Team Mega Hurtz
Final Group Report

Charlie Saechou
Jonathan Saechou
Vincent Saechao
Will Yang

March 20th, 2018

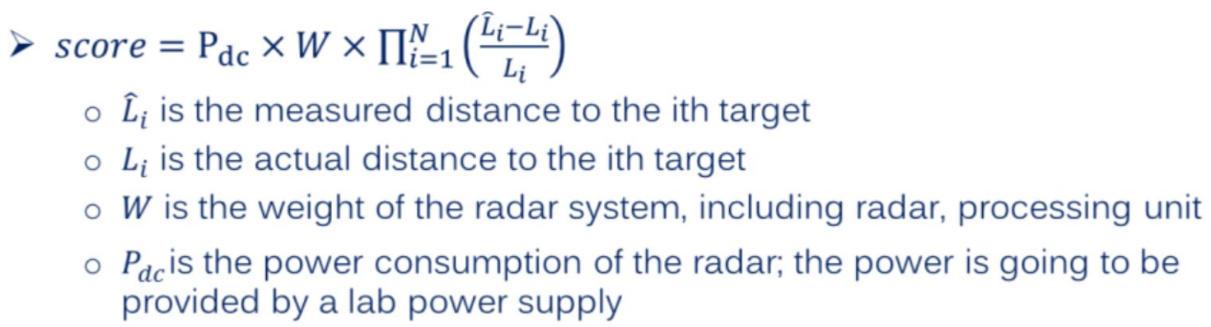
Table Of Contents

Abstract:

This project was split up into two quarters. During fall quarter, we built our own FMCW radar system given helpful guidance of lab manuals and TAs. This quarter, we implemented what we learned about the FMCW radar and attempted to design and build our own improved FMCW radar system. This 2.4 GHz system will be used in an outdoor environment to detect targets at various distances. Our goals for the radar system was to have low power consumption, low overall weight, and high accuracy.

Design Rules/Scoring:

1) Be able to detect a 0.3 x 0.3 m^2 target ranging from 5 meters to 50 meters
2) Budget will be up to $300
3) Scoring



- $score = \mathrm{P_{dc}} \times W \times \prod_{i=1}^{N} \left(\frac{\hat{L}_i - L_i}{L_i}\right)$
  - $\hat{L}_i$ is the measured distance to the ith target
  - L_i is the actual distance to the ith target
  - W is the weight of the radar system, including radar, processing unit
  - P_{dc} is the power consumption of the radar; the power is going to be provided by a lab power supply

Figure (1). Scoring Details

Overall Design Details:

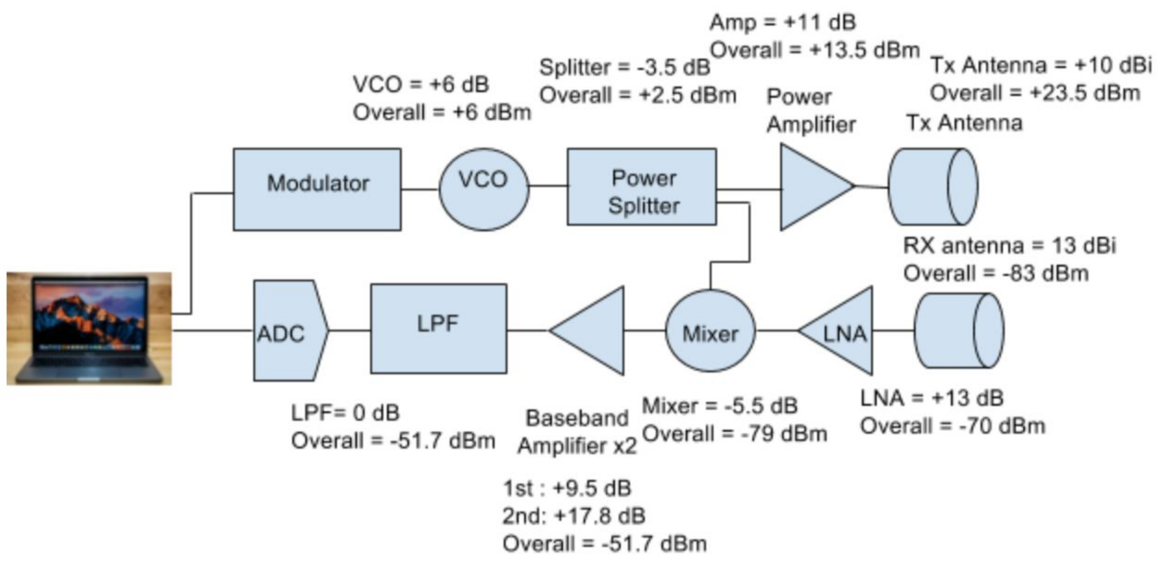
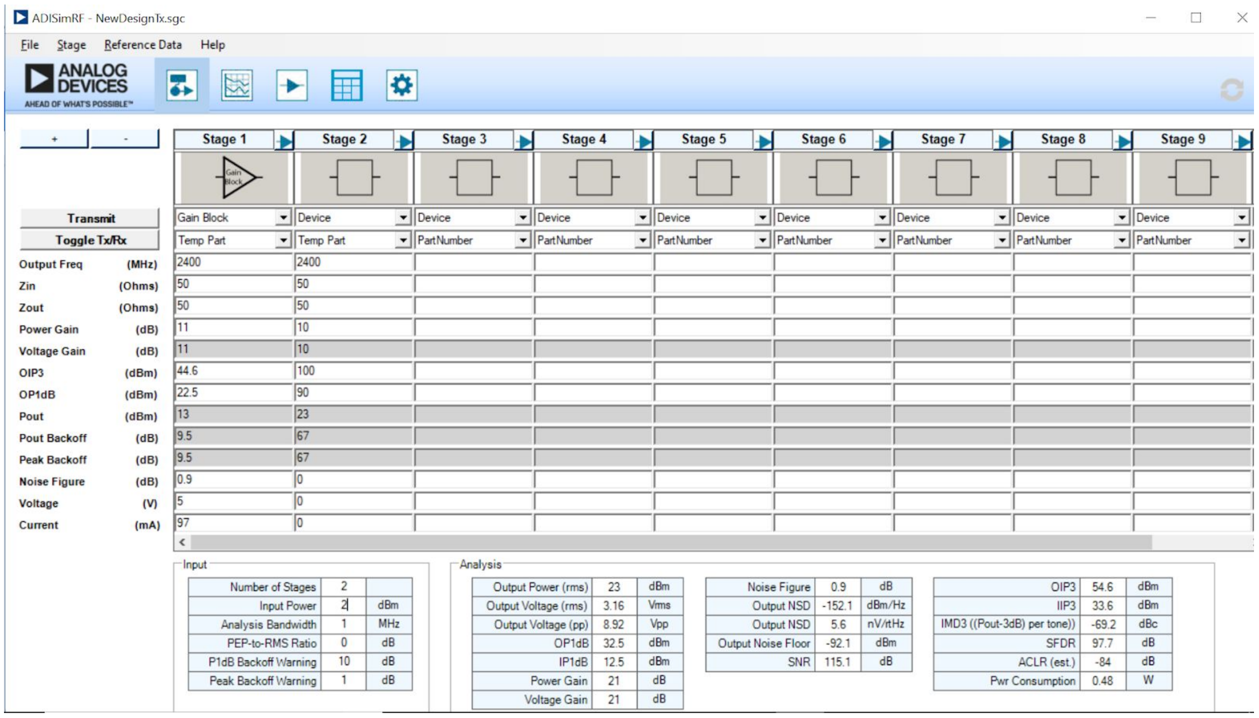


Fig. () Block Diagram

After creating our block diagram based off quarter 1 lab designs, we used ADIsimRF to calculate the gain of the transmitting and receiving power. We had to split the transmitting side and receiving side into two seperate ADI simulations. To implement both parts of the antenna, we found the desired component on websites such as Digikey or Mini-circuits. These websites had specific information about the components to help us determine the gain. Once we put all the components on the simulation, replaced some components to have no errors. The receiving and transmit antenna simulations are shown below.


ADISimRF - NewDesignTx.sgc
File Stage Reference Data Help
ANALOG DEVICES
AHEAD OF WHAT'S POSSIBLE™
+
-
Transmit
Toggle Tx/Rx
Gain Block

	Stage 1	Stage 2	Stage 3	Stage 4	Stage 5	Stage 6	Stage 7	Stage 8	Stage 9
	Gain Block	Device	Device	Device	Device	Device	Device	Device	Device
	Temp Part	Temp Part	PartNumber	PartNumber	PartNumber	PartNumber	PartNumber	PartNumber	PartNumber
Output Freq (MHz)	2400	2400							
Zin (Ohms)	50	50							
Zout (Ohms)	50	50							
Power Gain (dB)	11	10							
Voltage Gain (dB)	11	10							
OIP3 (dBm)	44.6	100							
OP1dB (dBm)	22.5	90							
Pout (dBm)	13	23							
Pout Backoff (dB)	9.5	67							
Peak Backoff (dB)	9.5	67							
Noise Figure (dB)	0.9	0							
Voltage (V)	5	0							
Current (mA)	97	0							

Input

Number of Stages	2	
Input Power	2	dBm
Analysis Bandwidth	1	MHz
PEP-to-RMS Ratio	0	dB
P1dB Backoff Warning	10	dB
Peak Backoff Warning	1	dB

