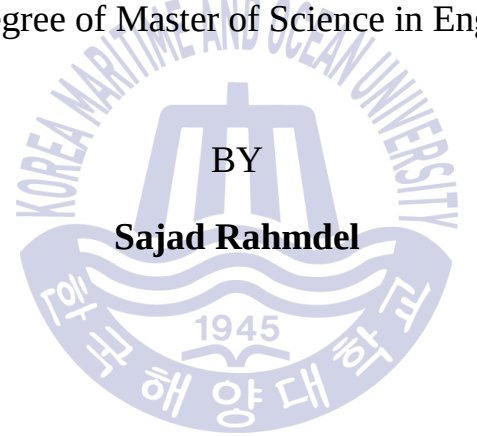
SONAR SYSTEM SOFTWARE AND HARDWARE DEVELOPMENT: FISH FINDER APPLICATION

A THESIS

Submitted to the

graduate school of
Korea Maritime and Ocean University, in partial fulfillment of the requirements
for the degree of Master of Science in Engineering



BY

Sajad Rahmdel

February 2018

Advisor:

Professor Gang-Gyoo Jin

SONAR SYSTEM SOFTWARE AND HARDWARE DEVELOPMENT: FISH FINDER APPLICATOIN

By

Sajad Rahmdel

Submitted in partial fulfillment of the requirements for the degree of Master of Science in Engineering

Approved by:

Professor Gi Ryong Jeong .....

Professor Serng Bae Moon .....

Professor Gang Gyoo Jin .....

December 2017

Department of Control and Instrumentation Engineering

Graduate School, Korea Maritime and Ocean University

ACKNOWLEDGEMENTS

This thesis owes its existence to the help, support and inspiration of several people. Firstly, I would like to express my sincere gratitude to Prof. Yu and my family for their supports, understanding, patience, enthusiasm and encouragement. I would also like to thank Dr. Jong-Hwa Kim and professor Gang-Gyoo Jin for their assistance and suggestions to write this thesis.

I am grateful to my teammates and friends here in Korea Maritime and Ocean University, especially to Dr. Soo-Jong Mo. I appreciate every single assistance from my best friends, Amin Beirami and Hamid Reza Islami who stood by my side and gave me support and encouragement. Last but not least, I thank Dr. Jong-Heung Kim and other members of FemTek company for their assistance in designing and debugging the sonar hardware.

Table of Contents

List of Figures

SONAR SYSTEM SOFTWARE AND HARDWARE DEVELOPMENT: FISH FINDER APPLICATOIN

Sajad Rahmdel

Department of Control and Instrumentation Engineering
Graduate School of Korea Maritime and Ocean University

ABSTRACT

A study of how to develop a sonar system is presented in this thesis. A typical sonar system has two sub circuits, namely, transmit path and receive path. On transmit path, HV (high voltage) pulses are generated and transmitted into the water using a transducer. On the receive path, the echo of the transmitted signal is received. This received signal is then amplified and converted into digital data using an ADC. The digital data is then sent to a PC for signal processing. After signal processing, the results are shown in a GUI.

Software technologies used in this study include Fish Finder GUI, Visual C++ GUI and back-end code, AFE5809 GUI, and High Speed Data Converter Pro (by Texas Instruments). Matlab was used to develop the Fish Finder GUI similar to commercial fish finders. The Fish Finder GUI also does noise filtering as well as calculating the depth and distance to the detected objects. Visual C++ was used to develop a software, called, Control GUI. This software, receives the data from the TSW1400, saves the data as a .CSV file. It also connects all used software programs together to work as a complete fish finder system.

The developed sonar system was tested in 1-meter water, as well as 10-meter shore side test. First, HV pulse was generated and transmitted into the water. After that, the echo of the pulse

was captured. Then, the echo was converted into digital data. Finally, the digital data was the digital data results were successfully shown in the GUIs.

Chapter 1. Introduction

In this chapter, an overview of sonar systems as well as background researches in this area is explained. Also, at the end of the chapter, the purpose of this thesis is described.

SONAR is the acronym for SOund Navigation And Ranging. By using a sonar system, we can detect and map objects underwater. Usually, sonar systems have a frequency between 20KHz up to 800KHz. Sonar has several applications including fish finders, ocean floor mapping, and vessel navigation.

Sonar systems fall into two categories, namely, passive sonars and active sonars. In passive sonars, no signal is transmitted and the system only listens and receives echoes. Passive sonars are typically used for military vessels, where the vessel would stay undetectable.

The second category is called active sonars. In active sonars, the system transmits a pulse or ping using a transmitter. Then the system waits and listens for the response or echo of the transmitted signal. The echo is captured by a receiver. However, some systems have a transceiver, which is the combination of both transmitter and receiver [1]. If the transmitted signal hits an object, such as a school of fish, an attenuated reflection will move backward towards the sonar system. This reflection is called echo. When the echo reaches the system, the receiver will capture this echo. After that the receiver will convert this echo into electrical signal. The electrical signal is very weak and requires amplification. In addition, this signal contains noise which needs to be removed before the signal can be processed. An example of a commercial fish finder is shown in Figure 1.

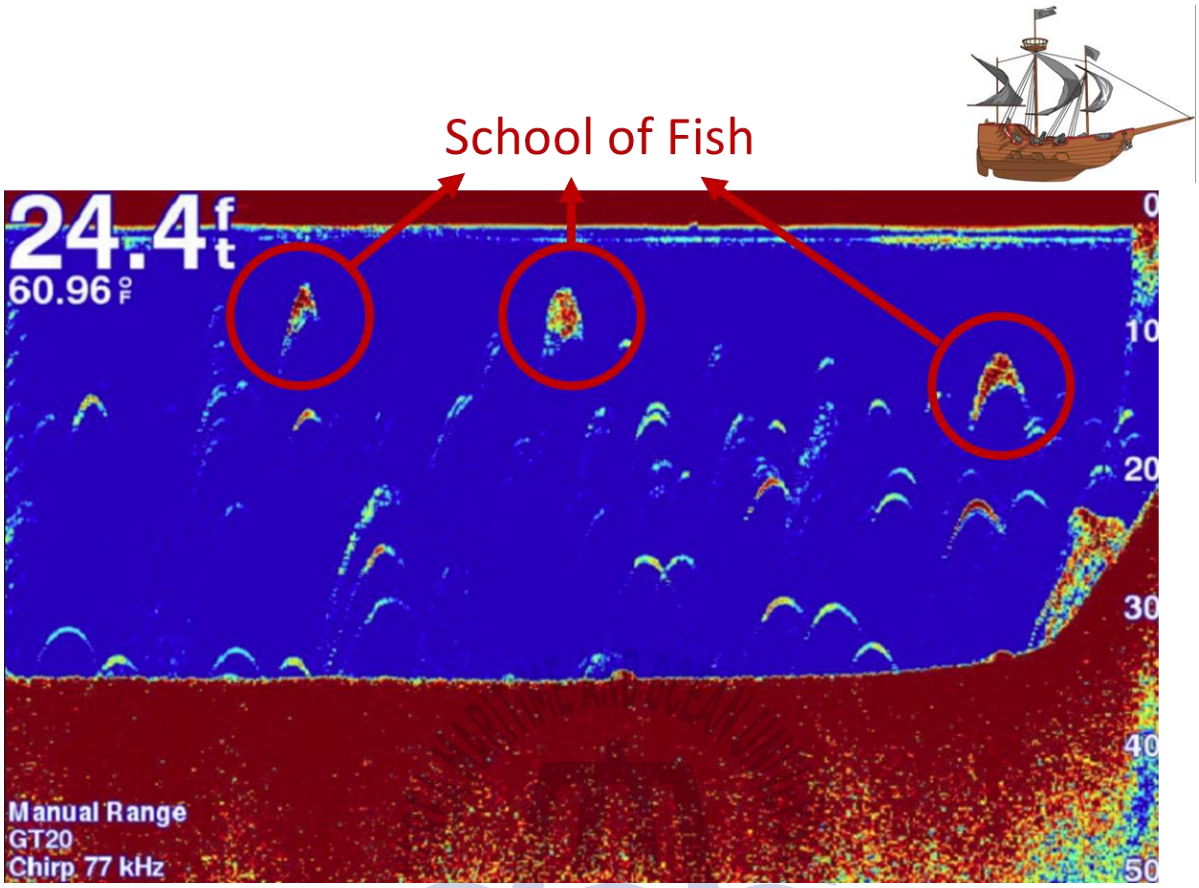


Figure 1 Example of Commercial Fish Finder Display

Several studies [2-6] explain the application of sonar system in various scientific areas. Fornshell and Tesei explained the history of sonar system development as a tool for measuring depths and finding schools of fish in the ocean. Their paper describes the rapid technological development in the field of acoustic oceanography from 1920 up to the present day [7]. In another study by Hansen, the author explains the basic principles of sonar with three different applications, namely, fish finding, imaging of the seafloor and mapping of the seafloor. The paper explains basic physics of waves and how they propagate into water. In addition, more advanced physic topics such as spherical spread of the sound, absorption, refraction, and reflection of sound were covered [8]. A group of Japanese researchers developed a remote fish finder system which

helped fishers visualize the approximate amount of fish and the fish types. The study focused on developing a sonar system with a new set-net monitoring solution which would improve the conventional equipment. Some of these improvements include lower price, smaller size and less maintenance requirement [9]. In another study, Manik explains the application of a quantified sonar system to detect and quantify the bottom echoes at the ocean. The process to transmit and receive the echo from the bottom of the sea is explained and the mathematical theory is proved by experimental results [10].

Several other papers and books [11-16] explain both the theory and application of sonar systems. While these studies represent important information about sonar systems, none of them explain the practical process of building the hardware and software of a sonar system. This lack of background in explaining how to build a complete sonar system motivated the researcher to write this manuscript. The purpose of this study is to explain the process to build both the software and the hardware necessary for a sonar system. A prototype of the sonar system was also developed to validate the proposed system. The results showed satisfactory echo of the transmitted pulses.

Chapter 2. Theory of Sonar Systems

2.1 Overview

Sonar systems are divided into two sub-categories, namely, passive and active. Passive sonars do not emit any pulse and only listen for incoming signals which makes them perfect for military applications. For example, a military vessel could detect the existence of the enemy ships or submarines by listening to the signals generated by the enemy vessels.

On the other hand, active sonars transmit pulses into the water. These pulses are called “pings” and are generated by a transmitter element array. If the transmitted signal hits an object, such as, a school of fish or an enemy vessel, a distorted signal will be reflected backward, toward the transmitter. This reflected signal is called an “echo”. Active sonars, similar with passive sonars, have a listener which is called a “receiver”. So they can capture this echo and using signal processing, the echo could be interpreted and the position and size of the object could be estimated. The echo is usually very weak and contains noise, as a result, before converting the echo into digital signal, it should be amplified and its noise should be reduced. The objective of the receiver is to find out where the signal was originated. This decision is mainly depending on the signal-to-noise ratio (SNR) of the system.

2.2 Traditional pulse and Chirp

The transmitted pulse is divided into two main types, namely, traditional tone burst (ping) and Chirp. The traditional sonar transmits a single powerful ping with a constant frequency into the water column, then it listens for the echo of the transmitted pulse. Typically, traditional sonars transmit high energy pulse but with very short duration. As a result, the total amount of energy that can be transmitted is limited. In addition, different frequencies reveal different levels of

detail. For example, 50Khz frequency can penetrate deeper in water but it provides less detail, however, 200Khz frequency can have less penetration but can show more detail. As a result, most traditional sonars use two or three different frequencies to provide better combination of details and depth.

On the other hand, the Chirp sonar (an acronym for Compressed High Intensity Radiated Pulse) is capable of transmitting longer pulses than the traditional sonar, putting more energy into the water. Chirp sonars can also generate pulses with varying frequencies as a result they capture better details of the objects inside water. In this study, traditional pulses are used, however, the structure is designed in a way that system can be upgraded to use Chirp technology.

2.3 Sonar receive sub-path

A typical receive sub-path circuit includes methods to amplify the analog signal as well as some filters to increase the dynamic range of the ADC. These improvements could be done both in analog domain and in digital domain.

In analog domain, a typical filter used before most ADCs is a low-pass filter. This filter behaves as an anti-aliasing filter. As a result, it will limit the higher-order harmonics and high-frequency noise that may aliased into the first Nyquist zone. Also in digital domain, several methods including decimation and filtering are used to improve the SNR performance and reduce the data rates to reduce the necessary FPGA processing of the data. Another digital method is called spatial filtering which is often used to provide directivity to the transmitted ping through beam forming.

2.4 Beamforming

In a Sonar system, acoustic waves are transmitted with a specific frequency into the imaging medium (in this case into water). Then the echo of the transmitted wave is received and saved as a function of time and position. Because the transmission and reception of sound waves are performed at the same location (that is, at the transducer probe), the distances through which these waves propagate can be computed through simple trigonometric geometry. This concept is schematically represented in figure below.

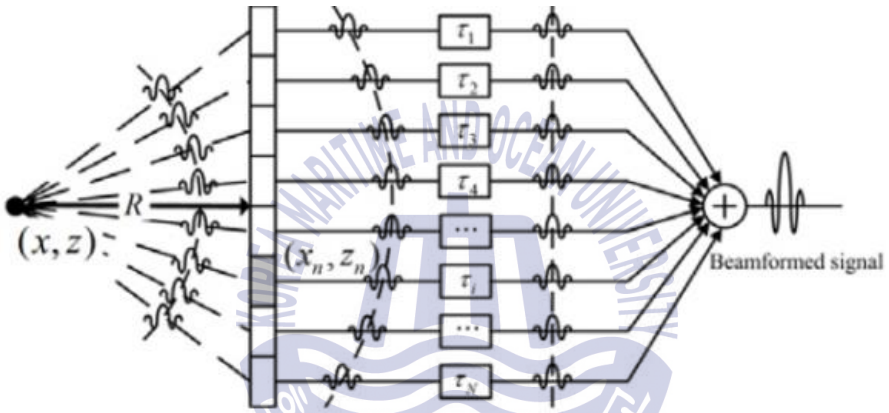


Figure 2 Principle of Delay-and-Sum Beamforming

Figure 3 represents two neighboring and independent transmitted signals from two different transducer elements. The red arrow shows the signal transmitted and received from the first transducer element, while the pink one represents the second transducer element. The received waveforms by these two independent channels include separate and different datasets. As seen in Figure *3*, the distances are different, and assuming constant speed of sound in water, the propagation times of the two channels are unequal.

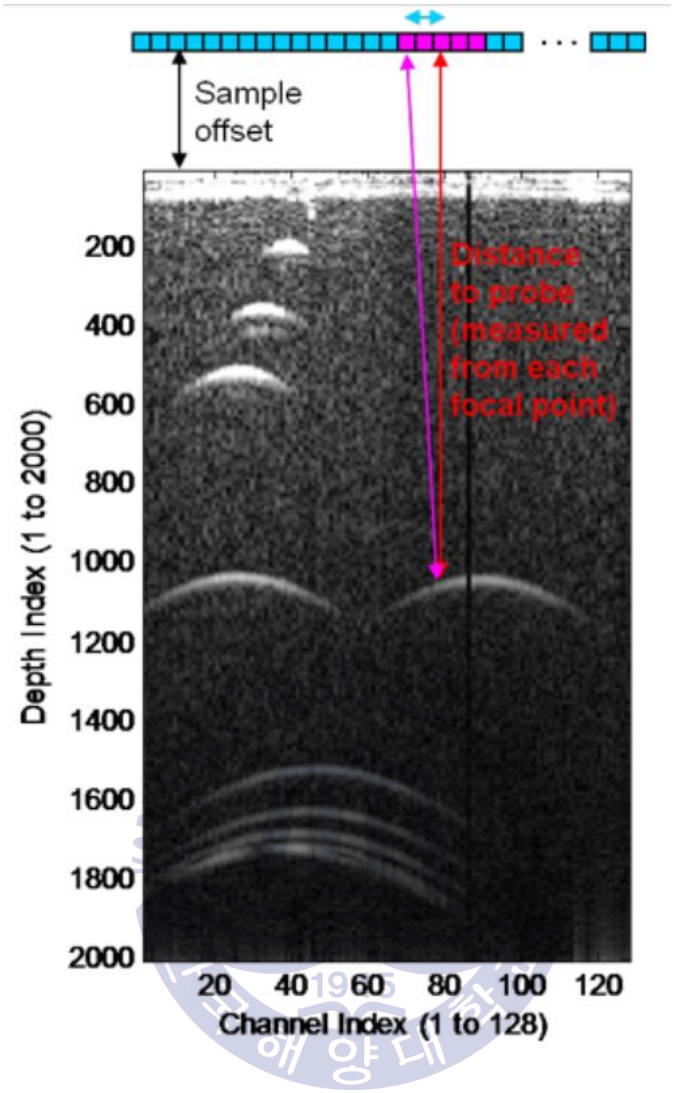


Figure 3 Geometry of Propagation Paths

2.5 Signal-to-Noise Ratio (SNR)

There are a number of methods to improve the SNR of a Sonar system both in analog and digital domains. Implementing these methods can result in better quality or the range over which the Sonar system could be used. Most Sonar systems have a transceiver which can do the job of a transmitter and a receiver at the same time.

Typically, two reasons cause the low SNR of the received echo, namely, the attenuation of the acoustic wave in the medium and the limited power output for wave transmitted by transducer.

A common approach to increase the SNR is by using multiple neighboring channels to capture the echo following the beamforming theory. The difference is timing of the neighboring channels could be cancelled out because the physical location of the transducer elements is known at the time of beamforming. As a result, the delay of the data for different channels are removed and the data are cumulatively summed to achieve higher SNR. By summing the neighboring data, the noises will cancel out each other, as a result better SNR is achieved [17].

2.6 Velocity of sound in water

The velocity of sound in water depends on several factors including water temperature, salinity, and pressure. The speed of sound in sea water is calculated from the following empirical equation:

$$c(T,S,z) = a_1 + a_2T + a_3T^2 + a_4T^3 + a_5(S-35) + a_6z + a_7z^2 + a_8T(S-35) + a_9Tz^3 \qquad (1)$$

where T is the temperature in degrees Celsius, S is the salinity in parts per thousand, z is the depth in meters, and the constants are:

$$\begin{aligned} a_1 &= 1,448.96, & a_2 &= 4.591, & a_3 &= -5.304 \times 10^{-2}, \\ a_4 &= 2.374 \times 10^{-4}, & a_5 &= 1.340, & a_6 &= 1.630 \times 10^{-2}, \\ a_7 &= 1.675 \times 10^{-7}, & a_8 &= -1.025 \times 10^{-2}, & a_9 &= -7.139 \times 10^{-13}, \end{aligned}$$

For example, the velocity of sound in water with 10℃, 1000 Kilopascals, and 3% salinity is calculated to be 1500 m/s. The below figure shows the speed of sound profile in sea water [18].

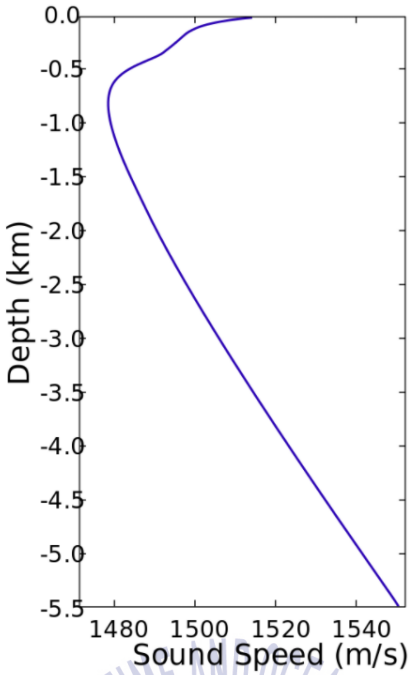


Figure 4 Speed of Sound Profile in Sea Water [19]

2.7 Sonar Transducer

A transducer is a device that converts one form of energy to another form. A sonar transducer converts electrical energy directly into mechanical energy (vibration) and vice versa. There are two main categories of transducers, namely, magnetostrictive and ceramic. Magnetostrictive transducers can tolerate very high powers without being damaged. As a result, magnetostrictive transducers are suitable for high power low frequency applications. On the other hands, ceramic sonars have better efficiency factor but they can get damaged if overloaded [20]. This 300 W plastic transom mount with CHIRP technology has a maximum depth of 1,500 ft; an operating frequency of 95 to 155 kHz; a beamwidth of 26/17 degrees; and mounts on a 3 to 20-degree transom. More information about our transducer is available at [21].

Chapter 3. Sonar Prototype Hardware

In Chapter 3, the hardware section of the proposed sonar system is described. The system is divided into two parts. First part is called transmit path sub-circuit and the second part is called receive path sub-circuit. Each sub-circuit is explained in detail.

3.1 Sonar System Prototype

In order to better understand how sonar systems work, many researchers have developed the required hardware and software for sonar systems. Sonar systems are usually divided into two sub-circuits, namely, Transmit Path Sub-Circuit and Receive Path Sub-Circuit. Figure 5 shows a sonar system diagram with its sub-circuits. The parts shown in are explained in details below.

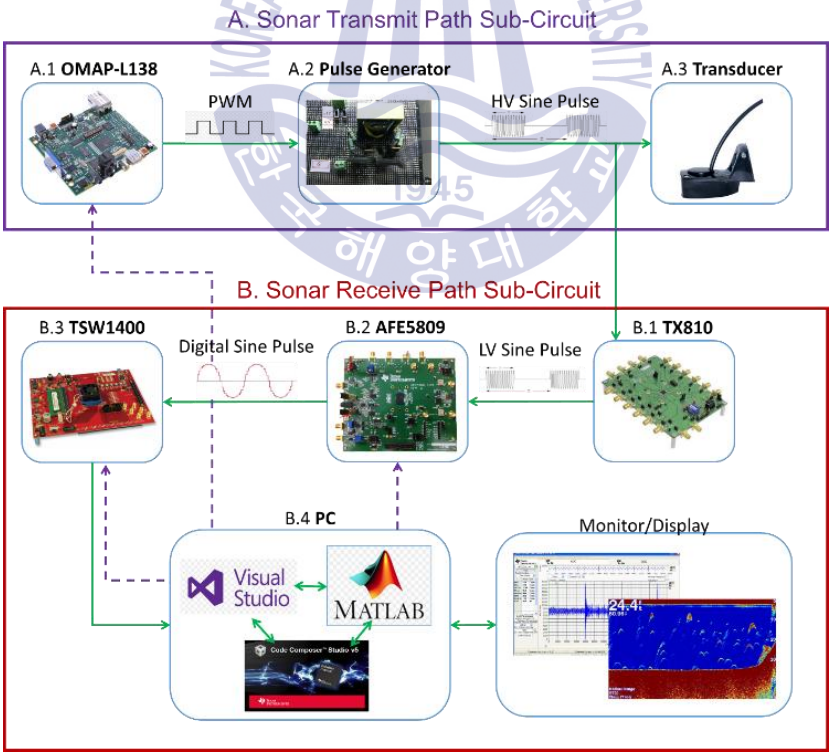


Figure 5 Sonar System Diagram

3.2. Sonar Transmit Path Sub-Circuit

The high voltage pulse is generated and transmitted in sonar transmit path sub-circuit. Figure 6 illustrates this sub-circuit.

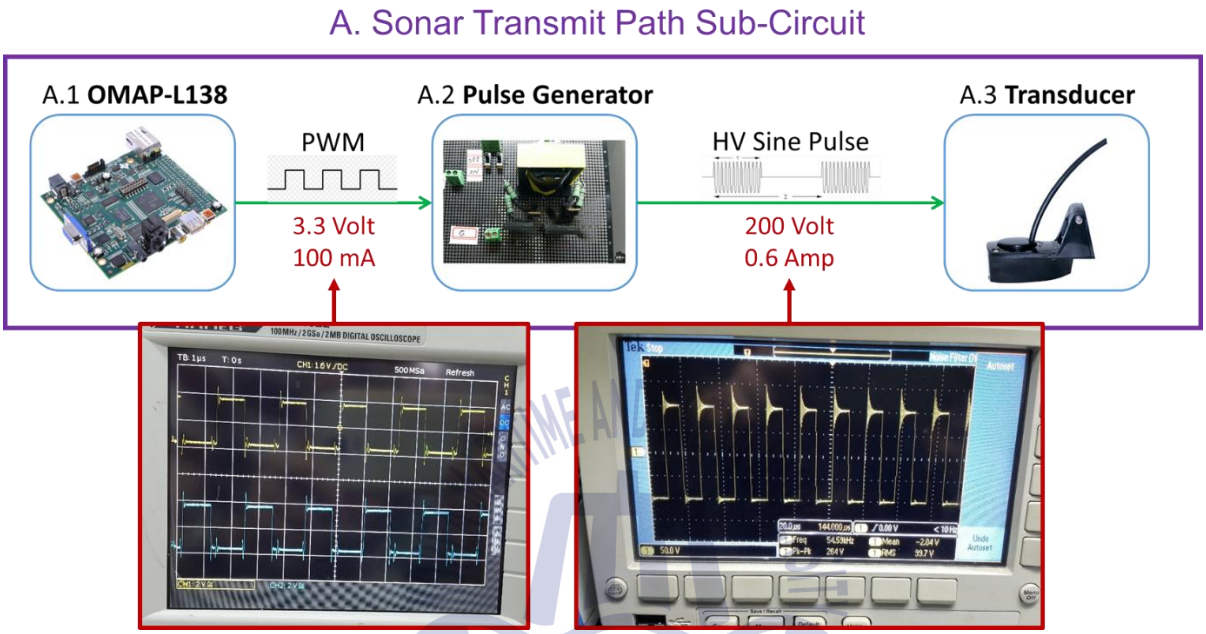


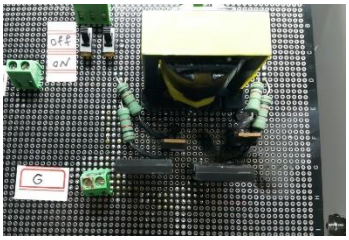
Figure 6 Sonar Transmit Path Sub-Circuit

Each part of the transmit path sub-circuit is explained in more detail below.

3.2.1 OMAP-L138:

The ***OMAP-L138*** C6000 DSP+ARM processor is a low-power applications processor based on an ARM926EJ-S and a C674x DSP core. The two Enhanced High-Resolution Pulse-Width Modulator (eHRPWM) of ***OMAP-L138*** are used to generate two PWM signals. These PWM signals are given as input to ***HV Pulse Generator.***

3.2.2 HV (High-Voltage) Pulse Generator:



A custom high voltage pulse generator was developed. This pulse generator gets the ePWM output of OMAP-L138 as input, after that, the ePWM is amplified to about 250 volts peak-to-peak. Then, a sine shape is made of the amplified ePWM. The high-voltage sine shape is connected to the Transducer and the TX810.

3.2.3 Transducer:

The ***A.3 Transducer*** transmits sine shape pulses into the water. Then it receives the echo of those pulses and gives the echo to ***B.1 TX810***.

3.3. Sonar Receive Path Sub-Circuit

The echo of the received signal is captured and displayed in a PC using the receive path sub-circuit. Figure *7* shows the diagram of this sub-circuit.

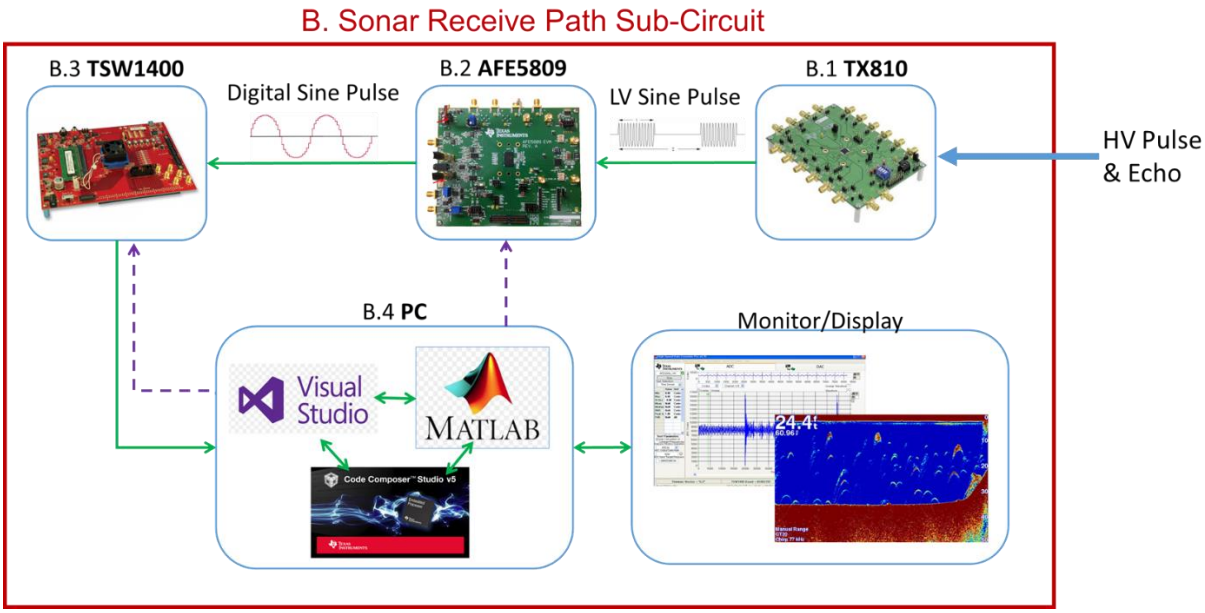
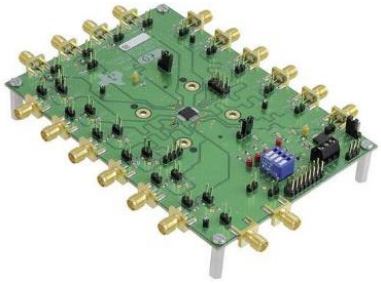


Figure 7 Receive Path Sub-Circuit

Each component of this sub-circuit is explained in detail below.

