Acoustic Pinger Locator (APL) Subsystem

Senior Design, Group 19: Zhen Cai, Jonathan Mohlenhoff,
Cassondra Puklavage [1]

December 14, 2009

[1]Sponsored by the Robotics Club at the University of Central Florida

Contents

Executive Summary

The overall goal of this project is to have some device be able to pick up underwater acoustics from a pinger and triangulate a relative position from the device to the pinger. After this prototype is designed, constructed, and tested it will be utilized on the Autonomous Underwater Vehicle (AUV) of the Robotics Club at UCF. This system is being developed specifically for the 2010 Association for Unmanned Vehicle Systems International (AUVSI) and Office of Naval Research (ONR) Autonomous Underwater Vehicle Competition in San Diego, California. The competition involves several missions the robot must autonomously complete, one of them including the pinger localization.

The preliminary design consists of an array of four or five hydrophones arranged in a particular way to allow for phase analysis on the signals to triangulate a heading, depth, and distance. Several different techniques were researched to achieve acoustic localization, a simulator was created to perform some preliminary analysis on these different methods to facilitate a final design. Another important aspect of this project is to receive the acoustic pinger signals from the underwater environment. The proposed method of achieving this goal is a passive hydrophone array mounted on the vehicle that converts acoustic energy into electrical energy. These attenuated signals are then conditioned by first pre-amplification with a variable gain to a utilizable signal. The signal is then filtered at the pinger's specified frequency using a third order Butterworth bandpass filter to remove unwanted noise. The final analog signal processing stage needs to adjust the signal for the appropriate range of the analog to digital converter.

The analog signals are captured simultaneously from the analog to digital converter at a sampling rate that exceeds the Nyquist sampling theorem to recieve unaliased digital signal data. The data is then processed by a field programmable gate array (FPGA) which contains the digital signal processing to calculate the acoustic localization of the pinger. The first step is to analyze each individual hydrophone signal by performing the Fast Fourier Transform (FFT) on the signals and determine relative phase differences between the hydrophone signals. The phase differential information is then used in the mathematical multilateration technique implemented on the FPGA to calculate the pingers location. The pingers heading, depth, and distance relative to the hydrophone array is then communicated over serial to the AUV's host computer which is then used to navigate to the pinger in the competition.

Chapter 1

Acoustic Pinger Locator Project Overview

1.1 Introduction

The purpose of this project is to create an independent system for an autonomous underwater vehicle (AUV) that can listen for and locate an underwater pinger. This system would send heading and depth information to the AUV, enabling the AUV to travel to the pinger. This project includes knowledge and implementation of several areas of electrical and computer engineering. The system will have to use analog and digital hardware, along with software and mathematics to decipher the incoming information. This chapter provides information about why this project was chosen and some general requirements.

1.2 Purpose

The organization AUVSI (Association for Unmanned Vehicle Systems International) holds several autonomous robotic student competitions every year. These competitions include building robots that perform in ground, aerial, surface and underwater environments. The main purpose of these competitions are to give students an opportunity to work and understand unmanned vehicles and introduce them to military and industry leaders. Every year, for each competition, a new set of rules are released that contain vehicle specifications and what the missions will be. It is then up to the participating student groups to design systems that can perform these missions autonomously. The Robotics Club at the University of Central Florida enters several of the AUVSI competitions each year. For the past 7 years the AUV Competition has been attended. One of the most important missions has been to locate and travel to an underwater pinger. The AUV team has requested that a new system be created from scratch that can perform this important task.

1.3 AUVSI and ONR's 13th International AUV Competition

AUVSI and ONR (Office of Naval Research) hold and international AUV competition each year, specifically aimed towards students. The competition is held every August at the Point Loma Naval Base in San Diego, California. The Pacific SPAWAR center training facility is where the competition is set up. This location has a large pool where the navy tests other small submarines and the affects of RADAR. In general, about 30 teams from the US, Canada, Japan and Korea compete. Several of the universities from the US include MIT, Cornell, VT and UF. Over a five day period students compete to gain the most points through static judging (paper, website, t-shirts), qualifier runs and the final challenge. Before this can happen rules are created that give a detailed explanation of what the course will be and what is expected of the teams.

These rules contain physical limitations for competing groups' AUVs. The vehicles must be within a 3' x 3' x 6' area. In addition teams receive points for a lighter AUV and lose points if their vehicle surpasses a specified 125 pound weight limit, ultimately being disqualified if too heavy. The rules specify other requirements that will be used for static judging including paper specifications. The final section of the rules explain the different missions the AUV will have to perform along with information about points and specific hardware that will be used to create obstacles. Here is the list of last year's missions:

- Submerge and travel through a starting gate
- Recognize orange paths along the floor at several locations in the competition area that will point the AUV to the next task
- Find and ram into a red buoy moored underwater
- Travel under two green PVC pipes that represent "barb wire"
- Shoot two torpedoes through a target
- Recognize a primary and secondary target, specified before leaving dock, and drop a marker on them
- Locate a pinger, travel to it, acquire the "brief case" then surface in an octagon on the surface

A team is awarded points for each mission there robot completes. The amount of points for each mission clearly defines what the most important tasks are to complete. An AUV that follows a path will gain 50 points for each path. On the other hand an AUV that surfaces in the correct octagon immediately gains 2000 points. What generally happens are that teams that can complete a single mission like the octagon will make it to the top three. To further help clarify the missions of the competition a layout of the test facility with the mission layout is illustrated in Figure 1.1.

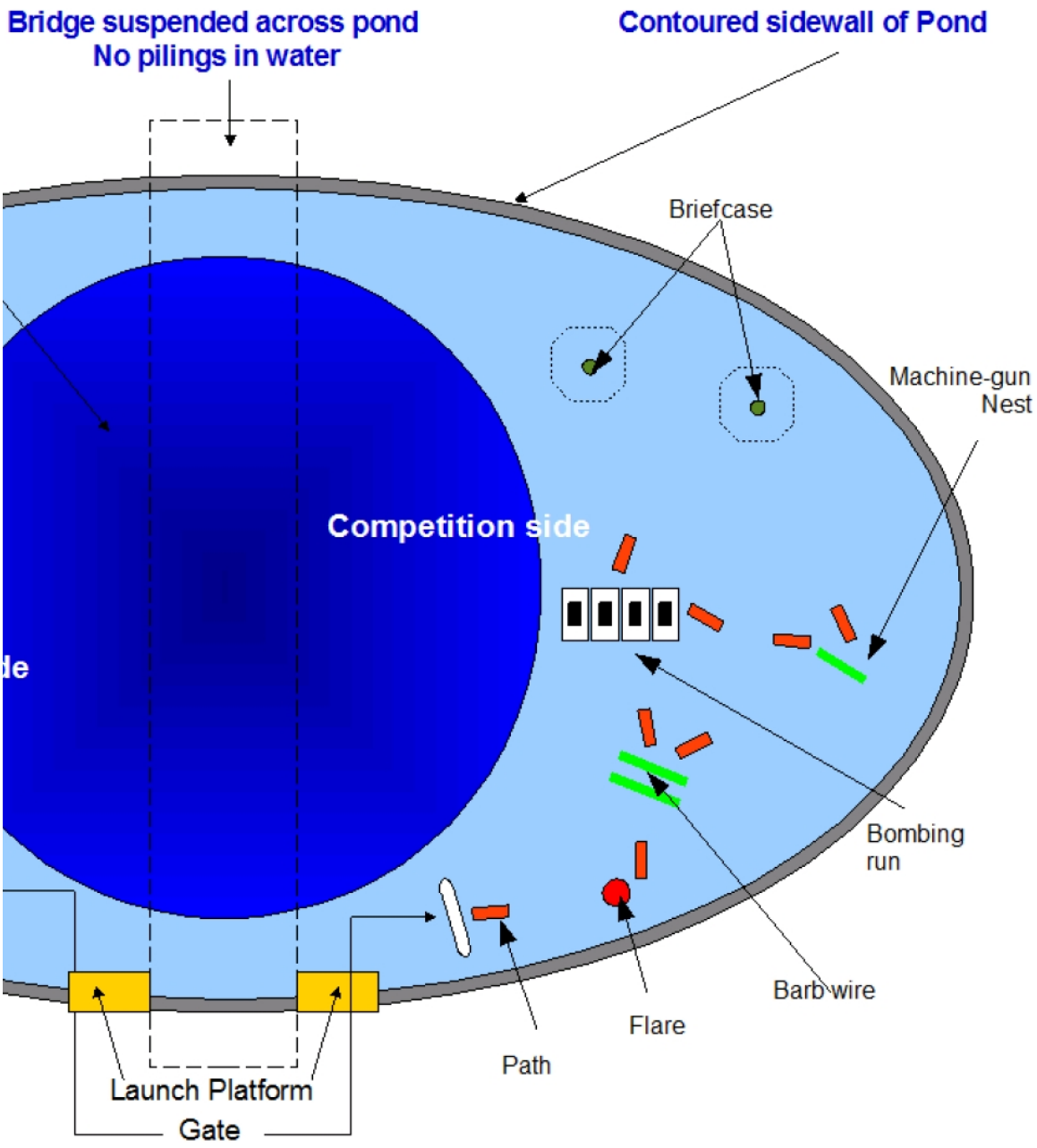


Figure 1.1: An overhead view of the competition site and mission layout. This image is reprinted with permission from the AUVSI organization, see the Appendix Section A.4.

1.4 Official Competition Specifications for the Pinger Mission

The official rules define that the pinger will emit a sine wave at anywhere between 20-30KHz. This will be done every other second in a 7ms burst. The waveform and

positioning information can be seen in Figure 1.2

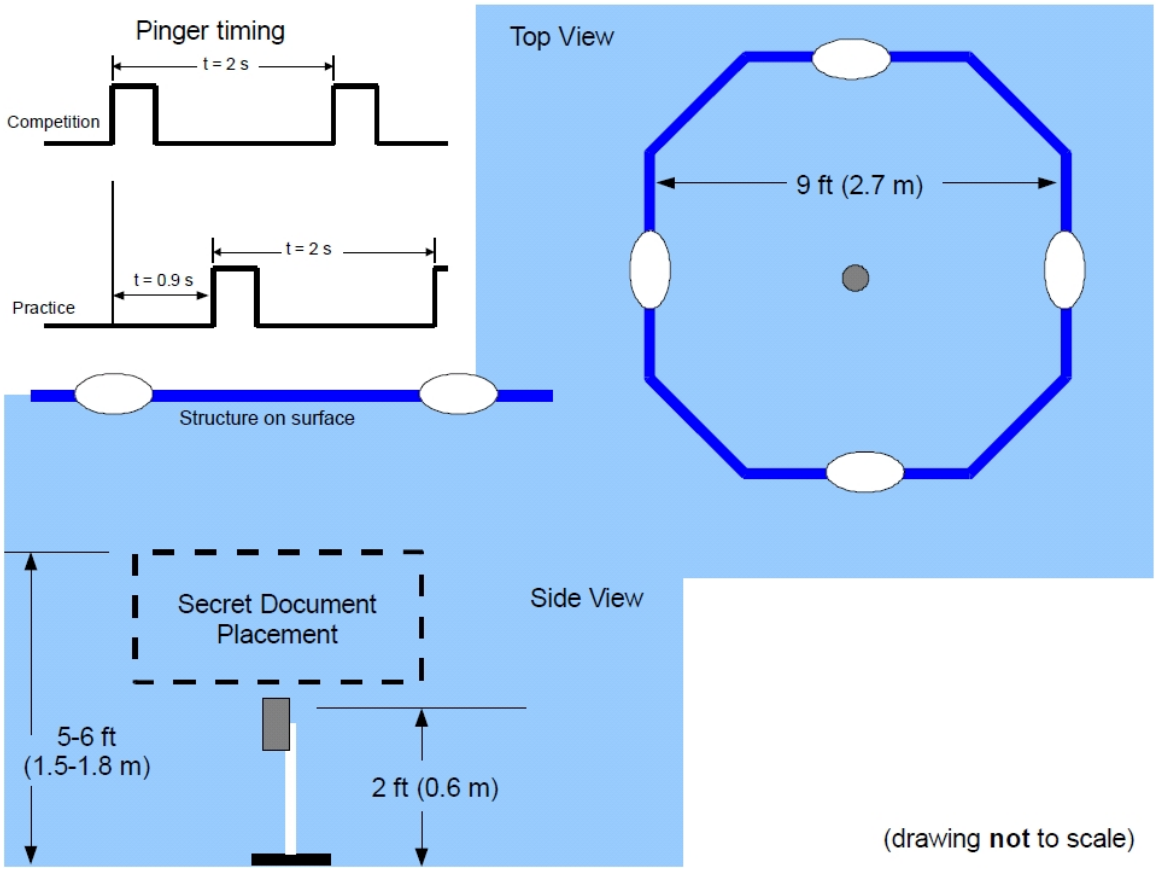


Figure 1.2: Wave form and position of pinger at competition site. This image is reprinted with permission from the AUVSI organization, see the Appendix Section A.4.

Because of these specifications the analog board that will take in, amplify and clean up the signal will need to have an adjustable filter frequency. Another concern is because the signal is sent out in 7ms bursts, a system must be put in place that can recognize when an actual signal from the pinger is being sent out and not recognize noise that is constantly present in the water.

1.5 AUV Team's Objectives

Last year the AUV team won 4th place of about 30 teams. Because of this they want to improve their vehicle, sensors and AI to place higher this year. Because the pinger locating mission is vital to winning the competition the system will be extensively worked on by this senior design group. The AUV Team plans to radically change the design of their vehicle.

1.5.1 Mechanical

The team is completely scrapping their frame from the previous year and redesigning a new one. It will consist of a single tube that holds all of the electronics. The frame that holds this acrylic tube will be made of 8020 1" x 1" extruded aluminum, allowing for mounting location all around the vehicle. To extend the AUV's capabilities 3 more Seabotix motors will be added to allow for 6 degrees of freedom including the ability to strafe. The initial design can be seen in Figure 1.3.

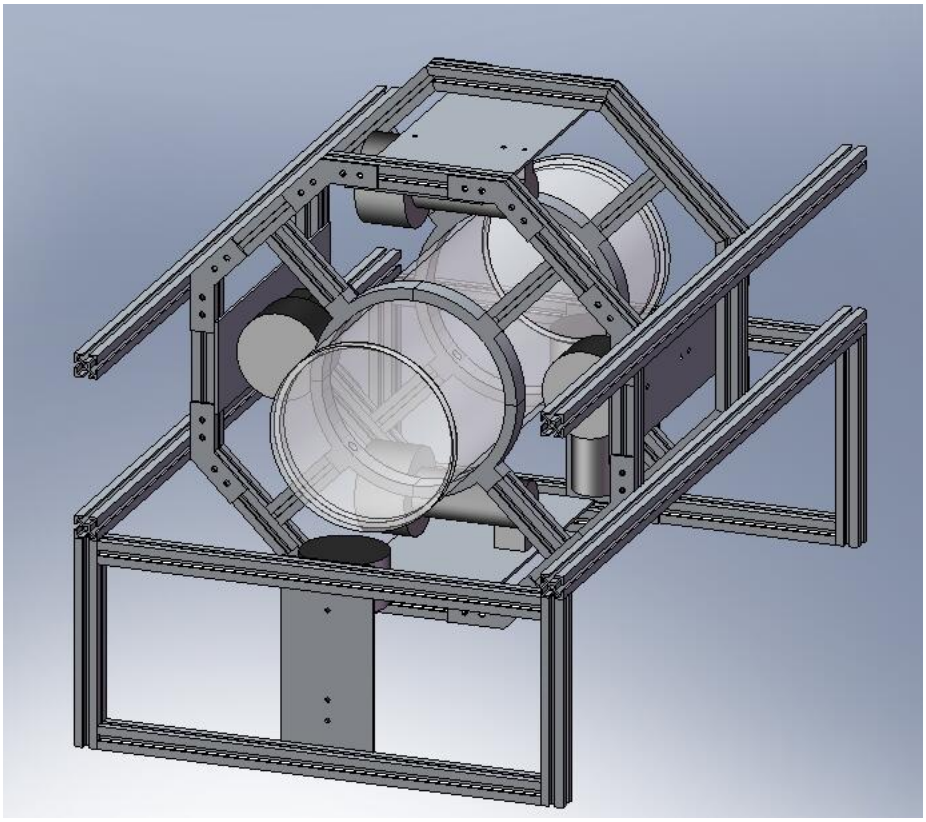


Figure 1.3: Mechanical design of the 2010 AUV. This image also shows the different mounting points for the hydrophone array.

1.5.2 Electrical

A new computer and microcontroller will be purchased to further computation capabilities. In terms of lower level electronics a power board, with enhanced shore power capabilities, will be redesigned. A new set of batteries will also be purchased so that all systems run on 22.2 V. New sensors will include a Bumblebee stereo vision camera from Point Grey and the possible implementation of a LIDAR system.

1.5.3 Software

The underlying framework will stay the same. The (Joint Architecture for Unmanned Systems) JAUS software base will be upgraded to the new JAUS SAE standards. Vision will be improved by implementing a new method that is less reliant on colors. The AI will also be redesigned to better handle unexpected problems.

1.6 Proposed Solution

To meet the demands of both the competition rules and the needs of the AUV team a system that is completely self-contained will be created. This will give the AUV team a simple interface to use where they will only have the final heading and depth given to them. An analog circuit board will be used to amplify and filter the incoming signals. From there it will be sent through high-speed analog to digital converters to an FPGA or DSP. This digital hardware will decide when to sample and may control some aspects of the analog board. Once a pinger signal has been recognized the digital hardware will take in and process this information resulting in only a heading and distance from the AUV to the pinger. Finally, this information will be sent to the AUV's computer. Ideally the underwater acoustic pinger locator system will listen for the pinger, clean the incoming signals appropriately, sample at the correct time, use mathematical methods to locate the pinger and respond by sending a simple heading and distance to the AUV.

Chapter 2

Specifications, Budget and Timeline

This chapter explains the specifications given to the Senior Design group by the AUV team. The main goal of the project will be to create a self-contained system that is able to provide a heading and possible distance from any point in the water to a specified acoustic pinger. A brief discussion of the current budget is given including a breakdown of possible items and their statuses. Finally, an in-depth explanation of our current testing goals and information about where, how and who will be helping the senior design team.

2.1 Technical Objectives

This section provides a list of all the specifications the system must meet. These specifications are derived from both the needs of the AUV team and the AUVSI competition rules. These requirements must be met to the best of the ability of the senior design team.

- Be able to operate off of a 22.2 V lithium-polymer battery
- Recognize a pinger with a frequency range of 20 - 30KHz
- Use a 4 or 5 hydrophone array
- Pre-amplify the incoming signal with the use of op-amps or similar methods with a gain of at least 400
- Use a filtering method that allows for the easy adjustment of the center frequency
- Use an analog to digital converter that samples the incoming signals simultaneously with at least 12 bits of resolution
- The analog to digital converter must sample at least 10 times the pinger frequency (12*4*1,000,000 = 6 MB/s)
- Have a working method of when to sample the incoming signals
- Perform a Fourier Transform on the incoming signals
- Use an mathematical method to obtain the heading and possibly a distance

- Send the heading and distance to the AUV

2.1.1 Goals

Here are a set of goals the senior design team will make in response to the needs of the AUV team.

- Try to keep the cost of the system down
- Allow for the easy adjustment of the filters
- Allow for the easy adjustment of the pre and post amplification gains
- Must know when to sample
- Include more accurate heading calculations
- Try to include accurate distance information
- Provide the appropriate information to the AUV
- Other than allowing the AUV to do some adjustments, make the overall system as independent as possible

2.2 Budget and Financing

This project will be completely financed by the AUV team and the robotics club. A set of five hydrophones, which are the most expensive item, were purchased several years ago by a previous AUV team. The most expensive item to be purchased this year will be a new computer for the AUV. At this point many items that can be used by the senior design team, including an FPGA, have been purchased. Here is a table describing the status of most of the components that will be used for this project:

2.3 Timeline

A gantt chart has been created that describes the timeline of this project. By the completion of this paper several components should have already been completed. The goals of this past semester included the investigation of several different areas of interesting and some designing. The first element that was investigated included mathematical methods to locate the pinger. Several implementations and a final design were decided for the analog hardware portion, which amplifies and filters the incoming signal. A preliminary investigation of digital hardware including FPGAs, microcontrollers, ADCs and DSPs was also conducted.
The next semester will begin the building, testing and integration phases. Because of the modular characteristics of the project the analog hardware, digital hardware and software can be worked on in parallel. After testing is completed integration of each component will take place followed by final testing and tweaking.

Item	Status	Finance	Approximate Cost
AUV Computer	NP	AUV Team	850.0
Hydrophone 1	P	AUV Team	1000.00
Hydrophone 2	P	AUV Team	1000.00
Hydrophone 3	P	AUV Team	1000.00
Hydrophone 4	P	AUV Team	1000.00
Hydrophone 5	P	AUV Team	1000.00
FPGA Board	P	AUV Team	150.00
ADC 1	P	AUV Team	30.00
ADC 2	P	AUV Team	30.00
Custom Filter Board	NP	Sponsorship	0.00
Board Components	NP	AUV Team	50.00
Hydrophone Mount	NP	In House	50.00
Waterproof Box	NP	AUV Team	50.00
Subconn connectors	NP	AUV Team	75.00
Total Cost: This year's Cost:			6285.00 1075.00

Figure 2.1: Table showing the estimated cost of the project.

2.4 Testing Schedule

As mentioned before, the modular nature of this system allows for the major components to be worked on in parallel. These components include the analog hardware, digital hardware and software. Each section below goes into detail about how they will be tested for each of these components. Once these individual components are tested and it is verified that they work, the different sections will be combined together and again tested. The last testing phase will be to ensure that the entire system works, from taking in raw hydrophone signals to outputting a heading and distance.

2.4.1 Analog Hardware Testing

For all these tests until the last phase a single circuit will be created for testing a single hydrophone, assuming that the results will be the same for the other three or four. The purpose of the analog hardware is to amplify and filter the incoming raw signal without introducing noise or other distortions like phase shifts. The first step in testing the analog hardware will be to set up the pre-amplification portion. Instead of first using the raw signal from a hydrophone, a function generator will be used to mimic the signal. The output will be verified through the use of an oscilloscope. If the circuit functions as necessary, the next phase of testing will begin. If the circuit fails to complete its task a new design or hardware component will be implemented and the testing process will be reevaluated.
The next analog circuit to be tested will be the filtering portion. This will be done in a similar way, in that, only one circuit will be put together. The circuit will then be

tested by applying a 20 to 30KHz sine wave with a function generator. If the circuit is able to properly attenuate any unwanted frequencies while not distorting the wanted signal, it will be considered that it works properly. If a programmable filter is used, more intense testing will take place. An example of this includes generating a 22kHz signal and making sure close frequencies like 24kHz are being attenuated enough. The center frequency will also be changed on the fly to test the capabilities of the programmable filter. If this circuit works well, the next phase of testing can begin. If the circuit does not work, a new design will have to be implemented and tested again.

The next analog circuit to be tested would then be the final amplification stage. This would be very similar to the first pre-amplification stage with the added task of shifting and amplifying or attenuating the signal. A single circuit will be set up and a simulated signal from a function generator will be used. First the amplification or attenuation will occur. This will be followed by a circuit that shifts the signal above zero. It must be ensured that after these two steps the signal voltage is no greater than what the analog to digital converter can accept. If this circuit works the final process of testing can begin. If the circuit does not complete this task it will be redesigned and tested again.

The final step for testing the analog circuitry is to put all the individual sections that have already been tested together. This may also happen in steps. The pre-amplification and filter might be tested together first, as an example. Once all phases are integrated together, testing will begin on this final circuit. First a signal from a function generator will be used. If the circuit works properly testing with the real pinger will take place. This will be done by placing the pinger in an aquarium or small pool and seeing if the signal is properly amplified and filtered. If this final circuit works, a board that has four or five input channels will be designed and sent out to be created.

2.4.2 Digital Hardware Testing

The digital hardware will be tested with a series of test programs that will run on the FPGA or other microcontroller that will be used. Signals will be simulated with a function generator feeding directly into the analog to digital converters. The programs written will encompass each module of functionality. The first test will be to properly sample from the ADCs. The signal must be sampled fast enough and sampled at the same time. The second test is to choose when to sample, meaning how to choose that the incoming signal is from the pinger and not noise. Maintenance programs that will be created involve controlling some components on the analog board, for example, if a programmable filter or VGA was used. Once the signals are coming in properly the next phase is to take the fast Fourier transform and confirm that it works. The final and most important phase is implementing the math on the digital hardware. This will be tested by comparing the hardware's results with the simulated results, which are known to be true. Each of these programs will be created individually and then tested. The final phase would be to combine these programs and test that the overall system works properly.

2.4.3 Software Testing

The first phase of testing the software is assuring that the math, which will be used to calculate heading and distance, are giving adequate results. This is done by first creating a simulator. From the information provided by the simulator a mathematical method will be formed to correctly gain data from the outside world and process this information to gain the heading and distance needed.
The next step would be to implement this code on the microcontroller or FPGA. This will be tested by using data gained from a pinger owned by the AUV Team. If this works sufficiently then the software phase will be ready for integration. If it does not work it may be that new mathematics will have to be used.
Another software section that has to be tested is the communication with the AUV. This will be done by created program that simulate the heading and distance information. It will then be tested that the AUV is getting this information. A similar test will be conducted for other communication that will take place, like the ability for the AUV to change the frequency the pinger will be.

2.4.4 Final Integration and Testing

The final testing phase would be to combine the analog and digital hardware. If this works properly the software portion would be added. At this point the entire system would be set and testing to make sure that information is properly being processed.

2.5 Consultants and Suppliers

The next few sections describe the consultants and suppliers that the senior design team and AUV team will be working with throughout this project. They include mentors from the university and also companies that may contribute.

2.5.1 AUV Team

Two of the members of this senior design group are also on the AUV Team last year and continue to be a part of the team this year. Because of this the senior design group will be very involved in the overall development of the AUV and the acoustic pinger locator system. This will allow for much more testing and easier integration of the system.

2.5.2 Advisors

The two main advisors will be Gary Stein and the robotics club academic advisor Daniel Barber. Both are previous members of the robotics club and have worked on robots for the IGVC, IARC, ASV and AUV Competitions also held by AUVSI. They will help keep the group on track and propose solutions if the senior design team get stuck.

2.5.3 Suppliers

Many suppliers will be used for this acoustic pinger locator system. This project will need supplies from many companies because of all the different aspects this project involves. Integrated components will be used for the filter board, an FPGA or other microcontroller will be used and a new computer will be purchased for the AUV. The AUV Team will try to gain sponsorships from as many of these companies as possible on behalf of the senior design group.

2.6 Facilities and Equipment

This section describes the facilities and equipment that will be used by the senior design and AUV teams.

2.6.1 Robotics Laboratory

Because of the close relationship with the AUV team, the senior design group will be using the robotics club's laboratory located at Partnership II in Research Park. The lab is divided into several sections including a machine shop area, electronics workbench and a section for the ASV and AUV teams. This location will allow the senior design group to work, test and store equipment there.

2.6.2 Equipment

The electronic workbench in the robotics club's lab has an array of tools including oscilloscopes, power supplies, function generators, multimeters and other general tools like wire cutters and breadboards. It also holds a wide variety of previously purchased IC chips and other generally used electronic components like capacitors and resistors. The machining area at the robotics club's lab will be used to construct the hydrophone mount array. A CNC milling machine will most likely be used. This includes the software necessary for taking a SolidWorks cad and generating the proper G codes. If any other tools are needed they can be found there.

2.6.3 Companies

Several companies have already agreed to provide services or equipment to the senior design and AUV team. So far these two companies include Advanced Circuits, who have agreed to make our printed circuit boards (PCBs) for free. Digilent Inc. has also provided the group with two of the high speed ADCs for the FPGA that was purchased last year.

2.6.4 Software Environments: Windows

A windows environment will be used for several parts of the development and research phases. This is because some applications only exist in this environment. Some of

these are discussed in the section below:

MATLAB

MATLAB will be used to test the mathematical methods for finding the pinger. MATLAB allows for the quick creation of mathematically intense program. This would allow the senior design group to quickly go through different methods that may be implemented on the final system. It can also be used to create a basic simulator to further test these methods.

Xilinx

The Xilinx SDK will be used to program the FPGA, if this device is chosen. The group will be using the free student version provided by the company. This software allows for components to be dragged and dropped or written in Verilog. It also allows for the easy assignment of pins to the outside world and what they will be used for.

Eagle

Eagle is a circuit layout software that has a free version available to students. The format generated is what the company Advanced Circuits uses and will allow them to make our boards one testing is complete. There is a free student version that we will be using. The format that the designs are saved is in also used by Advanced Circuits, who will be creating our boards.

2.6.5 Software Environments: Linux

Linux will also be used because all of the robotics club projects are developed in Ubuntu. This will be necessary to write the software that will be interfacing the acoustic pinger locator system and the AUV.

Codelite

Codelite is a free software environment that is used by the AUV team for organizing, building and compiling all there code. This will be used so that the acoustic pinger locator system can be easily integrated.

Chapter 3

Research and Investigation

3.1 Mathematical Analysis

After conducting research in the subject matter of acoustic localization, currently there are several known techniques for obtaining the unknown position of the emitter with a set of receivers. The following sections will describe the analysis of each technique and conclude with the features and drawbacks of the particular method.

The following is an overall description of the problem with known and unknown information obtained by the system: The acoustic pinger is located underwater at some unknown position with a specified ping duration, period, and approximate frequency. The Acoustic Pinger Locator passively listens for the acoustic ping produced by the pinger and captures the sound waves via an underwater microphone, known as a hydrophone, converting the sound energy into an electrical signal of voltage versus time. The voltage of the electrical signal generated is proportional to the strength of the sound wave. Once the sound waves produced by the underwater acoustic pinger are captured, analysis and processing needs to be performed on the received signals to calculate the location of the ping source.

3.1.1 Trilateration

Trilateration is a method for determining the common intersection point given three sphere surfaces. Utilizing this method requires only simple calculations of solving the equations of three spheres with three unknowns the position of the unknown emitter (x, y, z). This method is also known as Time of Arrival (ToA) and requires that the receivers be synchronized with each other as well as the emitter. With the receivers and emitter synchronized in time an absolute distance can be calculated by multiplying the travel time of the wave with the wave propagation speed. These distances are the radii of the spheres about the relative center point to the corresponding receiver.

The following is the mathematical derivation for the trilateration solution.

The equations of three spheres

$$r_a^2 = x_a^2 + y_a^2 + z_a^2 \tag{3.1}$$
$$r_b^2 = x_b^2 + y_b^2 + z_b^2 \tag{3.2}$$
$$r_c^2 = x_c^2 + y_c^2 + z_c^2 \tag{3.3}$$

By letting all the spheres be referenced to sphere a

$$x_a = (x - 0) \tag{3.4}$$
$$y_a = (y - 0) \tag{3.5}$$
$$z_a = (z - 0) \tag{3.6}$$

and simplifying the locations of sphere b

$$x_b = (x_a - x) = (x - x_b) \tag{3.7}$$
$$y_b = (y_a - 0) = y \tag{3.8}$$
$$z_b = (z_a - 0) = z \tag{3.9}$$

and sphere c

$$x_c = (x_a - x_c) = (x - x_c) \tag{3.10}$$
$$y_c = (y_a - y_c) = (y - y_c) \tag{3.11}$$
$$z_c = (z_a - 0) = z \tag{3.12}$$

results in

$$r_a^2 = x^2 + y^2 + z^2 \tag{3.13}$$
$$r_b^2 = (x - x_b)^2 + y^2 + z^2 \tag{3.14}$$
$$r_c^2 = (x - x_c)^2 + (y - y_c)^2 + z^2 \tag{3.15}$$

with the point (x, y, z) being the location of the unknown emitter.

Solving for the closed form of x, y, z reduces to

$$x = \frac{r_a^2 - r_b^2 + x_b^2}{2x_b} \tag{3.16}$$
$$y = \frac{r_a^2 - r_c^2 - x^2 + (x - x_c)^2 + y_c^2}{2y_c} \tag{3.17}$$
$$z = \pm\sqrt{r_a^2 - x^2 - y^2} \tag{3.18}$$

As illustrated in the mathematical analysis, the solution is simple, however implementing trilateration for the current application will not work because the receivers cannot be synchronized with the emitter.

3.1.2 Multilateration

Multilateration, also known as hyperbolic positioning, is another approach for localizing the emitter. This process utilizes the computation of the time difference of arrival of the signal to the set of receivers. Multilateration is different from trilateration because it measures the differences in between receivers rather than the absolute time.

