Final group report

Brian Ratledge, Hui-Ying Siao, John Villahermosa, and Bryant Vu

***Abstract*—The radar system operates at 2.4GHz and uses a frequency modulated continuous wave (FMCW) to determine the range of objects at distance. The waves are generated by the RF circuit and blasted out the transmitter at the metal plate target. The signals are then reflected back towards the receiver and processed to determine the distance of the plate. In the span of a single quarter we went through the process of building a full radar system. This involved selecting suitable components, designing the PCB, assembling the system, and lastly testing out the radar to debug the problems to reach the end goal of the project.**

I. Introduction

With its roots in the radar we built in Quarter 1, our system is redesigned with different components in order to improve upon its efficiency and performance. Within a limited time, we chose components and designed the circuit schematic with the reference of component datasheets. The block diagram of the radar system is shown in Fig.1, the target is a metal plate used on the competition day to reflect the signals transmitting from a Yagi antenna.

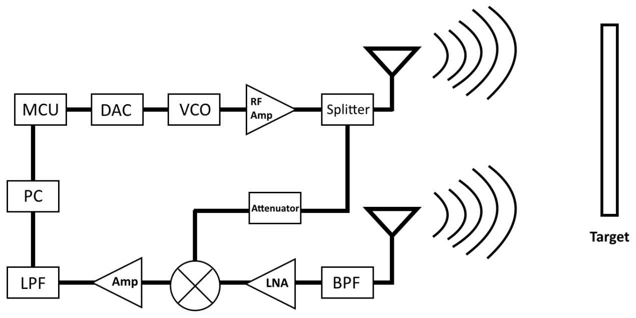
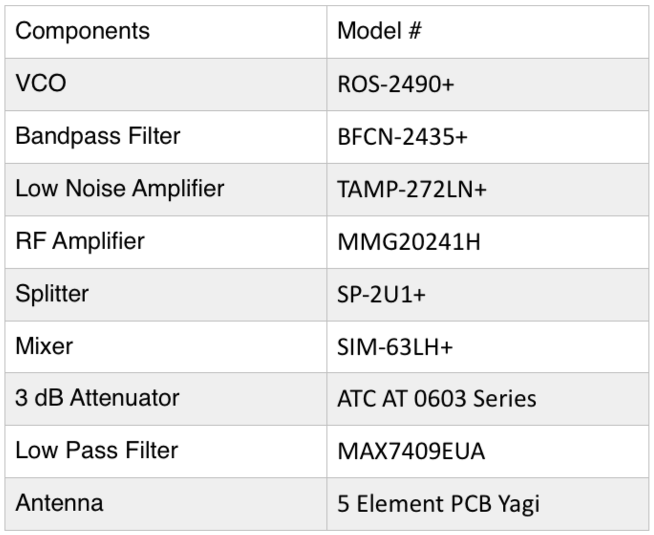


Fig. 1. RF Block Diagram

II. Component selections

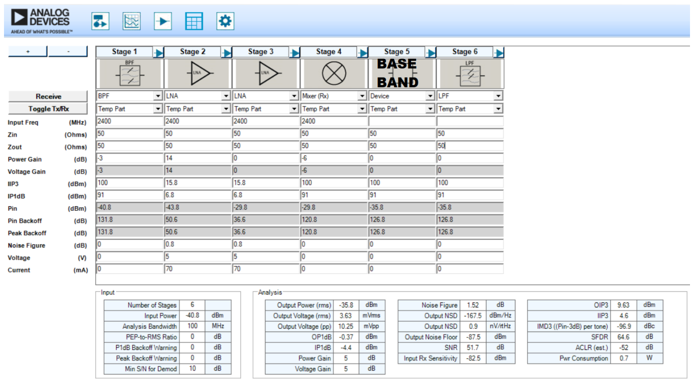
There are devices and components with certain specifications and performance parameters. Since the system level design results in the determination of the components, the component selection is important. The safety range value of all the parameters of all terminals, inputs and outputs should be set at first. The best matched component should be chosen based on its specifications from the datasheet. Table 1 shows the components and its model number that we selected for the final design.

In order to calculate the gain of the transmitting and receiving power, ADIsim was used to simulate the theoretical values. We obtained a theoretical output power of -35.8 dBm. The simulations for the transmitter and receiver are shown in Fig.2 and Fig.3.