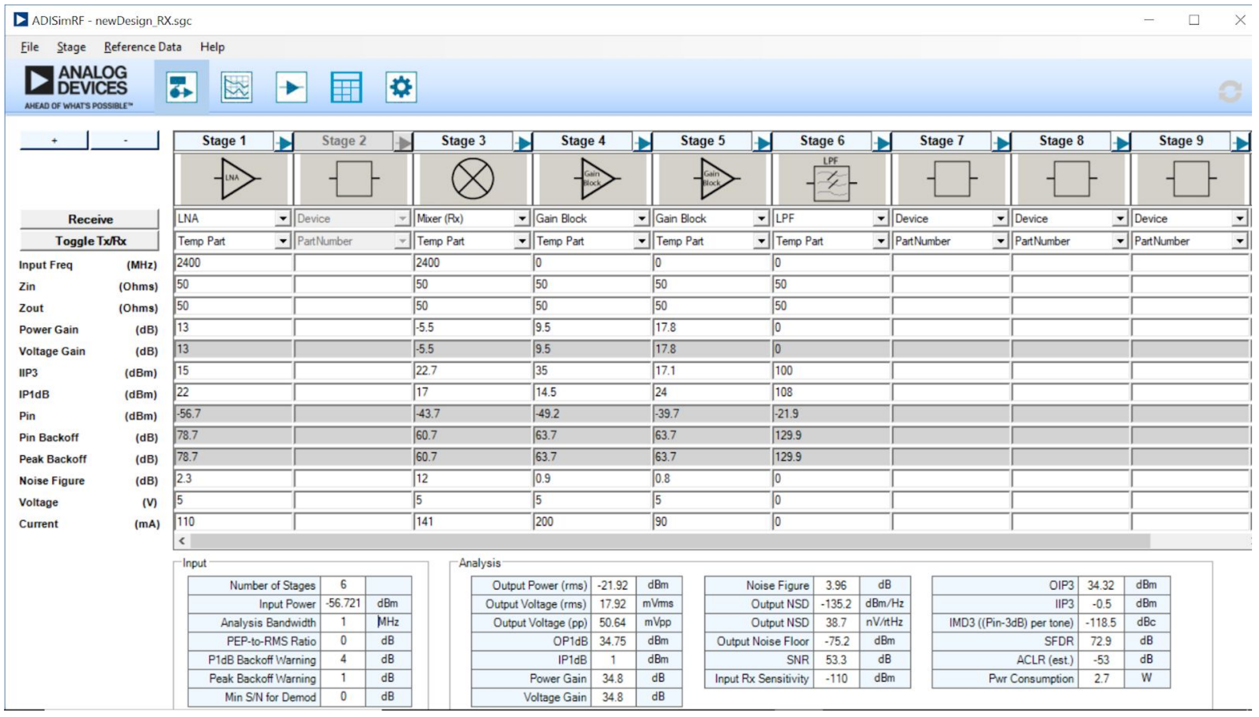
Analysis

Output Power (rms)	23	dBm
Output Voltage (rms)	3.16	Vrms
Output Voltage (pp)	8.92	Vpp
OP1dB	32.5	dBm
IP1dB	12.5	dBm
Power Gain	21	dB
Voltage Gain	21	dB

Noise Figure	0.9	dB
Output NSD	-152.1	dBm/Hz
Output NSD	5.6	nV/rtHz
Output Noise Floor	-92.1	dBm
SNR	115.1	dB

OIP3	54.6	dBm
IIP3	33.6	dBm
IMD3 ((Pout-3dB) per tone))	-69.2	dBc
SFDR	97.7	dB
ACLR (est.)	-84	dB
Pwr Consumption	0.48	W

Fig. () ADIsim for Transmitting Antenna


ADISimRF - newDesign_RX.sgc
File Stage Reference Data Help
ANALOG DEVICES
AHEAD OF WHAT'S POSSIBLE™
+
-
Receive
Toggle Tx/Rx
LNA
Gain Block
Gain Block
LPF

	Stage 1	Stage 2	Stage 3	Stage 4	Stage 5	Stage 6	Stage 7	Stage 8	Stage 9
	LNA	Device	Mixer (Rx)	Gain Block	Gain Block	LPF	Device	Device	Device
	Temp Part	PartNumber	Temp Part	Temp Part	Temp Part	Temp Part	PartNumber	PartNumber	PartNumber
Input Freq (MHz)	2400		2400	0	0	0			
Zin (Ohms)	50		50	50	50	50			
Zout (Ohms)	50		50	50	50	50			
Power Gain (dB)	13		-5.5	9.5	17.8	0			
Voltage Gain (dB)	13		-5.5	9.5	17.8	0			
IIP3 (dBm)	15		22.7	35	17.1	100			
IP1dB (dBm)	22		17	14.5	24	108			
Pin (dBm)	-56.7		-43.7	-49.2	-39.7	-21.9			
Pin Backoff (dB)	78.7		60.7	63.7	63.7	129.9			
Peak Backoff (dB)	78.7		60.7	63.7	63.7	129.9			
Noise Figure (dB)	2.3		12	0.9	0.8	0			
Voltage (V)	5		5	5	5	0			
Current (mA)	110		141	200	90	0			

Input

Number of Stages	6	
Input Power	-56.721	dBm
Analysis Bandwidth	1	MHz
PEP-to-RMS Ratio	0	dB
P1dB Backoff Warning	4	dB
Peak Backoff Warning	1	dB
Min S/N for Demod	0	dB

Analysis

Output Power (rms)	-21.92	dBm
Output Voltage (rms)	17.92	mVrms
Output Voltage (pp)	50.64	mVpp
OP1dB	34.75	dBm
IP1dB	1	dBm
Power Gain	34.8	dB
Voltage Gain	34.8	dB

Noise Figure	3.96	dB
Output NSD	-135.2	dBm/Hz
Output NSD	38.7	nV/rtHz
Output Noise Floor	-75.2	dBm
SNR	53.3	dB
Input Rx Sensitivity	-110	dBm

OIP3	34.32	dBm
IIP3	-0.5	dBm
IMD3 ((Pin-3dB) per tone)	-118.5	dBc
SFDR	72.9	dB
ACLR (est.)	-53	dB
Pwr Consumption	2.7	W

Fig. () ADIsim for Receiving Antenna

Component	Model Number	URL
LNA (1)	HMC 639ST89E	https://www.digikey.com/product-detail/en/analog-devices-inc/HMC639ST89E/1127-3004-ND/5359984
VCO	ROS-2490+	https://www.minicircuits.com/WebStore/dashboard.html?model=ROS-2490%2B
Mixer	Sim-63LH+	https://www.minicircuits.com/WebStore/dashboard.html?model=SIM-63LH%2B
LPF	MAX291ESA+	https://www.digikey.com/product-detail/en/maxim-integrated/MAX291ESA/MAX291ESA-ND/1513302
Amplifier (Rx) #1	PGA-103	https://www.minicircuits.com/WebStore/dashboard.html?model=PGA-103%2B
Amplifier (Tx)	PGA-103	https://www.minicircuits.com/WebStore/dashboard.html?model=PGA-103%2B
Amplifer (Rx) #2	MMG20241H	https://www.mouser.com/productdetail/nxp-freescale/mmg20241ht1?qs=sGAEpiMZZMvlz5n0fllKWCPT5hyshv%2FsuQS0BdS5sXs%3D
Antenna	Yagi	http://www.wa5vjb.com/pcb-pdfs/Yagi2400.pdf or http://www.wa5vjb.com/products2.html
Splitter	BP2U+	https://www.minicircuits.com/WebStore/dashboard.html?model=BP2U%2B

Table () Component List

PCB Design:

To design our pcb schematics, we used the datasheets based off the components we chose. Each component had a specific layout consisting of resistors, capacitors and inductors. The data sheet also gave us the specific values for them. Once we had our schematic prepared, we had to create the footprints of each component based off of the dimensions given in the datasheet. Luckily, we went through many pcb designs in the first quarter to know that creating the footprints correctly is a big deal because if it is wrong, it will be hard to solder the components on when the pcb board comes back. An important thing to remember is that all components that are not on the KiCad software requires a new footprint. Though KiCad may have some equivalent footprint design, there most likely is something that is different. As a caution, we created new footprints for each component we used to ensure the size on the PCB was correct.

For our design, we chose to split the baseband and RF pcb into two seperate pcbs. This will allow us to test the two pcbs separately and will make it easier for us to troubleshoot when a problem arises. Also, if one were to not work, we wouldn't have to through the whole design away. We would be able to use the half that does work and build the other half based off of quarter one designs.

As we designed the RF board, there were many things we had to take into consideration. The RF PCB contained the RF signals, which is very important in testing the radar system. So when designing, we had to make sure the traces for the RF board were completely straight as it can be to avoid loss in signal from sharp edges. When the board was complete, we added via fences to connect the top and bottom ground fills. This same process was repeated for the baseband PCB.

