FAMU-FSU College of Engineering

Department of Electrical and Computer Engineering

User Manual - 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

I. Introduction

1.1 Executive Summary

The component housing is enclosing each single electrical device and connection that is used for the Synthetic Aperture Radar Imager. Its main function is to safely house the hardware and protect it from physical damage, overheating, and tampering or theft. It is to allow easy access for testing and configuration purposes as well as to act as a modular "black box" to avoid having the components exposed unnecessarily.

1.2 Functional Diagram

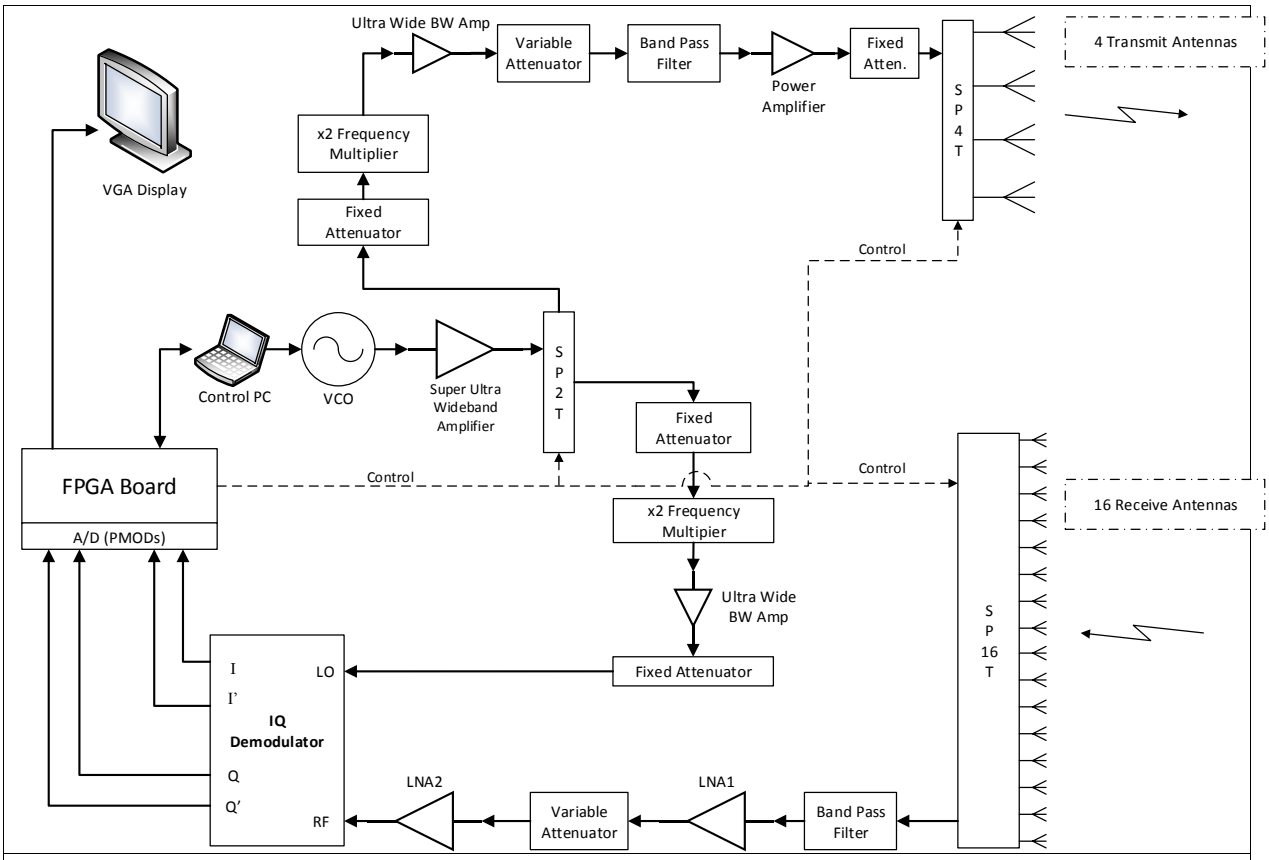

Figure 1.2 : Top-down block diagram

1.3 Project Specifications

The product specifications were defined early on based upon possible improvements from the legacy component housing. Areas of improvement include the overall weight and size of the component housing which lead to a lack of mobility to the already immobile structure of last year.

Further, there was so safe securing mechanism for the housing to the actual frame, there were two aluminum bars extruded from the frame that the housing sat onto with two equivalent sized slits. The housing was slid on and rested on the bars and was depended on friction making it unideal and a potential safety hazard. Further, most of the components were mounted on a vertical plate which made the configuration and testing tedious for the electrical engineers since there was not much room for their hands. Lastly, the entire housing was open leaving all of the electrical components exposed which was not ideal since there are expensive pieces of hardware which can potentially be damaged, tampered, or stolen. To remedy this, the design is not only enclosed but also has latches. The exposure also could have led to electromagnetic interference.

The overall weight of the newly designed component housing before the actual components would are mounted and installed is 12.9 lbs. This is a vast improvement from the legacy component housing, mainly due to the material selection. The new housing was made with 0.09” think aluminum as oppose to ¼” steel from the previous housing. The new component housing dimensions are 17.25” x 30” x 5” as shown below in figure 1.3. The mobility of the housing improved not only to the size and weight reduction over last year, but also with welded handles on sides of the housing. This allows the housing to be lifted and moved with more simplicity, and with the decrease in weight, it can easily be carried by one individual.

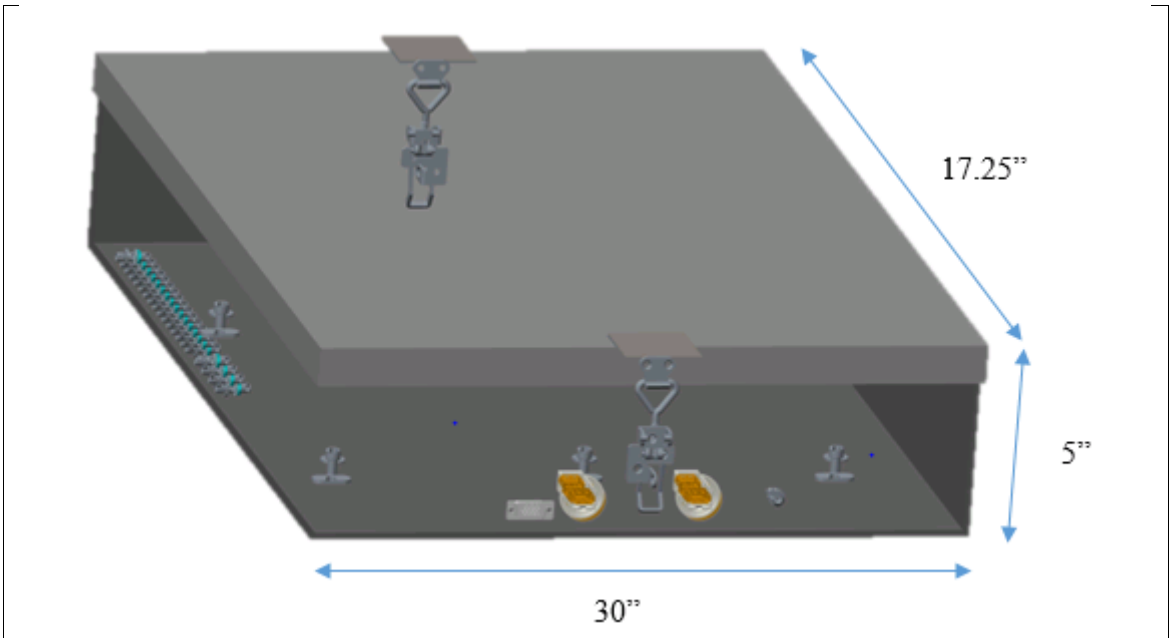

Figure 1.3: CAD image of final component housing design iteration

As mentioned before, the housing needs to be thermally efficient to prevent the components from overheating, allowing them to perform properly. Based on the surface area, power supplied to the components, and material/finish, the component housing is able to safely operate and will experience a 9.5℃ temperature increase within the enclosure once steady state has been reached. Since the component housing will be mostly operated at room temperature (21℃), this translates to an enclosure temperature of approximately 30.5℃. This is deemed safe and appropriate since the most sensitive electrical component's operating temperature limit is 50℃.

