14-th bit. The system output with a 20.1 MHz input test signal was simulated, and the results are shown in Figure 7.1.3 (in analog waveform view) and in Figure 7.1.4 (in logical view to get a sense of the timing).

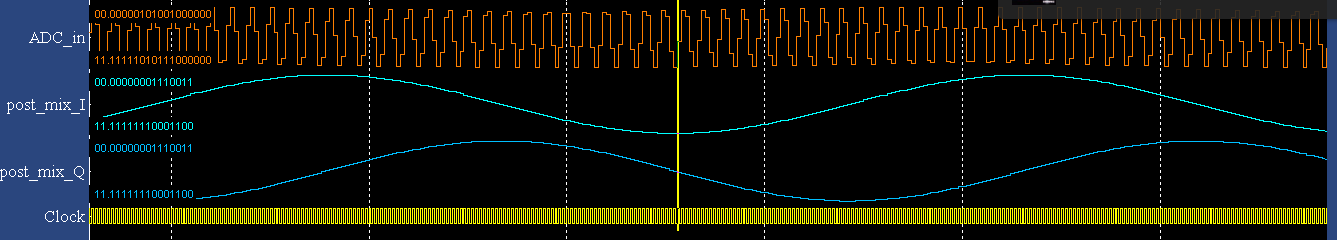


Figure 7.1.3 – Mixer Output Simulation (Analog)

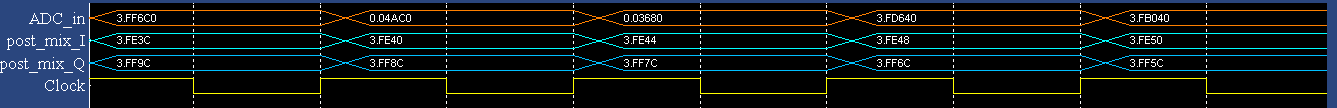


Figure 7.1.4 – Mixer Output Simulation (Logic)

The results show that the sampled input signal has been successfully downconverted to 1 MHz at a parallel rate (the clock rate is 100 MHz).

## Section 7.2 – Decimation Stage 1

Instead of attempting to decimate the signal by a factor of 20 in a single stage, the decimation was accomplished over two cascaded stages. In the first stage, the signal was decimated by a factor of 4. This resulted in a sampling rate of 25 MHz. Instead of true CIC filtering, the signal was simply downsampled by a factor of 4, and then filtered with another distributed arithmetic filter. The filter was made parallel again, however, the input streams were serially combined to take advantage of the lower sampling rate but the still-existent 100 MHz FPGA clock. The block diagram is shown in Figure 7.2.1.

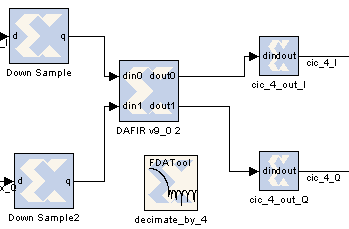


Figure 7.2.1 – Decimation Stage 1 Block Diagram

The FIR filter magnitude response is shown in Figure 7.2.2.

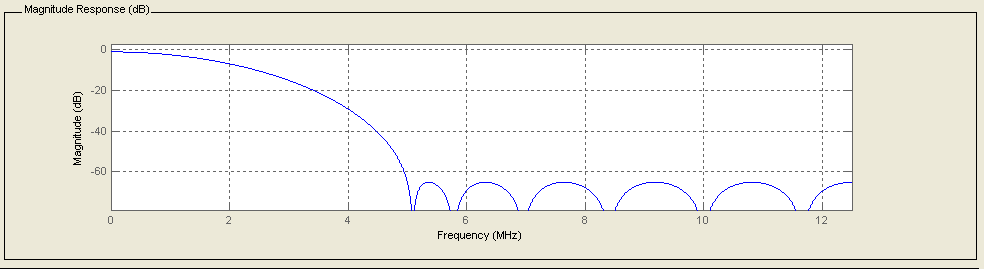


Figure 7.2.2 – Filter Magnitude Response (12 Tap FIR, fs = 25 MHz)

The system output up to this point was simulated, and the results are shown in analog form (Figure 7.2.3) and in logical form (Figure 7.2.4).