3.3.1 TX810

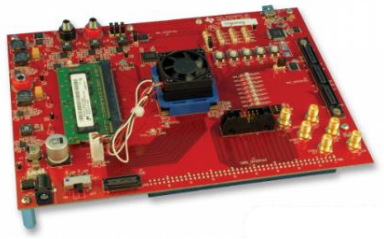


The **TX810** [22] is a T/R switch which is used to prevent high voltage signals from the ***A.2 HV Pulse Generator*** directly getting to the ultrasound ***B.2 AFE5809***'s lower voltage amplifier input pins.

3.3.2 AFE5809

The ***B.2 AFE5809*** device [23] is a highly-integrated analog front-end (AFE) solution specifically designed for ultrasound systems in which high performance and small size are required. The ***B.2 AFE5809*** receives the analog echo of the pulse from ***B.1 TX810.*** Then, it converts this analog signal into digital signal (ADC). This digital signal is then given to the ***B.3 TSW1400*** as input.

3.3.3 TSW1400

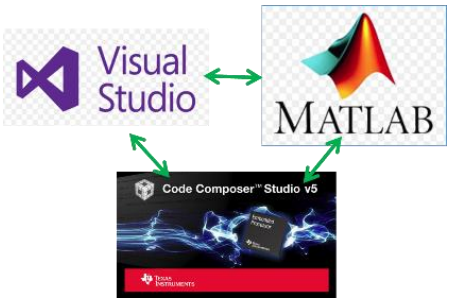


The **TSW1400** [24] is a complete data capture circuit board used to capture and analyze the digital output of ***B.2 AFE5809***.

3.2.3 Transducer

The ***A.3 Transducer*** transmits sine shape pulses into the water. Then it receives the echo of those pulses and gives the echo to ***B.1 TX810***.

3.3.4 PC



The ***B.4 PC*** controls three components of this sonar system, namely, ***A.1 OMAP-L138***, ***B.2 AFE5809*** and ***B.3 TSW1400***. A total of five different software are used (Visual Studio, Code Composer Studio, Matlab, AFE5809 GUI, and HSDC Pro).

3.4. Details of each sonar component

In the next sections, each component of this sonar system is explained in detail.

3.4.1 OMAP-L138

The OMAP-L138 C6000 DSP+ARM processor is a low-power applications processor based on an ARM926EJ-S and a C674x DSP core. This processor provides significantly lower power than other members of the TMS320C6000™ platform of DSPs. Figure 8 shows the OMAP-L138 EVM (Evaluation Module).

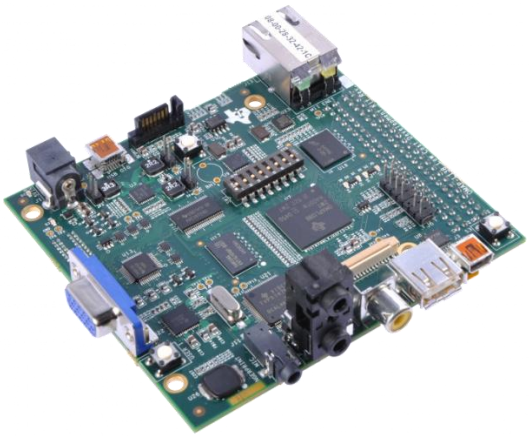


Figure 8 OMAP-L138 LCDK [25]

This device has double cores. A combination of TMS320C674x DSP core and ARM926EJ-S core. OMAP L-138 is used to generate custom ePWM signals for our sonar system prototype. Figure 9 shows the functional block diagram of OMAP-L138 [25].

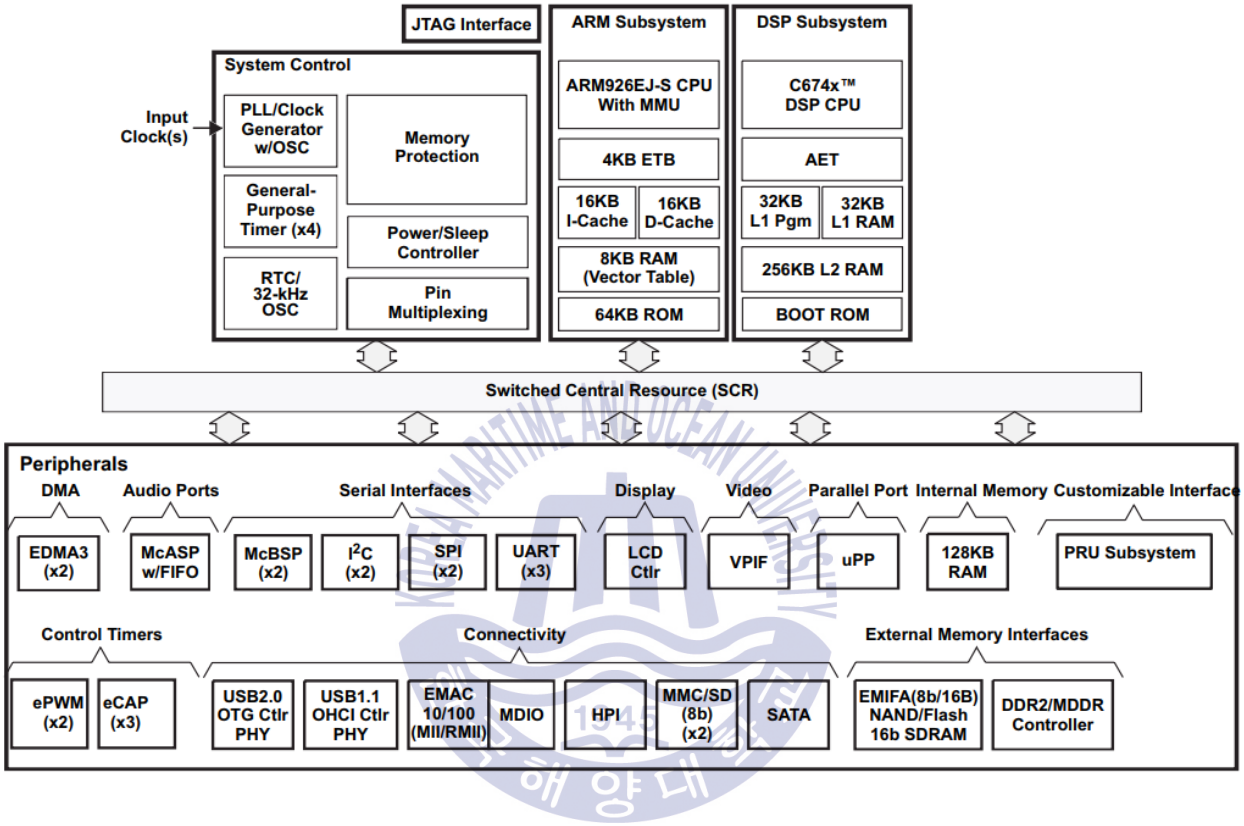


Figure 9 OMAP-L138 Functional Block Diagram [25]

The schematic of OMAP-L138 is shown in Appendix H.

3.4.2 OMAP-L138's Enhanced High-Resolution Pulse-Width Modulator (eHRPWM)

The OMAP-L138 contains two enhanced PWM modules (eHRPWM). These eHRPWMs are used to generate two PWM pulses. The generated pulses are then given to ***A.2 HV Pulse Generator*** as input. Figure 10 shows the block diagram of these two eHRPWM modules.

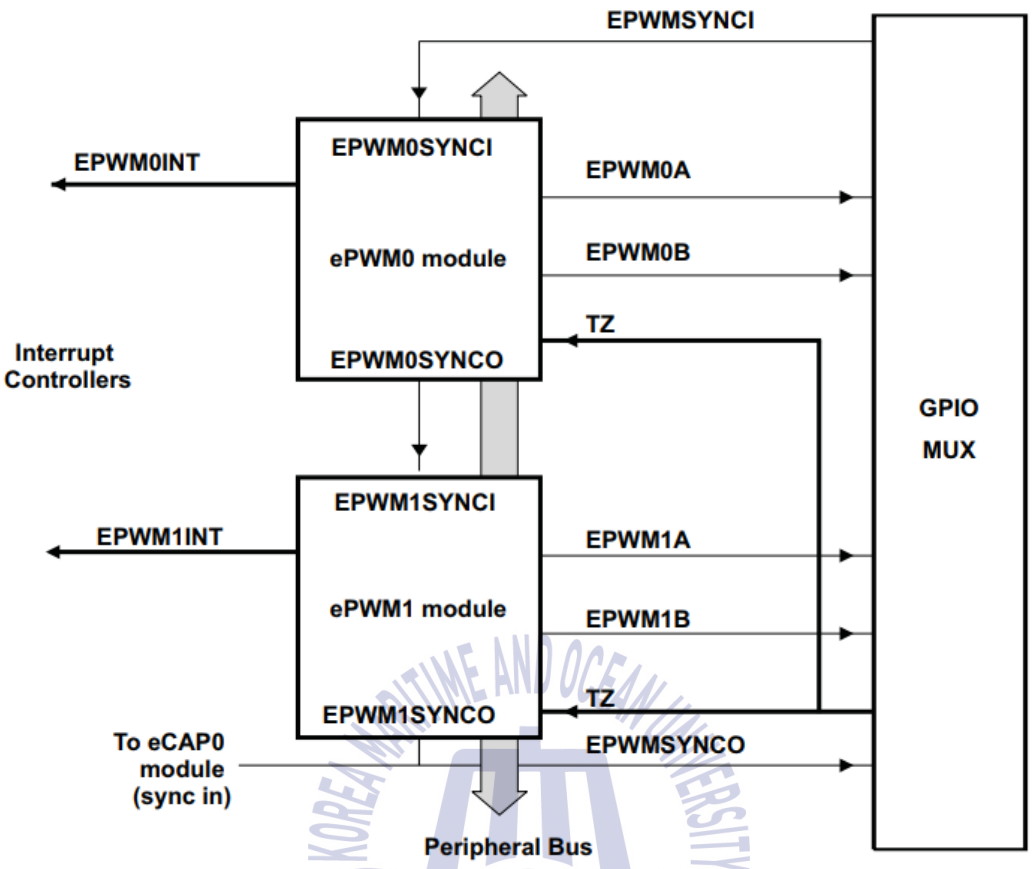


Figure 10 Multiple PWM Modules in a OMAP-L138 System [25]

Figure 11 shows the signal interconnections with the eHRPWM. Each submodule has its own functionality. Code Composer Studio and C++ language are used to set the proper register values for each submodule.

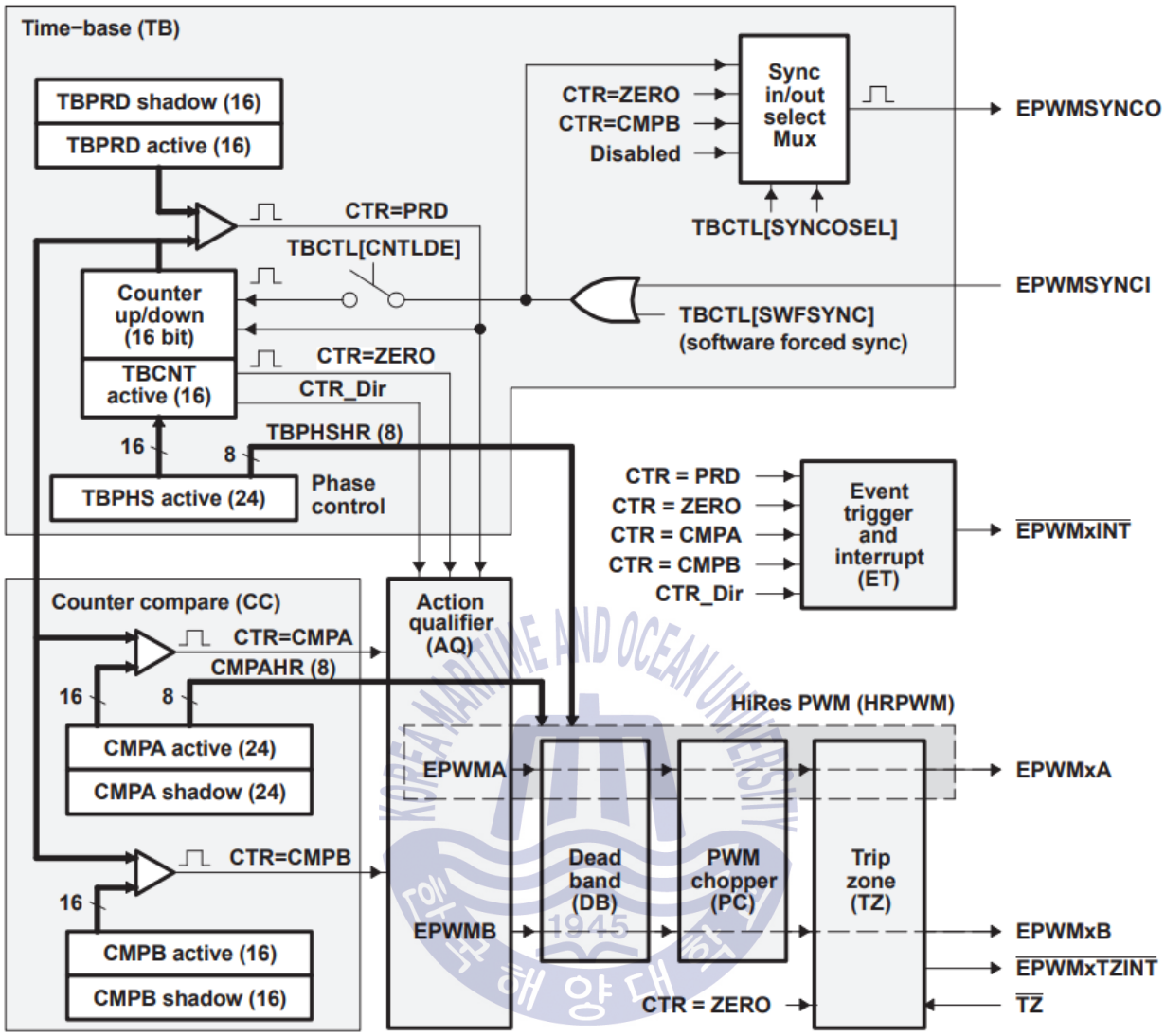


Figure 11 eHRPWM Sub-Modules Showing Critical Internal Signal Interconnections [25]

3.4.3. HV (High-Voltage) Pulse Generator

A custom board was designed to convert the low voltage ePWM generated by OMAP L138 to high-voltage sine pulse. Below is a photo of this pulse generator.

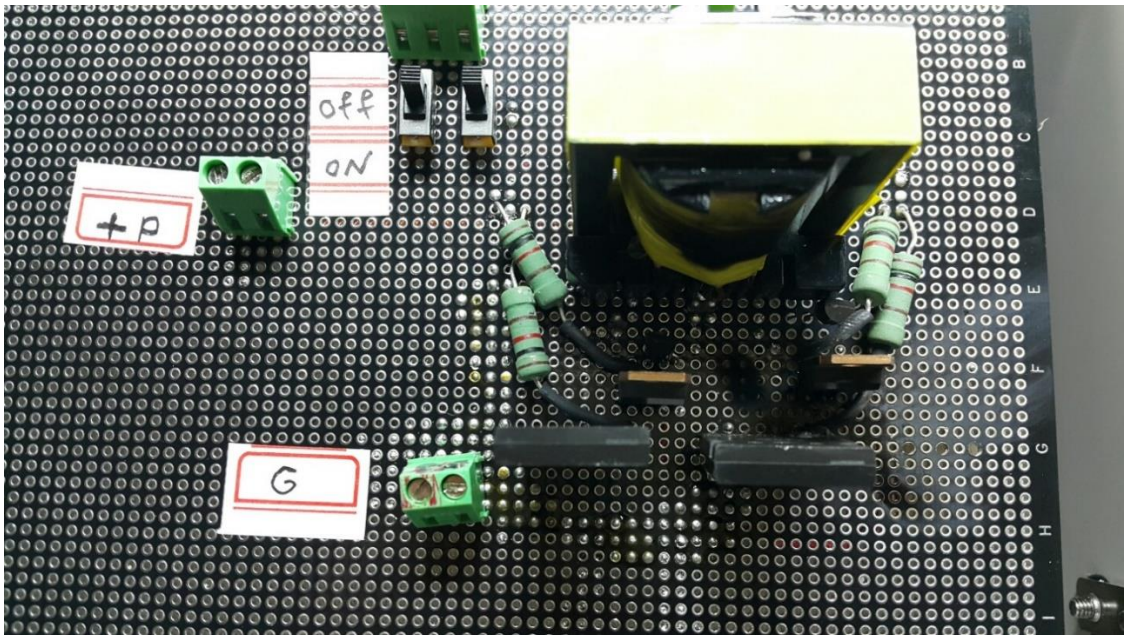


Figure 12 Custom HV pulse generator

The HV pulse generator has two main inputs. A power input with DC 24 volt/ 30 ampere and a pair of ePWM inputs. Two low voltage ePWMs are generated by OMAP L138. These ePWMs are then connected to the inputs of the custom HV pulse generator. These low voltage ePWMs will then activate two transistors. These two transistors in turn will activate another two power MOSFETs. The MOSFETs will then transmit the high voltage input into the transformer. The transformer will then amplify the input voltage 4 times. So, the input 24 volt will be about 100 volts (RMS) on the other side of the transformer. Figure 13 shows the shape of signal after the transformer. The schematic of HV pulse generator is shown in Figure 13.

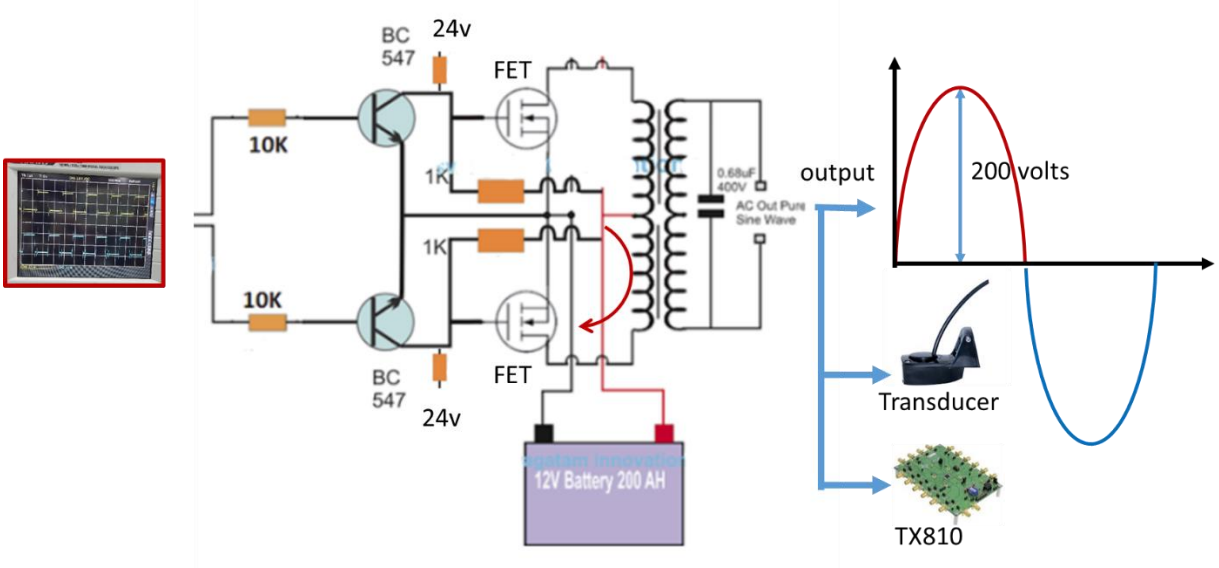


Figure 13 Schematic of HV Pulse Generator

A capacitor is then added after the transformer to convert the ePWM shape into the sine shape. Various capacitors with different capacities have been tested to find out the best sine shape. Below a photo of the high-voltage sine shape (after adding capacitor) is shown. A schematic of the custom HV pulse generator is shown at Appendix B.

3.4.4. Transducer

Ultrasonic transducers are divided into three broad categories: transmitters, receivers and transceivers. Transmitters convert electrical signals into ultrasound, receivers convert ultrasound into electrical signals, and transceivers can both transmit and receive ultrasound. Figure 14 shows the sonar transceiver used in this work. The frequency of this transducer is 95 KHz and it requires 300W power.

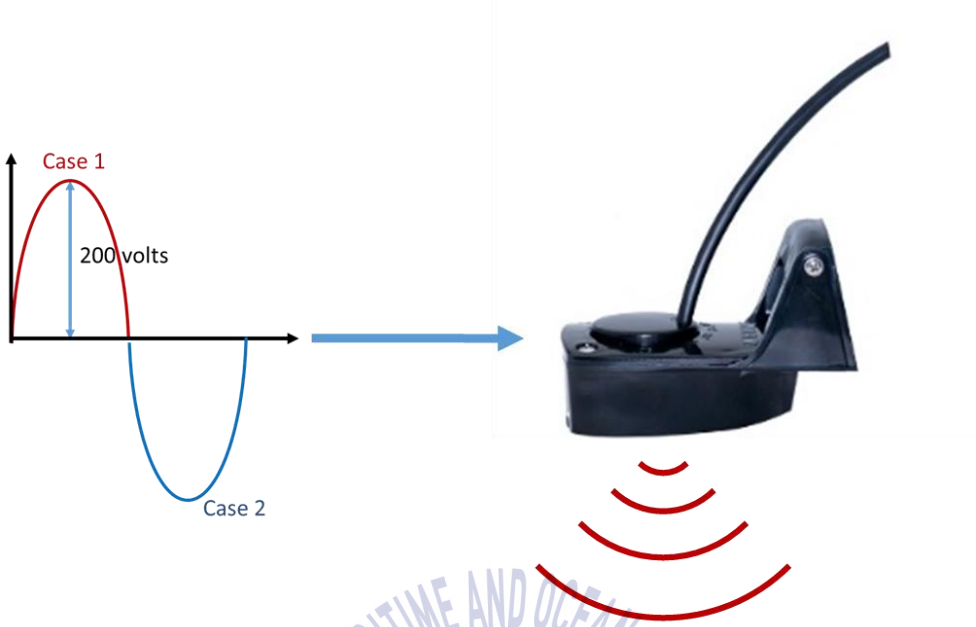


Figure 14 Sonar Transducer (TM150m)

3.4.5. TX810:

The **TX810** is a transmit/receive (T/R) switch which is used to prevent high voltage signals from the ***A.2 HV Pulse Generator*** directly getting to the ultrasound ***B.2 AFE5809***'s lower voltage amplifier input pins.

This transmit/receive switch alternately connects the ***A.2 HV Pulse Generator*** and ***B.2 AFE5809*** to a shared transducer. When the ***A.2 HV Pulse Generator*** is active, the resulting high voltage causes the circuit to conduct, shorting together the ***B.2 AFE5809's*** terminals to protect it.

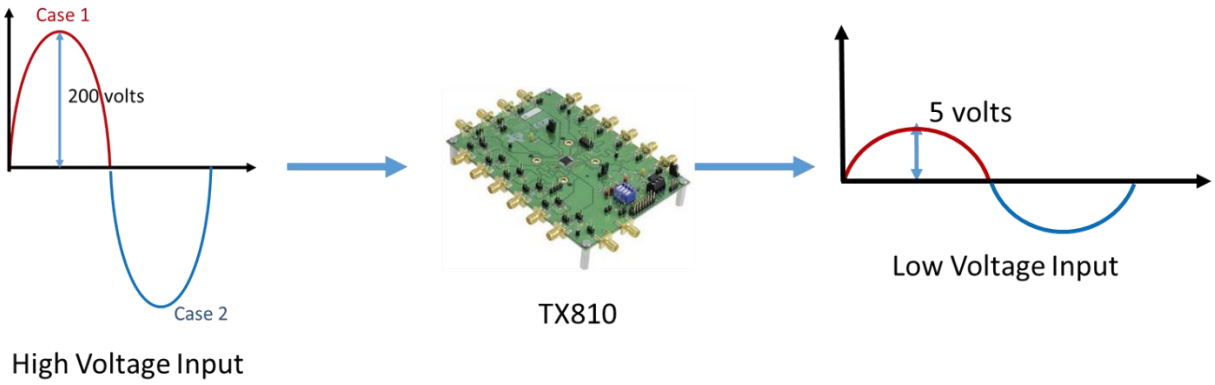


Figure 15 TX810EVM

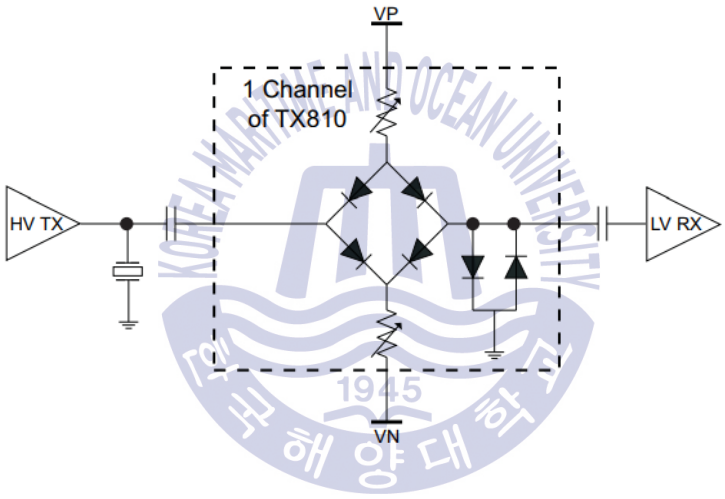


Figure 16 Block Diagram of TX810

3.4.6. AFE5809

The AFE5809 device is a highly-integrated analog front-end (AFE) solution specifically designed for ultrasound systems in which high performance and small size are required. The AFE5809 device integrates a complete time-gain-control (TGC) imaging path and a CWD path. The device also enables users to select one of various power and noise combinations to optimize system performance. Therefore, the AFE5809 device is a suitable ultrasound AFE solution not

only for high-end systems, but also for portable ones [23]. Figure 17 shows the AFE5809 GUI and the name of important parts of the AFE5809 are magnified.

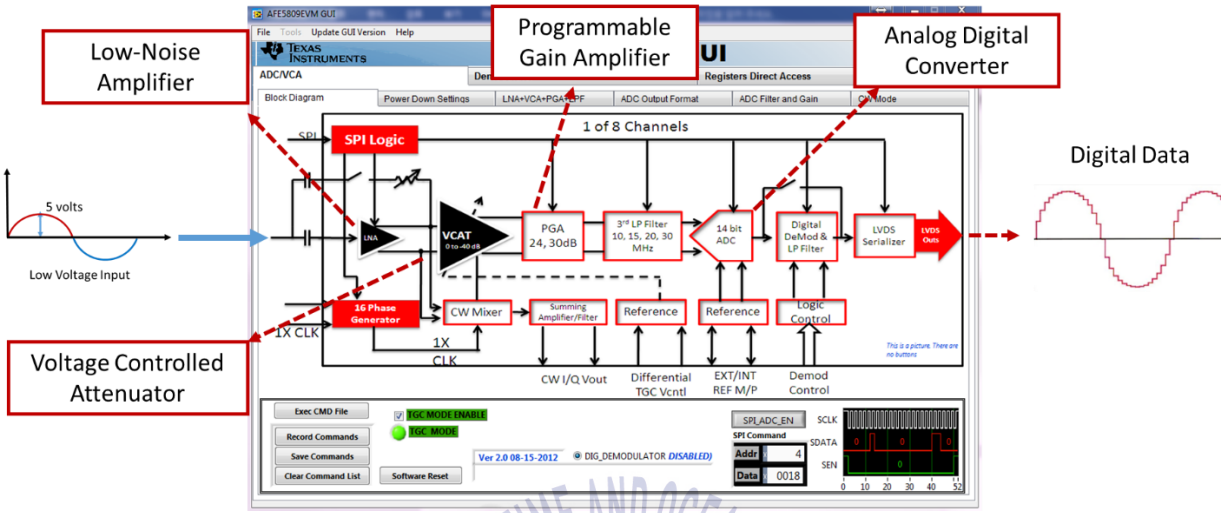


Figure 17 AFE5809 Evaluation Module [23]

The AFE5809 device contains several sub-modules, including 8 echo input channels, 8 voltage controlled amplifier (VCA), 14, and 12-bit ADC, and CW mixer. The AFE5809's functional diagram is shown in Figure 18.

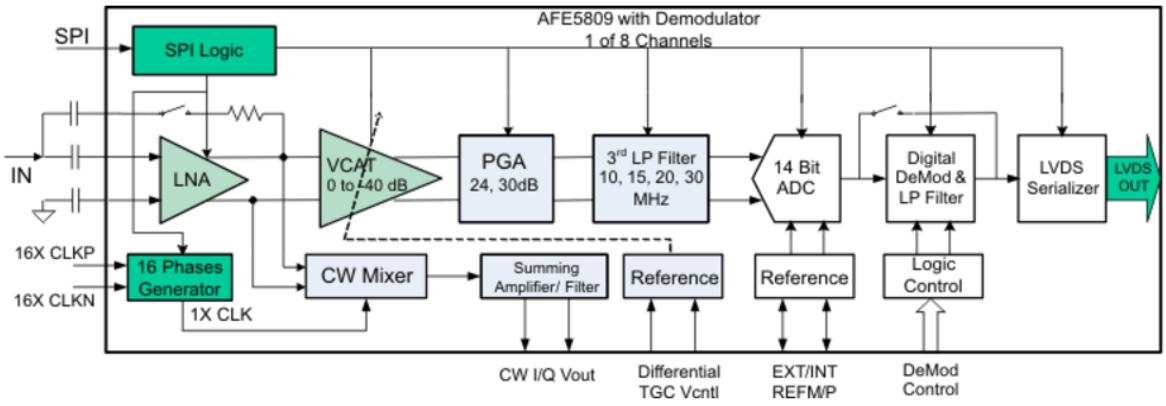


Figure 18 AFE5809 Functional Diagram [23]

In our prototype, we only used one channel of the AFE5809 because we used one transducer. The echo was received by AFE5809, voltage was amplified and noise was reduced. After that, using the 12-bit Analog-Digital Convertor (ADC) submodule, the signal was converted into digital values. A sample of these digital values (ADV values) are shown in Figure 19. Also, the schematic of AFE5809 EVM is shown in Appendix F.