To minimize electromagnetic interference the new housing is completely enclosed and electromagnetic interference shielding was applied in the form of a gasket. This EMI gasket was installed along the inside perimeter of the lid, so any seals or gaps from the lid to the top surface of the housing would be in contact. Lastly, the housing's safety and security were upgraded by having the housing installed to the frame via bolts as well as being able to be locked when stored so that the housing cannot be opened when not wanted.

II. Components

2.1 Product Assembly

All of the electrical components are mounted onto a drop-in plate which sits on ¾" risers above the bottom of the housing which is then mounted to the frame. Below in figure XXB is the layout of each electrical component in their final arrangement. The electrical components are labeled and tabulated below.

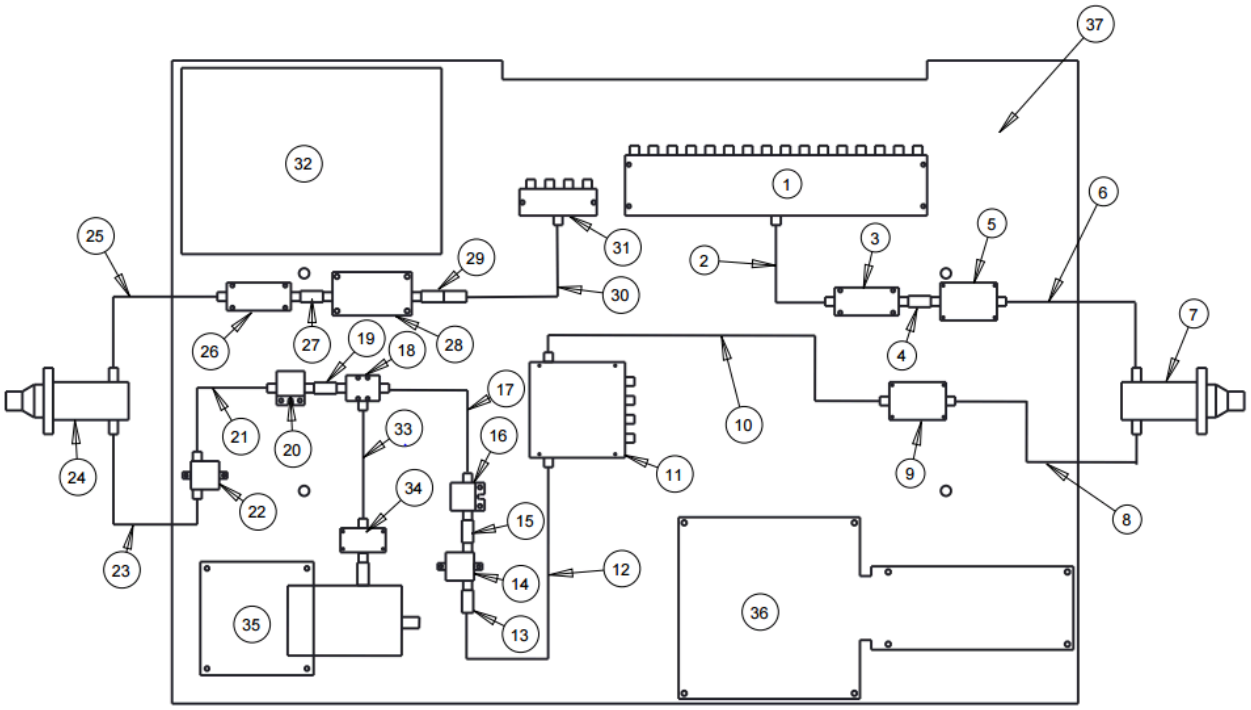

Figure 2.1a: Layout of electrical components mounted onto drop-in plate

Plate#	Name	Designation/Description	Power
1	SP16T Switch	SR-L010-16S	+5,-12, GND
2	SMA Cable		
3	10.5 GHz Bandpass Filter	Marki 1050	
4	SMA M-M		
5	LNA 1	SLNA-120-28-22-SMA	+12, GND
6	SMA Cable		
7	Variable Attenuator	SA4077	
8	SMA Cable		
9	LNA 2	SLNA-180-38-25-SMA	+12, GND
10	SMA Cable		

11	IQ Demodulator	AD60100B	+5,-5,GND
12	SMA Cable		
13	-3dB Attenuator		
14	AMP 3	Ultra Wide Bandwidth Amplifier ZX60-14012L-S+	+12, GND
15	SMA M-M		
16	2x Frequency Multiplier	ZX90-2-50-S+	
17	SMA Cable		
18	SPDT Switch	HMC-C011	
19	SMA M-M		
20	2x Frequency Multiplier	ZX90-2-50-S+	
21	SMA Cable		
22	AMP 2	Ultra Wide Bandwidth Amplifier ZX60-14012L-S+	+12, GND
23	SMA Cable		
24	Variable Attenuator	SA4077	
25	SMA Cable		
26	10.5 GHz Bandpass Filter	Marki FB1050	
27	SMA F-F		
28	Power Amplifier	SPA-110-30-01-SMA	+15, GND
29	-3dB Attenuator		
30	SMA Cable		
31	SP4T Switch	RFSP4TA0812G	+5, -5, GND
32	Power Supply		+ 19 VDC
	-6 dB Attenuator	To pad RF COM port of SPDT. Not shown in the component plate diagram	

33	SMA Cable		
34	AMP 1	Super Ultra Wideband Amplifier	+12, GND
35	VCO	Voltage-Controlled Oscillator (PLL)	+5, GND
36	FPGA	Xilinx Nexys 3 & space for VHDC breakout board	USB Power

The drop-in plate is then seated on four ¾" risers which are screwed through the plate as well as the bottom of the component housing. This is shown below in figures XXD and XXE, where the lid of the component housing is removed for the sake of demonstrating how the drop-in plate sits inside. Further, figure 2.1b and 2.1c shows how the component housing will install onto the frame, displayed with its lid closed.