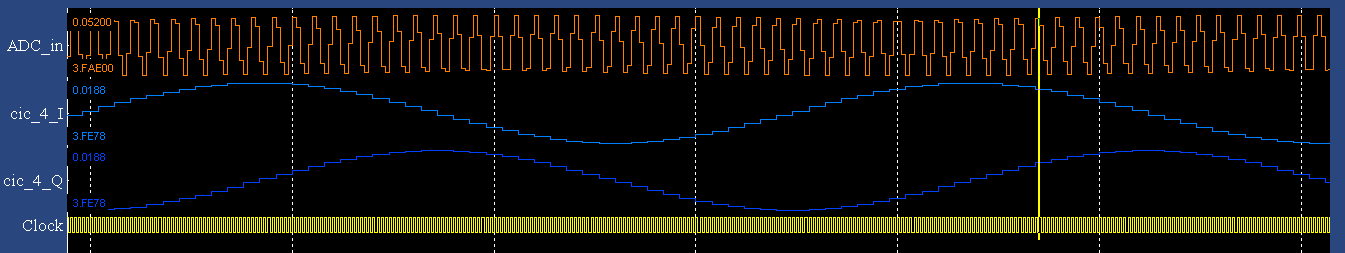


Figure 7.2.3 – Decimation Stage 1 Output (Analog)



Figure 7.2.4 – Decimation Stage 1 Output (Logical)

## Section 7.3 – Decimation Stage 2

The signal was decimated again by a factor of 5, thus bringing the sampling rate down to 5 MHz. This stage of the FPGA block diagram takes the same basic structure as the first decimation stage: there is a downsampling block, followed by an FIR filter, and finally the signal is recast back to a 16-bit width. The FIR filter at this point could theoretically be serialized to a high degree. The two input signals are multiplexed in the filter block, which reduces the overall available clock cycles for each stage by a factor of two. However, that still leaves 10 clock cycles to compute the filtering operations for 16 bits of data. In the actual implementation, this filter was left as a completely parallel filter with a multiplexed input. However, there is the possibility of serialization, which would *theoretically* greatly reduce FPGA resource consumption while maintaining system throughput. The FIR filter magnitude response is shown in Figure 7.3.1.

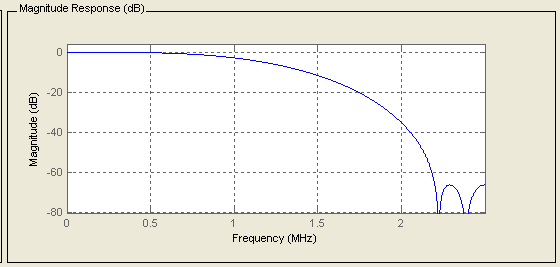


Figure 7.3.1 – Filter Magnitude Response (6 Tap FIR, fs = 5 MHz)

The system output up to this point was simulated, and the results are shown in analog form (Figure 7.3.2) and in logical form (Figure 7.3.3).

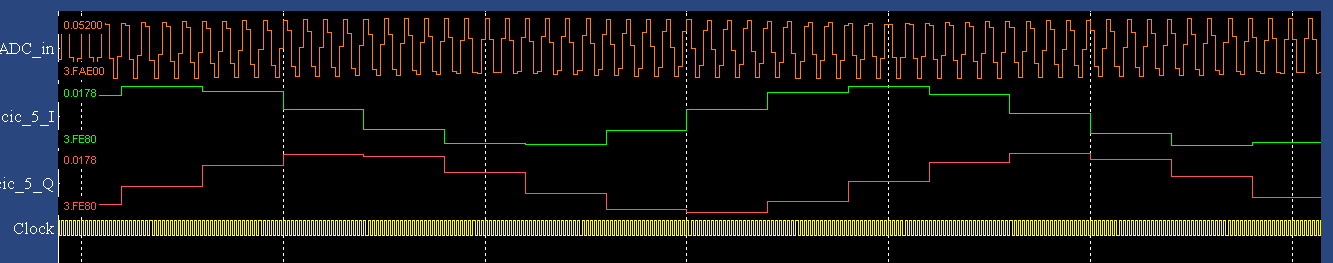


Figure 7.3.2 – Decimation Stage 2 Output (Analog)

