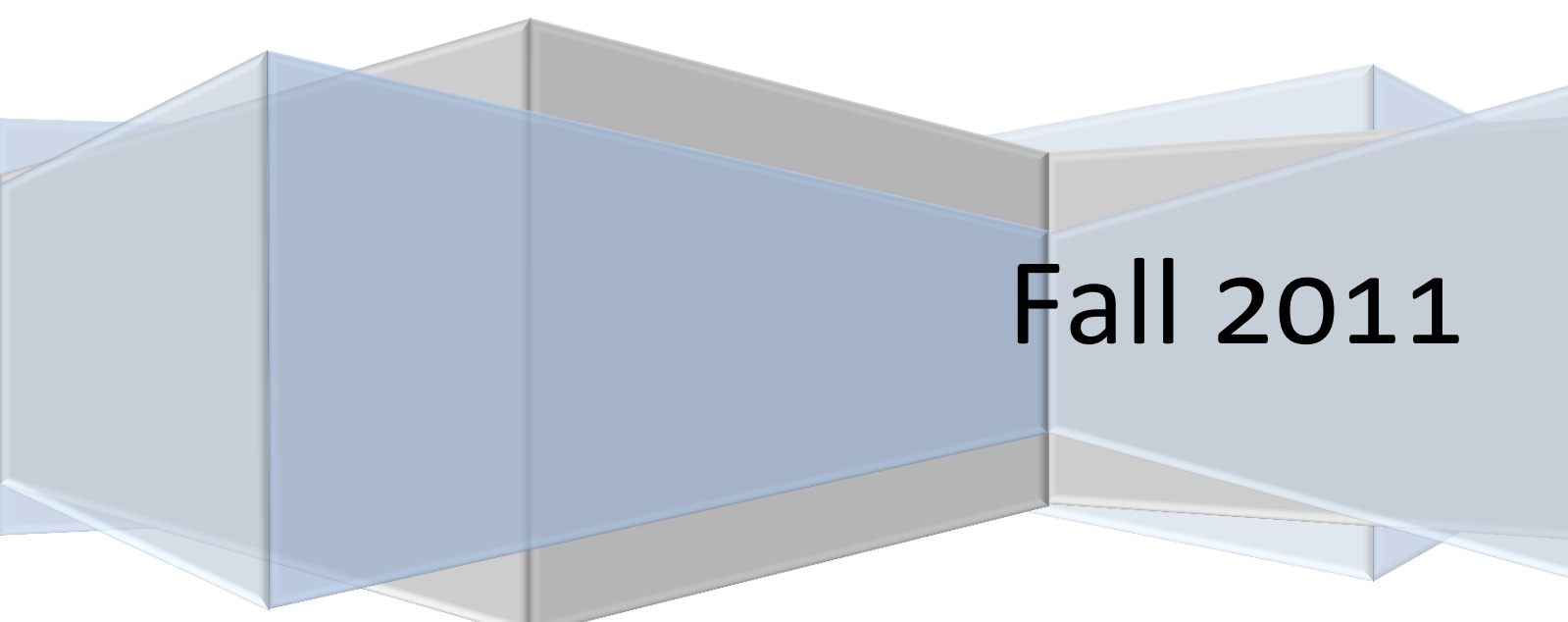


Michigan State University College of Engineering

Remotely Operated Intrusion Alarm

Sponsor – MIT Lincoln Laboratory

**Design Team 6: Peter Deacon, Ryan Hunt, Darci
Koenigsknecht, Chris Leonard, and Chris Oakley**

An abstract graphic at the bottom of the page consisting of several overlapping, semi-transparent, light blue and grey geometric shapes that resemble folded paper or crystalline structures. The shapes are arranged in a way that creates a sense of depth and movement.

Fall 2011

Executive Summary

MIT Lincoln Laboratory has developed an Open Courseware class that uses a simple can antenna to radar imaging, ranging, and Doppler. Lincoln Laboratory has asked this design team to expand on the course and demonstrate that the radar can be implemented as a remote intrusion detection system in real-time. The final design had to monitor a room in real time, be small enough to mount in a room, communicate to a PC over long distances, and be powered, function, and communicate remotely.

Design Team 6 demonstrated that sampling and processing can be done in real time. The program demonstrated the ability to process an ideal signal, but amplitude variation due to improper coordination with the sync signal generated an incorrect output from the output signal. Advanced knowledge of signal processing could alleviate this problem. Lack of memory available prevented the team from implementing the processing on the microcontroller, but would be possible with more available RAM. Communication over Ethernet and an email alert were demonstrated. Therefore, the concept was proven to be viable, but further signal processing experience will be necessary to perfect the design.

Acknowledgements

We would like to give special thanks to Dr. Edward Rothwell, the group's facilitator, for keeping the team on track and granting us access to the EM lab. Thanks also to graduate students Andrew Temme, for general advice and granting us access to equipment in the ERC; and Junyan Tang, for assisting us in photo-fabrication of our PCBs. We would also like to thank Roxanne Peacock for her assistance in ordering parts, as well as lab techs Gregg Mulder and Brian Wright for their assistance in the ECE shop. Finally, we would like to thank the staff at MIT's Lincoln Laboratory for giving us this opportunity, especially Brad Perry and Michael Sheppard for their advice and assistance.

Table of Contents

Executive Summary.....	1
Acknowledgements.....	1
Chapter 1: Introduction and Background	4
Introduction	4
Background	4
Chapter 2: Exploring the Solution Space and Selecting a Specific Approach	6
FAST Diagram	6
Conceptual Design Description	6
Feasibility Matrix.....	8
Proposed Design Solution	8
Budget	9
Schedule and GANTT Chart.....	9
Chapter 3: Technical Description of Work Performed.....	10
Hardware Design.....	10
Ramp and Filter Design	10
PCB Design	10
Hardware Implementation	13
Radar Construction and Tuning	13
Ramp and Filter Construction and Testing.....	14
Software and Interface Design.....	15
I ² S Design.....	15
Ethernet Design.....	15
PC Processing Design	15
Software Implementation.....	16
I ² S Implementation	16
Ethernet Implementation	17
Fourier Transform Implementation	18
Target Identification.....	18
Chapter 4: Test Data and Proof of Functional Design.....	19
Testing of RF components	19
WAV File Processing.....	20

FFT Development and Validation.....	20
Ethernet Testing.....	22
Target Identification and Validation	23
Chapter 5: Summary and Conclusions	27
Appendix 1: Technical Roles, Responsibilities, and Work Accomplished	29
Peter Deacon.....	29
Ryan Hunt.....	30
Darci Koenigsknecht.....	30
Chris Leonard	31
Chris Oakley	31
Appendix 2: References	32
Appendix 3: Technical Attachments	33
Ramp and filter schematic	33
Tuning the antennas	34
Low Pass Filter.....	36
I ² S Bus Overview	36
ADC PCB Design.....	37
Filter, Ramp, and ADC PCB Design	38
Itemized Budget.....	40
Stellaris Microcontroller Code	42
PC C# Code	52
Complex.cs	52
Form1.cs.....	54
fourier.cs	61
SineGen.cs.....	63
Appendix 4: GANTT Charts.....	65
Original GANTT Chart.....	65
Final GANTT Chart.....	66

Chapter 1: Introduction and Background

Introduction

The design project is to develop a remotely operated intrusion alarm, which will alert the user via email and text message that there has been an intrusion. For this project, a radar working in stand-alone mode will be used to detect the intrusion. This information about the intruder will then be sent to the user's cell phone. A less complex version of this project has been used by other students, which was previously developed by MIT Lincoln Laboratory as a classroom project. However, the earlier system has many shortcomings such as a lack of speed, accuracy, and portability. These three parts are imperative to fulfilling the design requirements.

A major piece of this design is to build a 2.4 GHz radar which will detect an intruder in a given area. The radar will calculate the range and speed of the intruder, which will be sent to the user when an intrusion occurs. For the radar, there will need to be an interface constructed to a smart phone, laptop, or a basic cell phone. The antenna is the first component for this design, leaving the data processing section as a major design component. This would eventually pass the data to the laptop for transmission to a cell phone via text message, and send an email to the user. A way to incorporate the processing between the antenna and the laptop is using a TI ADC PCM1808 in combination with TI Stellaris LM3S9D92 microcontroller. By using these two components, most of the interfacing is already incorporated between the two. After that, the information on the Stellaris is ported to the PC via Ethernet, and the PC sends the message to the user. This design is small, power efficient, and will allow an optimal sampling rate in order to obtain real-time data, as well as system portability.

Background

The remotely operated intrusion alarm is a modification to a system previously developed by MIT Lincoln Lab. The previous system was used in a classroom environment as a teaching tool¹.

The intrusion detection system utilizes a frequency modulated continuous-wave radar. The system functions by constantly transmitting a high frequency wave that is varied from 2.26GHz to 2.59GHz. The signal is reflected off of an object, and received at an antenna which is located near the transmitting antenna. This received signal is identical to the transmitted signal, only with a delay in time due to the propagation speed of the electromagnetic wave, and reduced in amplitude. This time-delayed signal is then mixed with the original signal, producing an output signal with a frequency below 20 kHz, which can be easily measured with a microphone input to a computer.

A ramp signal generator is used to vary our signal source over our full 330MHz range. A ramp signal with a frequency of 25 Hz and a peak-to-peak voltage of five volts is given as the modulation input to a voltage controlled oscillator, which produces the transmitted signal. This also generates a synchronization signal which is used to identify periods when data is to be processed for target identification.

¹ <http://ocw.mit.edu/resources/res-ll-003-build-a-small-radar-system-capable-of-sensing-range-doppler-and-synthetic-aperture-radar-imaging-january-iap-2011/index.htm>

The audio range signal created from the mixing of the transmitted and received signals is measured using the right channel of a stereo microphone input on a computer. The synchronization signal created from the ramp signal generator is measured using the left channel of the same stereo microphone input. This stereo audio signal is recorded for some period of time by a user, and later processed using a MATLAB program. The result of the output from this MATLAB program is to be interpreted by the user to extract any useful information.

The intrusion alarm radar was developed as a teaching tool, which can be assembled by students, and allow them to quickly design the signal processing required to detect objects. This has several shortcomings which need to be addressed, including speed, accuracy and portability, which make the radar designed by Lincoln Lab impractical for use in an intrusion detection system.

Speed is crucial to the proper operation of any intrusion monitoring system. The system developed by Lincoln Lab requires a significant amount of user interaction, a result in which is very slow operation. Currently, the audio signal, which comes from this system, is recorded for a duration of time by the user. This signal is then processed by MATLAB, which returns an image displaying range to an object versus time. Due to the very slow operation of this system, an intruder cannot be detected in real time. Instead, the intrusion can only be detected after the event has passed. For any sensor to be used in a secure environment, this response time is not acceptable.

In addition to the slow response time of this system, accuracy is another area of potential failure. Currently, the image MATLAB returns of the processed signal must be interpreted by a user. This image can be difficult for a user to interpret. The image is color coded to convey the amplitude of the signal returned by the detected object. Should the received signal be of sufficiently low amplitude, it may be difficult to distinguish from colors of other received signals with similar amplitude. As a result, should this low-amplitude signal be shown to change range over time, a user can misinterpret the data and fail to detect an intruder.

Portability is a significant factor in the design of any intrusion detecting system. The system designed by Lincoln Lab requires a computer near the radar platform to receive and process the audio signal generated by the system. This is, quite often, not a practical requirement in an environment to be secured. In this configuration, a power source must be available for the computer to operate for any significant duration. This requirement increases the difficulty of moving the platform to a new location, should it be discovered it was not originally placed in an optimal location. The system should be easy to adapt to changing conditions.

For any system to be used to detect an intruder, the system must respond quickly enough to detect when an object moves. The system should be easy to deploy in a multitude of locations, as well as easy to change the desired location of the sensor. Finally, the system should be able to operate with autonomy. Unfortunately, the intrusion detection system as currently designed is not capable of fulfilling any of these requirements.

Chapter 2: Exploring the Solution Space and Selecting a Specific Approach

FAST Diagram

FAST DIAGRAM:

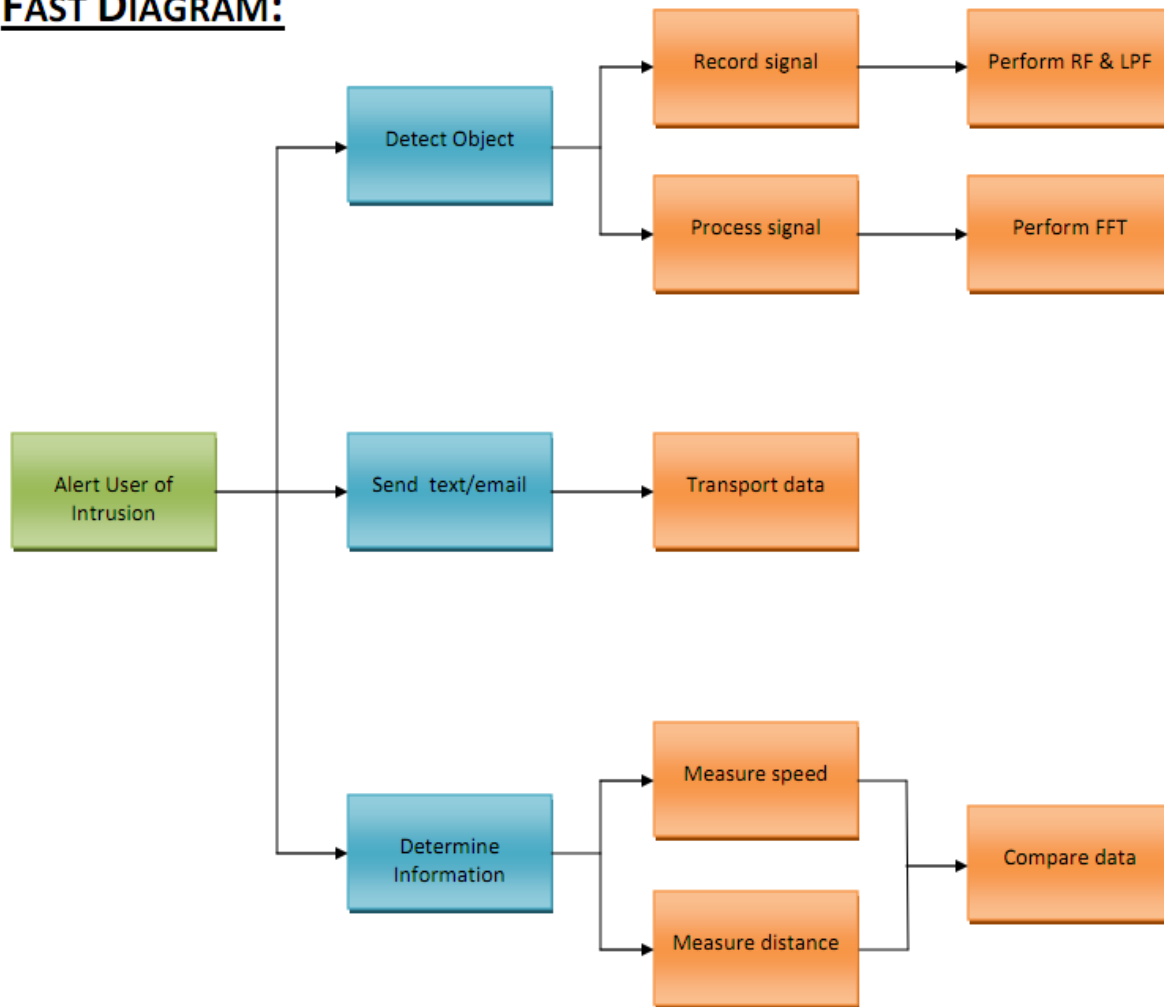


Figure 1 - FAST Diagram

Conceptual Design Description

This system can be designed in one of two ways, as shown in Figure 2. One way to design the system is to do the processing on a PC and the other way is to do the processing on an on board chip. The basic system includes a radar antenna, a means of processing, and a way to set off an alarm or alert. The alert could be an e-mail, text message, or a combination of the two. The radar stage of the system which has been provided will not change unless an antenna with different dimensions is used. If a

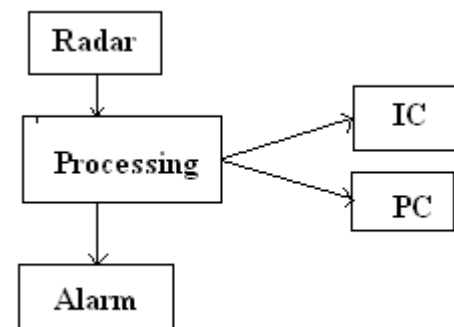


Figure 2 - Processing Options

different sized can is used, minor calculations will be required in order to achieve the same signal.

If a PC is chosen to do the processing, MATLAB and LabView can be used to do the processing and C# can be used to send the alert. In order to send the signal from the radar system to the PC an analog to digital converter and a digital to analog converter will be required. More ICs are required to send information to do this. The advantages to using a PC for processing are simpler coding, large amounts of memory, and an interface to monitor code. The disadvantages to using a PC to do the processing are added noise due to the added complexity with the digital to analog converter and added ICs to couple with different interfaces. The time needed to send a full signal to the PC will limit the time that the system is able to scan and take in information.

The signal from the RF/antenna stage can be sent via a wired connection or a wireless connection. Wireless connections include Wi-Fi and Bluetooth. Advantages of wireless communication include mobility, being able to deploy the system anywhere, and not being limited to the length of a cord. A disadvantage to using a wireless communication system is that the radar will not be able to sweep while communication with the PC due to frequency interferences. The rate at which the signals can be transmitted over a wireless connection is limited to 256kbps. A faster transmission time is required in order to spend more time detecting and less time transmitting signals to process. Another major disadvantage to a wireless connection is the systems vulnerability to an intruder jamming the system.

A wired connection to the PC allows faster transmission time which can be done using a serial input, USB, or an Ethernet connection. Using a serial communication system is simple to interface, more secure than wireless, and easy to troubleshoot. Serial communication only allows a transmission rate of up to 200kbps, which again is too slow for the size of signal that will be sent.

Connecting to the PC with a USB interface allows faster transmission speeds. USB 2.0 enables transmission speeds up to 144Mbps, which is sufficient to transmit the signal to the PC. The main disadvantage to a USB connection is the short length of cable available, limiting the range of the processing unit. If more than one USB is connected together via a repeater, the signal to noise ratio is increased as well as the cost of the system for extra controllers and connectors. The addition of controllers to drive the extra connectors makes the system more complex.

An Ethernet connection between the RF stage and the processing unit allows transmission speeds between 10 and 100 Mbps. These speeds are sufficient to transmit a signal to the PC for processing while the radar continues to sweep. Ethernet is inexpensive and provides as much distance between the RF stage and the PC as needed. Ethernet cables have been tested up to 100 yards with zero noise added to a signal. A disadvantage to using Ethernet is that an additional controller is required to interface the Ethernet as well as additional code to connect to the system.

Processing of the signals can also be done on an on board microchip. Processing of signals on a microchip eliminates the transmission time required to send the signals to a PC. The digital to analog converter will also be removed from the system because the signal will not be sent to the PC. There are many microchips available to do the necessary processing. The Microchip-PIC32 and TI-Stellaris 9000 series are two microprocessors that have everything needed to implement into the system.

Microchip PIC32 has an on board Ethernet interface which eliminates an extra controller and extra code to interface the Ethernet. In addition to the Ethernet interface, the controller has a USB, I²C, and UART interfaces. The disadvantage of using the PIC32 is that it does not have an I²S interface needed to communicate to the analog to digital converter. The Stellaris 9000 series controllers have all of the interfaces needed to communicate with the system as well as an I²S interface. Having an I²S interface on eliminates the need for a dsPIC to couple between I²C and I²S interfaces. Without the need for a dsPIC the cost of the system decreases as well as the systems complexity.

Feasibility Matrix

We used a feasibility matrix to help us make our design decisions, seen in Figure 3. The matrix shows that the best options are Ethernet for our communication, and the Stellaris microcontroller for the processing.

Criteria	Importance	Possible Solutions					
		Wireless	Serial	USB	Ethernet	PIC32	Stellaris
Speed	5	1	1	9	9	3	9
Size	3					3	9
Communication	4	3	3	9	9		
Remote	4	9	9	1	9		
Totals		53	53	85	117	24	72

Figure 3 - Feasibility Matrix

Proposed Design Solution

The following is a description of the design being implemented. This design can be broken into several components and subcomponents, each with a specialized function to allow for successful communication of an intrusion message. The high level block diagram of the complete system is shown in Figure 4.

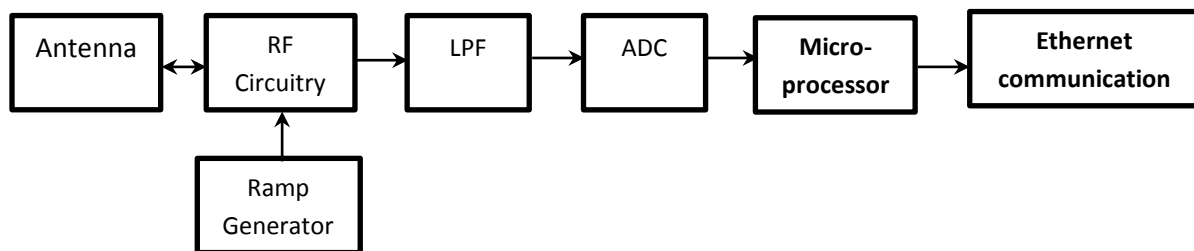


Figure 4 - Complete System

The RF circuitry is comprised of a group of subcomponents. The block diagram of the RF system is shown in Figure 5.

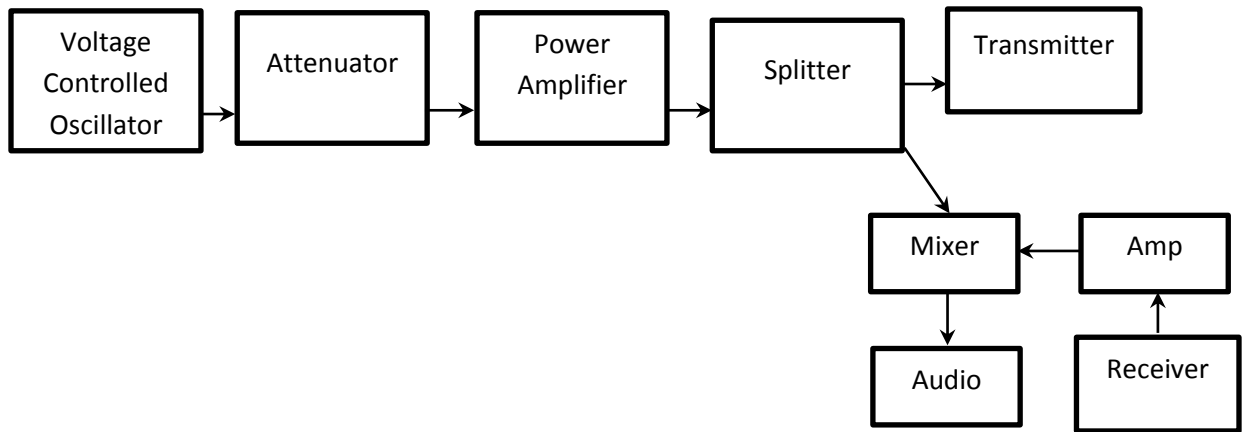


Figure 5 - RF Block Diagram

In order to analyze the differences between successive analog signals from the RF component, the mixed analog signal will be passed through a low pass filter into the PCM1808 analog to digital converter (ADC). The PCM1808 is a 24 bit ADC; 24 bit resolution is desirable because the output digital signal will allow for high sensitivity to motion. Filtered digital signal will then be passed using an I2S data link into a Stellaris LM3S9D92 microprocessor. The microprocessor will be used to perform a Fourier Transform on the input to the microprocessor. The output from the Fourier transform will be an array of frequencies and amplitudes. Successive Fourier transforms will be compared to determine if there is a change in amplitude at each frequency. These changes will correspond to motion in the room being scanned. If motion is detected a flag on the microprocessor will be set to trigger an alert. The alert as well as the data from the successive Fourier transforms will be sent to a PC using the Ethernet port on the Stellaris microprocessor.