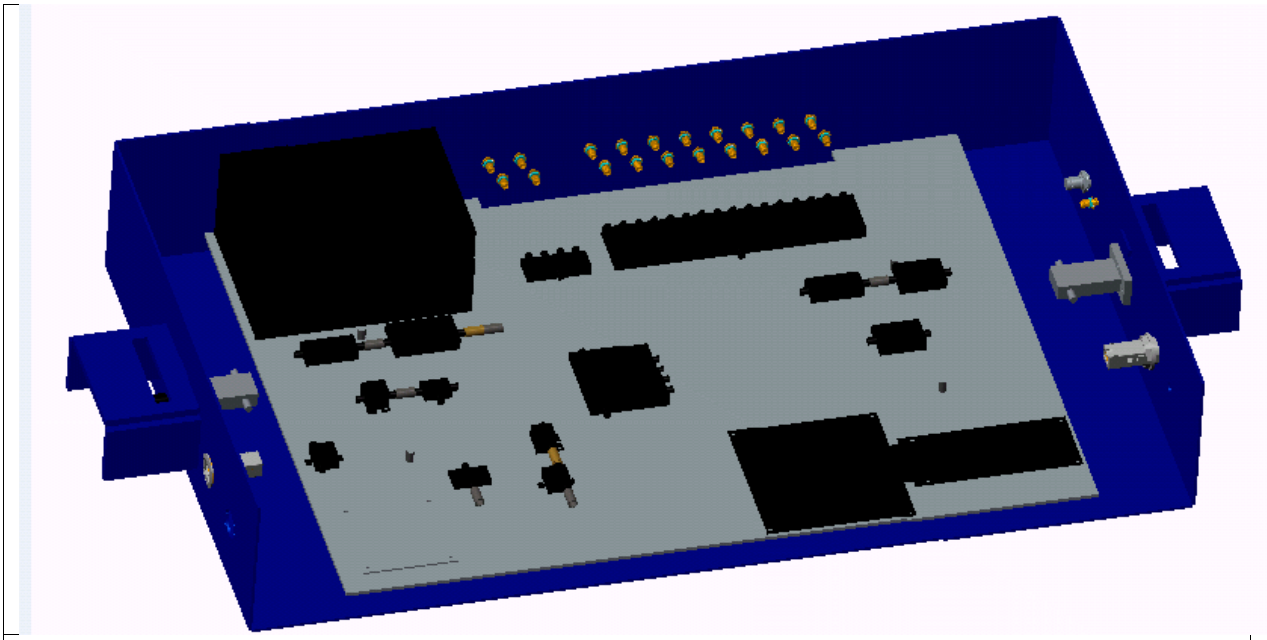

Figure 2.1b: Drop-in plate mounted to component housing

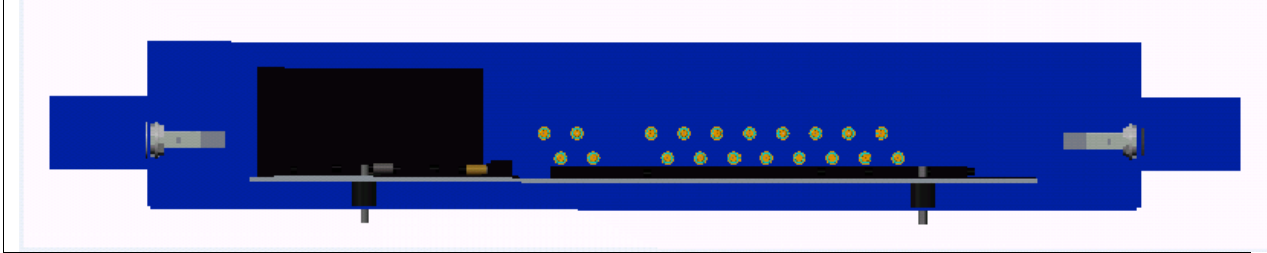

Figure 2.1c: Drop-in plate mounted to component housing

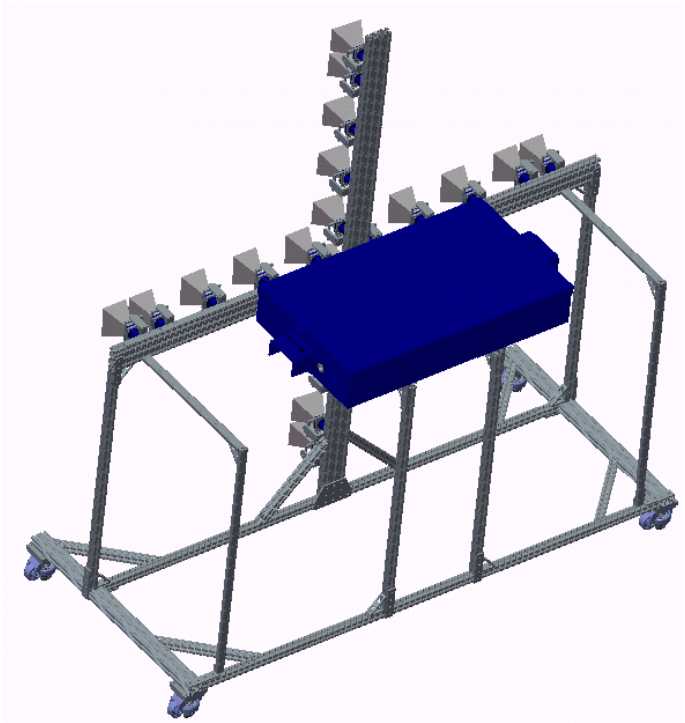

Figure 2.1d: Component housing mounted to frame

III. Setup

3.1 Installation

1) First, the four 3/4” standoff risers must be inserted through the four holes on the bottom of the component housing. Then four T-nuts must be threaded from the outside of the bottom face of the housing, these will slide along the two channels of the rear legs. Make sure that the SMA connectors are facing the structure when sliding the housing toward the structure within the channels.
2) Next, the drop-in plate must be placed inside the component housing so that the four holes on the plate align with the standoff risers. Once in place, tighten a nut on each of the protruding standoff threads. Tighten until each nut is snug and locked in place.
3) After, the twenty SMA connectors along the front face of the housing need to be connected to their corresponding switch ports. These include: the SP16T, SP4T, as well as the variable attenuators. Make sure to follow the labels on the outside of the housing to correctly connect the SMA adaptors to each connector.

WARNING: Use 50Ω terminations (SMA screw-on caps marked "ST1819") on any unused transmit (SP4T) lines during testing to prevent signal reflection.

4) Now the panel mounts need to be connected to their corresponding input and output sources. The FPGA has one VGA port that needs to be connected to its corresponding panel mounted connector and one USB port that needs to be connected as well. The VCO also has one USB port that needs connecting. Ensure that the power supply's switch is turned OFF before connecting it to its corresponding panel mounted connector.

3.2 Safety Precautions

Once the component housing is properly installed onto the frame, safety precautions should be taken to ensure the housing is properly mounted so it will not fall or be accidently pushed off of the frame.

1) Check the tightness of the bolts mounted through the drop-in plate through the frame. Ensure none are loose, if so, tighten until each bolt is snug and cannot move freely.
2) When removing or reattaching the lid from the component box, make sure to always grip the lid from the handles on both sides to avoid the user pinching his/her fingers.
3) When plugging or unplugging any cable, make sure to grasp by the head of the plug and not the actual cable to prevent pulling and creating tension to the component and the cable itself.
4) If the lid is to be removed, make certain there are no wires or cables in the path that can obstruct the lid and potentially pull or tug the wires or cables. This will warrant that the wires, cables, or electrical components will not be harmed by tension.
5) If the lid is to be removed, ensure placement of lid is on a stable and safe location to minimize the risk of having it bumped or pushed to the ground, potentially harming someone or the lid itself.
6) Always ensure an electrostatic discharge (ESD) strap is being used properly before handling internal components. Solid-state switches (SPDT, SP4T, SP16T) are extremely sensitive to ESD.
7) Ensure the frame structure, the component housing, and the component plate are properly grounded before applying power or working inside the component housing.

8) The SAR and the high-frequency signal generator provided by Northrop Grumman are capable of generating enough power to damage test equipment. Pad the input of the spectrum analyzer with a suitable fixed attenuator (usually -10 dB or so) when conducting testing to prevent overload.

IV. Operation

4.1 Turn-On Procedure

Read and understand all the following warnings before attempting to power on the SAR.

Warnings:

1. Ensure frame assembly, component housing, and component plate are properly grounded prior to applying power.
2. Always ensure an electrostatic discharge (ESD) strap is being used properly before handling internal components. Solid-state switches (SPDT, SP4T, SP16T) are extremely sensitive to ESD damage.
3. Before applying power to the SAR, or to any active component in the component housing, check for loose pin sheaths or connections that might be in contact with the component plate, causing a short-circuit. When in doubt, check with a multi-meter.
4. If using a programmable DC power supply instead of the power cable provided, ensure that +19 VDC is being supplied **with a voltmeter** before plugging into the component housing power supply. The system should draw about 2.18A with everything connected and powered.
5. RF Radiation: Although the SAR transmits non-ionizing RF radiation at safe power levels, electromagnetic radiation (EMR) levels inside or directly on the antenna horns can exceed the ANSI/IEEE and FCC limit for safe human exposure to EMR of $1 mW/cm^2$. This drops to safe levels within an inch or two of the transmitting horn. Do not place fingers or body parts inside a transmitting antenna horn or waveguide adapter assembly. An EMR meter is provided in the project supplies.

Turn-On Procedure

1. With SAR power off, connect the control PC to the VCO and FPGA via USB. If using a signal generator to drive the SPDT, ensure that it is connected with the output signal turned off.
2. Ensure the switch on the component housing power supply is in the OFF position.
3. Plug in a 19 VDC power source to the component housing power supply through the panel connector marked "19V".
4. Turn on the switch on the component housing power supply. A red LED next to the switch should illuminate.
5. If driving the SPDT switch with an external function generator, turn on the output. See Table 4.1 for appropriate function generator settings.
6. Turn on VCO from the control PC according to Section 4.2: Voltage Controlled Oscillator.
7. Turn on the FPGA using the POWER slider switch on the board.
8. Set the SAR to the desired operation mode according to Section 4.3: FPGA Control.

Turn-Off Procedure

1. Turn off the components in the opposite order as the turn-on procedure.

Transmit/Receive Mode Switching with the Tektronix AFG 3022B Dual Channel 25MHz Function Generator

Using the 25 MHz function generator is not ideal, but it is the most suitable piece of equipment in the senior design lab at the time of this writing. A 50 MHz function generator should be used if available.