	A	B	C	D	E	F	G	H
100776	-2	2256	1	-16	-19	-7	11	11
100777	-3	2307	-2	-13	-19	-10	6	14
100778	-3	2346	-4	-15	-20	-7	8	12
100779	-3	2433	-1	-14	-18	-11	8	12
100780	-4	2496	1	-14	-16	-8	8	12
100781	-5	2515	0	-13	-19	-7	6	12
100782	-3	2614	2	-12	-20	-7	4	13
100783	-1	2675	-1	-10	-19	-9	3	10
100784	-4	2690	0	-14	-21	-8	4	11
100785	-2	2811	2	-10	-15	-6	6	11
100786	-4	2862	2	-11	-18	-9	6	13
100787	-4	2899	1	-12	-16	-7	6	16
100788	-1	2999	3	-11	-15	-8	7	13
100789	-3	3056	2	-10	-15	-7	6	10
100790	-1	3122	5	-9	-17	-8	8	12
100791	-3	3182	7	-13	-14	-4	9	16
100792	1	3271	3	-11	-16	-5	11	14
100793	0	3321	4	-13	-16	-5	6	15
100794	-1	3382	4	-13	-16	-3	8	17
100795	-2	3454	3	-13	-15	-2	11	13
100796	4	3530	7	-13	-17	0	13	16
100797	2	3573	4	-12	-18	0	9	15
100798	1	3637	8	-15	-17	-5	9	13
100799	3	3705	6	-9	-14	-3	12	15
100800	2	3747	7	-14	-16	-5	11	15
100801	3	3812	8	-16	-17	-7	13	15
100802	1	3866	4	-14	-16	-7	14	17
100803	5	3913	4	-15	-15	-5	17	18
100804	3	3964	5	-16	-16	-7	18	17
100805	3	4010	5	-12	-16	-2	16	19
100806	0	4050	6	-12	-14	-3	20	17
100807	-1	4102	4	-13	-14	-5	21	17
100808	0	4143	6	-9	-17	-5	18	19
100809	2	4188	4	-10	-18	-3	22	20
100810	-4	4218	0	-10	-17	-1	21	20
100811	-1	4254	-1	-11	-14	-4	21	21
100812	1	4291	-1	-13	-16	-2	21	25

Figure 19 Parts of one capture of the signal. Columns A-H represent 8 channels of AFE5809 and rows show ADC values

3.4.7 TSW1400

The **TSW1400** is a complete data capture circuit board used to capture and analyze the digital output of ***B.2 AFE5809***. Figure 20 shows the TSW1400EVM used in this study.

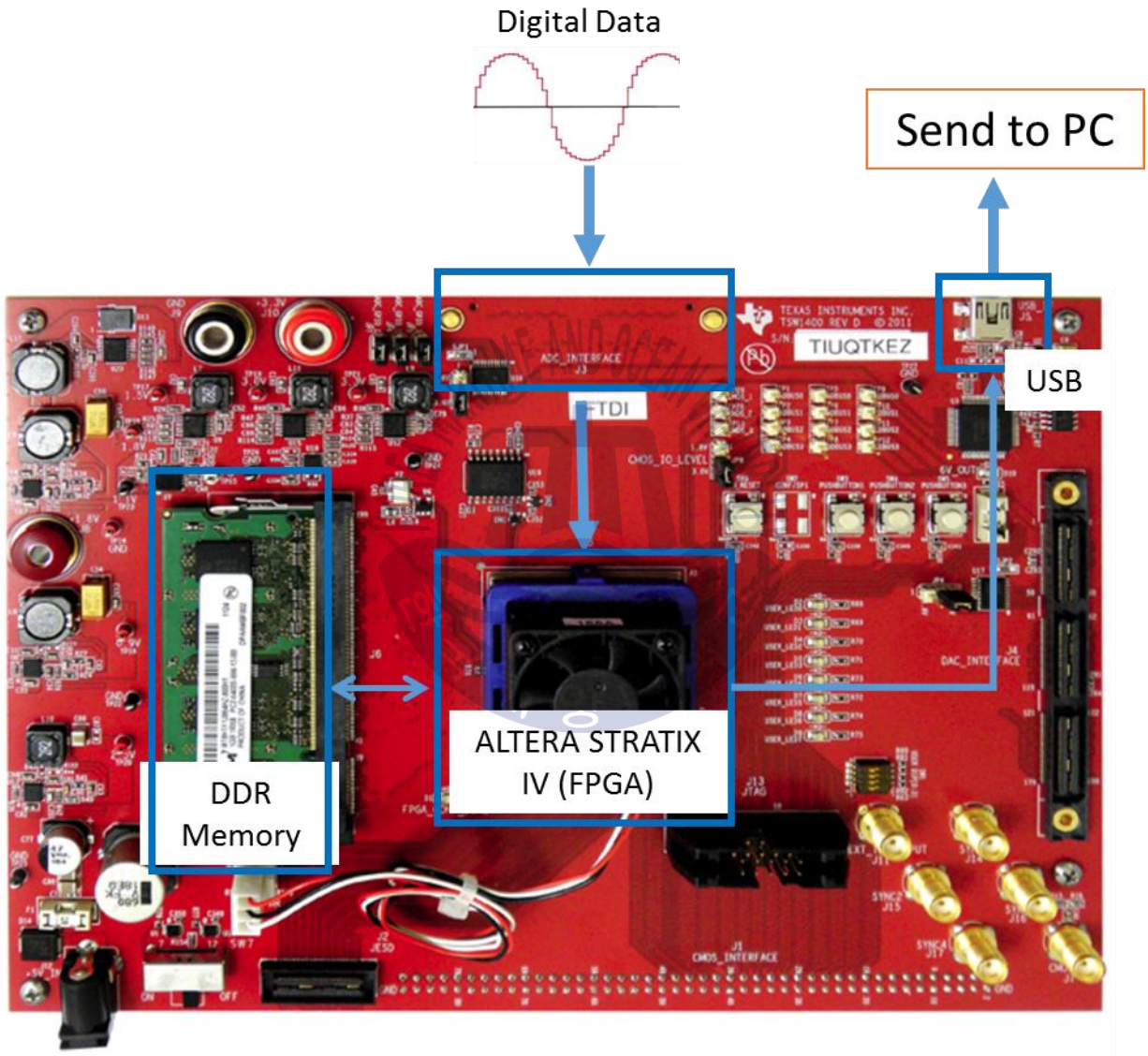


Figure 20 TSW1400EVM Data Capture / Pattern Generator

The TSW1400EVM features a high speed LVDS bus capable of providing 16-bits of data at 1.5 GSPS. Additionally, the board will support a dual 16-bit CMOS interface that will be enabled in a future firmware upgrade. The board comes with 1GB of memory, which provides a 16-bit sample depth of 512 MB. This improved memory enables better FFT frequency resolution and larger capture sizes for more realistic tests. TSW1400EVM's Block Diagram is shown in Figure 21.

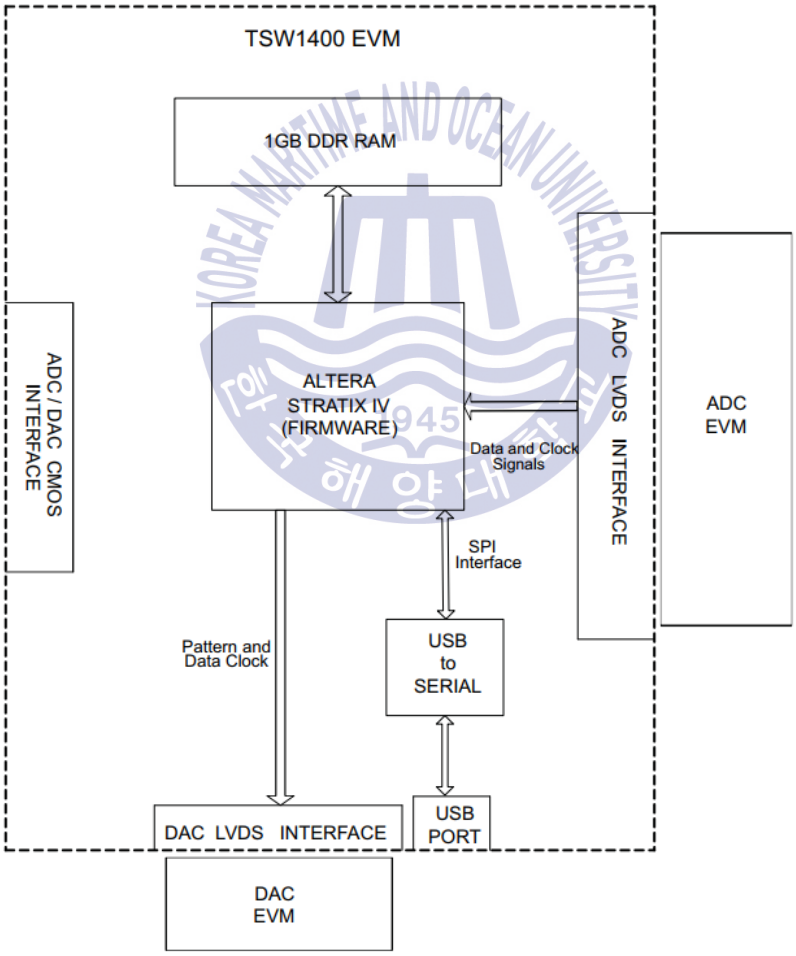


Figure 21 TSW1400EVM Block Diagram [24]

Figure 22 shows the ***B.2 AFE5809*** connected to the ***B.3 TSW1400***.

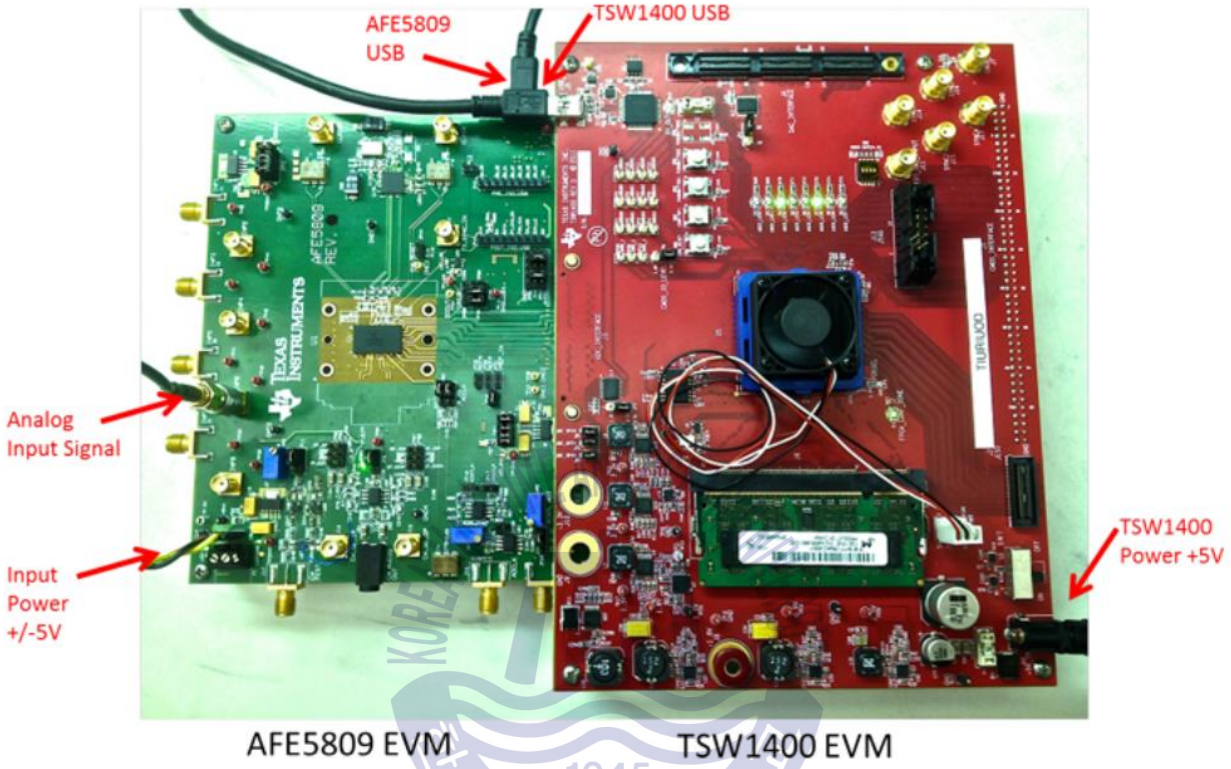


Figure 22 Receive Sub-Circuit EVM Setup

The AFE5809 generates raw ADV values. These values are then transferred to TSW1400 using LVDS connection. The TSW1400 then captures these raw values and temporarily stores them in an on-board 1 Gigabyte memory using the FPGA processor. These data are then transferred into the PC using a USB cable for further analysis.

Chapter 4. Sonar Prototype Software

Chapter 4 deals with the software part of the sonar system. As shown in

Figure 5 *Sonar System Diagram*, the *B.4 PC* controls three components of this sonar system, namely, *A.1 OMAP-L138*, *B.2 AFE5809* and *B.3 TSW1400*. A total of five different software are used (Visual Studio, Code Composer Studio, Matlab, AFE5809 GUI, and HSDC Pro). Figure 23 shows the PC software block diagram.

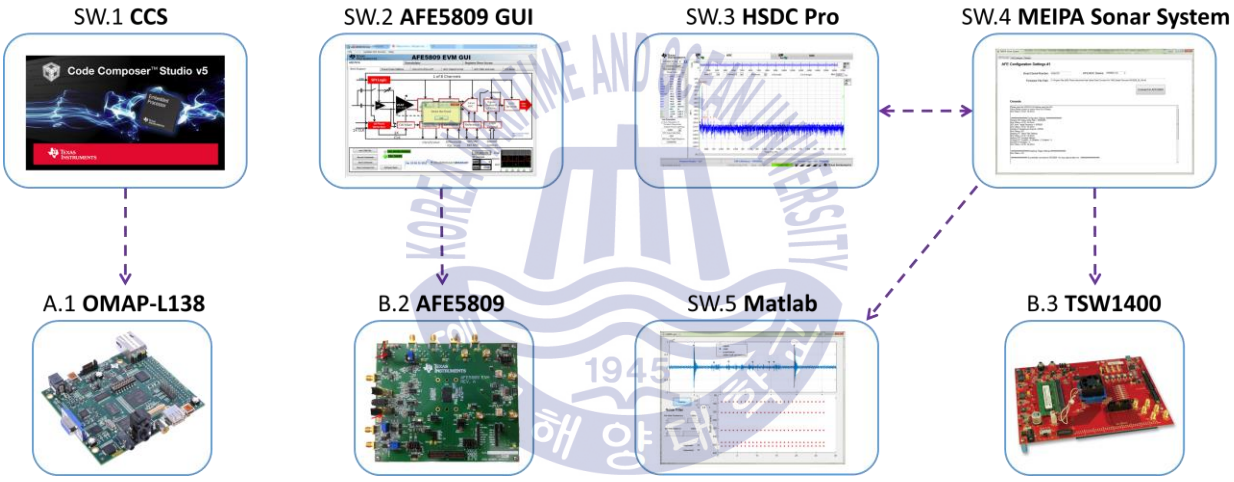


Figure 23 PC Software Block Diagram

4.1 SW.1 CCS: Control of *A.1 OMAP-L138*

Code Composer Studio (CCS) is an integrated development environment (IDE) that supports TI's Microcontroller and Embedded Processors portfolio. Code Composer Studio comprises a suite of tools used to develop and debug embedded applications. It includes an optimizing C/C++

compiler, source code editor, project build environment, debugger, profiler, and many other features.

By using C++ and CCS, ***A.1 OMAP-L138's*** eHRPWM module is configured and two PWM signals are generated.

4.2 SW.2 AFE5809 GUI: Control of *B.2 AFE5809*

Different register values of ***B.2 AFE5809*** should be configured to ensure the proper functionality of this module. AFE5809 GUI, a software developed by TI, is used for this purpose. A screenshot of this software is shown in Figure 24.

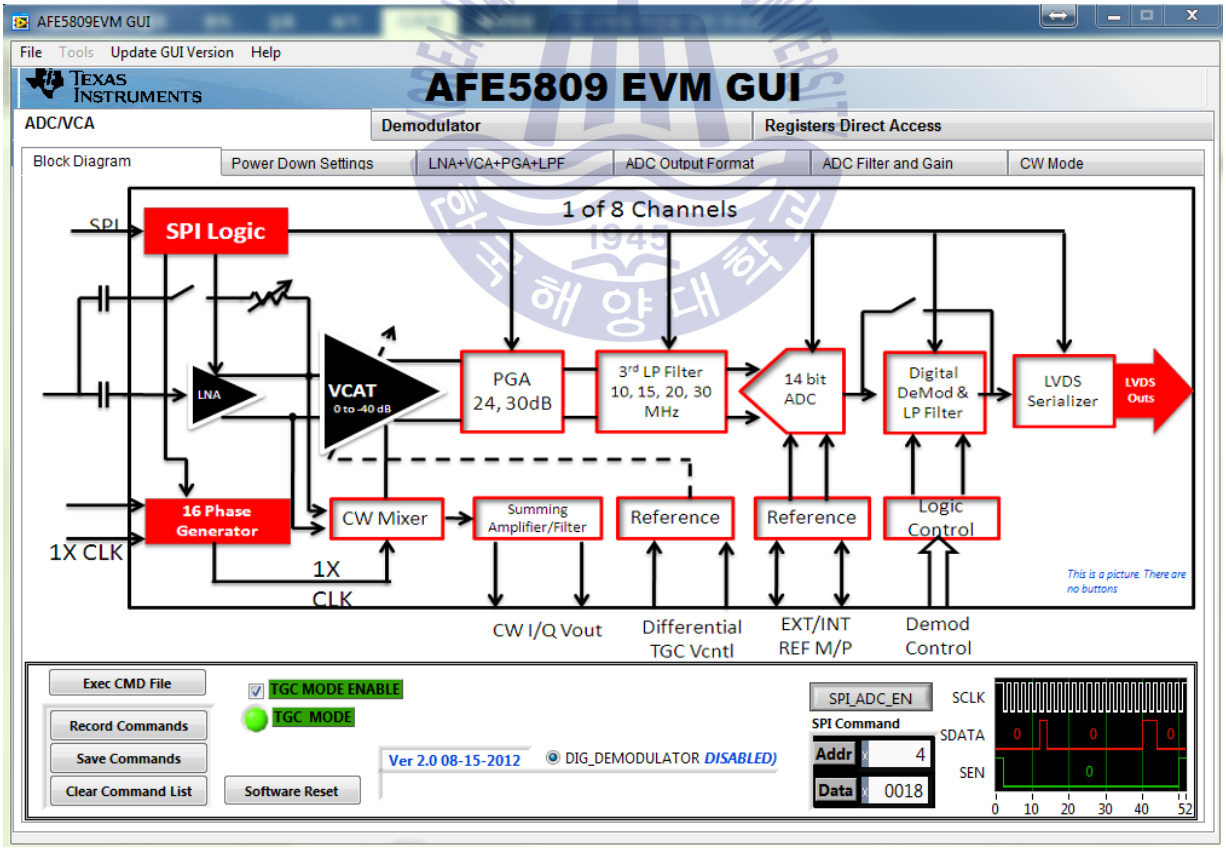


Figure 24 AFE5809 GUI

4.3 SW.3 HSDC Pro: Control of *B.3 TSW1400*

Several different register values of ***B.3 TSW1400*** should be configured to make sure the board functions properly. As a result, High speed data converter (HSDC) Pro software was used. HSDC Pro provides the tools to connect and capture data from ***B.2 AFE5809*** board.

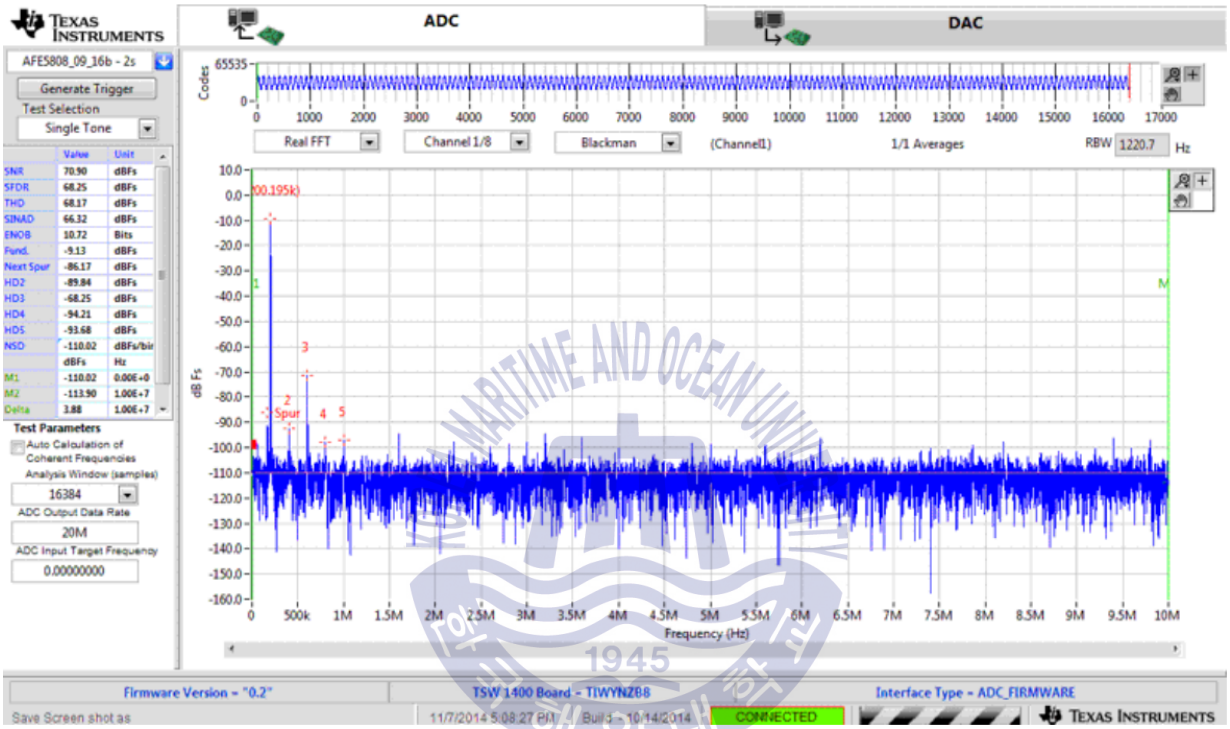


Figure 25 HSDC Pro - Capture Signal from **B.2 AFE5809**

The HSDC Pro software can capture the data from the ***B.2 AFE5809*** manually. However, for this application, the whole process should be automated. To achieve this automation goal, Visual C++ was used to make a custom code and GUI. Using this custom code, the process of capturing data from ***B.2 AFE5809*** was automated. Also, the data was then saved as a .csv file in a directory. This process is explained in more detail below.

4.4 SW.4 Control GUI:

Using Visual C++, a custom code and GUI, namely, *Control GUI* Software, was design to automate the process of capturing data from ***B.2 AFE5809*** and saving the data in a directory. *Control GUI* Software has 3 tabs, namely, AFE Connect, AFE Configure, and Results tabs. Each tab is explained in more details below.

4.4.1 Tab #1 AFE Connect:

In AFE Connect tab, one can select the AFE device type (in this case AFE5809) and file path firmware to the device. Using these information, then we can connect to the AFE board. Figure 40 shows a screenshot of AFE Connect tab of Control GUI Software.

A snippet of the code behind the Control GUI Software is shown in Appendix A.

4.4.2 Tab#2 AFE configure:

As the name implies, in this tab, different configurations of AFE are set. These configurations include, FFT settings, number of bins to remove after DC, File save settings, FTT Averaging, and Trigger settings. Figure 41 shows a screenshot of AFE Configure tab of Control GUI Software.

4.4.3 Tab#3 Results:

The results of captures data are shown in this tab. Also, by clicking on Capture data, the result will be saved in a directory in the hard disk. Saving this data automatically in a directory is

important because this data is later used in Fish Finder GUI to carry out signal processing. The process is explained in details in next section.

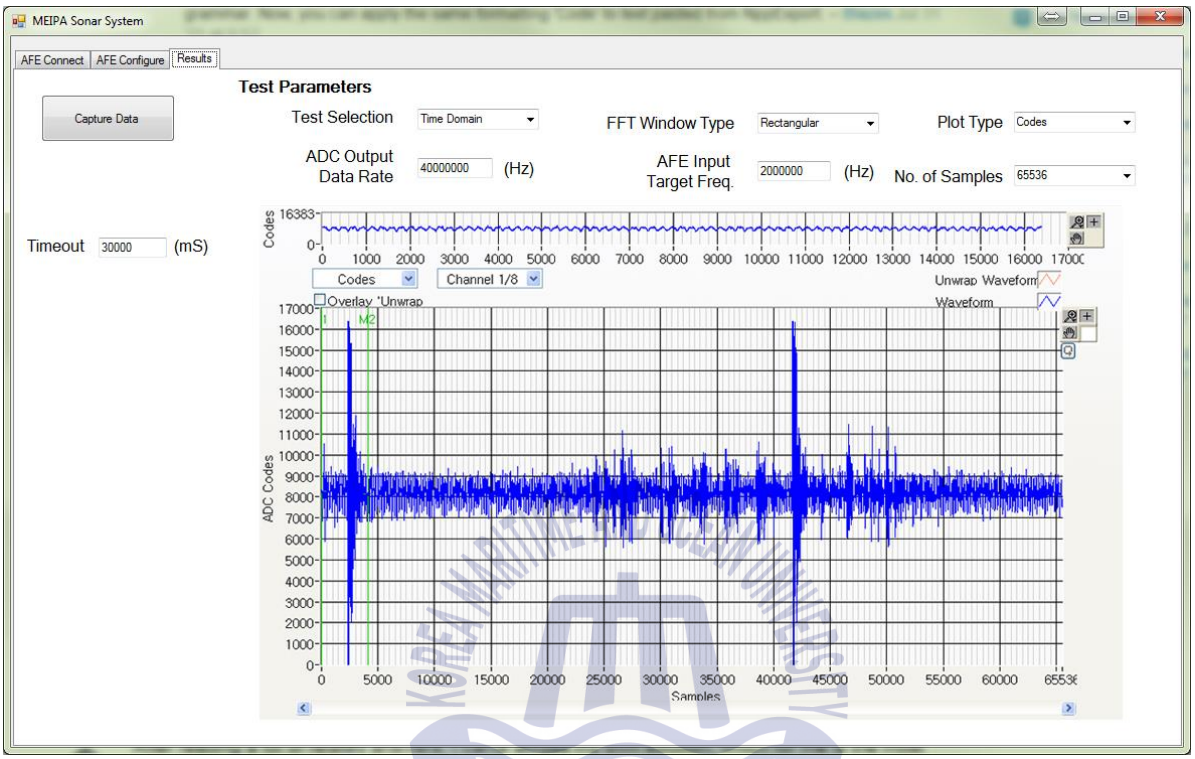
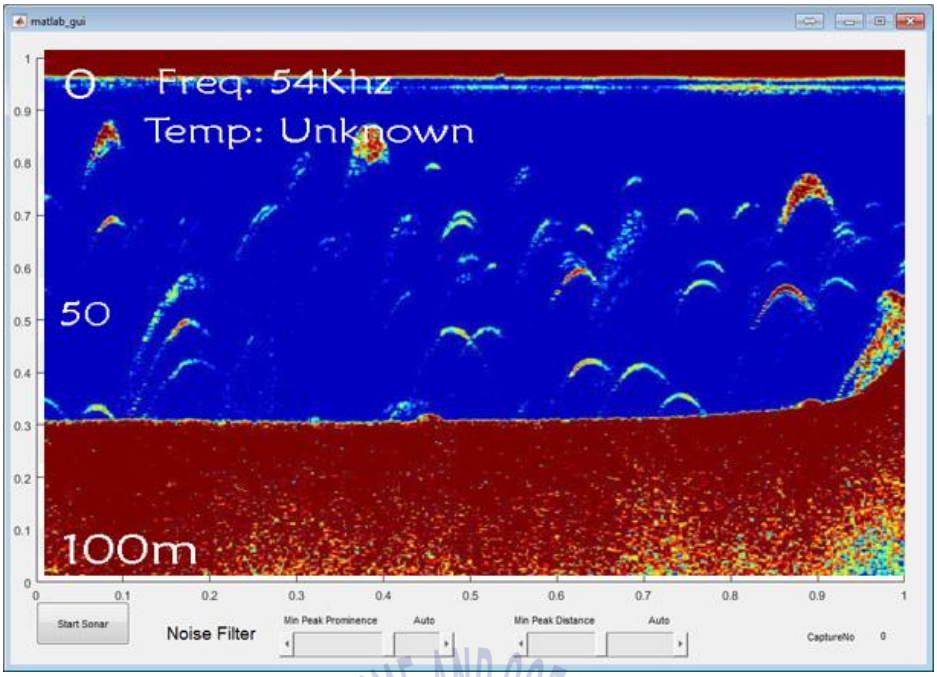


Figure 26 Results tab of Control GUI Software

4.5 SW.5 Fish Finder GUI: Signal Processing and GUI

Matlab Signal Processing toolbox provides an ideal solution to our signal processing requirements. As a result, the Fish Finder GUI code was written in Matlab Language. A screenshot of this Fish Finder GUI is shown in Figure 27.

