FAMU-FSU College of Engineering

Department of Electrical and Computer Engineering

Final Report - Spring 2016

Synthetic Aperture Radar Imager

ECE Design Team 11

Members:

Olivier Cedric Barbier (ECE)	oliver.barbier
Jordan Bolduc (ECE)	jpb12c
Scott Nicewonger (ECE)	swn10
Julian Rodriguez (ME)	jar12g
Kegan Stack (ME)	kts11d

Date:

April 11, 2016

Table of Contents

Executive Summary

Problem Statement

Design and implement a proof of concept prototype of a static, stationary radar imager that exploits Synthetic Aperture Radar (SAR) technology.

Selected Method

Two perpendicularly aligned linear arrays provide transmit to receive displacement to produce phase fronts which correspond to reflection angles of arrival.

Key Features

The SAR uses 10GHz pulsed RF to image a 40inch x 40 inch scene at a distance of 20 feet.

Evaluation method

Evaluation is based on the ability to detect an RF corner reflector in the downrange scene.

Important results

The SAR successfully detected a corner reflector at boresight and generated a phase slope difference as the corner reflector was moved radially about boresight of the array.

ACKNOWLEDGMENTS

I. Introduction

1.1 Problem & Users

In partnership with the FAMU/FSU College of Engineering and Northrop Grumman, the objective of the Synthetic Aperture Radar (SAR) Imager Project is to develop a low-cost weapon detection system that provides suitable imagery resolution for physical security and military force protection applications.

Current detection technologies commonly employed in the security industry such as metal detectors, Advanced Imaging Technology (AIT) scanners, and x-ray scanners can be expensive, obtrusive, and require the subject to be inside the apparatus. An imager based on SAR technology, composed primarily of commercial-off-the-shelf (COTS) components, can be implemented at a lower cost than many industry-standard scanners; it may be placed behind a barrier, out of view from subjects; and most importantly, it can identify concealed metal objects from a distance.

In environments with multi-layered physical security protocols, the SAR imager's superior range could alert security professionals to potential threats before they reach an access control point, or before they progress further into a secure area, depending in which security layer the SAR is deployed. Some environments may be vulnerable to physical attack, but conventional AIT body scanners are too obtrusive or inefficient. An amusement park, for instance, might have high-level security needs, but their customers would not tolerate stepping into a full-body scanner.

Furthermore, random screening protocols have been widely criticized for being culturally or racially biased in practice. With SAR capability, guests can be discreetly imaged while queuing, and persons of interest can be identified for additional screening based on the presence of metal signatures rather than the caprice of a human screener.

1.2 Assumptions & Limitations

The scope of this project is limited to demonstration of the underlying theory of operation. A radar reflector is placed at known range and angle from boresight, rather than tracking a weapon on a person in the final operational environment.

1.3 End Product

The final design iteration of Spring 2016 produced a greatly improved implementation of a legacy SAR design. The product is capable of detecting the presence of a corner reflector in the downrange scene.

Manual scanning mode is fully implemented in the FPGA, while real-time automatic scanning logic has been coded but not fully realized. Signal processing backbone has been coded and prototyped in MATLAB and is ready for field testing.

II. System Design

2.1 System-Level Design Overview

Figure-1 shows the top-down block diagram of the SAR system. From the voltage controlled oscillator (VCO) that generates the 5 GHz intermediate frequency (IF) analog signal, the signal travels through three component chains: the Transmission Path, the Reference or Local Oscillator (LO) Path, and the Receive Path. Movement between these paths is governed by the Xilinx Nexys-3 FPGA, which provides timing and control voltages to the switches.

Three high-frequency, solid-state switches are used to set the system into the appropriate state. These are the Single Pole Double Throw (SPDT), Single Pole Four Throw (SP4T), and Single Pole 16 Throw (SP16T) switches. The state of the SPDT switch determines whether the system is in Transmit Mode or Receive Mode. The SP4T and SP16T switches determine which transmit antennas and which receive antennas are active, respectively.

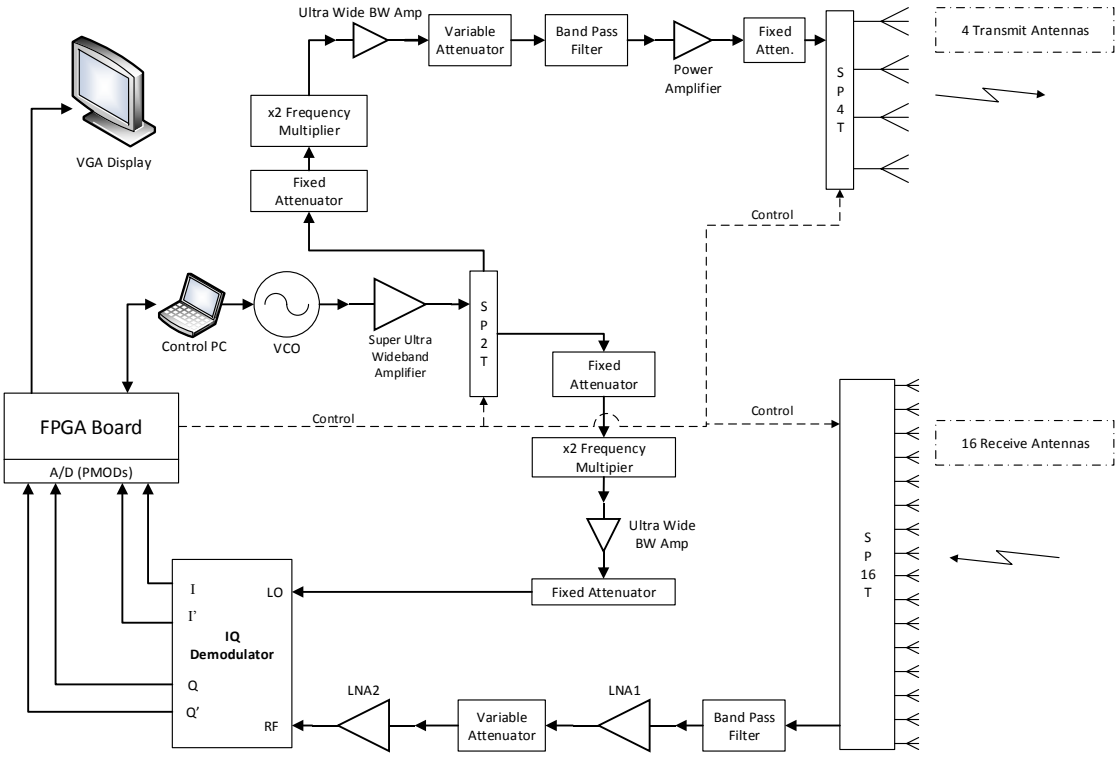


Figure-1: SAR Top-down block diagram

The following sections will elaborate in detail on the SAR subsystems, their functions, and how their functions are accomplished, but at the simplest level, the SAR must transmit a known signal, capture its reflection, compare the reflection to the signal that was sent, and process the data. Figure-2 shows a simple process diagram for a single transmit pulse.

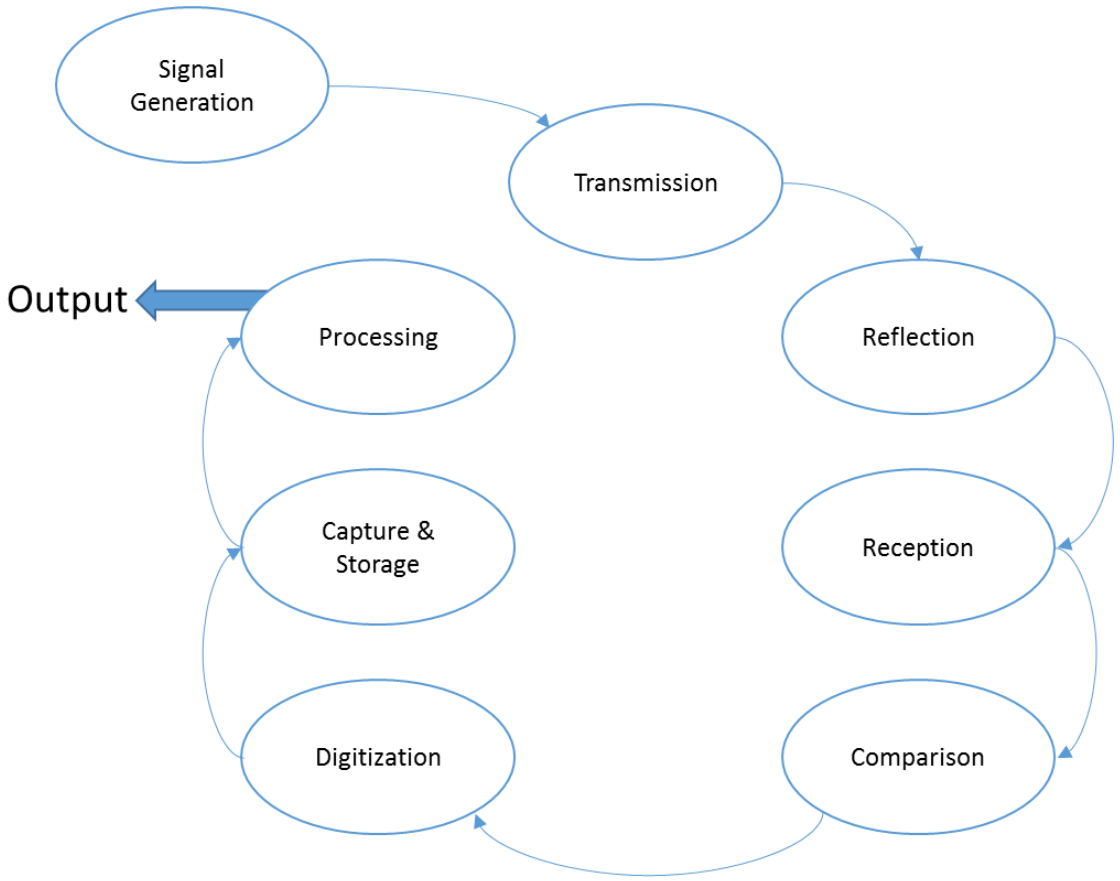


Figure-2: Process diagram for a single pulse

Signal Generation is shown as a block outside the cycle since the VCO is always generating the phase-locked analog signal. In Transmission Mode, the signal is amplified and pulsed to the physical target downrange. In Receive Mode, the signal generated by the VCO is used as a reference to compare the phase of the reflected signal. The SPDT switch reroutes the signal to the LO port of the IQ Demodulator. The IQ Demodulator compares the phase difference between the LO signal and the received signal reflected from the target, and outputs the In-phase and Quadrature components as voltages. These I and Q analog voltages are digitized, encoded, and stored for processing with the data from other pulses.

The sequence described is repeated for each transmitter/receiver antenna pair along the linear array. The design uses 16 transmitter/receiver pairs along each axis to generate a phase slope about the 16 phase centers created. Each phase center maps to a region of the downrange scene and corresponds to a unique, complex basis function. In signal processing, a Discrete Fourier Transform algorithm convolves the received data with the basis functions to associate energy (amplitude) with angle of arrival (since each phase center corresponds to an angle of arrival).

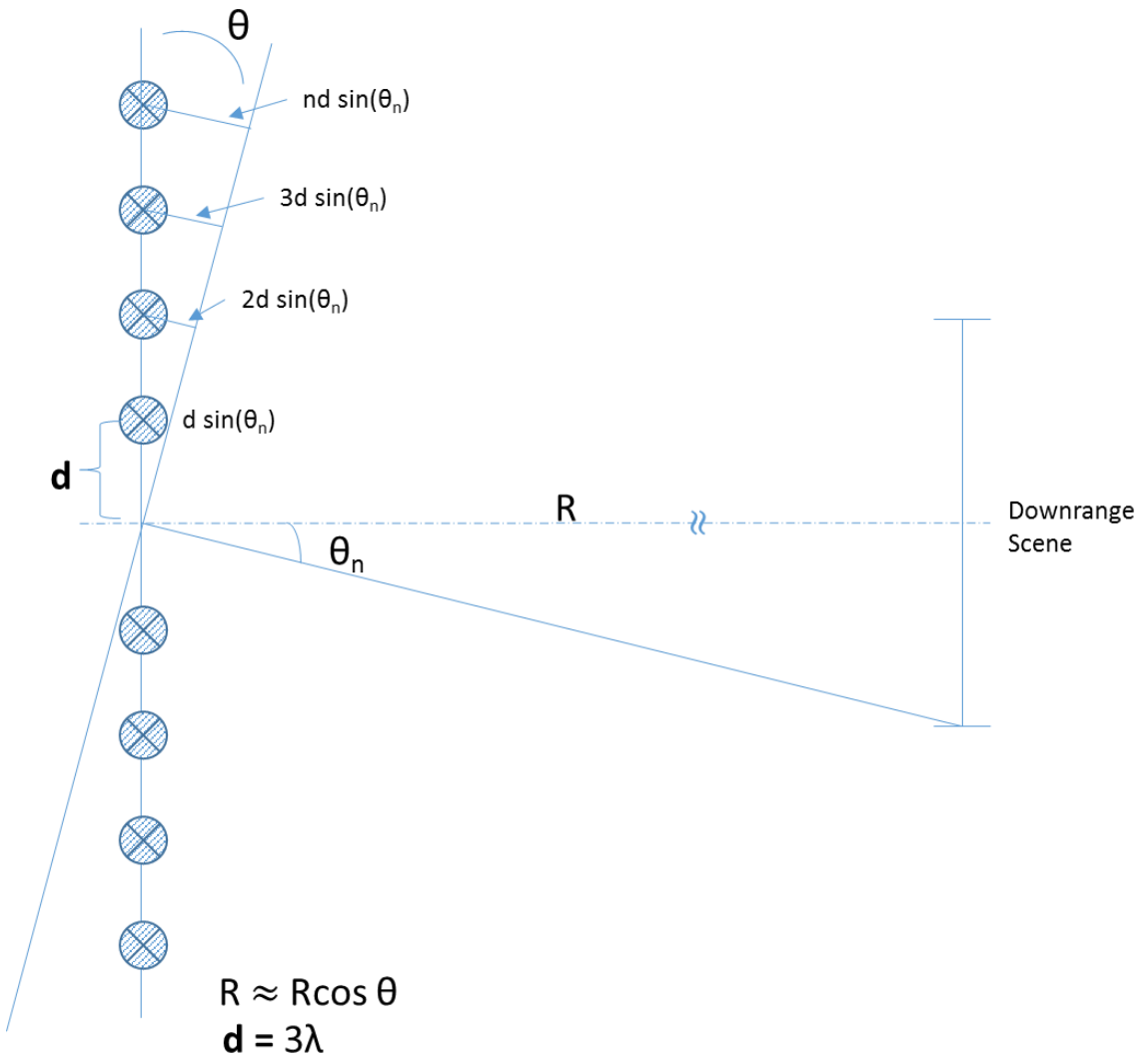


Figure-3: Linear phase slope

2.2 System Requirements

- REQF-001: Frequency Range
  - The frequency range of the imager should be within FAMU-FSU College of Engineering policies. The SAR will emit 10 GHz X-band energy at low power levels.
- REQF-002: Operating Range
  - The range from the imager to the target needs to be 20 feet.
- REQF-003: Extent of the Scene
  - The area that must be imaged should be the width of a normal sized person.
- REQF-004: Cross Range Resolution
  - The cross range resolution must be enough to discern whether there is a possible threat by showing there is large scatter on a specific portion of the body, but it need not be sharp enough to outline the particular type of weapon. The designed pixels size to divide the scene into would be 2.5 inches x 2.5 inches.
- REQF-005: Down Range Resolution
  - Down range resolution will be a future capability. For ethical reasons, the down range resolution should not be so high as to be capable of producing a high-resolution image of the human body.
- REQF-006: Pulse Width
  - The pulse width is nominally 20 ns since target (20 feet away) round trip is 40nS away as pulse travels 1nS per foot leaving 20nS for system to switch to receive mode at the end of the transmit pulse to detect the reflected pulse .
- REQF-007: Voltage Controlled Oscillator
  - The VCO used for converting the signal must be functional at 5 GHz.

- REQN-001: FCC Rules and Regulations
  - The Radio Frequency (RF) emitted from the SAR imager must be within ANSI/IEEE C95.1-1992 guidelines and FCC Rules and Regulations 47 C.F.R. 1.1307(b), 1.1310, 2.1091, 2.1093 regarding safe RF exposure for humans.
- REQN-002: Components Interference

- The Radio Frequency (RF) from the SAR imager should not affect communication between any other electrical components in the design.

- REQN-003: Interference
  - The Radio Frequency (RF) from the SAR imager should not interfere with common communications systems in the environment.
- REQN-004: Logic Implementation
  - The programing language used to communicate with the FPGA should be VHDL.

- REQM-001: Structural Redesign
  - The redesigned structure should be optimized for weight and stability. This includes a target weight goal of 80 lbs. Must still maintain rigidity. Increase the structure's mobility, preferably by introducing wheels to the structure.
- REQM-002: Horn Calibration
  - Horns must be adjustable in both azimuth and elevation angles. Horn's angular setting must be within 5 degrees. Tolerance must be less than 1/10th of an inch. Must focus within a circle of 1ft diameter from 20 ft. away.
- REQM-003: Component Box
  - Component box must dissipate the heat generated by the electrical hardware. Its weight and size must also decrease while being configured to the newly designed structure.

III. Design of Major Components

3.1 Structure designs

The design of the structure is strictly dictated by the geometry of the antenna array. As long as the structure can support the 20 antenna horns and hardware box, the secondary goal of reducing weight and cost was pursued in design.

3.1.1 Design S-1: 80/20 structure

The first design, Structure S-1, focuses around the use of 80-20, an industrial grade building structure and test platform as shown in **Error! Reference source not found.**. 80-20 is very modular due to its extruded aluminum profile and con be combined to other pieces through a variety of connectors. This design is also very flexible because different sized pieces of 80-20 with different channel numbers can be selected if more strength or surface area is desired.

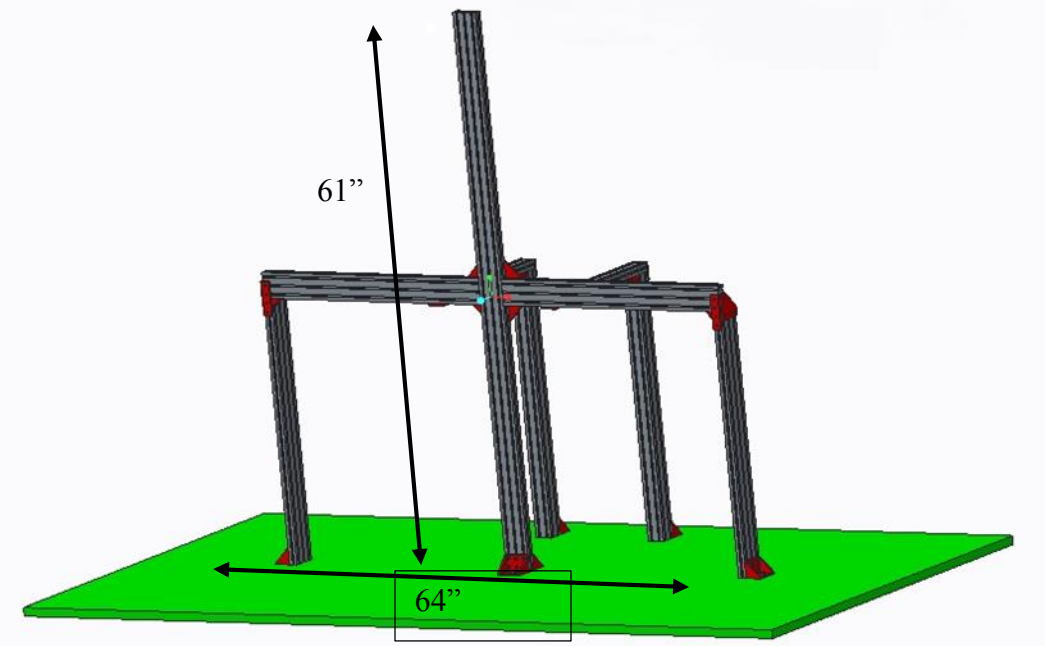


FIGURE-4: DESIGN S-1,3D (INCHES)

From the particular SAR radar array specified by Northrop Grumman, a 3x1channeled piece of 80-20 was used as the main vertical and horizontal bars which hold the antennas in place. Four angled brackets are used on the back of the structure to provide rigidity to the structure. This allows for near endless translation of the waveguide holders so that they can be aligned relative to each other. 80-20’s modular nature allows support beams to be attached anywhere. At the end of each horizontal beam, another 3x1 piece is used to support the far side. In order to keep the device from toppling forwards or backwards, two legs are added to each horizontal beam. This leg also serves to balance the weight of a central rear mounted control box if this location becomes specified by the EE team. The green base plate is an arbitrary ground; it shows how the structure would be mounted to a cart surface or floor with 45 angle brackets in red. Structure S-1 stands 64” tall and 61” wide from the extreme ends of the cross beams. The top of the 3” wide arm is 33.5” above the ground making the center exactly at 32” above the ground. The rear leg stands 19” away from the front and connects to the very bottom channel of the horizontal arm at 31.5” high.

3.1.2 Design s-2: Custom aluminum structure

Design S-2 features influences from last year's design or Generation 1 (Gen1) and is shown below in **Error! Reference source not found.**. Four pieces of Aluminum are bent or welded into an L shape and are attached together at their ends. The connectors at each horizontal end extend down to the floor to provide stability and weight relief to the center ground piece. Each waveguide adapter is sandwiched between two different pieces with a rectangular cutout placed in the proper distances for the antennas. There are four plastic gutters which protect and conceal the wires and are shown to be clear attached to the rear of the L beams.

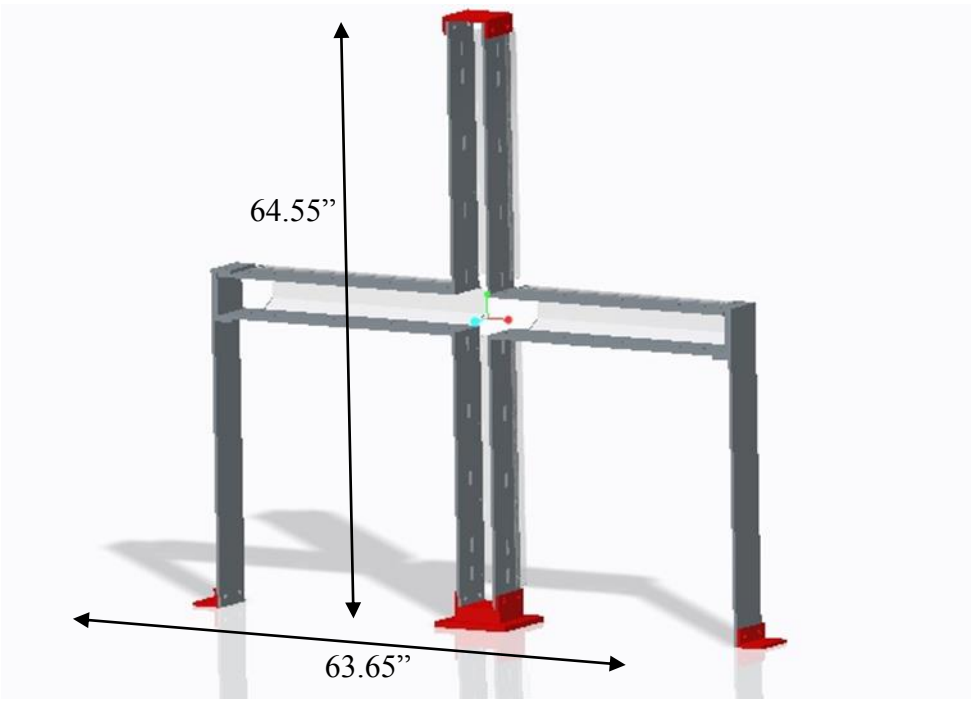


Figure-5: DesignS-2,3D (inches)

Detailed drawings of this design concept can be found in Appendix B. Each L beam is made of 0.375” Aluminum and is spaced 4 inches apart from each other to offer clearance for the waveguide to rotate freely without interference. The rectangle which anchors the waveguide adapter and rotation mechanism are spaced 1.5” x 0.5” to all some adjustment room to fine tune their translation. This structure stands 64.55” tall and 63.65” wide with each arm 29” long. At the side of the structure, each end cap stands 35.85” tall and 29.075” away from the downward side of the center. The inside of each gutter is 4.75” apart and 26” long so that it doesn't interfere with the end caps. The component box will be mounted to the back the horizontal sections of the L beams.

3.1.3 S-1, Version 2

While there were no issues with the stress analysis of the structure, additional components were added for convenience. The main horizontal and vertical bars were increased in thickness to accommodate the new horn holder design. There were additional horizontal bars added in the middle of the structure in order to act as something to grab in order to move the structure. The bottom of the structure was framed as well so that castors can be mounted. Figure 19 shows a rough representation of what these addition look like.

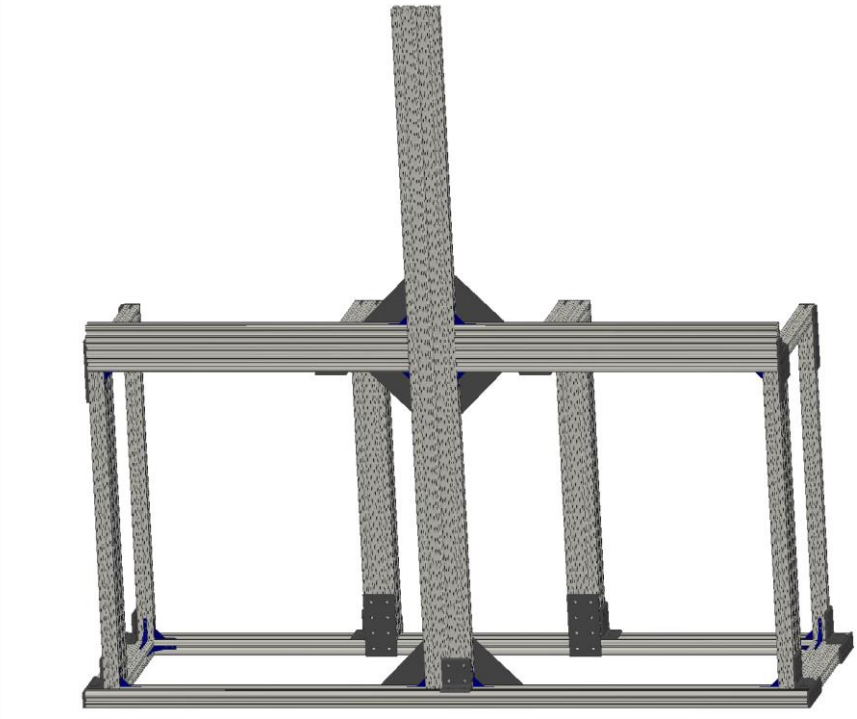


Figure-6 : Design S-1 V2

3.1.4 S-1, Version 3

A slight modification was added to Version 2 was to extend the bottom forward bar out from the structure. This addressed a few areas of concern:

- Sponsor requested a laser pointer based testing system that could be mounted to the structure. The bottom platform could be used to mount this to.
- Although tipping would not be a problem when stationary, the extended bar would ensure that if any unexpected forces were applied (i.e., in transit being rolled on wheels), there would be no risk of tipping
- More structure if design were to change

To account for this potential issue, the front of the base was extended 9” forward, while the cross remain in the same position with respect to the rack of the base. This also increase the wheel base depth to 30” which would increase stability of the structure rolling over a tilled floor as shown in Figure 7.1.3 below. A piece of 1545-8020 was added in the middle of the rectangular base to give the bottom of the vertical horn beam support. Due to the restructuring, different attachment plates T-slotted nuts can be used to secure the parts together. This results in a cost difference of $233.17.

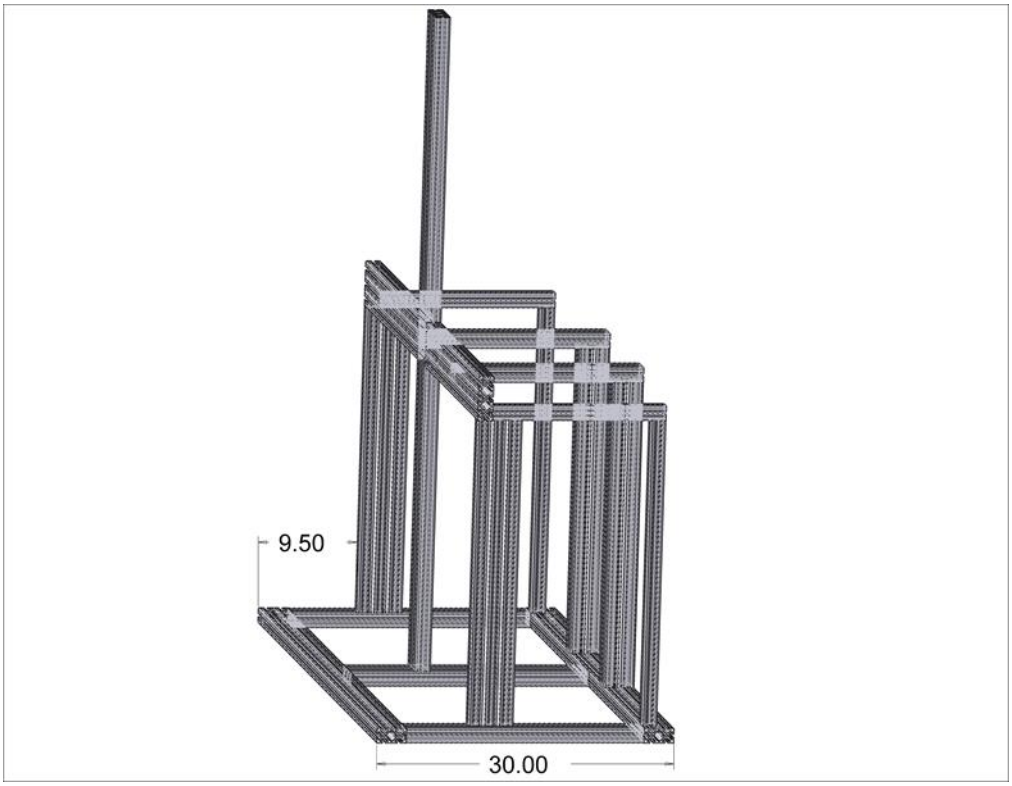


Figure-7: S-1,V3

3.1.5 S-1, Version 4

To quantify the improvements in weight achieved by this year’s project over last year, weight analysis of all the 8020 beams and needed attachment plates was performed. From the numbers given on the 8020 website, the weight of Version 3 was calculated to be 174lbs. This value was not a drastic improvement over the structure of last year at 220lbs. As a result, the structure was changed back to 10 series to get the weight closer to the 80lbs two person Military goal. This design iteration can be seen in Figure-7.