Table 4.1: Tektronix AFG 3022B Settings for Driving Tx/Rx Mode (SPDT)

Function	Pulse Continuous
Period	83.33 ns
Delay	0.0 Ns
High	2.200V
Low	0 mV

Duty	39.182 %
Leading	18.00 ns
Trailing	18.00 ns

4.2 Voltage Controlled Oscillator

The Hittite (Analog Devices) HMC820LP6CE Fractional-N PLL with Integrated VCO (herein referred to as simply "VCO") provides the analog signal that is multiplied, amplified, and transmitted by the transmission subsystem, as well as the reference signal for the Local Oscillator (LO) subsystem.

To generate the 5GHz sine wave that drives the SAR:

1. Connect SMA port labeled RF OUT to the Super Ultra Wideband Amplifier (Amp 1) in the component housing, or to an appropriate 50Ω load or test equipment if on the bench.
2. Connect the VCO to a PC with the Hittite software installed (boot CD-ROM is in the VCO box) using a USB to Type 2 connection wire.
3. Turn on the component housing power supply if on the SAR. If bench testing, apply +5 VDC from a DC power supply to TP3 (+5V) and TP4(GND) on the VCO.
4. Launch "Launch Hittite x64 PLL Eval Software V3250.exe". The design tool and design.exe are not used to generate the signal, although there are likely shortcuts for these on the desktop.
5. Go to the "PLL with Integrated RF VCOs" drop-down menu and select "HMC820LP6CE". Select Done.