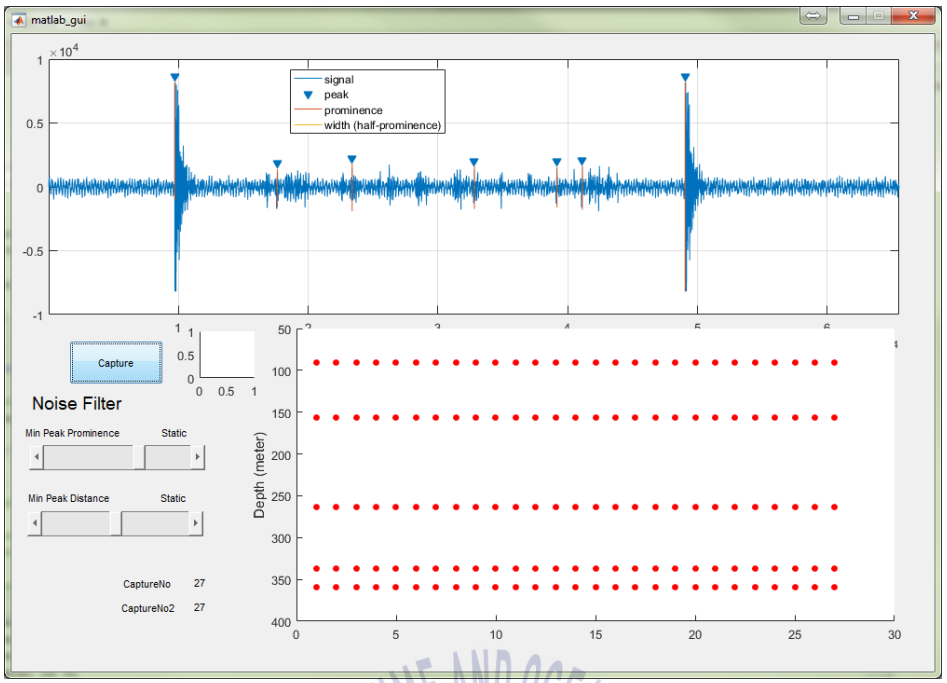


(b)

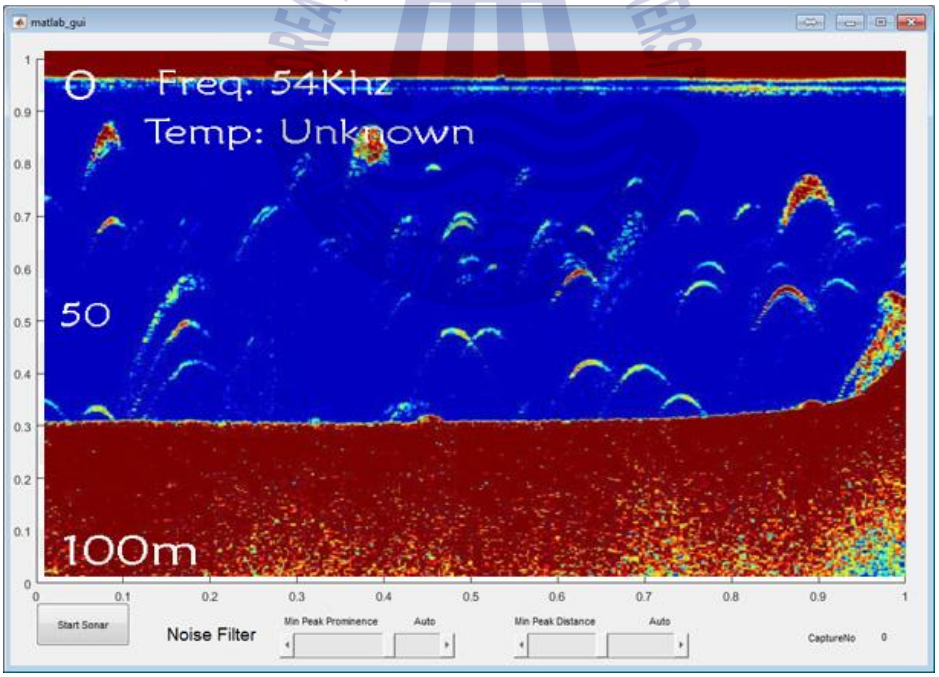
Figure 27 Fish Finder GUI with Garmin Fish Finder Data

The below figure shows a GUI similar to fish finder products. The developed Fish Finder GUI shows the depth of the sea and the frequency of the signal. The intensity of echo is shown by different colors. For example, for high intensity echo, a red color is shown, while a low intensity echo has a color close to blue.

Important Note: The data shown in the figure below is from a manufactured fish finder. We have used this data to verify that our GUI code works as expected.



(a)



(b)

Figure 27 (a): Fish Finder GUI, Transmitted pulse and received echo. (b): converting ADC values into image maps

Process of calculating sea floor and fish position

The Fish Finder GUI code calculates the fish position and the floor of the sea. Here is the process:

1. The signal data is received from Control GUI Software, which was explained in previous section.
2. The noise is filtered by using Min Peak Prominence and Min Peak Distance variables. These variables could be adjusted in GUI, as shown in Figure 27.
3. Using signal processing, the local maximas of the echo of the signal is calculated. Each local maxima indicates an object (fish).
4. The time it takes for the sonar wave to hit the object and get back to the transducer is calculated using: time = diffBetweenTXandRXPosition/samplingTime; % sec
5. Then the distance is calculated using: distance = SoundVelocity * time / 2; % meter
6. This process is repeated for each object (fish).

A snippet of the code behind Fish Finder GUI is shown in Appendix E.

Chapter 5. Experiments

The functionality of the proposed sonar system was verified by carrying out several experimental tests. The results of these experiments are shown and explained in this chapter.

5.1 Transmit Sub-Circuit Test

The purpose of this test is to verify that the proposed transmit sub-circuit works as expected. First, two PWM signals are generated by OMAP-L138. Then, the PWM signals are converted to HV pulses by the HV Pulse Generator. Finally, The HV pulse is transmitted into water and the echo is received by the transducer. The transmitted pulse and received echo were verified on an oscilloscope.

5.2 Receive Sub-Circuit Test

The purpose of this test is to verify that the receive path sub-circuit works as expected. Four different signal shapes, namely, sine, pulse package, ramp, and toggle shapes are generated using a signal generator. These shapes are given as input to the receive path sub-circuit as shown in Figure 28.

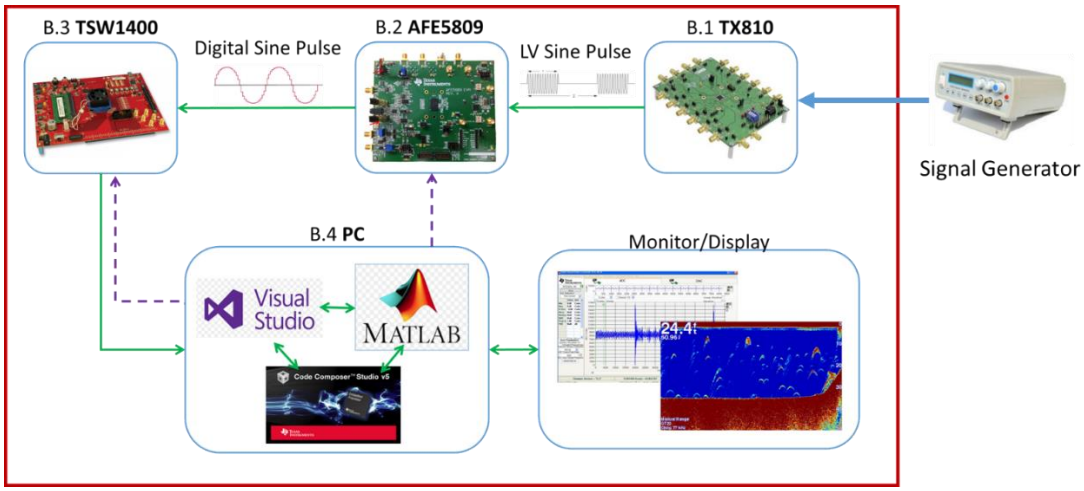


Figure 28 Receive path sub-circuit test

These signals are captured by AFE5809 and then converted into digital signals. Then, the digital signals are temporarily stored in memory of TSW1400EVM. The digital signals are then transferred to a PC and are displayed using HSDC Pro software. The results are shown below.

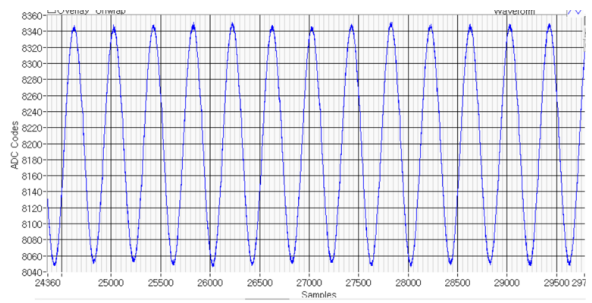


Figure 29 Sine Shape

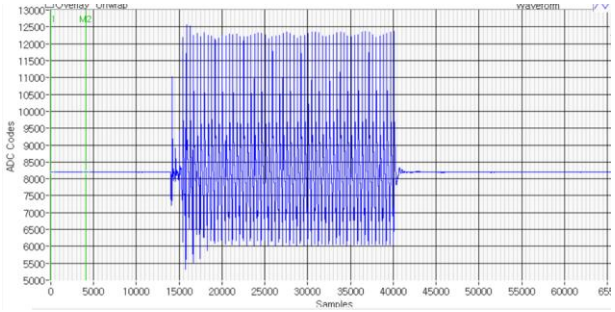


Figure 30 Pulse Package Shape

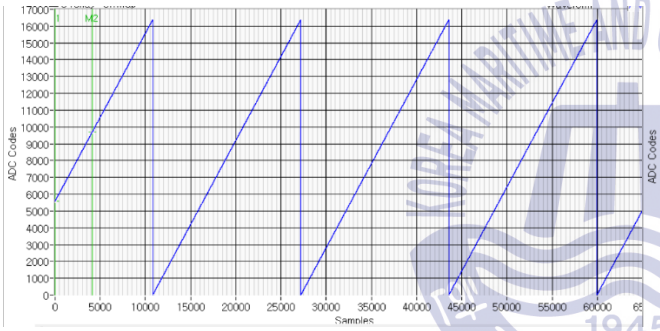


Figure 31 Ramp Shape

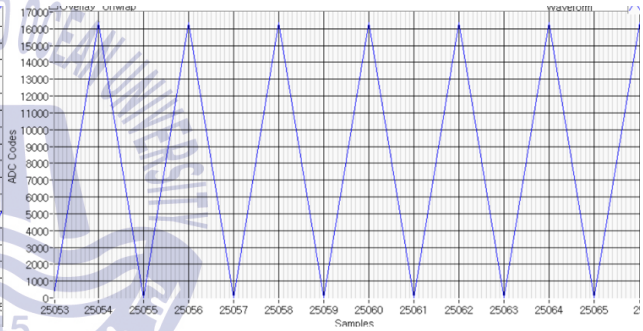


Figure 32 Toggle Shape

Figure 29 to Figure 32 display the received shapes generated by the signal generator. As seen above, the receive path sub-circuit is working properly.

5.3 1-meter test

The proposed sonar system was tested using a 1-meter test. First, two ePWM signals are generated using the OMAP L-138. The output result of OMAP L138 is shown below:

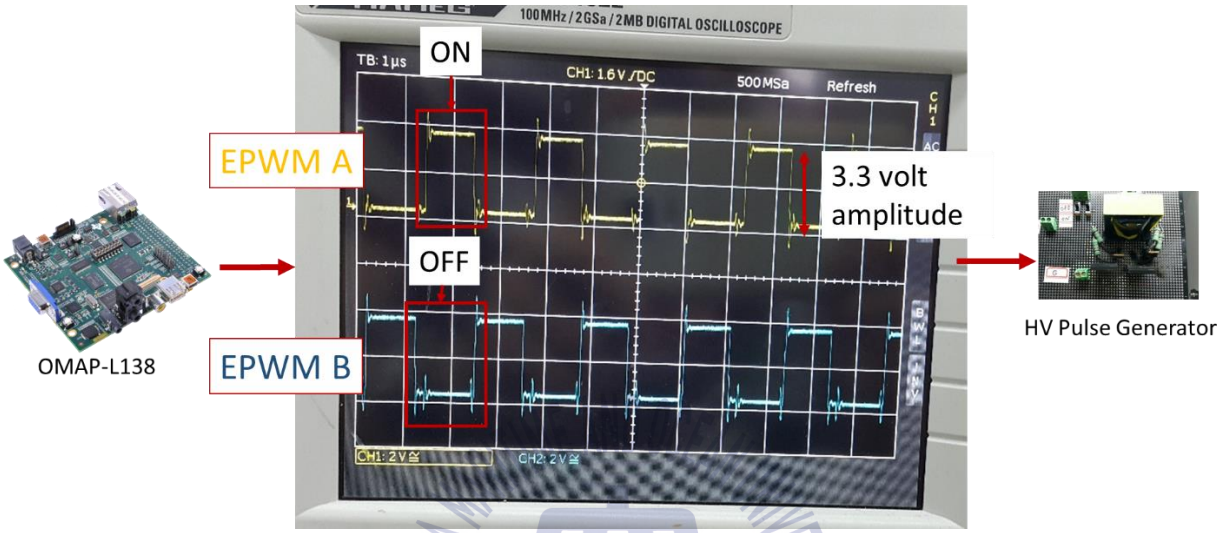


Figure 33 two ePWM signals generated by OMAP L138 with 95Khz frequency and 3.3 volt amplitude

Second, the two ePWM signals shown above are given to the custom HV pulse generator as input. The HV pulse generator will first increase the voltage of the signal and then transforms the square shape into sine shape. Below two figures show the HV pulse generator after amplification of the ePWM signals and after transforming the square shape into sine shape, respectively.

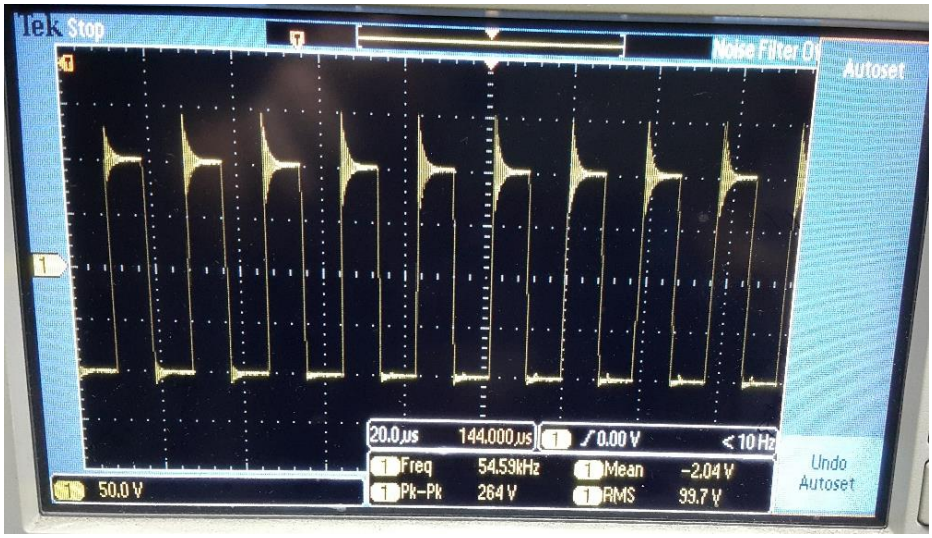


Figure 34 HV Pulse Generator amplifies the low voltage ePWM inputs

Figure 35 shows the final output of the HV pulse generator while the pulse is zoomed in for visual clarification.

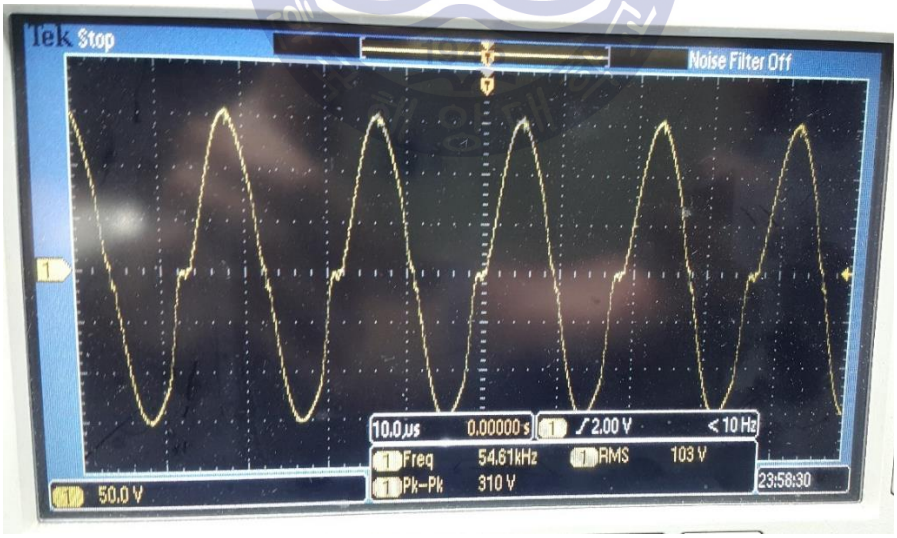


Figure 35 HV Pulse Generator output after (Pulse Zoomed-In) after transforming the square shape into sine shape

As seen above, the sine wave is not perfect but good enough to vibrate the transducer. Figure 36 shows three pulse packages. Each pulse package includes several pulses. Depending on the depth of water, timing of pulse generation and silence time is changed. For example, for deeper sea, longer pulse generation and longer silence time is used, when compared to shallow waters. Figure 36 shows the pulse package. Each pulse package contains several pulses. The number of pulses in a pulse package are chosen based on the depth of the ocean. While in shallow waters, less pulses are generated, as a result, less energy is dissipated into water. On the other hand, when in deep ocean (for example 500 meters), each pulse package contains more pulses, so, more energy is dissipated into the ocean.

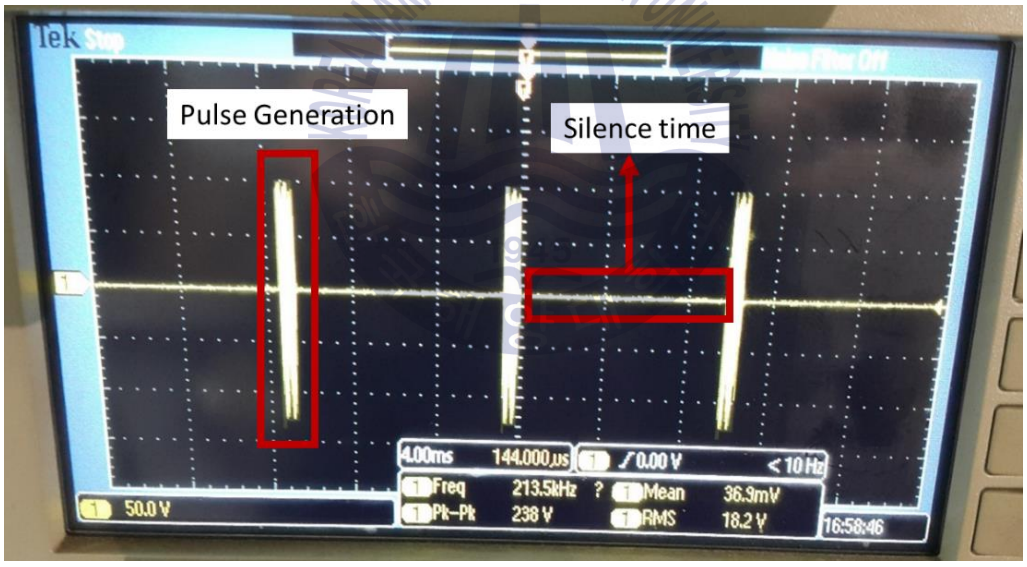
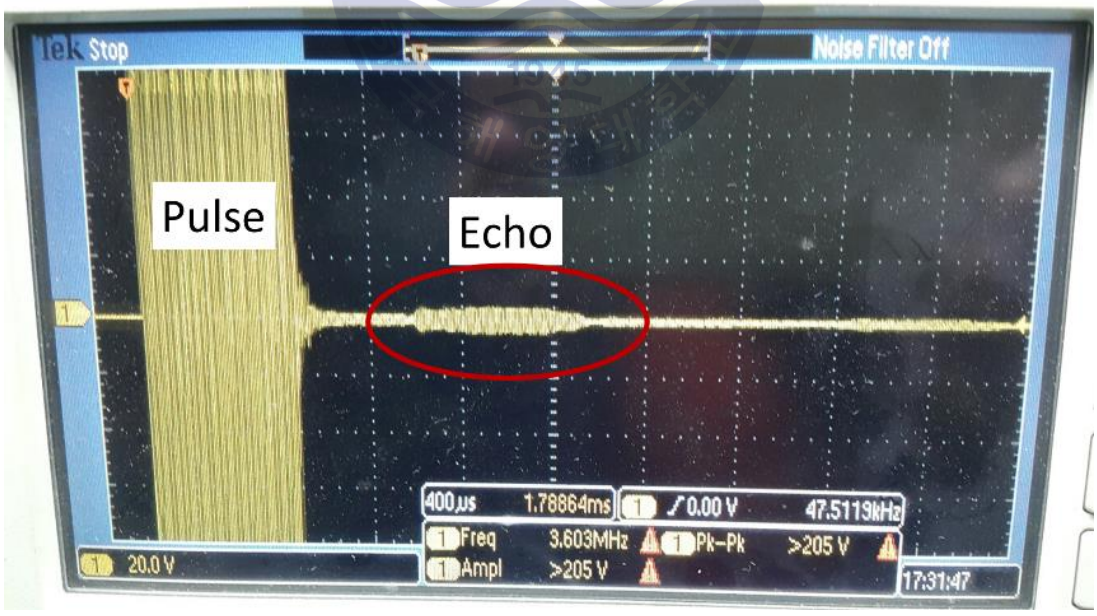


Figure 36 Final HV pulse generator output (pulse package)

Third, the HV sine pulse is transmitted into the water and the echo is captured by the transducer. Figure 37 show the transducer positioning and the received echo response.

(a)



(b)

Figure 37 1-meter test: (a) transducer position (b) echo response

In order to convert this analog signal, the high voltage pulse and echo, as shown above, should be given to the AFE5809 as input. Later, AFE5809 can convert this analog into digital signal. However, the pulse has very high voltage (about 200 Volt peak-to-peak). As a result, if this high voltage is given to the AFE5809 directly, it may damage the board permanently. To avoid such damage, the high voltage signal is first given to the T/R Switch (TX810) as input. The T/R Switch reduces the high voltage into low voltage. Then this low voltage analog signal is given to AFE5809 as input. The AFE5809 converts the analog signal into digital signal. The configuration of AFE5809 GUI used for this experiment is shown below:

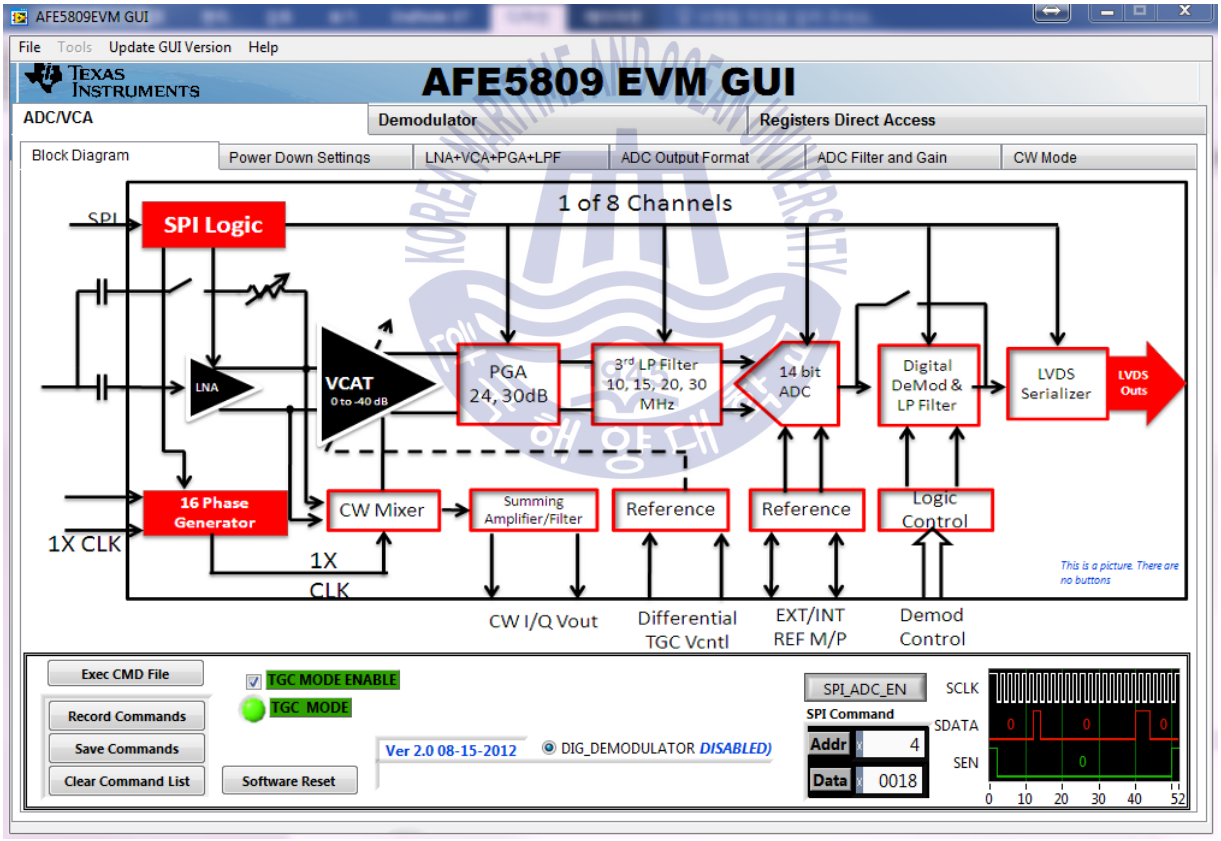


Figure 38 AFE5809 GUI control setting for converting analog to digital signal

After converting the analog into digital signal. The digital data is transferred to a PC using TSW1400 and the results are shown by HSDC Pro software. Here is a photo of the results.