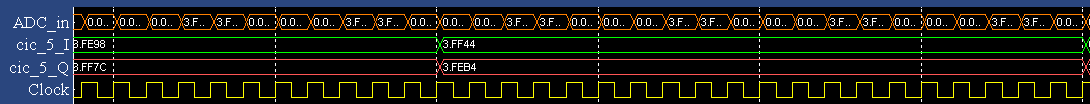


Figure 7.3.3 – Decimation Stage 2 Output (Logical)

## Section 7.4 – Time-Division Multiplexing

For the penultimate section of the FPGA block diagram, each of the two 16-bit streams are split into two 8 bit streams. Then, the 4 streams are time-division multiplexed. So, the four streams are transmitted in the time it would normally take to transmit one of them. So, the sampling rate increases by a factor of 4 (from 5 MHz to 20 MHz). Since there is a byte of data transmitted on every 20 MHz clock cycle, the overall data-rate going to the USB is 20 MBps. The block diagram is shown in Figure 7.4.1.

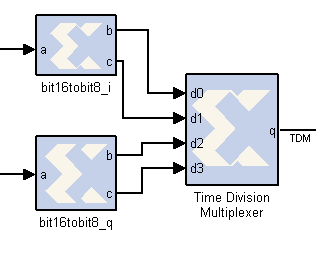


Figure 7.4.1 – Time Division Multiplexing Block Diagram

The system output up to this point was simulated. Only the logical implementation will be shown since it is most useful to see the bit-latch timing (Figure 7.4.2).

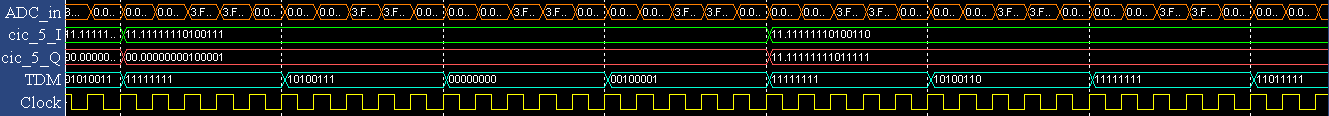


Figure 7.4.2 – Simulation of Time-Division Multiplexed Output

## Section 7.5 – USB Control

The USB controller was written as a transmit-only interface. The data is latched onto the data bus of the Cypress FX2 chip on the rising edge of the 20 MHz clock. The USB control lines on the Cypress FX2 chip are controlled from this module. Three onboard LED’s on the Nexys board are also used to show the status of the transmission and endpoint buffer. The block diagram is shown in Figure 7.5.1.

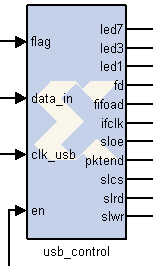


Figure 7.5.1 – USB Control Schematic Symbol

The full system output was simulated, and the results are shown in Figure 7.5.2.