Once the computer receives an alert package, it creates an email containing the speed and range of the intruder, as well as a timestamp of when the intrusion occurred. This email is then sent to a person or list of persons that should be alerted, as designated by the user.

Budget

The requirements of this project will be fulfilled with less than \$500. The RF stage of the system can be completed with \$235.55. The antenna design will be completed with \$53.48. The rest of the circuit, including the analog and power requirements, will be completed with \$71.60. An additional cost of \$11.72 has been added to the budget due to technical complications. An analog to digital converter break-out board needed to be designed and fabricated which added another \$13.08 to the budget. With these five sections of the system, the project comes to a grand total of \$384.76.

Schedule and GANTT Chart

The original project plan included additional time to address unanticipated problems with development at each stage. This did not account for the significant problems actually encountered throughout development and testing.

During testing of the analog circuitry, the maximum voltage limit of the MAX414 op-amp was exceeded, destroying the component. Testing of the radar components was delayed by a week while new components were shipped. Additional time delays were encountered during testing when inconsistent behavior was encountered between PCs used for testing.

Additional delays were encountered during development of the interface between the analog-to-digital converter and the Stellaris micro-controller. Assembly of a PCB to adapt this device for easy connection between a signal source and the micro-controller ended in failure. A different approach was taken to eliminate this problem, but a delay of several days occurred while additional parts were on order. Other problems occurred during software development of both the analog-to-digital interface, as well as network communication between the micro-controller and a PC, resulting in additional delays.

See Appendix 4 for the GANTT chart.

Chapter 3: Technical Description of Work Performed

Hardware Design

There were two parts to the design of this project: the radar components, and the processing components. The radar was provided by the sponsor, and as such, required no design. It consists of antennas, RF stage, the ramp generator and the filter. Several options for design of the processing stage were explored. These can be viewed in Chapter 2.

Ramp and Filter Design

The ramp generator and filter schematic were provided. The ramp generator consisted of a single IC. Aside from DC blocking caps and voltage dividers, it had a ramp output and a sync output. The ramp output was the actual ramp signal, while the sync output was a square wave of the same frequency as the ramp. The frequency of both of these was controlled by an attached potentiometer.

PCB Design

In addition, a custom PCB was designed by the group, which included MIT's design with the ADC used in the project.

A circuit was built initially on a proto-board using a schematic from the MIT courseware website and a decision had been made to take the proto-board and put it on a PCB. The circuit that had been built on the proto-board consisted of the ramp generator, filter, and analog to digital converter (ADC). These are all of the main parts that were incorporated in the PCB design. In order to reduce noise, surface mount components were chosen for the PCB.

In order to use surface mount parts as opposed to through-hole parts on the PCB more parts had to be ordered for the board. A Bill of Materials (BOM) was provided from our sponsor of through-hole parts for the filter and the ramp generator. The ADC was added to the PCB board being designed, which added to the number of parts required. The ADC that was used was the PCM1808 from Texas Instruments. Using the data sheet, it is easy to see that in order to implement the PCM1808 there are

only four extra capacitors required. While finding the surface mount equivalent parts from the BOM from MIT to use for the PCB, it was evident that many of the discrete components had very specific specifications and requirements which made the search for parts more meticulous.

Dip Trace was chosen as the PCB creation software for the project. Dip Trace was chosen because it is a single software package that is capable of producing each step of a PCB build. Dip Trace was used to create a schematic of the entire electrical system, with the exception of the RF components. This schematic can be found in Appendix 3, Figure 1. The schematic contains custom components created by members of the team. Dip Trace contains libraries of components that have been compiled from popular electronics manufactures and from lists of standard parts. Some of the parts used in the final design for did not exist in the Dip Trace libraries; therefore it was necessary to create components.

To create components members of the team went through the data sheets for each part and gathered information about the size and spacing of each pin requiring electrical connection. The team members then used to Dip Trace Pattern Editor to create pads slightly larger than each pin. Figure 6 shows a custom pad created for the SMA connectors used to connect the receiver antenna to the PCB.

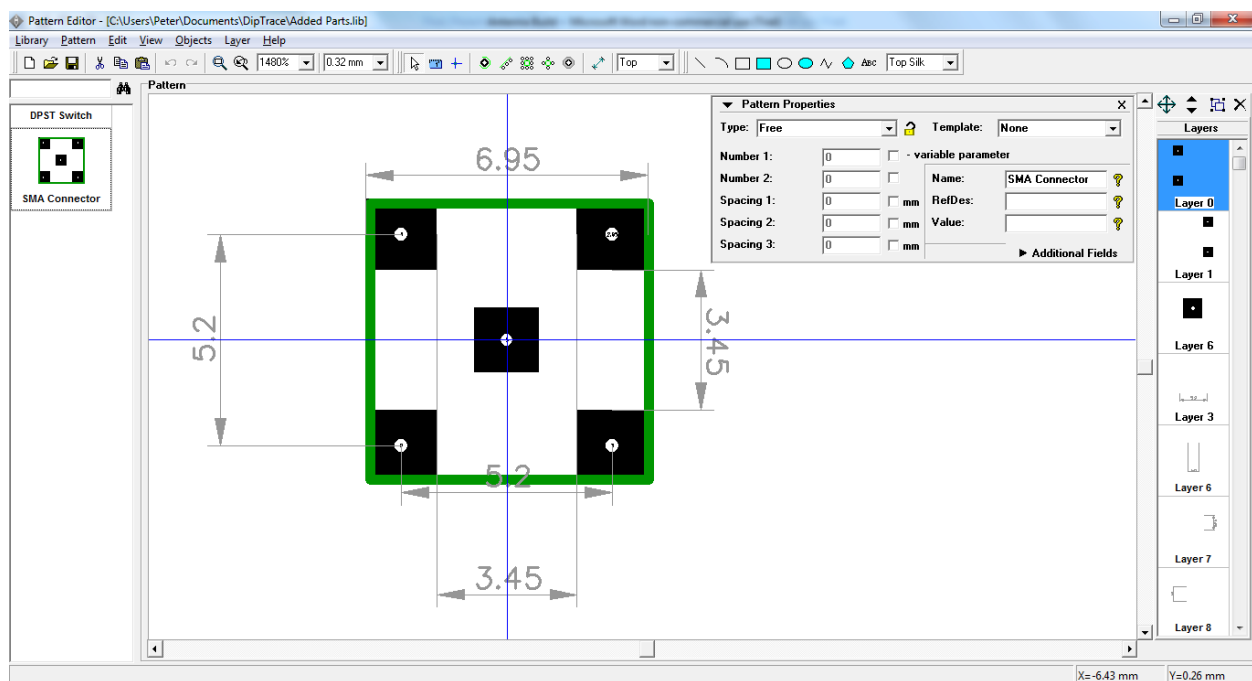


Figure 6 - Custom pad for SMA connectors

After creating custom pads for each component, components must be created. Dip Trace's Component Editor allows users to create symbols the symbols used to represent components on a schematic. The pad attached to the component represents the electrical connections for the component and is what gets displayed on the PCB. Design Team 6 created custom components for each element in the schematic. An example element is shown in Figure 7, which shows the SMA Connector with the custom pad attached.

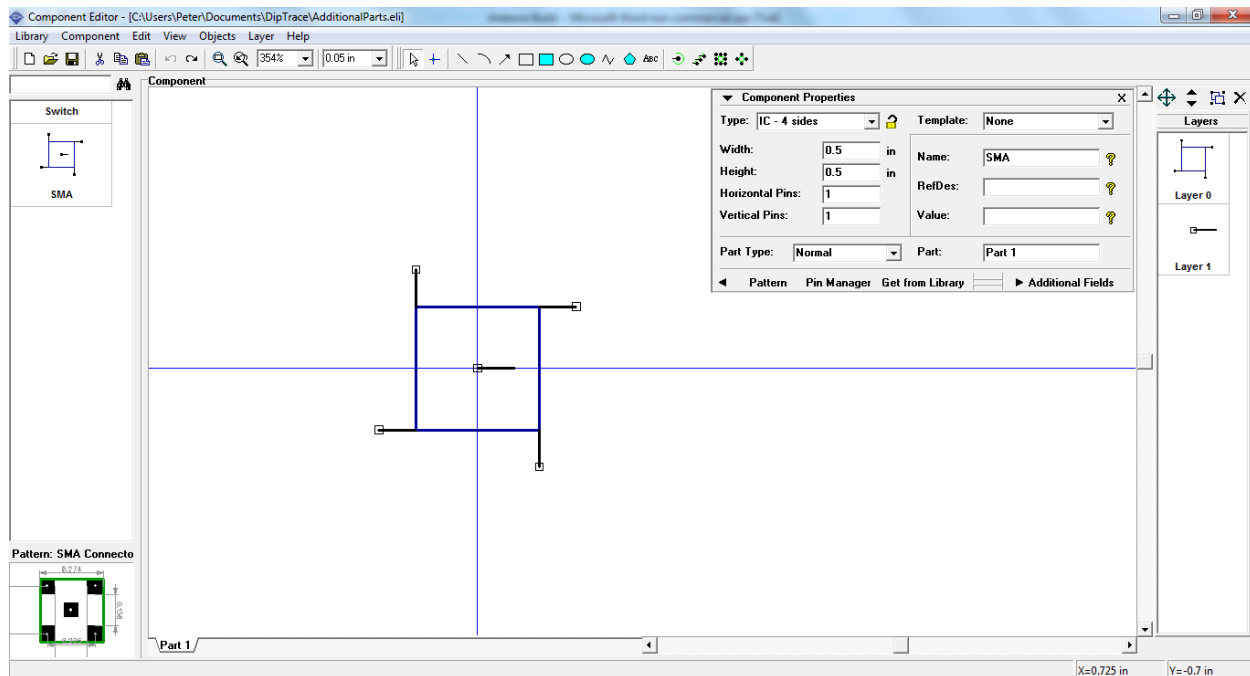


Figure 7 - SMA with custom pad

Once all the components were created they were placed in the schematic and wired together.

Once all of the individual components of the PCB had custom surface mount pads and patterns, the schematic needed to be laid out using DipTrace. To do this, the library of custom patterns made had to be added to the project workspace. The schematic for the filter and the ramp generator can be found on the MIT website in the courseware. When creating the schematic in DipTrace it is important to make sure everything has ground and power and that all of the connections to pads are made. A common mistake is adding a part to the schematic and not making the connection to the pad. One way to avoid this is to mouse over each connection after the schematic has been built and the net to which the connection belongs to will be highlighted along with its connections. It is also recommended to open datasheets of components with multiple connections when laying out a PCB in DipTrace.

After the schematic had been laid out, the PCB Layout software has to be opened in DipTrace. From there the schematic can be imported into the program just by opening the file with "PCB Layout". This part of the program offers features that automatically lay out the board and run traces. The problem with using its auto-run features is that the program does not always find a layout that makes all of the necessary connections. Under "Route-Setup" width parameters for the traces can be set. Using the auto-placement feature, DipTrace will run your schematic through all of the various ways to assemble the PCB and then chooses the optimal layout. When the program runs auto-placement, it takes each individual part and places it on the board, taking into account its connections with other components. Using auto-placement does not make the traces for the user; it configures all of the parts in a tight setting on the board and then allows the user to manually move some components around if necessary. When all of the components are in place, it is time to use auto router. Auto router will lay traces down, making as many connections as possible without moving any of the components. In many cases,

DipTrace will not be able to make every single connection in which case it will notify you with a blue line connecting the two pads that do not have a direct connection. At this point it is possible to move components, vias, and traces around and make manual connections.

After the board is laid out, all of the traces are in place and connections are made, a verification test needs to be run. The verification process checks the nets of the circuit then checks the design rules of the circuit. Once there are no errors in either the design rules or the nets of the circuit, the PCB is complete. Design team six's finished PCB board can be seen below in Figure 8.

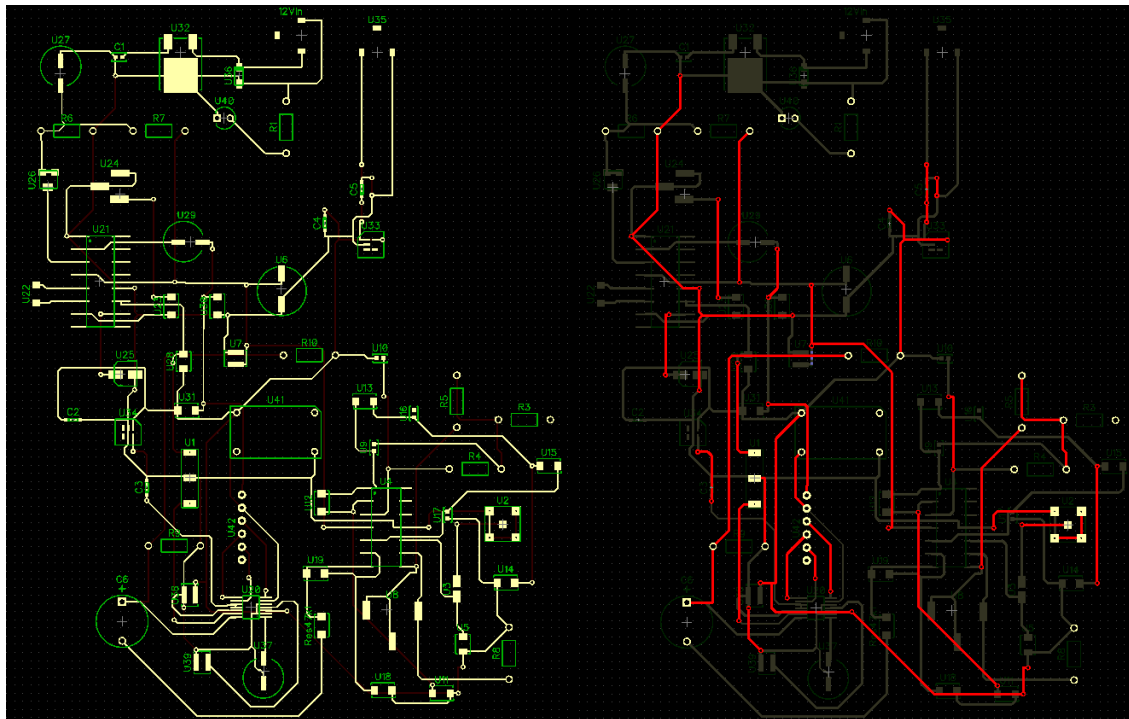


Figure 8 - PCB top (left) and bottom (right)

Hardware Implementation

Radar Construction and Tuning

The radar in used in this project requires two 2.4GHz antennas. The antennas were built based off of instructions for the design of an FMCW radar provided by MIT Lincoln Laboratory. These antennas are monopole cylindrical wave guide antennas. To build the antennas Design Team 6 calculated the appropriate length of the monopole, as well as the distance the monopole should be placed in front of the back of the waveguide. Monopole antennas are typically built to be a quarter of the wavelength of the desired signal long. The wavelength, λ , of a signal is represented by the equation 1 where c is the speed of light and f is the frequency.

$$\text{Equation 1: } \lambda = c / f$$

For this application, the desired frequency is 2.4 GHz this means that the monopole should be about 1.3 inches long. Within a waveguide the wavelength of a signal is longer than in free space. To calculate λ_g ,

the wavelength within a guide, Design Team 6 used Equation 2, where D is the diameter of the waveguide.

$$\text{Equation 2: } \lambda_g = \lambda / \sqrt{1 - \left(\frac{\lambda}{1.705 * D}\right)^2}$$

Using the Equation 2 the wavelength inside the waveguide was calculated to be 1.8”.

Design Team 6 drilled a hole into each wave guide 1.8” from the back wall, and then soldered a 1.3” monopole onto a SMA connector. The monopole was then mounted inside the wave guide. After mounting the antennas had to be tuned.

Once the antennas had been put together they needed to be tuned. The monopoles were designed to each be one-quarter of a wavelength which was calculated from the operation frequency. The target frequency is 2.4 GHz; however, the frequency is actually being swept from 2.4GHz to 2.5GHz. In order to tune the antennas, the original installed monopoles were longer than one-quarter of a wavelength of the operating frequency. This way they could be tuned by trimming them. Installing wires without leaving extra length could result in undershooting the target frequency due to loading from the cans or other electromagnetic interferences.

The Agilent 85032 network analyzer was chosen to test and tune the antennas because it had the ability to operate in the desired frequency range. The following parameters were the goals during testing: a reflection coefficient greater than -15.7 dB, a peak gain greater than 7 dBi, and a signal to noise ration greater than 2. All of these conditions were met over a frequency sweep from 2.4GHz to 2.5GHz.

Each antenna had to be tuned individually. One antenna was hooked up to the network analyzer at a time and placed in the anechoic chamber. Once placed there, the team ran multiple frequency sweeps, each time clipping off small pieces of the monopole antenna. This method was used in order to avoid overshooting the target range and starting over with a new monopole. The length of the wire is directly related to the resonant frequency of the antenna, which is why care had to be taken when shortening each monopole.

“Tuning the antennas” in Appendix 3 shows that the resonant frequency of both the monopole antennas is at about 2.4GHz and is below -15dB for most of the frequency sweep between 2.4GHz and 2.5GHz.

“Tuning the antennas” in Appendix 3 also shows that the signal to noise ratio is below two for the frequency sweep between 2.4GHz and 2.5GHz in both antennas.

Ramp and Filter Construction and Testing

The ramp generator and filter were built according to the provided schematic. When they were finished, the ramp generator output was viewed on a scope, and a potentiometer was used to adjust the ramp to a 25 Hz wave. The filter’s corner frequency was recorded at 16.4 kHz, which is acceptably close to the design corner of 15 kHz. Also, there was a -17.7 dB drop at one decade, which is close to the target -20 dB/decade (See “Low Pass Filter” in Appendix 3).

Software and Interface Design

The software on the microcontroller had to do three things: sample data from the ADC, detect an intruder, and send range and speed over Ethernet to a PC. The software on the PC simply had to take the range and speed data and automatically send an email to a list of users.

I²S Design

Design choices had to be made for I²S communication. The I²S bus can be configured so that the transmitter is the master and the receiver is the slave, the receiver as the master and the transmitter as the slave, or both transmitter and receiver as slaves and an external controller as the master (See “I²S Bus Overview” in Appendix 3). Following some investigation in the Stellaris manual, it was discovered that the controller, the receiver, could not output the master clock frequency if the internal PLL was used. As such, the receiver could not be used as the master easily. Instead, the transmitter, the ADC, was chosen as the master device, and the receiver was chosen to be configured as the slave. There was also an option to use either default, left-justified, or right-justified I²S format. The ADC was incapable of writing in right-justified format, so the default format was chosen out of simplicity.

Ethernet Design

Ethernet was chosen as the media for communicating from the microcontroller to the PC. The raw API functions for lightweight IP (lwIP) make programming the controller as a client simple and small in terms of memory.

PC Processing Design

Software was developed to be used during the testing phase to confirm output from each component is valid, as well as to display real time results of the intrusion detecting radar. To facilitate rapid development of a user interface, Microsoft C# was the language of choice. This provides easy to use TCP interfaces, as well as simple user interface components such as graphing tools, which allow for easy confirmation of proper system operation.

The PC side software functions by creating a TCP server and listening for connections on port 2000. When a connection is created, a new thread is spawned to handle the data transmitted from the microcontroller. The received data stream is stored as a byte array in little-endian format. This byte array is processed to extract the contained data points stored as unsigned integers. This time-domain data is processed using a fast Fourier transform algorithm, and stored with a time-stamp, for later comparison.

To facilitate data processing and storage, a RadarSample class was created. The constructor of this class requires two pieces of data: an array of time-domain samples, and a DateTime object containing the time when the sample was received by the PC. Upon object instantiation, the time-domain data is processed into its frequency-domain representation. This frequency-domain data is stored for later comparison, as well as displayed in a graph on the user interface.

The frequency-domain samples are compared by subtracting a previous sample from the latest sample. This will, in theory, eliminate noise common to both samples, and result in two points corresponding to the distance some object traveled over the comparison period. Noise in the system can result from both erroneous signals due to coupling between the transmitter and receiver, as well as other random noise

in the environment. The signals which would indicate a moving object will appear as both a maximum value spike representing the object's location at the time of the most recent sample, as well as a minimum value spike representing the object's location at the time of the previous sample against which the comparison is being executed.

After comparison of data samples, if an intruder is detected, information related to that intruder is displayed on the user interface. The speed of the intruder, as well as the distance to the intruder at the time of the most recent sample, are displayed in text boxes.

Software Implementation

I²S Implementation

For this application, the ADC had to be set as the master device in the I²S module. This is done by setting pins MD0 and MD1 high, generating a bit clock of 256f_s. In addition, FMT was set low, so that a normal-style I²S was sent, as opposed to left-justified.

The Stellaris microcontroller had to be programmed to communicate with the ADC over I²S. The microcontroller had to be enabled as a slave device, powered by an external oscillator. An interrupt handler was written to handle the dual channel input and place them into two separate left and right channel buffers. The interrupt handler also was designed so that the program would only record the audio channel when the sync signal from the ramp generator was high:

The I²S peripheral and GPIOs are set and enabled by the SysCtlPeripheralEnable, GPIOPinConfigure, and GPIOPinTypeI2S functions:

```
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOD);

SysCtlPeripheralEnable(SYSCTL_PERIPH_I2S0);
SysCtlPeripheralReset(SYSCTL_PERIPH_I2S0);

GPIOPinConfigure(GPIO_PA2_I2S0RXSD);
GPIOPinConfigure(GPIO_PD1_I2S0RXWS);
GPIOPinConfigure(GPIO_PD0_I2S0RXSCK);
GPIOPinConfigure(GPIO_PA3_I2S0RXMCLK);
```

Originally, the I²S bus was sending information, and the I²S Rx module registers were set correctly, but the FIFO was not being filled. The cause of this was that the GPIOPinTypeI2S function was not added, the pins were multiplexed to their correct signal, but the pins were not enabled as inputs.

```
GPIOPinTypeI2S(GPIO_PORTA_BASE, GPIO_PIN_2 | GPIO_PIN_3);
GPIOPinTypeI2S(GPIO_PORTD_BASE, GPIO_PIN_0 | GPIO_PIN_1);
```

The I²S Receive module is set to normal I²S format, dual mode stereo, salve configuration, 24-bit sample size with a 32 bit word size. The sample vs. word size is set as such because the ADC has a 24 bit

resolution, but it sends 32 bits of I²S data with each sample; the two most significant bytes are always 0x00. The module is configured by using the I2SRxConfigSet function:

```
I2SRxConfigSet(I2S0_BASE, I2S_CONFIG_FORMAT_I2S |  
               I2S_CONFIG_MODE_DUAL |  
               I2S_CONFIG_CLK_SLAVE |  
               I2S_CONFIG_SAMPLE_SIZE_24 |  
               I2S_CONFIG_WIRE_SIZE_32);
```

The FIFO is set to trigger a service request interrupt when it fills up with 4 samples, using the I2SRxFIFOLimitSet function. The custom interrupt handler is then registered with the I²S module, and all interrupts are enabled using the I2SIntRegister, I2SIntEnable, and IntEnable functions:

```
I2SIntRegister(I2S0_BASE, My_I2S_Int);  
I2SIntEnable(I2S0_BASE, I2S_INT_RXERR | I2S_INT_RXREQ);  
IntEnable(INT_I2S0);
```

Finally, the I²S receive module is enabled using I2SRxEnable, and the program is ready to run with all computations occurring at the interrupt level:

```
I2SRxEnable(I2S0_BASE);  
  
while(1)  
{  
}
```

The full code for the I²S can be seen in the final program code, in Appendix 3.