Given the receivers location of A, B, C, D, an unknown location of the receiver (x, y, z), the travel time (T), and the wave propagation (c)

$$T_A = \frac{1}{c}\sqrt{(x - x_A)^2 + (y - y_A)^2 + (z - z_A)^2} \tag{3.19}$$

$$T_B = \frac{1}{c}\sqrt{(x - x_B)^2 + (y - y_B)^2 + (z - z_B)^2} \tag{3.20}$$

$$T_C = \frac{1}{c}\sqrt{(x - x_C)^2 + (y - y_C)^2 + (z - z_C)^2} \tag{3.21}$$

$$T_D = \frac{1}{c}\sqrt{(x - x_D)^2 + (y - y_D)^2 + (z - z_D)^2} \tag{3.22}$$

Let the other receivers and emitter be in reference to receiver A, by setting A to be the origin of the coordinate system,

$$T_A = \frac{1}{c}\sqrt{(x)^2 + (y)^2 + (z)^2} \tag{3.23}$$

Substituting equation (3.23) into the above equations and simplifying reduces to

$$\tau_B = T_B - T_A = \frac{1}{c}(\sqrt{(x - x_B)^2 + (y - y_B)^2 + (z - z_B)^2} - \sqrt{x^2 + y^2 + z^2}) \tag{3.24}$$

$$\tau_C = T_C - T_A = \frac{1}{c}(\sqrt{(x - x_C)^2 + (y - y_C)^2 + (z - z_C)^2} - \sqrt{x^2 + y^2 + z^2}) \tag{3.25}$$

$$\tau_D = T_B - T_D = \frac{1}{c}(\sqrt{(x - x_D)^2 + (y - y_D)^2 + (z - z_D)^2} - \sqrt{x^2 + y^2 + z^2}) \tag{3.26}$$

These three equations define three separate hyperboloids in three-dimensional space. Computing the solution (x, y, z) is performed by solving the intersections of the hyperboloids. The analysis of multilateration has increased complexity than the previous method of trilateration.

With continued investigation into the multilateration problem, research was discovered on a closed form solution. Our senior design team designed a sophisticated and thorough computer simulation in C++, further description of the simulator can be found elsewhere in this report. With the use of the simulator the closed form solution of the multilateration method was confirmed. Further testing was performed to test the accuracy and precision of the algorithm by using a technique known as brute force method. Pinger positions were iterated through a 10 meter by 10 meter by 10 meter volume and error measurements were taken. The results proved to be successful.

An additional step was followed to determine the precision and accuracy of the algorithm given some error through the input of the system. It was discovered that

with only a 1% error of difference in the timing on one of the input signals the output diverged significantly from the desired solution. This correlates to the non-linearity of the system and proves to be unstable and cannot be utilized in the application of the acoustic pinger locator as less than 1% error cannot be ensured.

3.1.3 Data Mapping

The two previous methods would be ideal for solving the desired solution however, the system is non-ideal and prone to error. Thus, a more robust method for calculating the heading and distance to the pinger is desired. Further research was conducted and data mapping was discovered. Data mapping is the process of creating data element mappings between two separate and distinct sets, the input of hydrophone timing information and the output of the heading and distance to the pinger. With the system inherently being non-linear as described in the multilateration section, this poses to be a difficult problem. Data mapping is a technique that attempts to linearize the system about some non-linear function and reduce the error to within some specified tolerance. To implement this technique a simulator of the system needed to be created to brute force this technique. Once proven in simulation that this technique worked it can be implemented on the real system to be tested in a real environment.

3.2 Timing Acquisition Techniques

Acquiring synchronized and accurate timing of the emitter's signal to each individual hydrophone of the array requires specialized hardware and techniques. Through our research we discovered several different methods for solving this problem. Each of the methods has different strengths and weaknesses based on complexity, accuracy, being prone to error, and adaptability.

3.2.1 Counter Method

One of the more simple methods is implemented using high speed counter to measure the time differences for each hydrophone, this will be referred to as the counter method. This method utilizes the analog output from each of the hydrophones that is then compared to some reference voltage that triggers a high speed timer.

An example of a simple implementation can be seen in Figure 3.1 of the counter method. Addition logic is required for the full implementation. One high speed timer would be utilized for the set of the hydrophones, the trigger for the high speed timer would be or'ed with all the comparators with the hydrophones. The output of each comparator would also set a latch and set a register of the current timer value. After all latches are set the timing capture would be captured and the registers that hold the timer values for each hydrophone would be read into some processor to implement one of the acoustic locating techniques mentioned in Section 3.1.

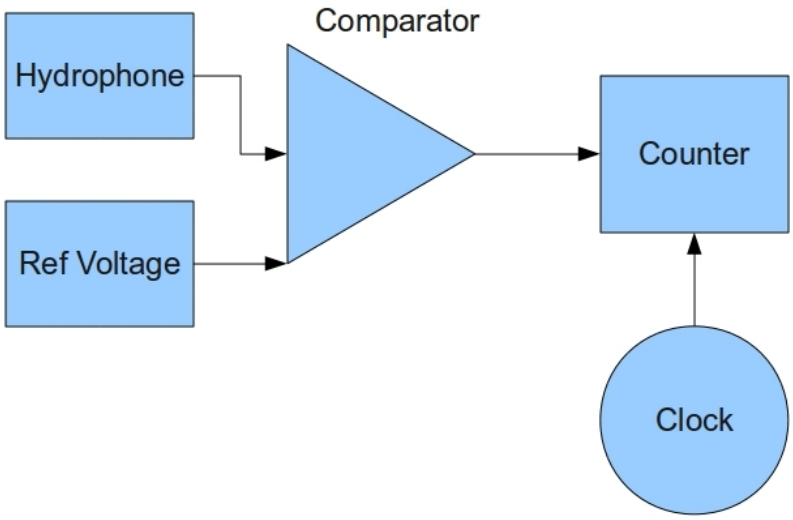


Figure 3.1: A high level block diagram of the counter method for capturing timing of the hydrophone signals.

The counter method is simple however, it suffers from being prone to error within a noisy environment. If there was a noise that was not from the pinger, but had magnitude large enough to trip the comparator, a false positive would be captured. This timing is also based on the condition that each component of the system, including the set of hydrophones, are exactly the same. In the real world this is entirely untrue and a lot of experimentation would need to be performed to see if the tolerances of the components would be within some value to only affect the output of the acoustic location method within some percent of error specified in the problem statement. Due to these constraints of the counter method it will not be useful in implementing it in our system.

3.2.2 Frequency Domain Analysis

Since one of the known specifications is the frequency of the pinger utilization of frequency domain domain analysis can be used to filter out spurious noises and achieve higher reliability on the hydrophone timing. Further, if the hydrophone array is constructed such that each receiver is within one half wavelength(λ) of the emitter signal from each other, phase analysis can be performed to achieve timing information for the system. See Figure 3.2 for a layout that meets the requirement of $\lambda/2$. This technique introduces a different approach to the problem and a transformation of the input parameters to better solve for the timing variables of the set of hydrophones to calculate the outputs, direction and distance.

Analyzing in the frequency domain gives other constraints to the system, in addition to maintaining the distance with each hydrophone. The transformation of the

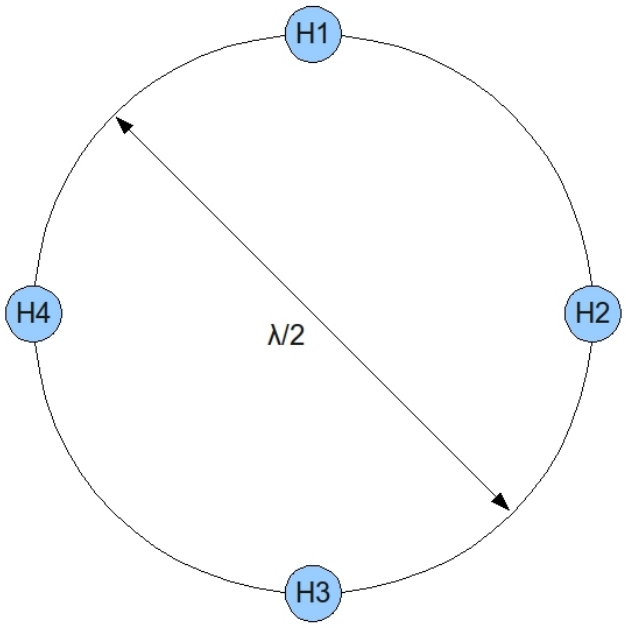


Figure 3.2: Shows an example of a hydrophone array that maintains the requirement of keeping each hydrophone within one half of the signals wavelength ($\lambda/2$).

data from the time domain in which the signals are captured to the frequency domain in how the signals will be analyzed is through the Fourier Transform, more specifically for our system, the discrete fast Fourier transform. To analyze in the frequency domain with digital hardware the signals need to be captured from its form of an analog signal of continuous voltage and continuous time a digital form of discrete values of voltage to discrete time. For sampling the data, the Nyquist Sampling theorem needs to be satisfied, that is the sampling period needs to be at least twice of the highest frequency component of the signal to prevent anti-aliasing of the captured signal. This requires that the system needs analog filters to attenuate all high frequencies and high speed analog to digital converters that samples at least twice the speed of the pinger frequency.

For the required hardware to perform phase analysis see Figure 3.3 of a block diagram. The specialized hardware required for performing phase analysis is an amplifier and filter, to condition the signal for an analog to digital converter, which is used to capture the signals for digital analysis.

There are several requirements for the amplifier, filter, and analog to digital converter. The amplifier needs to be designed such that the gain does not saturate the output signal. With this system being used in a variable environment where the magnitude of the pinger signal is dependent on distance from the sound source, the signal received is not a constant value and may increase by some factor that exceeds the input threshold of the amplifier for output signal saturation to occur. The amplifier needs to be dynamically configurable so that at greater distances the small

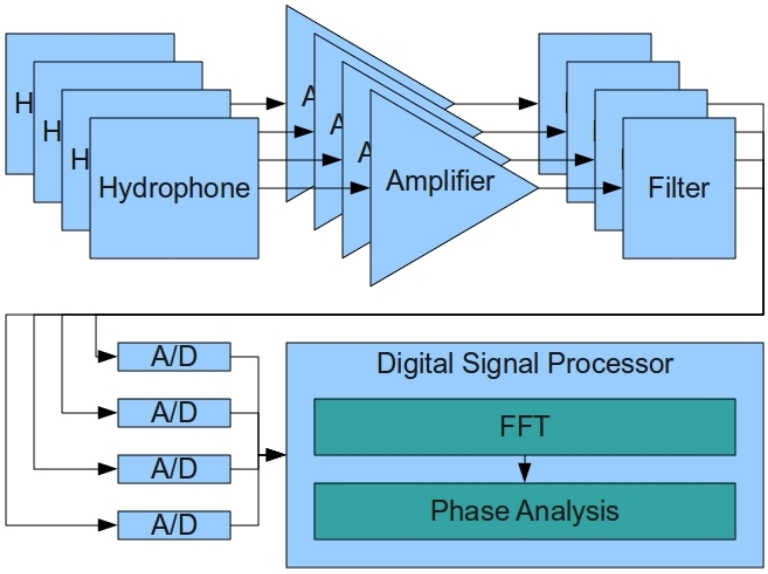


Figure 3.3: This is a high level block diagram of the hardware for analyzing the phases of the received signals from the hydrophones to determine timing information.

attenuated signals due to the power loss of the signal traveling in water can be 'seen', but when the system is close to the pinger source the amplifier does not saturate and the signal gain can be reduced. The approach would be that the digital system can detect if the amplifier is approaching its saturation limit and the system can then reduce the gain automatically. There is information regarding these types of amplifiers, known as Variable Gain Amplifies (VGAs), in the hardware research section of this documentation.

After the signal is appropriately amplified, filtered, and converted into its digital format it then needs to be processed for further analysis. Ensuring that the sampled data size is a power of two, the fast Fourier transform can be performed on the data. This results in two sets of values for each frequency component, the real and imaginary. Calculating the power of each of the frequencies, by:

$$P_i = \sqrt{\Re_i^2 + \Im_i^2} \tag{3.27}$$

Where P is the power, $\Re$ is the real component, and $\Im$ is the imaginary component of the ith frequency.

After calculating the power of each of the specific frequencies, the frequency with the largest amplitude should be the frequency of the pinger source signal:

$$f_i = i * (f_s/N) \tag{3.28}$$

Where f is the frequency of the ith indexed frequency, f_s is the sampling frequency, and N is the number of samples.

At this frequency the phase of each hydrophone can be calculated by:

$$\phi = \arctan \frac{\Im}{\Re} \tag{3.29}$$

Where ϕ is the phase of the hydrophone signal in radians.

After the phase is calculated for each individual hydrophone the phases need to be referenced to one particular hydrophone signal to calculate the time delay of arrival. Therefore, the reference hydrophone will be calculated and considered to have a zero phase, while the rest in the set will have phase in reference to it. This is simply done by taking the difference of the two phases and checking to ensure that the result is unwrapped within $\pm\pi$ that is the basis for specification of the hydrophone setup, one-half wavelength. This principle can be applied because this specification was met and that each of the signals phases are guaranteed to be within one-half wavelength and the timing can be appropriately calculated.

Once the phase differences are calculated the timing can be calculated by:

$$\tau_{x,y} = \frac{\phi_{x,y}}{2\pi} \frac{\lambda}{v} \tag{3.30}$$

$$\lambda = \frac{v}{f} \tag{3.31}$$

$$\tau_{x,y} = \frac{\phi_{x,y}}{2\pi} \frac{1}{f} \tag{3.32}$$

These results can be used in the methods calculating the direction and distance described in the mathematical analysis section. Although this method seems robust by filtering out most of the noise signals of different frequencies, it requires that the received signal is of a pure sine wave and this is simply not true in a real world environment. Errors can arise when calculating phase from the discrete Fourier transform if the input signal is something other than a pure sine wave due to the overlapping in the frequency domain. Also, if the frequency of the signal is not on a calculated multiple frequency of the discrete Fourier transform, 'spilling' will occur to the adjacent frequency components and the actual signal phase cannot easily be calculated.

3.2.3 Cross Correlation

In signal processing, the process of measuring the similarity between two waveforms as a function of a time-shift applied to one of the waveforms is known as cross correlation. For discrete functions, cross correlation is defined as:

$$(f \star g)[n] \stackrel{\text{def}}{=} \sum_{m=-\infty}^{\infty} f^*[m]\ g[n+m] \tag{3.33}$$

Imagine taking the two hydrophone signals that are separated an arbitrary distance away. If the signal from the pinger is propagating through the water at certain

wave-speed one of the hydrophones will start to receive the signal first and the other at a later point in time however, the wave-shape should be exactly the same. The only difference between the two would be attenuation of amplitude for the second hydrophone relative to the first hydrophone. Cross correlation is a good fit for this type of signal analysis as the two signals are only different by a shift in time.

Utilizing cross correlation for deriving the timing for the set of hydrophone signals is a robust method as it is not dependent on the signal being a pure sine wave, it can take any arbitrary shape, and the hydrophone array does not need to be configured in such a limiting manner of one-half wavelength as in the frequency domain analysis method. Cross correlation does have its limitations though, with the discretized signals the highest resolution is based on the actual sample frequency of system therefore, the higher the sampling rate the more distinct values of time shifting can be achieved. Also, provided the system has enough memory to capture the amount of data for the extreme case, the hydrophones can be spread out further for higher resolution.

To explain it more clearly, imagine having only a two hydrophone setup. If the hydrophones were in line with each other which were orthogonal and coplanar to the pinger, the signal received at each pinger would be in phase with a difference of zero time for the cross correlation technique, as illustrated in Figure 3.4. This is due to the distance from each hydrophone to the pinger source is equivalent and this is the path propagating wave.

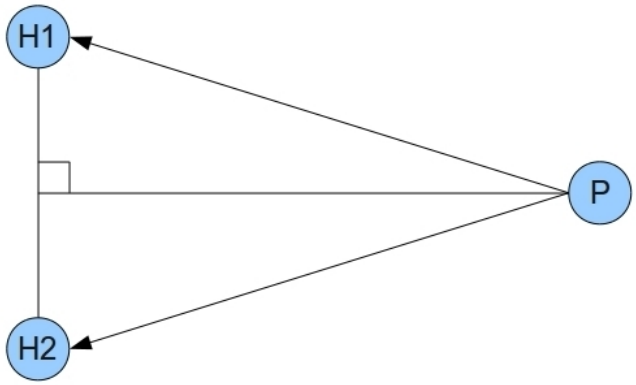


Figure 3.4: Illustrates the setup of having two hydrophones in line with each other, which are also orthogonal and coplanar to the pinger.

For the other extreme, imagine having two hydrophones in line with each other as well as the pinger source. The signal would propagate through the water and be received by the closer hydrophone and then received by the secondary hydrophone. This would provide the most difference of time possible with this arrangement as calculated with the cross correlation technique. This example is illustrated in Figure 3.5 with A being the difference in distance the wave had to travel.

Now imagine that the pinger can traverse along the circumference of a circle with a certain radius coplanar to the hydrophones. The signal phases at the hydrophones

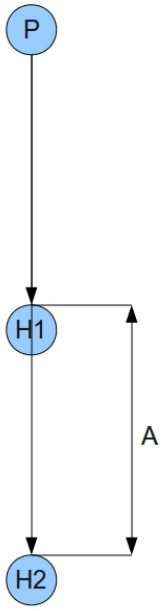


Figure 3.5: Illustrates the setup of having two hydrophones in line with each other, which are also in line with the pinger source. This shows the max difference in time that can be achieved for this setup with a max time shift using cross correlation of A.

will range from completely in phase to the maximum attainable time shift allowed by the geometry. This shows the minimum and maximum range of the input timing and the resolution is based on the maximum distance attainable between a pair of hydrophones.

3.2.4 Conclusion

The process of performing these acoustic localizations for underwater applications are known in general as Long Baseline (LBL), Ultra-Short Baseline (USBL), and Short Baseline (SBL) Acoustic Positioning Systems. The differences between the classes of the underwater acoustic positioning systems are the approximate distances between the hydrophone transducers or receivers relative to each other. For long baseline systems the receivers are placed along the sea floor tens to hundreds of meters apart for the system. This approach would take into account the first method of only having to only keep a high speed running counter to achieve sub-meter accuracy for determining the position of the pinger. However, this setup is not feasible for encapsulating the entire system on the Autonomous Underwater Vehicle (AUV).

For ultra-short baseline acoustic positioning systems the relative distance between the hydrophones are sub meter about 10cm depending on signal wavelength. This approach would utilize the second method of calculating the phase shifts between the signals. The size of this type of system would be feasible for placing on the

AUV. Different hardware tolerances and the environment itself with non-uniformity of underwater acoustics like signal refractions and reflections can have a significant effect on the systems reliability and precision.

3.3 Interface with Autonomous Underwater Vehicle (AUV)

As this system that we are designing is a for a specific application to the customer, the Robotics Club at UCF, the system requirement is for it to be integrated and have a defined interface to the Autonomous Underwater Vehicle (AUV). Per the request of the Robotics Club, research was performed to best get the data to the AUV's on-board intelligence system and the necessary power requirements. After the research was performed, a meeting with the AUV team was held to best determine how to further the design of the Acoustic Pinger Locator System, described in the conclusion of each sub section.

3.3.1 Communication

The communication protocol is an integral part in making a useful component to the AUV. This section contains information and research on the different standardized protocols for communicating with the on-board AUV computer and the Acoustic Pinger Locator (APL). The different communicating protocols have some advantages and disadvantages when it comes to both the hardware requirements, software overhead, and integration. The best solution will provide a robust standard communication protocol that can be supported on both pieces of hardware, the AUV computer and the APL, as well as be limited in the required connections to limit the use of expensive underwater connectors. The protocol has to have a simple software interface to the computer that can support our message set described in a later section.

Ethernet

Ethernet is of a frame-based or packet based computer networking technology primarily used for local area networks (LANs). Ethernet has been standardized by IEEE 802.3 and is the most widespread wired LAN technology. Ethernet has support up to 100 Mbit/s with the on-board AUV computer and is utilized for other purposes of shore power communications and remote logging in. With adding another component to the on-board AUV LAN an Ethernet Switch will need to be integrated into the AUV system, requiring additional power requirements and space inside the waterproof enclosure.

At the hardware level of Ethernet, physical connections need to be made in the form of twisted pair copper wire. For Ethernet in duplex mode only two twisted pairs are required, one for transmission and one for receiving, with a total composition of four individual conductors. Most Ethernet cable meets specifications defined by ANSI/TIA/EIA-568-A, these standards are to ensure that cables meet the necessary

noise rejection and limit interference, known as crosstalk, for high speed and distance data throughput. There are multiple specifications like Category 5 cable, Category 5 Enhanced cable, and Category 6 cable, all meeting different requirements. For our application Category 5 Enhanced cable would be sufficient for maintaining communication with the limited distance the cable has to travel and the data throughput required. The standard itself uses a 8 position 8 contact (8P8C) connector mostly incorrectly referred to RJ45 due to its non-compliance with the method of wiring and non-use for telephone communications. Even though eight conductors are present only four are necessary for Ethernet communications.

Out of all the research performed for the signal processor for the Acoustic Pinger Locator Subsystem, not all of the devices possible for performing the amount of data capture and processing has an on-board Ethernet. For external Ethernet to be added there is complex hardware requirements for the voltages required for Ethernet as well as software interface needed to be written for the protocol. Ethernet would support everthing else required as data rate and throughput.

On the computer side, Ethernet, supports several different Trasport Layers, most notably and widely used, Transmission Control Protocol (TCP) and User Datagram Protocol (UDP). TCP provides reliable, ordered delivery of a stream of bytes from an application on a computer to another application. TCP requires a connection port and hand-shaking for a data stream being transmitted and received. TCP operates on a high level and is a full featured protocol with data integrity and connection management. With these features comes some complexity in having a low level processor to handle the protocol and UDP may be a better choice for ease of implementation.

UDP allows applications to send messages, known as datagrams, to other hosts over an Internet Protocol (IP) network without requiring prior communications to set up special transmission channels or data paths as with TCP. UDP is also compatiple with packet broadcasts, to send to all on a local network, and multicastins, to sent to all current subscribers. These techniques make it preferrable to use UDP for this particular application as to maintain a simpler communication protocol for transmitting and receiving data between the AUV computer and Acoustic Pinger Locator subsystem.

For the AUV computer interface to ethernet, a well used library by the Robotics Club will be utilized, Cross-Platform Utilities (CxUtils) 2.0. This open source software library provides an easy to use interface for communication over Ethernet with TCP or UDP protocols. Further description of CxUtils can be found in the software section of this report.

CxUtils is a multiplatform C++ library containing many useful functions and classes for rapid development of applications. It contains tools for threads, network communication, joysticks, serial communication, shared memory, timers, and basic math operations (matrices, quaternion rotations, coordinate transformations). Using this library it should be a simple task to create a C++ application that can easily be ported between Windows, Linux, and other platforms. CxUtils is also released under the BSD License and is open for use in our project.

Universal Serial Bus (USB)

The Universal Serial Bus is a widely used protocol for computer system peripherals, providing both data transfer and power for the subsystem. The USB standard defines both the signaling as well as the connector. USB is supported by many systems and is readily expandable with hubs for more devices. The protocol itself is based on a host and a slave end point device. In our case the on-board AUV computer would be implemented as the host controller and the Acoustic Pinger Locator subsystem would be the slave end point device. USB supports the necessary bandwidth that would be required for our subsystem to communicate with the host computer and even transferring large amounts of data such as the captured input signals.

The defined signals used in the USB Standard require a four pin interface $5V$, GND, D+, and D-. The power rails are provided by the host controller and can source upto $500mA$ without any custom hardware power system. D+ and D- signals is a two wire interface that provides the means of communication defined by the standard. Without any active USB component cables the max recommended length of a USB cable is 14 feet, due to the high speed bus and susceptibility to noise and signal attenuation. Compared to Ethernet less conductors are needed for communication and power can be provided over the same standard connection. Although, for our project running off of the $5V$ system rail to power the subsystem with the current limitation of $500mA$ is a concern. First, our system is going to require more complex power for the analog circuitry involved for amplifying the AC couple signals of the piezoelectric material of the hydrophones, for example $\pm 12V$ up to a few hundred milliamps. Performing the power conversions of the previous example would show the subsystem would exceed the provided $5V$ at $500mA$.

$$(5V * 0.5A = 2.5W) < (12V * .3A = 3.6W)$$

Second, the sensitivity of the USB signal lines to noise and attenuation make it difficult to splice and place multiple connection points for using waterproof connectors on the AUV, if encapsulation of the subsystem is choosen. The harsh environment of underwater provides a place for significant corrosion and a system capable of handling dropped packets and signal noise is important.

Researching software interfaces for USB support was a more difficult task. Custom drivers are mostly needed when interfacing with a custom solution, like our APL subsystem, however there were several examples discovered on the internet with Linux support of the human-interface device (HID) drivers. The research shows that this is a difficult task with a lengthly learning curve on both the computer end and the signal processor end. Utilizing libraries for this would be integral in being able to develop for this. At this time research was discontinued for USB support to find a more simple method, if deemed appropriate to continue looking into USB the research will only continue then.

Inter-Integrated Circuit (I^2C) Bus

Inter-Integrated Circuit Bus better known as I^2C is a communication protocol invented by Philips that is commonly used to connect low-speed subsystem peripherals to a system. The communication protocol is based on a primary host and addressable slave devices. I^2C utilizes two bidirectional open-drain lines, the Serial Data Line (SDL) and the Serial Clock Line (SCL). Most microcontrollers and FPGA's support the popular standard I^2C with a hardware peripheral or module. Figure 3.6 is an illustration of a typical I^2C setup, showing the two bus lines pulled up to the system voltage and the Master device and Slave devices.

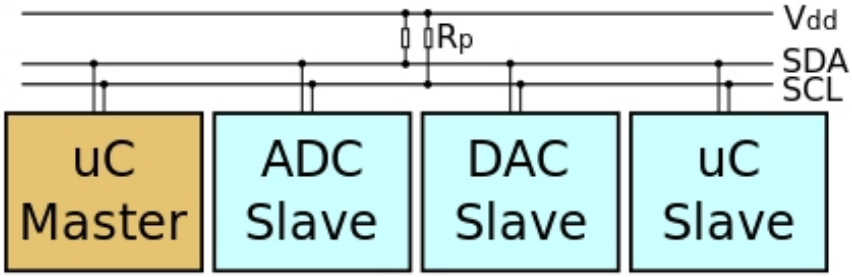


Figure 3.6: A high level diagram of a typical I^2C setup, reprinted with permission granted from Wikipedia GNU Free Documentation License.

The overall architecture of the I^2C is dependent on the master and slave nodes. I^2C uses a 7-bit addressable space with several reserved addresses with a maximum of 112 addressable slave nodes. The master node issues the clock and addresses the slaves whereas the slave nodes receives the clock line and address, performing the next directed action by the master.

The current autonomous underwater vehicle (AUV) design incorporates an I^2C interface for communication with the on-board thrusters and other peripherals. The master device is already defined as a Z-World Rabbit Microcontroller and is connected to the on-board computer via a Asynchronous Serial link with a defined message set. Integrating the Acoustic Pinger Locator Subsystem with the current I^2C would be a simple approach. The current Master device would have to be reprogrammed and modified for interfacing with our subsystem design and additional messages would have to be included.

The one drawback is the layers of communication necessary for the APL Subsystem to communicate with the host computer. To go from the computer to query the Rabbit microcontroller via Asychronous Serial about the APL, and now from the Rabbit to query the APL via I^2C and then for the subsystem to respond and report back the message, there are two of the four layers not necessary. Also, if the hardware architecture of AUV is modified or changed it may be difficult to provide backwards compatability with the APL Subsystem. In the end, it may be easier and simpler to communicate via a more direct route to the on-board host computer.

Asynchronous Serial

A common two-wire communication protocol standard on most computers, microcontrollers, and FPGAs is via a subset of RS-232. Many systems support a simple asynchronous serial communication based on the RS-232 standard.

The two devices that are communicating are connected together with a crossover connection of Transmitted Data and Received Data with a common ground for reference. Each device needs to be configured for an appropriate communication speed with the defined baud rate becuase there is no clock line. Many of the common baud rates are 9600, 19200, 38400, 57600, and 115200. These baud rates are often supported by the host computer and peripheral devices. Using asynchronous serial for communicating with the Acoustic Pinger Locator Subsystem, simplifies development and interoperability for the subsystem by not requiring another hardware interface like in I^2C.

The digital processor that is picked for the design will support RS-232 and for the computer side a USB-to-Serial Adapter will be utilized for the serial support. A state machine will be used to control the hardware universal serial asynchronous receiver transmitter (USART) module for maintaing proper communication to the on-board computer. With the on-board computer the interface to the serial communication will be implemented with the cross platform utilities commonly used in the Robotics Club at UCF known as CxUtils which is described in a different section of this report.

The computer interface to the APL Subsystem will be implemented in a standalone library that can be used in applications that require communication with the APL Subsystem. This standalone library is important to maintain an abstract, object-oriented approach to the overall system. Utilizing a library to interface with the serial port simplifies the system and eliminates development time for communication so more time can be spent on developing other aspects of the subsystem.

Conclusion

Developing a robust and simple communication system will provide a superior system for use in the Acoustic Pinger Locator Subsystem. Using a standard protocol that has already been developed and is supported by the current Autonomous Underwater Vehicle (AUV) of the UCF Robotics Club is an important feature for the APL Subsystem. Limiting the amount of conductors that have to be passed from the AUV System to the APL Subsystem while providing waterproof connections is important to minimize for both cost and eliminate system failures. Support for the protocol on a software and hardware end is another necessary feature for the APL Subsystem. Due to these constraints and features of the APL Subsystem the best choice by both us the Senior Design Team and recommendations from the Robotics Club, we have chose Asynchronous RS-232 Serial Communication Protocol, utilizing a standard baud rate over Data Transmit, Data Receive, and a common reference ground. For interfacing to the computer, the CxUtils library will be used.

3.3.2 Supported Message Set

Defining a message set for the Acoustic Pinger Locator Subsystem will keep the focus of the project on target for what is required. This will define the expectations for the Robotics Club at UCF. The following section describes the general messages and Table 3.1 gives an overview of the message format of the MCU Library of the Zebulon Code Bank maintained by the Robotics Club at UCF.

Field	**Size**	**Description**
Start	2 Bytes	Indicates start of message has a value of '#%'
Command	Byte	Message Type / Command Code 0x00 - 0x0A: Query Messages 0x0B - 0x7F: Command Messages 0x80 - 0xFF: Report Messages
Data	4 Bytes	Fixed size payload, 4 bytes, set to null if no data is used
Checksum	Byte	Sum of all bytes starting from Start field up unitl and not inlcuding Checksum field: Checksum = Start byte 0 + Start byte 1 + Command + Data[0] + Data[1] + Data[2] + Data[3]

Table 3.1: This table shows the message format for the MCU Library of the Zebulon Code Bank.

Query Messages

Query Report - 0x00		
Field	**Size**	**Description**
Type	Byte	The report message you are requesting. Range [0x80, 0x83]
Additional Info	3 Bytes	Specific to the Report Message.

Table 3.2: Shows the format for a Query Report Message.

Payloads for specific Report Messages			
Byte[0] - Type	**Byte[1]**	**Byte[2]**	**Byte[3]**
Report Message Status - 0x80	Motor Address	0	0
Report Depth Sensor - 0x81	0	0	0
Report Digital Inputs - 0x82	Port	0	0
Report Analog Inputs - 0x83	Port	0	0
Report Pinger Heading - 0x84	0	0	0

Table 3.3: Shows the payloads for the particular Report Messages supported by the current AUV and the additional custom messages that will be added for use of the Acoustic Pinger Locator Subsystem.

Command Messages

The following are examples of the already supported messages of the MCU Class for the AUV. These are used to illustrate the use and format of the messaging style already implemented so the APL Subsystem can follow the similar standards and techniques that have already been proven to work.

Set Motor Thrust - 0x0B Description: This command will cause the receiver to set a thrust value for a specific motor.		
Field	**Size**	**Description**
Motor	Byte	Motor Number [0x00, 0xFF] 0x60: Left Motor 0x70: Right Motor 0x80: Vertical Thruster 0xFE: Horizontal Combined 0xFF: All Three Motors Combined
Payload for 0x60, 0x70, 0x80: Thrust	Byte	Desired thrust values [0x1C, 0xE4] 0x80 = Off 0x1C = Full Reverse 0xE4 = Full Forward
Payload for 0xFE: Thrust	2 Bytes	Byte 0: Left Motor Thrust Byte 1: Right Motor Thrust
Payload for 0xFF: Thrust	3 Bytes	Byte 0: Left Motor Thrust Byte 1: Right Motor Thrust Byte 2: Vertical Motor Thrust

Table 3.4: This shows the message format for the Set Motor Thrust command an example of the MCU Library message set.

Set Digital Outputs - 0x0C Description: This command will cause the receiver to set an 8 bit digital output port to the desired values. Can be used for general purpose outputs like LEDs.		
Field	**Size**	**Description**
Port	Byte	Port Number [0x00, 0xFF] Application Specific
Value	Byte	0 in bit field indicates off. 1 in bit field indicates on.

Table 3.5: Shows the message format for the Set Digital Output Message used for referencing when creating the custom message set for the APL Subsystem.

Ping - 0x0D Description: Directs MCU to respond with a Pong Message.		
Field	**Size**	**Description**
Data	4 Bytes	'PONG'

Table 3.6: Shows the format of the Ping message sent from the host computer, must be implemented on the peripheral device, such as the APL Subsystem.

Set Motor Address - 0x0E Description: Directs MCU to configure thruster with new address.		
Field	**Size**	**Description**
Current Address	Byte	Current Motor Address to Change
New Address	Byte	New Address for Motor

Table 3.7: Another example of formatting for a MCU Message implemented on the AUV.