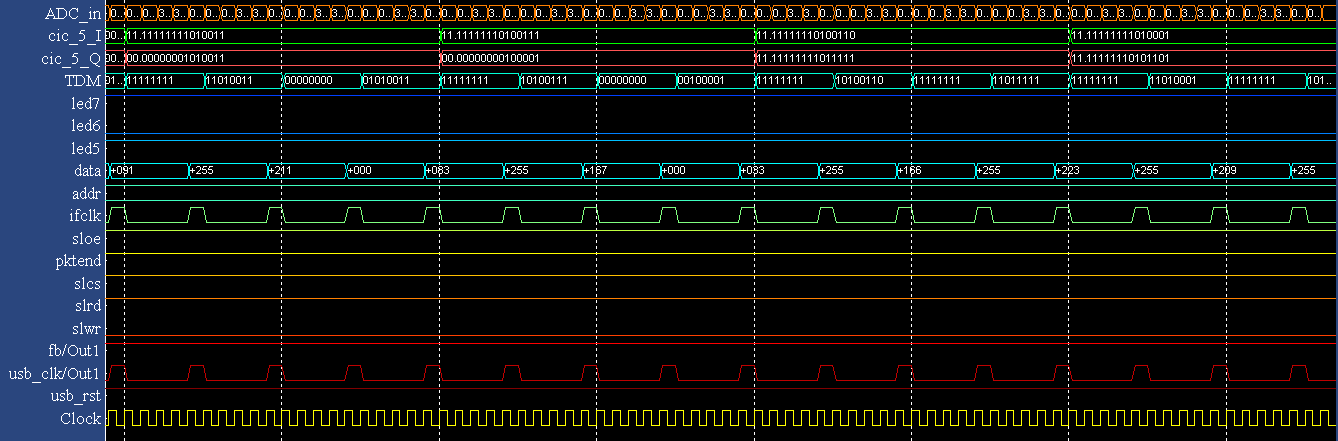


Figure 7.5.2 – USB Controller Simulation

# Section 8.0: Software Installation

This guide assumes you are using Ubuntu 11.04. The software has been tested on Arch Linux, but installation on Arch will not be covered.

## Sectoin 8.1: Dependencies

The SDR Jove software requires GNU Radio. The version used to develop and test the software was GNU Radio version 3.3.1git-144-gf2940603d. Using a version older than this is not recommended. I recommend downloading the latest copy of GNU Radio from the git or svn repositories.

First, install the following packages from the Synaptic:

sudo apt-get build-dep gnuradio

sudo apt-get install python-dev

sudo apt-get install autoconf automake autotools-dev

sudo apt-get install libboost-dev libboost-python-dev libboost-program-options-dev

sudo apt-get install guile guile-dev

Clone the latest GNU Radio version by issuing the following command:

git clone http://gnuradio.org/git/gnuradio.git

Once these are installed, change directories to where GNU Radio was cloned to, and run:

./bootstrap

./configure

make

make check

sudo make install

The firmware for the USB transceiver is already built, but if you would like to recompile it, you will need the Small Device C Compile (SDCC).

sudo apt-get install sdcc

To get a copy of the SDR Jove software, use the following command to clone to the repository:

git clone git://gitorious.org/sdrjove/sdrjove.git

This will copy the directory structure of SDR Jove to your current directory. The directory looks like the following:

/sdrjove

/debug -- Development Tools, fx2load, udev rule file

/fx2\_firmware -- Firmware source files for the Cypress FX2 USB Transceiver

/gui -- Python files for the GUI and GNU Radio

/hardware -- Hardware files, FPGA source, schematics, etc.

/usb\_driver -- Source code for the usb driver.

The debug folder contains the following files:

/debug

/fx2 --Contains python files to load firmware onto the FX2 (from

fx2lib). Run “sudo python setup.py install” in this directory to install.

donbusb.iic --The firmware that comes with the Nexys2 board (provided by

Digilent)

load.py --Loads the SDRJove firmware to the FX2’s RAM

30-sdrjove.rules --Move this file to /etc/udev/rules.d/

It sets up a udev rule to set permissions on the receiver when it is plugged into the USB port.

Vend\_Ax.hex --Stage 2 loader to download new firmware to the FX2 EEPROM.

The fx2\_firmware directory contains the following directories:

/fx2\_firmware

/cystream --Firmware used to stream data from the FX2.

/fx2lib --Library to write firmware for the FX2.

The gui directory contains the following directories and files:

/gui

sdrjove.py --This is the main GUI file. Run this file to start the application.

time\_database.txt --Timestamp database. Any time the software writes data to a file,

the filename and time are appended to this file.

\_\_init\_\_.py --This python this directory is part of a package.

/gnu\_radio --GNU Radio processing blocks. Described below.

/gui\_components --GUI component files. Each frame of the GUI is contained in a separate python file.

The gnu\_radio directory of the gui/ directory contains the GNU Radio python files and the GRC files used to generate most of the code.

/gnu\_radio

\_\_init\_\_.py --Tells python this is part of a package.

live\_mode.py --Python script to use the live mode from the GUI.

live\_mode.grc --GRC file to describe the flow of the live mode

radio\_astronomy.py --Python script to use the radio astronomy mode from the GUI.

radio\_astronomy.grc --GRC file to describe the flow of radio astronomy mode.

record\_mode.py --Python script to use record mode from the GUI.

record\_mode.grc --GRC file to describe record mode.

The usb\_driver directory contains the following files:

/usb\_driver

dataGrabber.c --The main driver file.

dataGrabber\_timing\_test.c --Source for running tests to time the driver.