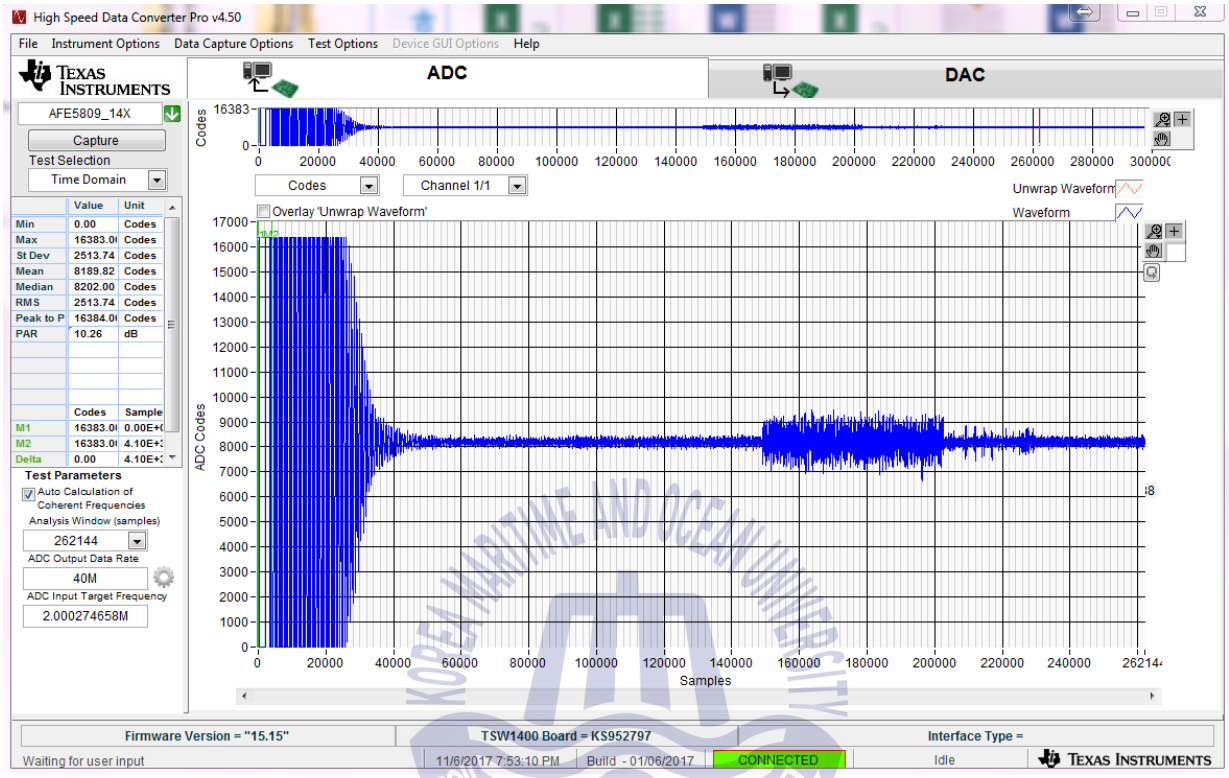


Figure 39 Received echo shown in PC by HSDC Pro software

A custom GUI, namely, Control GUI is developed to automatize the process of capturing and displaying the results in Fish Finder GUI. Control GUI controls HSDC Pro software. Whenever a new echo is captured, Control GUI commands the HSDC Pro to capture the echo and save it as a .CSV file on the hard disk. This process is done in two steps. In Step 1, Control GUI commands HSDC Pro to connect to the AFE5809 board. This step is shown below:

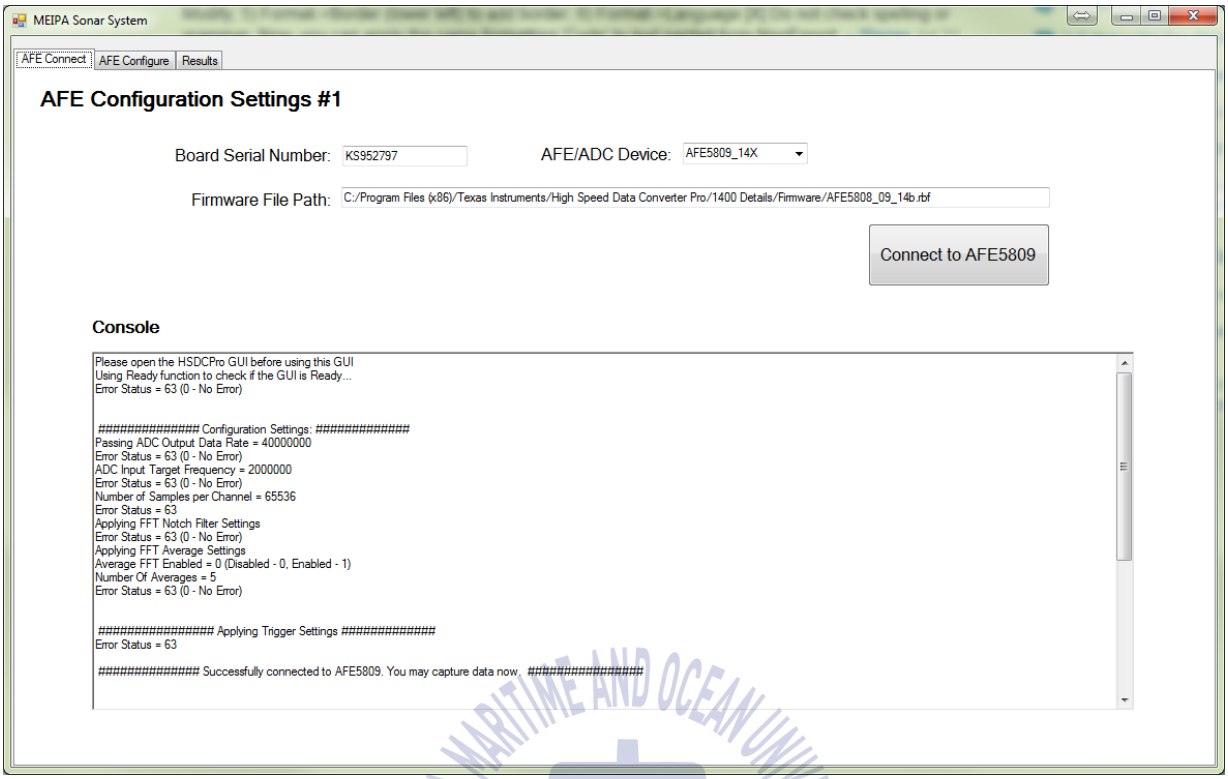


Figure 40 Control GUI AFE Connect Tab: in here the GUI commands HSDC Pro to connect to the AFE5809 board.

In Step 2, Control GUI commands HSDC Pro to capture the digital signal data from AFE5809 and save it as a .CSV file. The GUI provides the capability to control several variables. Variables such as FFT type, number of harmonics, path to save .CSV data file, trigger settings, and etc. This step is shown below:

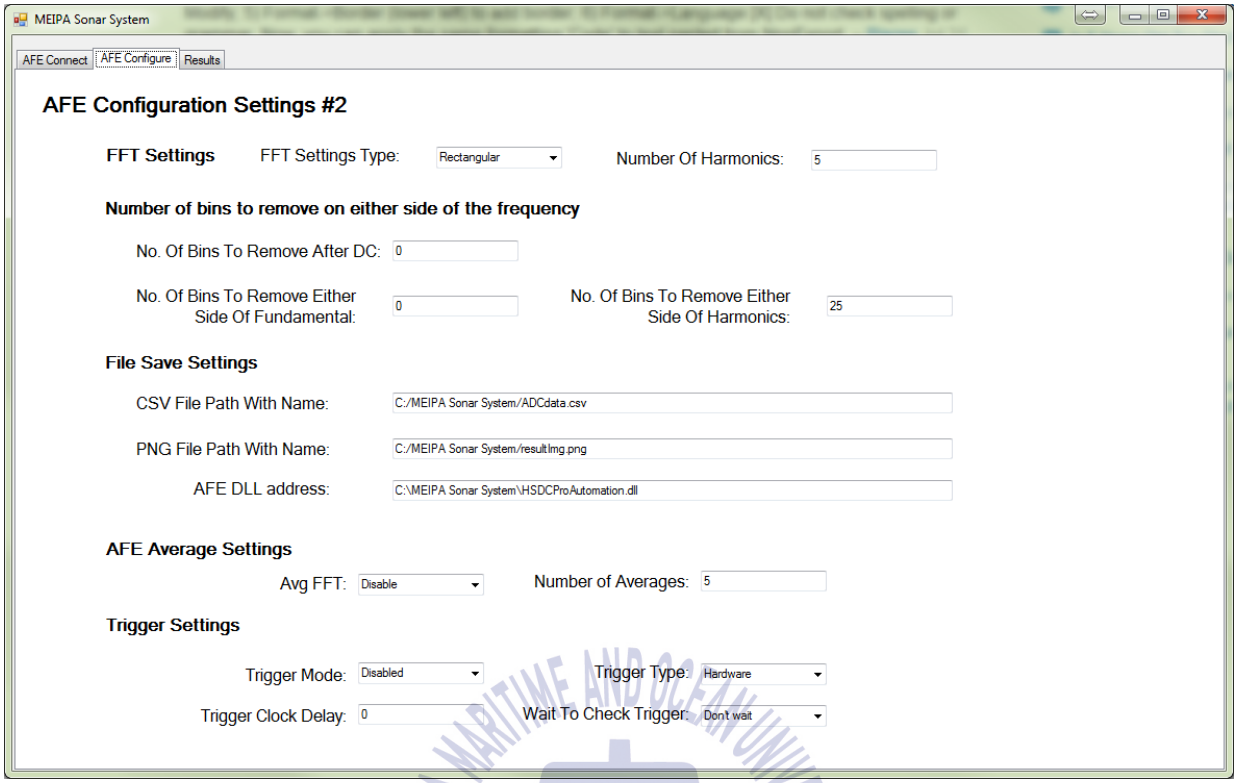


Figure 41 AFE Configure tab of Control GUI Software

After the two steps explained above, the results are saved in a .CSV file. As an example, some parts of a .CSV file (1 channel) are shown in below:

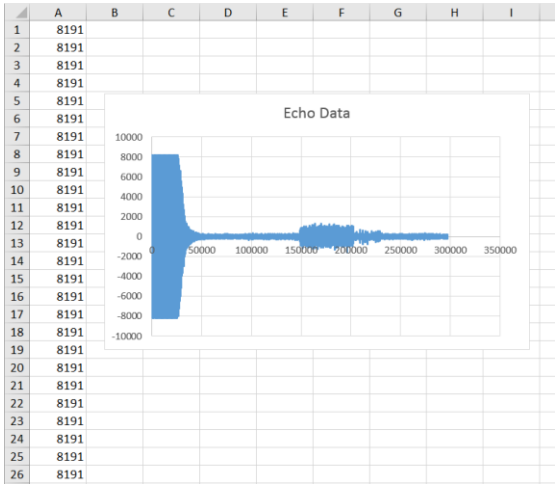


Figure 42 Parts of saved .CSV data (digital echo results)

The Control GUI notifies the Fish Finder GUI about the received signal. So, Fish Finder GUI reads the .CSV file and illustrates the data in the Fish Finder GUI. The results are shown in below.

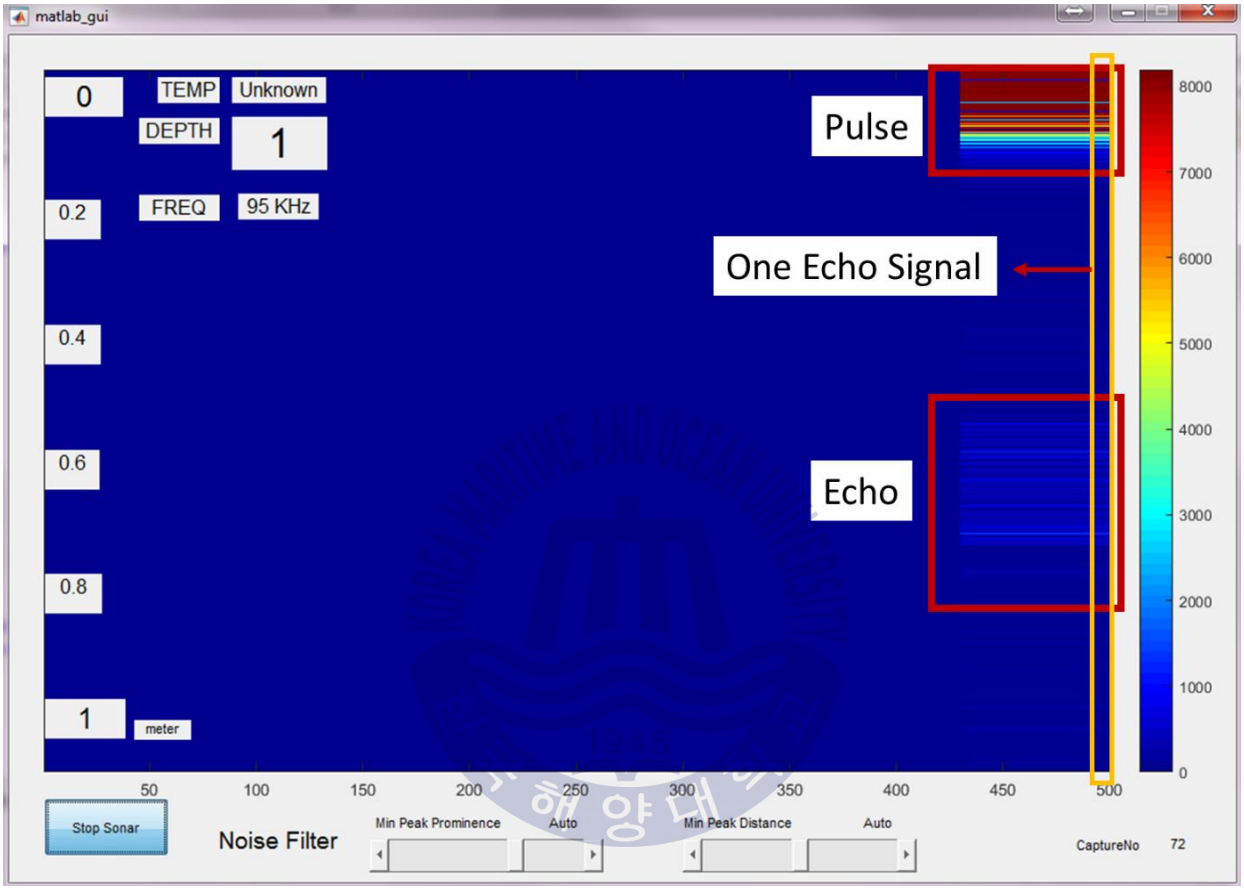


Figure 43 Fish Finder GUI: Showing signal results in colorjet

As shown in figure above, the pulse and echo are shown in color chart. Blue indicates weak signal and red indicates strong signal. The pulse has a strong signal, so it looks completely red while the echo is in light blue.

5.4 10-meter shore test

The prototype was tested beside the shore with a depth ranging from 1 to 10 meters. The below figures show a sample of transmitted pulse package and it's received echo. As shown below, the

echo is very weak. The reason is because the transducer requires 300W of energy to work properly, however, our pulse generator, currently, can generate about 40W of energy (100 RMS volts with 0.4 ampere).

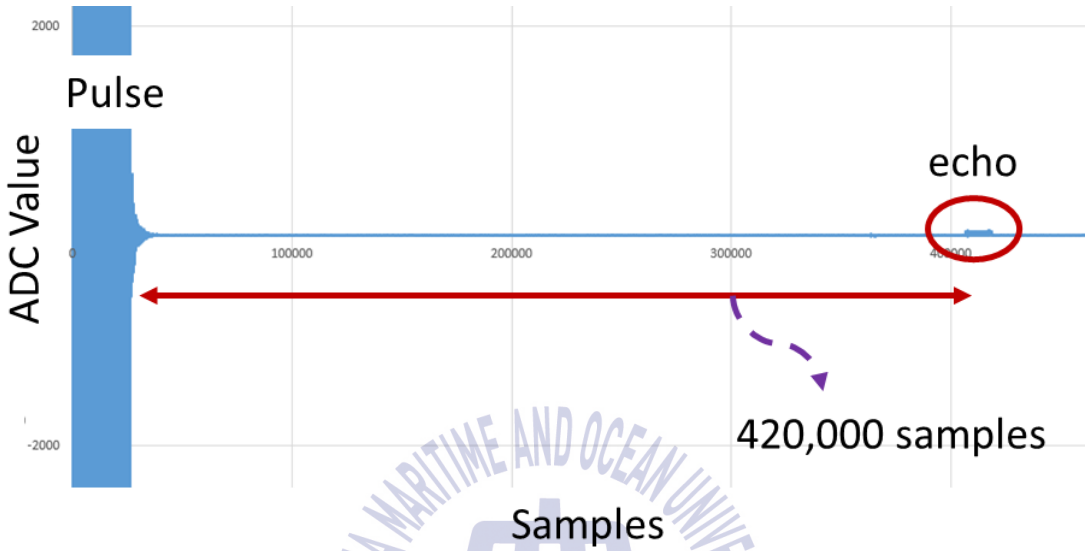


Figure 44 10-meter shore test echo signal

First, the time (T) traveled by the pulse should be calculated from the samples. The ADC sample rate is 65 MSPS (Mega Sample Per Second). Using this sampling rate, for the data shown in the Figure 44, the time is calculated by:

$$T = \frac{420{,}000 \times 1}{65 \times 10^6} = 6.46\ mSec$$

Now, using the calculated time, the distance is calculated by:

$$\mathrm{D} = \frac{\mathrm{V} \times \mathrm{T}}{2} = \frac{1500 \times 6.46 \times 10^{-3}}{2} = 4.85\ meter$$

where D is distance to object, V velocity of wave in water and T time traveled to reach the object and come back to the transducer.

The calculated distance (4.85 meters) is pretty close to the actual distance (5 meters). This indicates the correct functionality of the prototype.

As show in Figure 44, the echo of this test was weak but still readable. We tested for longer ranges including 10 and 50 meters, however, the signal was very weak and not readable anymore. The reason for this weak echo was because we need much more power to transmit the pulse signal into water. Unfortunately, currently we can produce only about 40W of energy but in order to work optimally, the transducer needs 300W of energy.

Chapter 6. Conclusion

In this thesis, the process to develop a sonar system (fish finder) was explained. The process was divided into two major parts, namely, sonar hardware and sonar software. A prototype of a sonar system, with two sub path circuit, was developed. This prototype was divided into two sub-circuits, namely, Transmit Path Sub-Circuit and Receive Path Sub-Circuit. The necessary software to control and obtain the results of the prototype was also developed. The prototype was tested twice, first, in a 1-meter test, and second, in a 10-meter shore test. The hardware worked as expected and the software captured the data successfully. The data was also shown in a custom GUI similar to a manufacturer fish finder.

Future direction

- **Increase the power output of pulse generator**: The HV pulse generator currently can produce about 40W of energy. This is no sufficient to fully activate the transducer. It is worth noting that the transducer vibrated with 40W of energy. However, this energy is not enough for depth over 3 meters. The new pulse generator should output about 300W energy to fully activate the transducer
- **FPGA coding instead of PC:** Currently, the ADC data is transferred to PC via a USB connection. Then, signal processing is done. USB connection is very slow, which makes the whole process of capture and signal processing slow. Alternatively, this signal processing should be done by the FPGA of the TSW1400. This is the optimal solution
- **Test in deep water:** After increasing the pulse power, the prototype should be tested in deep water (over 10 meters). Also, using the test results, the codes should be optimized to calculate the depth properly.

References

[1] Instruments, T. *SONAR Receiver Path Sub-System Reference Design Using the AFE5809*. Available from: http://www.ti.com/lit/ug/tidu702/tidu702.pdf. 2015 Oct. 2017

[2] Li, X., et al., *80-MHz intravascular ultrasound transducer using PMN-PT free-standing film.* IEEE transactions on ultrasonics, ferroelectrics, and frequency control. **58**(11): p. 2281-2288. 2011

[3] Brown, C.J. and P. Blondel, *Developments in the application of multibeam sonar backscatter for seafloor habitat mapping.* Applied Acoustics. **70**(10): p. 1242-1247. 2009

[4] Trucco, A., M. Palmese, and S. Repetto, *Devising an affordable sonar system for underwater 3-D vision.* IEEE Transactions on Instrumentation and Measurement. **57**(10): p. 2348-2354. 2008

[5] Akyildiz, I.F., D. Pompili, and T. Melodia, *Underwater acoustic sensor networks: research challenges.* Ad hoc networks. **3**(3): p. 257-279. 2005

[6] Gupta, S.D., et al., *Design and Implementation of Water Depth Measurement and Object Detection Model Using Ultrasonic Signal System.* International Journal of Engineering Research and Development. **4**(3): p. 62-69. 2012

[7] Fornshell, J.A. and A. Tesei, *The development of SONAR as a tool in marine biological research in the twentieth century.* International Journal of Oceanography. **2013**. 2013

[8] Hansen, R.E., *Introduction to sonar*. Course Material to INF-GEO4310, University of Oslo,(Oct. 7, 2009). 2009

[9] Wada, M., et al. *The development of a remote fish finder system for set-net fishery*. in *Oceans-St. John's, 2014*. 2014. IEEE.

[10] Manik, H.M., *Seabed identification and characterization using sonar.* Advances in Acoustics and Vibration. **2012**. 2012

[11] Stergiopoulos, S., *Advanced signal processing handbook: theory and implementation for radar, sonar, and medical imaging real time systems*. CRC press. 2000

[12] Sherfey III, S.R., *A mobile robot sonar system*, NAVAL POSTGRADUATE SCHOOL MONTEREY CA. 1991

[13] Lochner, J.T., *Analysis and Improvement of an Ultrasonic Sonar System on an Autonomous Mobile Robot*, NAVAL POSTGRADUATE SCHOOL MONTEREY CA. 1994

[14] Kelly, I. and D. Keating. *Flocking by the fusion of sonar and active infrared sensors on physical autonomous mobile robots*. in *Proceedings of The Third Int. Conf. on Mechatronics and Machine Vision in Practice*. 1996.

[15] Burckhardt, C., P.-A. Grandchamp, and H. Hoffmann, *An experimental 2 MHz synthetic aperture sonar system intended for medical use.* IEEE Transactions on Sonics and Ultrasonics. **21**(1): p. 1-6. 1974

[16] Barshan, B. and R. Kuc, *A bat-like sonar system for obstacle localization.* IEEE Transactions on systems, man, and cybernetics. **22**(4): p. 636-646. 1992

[17] Van Veen, B.D. and K.M. Buckley, *Beamforming: A versatile approach to spatial filtering.* IEEE assp magazine. **5**(2): p. 4-24. 1988

[18] Wong, G.S. and S.m. Zhu, *Speed of sound in seawater as a function of salinity, temperature, and pressure*. the Journal of the Acoustical Society of America. **97**(3): p. 1732-1736. 1995

[19] wikipedia. *Speed of sound*. Available from: https://en.wikipedia.org/wiki/Speed_of_sound#Seawater. 2017 Nov. 2017

[20] vexilar. *How SONAR Works*. Available from: http://www.vexilar.com/blog/2014/08/28/how-sonar-works. 2017 Nov. 2017
[21] Garmin. *Airmar TM150M*. Available from: https://buy.garmin.com/en-US/US/p/120791. 2017
[22] Instruments, T. *TX810: 8-Channel, Programmable T/R Switch for Ultrasound*. Available from: http://www.ti.com/lit/ds/symlink/tx810.pdf. 2010 Oct. 2017
[23] Instruments, T. *AFE5809 Fully Integrated, 8-Channel Ultrasound Analog Front End With Passive CW*

Mixer, and Digital I/Q Demodulator, 0.75 nV/rtHz, 14, 12-Bit, 65 MSPS, 158 mW/CH. Available from: http://www.ti.com/lit/ds/symlink/afe5809.pdf. 2015 Oct. 2017
[24] Instruments, T. *TSW140x High Speed Data Capture/Pattern Generator*

Card. Available from: http://www.ti.com/lit/ug/slwu079d/slwu079d.pdf. 2016 Oct. 2017
[25] Instruments, T. *OMAP-L138 C6000™ DSP+ ARM® Processor*. Available from: http://www.ti.com/lit/ds/symlink/omap-l138.pdf. 2017 Oct. 2017