These previous command messages are currently supported by the current AUV MCU Library, the following is a new command that will be developed and implemented for the APL Subsystem and possibly utilized on other systems later for the Robotics Club. We feel that these command messages are required for the successful implementation of the APL Subsystem. Support will be required on both the host computer end as well as on the low level hardware software end. The computer software utilizing the Zebulon code bank will need to have updates to the MCU Class and the support for these messages. The signal processor that will be utilized for the AUV will need to also contain support for these additional messages.

Subscription - 0x0F Description: Directs MCU to report messages regularly without being queried for it.		
Field	**Size**	**Description**
Report Message	Byte	The desired Report Message to subscribe to the MCU
Payload for Report Message	Byte	Specific to application, for example if subscribing to Digital Input Message, it will contain the desired Port for subscription. This field is specific to a particular application, if not applicable leave null.
Enable / Disable	Byte	This field enables or disables that particular subscription. 0xFF - Enables Subscription, 0x00 - Disables Subscription.
Update Rate	Byte	This field is used for commanding the update rate, specific per application. For the APL Subsystem a value 0x00 - is on change.

Table 3.8: A new command message to the MCU library class that needs to be implemented for the APL Subsystem.

APL - 0x10 Description: Directs MCU if an APL Subsystem to perform a particular configuration and setup.		
Field	**Size**	**Description**
Mode	Byte	Normal Mode - 0x00 Debug Mode - 0x01 Reset - 0x02 Raw - 0x03
Math Configuration	Byte	Perform different math on hydrophone data: Multilateration - 0x00 Data Mapping - 0x01

Table 3.9: A new command message to the MCU library class that needs to be implemented for the APL Subsystem.

Report Messages

Report Motor Status - 0x80 Description: This message is used to query motor status and Report Motor Status 1 and Report Motor Status 2 messages are sent in response.		
Field	**Size**	**Description**
Motor Address	Byte	The address for the requested motor status.

Table 3.10: This shows the format of a report message implemented for the current AUV, specifically the Report Motor Status Message.

Report Depth Sensor - 0x81 Description: This message contains the values for the custom depth sensor.		
Field	**Size**	**Description**
Value	2 Bytes	Values is the raw unfiltered data from the pressure sensor for measuring depth on the UCF AUV 2009. Range is [0, 1023].

Table 3.11: Is the formatting for the Report Depth Sensor Message implemented on the current AUV.

Report Digital Inputs - 0x82 Description: This message contains the values of an 8 bit digital input port. Can be used for general inputs like buttons and switches.		
Field	**Size**	**Description**
Port	Byte	Port number [0x00, 0xFF] Application specific
Value	Byte	0 in bit field indicates off 1 in bit field indicates on

Table 3.12: Shows the formatting for another example report message, Report Digital Inputs, that is currently implemented on the AUV.

Report Analog Inputs - 0x83 Description: This message contains the values of a read on an analog input port.		
Field	**Size**	**Description**
Port	Byte	Port number [0x00, 0xFF] Application specific
Value	Byte	Value of Analog to Digital Conversion, Range [0, 255]

Table 3.13: Shows the formatting for another example report message, Report Analog Inputs, that is currently implemented on the AUV.

*Pong - 0x84 *CANNOT QUERY THIS MESSAGE*		
Description: This message is generated as a response to a Ping command.		
Field	**Size**	**Description**
Data	4 Bytes	'PONG'

Table 3.14: Shows the formatting for another example message, Pong, that is currently implemented on the AUV, this must be implemented for all MCU's that are connected with the Zebulon MCU Library. This message response is required for confirmation on connecting to an MCU with the appropirate baud rate.

Report Pinger Heading - 0x85		
Description:		
Field	**Size**	**Description**
Data	4 Bytes	Contains the heading of the pinger relative to the configuration of the array, measured in radians $[+\pi,-\pi]$ scaled to over the four byte value $-\pi \rightarrow 0$, $+\pi \rightarrow 4,294,967,295$. Data range is [0, 4294967295].

Table 3.15: Report Pinger Heading Message must be implemented on the APL Subsystem to communicate with the AUV host computer using the MCU Library.

Report Pinger Depth - 0x86		
Description:		
Field	**Size**	**Description**
Data	4 Bytes	Contains the depth of pinger relative to the placement and configuration of the hydrophone array. Measured in meters, [-100, 100] scaled over the four byte range of $-100 \rightarrow 0$, $+100 \rightarrow 4,294,967,295$. Data range is [0, 4294967295].

Table 3.16: Report Pinger Depth Message must be implemented on the APL Subsystem to communicate with the AUV host computer using the MCU Library.

Report Pinger Distance - 0x87		
Description:		
Field	**Size**	**Description**
Data	4 Bytes	Contains the distance to the pinger relative to the location and configuration of the hydrophone array. Measured in meters, [0, 100] scaled over the four byte of $0 \rightarrow 0$, $100 \rightarrow 4,294,967,295$. Data range is [0, 4294967295].

Table 3.17: Report Pinger Distance Message must be implemented on the APL Substystem to communicate with AUV host computer using the MCU Library.

Report Pinger Position (x) - 0x88 Description:		
Field	**Size**	**Description**
Data	4 Bytes	Contains the (x) position of the pinger relative to the location and configuration of the hydrophone array. Measured in meters, [-100, 100] scaled over the four byte range of $-100 \rightarrow 0$, $+100 \rightarrow 4,294,967,295$. Data range is [0, 4294967295].

Table 3.18: Report Pinger Position (x) must be implemented on the APL Subsystem to communicate with the AUV host computer using the MCU Library.

Report Pinger Position (y) - 0x89 Description:		
Field	**Size**	**Description**
Data	4 Bytes	Contains the (y) position of the pinger relative to the location and configuration of the hydrophone array. Measured in meters, [-100, 100] scaled over the four byte range of $-100 \rightarrow 0$, $+100 \rightarrow 4,294,967,295$. Data range is [0, 4294967295].

Table 3.19: Report Pinger Position (y) must be implemented on the APL Subsystem to communicate with the AUV host computer using the MCU Library.

Report Pinger Position (z) - 0x8A Description:		
Field	**Size**	**Description**
Data	4 Bytes	Contains the (z) position of the pinger relative to the location and configuration of the hydrophone array. Measured in meters, [-100, 100] scaled over the four byte range of $-100 \rightarrow 0$, $+100 \rightarrow 4,294,967,295$. Data range is [0, 4294967295].

Table 3.20: Report Pinger Position (z) must be implemented on the APL Subsystem to communicate with the AUV host computer using the MCU Library.

Summary

Here is a summary of the supported messages for the APL Subsystem.

Code	Description	Reference
0x00	Query Report	Table 3.2
0x0C	Set Digital Outputs	Table 3.5
0x0D	Ping	Table 3.6
0x0F	Subscription	Table 3.8
0x10	APL	Table 3.9
0x82	Report Digital Inputs	Table 3.12
0x83	Report Analog Inputs	Table 3.13
0x84	Pong	Table 3.14
0x85	Report Pinger Heading	Table 3.15
0x86	Report Pinger Depth	Table 3.16
0x87	Report Pinger Distance	Table 3.17
0x88	Report Pinger Position (x)	Table 3.18
0x89	Report Pinger Position (y)	Table 3.19
0x8A	Report Pinger Position (z)	Table 3.20

Table 3.21: Summary of all the messages that must be supported by both the APL Subsystem and the MCU Library Class of Zebulon.

3.3.3 Power

Since this subsystem has a specific purpose of operating and being used on the Robotics Club AUV, consulting with the current AUV team on power availability and requirements was necessary for the development of the APL Subsystem. We conducted research and evalutated the different necessary maximum power requirements, voltages, and current loads for the components planning on being used. The current requirements were also derated to ensure ample current load from the AUV host interface to the APL Subsystem. There are several different approaches that our senior design team can take when implementing the power interface. Some of the design requirements stipulated by the Robotics Club AUV team was that the interface had to limit the amount of waterproof connections necessary for power much like the communications interface. The following section will give an overview of the different power interface options.

Determining the necessary power requirements for the APL Subsystem, we needed to research in the voltage and current requirements for the different components that will be utilized in the APL Subsystem. After performing some research on the current AUV System we found out the different voltages available on the system. The AUV System provides the following voltages:

- 3.3V , Regulated
- 5.0V , Regulated
- 12.0V , Regulated
- -12.0V , Regulated
- 20.0V , Regulated
- 12-14V , Unregulated

- 20-24V , Unregulated

With the current research done for the APL Subsystem the required voltages are as follows:

- 5.0V, regulated for the digital components
- up to ± 12 regulated for the analog components

Regulated

For the power interface from the AUV to the APL Subsystem to provide regulated voltages a number of connections would need to be made including a common ground. According to the Dr. Weeks, the analog and digital power rails should be isolated until the point at the analog to digital converter. The reason for performing this isolation is that on digital power rails there could be up to 100mA of noise or more which would cause interfere with the analog signals that are being captured.

Unregulated

The AUV would be able to provide unregulated voltage to the APL Subsystem. This would provide only limited connections for power to the APL Subsystem. Although, the APL Subsystem would need to implement its one power regulator ciruitry and isolation from one of the unregulated voltage sources of the AUV.

Conclusion

Utilizing one of the unregulated voltage sources from the AUV electrical system to provide power to the APL Subsystem. However, extra development and research needs to be performed for the proper regulation. The current AUV electrical system does not provide the voltage isolation that the APL Subsystem requires. If the AUV were to provide the regulated voltages from the interior electrical system to the APL Subsystem and there is a significant current draw through the multiple connections, a significant voltage drop would occur accross the connection and not provide the necessary voltages to the APL Subsystem. Having the APL Subsystem be able to independently regulate its own voltages this would allow this subsystem to be simply integrated into other versions of the AUV.

3.3.4 Physical Interface

Research into the actual physical interface was an important for both the Robotics Club AUV Team and us the Senior Design Team. The physical interface is the connection from the AUV to the APL. This physical interface had to be agreed upon by us and the customer so that when the design comes to be implemented and integrated into the AUV no issues would arise. There are two basic approaches to designing this subsystem. The first would be to design it as an independent system with a simple physical interface that can be easily integrated into the overall AUV.

This is known as a black box design because the subsystem just works and the system simply interfaces with it not having to be heavily coupled with the subsystem. The other design would be integrating the entire subsystem into the AUV system, enclosed in the waterproof housing of the AUV circuitry. The following further describes the two designs.

Black Box

The black box approach requires more development and further research into encapsulating the subsystem from the interface. The interface provided needs to be easy to use, simple, and work properly. This design will provide better usability in the long run for the Robotics Club AUV team. The ability to move this subsystem from one version of the AUV to another would be a product that the AUV Team would like and will be emphasized in the overall design. For the AUV Team to be able to integrate our subsystem with a simple external waterproof connector that supplies both power and communications to the APL Subsystem will lessen the impact of integrating and prevent errors.

Integrated

Implementing an integrated subsystem into the AUV will simplify the external enclosures required to waterproof the subsystem. There is mechanical engineering research to design a waterproof enclosure up to the desired depth for APL Subsystem. This type of design is out of the scope of an electrical and computer engineering design team. If the subsystem is enclosed in the already designed and fabricated enclosure of the AUV this would eliminate the need for this type of research. Although, the external waterproof connections on the current AUV enclosure would increase for each individual hydrophones. The hydrophone array would have to be integrated onto the system properly for acoustic localization as well as callibration for the system would be different for the system based on arrangment and may prove more difficult.

Conclusion

The idea of implementing a black box or integrated subsystem into the AUV was discussed greatly with the AUV Team. Considering the additional mechanical work required, it was determined that if time permitting the subsystem would be integrated as a black box to better facilitate simplicity of use. However, for scalability our Senior Design Team will have the option of designing an integrated system that will have the ability to be converted later to an encapsulated black box design.

Chapter 4

Analog Hardware

4.1 Introduction: Filter and Amplification

This board is an essential part of the project that will be used to take in and clean all the raw hydrophones signals. These occur in several phases starting with an amplification stage. The next step is to filter out all unwanted frequency which eliminates some noise and echoes from the water environment. The last stage amplifies or attenuates the signal as necessary and shifts it into non-negative voltages so that an ADC can send the information out to the FPGA or other microcontroller. Another feature that will be added to the board is a way of easily changing some variables like frequency and the amount of amplification that is taking place. The next few sections explain the approach that will be taken, the specifications for several components that were researched and the final design.

4.1.1 Hydrophones

The choice of hydrophones is very important. They serve as the only sensors that will be taking in the pinger signals from the water. Important attributes that must be checked include working frequency range, sensitivity, amplification, operational depths and size. Here is a comparison between the current model hydrophone that the AUV team has been using for the past few years and a different brand.

Hydrophone TC4013 miniature reference by RESON

The TC4013 from RESON is the model hydrophone that has been used for the past several years. There have been no noticeable problems with them, but an investigation in new technology is still appropriate. Below in table 4.1 a list of important specifications are given.

Frequency	Sensitivity	Gain	Operational Depth	Size
1Hz to 140KHz	-211dB to 3dB	none	700m	63mm

Figure 4.1: Table showing useful information about the TC4013

Frequency	Sensitivity	Gain	Operational Depth	Size
0 to 125KHz	-250dB	none	250m	50mm

Figure 4.2: Table showing useful information about the CR3 hydrophone.

CR3 Hydrophone by Cetacean Research

This hydrophone from Cetacean Research was interesting because it possessed many of the traits the old hydrophone did along with several other features. These included analysis software and some hardware that allows the setting of a high-pass filter and gain. Unfortunately, these components are two bulky and wouldn't be able to be integrated. Table 4.2 shows specifications about the CR3.

Final Decision

The decision was made to keep the hydrophones that have been used for the past several years. This was because there have been no problems with the current hydrophones. Another reason that the other hydrophones were not chosen was because, as it can be seen by looking at tables 4.1 and 4.2 that there is no real advantage. Finally because the hydrophones cost about a thousand dollars each and that would push the budget over the edge.

4.2 Amplification and Filtering

The following sections discuss the possible approaches that will be taken to complete the tasks of amplification, filtering, final amplification and signal shifting. There are also discussions of different components that were found to complete these tasks.

4.2.1 Amplification

For the acoustic pinger locator system the input signal from the hydrophones will be in the millivolt range. Based on research at last year's AUV competition the circuit gain must be able to magnify a signal by at least 500. If it is any less than this, the AUV will have to be very close to the pinger before it is recognized. Ideally the pinger should be recognized from at least half the distance of the competition site. This task has to be completed without introducing noise that overshadows the incoming hydrophone signals.

4.2.2 Power

The AUV will be running off of 22.2V LiPo (lithium polymer) batteries. This analog board will be fed this raw voltage and regulate it as necessary. One concern is that most op-amps need both a positive and negative input voltages. This means a voltage regulating circuit will have to be build that takes in the 22.2V and outputs a voltage of positive and negative voltage range. This will not be difficult, but op-amps that

run off a single positive voltage will be most desired. The only concerns that may arise from voltage regulation will be noise and heat.

4.2.3 Bandwidth

This is another important attribute of a circuit taking in signals of different frequencies. Components such as op-amps have to been researched with bandwidth specifically in mind. If the op-amp does not have a large enough bandwidth the signals will be attenuated. An example is an amplifier for a cell phone that must function in the Mega Hertz range. This is opposed to an op-amp that designed specifically for audio equipment. Op-amps that are being investigated will be checked to make sure they are able to sufficiently amplify signals between 20-30 KHz as specified in the official rules.

4.2.4 Slew Rate

Slew rate is the op-amps ability to change voltage over time. A high slew rate will allow an op-amp to quickly and accurately magnify a fast signal, which is necessary in the case of the acoustic pinger locator. Slew rate is also very important because once that bandwidth specification is satisfied the slew rate will then exhibit the main limitations of an op-amp. This is especially important in this application because the signal needs to be precise.

4.3 Circuit Components

Here is a discussion of circuit components that will be able to help complete the task.

4.3.1 Operational Amplifier

Amplification will be done with the use of operational amplifiers (op-amps) or a circuitry that uses them. They are very useful in that a simple resistance ratio can be used to amplify a signal without introducing much noise. They are also widely used and thousands are available through manufacturers meaning that an op-amp that meets our specifications can be found.

4.3.2 Variable Gain Amplifier

A variable gain amplifier or VGA is a device that, through digital or analog input, will adjust the gain of an input signal. This would be beneficial because, if used to keep an incoming signal stable, this signal can then be sent to ADCs. Since the signal is always at the same voltage range it will be understood by the FPGA to be the same. This could help in the specific case of figure out when to sample they hydrophones. This would also eliminate several other problems that could arise. One problem that can occur is accidentally making the gain to high. This configuration would work

if the AUV is far away from the pinger, but once the AUV is near the pinger the amplifying circuit may saturate and render this information useless. VGAs come in both analog and digital packages, meaning they can be adjusted digitally or through voltage changes.

4.3.3 Digital Potentiometer

A digital potentiometer is a device that takes in information and translates that to a change in wiper position of a potentiometer. It is like a regular potentiometer except that instead of turning or sliding a mechanical wiper, messages through a communication protocol or setting bits make the changes. This would serve the same purpose as the VGA. The only difference is that this device would be used to change the ratio of resistance on an op-amp circuit or analog VGA to change the amount of amplification.

4.3.4 Digital: VGA and Digital Potentiometer

The digital VGAs and potentiometers are adjusted through a type of communication level where bits are sent to it in several different architectures. Possible downsides of these components are that they have discrete gain steps as opposed to continuous ones. If the resolution is too low it may cause problems when amplifying. These steps are usually logarithmically linear. For a most of the components found the communication protocols are listed below:

I^2C

I^2C stands for Inter-Integrated Circuit. It is a communication protocol that allows for many different devices or nodes to exist on a single bus. To use this protocol the system only need two wires, one is a clock and the other is the bi-directional communication line. This protocol would more easily integrate into the overall system because it has less hardware needs and several other components on the vehicle, such as the motors, will be communicated through I^2C.

SPI

SPI stands for serial peripheral interface bus. This communication protocol like I^2C allows for several components to exist on the same bus. Then through a clock line, a slave select line and two data transfer lines it allows for communication between devices. This protocol is not as desired because no other component on the AUV uses it. Also, it requires more wires to run. Even though these problems exist, a component with the communication standard is still a viable option.

Other

Some components used other standards that are not widely used and were disregarded because of it.

4.4 Researched Components

This section gives more information about specific components found and why they are appropriate for the acoustic pinger locator project. This involved investigating important attributes of each components and how they meet the necessary design requirements.

4.4.1 Operational Amplifier

LT1028 - Ultra Low Noise Precision High Speed Op Amps

This op-amp is from Linear Technology and is specifically listed as a high-speed precision op-amp. Here is a list of important features in Figure 4.3.

Noise	Power	Gain-BW Product	Slew Rate
$1.1\frac{nV}{\sqrt{Hz}}$	± 22V	50 to 75 MHz	15 $\frac{V}{\mu V}$

Figure 4.3: Table showing useful information about the LT1028.

This op-amp is particularly interesting because its uses were listed specifically for hydrophones.

Pros: As the table shows, the bandwidth is great enough that even if the op-amp is used with a gain of 1000, the 20 to 30KHz signal will not be attenuated. The op-amp also runs off of 22V, which means that voltage regulation might not be necessary. This op-amp also comes in a DIP package, making it easier to use.

Cons: The only real disadvantage is that the slew rate might not be great enough to accurately keep up with the signal while amplifying it, which would result in signal distortion. Although a DIP package is available, there is only one op-amp per package which takes up more room and costs more.

MAX414 - Quad, 28MHz, Low-Noise, Low-Voltage, Precision Op Amps

This op-amp is from Maxim and is specifically listed as a low noise precision op-amp. Here is a list of important features in Figure 4.4.

Noise	Power	Gain-BW Product	Slew Rate
$4.5\frac{nV}{\sqrt{Hz}}$	± 2.4V to ± 10.5V	28 MHz	4.5 $\frac{V}{\mu V}$

Figure 4.4: Table showing useful information about the MAX414.

It can be observed, through the provided table, that this op-amp meets almost every need of the pinger locator system.

Pros: A benefit of this specific model is that it comes in a DIP. This package has four op-amps per chip and is cheaper than the LT1028. Depending on how many op-amps will be needed in the final design it would be very advantageous because it lowers cost and used space.

Cons: The main issue is the bandwidth and amplification. If a gain of 1000 is needed the usable bandwidth has dropped to 28KHz. This might be a problem if the pinger is emitting a signal at 30KHz.

4.4.2 Variable Gain Amplifier

AD8369: 600 MHz, 45 Db Digitally Controlled Variable Gain Amplifier

This variable gain amplifier is from Analog Devices and is specifically listed as a high performance digitally controlled VGA. It has a large bandwidth and a maximum gain of 100. Here is a list of important features in Figure 4.5.

Noise	Power	Gain-BW Prod	Slew Rate	Max Gain	Resolution
$2\frac{nV}{\sqrt{Hz}}$	$\pm$ 3.0V to $\pm$ 5.0V	600 MHz	1200 $\frac{V}{\mu V}$	40 dB	16

Figure 4.5: Table showing useful information about the AD8369.

This device is controlled through four pins that give 16 discrete steps. These steps allow for very precise gain management.

Pros: A nice feature of this VGA is that is has a gain step of 1.4. This will allow for precise setting of amplification.

Cons: Unfortunately, the maximum gain of this device is 100. For the purposes of this project two of these VGAs would have to be put in series for a maximum gain of 10,000. Another difficulty with this VGA is that it does not come in a DIP package, making it more difficult for soldering.

LMH6518: 900 MHz, Digitally Controlled, Variable Gain Amplifier

This variable gain amplifier is from National Semiconductor. It is listed as working with a specific ADC to make a fast data acquisition system. Because of this, the LMH6518 would be used for the final amplification/attenuation step which is where the information is sent to the FPGA through ADCs.

Noise	Power	Gain-BW Prod	Slew Rate	Max Gain	Resolution
$0.98\frac{nV}{\sqrt{Hz}}$	$\pm$ 3.6V to $\pm$ 5.5V	900 MHz	38.8 dB	N/A	20

Figure 4.6: Table showing useful information about the LMH6518.

This device is also a digital VGA. Unlike the AD8369, it is controlled through a serial connection.

Pros: Several advantages to using this VGA include the low noise, large bandwidth and more precise gain steps.

Cons: Like the AD8369 the maximum gain is about 100, so two chips would have to be used in series to get the necessary amount of gain. Unfortunately, the slew rate was not provided in the datasheet. Also, the datasheet seems to assume it will be going directly into the ADC which is not what we will be doing for this project.

AD8330: Low Cost, DC to 150MHz Variable Gain Amplifier

This variable gain amplifier is from Analog Devices. Unlike the previous Analog Device's VGA, this device's gain is adjusted based on voltage. It is specifically listed as a wideband, low noise VGA with moderately low distortion.

Noise	Power	Gain-BW Prod	Slew Rate	Max Gain	Resolution
$5\frac{nV}{\sqrt{Hz}}$	±2.7V to ±6V	150 MHz	50dB	1500 $\frac{V}{\mu V}$	infinite

Figure 4.7: Table showing useful information about the AD8330.

The major difference between this VGA and the other two VGAs that were investigated was that it is adjusted through a voltage source, not digitally. This means that another device or method will be needed to control the gain, not directly from the computer.

Pros: This device has several interesting features. The bandwidth automatically adjusts so that it is constantly 150MHz, regardless of the gain. Also it runs off of a single positive voltage source, this would result in one less regulator.

Cons: Like the other two VGAs, its max gain is about 316, this would mean that two of these devices will have to be used to acquire the necessary gain. Another issue is that there is no DIP package available, which will make it more difficult to use.

4.4.3 Digital Potentiometers

DS1803: Addressable Dual Digital Potentiometer

This digital potentiometer is from Maxim. This device comes in several packages and 10KΩ, 50KΩ and 100KΩ versions.

Max Resistance	Power	Protocol	Resolution
10KΩ, 50KΩ, 100KΩ	3V to 5V	2-Wire Serial Interface	256

Figure 4.8: Table showing useful information about the DS1803.

This device may allow for more discrete gain levels when used in conjunction with an voltage controlled VGA.

Pros: This digital potentiometer has several advantages. For one it is controlled through I^2C, which will be used extensively throughout the AUV. Another interesting feature is that it allows for 256 discrete steps that would give, with the 10KΩ resistance, steps of about 40Ω. This could allow for precise control over the gain. Another advantage is that it comes in a DIP package and has two separate digital potentiometers built into a single chip.

Cons: There were no really noticeable disadvantages to this device except that it will add to the complexity of the board because an op-amp or VGA and digital potentiometer need to be used.

FN8164: Quad Digital Controlled Potentionmeter

This digital potentiometer is available through Intersil. It comes in several package types and different resistor ratings.

Max Resistance	Power	Protocol	Resolution
2KΩ, 10KΩ, 50KΩ	-3V, 5V	2-Wire Serial Interface	64

Figure 4.9: Table showing useful information about the FN8164.

Pros: Like the DS1803, the communication protocol used is I^2C, which is extensively used throughout the AUV. The chip also comes in several versions. Finally, a DIP version is available, allowing it to be easily used.

Cons: Several problems with this chip include that a -3V and 5V voltage source are needed to run the chip. This would cause additional regulators to be added to the board. Another issue is that unless the 2KΩ version is used; it is not as precise as the DS1803.

4.5 Possible Configurations

This section encompasses many possible configurations for amplification. These circuits can be applied to the pre-amplification or post-filtering amplification. They are produced using the devices and components that were discussed in the previous section.

4.5.1 Op-Amp with Resistor

This is the simplest configuration; it involves amplification through the use of a single op-amp and a single non-adjustable resistor.

Pros: This configuration would guarantee that amplification will happen. This also allows for a specific gain to be set.

Cons: Unfortunately, with this configuration there is no way to easily change the gain. To change the gain, the resistor would have to be de-soldered and then a new resistor would have to be put in its place, making it very limited.

4.5.2 Op-Amp with Potentiometer

Pros: This is similar to the previous configuration. A single op-amp and a potentiometer would be used. The potentiometer would be set up so that it acts like an adjustable resistor. As this resistance changes, so would the op-amps' gain.

Pros: This produces a guaranteed amplification with an easier way of adjusting the gain.

Cons: Although better than the previous configuration, in the long run, it will still be difficult to tune because this analog board will be located in a water-tight box or inside the AUV's electronics tube. To make an adjustment, the tube will have to

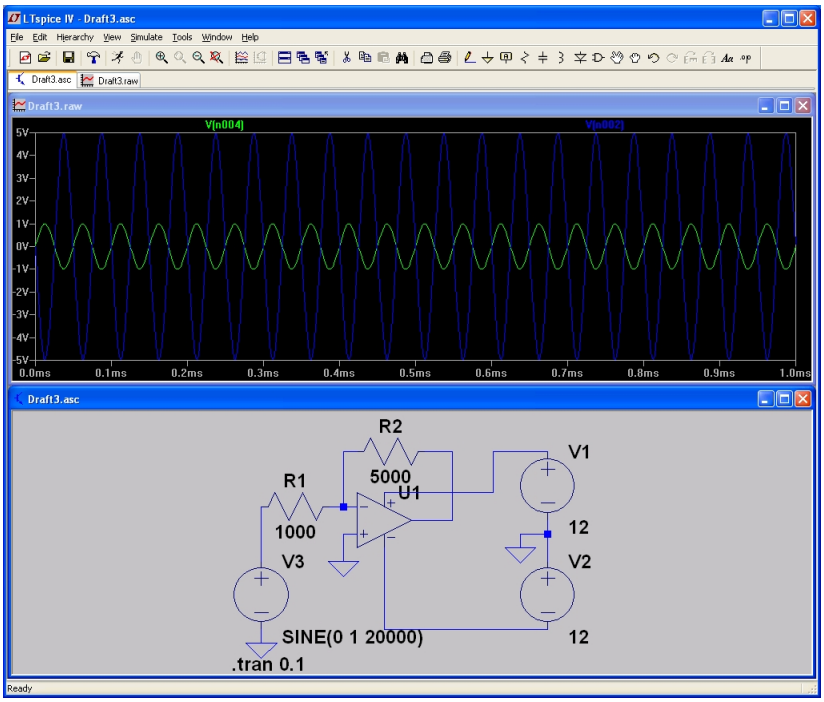


Figure 4.10: Here is the basic configuration that also shows the input and output voltages. The other circuits work on this same principle.

be removed and opened which may be cumbersome. Also, the potentiometer has a limit to how much it increases the gain, and then it suffers the same problem as a normal resistor.

4.5.3 Op-Amp with Digital Potentiometer

This configuration still follows the basic idea of the previous two configurations. This circuit replaces the regular potentiometer with a digital potentiometer.

Pros: This configuration eliminates the problems of the previous two configurations by allowing the gain to be changed through software. The AUV can stay intact while, through software, the gain is adjusted. This would also allow for a continuously changing gain as the AUV moves closer or farther away from the pinger.

Cons: By using the digital potentiometer the gains are limited to discrete values. These devices are also limited because the resistance values that exist may be too small to provide a high enough gain.

4.5.4 Analog VGA with Digital Potentiometer

This configuration uses a variable gain amplifier instead of an op-amp. This VGA is then controlled by a digital potentiometer.

Pros: The VGA may be more stable than a simple op-amp because it may be constructed to improve noise reduction and limit other signal distortions. Another

benefit of this configuration is that a very small voltage change will be able to control the entire gain range of the device.

Cons: This configuration suffers from the same, discrete values, problem as the previous circuit. It also adds complexity to the circuit along with cost and room used on the board.

4.5.5 Analog VGA with Analog Feedback

This configuration would involve a single analog variable gain amplifier coupled with other circuitry to adjust the gain.

Pros: This method could automatically compensate for changes in incoming signals. This would eliminate the need for any other control of the system.

Cons: The extra circuitry to make it work would result in more space being used and a higher cost. It would also be more complex and take more time to test. Another drawback of this method is that once completed there is no simple way of changing the output.

4.5.6 Digital VGA

This configuration involves the use of a single variable gain amplifier that is controlled through software.

Pros: This configuration allows for the analog gain to be changed easily through serial or pin setting. This gets rid of the need to interact with the hardware directly. This could be configured to automatically adjust as the AUV travels through the water. Another benefit is, unlike several other proposed methods, only a single chip is needed to implement this method. Making the board smaller and more cost efficient.

Cons: These values are discrete like with the digital potentiometer. Also, some of the programming architectures may be more difficult to implement.

4.5.7 Final Choice

With many options explored and many components researched the decision was made to use the combination of the analog variable gain amplifier controlled by a digital potentiometer. This will hopefully prove to be more stable while still allowing precise control of the gain.

4.6 Filtering

One requirement of the system is that it must be able to accurately find signals at a certain frequency. To do this a filter will be used that attenuates all unwanted frequencies. This section discusses different types of filters and what options are available.

4.6.1 Filter Types

Several base filter types exist. This section discuses each type and their ability to work for the acoustic pinger locator system.

Lowpass

A lowpass filter is a signal cleaning circuit that allows lower frequencies to pass through a circuit. This means that any frequency that is at a low enough frequency will pass through a designed threshold, the rest will be attenuated.

This would be more useful if we expected to have noise only from high frequency sources. This will probably not work because it allows for all frequencies below the 20KHz minimum. This could cause the amplification and filter board to amplify noise and other miscellaneous sounds. This would especially be a problem if two pingers are in the water at different frequencies.

Highpass

A highpass filter is a signal cleaning circuit that allows high frequencies to pass through a circuit. That means any frequency above a designed threshold will not be affected, while lower frequencies will be attenuated.

This would be useful if we knew our noise would only come from one source, like an AC power source at 60Hz. This will probably not work because it allows for all frequencies above the 20KHz minimum. This could cause the amplification and filter board to amplify noise and other miscellaneous sounds.

Band-Pass

A band-pass filter is a filter that only allows a designed range of frequencies to pass through the circuit. Frequencies that are within the threshold, around a center frequency, are allowed through. All other signals will be attenuated.

If this filter had a narrow band it could work for this project. By having a small range of frequencies that are allowed, the pinger frequency can be specifically targeted. It would also remove all noise above and below the wanted frequency. This would provide a clean signal with minimal noise.

Notch or Band Gap

A band gap filter is a filter that allows all frequencies outside of a designed range through a circuit. This essentially eliminates a band of frequencies.

This filter would be useful if a specified frequency needed to be eliminated, but the opposite is needed for this application.

4.6.2 Attenuation

The ability of the filter to eliminate unwanted signals is a very important component of filters. A filter's ability to eliminate these frequencies is specified by its "order". The

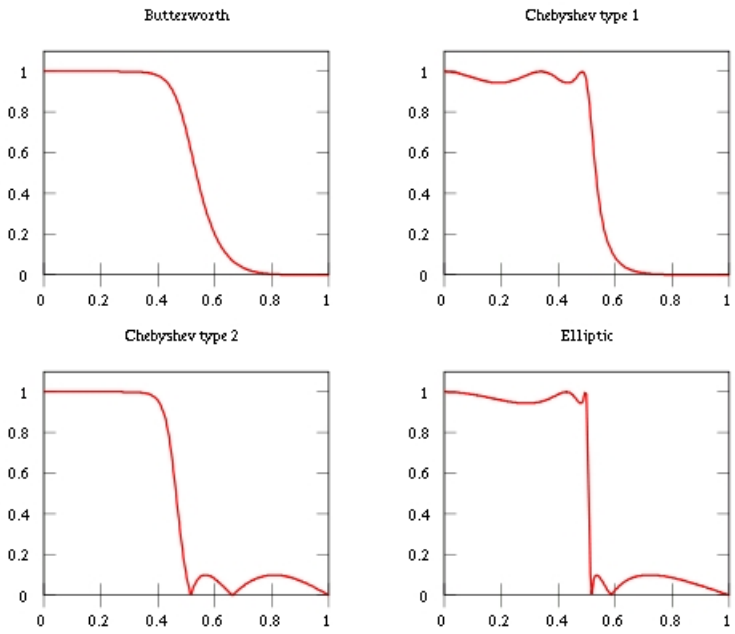


Figure 4.11: Several graphs showing the different characteristics and shapes of different filters. This image is reprinted with permission granted from the Wikipedia GNU free documentation license

higher the "order", the more capable the filter is to eliminate unwanted frequencies. A first order filter will begin to decline at a 20 dB/decade slope at the cutoff frequency. This translates to dividing the signal power by 10. If a second order filter was used, the decline would become 40 dB/decade or a decline of 100.