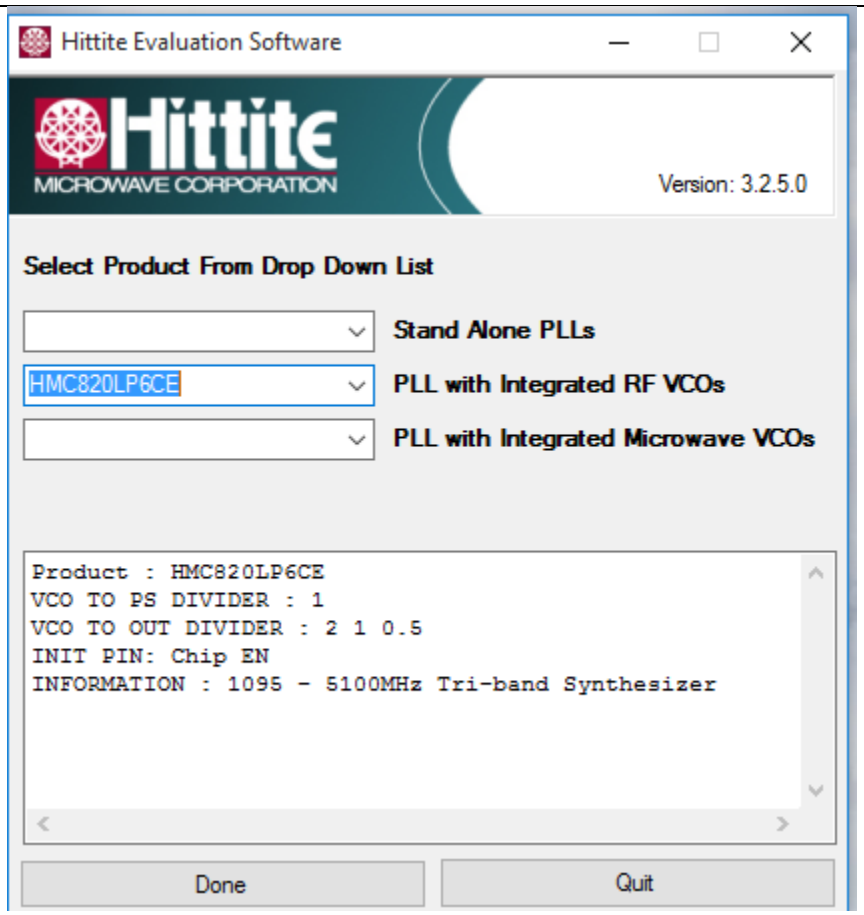

Figure 4.1a: Hittite evaluation software

6. This should launch the main GUI. “Load Reg File” will be flashing in red in the bottom right corner. Click it.

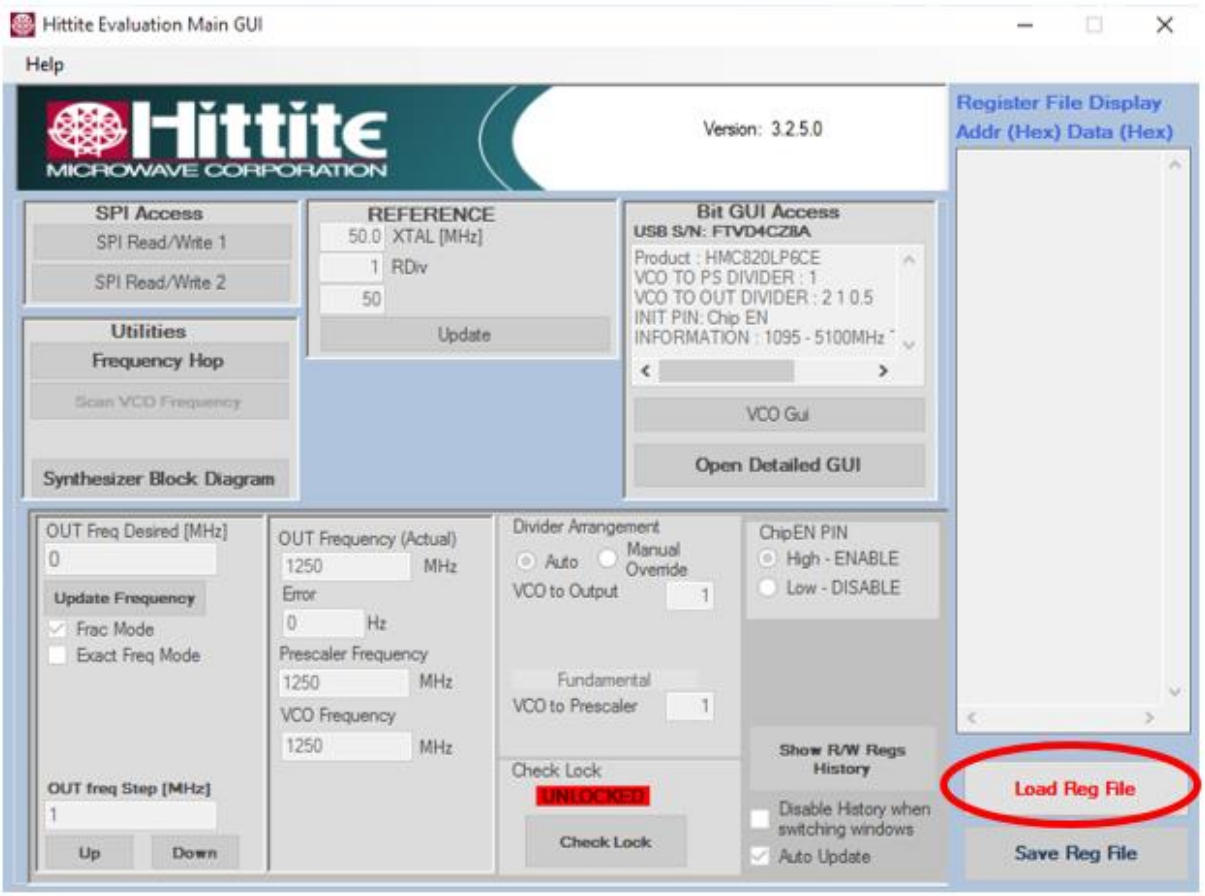

Figure 4.1b: XX: Hittite evaluation main GUI

7. In the dialog box that opens, select the “PLL with Integrated RF VCOs” folder and open the text file “hmc820_2x”.

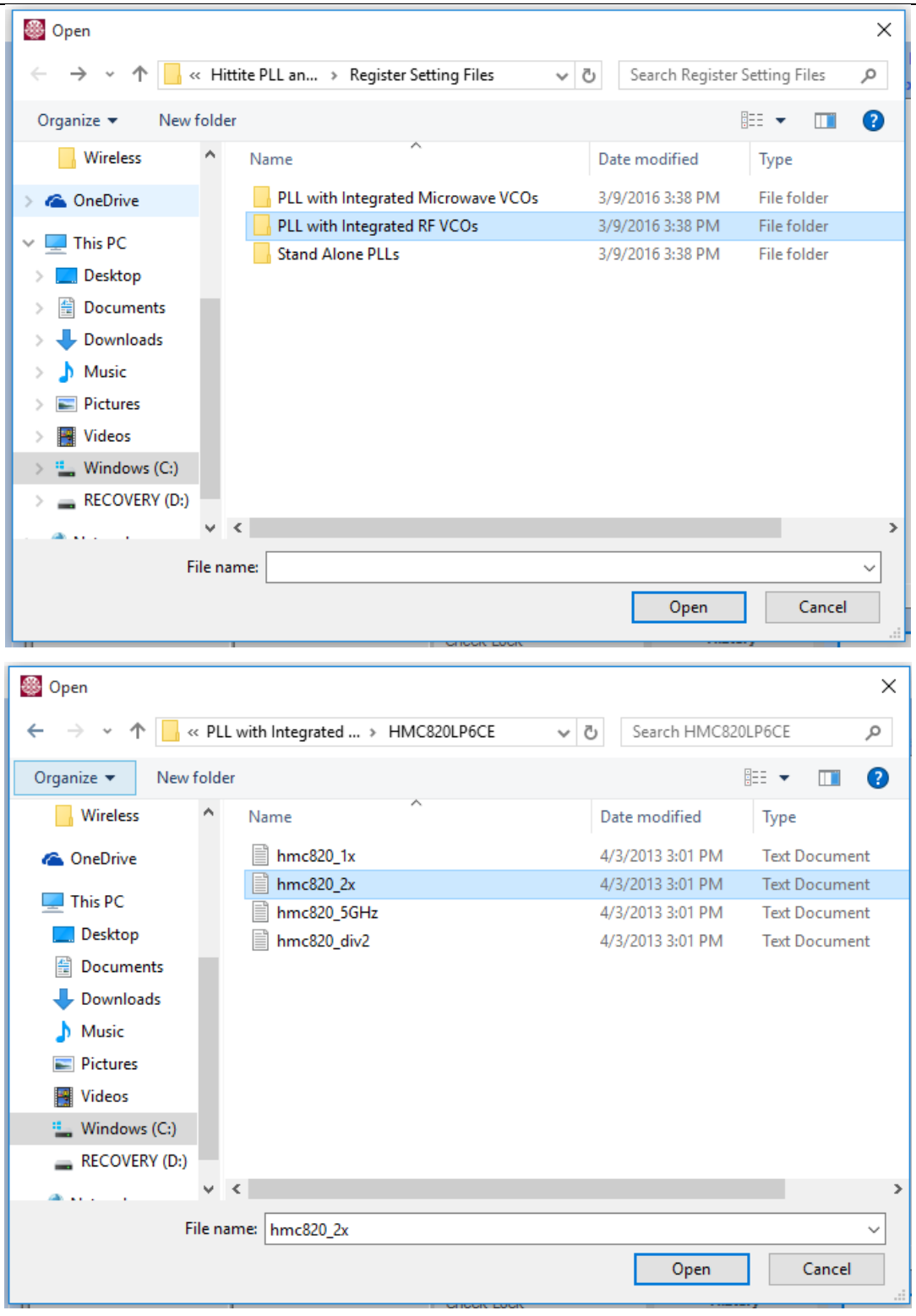

Figure 4.1c: VCO register file

8. After a few moments, the default output frequency will populate, as well as several other parameters. The VCO is now transmitting at the default frequency. If running the system live, the transmit horns are now transmitting RF at twice this default frequency. If on the bench, the default frequency can be seen on the spectrum analyzer using the peak search function.

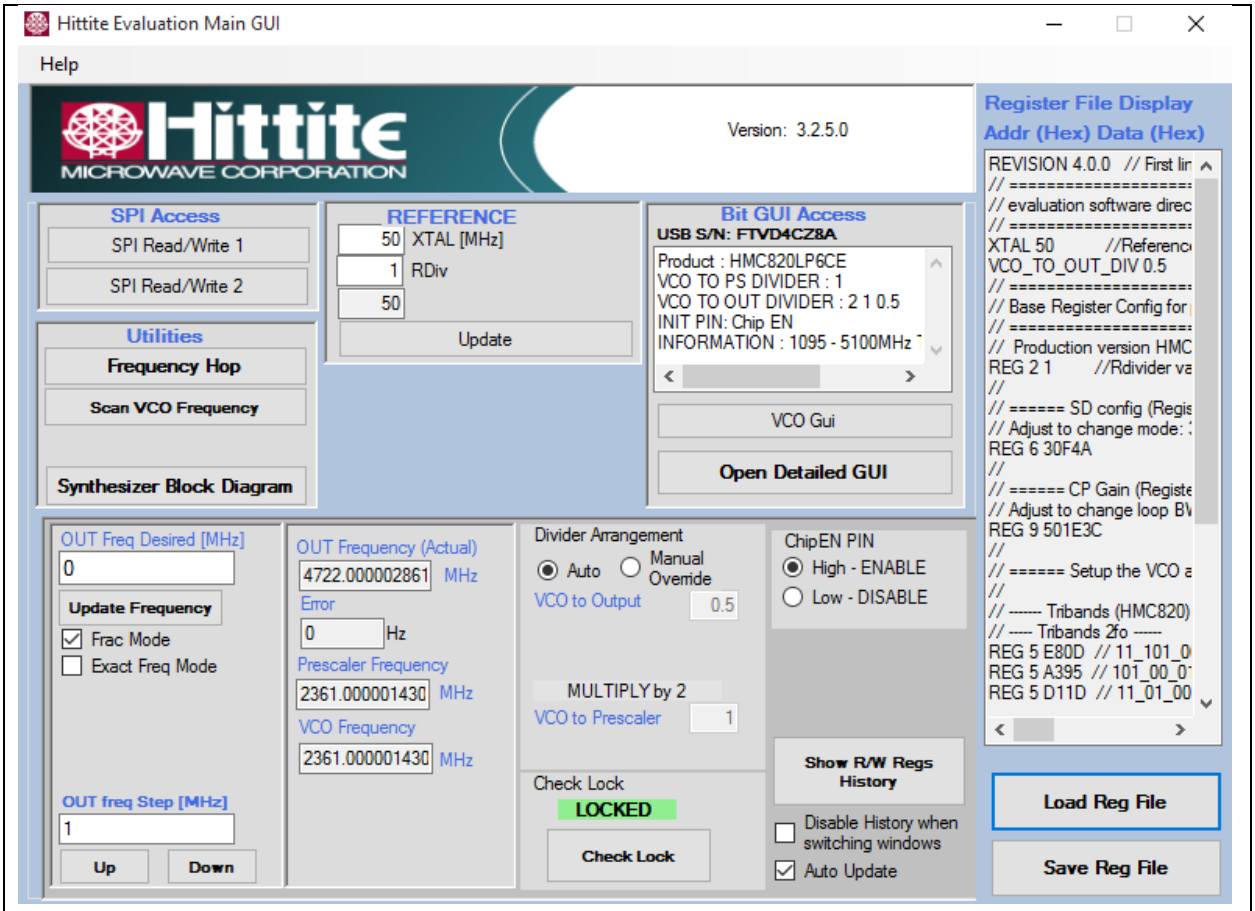

Figure 4.1d: VCO default frequency

9. Enter “5000” in the “OUT Freq Desired [MHz]” field and click “Update Frequency”. After a moment, the “OUT Frequency (Actual)” should populate the new frequency and the “Check Lock” will display “LOCKED” in green, indicating phase lock has been achieved. The VCO is not generating a 5 GHz phase-locked analog signal.