Appendix A. Control GUI

Note: Part of the code below is taken from Texas Instrument's High Speed Data Converter Pro software which is found in

```
private: System::Void button1_Click_2(System::Object^  sender, System::EventArgs^  e) {
                button1->Text = "Connecting";

                richTextBox1->Text = "";
                richTextBox1->Text += "Connecting to AFE5809 Started. Please wait... \r\n";


                // ################### Connect code ########################

//*****************************************************************************//

//************************** ADC Configuration Settings **************************//

//*****************************************************************************//
                IntPtr ptrToStrBoardSerialNumber = Marshal::StringToHGlobalAnsi(textBox1->Text);
                char* charBoardSerialNumber = static_cast<char*>(ptrToStrBoardSerialNumber.ToPointer());
                strcpy(BoardSerialNumber, charBoardSerialNumber); //Serial Number of board to be connected. Eg: "TIVHIV9Z" or "TIVHIV9Z-TSW1400"
                //strcpy(BoardSerialNumber, "KS952797"); //Serial Number of board to be connected. Eg: "TIVHIV9Z" or "TIVHIV9Z-TSW1400"

                IntPtr ptrToStrFirmwareFilePath = Marshal::StringToHGlobalAnsi(textBox13->Text);
                char* charFirmwareFilePath = static_cast<char*>(ptrToStrFirmwareFilePath.ToPointer());
                strcpy(FirmwareFilePath, charFirmwareFilePath); //Firmware file path which needs to be loaded to the board.
                WaitToCheck = 1; //Wait to check if firmware is downloaded properly? 0 - No, 1 - Yes. If yes, timeout should be greater than 60sec.

                IntPtr ptrToStrADCDevice;


//richTextBox1->Text += "Sajad  " + comboBox4->SelectedIndex + " dsfgsd\r\n";
                if (comboBox4->SelectedIndex == 0) {
                        ptrToStrADCDevice = Marshal::StringToHGlobalAnsi("AFE5809_12X");
                }
                else if (comboBox4->SelectedIndex == 1) {
                        ptrToStrADCDevice = Marshal::StringToHGlobalAnsi("AFE5809_14X");
                }
                else if (comboBox4->SelectedIndex == 2) {
                        ptrToStrADCDevice = Marshal::StringToHGlobalAnsi("AFE5809_16X");
                }

                // sample number

                char* charADCDevice = static_cast<char*>(ptrToStrADCDevice.ToPointer());
                strcpy(ADCDevice, charADCDevice); //ADC device to be selected(should be same as what appears in the HSDC Pro GUI selection drop down box.

                // ################ get GUI value and put inside variables ################
                ADCOutputDataRate = Double::Parse(textBox2->Text); //ADC output Data Rate
```

```
ADCInputTargetFrequency = Double::Parse(textBox3->Text); //ADC Input Target Frequency
TimeoutInMs = Double::Parse(textBox4->Text); //TimeoutInMs for each function

//Number of Samples per Channel
if (comboBox11->SelectedIndex == 0) {
	NumberOfSamplesPerChannel = 500001;
}
else if (comboBox11->SelectedIndex == 1) {
	NumberOfSamplesPerChannel = 1000001;
}
else if (comboBox11->SelectedIndex == 2) {
	NumberOfSamplesPerChannel = 2000001;
}
else if (comboBox11->SelectedIndex == 3) {
	NumberOfSamplesPerChannel = 4000001;
}
else if (comboBox11->SelectedIndex == 4) {
	NumberOfSamplesPerChannel = 8000001;
}
else if (comboBox11->SelectedIndex == 5) {
	NumberOfSamplesPerChannel = 16000001;
}

ChannelIndex = 0; //Interested channel's index(0-based)

//ADC Average Settings
AverageFFTOn = comboBox6->SelectedIndex; //0-Disable Avg FFT; 1-Enable Avg FFT;

NumberOfAverages = Convert::ToInt32(textBox5->Text); //Number of captures for which Avg FFT needs to be calculated

//Trigger Settings
TriggerModeEnable = comboBox7->SelectedIndex; //Enable Trigger - 1; Disable Trigger - 0
SoftwareTriggerEnable = comboBox8->SelectedIndex; //Hardware Trigger - 0; Software Trigger - 1; Arm On Next Capture Button Press - 0
ArmOnNextCaptureButtonPress = 0; //Hardware Trigger - 0; Software Trigger - 0; Arm On Next Capture Button Press - 1
//Arm On Next Capture Button Press is similar to Software Trigger Mode. After user presses the capture button, the trigger is armed.
TriggerCLKDelays = Convert::ToInt32(textBox6->Text); //The number of clock delays for the trigger
WaitToCheckTrigger = comboBox9->SelectedIndex; //0 - Don't Wait, 1 - Wait and check whether trigger has occurred.

//FFT Settings
FFTSettingsType = comboBox10->SelectedIndex; //0 - Rectangular ; 1 - Other Windows
NumberOfHarmonics = Convert::ToInt32(textBox7->Text); // Number of harmonics to be considered
//Number of bins to remove on either side of the frequency
NoOfBinsToRemoveAfterDC = Convert::ToInt32(textBox8->Text);
NoOfBinsToRemoveEitherSideOfFundamental = Convert::ToInt32(textBox9->Text);
NoOfBinsToRemoveEitherSideOfHarmonics = Convert::ToInt32(textBox10->Text);
//Custom frequency and the corresponding number of bins to remove for that frequency
// WARNING: ADD TO GUI LATER
CustomNotchFrequencies[0] = 5000000;
NoOfBinsToRemoveOnEitherSideOfCustomFrequencies[0] = 25;
CustomNotchFrequencies[1] = 20000000;
NoOfBinsToRemoveOnEitherSideOfCustomFrequencies[1] = 20;
```

```
                NumberOfCustomFreq = 1; //Number of custom frequencies
                                                        //For Real FFT, FFT Array length is half the
number of samples
                                                        //For Complex FFT, FFT Array length is
equal to the number of samples
                FFTArrayLength = 32768;

                //Enable/Disable the automatic notching of (Fs/2 - Fin) frequency, when Fs/2 or Fin is changed.
                // WARNING: ADD TO GUI LATER
                enableFsby2MinusFinNotching = 1; //0 - Disable, 1 - Enable
                binsToRemoveOnEitherSideOfFsby2 = 10; // Number of Bins to remove on either side of (Fs/2 -
Fin) frequency

                //File Save Settings. Please provide file path with the respective extension
                IntPtr ptrToStrCSVFilePathWithName = Marshal::StringToHGlobalAnsi(textBox11->Text);
                char*                                   charCSVFilePathWithName                         =
static_cast<char*>(ptrToStrCSVFilePathWithName.ToPointer());
                strcpy(CSVFilePathWithName, charCSVFilePathWithName);
                IntPtr ptrToStrPNGFilePathWithName = Marshal::StringToHGlobalAnsi(textBox12->Text);
                char*                                   charPNGFilePathWithName                         =
static_cast<char*>(ptrToStrPNGFilePathWithName.ToPointer());
                strcpy(PNGFilePathWithName, charPNGFilePathWithName);

                //Selection Settings
                TestSelection = comboBox1->SelectedIndex; //Time Domain-0; Single Tone-1; Two Tone-2;
Channel Power-3
                PlotType = comboBox2->SelectedIndex; //Codes - 0; Bits - 1; Real FFT - 2; Complex FFT - 3
                FFTWindowType = comboBox3->SelectedIndex;; //Rectangular - 0; Hamming - 1; Hanning - 2;
Blackman - 3

                //Configuring Bandwidth Integration Markers
                // WARNING: ADD TO GUI LATER
                EnableBIM = 0; //0-Disable BIM Markers; 1-Enable BIM Markers;
//Frequency for Bandwidth Integration Markers, within which the Single Tone Parameter values need to be calculated
                BIM0 = 0;
                BIM1 = 20e6;

                //Getting the Single Tone Parameters
                //The parameters whose values are needed must be sent separated by ";" as a string(char array)
                strcpy(ParametersIn,                    "SNR;SFDR;THD;SINAD;ENOB;Fund.;Next
Spur;HD2;HD3;HD4;HD5;NSD");
                ParameterValueLength = 12; //Size of "ParameterValues" array. Should be at least equal to the no.
of parameters requested
                dBFs = 1; //dBc - 0, dBFs - 1. Required Unit for Parameters - SNR, THD, SINAD, and Next Spur

                //Setting and Getting the Channel Power Parameters
                ADCCenterFrequency = ADCInputTargetFrequency;
                ChannelPwrNumberOfChannels = 2;
                SignalWidth = 2e6;
                ChannelSeparation = 4e6;
                //The parameters whose values are needed must be sent separated by ";" as a string(char array)
                strcpy(ChannelPwrParametersIn, "C1 PWR;C2 PWR;C3 PWR;C4 PWR;C5 PWR");
                ChannelPwrNumberOfParameters = ChannelPwrNumberOfChannels; //Size of "ParameterValues"
array. Should be at least equal to the no. of parameters requested


                                        //********************* Loading the HSDCPro Automation DLL
**********************//
```

```
                                        //The below path needs to be changed according to 32-bit/64-bit systems

                                        //For 32-Bit OS, uncomment below line and comment out the line specified for
64-Bit

                                        //HINSTANCE hGetProcIDDLL = LoadLibraryW(L"C:\\Program Files\\Texas
Instruments\\High Speed Data Converter Pro\\HSDCPro Automation DLL\\32Bit DLL\\HSDCProAutomation.dll"); /*
get handle to dll */

                                        //For 64-Bit OS, comment out above line and uncomment below line
                    IntPtr ptrToStrhGetProcIDDLL = Marshal::StringToHGlobalAnsi(textBox14->Text);
                    char* charhGetProcIDDLL = static_cast<char*>(ptrToStrhGetProcIDDLL.ToPointer());
                    HINSTANCE          hGetProcIDDLL          =          LoadLibraryW(L"C:\\MEIPA          Sonar
System\\HSDCProAutomation.dll"); /* get handle to dll */

                    //  int32_t  __cdecl  Pass_ADC_Output_Data_Rate(double  ADCOutputDataRate,  int32_t
TimeoutInMs);
                    FARPROC          lpfnGetProcessID          =          GetProcAddress(HMODULE(hGetProcIDDLL),
"Pass_ADC_Output_Data_Rate");  /* get pointer to the function in the dll*/
                    typedef int(*pICFUNC1)(double, int); /* define the Function in the DLL for reuse.*/
                    pICFUNC1 Pass_ADC_Output_Data_Rate;
                    Pass_ADC_Output_Data_Rate = pICFUNC1(lpfnGetProcessID);

                    //int32_t __cdecl Set_ADC_Input_Target_Frequency(double ADCInputTargetFrequency, int32_t
TimeoutInMs);
                    lpfnGetProcessID                =                GetProcAddress(HMODULE(hGetProcIDDLL),
"Set_ADC_Input_Target_Frequency");
                    typedef int(*pICFUNC1)(double, int);
                    pICFUNC1 Set_ADC_Input_Target_Frequency;
                    Set_ADC_Input_Target_Frequency = pICFUNC1(lpfnGetProcessID);

                    //int32_t  __cdecl  Set_Number_of_Samples(uint64_t  NumberOfSamplesPerChannel,int32_t
TimeoutInMs);
                    lpfnGetProcessID = GetProcAddress(HMODULE(hGetProcIDDLL), "Set_Number_of_Samples");
                    typedef int(*pICFUNC2)(unsigned long long, int);
                    pICFUNC2 Set_Number_of_Samples;
                    Set_Number_of_Samples = pICFUNC2(lpfnGetProcessID);

                    //int32_t  __cdecl  Trigger_Option(int32_t  TriggerModeEnable,int32_t  SoftwareTriggerEnable,
int32_t ArmOnNextCaptureButtonPress,uint8_t TriggerCLKDelays, int32_t TimeoutInMs);
                    lpfnGetProcessID = GetProcAddress(HMODULE(hGetProcIDDLL), "Trigger_Option");
                    typedef int(*pICFUNC3)(int, int, int, unsigned char, int);
                    pICFUNC3 Trigger_Option;
                    Trigger_Option = pICFUNC3(lpfnGetProcessID);

                    //int32_t __cdecl Pass_Capture_Event(int32_t TimeoutInMs);
                    lpfnGetProcessID = GetProcAddress(HMODULE(hGetProcIDDLL), "Pass_Capture_Event");
                    Pass_Capture_Event = pICFUNC4(lpfnGetProcessID);

                    //int32_t __cdecl Generate_Software_Trigger(int32_t WaitToCheckTrigger,int32_t TimeoutInMs);
                    lpfnGetProcessID                =                GetProcAddress(HMODULE(hGetProcIDDLL),
"Generate_Software_Trigger");

                    Generate_Software_Trigger = pICFUNC5(lpfnGetProcessID);

                    //int32_t __cdecl Read_DDR_Memory(int32_t WaitToCheckTrigger,int32_t TimeoutInMs);
                    lpfnGetProcessID = GetProcAddress(HMODULE(hGetProcIDDLL), "Read_DDR_Memory");

                    Read_DDR_Memory = pICFUNC5(lpfnGetProcessID);
```

```
//int32_t __cdecl HSDC_Ready(int32_t TimeoutInMs);
                lpfnGetProcessID = GetProcAddress(HMODULE(hGetProcIDDLL), "HSDC_Ready");

                HSDC_Ready = pICFUNC4(lpfnGetProcessID);

                //int32_t __cdecl Save_Raw_Data_As_CSV(char CSVFilePathWithName[],int32_t TimeoutInMs);
                lpfnGetProcessID = GetProcAddress(HMODULE(hGetProcIDDLL), "Save_Raw_Data_As_CSV");

                Save_Raw_Data_As_CSV = pICFUNC6(lpfnGetProcessID);

                //int32_t __cdecl ADC_Test_Selection(uint16_t TestSelection, int32_t TimeoutInMs);
                lpfnGetProcessID = GetProcAddress(HMODULE(hGetProcIDDLL), "ADC_Test_Selection");

                ADC_Test_Selection = pICFUNC7(lpfnGetProcessID);

                //int32_t __cdecl ADC_Plot_Type(uint16_t PlotType, int32_t TimeoutInMs);
                lpfnGetProcessID = GetProcAddress(HMODULE(hGetProcIDDLL), "ADC_Plot_Type");

                ADC_Plot_Type = pICFUNC7(lpfnGetProcessID);

                //int32_t __cdecl FFT_Window(uint16_t FFTWindowType, int32_t TimeoutInMs);
                lpfnGetProcessID = GetProcAddress(HMODULE(hGetProcIDDLL), "FFT_Window");

                FFT_Window = pICFUNC7(lpfnGetProcessID);

                //int32_t __cdecl Select_ADC_Channel(uint16_t ChannelIndex, int32_t TimeoutInMs);
                lpfnGetProcessID = GetProcAddress(HMODULE(hGetProcIDDLL), "Select_ADC_Channel");

                Select_ADC_Channel = pICFUNC7(lpfnGetProcessID);

                //int32_t __cdecl FFT_Window_Notching(uint16_t FFTSettingsType, uint32_t
NumberOfHarmonics, uint32_t NoOfBinsToRemoveEitherSideOfHarmonics,
                //uint32_t NoOfBinsToRemoveAfterDC, uint32_t NoOfBinsToRemoveEitherSideOfFundamental,
double CustomNotchFrequencies[],
                //uint32_t NoOfBinsToRemoveOnEitherSideOfCustomFrequencies[], uint32_t
NumberOfCustomFreq,int32_t enableFsby2MinusFinNotching, int32_t binsToRemoveOnEitherSideOfFsby2,int32_t
TimeoutInMs);
                lpfnGetProcessID = GetProcAddress(HMODULE(hGetProcIDDLL), "FFT_Window_Notching");

                FFT_Window_Notching = pICFUNC8(lpfnGetProcessID);

                //int32_t __cdecl Get_FFT_Data(int32_t TimeoutInMs, double *f0, double *df,double
ActiveChannelFFT[], int32_t *FFTArrayLength);
                lpfnGetProcessID = GetProcAddress(HMODULE(hGetProcIDDLL), "Get_FFT_Data");

                Get_FFT_Data = pICFUNC9(lpfnGetProcessID);

                //int32_t __cdecl Save_FFT_As_PNG(uint16_t ChannelIndex, char PNGFilePathWithName[],
int32_t TimeoutInMs);
                lpfnGetProcessID = GetProcAddress(HMODULE(hGetProcIDDLL), "Save_FFT_As_PNG");

                Save_FFT_As_PNG = pICFUNC10(lpfnGetProcessID);

                //int32_t __cdecl Single_Tone_Parameters(char ParametersIn[], int32_t dBFs, int32_t TimeoutInMs,
double ParameterValues[], int32_t ParameterValuesLength);
                lpfnGetProcessID = GetProcAddress(HMODULE(hGetProcIDDLL), "Single_Tone_Parameters");

                Single_Tone_Parameters = pICFUNC11(lpfnGetProcessID);
```

```
		//int32_t __cdecl Get_Error_Status(int32_t StringLengthIn, int32_t TimeoutInMs, char ErrorString[]);
		lpfnGetProcessID = GetProcAddress(HMODULE(hGetProcIDDLL), "Get_Error_Status");

		Get_Error_Status = pICFUNC12(lpfnGetProcessID);

		//double __cdecl Automation_DLL_Version(void);
		lpfnGetProcessID = GetProcAddress(HMODULE(hGetProcIDDLL), "Automation_DLL_Version");

		Automation_DLL_Version = pICFUNC13(lpfnGetProcessID);

		//int32_t __cdecl Connect_Board(char BoardSerialNumber[], int32_t TimeoutInMs);
		lpfnGetProcessID = GetProcAddress(HMODULE(hGetProcIDDLL), "Connect_Board");

		Connect_Board = pICFUNC6(lpfnGetProcessID);

		//int32_t __cdecl Disconnect_Board(int32_t TimeoutInMs);
		lpfnGetProcessID = GetProcAddress(HMODULE(hGetProcIDDLL), "Disconnect_Board");

		Disconnect_Board = pICFUNC4(lpfnGetProcessID);

		//int32_t __cdecl Download_Firmware(char FirmwareFilePath[],int32_t WaitToCheck, int32_t TimeoutInMs);
		lpfnGetProcessID = GetProcAddress(HMODULE(hGetProcIDDLL), "Download_Firmware");

		Download_Firmware = pICFUNC14(lpfnGetProcessID);

		//int32_t __cdecl Select_ADC_Device(char ADCDevice[], int32_t TimeoutInMs);
		lpfnGetProcessID = GetProcAddress(HMODULE(hGetProcIDDLL), "Select_ADC_Device");

		Select_ADC_Device = pICFUNC6(lpfnGetProcessID);

//int32_t __cdecl ADC_Average_Settings(int32_t AverageFFTOn,int32_t NumberOfAverages, int32_t TimeoutInMs);
		lpfnGetProcessID = GetProcAddress(HMODULE(hGetProcIDDLL), "ADC_Average_Settings");

		ADC_Average_Settings = pICFUNC15(lpfnGetProcessID);

		//int32_t __cdecl Set_ADC_BIM(int32_t EnableBIM, double BIM0, double BIM1,int32_t TimeoutInMs);
		lpfnGetProcessID = GetProcAddress(HMODULE(hGetProcIDDLL), "Set_ADC_BIM");

		Set_ADC_BIM = pICFUNC16(lpfnGetProcessID);

		//int32_t __cdecl ADC_Channel_Power_Settings(double ADCCenterFrequency,int32_t NumberOfChannels, double SignalWidth, double ChannelSeparation,
		//int32_t TimeoutInMs);
		lpfnGetProcessID = GetProcAddress(HMODULE(hGetProcIDDLL), "ADC_Channel_Power_Settings");

		ADC_Channel_Power_Settings = pICFUNC17(lpfnGetProcessID);

		//int32_t __cdecl ADC_Channel_Power_Parameters(char ParametersIn[],int32_t TimeoutInMs, double ParameterValues[], int32_t NumberOfParameters);
		lpfnGetProcessID = GetProcAddress(HMODULE(hGetProcIDDLL), "ADC_Channel_Power_Parameters");

		ADC_Channel_Power_Parameters = pICFUNC18(lpfnGetProcessID);
```

```
        //*************************************************************************************//
                //********** The actual call to the function contained in the dll ***************//

        //*************************************************************************************//
                printf("\nPlease open the HSDCPro GUI before using these Automation DLL functions.");
                richTextBox1->Text += "Please open the HSDCPro GUI before using this GUI \r\n";
                Application::DoEvents();
                printf("\nPress any key to start...");
                Application::DoEvents();
                //getch();

                //Connecting to the board and selecting ADC device
                printf("\n\n Connecting to board : %s", BoardSerialNumber);
                richTextBox1->Text += "#################### Connecting to board : " + textBox1->Text + "
#################### \r\n";
                Application::DoEvents();
                Error_Status = Connect_Board(BoardSerialNumber, TimeoutInMs);
                printf("\nError Status = %d", Error_Status);
                richTextBox1->Text += "Error Status = " + Error_Status + " (0 - No Error)\r\n";
                Application::DoEvents();
                //Selecting the device.It will automatically download its firmware.
                //Please provide sufficient timeout for completing firmware downlaod operation
                printf("\n\nSelecting ADC Device : %s", ADCDevice);
                richTextBox1->Text += "Selecting ADC Device: " + comboBox4->Text + " \r\n";
                richTextBox1->Text += "This may take a few minutes. Please wait...\r\n";
                Application::DoEvents();
                Error_Status = Select_ADC_Device(ADCDevice, 120000); //120sec timeout value to allow
firmware download
                printf("\nError Status = %d", Error_Status);
                richTextBox1->Text += "Error Status = " + Error_Status + " (0 - No Error)\r\n";
                Application::DoEvents();
                //Use HSDC Ready function to check if the GUI has completed all its operations(like downloading
firmware)...
                printf("\n\nUsing HSDC Ready function to check if the GUI is Ready...");
                richTextBox1->Text += "Using Ready function to check if the GUI is Ready... \r\n";
                Application::DoEvents();
                Error_Status = HSDC_Ready(60000); //Waiting to check if HSDCPro has completed all its
operations.
                printf("\nError Status = %d", Error_Status);
                richTextBox1->Text += "Error Status = " + Error_Status + " (0 - No Error)\r\n";
                Application::DoEvents();
                //Download Firmware(OPTIONAL). Selecting device itself(above function) will automatically
download the firmware
                //printf("\n\nDownloading Firmware : %s",FirmwareFilePath);
                //Error_Status = Download_Firmware(FirmwareFilePath,WaitToCheck,60000);
                //printf("\nError Status = %d",Error_Status);
//Configuration Settings
                printf("\n\nConfiguration Settings:");
                richTextBox1->Text += "\r\n \r\n ############## Configuration Settings: ############# \r\n";
                Application::DoEvents();
                printf("\n\nPassing ADC Output Data Rate = %g", ADCOutputDataRate);
                richTextBox1->Text += "Passing ADC Output Data Rate = " + ADCOutputDataRate + "\r\n";
                Application::DoEvents();
                Error_Status = Pass_ADC_Output_Data_Rate(ADCOutputDataRate, TimeoutInMs);
                printf("\nError Status = %d", Error_Status);
                richTextBox1->Text += "Error Status = " + Error_Status + " (0 - No Error)\r\n";
```

```
                Application::DoEvents();

                printf("\n\nADC Input Target Frequency = %g", ADCInputTargetFrequency);
                richTextBox1->Text += "ADC Input Target Frequency = " + ADCInputTargetFrequency + "\r\n";
                Application::DoEvents();
                Error_Status = Set_ADC_Input_Target_Frequency(ADCInputTargetFrequency, TimeoutInMs);
                printf("\nError Status = %d", Error_Status);
                richTextBox1->Text += "Error Status = " + Error_Status + " (0 - No Error)\r\n";
                Application::DoEvents();

                printf("\n\nNumber of Samples per Channel = %d", NumberOfSamplesPerChannel);
                richTextBox1->Text += "Number of Samples per Channel = " + NumberOfSamplesPerChannel
                        + "\r\n";
                Application::DoEvents();
                Error_Status = Set_Number_of_Samples(NumberOfSamplesPerChannel, TimeoutInMs);
                printf("\nError Status = %d", Error_Status);
                richTextBox1->Text += "Error Status = " + Error_Status + "\r\n";
                Application::DoEvents();

                printf("\n\nApplying FFT Notch Filter Settings");
                richTextBox1->Text += "Applying FFT Notch Filter Settings \r\n";
                Application::DoEvents();
                Error_Status = FFT_Window_Notching(FFTSettingsType, NumberOfHarmonics,
NoOfBinsToRemoveEitherSideOfHarmonics, NoOfBinsToRemoveAfterDC,
NoOfBinsToRemoveEitherSideOfFundamental, CustomNotchFrequencies,
NoOfBinsToRemoveOnEitherSideOfCustomFrequencies, NumberOfCustomFreq, enableFsby2MinusFinNotching,
binsToRemoveOnEitherSideOfFsby2, TimeoutInMs);
                printf("\nError Status = %d", Error_Status);
                richTextBox1->Text += "Error Status = " + Error_Status + " (0 - No Error)\r\n";
                Application::DoEvents();

                printf("\n\nApplying FFT Average Settings");
                richTextBox1->Text += "Applying FFT Average Settings \r\n";
                Application::DoEvents();
                printf("\nAverage FFT Enabled = %d, (Disabled - 0, Enabled - 1)", AverageFFTOn);
                richTextBox1->Text += "Average FFT Enabled = " + AverageFFTOn + " (Disabled - 0, Enabled -
1)\r\n";
                Application::DoEvents();
                printf("\nNumber Of Averages = %d", NumberOfAverages);
                richTextBox1->Text += "Number Of Averages = " + NumberOfAverages + "\r\n";
                Application::DoEvents();
                Error_Status = ADC_Average_Settings(AverageFFTOn, NumberOfAverages, TimeoutInMs);
                printf("\nError Status = %d", Error_Status);
                richTextBox1->Text += "Error Status = " + Error_Status + " (0 - No Error)\r\n";
                Application::DoEvents();

                printf("\n\nApplying Trigger Settings");
                richTextBox1->Text += "\r\n\r\n ################ Applying Trigger Settings #############
\r\n";
                Application::DoEvents();
                Error_Status = Trigger_Option(TriggerModeEnable, SoftwareTriggerEnable,
ArmOnNextCaptureButtonPress, TriggerCLKDelays, TimeoutInMs);
                printf("\nError Status = %d", Error_Status);
                richTextBox1->Text += "Error Status = " + Error_Status + "\r\n";
                Application::DoEvents();

                printf("\n\nPress any key to exit..");
                richTextBox1->Text += "\r\n ############## Successfully connected to AFE5809. You may
capture data now. ################\r\n";
```

```
                button1->Text = "Connected";
Application::DoEvents();
                //pictureBox1->Image = Image::FromFile("C:\\MEIPA Sonar System\\resultImg.png");


                //Disconnecting from the board
                //printf("\n\nDisconnecting from the board");
                //Error_Status = Disconnect_Board(TimeoutInMs);
                //printf("\nError Status = %d",Error_Status);


                //****************************** Release the Dll ********************************//

                //FreeLibrary(hGetProcIDDLL);

                /* The return val from the dll */
                //return 0;// intMyReturnVal;

// ################### capture button ########################
                        // ################### capture button ########################

        private: System::Void button3_Click(System::Object^ sender, System::EventArgs^ e) {
                button3->Text = "Capturing...";


                //Use HSDC Ready function to check if the GUI has completed all its operations(like downloading firmware)...
                printf("\n\nUsing HSDC Ready function to check if the GUI is Ready...");
                richTextBox1->Text += "Using Ready function to check if the GUI is Ready... \r\n";
                Error_Status = HSDC_Ready(60000); //Waiting to check if HSDCPro has completed all its operations.
                printf("\nError Status = %d", Error_Status);
                richTextBox1->Text += "Error Status = " + Error_Status + " (0 - No Error)\r\n";


                // ################# Data Acquisition ##################
                richTextBox1->Text += "\r\n\r\n\r\n##################### Data Acquisition Start ##################### \r\n\r\n";

                if (TriggerModeEnable == 0) //Normal Capture
                {
                        printf("\n\nStart Normal Capture. Press any key to continue...");
                        richTextBox1->Text += "Start Normal Capture. Press any key to continue... \r\n";

                        printf("\nStarting Normal Capture...");
                        richTextBox1->Text += "Starting Normal Capture... \r\n";
                        Error_Status = Pass_Capture_Event(TimeoutInMs);
                        printf("\nError Status = %d", Error_Status);
                        richTextBox1->Text += "Error Status = " + Error_Status + " (0 - No Error)\r\n";

                }
                else if (TriggerModeEnable == 1 && SoftwareTriggerEnable == 0) //External Hardware Trigger
                {
                        printf("\n\nRead DDR Memory. Press any key to continue...");
                        richTextBox1->Text += "Read DDR Memory. Press any key to continue... \r\n";

                        printf("\nReading DDR Memory...");
```

```
                richTextBox1->Text += "Reading DDR Memory... \r\n";
                Error_Status = Read_DDR_Memory(WaitToCheckTrigger, TimeoutInMs);
                printf("\nError Status = %d", Error_Status);
                richTextBox1->Text += "Error Status = " + Error_Status + " (0 - No Error)\r\n";
            }
            else if (TriggerModeEnable == 1 && SoftwareTriggerEnable == 1) //Software Trigger
            {
                printf("\n\nGenerate Software Trigger. Press any key to continue...");
                richTextBox1->Text += "Generate Software Trigger. Press any key to continue... \r\n";
                //getch();
                printf("\nGenerating Software Trigger...");
                richTextBox1->Text += "Generating Software Trigger... \r\n";
                Error_Status = Generate_Software_Trigger(WaitToCheckTrigger, TimeoutInMs);
                printf("\nError Status = %d", Error_Status);
                richTextBox1->Text += "Error Status = " + Error_Status + " (0 - No Error)\r\n";
            }

            printf("\n\nChecking if GUI has completed all its operations...");
            richTextBox1->Text += "Checking if AFE GUI has completed all its operations... \r\n";
            //Maximum waiting time will depend on the number of averages configured
            Error_Status = HSDC_Ready(TimeoutInMs*NumberOfAverages); //Waiting to check if HSDCPro has completed all its operations.
            printf("\nError Status = %d", Error_Status);
            richTextBox1->Text += "Error Status = " + Error_Status + " (0 - No Error)\r\n";

            printf("\n\nSaving ADC Raw Data as CSV file at %s", CSVFilePathWithName);
            richTextBox1->Text += "Saving ADC Raw Data as CSV file at: " + textBox11->Text + "\r\n ";
            Error_Status = Save_Raw_Data_As_CSV(CSVFilePathWithName, TimeoutInMs);
            printf("\nError Status = %d", Error_Status);
            richTextBox1->Text += "Error Status = " + Error_Status + " (0 - No Error)\r\n";

            richTextBox1->Text += "#################### Data Acquisition Finished #################### \r\n";

            // trigger matlab to start capture
            String^ fileName = "matlab_vs_connector.txt";
            StreamWriter^ sw = gcnew StreamWriter(fileName);
            sw->WriteLine("1");
            sw->Close();
            richTextBox1->Text += "matlab triggered and runCode = 1 \r\n";
            Application::DoEvents();

            button3->Text = "Continuous Capture";

            button3->PerformClick();}
```