Figure-8: S-1,V4

One key addition to this design is moving the forward most chassis beam to directly under the Vertical horn holding beam. After talking with the electrical, having any metal in front of the antennas would cause radar reflections and produce inaccurate measurements. The 30" wheelbase was kept to keep the stability of the structure high. The weight of the simple handles on the sides of the structure was also reduced to a lighter series. Leveling casters were implemented as the preferred leveling and movement solution. Additional 45 deg cut pieces of 1010 Aluminum were order to give more rigidity to the structure.

3.1.6 S-1,Version5

In order to progress with our project, the physical prototype of Version 4 needed to be constructed. After the prototype was completed some key problems arose. Economy Tnuts can only be inserted in the end of the 8020. The nature of how some beams are joined together blocks of the end of some pieces. Slide in nuts from the top face of the channel should be used if any other 8020 beams are to be added later to give more rigidity. The Beam Analysis as performed in an earlier section came about due to the fact that the center beams were drooping down in the middle. Also upon the application of a force, the Rear chassis beam would vibrate in a different motion than the front beam. Going forward is it paramount that we connect these two different pieces to make sure the structure acts as one entire unit.

Even though the leveling casters were a nice in that they took care of the relative mobility of the structure and leveling portion, the small heavily grouted tiles in the A-Side of the Engineering Building did not go together well with the small 2” hard plastic wheels of the caster. During transit over this type of tile flooring, the entire structure would shake violently and even cause some hardware to fall out. All of these problems were taken car of in a final iteration Version 5 as shown below in Figure-8.

Figure-9: S-1, V5

The Front and Rear Chassis Beams are upgraded to 1020 with the second X channel in the vertical direction. Since the front and back have the same new height, the other beams do not need to be cut but just get raised one inch higher above the bottom of the structure than before. The leveling casters were replaced with 4” soft rubber wheels to give much better mobility over a variety of surfaces. As a result, new leveling hardware needed to be used since the leveling casters did not have a bigger wheel size.

To resolve the problem of not having a leveling option to the structure a two part solution was devised. First, a locking foot that could be locked in a used adjusted position was chosen to rise the rear of the structure above the casters. Next, Special Mitey Mount Anti Vibrational feet as shown in yellow were used to individually align the front of the structure on both the left and right sides. These feet are adjusted by a common socket wrench. The resulting tripod is much easier to adjust than the four leveling casters which is actually an improvement over Version 4. Also to connect the Front and Back chassis beams, a 18" piece of 1030 is used, which conveniently doubles as extra mounting holes for the lockdown foot. Finally a decent amount of milled 45 degree 1010 beams were added to the stricter to resist the Vertical Horn Holder Beam vibration and moment of inertia. These parts have yet to be included on the new structure since the electrical team is performing last minuet tests but they have been delivered to the Engineering College.

3.2 Horn Holders

3.2.1 Legacy Design

The horn holders act as the housing and the adjusting platform for the horns which emit and receive the radio frequency signal. It is imperative that this action functions properly and that the error in the accuracy should be minimized. The horn holders should allow the user to seamlessly calibrate the horns, secure their position when required, and be able to adjust them if needed. The horn holder design from the 2014-2015 team is in need of a complete redesign. This design only allowed for one degree of freedom; Northrop Grumman has stated that the horn holders must possess two degrees of freedom (azimuth and elevation).

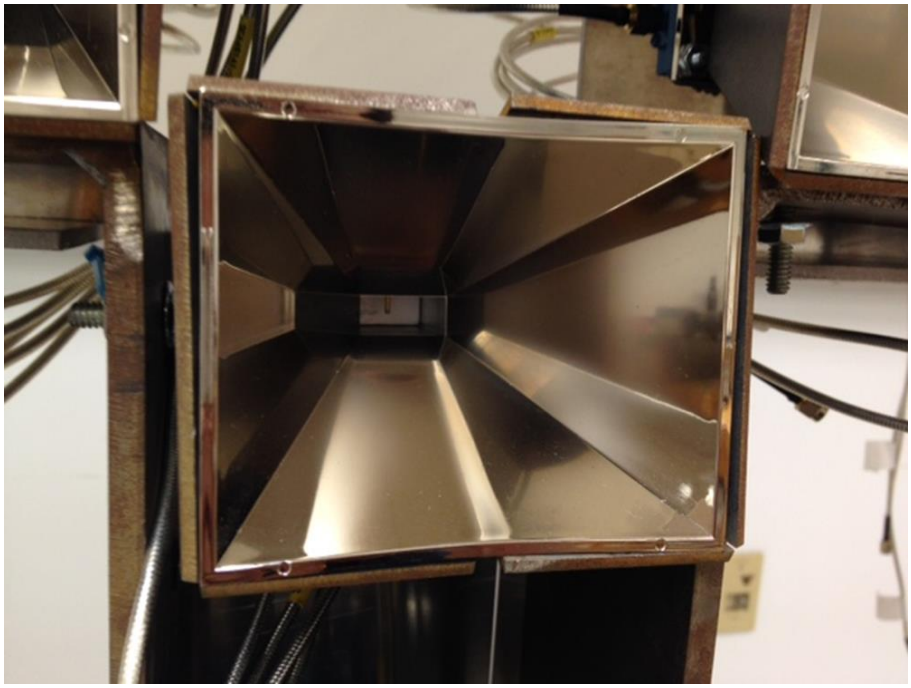


Figure-10: Gen 1 horn holder proof that the design caused deformation along the bottom of the horn

Above in Figure-10 the last year's horn holder is displayed. This shows how the holder is a shell that is screwed into the antenna structure. The horn sits within the shell and is rigidly attached to structure once the shell is screwed in. The horns in the horizontal array can only have the azimuth angle adjusted whereas the horns in the vertical array can only have the elevation angle adjusted. This limitation in movement is due to shells fitting along the ¼" steel -frame.

Another point of concern lies within the actual adjustment of the horns. To adjust the angle of the horn, one must unscrew the shell from the frame, tilt the horn to the desired position, and then screw down the shell to the structure. The torque exerted by the screwing motion is significant enough to potentially move the shell from its set position. This results in a tedious calibration process and detracts from the effectiveness of the design.

Lastly, the last year's horn holder design is presenting a compressive force against the horn in order to keep it in place. This is problematic because, in some cases, the horns are deforming due to this force. As shown below in figure-10 one can see that the bottom lip of the horn is deformed. This is not ideal for transmitting or receiving a radio frequency because the warped shell can potentially distort the trajectory, thus skewing data or making it difficult to hit the desired target. It is also possible for the horns to crack. Since they are made of an inexpensive plastic material, the horns lack the strength to withstand strong compressive forces.

3.2.2 Horn Holder Concept H-1

To amend these issues with the current design, Mechanical Engineering team #18 has been working on multiple concepts, alongside the two mechanical engineers on our team. Three preliminary designs were generated, all of which address the concern of being able to adjust the horns with two degrees of rotational freedom. After deliberation with Northrop Grumman, one concept was selected as a preferred design and will now be discussed in further detail.

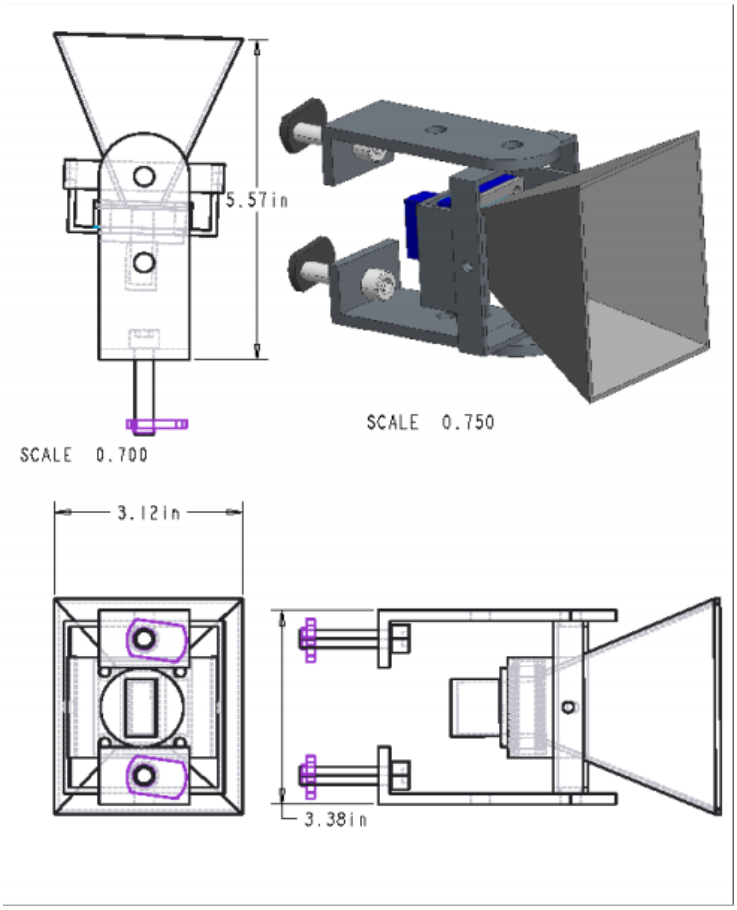


Figure-11: Cad image of the preliminary design iteration for redesigned horn holder

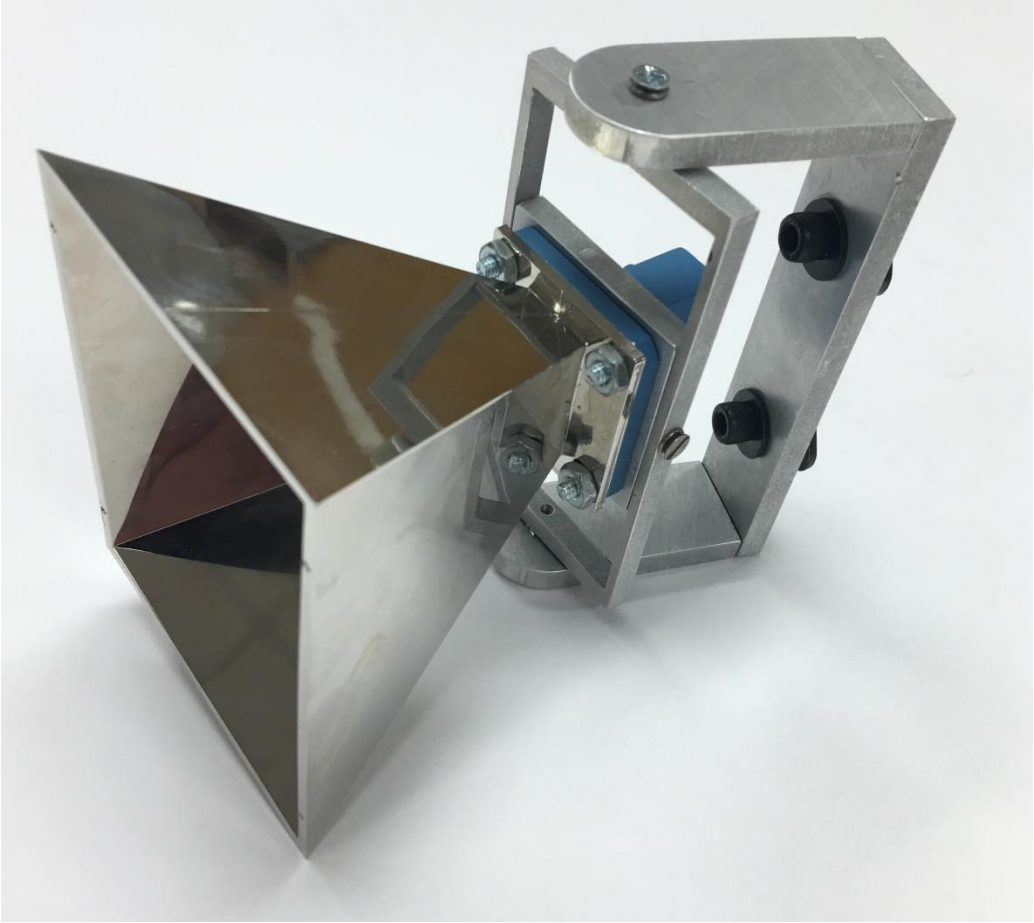
Displayed above in Figure-11 is one of the preliminary horn holder designs. This concept allows for the two degrees of rotational freedom needed, something not achieved in the current design. Its fulcrum point is located near the center of the horn so that adjustments to the angle will be as accurate as possible while calibrating the horn. This is contrasted to as if the fulcrum point where located at the rear of the horn, making micro adjustments difficult because a minimal rotation at the rear of the horn would translate into a large rotation at the front of the horn. This design also allows for a laser to be attached, making for

a more efficient calibration process. Further, the design is to be compatible with the 80/20 fasteners allowing for easy mounting of each horn. This is because, the structural redesign will utilize the 80/20 extruded aluminum.

However, there were still some design flaws with this iteration. The main one being that the fulcrum point, although ideally sound due to the adjusting closer to the center, would cause for interference when tightening down. Because of this, the design was iterated once more so that the fulcrum point was pushed back towards the wave guide, near the rear of the horn. Since the calibration will be done with an attached laser, it is not necessary to have the pivot point at the center of the horn. The calibration will simply follow the laser's sight. This will ensure both accuracy of the horn's position as well as simplifying the securing mechanism's adjustment. Thumbscrews were added to the design so that a tool would not be needed, this will decrease the adjustment and calibration process.

3.2.3 Horn Holder Concept H-1 Version2

Design H-1 has been modified slightly to be fully compatible with the updated structure iteration. The two 'L' brackets have been replaced by one solid bracket to provide more assurance to the holder's strength. To secure the azimuth and elevation positions, four combinations of a wing bolt, star washer, and lock washer will be used. Recently, the ideal distances between the horns for optimal performance were received from the electrical team. To satisfy those distances, the width of the outer bracket piece was reduced so that there will not be any clearance issues. The shortest distance between horns will be between the transmitter and adjacent receiving horns. To be sure that there will be no clearance issues between these horns, smaller thumb bolts will be used instead of the wing bolts. A total CAD model of this final assembly is shown in Figure-12. The material selected was Aluminum 6061 and once assembled and connected with the horn and wave guide, the weight was approximately 1.5lbs. Also, the cost of one horn holder came out to be $15 which of course would go down if more were to be manufactured.



Figure–12: Final horn holder design

3.3 Component Housing

3.3.1 Legacy Component Housing

The component box houses all of the electrical components with the exception of a laptop and a monitor screen. The box will mount on the back of the radar structure, making it a self-contained system. The weight of the box should be minimal while maintaining easy access to all components for testing purposes. The team from last year finalized the housing as seen in figure-13 which was designed in an "L" shape in order to sit almost vertically on the structure. They decided to leave the housing exposed to the ambient air to allow for sufficient cooling. Made of both steel and aluminum, this design was too heavy and bulky which only added to the immobility of the structure and its size was larger than necessary. The housing attaches to the frame with a slot and pin-type system where rectangular pins extrude from the frame. The matching rectangular slots on the box allow for the two to connect. This design relies on

friction and gravity with no real locking mechanism, making it potentially dangerous to the radar itself and the users around it.

Figure-13 : Image of legacy component housing

3.3.2 Component Housing C-1

A complete redesign for the component housing was constructed that not only reduced the weight and increased mobility, but integrated safety, protection of electrical components, electromagnetic interference shielding, and eased electrical connections. Early design iterations consisted of an open flat tray that would be able to sit on the structure to really minimize weight and allow for easy access to the electrical hardware but this lacked a securing feature for the hardware as well as a lack of electromagnetic interference shielding. As shown below in figure-14 is the final design iteration for the component housing. The design philosophy for the housing was to firstly have ample space for testing to take place and for a user to be able to access any component and simplify testing, and secondly to be as enclosed as possible while still allowing for adequate heat dissipation. The reason behind constructing a sealed enclosure was to minimize the amount of electromagnetic interference between the outside environment and the electrical components inside, as well as being able to safely contain the electrical components and

have the ability to lock and secure them. Further, a conductive elastomer was utilized along the inside of the housing lid in order to ensure that any gaps in the housing would be properly sealed with an appropriate electromagnetic gasket to again minimize interference.

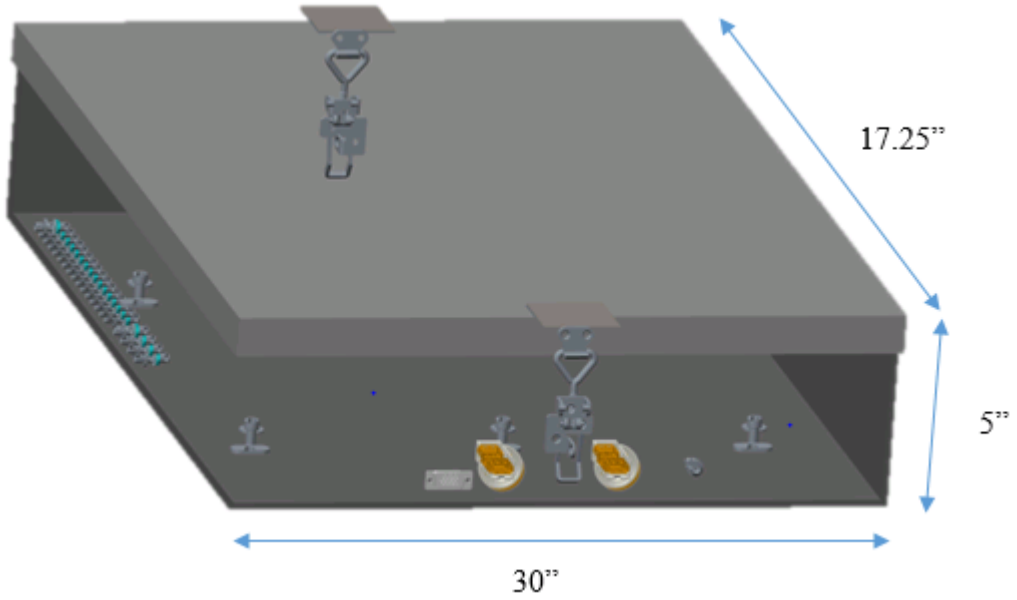


Figure-14: CAD image of final component housing design iteration

The first difference compared to last year’s design is the fact that this is now an enclosure and acts as a black box where any inputs and outputs are configured on the exterior of the component housing so that the enclosure can remain sealed and does not need to be opened or exposed in order to operate. The external panel mounted connectors consisted of one VGA port for the FPGA, two USB connectors for the FPGA and the VCO, one power connector and twenty SMA connectors. Also visible from figure-14 are the pad-lockable latches on two sides which will allow for the housing to be locked, again ensuring that the electrical components cannot be easily tampered with by an unknown user. The electrical components are mounted on a drop-in tray that sits on risers within the enclosure; this allows the components to be horizontal and thus easier to work with.

The housing’s dimensions are 30” x 17.25” x 5” with 0.09” thick Aluminum 6061. This resulted in a huge weight reduction to 12.9 lbs before electrical components were installed. This decrease in weight in turn increases the mobility. The box is able to sit and bolt onto the rear legs of the redesigned structure

which allows for the housing to be more secure and improves on safety from last year with a more permanent mounting design. The housing is shown mounted onto the structure below in figure-15

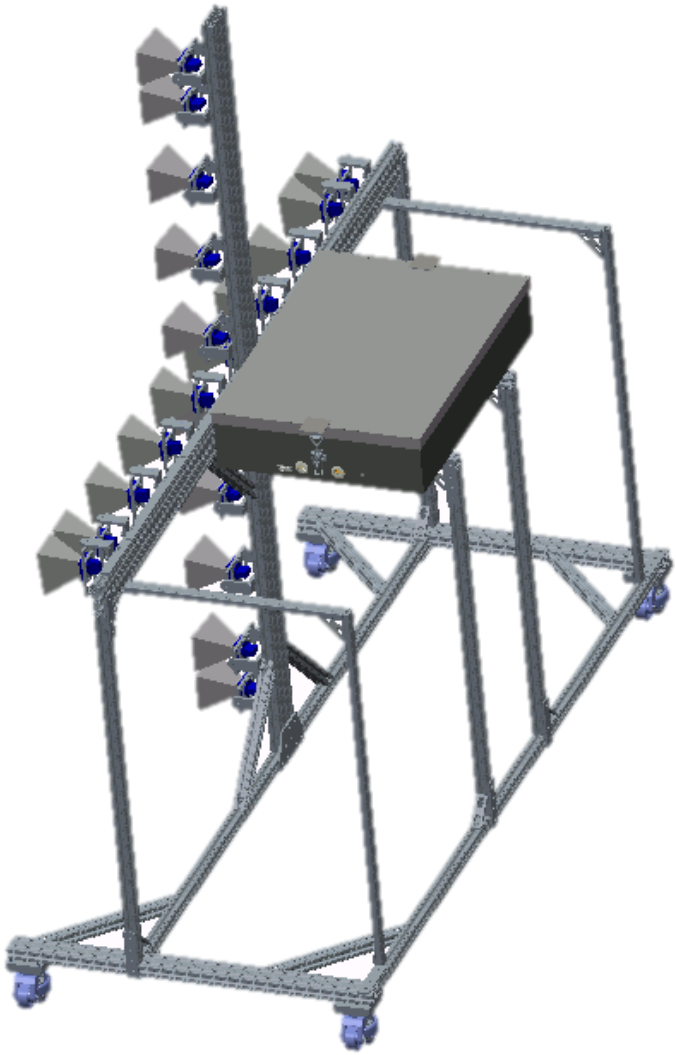


Figure-15 : Image of component housing mounted onto structure

Once the design was finalized, the manufacturing drawings were taken to the machine shop so that the sides of the housing could be first water jetted. After the pieces were appropriately cut, they were to be welded for assembly. Lastly, the housing was powder coated blue and then the panel mounted connectors were installed along with the fasteners needed for installation. Shown below in figure-16 is the component housing completed along with the rewired and installed components sitting on the drop in tray in figure-17.

Figure-16 : Completed and assembled component housing

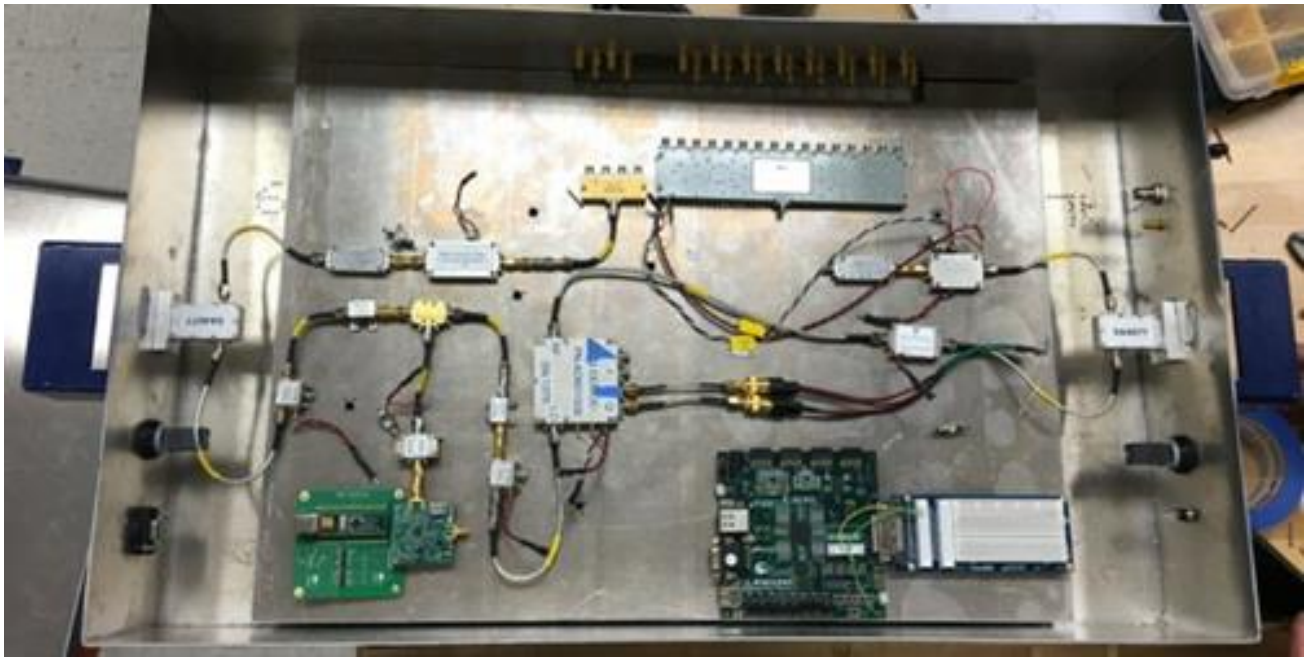


Figure-17: Rewired and installed electrical components

3.4 Component Housing Thermal Analysis

To begin thermal analysis on the component box, the first step was to determine the amount of power supplied to each component to understand how much heat must be dissipated. In order to find this value, the power supplied to each component was found by multiplying the rated current and voltage of each component and then summed together as a whole. This is shown below in Table-1

Table-1: Power supplied to electrical components

Component	Area (ft2)	Voltage (V)	Current (A)	Dissipated Power (W)
VCO	0.174	3.300	0.045	0.149
FPGA	0.278	3.300	0.200	0.660
SPDT	0.005	5.000	0.001	0.007
SP4T	0.010	5.000	0.160	0.800
SP16T	0.089	5.000	0.550	2.750
IQ DEMOD	0.075	5.000	0.110	0.550
SUPER ULTRA WBA	0.005	12.000	0.400	4.800
ULTRA WBA	0.005	12.000	0.400	4.800
LOW NOISE AMP	0.012	12.000	0.320	3.840
POWER AMP	0.017	15.000	1.100	16.500
Total	X	X	X	34.856

By adding each electrical component's power supply, we received a value of 34.8 W. Next, obtaining the inside surface area of the box will allow us to obtain the heat flux by dividing the input power over the total inside surface area. As of now, the calculated inside surface area is 10.47 ft^2. This introduces a heat flux value of approximately 3.324 W/ft^2. Using figure-18 as shown below, one can obtain the rise in temperature inside the enclosure by assuming an "unfinished aluminum and steel enclosure" giving us a value of approximately 16.65℃. This concludes that the temperature inside the enclosure is about 16.65℃ higher than ambient temperature, which is taken as 21℃. This translates to an enclosure temperature of about 37.65℃ during steady state operation.

This is shown in purple in Figure figure-18 below. This however changed once the component housing was powder coated and thus was considered a "painted metallic" and therefore due to the unfinished metal's inefficient radiation heat transfer, more heat is radiated once the enclosure is painted. This results in an enclosure temperature rise of about 9.5℃. Assuming room temperature, the final enclosure temperature reaches approximately 30.5℃. Since the most sensitive component's failure temperature is (50℃) we are able to safely determine that the enclosure is constructed so that the heat can effectively dissipate to allow proper functionality for the electrical hardware without risk of overheating.

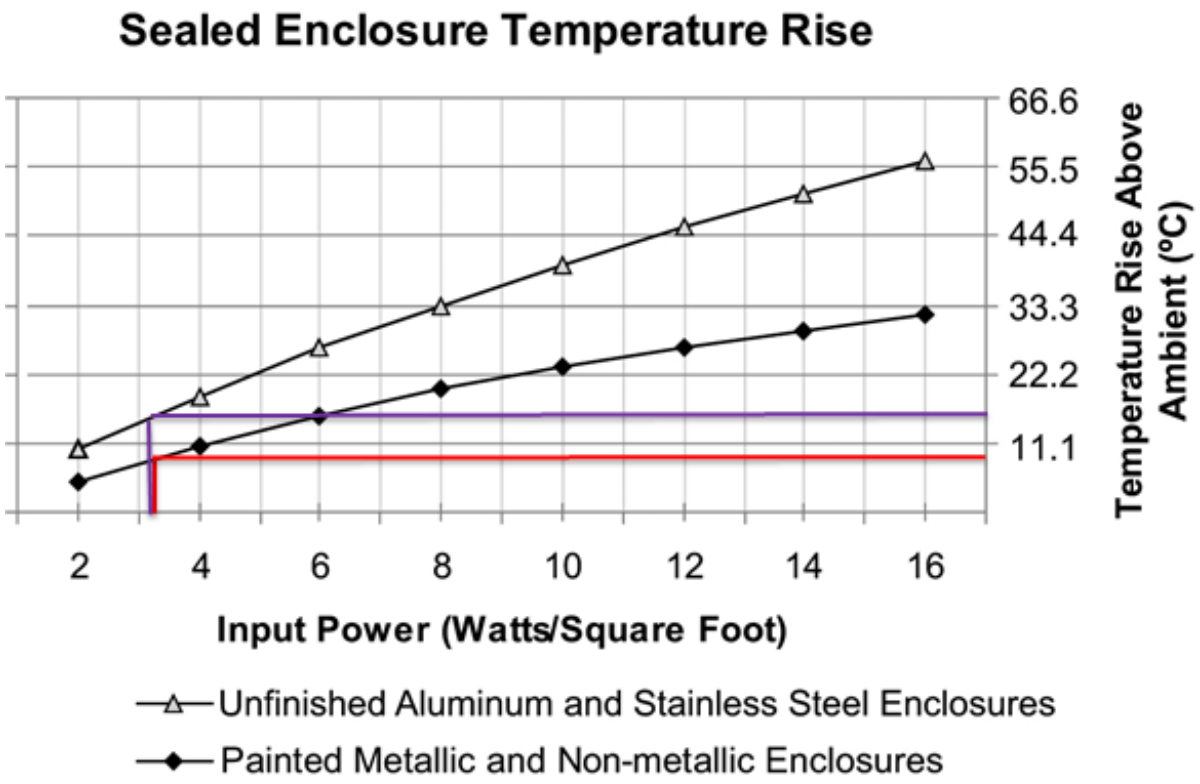


Figure-18 : Temperature Rise within Enclosures

3.5 Transmission Subsystem

3.5.1 Purpose

The Transmission Subsystem generates, amplifies, and transmits a 10 GHz RF pulse.