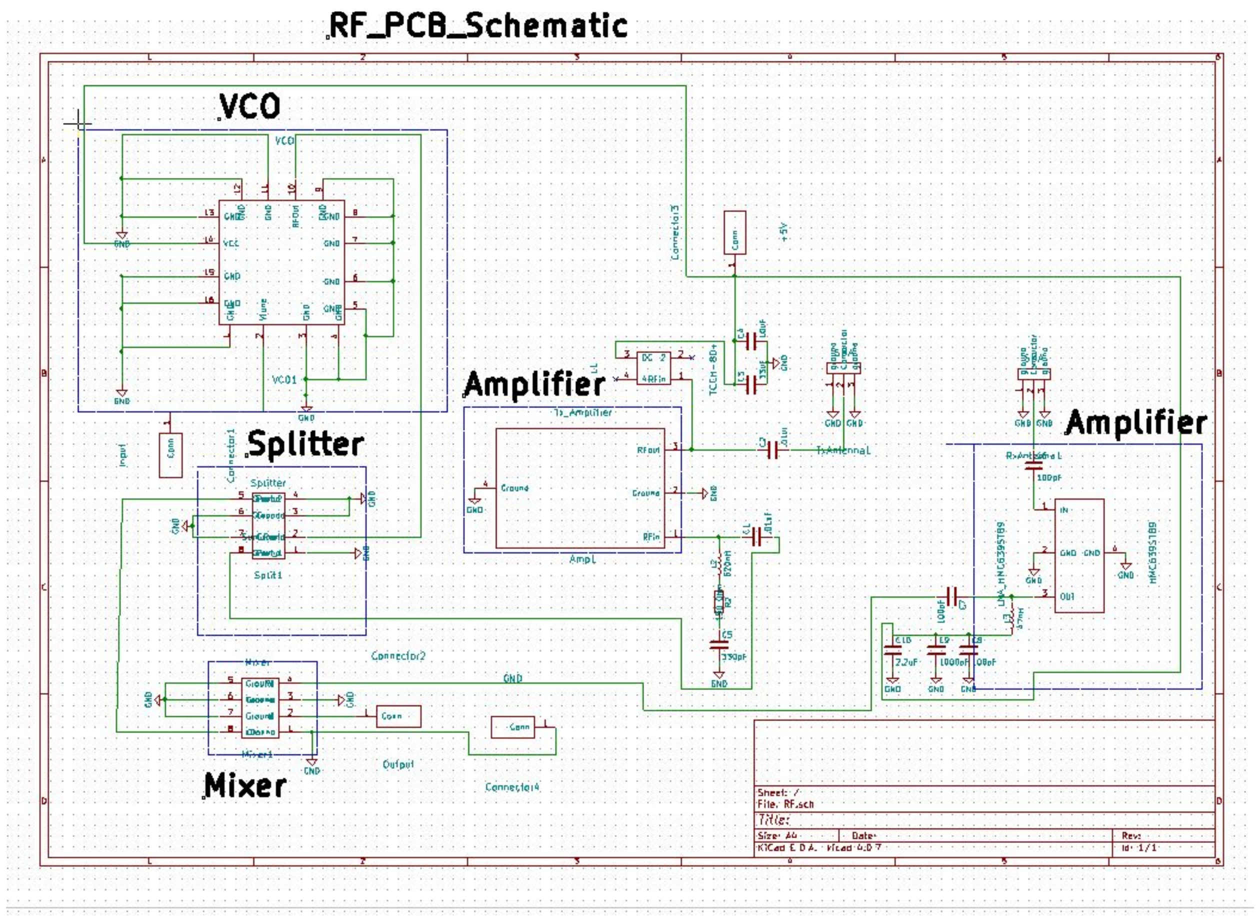


Fig. () RF Schematic

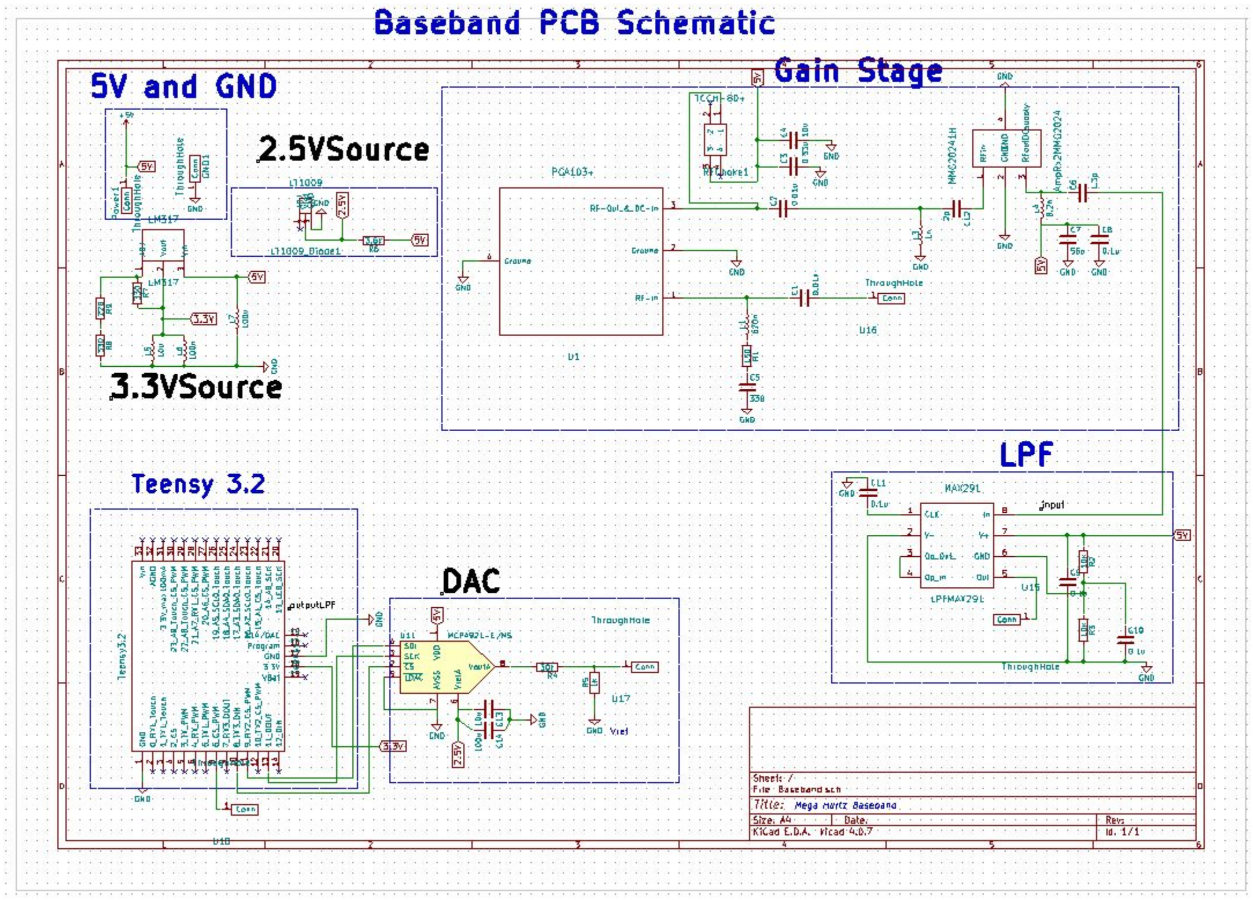


Fig. () Baseband Schematic

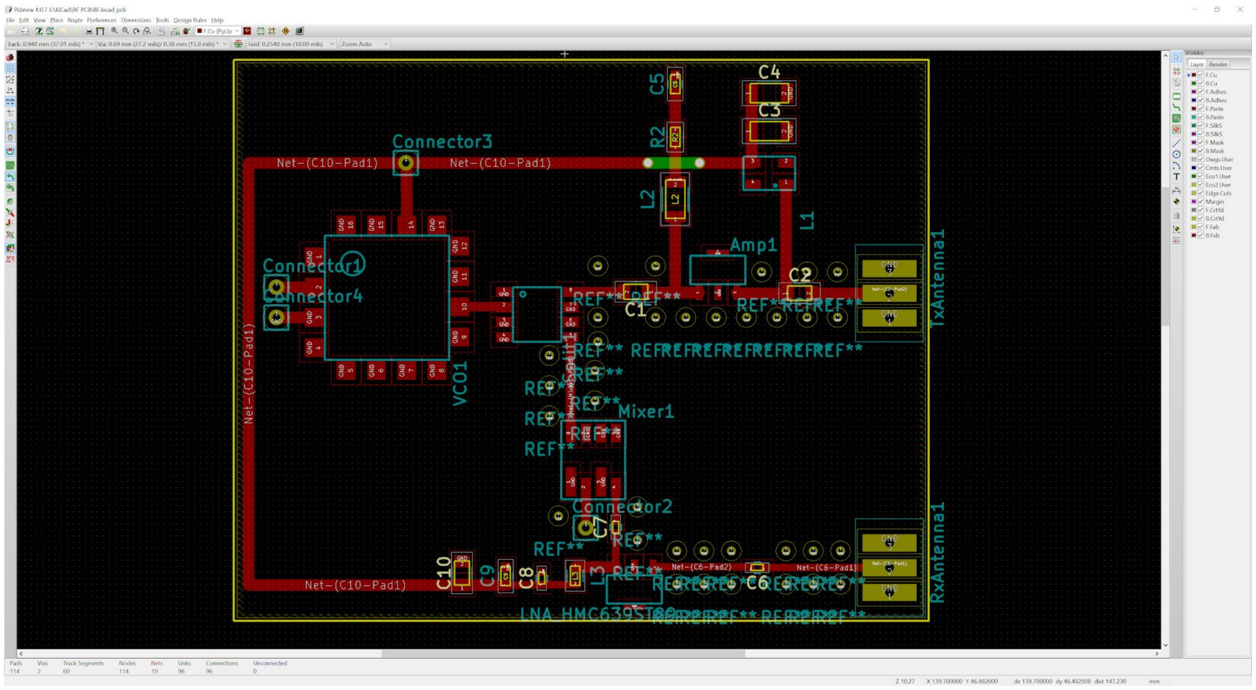


Fig. () RF PCB Layout

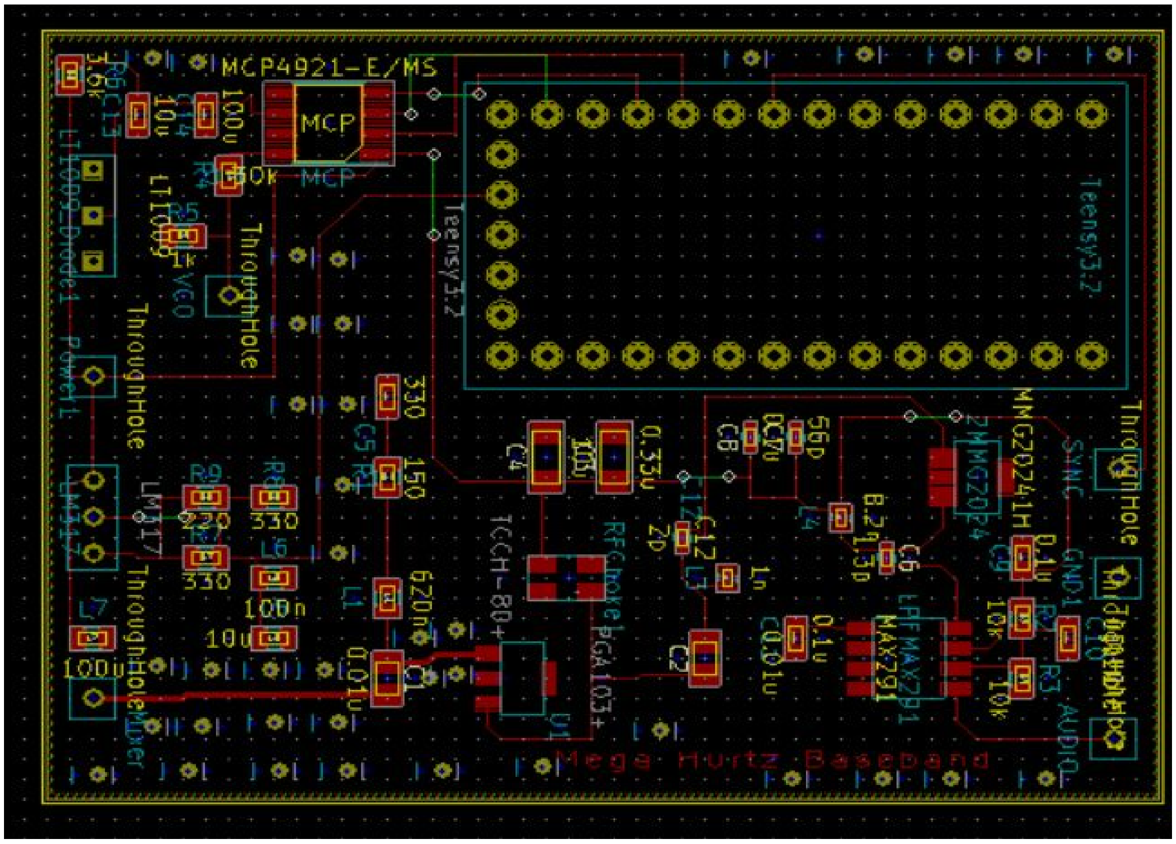


Fig. () Baseband PCB Layout

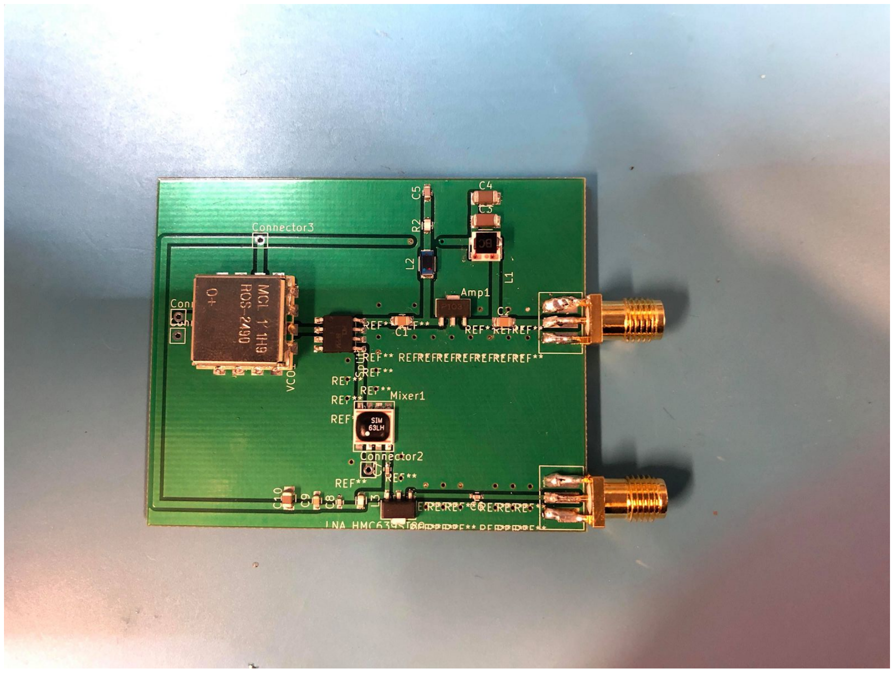


Fig. () RF PCB Soldered

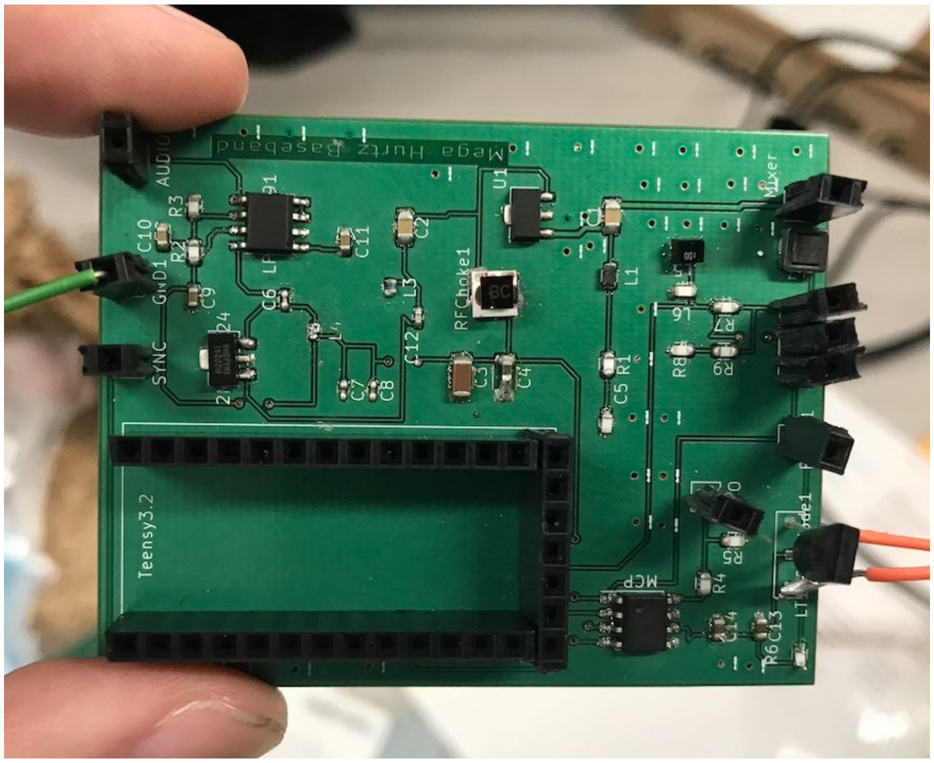


Fig. () Baseband PCB Soldered

Antenna Choice:

Based off previous teams, we chose to go with the Yagi Antenna for our Transmitting Antenna and the coffee cans from quarter 1 as our Receiving antenna. This antenna gave our radar system a high gain. According to the Yagi datasheet, the antenna had a maximum signal of 10-11dBi at 2.4 GHz. However, when actually using the antenna, the bandwidth was a lot smaller than advertised.

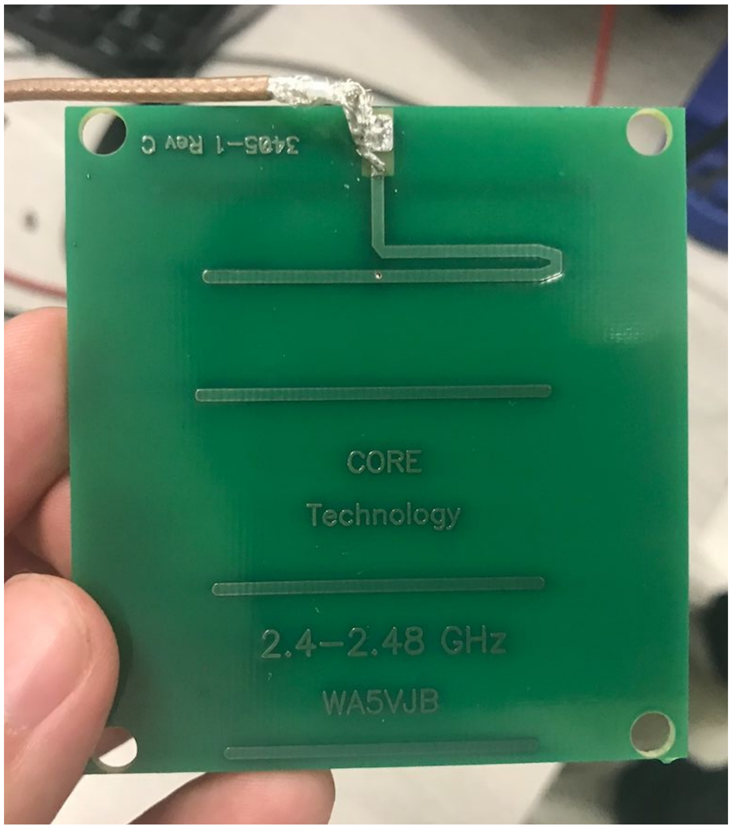


Fig. () Yagi Antenna (Transmit)