The project calls for the ability of the system to listen for a single frequency that may range between 20 to 30KHz. Based on the competition from last year, several pingers were in the water at a single time. This means that a relatively high order filter should be used so that all noise and other pingers will essentially be eliminated.

4.6.3 Shape

Another attribute that a filter can have deals with its shape and how it reacts near the cutoff frequency. Several types exist including Bessel, Butterworth, Elliptic and Chebyshev filters to name a few. Figure 4.11 shows how several of these filters react.

4.6.4 Final Choice

To complete the task of listening and recognizing and underwater acoustic pinger a filter must have several characteristics. It will be a band-pass type filter because the pinger sends a signal at only one frequency. Noise and other pingers are also a concern. Because of that a relatively high order filter will be used, more experimentation will have to be done to know the exact order. Finally, because the frequency band being examined will be small and as little distortion as possible is wanted a Butterworth

filter will be used. If these criteria are met, an ideal filter for this application may be found.

4.7 Programmable Filters

MAX263/MAX264: Pin Programmable Universal and Band-Pass Filters

This programmable filter is available through Maxim. This device allows for the user to program the center frequency, Q value, attenuation, type of filter and shape. Figure 4.12 shows the chip's interesting features.

Frequency Range	Power	Type	Resolution f_0	Resolution Q
0 to 75KHz	$pm5$V	LP,HP,BP,N	32	128

Figure 4.12: Table showing useful information about the MAX263/MAX264.

This is a highly configurable filter chip that allows almost every filter variable to be adjusted. This can be used for this project because it works in the needed frequency range and has all the features that were discussed in the previous section.

Pros: This filter can be configured to implement a 2 to 8^{th} order Butterworth band-pass filter with an adjustable Q. This is the desired filter type as discussed before. The fact that not many external capacitors and resistors are needed is also nice. Another feature is that a DIP model is available that will make it easier to use.

Cons: The only real problem with this chip is that there is no software provided by the company to automatically generate the needed input voltages, capacitors and resistors.

LTC1059: High Performance Switched Capacitor Universal Filter

This programmable filter is available through Linear Technology. This device allows for the user to program the center frequency, Q value, type of filter and shape. Figure 4.13 shows the chip's interesting features.

Frequency Range	Power	Type	Resolution f_0	Resolution Q
0.1 to 40KHz	4.74V to 16V	LP,HP,BP,N,A	continuous	continuous

Figure 4.13: Table showing useful information about the LTC1059.

This is a configurable filter chip that allows for many filter configurations. The filters that are producible can be applied to the acoustic pinger locator system.

Pros: This component can be configured to implement a 2^{nd} order Butterworth band-pass filter that also has a configured Q. This is the desired filter as discussed in the previous section. This is done by adjusting a clock signal. The chip comes with an extra unused op-amp, which can help with space on the board.

Cons: This device has several problems, one of which is that it is limited to 2^{nd} order filters. To create a more powerful filter, more than one of these chips would

have to be used, taking up space and money. Another issue is that the datasheet does not give enough information to properly configure the device and separate software is needed.

4.7.1 Final Choice

After considering the LT1059 and the MAX263/MAX264 chips, the decision was made to use the second chip. This is because it allows for so much more control. It allows the user to change Q values, N orders and center frequencies through pin setting and a pulse width modulated signal. Although Linear Technology provides software to help configure their different chips, the MAX263/MAX264 meets the specifications exactly.

4.8 Final Amplification/Attenuation and Shifting

The final stage of this board is to make sure that the signal voltages are within a usable range. This is so the ADCs will be able to use this information. If the voltage is too high when it enters the ADC, which might have a max voltage of 3.3V, it might break the ADC or read incorrectly. This may also occur if the range is wrong. For example the ADC might take voltages from 0 to 3.3V, but the incoming signal ranges from -1 to 1V, resulting in breaking the ADC or interpreting the information incorrectly. The ADCs that were researched can be found in the next chapter.

4.9 Possible Configurations

Because op-amps, VGAs and digital potentiometers have already been researched, those same components will apply to this section of the circuit. These next sections are about the possible configurations to properly modify the signal.

4.9.1 Amplification/Attenuation Options

This part of the circuit will serve as a final amplification or attenuation stage. It will depend on the initial amplification, whether the signal needs to be further amplified or attenuated. Two circuits, similar to the pre-amplification stage, will be investigated for this section.

Op-Amp and Potentiometer

This circuit would be set up so that the potentiometer adjusts the gain of the op-amp. To adjust this circuit the container would have to be opened.

Pros: This circuit may work as opposed to its previous implementation for the pre-amplification stage. The reason for this is because once the pre-amplification stage gain is found; the system will automatically try to adjust to that level in the

first phase. Technically, the final amplification stage should always receive the same voltages.

Cons: The issues with this configuration are that it is still difficult to adjust the potentiometer, but it will not have to be changed as often. Also, if a static pre-amplification stage is used, this configuration might not work.

VGA and Digital Potentiometer

This circuit would be configured in the same way with the VGA adjusted by the digital potentiometer.

Pros: Unlike the previous circuit this would allow for a static or dynamic gain design. This would have all the capabilities of the previous circuit and not need the circuit to be accessed to change the gain/attenuation. This would make testing, design adjustments and quick changes easier.

Cons: The only downside of this configuration is that it would increase the cost and more space would be taken up on the board.

4.9.2 Shifting Circuit Options

This circuit portion will serve as the final step of the analog phase. This step shifts the voltage into a positive voltage range. This will allow an ADC to gain the signal's information. Two circuits were investigates to fulfill this task.

Adder Circuit with Potentiometer

This design would implement an op-amp adding circuit. This configuration involves having two input voltages go into a single op-amp terminal, these input voltages are then added together and adjusted based on their resistances.

Pros: The advantages of this configuration are that the output can easily be adjusted by using regular or digital potentiometers.

Cons: The disadvantages of this circuit are that it involves the use of another chip and external resistors.

Villard Voltage Level Shifting Circuit

This design implements a voltage shifting circuit through the use of capacitors and a diode circuit. The values of C_1 and C_2 are adjusted to affect the output. It must be kept in mind that this method halves the input voltage.

Pros: The advantage of using this design is that no new ICs would have to be used. The only components need would be two capacitors and a diode. This would save money, power and space.

Cons: This circuit halves the input voltage. Another disadvantage would be that it might not be as stable. Another problem is that if the final voltage needs to be changed, the capacitors would have to be removed and new ones re-soldered in their place.

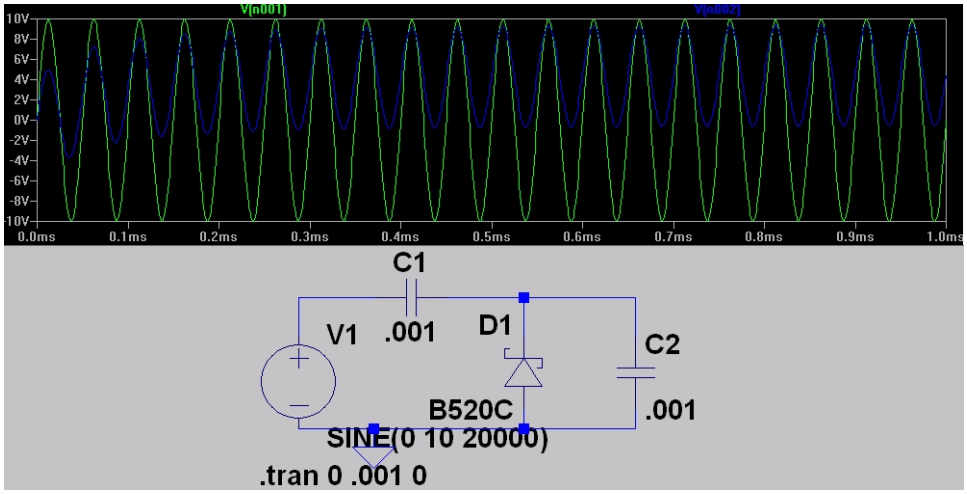


Figure 4.14: Here is a circuit showing the configuration of the Villard voltage level shifter. Simulation created in LTSpice with values C1 = .001F, C2 = .001F, f = 20KHz, Vin = 10V

Final Choice

Despite the additional cost and space used by the op-amp, the op-amp adder will be used. This is because it is assumed that it will be more stable and easier to configure then the Villard voltage shifting circuit.

4.10 Final Analog Hardware Design

Each portion of the amplification and filter board has been discussed. In each of these sections the important attributes have been brought up, along with components that can be used for each portion and finally the possible circuits that can be produced from them. Because of the complexity of the analog hardware board it is broken up into the individual modules discussed throughout this chapter. The power regulation is not shown in these schematics because it is has not been decided if the analog board will receive raw or pre-regulated power. Each schematic was produced in the Eagle software.

This first circuit is the section of the analog board where the raw signals from the hydrophones are initially captured. The hydrophones have two terminals, one is the signal and the other is the reference, which is connected to ground on the board. The signals are sent though a unity-gain op-amp circuit to preserve the signal from interference. The four outputs of HYDROPHONES (1 - 4) are then sent to the pre-amplification stage. Below, in figure 4.15, is the schematic. The hydrophone array is sent into the MAX414 chip that was discussed earlier in this chapter.

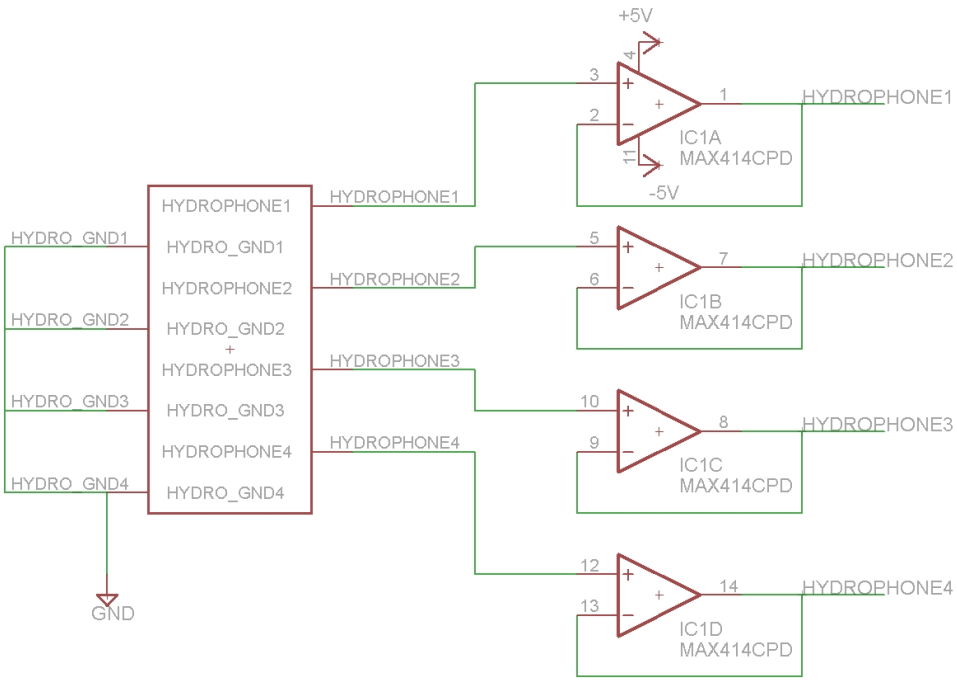


Figure 4.15: Here is a schematic showing how the MAX414 will be used to maintain the integrity of the incoming signals.

The pre-amplification stage will then take place, the inputs being the signals from the acquisition stage. In this stage a simple hand-adjustable gain stage is implemented with op-amps and potentiometers. This is because the VGAs could not provide enough gain to boost the signal in one step. This first gain will be multiplied by the VGAs' gain to solve this problem. The AD8330 voltage controlled VGAs were chosen to be used in conjunction with the DS1803 digital potentiometer. The FPGA will control the digital potentiometer, which then adjusts the voltage on the VGA changing the gain of the circuit. The digital potentiometer is addressed for I^2C through the A0, A1 and A2 pins, this allows for up to eight DS1803s to be used together. If needed the second potentiometer of the DS1803 can be used to adjust the simple op-amp gain, but this will be static when running. These four outputs will then be filtered in the next stage. Figure 4.16 shows the circuits created using these components.

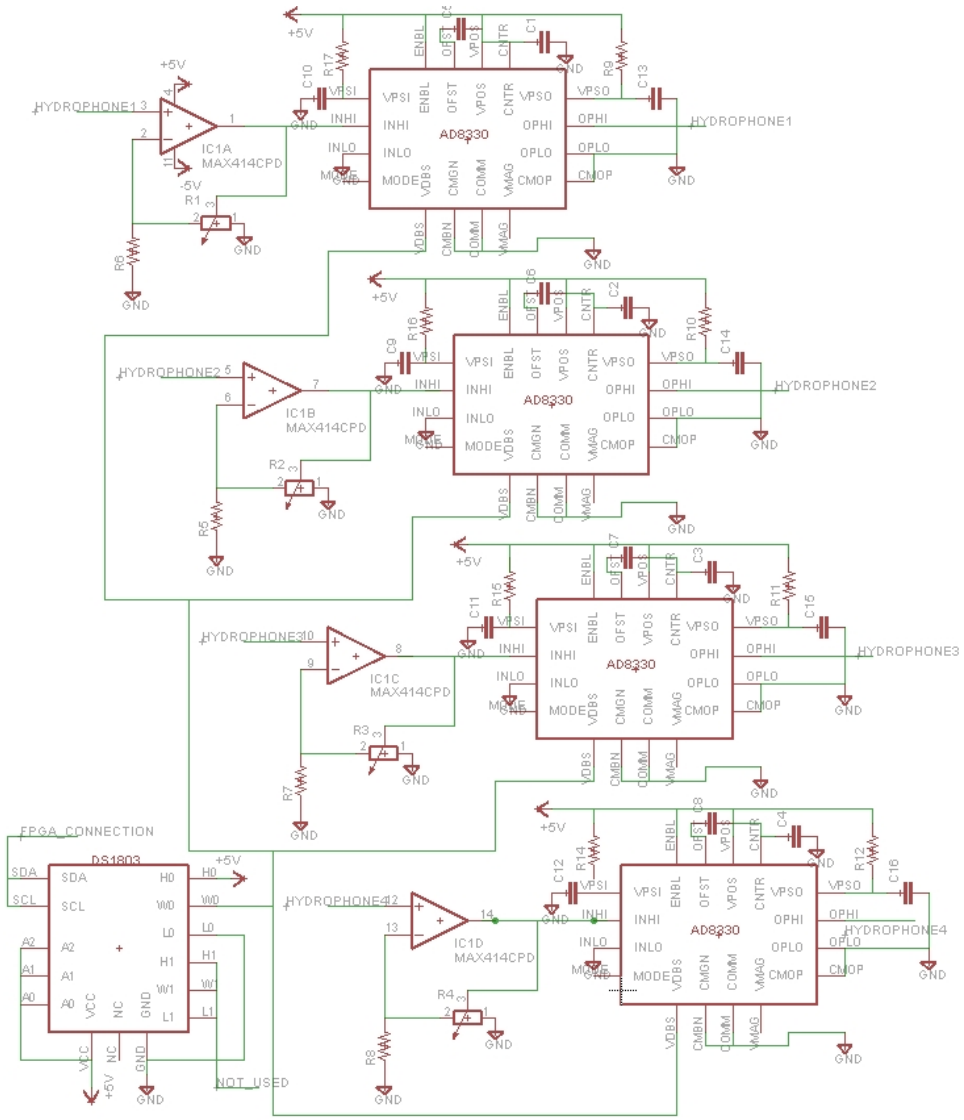


Figure 4.16: This figure shows the pre-amplification process that involves a hard amplification stage with MAX414. The next stage uses VGAs that are controlled by a digital potentiometer, for finer gain detail.

The filtering stage will take in the four amplified hydrophone signals and then filter out all unwanted information. The MAX267 chip, as discussed earlier, will be used. Only two will be needed because each contains two adjustable band-pass filters. For the project an adjustable center frequency was needed, this is done through CLKA and CLKB. A square wave between oscillating between 40Hz to 1.5MHz will adjust the frequency from 0Hz to 37KHz, which contains the needed 20 to 30KHz. A free op-amp is available on each chip, but is not used. The Q value, in this configuration, is set to 64. This is controlled through the Q0 - Q7 pins. The Nth order is set through the F0 - F4 pins and in this configuration is set to 3. These pin adjustments can be expanded by adding jumpers to each Q and F pin to allow access to all configurations. Figure 4.17 shows the MAX267 being used to implement a Butterworth 3rd order

band-pass filter with a Q value of 64. This is the exact configuration discussed in earlier sections. The outputs of this circuit are listed as PROC_SIGNAL, these signals will feed into the final phase of the analog hardware board.

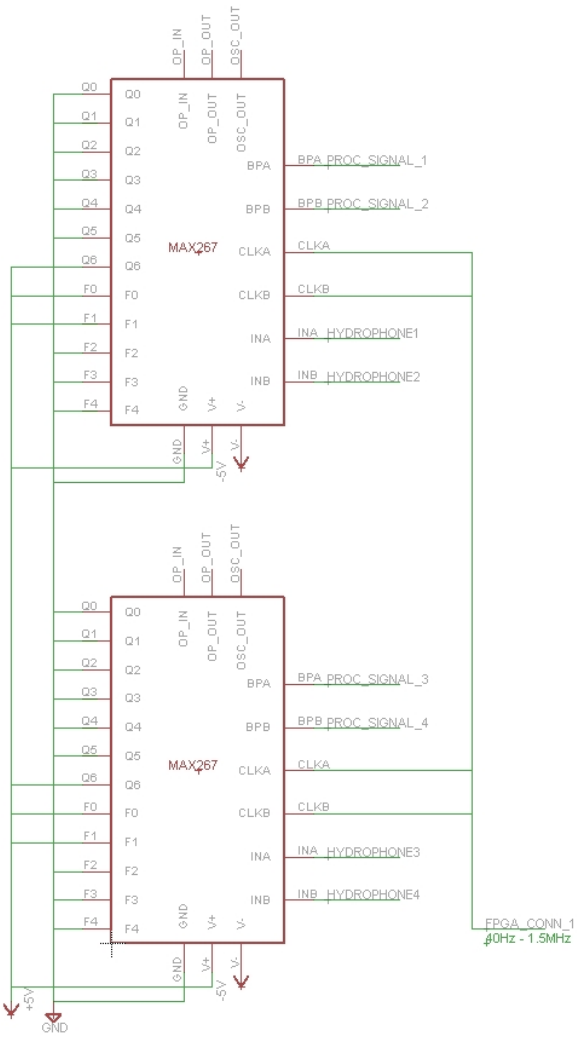


Figure 4.17: The circuit shows the two MAX267 chips filtering the amplified signal. It is configured through the Q0 - Q6 and F0 - F4 pins.

The final section is the amplification/attenuation and shifting stage that prepares the filtered signal for the analog to digital converters. This is implemented through the use of four more DS1803 digital potentiometers and two more MAX414 quad op-amp chips. In this phase the FPGA will be in control of the two digital potentiometers in each DS1803. The addresses are set so that none are the same, allowing for the use of more than one. The first digital potentiometer attenuates the signal as the PROC_SIGNAL passes through the chip. This is then sent through a unity-gain circuit to maintain its strength. In the final step, the signal passes through an adder circuit. This is done by sending a reference voltage and the signal through a single terminal of an op-amp. The output of this will be the added signals. This will shift

the voltage into a positive range. Figure 4.18, shows the circuit discussed in this section. The output is listed as FINAL_SIGNAL, this signal will feed directly into the ADCs and then to the FPGA to be further processed.

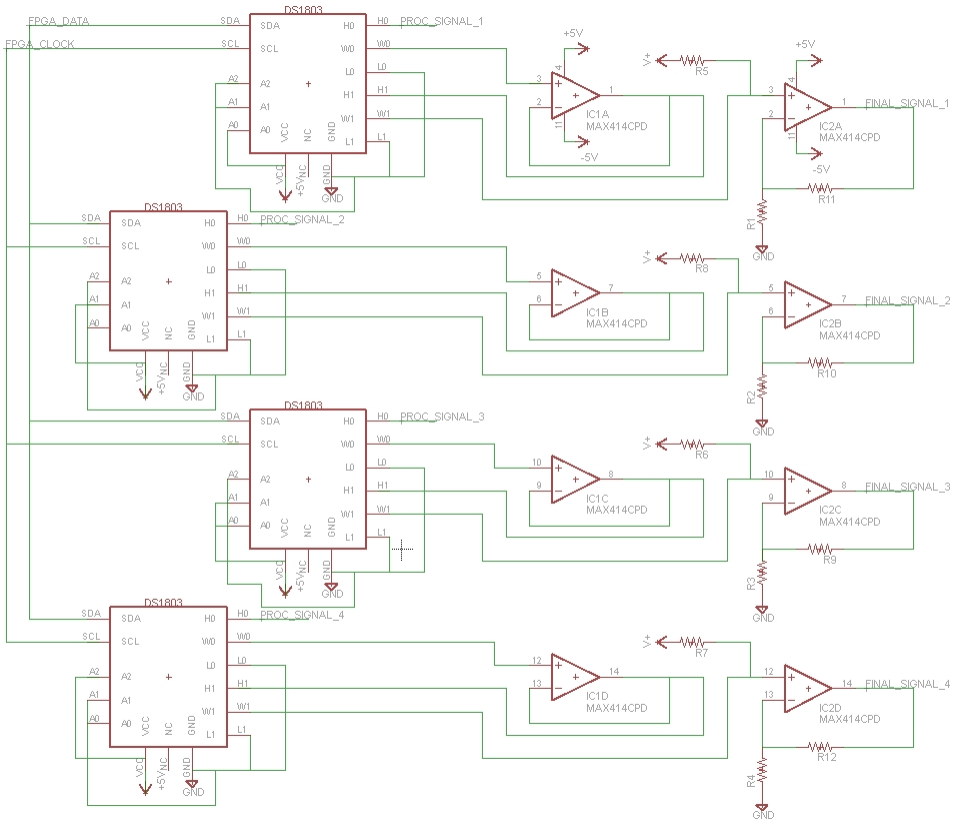


Figure 4.18: This figure shows the components and methods used for the final amplification/attenuation and shifting. This will allow the signals to be used by an ADC.

As designed throughout this chapter, the analog board is meant to take in, filter and adjust for later use, the raw hydrophone signals. This must also be done while not distorting or adding additional noise to the signals. Many components have been examined and many circuits have been considered. The result of this research has been shown in the past few figures. As all design requirements were met, it will be up to testing, to see what the final outcome of the analog board will be.

Chapter 5

Digital Hardware

5.1 Introduction

The hardware selection is one of the most important parts of this project. There are quite a lot of different implementations and combinations of boards we can use to complete our design. Since our project requires the frequency range to be around 20 kHz to 30 kHz, operation voltage to be 12v, 5v or 3.3v voltages, the ability to filter noise signals to about 10 times the pinger frequency(12*4*1,000,000 = 6 MB/s), and processed the vast majority of FPGA boards, Microcontrollers, Analog to Digital devices were immediately ruled out because of their lack of computational abilities. We narrowed down our choices to 4 possibilities, and they will all be described later on in this chapter.

5.1.1 Signal Capture Process and Requirements

We will have 4 hydrophones that capture analog signal, and all the hydrophones can not really do any of the processing jobs. So it would be hard for analog circuitry to do all that. Instead of capturing the data or signal in the analog world with all our hydrophones, we need to convert all the signals into the digital world. So initially, we would have the 4 signals captured from hydrophones. Our goal is to design and use one of the hydrophone signals and then multiply it or copy it 4 times. Because of this, we would be able to use all 4 of them. Fortunately, using this method will simplify our design in many different ways, hence it saves us a lot of time and get we can get full grasp of the pictures of what the signals looks like. To not confuse anyone, one can think of just using one hydrophone and capture all the signals. Once we have captured the signals from the pinger beneath the water, we are going to amplify the signals using one of our top choice boards. The signal we captured is going to be in the millivolt range because it is really attenuate signal. The hydrophone itself does not do any signal amplification because the hydrophones themselves just capture the signal. Therefore, we have to amplify the signal from the hydrophone. Upon completion we will then filter the signal to get rid of different noises for low frequencies and high frequencies to meet our requirements of frequency range at around 20 to 30 kHz. The type of filter we are going to use to filter the signal would be like a band pass filter

that might be configurable depending on the type of pinger that we are looking at. In the competition itself, there will be many different types of frequencies provided for us, from there we can configure it to our needs.

5.1.2 Analog to Digital Overview

This signal capture process has to be done in the analog sense. It will then be process in the digital world. There are so many different approaches to do this. One way to accomplish this is to capture the signals in analog way, and the other way is capture the signal using A/D (Analog to Digital converter) to convert the analog signal and convert it to digital. There are thousands of types and makes of Analog to Digital converters in the market; they all have different specifications for each individual ones. Most of them have a lot of different sampling rates. Some questions we may ask ourselves are how fast can they sample? What type of techniques does it involve? How are the actual sampling works? Base on our initial research, we thought that the Analog to Digital converter part that is provided by a company called Digilent would help us with this project. They actually make P mod module that will be use as Analog to Digital converter to process the signals. The price is really good for the performance and part that is involved. Also, this product shall provide us with high enough frequency.

5.1.3 Sampling Technique

We need to use Nyquist Sampling theorem to sample the signal that is already been sampled. This technique shall help us reconstruct the signals from an infinite sequence of samples if the sampling rate exceeds 2B samples per second, where B is the highest frequency that was showed in the original signal. Even though this technique is perfect reconstruction of the signal in the mathematical theory, but it is not 100% match in the real world applications. We can only use this for the approximation, which will come pretty close to the final result. This sampling theorem only provides sufficient condition, but not necessary one for perfect reconstruct. So using this technique, we have to sample our frequency to twice as fast or at least twice as fast as the original signal or else we will get some type of Anti-aliasing and we won't be able to capture the entire signal. For example, if our highest frequency is going to be 35 kHz or so, then we need to at least sample 2x35 kHz, which is 70 kHz of frequency in order to get the proper result. We can also sample it to 3 or 4 times or 10 times or even higher sampling rate than the original one to get the best and most accurate data.

5.1.4 Different Design

There are also other steps we can take and use it, such as Digital Signal Processing techniques. There are a few different companies that have the Digital Signal Processing board that we can use to meet our requirements. Our research found that there is a device called Blackfin Digital Signal Processing; although it is expensive, but it is made for filtering out the noise and sampling the signals. Compare the BlackFin

processor and FPGA, FPGA has certain limitations. For example, FPGA can be used as Digital Signal Processing but does not have enough computing powers for our complex mathematics. One of advantages of using FPGA is it can be used to program to do whatever users need; it is very scalable and flexible with user's needs and can be dynamically changed. Whereas BlackFin Processor is pretty much set in stone and can only be use for signal processing, it cannot do all the features FPGA can do, but it does the digital signal processing a lot faster and better than an FPGA can do. BlackFin has specific tools, software features, hardware features designed for just for the signal processing features. For the most part, BlackFin microprocessor do a lot of processing in parallel, just like graphic card where there are a lot of good data manipulations.

5.2 Challenges

5.2.1 Mathematic Challenges

There are always risks associated with our system; these risks could come when interfacing our various components together to function as whole system unit in sync. Risks could be the interface between hydrophone with FPGA or Central Processing system. Another potential risk is going to be the position calculation using either multilateration or triangulation technique. Simply because the technique involves the complicated mathematic problem solving and the group has to determine the exact coordinate to use in the derived equations. We have to make sure whatever devices we choose have to have enough computing power and processing power to compute all the signal calculations and locate the coordinate location.

5.2.2 Hardware Selection Challenges

Since we have never work with the BlackFin microprocessors before, it would be a big challenge for us. This means we have a lot of research homework and learning to do. We are a lot more familiar with FPGA platform and have programmed the device with Verilog software coding that drives the FPGA. In both hardware and software side, we are pretty much set if we choose FPGA. If we do use FPGA, then Xilinx ISE is good type of software to use since we have used it in class lab before and understood all the features it has in it. So these are pretty much two of the largest hardware pieces that we can choose from.

5.2.3 Alternative Devices

Some of the other alternative parts we can consider could be the ARMmite microcontroller and RabbitCore. One of the risk or challenge posed by using ARMmite microcontroller come from the fact that this device does not have any wireless embedded on board, this would mean we will not be able to transfer any video or audio data from the sub system to the base computer. Another risk for using ARMmite

microcontroller is the processing power of this microprocessor is limited. This mean less process speed, hence the system would be slow and hard to accomplish our goal. RabbitCore module is another device can be considered, simply because it has almost all the functionalities we need. However, one of the disadvantages of using this device is pretty much similar with the ARMmite controller. It does have an Ethernet port to transfer real time data, but it does not have the wireless capability.

5.3 Hardware

5.3.1 Nexys2 FPGA Board

Nexys FPGA Board is one of our top choices of the FPGA board, it is a very powerful digital system design device. It has Xilinx Spartan 3E FPGA build in the device itself. It includes a 16 Mbytes of the fast SDRAM along with 16 Mbytes of Flash ROM, this board has the Xilinxs 32-bit RISC MicroblazeTM embedded on-board. This board can accomplish most of the designs without any complications, thanks to the building high-speed USB2 with fast transfer rate. It has many of the I/O (Input/Out) devices, expansion ports, and many of the data ports allows for more freedom of input and output the data. The cost of the board is inexpensive; it cost around $99 with Academic discount. To utilize and communicate with the board, we need to use the free program from Digilent called Digilent Adept Suite Xlinx Spartan-3E FPGA.

The figure 5.1 below shows a simple picture of what the device looks like, all the parts including USB2, 16Mbytes of fast Micron PSDRAM, 16MB Intel StrataFlash Flash R, Flash Rom are embedded on the board.Figure 5.2 shows the FPGA board with higher level schematics and it has a build in 1200k of the Logic gates USB2 port provide the board power, really high speed data transfers, and easy to configure the device. This device ideal for all battery power applications. Some of the features offers are:

- Xilinx Spartan-3E FPGA, 500K or 1200K gate
- USB2 port providing board power, device configuration, and high-speed data transfers
- Works with ISE/Webpack and EDK
- 16MB fast Micron PSDRAM
- 16MB Intel StrataFlash Flash R
- Xilinx Platform Flash ROM
- High-efficiency switching power supplies (good for battery-powered applications
- 50MHz oscillator, plus a socket for a second oscillator
- 75 FPGA I/Os routed to expansion connectors (one high-speed Hirose FX2 connector with 43 signals and four 2x6 Pmod connectors)
- All I/O signals are ESD and short-circuit protected, ensuring a long operating life in any environment
- On-board I/O includes eight LEDs, four-digit seven-segment display, four push-buttons, eight slide switches

Figure 5.1: Figure of Nexys2 FPGA board with all the I/, Data Ports and Expansions. Reprinted with permission from Digilent Inc., see the Appendix Section A.2.

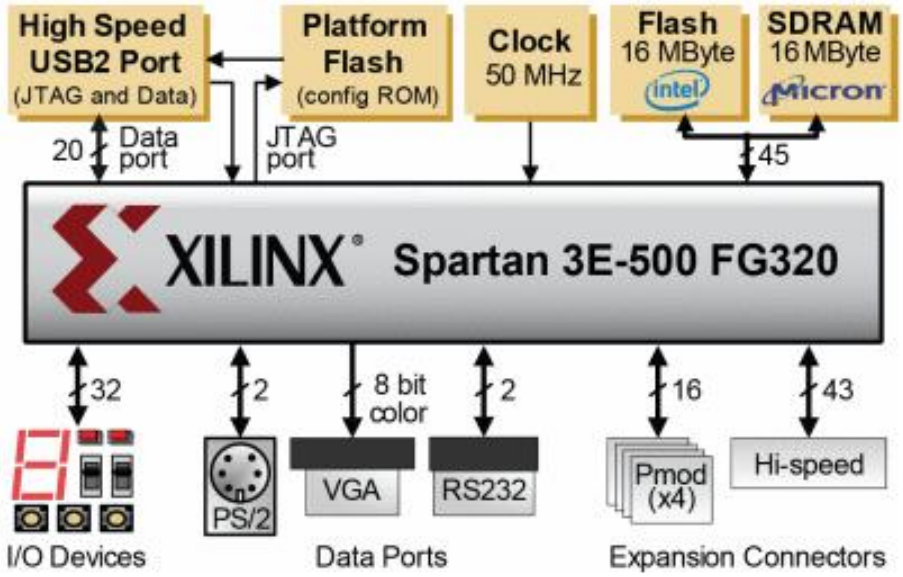


Figure 5.2: A High level block diagram of Nexys2 FPGA with Spartan 3E. Reprinted with permission from Digilent Inc., see the Appendix Section A.2.