3.5.2 Components

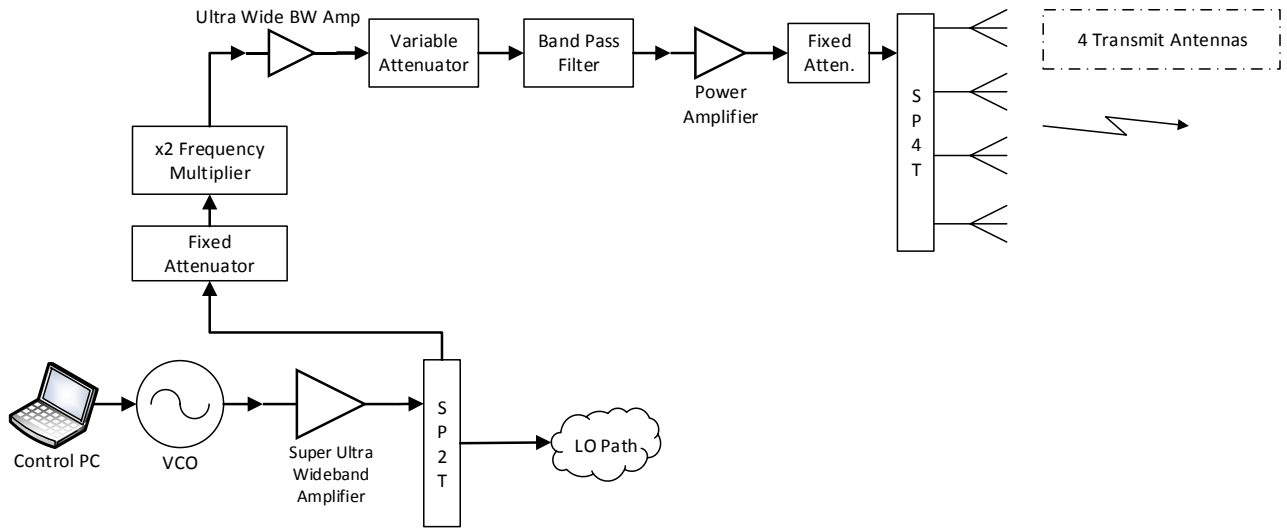


Figure-19: Transmission Subsystem components

The state of the SPDT switch determines when the system is in Transmission Mode. When in Transmission Mode, the 5 GHz signal from the VCO is amplified, multiplied in frequency to 10 GHz, and then amplified twice more before transmission. The fixed attenuators are in the design to prevent the frequency multiplier and the switches from being over-driven. The variable attenuator provides fine-tuning capability before the final amplification.

3.6 Reference (LO) Subsystem

3.6.1 Purpose

The Reference Subsystem provides the IQ Demodulator Local Oscillator with a 10GHz phase-locked signal.

3.6.2 Components

The Reference Subsystem is identical to the Transmission Subsystem until the final stage of filtering and power amplification that the reference path does not do. This is by design, since the purpose of the signal is to compare with the returned signal, drift from the transmitted signal should be minimized. This is why the Hittite Phase Lock Loop (PLL) VCO was selected. An internal feedback in the VCO corrects any phase drift in the signal.

The state of the SPDT determines when the Reference Path is active. During this time, the system is simultaneously receiving reflected signals, so this is referred to as Receive Mode. In all SAR

documentation, “receive mode” should be interpreted to mean when the Reference Path and the Receive Path are active, unless expressly stated otherwise.

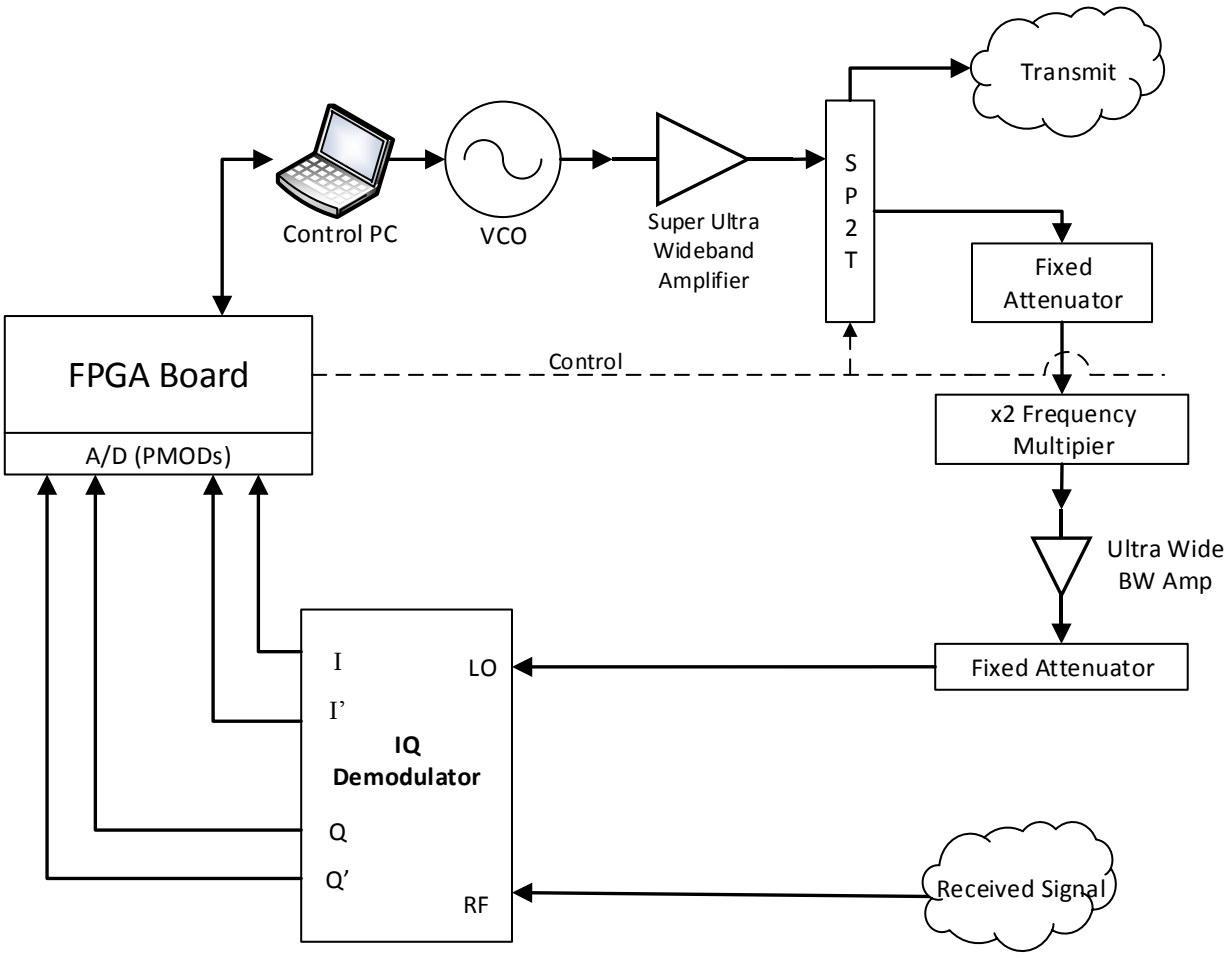


Figure-20: Reference Subsystem

3.7 Receive Subsystem

3.7.1 Purpose

The Receive Subsystem collects the reflected signal from the downrange target, amplifies the signal, and routes it to the RF port of the IQ Demodulator.

3.7.2 Components

After being received by the antennas, a bandpass filter removes extraneous signals not near the 10 GHz desired frequency. Two Low Noise Amplifiers (LNA) amplify the received signal to usable levels. The variable attenuator prevents the first LNA from compressing the second LNA, and allows for precisely tuning the receive level needed to the IQ Demodulator.

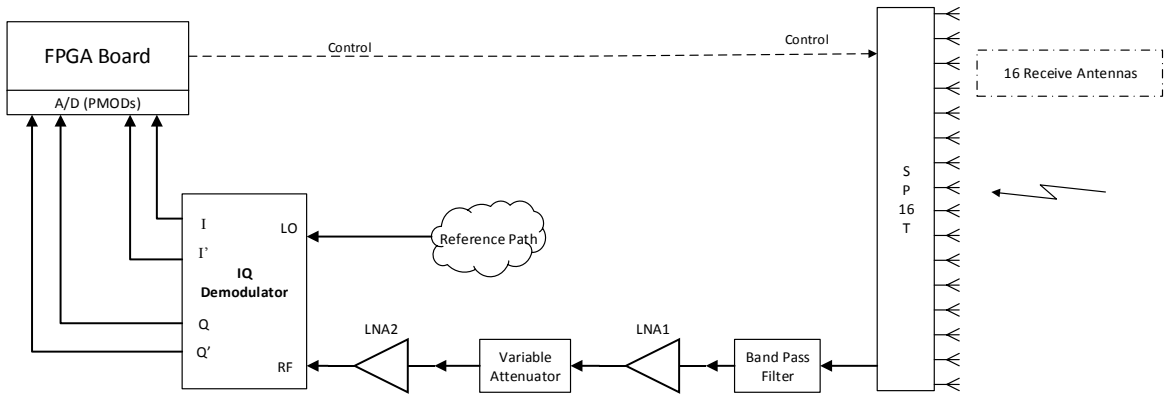


Figure-21: Receive Subsystem

3.8 System Control (FPGA)

3.8.1 Purpose

SAR radar is a synchronous system, which makes it important that a processing machine control the components that makes up the radar. The processing machine used in the system is the NEXYS3 Digilent board. This FPGA board is equipped with enough I/O devices and ports to host a wide variety of digital systems. Very High Speed Description Language (VHDL) language is used to program this board. The speed of this devices along with its extended resource documentation made it suitable for the system.

3.8.2 Requirements

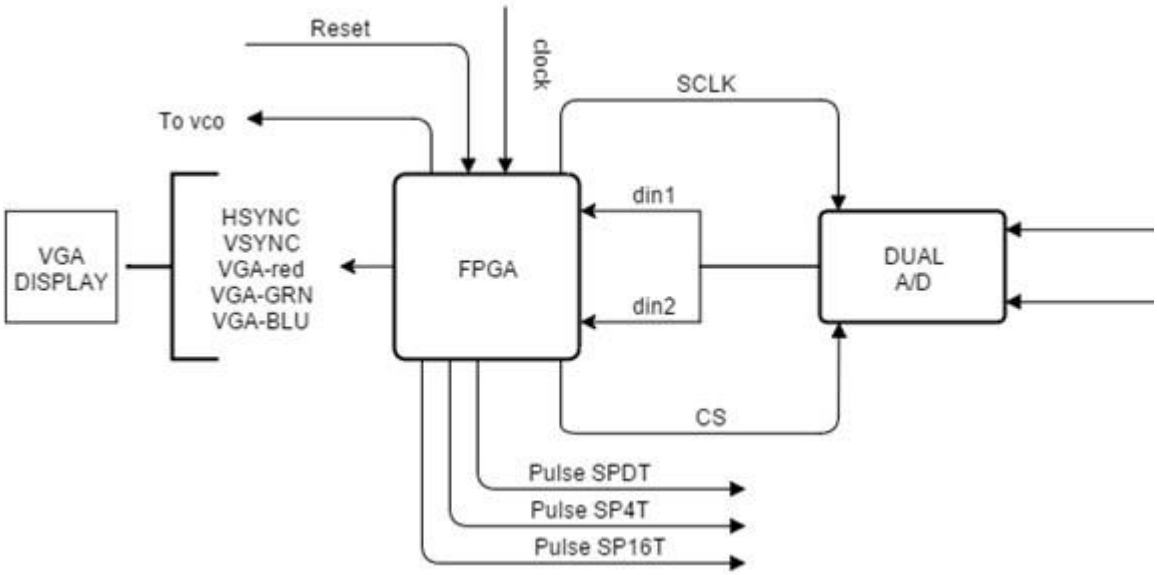


Figure-22: Top-level design diagram

The FPGA is required to control the Single pole double switch (SPDT). This switch activate the transmitting path and the receiving path. Since the radar has to switch between transmitting and receiving path in 60ns, this switch require a high-speed pulse to drive it. Each component in a radar is required to be only active for a certain amount of time. In addition, the FPGA control the switching of the transmitting antenna and receiving antenna by sending signal to the SP4T and SP16T respectively.

In addition to controlling the switches, the FPGA is required to take in positive and negative voltages from the In-phase and Quadrature-phase demodulator (IQ Demodulator). This is done by sending signal to control the Analog to digital converter device purchased from the Digilent store.

Finally the FPGA will use its processing power to implement a Fast Fourier Transform in order to process the incoming signal from the scene. This signal processing ouput will be display on the VGA monitor which is also controlled by the FPGA.

A complete diagram of the system design is offered in Figure-22.

1. Method

Figure-23: A/D converter from Digilent

In order to accomplish the requirements of the system control, the team have completed different coding assignment that control a specific components in the radar. At the end, all the code where converged into one code that controlled the whole system.

Analog to digital:

The PmodDA4 ADC device from diligent are used to communicate with the NEXYS3 board. This component is shown in figure-23. Two ADC chips were acquired and placed between the IQ demodulator and the FPGA. The PMOD connectors are used to interface with the chips. These ports are used for low frequency clock signals. This ADC chips digitalize the voltage range from 0 to 3.3 volts into a 12 bits binary value. The code was tested using a DC voltage supply. The DC voltage where able to digitalize and displayed on the 7-segment display. This code was improved from the 2014-2015 code. The conversion formula between DC voltages to HEX is as follow.

$$V_{Hex} = \left(\frac{Voltage}{3.3}\ X\ 4095\right)_{Hex} \quad eq(3.7.3.1)$$

SP4T, SP16T, SPDT Switching logic:

The SPDT is responsible to activate the transmit path or activate the receive path. This switch need to be transmit for 20 ns where a pulse will be sent out of the transmitting antenna. 100 pulse is sent out of the transmitting antenna this will help build up the DC voltage in the IQ demodulator in order to compare with the receiving signal. Since the signal that drive the SPDT is a high speed signal, the VHDCI connector of the FPGA board had to be used. In addition, the SP4T control which antenna is transmitting at the time and SP16T switches control which antenna is receiving; this components were connected to the PMOD converter because their signal is a low speed signal. A complete signal diagram is presented in appendix B. A strip down version of this diagram is showed in figure 3.7.3.2 where the relation between SPDT, SP16T is SP4T is pictured. In the figure, the receive window is 40 ns because the signal need to propagate through the air reflect from the metal and propagate back to the receiving antenna.

VHDL library

In order to make the code modular, a VHDL library was implemented. The library contain components that are used to perform different tasks. An explanation of the different components follows.

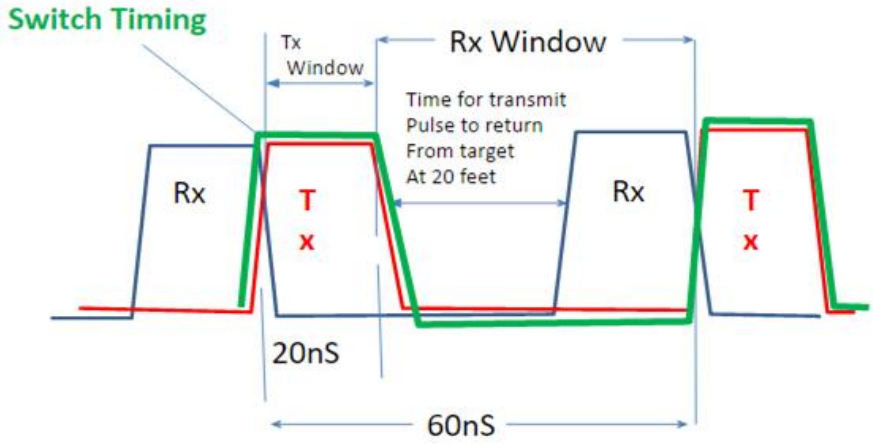


Figure-24: SPDT Switching timing

1. ADC components

```
--8. Analog to digital conversion with negative reference
COMPONENT A_TO_D
PORT(
   CLK : IN std_logic;
   RST : IN std_logic;
   SDATA1 : IN std_logic;
   SDATA2 : IN std_logic;
   SDATA3 : IN std_logic;
   SDATA4 : IN std_logic;
   START : IN std_logic;
   SCLK1 : OUT std_logic;
   nCS1 : OUT std_logic;
   SCLK2 : OUT std_logic;
   nCS2 : OUT std_logic;
   DONE1 : OUT std_logic;
   DONE2 : OUT std_logic
   );
END COMPONENT;
```

Figure-25: ADC components

The ADC components picture in figure-24 control two ports of the PMOD (Bank A, and Bank B). This components takes as input a 100 MHZ sample it to make a 25Mhz clock that are used to drive the chips. The 25 MHz clock is assigned to SCLK1 and SCLK2. The code sample the input voltage to SDATA1, SDATA2, SDATA3 and SDATA4. When done this code activate DONE1 and DONE2 to let the users know that program is done. This is a good practice as it help to implement the final code.

2. 7-segment Driver components.

```
component segdriver is
Port  (
    CLK     : In std_logic; --100 MHZ
    RST     : In std_logic; --button(0) B8
    sig_in  : In Std_logic_vector(15 DOWNTO 0);
    seg     : out STD_logic_vector(6 downto 0);
    an      : out std_logic_vector(3 downto 0));
end component;
```

Figure-26: 7-Segment driver

The driver accept a 16 bits binary in port sig_in and output the hexadecimal value on the 7 Segment display.

3. Memory driver

```
component memdriver is
PORT
   (
   clk_i    : in std_logic;
   RST      : in std_logic;
   btn      : in std_logic;

   adr_in   : in STD_LOGIC_VECTOR(22 downto 0);

   MemAdr : out std_logic_vector (23 downto 1);    -- Address
   RAM_OEb  : out std_logic;                       -- Output Enable
   RAM_WEb  : out std_logic;                       -- Write Enable
   RAMAdv  : out std_logic;                        -- Address Valid
   RAMClk  : out std_logic;                        -- RAM clock
   RAMCre   : out std_logic;                       -- Control Register enable
   RAM_CEb  : out std_logic;                       -- Chep Enable
   RAM_LB   : out std_logic;                       -- Lower Byte
   RAM_UB   : out std_logic;                       -- Upper Byte
   MemDB_io : inout std_ulogic_vector (15 downto 0);  -- Bidirectional data
   MemDB_in : in  std_ulogic_vector (15 downto 0);
   MemDB_out: out std_ulogic_vector  (15 downto 0)
   );
end component;
```

Figure-27: Memory driver

The FPGA board has 16 Mbytes RAM. This components allow the user to save and read the value into the memory. When BTN equals zero, the driver allows the user to write data to the memory. Adr_in takes in an address, and MemDB_in take in the data and put it at the specified address. When BTN equals 1, the driver allows the user to write to the output.

These components are easy to initialize when the library is declared. In order to use the library, use:

Copy and paste the SAR_LIB located in the school website/ the project flash drive into the project folder. In the ISE driver go to project>>New VHDL Library. Fill in “New VHDL library name” box, for example my_lib. Locate the folder SAR_LIB. Click ok. In the VHDL, write

```
Library my_lib;
USE my_lib.sar_design.all;
```

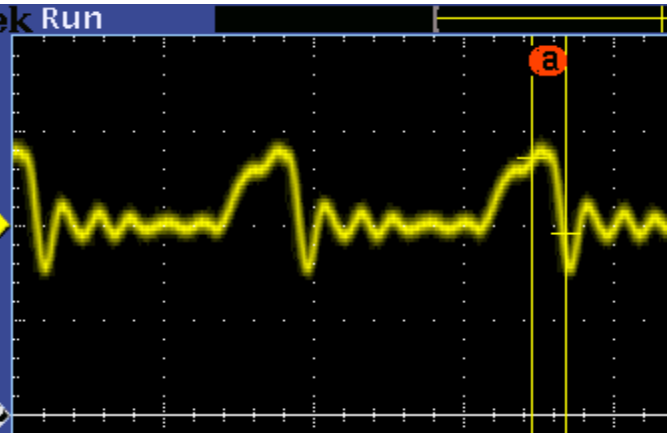
Figure-28: Declaring library

3.8.3 Challenges

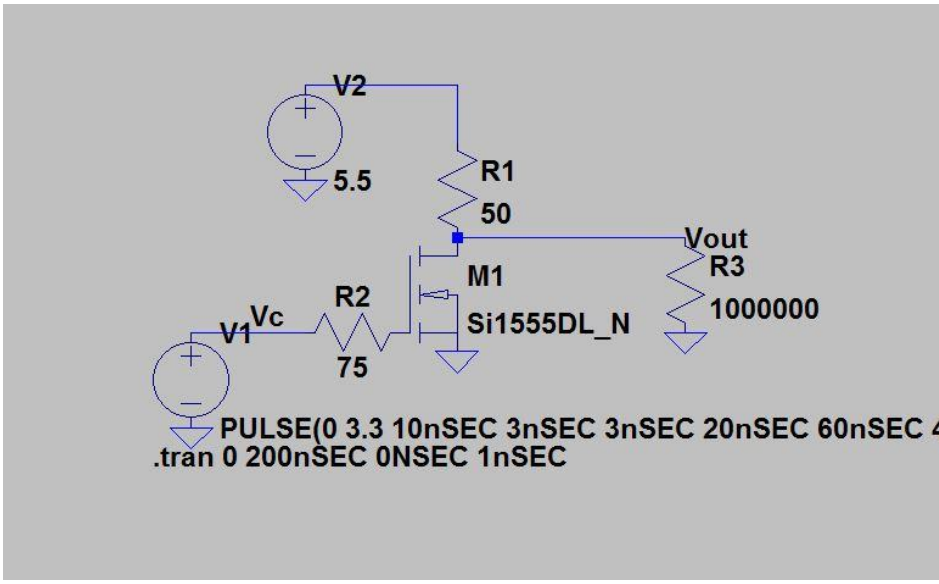
Before the completion of this program, many obstacles needed to be overcome. Those obstacles forces alternative path to be made. First the ADC voltage only digitalize voltage between 0V and + 3.3 V. The IQ Demodulator on the other hands voltage range are from -300 mV to +300 mV. Therefore, negative voltage were not able to be digitalize. Using a step up circuit will add noise and will require additional

circuit to be made. In order to solve this problem, the I-bar and Q-bar channel of the IQ demodulator are used. Using the fact that I and I-bar channel are complex conjugate, when one is positive the other is negative, a logic circuit was able to be implemented using VHDL. This circuit was implemented and tested. The output was a 13 bit binary where the 13th bit represent whether or not the value is positive or negative. Although this circuit was able to be implemented, it was not used in the completion of the coding because, all 4 channels the IQ demodulator value (I, I-bar, Q, andQ-bar) were saved to the memory and exported to MATLAB.

Another important challenge, was the fast pulse signal that drives the SPDT. A square wave with 0V low and at least +3.1V was required to switch the SPDT. When using the VHDC connectors, at high speed the signal become distorted and the +3.1V was not always outputted see Figure 3.7.4.1.a. In order to solve this, a circuit was used. This circuit is pictured in Figure 3.7.3.4.1.b. When no voltage from the FPGA is applied, the MOSFET is in cut off region and the output is +5.5V. On the other hand, when a voltage is applied by the FPGA, the output voltage is +0 V. A waveform of this characteristic is shown on the Figure 3.7.4.2. The blue line represent output from the FPGA, and the green line is the output of the circuit.



a)



b)

Figure-29: a) Output waveform from VHDC connectors b) Circuit

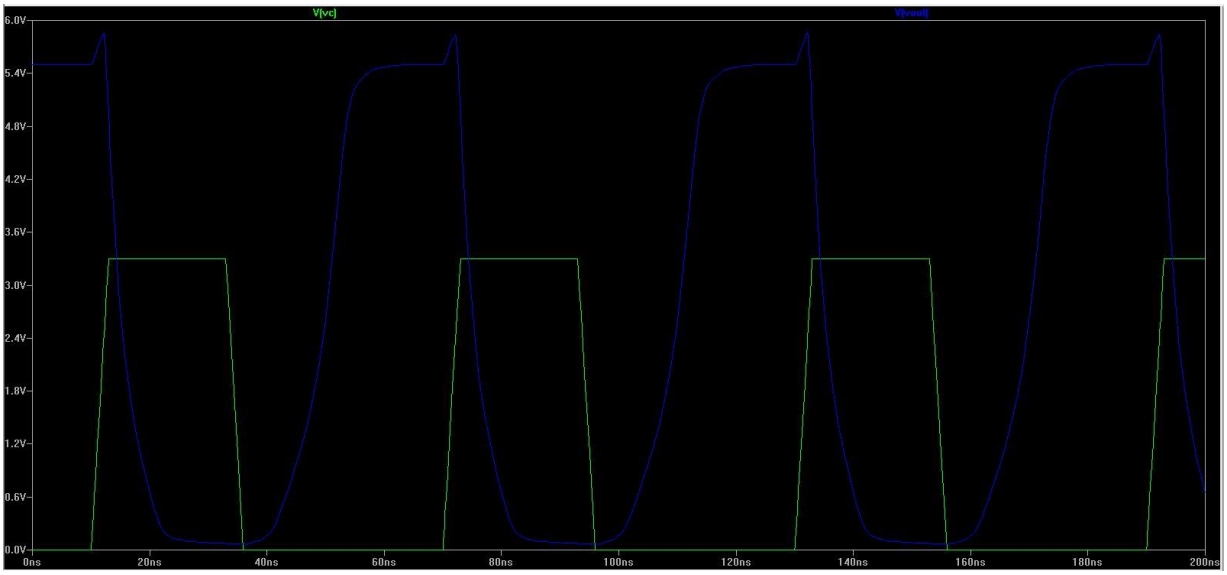


Figure-30: LT SPICE output waveform

Different generation of this circuit was made, but the circuit was not used in the completion of this project. Instead a waveform was used using the waveform generator.

3.8.4 Results

A final code was implemented. TOP_range is the name of the project file and can be found on the project flash drive. The code consisted of a Moore finite state machine that controlled the antenna. This code allows the radar to work in two modes of operation (Debug mode and Real time mode). In debug mode the FPGA switches are used to control which transmit/ receive pair is activated. The sequence of the antenna along with the combination of switches (SW) that activate the radar is presented in the table 3.7. 5.1. TX1, TX2 and receive 1 to 8 are located on the X-axis of the antenna. In this version of the SAR radar, only the X-axis antenna was used in order to make the system simpler to troubleshoot. When in debug mode the user is allows to press the push button on the FPGA to see the DC voltage input of the I, Q, I-bar , and Q-bar channel.

Table-2 : Switching Control

Transmit	Receiver	**SW5**	**SW4**	**SW3**	**SW2**	**SW1**	**SW0**
TX1	1	0	0	0	0	0	1
	2	0	0	0	0	1	0
	3	0	0	0	0	1	1
	4	0	0	0	1	0	0
	5	0	0	0	1	0	1

	6	0	0	0	1	1	0
	7	0	0	0	1	1	1
	8	0	0	1	0	0	0
TX2	1	0	0	1	0	0	1
	2	0	0	1	0	1	0
	3	0	0	1	0	1	1
	4	0	0	1	1	0	0
	5	0	0	1	1	0	1
	6	0	0	1	1	1	0
	7	0	0	1	1	1	1
	8	0	1	0	0	0	0
TX3	9	0	1	0	0	0	1
	10	0	1	0	0	1	0
	11	0	1	0	0	1	1
	12	0	1	0	1	0	0
	13	0	1	0	1	0	1
	14	0	1	0	1	1	0
	15	0	1	0	1	1	1
	16	0	1	1	0	0	0
TX4	9	0	1	1	0	0	1
	10	0	1	1	0	1	0
	11	0	1	1	0	1	1
	12	0	1	1	1	0	0
	13	0	1	1	1	0	1
	14	0	1	1	1	1	0
	15	0	1	1	1	1	1
	16	1	0	0	0	0	0

When all the switches are up the program is on real time mode. The receive antenna are switched sequentially using the pattern showed in the appendix B. The VGA display was not implemented in this version of the code because the data processing was done in MATLAB. However, code for the VGA is also present in the library.

Finally, this code let the user save the data of the from the 4 channels IQ demodulator into the RAM storage of the FPGA. Refer to user manual on how to do that. This code not fully tested, as the time between the construction of the radar and the implementation of the code did not allow too many full range testing. However, loading the data in the memory should work because it was tested using the DC voltage supply.

3.9 Display

3.9.1 Purpose

The purpose of the display was to implement a control a VGA monitor in order to inform the user about which phase center was picking up the most reflected signal. The display does not necessarily tells the user where in the scene the metal is, it only give an idea as to what region of the scene were have the most reflection.

3.9.2 Requirements

This version of the VGA monitor code is an initial step to show that the outputting data on the monitor is possible. The display consisted of displaying 16 columns each representing the phase magnitude of the Fast Fourier Transform at each of the angle bin.

3.9.3 Method

Table-3: VGA monitor Parameter

	Pixel Clock	pulse (pixels)	Back Porch (pixels)	active (pixels)	Front Porch (pixels)	Total (pixels)
Horizontal	108 MHz	112	248	1280	48	1688
Vertical	108 MHz	3	38	1024	1	1066

The code for the VGA is done and is located on the project flash drive. The VGA code use a 108 MHz clock to produce 1280 x 1024 resolution. The refresh rate of the monitor is 60Hz which allows any data from the scene to come and be display in little bit more than 17 milliseconds. Furthermore, this resolution divided the monitor into 1640 Horizontal lines and 1065 vertical lines. The parameters used to program the VGA is presented in the table-3.