Ethernet Implementation

Porting via Ethernet is a very in depth task. This type of implementation requires one to know how to connect between two remote objects via networking. To accomplish this task a file was found that implemented an Ethernet connection between the Stellaris and a web server on the laptop. After further investigation of how a web server implementation works and research through the E2E community for TI online, a TCP connection was established. The TCP connection allows the Stellaris to make a connection from the Ethernet to the laptop. This is done by first creating a new socket and then binding to the port that is connected to the laptop, once the bind is complete the code tries to connect to the address and port entered for the laptop. Once the connection is established it will call a callback function which will then send the data across the Ethernet to an interface developed on the laptop side for converting little-endian to big-endian so that it is converted for readability. When all data has been sent, the connection is then closed on both ends of the connection.

Problems that were encountered during Ethernet porting were the ability to find examples of how to develop a network connection between the Stellaris and the laptop. These problems were overcome by reading threads posted on the TI E2E Community, reading over data sheets for given functions, go over the web server connection code, and posting for help on the E2E Community page. Once code was established and ran, another encounter was data not sending over the Ethernet connection that was made. After research on a particular function, tcp_write, it was found that a parameter for the function was incorrect, and the problem was solved immediately.

Fourier Transform Implementation

Spectral analysis of the analog signal output from the intrusion detection radar is required to determine range to an intruder, as well as the speed of said intruder. In order to accomplish this, a Fourier transform algorithm was developed for use on a PC, which could then be ported to a micro-controller for a fully embedded system. The Fourier transform is a mathematical operation which decomposes a signal into its frequency components.

A discrete Fourier transform (DFT) method was developed first. This is to be used to validate output of a Fast Fourier Transform (FFT) to be developed. The discrete Fourier transform is defined as:

$$X_k = \sum_{n=0}^{N-1} x_n e^{-i2\pi k \frac{n}{N}}, k = 0, \dots, N-1$$

The DFT, while accurate, is a rather slow operation. For a signal with N samples, the DFT requires $O(N^2)$ operations to calculate the magnitude of each frequency component. As the number of samples increases linearly, the number of operations required to calculate the frequency spectrum increases rapidly. This requirement leads to an unacceptably long time to calculate the frequency components of each sample set, resulting in a system which is not capable of operating in a real-time fashion.

To combat this time constraint, a fast Fourier transform (FFT) method was developed using the common Cooley-Tukey algorithm. The Cooley-Tukey algorithm computes the Fourier transform of a given by first splitting the time-domain signal of size N into two smaller arrays of size $\frac{N}{2}$. These new arrays are then processed in a recursive manner to reduce the number of computations required for each data point. The Cooley-Tukey algorithm is defined as:

$$X_k = \sum_{m=0}^{\frac{N}{2}-1} x_{2m} e^{-\frac{2i\pi}{N}mk} + e^{-\frac{2i\pi}{N}k} \sum_{m=0}^{\frac{N}{2}-1} x_{2m+1} e^{-\frac{2i\pi}{N}mk}$$

The Cooley-Tukey algorithm only requires $O(N \log N)$ operations to compute the Fourier transform, resulting in a significant speed advantage over the DFT. However, the Cooley-Tukey algorithm as applied requires the sample size N of the given signal to be some power of 2. While the sample set is not always of this size, additional samples with a value of zero can easily be added to the end of the sample data to reach a sample set to conform to this constraint. Due to the speed with which the Cooley-Tukey algorithm computes the Fourier transform, the sample size constraint still results in a computation time which will allow for very fast operation of the intrusion detection system.

Target Identification

The Fourier transform algorithm used will yield an array with each element containing the magnitude of some frequency. In order to make use of this information, there needs to be some method to correlate an array index to a particular frequency. For an array index “i” containing, sample rate f_s and number of time-domain samples N, the corresponding frequency can be calculated as:

$$f_i = i * f_s$$

The distance to an object can easily be calculated using this information. Given a distance “d” to a target, signal bandwidth “B”, and ramp time “ t_s ”, the frequency corresponding to a target can be calculated as follows:

$$f_t = \frac{2dB}{c*t_s}$$

Given a target identified at array index “i”, the distance to that target can be calculated by combining the two above equations. The resulting equation is:

$$d = \frac{ict_s f_s}{2B}$$

After subtracting frequency-domain samples, two data points need to be found. The first data point is the index of the maximum value. This corresponds to the location of the target in the most recent sample. The second data point is the index of the minimum value. This corresponds to the location of the target in the sample against which the comparison is being executed. By subtracting the index of the minimum value from the index of the maximum value, and applying the distance calculation from above, the distance over which the target traveled can be calculated. The sign of this value indicates direction of travel, either towards or away from the radar. A positive value indicates the target is moving away from the radar, while a negative value indicates the target is moving towards the radar.

By storing a time-stamp in epoch format with each sample, the time, in seconds, over which the comparison takes place can be easily calculated. Combining the time between compared samples with the calculated distance traveled, target speed can be computed. These results can be displayed for a user to observe, or sent in a message to a remote device for use in an alarm system.

Chapter 4: Test Data and Proof of Functional Design

Testing of RF components

During the testing phase of the radar system assembly, the radio-frequency components appeared to have failed. There was no apparent output which could be measured either through the use of the microphone input to a PC, or with an oscilloscope.

The RF components were removed from the system, and each individual component verified with the use of a spectrum analyzer. The voltage controlled oscillator was first connected directly to the spectrum analyzer, with the output frequency held constant at 2.26GHz. The output power measured was approximately 3dBm.

Each amplifier was then connected, using the VCO as a signal source, and measured with the spectrum analyzer. To protect the input to the spectrum analyzer, a 30dB attenuator was placed between the amplifier and the spectrum analyzer input. The output of these components was verified to be correct.

Finally, the signal splitter, mixer and attenuator were each connected, keeping the attenuator attached to the spectrum analyzer input. These components each had the correct output power and frequency.

No problem could be found with any individual component. These components were then reassembled and connected back to the analog circuitry. Measurements with both an oscilloscope and a PC verified the system to be once again functional.

WAV File Processing

A class was developed to process data stored in a WAV file, which could be used during testing of methods to detect a target. WAV files have a very specific format, which is simple to process using basic file manipulation techniques.

In a WAV file, the first 44 bytes contain data related to the file itself. This includes the sample rate, number of bits per sample and the number of channels. The rest of the data contains the audio data. For a stereo WAV file, there are only two channels: left and right. For any given sample, the left channel data occurs first, with the right channel data immediately following.

The class to process a WAV file stores the sample frequency, and number of bits per sample. Two functions were developed. The first function fetches the next sample in the file. The second function will return an array of audio samples where the synchronization signal is high.

FFT Development and Validation

Throughout the development of the fast Fourier transform, operation of the transform needed verification. To accomplish this verification, a sinusoid was generated in software, which could then be processed by the FFT algorithm, and the results displayed for a user to verify. By utilizing a sinusoid with a known frequency, verification is trivial. This sinusoid output was graphed to verify its functionality, which can be seen in Figure 9.

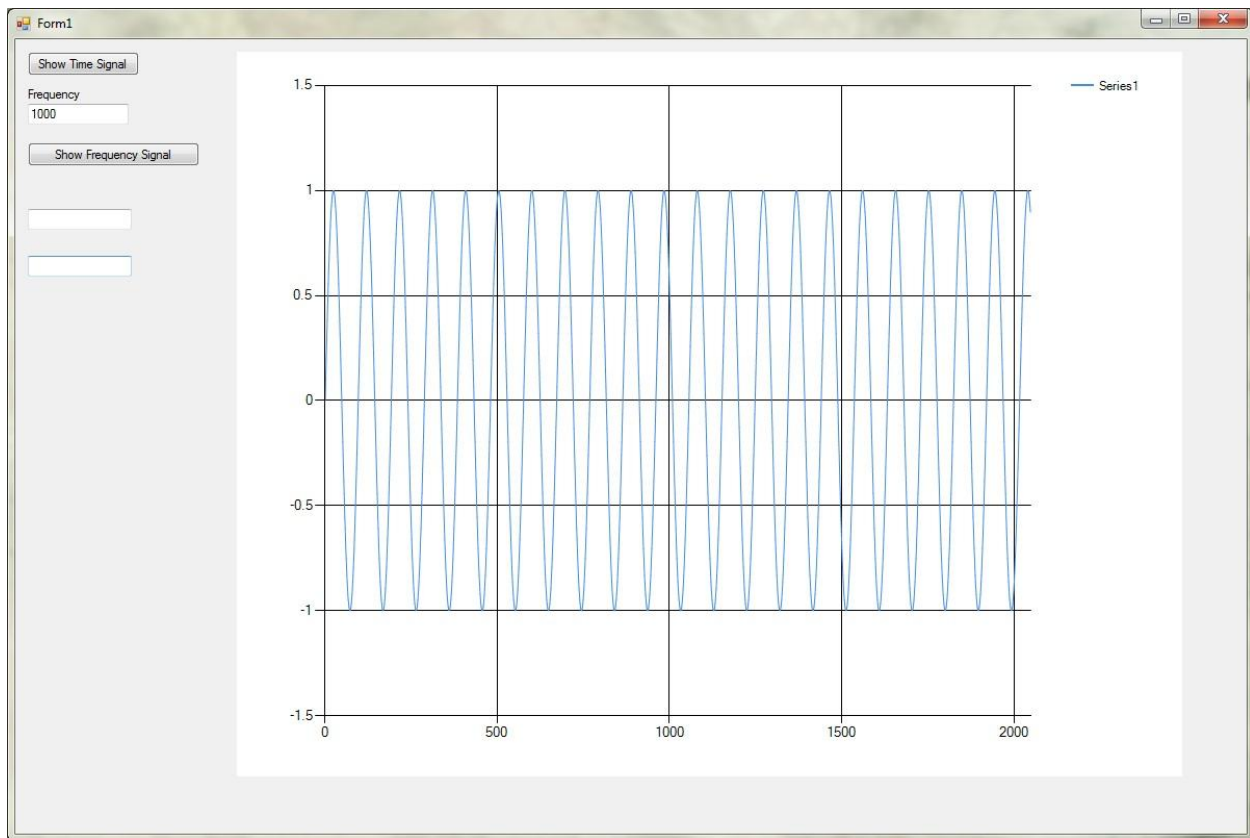


Figure 9 - Generated sinusoid

After verifying the sinusoid was functioning properly, a discrete Fourier transform was developed. The DFT was developed based on the definition of the discrete Fourier transform. This operation, while accurate, was slow. The fundamental frequency of the sinusoid used for the DFT was verified by locating the array index of the maximum value, and correlating this index to its corresponding frequency.

The next step was to develop a fast Fourier transform, comparing the output of this to the output from the DFT. To comply with the restriction of using a sample size that is some power of two, the sinusoid generated was restricted to 2048 data points. The output was then verified using the index of the maximum data point, and calculating the corresponding frequency. The output of this operation is shown in Figure 10.

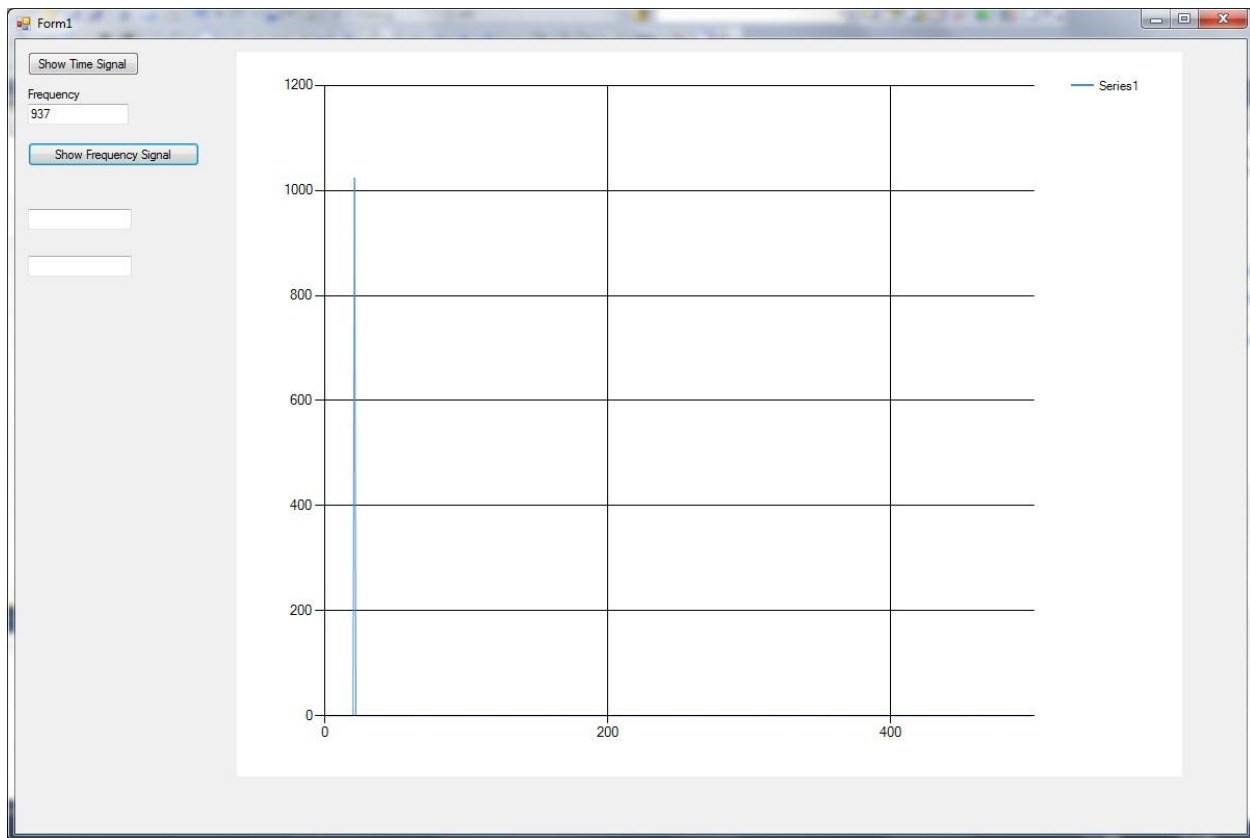


Figure 10 - Fourier transform

Ethernet Testing

In order to test the Ethernet portion of the project there was need for two laptops, a network router due to the inability to get into the EGR network for testing purposes, and the Stellaris connected between the two. The laptop running the code for the Stellaris was connected by the debugger on the Stellaris. The Stellaris was connected via Ethernet to one port on the network router, and another port was leaving the network router and connecting to the other laptop which was to receive the data from the Stellaris. When a network router was found the laptop running the program for the Stellaris was configured to be address 192.168.1.2, and the laptop that was going to receive the data was configured to 192.168.1.1. By doing this, the IP address for the laptop getting the data was put into the programming that was sent to the Stellaris so that it could call out to the network for that specific address to make a connection with. When the program was working, but not sending data, WireShark was downloaded to ensure proper connection between the Stellaris and the laptop receiving data. Knowing there was a connection ensured that it was the Stellaris code that was the problem. Once the code was resolved an interface established on the laptop receiving data would display the data send over the Stellaris exactly as it was supposed to look. A dummy array with the numbers 1-10 in order was sent from the Stellaris to the PC, and printed out in the exact form of:

Sent from Stellaris: `unsigned int dummy[10] = {1,2,3,4,5,6,7,8,9,10}`

```
tcp_write(pcb, dummy, sizeof(dummy), 1);
```

Printed from receiving laptop: [0]:1

[1]:2

[2]:3

[3]:4

[4]:5

[5]:6

[6]:7

[7]:8

[8]:9

[9]:10

A C# application was developed for validating data sent across a network from the micro-controller to a PC. This application creates a new thread to listen for TCP connections on port 2000. Data transmitted from the micro-controller is received as an array of bytes, which is then combined and reordered based on the parameters of the original data. These values are then displayed in a text box for verification by the user.

Target Identification and Validation

Validation of the proposed technique to identify a target was carried out in two steps. The first step was to simulate an ideal signal using a signal source swept from over a range of frequencies. The second step was to record the analog output of the radar, stored as a WAV file on a PC, and processed. This data was recorded using a sampling rate of 44.1 kHz, and a resolution of 16 bits.

This data was utilized by taking each sample, and checking the data corresponding to the left channel. While this left channel sample had a value less than zero, the data was discarded and the next sample was taken. While the left channel had a value greater than zero, the right channel data was added to an array. Once the left channel value returned to below zero, the right channel data was processed using the developed FFT.

Testing with a simulated ideal signal proved our concept to be viable. Testing was conducted using a signal source swept from 200Hz to 2 kHz, over a period of approximately 60 seconds. This simulated a target walking away from the radar, beginning at a distance of two meters, and ending at a distance of approximately 20 meters. This resulted in both a maximum value peak and a minimum value valley, which can easily be used for computation of the target speed and range. An example of this output can be seen in Figure 11.

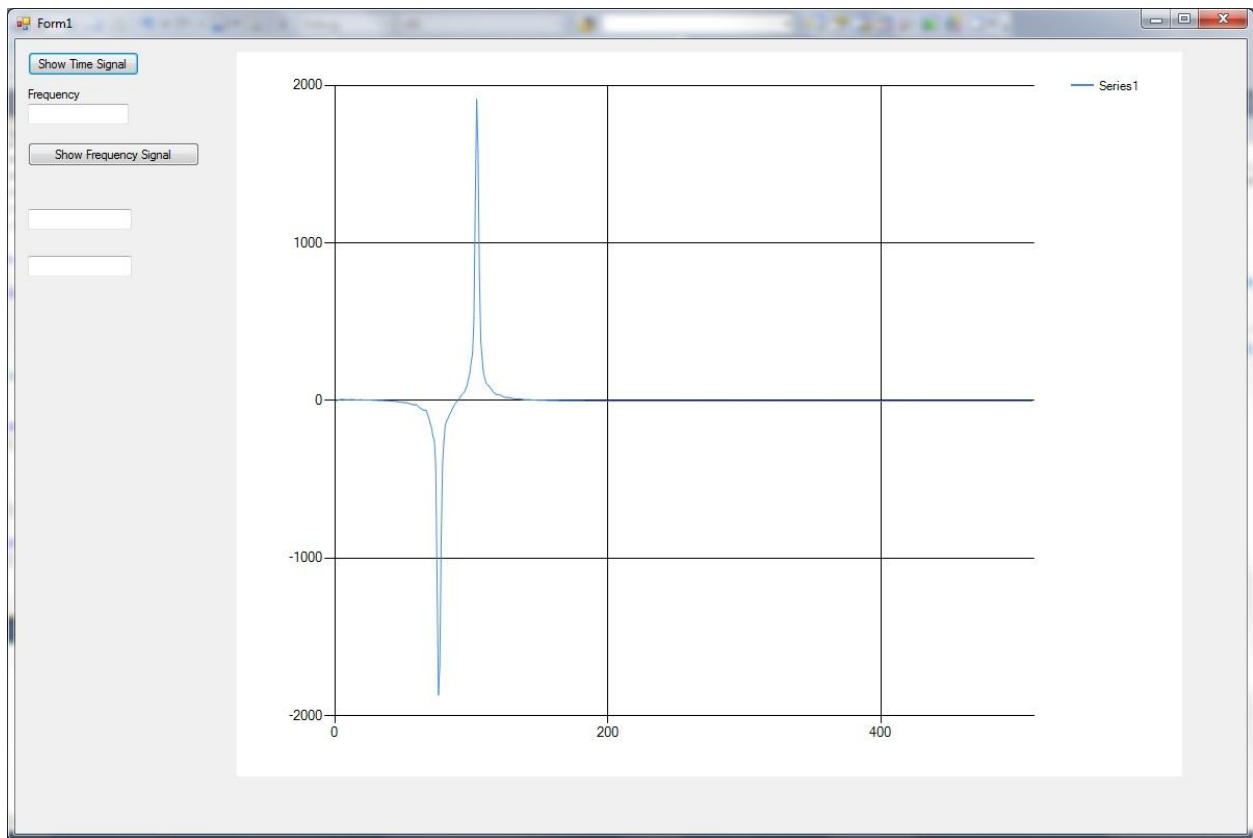


Figure 11 - FFT difference signal

The next step consisted of using the same method with actual output from the radar. During this process, a significant amount of variation was found to exist between each sample, leading to a signal which was rather noisy after processing. Throughout the development process, no method was discovered which resulted in enough noise reduction to allow for automated target identification.

The first attempt at identification of a moving target was to take the Fourier transform of one sample from the radar, and subtract it from a sample taken five seconds later. This resulted in an excessively noisy signal, in which no target could be accurately picked out. An example of this failure can be seen in Figure 12.

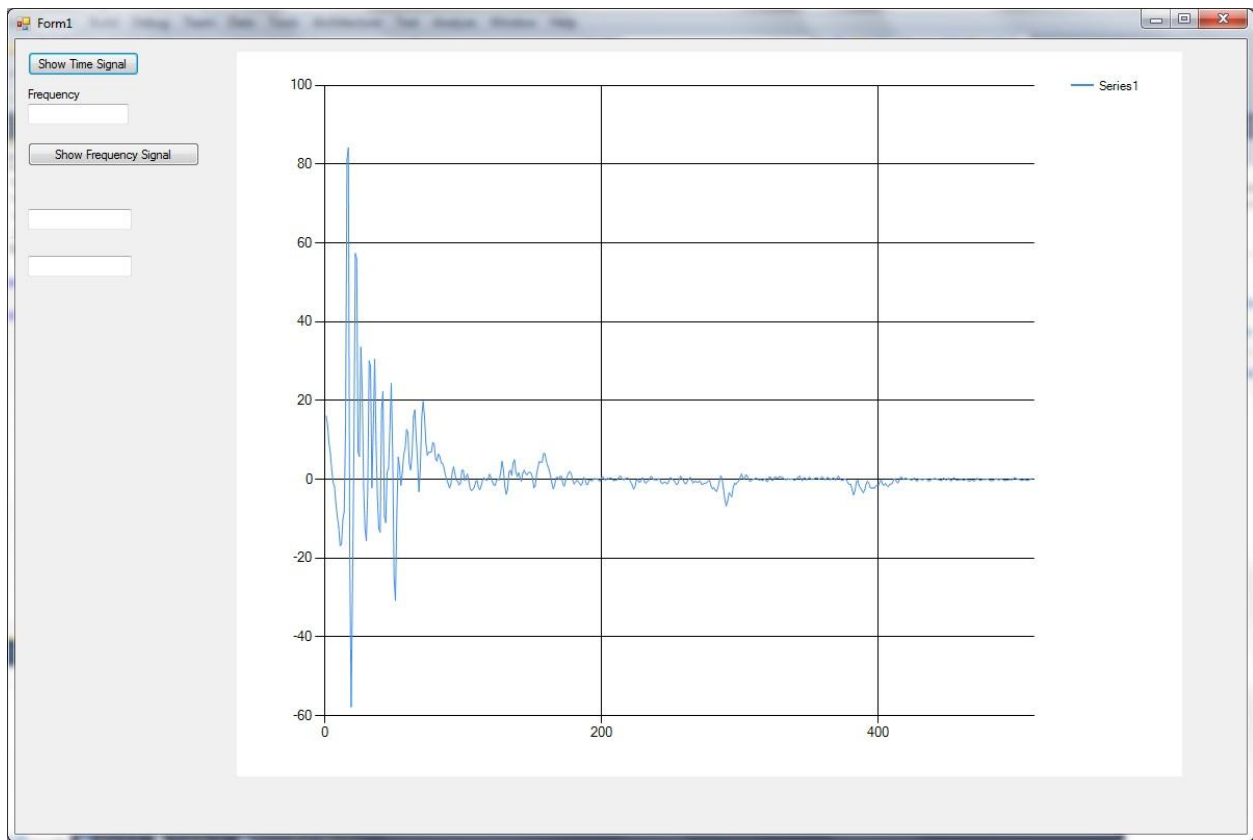


Figure 12 - FFT of sampled data

An example of this variation in signal amplitude can be seen in Figure 13.