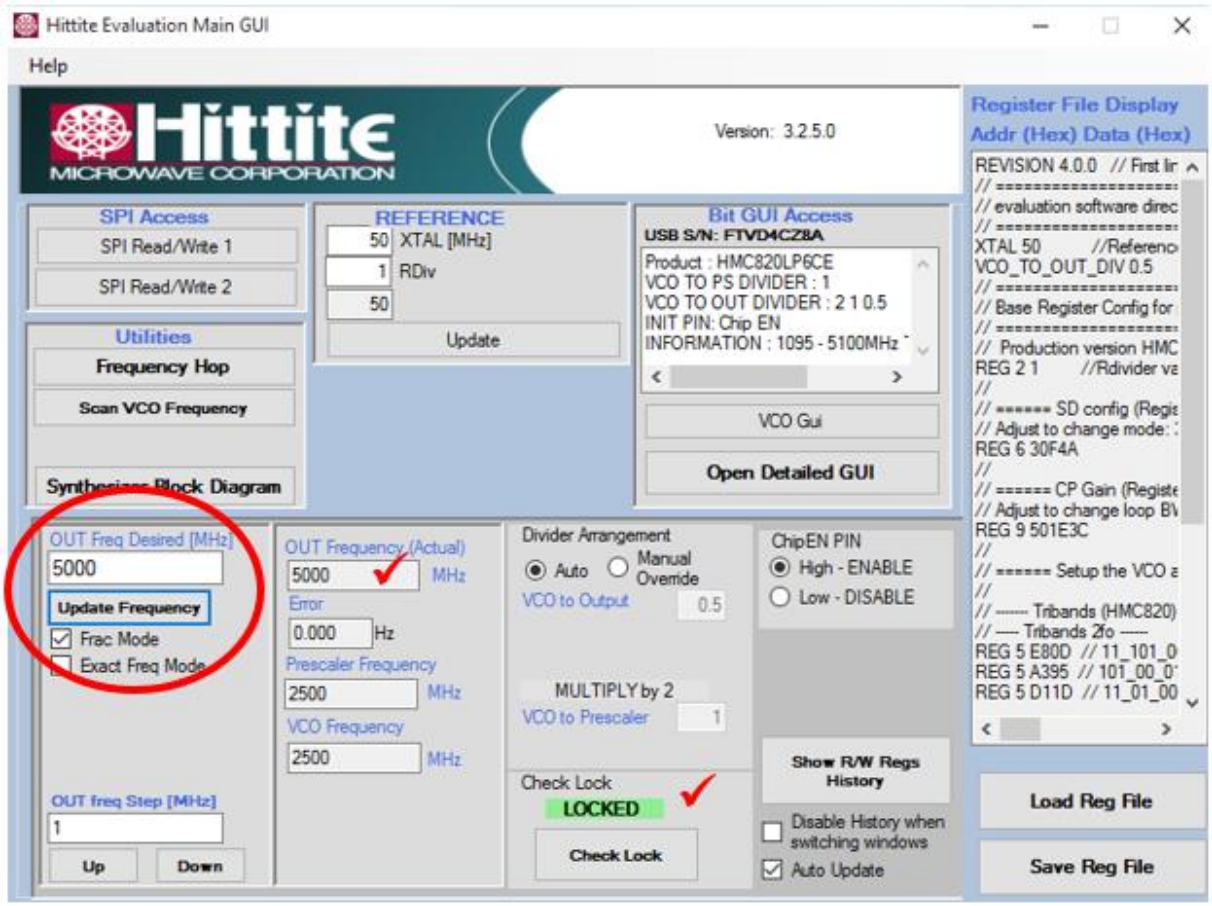

Figure 4.1e: VCO generating 5GHz signal

4.3 FPGA Control

The Nexys-3 Spartan-6 FPGA Board by Xilinx is used to control the radar. The FPGA is plugged into the laptop via USB port on the computer and micro USB port on the FPGA.

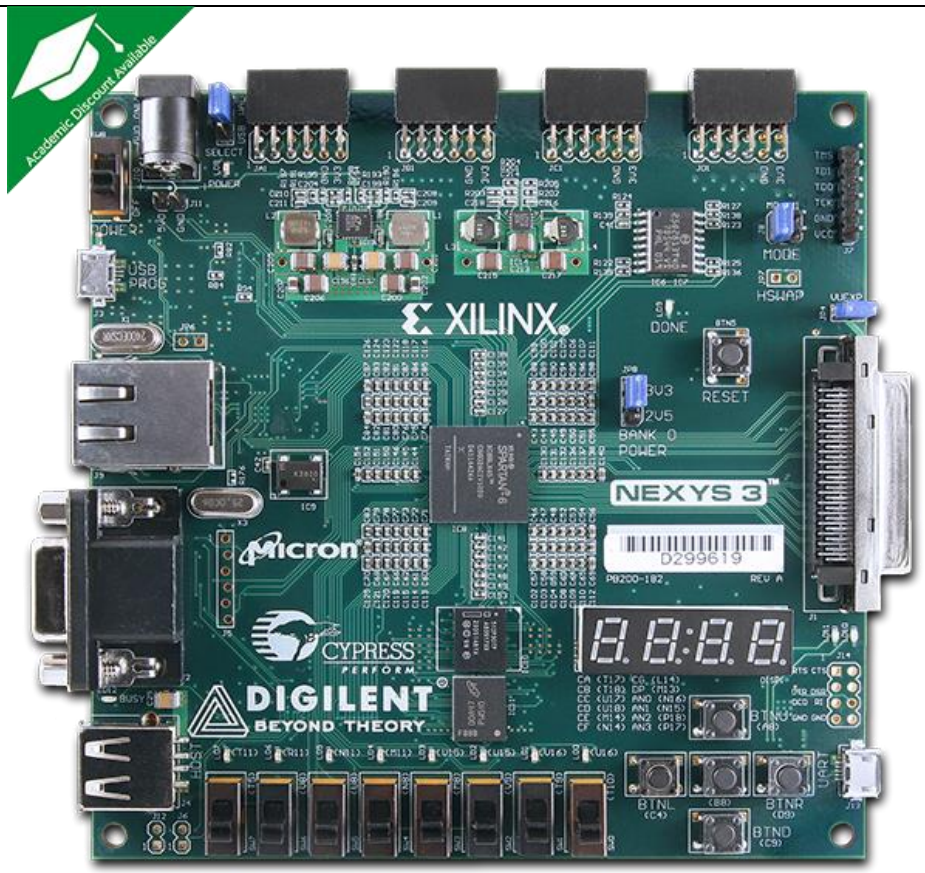

Figure 4.3a: Nexys-3 FPGA Board

To program and use the board, the **ISE DESIGN SUITE 13.4** and the **ADEPT2** software have to be downloaded. The ISE DESIGN SUITE is a Platform Studio and Embedded Development Kit for the board. The ADEPT2 is a unique and powerful solution which allows the user to communicate with Digilent system boards and a wide assortment of logic devices. Both software can be found on the XILINX website. When a file is compiled, a .bit file is created. This .bit file is what is downloaded onto the FPGA.

1. Once you download Xilinx Design Suite 13.4, a node locked license needs to be acquired. Instructions on how to do this specifically are given when downloading Xilinx Design Suite 13.4. For more detailed instructions, look on the Xilinx website. Use the ISE software to reconfigure the code.

a User is not advised to change any part of the code until complete mastering of the SAR project is acquired. Further information about the project code and software development can be found on the final report.

2. The code for this project is on the school website as well as on the project USB flash drive. Under 2014-2015 SAR CODE >> Code version >> library testing >> test library >> Range_Test

3. Go to the adept software, browse to correct address, and select TOP_range.bit. Click program and then the FPGA will be programmed.

4. It is recommended to reset the program from here, using the SW[7] switch up to reset and switch it back down. See, Figure 4.3c for reference. The 7-segment display should 0 at that point.

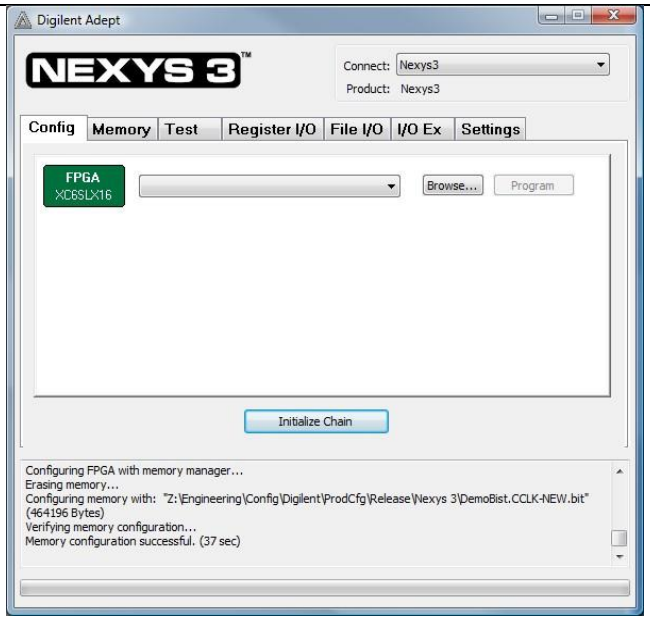

Figure 4.3b: Digilent Adept software

The FPGA has different modes of operation that can controlled by the position of the slider switches on the board. The two main modes of operation are

- Debugging mode
- Real Time mode

In Debugging mode, the user is allow to activate each transmit and receive antenna pair in order to check the fidelity of the signal that is coming in that particular path. This mode is activate at the beginning, as long as the SW[0] SW[1] SW[2] SW[3] SW[4] SW[5] and SW[0]is not all high.