3.9.4 Challenges

The challenging part of the VGA display code is to create a 108 MHz clock. In practice in order to create a clock a faster clock need to be sampled. Since the FPGA clock is 100MHz, this clock was not able to drive the VGA components. In order to solve the problem, Xilinx IP core generator clock wizards

was used to create a 108 MHz clock. If one need to reconfigure the code, he/she has to recreate the IP core clock wizard. Although using the core generator is a good practice, it need to be understood before it is used in the code. For example, when declaring the code, one has to make sure that the input CLK is not declared as a buffer or else the program will not compile and will give an error. It is advised to look up the datasheet about any core used in Xilinx.

3.9.5 Results

Although this code is completed, it was not used to in the code because the processing was done in MATLAB as a way to prove the concepts. However the code was tested using the switches and a DC voltage generator. The ADC will take in value from the DC supply. The switches where used to determine which column on the monitor was activate. Changing the voltage from the DC supply will vary the height of the column.

3.10 Signal Processing (MATLAB)

3.10.1 Purpose

The purpose of implementing MATLAB is to serve as a base interim goal for the signal processing on this project. At the beginning of the year, the main goal was to implement all of the signal processing on the FPGA, however, this goal was adjusted January of 2016. MATLAB will serve as a good testing tool for signal processing work in the future upon new iterations of the SAR system. Originally, MATLAB processing code was to be worked on in parallel with the signal processing code in VHDL, however, this was also altered when the team realized that real time signal processing would require a lot more time to implement that was just not available during the year.

Another reason why MATLAB was chosen as a medium for performing signal processing work was to allow for future development of the system as a whole. Using MATLAB allows for a Neural Network to be more easily implemented, allowing for greater image recognition and image formation accuracy.

Finally, MATLAB offers many toolboxes, such as the Phased Array toolkit and Simulink, which will help with testing the radar in future iterations of this project. Another reason for deciding to implement the signal processing work on MATLAB was the amount of image and data visualization tools that is offered with the software. It is possible to simultaneously generate a 2-Dimensional Cartesian plot, a 2D Bar Graph plot (simulating the “stripe” that would appear on the

dedicated VGA display in real time targeting), as well as a 3D view of the scene when trying to visualize the energy scatter at the scene after performing the Fourier Transform on the received signals. Finally, MATLAB also makes it easier to create custom functions that can perform boresight calibration of the data, which will be expanded upon further in this section.

3.10.2 Requirements

The main requirement for the MATLAB signal processing is to implement code that will perform the Discrete Fourier Transform (DFT) on the reflected signals coming back from the target at the scene. The Fourier transform is one of the key techniques that is used in almost all signal processing. The Transform will be able to take the signal, pick up different energies and distances once put through the transform, then using this data, be able to form a certain scattering of these energies and give all of the necessary information to generate the image that was captured at the scene. Specifically, the transform is used to decompose the signal into different sine and cosine waves.

Along with writing the DFT code in MATLAB, data visualization must also be implemented simultaneously to the DFT calculations, so that somebody can look at a graph and see where the highest energy signature in the scene is coming from.

The next requirement is to save the basis functions that are created by the phase centers between the transmit antenna pairs into MATLAB and be able to do some mathematical manipulations on them. The requirement that may be the second most important is to be able to allow the FPGA and MATLAB to "talk" to each other. More specifically, a method that is used to convert the I and Q data taken from the FPGA into a suitable form that can be used in MATLAB. Finally, the last requirement is to write code that will implement the calibration code that is used to correct for the phase slope error that happens in the near field transmission.

3.10.3 Method

When attempting the write the code to perform the Fourier Transform in MATLAB, a few things must be considered. The first is that all of the calculations must be saved into a singular function that takes in an array of 16 I values and a second array of 16 Q values, in order to speed up the processing. With the Fourier Transform is able to be calculated with one line of code, it shows great improvement over the excel spreadsheet with test values that was used to do signal

processing in the year prior. Within this matlab function that was written, the phase angles for all of the phase centers were saved into an array, which these are generated by the spacing of the actual transmit and receive antenna horns. The basis functions must be saved in this code as well. When creating these basis functions based on these phase angles, a 16 by 16 matrix was created. Now this is done, the Fourier transform can be computed. Using the received I and Q values, complex multiplication with this basis function matrix can be completed. An example of this is shown below:

```
cmplxRIQ15 = ((freal(15,:)*I(:,15)) - (fim(15,:)*Q(:,15)));
cmplxRIQ16 = ((freal(16,:)*I(:,16)) - (fim(16,:)*Q(:,16)));
cmplxImIQ1 = ((freal(1,:)*Q(:,1)) + (fim(1,:)*I(:,1)));
cmplxImIQ2 = ((freal(2,:)*Q(:,2)) + (fim(2,:)*I(:,2)));
cmplxImIQ3 = ((freal(3,:)*Q(:,3)) + (fim(3,:)*I(:,3)));
```

Figure-31 : Snippet of MATLAB code showing complex multiplication of I and Q values with the basis functions

Once this is completed, these I and Q values must be added together and converted to the decibel (dB) scale. Finally, these values must be plotted in Cartesian form and bar stripe form against the bin angles, or the angle of the corner reflector downrange in the scene, relative to boresight. Both of these plots are shown below:

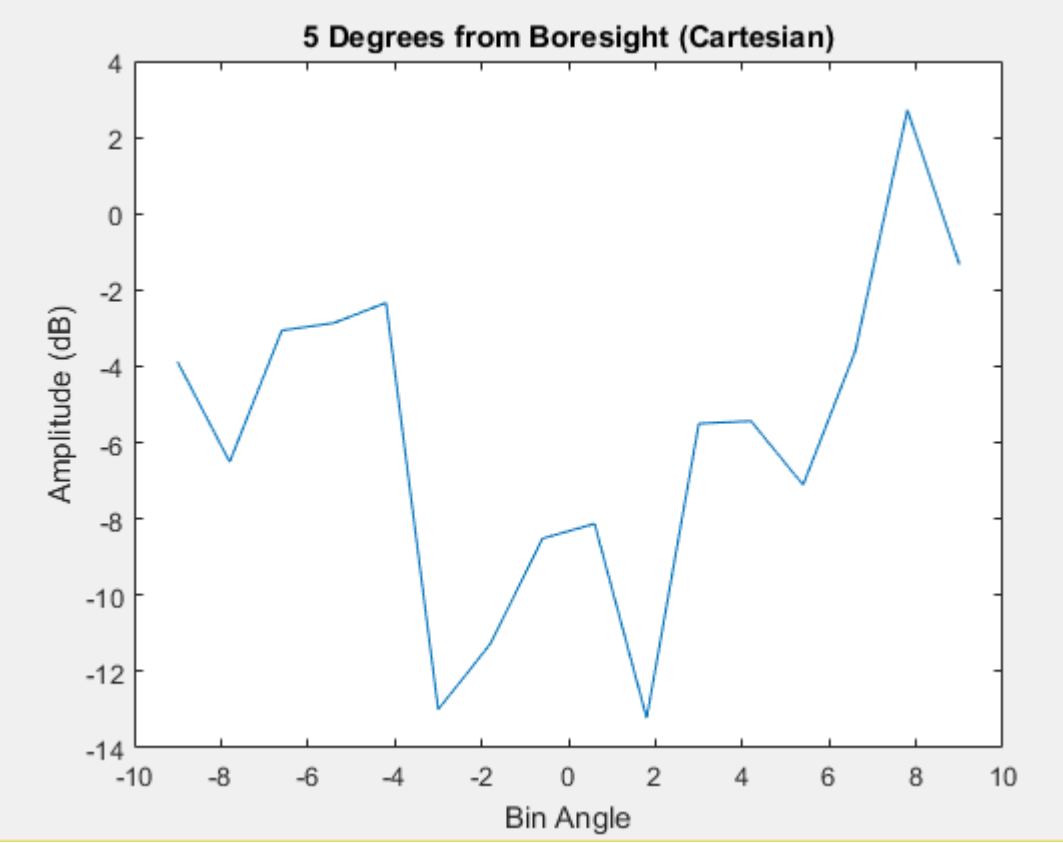


Figure-32: 2D Cartesian plot of energy scatter

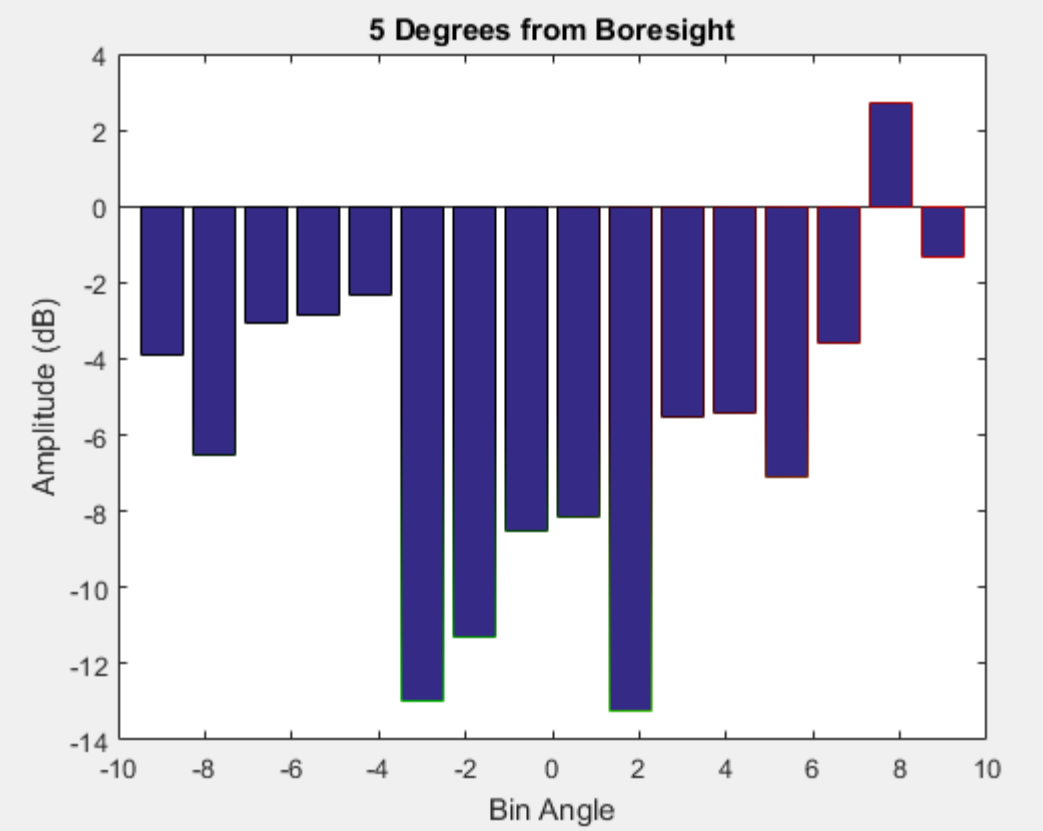


Figure-33: 2D bar plot of energy scatter "stripe form"

Now that the Fourier transform code has been completed, the code for taking the received data from the FPGA needs to be completed. The method that was used to go about this was to take the hex values that were stored in as an upper byte and a lower byte for each I, Ibar, Q and Qbar channel, then saved as a binary file from the FPGA memory and convert them into voltage values and save these values into two 1 by 16 sized arrays. To do so, the hex values had to be converted back to a DC voltage value, which was done by the following computation, as seen by the snippet of code:

```
Iarr1 = ((dat(2).*((16).^2)) + dat(1)); % Grabs Upper Byte and Lower Byte, Does Computation

Qarr1 = ((dat(66).*((16).^2)) + dat(65)); % Grabs Upper Byte and Lower Byte, Does Computation
```

Figure-34: Snippet of MATLAB code showing how to convert hex I and Q data from upper byte and lower byte notation in a binary file to a DC voltage value

After this, the I and Q data was sorted into the 1 by 16 sized arrays for computation in the DFT algorithm, which is given as the output of the matlab files used. To note, there were two separate matlab functions that read in and converted all of this data from the FPGA, the first to read the I

data and output the correct values measured, and the second to do the same with the Q values that were measured. Finally the method that was used to do the calibration calculation was to create a separate function that takes the complex conjugate of the received I and Q data at boresight and multiplies it with any received values that are taken at any angle away from boresight. In theory, this should isolate the single energy peak at the bin angle where energy is being detected, however this did not turn out as planned, as this was an unexpected error. Essentially regardless of where the corner reflector was placed at the scene, after calibration, the peak amplitude would always converge towards 0 degrees, as if something were being targeted at boresight. This issue most likely comes from some multipath error, as discussed further on in RF testing results.

3.10.4 Challenges

There were many challenges when it actually came to doing the signal processing on MATLAB, but first, the challenges that were encountered during the actual coding must be addressed. To start, there was an initial misunderstanding between teammates on how the data coming from the FPGA will actually be represented. This was because there was the initial idea that the only values that are to be saved to the FPGA are from the I and the Q channels, so separate logic needed to be written in order for the code to recognize whether or not the values coming in are negative.

This initial challenge was met by creating a reference that the received value will be negative if the decimal value if above the integer value 4096. This was done after raw conversion of the hex value, but before "normalizing" the value to a voltage value that exists in the millivolt range. Specifically, this "normalization" challenge was tackled by simply dividing the decimal value after hex conversion by 4095 then multiplying that result by 3.3 volts, or the voltage that controls the ADC switching logic. However, this ended up being incorrect and incomplete, due to the fact that the code for controlling the inverse I (Ibar) and inverse Q (Qbar) channels from the IQ demodulator had been completed.

This code is the most recent challenge met, also the last MATLAB challenge to tackle. Reading all four channels meant that there would actually be another set of 64 bytes representing the inverse I and inverse Q channels, also forcing for the need to alter the logic used to detect a negative value. Now the negative value would show up as a positive number in the inverse channels, while the normal I and Q channels will show a value of zero in that location in the array. This was logic was

corrected by setting up a check in the code. If the I or Q channel has a hex value of 0 for both their upper byte and lower byte locations, then then data must be taken from the inverse channel and then the normalization calculation can be completed. If the inverse channel was used, then the code will simply make that value negative and place it in the new array for I and Q. A snippet of this code is seen below:

```
if Qarr3 ~= check
    Qv3 = (((Qarr3)./4095).*(3.3));
else
    Qv3 = -(((Qbarr3)./4095).*(3.3));
end
```

Figure-35: Snippet of MATLAB code logic for negative and positive values from ADC and FPGA along with normalization code. (Check is set as 0 in beginning of code)

The next challenge that was to be dealt with is the issue of calibration. Due to the fact that there will be error in the data from the radar not operating in the far field, there will be some phase slope that is created when going off at more extreme angles in the scene, so to correct this, there needs to be code to make up for that error. The only challenge that was presented was a more simple way of doing the complex conjugate multiplication with the data taken at boresight, then saving those values, and reusing all of them for any data taken at other angles. Luckily, the decision to use MATLAB allows for data to be saved into variables and then brought up whenever is necessary, so that challenge was easily met.

A challenge that was foreseen would be if the energy scatter at the scene had too much extra energy signatures from foreign objects, making the signal processing much more complicated than can be dealt with in the time provided for full work on the design project. Finally, there is a challenge that MATLAB inherently presents, and that is the issue of not being real time signal processing and image formation. After discussing this with our sponsor, the latency between bringing the data from the FPGA to MATLAB, then the processing speed of MATLAB, was not too much of an issue, because it shows very clear progress from the previous year, in terms of efficiency, effectiveness, versatility and speed.

3.10.5 Results

Overall, the code itself does operate as expected. The fourier transform MATLAB code was verified using the excel signal processing file provided by the sponsor from the previous year. The

stark difference is being able to quickly perform the signal processing, being able to read data straight from the system itself and performing the calculations. The code that reads the I and Q data off of the FPGA memory does work in practice, but only with the slider switches and the values that they generate by writing to memory. This was not able to be tested with actual data that was taken with the radar in the field, but not due to an error with the MATLAB code, but rather the results that were taken as the radar was operating were not the correct values that were coming in and there were issues with writing these data points to the memory of the FPGA.

Furthermore, this MATLAB code was also verified and tested by creating an array of 128 values, chosen at random by the randn MATLAB function. The resulting output from the readI and readQ MATLAB functions were in fact a 1 by 16 size array for both I and Q, showing both negative and positive values. This proved that the logic for pulling negative and positive values from all 4 channels from the IQ demodulator did work as intended.

Finally, while the calibration code was written as the sponsor directed it to be written, the actual results of this calibration code were not ideal, however it is important to note that the calibration code can easily be changed if a new method if figured out in future development of the project.

No coding work on the neural network has been completed by the students on the team, simply because the main goal was to get a functioning radar system first. Given more time, or opportunities with future teams on the project, this can also be easily implemented given that the I and Q data does get stored in matrix form, allowing for easy implementation into the neural network algorithm that the instructor has created. The resulting MATLAB code as it stands now can be considered a success and a vast improvement over the previous year in implementing signal processing, closer to real time than the excel spreadsheet presented from the prior team. Moving forward, this MATLAB code can be handed off to any future team on the project, as a way to assist in testing the radar once the issue of multipath error has been solved, as well as a toll to utilize when creating and debugging the code that will be used to implement the signal processing natively on the FPGA.

3.11 Signal Processing (VHDL)

3.11.1 Purpose

The purpose of implementing the signal processing using VHDL natively on the FPGA is to create a SAR system that more closely represents a final product that will be used in industry. Performing all of the signal processing on the FPGA effectively cuts out the need for an extra computer being in the loop and performing the processing on MATLAB. This also allows for real time signal processing, allowing for a great increase in speed of processing and near immediate image formation on the dedicated VGA display.

3.11.2 Requirements

The first requirement is that the FPGA needs to have the basis functions stored onto the board's own memory. The next requirement is that the FPGA be able to store the results of the Fourier transform onto the memory to be able to access it any time after for whatever reason. The FPGA, and most importantly, must be able to implement the Fourier transform completely independent of another computer.

3.11.3 Method

The signal processing using VHDL must be implemented using a Fast Fourier Transform (FFT) algorithm. This can be done in three ways, either by utilizing an IP core provided by Xilinx, which is difficult for several reasons, implementing a hard coded FFT algorithm, or by purchasing a separate FFT module that can be added onto the FPGA, however this would be a very expensive option, roughly over $2000.

The downside to the second option is that it would take much more time to develop an algorithm and implement it on VHDL than it would be to order the separate module and spare a lot of VHDL coding. Using code on the FPGA board that was developed from an algorithm will also take more processing time to do all of the calculations than the time it would take the FFT module to do the same work. This, however, is a much better option that paying for a separate module. This algorithm was actually found during research in the College of Engineering's own Dr Meyer-Basae's book "Digital Signal Processing with FPGAs." This specific algorithm can be seen below, implementing 5 additions and only 3 multiplications, being seen as fairly efficient use of the FPGAs processing power and being more cost effective, not needing nearly as large of logic

circuits to perform these calculations, as compared to the complexity of implementing a fourier transform and complex multiplication through other means in developing such an algorithm from scratch.

$$s[1]=a-b; \quad s[2]=c-d; \quad s[3]=c+d;$$
$$m[1]=s[1]d; \quad m[2]=s[2]a; \quad m[3]=s[3]b;$$
$$s[4]=m[1]+m[2]; \quad s[5]=m[1]+m[3];$$
$$(a+jb)(c+jd) = s[4]+js[5];$$

Figure-35: FFT algorithm for complex multiplication between I and Q data received, with the basis functions

Finally, the other method of using an IP Core provided by Xilinx is most likely the best option because all of the coding is pretty much provided, with only some minor modifications needed to be made to include the basis functions that are used during the transform.

3.11.4 Challenges

The biggest challenge with implementing the signal processing on the FPGA using VHDL is simply the time it would take to fully realize the code and get it to a working state. As described above, the largest challenge with buying a separate FFT module for the FPGA is simply that the cost is too large in comparison to the rest of the budget to actually justify purchasing. Another challenge with that method would be learning how to interface the actual module with the FPGA board itself. Finally, another challenge would be how to make the signal processing VHDL code work in tandem with the rest of the FPGA code and integrating that into the state machines created to control the rest of the system, making the control logic just a bit more complicated. Finally, another challenge would be to somehow communicate the results given from the FFT and have that integrate with the dedicated VGA display.

3.11.5 Results

After about February, the decision to hold back on implementing the signal processing on the FPGA was made so that it would be easier to focus on implementing the signal processing on

MATLAB. This was to create an incremental goal, and allow for greater testing of the signal processing and of the system. This also allows for a better method of debugging the VHDL code when that will eventually be written. While the FPGA certainly allows for near real time signal processing and it empirically faster at doing the signal processing than it is the do all of the signal processing externally on MATLAB, there simply wasn't time to further work on the signal processing on the board itself.

This was one of the most important engineering decisions made during the course of the project this year, teaching the team lessons about defending their reasoning for going through with certain options, as well as serving as a real world lesson in that iterative designs can be used to demonstrate certain functions of a system being engineered. For future recommendations on how the signal processing should be implemented on the FPGA is to use the resources that have already been given. Using MATLAB to troubleshoot the coding and to compare results of the FFT during targeting will greatly speed up the process of making sure the signal processing is being implemented correctly on the FPGA board.

Any future team should also seriously look into implementing the Xilinx FFT IP Core rather than creating and coding a FFT algorithm by hand, because the IP Core FFT is most likely better optimized for the specific Xilinx FPGA than any code that is written from scratch.

IV. Test Plan

4.1 Test Phases

4.1.1 Phase 1: Legacy Verification Testing

The first phase of testing focused on verification and evaluation of the legacy prototype. In addition to establishing a baseline for the situation the team was inheriting, the initial battery of tests were an important familiarization exercise. There was a steep learning curve, not just for radar, but for microwave components in general and their associated test equipment. It was the first exposure for most of the team to high-frequency function generators and the spectrum analyzer and network analyzer.

4.1.2 Phase 2: Re-design Iterative Testing

As the picture of the legacy design's status became clearer, focus was shifted to improving the design and completing the signal path. The process of synthesizing new code occurred in this phase.

Development began with top-level functional diagrams, then moved on to state diagrams, timing diagrams, pseudocode, writing, and iterative testing and debugging.

During this phase, it became clear that MATLAB was far better suited to solving the signal processing problems given its wider range of native signal processing tools and ease of use over VHDL. As it was initially set out, the FPGA was to precisely drive the control lines at high frequency, provide signal processing, run calibration logic, and drive a custom VGA display system. The scope and volume of VHDL code to be written reinforced the decision to outsource most of those functions to MATLAB, even at the cost of stand-alone functionality and real-time processing.

The FPGA-driven VGA display would only be stipes on the screen representing the phase centers, so if the user ever wanted to manipulate that data in any interesting way, they would need to export it to a program like MATLAB anyway. A limited VGA display framework was set forth at the request of the sponsor, but writing custom display drivers from scratch in hardware language was not the most effective use of very limited human resources, especially given that critical FPGA system control code remained. Work continued on the FPGA system control code modules, but signal processing resources were shifted to MATLAB development.

Phase 1 testing should have revealed that the SPDT switch that controlled transmit/receive mode switching was faulty, but the suitable test equipment that had to be sent by Northrop Grumman arrived damaged. After the lead time to get the proper equipment, lack of experience in high-speed microwave switches and lack of experience with the test equipment resulted in numerous inconclusive results. While this was frustrating, valuable insights were gained regarding the proper procedures for testing microwave components, as well as experience in operating and interpreting the readings on the spectrum analyzer.

Spare switches of a different (cheaper) model were available and implemented, but the FPGA lacked the output voltage to reliably switch them. A voltage step-up circuit requiring miniscule, high-frequency surface-mount components was designed and prototyped. In terms of learning objectives achieved, the exercise was incredibly beneficial. The team was exposed to Eagle CAD, PCB milling, high-frequency circuitry, and precision soldering for the first time. However, the cost in terms of time wasted on a part that was blown up anyway was immense. Not having this part outsourced to a PCB fabrication service was a major blunder in project management. It was clear at the onset that the college did not have the resources to properly fabricate this component, and action should have been taken earlier.

Meanwhile, the mechanical engineers went through design and testing iterations of the frame, component housing, horn holder assemblies, and optical alignment assemblies. A major lesson learned in

this process was to always make sure to leverage the experience and expertise of machinists and technicians. Input from the machine shop influenced several design changes for the better. At times, productive communication between the engineering team and the machine shop broke down, and it resulted in the component housing being built backwards. This forced the re-design of the electrical component layout.

4.1.3 Phase 3: Migration Testing

The objective of this phase of testing was to verify that the SAR electronic system still worked in the newly-fabricated structure and component housing. Phase 1 tests were conducted again on the newly designed implementation successfully. Changes in cable run lengths required that the fixed attenuators in the system be changed to preserve the power budget. Future teams should consider the fixed attenuators a tool for optimizing the gain/loss characteristics along the signal path and not something written in stone. Some of the components are running near compression and some are well below. In addition to this, the team learned some valuable lessons on optimizing signal strength in an RF electronic signal path. The most important being on how to pad the attenuation of the signal in order to most effectively increase the transmitted signal power without reaching saturation levels for any single component, effectively driving signal performance down. While this was not achieved fully, it did allow for greater exposure to solving engineering challenges dealing specifically with the optimizing electronic signal paths and how to balance these changes in any given path with other branching paths in the system. Specifically if there is a very high transmit power that is needed, knowing how to pad and attenuate that power on the receive chain is imperative so that no sensitive component is being fed too powerful of a signal, while also making sure it is not being fed too low of a power signal. More extensive field testing will reveal the optimal solution since this team did not collect enough field data to verify what the optimal variable attenuator settings are for the chosen operating environment.

4.2 RF Range Test Results

4.2.1 Purpose

RF range testing represents the final phase of testing. The 2016 team sought to demonstrate the detection functionality of the SAR Imager and to collect the first data sets of what will be the extensive field testing required to fully characterize the full SAR imaging problem.

4.2.2 Method

Initial range testing was conducted in AME 234. This room was chosen for its relatively open floor space and after-hours access.

In lieu of a suitable anechoic testing venue, anechoic RF absorbing panels were mounted onto wooden stands that were placed behind the reflector. This was meant to absorb background reflections with the hope that most other reflections would dissipate sufficiently to be negligible with respect to the direct target reflection.

The SAR pulsed the corner reflector at boresight using FPGA Manual Mode. Transmit/receive mode switching was driven by Tektronix AFG 3022B Dual Channel 25MHz Function Generator using the settings in the SAR user manual because the voltage step-up circuit that would allow for FPGA mode switching was overloaded previously.

Voltage was measured from the I and Q channels of the ID Demodulator for each transmit/receive pair and tabulated.

4.2.3 Results

Initial tests yielded I and Q voltages near or over the maximum output voltage of the IQ Demodulator of 300mV, particularly for the pairs closest together. Increasing attenuation at both variable attenuators, thereby reducing power transmitted and received, mitigated this to some degree, but the transmit/receive pairs closest together were still noticeably higher than the other pairs even though the reflector was at boresight.

A great amount of time was spent moving the RF panels around the room, trying to find an arrangement that would mitigate what appeared to be the junk data. This proved difficult, not only because a single transmit/receiver pair was observable at a time and the data was fluctuating +/- 10mV when everything was stationary, but also because very small movements of an anechoic panel would result in large spikes on the meters, or nothing at all. This should not have been surprising since the wavelength at 10 GHz is only 3cm. At such small wavelengths, changes on the order of inches could be quite significant with respect to a 3cm wave.

However, removing the corner reflector from the scene made a clear difference, so it was decided to move the reflector radially about boresight as planned so as to collect enough data to conduct meaningful analysis. When the reflector was -5 degrees from boresight, the data in figure-36 was collected. The figure

shows the recorded data in orange over the basis function for -5 degrees shown in blue. A best fit line of the measured data reveals that the results may be more in line the ideal model than it seems at first.

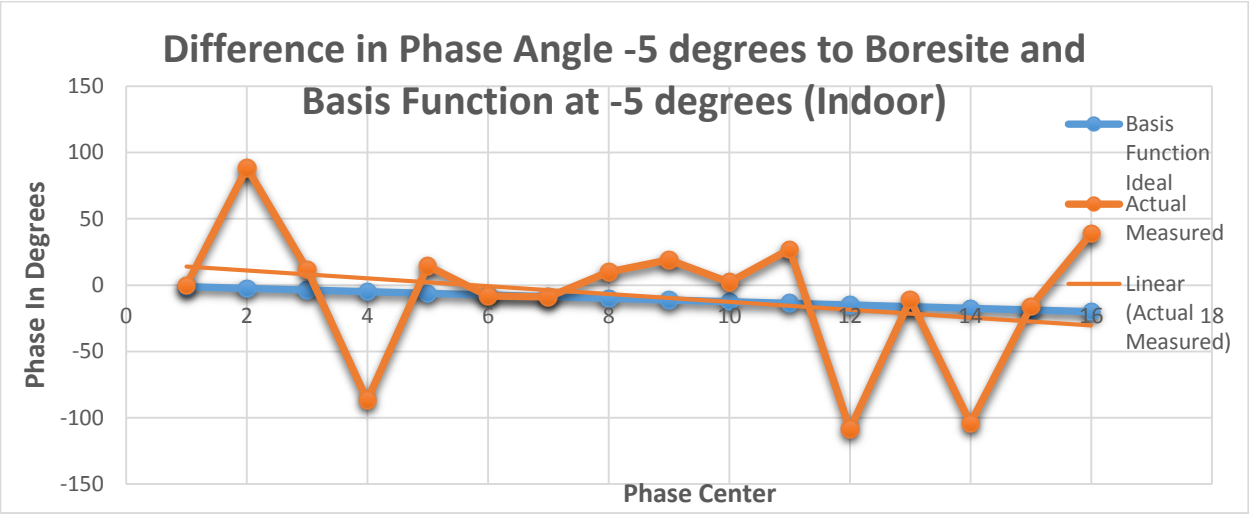


Figure-36: Difference in phase angles

As promising as this was, it was getting late in the evening on the last weekend of the project by this time. It was decided to set up the system outside in the empty parking lot to get an idea of how much difference being inside an enclosed room made. Outdoor data is shown in figure-37below.