Appendix B. The Schematics of the custom HV Pulse Generator

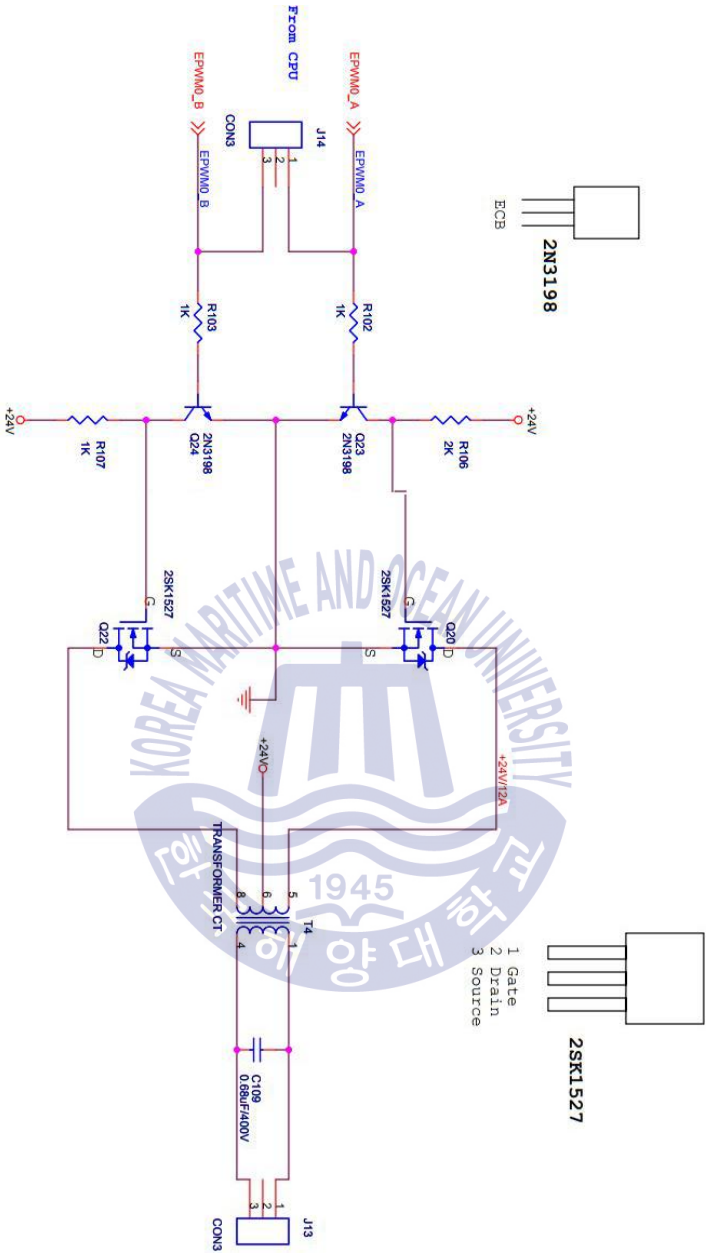


Figure 45 Schematics of the custom HV pulse generator

Appendix C. Prototype Photos

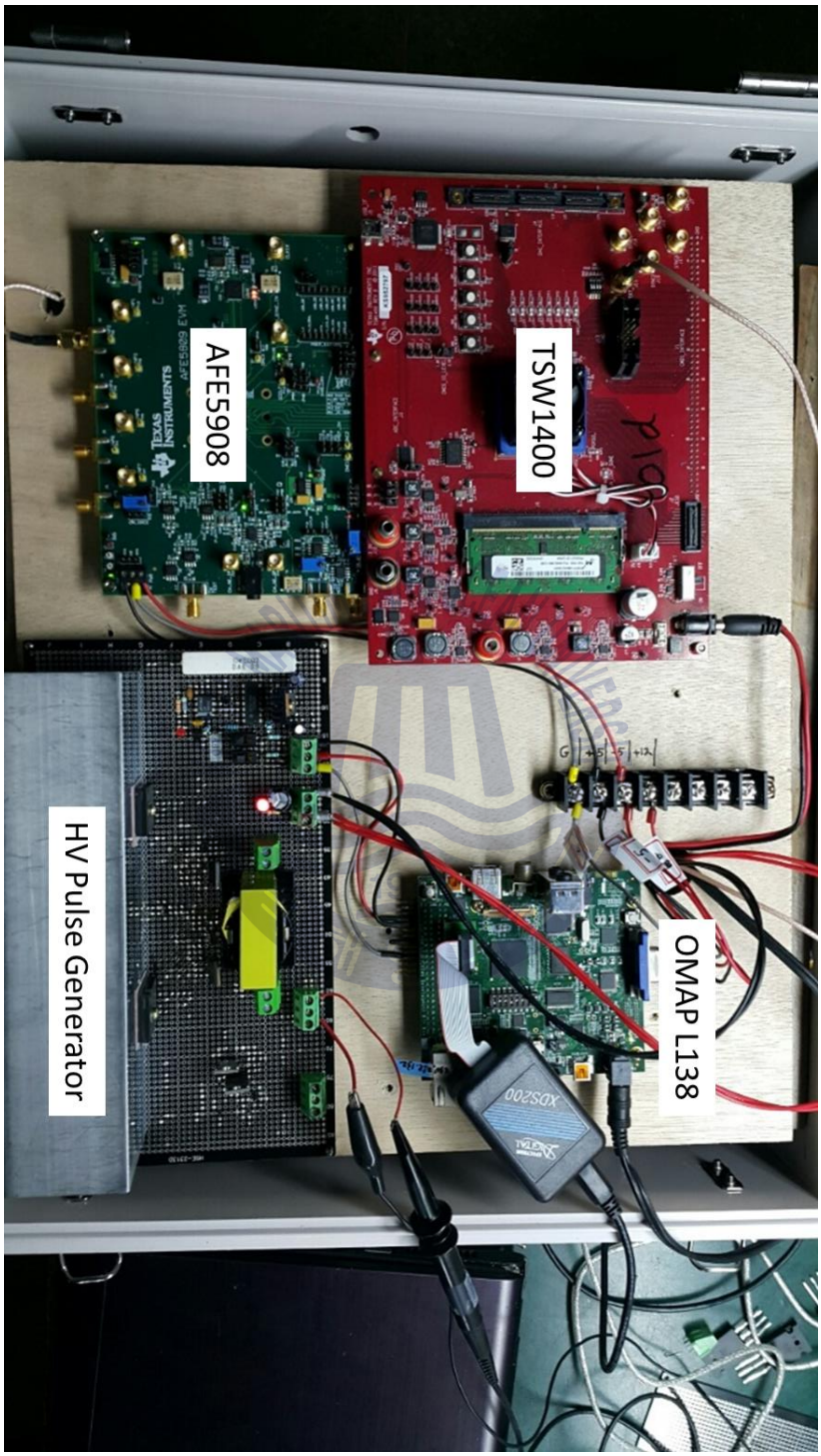


Figure 46 Sonar Prototype Photo: Top level

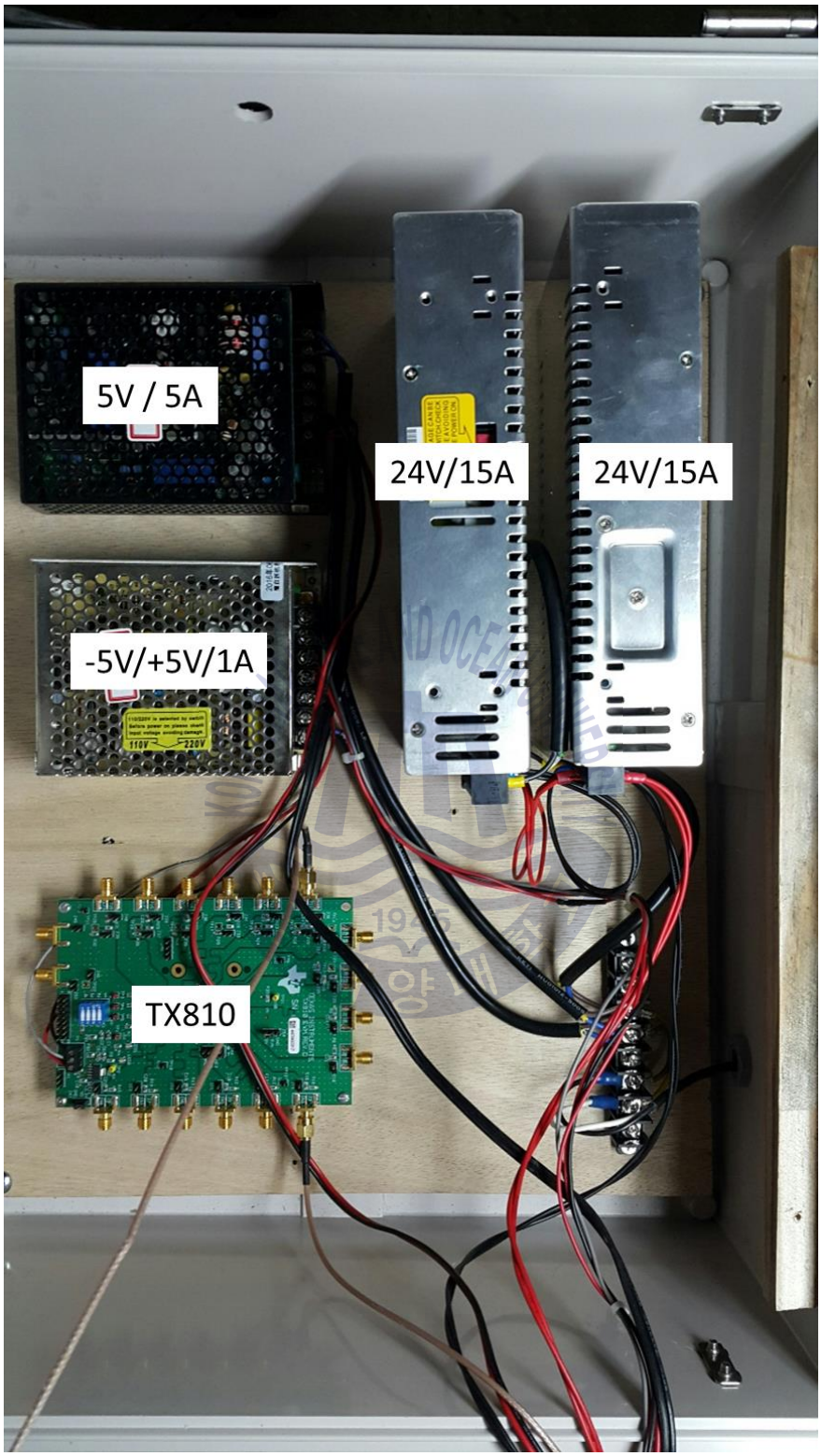


Figure 47 Sonar Prototype Photo: Bottom level

Appendix D. Snippet of C++ code used to configure A.1 OMAP-L138's eHRPWM

Snippet of C++ code used to configure ***A.1 OMAP-L138's*** eHRPWM is shown is Appendix D.

```
    SetupIntc();

  PSCModuleControl(SOC_PSC_0_REGS, HW_PSC_EHRPWM, PSC_POWERDOMAIN_ALWAYS_ON,
          PSC_MDCTL_NEXT_ENABLE);
  PSCModuleControl(SOC_PSC_1_REGS, HW_PSC_EHRPWM, PSC_POWERDOMAIN_ALWAYS_ON,
          PSC_MDCTL_NEXT_ENABLE);

  EHRPWM1PinMuxSetup();
  EHRPWM0PinMuxSetup();

  // timer
  /* Set up the Timer2 peripheral */
  TimerSetUp64Bit();
  /* Set up the AINTC to generate Timer2 interrupts */
  TimerIntrSetUp();
  /* Enable the timer interrupt */
  TimerIntEnable(SOC_TMR_2_REGS, TMR_INT_TMR12_NON_CAPT_MODE);
  /* Start the timer. Characters from cntArr will be sent from the ISR */
  TimerEnable(SOC_TMR_2_REGS, TMR_TIMER12, TMR_ENABLE_CONT);
  /* make sure all the characters from cntArray from ISR */

  /* TimeBase configuration */
  /* Configure the clock frequency */
  EHRPWMTimebaseClkConfig(SOC_EHRPWM_0_REGS,
                SOC_EHRPWM_0_MODULE_FREQ / CLOCK_DIV_VAL,
                SOC_EHRPWM_0_MODULE_FREQ);
  EHRPWMTimebaseClkConfig(SOC_EHRPWM_1_REGS,
                SOC_EHRPWM_1_MODULE_FREQ / CLOCK_DIV_VAL,
                SOC_EHRPWM_1_MODULE_FREQ);

  /* Configure the period of the output waveform */
  EHRPWMPWMOpFreqSet(SOC_EHRPWM_0_REGS,
            SOC_EHRPWM_0_MODULE_FREQ / CLOCK_DIV_VAL,
            (SOC_EHRPWM_0_MODULE_FREQ / CLOCK_DIV_VAL) / 0x157,
            EHRPWM_COUNT_UP, EHRPWM_SHADOW_WRITE_DISABLE);
  EHRPWMPWMOpFreqSet(SOC_EHRPWM_1_REGS,
            SOC_EHRPWM_1_MODULE_FREQ / CLOCK_DIV_VAL,
            (SOC_EHRPWM_1_MODULE_FREQ / CLOCK_DIV_VAL) / 0x157,
            EHRPWM_COUNT_UP, EHRPWM_SHADOW_WRITE_DISABLE);

  /* synchronization*/
  EHRPWMTimebaseSyncEnable(SOC_EHRPWM_0_REGS,                    250,
EHRPWM_COUNT_DOWN_AFTER_SYNC);
  EHRPWMTimebaseSyncEnable(SOC_EHRPWM_1_REGS,                    100,
EHRPWM_COUNT_DOWN_AFTER_SYNC);
```

```
   /* Disable syncout*/
   EHRPWMSyncOutModeSet(SOC_EHRPWM_0_REGS,
EHRPWM_SYNCOUT_COUNTER_EQUAL_ZERO);
   EHRPWMSyncOutModeSet(SOC_EHRPWM_1_REGS,
EHRPWM_SYNCOUT_COUNTER_EQUAL_ZERO);

   /* Configure Counter compare cub-module */
   /* Load Compare A value  */
   EHRPWMLoadCMPA(SOC_EHRPWM_0_REGS, 85, EHRPWM_SHADOW_WRITE_DISABLE,
   EHRPWM_COMPA_NO_LOAD,
            EHRPWM_CMPCTL_OVERWR_SH_FL);
   EHRPWMLoadCMPB(SOC_EHRPWM_0_REGS, 225, EHRPWM_SHADOW_WRITE_DISABLE,
   EHRPWM_COMPB_NO_LOAD,
            EHRPWM_CMPCTL_OVERWR_SH_FL);
   EHRPWMLoadCMPA(SOC_EHRPWM_1_REGS, 25, EHRPWM_SHADOW_WRITE_DISABLE,
   EHRPWM_COMPA_NO_LOAD,
            EHRPWM_CMPCTL_OVERWR_SH_FL);
   EHRPWMLoadCMPB(SOC_EHRPWM_1_REGS, 140, EHRPWM_SHADOW_WRITE_DISABLE,
   EHRPWM_COMPB_NO_LOAD,
            EHRPWM_CMPCTL_OVERWR_SH_FL);



   /* Configure Action qualifier */
   EHRPWMConfigureAQActionOnA(SOC_EHRPWM_0_REGS,
EHRPWM_AQCTLA_ZRO_EPWMXALOW,
                  EHRPWM_AQCTLA_PRD_DONOTHING,
                  EHRPWM_AQCTLA_CAU_EPWMXAHIGH,
                  EHRPWM_AQCTLA_CAD_DONOTHING,
                  EHRPWM_AQCTLA_CBU_EPWMXALOW,
                  EHRPWM_AQCTLA_CBD_DONOTHING,
                  EHRPWM_AQSFRC_ACTSFA_DONOTHING);
   /* Toggle when CTR = CMPA OR CMPB */
   EHRPWMConfigureAQActionOnA(SOC_EHRPWM_1_REGS,
EHRPWM_AQCTLA_ZRO_EPWMXAHIGH,
                  EHRPWM_AQCTLA_PRD_DONOTHING,
                  EHRPWM_AQCTLA_CAU_EPWMXATOGGLE,
                  EHRPWM_AQCTLA_CAD_DONOTHING,
                  EHRPWM_AQCTLA_CBU_EPWMXATOGGLE,
                  EHRPWM_AQCTLA_CBD_DONOTHING,
                  EHRPWM_AQSFRC_ACTSFA_DONOTHING);
   EHRPWMConfigureAQActionOnB(SOC_EHRPWM_1_REGS,
EHRPWM_AQCTLB_ZRO_EPWMXBHIGH,
                  EHRPWM_AQCTLB_PRD_DONOTHING,
                  EHRPWM_AQCTLB_CAU_DONOTHING
                  EHRPWM_AQCTLB_CAD_EPWMXBTOGGLE,
                  EHRPWM_AQCTLB_CBU_EPWMXBTOGGLE,
                  EHRPWM_AQCTLB_CBD_DONOTHING,
                  EHRPWM_AQSFRC_ACTSFB_DONOTHING);
```

Appendix E. Snippet of the code behind Fish Finder GUI (Matlab)

```
numCapture = str2double(get(handles.captureCount, 'string'));
numCapture2 = str2double(get(handles.captureCount2, 'string'));
% horizPos = str2double(get(handles.horizPos, 'string'));
MinPeakProminence = get(handles.MinPeakProminence, 'value');
MinPeakDistance = get(handles.MinPeakDistance, 'value');
%fprintf('\nMinPeakProminence %d', MinPeakProminence , '\n');

%sajad
%define variables
samplingTime = 65536; % we are using 16 bit ADC so our sampling time is 65536
SoundVelocity = 1500; % m/s - sound velocity in ocean water

filename = 'ADCdata.csv';
originalData = csvread(filename, 0,0,[0 0 65535 0]); % read data from AFE exceloutput
myData = transpose(originalData);
if(numCapture ~= 1)
   finalData1 = evalin('base','finalData1');
   %finalData2 = evalin('base','finalData2');
else
   finalData1 = [];
end

% if data larger than numCapture2, remove old data in matrix
if (numCapture2 >= 40)
   myMin = min(finalData1(1,:));
   finalData1(:, any(finalData1==myMin, 1)) = []; % find minium value in matrix, remove all the columns
end


axes(handles.axes1); % choose proper axis for top signal plot
findpeaks(myData,'MinPeakProminence',MinPeakProminence,'MinPeakDistance',MinPeakDistance,
'Annotate','extents')
% spectrogram(myData)
% colorbar

fishImg = imread('circle.jpg');
floor_piece = imread('floor_piece.jpg');

%assignin('base', 'myData1Column', myData(:, numCapture));
[peakValueTX, peakPositionTX] = findpeaks(myData,'MinPeakProminence',8000, 'NPeaks', 1,
'MinPeakDistance',MinPeakDistance, 'Annotate','extents');
[peakValue, peakPosition] =
findpeaks(myData,'MinPeakProminence',MinPeakProminence,'MinPeakDistance',MinPeakDistance,
'Annotate','extents');
[mm1, peakValueSize]= size(peakValue);

fprintf('\npeakValueSize is %d', peakValueSize);
fprintf('\n max finalData1 is %d', max(finalData1));

jj = 1;
[mm2, nn] = size(finalData1);
```

```
for i = 1:peakValueSize
   if (peakValue(i) > 1000 && peakValue(i) < 7000)
      %diffBetweenTXandRXPosition = peakPosition(i) - peakPositionTX(1,1);
      diffBetweenTXandRXPosition = peakPosition(i) - peakPositionTX(1);
      time = diffBetweenTXandRXPosition/samplingTime; % sec
      distance = round(SoundVelocity * time / 2); % meter
      finalData1(1, nn+jj) = numCapture;
      finalData1(2, nn+jj) = distance;
      jj = jj + 1;

   end
end

assignin('base', 'finalData1', finalData1);
%assignin('base', 'finalData2', finalData2);

% set(handles.horizPos, 'string', horizPos - 0.05);
axes(handles.axes25); % choose proper axis for top signal plot
scatter(finalData1(1,:), finalData1(2,:), 'wo','MarkerFaceColor','r');
set(gca,'Ydir','reverse')
%set(gca,'Color',[0.3 0.75 0.93]);

ylabel('Depth (meter)')
```

Appendix F. Schematics of AFE5809 EVM

The schematics of AFE5809 EVM is shown in the following figures.

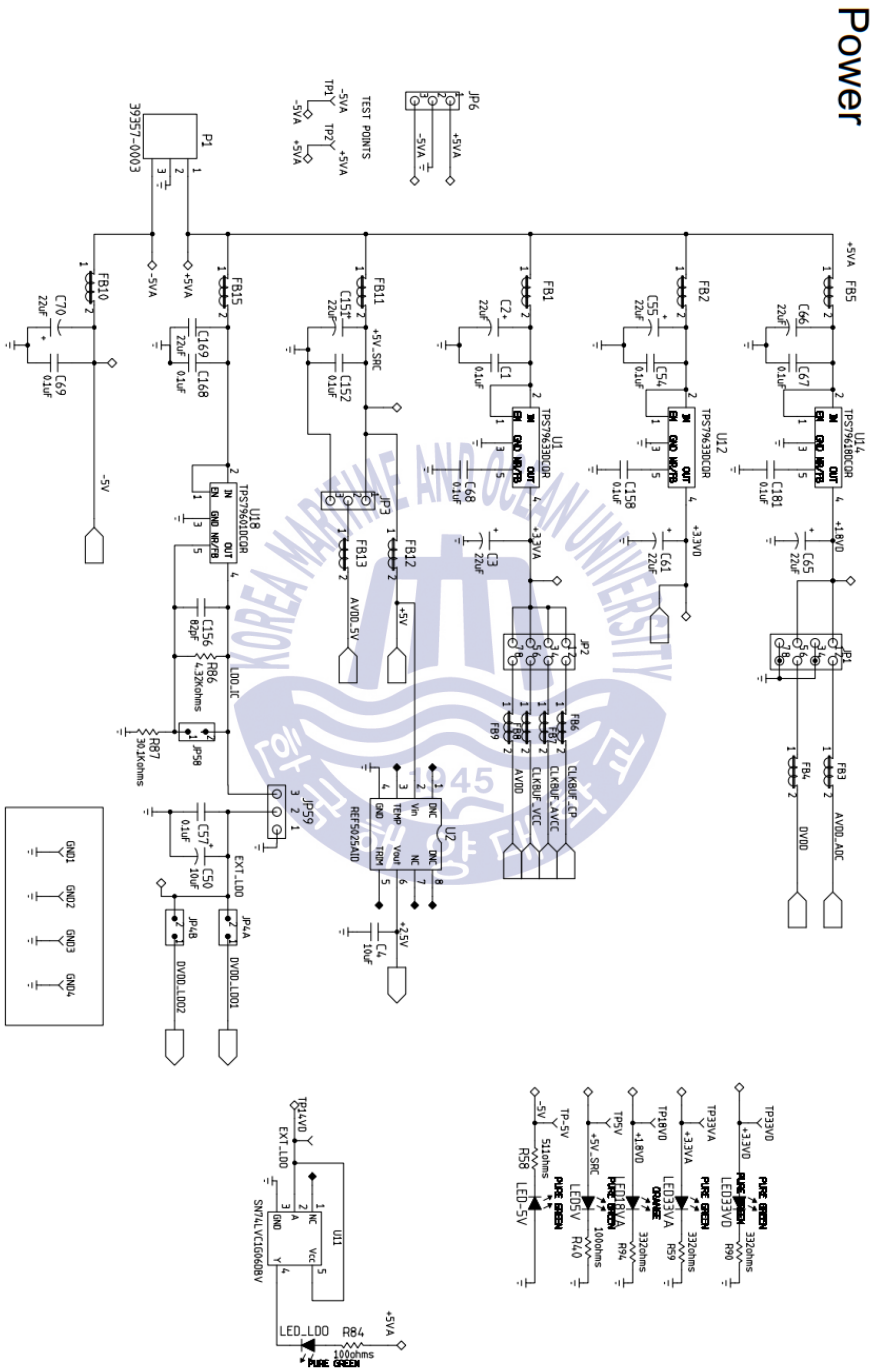


Figure 48 AFE5809 EVM Schematic (Sheet 1 of 10)

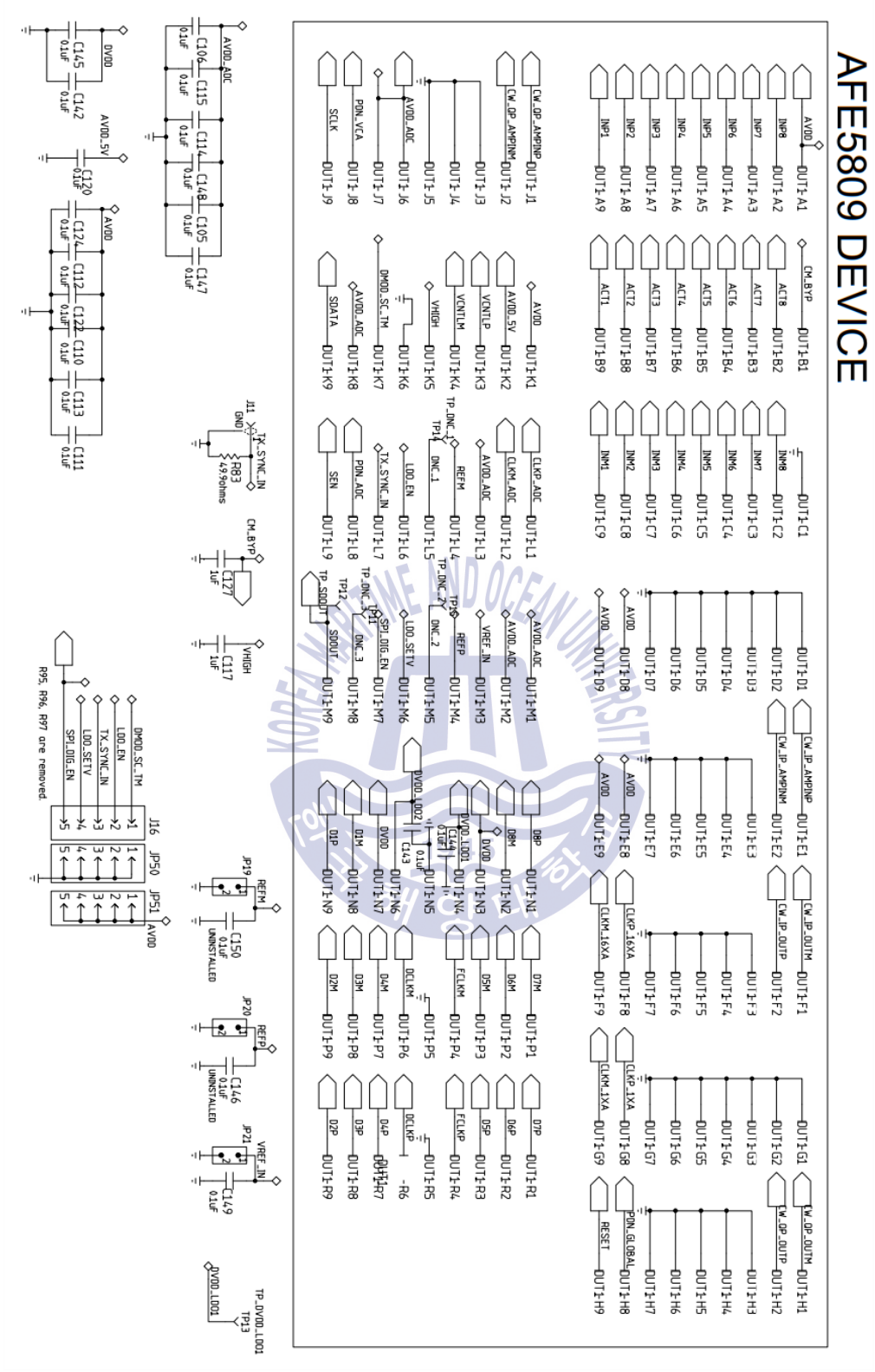


Figure 49 AFE5809 EVM Schematic (Sheet 2 of 10)

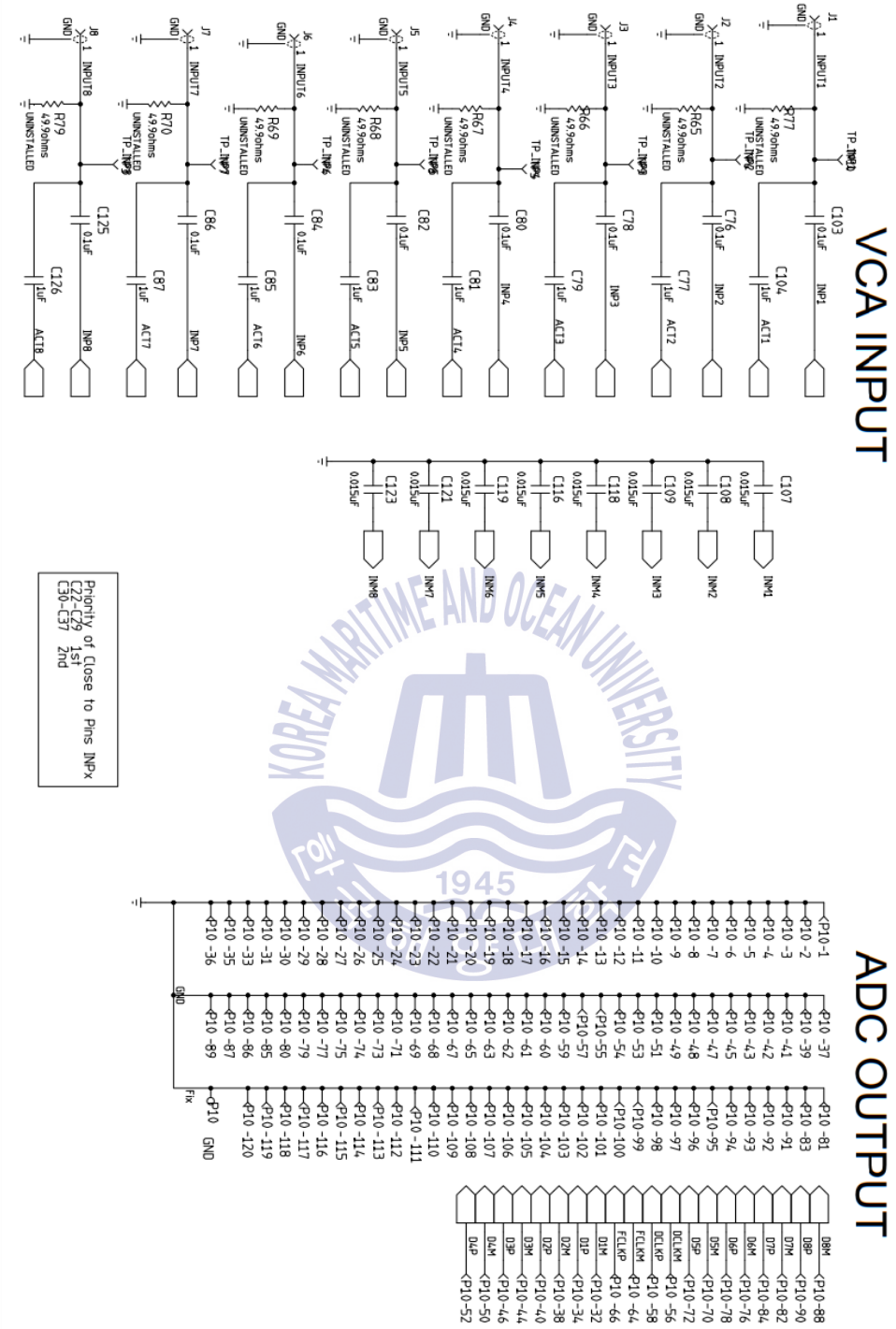


Figure 50 AFE5809 EVM Schematic (Sheet 3 of 10)

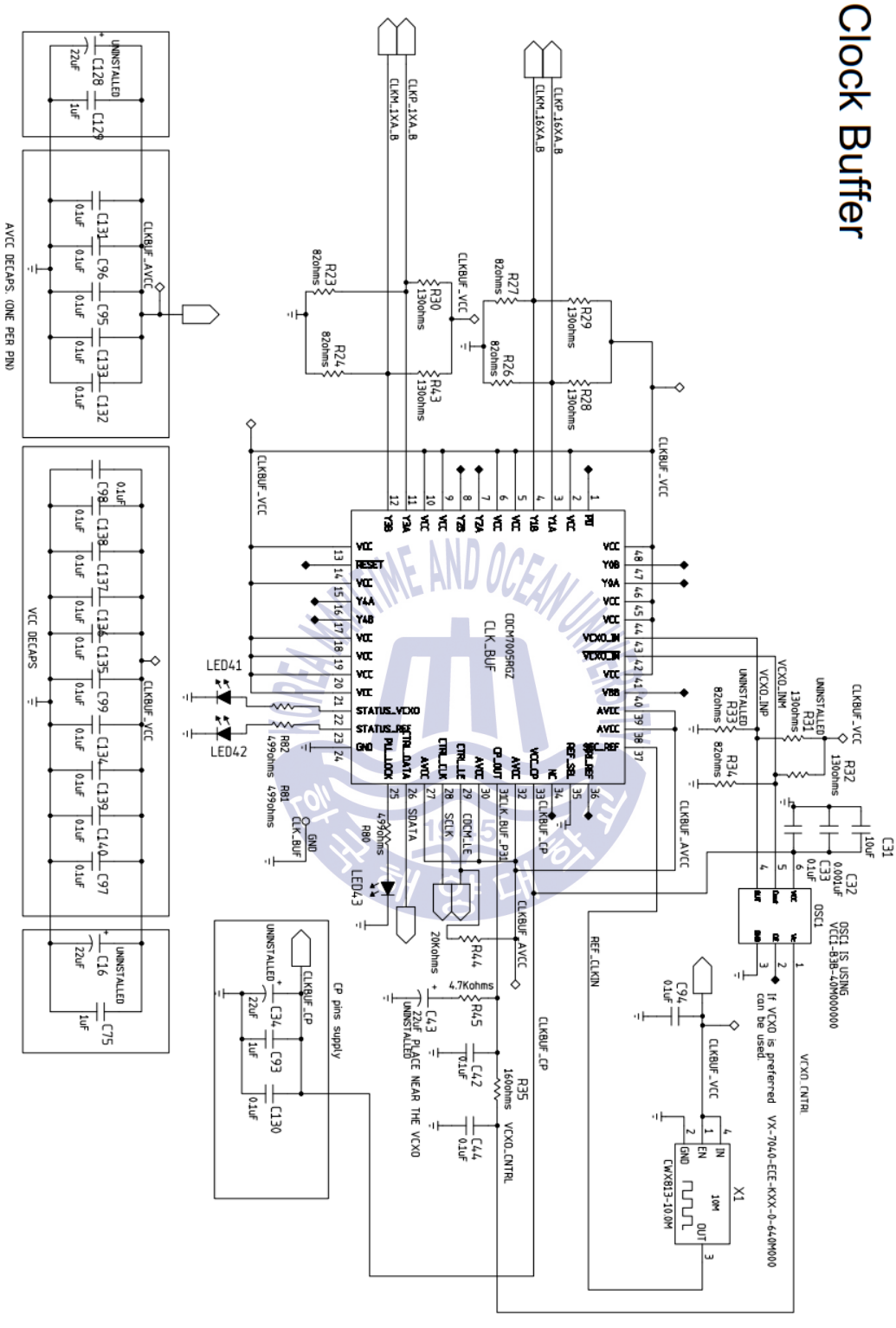


Figure 51 AFE5809 EVM Schematic (Sheet 4 of 10)