5.3.2 PmodAD1

One of the A/D converters we can use is the Pmod AD1. PmodAD1 is an Analog to Digital Module Converter board that can be use to connect to the Nexys2 FPGA board. Figure 5.3 below shows the device Pmod and its pin connectors, which has two 12-bit A/D inputs and this device can help us converts the signals at a maximum capability of rate of one million samples per second, this will be sufficient enough for

our device requirement. This device has 6 pins-header connectors, and the device is really compact which is smaller than 1 square inch, it come with a 6" 6-pin cable and a 6 pin header and it is ideal for any applications and can be located at any signal source. We can utilize this device to convert analog input signal from 0-3.3 volts to a 12 bit digital signal of 0 to 4095.

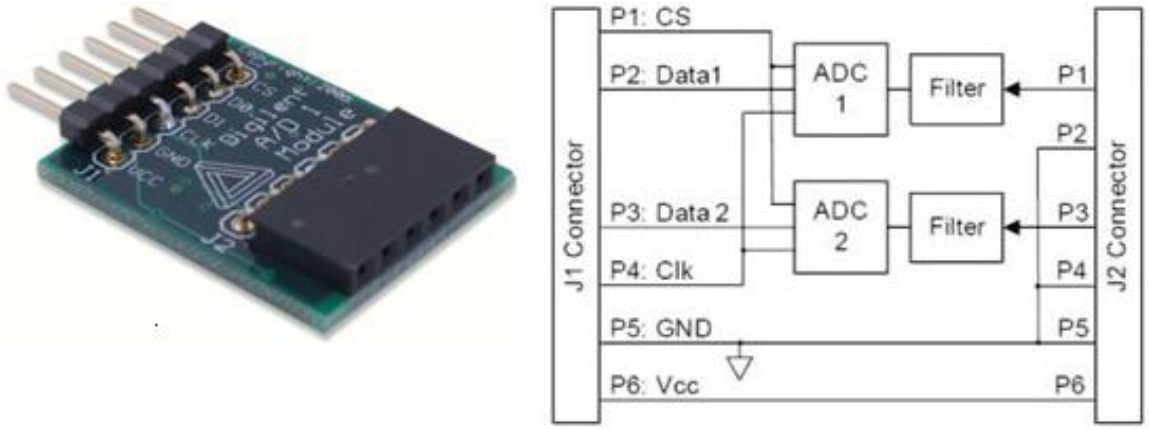


Figure 5.3: Image of the analog to digital peripheral module (PmodAD1) and from Digilent Incorporated. Reprinted with permission from Digilent Inc., see the Appendix Section A.2.

Some of the features of Pmod is provided below:

- Two 2-pole Sallen-Key anti-alias filters
- Two simultaneous A/D conversion channels at up to one MSa per channel
- Very low power consumption
- Small size (0.95" x 0.80")

Implement the Device

We can implement the device by connect the Nexys2 FPGA board with Pmod AD1 Analog to Digital converter. First we connecting the J2 Connector from Pmod to Nexys2 FPGA board by align all the pins. Pin p1, p2, p3, p4, p5, p6 from Pmod will connect to pin connectors on Nexys2 board with connector name Pmod 2x6 pin JA1, JA2, JA3, JA4, GND, VCC3V2. Figure 5.4 below shows all the available pins for the Nexys2 FPGA board has, which are four Pmod inputs from Nexys2 FPGA board, but we only need to connect 2 the Pmod AD1 and DA1 converter to it in our project.

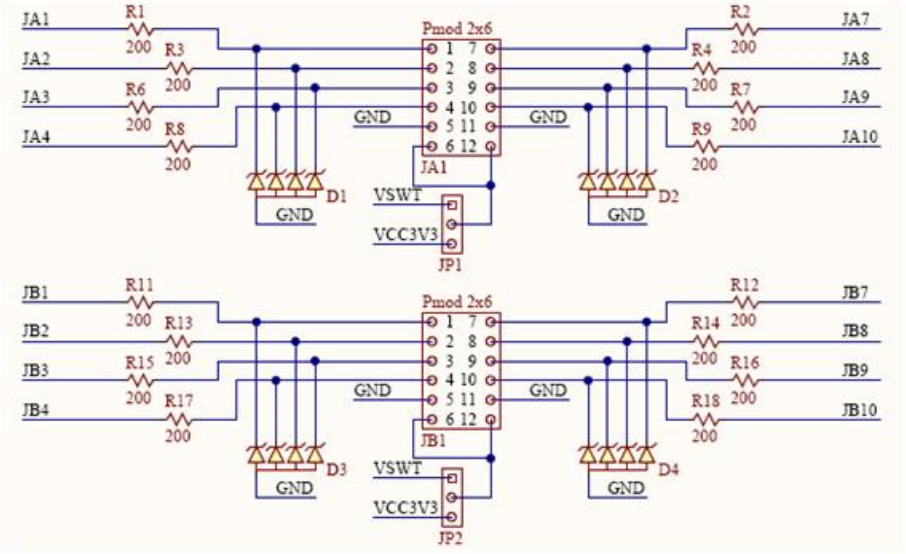


Figure 5.4: Figure illustrate two of the four connectors will be used for FPGA and Pmod connection. Reprinted with permission from Digilent Inc., see the Appendix Section A.2.

Disable Unused RAM

There are a few connections we do not really need to use in our design, such as Intel/Numonyx StrataFlash and the Micron Pseudo-static RAM. We have to disable these devices to make sure they do not interfere with all the rest of the logic parts. We could use the following command in Verilog code line language to achieve this goal:

```
1 assign {St_ce_bar, St_rp_bar, Mt_ce_bar, Mt_St_we_bar, Mt_St_oe_bar}
    =
2 5'b11111;
3 NET St_ce_bar LOC = R5; NET St_rp_bar LOC = T5; NET Mt_ce_bar LOC =
    R6;
4 NET Mt_St_oe_bar LOC = T2; NET Mt_St_we_bar LOC = N7;
```

5.3.3 Jumper and Power Supply Configuration

Peripheral Modules Setting

Since our project requires the use of Pmod AD1 device to connect to the FPGA board,we have to set the jumper (JP1, JP2, JP3 and JP5) next to the Peripheral Module connectors. Figure 5.5 below shows this setting need the power source from the module, so we need to set the jumper on the VSWT side, not the 3V3 side.

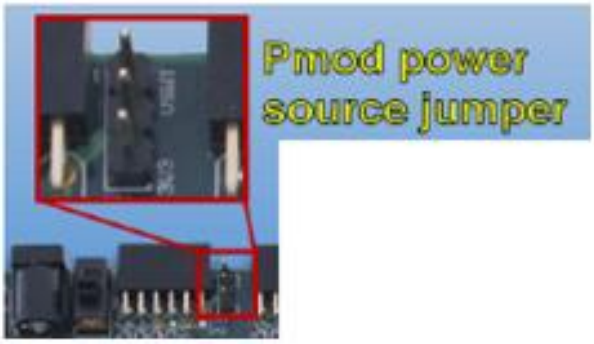


Figure 5.5: Figure shows the correct setting of the power jumper to VSWT side. Reprinted with permission from Digilent Inc., see the Appendix Section A.2.

PS2 keyboard/Mouse Interface Setting

We did not plan on use the PS2 Keyboard/Mouse interface, so we can disable this device. Figure 5.6 shows there will be a jumper named JP10 next to the PS2 connector located at the South West corner of the board. We will set the jumper to 3V3 side to disable it, not VSWT side.

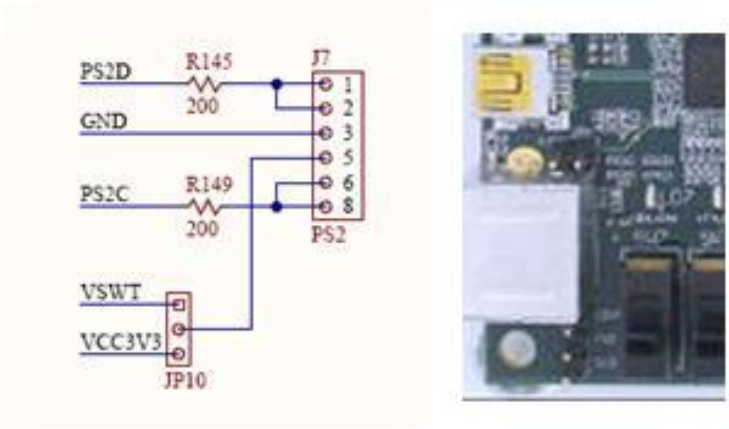


Figure 5.6: Figure shows how to disable the PS2 Mouse/Keyboard Interface by set the jumper JP10 to 3v3. Reprinted with permission from Digilent Inc., see the Appendix Section A.2.

JTAG Pins

We plan to use just the USB Serial Interface to communicate with the board, so the JTAG cable will be unnecessary in our case. We will disable the device replace the jumper pin on the JTAG header in the North East Corner of the board(Figure 5.7), cover the jumper on pin 2 (TDI) and pin 3 (TDO). We will move the spare jumper from the jumper storage location JP8 at the south East corner of the board and use the jumper to cover pin 2 and pin 3.

Figure 5.7: Figure shows the JTAG jumper setting should cover Pin2 and Pin3. Reprinted with permission from Digilent Inc., see the Appendix Section A.2.

Power

Power Jack will be use to power the board and USB cable to provide the data transfer. Figure 5.8 shows that We have to set the power jumper to WALL side, then connect the USB cable to the board (small USB side) and connect the other side of USB (the regular flat side) to the computer. After the setting is done, we can switch on the power anytime by switching the device on in the North West corner of the board.

Figure 5.8: Set the jumper to wall and connect the power cord to power the board. Reprinted with permission from Digilent Inc., see the Appendix Section A.2.

On-board Self test

The board has a build in self test, so we can do a self test to make sure all the functions are working. To do this, we need to set the jumper JP9 in the middle of the North East corner of the board to ROM side, but to leave the JTAG side open. After the setting is done, we can switch on the power and press the reset bottom(Figure 5.9) at the North East corner to let it run. The seven segment display(Figure 5.10) should show a 4 Digit character and will display run for a short period of time, and then it should display PASS or 128 alternatively. Unless if there is something wrong with board itself, the PASS character should display on the screen. When the board was send out by the manufacturer, it has pre-installed test program build into the Platform Flash ROM. The self test program can also perform the snake game to make sure the board itself is not defected. When we push the 4 pushbuttons, the self test sequence should be exited and the screen should enter the snake game mode. Pressing 4 of the bottoms at the different time will show the different sequence of the snake

game and we shall see it. Alternately, we could perform the self test using the build in software Adept and click on Test tab.

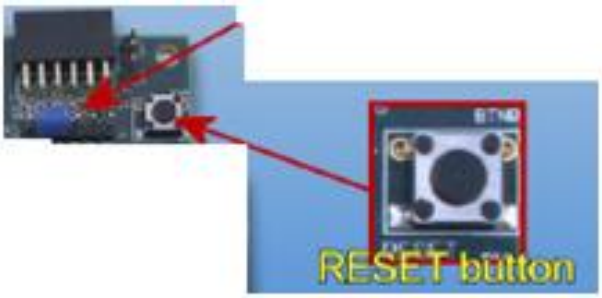


Figure 5.9: Set the jumper JP9 to ROM setting to get the self testing to work. Reprinted with permission from Digilent Inc., see the Appendix Section A.2.

Figure 5.10: The seven segment display with 4 pushbuttons. Reprinted with permission from Digilent Inc., see the Appendix Section A.2.

Configure Nexy2 FPGA with the Design

As mention before, figure 5.11 shows the connections of USB cable to manage all the data transfer between the board and computer itself. We will connect the USB cable to the USB connector on the board and computer USB port, and then we can start to download the bit files to the board. For jumper JP 9, we had it set on the ROM label when we perform the self test. Now we shall move the JP 9 setting to JTAG side and leave it there for the board to be ready to use. After the design sets up on the computer, then downloading process shall be started. Before we proceed, software tool shall be downloaded that was provided by the board that is called Adept. This tool with provide a sample test bit file named test_nexys2_verilog.bit.

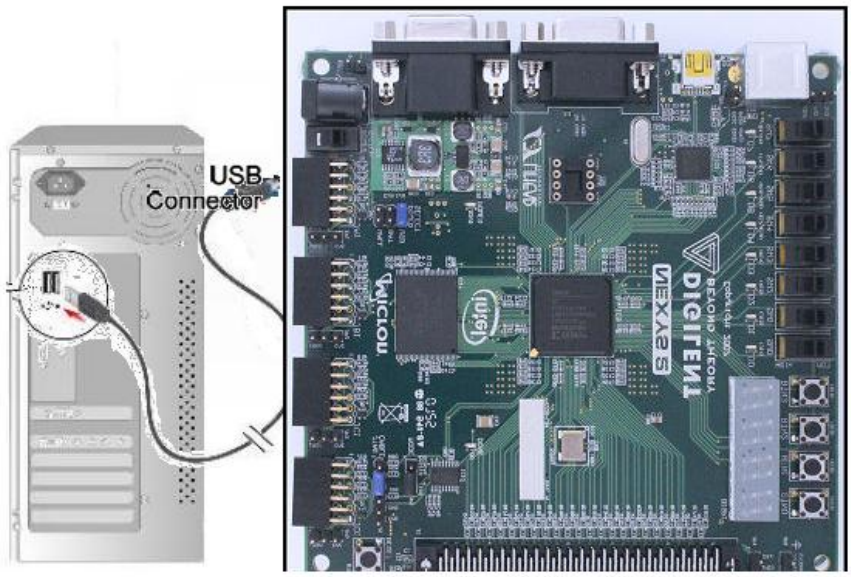


Figure 5.11: Connect USB2 from PC to Nexys2 FPGA board. Reprinted with permission from Digilent Inc., see the Appendix Section A.2.

Set up the board with New PC

For the first time use of a new computer, we shall correctly configure the computer itself and board itself to the setting by using the sample bit file with steps below:

1. Make sure the Jumper JP 9 was set it to the right position as it was mention on the paragraph above.
2. Install the Diligent software Adept 2.1 from Diligent website:

```
1 http://www.digilentinc.com/Products/Detail.cfm?NavTop=2&NavSub
    =69&Prod=ADEPT
```

3. Connect the board with new computer and switch the power on.
4. Go to Start menu, Programs, Digilent, Adept, Adept. Figure shows the where the software is located at(Figure 5.12).

Figure 5.12: Figure shows where the Adept program is located on PC. Reprinted with permission from Digilent Inc., see the Appendix Section A.2.

5. The Adept tool should recognize the FPGA board, we should hit browse tab to import the test_nexys2_verilog.bit file.
6. Click program button to configure the FPGA.
7. Figure 5.13 shows the 7 segment LED. A walking LEDs patterns should appear on the 7 segment display. The four characters should display 8 bit hex value set on the 8 switches. We shall be able to control it with the push buttons and switches to get the good ideas of how the sample bit files are programmed.

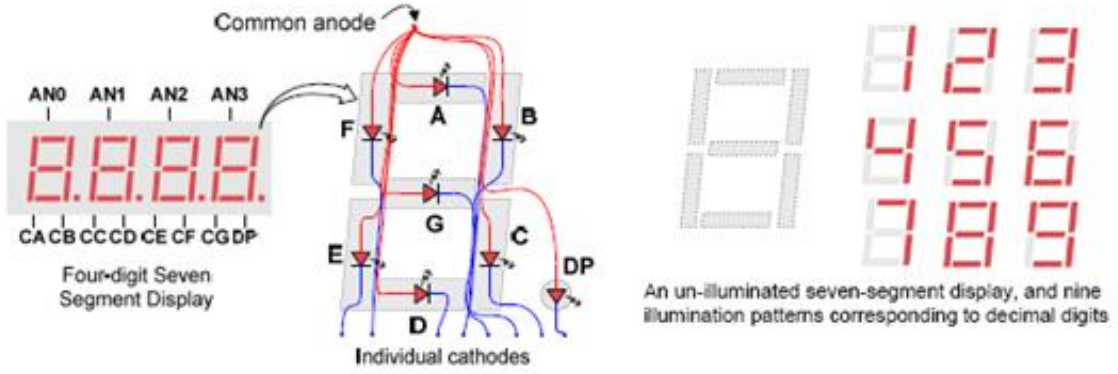


Figure 5.13: Figure shows the 7 segment LED display. Reprinted with permission from Digilent Inc., see the Appendix Section A.2.

5.3.4 AD7298 A/D Converter

An alternative solution of A/D converter we could consider is AD7298. The AD7298 is a 12 bit resolution high speed and low power consumption analog to digital converter. It has a temperature sensor embedded in the chip, and we can use this to collect the data of temperatures under the water for our sub system. The device itself is operates at the 3.3 v power supply, which meets our initial power requirement. The sampling speed is really important for our requirement; and since this device has the capability of one million samples per second, which fits our design. This device has the capability of frequency can be inputted up to 70 MHz, we can use the pre-programmed sequencer for this device to do the signal conversion. The low power consumption feature makes this chip an ideal device for our design.

Connection

The device has 8 single ended analog inputs that are multiplexed into the on chip track and hold. The input acceptance of each input is 0v to 2.5v. So we will use pin 1 to pin 4, and tie the pin5 to pin 8 to the ground GND1 to avoid all the noise signals. All the input signal data will be processed and output to the Dout serial data output. Each time the SCLK input is on the falling edge of the clock, this signal will be output it to the serial data output Dout. The data stream of this device has four

addresses bit to indicate which channel is the signal come out of. The output will be in the binary form for the voltage channels and 2s complements for the temperature sensor result. We will use the power supply input pin Vdrive pin to determine which interface operates base on the voltage that is driven by. Ground all the external input signals to the GND1 since it is an internal reference, all the GND1 pins should configure to be the same potential and shouldn't be more than 0.3v apart. Unlike GND1, the GND is the reference point for all the analog and digital circuitry and should be connect to the GND plane of the system. Same as GND1, all the reference points should be at the same potential and must not exceed 0.3v. We can program the content of our control registers to select the different modes, which increase our design flexibility. The device contains a power on reset circuit(Figure 5.14), we can use this reset to set all the settings to 0s for default and configure it for normal mode of operation. The time it takes to power up the device is about 100 us when is using internal reference. In order to use the reset, we must enable the reset operation TD pin should set to low, and this is asynchronous to the clock.

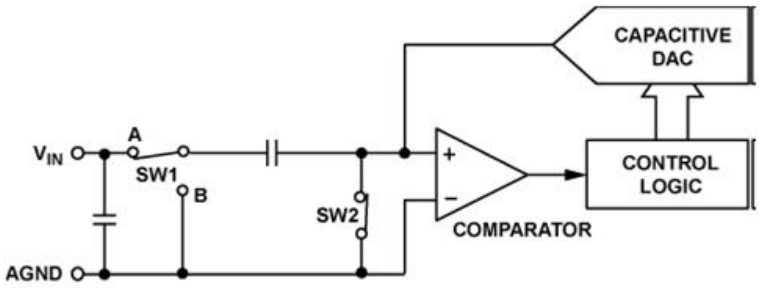


Figure 5.14: Figure shows the Acquisition Phase of the A/D. Reprinted with pending permission from Analog Devices Inc., see the Appendix Section A.2.

Conversion

Figure 5.15 below shows schematics for the operation conversion, SW2 is close and SW1 will be in position A. The comparator will be held in balance condition and sampling capacitor will acquire the signals on the selected Vin channel. Soon after the ADC starts the conversion, SW1 will be in position B and SW2 will be open. The two diode D1 and D2 provides the electro static discharge function, so we can have peace of mind when we collect the analog signals. The analog input shall not exceed internally generated voltage of 2.5v by more than 300mv. The diodes will become forward bias and conducting current into the substrate. Figure below shows the resistor R1 component made up of the on resistance of a switch and also shows the on resistance of input multiplexer. This made up the total resistance of TBD ohms. The C2 is a sampling capacitor that samples the input signals. Since our design is using the DC inputs, we dont need to remove the high frequency components from the analog input signals.

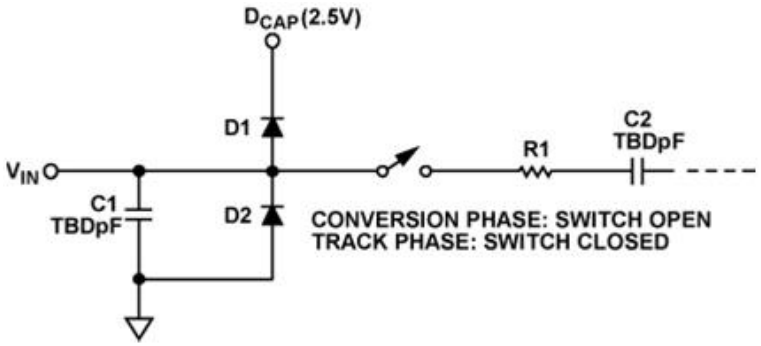


Figure 5.15: Figure shows equivalent analog circuit. Reprinted with pending permission from Analog Devices Inc., see the Appendix Section A.6.

5.3.5 Xilinx ISE WebPack Design Software

Xlinx ISE WebPack design software will be used in our design simply because it is free and it is design to be use with Nexys2 FPGA Spartan 3E; it has a lot of features with full featured FPGA design. We can combine this software with our FPGA board to bring out the full features of the hardware and software. The advantages of using this WebPack Design Software is users can implement the hardware with almost any operating systems, such as Linux system, Windows XP operating system (Figure 5.16 and 5.17 shows the capability of the different versions of Windows OS), and Windows Vista Operating system. The ISE Design Suit is compatible with the FPGA Nexys2 Spartan 3E board according to the Specification table.

ISE Design Suite Device Support	ISE WebPACK	ISE Design Suite Logic Edition Embedded Edition DSP Edition System Edition
Spartan™ Series	Spartan-3: XC3S50 – XC3S1500 Spartan-3A: All Spartan-3AN: All Spartan-3A DSP: XC3SD1800A Spartan-3E: All Spartan-6: XC6SLX4 – XC6SL75/T XA (Xilinx Automotive) Spartan-3: All	Spartan-3: All Spartan-3A: All Spartan-3AN: All Spartan-3A DSP: All Spartan-3E: All Spartan-6: All XA (Xilinx Automotive) Spartan-3: All
CoolRunner™ XPLA3 CoolRunner-II™ CoolRunner-IIA™	All	
XC9500™ Series	All (Except 9500XV Family)	

Figure 5.16: Figure shows equivalent analog circuit. Reprinted with pending permission from Xilinx Inc., see the Appendix Section A.5.

Product	XP Professional		Vista Business	
	32-bit	64-bit	32-bit	64-bit
Design Entry and Implementation Tools (ISE Foundation™)	Yes	Yes	Yes	Yes
ISE Simulator (ISim)	Yes	No	Yes	No
ISE WebPACK™	Yes	No	Yes	No
ChipScope™ Pro and ChipScope Pro Serial I/O Toolkit	Yes	Yes	Yes	Yes
Embedded Development Kit (EDK)	Yes	No	Yes	No
System Generator for DSP*	Yes	No	Yes	No
AccelDSP™ Synthesis Tool*	Yes	No	No	No
ModelSim Xilinx Edition-III (MXE-III)	Yes	No	Yes	No

Figure 5.17: Figure shows windows capability. Reprinted with pending permission from Xilinx Inc., see the Appendix Section A.5.

This ISE software is most desired software solution for FPGA and CPLD design; it includes and uses the HDL language to synthesis and to simulate. It also has the implementation, JTAG programming, and device fitting features. The ISE WebPack Design Software brought the high performance and it is free with complete functionalities all around. It provides instant access to all the excellent features. This software also can provide us with high productivities with its free update and error, and free downloading with its single file installation. Table 5.18 gives detail features of the Xilinx ISE WebPack:

Features
A downloadable PLD design for both Microsoft and Linux
The fastest timing closure with Xilinx SmartCompile
Complete, front-to-back design environment
Integrated HDL verification with the Lite version
The way to get started with productivity, performance, and power
Easily upgradeable to any of the ISE Design Suite E

Figure 5.18: Most of the Features Xilinx ISE Webpack offers

Software and Hardware Recommendation Requirements

The Directory permissions must have the requirement of write permissions must exist for all directories containing design files to be edited. The monitor must have 16-bit color VGA with minimum recommended resolution of 1024 by 768 pixels. The drive must have DVD-ROM for ISE Design Suit. The system must have an available parallel, or USB port appropriate for the Xlinx programming cable, the installation of the cable driver software required Windows XP Pro SP1 or newer version, or Windows Vista Business version, or else cable will not be able to function properly if you are not using these types of operating systems.Figure 5.19 shows the table of the Recommendation of using the software. Even if our system meets the software requirements, it does not mean we will be successful without the hardware requirements Some of the hardware recommendations are giving in the table below.

Recommendation	Notes
2.00 GB of RAM	
600 MB of hard disk space	Minimum Requirement
Xilinx Hardware Co-Simulation Platform	Required for the Hardware Co-Simulation Flow

Figure 5.19: Figure shows minimum requirements and software recommendation. Reprinted with pending permission from Xilinx Inc., see the Appendix Section A.5.

5.3.6 ISE WebPack with Nexys2 FPGA

User Define I/O

The fact that Nexys2 has many of input devices, output devices and data ports make our design a little bit easier, simply because we dont need any other components. The 4 push buttons in the figure 5.20 are set to low and is driven high when the push button is pressed. The 8 slide switch can generate a high or a low depend on the up and down position. The 4 push button and 8 slide switch input are design to protect again short circuit by placing the resistors in the series. We have to be cautious that we do not assign the FPGA pin to a push button or define slide switches as the output, because this will cause the circuit to be shorted. As mentioned previously, the board contains 4 digit seven segments LED display.

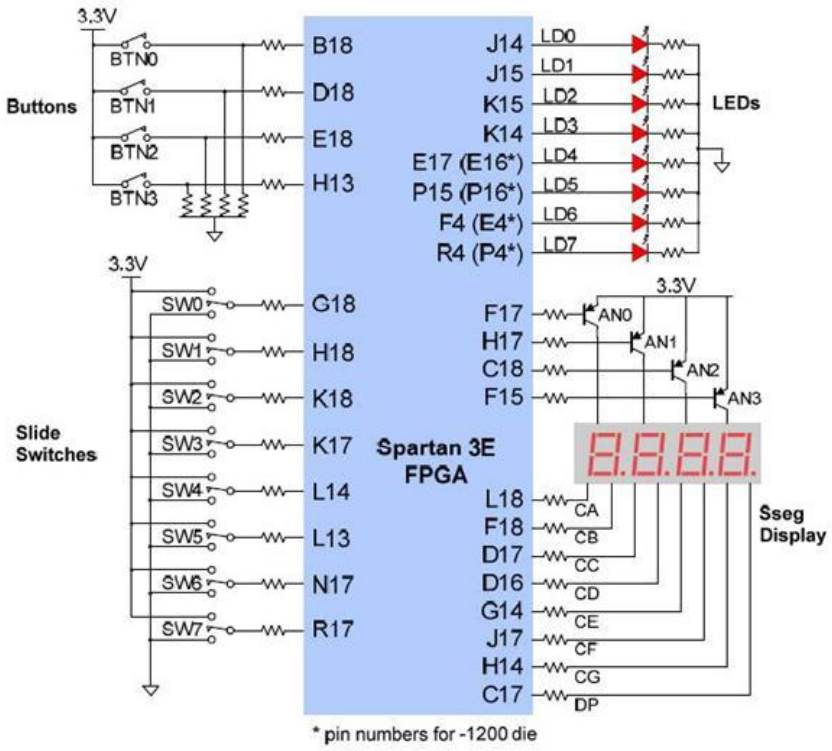


Figure 5.20: Figure shows the 4 push buttons and 8 switch buttons resistors are connect in series. Reprinted with permission from Digilent Inc., see the Appendix Section A.2.

Configuration Using ISE WebPack

In order to configure the device with the ISE WebPack, we would need to test it out by write a code to implement the board to make sure it will communicate properly. We have to create new project and specify the project name, pick a location without any spaces in the path. If there is space in between, then sometimes the ISE software might not behave properly. The next step could be targeting the Nexys2 FPGA board and leave all other options unchanged, then pick the VHDL module option and choose the VHDL file that we designed to begin the design process. We can then assign the ports, inputs and outputs to the design pins, otherwise the software will assign the pins to random places and this will cause problems. The way pins are assign is described in the .ucf (Universal Constraints File). After we have completed the in assign,

5.3.7 BlackFin DSP Processors

BlackFin Microprocessor Overview

BlackFin microprocessor has a lot of library and tolls for process transforms of the signal, which will help us with our design project. It is very neat to use it, simply

because BlackFin microprocessor has all the capability of running an operating system on the processor, it has on-board Linux based operating system(Figure 5.21 shows the architecture of the CPU). We shall be able to run more than just the digital signal processing. As mention before, we can also use this processor to process and calculate the complex math that will involve in our design project. Basically the math will involve actually triangulating the pinger and allocating its exact position. So this is one of another best feature that is offer by this board, but once again the cost of this board is one of our biggest concerns. The cost of the BlackFin microprocessor is about $26 per unit, but the Evaluation board itself cost about $900 to $100 a piece. We would need to purchase this Evaluation kits if we want to utilize the BlackFin microprocessor. The Evaluation kit will have a development board, along with A/V extender and external FPGA E-Z extender, however they cost about another $600. The evaluation kit will have DSP on it and external chips that come with it, such as RAMS, ROMS, couple different other things, as well as on board Analog to Digital converters, which we can rather get the Analog to Digital devices and Nexys2 FPGA board from Digilent. We can actually tie into our hydrophone signal. They always handles different types of network and communications, it would be an easy way to communicate with our computer base system for the Sub vehicle.

BlackFin DSP processor is one of our top choices because it offers software flexibility and scalability for quality applications, this processor can perform multi-format audio, video, voice and image processing, multi-mode baseband and packet processing, control processing, and real time processing security. Figure below shows the architecture of the Blackfin core:

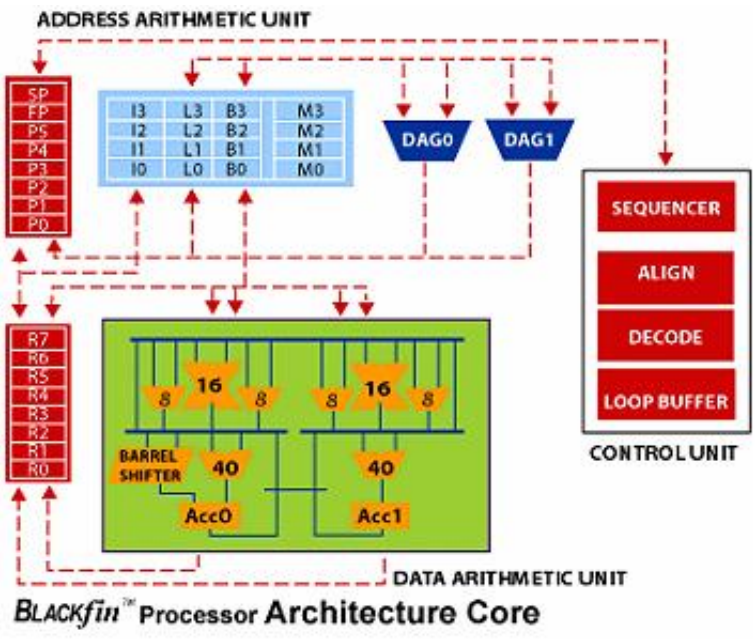


Figure 5.21: Figure shows the architecture of the Blackfin Core. Reprinted with pending permission from Analog Devices Inc., see the Appendix Section A.6.

Model ADSP-BF527

One of the models we are considering to use is the ADSP-BF527 processor(Figure 5.22 below show the functional block diagram for the processor). This processor has the processing power of 16/32 bit embedded processor core, which meets our requirement of at least of 12 bit resolution. Below are a list of the full features this processor offers:

- Lockbox Secure Technology: Hardware-enabled security for code and content protection.
- Blackfin Processor Core with up to 600 MHz (1200 MMACS) performance
- 2 dual-channel, full-duplex synchronous serial ports supporting 8 stereo I2S channels
- 12 peripheral DMA channels supporting one- and two-dimensional data transfers
- NAND Flash Controller with 8-Bit interface for commands, addresses and data.
- Ethernet 10/100 MII interface
- Memory controller providing glue-less connection to multiple banks of external SDRAM, SRAM, Flash, or ROM
- 289-ball, 12x12 mm, 0.5 mm pitch mini-BGA (Commercial temperature range 0C to +70C)
- 208-ball, 17x17 mm, 0.8 mm pitch mini-BGA (Commercial temperature range 0C to +70C; Industrial temperature range -40C to +85C*) * 533 MHz max operating speed

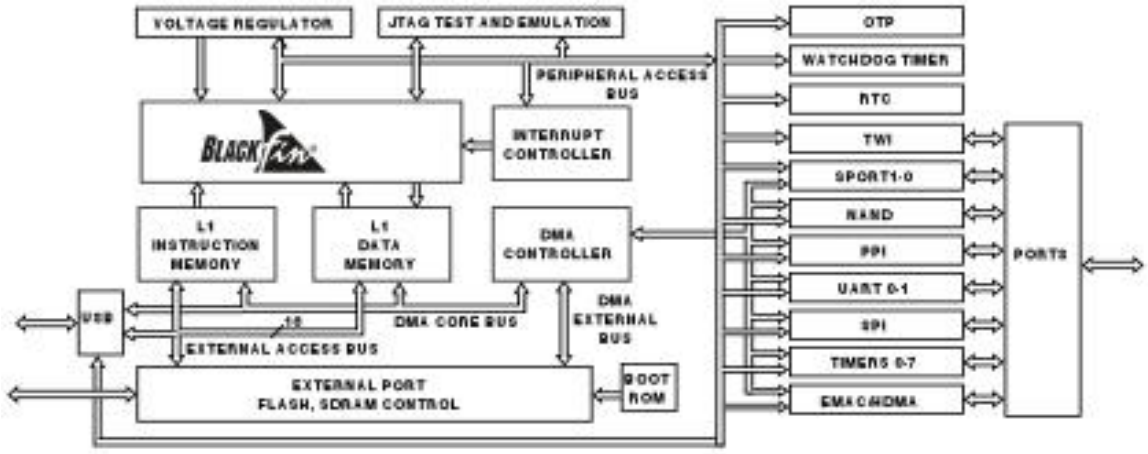


Figure 5.22: Figure shows function blocks of the Blackfin core. Reprinted with pending permission from Analog Devices Inc., see the Appendix Section A.6.