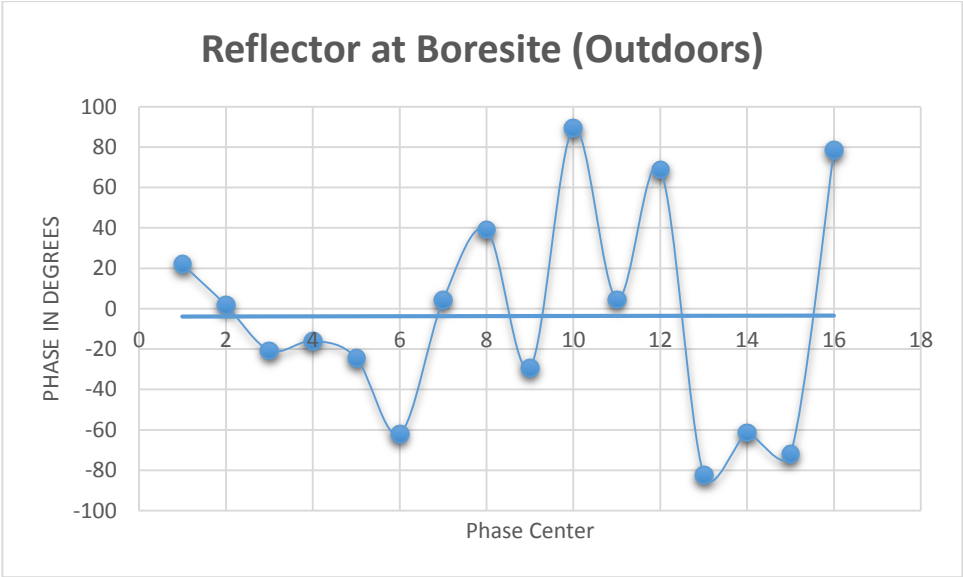


Figure-37: Phase vs Phase center with corner reflector

As can be seen from the data taken with the radar in the outdoor, empty parking lot, the phase slope for the corner reflector being detected at boresight is roughly 0, with a phase offset of approximately 3 degrees. Interestingly enough, the phase slope for an image at boresight given the basis function image formation also is a flat line, but positioned at 0 degrees, with no offset. This offset represents some error in the system, but being off by a small amount is not as important as the actual phase slope very closely resembling the ideal phase slope in a vacuum type setting. This shows that the radar system is functioning correctly, while not being completely perfect. This is due to the issue of multipath error, as a result from multiple reflections at the scene. However, this data is much more accurate than the data that was taken in the indoor lab, showing how much of an effect multiple reflections can have on an imaging system.

4.2.4 Discussion

Unfortunately, workflow in the laboratory portion of the project had to be stopped in order to produce the project deliverables and the team was already pushing the timeline to the limit. The -5 degrees from boresight data that would be directly comparable to the indoor data collected was invalid due to loose connectors that were hastily fastened by flashlight.

Much more data needs to be taken in the field before the following educated conjectures could be considered conclusions; but nevertheless, we believe the deviations in the measured phases from the ideal phase slope are likely due to lack of transmit/receive antenna isolation and multipath reflections.

4.2.4.1 Near Field Effects/ Isolation

The figure below shows the radiation pattern of a pyramidal horn antenna similar to the antennas used in the SAR design. Notice the side lobes and rear lobes radiating away from the main directive beam.

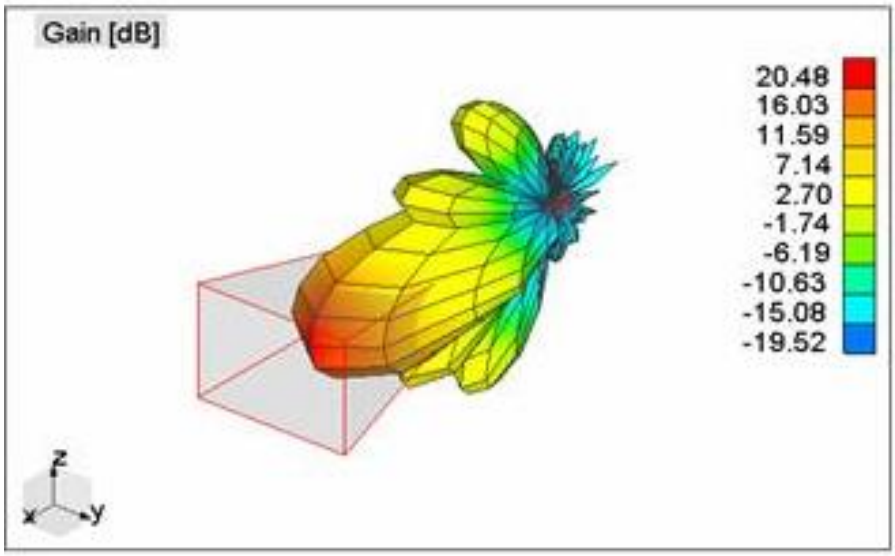


Figure-38: Horn antenna radiation pattern
http://www.wipl-d.com/applications.php?cont=antenna-design/aperture-antennas

The spacing of the transmit antennas from the nearest receive antenna was based on a ray model similar to figure with generating an idealized phase center in mind. However, this model fails to take into account the near-field effects from the transmitter. The side lobes and/or rear lobes of the transmitter are impinging on the adjacent receiver. This would explain why the transmit/receive pairs on the ends of the array (i.e. closest together) were experiencing higher voltage levels at the output than the other pairs.

In transmission mode, the signal is being routed to the transmit antennas, but the Low Noise Amplifiers (LNA) are still powered and amplifying whatever they are receiving, regardless of whether the system is reading that signal or not. It is possible that LNA 1 is receiving the near-field radiation from the adjacent transmit antenna, amplifying it, and overdriving LNA 2. In fact, both may have been at or near compression when the variable attenuators were set low.

When the system switches to receive mode, the LNAs may still be in recovery time from being saturated, causing errors.

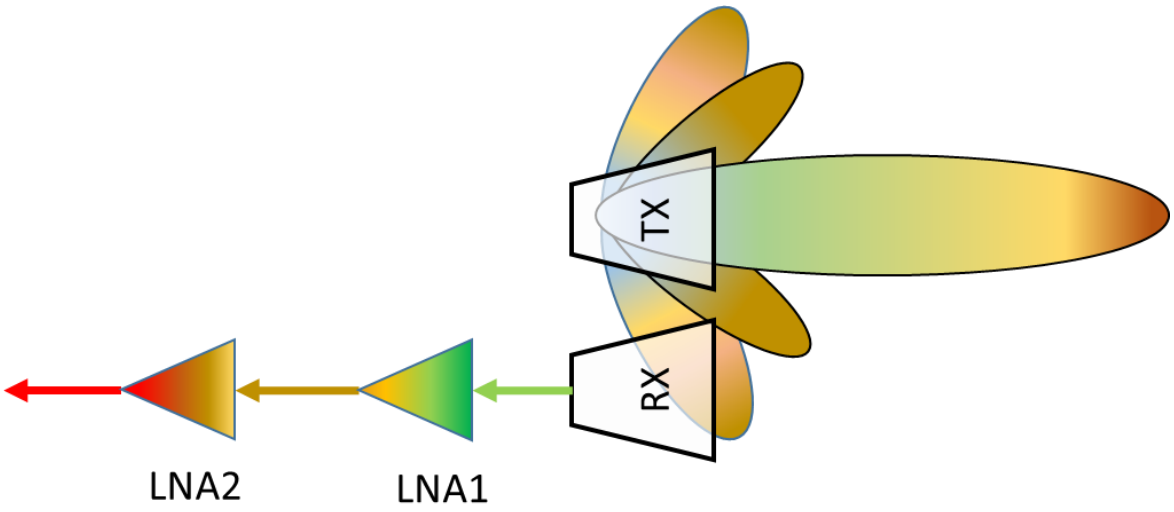


Figure-39: Near-field bleed-over from adjacent transmitter

4.2.4.2 Near-Field Mitigation: Physical Dimensioning

The most direct approach to mitigating the near-field bleed-over is to simply increase the distance between the transmitters and adjacent receivers. One could take the term “isolation” literally and separate the antennas with RF absorber material, to boot. This approach is shown in the figure below.

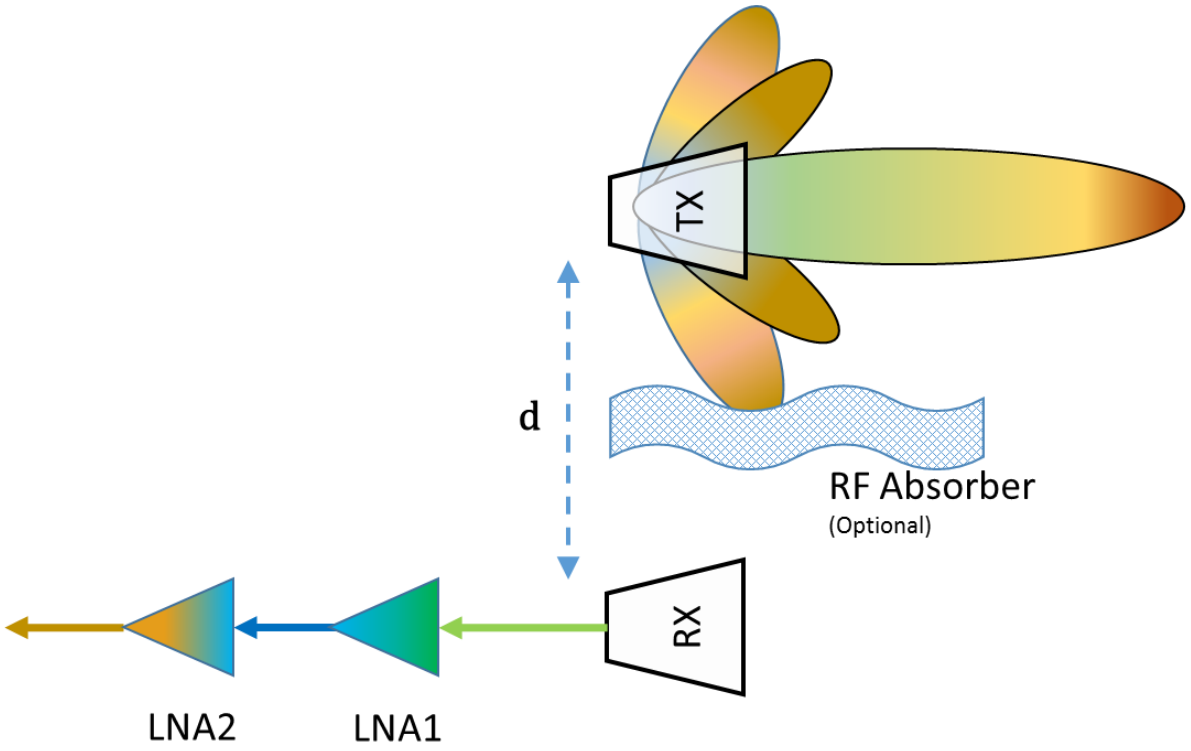


Figure-40: Near-field effect mitigation by way of increasing distance

This approach would have consequences in signal processing since it changes the location of the phase centers. Unless you wanted to simply throw out four receive horns and four phase centers, the whole system would have to be modeled again (which might not be a bad idea).

4.2.4.3 Transmit/Receive Isolation Switch

Another approach would be to utilize the spare SPDT switch already in inventory to direct the receive signal to a 50 ohm load while the SAR is in transmit mode. Since it is the exactly the same as the switch that controls transmit/receive mode switching, one can essentially copy and paste the VHDL control logic to another FPGA port and run both switches on the same clock. Two possible implementations of this solution are shown below.

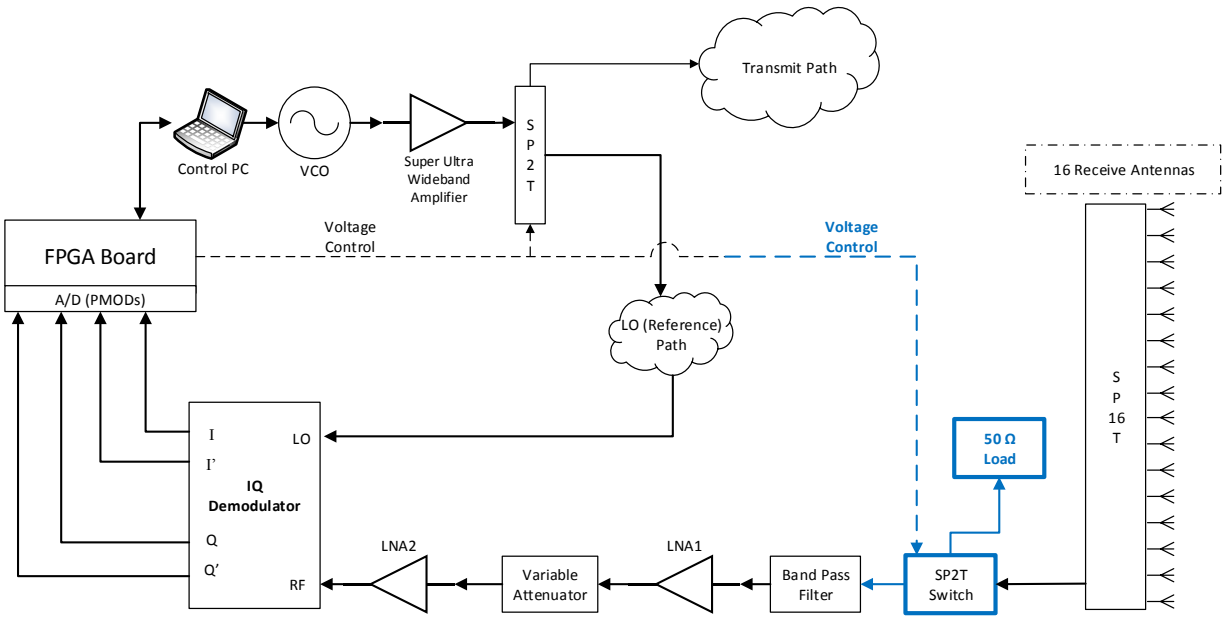


Figure-41: Isolation switch design 1

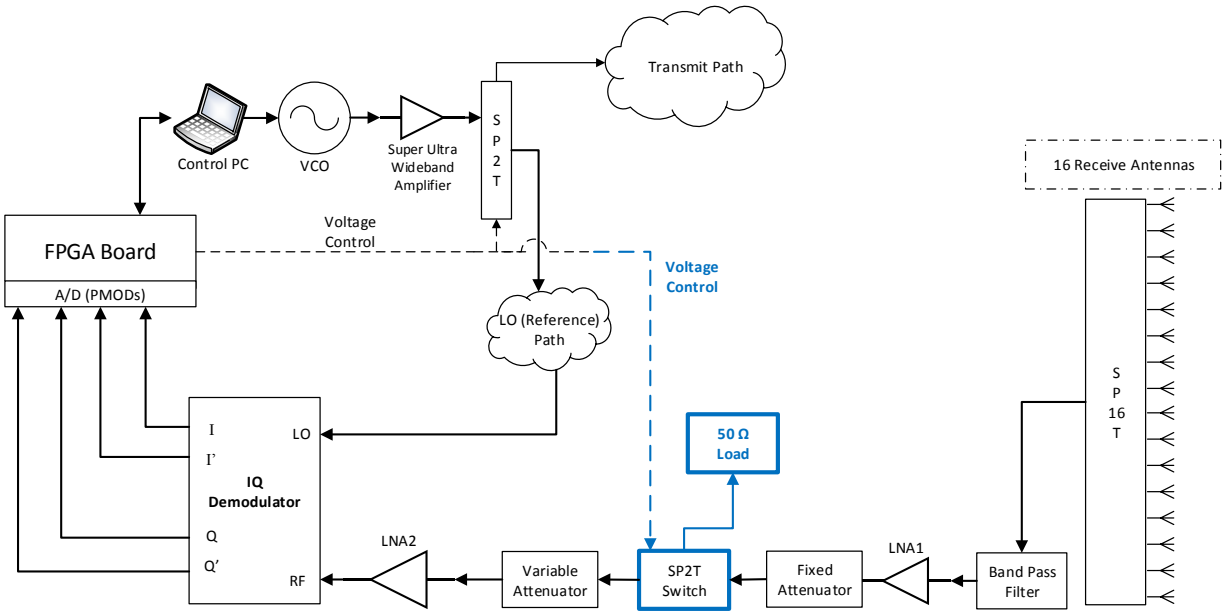


Figure-42: Isolation switch design 2

4.2.4.4 MULTIPATH INTERFERENCE

The other source of phase distortion is clearly multipath fading. Moving around the room holding an anechoic panel and watching the IQ Demodulator output change strongly suggests that the effect of multipath fading was underestimated.

It is not simply a matter of filtering out the noise floor as we initially thought. Each reflected signal returns to the receiver having taken a different path, and thus arriving at a different phase, and the receiver is being bombarded by countless reflected signals with unique phase shifts. This presents a fundamental problem since measuring the phase shift of the target reflection is what a radar is meant to do!

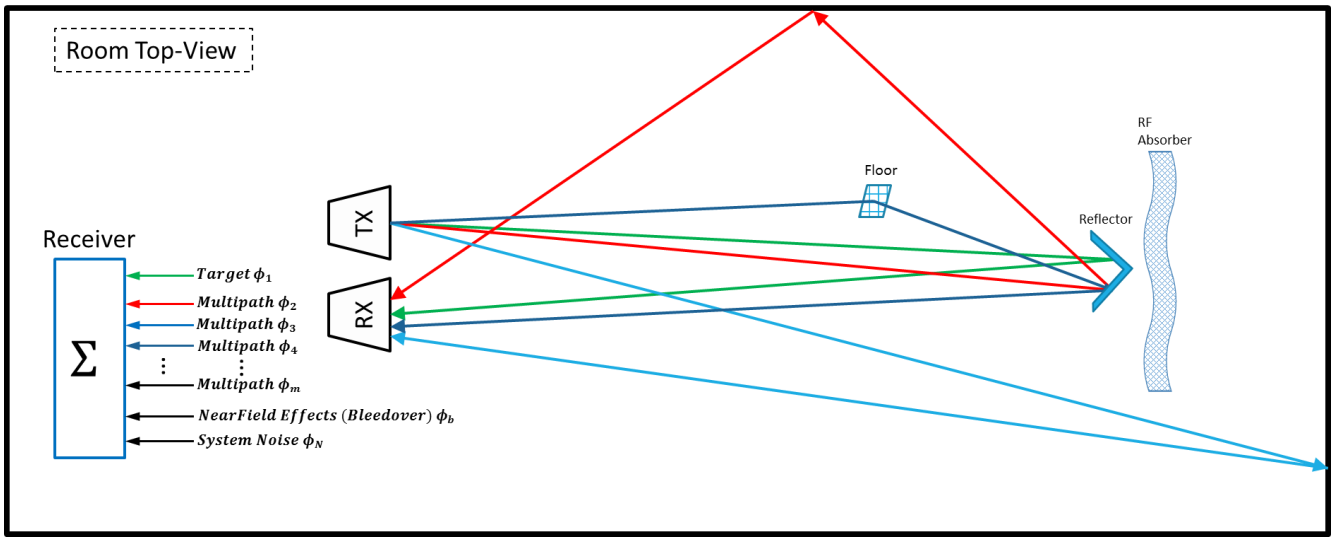


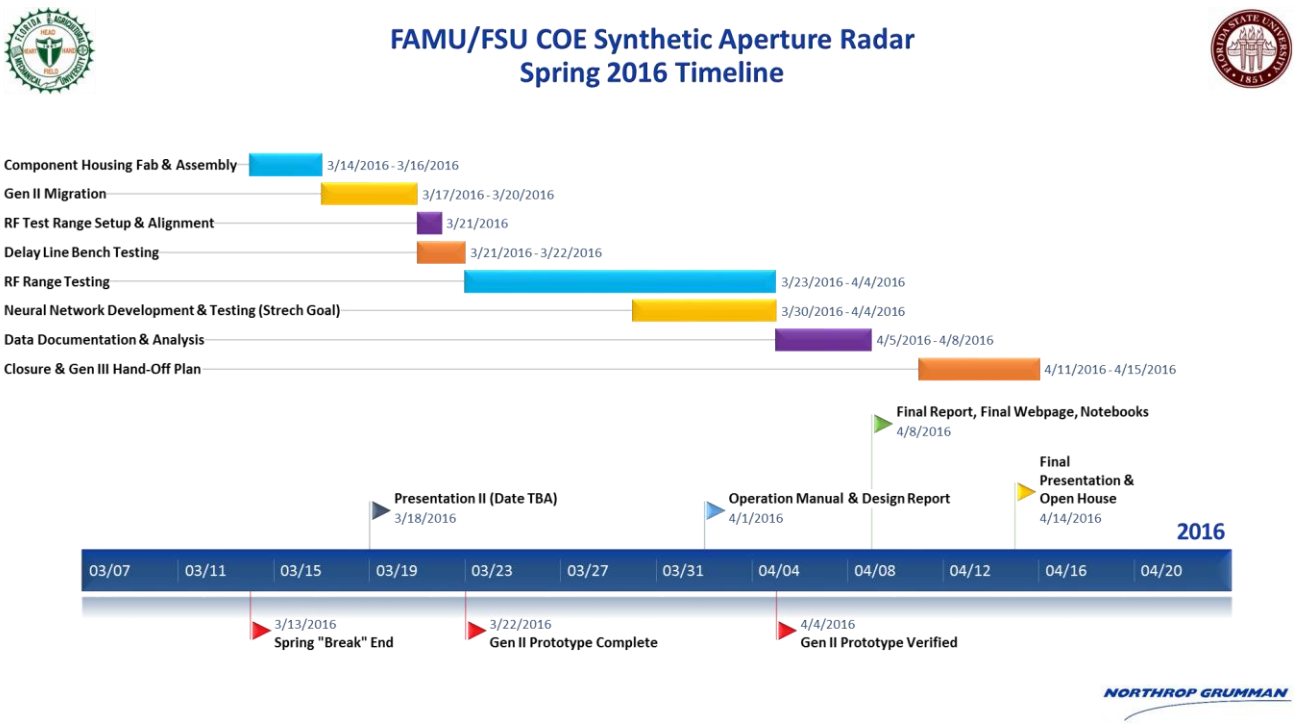
Figure-43: Diagram of multipath

Statistical models for multipath fading exist, Rayleigh and Rician fading come to mind, but they don't seem to be applicable to this situation. Developing an empirical model of the room might get one close to something, but it would require mathematics far beyond the undergraduate level to accomplish. And it is doubtful anyone wants to write a PhD thesis about AME 234.

4.2.4.5 RECOMMEND OUTDOOR TESTING

On the other hand, results in the outdoor environment seemed more reasonable, though there was not enough time to collect very much data. All the learning objectives for the project can still be accomplished is the SAR is run in an unobstructed outdoor environment.

V. Project Schedule



VI. Budget

This year, Northrop Grumman allocated the team with an amount of $5,000 for the initial budget, with a pool of another $5,000 if necessary. These funds were provided by our sponsor through the Electrical and Computer Engineering Department. The team's goal was to remain under the initial budget in order to remain cost effective. The first generation of the project had spent a majority of its budget on electrical components: approximately $31,730.83. The initial assumption was that the majority of the electrical components were purchased so that this year's budget can focus on test equipment and materials for the component housing. With this, our initial budget was estimated to be very well under $5,000 as shown below in figure-44

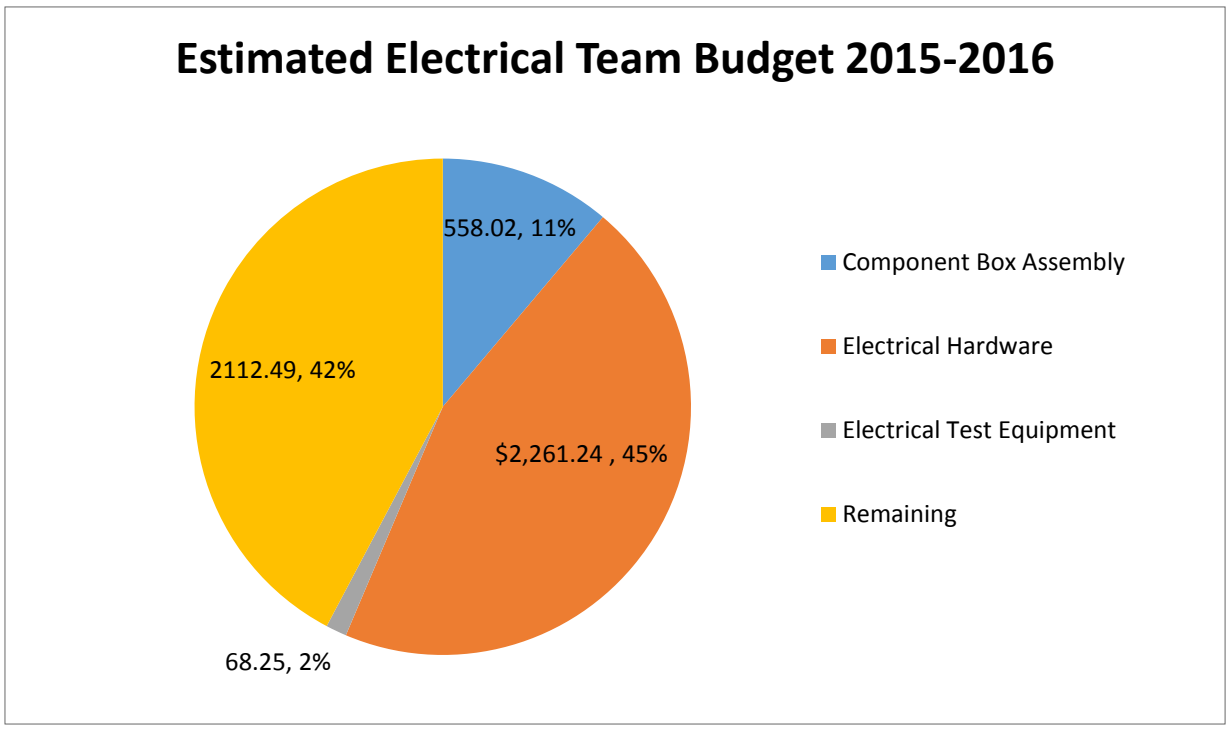


Figure-44: Estimated Electrical Team Budget 2015-2016

Table-4: Initial Expense Report

Estimated Budget 2015-2016					
Part Name	Part Number	Quantity	Unit Price	Total Cost	Vendor
USB panel mount	1195-3481-ND	2	$ 21.65	$ 43.30	Digikey
SMA panel mount	ACX1244-ND	20	$ 7.22	$ 144.44	
3” SMA cable	744-1446-ND	20	$ 18.53	$ 370.50	
12” USB type b cable	UR022-001-ND	1	$ 4.49	$ 4.49	
VGA cable 3’	TL651-ND	2	$ 10.32	$ 20.64	
Power Cable	CP-2224-ND	1	$ 3.61	$ 3.61	
Power block	T1203-P5P-ND	1	$ 24.65	$ 24.65	
Power Connection	CP-064A-ND	1	$ 2.70	$ 2.70	
HMC-C011 SPDT Switch	1127-1638 -ND	1	$ 1,652.45	$ 1,652.45	

Conductive Foam	6180T43	3	$ 45.98	$ 137.94	McMasterCarr
Conductive Foam	6180T18	1	$ 56.07	$ 56.07	
Conductive Sheet	6021T13	1	$ 39.07	$ 39.07	
Latches	13435A31	2	$ 12.78	$ 25.56	
Aluminum Sheet	89015K59	1	$ 219.47	$ 219.47	
D-Sub Panel Mount	093-112	1	$ 20.42	$ 20.42	Parts Express
Anti-Static Wrist Strap	EC-900-022	1	$ 6.99	$ 6.99	FourAker Electronics
LED Screwdriver	EC-SD-804-1	1	$ 3.99	$ 3.99	
Color Tape	67-LT-5CP	1	$ 6.55	$ 6.55	
VmodBB - VHDC-Breadboard	Vmod-BB	1	$ 57.71	$ 57.71	Diligent
Times Mic LMR-100A-PVC	16346	39	$ 0.49	$ 19.11	The Antenna Farm
RFI RSA-3050-B	33054	2	$ 4.95	$ 9.90	
Install Conn Dual	DCI	1	$ 8.00	$ 8.00	
Shipping		1	$ 9.95	$ 9.95	

The allocated $5,000 budget was not exceeded, the remaining funds for the team is $4,191.38. This is largely due to the fact that an SPDT switch purchased from Digikey by last year's team was sent back to Digikey to be repaired when in fact they determined the part was defected and offered a refund of $2,493.27. A new SPDT switch was purchased for $1,652.45. Below in figure-45 is a pie chart displaying the breakdown of the budget showing that with the reimbursement from Digikey, there is a remainder of 84%. Most of the oversights in the initial budget were the electrical connectors since rewiring was one of the last processes and therefore unexpected purchases needed to be made. In table-5 is the detailed expense report that is segmented by vendor. Preferred vendors such as Digikey were utilized as often as possible however local stores were also visited when possible in order to increase productivity time.