Makefile --Makefile to compile the driver

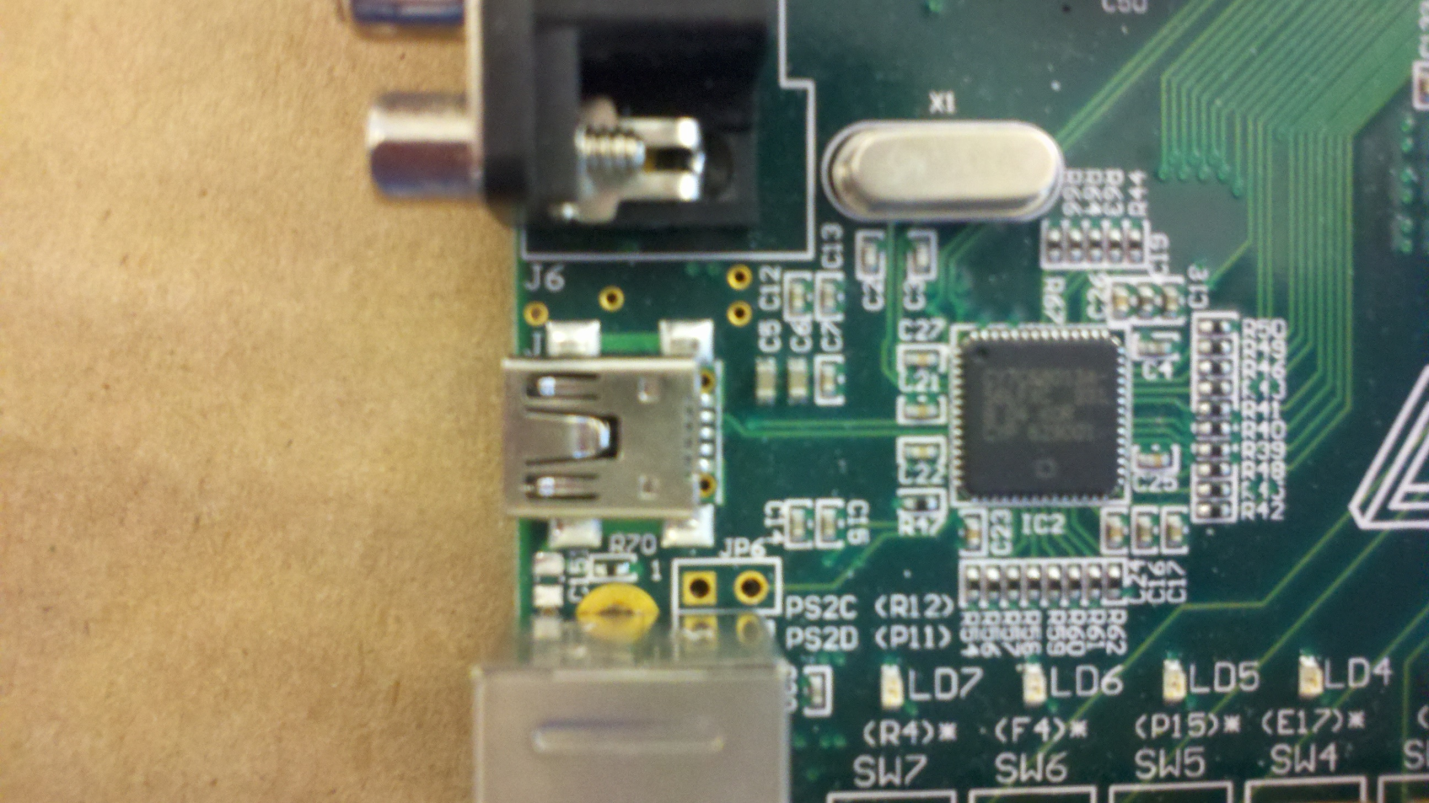
# Section 9.0: Software Usage

When initially setting up the SDRJove hardware, the first step is to download the FPGA bit file to the Nexys2 board. The recommended way to do this is with the Digilent Adept tool. This tool runs on Windows and can be downloaded here:

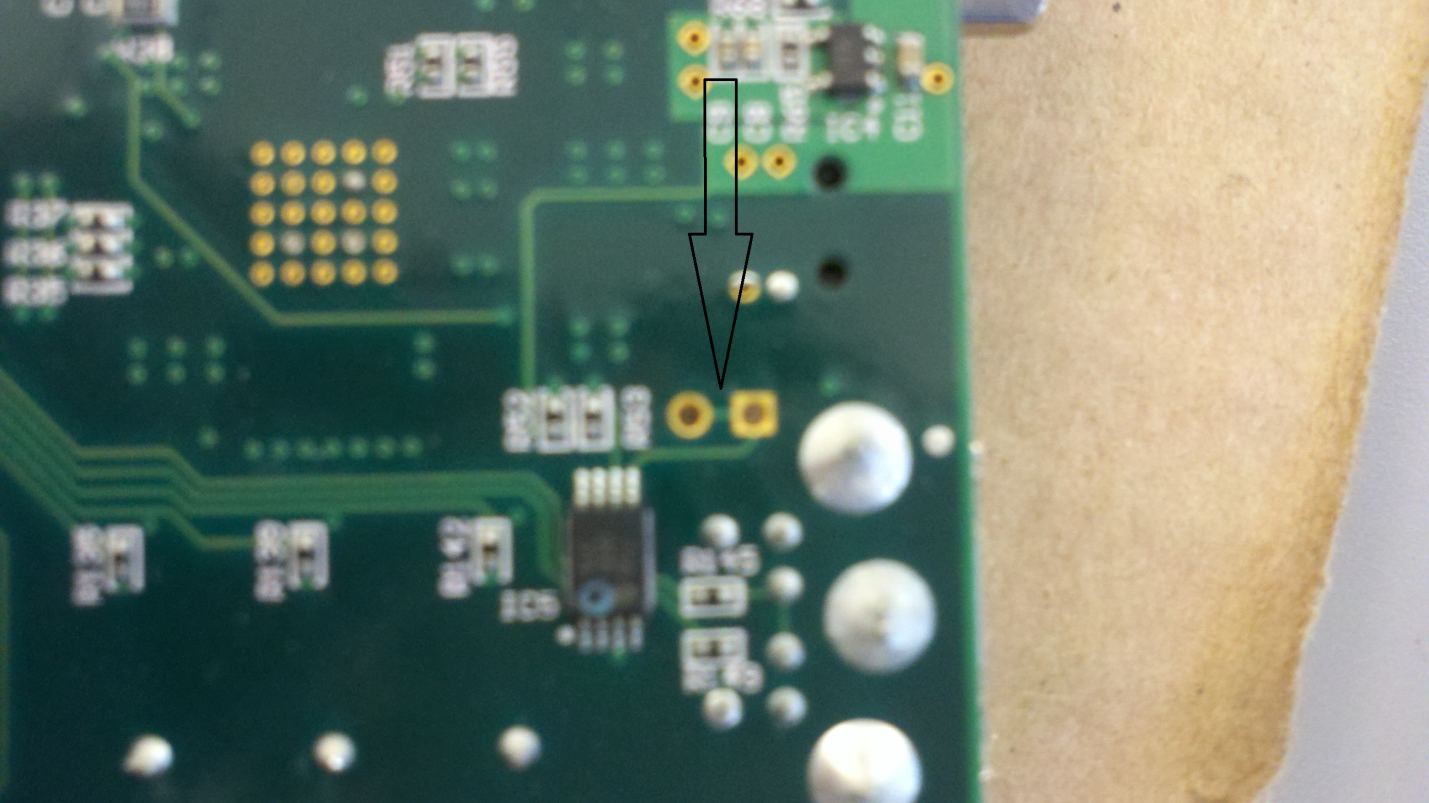
<http://www.digilentinc.com/Products/Detail.cfm?Prod=ADEPT2>

If doing debugging, the trace connecting the FX2 chip to the EEPROM can be cut. To do this, first located JP6 located next to the FX2 USB chip.

THIS IS NOT REQUIRED. ONLY CUT THIS TRACE IF DEBUGGING USB FIRMWARE.



Flip the board over and notice there is a trace connecting both pins of JP6.



Using an X-Acto knive or similar tool, cut this trace. It can be verified that the trace is disconnected by plugging the board into the USB port. The board will not power on. By running the lsusb command, it can be verified that a default Cypress FX2 chip is connected. Once this is verified, disconnect the USB power and solder two header pins to JP6. Using the extra jumper located on the “Jump Storage” jumper, the FX2 EEPROM can be reconnected. Keep this jumper connected, and connect to the USB bus. Again using the lsusb command, it can be verified that the device enumerates as “Digilent.” The device should also power on.