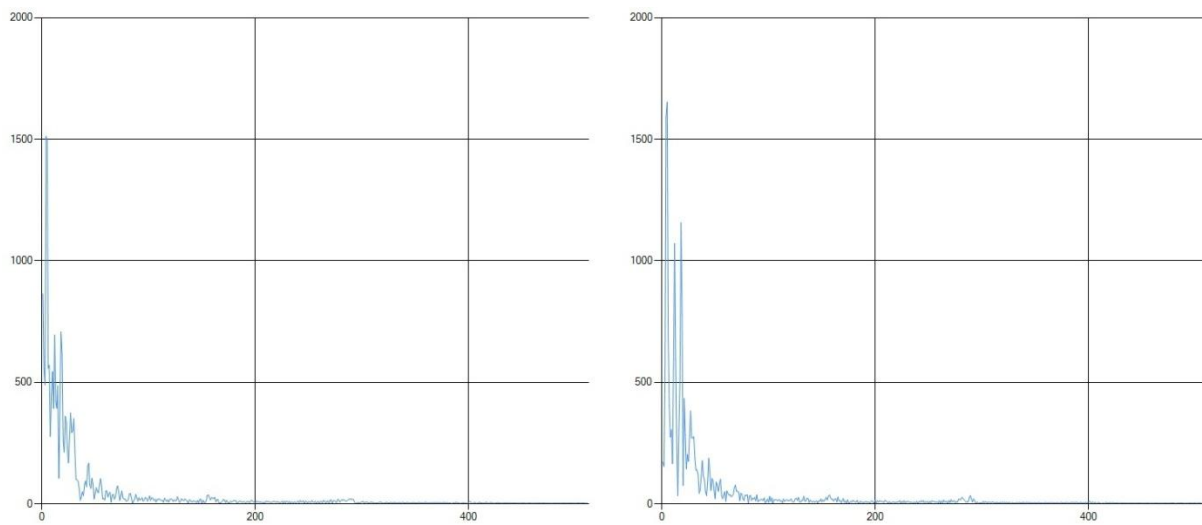


Figure 13 - Signal amplitude variation

After this, an attempt was made to filter these signals. This was accomplished by taking the average of every two data points in each sample. This also did not produce usable results. The output of this attempted averaging can be seen in Figure 14.

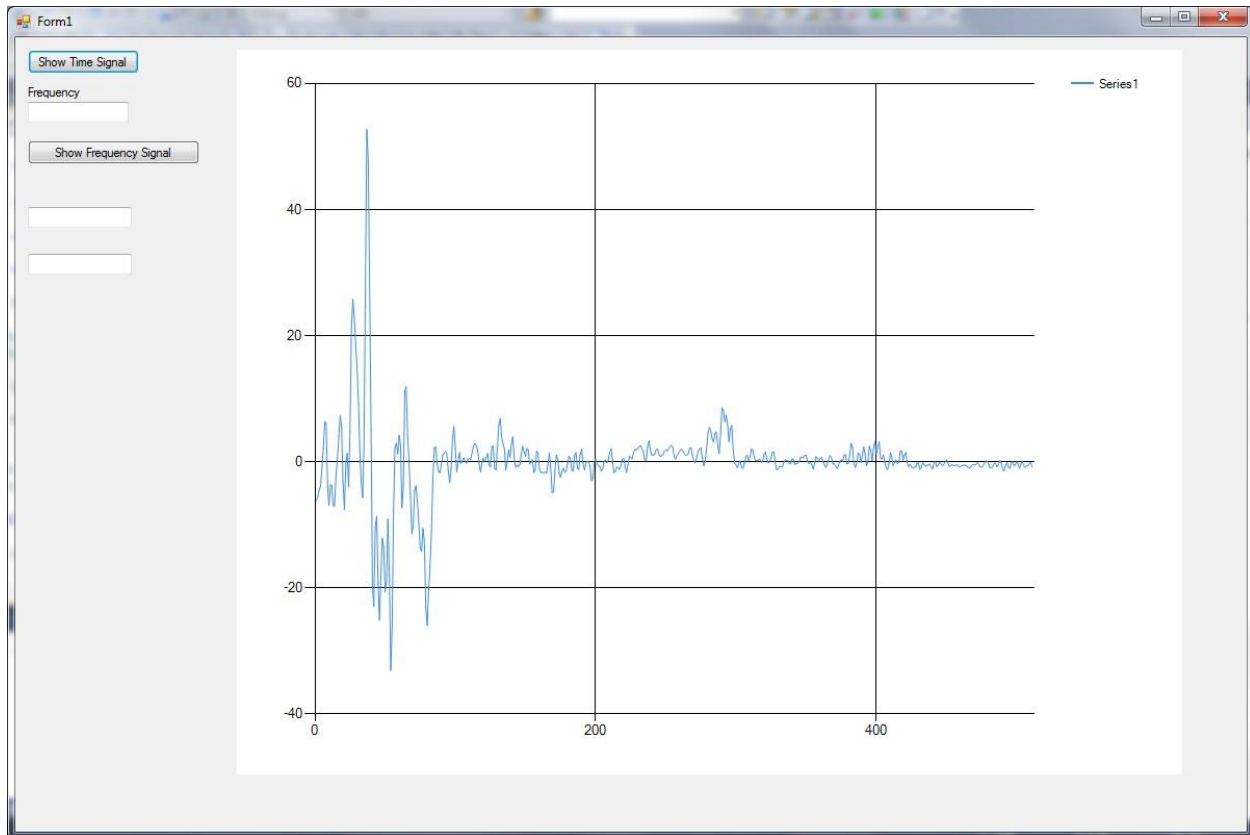


Figure 14 - Failed difference with two points averaged

The final attempt at filtering of the comparison signal was taking blocks of N samples, and calculating their average. Unfortunately, this also resulted in failure. The result of this attempt can be seen in Figure 15.

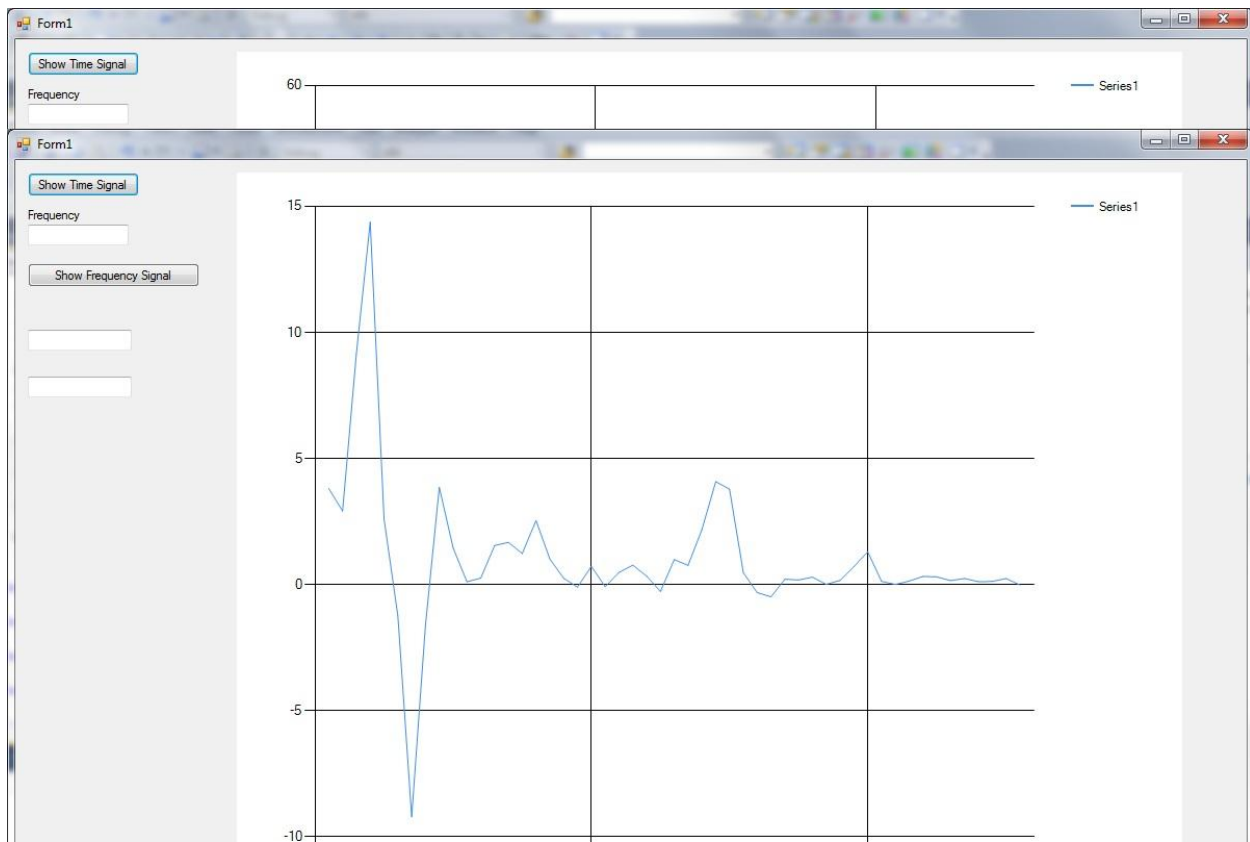


Figure 15 - Difference signal, multiple averages

Chapter 5: Summary and Conclusions

MIT Lincoln Laboratory developed a classroom project that is three weeks long in which students develop a radar to detect objects in a given area. Design Team 6 was tasked with expanding this design. The earlier system had many shortcomings such as lack of speed, accuracy, and portability. MIT Lincoln Laboratory has tasked DT6 with developing a remotely operated intrusion alarm, which would make the shortcomings become three key parts to fulfilling the design requirements. DT6 had the following four main tasks to accomplish with this design project.

- Speed – be able to monitor real-time movement and alert a user within seconds.
- Size – small enough to mount discretely in home or office.
- Remote Operation – have on board power, and remote communication between the radar and PC.
- Reliable Communication – have reliable detection, low amount of false triggers, and communicate with devices around the world.

Task 1 was to sample data fast enough to obtain real-time data that can calculate range and speed to send to the user via email or text message. In order to do this DT6 had to select hardware (ADC, microcontroller, etc.) that could send data at a fast rate. DT6 decided to go with an ADC and TI Stellaris microcontroller. The Stellaris had an on board Ethernet port which could pass data directly to a PC for

processing. Although there are some shortcomings on correct signal processing for the receiving of data, the data is able to be processed in real time.

Task 2 was to make a small design which could be mounted for implementation. This was a team decision, based on the use for this design project. The goal of this task was to develop all boards (ADC, ramp generator, etc.) for the project in as small a manner as possible so that all of the project could fit on 12x12 size wooden board for mounting.

Task 3 was to make the design project remotely operated. This means all processing was to be done on the microcontroller and then sent to a PC to be sent off to the user, as well as having on board power. On board power was established, however, because of the microcontroller that DT6 selected there was not enough RAM to run signal processing on the microcontroller. Because of this, DT6 opted to put all signal processing on the PC side to determine the range and speed calculations. A microcontroller with additional RAM would be able to run these processes.

Task 4 was to establish reliable communication for detection. Although DT6 is able to communicate detections, the raw signal from the radar has amplitude variations that are not uniform across the spectrum. As such digital interpretation of the data is inaccurate. Therefore, the Fourier Transforms and speed and range calculations are only successful on a computer-generated frequency sweep. Advanced knowledge of signal processing could alleviate these variations and allow for range and speed calculations.

With a budget of \$500 for the project this semester, Design Team 6 used the majority of the allotted amount. The final cost of the project came to \$422.58 with just \$77.42 to spare. The bulk of the cost came from the RF stage of the project which cost \$235.55. This section of the project was assembled very carefully and checked multiple times before running any experiments or testing with the system because the individual components are so expensive. After buying all of the parts required for the project that was provided from MIT, with their bill of materials, the budget was left with \$177.56. In addition to the bill of materials, the project required a processor and an analog to digital converter and all of the discrete components required to implement these parts. Towards the end of the project a decision was made to create a PCB for all of the parts of the project with surface mount components. With the extra surface mount parts and a couple of parts that were re-ordered due to burnt chips, the final budget came down to \$422.48 used for the project.

Design:	Amount (\$)
Antenna/Radar	\$ 289.03
Integrated Circuits	\$ 35.80
Discrete Components	\$ 97.75
TOTAL	\$ 422.80

There is still much room for project improvements with this specific design project. One item that could be improved is the signal processing in order to reduce the amplitude variation so that information could be processed. A second improvement would be picking a microcontroller that had adequate RAM to be able to store the signal processing on the microcontroller instead of doing it all on the PC side of the project. These two improvements would make a successful working remotely operated intrusion alarm which would meet all the requirements.

Overall, the project completed its objective as a proof of concept, with room for optimization. A few minor setbacks put the timeline behind schedule, leaving some parts of the project being unable to get started on time. Though DT6 was not able to meet all requirements, a fully functioning project was still established. DT6 considers this project a success, and was honored to work with MIT Lincoln Laboratory this semester.

Appendix 1: Technical Roles, Responsibilities, and Work Accomplished

Peter Deacon

Peter's main technical tasks were related to hardware development, fabrication, and testing. The first task that Peter completed was the build and test of the FMCW Radar. The second was the design of a printed circuit board that contained all of the radar's support circuitry.

For the first task Peter worked with Hunt, Oakley, and Leonard on individual parts of the hardware. MIT Lincoln Laboratory designed FMCW Radar and support circuitry used in the initial phase of this project. With Leonard, Peter built the coffee can transmitter and receiver antennas. This task involved placing a monopole in a cylindrical wave guide and trimming the monopole to have its maximum gain at 2.4GHz. The testing for this task was done on a network analyzer at the Engineering Research Center with the aid of an MSU graduate student Andrew Temme. Temme granted Design Team 6 access to the high frequency network analyzer required for the tuning of the antennas.

Peter worked with Hunt on the build of the radar's support circuitry. This included building and testing a 25Hz ramp generator which was used to drive a voltage controlled oscillator for the radar. The ramp generator had to be tuned by measuring the output frequency on an oscilloscope and adjusting a potentiometer until the desired frequency was reached. The radar support circuitry also included a low pass filter with an amplifier. The testing for this part of the circuitry included tuning the corner frequency of the filter to 15 kHz to insure that high frequency noise would be removed from the received radar signal. The roll off rate of the filter was also tested to determine if high frequency signals would be attenuated properly.

For the final product, Peter worked with Leonard to create a PCB containing all of the radar support circuitry, as well as an ADC and a Stellaris microcontroller and the discrete components required for their operation. For the PCB build Peter had to learn how to use Dip Trace a PCB creation tool which allows for the creation of custom pads and components. Peter and Leonard created custom pads for each of the components on the board and converted those pads into components that could be placed

on a schematic. Peter and Leonard then created a schematic for the circuitry and converted it into an auto-routed PCB. Several of the traces created by the auto-routing software had to be changed to be compatible with fabrication techniques available to Design Team 6.

Ryan Hunt

Ryan worked with Peter on building the ramp generator and filter according to the design provided by Lincoln Lab. He and Peter built the components on a breadboard as a prototype, then tested and calibrated them to what the project required.

Ryan worked with the entire group on calibrating the antennas. He and the rest of the group spent a day at the ERC using a network analyzer and an anechoic chamber to measure the SWR. They trimmed the each antenna until the SWR was below NUMBER between 2.4 and 2.6 GHz. Ryan was also involved in testing the radar stage to ensure it was working properly. This was done by repeating the experiments that were performed in MIT, and were successful.

Ryan's primary individual contribution to the project was creating communication between the ADC and the microcontroller. He had to use the ADC data sheets to determine how to configure the ADC as a master. He then had to use the TI data sheets to determine which GPIO pins to configure and connect to the ADC for I²S communication. He had to use TI's StellarisWare peripheral library to find the functions to configure the I²S to operate as a slave, in normal I²S mode, with a sample size of 24 bits. He also wrote the interrupt handler to save ADC data to an array, and enabled the I²S receive module.

Finally, Ryan worked with Darci on combining the Ethernet and ADC sampling programs into one program. Care had to be taken to ensure that all relevant files and headers were linked to the project, and that interrupts had the correct priorities.

Darci Koenigsknecht

Darci had four main tasks that she needed to complete for our project to be successful. Two tasks were hardware based and two were software based.

For the hardware, she had to implement the RF stage components on the design board, properly connecting each piece and screwing them to the board. Another task for this was connecting all the VDD and ground wires to the RF stage components by soldering them to the correct pieces. The second hardware task was contributing to the tuning of the antennas on the radar portion of our design project, as well as helping to test the component was available.

Darci's main software task was to develop network connection between the Stellaris microcontroller and our laptop. This was a difficult task because she had never used code composer before, and there was minimal examples available to learn how to make a network connection. She found an already available code to implement Ethernet connection between the Stellaris and the laptop; however the connection was through a web server. In viewing the code that was used to connect to a web server, and code that was found through the TI E2E Community website, she was able to work with another team member to properly piece the parts together and get a static IP connection between our laptop and the Stellaris and sent an array across to make sure we could accept data properly. When this was

completed, she worked with Ryan to implement his I2S code into her program so that they could send the data from the ADC through the Stellaris and over Ethernet to the PC where the signal processing was going to be done.

A smaller and short-lived task that Darci worked on was code to calculate the range and speed of the intruder from the data received about the intrusion. This task was short-lived because problems arose with the project that was limiting the ability to produce remote signal processing on the controller. Therefore the range and speed would be implemented on the laptop and much easier to develop.

Chris Leonard

Chris contributed to many technical roles throughout the design project. Some of his technical contributions include construction of the antennas, tuning of the antennas, writing a Fast Fourier Transform (FFT) in Microsoft Visio, testing the radar, and laying out a PCB for the final design of the project.

Chris's first task was helping the group assemble the two antennas and then tune them at the engineering research center. Chris and another group member spent some time writing a FFT in Microsoft Visio in order to test other group members' code. This code was ultimately not used because another group member had also written code for an FFT in order to test code. After other group members had some code to test, Chris and other group members were able to test the antennas with a sample simulation file provided from MIT as well as the data acquisition code and were able to get useable results.

A good deal of Chris's time was spent designing and assembling the final PCB for the project. He first had to find all of the parts that he needed for the final board in surface mount form and make sure that they were all correct parts. Then he taught himself a new program that was used to lay out a PCB. This included making pads for each component of the board and making sure that the dimensions of each pad were the right size and dimension. Then he laid out the entire board in a schematic form and then used the program to find the optimal trace routes for the PCB. The board then required some debugging in order to make sure that every trace was in the right spot and that all of the parts of the board were again connected correctly. Chris was able to complete the PCB as well as helping with the testing of the antenna and code.

Chris Oakley

Chris Oakley had a leading role in the selection of components used in the design of this project. He was also responsible for development of all C# applications used during the design phase, as well as aided in troubleshooting of analog and radio-frequency circuitry used. Chris provided information on how the system developed by MIT Lincoln Laboratory functioned. He used this information to aid in the selection of a micro-controller and analog-to-digital converter.

PCBs were also designed to adapt the small components for use in a prototyping board. The PCBs were assembled, and additional circuitry required was installed to a prototyping board so testing could be carried out, while allowing for easy changes to the system configuration.

Early in the build process, difficulties were encountered with both the analog circuitry, as well as with the radio-frequency components. Chris assisted in the troubleshooting of these errors, as well as developing additional circuitry to prevent future errors and verifying correct operation of each radio-frequency component. Operation of these components was verified through the use of a spectrum analyzer. The voltage controlled oscillator was first verified to be operating correctly by verifying the power level at the desired output frequency. All other components were verified by using the voltage controlled oscillator as a signal source, and again verifying the correct output power and frequency with a spectrum analyzer.

PC applications were developed for use both during the design phase, as well as for demonstration purposes. Chris developed these applications using the Microsoft C# programming language. A Fourier transform application was first developed to ensure proper spectral analysis techniques were implemented. This included developing a sinusoid generator, DFT function, and finally an FFT function. The results of these were graphed for a user to verify correct operation. A WAV processing library was developed for use during testing of the radar signal processing software. This enabled a user to both display time data, and display spectral data for each sample of a signal. An additional application was developed for use during development of a network communication system used between a PC and the micro-controller. This application allowed a user to view the raw data being passed from the micro-controller to the PC, as well as displaying the number of bytes received with each transfer.