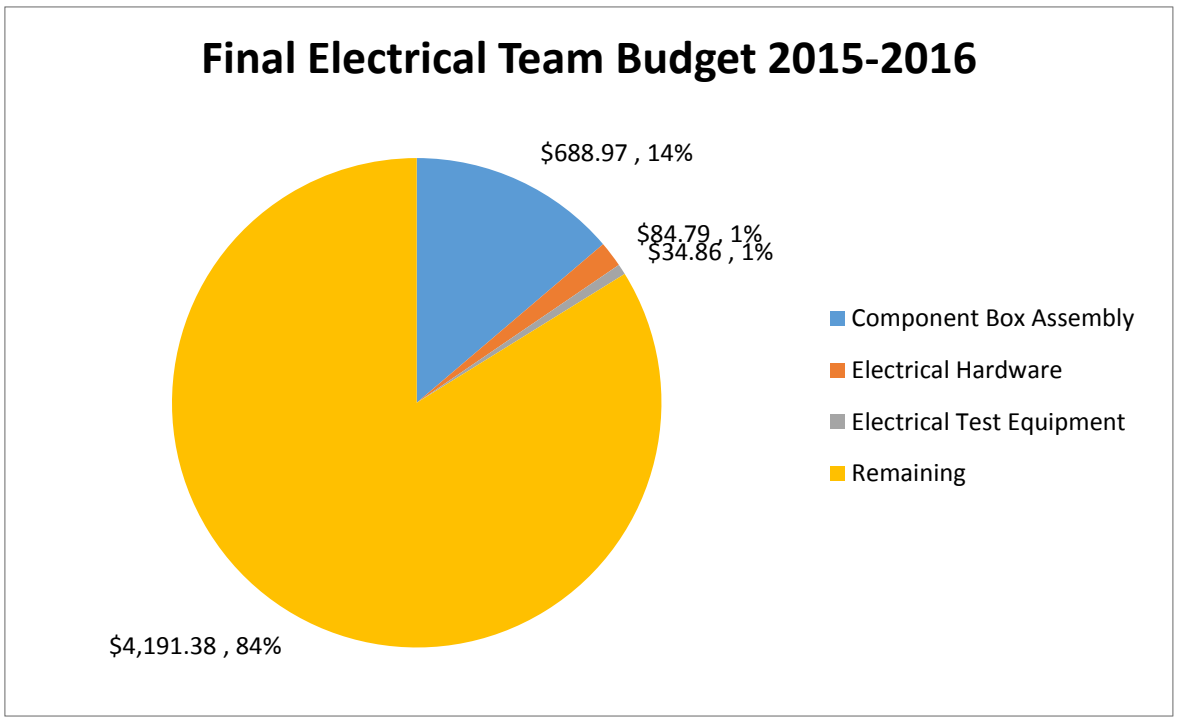


Figure-45: Final Electrical Team Budget 2015-2016

Table-5: Final Expense Report

Budget 2015-2016					
Part Name	Part Number	Quantity	Unit Price	Total Cost	Vendor
USB panel mount	1195-3481-ND	2	$ 21.65	$ 43.30	Digikey
SMA panel mount	ACX1244-ND	20	$ 7.22	$ 144.44	
3” SMA cable	744-1446-ND	20	$ 18.53	$ 370.50	
12” USB type b cable	UR022-001-ND	1	$ 4.49	$ 4.49	
VGA cable 3’	TL651-ND	2	$ 10.32	$ 20.64	

Power Cable	CP-2224-ND	1	$ 3.61	$ 3.61	
Power block	T1203-P5P-ND	1	$ 24.65	$ 24.65	
Power Connection	CP-064A-ND	1	$ 2.70	$ 2.70	
HMC-C011 SPDT Switch	1127-1638 -ND	1	$ 1,652.45	$ 1,652.45	
Jack Banana Connector	J152-ND	1	$ 0.78	$ 0.78	
USB 2.0 Male 3M	AE10625-ND	2	$ 3.98	$ 7.96	
Tax/Shipping		1	$ 4.16	$ 4.16	
HMC-C058 SPDT Switch	1127-1646-ND	1	$ (2,493.27)	$ (2,493.27)	
Conductive Foam	6180T43	3	$ 45.98	$ 137.94	McMasterCarr
Conductive Foam	6180T18	1	$ 56.07	$ 56.07	
Conductive Sheet	6021T13	1	$ 39.07	$ 39.07	
Latches	13435A31	2	$ 12.78	$ 25.56	
Aluminum Sheet	89015K59	1	$ 219.47	$ 219.47	
D-Sub Panel Mount	093-112	1	$ 20.42	$ 20.42	Parts Express
Anti-Static Wrist Strap	EC-900-022	1	$ 6.99	$ 6.99	FourAker Electronics

LED Screwdriver	EC-SD-804-1	1	$ 3.99	$ 3.99	
Color Tape	67-LT-5CP	1	$ 6.55	$ 6.55	
VmodBB - VHDC-Breadboard	Vmod-BB	1	$ 57.71	$ 57.71	Diligent
Times Mic LMR-100A-PVC	16346	39	$ 0.49	$ 19.11	The Antenna Farm
RFI RSA-3050-B	33054	2	$ 4.95	$ 9.90	
Install Conn Dual	DCI	1	$ 8.00	$ 8.00	
Shipping		1	$ 9.95	$ 9.95	
Startech Self Adhesive Cable Tie Mounts	HC102	1	$ 9.99	$ 9.99	Amazon
Thin Self-Gripping Cable Ties	-	1	$ 7.34	$ 7.34	
Metal Oxide Resistors	71-RNX0381M00DHLB	5	$ 3.31	$ 16.55	Mouser Electronics
High Frequency/RF Resistors	71-FC0603E50R0BTBST1	5	$ 2.10	$ 10.50	
High Frequency/RF Resistors	71-FC0402H75R0BTS	5	$ 4.86	$ 24.30	
Tax/Shipping		1	$ 7.99	$ 7.99	
SMA Female-Female Cable	PE3C3043-336	1	$ 258.27	$ 258.27	Pasternack

EMS Unassembled Female Connectors Futaba J	EMOM0116	3	$ 11.59	$ 34.77	TowerHobbies
EMS Unassembled Male Connectors Futaba J	EMOM0115	3	$ 10.59	$ 31.77	

VII. Conclusion

7.1 Accomplishments

The 2016 SAR team designed and implemented a far more practical, modular SAR Imager that can be foundational for future projects. Mechanically, the team reduced the weight of the frame by over 100 pounds, added mobility options and noticeably improved the stability of the overall structure. The updated component housing was designed and fabricated in such a way that this SAR system, at least aesthetically, is closer to what the actual design for market would be, while more importantly providing greater shielding against electromagnetic interference or leakage coming in or out of the housing. This component housing is also considerably safer and more mobile than the previous L shaped design, along with a greater ease of access for testing the electronic system, as well as to more easily assemble and connect the component housing with the transmit and receive horns on the actual frame. On the RF side of the project, the system can now generate its own signal for transmit and receive through use of the voltage controlled oscillator, without the need for any external signal source to do this in its stead. The signal path power has been better optimized for both transmit and receive, with less saturation and compression of components, whereas the previous year, this could be deemed questionable. Great strides with the FPGA software and coding have been made this year, with code being written to perform all of the system controls for switching between different paths, as well as between different antenna transmit and receive pairs. ADC logic has been implemented to read the voltage values coming from the IQ demodulator, representing the reflected signals out at the scene, and are converted to a digital word for use in processing on the FPGA board. Advancements with the dedicated VGA display have also been made to show progress to reaching the end goal of a standalone radar system that requires no external PC for operation, processing and

detection. There was also manual switching logic created for the antenna paths, along with some great work that has been done with reading and writing data to the on board memory of the FPGA Finally, achievements with the signal processing have certainly been attained. Specifically, improving upon the slow, archaic signal processing implementing the Discrete Fourier Transform using excel has been completely reworked. All signal processing has been successfully coded into custom MATLAB functions, allowing for computation of the Fourier Transform for any reflected signals from the system. This also includes code for calibration and error correction due to the issue of operating in the near field, rather than the far field, fixing some issues with a reflector being targeted at too extreme of an angle. In addition, logic and code has been successfully written to convert the digital data of the reflected I and Q values that was stored on the FPGA memory, and bring these values to MATLAB as a DC voltage value that can be used for processing. This MATLAB code will serve as a launching pad for any future team that will implement signal processing natively on the FPGA, which has also been started with work on FFT algorithms being researched and presented, along with VHDL code being provided in the form of an IP Core provided by Xilinx. Overall, a functioning radar system that is structurally sound has been realized, but can certainly be improved upon with future iterations of the project.

7.2 Proposed Improvements

The most important improvement the team would suggest going forward is to evaluate the design for the step-up voltage circuit and get a prototype implemented early on. Getting the FPGA to drive the SPDT switch should be a top priority since it would eliminate the reliance on an external pulse generator.

Implementing the isolation switch proposed in Section 6.2.4.3 and then evaluating its effectiveness would be a very valuable contribution to the project that can be done relatively simply and early in the semester next year.

More data needs to be taken both indoors and outdoors for each angle of arrival there is a basis function for. Once that data is in hand, its analysis should inform future decision-making regarding the scope of the project. It seems an honest evaluation of the SAR's feasibility in indoor environments is in order.

Be that as it may, subsequent teams should consider conducting signal path bench testing as early in the first semester as possible so that they can become familiarized enough with the system.

7.3 Project Future

Future iterations of the project could include replacing the plastic pyramidal horn antennas with patch antennas. Inquiries have been made, and it is possible to have patch antennas fabricated at the College of Engineering with the cooperation of certain graduate students. Designing these antennas would be a valuable learning challenge and expose undergraduates to collaboration with graduate students.

Implementing a phase slope generator using coaxial line stretchers could be considered so a target can be synthesized.

To implement a neural network in the future is to create a "smarter" system that uses pattern recognition for improved image formation. By taking a lot of different measurements with the system, for many different angles away from boresight, then implementing this into a prewritten neural network algorithm given in MATLAB, the system can learn to recognize targets at specific angles with greater accuracy. This I and Q data taken for many different scenarios will be saved into a large matrix, and then simply run into the neural network algorithm in MATLAB. This algorithm simply contains another large matrix that then gets multiplied by the test data, and will help with the pattern recognition for image formation, for any of the 16 bin angles at the scene. The senior design professor, Dr Jerris Hooker, will assist in implementing such a network.

VIII. Team

Jordan Bolduc – Jordan is graduating FSU with his B.S. in Electrical Engineering in April 2016. He is going into his professional career in the upcoming months, most likely accepting a position as an electronics engineer with the federal government, helping serve the United States Navy, contingent upon receiving his Secret security clearance. Eventually, Jordan wants to do work in signal processing with a defense contractor or an audio processing company such as Dolb

Appendix A: Test Plan Documentation

Test Identifier	Title	Description	Comments
FAM-000	Test Equipment Familiarization & Safety	Feed Pulse Generator directly to Spectrum Analyzer. Study Span, ResBW, PRF spectral lines, envelope, etc.	COMPLETE: PASS
CTRL-001	FPGA ADC Input Voltage	Apply voltage from 0 – 3.3 V to FPGA PMOD pins & verify 12-bit hex words on 7-seg display.	COMPLETE: PASS
DISP-015	Display Verification	Demonstrate using display as volt meter for the AtoD signal from test **CTRL-001**	COMPLETE: PASS
HW-016	SPDT Hardware TS	Use Pulse Gen & Spec Anny to verify switch function	COMPLETE: FAIL, Resolved 2 trials show no switching capability. Switch replaced & verified.
CTRL-002	FPGA Switching Logic	Use slider switches on FPGA board to simulate every switching state.	COMPLETE: PASS
TXRF-003, A-F	TX Output Power & Gain (Constant)	Verify RF power of each component along TX Path to PA w/constant source. No Switching, No Timing.	COMPLETE: PASS Post-Migration: PASS
TXLO-004, A-D	LO Output Power & Gain (Constant)	Verify IF power of each component along LO Path up to	COMPLETE: PASS Post-Migration: PASS

		IQ Demod w/constant source. No Switching, No Timing.	
RXRF-005, A-D	RX Power & Gain (Constant)	Verify RF power and gain from BPF to IQ Demod w/constant source. No Switching, No Timing.	COMPLETE: PASS Post-Migration: PASS
TXIF-006	SPDT Switching (Manual)	Use FPGA slider switch controls to verify both switch paths.	Need step-up circuit so FPGA output can drive switch
TXRF-007	SP4T Switching (Manual)	Use FPGA slider switch controls to verify all switch paths.	COMPLETE: PASS Post-Migration: PASS
RXRF-008	SP16T Switching (Manual)	Use FPGA slider switch controls to verify all switch paths.	COMPLETE: PASS Post-Migration: PASS
TXIF-009	VCO	Generate 5GHz, -4dBm signal from VCO.	COMPLETE: PASS VCO 1 & 2 operational
CTRL-010	FPGA Fast Pulse	Verify FPGA 50MHz, 0.333 duty operational pulse.	Timing and duty: PASS. Not enough voltage output to drive SPDT
TXIF-011	IF Output Power And Pulse Fidelity of SPDT Switch w/ VCO & FPGA fast pulse	Verify output power and pulse fidelity to TX/LO common SPDT using VCO and FPGA fast pulse.	PASS with VCO and 25MHz pulse generator
LORF-012, A-D	LO Power & Pulse Fidelity w/ VCO & FPGA fast pulse	Verify RF power & pulse fidelity along LO Path using VCO	PASS with VCO and 25MHz pulse generator

		and FPGA fast pulse.	
TXRF-013, A-J	TX Output Power & Pulse Fidelity	Verify RF power & pulse fidelity along TX Path using VCO and FPGA fast pulse	PASS with VCO and 25MHz pulse generator
RXRF-014	RX Gain & Pulse Fidelity	Verify RF power and gain to IQ Demod (including SP16T) using VCO and FPGA fast pulse.	PASS with VCO and 25MHz pulse generator
TXRX-015	Tx-Rx Delay Line Hard Loopback	Tx/Rx hard loop with delay-matched coax run & line extender.	COMPLETE: PASS Signal path verified for completeness and loss, NOT for fast pulse operation. Post-Migration: PASS
IQDM-016	IQ Demodulator Voltage Test	Verify voltage from each IQ Demod pin to FPGA PMODs w/Delay Line	COMPLETE: PASS

Appendix B: VHDL Code Modules

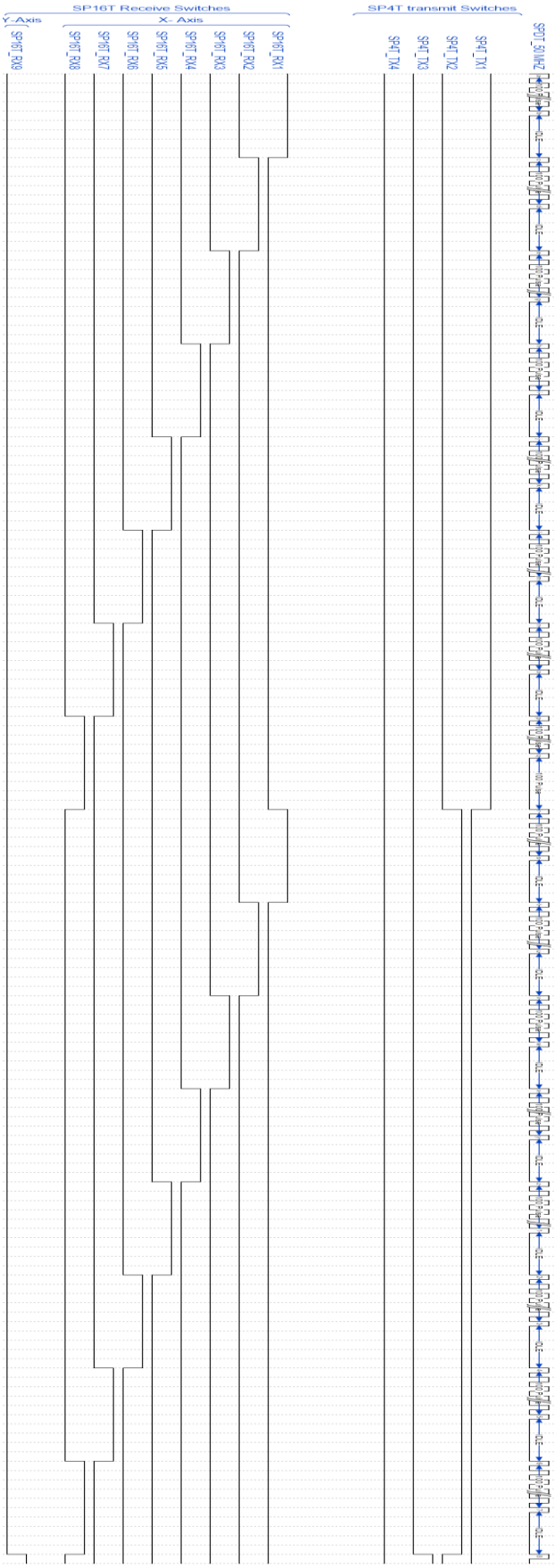

SP16T Receive Switches
Y- Axis
X- Axis
SP4T transmit Switches
SP16T_RX9
SP16T_RX8
SP16T_RX7
SP16T_RX6
SP16T_RX5
SP16T_RX4
SP16T_RX3
SP16T_RX2
SP16T_RX1
SP4T_TX4
SP4T_TX3
SP4T_TX2
SP4T_TX1
SP0T_50MHZ