Processor Speed

In order to calculate the position of the pinger, our research has several This core offers the core speed of 600 MHz, which means more than enough speed for our

project.

Wireless Capability

The peripherals enable the connections between the WiFi 802.11 a/b/g modules or Ethernet. The low power means more power efficiency on digital signal processing. We can use this feature to transmit and receive video and audio data wirelessly.

Cost effective

One of our biggest concerns is the budget and cost effective, the unit cost of this Black fin processor is about $18. So the chip itself is not a big concern. However, if we get this chip, then we would need to purchase the Evaluation board ADSP-BF527 EZ-KIT lite, Blackfin A-V EZ-extender, BlackFin FPGA EZ-extender, and all the software. Just the Evaluation board itself will cost about $1000 a piece, and this will be a budget problem for us.

Wireless IP protection

The ADSP-BF527 has a special feature that helps secure the access to the program code. This way, we wouldn't have to worry about the security issues during the competition.

Performance

Unlike the Digital Signal processing and micro controller perform by itself individually, the Blackfin ADSP-BF527 architecture combines both functionality in one Chip. This design provides improvements in performance, programmability and power consumption over traditional DSP and RISC architecture designs. This special design saves a lot of design spaces for our project.

Audio

Most Analog devices today in the market offer a wide range of portfolio processor that feature comprehensive audio centric peripherals. This board provides the precise audio processing for any applications includes automotive audio, portable audio devices, high quality home theatre systems and professional audio equipments.

Security and Surveillance

This processor contains a wide range of device portfolio along with all the best performance software out there. It supports third party network, supports and all the extra features. One of the main reason why we make this processor our number 1 choice is this processor has the ability to delivers a big amount of performance in the area of digital signal processing.

Processor in Automotive

This processor can be made for automobiles audio system as well, it supports wide range of advanced driver assistance product to market. One of the advantage of this processor is it offers low power consumption, really high performance, code secure architectures supported by the company. The Blackfin is currently used in Audi A5 vehicle, this processor powers the A5s multimedia interface radio, it delivers the phenomenal CD quality to the drivers and passengers. This processor has the optional surround sound amplifier delivers the top the line audio quality. Base on our research, this Blackfin processor was ideal for our project simply because it offers us with best outstanding performance, scalability, and connectivity to handle audio decoding, and digital audio broadcasting. This processor is an ideal processor for automotive developers, simply because the processors fast performance and connectivity they need for electronic design applications. Because of the fast growing technology, software flexibility is always an issue. However, the Blackfin processor offers extensive software flexibility since this is critical for automotive applications because media formats and communications standards are always changing.

Because the Blackfin processor can perform both control and signal processing makes it capable of handling both audio processing and external device management. This processor used in Audi enables users to control audio sources, such as USB devices, and Ipod device through audio system. This processor can enable the Ipod interface on the audio system, it displays the song's name and title on the screen so the users can view it and control it through the audio system. The Blackfin earn its name because of its outstanding performance, rich audio feature set and it is the foundation of surround sound amplifier. This processor was used by many other applications too, such as network Ethernet I/O devices, it used the processor's feature of low current drawing and reasonable price to process the signal. The Blackfin processor process the control of traffic on the switch through its build in fast Ethernet Medium Access Controller (MAC), this support both 10BaseT and 100BaseT based operation. At the speed of 600 MHz and on-board memory of 132 Kbytes, it combines with MAC controller for signal processing, produce advantage of a clean single instruction set architecture. Hence, this cut the cost of materials and simplifies the hardware and software implementation, this also eliminates the need of separation of digital signal and control signal. One of the top end surveillance camera company utilized the Blackfin processor to process the top end image. Compare to the previous uses FPGA and microcontrollers, the BlackFin Processor is the way to go.

Processor in test and measurement

We can choose this processor because it offers the capability of best in class data acquisition and data analysis for test and measurements. The processor can support from precise signal procession to many sensor supporting. Our initial intend was using Fast Fourier Transform algorithm to compute the discrete Fourier Transform and its inverse. By using this method, we can break down the signal sequence of values that we pick up from hydrophone and split them into components of different frequencies,

and using the Fast Fourier transform method will help us compute the desire signal even faster. If we were just using discrete Fourier Transform then the running time to computer the arithmetic operation would be $O(N^2)$ (Big O of N square time), compare to Fast Fourier transform by computing the same result with running time of O(NlogN) where N is number of points we are calculation. The speed difference between the Discrete Fourier transforms and Fast Fourier transform could be really important when it come to large package of data. We chose Fast Fourier transform method to do our Digital Signal Processing because it is fast in real time.

Real time control

One of the best features about Blackfin processor is that we can use it as wired system or wireless system for our control requirements. The software this processors come with can help us achieve the real time control and can be programmable. We can have different operating system installed with the processor according to our needs. Some of the examples are Fusion RTOs that is best for port and optimize networking. RTXC Quadros operating system that has Capability of convergent processing, it combines with traditional architecture of real time control processing, alone with executive for Digital Signal Processing that deal with data flow operation. We could use VisualDSP++ Development Software, which is a really easy to use software that offers all the following features:

- Fully integrated user interface including project management, debugging, profiling, plotting
- Support a variety of debug targets (emulation, simulation, compiled simulation, and 3rd party offerings)
- C/C++ compiler, assembler (with C data type support), expert linker, loader
- VisualDSP++ Kernel (VDK) with multiprocessor messaging capability
- Automation API and Automation Aware Scripting Engine
- Background Telemetry Channel (BTC) support with data streaming capability
- Profile-Guided Optimization(PGO)
- TCP/IP & USB Support and Processor configuration/Start-up code wizard (Blackfin processors)
- Multiple project management (SHARC and TigerSHARC processors)

Evaluation Kit

In order for the processor to work, we need to purchase the Evaluation kit for the BlackFin ADSP-BF 527 processor. This will allow us to connect the BlackFin processor to the evaluation board; the board can then communicated with the PC through serial interface USB cable. This will allow us to configure both hardware and software for our project. The only down side is the board cost about $1000 each.We have included a list of the features this evaluation board had come with:

- ADSP-BF527 Processor: 289 Ball mBGA, 600MHz; LFBGA-SS2, 12mmX12mm/0.5 pitch, 4L, (A02)

- SDRAM: 64 MB, 32M x 16; Micron MT48LC32M16A2 3.3V
- FLASH: 4 MB (2M x 16); Numonyx M29W320EB70ZE6E 3.3V
- NAND FLASH: 4 Gb Numonyx NAND04; NAND04GW3B2B 3.3V
- SPI FLASH: 16 Mb Numonyx M25P16-VMW6G 3.3V
- Audio CODEC: Audio CODEC-Low power, portable applications; External Codec for testing purposes only on Rev 1.0
- Power Analysis Interface: Sense resistors
- USB Debug Agent: ADSP-BF535; XC3S250E Spartan IIIe FPGA; USB port connector
- LCD Display: Varitronix - Landscape QVGA, 8 bit serial 320x240, Touch Screen
- Ethernet PHY: SMSC (RMII); LAN8700
- JTAG ICE 14-pin header
- USB cable
- CE compliant external power supply (US or European)
- Touch Screen Controller: Maxim Semiconductor, MAX1233
- Thumbwheel: CTS Corp, Rotary Encoder, 2 Bit Binary, CT2999-ND
- Keypad: ACT-07-30008-000-R
- Keypad Controller: Maxim Semiconductor, MAX1233
- UART: ADM3202 RS-232 line driver/receiver; DB9 female connector
- RTC Battery: 3.0 Volt Li-ION
- LEDs: 8 LEDs: 1 power (green), 1 board reset (red), 3 general purpose (amber), 1 USB monitor (amber), 2 Ethernet (amber)
- Push Buttons: 3 push buttons w/ debounce logic: 1 reset, 2 programmable flag
- Connectors: Keypad, USB OTG, Ethernet, HOST, SPORT0 (STAMP), SPORT 1 (STAMP), SPI (STAMP), TWI0 (STAMP), Timers (STAMP), PPI0 (STAMP), UART0, UART1 (STAMP), Expansion Interface (3)

Touch Screen LCD Panel

This board came with a low power consumption LCD touch screen panel which allows us to be able to configure the board without using the PC. This also comes with a set of stereo headphone and all the necessary cables.

5.3.8 Software Options

VisualDSP++

The Evaluation kit comes with evaluation board and an evaluation suite of the VisualDSP++ development and debugging environment. The software can compile the C++/C, it has assembler and linker in the tool set.

uClinux

Although Visual DSP++ offers all these great features, however it cost money. We are probably going to choose the open source Operating system that is also supported

by Linux, it is called uClinux. It stands for Microcontroller Linux. It was a fork of Linux kernel for microcontrollers, but with no memory managements. The reason why we chose this operating system is because the operating system itself is open source and it is available to use with Blackfin processor. They have a website/forum where we can get the free software and hardware projects that aims for the Blackfin processors. It has all the open source software, offers easy access to the best in source code, mailing lists, bug tracking, and message boards. All we have to do is register as a new user in order to use the site.

This uClinux project can also generate C standard library called uClibc, within the uClibc it contains libraries, all the useful tools and applications. We can configured and build into a kernel with root file system.

5.3.9 Coridium ARMmite

We have the option of using the Coridium ARMmite microcontroller board(figure 5.23); this board has the CPU frequency of 60 MHz. This board is really small; it is the size a credit card with prototype area measuring around 2 x 0.75 inches. We can use a host computer to connect to the Coridium ARMmite with a USB port interface to configure to emulate the venerable asynchronous serial port. Some of the great features of this board are following:

- ARM7 CPU running at 60 MHz
- BASIC compiler runs >10 million lines of codes/sec
- Pre-configured C compiler
- Programmed via USB interface
- Stand-alone operation
- 32K Flash memory and 8K SRAM memory
- 24 TTL compatible digital I/O
- 8 10-bit A/D converter channels, 100 kHz rate
- 8 Hardware PWM channels
- Powered from USB or 7-12V DC input
- Optional rechargeable battery backed real time clock
- internal supplies of 3.3V and 1.8V

Figure 5.23: Figure shows diagram of the ARMmite microcontroller. Reprinted with permission from Coridium Inc., see the Appendix Section A.1.

Process Power

The CPU speed is decent for most small applications, with 60 MHz of speed on microcontroller, we can finish the design but it might be a little bit slow to compute our signals, just the matter of time it takes to finish processing the capture signals. This microcontroller's main features are simplicity, which many control applications can be accomplished in a very small program, the program is easy to install and can start to program really quick.

Software

This microcontroller board also comes with a very useful suit of window based software development tools with a lot of useful documentation and all the helpful schematics. They also provided a useful forum at Yahoo Groups called ARMexpress.

If someone doesn't really know a lot of complex programming languages then this board is way to go, because the board has build-in hardware functions and the speed of compiled code that can outperform other complex languages, it is also easy to debug its program and ARMbasic program that came with the board can be enter directly from any console or can use edit using just basic notepad.

Alternative Software

However, we would prefer to interface this microcontroller board with Ubuntu Linux based operating system simply because all of our environments are store in our Unix based system. We found that this is possible to do, but it is not directly supported it Coridium product company. Since this board is interfaces with USB-Serial port and

Feisty release disable the USB-Serial port which normally show up as devices under the devices directory. Base on our research, we found that we can just uninstall the braille device drives by using the following command: sudo apt-get remove brltty brltty-xll to disable the drive. We can test the USB-Serial port using Windows machine, we can send some data to USB-Serial port while create a small program to activated the LED.

For example:

```
1 DIR(15) =1' //Use pin 15 as the output and set it =1.
2 WHILE (i<30)
3     OUT(15) = i AND 1  // set the pin 15 high =1 when i is odd, low
          when i is even.
4     i = i+1
5     WAIT(600) //wait 600us
6     PRINT "I'm here", i
7 LOOP
```

We can use this sample code to verify the data in the Windows platform first before move into the Linux environment. After get the code to work, we can unplug the USB port and connect it in Linux environment. If the USB port works, then you can be able to verify that the new port at /dev/ttyUSB0 or something close to that. After that we can verify with terminal emulator with TCL or Tool Command Language, we use it rapid prototyping, scripted applications and testing. The first line has to point to the correct USB-Serial port, you can name it poll.tcl.

Following is a simple code that demonstrate this:

Example: Poll the comport periodically

```
set serial [open /dev/ttyUSB0 r+]
fconfigure $serial -mode "19200,n,8,1" -handshake xonxoff -blocking 0
-buffering none -ttycontrol {RTS 0 DTR 0}
while {1} {
set data [read $serial] ;# read ALL incoming bytes
set size [string length $data] ; # number of received byte, may be 0
if { $size } {
puts "received $size bytes: $data"
} update }
```

We start the terminal emulator program the shell and making sure that we are in the same director we use previous which is poll.tcl. $ tclsh poll.tcl After executing the code and reset the ARMmite microcontroller, the print statement should pop up every time when LED blinks.

I'm here 1
I'm here 2
I'm here 3
....

5.3.10 RCM3365 RabbitCore

The RCM3365 RabbitCore(figure 5.24) is a core module that is design for the embedded systems. This module comes with development tool that supports all the software compiling, linking, debugging, and editing abilities all in one. This board supports serial port connection or USB connection from Personal computer to the core module. It is design to easy use of software and hardware, the Dynamic C capability helps to speed up the design process without any hassles; users can debug right on hardware itself so that there is no in-circuit emulation required. The company also provided the library for all the codes for drivers and sample programs is also available through the website.

Figure 5.24: Figure shows the diagram of RabbitCore with removable memory card and Ethernet connection and on-board microprocessors. Reprinted with permission from Z-World Inc., see the Appendix Section A.3.

Embedded Microprocessor

This board unit has a microprocessor embedded in it(Figure 5.25); it has low Electromagnetic Interference (EMI) design for the communications and embedded control. This processor is 8 bit architecture, it is driven by C instruction. With speed of 55MHz and software support up to 1 MB of code/data space, this speed is good for most of applications. The voltage is running at 3.3 v with 5V tolerant I/O.

Figure 5.25: Figure shows the on-board microprocessor embedded on the RabbitCore. Reprinted with permission from Z-World Inc., see the Appendix Section A.3.

Removable Memory Card

RCM3365 came with a removable memory card(figure 5.26) unit that is hot swappable. The module itself has a on board memory of 16 MByte of Nand flash, where Nand flash uses floating gate transistors and connected in the way where Nand gate looks like. The transistors are connected in series and if all the word lines are high then its bit line will pull low; in order to use write and erase function, the nand flash has to use tunnel injection and tunnel release respectively. This essentially works like a USB flash drive and most memory cards out there. Users can also expand the memory size with external memory support; the limit to this external card is up to 128 Mb. This memory cards are formatted into Dynamic C's and it is stored in the FAT or File Allocation Table, so users can not use the memory card and read it in xD picture card readers. However, this device does come with the accessory of USB card reader that is use to serve the purpose of reading the memory card with that USB port.

Figure 5.26: Figure shows external removable USB reader for the Memory. Reprinted with permission from Z-World Inc., see the Appendix Section A.3.

Design Advantages

This board is fully equipped with Microprocessor speed of 44 MHz clock that can process any type of signal with a very fast speed. The 10/100Base-T Ethernet allows connecting the board with high speed download and transferring data. This makes it ideal for any network enabling security and access system, and Heating, Ventilating, and Air Conditioning technology of indoor environmental comfort. It came with 512 KByte of Flash and 512 KByte of program execution Static random access memory. This device come with about 50 digital Input to Output pins that can share with about 6 serial ports, and all these are operated at 3.3V for low power consumption. The board can be use as a controlling microprocessor that mounts on the motherboard directly to save spaces and re-usability. We can interface the module with any of the CMOS compatible digital devices that are controlled by the Motherboard.

RabbitCore RCM 3365 Specifications:

Features RCM3365 RCM3375

- Microprocessor Rabbit 3000 @ 44.2 MHz
- Ethernet Port 10/100Base-T, RJ-45, 3 LEDs
- Flash 512K
- SRAM 512K program + 512K data
- Extended Memory 32 MB NAND Flash (fixed); xD-picture card socket supporting up to 128 MB xD-picture card socket supporting up to 128 MB
- Backup Battery Connection for user-supplied battery (to support RTC and SRAM)
- LED Indicators 5: ACT (activity), LINK (link), SPEED (10/100 Base-T), FM (flash memory), USR (user-programmable)

- General-Purpose I/O 52 parallel digital I/O: 44 configurable / 4 fixed inputs / 4 fixed outputs
- Additional Inputs 2 Startup Mode, Reset In
- Additional Outputs Status, Reset Out
- Auxiliary I/O Bus 8 data and 5 address (shared with I/O), plus I/O read-write
- Serial Ports Six 3.3 V CMOS-compatible: 6 configurable as asynchronous (with IrDA), 4 configurable as clocked serial (SPI), 2 configurable as SDLC/HDLC, 1 asynchronous serial port dedicated for programming
- Serial Rate Max. asynchronous baud rate = CLK/8
- Slave Interface Slave port permits use as master or intelligent peripheral with Rabbit-based or other master controller
- Real-Time Clock Yes
- Timers Ten 8-bit timers (6 cascadable from the first) and one 10-bit timer with 2 match registers
- Watchdog/Supervisor Yes
- Pulse-Width Modulators 4 PWM based on a 10-bit free-running counter and priority interrupts
- Input Capture 2-channel input capture can be used to time input signals from various port pins.
- Quadrature Decoder 2-channel quadrature decoder accepts inputs from external incremental encoder modules.
- Power 3.153.45 V DC, 250 mA @ 44.2 MHz 3.3 V
- Operating Temp. 0C to +70C
- Humidity 5-95 percent, non-condensing
- Connectors - Headers Two 2 x 17 (2 mm pitch), One 2 x 5, 1.27 mm programming, xD-picture card socket
- Board Size 1.850” 2.725” 0.86” (47 mm 69 mm 22 mm)

Software

This Rabbit module is programmed using Dynamic C software. It is an integrated C compiler, editor, loader and debugger. The software allows users to write and assemble the codes without exiting the environments. Because the Dynamic C software is provided with the module, it will save us a lot of time to write the code to implement the embedded systems. Since the Dynamic C software is base off C, so most of the functions are the same as C, such as IF statements, For Loop, While Loop, etc. Besides the traditional C, Dynamic C also includes some of the extensions for special use that makes the design process easier. The figure 5.27 are some of extensions that provided by Dynamic C:

Costatements	Simplify the implementation of state machines
Function Chaining	Control flow mechanism that allows callable segments of code to be embedded within one or more functions
Cofunctions	Allow cooperative processes to be implemented in a single program
Slice Statements	Enable preemptive processes in a single program

Figure 5.27: Figure shows the 4 push buttons and 8 switch buttons' resistors are connect in series. Reprinted with permission from Z-World Inc., see the Appendix Section A.3.

Hardware Connection and Debug

The build in tool helps to speed up the debug process, we dont even need the in circuit emulator or analyzer. The design does not need the prototype, it can be execute and debug directly on the final product, hence shall save us tremendous amount of time. The USB serial connection cable will be use to connect the module and PC to transmit the data into the module board, and then we shall be able to debug the device on machine level code or source code.

We can use either USB serial connection or wireless Ethernet that is build in the module, and all the drivers will be provided for ease installation. The module provides over 400 sample programs that will help the group to understand how the program is written and executed. The group shall be able to control the signal collection rate, control the frequency rate and filter the noise out of the signal, expand the variable gains and etc. With the wireless Ethernet connection, this will help us to transfer data such as videos and audios to the base station.

5.3.11 Conclusion

The Analog to Digital converter from Digilent can sample up to a million samples per second at 1 MHz speed, this is more than enough sampling speed that we needed. It provided 12 bit resolution, so this is pretty high resolution for the actual signal. This is the option that we can use, but out of all the hardwares we have above, there are a lot more to choose from. For example, ARMmite microcontroller, it has on-board Analog to digital converters, but the CPU processor on board is too slow comparing to others. It only has 8 bit resolution that is a little bit smaller compare to the Digilent 12 bit Analog to Digital converter and which is slightly below our requirement. If we do choose this microcontroller, then we probably wont be able to get the sampling speed out of it. Especially if we want to use the Nyquist Sampling technique at the higher sampling rate, like at the 6 or 10 times of simultaneous sampling rate. Compare the ARMmite Analog to Digital converter, Digilent Analog to Digital converter actually has this library that is embedded in VHDL (Verilog Hardware Descriptive Language). We can actually connect and talk to the device to perform all sorts of communication as we need, this will make the software side a lot easier for us to do. Another top choice is once again the BlackFin Processor, it is the ideal device to use for our design

but the price is our biggest concern. The process speed is more than enough to get our math computation, signal processing and converting, noise cancelling and etc. However, because of the higher cost. Digilent Nexys2 FPGA board is one of our top device to use to hook up to the 2 Analog to Digital converter to achieve our goal.

Chapter 6

Software

6.1 Simulator

A software simulation was developed throughout the research and design portion of our Senior Design Team schedule to better understand the mathematical analyses and approaches to the acoustic localization problem. Several features were developed to this approach. The following section delves into the simulation design approaches, including the choice of language for programming, various libraries used, classes designed, code snippets, and results of the simulator. This simulator will be used extensively in the next part of the Senior Design schedule, throughout construction for the hydrophone locations, to testing of the accuracy and precision of the actual implementation of the APL Subsystem.

The simulator integrates together various simulated components of the APL Subsystem to be tested in a simulated environment with appropriate physics of propagating wave, as well as the ability to inject noise, and graphically represent the data. The approach of the simulator was object-oriented and abstract with the majority of the components implemented as a library independent of the actual configuration and testing environment. This allowed for multiple testing configurations and seamless integration, for example if a bug was found in the library or an extensive modification was necessary the multiple applications would that utilize the library would be updated as well, because it is compiled from a single source.

6.1.1 Languages

Selection of a language is a very important aspect of when creating a simulator. There are many different languages to choose with different benefits and features. Most languages also have large library support that can provide great features to an application without having to invest time in reinventing a particular feature. Some languages can have simpler syntax but may lack in performance, however if ran a current computer, for this particular application, there will most likely be no issues concerning performance of the simulator. Sometimes for computationally intensive software packages, high performance computers are required to get the appropriate performance level required.

The following section will delve into the research and different aspects of the languages considered for developing the simulator. It will go into the pros and cons associated with the language and conclude with the language of choice.

MATLAB

MATLAB or in Linux a compatible language is GNU Octave are both a popular numerical computing environment. According to Wikipedia, "MATLAB allows matrix manipulation, plotting of functions and data, implementation of algorithms, creation of user interfaces, and interfacing with programs in other languages. Although it is numeric only, an optional toolbox uses the MuPAD symbolic engine, allowing access to computer algebra capabilities. An additional package, Simulink, adds graphical multidomain simulation and Model-Based Design for dynamic and embedded systems."

MATLAB is both popular in industry as well as academia, and is taught and used in several classes here at UCF. MATLAB has many features that makes it a great environment for quick calculation and analysis with many ways to illustrate the results. For our Senior Design Team we have some familiarity with MATLAB and have used it in several classes. Although, none of us have used it extensively to develop a full feature simulator in MATLAB.

Initial work was done it MATLAB for a rough draft and quick analysis there is a code segment of some of the work Figure 6.1.

```
format long g
%fid=fopen("phase.log","w");

%speed of sound underwater 1450 m/s
sos=1450;
%max distance between hydrophones 3 cm
d=3/100;
%frequency of pinger 30 khz
f=30000;
%wavelength
wlen=sos/f;

%sample size
s=512;
%capture speed 400 khz
c=400000;

%sample point times
t=[0:s-1]/c;

%depth of pinger, about 20 feet
pz=6;

%Loop for different position


```

```
%position of pinger
%(sub assumed at 0,0 facing north)
px=3;
py=5;
%for px=-10:.1:10
%for py=-10:.1:10

%distance from pinger to sub sensor center
dist=sqrt(px*px+py*py+pz*pz);
disp("Distance to Center");
disp(dist);

hangle=atan2(py,px);
disp("Angle to Pinger");
disp(180/pi*hangle);

%amplitude at source in Watts
watt=12;

%Approximate intensity at pinger
inten=watt/(4*pi*dist*dist);
%disp(inten);

%calculate position of hydrophones
%assuming mounted in circle front, left, etc
dist1=sqrt((px-0).^2+(py-(d/2)).^2+pz.^2);
dist2=sqrt((px-(d/2)).^2+(py-0).^2+pz.^2);
dist3=sqrt((px-0).^2+(py-(-d/2)).^2+pz.^2);
dist4=sqrt((px-(-d/2)).^2+(py-0).^2+pz.^2);
disp("Distance to Hydrophones");
disp([dist1,dist2,dist3,dist4]);

%make intensity for each
inten1=watt/(4*pi*dist1.^2);
inten2=watt/(4*pi*dist2.^2);
inten3=watt/(4*pi*dist3.^2);
inten4=watt/(4*pi*dist4.^2);
disp("Intensity at each Hydrophone");
disp([inten1,inten2,inten3,inten4]);

%Calculate number of wavelength to center
numwave=dist/wlen;

%Offset of wave is numwave of hydrophone
%versus the numwave to center
offset1=dist1/wlen-numwave;
offset2=dist2/wlen-numwave;
offset3=dist3/wlen-numwave;
offset4=dist4/wlen-numwave;
disp("Offset to each Hydropfronthone");
disp([offset1,offset2,offset3,offset4]);


function ra=dowrap(a)
```

```
81      ra=a;
82      if(a>.5)
83          ra=a-1;
84      end
85      if(a<-.5)
86          ra=a+1;
87      end
88  endfunction
89
90  %Relative offset based on
91  roffset1=dowrap(offset1-offset1);
92  roffset2=dowrap(offset2-offset1);
93  roffset3=dowrap(offset3-offset1);
94  roffset4=dowrap(offset4-offset1);
95
96  disp("Relative Offset to each Hydrophone");
97  disp([roffset1,roffset2,roffset3,roffset4]);
98
99
100 %Caclulate the wave form received for each hydrophone
101 hwave1=inten1*sin(2*pi*f*t+2*pi*offset1);
102 hwave2=inten2*sin(2*pi*f*t+2*pi*offset2);
103 hwave3=inten3*sin(2*pi*f*t+2*pi*offset3);
104 hwave4=inten4*sin(2*pi*f*t+2*pi*offset4);
105 %plot(t,hwave1,t,hwave2,t,hwave3,t,hwave4);
106
107 %Do FFT of signals
108 hfft1=fft(hwave1);
109 hfft2=fft(hwave2);
110 hfft3=fft(hwave3);
111 hfft4=fft(hwave4);
112
113 %make the frequency plot
114 freqx=[0:c/s:c/2-1];
115
116 %fancy way of finding the max index of signal
117 [a,index1]=max(abs(hfft1(1:s/2)));
118 [a,index2]=max(abs(hfft2(1:s/2)));
119 [a,index3]=max(abs(hfft3(1:s/2)));
120 [a,index4]=max(abs(hfft4(1:s/2)));
121
122 %calculate max frequency from index
123 hfreq1=freqx(index1);
124 hfreq2=freqx(index2);
125 hfreq3=freqx(index3);
126 hfreq4=freqx(index4);
127 disp("Max frequency at each Hydrophone");
128 disp([hfreq1,hfreq2,hfreq3,hfreq4]);
129
130 %find the offset of max freq
131 hcoffset1=angle(hfft1(index1))/(2*pi);
132 hcoffset2=angle(hfft2(index2))/(2*pi);
133 hcoffset3=angle(hfft3(index3))/(2*pi);
134 hcoffset4=angle(hfft4(index4))/(2*pi);
```

```
135 disp("Calculated offset at each Hydrophone");
136 disp([hcoffset1,hcoffset2,hcoffset3,hcoffset4]);
137
138 %find the relative offset of max freq
139 hcroffset1=dowrap(hcoffset1-hcoffset1);
140 hcroffset2=dowrap(hcoffset2-hcoffset1);
141 hcroffset3=dowrap(hcoffset3-hcoffset1);
142 hcroffset4=dowrap(hcoffset4-hcoffset1);
143 disp("Calculated relative Offset to each Hydrophone");
144 disp([hcroffset1,hcroffset2,hcroffset3,hcroffset4]);
```

Figure 6.1: MATLAB example code for preliminary hydrophone simulator.

Java

Java is a computer software platform developed from Sun Microsystems that provides a system for developing application software and deploys it across multiple platform environments, like windows, Linux, and mobile devices. The Java programming is very popular and has large user support and feature rich libraries. Java requires to be run in the Java Virtual Environment (JVM) and is not compile directly to be run on a processor. This can have significant drawbacks when trying to develop a high performance library. Java does offer simple development features when it comes to memory management, like the integrated automatic garbage collection mechanism to give the user freedom of having to handle pointer and memory allocation and deletion. Java is an object-oriented language used for creating abstract classes and inheritance of classes, providing a

Our Senior Design Team has had some familiarity with Java, but not extensive experience. The Robotics Club does not utilize Java for many of their applications therefore, Java was not chosen to be used for the simulator.

C++

C++ according to Wikipedia is "a statically typed, free-form, multi-paradigm, compiled, general-purpose programming language." C++ has both high level and low level language features providing the developer with strong constructs. C++ is a popular and widely used software in both industry and academia. It is the language of choice for the Robotics Club due to its high performance capabilities, user memory management, and many open-source supported libraries.

C++ was the language of choice for implementing the simulator due the our Senior Design Teams working knowledge and the support from the Robotics Club.

6.1.2 Libraries

Utilizing open-source libraries often gives great features when creating applications. There is no exception for the software simulator of the APL Subsystem. The following are some descriptions of the open source libraries used for the simulator.

CxUtils

CxUtils is a multi-platform C++ library containing many useful functions and classes for rapid development of applications released under the BSD License. It contains tools for threads, network communication, joysticks, RS-232/serial communication, shared/mapped memory interfaces (e.g. message box, message server/client), timers, keyboard and mouse capture/emulation, and basic math operations (matrices, quaternions, coordinate transformations). Using this library it should be a simple task to create a C++ application that can easily be ported between Windows, Linux, and other platforms. The following sections describe some of the available interfaces included with CxUtils. Example programs and additional documentation for developers is available with the library. [1]

FFTW

FFTW is a C subroutine library for computing the discrete Fourier transform (DFT) in one or more dimensions, of arbitrary input size, and of both real and complex data (as well as of even/odd data, i.e. the discrete cosine/sine transforms or DCT/DST). Benchmarks performed on on a variety of platforms, show that FFTW's performance is typically superior to that of other publicly available FFT software, and is even competitive with vendor-tuned codes. In contrast to vendor-tuned codes, however, FFTW's performance is portable: the same program will perform well on most architectures without modification. Hence the name, "FFTW," which stands for the somewhat whimsical title of "Fastest Fourier Transform in the West." The FFTW package was developed at MIT by Matteo Frigo and Steven G. Johnson. [2]

OpenCV

OpenCV (Open Source Computer Vision) is a library of programming functions for real time computer vision. OpenCV is released under a BSD license, it is free for both academic and commercial use. Example applications of the OpenCV library are Human-Computer Interaction (HCI); Object Identification, Segmentation and Recognition; Face Recognition; Gesture Recognition; Camera and Motion Tracking, Ego Motion, Motion Understanding; Structure From Motion (SFM); Stereo and Multi-Camera Calibration and Depth Computation; Mobile Robotics. [3]

The OpenCV library is used for visualizing the simulated data. For example, both the time domain representation of the signals captured by the hydrophones can be seen in Figure 6.2 and frequency domain in Figure 6.3.