Appendix 2: References

Charvat, Gregory, et al. "Build a Small Radar System Capable of Sensing Range, Doppler, and Synthetic Aperture Radar Imaging" MIT Open Courseware. 2011. <<http://ocw.mit.edu/resources/res-ll-003-build-a-small-radar-system-capable-of-sensing-range-doppler-and-synthetic-aperture-radar-imaging-january-iap-2011/index.htm>>

"Cooley-Tukey FFT Algorithm (C)" Literate Programs. 17 November 2008.
<http://en.literateprograms.org/Cooley-Tukey_FFT_algorithm_%28C%29>

Dunkels, Adam, et al. "Raw TCP/IP interface for lwIP".
<http://cvs.savannah.gnu.org/viewvc/*checkout*/lwip/lwip/doc/rawapi.txt?view=auto&revision=1>

"I2S Bus Specifications" Philips Semiconductors. June 5, 1996. <<http://www.sparkfun.com/datasheets/BreakoutBoards/I2SBUS.pdf>>

"SINGLE-ENDED, ANALOG-INPUT 24-BIT, 96-kHz STEREO A/D CONVERTER". SLES177A. Rev August 2006. Texas Instruments Incorporated, 2007-2011. <<http://www.ti.com/lit/ds/sles177a/sles177a.pdf>>

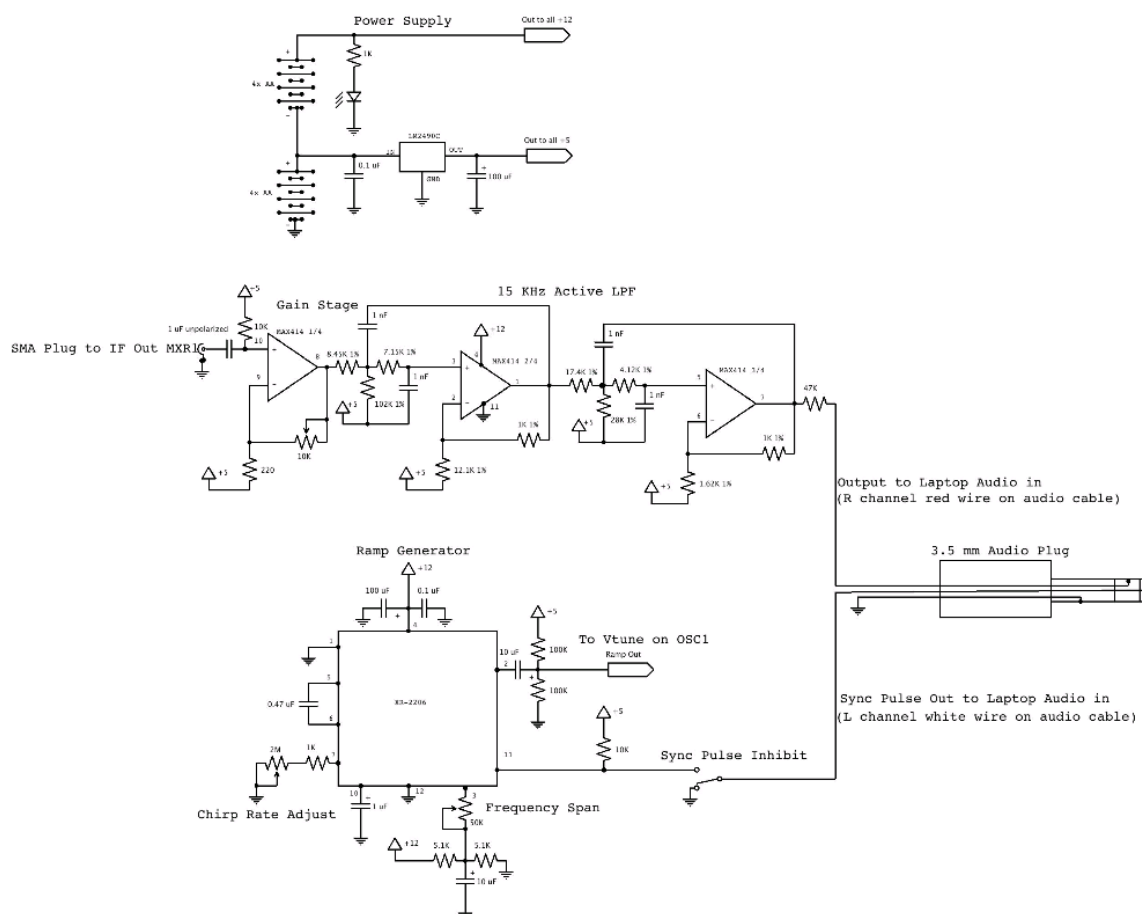
"Stellaris LM3S9B90 Microcontroller Data Sheet". DS-LM3S9B90-9970. Texas Instruments Incorporated, 2007-2011. <<http://www.ti.com/lit/ds/symmlink/lm3s9b90.pdf>>

“Stellaris Peripheral Driver Library User’s Guide”. SW-DRL-UG-8264. Texas Instruments Incorporated, 2007-2011. < <http://www.ti.com/lit/ug/spmu019m/spmu019m.pdf>>

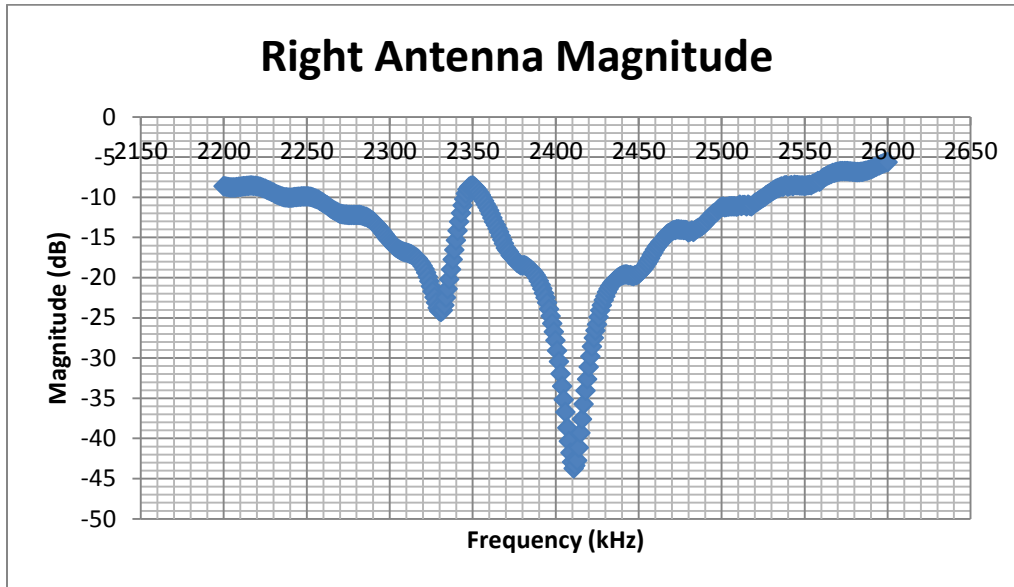
“TCP Client for Stellaris using lwIP”. TI E2E Community. User gabriellegarda. March 22, 2010. <http://e2e.ti.com/support/microcontrollers/stellaris_arm_cortex-m3_microcontroller/f/471/p/44726/163526.aspx#163526> Edited March 22, 2010