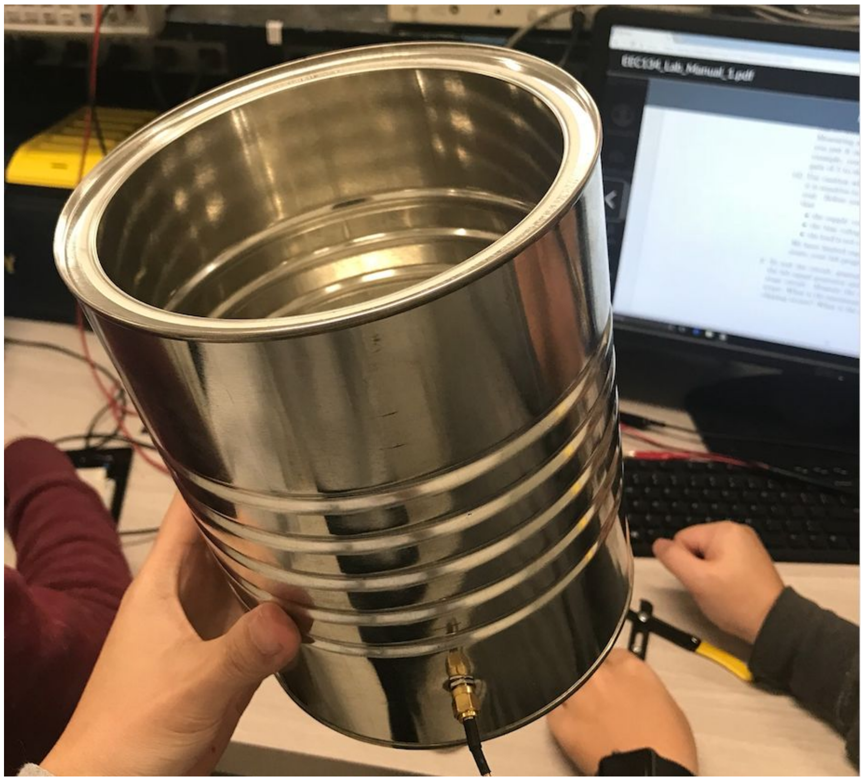


Fig. () Coffee Can (Receive)

Final Design Testing:

When we tested our RF pcb system in lab, we were able to amplify the signal as we moved a metal plate away and to the antennas. As shown in the figures below, the signal was a little messy but you can clearly see the signal does get larger in addition to the change in frequency. This made us believe our RF pcb board worked.

When we tried to test our baseband pcb, we kept short circuiting the board. It was hard to determine what caused the short circuit because there were many possibilities such as the soldering touching, nodes connected to the wrong place in the pcb design, forgetting to ground a component, etc. This being the case, and time running out, we chose to not use the baseband design and to build our quarter 1 system to compete with.