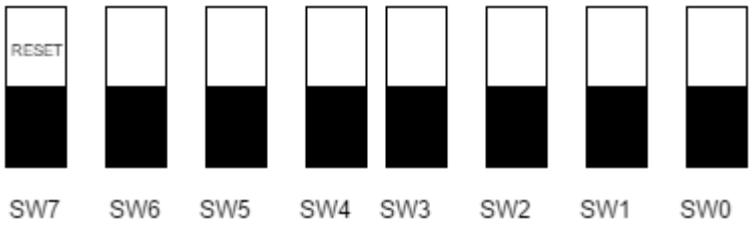

Figure 4.3c: FPGA Switches

In order to switch between transmit/receive antenna pair, the following table is used. This combination is only used to switch the antenna located on the X-axis. In the table, DOWN is referred to a (↓) motion of the switch whereas UP refereed to the (↑) motion of the switch.

Table 4.3a: Transmit/receive manual switching

TRANMIT	RECEIVE	SW[5]	SW[4]	SW[3]	SW[2]	SW[1]	SW[0]
TR1	R1	DOWN	DOWN	DOWN	DOWN	DOWN	UP
	R2	DOWN	DOWN	DOWN	DOWN	UP	DOWN
	R3	DOWN	DOWN	DOWN	DOWN	UP	UP
	R4	DOWN	DOWN	DOWN	UP	DOWN	DOWN
	R5	DOWN	DOWN	DOWN	UP	DOWN	UP
	R6	DOWN	DOWN	DOWN	UP	UP	DOWN
	R7	DOWN	DOWN	DOWN	UP	UP	UP
	R8	DOWN	DOWN	UP	DOWN	DOWN	DOWN
TR2	R1	DOWN	DOWN	UP	DOWN	DOWN	UP
	R2	DOWN	DOWN	UP	DOWN	UP	DOWN
	R3	DOWN	DOWN	UP	DOWN	UP	UP
	R4	DOWN	DOWN	UP	UP	DOWN	DOWN
	R5	DOWN	DOWN	UP	UP	DOWN	UP
	R6	DOWN	DOWN	UP	UP	UP	DOWN
	R7	DOWN	DOWN	UP	UP	UP	UP

	R8	DOWN	UP	DOWN	DOWN	DOWN	DOWN

In addition, with Transmit/Receive path selected, the user can select to look at the digital voltage of the I Channel, I-Bar Channel, and The Q-bar Chanel. This voltage is represented as hexadecimal value and only show positive voltages. Negative voltage are represent as 000

Notes: The voltage is only display when the button is pressed.

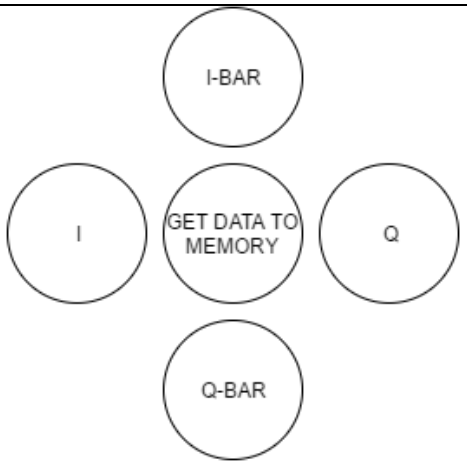

Figure 4.3d: FPGA pushbutton functions

Real Time mode. To switch to real time mode, the user has to switch SW[5] to SW[0] up. At this point, the user should see, the on board led light up sequentially. This mode is to be used with the VGA display.

Memory mode. This mode is there for the user to take data from the scene into the memory of the FPGA. This is because this version of the SAR radar is intended to do the processing in MATLAB. In order to get to that mode, it is recommended to do the following:

1. Put all the switches down.
2. Perform a reset by switching SW[7] up and down.
3. And press the middle button in the push button (see figure 4).
4. The LED on the board should do a full cycle and then stop.
5. The data is ready to be extracted.
6. Go to Adept software and select the Memory tab.

7. Select RAM.
8. In the “read memory content to file” field box browse to where the data is expecting, and select read.

A binary file is created, and the data can now be read in MATLAB, or other program.

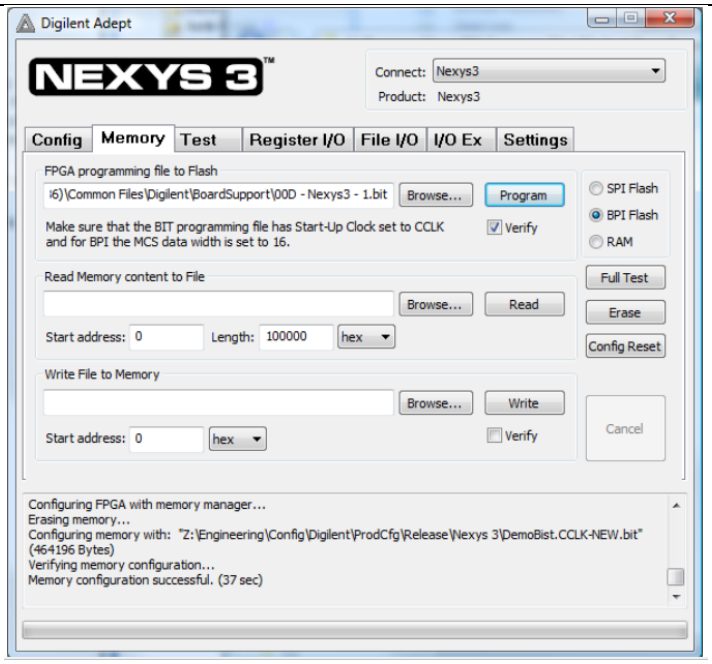

Figure 4.3e: Memory mode

4.4 Signal Processing

Once the SAR system has taken enough samples of the In Phase (I) and Quadrature (Q) data to reach a stable DC voltage level from the target downrange, the next step will be to process this data using MATLAB. It is important to note that the first set of data collected and processed be from the corner reflector at boresight, and done every first every time the system is turned on, as this data is necessary for calibration purposes. There are several .m files that need to be added to the MATLAB workspace in order to do these operations. These files are called: ‘sarFFT.m’, ‘readI.m’, ‘readQ.m’, ‘calibrationI.m’ and ‘calibrationQ.m’. These are the MATLAB code files that were designed and written to perform the signal processing for this system, from data collection, to the Fast Fourier Transform (FFT) and for the correction calculations to fix error when going off at a larger phase slope. The entire process of signal processing will take roughly 1 to 3 minutes, depending on the user.

1) From the computer controlling the FPGA, save the 1 by 128 array binary file ‘fpga_data.bin’ containing the data from the I, Ibar, Q and Qbar channels to the same MATLAB workspace containing the rest of the .m files used for processing.
2) To open the data and from the binary file, use the following line of MATLAB code:

   >> a = fopen(‘fpga_data.bin’);

   This will save the actual binary data file to MATLAB in order to further use this data.
3) The next step would be to open this binary file in order the read the 1 by 128 size array containing the upper byte and lower byte of all received I and Q data. Use the following line of code to save this into an array form that MATLAB can use to process:

   >> dat = fread(a);

   It is important that the array be saved into the variable with the exact name ‘dat’ as this is necessary to run the specified MATLAB functions for processing.
4) To convert the data from the FPGA file into voltage values given by two 1 by 16 size arrays representing the 16 I and Q pairs, the readI.m and readQ.m functions must be utilized. This is done by the following commands:

   >> Ival = readI(dat);

   >> Qval = readQ(dat);
5) Now that the data from the FPGA has been stored into MATLAB, this data from boresight must be manipulated for calibration purposes. This calibration is simply the complex conjugate of the I and Q data taken at the scene, then multiplying this data by that same data. To perform this calibration, simply us the code:

   >> Ical = calibrationI(Ival);

   >> Qcal = calibration(Qval);
6) Now the FFT can be implemented by saving these calibration values using the following code:

   >> I = (Ical .* Ival) + (Qcal .*Qval);

   >> Q = [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0];

   >> sarFFT(I,Q)

   This will result in the data becoming symmetrical on either side, after implementing the FFT function, showing that the corner reflector is located at boresight, as seen below in Figure 4.3g.