Appendix 3: Technical Attachments

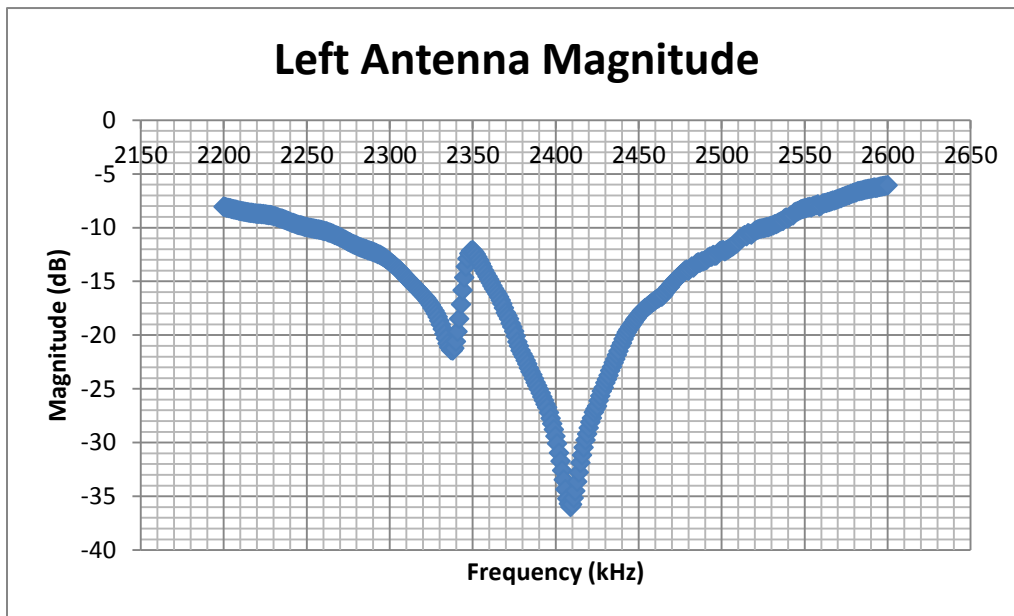
Ramp and filter schematic



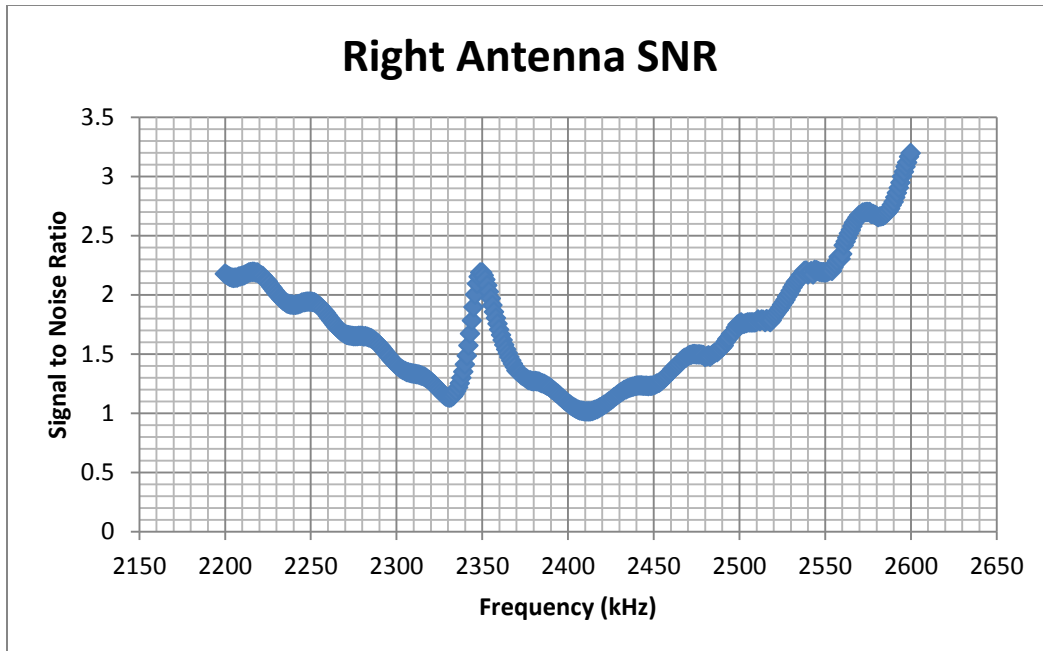
Tuning the antennas



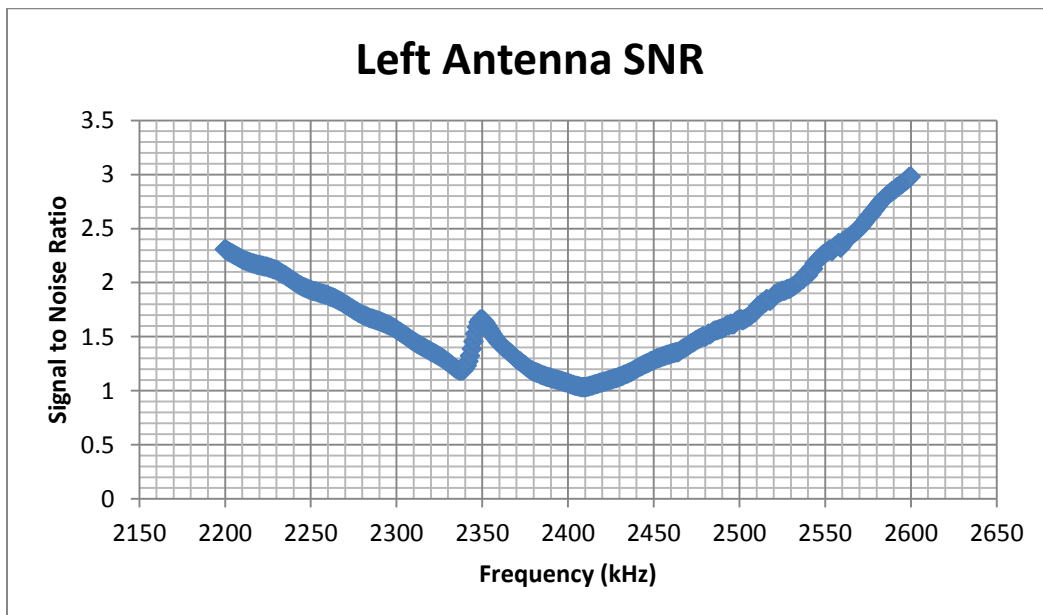
Right antenna resonant frequency



Left antenna resonant frequency

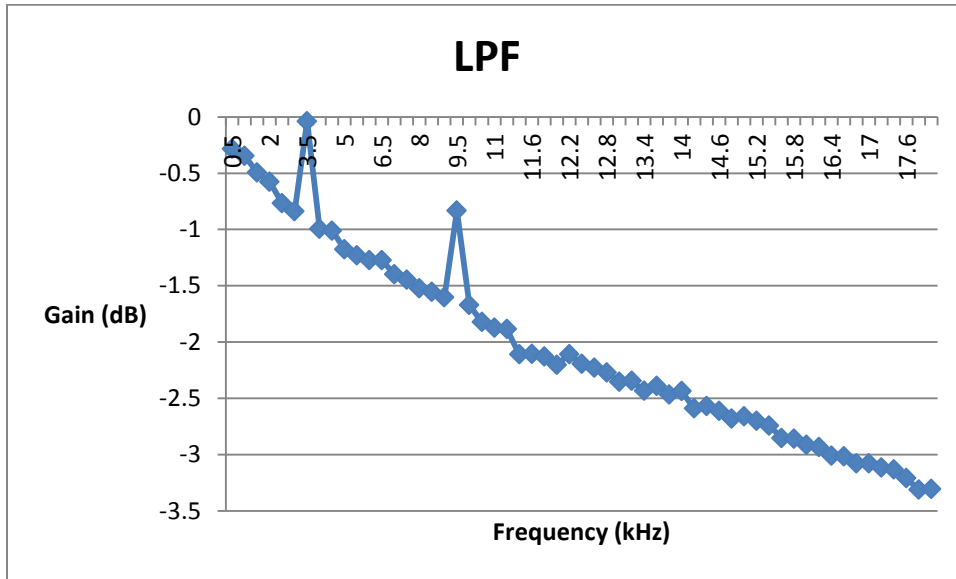


Right antenna SNR

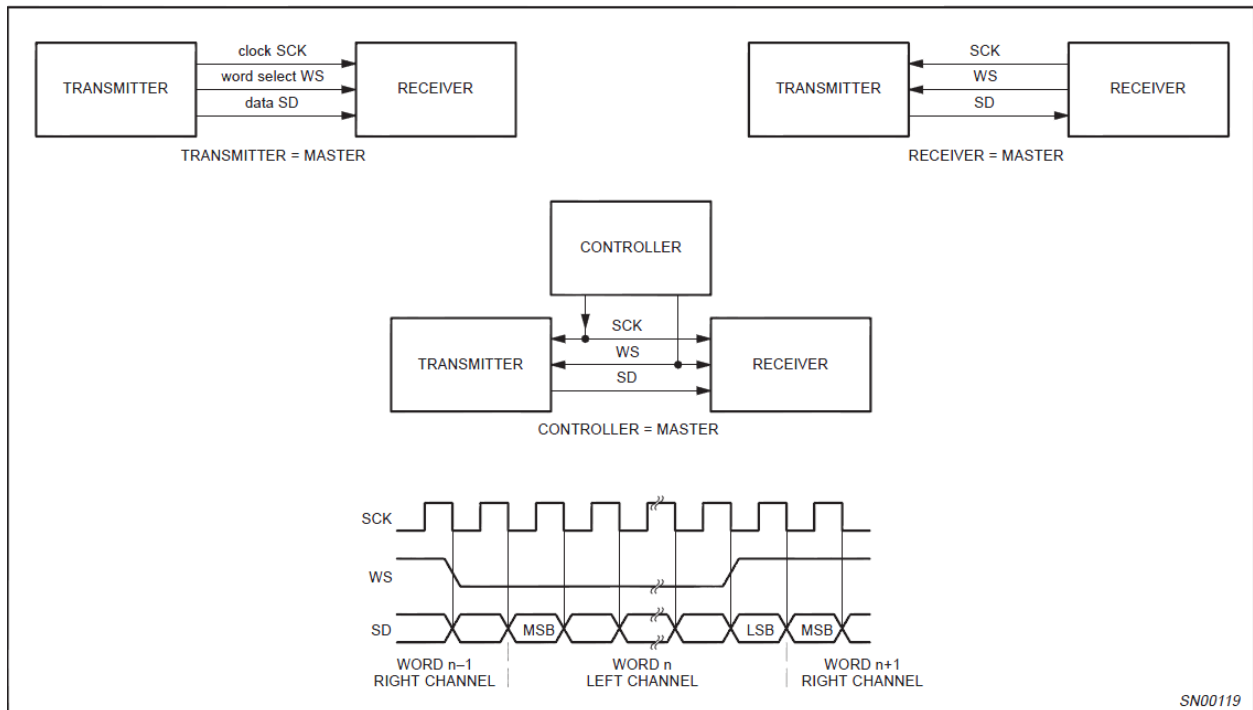


Left antenna SNR

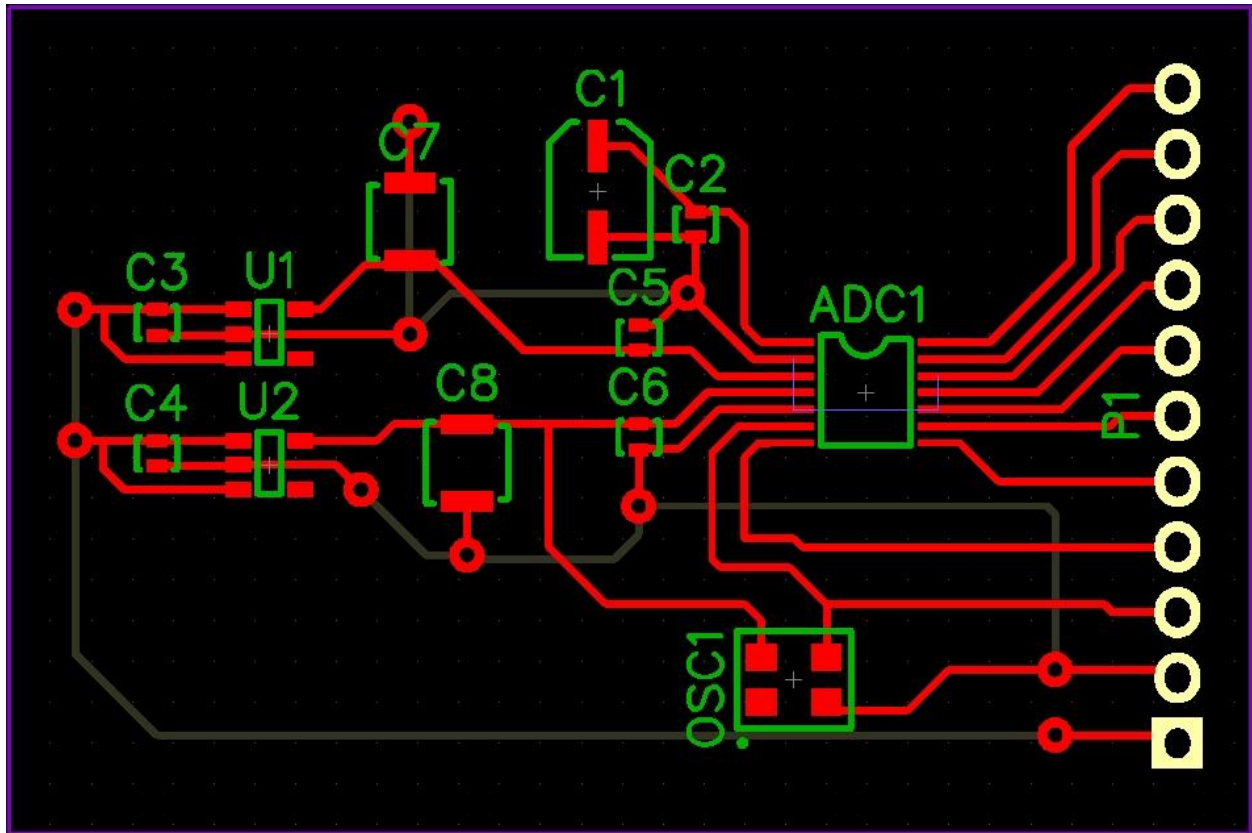
Low Pass Filter



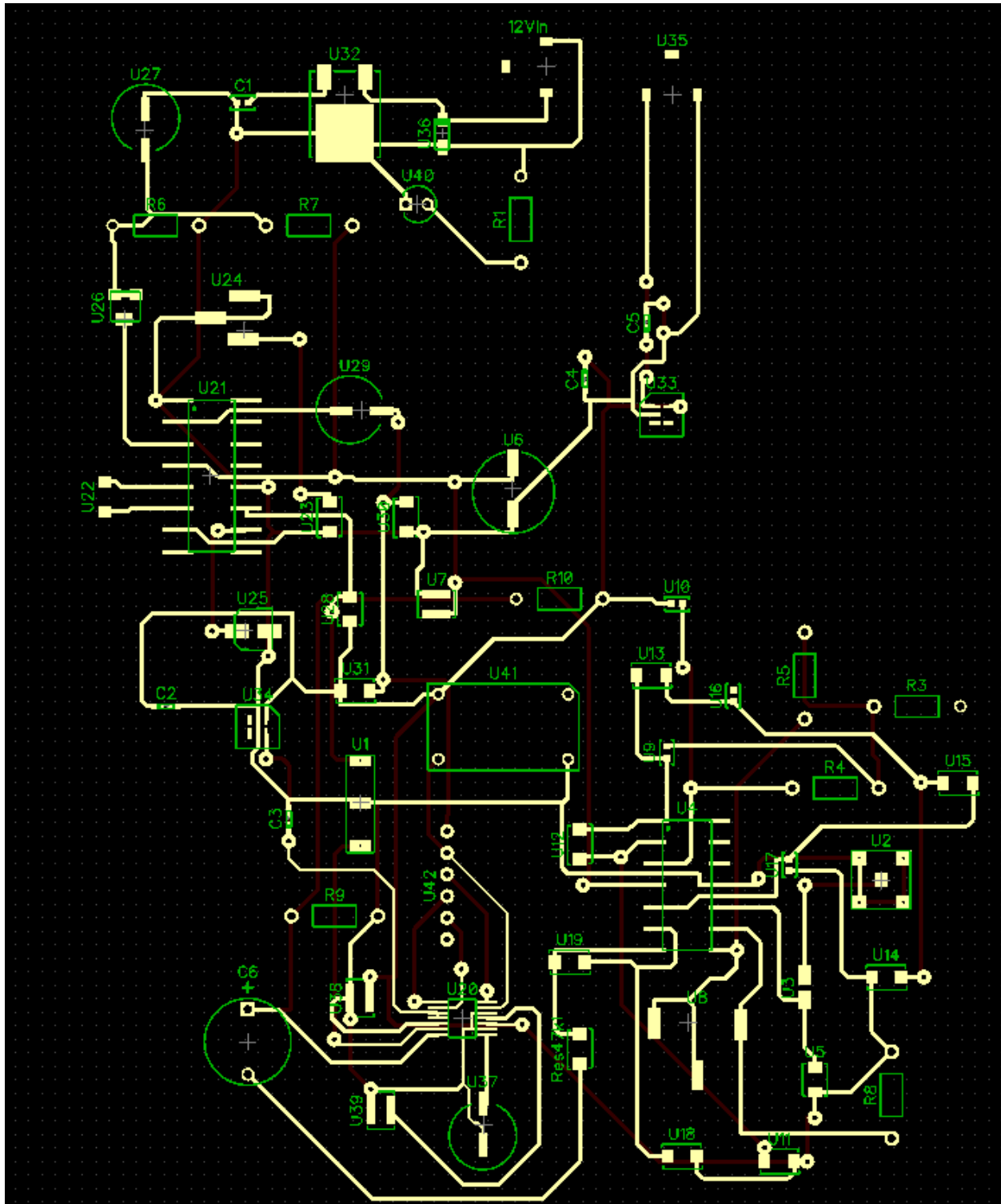
I²S Bus Overview



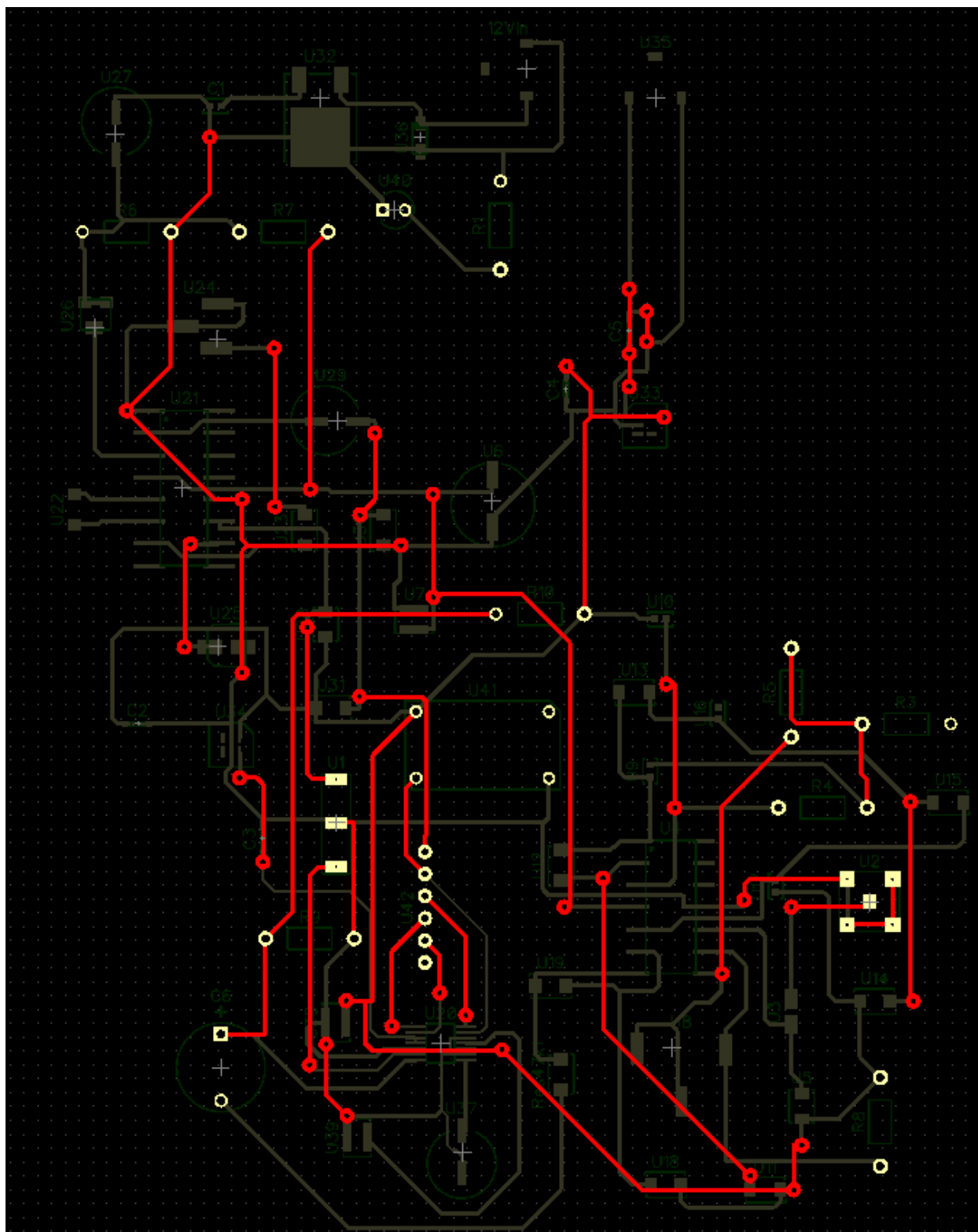
ADC PCB Design

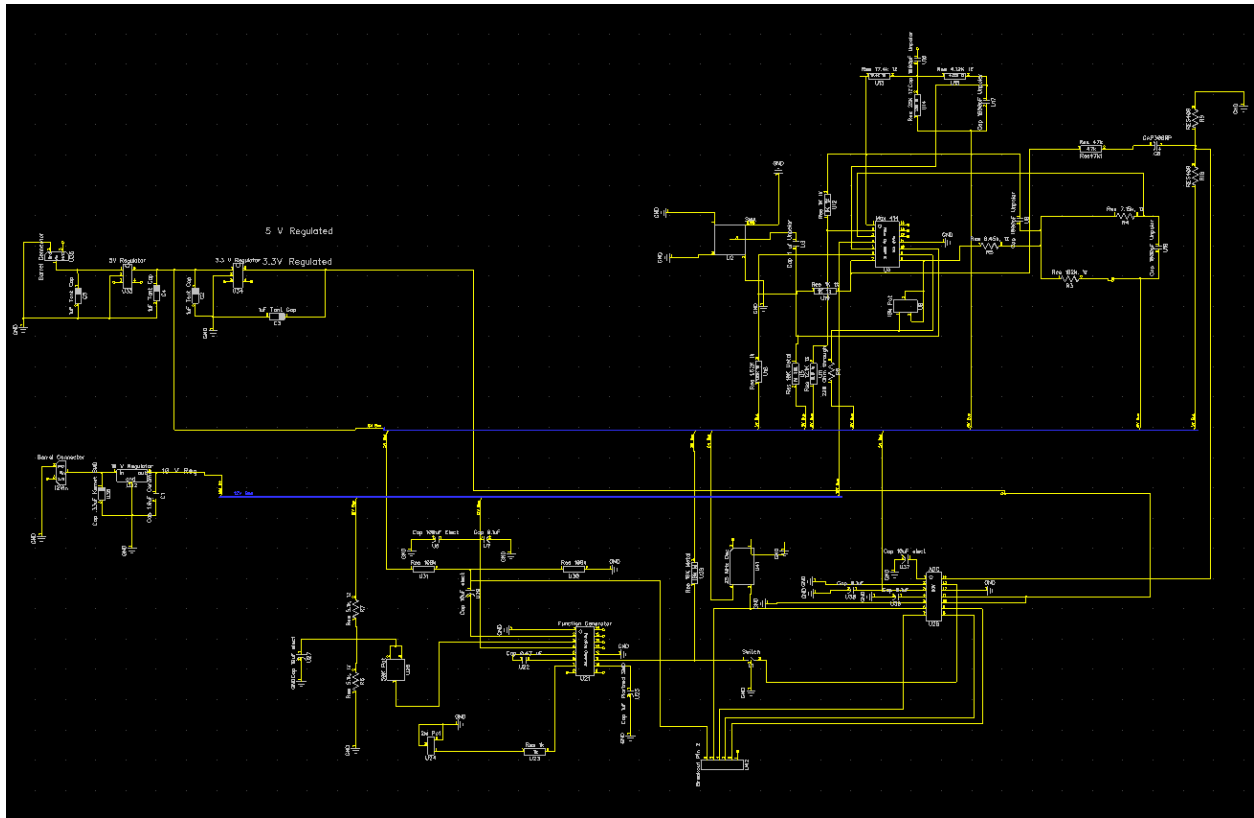


Filter, Ramp, and ADC PCB Design



PCB Top





Schematic used for PCB

Itemized Budget

Qty	Mfg Part Number	Supplier Part Number	Manufacture / Description	Cost each	Line Item Total
1	ZX95-2536C+	ZX95-2536C+	2315-2536 MC VCO, +6dBm Out	\$ 44.950	\$ 44.950
1	VAT-3+	VAT-3+	3dB SMA M-F attenuator	\$ 11.950	\$ 11.950
2	ZX60-272LN-S+	ZX60-272LN-S+	Gain 14 dB, NF=1.2 dB, IP1=18.5 dBm	\$ 39.950	\$ 79.900
1	ZX10-2-42+	ZX10-2-42+	1900-4200 Mc, 1.0 dB insertion loss	\$ 34.950	\$ 34.950
1	ZX05-43MH-S+	ZX05-43MH-S+	13 dB LO, RF to LO loss 6.1 dB, IP1 9 dBm	\$ 46.450	\$ 46.450
4	SM-SM50+	SM-SM50+	SMA-SMA M-M barrel	\$ 5.950	\$ 23.800
3	086-12SM+	086-12SM+	SMA-SMA M-M 6" cable	\$ 9.950	\$ 29.850

Qty	Mfg Part Number	Supplier Part Number	Manufacture / Description	Cost each	Line Item Total
2	901-9889-RFX	523-901-9889-RFX	SMA bulkhead F solder cup	\$ 4.270	\$ 8.540
1	XR-2206	NA - Obsolete	Function Generator Chip	\$ 0.000	\$ 0.000
2	K104Z15YSVE5TH5	594-K104Z15YSVE5TH5	Decoupling Cap 0.1 uF	\$ 0.050	\$ 0.100
2	UVR1E101MED1TD	647-UVR1E101MED1TD	100 uF cap	\$ 0.070	\$ 0.140
1	PV36Y103C01B00	81-PV36Y103C01B00	10k pot	\$ 1.360	\$ 1.360
1	CFP1/4CT52R201J	660-CFP1/4CT52R201J	200 ohm, 5%	\$ 0.029	\$ 0.029
2	41931	517-41931	4" cable ties	\$ 0.040	\$ 0.080
1	FK28Y5V1E474Z	No stock	0.47 uF capacitor	\$ 0.000	\$ 0.000
1	PV36W205C01B00	81-PV36W205C01B00	2M trimmer potentiometer	\$ 1.360	\$ 1.360

Qty	Mfg Part Number	Supplier Part Number	Manufacture / Description	Cost each	Line Item Total
1	PV36W503C01B00	81-PV36W503C01B00	50K trimmer potentiometer	\$ 1.360	\$ 1.360
1	UVR1H010MDD1TD	647-UVR1H010MDD1TD	1 uF electrolytic cap	\$ 0.060	\$ 0.060
1	UVR1H100MDD1TA	647-UVR1H100MDD1TA	10 uF electrolytic cap	\$ 0.060	\$ 0.060
2	MF1/4DCT52R5101F	660-MF1/4DCT52R5101F	5.1K resistor	\$ 0.060	\$ 0.120
2	CCF0710K0JKE36	71-CCF0710K0JKE36	10K resistor	\$ 0.080	\$ 0.160
1	TLHR5400	78-TLHR5400	Red LED	\$ 0.080	\$ 0.080
12	CCF0747K0JKR36	71-CCF07-J-47K	47K 5% resistor	\$ 0.040	\$ 0.480
1	T356A105M020AT73 01	80-T356A105M020AT-TR	1 uF tantalum capacitor	\$ 1.320	\$ 1.320
0				\$ 0.000	\$ 0.000

Qty	Mfg Part Number	Supplier Part Number	Manufacture / Description	Cost each	Line Item Total
2	XR2206CP		Function generator IC	\$ 6.490	\$ 12.980

Qty	Mfg Part Number	Supplier Part Number	Manufacture / Description	Cost each	Line Item Total
1		595-LM3S9D92-ICQ80A2	Stellaris MCU	\$ 14.520	\$ 14.520

Qty	Mfg Part Number	Supplier Part Number	Manufacture / Description	Cost each	Line Item Total
1	ASFLMB-24.576MHZ-LR-	815-ASFLMB-24.5-LR-T	24.576MHz Oscillator	\$ 2.250	\$ 2.250
10	VE-100M1ATR-0405	140-VE-100M1ATR0405	10uF electrolytic Cap	\$ 0.120	\$ 1.200
10	C0402C104K8PACTU	80-C0402C104K8P	0.1uF ceramic Cap	\$ 0.070	\$ 0.700
3	NCV551SN50T1G	863-NCV551SN50T1G	5V regulator	\$ 0.420	\$ 1.260
3	NCP511SN33T1G	863-NCP511SN33T1G	3.3V regulator	\$ 0.550	\$ 1.650
4	C3216X8R1E105KT	810-C3216X8R1E105K	Low ESR Cap	\$ 0.630	\$ 2.520
1	3-826629-6	571-3-826629-6	Header	\$ 3.500	\$ 3.500

Qty	Mfg Part Number	Supplier Part Number	Manufacture / Description	Cost each	Line Item Total
2	1MAX141CPD+	MAX141CPD+-ND	low-noise quad op-amp	\$ 5.860	\$ 11.720

Qty	Mfg Part Number	Supplier Part Number	Manufacture / Description	Cost each	Line Item Total
2	PP3-002A	CP3-1000-ND	Barrel Connector 2.1mm ID, 5.5mm OD	\$ 1.360	\$ 2.720
2	PJ-102A	CP-102A-ND	Barrel Connector 2.1mm ID, 5.5mm OD	\$ 0.920	\$ 1.840
2	ERJ-8GEYJ101V	P100ECT-ND	Resistor 100 ohm	\$ 0.060	\$ 0.120
2	EJR-8GEYJ102V	P1.0KECT-ND	Resistor 1K ohm	\$ 0.060	\$ 0.120
4	F981C105MMA	493-2883-1-ND	Capacitors 1.0 uF (tantalum)	\$ 0.910	\$ 3.640
1	XC6201P33	893-1054-1-ND	Regulator 3.3 V	\$ 0.650	\$ 0.650
1	XC6201P50	893-1055-1-ND	Regulator 5V	\$ 0.650	\$ 0.650
2	445-1328-1-ND	C1608Y5V1A105Z	Capacitors 1.0 uF (ceramic)	\$ 0.140	\$ 0.280
1	T491A334K035AT	399-3693-1-ND	Capacitors 0.33 uF	\$ 0.480	\$ 0.480

Qty	Mfg Part Number	Supplier Part Number	Manufacture / Description	Cost each	Line Item Total
1	EEE-1HAR10AR	PCE4625CT-ND	Capacitors 0.1 uF electrolytic	\$ 0.500	\$ 0.500
1	ACH-24.576MHZ-EK	No Stock	Oscillator 25MHz	\$ 0.000	\$ 0.000
3	ERJ-8ENF1001V	P1.00KFCT-ND	Resistor 1K 1%	\$ 0.100	\$ 0.300
1	ERJ-8ENF1212V	P12.1KFCT-ND	Resistor 12.1k 1%	\$ 0.100	\$ 0.100
1	ERJ-8ENF1742V	P17.4KFCT-ND	Resistor 17.4K 1%	\$ 0.100	\$ 0.100
1	ERJ-8ENF2802V	P28.0KFCT-ND	Resistor 28.0k 1%	\$ 0.100	\$ 0.100
1	ERJ-8ENF4121V	P4.12KFCT-ND	Resistor 4.12K 1%	\$ 0.100	\$ 0.100
1	ERJ-8ENF1621V	P1.62KFCT-ND	Resistor 1.62K 1%	\$ 0.100	\$ 0.100
3	LLL31MR71H104MA01L	490-4376-1-ND	Capacitor 0.1uF	\$ 0.650	\$ 1.950

Qty	Mfg Part Number	Supplier Part Number	Manufacture / Description	Cost each	Line Item Total
5	C1608X7R1H102J	445-5079-1-ND	Capacitor 1000 pF 5%	\$ 0.160	\$ 0.800
1	EEE-1HA010NR	PCE4301CT-ND	Capacitor 1uF Electrolytic	\$ 0.510	\$ 0.510
1	EEE-1VA100NP	PCE4297CT-ND	Capacitor 10uF Electrolytic	\$ 0.700	\$ 0.700

Qty	Mfg Part Number	Supplier Part Number	Manufacture / Description	Cost each	Line Item Total
2	EEE-FC1V1R0R	667-EEE-FC1V1R0R	Cap 1 uF electrolytic	\$ 0.470	\$ 0.940
3	UUD1E100MCL1GS	647-UUD1E100MCL	Cap 10 uF electrolytic	\$ 0.580	\$ 1.740
1	T498A105M035ATE10K	80-T498A105M35TE10K	Cap 1 uF unpolarized	\$ 0.460	\$ 0.460
12	CR1206-JW-473GLF	652-CR1206-JW-473GLF	Resistor 47K ohm	\$ 0.050	\$ 0.600
2	CR1206-FX-1003GLF	652-CR1206-FX1003GLF	Resistor 100k ohm	\$ 0.050	\$ 0.100
1	LS Q976-NR-1	720-LSQ976-NR-1	Red LED	\$ 0.100	\$ 0.100
2	SMM02040C1002FB300	71-SMM02040C1002FB30	Resistor 10K Metal Film	\$ 0.110	\$ 0.220
1	44WR50KLFT7	858-44WR50KLFT7	Pot 50K Potentiometer	\$ 3.210	\$ 3.210
1	3361S-1-205GLF	652-3361S-1-205GLF	Pot 2M Potentiometer	\$ 1.020	\$ 1.020

Qty	Mfg Part Number	Supplier Part Number	Manufacture / Description	Cost each	Line Item Total
1	CGA5L2X8R1H474K	810-CGA5L2X8R1H474K	Tuning Cap 0.47 uF cap	\$ 0.780	\$ 0.780
1	3361P-1-103GLF	652-3361P-1-103GLF	Trimmer Pot 10K Pot	\$ 1.020	\$ 1.020
2	UCZ1E101MCL1GS	647-UCZ1E101MCL1GS	Decoupling Cap 100 uF	\$ 0.970	\$ 1.940
2	MF1/4DCT52R5101F	660-MF1/4DCT52R5101F	Resistor 5.1k Metal film	\$ 0.050	\$ 0.100
1		815-ACOL-24.576-EK	Oscillator, 24.576MHz	\$ 1.600	\$ 1.600

Stellaris Microcontroller Code

```
//*****
**
//
// enet_lwip.c - Sample WebServer Application using lwIP.
//
// Copyright (c) 2009-2011 Texas Instruments Incorporated. All rights
reserved.
// Software License Agreement
//
// Texas Instruments (TI) is supplying this software for use solely and
// exclusively on TI's microcontroller products. The software is owned by
// TI and/or its suppliers, and is protected under applicable copyright
// laws. You may not combine this software with "viral" open-source
// software in order to form a larger program.
//
// THIS SOFTWARE IS PROVIDED "AS IS" AND WITH ALL FAULTS.
// NO WARRANTIES, WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT
// NOT LIMITED TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
// A PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. TI SHALL NOT, UNDER ANY
// CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL, OR CONSEQUENTIAL
// DAMAGES, FOR ANY REASON WHATSOEVER.
//
// This is part of revision 8049 of the EK-LM3S9B92 Firmware Package.
//
//*****
**

#include "inc/hw_ints.h"
#include "inc/hw_memmap.h"
#include "inc/hw_nvic.h"
#include "inc/hw_types.h"
#include "driverlib/timer.h"
#include "driverlib/ethernet.h"
#include "driverlib/flash.h"
#include "driverlib/gpio.h"
#include "driverlib/interrupt.h"
```

```

#include "driverlib/pin_map.h"
#include "driverlib/rom.h"
#include "driverlib/sysctl.h"
#include "driverlib/systick.h"
#include "driverlib/i2s.h"
#include "utils/locator.h"
#include "utils/lwiplib.h"
#include "lwip/tcp.h"
#include "lwip/err.h"
// #include "driverlib/drivers/set_pinout.h"

// *****
**
//
//! \addtogroup example_list
//! <h1>Ethernet with lwIP (enet_lwip)</h1>
//!
//! This example application demonstrates the operation of the Stellaris
//! Ethernet controller using the lwIP TCP/IP Stack. DHCP is used to obtain
//! an Ethernet address. If DHCP times out without obtaining an address,
//! AutoIP will be used to obtain a link-local address. The address that is
//! selected will be shown on the UART.
//!
//! UART0, connected to the FTDI virtual COM port and running at 115,200,
//! 8-N-1, is used to display messages from this application.
//!
//! For additional details on lwIP, refer to the lwIP web page at:
//! http://savannah.nongnu.org/projects/lwip/
//
// *****
**

// *****
**
//
// Defines for setting up the system clock.
//
// *****
**
#define SYSTICKHZ 100
#define SYSTICKMS (1000 / SYSTICKHZ)

// *****
**
//
// A twirling line used to indicate that DHCP/AutoIP address acquisition is
// in
// progress.
//
// *****
**
// static char g_pcTwirl[4] = { '\\', '|', '/', '-' };

// *****
**
//

```

```

// The index into the twirling line array of the next line orientation to be
// printed.
//
//*****
**
static unsigned long g_ulTwirlPos = 0;

//*****
**
//
// The most recently assigned IP address. This is used to detect when the IP
// address has changed (due to DHCP/AutoIP) so that the new address can be
// printed.
//
//*****
**
static unsigned long g_ulLastIPAddr = 0;

//*****
***
//
// Buffer management and flags.
//
//*****
***
#define AUDIO_BUFFER_SIZE      2048
static unsigned long Left_Buffer; //Left channel, the sync signal
static unsigned long Right_Buffer[AUDIO_BUFFER_SIZE]; //Right channel, the
audio data
static int Left_Sample = true; //flag to show whether left or right sample is
the next one in the FIFO, starting with the left sample
static int RBS = 0; //marker for the right buffer
static int tx_ready = false; //flag for if the data sampled should be
transmitted
static unsigned long Trash_Buffer; //Garbage variable to discard unwanted
data in the FIFO

static struct tcp_pcb *pcb; //constant PCB for TCP, to be called in many
functions

unsigned long ulSampleRate = 96000;
unsigned long usBitsPerSample = 24;

//*****
**
//
// The error routine that is called if the driver library encounters an
error.
//
//*****
**
#ifdef DEBUG
void
__error__(char *pcFilename, unsigned long ulLine)
{
}
#endif

```

```

//*****
**
//
// Required by lwIP library to support any host-related timer functions.
//
//*****
**
void
lwIPHostTimerHandler(void)
{
    unsigned long ulIPAddress;

    //
    // Get the local IP address.
    //
    ulIPAddress = lwIPLocalIPAddrGet();

    //
    // See if an IP address has been assigned.
    //
    if(ulIPAddress == 0)
    {
        //
        // Draw a spinning line to indicate that the IP address is being
        // discovered.
        //
        //UARTprintf("\b%c", g_pcTwirl[g_ulTwirlPos]);

        //
        // Update the index into the twirl.
        //
        g_ulTwirlPos = (g_ulTwirlPos + 1) & 3;
    }

    //
    // Check if IP address has changed, and display if it has.
    //
    else if(ulIPAddress != g_ulLastIPAddr)
    {
        //
        // Display the new IP address.
        //
        //UARTprintf("\rIP: %d.%d.%d.%d      \n", ulIPAddress & 0xff,
        //          (ulIPAddress >> 8) & 0xff, (ulIPAddress >> 16) & 0xff,
        //          (ulIPAddress >> 24) & 0xff);

        //
        // Save the new IP address.
        //
        g_ulLastIPAddr = ulIPAddress;

        //
        // Display the new network mask.
        //
        ulIPAddress = lwIPLocalNetMaskGet();
        //UARTprintf("Netmask: %d.%d.%d.%d\n", ulIPAddress & 0xff,

```

```

        //          (ulIPAddress >> 8) & 0xff, (ulIPAddress >> 16) & 0xff,
        //          (ulIPAddress >> 24) & 0xff);

        //
        // Display the new gateway address.
        //
        ulIPAddress = lwIPLocalGWAddrGet();
        //UARTprintf("Gateway: %d.%d.%d.%d\n", ulIPAddress & 0xff,
        //          (ulIPAddress >> 8) & 0xff, (ulIPAddress >> 16) & 0xff,
        //          (ulIPAddress >> 24) & 0xff);
    }
}

//*****
//
// The interrupt handler for the SysTick interrupt.
//
//*****
void
SysTickIntHandler(void)
{
    //
    // Call the lwIP timer handler.
    //
    lwIPTimer(SYSTICKMS);
}

//*****
//
// Close all connections between the Stellaris and the PC.
//
//*****

static void client_close(struct tcp_pcb *pcb)
{
    tcp_arg(pcb, NULL);
    tcp_sent(pcb, NULL);
    tcp_close(pcb);
}

//*****
//
// Make sure all data has been received by PC, then call the close function.
//
//*****

static err_t client_sent(void *arg, struct tcp_pcb *pcb, u16_t len)
{
    LWIP_UNUSED_ARG(arg);
    //client_close(pcb);
    return ERR_OK;
}