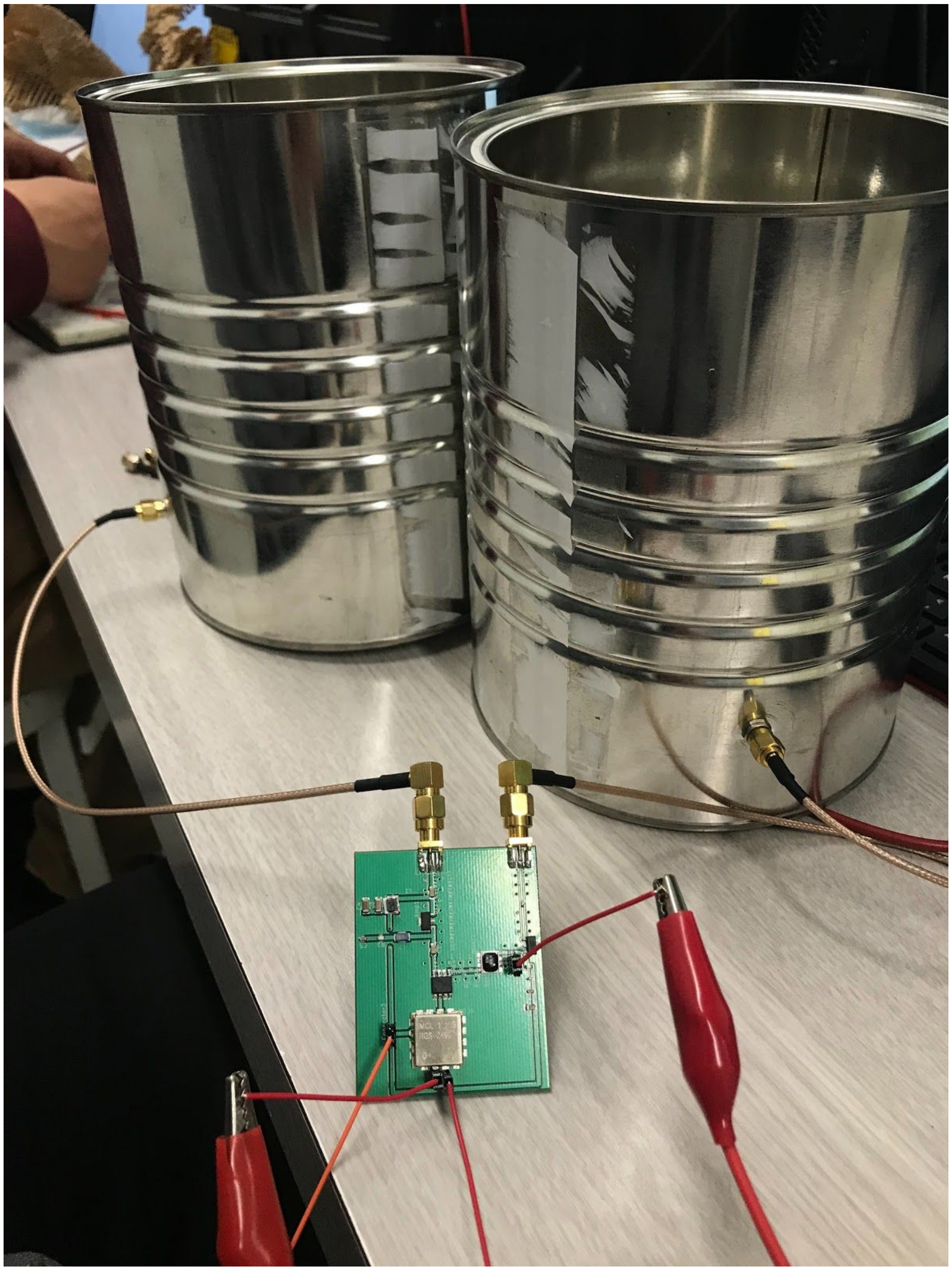


Fig. () Setup Test of RF PCB in lab room

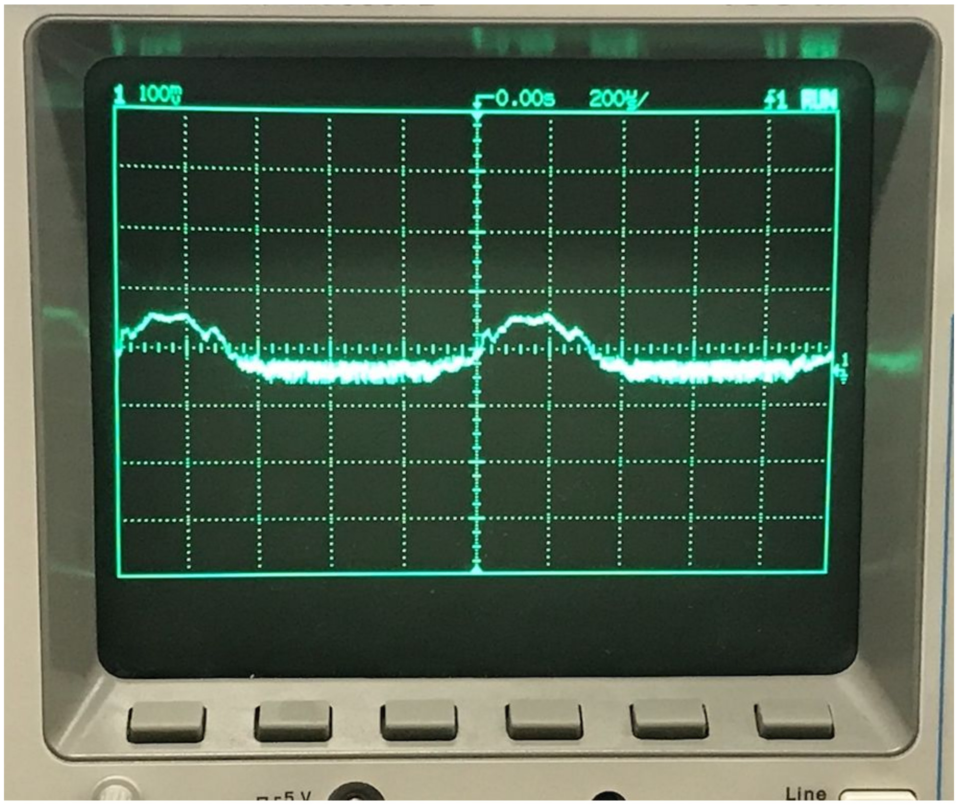


Fig. () Output of RF PCB with no plate

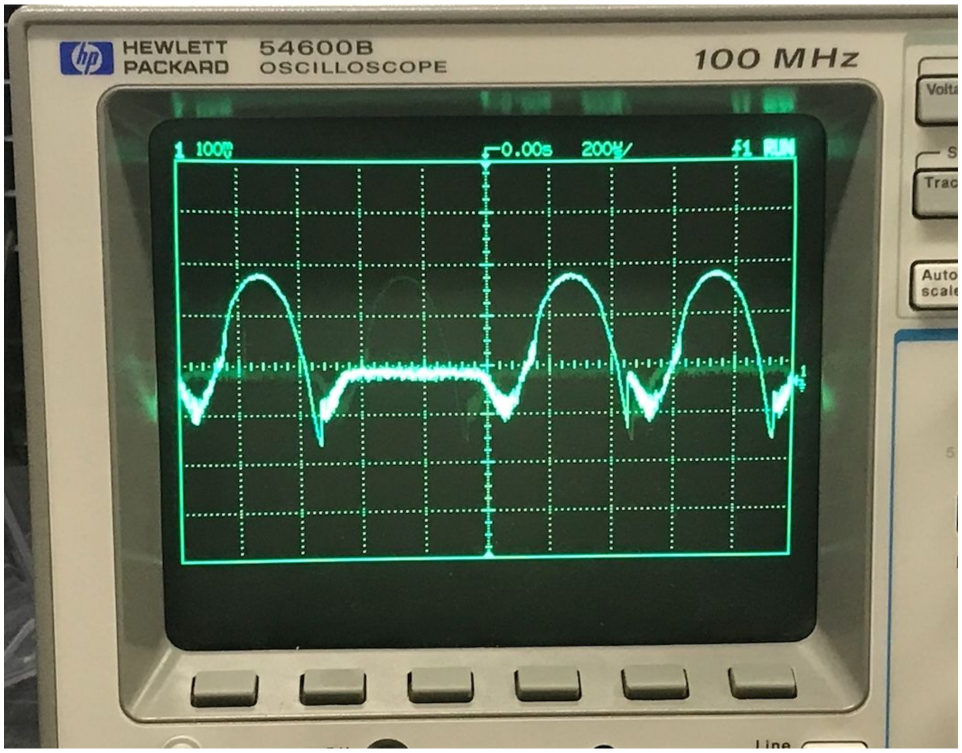


Fig. () Output of RF PCB with plate

Testing the Gain stage:

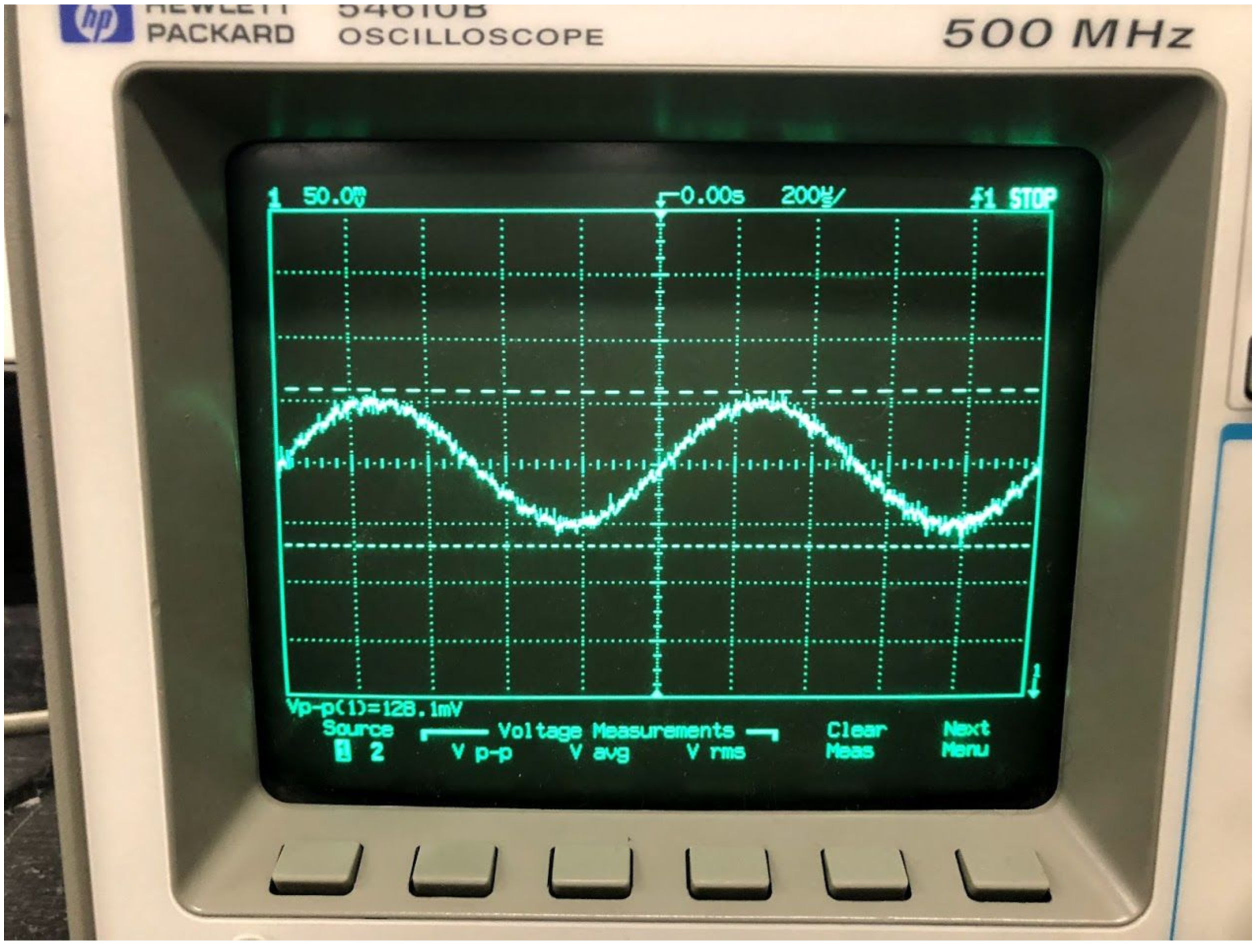


Figure () Input Signal with Vpp of 128.1mV

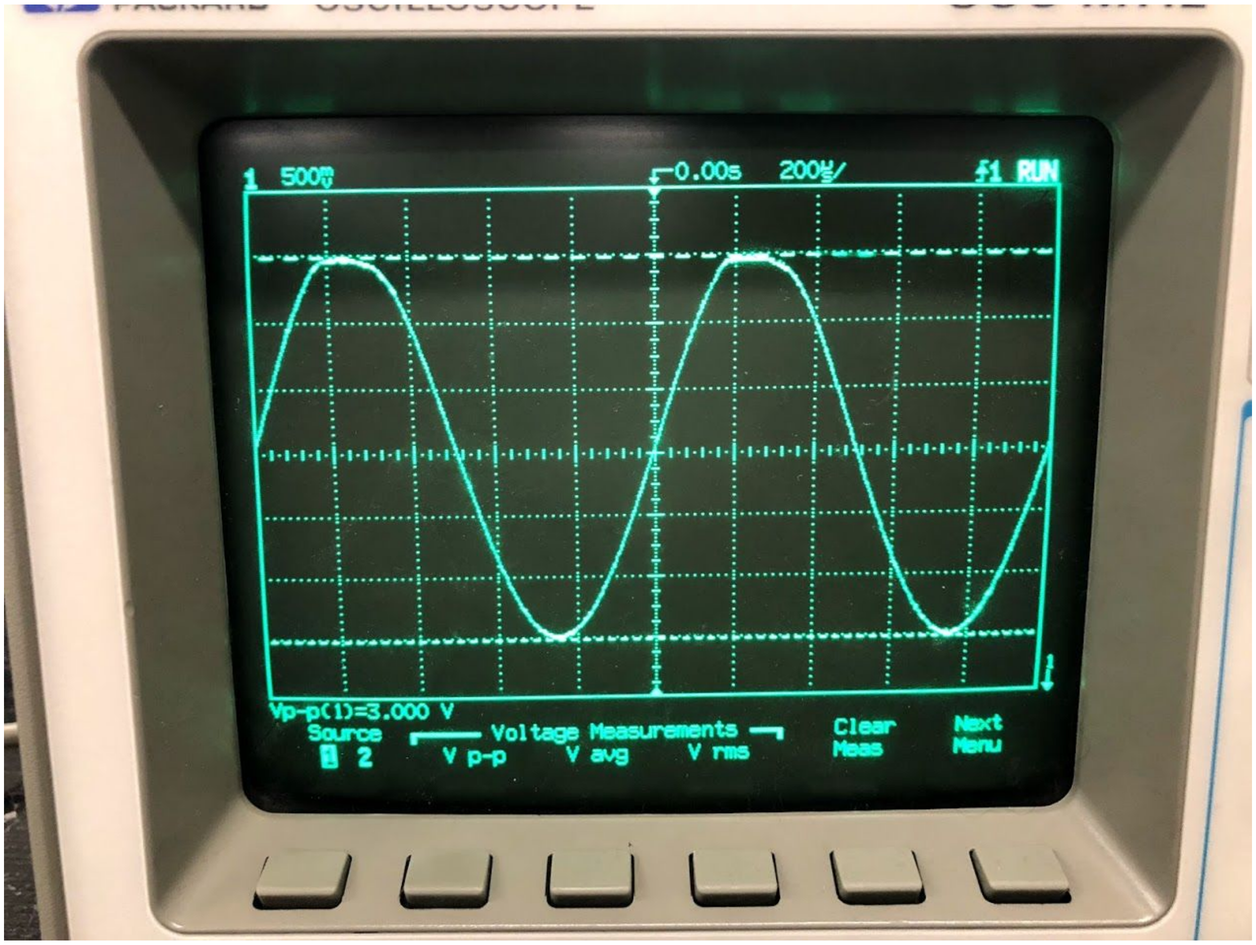
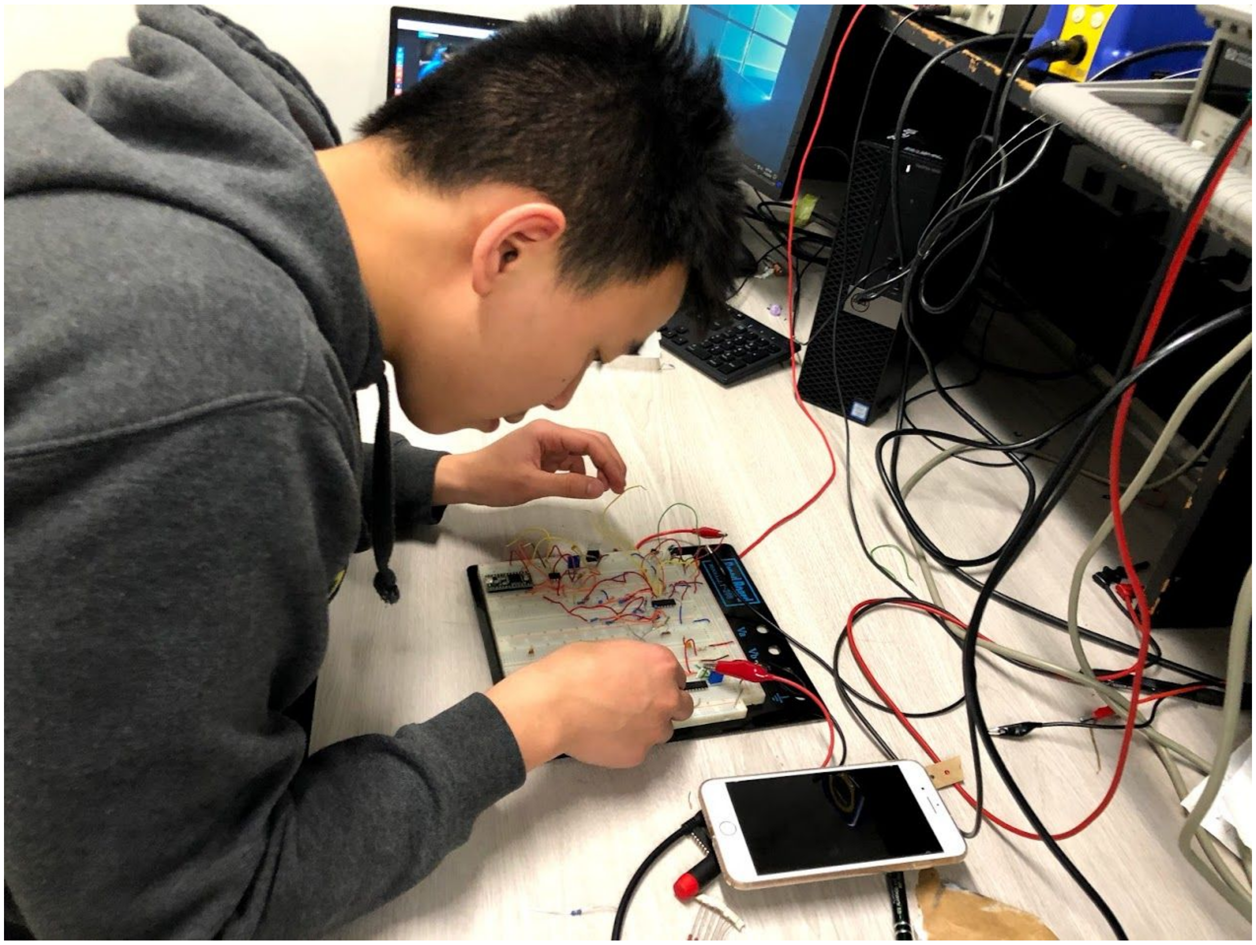
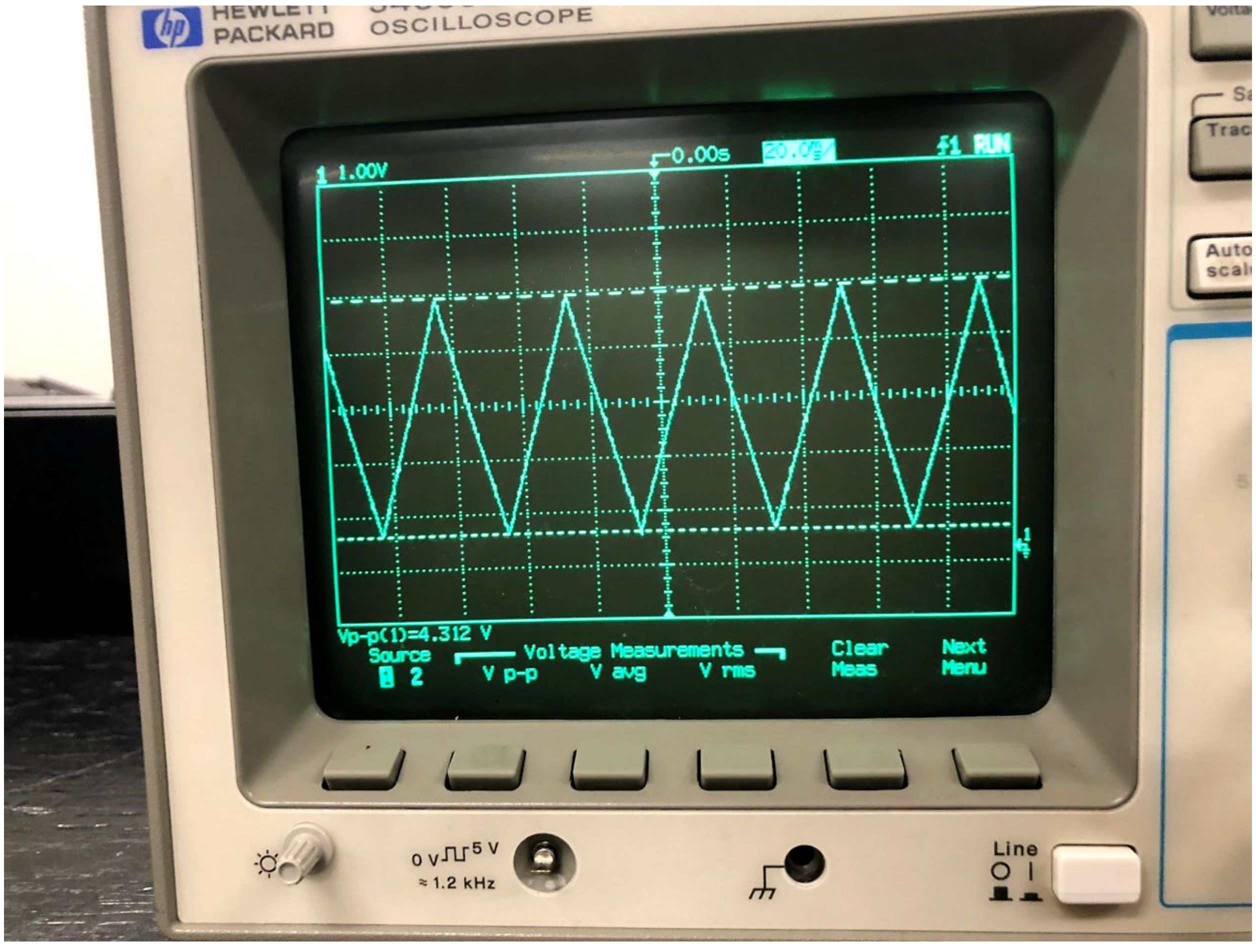


Figure () Vpp of the output of the gain stage of 3V

Testing the LPF:



DAC Output:



Complete System Results

We were able to produce a wav file from the quarter one system and run it through the python code to see the distances indoors but it didn't quite work as well outdoors on the test day.

Competition Day

Due to the rain on the competition day, we were not able to get very accurate results on some of the measurements in addition to the system being just a remake of the quarter one system. Using the quarter one system, we were able to grab the wav file from the field test and find the distances to send to the TAs with the python code.