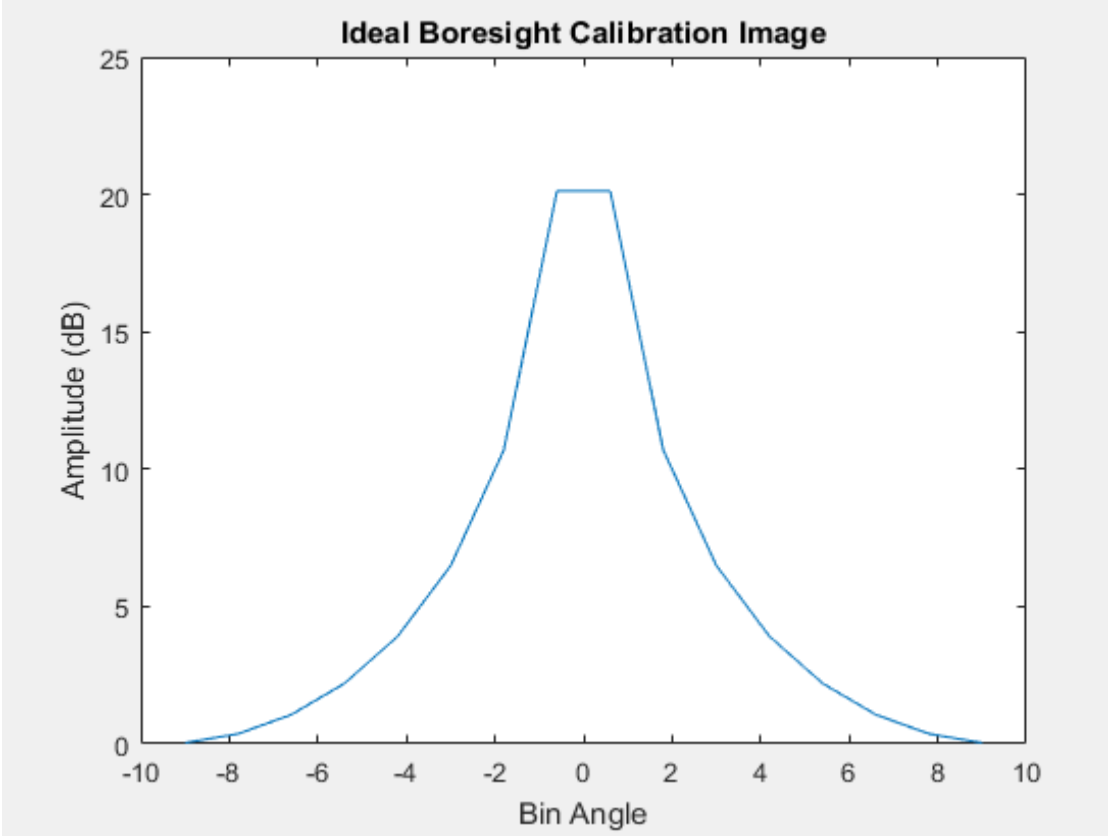

Figure 4.3g: Image at boresight after calibration and FFT (Ideal)

7) Now after calibration has been completed, showing that there is a peak energy level at boresight, data can be taken for any new set of data. To do so, repeat steps 1) – 3) to acquire the new set of data from the FPGA.
8) Run the code to convert the I and Q data into the correct DC voltage array format, with slight alteration from the way this was done during calibration. Run the following code to do so:

```
>> I = readI(dat);
>> Q = readQ(dat);
```

9) Finally, to do the signal processing for any targeting that is not at boresight, run the FFT function written to do the complex multiplication with the basis functions. The code is as follows:

```
>> sarFFT(I,Q)
```

V. Troubleshooting

5.1 Troubleshooting

Common problems will point back to a weak connection in either the wiring or the physical connections. Ensure that all connections are in properly and there are no cuts or tires in any cables or wires. If an electrical component is not secured in place, the standoffs could potentially be loose or the screws simply need to be tightened. External panel mounted connectors being loosely fitted should not affect the actual connections, however it is important that they are also secured tightly onto the sides of the housing to make sure they do not come off or fall during operation or transport.

Loose SMA connections will still conduct a signal, but with far greater loss. If one receive channel is very low compared to the others, check the SMA connections from that particular antenna, through to the SP16T switch, including inner and outer panel mounts. If half of the channels are very low compared to the other half on the same axis, check the SMA connections from the transmit antenna to the SP4T.

The weakest physical connections in the system are from the SP16T pins to the FPGA PMOD ADCs. The pins that control these channels are connected to internal pull-up resistors. Inactive channels should read 3.3V on a multi-meter, and the active channel should read open with respect to ground. Use probe leads to test the pin on the switch, the connector housing, and the FPGA connection. Note: The transmit (SP4T) pins are configured likewise, so when the FPGA is turned off with the rest of the system still powered, the transmit antennas will all transmit because the SP4T no longer has control voltage.

VI. Inventory

Inventory					
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	

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 IQ Demod w/constant source.	COMPLETE: PASS Post-Migration: PASS

		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 and FPGA fast pulse.	PASS with VCO and 25MHz pulse generator

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

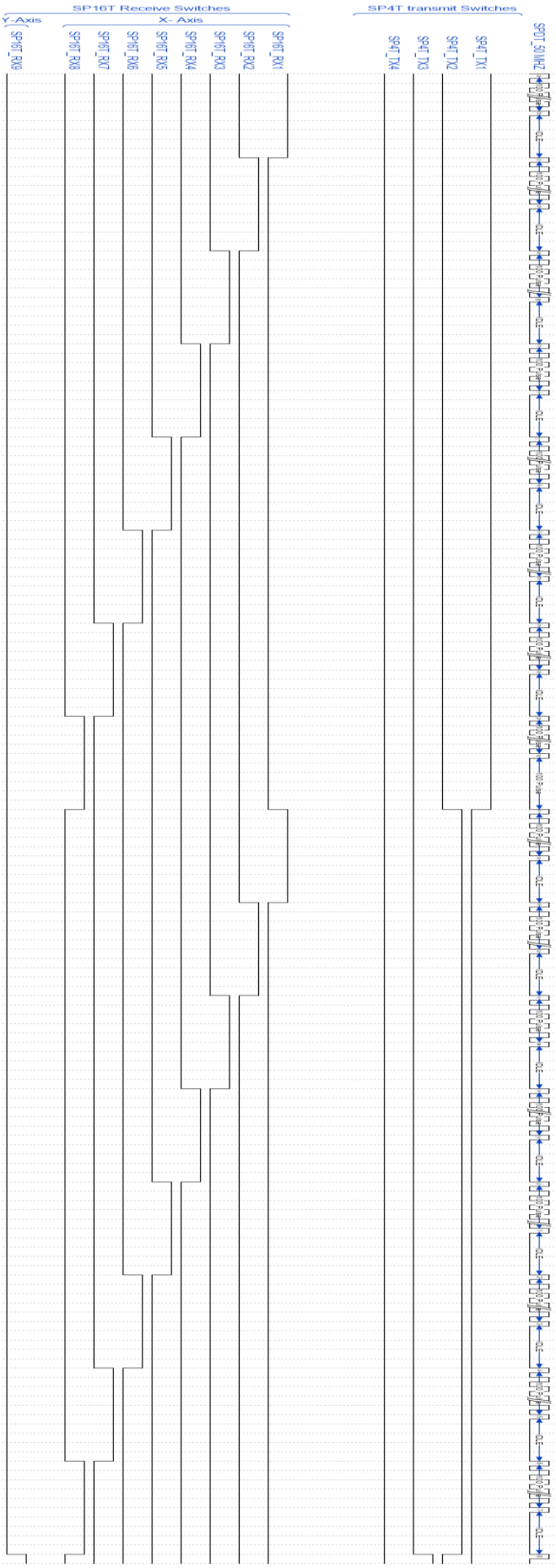
SP16T Receive Switches
Y- Axis
X- Axis
SP16T_RX9
SP16T_RX8
SP16T_RX7
SP16T_RX6
SP16T_RX5
SP16T_RX4
SP16T_RX3
SP16T_RX2
SP16T_RX1
SP4T transmit Switches
SP4T_TX4
SP4T_TX3
SP4T_TX2
SP4T_TX1
SPDT_5GHZ

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(t
h);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);15*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
```