```

```

}

//*****
**
//
// Upon connection, call the write function to send data to the PC.
//
//*****
**

static err_t client_connected(void *arg, struct tcp_pcb *pcb, err_t err)
{
    unsigned long dummy[10] = {1,2,3,4,5,6,7,8,9,10};
    LWIP_UNUSED_ARG(arg);

    if (err != ERR_OK)
    {
    }
    else
    {
        tcp_sent(pcb, client_sent);
        //tcp_write(pcb, dummy, sizeof(dummy), 1);
    }
    return err;
}
//*****
*
* Interrupt handler for the I2S
*
* *****/
void My_I2S_Int (void)
{
    int ulStatus;
    int i = 0;

    //
    // Get the interrupt status to see what the interrupt is.
    //

    ulStatus = I2SIntStatus(I2S0_BASE, true);

    //
    // Clear the pending interrupts.
    //

    I2SIntClear(I2S0_BASE, ulStatus);

    //
    // Determine if there was an error
    //
    if(ulStatus & I2S_INT_RXERR)
    {
        // handle the error
    }

    //
    // Handle the RX service request

```

```

//
if(ulStatus & I2S_INT_RXREQ)
{
    //
    //Retrive the data and put it in the buffer
    //
    if (Left_Sample) //check if left or right sample is next in the
FIFO
    {
        I2SRxDataGet(I2S0_BASE, &Left_Buffer);
        if((Left_Buffer<<16) < 10000) //check if sync buffer is
high or low
        {
            if(tx_ready == true) //if sync is low and transmit is
ready, send the data, clear the buffer, and set the data transmit as not
ready
            {
                tcp_write(pcb, Right_Buffer,RBS*4,1); //transmit
data
                for(i=0; i<2048; i++) //clear the buffer
                {
                    Right_Buffer[i] = 0;
                }
                RBS = 0;
                tx_ready = false; //do not transmit until sync
is high again
            }
            else //if the transmit flag is low, keep it low
            {
                tx_ready = false;
            }
        }
        else //if sync is high, set the transmit flag
        {
            tx_ready = true;
        }
        Left_Sample = false; //toggle the sample
    }
    else //right sample in FIFO
    {
        if(tx_ready == true) //if the sync is high, transmit flag
will be high and right channel should be sampled
        {
            I2SRxDataGet(I2S0_BASE, &Right_Buffer[RBS]); //sample
RBS++;
            if(RBS == 2048)
            {
                RBS = 0;
            }
        }
        else //if sync is low, discard the right channel, keep the
transmit flag low
        {
            I2SRxDataGet(I2S0_BASE, &Trash_Buffer); //put the
current FIFO in the trash buffer

```

```

        tx_ready = false;
    }
    Left_Sample = true; //toggle the sample
}

}

}

//*****
**
//
// This example demonstrates the use of the Ethernet Controller.
//
//*****
**
int
main(void)
{
    unsigned long ulUser0, ulUser1;
    unsigned char pucMACArray[8];
    struct ip_addr ip_ordinateur;
    unsigned long ulIPAddrordi;

    //
    // Set the clocking to run directly from the crystal.
    //
    ROM_SysCtlClockSet(SYSCTL_SYSDIV_1 | SYSCTL_USE_OSC | SYSCTL_OSC_MAIN |
        SYSCTL_XTAL_16MHZ);

    //
    // Initialize the UART.
    //
    /*
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
    GPIOPinConfigure(GPIO_PA0_U0RX);
    GPIOPinConfigure(GPIO_PA1_U0TX);
    ROM_GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);
    UARTStdioInit(0);
    UARTprintf("\033[2JEthernet with lwIP\n");
    */

    //
    // Enable and Reset the Ethernet Controller.
    //
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_ETH);
    ROM_SysCtlPeripheralReset(SYSCTL_PERIPH_ETH);

    //
    // Configure the I2S peripheral.
    //
    //
    //
    // Enable the I2S peripheral
    //
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);

```

```

SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOD);

SysCtlPeripheralEnable(SYSCTL_PERIPH_I2S0);
SysCtlPeripheralReset(SYSCTL_PERIPH_I2S0);

//
// Enable Port F for Ethernet LEDs.
//
ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
GPIOPinConfigure(GPIO_PF2_LED1);
GPIOPinConfigure(GPIO_PF3_LED0);
GPIOPinTypeEthernetLED(GPIO_PORTF_BASE, GPIO_PIN_2 | GPIO_PIN_3);

//
//Configure the GPIO pins for I2S
//
GPIOPinConfigure(GPIO_PA2_I2S0RXSD);
GPIOPinConfigure(GPIO_PD1_I2S0RXWS);
GPIOPinConfigure(GPIO_PD0_I2S0RXSCK);
GPIOPinConfigure(GPIO_PA3_I2S0RXMCLK);

GPIOPinTypeI2S(GPIO_PORTA_BASE, GPIO_PIN_2 | GPIO_PIN_3);
GPIOPinTypeI2S(GPIO_PORTD_BASE, GPIO_PIN_0 | GPIO_PIN_1);

//
// Configure SysTick for a periodic interrupt.
//
ROM_SysTickPeriodSet(ROM_SysCtlClockGet() / SYSTICKHZ);
ROM_SysTickEnable();
ROM_SysTickIntEnable();

//
// Enable processor interrupts.
//
ROM_IntMasterEnable();

//
// Read the MAC address from the user registers.
//
ROM_FlashUserGet(&ulUser0, &ulUser1);
if((ulUser0 == 0xffffffff) || (ulUser1 == 0xffffffff))
{
    //
    // We should never get here. This is an error if the MAC address has
    // not been programmed into the device. Exit the program.
    //
    //UARTprintf("MAC Address Not Programmed!\n");
    while(1)
    {
    }
}

//
// Convert the 24/24 split MAC address from NV ram into a 32/16 split MAC
// address needed to program the hardware registers, then program the MAC
// address into the Ethernet Controller registers.
//

```

```

pucMACArray[0] = ((ulUser0 >> 0) & 0xff);
pucMACArray[1] = ((ulUser0 >> 8) & 0xff);
pucMACArray[2] = ((ulUser0 >> 16) & 0xff);
pucMACArray[3] = ((ulUser1 >> 0) & 0xff);
pucMACArray[4] = ((ulUser1 >> 8) & 0xff);
pucMACArray[5] = ((ulUser1 >> 16) & 0xff);

//
// Initialize the lwIP library, using STATIC IP.
//
lwIPInit(pucMACArray, 0xC0A80102, 0xFFFFFFFF00, 0, IPADDR_USE_STATIC);
//lwIPInit(pucMACArray, 0, 0, 0, IPADDR_USE_DHCP);

//
// Setup the device locator service.
//
LocatorInit();
LocatorMACAddrSet(pucMACArray);
LocatorAppTitleSet("EK-LM3S9B92 enet_lwip");

//
// Configure the RX format and mode. Use I2S mode with
// 16-bit compact sample format. Word size is 24 but 32
// bits are on the wire.
//
I2SRxConfigSet(I2S0_BASE, I2S_CONFIG_FORMAT_I2S |
                I2S_CONFIG_MODE_DUAL |
                I2S_CONFIG_CLK_SLAVE |
                I2S_CONFIG_SAMPLE_SIZE_24 |
                I2S_CONFIG_WIRE_SIZE_32);
//
// Set the RX FIFO limit to trigger when there are 4 or fewer
// samples left in the FIFO.
//
I2SRxFIFOLimitSet(I2S0_BASE, 4);

//
// Set up the master clock source to use the master clock
// from an external pin.
//
I2SMasterClockSelect(I2S0_BASE, I2S_RX_MCLK_EXT);

//
// Setup and initialize connection between Stellaris and PC.
//
pcb=tcp_new(); // make a new thread
ulIPAddrordi=0xC0A80101; // address of the PC
tcp_bind(pcb,IP_ADDR_ANY,2000); // bind to a port
ip_ordinateur.addr =htonl(ulIPAddrordi);

//
// Go to callback function upon connection to PC.
//
tcp_connect(pcb,&ip_ordinateur,2000,client_connected);

//
// Clear out all pending interrupts.

```

```

//
I2SIntClear(I2S0_BASE, I2S_INT_RXERR | I2S_INT_RXREQ );

//
//Register my interrupt handler with the I2S controller
//
I2SIntRegister(I2S0_BASE, My_I2S_Int);

//
// Enable the interrupts for error and service request. Also,
// because the interrupt vector was allocated at compile-time, the
// peripheral interrupt must be enabled on the master controller.
//
I2SIntEnable(I2S0_BASE, I2S_INT_RXERR | I2S_INT_RXREQ);
IntEnable(INT_I2S0);

//
// Finally, the I2S receiver must be enabled so it can
// start receiving data.
//
I2SRxEnable(I2S0_BASE);

//
// Loop forever. All the work is done in interrupt handlers.
//
while(1)
{
}
}

```

PC C# Code

Complex.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

/*
 *
 * Class for complex arithmetic
 *
 */

namespace _480_Project_Testing
{
    class complex
    {
        //Initialize values for both the real and imaginary components
        public double re = 0.0;
        public double im = 0.0;

        //Empty constructor
        public complex()
        {
        }
    }
}

```

```

//Overloaded constructor
public complex(double re, double im)
{
    this.re = re;
    this.im = im;
}

//Return string of complex value
public string ToString()
{
    string data = re.ToString() + " " + im.ToString() + "i";
    return data;
}

//Convert from polar to rectangular
public static complex from_polar(double r, double radians)
{
    complex data = new complex(r * Math.Cos(radians), r *
Math.Sin(radians));
    return data;
}

//Override addition operator
public static complex operator +(complex a, complex b)
{
    complex data = new complex(a.re + b.re, a.im + b.im);
    return data;
}

//Override subtraction operator
public static complex operator -(complex a, complex b)
{
    complex data = new complex(a.re - b.re, a.im - b.im);
    return data;
}

//Override multiplication operator
public static complex operator *(complex a, complex b)
{
    complex data = new complex((a.re * b.re) - (a.im * b.im), (a.re *
b.im) + (a.im * b.re));
    return data;
}

//Return magnitude of complex number
public double magnitude
{
    get
    {
        return Math.Sqrt(Math.Pow(re, 2) + Math.Pow(im, 2));
    }
}

//Return phase of complex number
public double phase
{

```

```

        get
        {
            return Math.Atan(im / re);
        }
    }
}

```

Form1.cs

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Threading;
using System.Net;
using System.Net.Sockets;

namespace _480_Project_Testing
{
    public partial class Form1 : Form
    {
        //      AudioTimeData time_data;
        //      AudioFrequencyData frequency_data;

        AudioData test_wav = new AudioData("hallwayTest1.wav");

        public Form1()
        {
            long first_val = 200232342;
            double test_val = (double)first_val;
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            //ShowSyncSignal(4000);
            //ShowRadarSample();
            //ShowRawFFT();
            //ShowFilteredFFT();
            ShowSubtractedFFT();
            //CompareTwoPulse();
            //ShowLimitedFilter(200);
            //ShowTwoPulse();

            WindowFreqSamp(10);
        }

        //Show the sync signal pulled directly from the audio data
        private void ShowSyncSignal(int num_points)

```

```

    {
        int i = 0;

        //Array to hold left channel data
        double[] left_data = new double[num_points];

        for (i = 0; i < left_data.Length; i++)
        {
            //Sample data returned as 2 element array. First element
holds left channel sample
            left_data[i] = test_wav.GetNextSample()[0];
        }

        //Plot the data
        ShowOnChart(left_data);
    }

    //Show the radar signal pulled directly from the audio data
    private void ShowRadarSignal(int num_points)
    {
        int i = 0;

        //Array to hold right channel data
        double[] right_data = new double[num_points];

        for (i = 0; i < right_data.Length; i++)
        {
            //Sample data returned as 2 element array. First element
holds right channel sample
            right_data[i] = test_wav.GetNextSample()[1];
        }

        //Plot the data
        ShowOnChart(right_data);
    }

    //Show sample of time-domain radar data taken while the sync signal
is high
    private void ShowRadarSample()
    {
        //Get radar data from the audio sample
        double[] radar_sample = test_wav.GetRadarData();

        //Plot the data
        ShowOnChart(radar_sample);
    }

    //Show the non-averaged FFT data of a radar sample
    private void ShowRawFFT()
    {
        //Store the time-domain data with other info related to the audio
file
        AudioTimeData td1 = new AudioTimeData(test_wav.GetRadarData());

        //Take the FFT of the data
        AudioFrequencyData fd1 = new AudioFrequencyData(td1);
    }
}

```

```

        //Plot the data
        ShowOnChart(fd1.data);
    }

    //Show the FFT using averaging between data points
    private void ShowFilteredFFT()
    {
        //Get one full sample and take the FFT
        AudioTimeData td1 = new AudioTimeData(test_wav.GetRadarData());
        AudioFrequencyData fd1 = new AudioFrequencyData(td1);

        //Average the data points
        double[] filtered_data = FilterSignal(fd1.data);

        //Plot the resulting averaged data
        ShowOnChart(filtered_data);
    }

    private void ShowSubtractedFFT()
    {
        int i = 0;

        AudioTimeData td1 = new AudioTimeData(test_wav.GetRadarData());
        AudioFrequencyData fd1 = new AudioFrequencyData(td1);

        for (i = 0; i < 50; i++)
        {
            test_wav.GetRadarData();
        }

        AudioTimeData td2 = new AudioTimeData(test_wav.GetRadarData());
        AudioFrequencyData fd2 = new AudioFrequencyData(td2);

        double[] sub_data = new double[fd2.data.Length];

        for (i = 0; i < sub_data.Length; i++)
        {
            sub_data[i] = (fd2.data[i] - fd1.data[i]);
        }

        sub_data = FilterSignal(FilterSignal(sub_data));

        chart1.Series[0].Points.DataBindY(sub_data);
    }

    //Two-Pulse Cancellation Output
    private double[] TwoPulseCanceledSignal()
    {
        int i = 0;

        //Grab first sample of radar data
        AudioTimeData td1 = new AudioTimeData(test_wav.GetRadarData());
        AudioFrequencyData fd1 = new AudioFrequencyData(td1);

        //Grab the next immediate sample
        AudioTimeData td2 = new AudioTimeData(test_wav.GetRadarData());
    }

```

```

        AudioFrequencyData fd2 = new AudioFrequencyData(td2);

        //Create a new array to store the subtraction of each sample
        double[] sub_data = new double[fd2.data.Length];

        //Subtract the first sample from the second sample
        for (i = 0; i < sub_data.Length; i++)
        {
            sub_data[i] = fd2.data[i] - fd1.data[i];
        }

        //Average the data points
        sub_data = FilterSignal(sub_data);

        //Return this data
        return sub_data;
    }

    //Plot two-pulse cancelled signal
    private void ShowTwoPulse()
    {
        ShowOnChart(TwoPulseCanceledSignal());
    }

    //Compare results of two two-pulse cancelled signals
    private void CompareTwoPulse()
    {
        int i = 0;

        //Get first two-pulse cancelled signal
        double[] pulse1 = TwoPulseCanceledSignal();

        //Delay by 1 second
        for (i = 0; i < 25; i++)
        {
            test_wav.GetRadarData();
        }

        //Get the next two-pulse cancelled signal
        double[] pulse2 = TwoPulseCanceledSignal();

        //Array to store output data
        double[] sig_out = new double[pulse1.Length];

        //Subtract the two signals
        for (i = 0; i < sig_out.Length; i++)
        {
            sig_out[i] = pulse2[i] - pulse1[i];
        }

        //Filter the comparison
        sig_out = FilterSignal(sig_out);

        //Plot the comparison
        ShowOnChart(sig_out);
    }
}

```

```

//Show subset of data of filtered output
private void ShowLimitedFilter(int length)
{
    //Array to hold data points
    double[] data = new double[length];

    //Get radar sample
    AudioTimeData td1 = new AudioTimeData(test_wav.GetRadarData());
    AudioFrequencyData fd1 = new AudioFrequencyData(td1);

    //Array to hold filtered data signal
    double[] filtered_data = FilterSignal(fd1.data);

    //Copy filtered data into output array
    for (int i = 0; i < length; i++)
    {
        data[i] = filtered_data[i];
    }

    //Plot limited array
    ShowOnChart(data);
}

//Use moving average window to do basic filtering of data
private double[] FilterSignal(double[] data)
{
    //Array to hold return data
    double[] return_data = new double[data.Length];

    //Copy first data point
    return_data[0] = data[0];

    //Take average of current data point and previous data point
    for (int i = 1; i < return_data.Length; i++)
    {
        return_data[i] = (data[i] + data[i - 1]) / 2;
    }

    //Return the array of averaged samples
    return return_data;
}

//Show average window of frequency domain samples
private void WindowFreqSamp(int window_size)
{
    int i = 0;

    //Get first set of data
    AudioTimeData td1 = new AudioTimeData(test_wav.GetRadarData());
    AudioFrequencyData fd1 = new AudioFrequencyData(td1);
    //Use average window of window_size
    double[] t1 = FFTBucket(fd1.data, window_size);

    //Delay 1 second
    for (i = 0; i < 25; i++)
    {

```

```

        test_wav.GetRadarData();
    }

    //Get second set of data
    AudioTimeData td2 = new AudioTimeData(test_wav.GetRadarData());
    AudioFrequencyData fd2 = new AudioFrequencyData(td2);
    //Use average window of window_size
    double[] t2 = FFTBucket(fd2.data, window_size);

    //Array to store output data
    double[] data = new double[t2.Length];

    //Compare points
    for (i = 0; i < data.Length; i++)
    {
        data[i] = t2[i] - t1[i];
    }

    //Plot data
    ShowOnChart(data);
}

//Window averaging for data points
private double[] FFTBucket(double[] data, int window_size)
{
    //New return array
    double[] return_data = new double[data.Length / window_size];
    int i = 0;

    //Populate array
    for (i = 0; i < return_data.Length; i++)
    {
        //Initialize the data point to be stored
        return_data[i] = 0;
        //Take average of window_size points
        for (int j = 0; j < window_size; j++)
        {
            return_data[i] += data[j + i * window_size] /
window_size;
        }
    }

    //Return array of averaged points
    return return_data;
}

//Find array index containing maximum data point
private int FindMaxIndex(double[] data)
{
    int max = 0;

    //Find the maximum point
    for (int i = 0; i < data.Length; i++)
    {
        if (data[i] > data[max])
        {
            max = i;
        }
    }
}

```

```

        }
    }

    //Return the found value
    return max;
}

//Plot data
private void ShowOnChart(double[] data)
{
    chart1.Series[0].Points.DataBindY(data);
}

//Used for testing other functionality
private void button2_Click(object sender, EventArgs e)
{
    int i = 0;
    SineGen sine_data_1 = new SineGen(937.5, 0.0);
    //SineGen sine_data_2 = new SineGen(1345.8251953125, 0.0);

    AudioTimeData time_data_1 = new AudioTimeData(sine_data_1.data);
    //AudioTimeData time_data_2 = new
AudioTimeData(sine_data_2.data);

    //AudioFrequencyData frequency_data_1 = new
AudioFrequencyData(time_data_1);

    complex[] time_1 = new complex[sine_data_1.data.Length];

    for (i = 0; i < time_1.Length; i++)
    {
        time_1[i] = new complex(sine_data_1.data[i], 0);
    }

    complex[] freq_1 = fourier.FFT(time_1);
    double[] freq_1_mag = new double[500];

    for (i = 0; i < freq_1_mag.Length; i++)
    {
        freq_1_mag[i] = freq_1[i].magnitude;
    }

    int max = FindMaxIndex(freq_1_mag);

    int max_freq = (int) ((float)max * (float)96000 / (float)2048);
    frequency_box.Text = max_freq.ToString();

    //AudioFrequencyData frequency_data_2 = new
AudioFrequencyData(time_data_2);

    /*
    double[] result = new double[frequency_data_1.data.Length];

    int i = 0;

    for (i = 0; i < result.Length; i++)

```

```

        {
            result[i] = frequency_data_2.data[i] -
frequency_data_1.data[i];
        }
        */
        chart1.Series[0].Points.DataBindY(freq_1_mag);
    }

    private void button3_Click(object sender, EventArgs e)
    {

    }

}
}

```

fourier.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
/*
 *
 *
 * This is an adaptation of code found at
 * http://en.literateprograms.org/Cooley-Tukey\_FFT\_algorithm\_%28C%29
 *
 *
 *
 *
 *
 *
 */
namespace _480_Project_Testing
{
    class fourier
    {
        //Calculate complex DFT of N samples
        public static complex[] DFT(complex[] x)
        {
            //Store number of samples
            int N = x.Length;

            //Create output array of size N
            complex[] X = new complex[N];

            //Calculate for each point from 0 to N
            for (int k = 0; k < N; k++)
            {
                //Initialize the new data point
                X[k] = new complex(0, 0);

                //Sum samples from 0 to N
                for (int n = 0; n < N; n++)
                {
                    //Multiply the n data point with a complex number

```

```

        complex temp = complex.from_polar(1, -2 * Math.PI * n * k
/ N);

        temp *= x[n];
        X[k] += temp;
    }
}

return X;
}

//Calculate complex FFT of N samples
public static complex[] FFT(complex[] x)
{
    //Store number of samples
    int N = x.Length;

    //Create output arraya of size N
    complex[] X = new complex[N];

    //If only one sample in set, return only that sample
    if (N == 1)
    {
        X[0] = x[0];
        return X;
    }

    int k;

    //Initalize even and odd arrays
    complex[] e = new complex[N / 2];
    complex[] d = new complex[N / 2];

    for (k = 0; k < N / 2; k++)
    {
        //Assign even numbered samples to even array
        e[k] = x[2 * k];

        //Assign odd numbered samples to odd array
        d[k] = x[2 * k + 1];
    }

    //Arrays to hold transforms of even and odd arrays
    complex[] D, E;

    //Calculate transform of odd samples
    D = FFT(d);

    //Calculate transform of even samples
    E = FFT(e);

    //Multiply odd samples by complex number
    for (k = 0; k < N / 2; k++)
    {
        complex temp = complex.from_polar(1, -2 * Math.PI * k / N);
        D[k] *= temp;
    }
}

```

```

        //First N/2 samples are even + (odd * complex)
        //Second N/2 samples are even - (odd * complex)
        for (k = 0; k < N / 2; k++)
        {
            X[k] = E[k] + D[k];
            X[k + N / 2] = E[k] - D[k];
        }

        //Return the transformed array
        return X;
    }
}

```

SineGen.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

/*
 *
 * Class to generate sinusoid for testing of FFT
 *
 */

namespace _480_Project_Testing
{
    class SineGen
    {
        //Initialize variables
        public double frequency;
        public double t0;
        public double[] data;
        public uint sample_rate = 96000;
        public uint samples = 2048;
        public double dt;

        //Constructor
        public SineGen(double frequency, double t0 = 0.0)
        {
            //Store frequency and starting point
            this.frequency = frequency;
            this.t0 = t0;

            //Store time between each data point
            dt = 1 / ((double)sample_rate);

            //Initialize the array
            data = new double[samples];
        }
    }
}

```

```
//Calculate and store each data point
for (uint i = 0; i < samples; i++)
{
    data[i] = Math.Sin(2 * Math.PI * frequency * i * (dt + t0));
}
}
}
```