The next thing to do prior to using the software is to load the FX2 firmware onto the onboard FX2 ROM on the Nexys2 board. You will need the Vend\_Ax.hex file in the debug/ folder. You will also need to install the fxload program from the Ubuntu repository. Do this with the following command:

sudo apt-get install fxload

Next, connect the Nexy2 to the USB bus. If you cut the USB EEPROM trace as outlined above, make sure it is connected. Then, run lsusb to determine the bus and device numbers from usbfs. You may need to remove the usbtest kernel module using:

sudo rmmod usbtest

Then issue the following command:

sudo fxload -D /dev/bus/usb/XXX/YYY -s ~/SDR\_JOVE/debug/Vend\_Ax.hex -t fx2lp -I ~/SDR\_JOVE/fx2\_firmware/cystream/fx2\_firmware.ihx -c 0x01

where XXX is the bus number from lsusb and YYY is the device number from lsusb. Note that the paths to Vend\_Ax.hex and fx2\_firmware.ihx assume SDRJove was cloned to your home directory. Note: you may have to run this command twice if you receive an error the first time.

This will write the SDRJove FX2 firmware to the EEPROM so its configuration will be remembered each time the device is plugged in. To temporarily load the new firmware, use the load.py file located in the debug directory. Using this program, firmware can be loaded onto the FX2, but it is erased each time the Nexys board is unplugged. When doing this, you will need to make sure the jumper on JP6 is disconnected prior to running load.py. Also, you may need to remove the usbtest kernel module using:

sudo rmmod usbtest

Next, unplug and reconnect the Nexys board to the USB bus. Run lsusb to make sure a new “Cystream” device is connected, with VID of 0x04b4 and PID of 0x0410.

Next, the USB driver must be compiled. Change directories to the usb\_driver directory and type “make” to compile.

You will also need to copy the udev rule to the correct directory. Change directories into the debug/ directory and issue the following command:

sudo cp 30-sdrjove.rule /etc/udev/rule.d/

Next, make sure all the hardware is connected and change directories to the “gui” directory. Type “./sdrjove.py” to run the application.

The software can be run in one of four modes of operation. Those modes are:

Real Time Mode

Full Record Mode

Radio Astronomy Mode

Data Analysis Mode

In Real Time mode, the user can select to display a waterfall plot or mix a portion of the bandwidth to audio, which can optionally be saved as a wav file or output through the speakers. When the audio is getting mixed down, the 1MHz signal is mixed with a digital 700kHz oscillator to mix the signal to the audio region. This mixed frequency is adjustable in the live mode configuration. The “adjust mixer frequency” slider in the configuration will change the mixer frequency by the specified amount (in hertz). Note: audio output through speakers is experimental, and highly dependent on the audio hardware and software installed.

In Full Record mode, the data stream can be recorded to a file, and a live waterfall plot of the band can be displayed.

In Radio Astronomy mode, the integrated RMS signal strength over the entire band is displayed as a function of time. This stream of data can be optionally be saved to a file. This file is saved as a binary file with 16 bits (2 bytes) allocated to each data point.

In Data Analysis Mode, a data file from full record mode can be run through radio astronomy mode to analyze the power in the signal.

Both full record and live mode require configuration before they can be used. Selecting a button on the toolbar will select that mode.

# Section 10.0: Software Inner Workings

The software can be broken down into three main sections: a libusb driver, the GUI frontend, and the GNU Radio backend. The libusb driver is responsible for getting data off the USB bus and into userland. The GUI front end displays information to the user, and controls the flow of the data. The GNU Radio backend uses the GNU Radio framework to do all the signal processing work.