Components	Model #
VCO	ROS-2490+
Bandpass Filter	BFCN-2435+
Low Noise Amplifier	TAMP-272LN+
RF Amplifier	MMG20241H
Splitter	SP-2U1+
Mixer	SIM-63LH+
3 dB Attenuator	ATC AT 0603 Series
Low Pass Filter	MAX7409EUA
Antenna	5 Element PCB Yagi

Fig. 2. RF Component List



		Stage 1	Stage 2	Stage 3	Stage 4	Stage 5	Stage 6
		BPF	LNA	LNA	Mixer (Rx)	Device	LPF
		Temp Part	Temp Part	Temp Part	Temp Part	Temp Part	Temp Part
Input Freq	(MHz)	2400	2400	2400	2400		
Zin	(Ohms)	50	50	50	50	50	50
Zout	(Ohms)	50	50	50	50	50	50
Power Gain	(dB)	-3	14	0	-6	0	0
Voltage Gain	(dB)	-3	14	0	-6	0	0
IIP3	(dBm)	100	15.8	15.8	100	100	100
IP1dB	(dBm)	91	6.8	6.8	91	91	91
Pin	(dBm)	-40.8	-43.8	-29.8	-29.8	-35.8	-35.8
Pin Backoff	(dB)	131.8	50.6	36.6	120.8	126.8	126.8
Peak Backoff	(dB)	131.8	50.6	36.6	120.8	126.8	126.8
Noise Figure	(dB)	0	0.8	0.8	0	0	0
Voltage	(V)	0	5	5	0	0	0
Current	(mA)	0	70	70	0	0	0

Input

Number of Stages	6	
Input Power	-40.8	dBm
Analysis Bandwidth	100	MHz
PEP-to-RMS Ratio	0	dB
P1dB Backoff Warning	0	dB
Peak Backoff Warning	0	dB
Min S/N for Demod	10	dB

Analysis

Output Power (rms)	-35.8	dBm
Output Voltage (rms)	3.63	mVrms
Output Voltage (pp)	10.25	mVpp
OP1dB	-0.37	dBm
IP1dB	-4.4	dBm
Power Gain	5	dB
Voltage Gain	5	dB

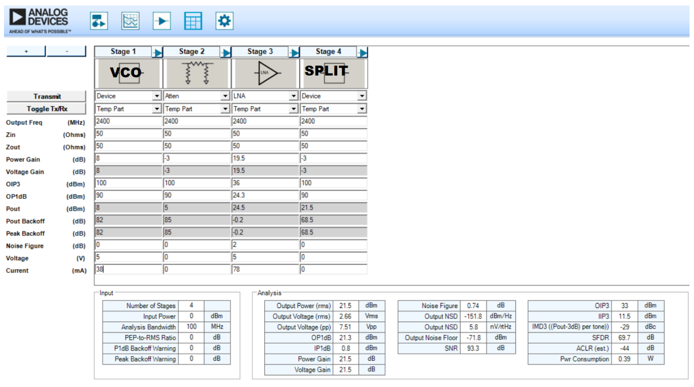
Noise Figure	1.52	dB
Output NSD	-167.5	dBm/Hz
Output NSD	0.9	nV/rtHz
Output Noise Floor	-87.5	dBm
SNR	51.7	dB
Input Rx Sensitivity	-82.5	dBm

OIP3	9.63	dBm
IIP3	4.6	dBm
IMD3 ((Pin-3dB) per tone)	-96.9	dBc
SFDR	64.6	dB
ACLR (est.)	-52	dB
Pwr Consumption	0.7	W

Fig. 3. Receiver Simulation

III. PCB designs

With the selection of the RF components complete, we spent considerable time fine-tuning our PCB layouts until we were satisfied enough to place the fabrication order. The footprints of the PCB designs are based on the datasheets of the components provided on Table1. In Fig.5 and Fig.6 are the RF and baseband schematics and the layouts. For our PCB designs, we used Circuit Maker to create the circuit schematics and PCB board layouts, as shown in Fig.7 and Fig.8. In addition, the layout of passives is also based on the component datasheets. In order to realize the on-board processing, we used the Raspberry Pi 3 to generate the SYNC and Vtune signals as well as process the received signals.