Appendix 4: GANTT Charts

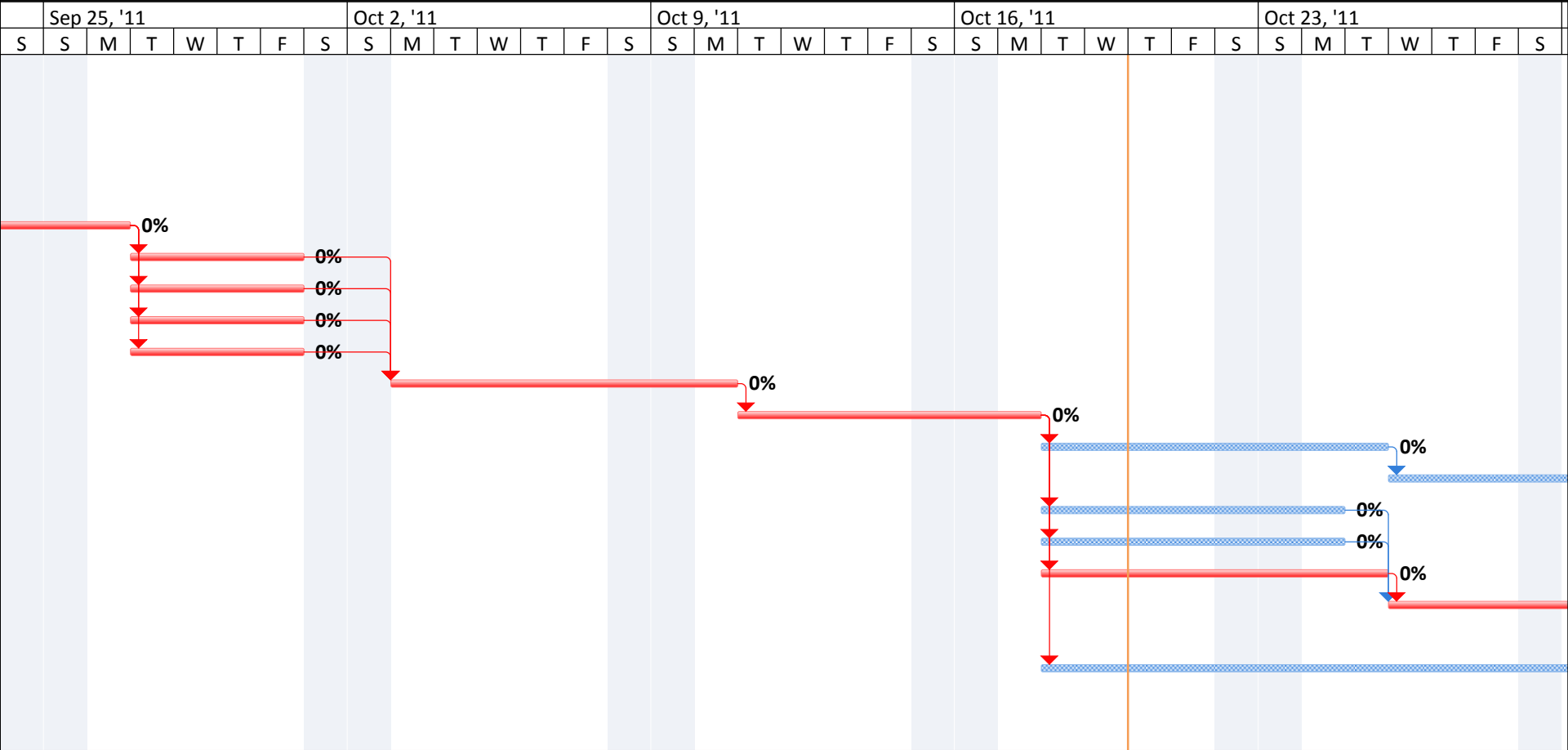
Original GANTT Chart

ID		Task Name	Duration	Start	Sep 11, '11														Sep 18, '11					
					T	W	T	F	S	S	M	T	W	T	F	S	S	M	T	W	T	F		
1		Meet with faculty facilitator	1 day	Wed 9/7/11																				
2		Conference call with sponsor	2 days	Wed 9/7/11																				
3		Team research of project	4 days	Fri 9/9/11																				
4		Specify design criteria	2 days	Thu 9/15/11																				
5		Research components required	3 days	Mon 9/19/11																				
6		Order components	3 days	Thu 9/22/11																				
7		Assemble ramp generator	4 days	Tue 9/27/11																				
8		Assemble filter and amplifier	4 days	Tue 9/27/11																				
9		Assemble RF components	4 days	Tue 9/27/11																				
10		Assemble antenna	4 days	Tue 9/27/11																				
11		Test RF components	6 days	Mon 10/3/11																				
12		Test complete system	5 days	Tue 10/11/11																				
13		Layout ADC breakout board	6 days	Tue 10/18/11																				
14		Populate ADC breakout board	6 days	Wed 10/26/11																				
15		Communication between uC and ADC	5 days	Tue 10/18/11																				
16		PC server application to receive uC data	5 days	Tue 10/18/11																				
17		uC client to connect to PC	6 days	Tue 10/18/11																				
18		Forward raw ADC data to PC	4 days	Wed 10/26/11																				
19		Test ADC with signal generator	1 day	Tue 11/1/11																				
20		Develop FFT on PC	10 days	Tue 10/18/11																				
21		Test FFT on PC with known data	3 days	Wed 11/2/11																				
22		Calculate range & speed from FFT	4 days	Mon 11/7/11																				

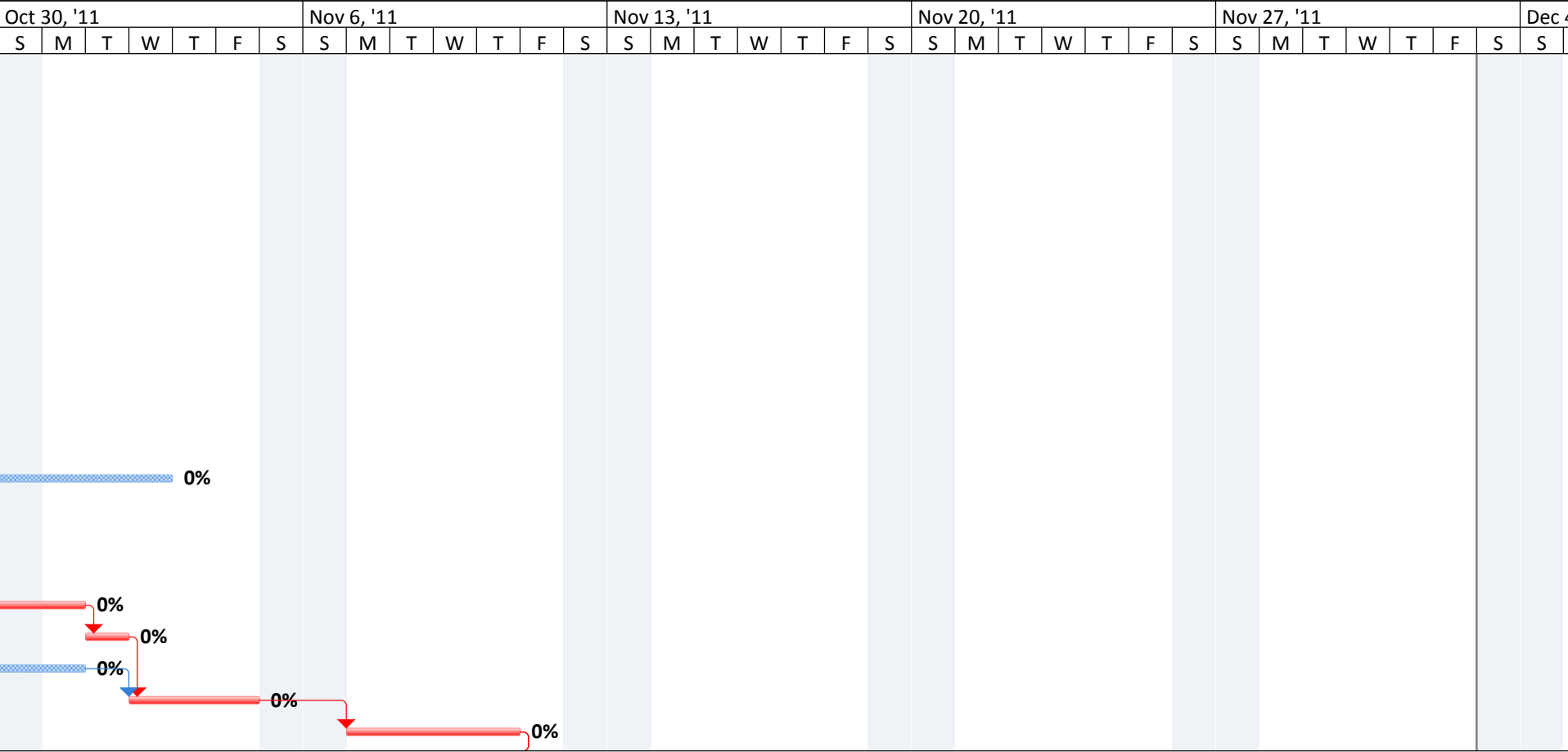
Critical		Finish-only		Manual Summary	
Critical Split		Duration-only		Project Summary	
Critical Progress		Baseline		External Tasks	
Task		Baseline Split		External Milestone	
Split		Baseline Milestone		Inactive Task	
Task Progress		Milestone		Inactive Milestone	
Manual Task		Summary Progress		Inactive Summary	
Start-only		Summary		Deadline	

ID		Task Name	Duration	Start						Sep 11, '11							Sep 18, '11						
					T	W	T	F	S	S	M	T	W	T	F	S	S	M	T	W	T	F	
23		Send alert email from PC	3 days	Fri 11/11/11																			
24		Port FFT to uC	3 days	Fri 11/11/11																			
25		Test FFT on uC	2 days	Wed 11/16/11																			
26		Forward FFT data to PC	1 day	Fri 11/18/11																			
27		Calculate range & speed on uC	2 days	Fri 11/18/11																			
28		Send alert message to PC	4 days	Tue 11/22/11																			
29	 	Layout uC and ADC board	10 days	Wed 10/26/11																			
30	 	Layout ramp & filter board	10 days	Wed 10/26/11																			
31		Populate uC and ADC board	7 days	Wed 11/9/11																			
32		Populate ramp & filter board	7 days	Wed 11/9/11																			
33		Test complete system	5 days	Mon 11/28/11																			

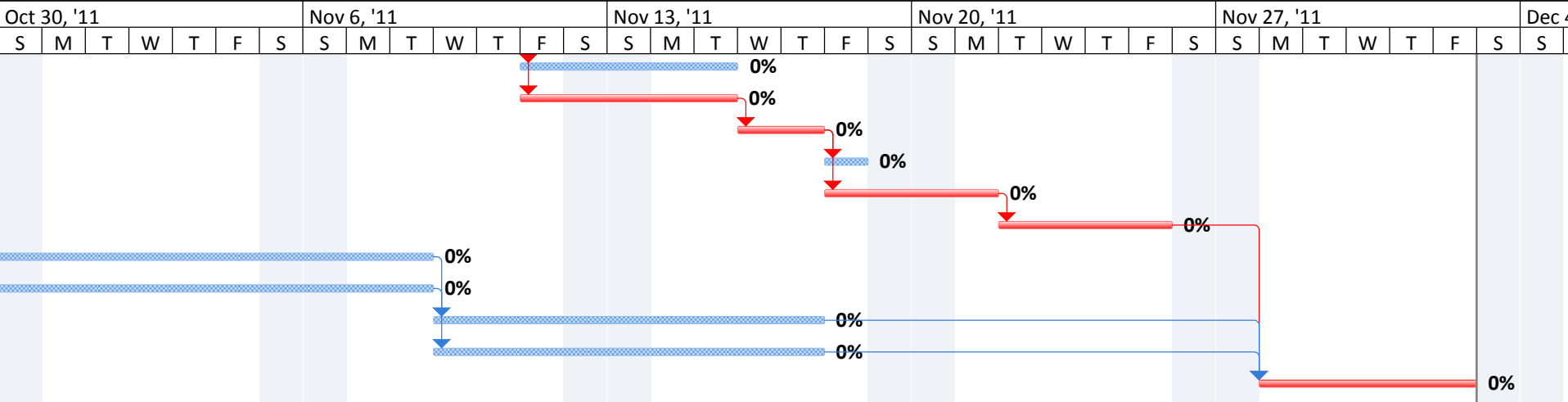
Critical		Finish-only		Manual Summary	
Critical Split		Duration-only		Project Summary	
Critical Progress		Baseline		External Tasks	
Task		Baseline Split		External Milestone	
Split		Baseline Milestone		Inactive Task	
Task Progress		Milestone		Inactive Milestone	
Manual Task		Summary Progress		Inactive Summary	
Start-only		Summary		Deadline	



Critical		Finish-only		Manual Summary	
Critical Split		Duration-only		Project Summary	
Critical Progress		Baseline		External Tasks	
Task		Baseline Split		External Milestone	
Split		Baseline Milestone		Inactive Task	
Task Progress		Milestone		Inactive Milestone	
Manual Task		Summary Progress		Inactive Summary	
Start-only		Summary		Deadline	



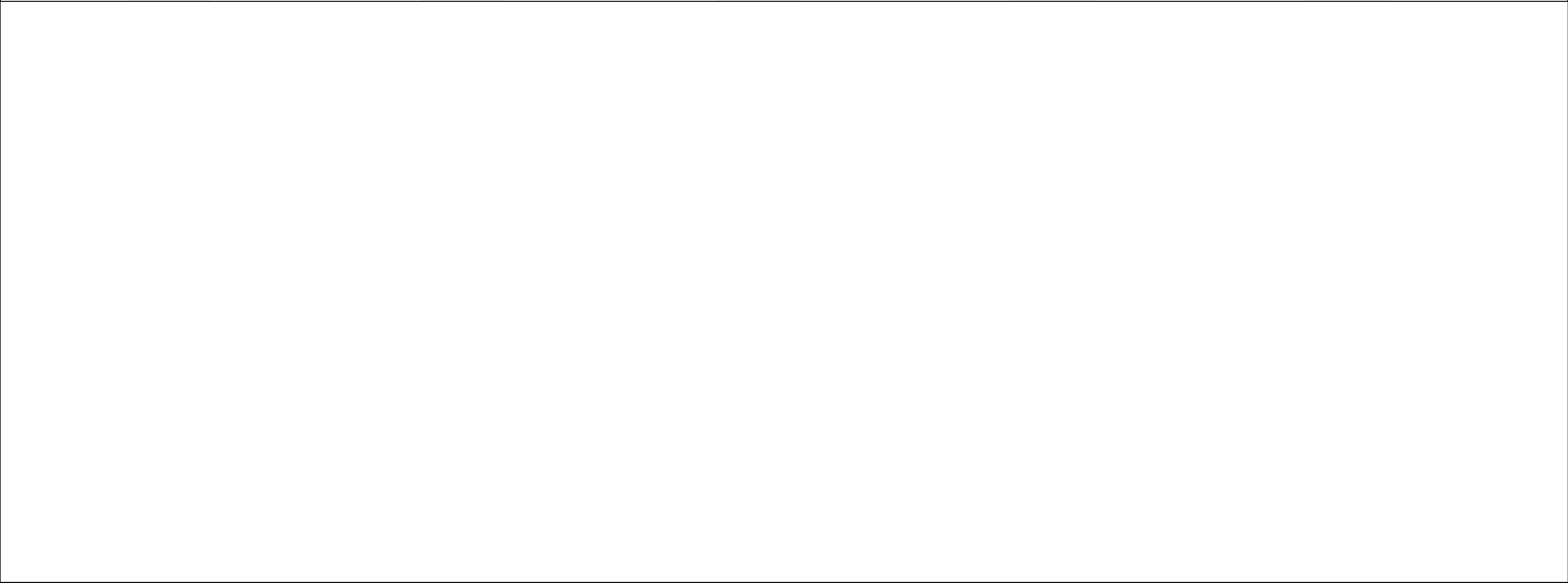
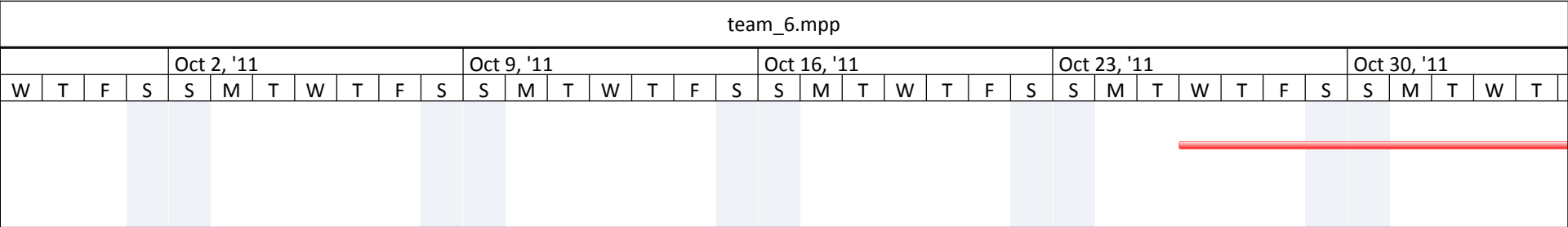
Critical		Finish-only		Manual Summary	
Critical Split		Duration-only		Project Summary	
Critical Progress		Baseline		External Tasks	
Task		Baseline Split		External Milestone	
Split		Baseline Milestone		Inactive Task	
Task Progress		Milestone		Inactive Milestone	
Manual Task		Summary Progress		Inactive Summary	
Start-only		Summary		Deadline	



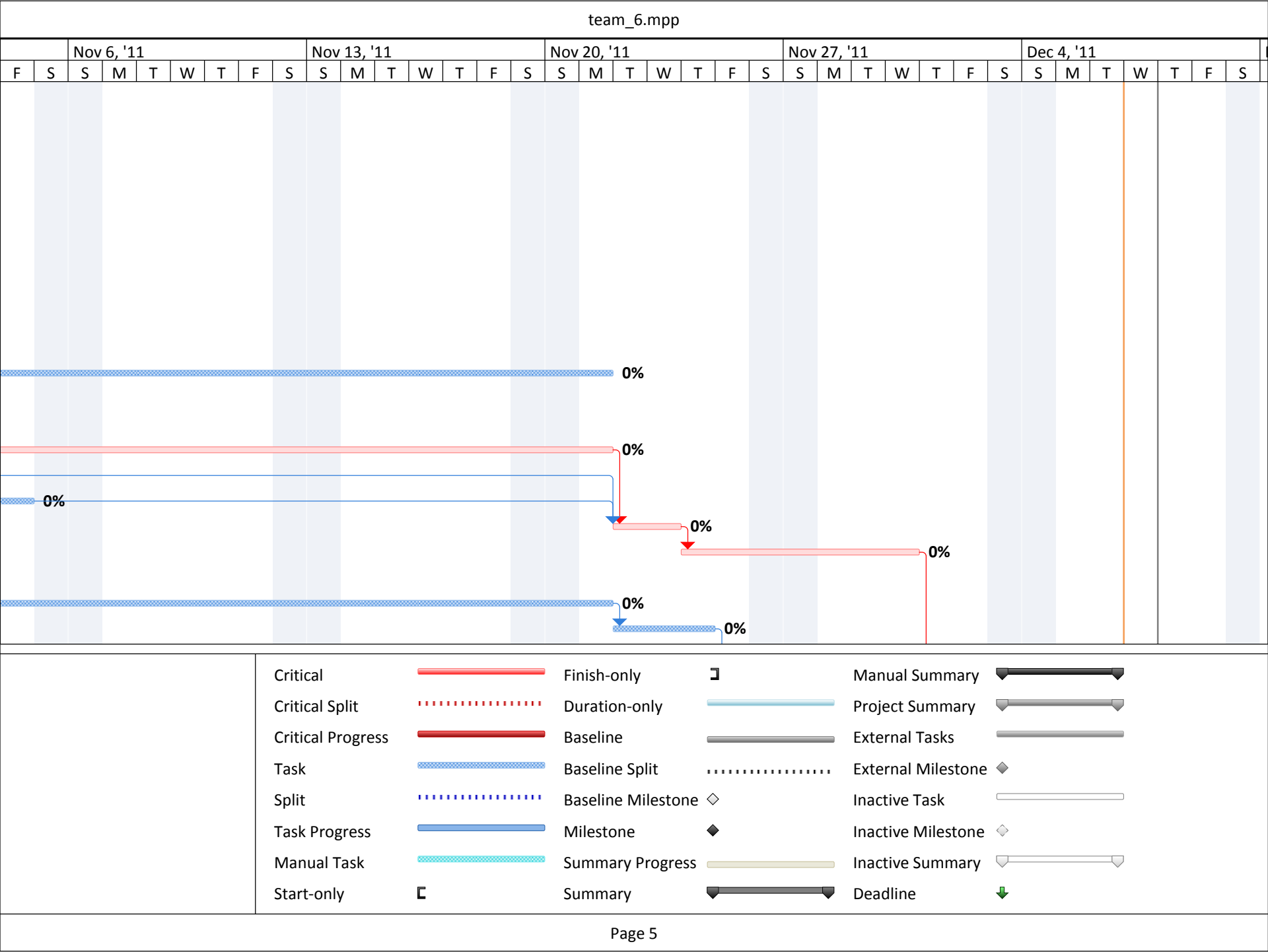
Critical		Finish-only		Manual Summary	
Critical Split		Duration-only		Project Summary	
Critical Progress		Baseline		External Tasks	
Task		Baseline Split		External Milestone	
Split		Baseline Milestone		Inactive Task	
Task Progress		Milestone		Inactive Milestone	
Manual Task		Summary Progress		Inactive Summary	
Start-only		Summary		Deadline	

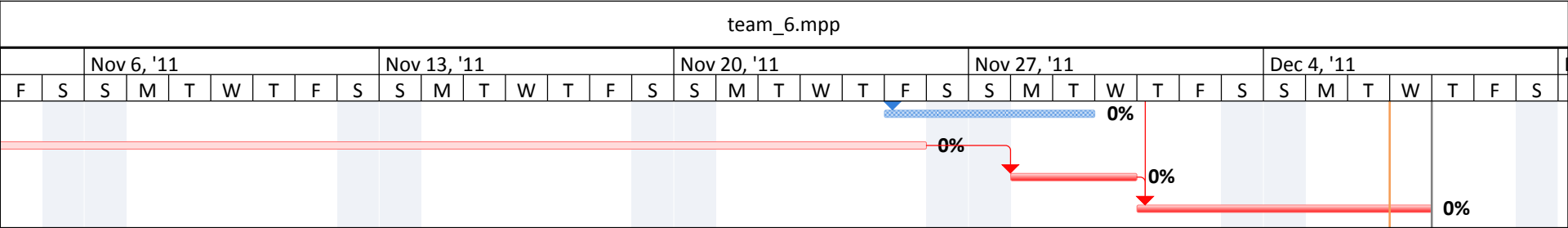
Final GANTT Chart

Critical		Finish-only		Manual Summary	
Critical Split		Duration-only		Project Summary	
Critical Progress		Baseline		External Tasks	
Task		Baseline Split		External Milestone	
Split		Baseline Milestone		Inactive Task	
Task Progress		Milestone		Inactive Milestone	
Manual Task		Summary Progress		Inactive Summary	
Start-only		Summary		Deadline	



Critical		Finish-only		Manual Summary	
Critical Split		Duration-only		Project Summary	
Critical Progress		Baseline		External Tasks	
Task		Baseline Split		External Milestone	
Split		Baseline Milestone		Inactive Task	
Task Progress		Milestone		Inactive Milestone	
Manual Task		Summary Progress		Inactive Summary	
Start-only		Summary		Deadline	





Critical		Finish-only		Manual Summary	
Critical Split		Duration-only		Project Summary	
Critical Progress		Baseline		External Tasks	
Task		Baseline Split		External Milestone	
Split		Baseline Milestone		Inactive Task	
Task Progress		Milestone		Inactive Milestone	
Manual Task		Summary Progress		Inactive Summary	
Start-only		Summary		Deadline	