Figure B.1- Switches timing diagram

```
----------------------------------------------------LIBRARY---------------------------------------------------
Library ieee;
Use ieee.std_logic_1164.all;
Use ieee.std_logic_unsigned.all;
Use ieee.std_logic_arith.all;

--use sar_design.all;

PACKAGE SAR_DESIGN is
--1.
component segdriver is
Port    (
        CLK         : In std_logic; --100 MHZ
        RST         : In std_logic; --button(0) B8
        sig_in : In Std_logic_vector(15 DOWNTO 0); -- 16 bits signal for 7 segments display
        seg         : out STD_logic_vector(6 downto 0); -- 8 bits per anodes
        an          : out std_logic_vector(3 downto 0));
end component;
--2.
component memdriver is
PORT
        (
        clk_i   : in std_logic;
```

```
        RST             : in std_logic;
        in_out          : in std_logic;

        adr_in  : in STD_LOGIC_VECTOR(22 downto 0);

  MemAdr : out std_logic_vector (23 downto 1);     -- Address
        RAM_OEb     : out std_logic;                                    --
Output Enable
        RAM_WEb     : out std_logic;                                    --
Write Enable
        RAMAdv  : out std_logic;                                    --      Address
Valid
        RAMClk  : out std_logic;                                    -- RAM clock
        RAMCre   : out std_logic;                                   --      Control
Register enable
        RAM_CEb     : out std_logic;                                    -- Chep
Enable
        RAM_LB      : out std_logic;                                    --
Lower Byte
  RAM_UB    : out std_logic;                                    -- Upper Byte
        MemDB_io    : inout std_ulogic_vector (15 downto 0);      -- Bidirectional data
        MemDB_in    : in  std_ulogic_vector (15 downto 0);
        MemDB_out: out std_ulogic_vector  (15 downto 0)
        );
end component;
--3.
        COMPONENT comp_50Mhz is
        PORT(
                CLK : IN std_logic;
                RST : IN std_logic;
                CLK_50Mhz_out : OUT std_logic
```

```
        );
    END COMPONENT;


--4.
function multiply (X,Y : std_logic_vector(12 downto 0)) return std_logic_vector;

--5.
function ADD (X,Y : std_logic_vector(24 downto 0)) return std_logic_vector;

--6.
COMPONENT data_processing is
PORT(
    clk : IN std_logic;
    enable : IN std_logic;
    MemDB_in : IN std_logic_vector(15 downto 0);
    MemDB_io : INOUT std_logic_vector(15 downto 0);
    MemAdr : OUT std_logic_vector(23 downto 1);
    RAM_OEb : OUT std_logic;
    RAM_WEb : OUT std_logic;
    RAMAdv : OUT std_logic;
    RAMClk : OUT std_logic;
    RAMCre : OUT std_logic;
    RAM_CEb : OUT std_logic;
    RAM_LB : OUT std_logic;
    RAM_UB : OUT std_logic
    );
END COMPONENT;

--7.
TYPE ONE_BY_16 is Array (1 to 16) of STD_LOGIC_vector(24 downto 0);
```

```
--Constant for Memory Address
-- i will be store 0 .. 15
-- i-bar will be stored 16 .. 31
-- q will be store 32 .. 47
-- q-bar will be stored 48 .. 63
TYPE MEM_ADDRESS is Array (1 to 64) of STD_LOGIC_vector(23 downto 0);
constant MEM_ADR : MEM_ADDRESS := (
--I
	x"000000",x"000001",x"000002",x"000003",x"000004",x"000005",x"000006",x"000007",x"000008",x"000009",x"00000a",x"00000b",x"00000c",x"00000d",
	x"00000e",x"00000f",
--I-bar
	x"000010",x"000011",x"000012",x"000013",x"000014",x"000015",x"000016",x"000017",x"000018",x"000019",x"00001a",x"00001b",x"00001c",x"00001d",
	x"00001e",x"00001f",
--Q
	x"000020",x"000021",x"000022",x"000023",x"000024",x"000025",x"000026",x"000027",x"000028",x"000029",x"00002a",x"00002b",x"00002c",x"00002d",
	x"00002e",x"00002f",
--Q- bar
	x"000030",x"000031",x"000032",x"000033",x"000034",x"000035",x"000036",x"000037",x"000038",x"000039",x"00003a",x"00003b",x"00003c",x"00003d",
	x"00003e",x"00003f"
	);

--Constant for BASIS VALUES
TYPE BASIS is Array (1 to 16, 1 to 16) of STD_LOGIC_vector(15 downto 0);
constant Basis_real : Basis := (
--	("xxxx" , "xxxx", "xxxx", "xxxx", "xxxx", "xxxx",
	"xxxx", "xxxx", "xxxx", "xxxx", "xxxx", "xxxx",
	"xxxx", "xxxx", "xxxx", "xxxx")
```

```
        (x"14C2"    ,x"047D"    ,x"140f"    ,x"0379"    ,x"12c4"    ,x"01f2"
        ,x"1110"    ,x"0022"    ,x"00cb"    ,x"11b3"    ,x"0289"    ,x"1349"
        ,x"03e7"    ,x"1462"    ,x"04b2"    ,x"14d7"    ),
        (x"140c"    ,x"01e7"    ,x"00dd"    ,x"135a"    ,x"04b9"    ,x"148a"    ,x"02d9"    ,x"1038"
        ,x"127d" ,x"045d" ,x"14ce" ,x"03a6" ,x"114b" ,x"1180"    ,x"03c9" ,x"14d4" ),
        (x"12b9"    ,x"11cc"    ,x"04bb"    ,x"1385"    ,x"10ca"    ,x"0466"
        ,x"1427"    ,x"0042"    ,x"03dc"    ,x"1498"    ,x"014a"    ,x"0324"
        ,x"14d2"    ,x"0243"    ,x"0246"    ,x"14d2"    ),
        (x"10fb"    ,x"1474"    ,x"02c6"    ,x"0355"    ,x"141f"    ,x"11ad"
        ,x"04cb"    ,x"1044"    ,x"14b1"    ,x"0227"    ,x"03d1"    ,x"13b2"
        ,x"1255"    ,x"04a2"    ,x"0075"    ,x"14d2"    ),
        (x"00ea"    ,x"1481"    ,x"129f"    ,x"0382"    ,x"03f2"    ,x"1205"
        ,x"14b6"    ,x"003c"    ,x"04cc"    ,x"0194"    ,x"1434"    ,x"132c"
        ,x"0300"    ,x"044e"    ,x"1160"    ,x"14d4"    ),
        (x"02ac"    ,x"11e6"    ,x"14c5"    ,x"135d"    ,x"010e"    ,x"0487"
        ,x"03f0"    ,x"1030"    ,x"1425"    ,x"1463"    ,x"10b3"    ,x"039e"
        ,x"04af"    ,x"018d"    ,x"12fa"    ,x"14d6"    ),
        (x"0405"    ,x"01d3"    ,x"10fe"    ,x"1379"    ,x"14c5"    ,x"1472"
        ,x"129c"    ,x"001e"    ,x"02cd"    ,x"0488"    ,x"04b8"    ,x"034c"
        ,x"00c1"    ,x"120c"    ,x"1427"    ,x"14d8"    ),
        (x"04c0"    ,x"0479" ,x"0405"    ,x"0369"    ,x"02ac"    ,x"01d3"    ,x"00e9"
        ,x"100b"    ,x"10fe"    ,x"11e7"    ,x"12be"    ,x"1379"    ,x"1411"
        ,x"1482"    ,x"14c5"    ,x"14d9" ),
-- symmetry
        (x"04c0"    ,x"0479" ,x"0405"    ,x"0369"    ,x"02ac"    ,x"01d3"    ,x"00e9"
        ,x"100b"    ,x"10fe"    ,x"11e7"    ,x"12be"    ,x"1379"    ,x"1411"
        ,x"1482"    ,x"14c5"    ,x"14d9" ),
        (x"0405"    ,x"01d3"    ,x"10fe"    ,x"1379"    ,x"14c5"    ,x"1472"
        ,x"129c"    ,x"001e"    ,x"02cd"    ,x"0488"    ,x"04b8"    ,x"034c"
        ,x"00c1"    ,x"120c"    ,x"1427"    ,x"14d8"    ),
```

```
        (x"02ac"    ,x"11e6"    ,x"14c5"    ,x"135d"    ,x"010e"    ,x"0487"
        ,x"03f0"    ,x"1030"    ,x"1425"    ,x"1463"    ,x"10b3"    ,x"039e"
        ,x"04af"    ,x"018d"    ,x"12fa"    ,x"14d6"    ),
        (x"00ea"    ,x"1481"    ,x"129f"    ,x"0382"    ,x"03f2"    ,x"1205"
        ,x"14b6"    ,x"003c"    ,x"04cc"    ,x"0194"    ,x"1434"    ,x"132c"
        ,x"0300"    ,x"044e"    ,x"1160"    ,x"14d4"    ),
        (x"10fb"    ,x"1474"    ,x"02c6"    ,x"0355"    ,x"141f"    ,x"11ad"
        ,x"04cb"    ,x"1044"    ,x"14b1"    ,x"0227"    ,x"03d1"    ,x"13b2"
        ,x"1255"    ,x"04a2"    ,x"0075"    ,x"14d2"    ),
        (x"12b9"    ,x"11cc"    ,x"04bb"    ,x"1385"    ,x"10ca"    ,x"0466"
        ,x"1427"    ,x"0042"    ,x"03dc"    ,x"1498"    ,x"014a"    ,x"0324"
        ,x"14d2"    ,x"0243"    ,x"0246"    ,x"14d2"    ),
        (x"140c"    ,x"01e7"    ,x"00dd"    ,x"135a"    ,x"04b9"    ,x"148a"    ,x"02d9"    ,x"1038"
        ,x"127d" ,x"045d" ,x"14ce" ,x"03a6" ,x"114b" ,x"1180"    ,x"03c9" ,x"14d4" ),
        (x"14C2"    ,x"047D"    ,x"140f"    ,x"0379"    ,x"12c4"    ,x"01f2"
        ,x"1110"    ,x"0022"    ,x"00cb"    ,x"11b3"    ,x"0289"    ,x"1349"
        ,x"03e7"    ,x"1462"    ,x"04b2"    ,x"14d7"    )
);

constant Basis_ima : Basis := (
--      ("xxxx"     ,     "xxxx",     "xxxx",     "xxxx",     "xxxx",     "xxxx",
        "xxxx",     "xxxx",     "xxxx",     "xxxx",     "xxxx",     "xxxx",
        "xxxx",     "xxxx",     "xxxx",     "xxxx")
        (x"0414"    ,x"0468"    ,x"00af"    ,x"13ac"    ,x"14a6"    ,x"115b"
        ,x"032f"    ,x"04cb"    ,x"01ff"    ,x"12a4"    ,x"14d9"    ,x"129a"
        ,x"0209"    ,x"04cd"    ,x"0326"    ,x"1166"    ),
        (x"11ee"    ,x"01d2"    ,x"12a7"    ,x"0361" ,x"13fc"    ,x"0470"    ,x"14bb"
        ,x"04d8"    ,x"14c9"    ,x"048a"    ,x"1422"    ,x"0391"    ,x"12e0"
        ,x"0212"    ,x"1132"    ,x"0045"    ),
```

```
(x"12ac" ,x"0475" ,x"14c5" ,x"0381" ,x"1115" ,x"11b4"
,x"03eb" ,x"14d8" ,x"0429" ,x"121b" ,x"10a7" ,x"0330"
,x"14ad" ,x"049c" ,x"1307" ,x"006f" ),
(x"1404" ,x"0480" ,x"110b" ,x"1356" ,x"04c8" ,x"1209"
,x"1281" ,x"04d7" ,x"12ef" ,x"118e" ,x"04ac" ,x"13b1"
,x"1088" ,x"0449" ,x"1448" ,x"0084" ),
(x"14c0" ,x"01ea" ,x"03f9" ,x"1385" ,x"128f" ,x"048c"
,x"00b8" ,x"14d8" ,x"013b" ,x"0457" ,x"12fd" ,x"1324"
,x"0440" ,x"016c" ,x"14d4" ,x"0085" ),
(x"14c3" ,x"11cd" ,x"0414" ,x"0357" ,x"12d1" ,x"1469"
,x"0125" ,x"04d7" ,x"00af" ,x"1496" ,x"126c" ,x"03aa"
,x"03ce" ,x"123b" ,x"14a7" ,x"0078" ),
(x"140c" ,x"1476" ,x"10e1" ,x"037e" ,x"04ba" ,x"01b9"
,x"12d4" ,x"14d9" ,x"1285" ,x"0210" ,x"04cc" ,x"0339"
,x"113d" ,x"1498" ,x"13d4" ,x"005e" ),
(x"12b5" ,x"147e" ,x"14bf" ,x"1363" ,x"10e0" ,x"01ef"
,x"0416" ,x"04d8" ,x"03f4" ,x"01b7" ,x"111b" ,x"138d"
,x"14ca" ,x"1466" ,x"1282" ,x"003c" ),
(x"10f4" ,x"11de" ,x"12b5" ,x"1372" ,x"140c" ,x"147e"
,x"14c3" ,x"14d9" ,x"14bf" ,x"1476" ,x"1400" ,x"1363"
,x"12a4" ,x"11cb" ,x"10e0" ,x"0014" ),
(x"00f3" ,x"01dd" ,x"02b4" ,x"0371" ,x"040b" ,x"047d"
,x"04c2" ,x"04d8" ,x"04be" ,x"0475" ,x"03ff" ,x"0362"
,x"02a3" ,x"01ca" ,x"00df" ,x"1015" ),
(x"02b4" ,x"047d" ,x"04be" ,x"0362" ,x"00df" ,x"11f0"
,x"1417" ,x"14d9" ,x"13f5" ,x"11b8" ,x"011a" ,x"038c"
,x"04c9" ,x"0465" ,x"0281" ,x"103d" ),
(x"040b" ,x"0475" ,x"00e0" ,x"137f" ,x"14bb" ,x"11ba"
,x"02d3" ,x"04d8" ,x"0284" ,x"1211" ,x"14cd" ,x"133a"
,x"013c" ,x"0497" ,x"03d3" ,x"105f" ),
```

```
        (x"04c2"    ,x"01cc"    ,x"1415"    ,x"1358"    ,x"02d0"    ,x"0468"
        ,x"1126"    ,x"14d8"    ,x"10b0"    ,x"0495"    ,x"026b"    ,x"13ab"
        ,x"13cf"    ,x"023a"    ,x"04a6"    ,x"1079"    ),
        (x"04bf"    ,x"11eb"    ,x"13fa"    ,x"0384"    ,x"028e"    ,x"148d"
        ,x"10b9"    ,x"04d7"    ,x"113c"    ,x"1458"    ,x"02fc"    ,x"0323"
        ,x"1441"    ,x"116d"    ,x"04d3"    ,x"1086"    ),
        (x"0403"    ,x"1481"    ,x"010a"    ,x"0355"    ,x"14c9"    ,x"0208"
        ,x"0280"    ,x"14d8"    ,x"02ee"    ,x"018d"    ,x"14ad"    ,x"03b0"
        ,x"0087"    ,x"144a"    ,x"0447"    ,x"1085"    ),
        (x"02ab"    ,x"1476"    ,x"04c4"    ,x"1382"    ,x"0114"    ,x"01b3"
        ,x"13ec"    ,x"04d7"    ,x"142a"    ,x"021a"    ,x"00a6"    ,x"1331"
        ,x"04ac"    ,x"149d"    ,x"0306"    ,x"1070" )
);


--8. Analog to digital conversion with negative reference
COMPONENT A_TO_D is
PORT(
        CLK : IN std_logic;
        RST : IN std_logic;
        SDATA1 : IN std_logic;
        SDATA2 : IN std_logic;
        SDATA3 : IN std_logic;
        SDATA4 : IN std_logic;
        START : IN std_logic;
        SCLK1 : OUT std_logic;
        nCS1 : OUT std_logic;
        SCLK2 : OUT std_logic;
        nCS2 : OUT std_logic;
        I               : out std_logic_vector(11 downto 0);
        I_bar   : out std_logic_vector(11 downto 0);
```

```
        Q                : out std_logic_vector(11 downto 0);
        Q_bar   :out std_logic_vector(11 downto 0);
        DONE1 : OUT std_logic;
        DONE2 : OUT std_logic
        );
END COMPONENT;
--9. DEBOUNCE circuit is important when using switches and push button
component Debounce is
   Port ( INP : in  STD_LOGIC_VECTOR (12 downto 0);
        CCLK : in  STD_LOGIC;
                        rst      : in STD_LOGIC;
        OUTP : out  STD_LOGIC_VECTOR (12 downto 0));
end component;

end SAR_DESIGN;



Package body SAR_DESIGN is
-- MULTIPLICATION
        function multiply (X,Y : std_logic_vector(12 downto 0)) return std_logic_vector is
                variable result : std_logic_vector(24 downto 0);
        begin
                result(24):= x(12) xor y(12);
                result(23 downto 0):= x(11 downto 0)* y(11 downto 0);
        return result(24 downto 0);
        end multiply;

-- ADDITION
        function ADD (X,Y : std_logic_vector(24 downto 0)) return std_logic_vector is
                variable result : std_logic_vector(24 downto 0);
        begin
```

```
            IF (X(24) = Y(24)) THEN
                RESULT := X(24) & (X(23 DOWNTO 0) + Y(23 DOWNTO 0));
            ELSIF (X(23 DOWNTO 0) >= Y(23 DOWNTO 0)) THEN
                RESULT := X(24) & (X(23 DOWNTO 0) - Y(23 DOWNTO 0));
            ELSE
                RESULT := Y(24) & (Y(23 DOWNTO 0) - X(23 DOWNTO 0));
            END IF;
        return result(24 downto 0);
        end ADD;



end SAR_DESIGN;
--------------------------------ANALOG TO DIGITAL CONVERTER---------------------------------


library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use work.all;
entity A_TO_D is
  Port    (
  --General usage
    CLK      : in std_logic;
    RST      : in std_logic;

  --Pmod interface signals for first adc
    SDATA1   : in std_logic;
    SDATA2   : in std_logic;
    SCLK1    : out std_logic;
    nCS1     : out std_logic;
```

```
 --Pmod interface signals for second adc
   SDATA3   : in std_logic;
   SDATA4   : in std_logic;
   SCLK2    : out std_logic;
   nCS2     : out std_logic;

   START    : in std_logic;
         --ouput
          I                    : out std_logic_vector(11 downto 0);
          I_bar    : out std_logic_vector(11 downto 0);
          Q                    : out std_logic_vector(11 downto 0);
          Q_bar  : out std_logic_vector(11 downto 0);
   DONE1    : out std_logic;   -- adc1
   DONE2    : out std_logic            -- adc2

         );

end A_TO_D ;

architecture AD of A_TO_D is

type states is (Idle,
            ShiftIn,
            SyncData);
       signal current_state : states;
       signal next_state    : states;

       signal temp1        : std_logic_vector(15 downto 0);
       signal temp2        : std_logic_vector(15 downto 0);

       signal temp3        : std_logic_vector(15 downto 0);
```

```
        signal temp4        : std_logic_vector(15 downto 0);



        signal clk_div      : std_logic;
        signal clk_counter   : std_logic_vector(1 downto 0);
        signal shiftCounter  : std_logic_vector(3 downto 0) := x"0";
        signal enShiftCounter: std_logic;
        signal enParalelLoad : std_logic;

                        signal DATA1                : std_logic_vector(11 downto 0);
                        signal DATA2                : std_logic_vector(11 downto 0);

                        signal DATA3                : std_logic_vector(11 downto 0);
                        signal DATA4                : std_logic_vector(11 downto 0);


begin

I<=DATA1;
I_bar<=DATA2;
Q<=DATA3;
Q_bar<=DATA4;

--------------------------------------------------------------------------------
-- Title         :     clock divider process
--
-- Description    : This is the process that will divide the 100 MHz clock
--                 down to a clock speed of 25 MHz to drive the ADC7476 chip.
--------------------------------------------------------------------------------
      clock_divide : process(rst,clk)
      begin
```

```
        if rst = '1' then
            clk_counter <= "00";
        elsif (clk = '1' and clk'event) then
            clk_counter <= clk_counter + '1';
        end if;
    end process;

    clk_div <= clk_counter(1);
    SCLK1 <=  not clk_counter(1);

    clk_div <= clk_counter(1);
    SCLK2 <=  not clk_counter(1);

--------------------------------------------------------------------------------
--
-- Title      : counter
--
-- Description: This is the process were the converted data will be colected and
--              output.When the enShiftCounter is activated, the 16-bits of data
--              from the ADC7476 chips will be shifted inside the temporary
--              registers. A 4-bit counter is used to keep shifting the data
--              inside temp1 and temp2 for 16 clock cycles. When the enParalelLoad
--              signal is generated inside the SyncData state, the converted data
--              in the temporary shift registers will be placed on the outputs
--              DATA1 and DATA2.
--
--------------------------------------------------------------------------------

counter : process(clk_div, enParalelLoad, enShiftCounter)
    begin
        if (clk_div = '1' and clk_div'event) then
```

```
            if (enShiftCounter = '1') then
               temp1 <= temp1(14 downto 0) & SDATA1;
                temp2 <= temp2(14 downto 0) & SDATA2;
                 temp3 <= temp3(14 downto 0) & SDATA3;
                  temp4 <= temp4(14 downto 0) & SDATA4;

               shiftCounter <= shiftCounter + '1';
            elsif (enParalelLoad = '1') then
               shiftCounter <= "0000";
               DATA1 <= temp1(11 downto 0);
                DATA2 <= temp2(11 downto 0);
                 DATA3 <= temp3(11 downto 0);
                  DATA4 <= temp4(11 downto 0);

            end if;
         end if;
      end process;

-------------------------------------------------------------------------------
--
-- Title      : Finite State Machine
--
-- Description: This 3 processes represent the FSM that contains three states.
--              The first state is the Idle state in which a temporary registers
--              are assigned the updated value of the input "DATA1" and "DATA2".
--              The next state is the ShiftIn state where the 16-bits of data
--              from each of the ADCS7476 chips are left shifted in the temp1
--              and temp2 shift registers. The third state, SyncData drives the
--              output signal nCS high for 1 clock period maintainig nCS high
--              also in the Idle state telling the ADCS7476 to mark the end of
```

```
--              the conversion.
-- Notes:       The data will change on the lower edge of the clock signal. There
--              is also an asynchronous reset that will reset all signals to
--              their original state.
--
----------------------------------------------------------------------------------


----------------------------------------------------------------------------------
--
-- Title      : SYNC_PROC
--
-- Description: This is the process were the states are changed synchronously. At
--              reset the current state becomes Idle state.
--
----------------------------------------------------------------------------------
SYNC_PROC: process (clk_div, rst)
  begin
    if (clk_div'event and clk_div = '1') then
      if (rst = '1') then
        current_state <= Idle;
      else
        current_state <= next_state;
      end if;
    end if;
  end process;


----------------------------------------------------------------------------------
--
-- Title      : OUTPUT_DECODE
--
-- Description: This is the process were the output signals are generated
```

```
--          unsynchronously based on the state only (Moore State Machine).
--
-------------------------------------------------------------------------------
OUTPUT_DECODE: process (current_state)
  begin
    if current_state = Idle then
        enShiftCounter <='0';
        DONE1 <='1';
        DONE2 <='1';

                                nCS1 <='1';
                                nCS2 <='1';

        enParalelLoad <= '0';
      elsif current_state = ShiftIn then
        enShiftCounter <='1';
        DONE1 <='0';
        nCS1 <='0';

        DONE2 <='0';
        nCS2 <='0';

        enParalelLoad <= '0';
      else --if current_state = SyncData then
        enShiftCounter <='0';
        DONE1 <='0';
        nCS1 <='1';

        DONE2 <='0';
        nCS2 <='1';
```

```
            enParalelLoad <= '1';
        end if;
    end process;

-------------------------------------------------------------------------------
--
-- Title      : NEXT_STATE_DECODE
--
-- Description: This is the process were the next state logic is generated
--              depending on the current state and the input signals.
--
-------------------------------------------------------------------------------
    NEXT_STATE_DECODE: process (current_state, START, shiftCounter)
    begin

      next_state <= current_state;  -- default is to stay in current state

      case (current_state) is
        when Idle =>
          if START = '1' then
            next_state <= ShiftIn;
          end if;
        when ShiftIn =>
          if shiftCounter = x"F" then
            next_state <= SyncData;
          end if;
        when SyncData =>
          if START = '0' then
          next_state <= Idle;
          end if;
        when others =>
```

```
        next_state <= Idle;
    end case;
  end process;



end AD;
----------------------------------------------------SEG DRIVER-----------------------------------------------
Library ieee;
Use ieee.std_logic_1164.all;
Use ieee.std_logic_unsigned.all;
Use ieee.std_logic_arith.all;

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity segdriver is
PORT  (
                CLK             : In std_logic; --100 MHZ
                RST             : In std_logic; --button(0) B8
                sig_in  : In Std_logic_vector(15 DOWNTO 0); -- 16 bits signal for 7 segments display
                seg             : out STD_logic_vector(6 downto 0); -- 8 bits per anodes
                an                      : out std_logic_vector(3 downto 0));
end segdriver;

architecture Behavioral of segdriver is
signal KHZpulse, syncr1                                                 : std_logic ;
                                                                                --1khz signal
signal clk_counter                                                      : std_logic_vector(20 downto
0);                                                             -- counter to make the 1khz signal
```

```
signal an_counter                                                    :    std_logic_vector(1
downto 0);                                                    -- counter to help switch the anodes
and seg data
signal char0,char1,char2,char3, indecode      :std_logic_vector       (3 downto 0);
                                    -- holding data value for the 15 bits inputs

begin

Process(CLK, sig_in, RST)
Begin
if (RST ='1')then
char0 <= (others=>'0');
char1 <= (others=>'0');
char2 <= (others=>'0');
char3 <= (others=>'0');
elsif rising_edge(clk) then
char0 <=sig_in(3 downto 0);
char1 <=sig_in(7 downto 4);
char2 <=sig_in(11 downto 8);
char3 <=sig_in(15 downto 12);
end if;
End Process;


---CREATING a 1 KHZ clock base on the 100MHZ signal
--The 7segment work best at 60 hz to 1khz
-- to get 1khz clk_counter needs to be 100,000
Process(CLK,RST, KHZpulse)
Begin
if (RST ='1')then
        clk_counter <= (others =>'0');
```

```
elsif rising_edge(clk) then
        if(syncr1 = '1') then
                clk_counter <= (others =>'0');
        else
                clk_counter <= clk_counter + 1;
        end if;
end if;
End Process;



syncr1  <='1' when (clk_counter="00001100001101010000") else '0';
KHZpulse<=syncr1;
----------lets switch anodes at 1KHz
Process(CLK,RST,KHZpulse)
Begin
if rst = '1' then
        an_counter <= (others=>'0');
elsif rising_edge(CLK) then
        if(KHZpulse = '1') then
                an_counter <= an_counter + 1;
        end if;
end if;
end process;

with an_counter select
an                  <= "1110" when "00",
                               "1101" when "01",
                               "1011" when "10",
                               "0111" when "11",
                               "0000" when others;
```

```
with an_counter select
indecode <= char0 when "00",
                    char1 when "01",
                    char2 when "10",
                    char3 when "11",
                    "0000" when others;

with indecode select
                    seg             <="1000000" when x"0",
                                        "1111001" when x"1",
                                        "0100100" when x"2",
                                        "0110000" when x"3",
                                        "0011001" when x"4",
                                        "0010010" when x"5",
                                        "0000010" when x"6",
                                        "1111000" when x"7",
                                        "0000000" when x"8",
                                        "0010000" when x"9",
                                        "0001000" when x"A",
                                        "0000011" when x"B",
                                        "1000110" when x"C",
                                        "0100001" when x"D",
                                        "0000110" when x"E",
                                        "0001110" when x"F",
                                        "1111111" when others;



end Behavioral;
----------------------------------------------Memory Driver----------------------------------------
```

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.numeric_std.all;
use work.sar_design.all;

library UNISIM;
use UNISIM.VComponents.all;


entity memdriver is
PORT
        (
        clk_i    : in std_logic;
        RST              : in std_logic;
        in_out           : in std_logic;

        adr_in  : in STD_LOGIC_VECTOR(22 downto 0);

  MemAdr : out std_logic_vector (23 downto 1);      -- Address
        RAM_OEb      : out std_logic;                                    --
Output Enable
        RAM_WEb      : out std_logic;                                    --
Write Enable
        RAMAdv  : out std_logic;                                 --       Address
Valid
        RAMClk  : out std_logic;                                 -- RAM clock
        RAMCre   : out std_logic;                                --       Control
Register enable
        RAM_CEb      : out std_logic;                                    -- Chep
Enable
```

```
        RAM_LB      : out std_logic;                                    --
Lower Byte
  RAM_UB    : out std_logic;                                          -- Upper Byte
        MemDB_io    : inout std_ulogic_vector (15 downto 0);      -- Bidirectional data
        MemDB_in    : in  std_ulogic_vector (15 downto 0);
        MemDB_out: out std_ulogic_vector  (15 downto 0)
        );
end memdriver;

architecture Behavioral of memdriver is

begin

iobuf_inst: IOBUF
generic map(
        DRIVE =>12,
        IOSTANDARD=>"DEFAULT",
        SLEW=>"SLOW")
Port map(
        O=>MemDB_out(0),
        IO=>MemDB_io(0),
        I=> MemDB_in(0),
        T=>in_out               -- high: input  low: output
);

iobuf_inst1: IOBUF
generic map(
        DRIVE =>12,
        IOSTANDARD=>"DEFAULT",
        SLEW=>"SLOW")
Port map(
```

```
        O=>MemDB_out(1),
        IO=>MemDB_io(1),
        I=> MemDB_in(1),
        T=>in_out               -- high: input  low: output
);

iobuf_inst2: IOBUF
generic map(
        DRIVE =>12,
        IOSTANDARD=>"DEFAULT",
        SLEW=>"SLOW")
Port map(
        O=>MemDB_out(2),
        IO=>MemDB_io(2),
        I=> MemDB_in(2),
        T=>in_out               -- high: input  low: output
);

iobuf_inst3: IOBUF
generic map(
        DRIVE =>12,
        IOSTANDARD=>"DEFAULT",
        SLEW=>"SLOW")
Port map(
        O=>MemDB_out(3),
        IO=>MemDB_io(3),
        I=> MemDB_in(3),
        T=>in_out               -- high: input  low: output
);

iobuf_inst4: IOBUF
```

```
generic map(
        DRIVE =>12,
        IOSTANDARD=>"DEFAULT",
        SLEW=>"SLOW")
Port map(
        O=>MemDB_out(4),
        IO=>MemDB_io(4),
        I=> MemDB_in(4),
        T=>in_out               -- high: input  low: output
);

iobuf_inst5: IOBUF
generic map(
        DRIVE =>12,
        IOSTANDARD=>"DEFAULT",
        SLEW=>"SLOW")
Port map(
        O=>MemDB_out(5),
        IO=>MemDB_io(5),
        I=> MemDB_in(5),
        T=>in_out               -- high: input  low: output
);

iobuf_inst6: IOBUF
generic map(
        DRIVE =>12,
        IOSTANDARD=>"DEFAULT",
        SLEW=>"SLOW")
Port map(
        O=>MemDB_out(6),
        IO=>MemDB_io(6),
```

```
        I=> MemDB_in(6),
        T=>in_out               -- high: input  low: output
);

iobuf_inst7: IOBUF
generic map(
        DRIVE =>12,
        IOSTANDARD=>"DEFAULT",
        SLEW=>"SLOW")
Port map(
        O=>MemDB_out(7),
        IO=>MemDB_io(7),
        I=> MemDB_in(7),
        T=>in_out               -- high: input  low: output
);

iobuf_inst8: IOBUF
generic map(
        DRIVE =>12,
        IOSTANDARD=>"DEFAULT",
        SLEW=>"SLOW")
Port map(
        O=>MemDB_out(8),
        IO=>MemDB_io(8),
        I=> MemDB_in(8),
        T=>in_out               -- high: input  low: output
);

iobuf_inst9: IOBUF
generic map(
        DRIVE =>12,
```

```
        IOSTANDARD=>"DEFAULT",
        SLEW=>"SLOW")
Port map(
        O=>MemDB_out(9),
        IO=>MemDB_io(9),
        I=> MemDB_in(9),
        T=>in_out               -- high: input  low: output
);
iobuf_inst10: IOBUF
generic map(
        DRIVE =>12,
        IOSTANDARD=>"DEFAULT",
        SLEW=>"SLOW")
Port map(
        O=>MemDB_out(10),
        IO=>MemDB_io(10),
        I=> MemDB_in(10),
        T=>in_out               -- high: input  low: output
);
iobuf_inst11: IOBUF
generic map(
        DRIVE =>12,
        IOSTANDARD=>"DEFAULT",
        SLEW=>"SLOW")
Port map(
        O=>MemDB_out(11),
        IO=>MemDB_io(11),
        I=> MemDB_in(11),
        T=>in_out               -- high: input  low: output
);
iobuf_inst12: IOBUF
```

```
generic map(
        DRIVE =>12,
        IOSTANDARD=>"DEFAULT",
        SLEW=>"SLOW")
Port map(
        O=>MemDB_out(12),
        IO=>MemDB_io(12),
        I=> MemDB_in(12),
        T=>in_out               -- high: input  low: output
);
iobuf_inst13: IOBUF
generic map(
        DRIVE =>12,
        IOSTANDARD=>"DEFAULT",
        SLEW=>"SLOW")
Port map(
        O=>MemDB_out(13),
        IO=>MemDB_io(13),
        I=> MemDB_in(13),
        T=>in_out               -- high: input  low: output
);
iobuf_inst14: IOBUF
generic map(
        DRIVE =>12,
        IOSTANDARD=>"DEFAULT",
        SLEW=>"SLOW")
Port map(
        O=>MemDB_out(14),
        IO=>MemDB_io(14),
        I=> MemDB_in(14),
        T=>in_out               -- high: input  low: output
```

```
);
iobuf_inst15: IOBUF
generic map(
        DRIVE =>12,
        IOSTANDARD=>"DEFAULT",
        SLEW=>"SLOW")
Port map(
        O=>MemDB_out(15),
        IO=>MemDB_io(15),
        I=> MemDB_in(15),
        T=>in_out               -- high: input  low: output
);


RAMAdv <= '0';
RAMClk <= clk_i;
RAMCre <= '0';
RAM_CEb <= '0';
Ram_LB <='0';
Ram_UB <='0';

process (CLK_i, rst)
begin
if Rst= '1' then
MemAdr <= (others =>'0');
elsif rising_edge(CLK_i) then
memAdr <= ("00000000000000000000000" or Adr_in);
end if;
end process;

---read and write control
```

```
Ram_web <= in_out;          -- enable or disable writing
ram_oeb <= not in_out;      -- enable or disable output.



end Behavioral;
----------------------------------------------TOP_RANGE----------------------------------------------
----------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:    12:05:25 04/04/2016
-- Design Name:
-- Module Name:    Top_range - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.all;
library sar_design;
use sar_design.SAR_DESIGN.all;
```

```
entity Top_range is
port(
        led             :       out std_logic_vector(7 downto 0); --SATUS MODE
        clk             :       in std_logic;
        CHANNEL         :       in std_logic_vector(3 DOWNTO 0); -- which channel is enable on 7 seg display I, I_bar,Q, Q_bar
        rst             :       in std_logic;
        sw              :       in std_logic_vector (5 downto 0);
--      spdt            :       out std_logic; not used yet because of failure of reliable fast pulse square wave;
        sp4t            :       out std_logic_vector(3 downto 0);
        sp16t           :       out std_logic_vector(15 downto 0);

-----A_D_C PORT----------------------------------------------------------------------
        SDATA1          : IN std_logic;         --I
        SDATA2          : IN std_logic;         --I-BAR
        SDATA3          : IN std_logic;         --Q
        SDATA4          : IN std_logic;         --Q-BAR
        SCLK1           : OUT std_logic;
        nCS1            : OUT std_logic;
        SCLK2           : OUT std_logic;
        nCS2            : OUT std_logic;

----SEGDRIVER-----------------------------------------------------------------------
        seg             : out STD_logic_vector(6 downto 0);
        an              : out std_logic_vector(3 downto 0);

---MEMDRIVER-------------------------------------------------------------------------
        mem_en          : in std_logic;
        MemAdr          : out std_logic_vector (23 downto 1);           -- Address
```

```
        RAM_OEb         : out std_logic;
        -- Output Enable
        RAM_WEb         : out std_logic;
        -- Write Enable
        RAMAdv          : out std_logic;
        -- Address Valid
        RAMClk          : out std_logic;
        -- RAM clock
        RAMCre      : out std_logic;                                    --
Control Register enable
        RAM_CEb         : out std_logic;
        -- Chep Enable
        RAM_LB          : out std_logic;
        -- Lower Byte
  RAM_UB          : out std_logic;                                      --
Upper Byte
        MemDB                   : inout std_ulogic_vector (15 downto 0)-- Bidirectional data
);

end Top_range;

architecture Behavioral of Top_range is


TYPE MODE is (Debug, real_time, reset, idle_debug,idle_real_time, test_idle,
                                Memory,  memory_idle,  memory_adc,  LOAD_I,
LOAD_I_BAR,LOAD_Q, LOAD_Q_BAR);                                         -- debug mode
when switches can be used else real mode
SIGNAL pr_state, nx_state       : Mode;

SIGNAL TX_RX:STD_LOGIC_VECTOR(5 DOWNTO 0); -- tell which transmit receive pair is active
```

```
SIGNAL SEG_DATA: std_logic_vector(15 downto 0);                    -- data to go to the 7 segment display

SIGNAL ADC_START: STD_LOGIC; -- CONTROL THE STARTING OF THE ADC : ACTIVE HIGH
SIGNAL ADC_DONE1,ADC_DONE2: STD_LOGIC;-- CONTROL WHEN ADC1 AND ADC2 ARE DONE;
SIGNAL ADC_DATA1,ADC_DATA2,ADC_DATA3,ADC_DATA4: STD_LOGIC_VECTOR(11 downto 0);-- value out of the ADC1 and ADC2

SIGNAL MEM_CE:STD_LOGIC;-- enable chip or disable chip
SIGNAL MEM_RW:STD_LOGIC; -- read or write from/to the memory
SIGNAL MEM_ADDR: STD_LOGIC_VECTOR(22 downto 0); -- address to memory
SIGNAL MEM_I_DATA,MEM_O_DATA:STD_ULOGIC_VECTOR(15 downto 0);

TYPE DATA16 is Array (1 to 16) of STD_LOGIC_vector(11 downto 0);
SIGNAL I,I_bar,Q,Q_bar : DATA16;

SIGNAL CNT: integer range 1 to 10_000_000;
Signal MEM_hold: integer range 1 to 10_000;

SIGNAL ANTENNA: STD_LOGIC_VECTOR(5 downto 0);
SIGNAL INDEX: integer range 1 to 17;
SIGNAL ANTENNA_REAL_TIME_CLK:STD_LOGIC;
SIGNAL MEG_50HZ: STD_LOGIC;

begin


------COMPONENT INITIALIZATION----------------------------------------
Inst_A_TO_D: A_TO_D PORT MAP(
```

```
        CLK => CLK,
        RST => RST ,
        SDATA1 => SDATA1 ,
        SDATA2 => SDATA2,
        SCLK1 => SCLK1,
        nCS1 => NCS1,
        SDATA3 => SDATA3 ,
        SDATA4 => SDATA4,
        SCLK2 => SCLK2,
        nCS2 => NCS2,
        I=>ADC_DATA1,
        I_bar=>ADC_DATA2,
        Q=>ADC_DATA3,
        Q_bar=>ADC_DATA4,
        START => ADC_START,
        DONE1 => ADC_DONE1,
        DONE2 => ADC_DONE2
);

Inst_segdriver: segdriver PORT MAP(
        CLK =>CLK,
        RST =>RST,
        sig_in=>SEG_DATA,
        seg=> seg,
        an      =>an
);
Inst_memdriver: memdriver PORT MAP(
        clk_i => MEG_50HZ,
        RST=>RST,
        in_out=>'0',
        adr_in=> MEM_ADDR,
```

```
  MemAdr=>MemAdr,
        RAM_OEb=>RAM_OEb,
        RAM_WEb=>RAM_WEB,
        RAMAdv=>RAMAdv,
        RAMClk=>RAMCLK,
        RAMCre=> RAMCre,
        RAM_CEb=>RAM_CEb,
        RAM_LB=>RAM_LB,
  RAM_UB=>RAM_UB,
        MemDB_io=>MEMDB,
        MemDB_in=>MEM_I_DATA,
        MemDB_out=>MEM_O_DATA
);

INST_50MEG: comp_50Mhz PORT MAP(
                CLK=>clk,
                RST=>RST,
                CLK_50Mhz_out=>MEG_50HZ
);

-------------------------------------------------------------------------
-------------------------------------------------------------------------
--For Real time purpose we need a clock this section give a clock that
--switches the antenna one ofter the other one. Antenna count from 1 to 16
--and based on the truth table provided in the final report the TX_RX is
--changing
-------------------------------------------------------------------------

ANTENNA_CTR:PROCESS(ANTENNA_REAL_TIME_CLK, RST, ANTENNA,Index)
Variable antenna_cnt : integer range 1 to 16;
Variable Index_cnt     : integer range 1 to 16;
```

```
BEGIN
IF RST = '1' THEN
                ANTENNA_cnt := 1;
                INDEX_cnt := 1;
ELSIF RISING_EDGE(ANTENNA_REAL_TIME_CLK) THEN

                        if Index_cnt = 16 then                  -- index for the i q since
                                index_cnt:=1;                                   -- array is used
                        else
                                index_cnt := index_cnt+1;
                        end if;

                        if antenna_cnt = 16 then--sixteen values is being collected since we
                                antenna_cnt :=1;                -- are focusing on only the x-axis for now
                        else
                                Antenna_cnt := antenna_cnt +1;
                        end if;
END IF;
antenna<=std_logic_vector(to_unsigned(antenna_cnt,6));
index<=Index_cnt;
END PROCESS;



--------------------------------------------------------------------------
--This finite state machine is to describe wether debug mode is on or if
--the radar is taking real data
--------------------------------------------------------------------------
----- LOWER SECTION OF FSM-------------------
PROCESS(CLK,RST,mem_en)
VARIABLE cnt_INT: INTEGER RANGE 1 TO 10_000_000 ;
VARIABLE MEM_HOLD_INT: INTEGER RANGE 1 TO 10_000 ;
```

```
BEGIN
	IF RST = '1' then
		pr_state <= reset;
		mem_hold_int :=1;
		CNT_int :=1;
	elsif mem_en = '1' then
		CNT_int :=1;
		pr_state <=memory;
		mem_hold_int :=1;
	ELSIF rising_edge(CLK) then
		if mem_hold = 10_000 then
			mem_hold_int :=1;
		else
			mem_hold_int :=mem_hold_int +1;
		end if;

		if cnt = 10_000_000 then
			cnt_int :=1;
		else
			cnt_int := cnt_int + 1;
		end if;

		pr_state <= nx_state;
	end if;
mem_hold<= mem_hold_int;
cnt<=Cnt_int;
END PROCESS;
-------------------------UPPER SECTION OF FSM--------------------
-----UPPER SECTION OF FSM--------------------
FSM: Process(pr_state, SW, CHANNEL, ADC_DONE1,ADC_DONE2, CNT)
BEGIN
```