There is also custom firmware on the USB transceiver chip on the Nexys2 board. This firmware sets the chip up to communicate to the USB host PC in the correct way. The chip on the Nexys2 board is the Cypress FX2LP. The firmware makes heavy use of the FX2Lib project, and the CyStream program. FX2Lib can be found here:

<http://sourceforge.net/projects/fx2lib/>

CyStream can be found here:

<http://allmybrain.com/2009/04/14/fx2-cystream-throughput-test-with-sdcc-and-fx2lib/>

Fx2Lib provides many library functions to access the features of the chip, and CyStream is a data stream application for the chip which provides many interfaces for streaming data to or from a host. This chip gets set up using alternative interface 3. This sets up a single isochronous IN endpoint on endpoint #2. This endpoint transmits 3 packets per microframe, with 1024 bytes per packet (3072 bytes per microframe). Isochronous mode was used because it provides the best environment for streaming data. In isochronous mode, data is sent at a guaranteed rate, but there is no error correction mechanism.

The firmware sets the FX2 up in slave FIFO mode. This means data is clocked into the FX2 from the FPGA. It is set up to clock 8 bits of data in per cycle of the IFCLK line. The FPGA clocks data in at a rate of 20MHz. Since there are 16 bits per data point in both the I and Q components of the output stream, and each stream is sampled at 5MHz, 8 bits of data must enter the FX2 at a rate of 20MHz.

The data is sent in the following format:

I[0:7] Q[0:7] I[8:15] Q[8:15]

The 8 least significant bits of the I component are sent first, followed by the 8 least significant bits of the Q component. Then, the 8 most significant bits of the I component are sent, followed by the 8 most significant bits of the Q component.

Seeing the following application notes on streaming data from an FPGA to the FX2:

Cypress AN4053: Streaming Data Through Isochronous/Bulk Endpoints on EZ-USB FX2™ and EZ-USB FX2LP™

http://www.cypress.com/?docID=20041

Cypress AN61345: Implementing an FX2LP-FPGA Interface

http://www.cypress.com/?docID=27660

The LibUSB driver is responsible for communicating with the Linux kernel to get data off the USB bus and into userland, where it can be sent to GNU Radio for processing. The driver uses the asynchronous LibUSB 1.0 API. It creates 8 transfer request structures and submits them to the kernel. The driver then waits until data is ready, during which a callback function is called. While in this callback function, the driver examines the data in 64-byte chunks. To help determine whether the data received is valid, the FPGA writes four bytes of packet data for every 64 bytes. The packet data consists of the data string “NYAN”. If this string is located every 68 bytes (64 data bytes, 4 packet bytes), the driver knows the previous 64 bytes of data are correct. The driver then arranges the bytes so that the positions of the most and least significant bits are in the correct order, and interleaves the I and Q streams. Finally, it writes the interleaved data to a named pipe. The filename of the named pipe is passed to the driver in a command line argument. It is typically given a random name ending with the extension “.sdrjove”. This random name is generated by the GUI at startup. This file is removed when the GUI is closed.

The driver is loaded automatically by the GUI when I mode is selected which requires new data (Radio Astronomy mode, Real Time mode, and Full Record mode). Once it is loaded, the driver will claim the device, and then make a pause() system call. This system call causes the driver to wait until it receives a signal from the kernel. The driver registers signal handlers for SIGUSR1, SIGUSR2, and SIGINT. When the driver receives a SIGUSR2, it will again call pause(), thus halting the driver from getting new data. When it receives a SIGUSR1, it will continue getting new data. When the driver receives a SIGINT, it will exit gracefully, making sure to cancel any existing transfer requests and freeing any data allocated by malloc().

When the “Start” button is selected in the GUI, an instance of a GNU Radio top block object is created and executed. If live data is being used, the GNU Radio object will read incoming data from the named pipe. GNU Radio deinterleaves the I and Q streams. It then converts each stream of bytes into a stream of floating point numbers, which is further converted into a complex signal. This complex signal is what all the signal processing functions use. The general flow of the GNU Radio signal processing blocks can be seen using the GNU Radio Companion program, provided by GNU Radio. This program uses flow/graph diagrams to generate GNU Radio Python files which can be executed. While some aspects of the Python files have been modified, the general flow of the data can be seen by examining the .grc files inside the /gui/gnu\_radio directory with the GNU Radio Companion. For example, to view the flow diagram for real time mode, change directories to /SDR\_JOVE/gui/gnu\_radio and execute:

gnuradio-companion live\_mode.grc

1. See Application Note 3656 from Maxim Integrated Circuits “Single Transistor Reduces LDO Noise by 46 dB” <http://www.maxim-ic.com/app-notes/index.mvp/id/3656?sisint> [↑](#footnote-ref-1)