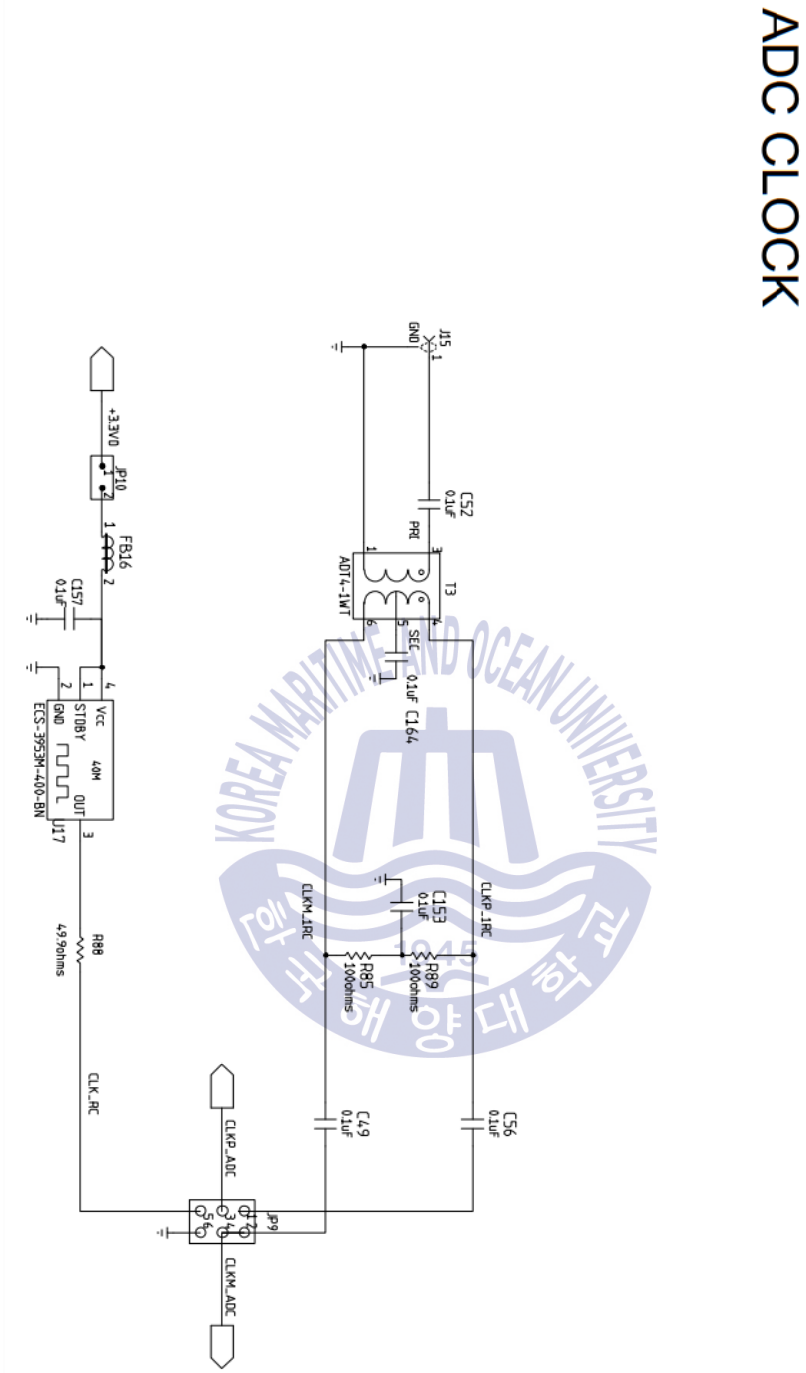


Figure 52 AFE5809 EVM Schematic (Sheet 5 of 10)

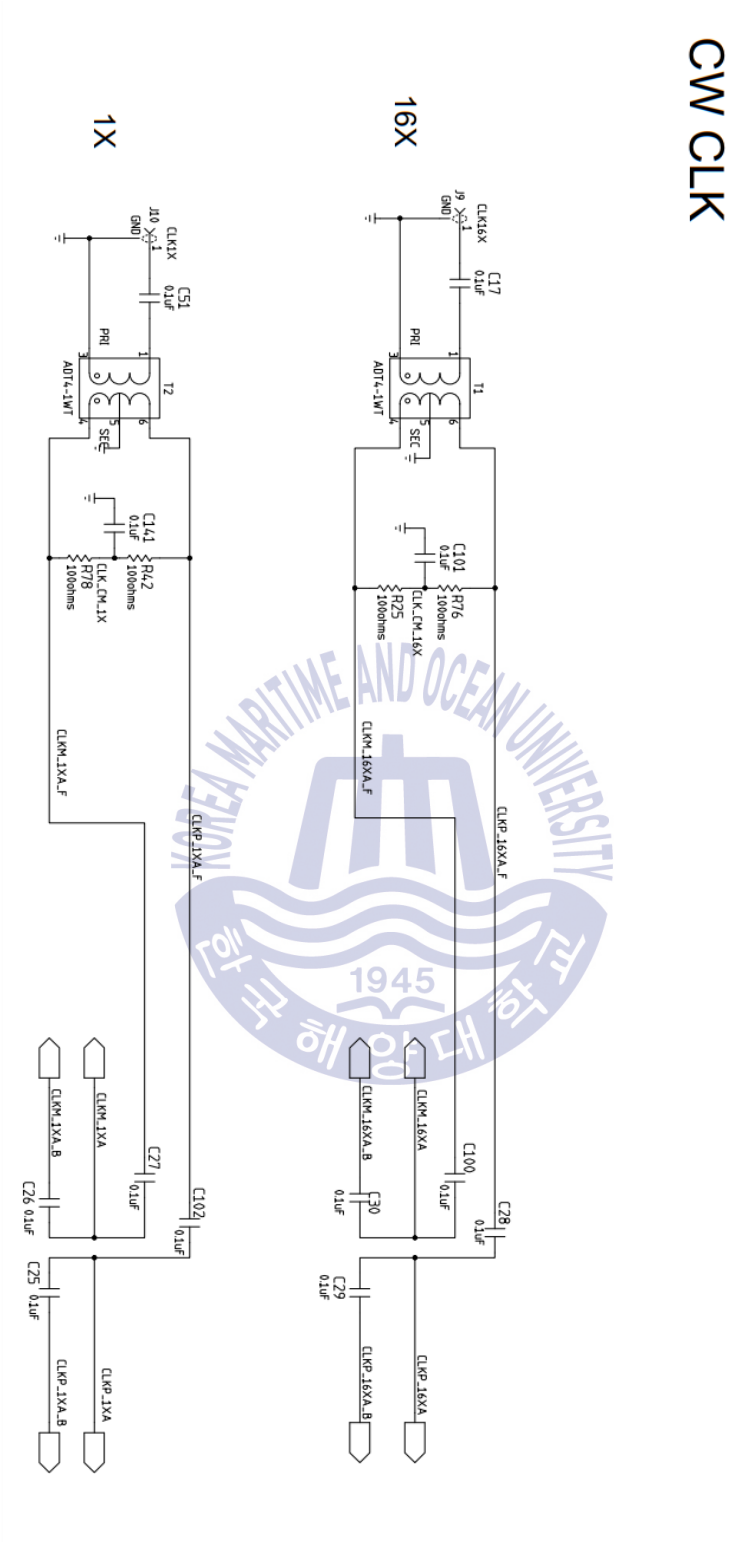


Figure 53 AFE5809 EVM Schematic (Sheet 6 of 10)

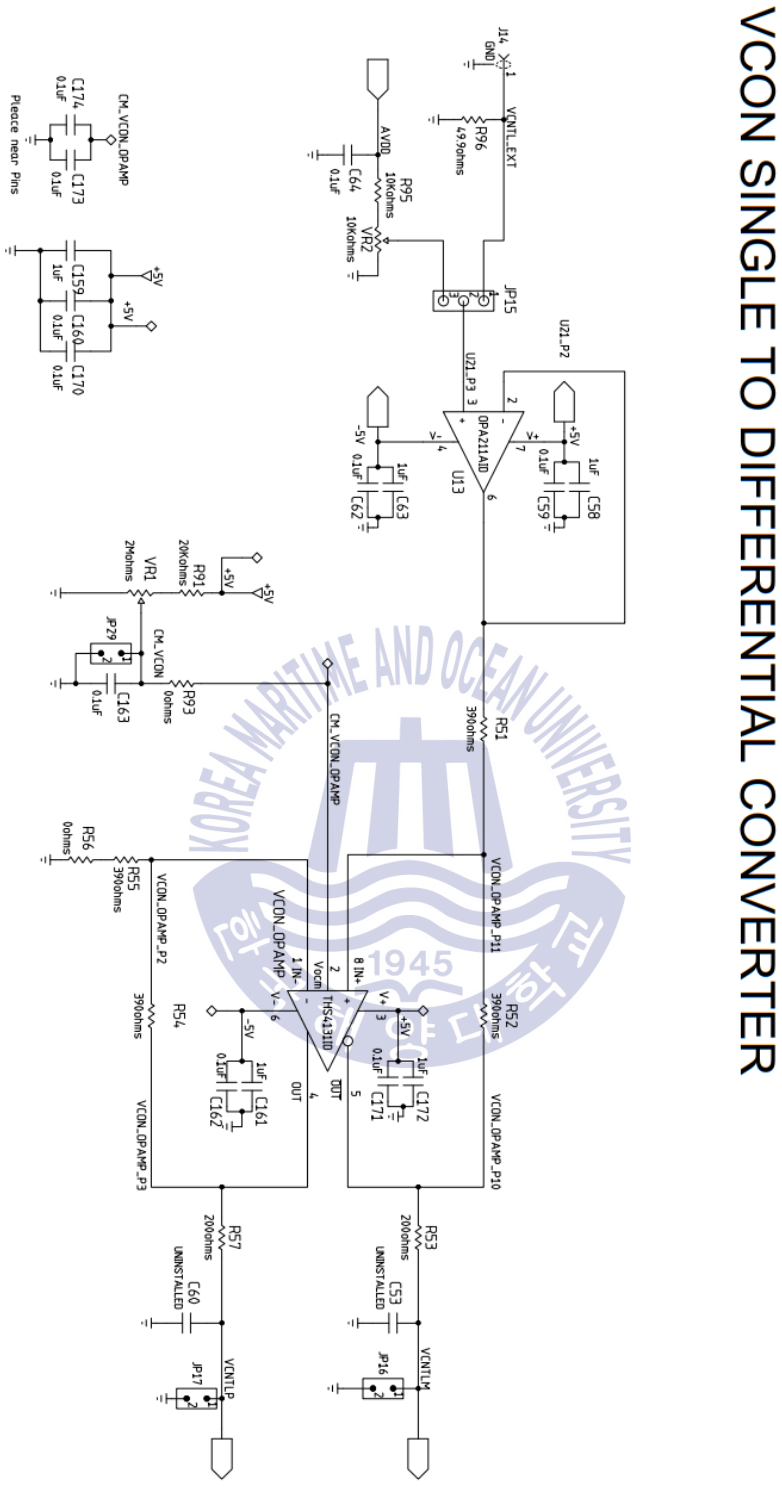


Figure 54 AFE5809 EVM Schematic (Sheet 7 of 10)

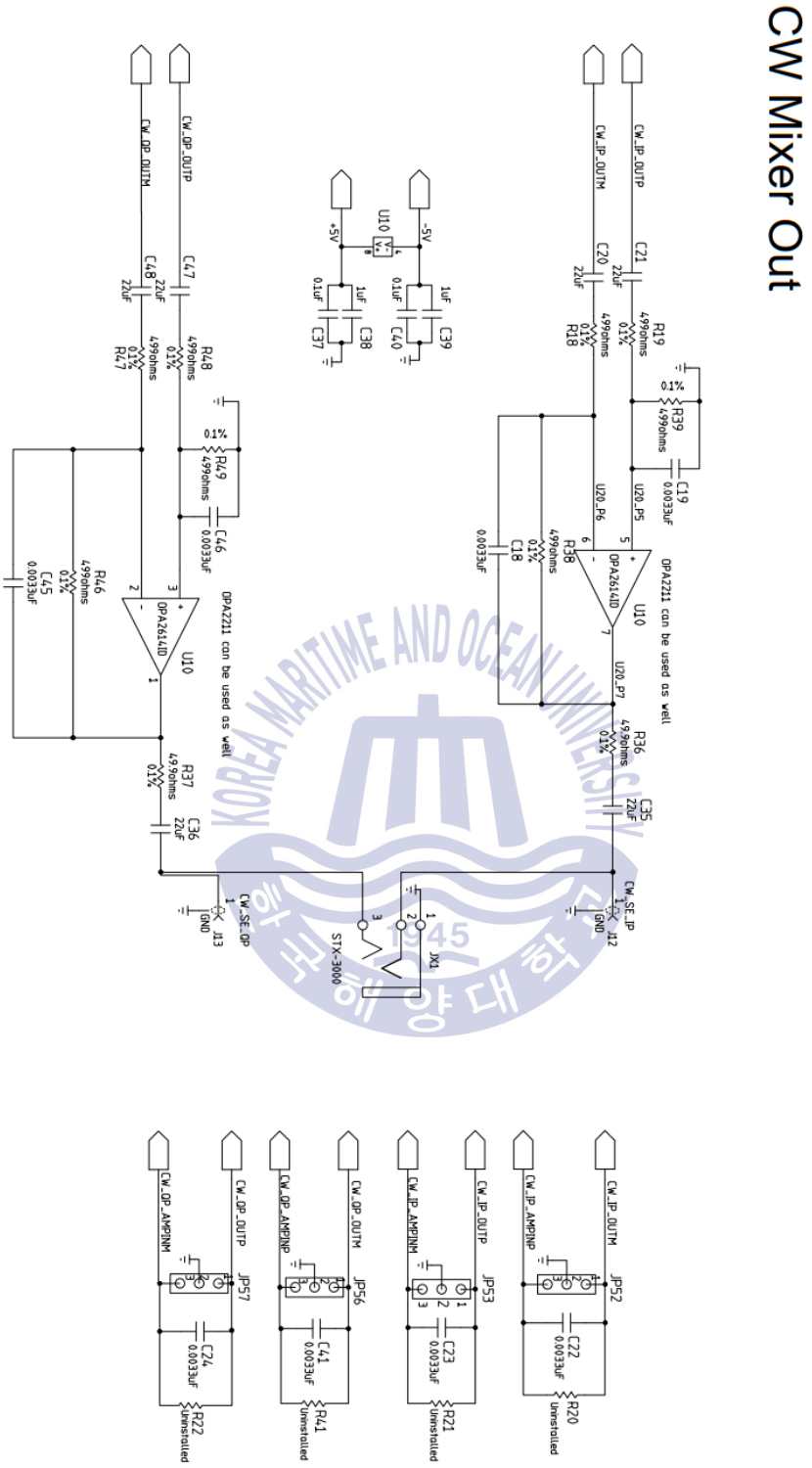


Figure 55 AFE5809 EVM Schematic (Sheet 8 of 10)

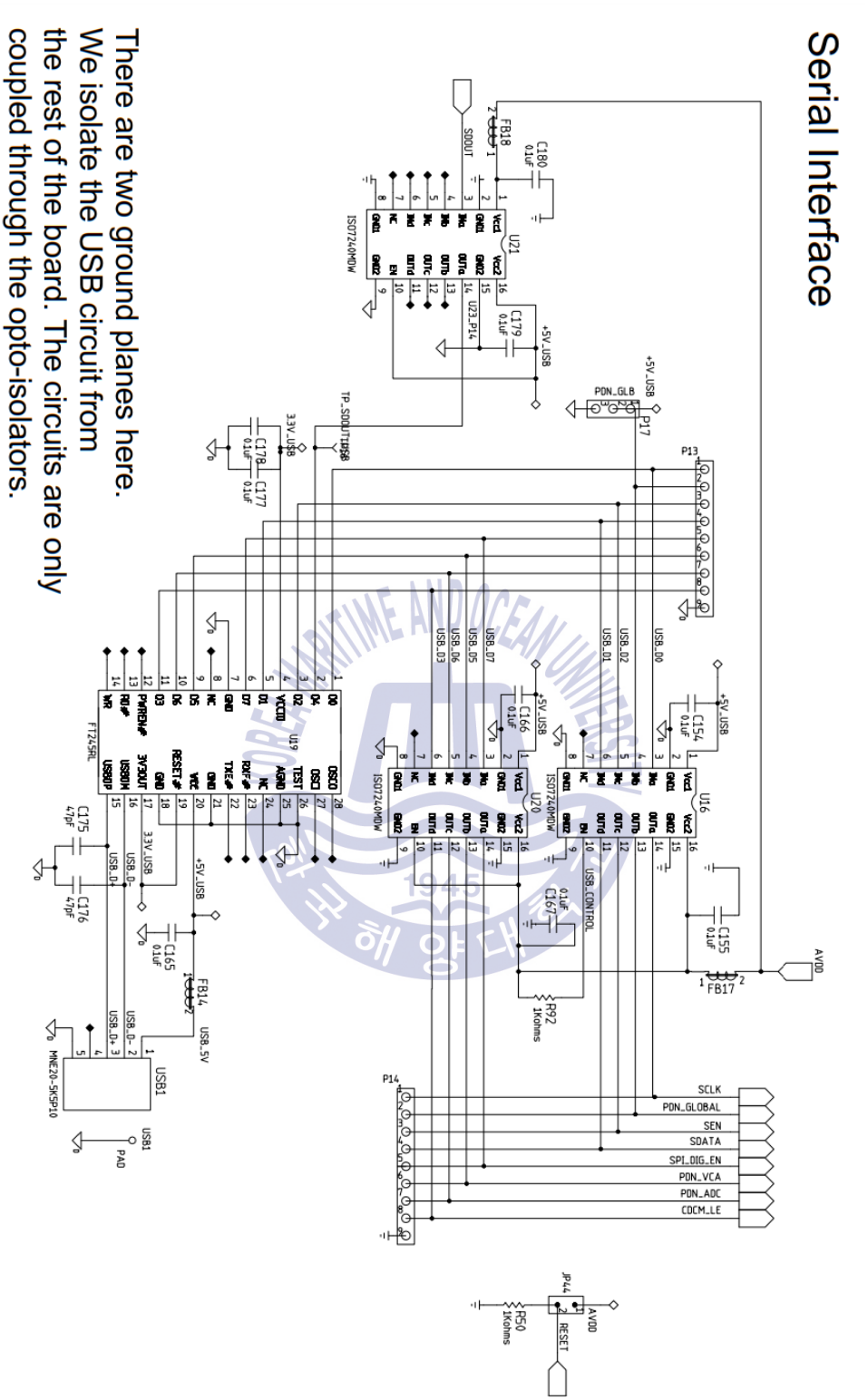


Figure 56 AFE5809 EVM Schematic (Sheet 9 of 10)

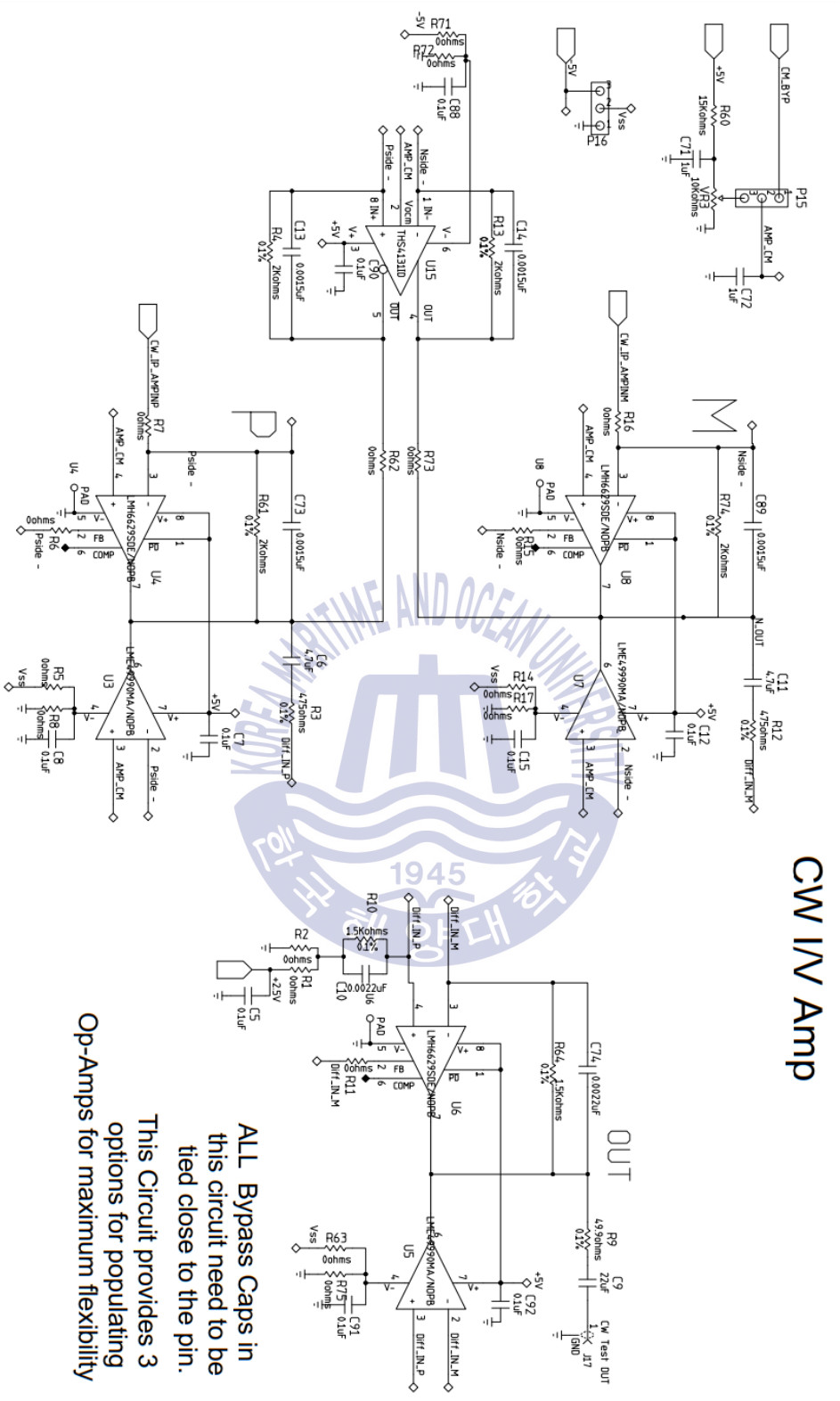


Figure 57 AFE5809 EVM Schematic (Sheet 10 of 10)

Appendix G. Schematic of TX810 EVM

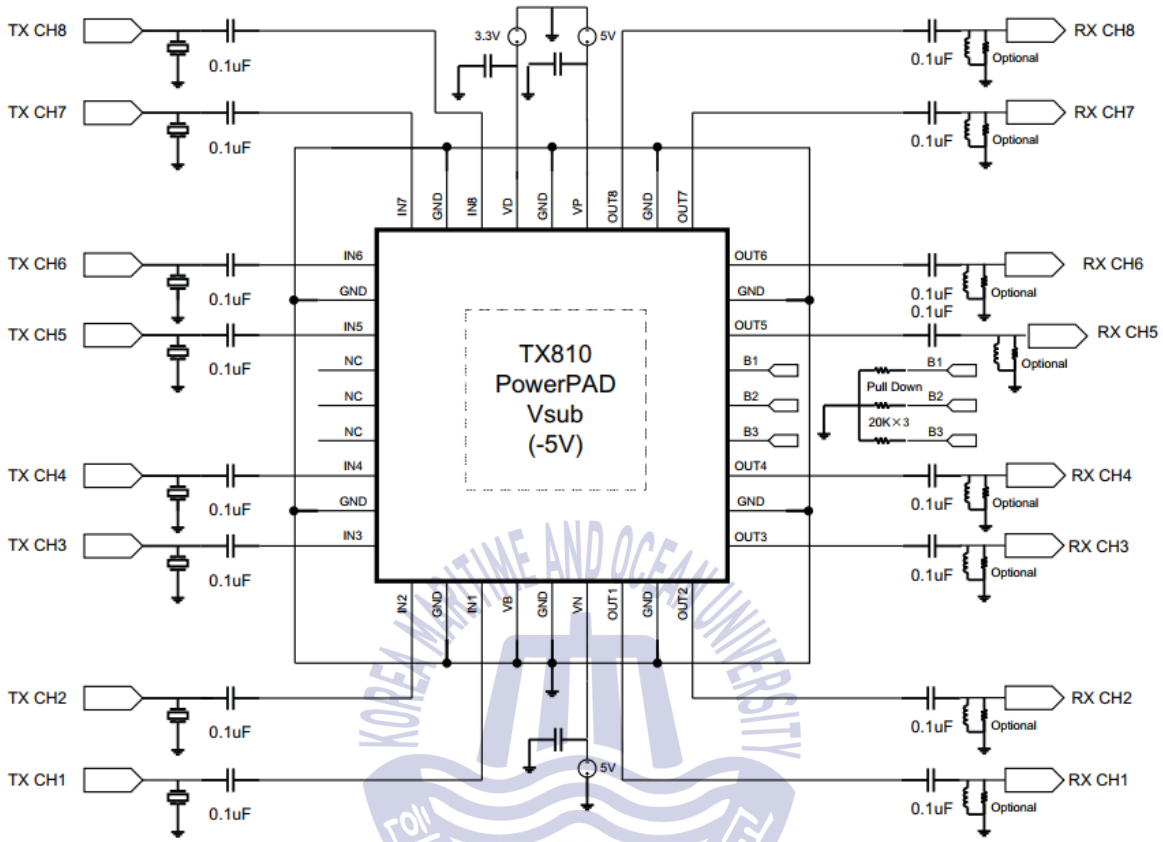


Figure 58 Schematic of TX810 EVM

Appendix H. Schematic of OMAP-L138 EVM

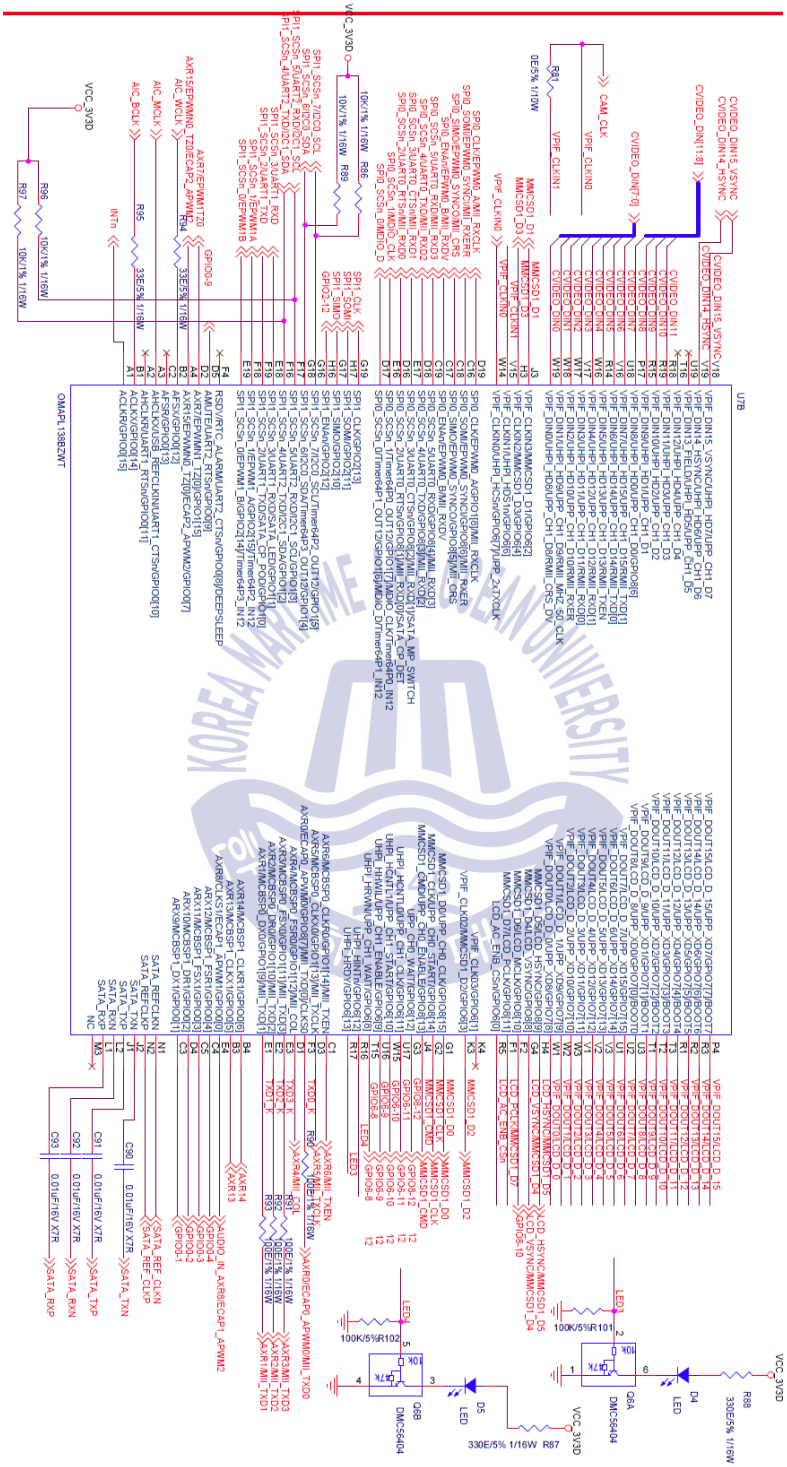


Figure 59 OMAP L-138 Schematic