```
case pr_state is
when reset =>
                    if(SW ="111111") then -- all 5 switches is up means real time data;
                        nx_state<= real_time;
                    elsif (SW/="111111") then
                        nx_state<= debug;
                    end if;

when debug =>
                    if(SW ="111111") then -- all 5 switches is up means real time data;
                        nx_state<= real_time;
                    elsif (SW/="111111") then
                        nx_state<= idle_debug;
                    end if;
when idle_debug =>
                        if(ADC_DONE1 = '1') then
                            if(SW ="111111") then -- all 5 switches is up means
real time data;
                                nx_state<=real_time;
                            elsif (SW/="111111") then
                                nx_state<= debug;
                            end if;
                        else
                            nx_state<= idle_debug;
                        end if;
when real_time=>
                    if(SW = "111111") then-- all 5 switches is up means real time data;
                        nx_state<= test_idle;
                    elsif (SW/="111111") then
                        nx_state<= debug;
                    end if;
```

```
when test_idle=>
                                    if cnt = 1 then
                                            nx_state<=Idle_real_time;
                                    else
                                        nx_state<= test_idle;
                                    end if;
when Idle_real_time=>
                                    if(ADC_DONE1 = '1') then
                                        if(SW ="111111") then -- all 5 switches is up means
real time data;
                                            nx_state<=real_time;
                                        elsif (SW/="111111") then
                                            nx_state<= debug;
                                        end if;
                                    else
                                        nx_state<= Idle_real_time;
                                    end if;
when memory=>
                                    nx_state<= memory_idle;
when memory_idle=>
                                    if cnt = 1 then
                                            nx_state<=memory_adc;
                                    else
                                        nx_state<= memory_idle;
                                    end if;
when memory_adc=>
                                    if(ADC_DONE1 = '1') then
                                            nx_state<=LOAD_I;
                                    else
                                        nx_state<= memory_adc;
                                    end if;
```

```
when   LOAD_I=>
                                    if mem_hold = 1 then
                                        nx_state<= LOAD_I_BAR;
                                    else
                                        nx_state<= LOAD_I;
                                    end if;
when   LOAD_I_BAR=>
                                    if mem_hold = 1 then
                                        nx_state<= LOAD_Q;
                                    else
                                        nx_state<= LOAD_I_BAR;

                                    end if;
when   LOAD_Q=>
                                    if mem_hold = 1 then
                                        nx_state<= LOAD_Q_BAR;
                                    else
                                        nx_state<= LOAD_Q;

                                    end if;
when   LOAD_Q_BAR=>
                                    if mem_hold = 1 then
                                        IF ANTENNA = "010000" THEN-- CHANGE
WHEN Y-AXIS IS BEING USED
                                            NX_STATE<=RESET;
                                        ELSE
                                            nx_state<= MEMORY;
                                        END IF;
                                    else
                                        nx_state<= LOAD_Q_BAR;
```

```
                                        end if;
when others => nx_state <=reset;
end case;
end process;

FSM_SIG: process (pr_state, SW, antenna, index,CHANNEL,
ADC_DATA1,ADC_DATA2,ADC_DATA3,ADC_DATA4)
begin
if pr_state = reset then
                SEG_DATA<="0000000000000000";
                ADC_START<= '0';
                antenna_REAL_TIME_CLK<='0';
                TX_RX<="111111";
                led<=(others=>'0'); -- status lED

elsif pr_state = debug then
                antenna_REAL_TIME_CLK<='0';
                ADC_START<= '1';
                TX_RX<= SW;
                led<= (others => '0'); -- status lED

                if CHANNEL = "0001" then
                        SEG_DATA<="0000000000000000" or ADC_DATA1;
                end if;
                if CHANNEL = "0010" then
                        SEG_DATA<="0000000000000000" or ADC_DATA2;
                end if;
                if CHANNEL = "0100" then
                        SEG_DATA<="0000000000000000" or ADC_DATA3;
                end if;
                if CHANNEL = "1000" then
```

```
                    SEG_DATA<="0000000000000000" or ADC_DATA4;
                end if;


elsif pr_state = Idle_debug then
                antenna_REAL_TIME_CLK<='0';
                ADC_START<= '0';
                TX_RX<= SW;
                led<= (others => '0'); -- status lED
                if CHANNEL = "0001" then
                    SEG_DATA<="0000000000000000" or ADC_DATA1;
                end if;
                if CHANNEL = "0010" then
                    SEG_DATA<="0000000000000000" or ADC_DATA2;
                end if;
                if CHANNEL = "0100" then
                    SEG_DATA<="0000000000000000" or ADC_DATA3;
                end if;
                if CHANNEL = "1000" then
                    SEG_DATA<="0000000000000000" or ADC_DATA4;
                end if;

elsif pr_state = real_time then
                    ADC_START<= '0';
                    antenna_REAL_TIME_CLK<='0';
--                  led<=(others=>'1');
                    led<= "00000000" or antenna; -- status lED

                    TX_RX<=antenna;

elsif pr_state = Idle_real_time then
```

```
                    ADC_START<= '0';

                    antenna_REAL_TIME_CLK<='0';
--                  led<=(others=>'1');
                    led<= "00000000" or antenna; -- status lED

                    TX_RX<=antenna;

                    I(index)<=ADC_DATA1;
                    I_bar(index)<=ADC_DATA2;
                    Q(index)<=ADC_DATA3;
                    Q_bar(index)<=ADC_DATA4;

 elsif pr_state = test_idle then
                    ADC_START<= '1';
--                  led<=(others=>'1');
                    led<= "00000000" or antenna; -- status lED

                    TX_RX<=Antenna;
                    antenna_REAL_TIME_CLK<='1';
elsif pr_state = memory then
                    ADC_START<= '0';
                    antenna_REAL_TIME_CLK<='1';
--                  led<=(others=>'1');
                    led<= "00000000" or antenna; -- status lED

                    TX_RX<=antenna;

 elsif pr_state = memory_idle then
                    ADC_START<= '1';
                    led<= "00000000" or antenna; -- status lED
```

```
                    TX_RX<=Antenna;
                    antenna_REAL_TIME_CLK<='0';

elsif pr_state = memory_adc then
                    ADC_START<= '0';
                    I(INDEX)<=ADC_DATA1;
                    I_bar(INDEX)<=ADC_DATA2;
                    Q(INDEX)<=ADC_DATA3;
                    Q_bar(INDEX)<=ADC_DATA4;

                    antenna_REAL_TIME_CLK<='0';
--                  led<=(others=>'1');
                    led<= "00000000" or antenna; -- status lED
                    TX_RX<=antenna;

        elsif pr_state = LOAD_I then
                    I(INDEX)<=ADC_DATA1;
                    Mem_ADDR<=MEM_ADR(48+INDEX)(22 DOWNTO 0);
                    Mem_I_data<=STD_ULOGIC_VECTOR(X"0000" or I(INDEX));
                    ADC_START<= '0';
                    antenna_REAL_TIME_CLK<='0';
        elsif pr_state = LOAD_I_BAR then
                    I_BAR(INDEX)<=ADC_DATA2;
                    Mem_ADDR<=MEM_ADR(16+INDEX)(22 DOWNTO 0);
                    Mem_I_data<=STD_ULOGIC_VECTOR(X"0000" or I_BAR(INDEX));
                    ADC_START<= '0';
                    antenna_REAL_TIME_CLK<='0';
        elsif pr_state = LOAD_Q then
                    Q(INDEX)<=ADC_DATA3;
                    Mem_ADDR<=MEM_ADR(32+INDEX)(22 DOWNTO 0);
```

```
                    Mem_I_data<=STD_ULOGIC_VECTOR(X"0000" or Q(INDEX));
                    ADC_START<= '0';
                    antenna_REAL_TIME_CLK<='0';
        elsif pr_state = LOAD_Q_BAR then
                    Q_BAR(INDEX)<=ADC_DATA4;
                    Mem_ADDR<=MEM_ADR(INDEX)(22 DOWNTO 0);
                    Mem_I_data<=STD_ULOGIC_VECTOR(X"0000" or Q_BAR(INDEX));
                    ADC_START<= '0';
                    antenna_REAL_TIME_CLK<='0';

 end if;
 end process;
-----------------------------------------


---------------------------------------
--All combination of the sp4t and sp16t is in the following
--this process statement is not related to any thing. it is a
--case statement base of the sw.
---------------------------------------
switching :process(CLK, sw, Tx_RX)
begin
If rising_edge(CLK) then

        if (TX_RX="000000")then
        sp4t<="1111";
        sp16t<="1111111111111111";
        end if;



        --first horizontal direction   starts at t1r1-t0r8
        if(TX_RX="000001")                  then
```

```
sp4t<="1110";
sp16t<="1111111111111110";
elsif(TX_RX="000010")       then
sp4t<="1110";
sp16t<="1111111111111101";
elsif(TX_RX="000011")       then
sp4t<="1110";
sp16t<="1111111111111011";
elsif(TX_RX="000100")       then
sp4t<="1110";
sp16t<="1111111111110111";
elsif(TX_RX="000101")       then
sp4t<="1110";
sp16t<="1111111111101111";
elsif(TX_RX="000110")       then
sp4t<="1110";
sp16t<="1111111111011111";
elsif(TX_RX="000111")       then
sp4t<="1110";
sp16t<="1111111110111111";
elsif(TX_RX="001000")       then
sp4t<="1110";
sp16t<="1111111101111111";

------Other horizontal direction    t2r1-t1r8
elsif(TX_RX="001001")                then
sp4t<="1101";
sp16t<="1111111111111110";
elsif(TX_RX="001010")       then
sp4t<="1101";
sp16t<="1111111111111101";
```

```
elsif(TX_RX="001011")       then
sp4t<="1101";
sp16t<="1111111111111011";
elsif(TX_RX="001100")       then
sp4t<="1101";
sp16t<="1111111111110111";
elsif(TX_RX="001101")       then
sp4t<="1101";
sp16t<="1111111111101111";
elsif(TX_RX="001110")       then
sp4t<="1101";
sp16t<="1111111111011111";
elsif(TX_RX="001111")       then
sp4t<="1101";
sp16t<="1111111110111111";
elsif(TX_RX="010000")       then
sp4t<="1101";
sp16t<="1111111101111111";

--vertical direction   t3r9-t3r16

elsif(TX_RX="010001")       then
sp4t<="1011";
sp16t<="1111111011111111";
elsif(TX_RX="010010")       then
sp4t<="1011";
sp16t<="1111110111111111";
elsif(TX_RX="010011")       then
sp4t<="1011";
sp16t<="1111101111111111";
elsif(TX_RX="010100")       then
```

```
sp4t<="1011";
sp16t<="1111011111111111";
elsif(TX_RX="010101")       then
sp4t<="1011";
sp16t<="1110111111111111";
elsif(TX_RX="010110")       then
sp4t<="1011";
sp16t<="1101111111111111";
elsif(TX_RX="010111")       then
sp4t<="1011";
sp16t<="1011111111111111";
elsif(TX_RX="011000")               then
sp4t<="1011";
sp16t<="0111111111111111";

--other vertical direction   t3r9-t3r16

elsif(TX_RX="011001")       then
sp4t<="0111";
sp16t<="1111111011111111";
elsif(TX_RX="011010")       then
sp4t<="0111";
sp16t<="1111110111111111";
elsif(TX_RX="011011")       then
sp4t<="0111";
sp16t<="1111101111111111";
elsif(TX_RX="011100")       then
sp4t<="0111";
sp16t<="1111011111111111";
elsif(TX_RX="011101")       then
sp4t<="0111";
```

```
        sp16t<="1110111111111111";
        elsif(TX_RX="011110")       then
        sp4t<="0111";
        sp16t<="1101111111111111";
        elsif(TX_RX="011111")       then
        sp4t<="0111";
        sp16t<="1011111111111111";
        elsif(TX_RX="100000")       then
        sp4t<="0111";
        sp16t<="0111111111111111";
        else null;
        end if;
end if;
end process;




end Behavioral;
```

Appendix C: MATLAB Code Modules

C.1 FAST FOURIER TRANSFORM MODULE

```
function ffasf = sarFFT(I,Q)
th = [-0.157079633 -0.136135682 -0.115191731 -0.09424778 -0.073303829 -0.052359878
-0.031415927 -0.010471976 0.010471976 0.031415927 0.052359878 0.073303829
0.09424778 0.115191731 0.136135682 0.157079633];
basis = [18.85*sin(th); 2*18.85*sin(th); 3*18.85*sin(th); 4*18.85*sin(th);
5*18.85*sin(th);6*18.85*sin(th);7*18.85*sin(th);8*18.85*sin(th);9*18.85*sin(th);10*
18.85*sin(th);11*18.85*sin(th);12*18.85*sin(th);13*18.85*sin(th);14*18.85*sin(th);1
5*18.85*sin(th);16*18.85*sin(th)];
freal = cos(basis);
fim = sin(basis);
cmplxRIQ1 = ((freal(1,:)*I(:,1)) - (fim(1,:)*Q(:,1)));
cmplxRIQ2 = ((freal(2,:)*I(:,2)) - (fim(2,:)*Q(:,2)));
cmplxRIQ3 = ((freal(3,:)*I(:,3)) - (fim(3,:)*Q(:,3)));
cmplxRIQ4 = ((freal(4,:)*I(:,4)) - (fim(4,:)*Q(:,4)));
cmplxRIQ5 = ((freal(5,:)*I(:,5)) - (fim(5,:)*Q(:,5)));
cmplxRIQ6 = ((freal(6,:)*I(:,6)) - (fim(6,:)*Q(:,6)));
cmplxRIQ7 = ((freal(7,:)*I(:,7)) - (fim(7,:)*Q(:,7)));
cmplxRIQ8 = ((freal(8,:)*I(:,8)) - (fim(8,:)*Q(:,8)));
cmplxRIQ9 = ((freal(9,:)*I(:,9)) - (fim(9,:)*Q(:,9)));
cmplxRIQ10 = ((freal(10,:)*I(:,10)) - (fim(10,:)*Q(:,10)));
cmplxRIQ11 = ((freal(11,:)*I(:,11)) - (fim(11,:)*Q(:,11)));
cmplxRIQ12 = ((freal(12,:)*I(:,12)) - (fim(12,:)*Q(:,12)));
cmplxRIQ13 = ((freal(13,:)*I(:,13)) - (fim(13,:)*Q(:,13)));
cmplxRIQ14 = ((freal(14,:)*I(:,14)) - (fim(14,:)*Q(:,14)));
cmplxRIQ15 = ((freal(15,:)*I(:,15)) - (fim(15,:)*Q(:,15)));
cmplxRIQ16 = ((freal(16,:)*I(:,16)) - (fim(16,:)*Q(:,16)));
cmplxImIQ1 = ((freal(1,:)*Q(:,1)) + (fim(1,:)*I(:,1)));
cmplxImIQ2 = ((freal(2,:)*Q(:,2)) + (fim(2,:)*I(:,2)));
cmplxImIQ3 = ((freal(3,:)*Q(:,3)) + (fim(3,:)*I(:,3)));
cmplxImIQ4 = ((freal(4,:)*Q(:,4)) + (fim(4,:)*I(:,4)));
cmplxImIQ5 = ((freal(5,:)*Q(:,5)) + (fim(5,:)*I(:,5)));
cmplxImIQ6 = ((freal(6,:)*Q(:,6)) + (fim(6,:)*I(:,6)));
cmplxImIQ7 = ((freal(7,:)*Q(:,7)) + (fim(7,:)*I(:,7)));
cmplxImIQ8 = ((freal(8,:)*Q(:,8)) + (fim(8,:)*I(:,8)));
cmplxImIQ9 = ((freal(9,:)*Q(:,9)) + (fim(9,:)*I(:,9)));
cmplxImIQ10 = ((freal(10,:)*Q(:,10)) + (fim(10,:)*I(:,10)));
cmplxImIQ11 = ((freal(11,:)*Q(:,11)) + (fim(11,:)*I(:,11)));
cmplxImIQ12 = ((freal(12,:)*Q(:,12)) + (fim(12,:)*I(:,12)));
cmplxImIQ13 = ((freal(13,:)*Q(:,13)) + (fim(13,:)*I(:,13)));
cmplxImIQ14 = ((freal(14,:)*Q(:,14)) + (fim(14,:)*I(:,14)));
cmplxImIQ15 = ((freal(15,:)*Q(:,15)) + (fim(15,:)*I(:,15)));
cmplxImIQ16 = ((freal(16,:)*Q(:,16)) + (fim(16,:)*I(:,16)));
realsum = cmplxRIQ1 + cmplxRIQ2 + cmplxRIQ3 + cmplxRIQ4 + cmplxRIQ5 + cmplxRIQ6 +
cmplxRIQ7 + cmplxRIQ8 + cmplxRIQ9 + cmplxRIQ10 + cmplxRIQ11 + cmplxRIQ12 +
cmplxRIQ13 + cmplxRIQ14 + cmplxRIQ15 + cmplxRIQ16;
rsumsq = (realsum).^(2);
imagsum = cmplxImIQ1 + cmplxImIQ2 + cmplxImIQ3 + cmplxImIQ4 + cmplxImIQ5 +
cmplxImIQ6 + cmplxImIQ7 + cmplxImIQ8 + cmplxImIQ9 + cmplxImIQ10 + cmplxImIQ11 +
cmplxImIQ12 + cmplxImIQ13 + cmplxImIQ14 + cmplxImIQ15 + cmplxImIQ16;
imsumsq = (imagsum).^(2);
```

```
ampl = [20*log10((rsumsq(:,1) + imsumsq(:,1)).^(0.5)); 20*log10((rsumsq(:,2) +
imsumsq(:,2)).^(0.5)); 20*log10((rsumsq(:,3) + imsumsq(:,3)).^(0.5));
20*log10((rsumsq(:,4) + imsumsq(:,4)).^(0.5)); 20*log10((rsumsq(:,5) +
imsumsq(:,5)).^(0.5)); 20*log10((rsumsq(:,6) + imsumsq(:,6)).^(0.5));
20*log10((rsumsq(:,7) + imsumsq(:,7)).^(0.5)); 20*log10((rsumsq(:,8) +
imsumsq(:,8)).^(0.5)); 20*log10((rsumsq(:,9) + imsumsq(:,9)).^(0.5));
20*log10((rsumsq(:,10) + imsumsq(:,10)).^(0.5)); 20*log10((rsumsq(:,11) +
imsumsq(:,11)).^(0.5)); 20*log10((rsumsq(:,12) + imsumsq(:,12)).^(0.5));
20*log10((rsumsq(:,13) + imsumsq(:,13)).^(0.5)); 20*log10((rsumsq(:,14) +
imsumsq(:,14)).^(0.5)); 20*log10((rsumsq(:,15) + imsumsq(:,15)).^(0.5));
20*log10((rsumsq(:,16) + imsumsq(:,16)).^(0.5))];
actres = ampl.^2;
thetainc = [-9; -7.8; -6.6; -5.4; -4.2; -3; -1.8; -0.6; 0.6; 1.8; 3; 4.2; 5.4; 6.6;
7.8; 9];
ffasf = [ampl thetainc];
plot(thetainc,ampl);
figure
bar(thetainc,ampl);
end
```

C.2 READING AND CONVERTING I AND Q DATA MODULES

```
function actual_dataI = readI(dat)
% Get 1st I value
check = 0;
Iarr1 = ((dat(2).*((16).^2)) + dat(1)); % Grabs Upper Byte and Lower Byte, Does
Computation
Ibarr1 = ((dat(34).*((16).^2)) + dat(33)); % Grabs UB and LB of Ibar for Negative
if Iarr1 ~= check % Checks if Positive From ADC logic
    Iv1 = (((Iarr1)./4095).*(3.3)); % Positive Normalization for MATLAB Computation
else % If Value is Negative (from Ibar Channel) From ADC Logic
    Iv1 = -(((Ibarr1)./4095).*(3.3)); % Negative Normalization for MATLAB
Computation
end

Iarr2 = ((dat(4).*((16).^2)) + dat(3));
Ibarr2 = ((dat(36).*((16).^2)) + dat(35));
if Iarr2 ~= check
    Iv2 = (((Iarr2)./4095).*(3.3));
else
    Iv2 = -(((Ibarr2)./4095).*(3.3));
end

Iarr3 = ((dat(6).*((16).^2)) + dat(5));
Ibarr3 = ((dat(38).*((16).^2)) + dat(37));
if Iarr3 ~= check
    Iv3 = (((Iarr3)./4095).*(3.3));
else
    Iv3 = -(((Ibarr3)./4095).*(3.3));
end

Iarr4 = ((dat(8).*((16).^2)) + dat(7));
Ibarr4 = ((dat(40).*((16).^2)) + dat(39));
if Iarr4 ~= check
    Iv4 = (((Iarr4)./4095).*(3.3));
```

```
else
    Iv4 = -(((Ibarr4)./4095).*(3.3));
end

Iarr5 = ((dat(10).*((16).^2)) + dat(9));
Ibarr5 = ((dat(42).*((16).^2)) + dat(41));
if Iarr5 ~= check
    Iv5 = (((Iarr5)./4095).*(3.3));
else
    Iv5 = -(((Ibarr5)./4095).*(3.3));
end

Iarr6 = ((dat(12).*((16).^2)) + dat(11));
Ibarr6 = ((dat(44).*((16).^2)) + dat(43));
if Iarr6 ~= check
    Iv6 = (((Iarr6)./4095).*(3.3));
else
    Iv6 = -(((Ibarr6)./4095).*(3.3));
end

Iarr7 = ((dat(14).*((16).^2)) + dat(13));
Ibarr7 = ((dat(46).*((16).^2)) + dat(45));
if Iarr7 ~= check
    Iv7 = (((Iarr7)./4095).*(3.3));
else
    Iv7 = -(((Ibarr7)./4095).*(3.3));
end

Iarr8 = ((dat(16).*((16).^2)) + dat(15));
Ibarr8 = ((dat(48).*((16).^2)) + dat(47));
if Iarr8 ~= check
    Iv8 = (((Iarr8)./4095).*(3.3));
else
    Iv8 = -(((Ibarr8)./4095).*(3.3));
end

Iarr9 = ((dat(18).*((16).^2)) + dat(17));
Ibarr9 = ((dat(50).*((16).^2)) + dat(49));
if Iarr9 ~= check
    Iv9 = (((Iarr9)./4095).*(3.3));
else
    Iv9 = -(((Ibarr9)./4095).*(3.3));
end

Iarr10 = ((dat(20).*((16).^2)) + dat(19));
Ibarr10 = ((dat(52).*((16).^2)) + dat(51));
if Iarr10 ~= check
    Iv10 = (((Iarr10)./4095).*(3.3));
else
    Iv10 = -(((Ibarr10)./4095).*(3.3));
end

Iarr11 = ((dat(22).*((16).^2)) + dat(21));
Ibarr11 = ((dat(54).*((16).^2)) + dat(53));
if Iarr11 ~= check
```

```
    Iv11 = (((Iarr11)./4095).*(3.3));
else
    Iv11 = -(((Ibarr11)./4095).*(3.3));
end

Iarr12 = ((dat(24).*((16).^2)) + dat(23));
Ibarr12 = ((dat(56).*((16).^2)) + dat(55));
if Iarr12 ~= check
    Iv12 = (((Iarr12)./4095).*(3.3));
else
    Iv12 = -(((Ibarr12)./4095).*(3.3));
end

Iarr13 = ((dat(26).*((16).^2)) + dat(25));
Ibarr13 = ((dat(58).*((16).^2)) + dat(57));
if Iarr13 ~= check
    Iv13 = (((Iarr13)./4095).*(3.3));
else
    Iv13 = -(((Ibarr13)./4095).*(3.3));
end

Iarr14 = ((dat(28).*((16).^2)) + dat(27));
Ibarr14 = ((dat(60).*((16).^2)) + dat(59));
if Iarr14 ~= check
    Iv14 = (((Iarr14)./4095).*(3.3));
else
    Iv14 = -(((Ibarr14)./4095).*(3.3));
end

Iarr15 = ((dat(30).*((16).^2)) + dat(29));
Ibarr15 = ((dat(62).*((16).^2)) + dat(61));
if Iarr15 ~= check
    Iv15 = (((Iarr15)./4095).*(3.3));
else
    Iv15 = -(((Ibarr15)./4095).*(3.3));
end

Iarr16 = ((dat(32).*((16).^2)) + dat(31));
Ibarr16 = ((dat(64).*((16).^2)) + dat(63));
if Iarr16 ~= check
    Iv16 = (((Iarr16)./4095).*(3.3));
else
    Iv16 = -(((Ibarr16)./4095).*(3.3));
end

actual_dataI = [Iv1 Iv2 Iv3 Iv4 Iv5 Iv6 Iv7 Iv8 Iv9 Iv10 Iv11 Iv12 Iv13 Iv14 Iv15
Iv16];
end



function actual_dataQ = readQ(dat)
% Get 1st Q value
check = 0;
```

```
Qarr1 = ((dat(66).*((16).^2)) + dat(65)); % Grabs Upper Byte and Lower Byte, Does
Computation
Qbarr1 = ((dat(98).*((16).^2)) + dat(97));
if Qarr1 ~= check % Checks if Positive From ADC logic
    Qv1 = (((Qarr1)./4095).*(3.3)); % Positive Normalization for MATLAB Computation
else % If Value ss Negative From ADC Logic
    Qv1 = -(((Qbarr1)./4095).*(3.3)); % Negative Normalization for MATLAB
Computation
end

Qarr2 = ((dat(68).*((16).^2)) + dat(67));
Qbarr2 = ((dat(100).*((16).^2)) + dat(99));
if Qarr2 ~= check
    Qv2 = (((Qarr2)./4095).*(3.3));
else
    Qv2 = -(((Qbarr2)./4095).*(3.3));
end

Qarr3 = ((dat(70).*((16).^2)) + dat(69));
Qbarr3 = ((dat(102).*((16).^2)) + dat(101));
if Qarr3 ~= check
    Qv3 = (((Qarr3)./4095).*(3.3));
else
    Qv3 = -(((Qbarr3)./4095).*(3.3));
end

Qarr4 = ((dat(72).*((16).^2)) + dat(71));
Qbarr4 = ((dat(104).*((16).^2)) + dat(103));
if Qarr4 ~= check
    Qv4 = (((Qarr4)./4095).*(3.3));
else
    Qv4 = -(((Qbarr4)./4095).*(3.3));
end

Qarr5 = ((dat(74).*((16).^2)) + dat(73));
Qbarr5 = ((dat(106).*((16).^2)) + dat(105));
if Qarr5 ~= check
    Qv5 = (((Qarr5)./4095).*(3.3));
else
    Qv5 = -(((Qbarr5)./4095).*(3.3));
end

Qarr6 = ((dat(76).*((16).^2)) + dat(75));
Qbarr6 = ((dat(108).*((16).^2)) + dat(107));
if Qarr6 ~= check
    Qv6 = (((Qarr6)./4095).*(3.3));
else
    Qv6 = -(((Qbarr6)./4095).*(3.3));
end

Qarr7 = ((dat(78).*((16).^2)) + dat(77));
Qbarr7 = ((dat(110).*((16).^2)) + dat(109));
if Qarr7 ~= check
    Qv7 = (((Qarr7)./4095).*(3.3));
else
    Qv7 = -(((Qbarr7)./4095).*(3.3));
```

```
end

Qarr8 = ((dat(80).*((16).^2)) + dat(79));
Qbarr8 = ((dat(112).*((16).^2)) + dat(111));
if Qarr8 ~= check
    Qv8 = (((Qarr8)./4095).*(3.3));
else
    Qv8 = -(((Qbarr8)./4095).*(3.3));
end

Qarr9 = ((dat(82).*((16).^2)) + dat(81));
Qbarr9 = ((dat(114).*((16).^2)) + dat(113));
if Qarr9 ~= check
    Qv9 = (((Qarr9)./4095).*(3.3));
else
    Qv9 = -(((Qbarr9)./4095).*(3.3));
end

Qarr10 = ((dat(84).*((16).^2)) + dat(83));
Qbarr10 = ((dat(116).*((16).^2)) + dat(115));
if Qarr10 ~= check
    Qv10 = (((Qarr10)./4095).*(3.3));
else
    Qv10 = -(((Qbarr10)./4095).*(3.3));
end

Qarr11 = ((dat(86).*((16).^2)) + dat(85));
Qbarr11 = ((dat(118).*((16).^2)) + dat(117));
if Qarr11 ~= check
    Qv11 = (((Qarr11)./4095).*(3.3));
else
    Qv11 = -(((Qbarr11)./4095).*(3.3));
end

Qarr12 = ((dat(88).*((16).^2)) + dat(87));
Qbarr12 = ((dat(120).*((16).^2)) + dat(119));
if Qarr12 ~= check
    Qv12 = (((Qarr12)./4095).*(3.3));
else
    Qv12 = -(((Qbarr12)./4095).*(3.3));
end

Qarr13 = ((dat(90).*((16).^2)) + dat(89));
Qbarr13 = ((dat(122).*((16).^2)) + dat(121));
if Qarr13 ~= check
    Qv13 = (((Qarr13)./4095).*(3.3));
else
    Qv13 = -(((Qbarr13)./4095).*(3.3));
end

Qarr14 = ((dat(92).*((16).^2)) + dat(91));
Qbarr14 = ((dat(124).*((16).^2)) + dat(123));
if Qarr14 ~= check
    Qv14 = (((Qarr14)./4095).*(3.3));
else
```

```
    Qv14 = -(((Qbarr14)./4095).*(3.3));
end

Qarr15 = ((dat(94).*((16).^2)) + dat(93));
Qbarr15 = ((dat(126).*((16).^2)) + dat(125));
if Qarr15 ~= check
    Qv15 = (((Qarr15)./4095).*(3.3));
else
    Qv15 = -(((Qbarr15)./4095).*(3.3));
end

Qarr16 = ((dat(96).*((16).^2)) + dat(95));
Qbarr16 = ((dat(128).*((16).^2)) + dat(127));
if Qarr16 ~= check
    Qv16 = (((Qarr16)./4095).*(3.3));
else
    Qv16 = -(((Qbarr16)./4095).*(3.3));
end

actual_dataQ = [Qv1 Qv2 Qv3 Qv4 Qv5 Qv6 Qv7 Qv8 Qv9 Qv10 Qv11 Qv12 Qv13 Qv14 Qv15
Qv16];
end
```

C.3 CALIBRATION CODE MODULES

```
function correctedQ = calibrationQ(Qval)
correctedQ = -Qval;
end

function correctedI = calibrationI(Ival)
correctedI = Ival;
end
```