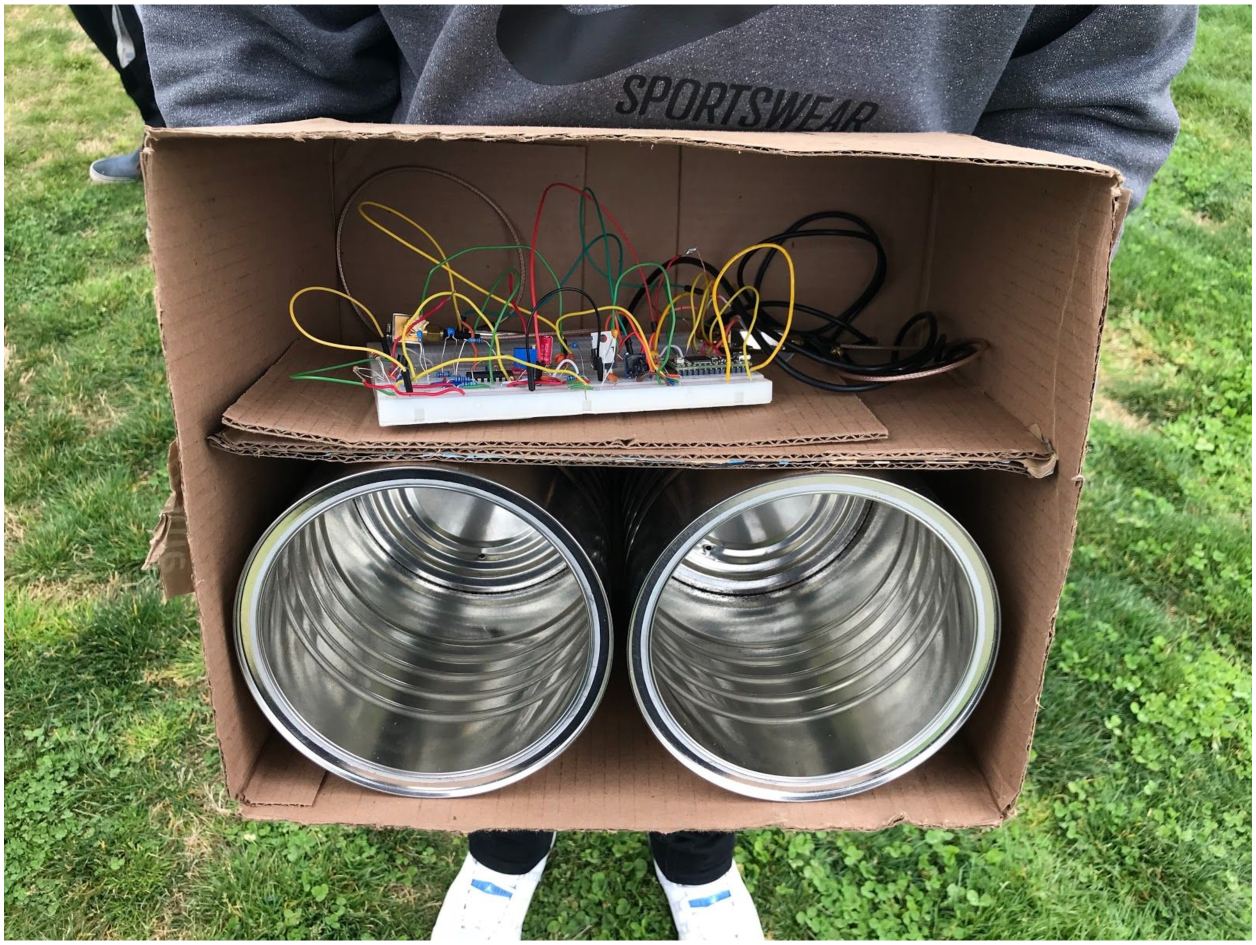


Fig. () Radar system set up

Conclusion

Our team learned a lot during this two quarter course. During the first quarter, our labs were guided so building the radar was quite simple since all the components and designs were given. Though we ran into problems with components not working, we were able to debug each section to determine where the problem occurred. In quarter 2, we realized that building a radar on our own without guidance is quite complicated. We had to decide our own components to use, and making sure each component had enough power to power the next component.

We learned how to create a PCB and then improved on this skill by learning more about how each connection should be made. When testing the actual PCBs and breadboards, we knew to check every connection. This ensured that the physical board we had was what we designed on KiCad. When testing the radar system outside, we noticed the system performs different than inside lab. However, we were able to get results and send them to the TAs for comparison after the competition.

Acknowledgements

Professor Xiaoguang Liu
TAs: Songjie Bi, Hind Reggad, Mahmoud Nafe

Appendix

Code to generate Sine wave from Triangle Wave

```
/*
Triangle wave and sync pulse generator to control a (0-5V input range) VCO for FMCW radar.
The MPC4921 DAC is used to generate a triangle wave with a period of 40ms.
PWM of the Arduino UNO is use to simultaneously generate the sync pulse,
used for signal processing.
*/

#include <SPI.h> // Include the SPI library

int indexx = 0;
bool flag = false;

const int slaveSelectPin = 10; //set the slave select (chip select) pin number
const int SYNC = 8;  //set the SYNC output pin number

int outputValue[] = {

};


void setup()
{
   // Set pins for output
   pinMode(SYNC, OUTPUT);                    // SYNC pin
   digitalWrite(SYNC, LOW);              // Sync pulse low
   pinMode(slaveSelectPin, OUTPUT);                    // Slave-select (SS) pin
   SPI.begin();                          // Activate the SPI bus
   SPI.beginTransaction(SPISettings(16000000, MSBFIRST, SPI_MODE0));  // Set up the SPI
transaction; this is not very elegant as there is never a close transaction action.
}

void loop()
{
   if (indexx == 4096){
     indexx = 0;
   }

   if (outputValue[indexx] == 2048 || (flag && outputValue[indexx] == 2045)){
     digitalWrite(SYNC, !digitalRead(SYNC));
   }

   if (outputValue[indexx] == 2045){
     flag = !flag;
   }


   byte HighByte =highByte(outputValue[indexx]);    // Take the upper byte
   HighByte = 0b00001111 & HighByte;       // Shift in the four upper bits (12 bit total)
   HighByte = 0b00010000 | HighByte;       // Keep the Gain at 1 and the Shutdown(active low)
pin off
   byte LowByte = lowByte(outputValue[indexx]);     // Shift in the 8 lower bits

   digitalWrite(slaveSelectPin, LOW);
   SPI.transfer(HighByte);            // Send the upper byte
   SPI.transfer(LowByte);             // Send the lower byte
digitalWrite(slaveSelectPin, HIGH); // Turn off the SPI transmission

   indexx = indexx + 1;
}
```

Python Code

```
# -*- coding: utf-8 -*-
#range radar, reading files from a WAV file
# Originially modified by Meng Wei, a summer exchange student (UCD GREAT Program, 2014)
from Zhejiang University, China, from Greg Charvat's matlab code
# Nov. 17th, 2015, modified by Xiaoguang "Leo" Liu, lxgliu@ucdavis.edu

import wave
import os
```

```
from struct import unpack import numpy as np
from numpy.fft import ifft import matplotlib.pyplot as plt from math import log

#constants
c= 3E8 #(m/s) speed of light
Tp = 20E-3 #(s) pulse duration T/2, single frequency sweep period.
fstart = 2260E6 #(Hz) LFM start frequency
fstop = 2590E6 #(Hz) LFM stop frequency
BW = fstop-fstart #(Hz) transmit bandwidth
trnc_time = 0 #number of seconds to discard at the begining of the wav file

window = False #whether to apply a Hammng window.

# for debugging purposes # log file
#logfile = 'log_new.txt' #logfh = open(logfile,'w') #logfh.write('start \n')

#read the raw data .wave file here
#get path to the .wav file
#filename = os.getcwd() + '\\running_outside_20ms.wav' filename = os.getcwd() +
'\\range_test2.wav' # The initial 1/6 of the above wav file. To save time in developing the code
#open .wav file
wavefile = wave.open(filename, "rb")

# number of channels
nchannels = wavefile.getnchannels()

# number of bits per sample sample_width = wavefile.getsampwidth()

# sampling rate
Fs = wavefile.getframerate()
trnc_smp = int(trnc_time*Fs) # number of samples to discard at the begining of the wav file

# number of samples per pulse
N = int(Tp*Fs) # number of samples per pulse

# number of frames (total samples) numframes = wavefile.getnframes()

# trig stores the sampled SYNC signal in the .wav file #trig = np.zeros([rows,N])
trig = np.zeros([numframes - trnc_smp])
# s stores the sampled radar return signal in the .wav file #s = np.zeros([rows,N])

s = np.zeros([numframes - trnc_smp]) # v stores ifft(s)
#v = np.zeros([rows,N])
```

```
v = np.zeros([numframes - trnc_smp])

#read data from wav file

data = wavefile.readframes(numframes)

for j in range(trnc_smp,numframes): # get the left (SYNC) channel
left = data[4*j:4*j+2]
# get the right (Data) channel right = data[4*j+2:4*j+4]

#.wav file store the sound level information in signed 16-bit integers stored in little-endian format

#The "struct" module provides functions to convert such information to python native formats, in
this case, integers.

if len(left) == 2:
l = unpack('h', left)[0]

if len(right) == 2:
r = unpack('h', right)[0]

#normalize the value to 1 and store them in a two dimensional array "s"

trig[j-trnc_smp] = l/32768.0 s[j-trnc_smp] = r/32768.0

#trigger at the rising edge of the SYNC signal trig[trig < 0] = 0;
trig[trig > 0] = 1;

#2D array for coherent processing s2 = np.zeros([int(len(s)/N),N])

rows = 0;
for j in range(10, len(trig)):

if trig[j] == 1 and np.mean(trig[j-10:j]) == 0: if j+N <= len(trig):

s2[rows,:] = s[j:j+N] rows += 1

s2 = s2[0:rows,:]

#pulse-to-pulse averaging to eliminate system performance drift overtime
for i in range(N):

s2[:,i] = s2[:,i] - np.mean(s2[:,i]) #2pulse cancelation
```

```
s3 = s2
for i in range(0, rows-1):

s3[i,:] = s2[i+1,:] - s2[i,:]

rows = rows-1 s3 = s3[0:rows,:]

#apply a Hamming window to reduce fft sidelobes if window=True
if window == True:

for i in range(rows): s3[i]=np.multiply(s3[i],np.hamming(N))

#####################################
# Range-Time-Intensity (RTI) plot
# inverse FFT. By default the ifft operates on the row v = ifft(s3)

#get magnitude
v = 20*np.log10(np.absolute(v)+1e-12)

#only the first half in each row contains unique information v = v[:,0:int(N/2)]

#normalized with respect to its maximum value so that maximum

is 0dB
m=np.max(v)
grid = v
grid=[[x-m for x in y] for y in v]

# maximum range max_range =c*Fs*Tp/4/BW # maximum time
max_time = Tp*rows

plt.figure(0)
plt.imshow(grid, extent=[0,max_range,0,max_time],aspect='auto', cmap =plt.get_cmap('gray'))
plt.colorbar()
plt.clim(0,-100)
plt.xlabel('Range[m]',{'fontsize':20})
plt.ylabel('time [s]',{'fontsize':20})
plt.title('RTI with 2-pulse clutter rejection',{'fontsize':20}) plt.tight_layout()
plt.show()

#plt.subplot(612) #plt.plot(grid[5])
```

```
#plt.subplot(613) #plt.plot(grid[6])

#plt.subplot(614) #plt.plot(grid[20]) # #plt.subplot(615) #plt.plot(grid[30])

#plt.subplot(616) #plt.plot(grid[40])
```