[1] http://active-ist.sourceforge.net/cxutils.php?menu=cxutils

[2] http://www.fftw.org/

[3] http://opencv.willowgarage.com/wiki/

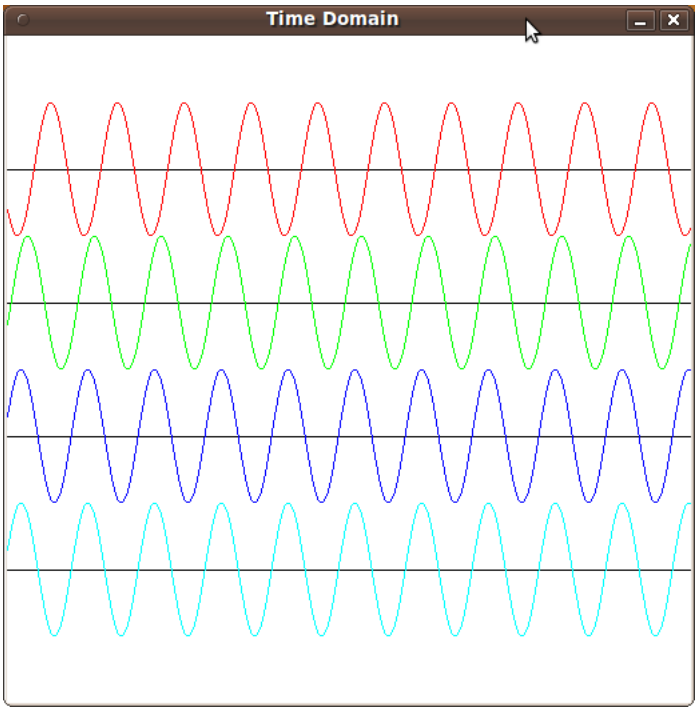


Figure 6.2: An example of the input signals of the simulated hydrophones digitally sampled.

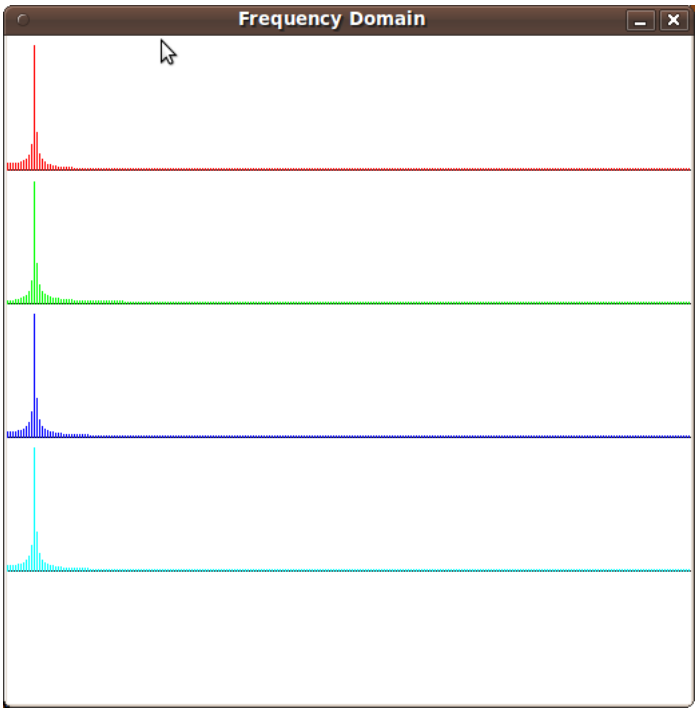


Figure 6.3: An example of the input signals of the simulated hydrophones digitally sampled then transforme into the frequency domain, displaying amplitude as a function of frequency.

6.1.3 Simulator Class

The following section goes into the code of the simulator and how it was developed. The Simulator Class encapsulates all the physics, environment parameters, sampling, calculations, and visualizations for the software simulation. Here is the code segment of the header file for the simulator class of all the contained data structures and organization of the code in Figure 6.4.

```
1  #ifndef __HYDROPHONE_SIMULATOR_H
2  #define __HYDROPHONE_SIMULATOR_H
3
4  #include "cxutils/math/cxmath.h"
5  #include <fftw3.h>
6  #include <vector>
7  #include "opencv/cv.h"
8  #include "opencv/highgui.h"
9  #include "opencv/cxcore.h"
10
11 namespace Zebulon
12 {
13     namespace Hydrophones
14     {
15         class Simulator
16         {
17             public:
18                 Simulator();
19                 ¬Simulator();
20
21                 static const double PingerFrequency;   //< Output
                       frequency of pinger measured in Hz.
22                 static const double WaveSpeed;          //< Wave
                       propagation speed measured in m/s, depends on
                       medium of travel, i.e. water.
23                 static const double Wavelength;
24                 static const double PingerPulsePeriod;     //< The
                       length of time until the pinger pings.
25                 static const double PingerPulseDuration;   //< The on
                        time of the pingers pulse.
26                 static const double PingerPowerOutput;     //< Rated
                       power output of acoustic energy at pinger source,
                       measured in watts? decibels?
27                 static const double AbsorptionCoefficient; //< Used
                       to calculate the power of signal at the receiver
                       due to loss of energy in meduim.
28
29                 static const int NumSamples;
30                 static const int NumHydrophones;
31                 static const double SampleFrequency;
32                 static const int mResolution;
33
34                 CxUtils::Point3D mPingerPosition;     //< Position of
                        pinger in meters.
```

```
35
36              double SignalAtReceiver(int receiverNumber, int
                    timeStep);
37              double PhaseAtReceiver(int receiverNumber, int time);
38
39              void DisplayTime();
40              void DisplayFreq();
41
42              void SetHydrophonePosition(int hydrophone, CxUtils::
                    Point3D positions);
43              double GetDistancePinger2Hydro(int hydrophone);
44              void Calc();
45              void PrintHydrophoneData();
46
47              double GetFreq(int hydrophone)
48              {
49                  return mHydrophones[hydrophone].mFrequency;
50              }
51
52              CxUtils::Point3D TimeDifferenceMulti(double ti,
                    double tj, double tk, double tl);
53              CxUtils::Point3D Multilateration(double rij, double
                    rik, double rkj, double rkl);
54              static double UnWrapPhase(double Ang1,double Ang2);
55
56              int CrossCorrelation(int hydrophoneA, int hydrophoneB
                    );
57              int CrossCorrelationFFT(int hydrophoneA, int
                    hydrophoneB);
58
59              void Test();
60
61              //Time Domain
62              fftw_complex *mH0Time;
63              fftw_complex *mH1Time;
64              fftw_complex *mH2Time;
65              fftw_complex *mH3Time;
66
67              //FFT Domain
68              fftw_complex *mH0Freq;
69              fftw_complex *mH1Freq;
70              fftw_complex *mH2Freq;
71              fftw_complex *mH3Freq;
72
73              fftw_plan mFFTPlan0;
74              fftw_plan mFFTPlan1;
75              fftw_plan mFFTPlan2;
76              fftw_plan mFFTPlan3;
77
78              fftw_plan mFFTPlan0b;
79              fftw_plan mFFTPlan1b;
80              fftw_plan mFFTPlan2b;
81              fftw_plan mFFTPlan3b;
82          private:
```

```
83                  IplImage *mTimeImage;
84                  IplImage *mFreqImage;
85
86                  class HydrophoneData
87                  {
88                      public:
89                          CxUtils::Point3D mPosition;
90                          int mFFTIndex;
91                          double mMaxMagnitude;
92                          double mFrequency;
93                          double mPhase;
94                  };
95
96                  HydrophoneData mHydrophones[4];
97                  void CalcValues(fftw_complex *mFreq, HydrophoneData &
                        Data);
98          };
99      }
100 }
101 #endif
102 /*  End of File */
```

Figure 6.4: C++ Header file displaying the member variables and organization of the code for the Simulator Class of the APL Subsystem software simulator.

6.1.4 Results

This section goes over some of the preliminary calculations resulting from using the developed simulator. Further simulations will be done in the second part of the Senior Design schedule to further test to implement the design. One example of the outputted data graphically displayed can be seen in Figure 6.5.

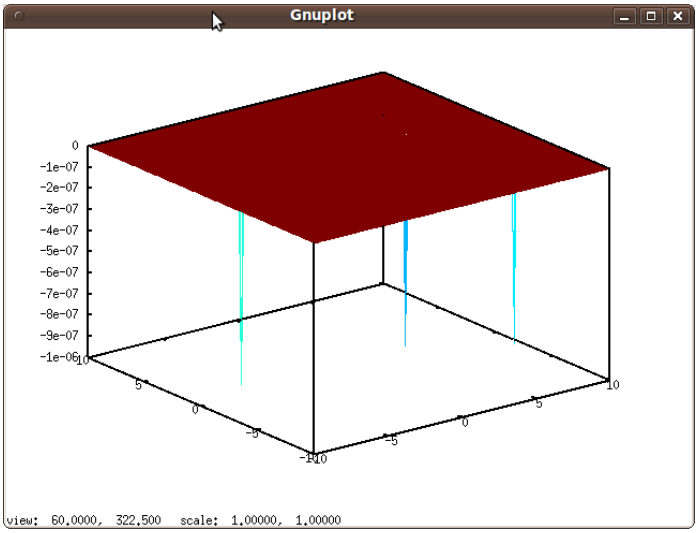


Figure 6.5: Data plotted using GNU Plot to display the differences in error of calculation utilizing multilateration with the pinger changing location in the XY plane.

Chapter 7

Explicit Design Summary

The preliminary design consists of an array of four or five hydrophones arranged in a particular way to allow for phase analysis on the signals to triangulate a heading, depth, and distance. Several different techniques were researched to achieve acoustic localization, a simulator was created to perform some preliminary analysis on these different methods to facilitate a final design. Another important aspect of this project is to receive the acoustic pinger signals from the underwater environment. The proposed method of achieving this goal is a passive hydrophone array mounted on the vehicle that converts acoustic energy into electrical energy. These attenuated signals are then conditioned by first pre-amplification with a variable gain to a utilizable signal. The signal is then filtered at the pinger's specified frequency using a third order Butterworth bandpass filter to remove unwanted noise. The final analog signal processing stage needs to adjust the signal for the appropriate range of the analog to digital converter. A more in-depth explanation including the specific components that were chosen and the final circuit schematics can be found in Chapter 4: Analog Hardware.

The analog signals are captured simultaneously from the analog to digital converter at a sampling rate that exceeds the Nyquist sampling theorem to recieve unaliased digital signal data. The data is then processed by a field programmable gate array (FPGA) which contains the digital signal processing to calculate the acoustic localization of the pinger. Information about the digital components investigated can be found in Chapter 5: Digital Hardware. The first step is to analyze each individual hydrophone signal by performing the Fast Fourier Transform (FFT) on the signals and determine relative phase differences between the hydrophone signals. The phase differential information is then used in the mathematical multilateration technique implemented on the FPGA to calculate the pingers location. The pingers heading, depth, and distance relative to the hydrophone array is then communicated over serial to the AUV's host computer which is then used to navigate to the pinger in the competition. For more information about the mathematical methods investigated see Chapter 3: Research and Investigation. See Figure 7.1 for a visualization of the process.

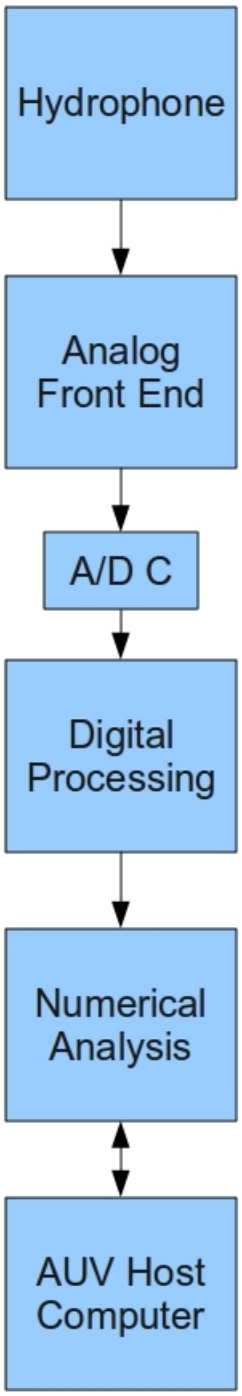


Figure 7.1: High level overview of the Acoustic Pinger Locator (APL) Subsystem integrated with the AUV Host Computer.

Appendix A

Copyright Permissions

A.1 Coridium Inc.

Jonathan Mohlenhoff <j.mohlenhoff@gmail.com>

Fwd: Information Request

zhen cai <zhencia@gmail.com> **Sun, Dec 13, 2009 at 4:32 PM**
To: Cassondra Puklavage <cassondrap@gmail.com>, Jonathan Mohlenhoff <j.mohlenhoff@gmail.com>

---------- Forwarded message ----------
From: **Bruce Eisenhard** <viskr@yahoo.com>
Date: Mon, Nov 23, 2009 at 10:09 AM
Subject: Re: Information Request
To: zhencia@gmail.com

Hi Zhen Cai

Please feel free to use any of our documents in you paper.

But we are not sponsoring any one at this time

Thank you for you interest

brucee

--- On Sun, 11/22/09, zhencia@gmail.com <zhencia@gmail.com> wrote:

> From: zhencia@gmail.com <zhencia@gmail.com>
> Subject: Information Request
> To: info@coridiumcorp.com
> Date: Sunday, November 22, 2009, 7:58 PM
> Hi, my name is Zhen Cai. I am a
> student at University of Central Florida and part of Robotic
> Club. I\'m currently working with my group on our senior
> design project called Autonomous Underwater Vehicle. It is
> also an IEEE competition project where all the universities
> in the whole nation are competing. We thought that Coridium
> ARMmite
> Single Board Programmable Controller from your
> company could help us accomplish this mission. So I would
> like to know if we can get the permissions for putting
> documentations/graphic/tables and any other contents from
> your website on our project paper. And also we would like to
> know if your company would like to be part of our
> sponsorship if it is possible, please let me know. If you
> need more information, the link to our website is
> www.auvsi.org/competitions/water.cfm and http://robotics.ucf.edu. Thank you for your time.
> sincerely, Zhen Cai University of Central Florida. Email: zhencia@gmail.com
> Phonte: 678-702-9995
>
>

A.2 Digilent Inc.

Jonathan Mohlenhoff <j.mohlenhoff@gmail.com>

Fwd: Nexys2 FPGA Board documentation permission

zhen cai <zhencia@gmail.com> **Sun, Dec 13, 2009 at 4:33 PM**
To: Cassondra Puklavage <cassondrap@gmail.com>, Jonathan Mohlenhoff <j.mohlenhoff@gmail.com>

---------- Forwarded message ----------
From: **Joe Harris** <joe@digilentinc.com>
Date: Mon, Nov 2, 2009 at 1:06 PM
Subject: RE: Nexys2 FPGA Board documentation permission
To: zhencia@gmail.com
Cc: sales@digilentinc.com

Hi Zhen,

You are welcome to use our documentation, graphics, photos, etc in your project.

What kind of sponsorship are you looking for?

Best Regards,

Joe Harris

www.digilentinc.com

509-334-6306

From: Fiona Cole [mailto:sales@digilentinc.com]
Sent: Monday, November 02, 2009 8:46 AM
To: 'Joe Harris'
Subject: FW: Nexys2 FPGA Board documentation permission

Fiona

Digilent Inc

A.3 Z-World

Jonathan Mohlenhoff <j.mohlenhoff@gmail.com>

Fwd: Documentation permission, price and sponsorships

zhen cai <zhencia@gmail.com> **Sun, Dec 13, 2009 at 4:32 PM**
To: Cassondra Puklavage <cassondrap@gmail.com>, Jonathan Mohlenhoff <j.mohlenhoff@gmail.com>

---------- Forwarded message ----------
From: <RabbitSales@rabbit.com>
Date: Mon, Nov 23, 2009 at 6:45 AM
Subject: Re:Documentation permission, price and sponsorships
To: zhencia@gmail.com

Good morning Zhen.

Regarding documentation for our product, if the information is available on line, you may certainly use it to support your design. We have development kits available which include the module, development board and software tools needed for development. Cost of the kit is $399.00. As a student, you are eligible for a 15% discount on the kit as well as modules. Below is a link which provides you with the descriptions and pricing.

http://www.rabbit.com/products/rcm3365/buyOnline.shtml

Though we don't offer sponsorships, we do wish you well in your design. If you have any questions, Cameron Haegle is the inside sales representative for the Southeast and is available at 952-912-3258 or Email: cameronh@digi.com

Good luck!

Gail Ovewrton-Welke
Digi Embedded Sales

From: zhencia@gmail.com
Sent: Sunday, November 22, 2009 10:00:54 PM
To: RabbitSales@Rabbit.com
CC:
Subject: Documentation permission, price and sponsorships

Hi,

my name is Zhen Cai. I am a student at University of Central Florida and part of Robotic Club. I'm currently working with my group on our senior design project called Autonomous Underwater Vehicle. It is also an IEEE competition project where all the universities in the whole nation are competing. We thought that RCM3365 **RabbitCore from your company could help us accomplish this mission. So I would like to know if we can get the permissions for putting documentations/graphic/tables and any other contents from your website on our project paper. And also we would like to know if your company would like to be part of our sponsorship if it is possible, please let me know. If you need more information, the link to our website is**

A.4 Association for Unmanned Vehicle Systems International (AUVSI)

Gmail by Google

Jonathan Mohlenhoff <j.mohlenhoff@gmail.com>

Fwd: Use of AUVSI pictures and information

Cassondra Puklavage <cassondrap@gmail.com> **Sun, Dec 13, 2009 at 9:07 PM**
To: Jonathan Mohlenhoff <j.mohlenhoff@gmail.com>

---------- Forwarded message ----------
From: **Leslie Hinton** <hinton@auvsi.org>
Date: Thu, Nov 19, 2009 at 4:02 PM
Subject: RE: Use of AUVSI pictures and information
To: Cassondra Puklavage <cassondrap@gmail.com>

Cassie,

Yes, you can go ahead and use the pictures. Do you have everything that you need?

Cheers,

Leslie

From: Cassondra Puklavage [mailto:cassondrap@gmail.com]
Sent: Thursday, November 19, 2009 3:59 PM
To: Leslie Hinton
Subject: Use of AUVSI pictures and information

Hello,

This is Cassondra Puklavage from the UCF team. I'm working on my Senior Design project this year and wanted know if I can use pictures and information from the competition rules document?

Thank you,

Cassie

A.5 Xilinx Inc.

Pending permission response.

A.6 Analog Devices Inc.

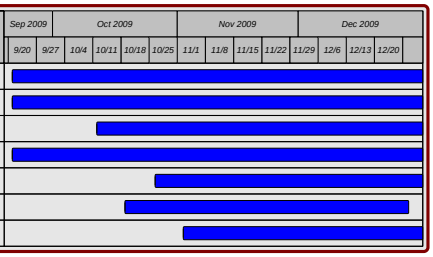
Pending permission response.

Appendix B

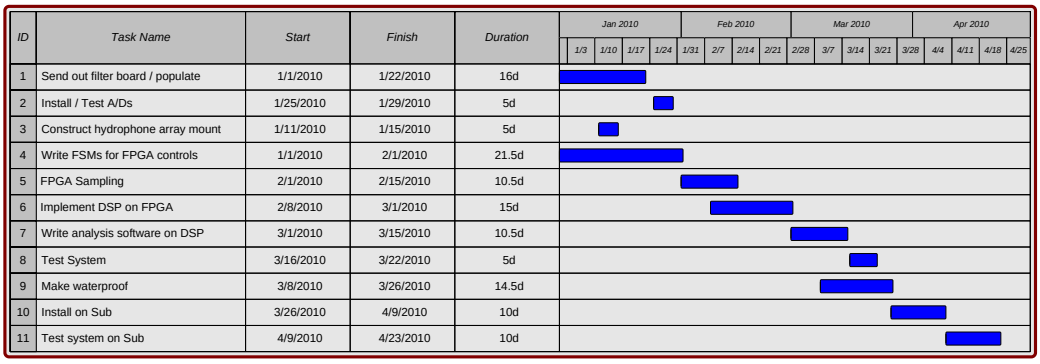
Milestone Charts

B.1 2009 Fall Semester

ID	Task Name	Start	Finish	Duration
1	Report	9/21/2009	2/21/2011	74.5d
2	Research Math	9/21/2009	4/12/2010	30d
3	Research FPGA/DSP	10/12/2009	5/31/2010	33.5d
4	Research Filter	9/21/2009	1/25/2010	18.5d
5	Simulation	10/26/2009	6/28/2010	35.5d
6	Prototype Filter Board	10/19/2009	12/28/2009	10.5d
7	Test/Adjust Prototype Filter Board	11/2/2009	3/22/2010	20.5d



B.2 2010 Spring Semester



ID	Task Name	Start	Finish	Duration
1	Send out filter board / populate	1/1/2010	1/22/2010	16d
2	Install / Test A/Ds	1/25/2010	1/29/2010	5d
3	Construct hydrophone array mount	1/11/2010	1/15/2010	5d
4	Write FSMs for FPGA controls	1/1/2010	2/1/2010	21.5d
5	FPGA Sampling	2/1/2010	2/15/2010	10.5d
6	Implement DSP on FPGA	2/8/2010	3/1/2010	15d
7	Write analysis software on DSP	3/1/2010	3/15/2010	10.5d
8	Test System	3/16/2010	3/22/2010	5d
9	Make waterproof	3/8/2010	3/26/2010	14.5d
10	Install on Sub	3/26/2010	4/9/2010	10d
11	Test system on Sub	4/9/2010	4/23/2010	10d

Appendix C

Software

C.1 /include/hydrophone_simulator.h

```
1  #ifndef __HYDROPHONE_SIMULATOR_H
2  #define __HYDROPHONE_SIMULATOR_H
3
4  #include "cxutils/math/cxmath.h"
5  #include <fftw3.h>
6  #include <vector>
7  #include "opencv/cv.h"
8  #include "opencv/highgui.h"
9  #include "opencv/cxcore.h"
10
11 namespace Zebulon
12 {
13     namespace Hydrophones
14     {
15         class Simulator
16         {
17             public:
18                 Simulator();
19                 ¬Simulator();
20
21                 static const double PingerFrequency;   //< Output
                       frequency of pinger measured in Hz.
22                 static const double WaveSpeed;          //< Wave
                       propagation speed measured in m/s, depends on
                       medium of travel, i.e. water.
23                 static const double Wavelength;
24                 static const double PingerPulsePeriod;     //< The
                       length of time until the pinger pings.
25                 static const double PingerPulseDuration;   //< The on
                        time of the pingers pulse.
26                 static const double PingerPowerOutput;     //< Rated
                       power output of acoustic energy at pinger source,
                       measured in watts? decibels?
```

```
27                  static const double AbsorptionCoefficient; //< Used
                        to calculate the power of signal at the receiver
                        due to loss of energy in meduim.
28
29                  static const int NumSamples;
30                  static const int NumHydrophones;
31                  static const double SampleFrequency;
32                  static const int mResolution;
33
34                  CxUtils::Point3D mPingerPosition;     //< Position of
                        pinger in meters.
35
36                  double SignalAtReceiver(int receiverNumber, int
                        timeStep);
37                  double PhaseAtReceiver(int receiverNumber, int time);
38
39                  void DisplayTime();
40                  void DisplayFreq();
41
42                  void SetHydrophonePosition(int hydrophone, CxUtils::
                        Point3D positions);
43                  double GetDistancePinger2Hydro(int hydrophone);
44                  void Calc();
45                  void PrintHydrophoneData();
46
47                  double GetFreq(int hydrophone)
48                  {
49                      return mHydrophones[hydrophone].mFrequency;
50                  }
51
52                  CxUtils::Point3D TimeDifferenceMulti(double ti,
                        double tj, double tk, double tl);
53                  CxUtils::Point3D Multilateration(double rij, double
                        rik, double rkj, double rkl);
54                  static double UnWrapPhase(double Ang1,double Ang2);
55
56                  int CrossCorrelation(int hydrophoneA, int hydrophoneB
                        );
57                  int CrossCorrelationFFT(int hydrophoneA, int
                        hydrophoneB);
58
59                  void Test();
60
61                  //Time Domain
62                  fftw_complex *mH0Time;
63                  fftw_complex *mH1Time;
64                  fftw_complex *mH2Time;
65                  fftw_complex *mH3Time;
66
67                  //FFT Domain
68                  fftw_complex *mH0Freq;
69                  fftw_complex *mH1Freq;
70                  fftw_complex *mH2Freq;
71                  fftw_complex *mH3Freq;
```

```
72
73                  fftw_plan mFFTPlan0;
74                  fftw_plan mFFTPlan1;
75                  fftw_plan mFFTPlan2;
76                  fftw_plan mFFTPlan3;
77
78                  fftw_plan mFFTPlan0b;
79                  fftw_plan mFFTPlan1b;
80                  fftw_plan mFFTPlan2b;
81                  fftw_plan mFFTPlan3b;
82              private:
83                  IplImage *mTimeImage;
84                  IplImage *mFreqImage;
85
86                  class HydrophoneData
87                  {
88                      public:
89                          CxUtils::Point3D mPosition;
90                          int mFFTIndex;
91                          double mMaxMagnitude;
92                          double mFrequency;
93                          double mPhase;
94                  };
95
96                  HydrophoneData mHydrophones[4];
97                  void CalcValues(fftw_complex *mFreq, HydrophoneData &
                        Data);
98          };
99      }
100 }
101 #endif
102 /*  End of File */
```

C.2 /src/hydrophone_simulator.cpp

```
1  #include "hydrophone_simulator.h"
2  #include <iostream>
3
4  using namespace std;
5  using namespace Zebulon;
6  using namespace Hydrophones;
7
8  const double Simulator::PingerFrequency = 20000.0;
9  const double Simulator::WaveSpeed = 1500.0;
10 const double Simulator::Wavelength = Simulator::WaveSpeed/Simulator::
       PingerFrequency;
11 const double Simulator::PingerPulsePeriod = 0.0;
12 const double Simulator::PingerPulseDuration = 0.0;
13 const double Simulator::PingerPowerOutput = 0.0;
14 const double Simulator::AbsorptionCoefficient = 0.0;
15
16 const int Simulator::NumSamples = 512;
17 const int Simulator::NumHydrophones = 4;
18 const double Simulator::SampleFrequency = 1000000.0;
19 const int Simulator::mResolution = 0;
20
21 Simulator::Simulator()
22 {
23     mH0Time = (fftw_complex*) fftw_malloc(sizeof(fftw_complex) *
          NumSamples);
24     mH1Time = (fftw_complex*) fftw_malloc(sizeof(fftw_complex) *
          NumSamples);
25     mH2Time = (fftw_complex*) fftw_malloc(sizeof(fftw_complex) *
          NumSamples);
26     mH3Time = (fftw_complex*) fftw_malloc(sizeof(fftw_complex) *
          NumSamples);
27
28     mH0Freq = (fftw_complex*) fftw_malloc(sizeof(fftw_complex) *
          NumSamples);
29     mH1Freq = (fftw_complex*) fftw_malloc(sizeof(fftw_complex) *
          NumSamples);
30     mH2Freq = (fftw_complex*) fftw_malloc(sizeof(fftw_complex) *
          NumSamples);
31     mH3Freq = (fftw_complex*) fftw_malloc(sizeof(fftw_complex) *
          NumSamples);
32
33     mFFTPlan0 = fftw_plan_dft_1d(NumSamples, mH0Time, mH0Freq,
          FFTW_FORWARD, FFTW_ESTIMATE);
34     mFFTPlan1 = fftw_plan_dft_1d(NumSamples, mH1Time, mH1Freq,
          FFTW_FORWARD, FFTW_ESTIMATE);
35     mFFTPlan2 = fftw_plan_dft_1d(NumSamples, mH2Time, mH2Freq,
          FFTW_FORWARD, FFTW_ESTIMATE);
36     mFFTPlan3 = fftw_plan_dft_1d(NumSamples, mH3Time, mH3Freq,
          FFTW_FORWARD, FFTW_ESTIMATE);
37
```

```
38      mFFTPlan0b = fftw_plan_dft_1d(NumSamples, mH0Time, mH0Freq,
            FFTW_BACKWARD, FFTW_ESTIMATE);
39      mFFTPlan1b = fftw_plan_dft_1d(NumSamples, mH1Time, mH1Freq,
            FFTW_BACKWARD, FFTW_ESTIMATE);
40      mFFTPlan2b = fftw_plan_dft_1d(NumSamples, mH2Time, mH2Freq,
            FFTW_BACKWARD, FFTW_ESTIMATE);
41      mFFTPlan3b = fftw_plan_dft_1d(NumSamples, mH3Freq, mH3Time,
            FFTW_BACKWARD, FFTW_ESTIMATE);
42
43      mTimeImage = cvCreateImage(cvSize(NumSamples, 500), 8, 3);
44      mFreqImage = cvCreateImage(cvSize(NumSamples, 500), 8, 3);
45
46      cvNamedWindow("Time Domain", CV_WINDOW_AUTOSIZE);
47      cvNamedWindow("Frequency Domain", CV_WINDOW_AUTOSIZE);
48  }
49
50
51  Simulator::¬Simulator()
52  {
53      fftw_free(mH0Time);
54      fftw_free(mH1Time);
55      fftw_free(mH2Time);
56      fftw_free(mH3Time);
57
58      fftw_free(mH0Freq);
59      fftw_free(mH1Freq);
60      fftw_free(mH2Freq);
61      fftw_free(mH3Freq);
62
63      fftw_destroy_plan(mFFTPlan0);
64      fftw_destroy_plan(mFFTPlan1);
65      fftw_destroy_plan(mFFTPlan2);
66      fftw_destroy_plan(mFFTPlan3);
67
68      cvReleaseImage(&mTimeImage);
69      cvReleaseImage(&mFreqImage);
70  }
71
72
73  void Simulator::SetHydrophonePosition(int hydrophone, CxUtils::
        Point3D position)
74  {
75      mHydrophones[hydrophone].mPosition = position;
76  }
77
78
79  double Simulator::GetDistancePinger2Hydro(int hydrophone)
80  {
81      return mPingerPosition.Distance(mHydrophones[hydrophone].
            mPosition);
82  }
83
84
85  double Simulator::SignalAtReceiver(int receiverNumber, int time)
```

```
86 {
87
88     double w = CxUtils::CX_TWO_PI*PingerFrequency;
89     double phi = mPingerPosition.Distance(mHydrophones[receiverNumber
           ].mPosition)*w/WaveSpeed;
90     double t = time/SampleFrequency;
91     return sin(w*t + phi);
92 }
93
94
95 double Simulator::PhaseAtReceiver(int receiverNumber, int time)
96 {
97
98     double w = CxUtils::CX_TWO_PI*PingerFrequency;
99     double phi = mPingerPosition.Distance(mHydrophones[receiverNumber
           ].mPosition)*w/WaveSpeed;
100    double t = time/SampleFrequency;
101    return phi;
102 }
103
104
105 CxUtils::Point3D Simulator::TimeDifferenceMulti(double ti, double tj,
        double tk, double tl)
106 {
107    double rij = ((ti - tj)*WaveSpeed);
108    double rik = ((ti - tk)*WaveSpeed);
109    double rkj = ((tk - tj)*WaveSpeed);
110    double rkl = ((tk - tl)*WaveSpeed);
111
112    return Multilateration(rij, rik, rkj, rkl);
113 }
114
115
116 CxUtils::Point3D Simulator::Multilateration(double rij, double rik,
       double rkj, double rkl)
117 {
118    double xi=mHydrophones[0].mPosition.mX;    double xj=mHydrophones
           [1].mPosition.mX;    double xk=mHydrophones[2].mPosition.mX;
              double xl=mHydrophones[3].mPosition.mX;
119    double yi=mHydrophones[0].mPosition.mY;    double yj=mHydrophones
           [1].mPosition.mY;    double yk=mHydrophones[2].mPosition.mY;
              double yl=mHydrophones[3].mPosition.mY;
120    double zi=mHydrophones[0].mPosition.mZ;    double zj=mHydrophones
           [1].mPosition.mZ;    double zk=mHydrophones[2].mPosition.mZ;
              double zl=mHydrophones[3].mPosition.mZ;
121
122    double xji = xj - xi; double xki = xk - xi; double xjk = xj - xk;
           double xlk = xl - xk;
123    double xik = xi - xk; double yji = yj - yi; double yki = yk - yi;
           double yjk = yj - yk;
124    double ylk = yl - yk; double yik = yi - yk; double zji = zj - zi;
           double zki = zk - zi;
125    double zik = zi - zk; double zjk = zj - zk; double zlk = zl - zk;
126
```

```
127     double s9  = rik*xji - rij*xki;  double s10 = rij*yki - rik*yji;
            double s11 = rik*zji - rij*zki;
128     double s12 = (rik*(rij*rij + xi*xi - xj*xj + yi*yi - yj*yj + zi*
            zi - zj*zj)
129                 - rij*(rik*rik + xi*xi - xk*xk + yi*yi - yk*yk + zi*
                        zi - zk*zk))/2;
130
131     double s13 = rkl*xjk - rkj*xlk;  double s14 = rkj*ylk - rkl*yjk;
            double s15 = rkl*zjk - rkj*zlk;
132     double s16 = (rkl*(rkj*rkj + xk*xk - xj*xj + yk*yk - yj*yj + zk*
            zk - zj*zj)
133                 - rkj*(rkl*rkl + xk*xk - xl*xl + yk*yk - yl*yl + zk*
                        zk - zl*zl))/2;
134
135     double a =  s9/(s10+0.000000000000001);  double b = s11/(s10
            +0.000000000000001);  double c = s12/(s10+0.000000000000001);
            double d = s13/(s14+0.000000000000001);
136     double e = s15/(s14+0.000000000000001);  double f = s16/(s14
            +0.000000000000001);  double g = (e-b)/(a-d+0.000000000000001)
            ;  double h = (f-c)/(a-d+0.000000000000001);
137     //double e = s15/(s14);  double f = s16/(s14);  double g = (e-b)
            /(a-d);  double h = (f-c)/(a-d);
138     double i = (a*g) + b;  double j = (a*h) + c;
139     double k = rik*rik + xi*xi - xk*xk + yi*yi - yk*yk + zi*zi - zk*
            zk + 2*h*xki + 2*j*yki;
140     double l = 2*(g*xki + i*yki + zki);
141     double m = 4*rik*rik*(g*g + i*i + 1) - l*l;
142     double n = 8*rik*rik*(g*(xi - h) + i*(yi - j) + zi) + 2*l*k;
143     double o = 4*rik*rik*((xi - h)*(xi -h) + (yi - j)*(yi - j) + zi*
            zi) - k*k;
144     double s28 = n/(2*m+0.000000000000001);       double s29 = (o/(m
            +0.000000000000001));     double s30 = (s28*s28) - s29;
145     double root = sqrt(fabs(s30));
146     //double root = sqrt(s30);
147     double z1 = s28 + root;
148     double z2 = s28 - root;
149     double x1 = g*z1 + h;
150     double x2 = g*z2 + h;
151     double y1 = a*x1 + b*z1 + c;
152     double y2 = a*x2 + b*z2 + c;
153
154     CxUtils::Point3D(x1, y1, z1).Print();
155     CxUtils::Point3D(x2, y2, z2).Print();
156
157     //return CxUtils::Point3D(x1, y1, z1);
158     return CxUtils::Point3D(x2, y2, z2);
159 }
160
161
162 void Simulator::Calc()
163 {
164     //individual hydrophones, calc phase
165     CalcValues(mH0Freq, mHydrophones[0]);
166     CalcValues(mH1Freq, mHydrophones[1]);
```

```
167     CalcValues(mH2Freq, mHydrophones[2]);
168     CalcValues(mH3Freq, mHydrophones[3]);
169
170     double iPhase = UnWrapPhase(mHydrophones[0].mPhase, mHydrophones
            [0].mPhase)/(2.0*M_PI)*WaveSpeed/GetFreq(0)/WaveSpeed;
171     double jPhase = UnWrapPhase(mHydrophones[0].mPhase, mHydrophones
            [1].mPhase)/(2.0*M_PI)*WaveSpeed/GetFreq(1)/WaveSpeed;
172     double kPhase = UnWrapPhase(mHydrophones[0].mPhase, mHydrophones
            [2].mPhase)/(2.0*M_PI)*WaveSpeed/GetFreq(2)/WaveSpeed;
173     double lPhase = UnWrapPhase(mHydrophones[0].mPhase, mHydrophones
            [3].mPhase)/(2.0*M_PI)*WaveSpeed/GetFreq(3)/WaveSpeed;
174
175     cout << "iTimeDiff: " << iPhase << endl;
176     cout << "jTimeDiff: " << jPhase << endl;
177     cout << "kTimeDiff: " << kPhase << endl;
178     cout << "lTimeDiff: " << lPhase << endl;
179
180     double rij = ((iPhase - jPhase)*WaveSpeed);
181     double rik = ((iPhase - kPhase)*WaveSpeed);
182     double rkj = ((kPhase - jPhase)*WaveSpeed);
183     double rkl = ((kPhase - lPhase)*WaveSpeed);
184
185     Multilateration(rij, rik, rkj, rkl);
186     /*
187     double rijPhase = UnWrapPhase(iPhase, jPhase);
188     double rikPhase = UnWrapPhase(iPhase, kPhase);
189     double rkjPhase = UnWrapPhase(kPhase, jPhase);
190     double rklPhase = UnWrapPhase(kPhase, lPhase);
191
192     Multilateration(rijPhase, rikPhase, rkjPhase, rklPhase);
193      */
194 }
195
196
197 double Simulator::UnWrapPhase(double Ang1,double Ang2)
198 {
199     double RetAng = Ang1 - Ang2;
200     while(RetAng < -M_PI)
201     {
202         RetAng += 2*M_PI;
203     }
204     while(RetAng > M_PI)
205     {
206         RetAng -= 2*M_PI;
207     }
208     return RetAng;
209 }
210
211
212 void Simulator::CalcValues(fftw_complex* mFreq, HydrophoneData& Data)
213 {
214     int maxFreqIndex = 0;
215     double mag = 0;
216
```

```
217     for(int i=0; i<NumSamples/2; i++)
218     {
219         double tempMag;
220         tempMag = sqrt(mFreq[i][0]*mFreq[i][0] + mFreq[i][1]*mFreq[i
                ][1]);
221         if(tempMag > mag)
222         {
223             maxFreqIndex = i;
224             mag = tempMag;
225         }
226     }
227
228     if(mag == 0.0)
229     {
230         mag = 1.0;
231     }
232     Data.mFFTIndex = maxFreqIndex;
233
234     double numeratorFreq = 0, numeratorPhaseR = 0, numeratorPhaseI =
            0;
235     double denominator = 0;
236     for(int i=maxFreqIndex-1; i<=maxFreqIndex+1; i++)
237     {
238         double tempMag = sqrt(mFreq[i][0]*mFreq[i][0] + mFreq[i][1]*
                mFreq[i][1]);
239         numeratorFreq += tempMag*((double)i/NumSamples*
                SampleFrequency);
240         numeratorPhaseI += mFreq[i][1] * tempMag;
241         numeratorPhaseR += mFreq[i][0] * tempMag;
242         denominator += tempMag;
243     }
244     Data.mFrequency = numeratorFreq/denominator; //Check divide by
            zero
245     Data.mPhase = atan2(numeratorPhaseI/denominator, numeratorPhaseR/
            denominator);
246     //Data.mFrequency = (double)maxFreqIndex/NumSamples*
            SampleFrequency;
247     //Data.mPhase = atan2(mFreq[maxFreqIndex][1], mFreq[maxFreqIndex
            ][0]);
248     Data.mMaxMagnitude = mag;
249 }
250
251
252 void Simulator::PrintHydrophoneData()
253 {
254     cout << "Name:\t\tHydrophone0\tHydrophone1\tHydrophone2\
            tHydrophone3" << endl;
255     //cout << "Position:\t"  << mHydrophones[0].mPosition.ToString()
            ;// << "\t" << mHydrophones[1].mPosition.Print() << "\t" <<
            mHydrophones[2].mPosition.Print() << "\t" << mHydrophones[3].
            mPosition.Print() << endl;
256     cout << "FFTIndex:\t"    << mHydrophones[0].mFFTIndex         <<
            "\t\t" << mHydrophones[1].mFFTIndex         << "\t\t" <<
            mHydrophones[2].mFFTIndex         << "\t\t" << mHydrophones
```

```
            [3].mFFTIndex         << endl;
257     cout << "Magnitude:\t"   << mHydrophones[0].mMaxMagnitude     <<
            "\t\t" << mHydrophones[1].mMaxMagnitude     << "\t\t" <<
            mHydrophones[2].mMaxMagnitude     << "\t\t" << mHydrophones
            [3].mMaxMagnitude        << endl;
258     cout << "Frequency:\t"   << mHydrophones[0].mFrequency        <<
            "\t\t" << mHydrophones[1].mFrequency        << "\t\t" <<
            mHydrophones[2].mFrequency        << "\t\t" << mHydrophones
            [3].mFrequency        << endl;
259     cout << "Phase:\t\t"     << mHydrophones[0].mPhase*180/M_PI
                    << "\t\t" << mHydrophones[1].mPhase*180/M_PI
                    << "\t\t" << mHydrophones[2].mPhase*180/M_PI
                    << "\t\t" << mHydrophones[3].mPhase*180/M_PI
                    << endl;
260 }
261
262
263 void Simulator::DisplayTime()
264 {
265     cvRectangle(mTimeImage, cvPoint(0,0), cvPoint(mTimeImage->width,
            mTimeImage->height), CV_RGB(255,255,255), -1);
266     cvLine(mTimeImage,cvPoint(0,100),cvPoint(mTimeImage->width,100),
            CV_RGB(0,0,0),1);
267     cvLine(mTimeImage,cvPoint(0,200),cvPoint(mTimeImage->width,200),
            CV_RGB(0,0,0),1);
268     cvLine(mTimeImage,cvPoint(0,300),cvPoint(mTimeImage->width,300),
            CV_RGB(0,0,0),1);
269     cvLine(mTimeImage,cvPoint(0,400),cvPoint(mTimeImage->width,400),
            CV_RGB(0,0,0),1);
270
271     double NormalRange = 1.0;
272
273     for(int i=0; i<NumSamples-1; i++)
274     {
275         cvLine(mTimeImage, cvPoint(i, -mH0Time[i][0]/NormalRange
                *50+100), cvPoint(i+1, -mH0Time[i+1][0]/NormalRange
                *50+100), CV_RGB(255,0,0), 1);
276         cvLine(mTimeImage, cvPoint(i, -mH1Time[i][0]/NormalRange
                *50+200), cvPoint(i+1, -mH1Time[i+1][0]/NormalRange
                *50+200), CV_RGB(0,255,0), 1);
277         cvLine(mTimeImage, cvPoint(i, -mH2Time[i][0]/NormalRange
                *50+300), cvPoint(i+1, -mH2Time[i+1][0]/NormalRange
                *50+300), CV_RGB(0,0,255), 1);
278         cvLine(mTimeImage, cvPoint(i, -mH3Time[i][0]/NormalRange
                *50+400), cvPoint(i+1, -mH3Time[i+1][0]/NormalRange
                *50+400), CV_RGB(0,255,255), 1);
279     }
280
281     cvShowImage("Time Domain",mTimeImage);
282 }
283
284
285 void Simulator::DisplayFreq()
286 {
```

```
287     cvRectangle(mFreqImage, cvPoint(0,0), cvPoint(mFreqImage->width,
            mFreqImage->height), CV_RGB(255,255,255), -1);
288     cvLine(mFreqImage,cvPoint(0,100),cvPoint(mFreqImage->width,100),
            CV_RGB(0,0,0),1);
289     cvLine(mFreqImage,cvPoint(0,200),cvPoint(mFreqImage->width,200),
            CV_RGB(0,0,0),1);
290     cvLine(mFreqImage,cvPoint(0,300),cvPoint(mFreqImage->width,300),
            CV_RGB(0,0,0),1);
291     cvLine(mFreqImage,cvPoint(0,400),cvPoint(mFreqImage->width,400),
            CV_RGB(0,0,0),1);
292
293     for(int i=0; i<NumSamples/2; i++)
294     {
295         double tmpmag;
296
297         tmpmag = sqrt(mH0Freq[i][0]*mH0Freq[i][0] + mH0Freq[i][1]*
                mH0Freq[i][1])/255*100;
298         cvLine(mFreqImage, cvPoint(i*2, 100), cvPoint(i*2, -tmpmag
                +100), CV_RGB(255, 0, 0), 1);
299
300         tmpmag = sqrt(mH1Freq[i][0]*mH1Freq[i][0] + mH1Freq[i][1]*
                mH1Freq[i][1])/255*100;
301         cvLine(mFreqImage, cvPoint(i*2, 200), cvPoint(i*2, -tmpmag
                +200), CV_RGB(0, 255, 0), 1);
302
303         tmpmag = sqrt(mH2Freq[i][0]*mH2Freq[i][0] + mH2Freq[i][1]*
                mH2Freq[i][1])/255.0*100;
304         cvLine(mFreqImage, cvPoint(i*2, 300), cvPoint(i*2, -tmpmag
                +300), CV_RGB(0, 0, 255), 1);
305
306         tmpmag = sqrt(mH3Freq[i][0]*mH3Freq[i][0] + mH3Freq[i][1]*
                mH3Freq[i][1])/255.0*100;
307         cvLine(mFreqImage, cvPoint(i*2, 400), cvPoint(i*2, -tmpmag
                +400), CV_RGB(0, 255, 255), 1);
308     }
309     cvShowImage("Frequency Domain",mFreqImage);
310 }
311
312 int Simulator::CrossCorrelation(int hydrophoneA, int hydrophoneB)
313 {
314     fftw_complex* x;
315     fftw_complex* y;
316     int n = NumSamples;
317     int maxdelay = 32;
318
319     switch(hydrophoneA)
320     {
321         case 0:
322             x = mH0Time;
323             break;
324         case 1:
325             x = mH1Time;
326             break;
327         case 2:
```

```
328             x = mH2Time;
329             break;
330         case 3:
331             x = mH3Time;
332             break;
333     }
334
335     switch(hydrophoneB)
336     {
337         case 0:
338             y = mH0Time;
339             break;
340         case 1:
341             y = mH1Time;
342             break;
343         case 2:
344             y = mH2Time;
345             break;
346         case 3:
347             y = mH3Time;
348             break;
349     }
350
351     int i, j, delay;
352     double mx,my,sx,sy,sxy,denom,r;
353     double rMax = 0.0;
354     int delayMax = 0;
355     double weightedNumerator = 0;
356     double rTotal = 0;
357
358     /* Calculate the mean of the two series x[], y[] */
359     mx = 0;
360     my = 0;
361     for (i=0;i<n;i++)
362     {
363         mx += x[i][0];
364         my += y[i][0];
365     }
366     mx /= n;
367     my /= n;
368
369     /* Calculate the denominator */
370     sx = 0;
371     sy = 0;
372     for (i=0; i<n; i++)
373     {
374         sx += (x[i][0] - mx) * (x[i][0] - mx);
375         sy += (y[i][0] - my) * (y[i][0] - my);
376     }
377     denom = sqrt(sx*sy);
378
379     /* Calculate the correlation series */
380     for (delay = -maxdelay; delay < maxdelay; delay++)
381     {
```

```
382         sxy = 0;
383         for (i=0; i<n; i++) {
384             j = i + delay;
385             if (j < 0 || j ≥ n)
386                 continue;
387             else
388             sxy += (x[i][0] − mx) * (y[j][0] − my);
389             /* Or should it be (?)
390             if (j < 0 || j ≥ n)
391                 sxy += (x[i] − mx) * (−my);
392             else
393                 sxy += (x[i] − mx) * (y[j] − my);
394             */
395         }
396         r = sxy / denom;
397         if(r > rMax)
398         {
399             rMax = r;
400             delayMax = delay;
401         }
402         //cout << "delay: " << delay << "\tr: " << r << endl;
403         /* r is the correlation coefficient at "delay" */
404     }
405
406
407     for(int offset = 0; offset < 32; offset++)
408     {
409         for (delay = delayMax − offset; delay ≤ delayMax + offset;
                delay++)
410         {
411             sxy = 0;
412             for (i=0;i<n;i++) {
413                 j = i + delay;
414                 if (j < 0 || j ≥ n)
415                     continue;
416                 else
417                 sxy += (x[i][0] − mx) * (y[j][0] − my);
418             }
419             r = sxy / denom;
420             weightedNumerator += delay*r;
421             rTotal += fabs(r);
422         }
423         cout << "Offset: " << offset << "\trMax: " << rMax << "\
                tdelayMax: " << delayMax << "\tWeightedDelay: " <<
                weightedNumerator/rTotal << endl;
424     }
425     return 0;
426 }
427
428 int Simulator::CrossCorrelationFFT(int hydrophoneA, int hydrophoneB)
429 {
430     fftw_execute(mFFTPlan0);
431     fftw_execute(mFFTPlan1);
432
```

```
433     for(int i=0; i<NumSamples; i++)
434     {
435         mH0Freq[i][1] *= -1;
436     }
437
438     for(int i=0; i<NumSamples; i++)
439     {
440        mH3Freq[i][0] = mH0Freq[i][0]*mH1Freq[i][0] - mH0Freq[i][1]*
              mH1Freq[i][1];
441        mH3Freq[i][1] = mH0Freq[i][0]*mH1Freq[i][1] + mH0Freq[i][1]*
              mH1Freq[i][0];
442     }
443
444     fftw_execute(mFFTPlan3b);
445     for(int i=0; i<NumSamples; i++)
446     {
447        cout << "i: " << i << "\tValue: " << mH3Time[i][0] << endl;
448     }
449
450     return 0;
451 }
452
453 void Simulator::Test()
454 {
455     double phaseA = UnWrapPhase(mHydrophones[0].mPhase, mHydrophones
          [0].mPhase)*180.0/M_PI;
456     double phaseB = UnWrapPhase(mHydrophones[0].mPhase, mHydrophones
          [1].mPhase)*180.0/M_PI;
457     double phaseC = UnWrapPhase(mHydrophones[0].mPhase, mHydrophones
          [2].mPhase)*180.0/M_PI;
458     double phaseD = UnWrapPhase(mHydrophones[0].mPhase, mHydrophones
          [3].mPhase)*180.0/M_PI;
459
460     cout << "A: " << phaseA << "\tB: " << phaseB << "\tC: " << phaseC
          << "\tD: " << phaseD << endl;
461 }
462
463 /*  End of File */
```

C.3 /src/example_hydrophone_sim.cpp

```
1  #include <iostream>
2
3  #include "hydrophone_simulator.h"
4
5  #include "opencv/cv.h"
6
7  using namespace std;
8  using namespace Zebulon;
9
10 int main(int argc, char **argv)
11 {
12     Hydrophones::Simulator sim;
13
14     sim.mPingerPosition = CxUtils::Point3D(7.5, 4.6, -3.15);
15
16     sim.SetHydrophonePosition(0, CxUtils::Point3D(0.00,  0.00, 0.00))
           ;
17     sim.SetHydrophonePosition(1, CxUtils::Point3D(0.00, -15.00, 0.00)
           );
18     sim.SetHydrophonePosition(2, CxUtils::Point3D(15.00,  15.00,
           0.00));
19     sim.SetHydrophonePosition(3, CxUtils::Point3D(0.00,  15.00, 0.00)
           );
20
21     double ati = (sim.GetDistancePinger2Hydro(0)/sim.WaveSpeed)*.96;
22     double atj = (sim.GetDistancePinger2Hydro(1)/sim.WaveSpeed)*1.03;
23     double atk = (sim.GetDistancePinger2Hydro(2)/sim.WaveSpeed)*1.01;
24     double atl = (sim.GetDistancePinger2Hydro(3)/sim.WaveSpeed)*.99;
25
26     cout << "Wavelength: " << sim.Wavelength << endl;
27
28     double ti = ati - ati;
29     double tj = ati - atj;
30     double tk = ati - atk;
31     double tl = ati - atl;
32
33     cout << "ati = " << ati << endl;
34     cout << "atj = " << atj << endl;
35     cout << "atk = " << atk << endl;
36     cout << "atl = " << atl << endl;
37
38     cout << endl;
39
40     cout << "ti = " << ti << endl;
41     cout << "tj = " << tj << endl;
42     cout << "tk = " << tk << endl;
43     cout << "tl = " << tl << endl;
44
45     cout << endl;
46
```

```
47      CxUtils::Point3D pingerSolution = sim.TimeDifferenceMulti(ti, tj,
            tk, tl);
48
49      int temp;
50      cin >> temp;
51      //return 0;
52
53      for(int i=0; i<sim.NumSamples; i++)
54      {
55          sim.mH0Time[i][0] = (double)(sim.SignalAtReceiver(0, i));
56          sim.mH0Time[i][1] = 0.0;
57
58          sim.mH1Time[i][0] = (double)(sim.SignalAtReceiver(1, i));
59          sim.mH1Time[i][1] = 0.0;
60
61          sim.mH2Time[i][0] = (double)(sim.SignalAtReceiver(2, i));
62          sim.mH2Time[i][1] = 0.0;
63
64          sim.mH3Time[i][0] = (double)(sim.SignalAtReceiver(3, i));
65          sim.mH3Time[i][1] = 0.0;
66      }
67
68      sim.DisplayTime();
69
70      fftw_execute(sim.mFFTPlan0);
71      fftw_execute(sim.mFFTPlan1);
72      fftw_execute(sim.mFFTPlan2);
73      fftw_execute(sim.mFFTPlan3);
74
75      sim.DisplayFreq();
76
77      cout << endl;
78
79      sim.Calc();
80
81      cout << endl;
82
83      sim.PrintHydrophoneData();
84
85      sim.Test();
86
87      double iPhase = sim.PhaseAtReceiver(0, 1);
88      double jPhase = sim.PhaseAtReceiver(1, 1);
89      double kPhase = sim.PhaseAtReceiver(2, 1);
90      double lPhase = sim.PhaseAtReceiver(3, 1);
91
92      double iTimeDiff = Hydrophones::Simulator::UnWrapPhase(iPhase,
            iPhase)/(2.0*M_PI)*sim.Wavelength/sim.WaveSpeed*1.000000;
93      double jTimeDiff = Hydrophones::Simulator::UnWrapPhase(iPhase,
            jPhase)/(2.0*M_PI)*sim.Wavelength/sim.WaveSpeed*1.000000;
94      double kTimeDiff = Hydrophones::Simulator::UnWrapPhase(iPhase,
            kPhase)/(2.0*M_PI)*sim.Wavelength/sim.WaveSpeed*1.000000;
95      double lTimeDiff = Hydrophones::Simulator::UnWrapPhase(iPhase,
            lPhase)/(2.0*M_PI)*sim.Wavelength/sim.WaveSpeed*1.000000;
```

```
96
97      double rij = (iTimeDiff - jTimeDiff)*sim.WaveSpeed;
98      double rik = (iTimeDiff - kTimeDiff)*sim.WaveSpeed;
99      double rkj = (kTimeDiff - jTimeDiff)*sim.WaveSpeed;
100     double rkl = (kTimeDiff - lTimeDiff)*sim.WaveSpeed;
101
102     sim.Multilateration(rij, rik, rkj, rkl);
103
104     cout << endl;
105
106     cout << "iTimeDiff: " << iTimeDiff << endl;
107     cout << "jTimeDiff: " << jTimeDiff << endl;
108     cout << "kTimeDiff: " << kTimeDiff << endl;
109     cout << "lTimeDiff: " << lTimeDiff << endl;
110
111     cout << endl;
112     sim.CrossCorrelation(0, 0);
113     sim.CrossCorrelation(0, 1);
114     sim.CrossCorrelation(0, 2);
115     sim.CrossCorrelation(0, 3);
116
117     cin >> temp;
118     cvWaitKey(0);
119     cin >> temp;
120
121     sim.CrossCorrelationFFT(0, 0);
122     sim.DisplayFreq();
123     sim.DisplayTime();
124     cvWaitKey(0);
125
126     return 0;
127 }
```

C.4 /src/example_multilateration.cpp

```
1  #include <iostream>
2  #include <math.h>
3  #include <stdlib.h>
4  #include <stdio.h>
5
6  using namespace std;
7
8  #define SAFEDIV(a,b) ((b==0.0)?1e6:a/b)
9
10 int main(int argc, char **argv)
11 {
12     double emitterX = 0.0;
13     double emitterY = 0.0;
14     double emitterZ = 0.0;
15     double xi=0.0;            double xj=0.01;            double xk
           =0.01;          double xl=0.0;
16     double yi=0.0;            double yj=0.0;             double yk
           =0.01;          double yl=0.01;
17     double zi=0.0;            double zj=0.0;             double zk
           =0.0;           double zl=0.0;
18
19     //double xi=0.0;            double xj=0.01;            double xk
           =0.0;          double xl=-0.01;
20     //double yi=0.0;            double yj=-0.01;             double
           yk=-0.2;          double yl=-0.01;
21     //double zi=0.0;            double zj=0.0;             double zk
           =0.0;           double zl=0.0;
22
23     int count = 0;
24     int nancount=0;
25
26     double rmseX1=0.0;
27     double rmseX2=0.0;
28     double rmseY1=0.0;
29     double rmseY2=0.0;
30     double rmseZ1=0.0;
31     double rmseZ2=0.0;
32
33
34     FILE* fpX;
35     FILE* fpY;
36     FILE* fpZ;
37     char filename[1024];
38     //fp = fopen("data.csv", "w");
39
40     for(emitterZ = -10.0; emitterZ < 10.0; emitterZ+=0.01)
41     {
42         sprintf(filename,"Xdata_%05.1lf.log",emitterZ);
43         fpX=fopen(filename,"w");
44         sprintf(filename,"Ydata_%05.1lf.log",emitterZ);
```

```
45          fpY=fopen(filename,"w");
46          sprintf(filename,"Zdata_%05.1lf.log",emitterZ);
47          fpZ=fopen(filename,"w");
48          printf("%lf\n",emitterZ);
49
50          nancount=0;
51          rmseX1=0.0;
52          rmseX2=0.0;
53          rmseY1=0.0;
54          rmseY2=0.0;
55          rmseZ1=0.0;
56          rmseZ2=0.0;
57
58          for(emitterX = -10.0; emitterX < 10.0; emitterX+=0.01)
59          {
60              for(emitterY = -10.0; emitterY < 10.0; emitterY+=0.01)
61              {
62                  double di=sqrt((emitterX - xi)*(emitterX - xi) + (
                        emitterY - yi)*(emitterY - yi) + (emitterZ - zi)*(
                        emitterZ - zi));
63                  double dj=sqrt((emitterX - xj)*(emitterX - xj) + (
                        emitterY - yj)*(emitterY - yj) + (emitterZ - zj)*(
                        emitterZ - zj));
64                  double dk=sqrt((emitterX - xk)*(emitterX - xk) + (
                        emitterY - yk)*(emitterY - yk) + (emitterZ - zk)*(
                        emitterZ - zk));
65                  double dl=sqrt((emitterX - xl)*(emitterX - xl) + (
                        emitterY - yl)*(emitterY - yl) + (emitterZ - zl)*(
                        emitterZ - zl));
66
67                  double ti=di/1500.0;
68                  double tj=dj/1500.0;
69                  double tk=dk/1500.0;
70                  double tl=dl/1500.0;
71
72                  //cout << "ti = " << ti << endl;      cout << "tj = "
                        << tj << endl;      cout << "tk = " << tk << endl
                        ;
73                  //cout << "tl = " << tl << endl;      cout << "xi = "
                        << xi << endl;      cout << "xj = " << xj << endl
                        ;
74                  //cout << "xk = " << xk << endl;      cout << "xl = "
                        << xl << endl;      cout << "yi = " << yi << endl
                        ;
75                  //cout << "yj = " << yj << endl;      cout << "yk = "
                        << yk << endl;      cout << "yl = " << yl << endl
                        ;
76                  //cout << "zi = " << zi << endl;      cout << "zj = "
                        << zj << endl;      cout << "zk = " << zk << endl
                        ;
77                  //cout << "zl = " << zl << endl;
78
79                  double xji = xj - xi; double xki = xk - xi; double
                        xjk = xj - xk; double xlk = xl - xk;
```

```
80                double xik = xi - xk; double yji = yj - yi; double
                      yki = yk - yi; double yjk = yj - yk;
81                double ylk = yl - yk; double yik = yi - yk; double
                      zji = zj - zi; double zki = zk - zi;
82                double zik = zi - zk; double zjk = zj - zk; double
                      zlk = zl - zk;
83
84                double rij = ((ti - tj)*1500.0); double rik = ((ti -
                      tk)*1500.0);
85                double rkj = ((tk - tj)*1500.0); double rkl = ((tk -
                      tl)*1500.0);
86
87                double s9  = rik*xji - rij*xki;  double s10 = rij*yki
                       - rik*yji;  double s11 = rik*zji - rij*zki;
88                double s12 = (rik*(rij*rij + xi*xi - xj*xj + yi*yi -
                      yj*yj + zi*zi - zj*zj)
89                           - rij*(rik*rik + xi*xi - xk*xk + yi*yi -
                                yk*yk + zi*zi - zk*zk))/2;
90
91                double s13 = rkl*xjk - rkj*xlk;  double s14 = rkj*ylk
                       - rkl*yjk;  double s15 = rkl*zjk - rkj*zlk;
92                double s16 = (rkl*(rkj*rkj + xk*xk - xj*xj + yk*yk -
                      yj*yj + zk*zk - zj*zj)
93                           - rkj*(rkl*rkl + xk*xk - xl*xl + yk*yk -
                                yl*yl + zk*zk - zl*zl))/2;
94
95                double a =  s9/(s10+0.000000000000001);  double b =
                      s11/(s10+0.000000000000001);  double c = s12/(s10
                      +0.000000000000001);  double d = s13/(s14
                      +0.000000000000001);
96                double e = s15/(s14+0.000000000000001);  double f =
                      s16/(s14+0.000000000000001);  double g = (e-b)/(a-
                      d+0.000000000000001);  double h = (f-c)/(a-d
                      +0.000000000000001);
97                //double e = s15/(s14);  double f = s16/(s14);
                      double g = (e-b)/(a-d);  double h = (f-c)/(a-d);
98                double i = (a*g) + b;  double j = (a*h) + c;
99                double k = rik*rik + xi*xi - xk*xk + yi*yi - yk*yk +
                      zi*zi - zk*zk + 2*h*xki + 2*j*yki;
100               double l = 2*(g*xki + i*yki + zki);
101               double m = 4*rik*rik*(g*g + i*i + 1) - l*l;
102               double n = 8*rik*rik*(g*(xi - h) + i*(yi - j) + zi) +
                       2*l*k;
103               double o = 4*rik*rik*((xi - h)*(xi -h) + (yi - j)*(yi
                       - j) + zi*zi) - k*k;
104               double s28 = n/(2*m+0.000000000000001);       double
                      s29 = (o/(m+0.000000000000001));     double s30 =
                      (s28*s28) - s29;
105               double root = sqrt(fabs(s30));
106               //double root = sqrt(s30);
107               double z1 = s28 + root;
108               double z2 = s28 - root;
109               double x1 = g*z1 + h;
110               double x2 = g*z2 + h;
```

```
111                 double y1 = a*x1 + b*z1 + c;
112                 double y2 = a*x2 + b*z2 + c;
113
114                 double errorX1 = (emitterX - x1);
115                 double errorX2 = (emitterX - x2);
116                 double errorY1 = (emitterY - y1);
117                 double errorY2 = (emitterY - y2);
118                 double errorZ1 = (emitterZ - z1);
119                 double errorZ2 = (emitterZ - z2);
120                 /*nancount+=isnan(x1);
121                 nancount+=isnan(x2);
122                 nancount+=isnan(x1);
123                 nancount+=isnan(y2);
124                 nancount+=isnan(z1);
125                 nancount+=isnan(z2);*/
126
127                 if(isnan(x1) || isnan(x2) || isnan(x1) || isnan(y2)
                        || isnan(z1) || isnan(z2))
128                 {
129                     nancount++;
130                     printf("%lf %lf %lf\n",emitterX,emitterY,emitterZ
                            );
131                 }
132
133
134                 rmseX1+=errorX1*errorX1;
135                 rmseX2+=errorX2*errorX2;
136                 rmseY1+=errorY1*errorY1;
137                 rmseY2+=errorY2*errorY2;
138                 rmseZ1+=errorZ1*errorZ1;
139                 rmseZ2+=errorZ2*errorZ2;
140                 count++;
141
142                 //fprintf(fpX,"%lf ",errorX1);
143                 //fprintf(fpY,"%lf ",errorY1);
144                 //fprintf(fpZ,"%lf ",errorZ1);
145 #if 0
146                 //fprintf(fp, "%0.3lf, %0.3lf, %0.3lf, %0.3lf, %0.3lf
                        , %0.3lf, %0.3lf, %0.3lf, %0.3lf\n", emitterX, x1,
                         x2, emitterY, y1, y2, emitterZ, z1, z2);
147                 /*
148                 cout << "emitterX: " << emitterX << "\tx1 = " << x1
                        << "\tx2 = " << x2 << endl;
149                 cout << "emitterY: " << emitterY << "\ty1 = " << y1
                        << "\ty2 = " << y2 << endl;
150                 cout << "emitterZ: " << emitterZ << "\tz1 = " << z1
                        << "\tz2 = " << z2 << endl;
151                 cout << endl;
152                 */
153
154
155
156                 if((fabs(errorX1) < 0.1 && fabs(errorX2) > 0.1) ||
157                    (fabs(errorY1) > 0.1 && fabs(errorY2) > 0.1) ||
```

```
158                     (fabs(errorZ1) > 0.1 && fabs(errorZ2) > 0.1))
159                 {
160                     count++;
161                     /*
162                     cout << count << endl;
163                     cout << "emitterX: " << emitterX << "\tx1 = " <<
                            x1 << "\tx2 = " << x2 << endl;
164                     cout << "emitterY: " << emitterY << "\ty1 = " <<
                            y1 << "\ty2 = " << y2 << endl;
165                     cout << "emitterZ: " << emitterZ << "\tz1 = " <<
                            z1 << "\tz2 = " << z2 << endl;
166                     cout << endl;
167                     */
168                 }
169 #endif
170             }
171
172             //fprintf(fpX,"\n");
173             //fprintf(fpY,"\n");
174             //fprintf(fpZ,"\n");
175
176         }
177         rmseX1=sqrt(rmseX1/count);
178         rmseX2=sqrt(rmseX2/count);
179         rmseY1=sqrt(rmseY1/count);
180         rmseY2=sqrt(rmseY2/count);
181         rmseZ1=sqrt(rmseZ1/count);
182         rmseZ2=sqrt(rmseZ2/count);
183         printf("X %lf %lf Y %lf %lf Z %lf %lf : %d\n",rmseX1,rmseX2,
              rmseY1,rmseY2,rmseZ1,rmseZ2,nancount);
184         fclose(fpX);
185         fclose(fpY);
186         fclose(fpZ);
187     }
188     //cout << count << " DONE\n";
189     //cin >> count;
190 }
```