		Stage 1	Stage 2	Stage 3	Stage 4
		Device	Atten	LNA	Device
		Temp Part	Temp Part	Temp Part	Temp Part
Output Freq	(MHz)	2400	2400	2400	2400
Zin	(Ohms)	50	50	50	50
Zout	(Ohms)	50	50	50	50
Power Gain	(dB)	8	-3	19.5	-3
Voltage Gain	(dB)	8	-3	19.5	-3
OIP3	(dBm)	100	100	36	100
OP1dB	(dBm)	90	90	24.3	90
Pout	(dBm)	8	5	24.5	21.5
Pout Backoff	(dB)	82	85	-0.2	68.5
Peak Backoff	(dB)	82	85	-0.2	68.5
Noise Figure	(dB)	0	0	2	0
Voltage	(V)	5	0	5	0
Current	(mA)	38	0	78	0

Input

Number of Stages	4	
Input Power	0	dBm
Analysis Bandwidth	100	MHz
PEP-to-RMS Ratio	0	dB
P1dB Backoff Warning	0	dB
Peak Backoff Warning	0	dB

Analysis

Output Power (rms)	21.5	dBm
Output Voltage (rms)	2.66	Vrms
Output Voltage (pp)	7.51	Vpp
OP1dB	21.3	dBm
IP1dB	0.8	dBm
Power Gain	21.5	dB
Voltage Gain	21.5	dB

Noise Figure	0.74	dB
Output NSD	-151.8	dBm/Hz
Output NSD	5.8	nV/rtHz
Output Noise Floor	-71.8	dBm
SNR	93.3	dB

OIP3	33	dBm
IIP3	11.5	dBm
IMD3 ((Pout-3dB) per tone))	-29	dBc
SFDR	69.7	dB
ACLR (est.)	-44	dB
Pwr Consumption	0.39	W

Fig. 4. Transmitter Simulation

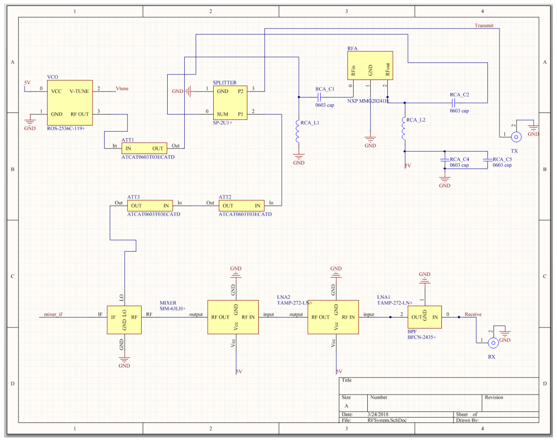


Fig. 5. RF Schematic

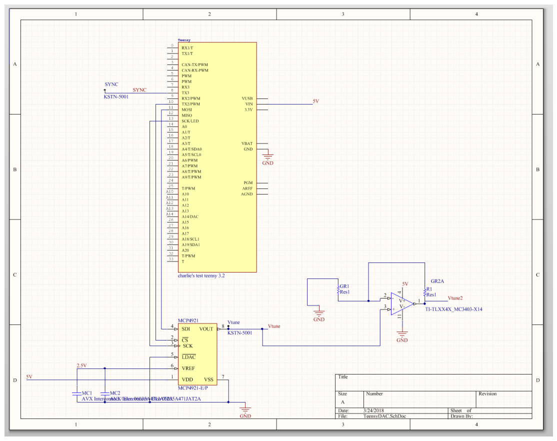


Fig. 6. Baseband Schematic.

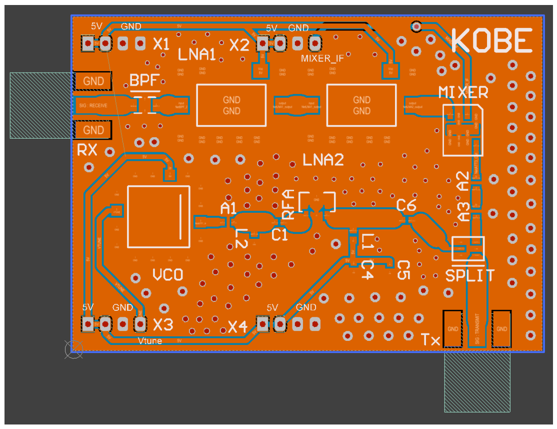


Fig. 7. Baseband layout.

IV. Antennas

The transmitting and receiving antennas play major roles in the radar system. In order to obtain high gain signals, the Yagi PCB antenna, as shown in Fig.10, was chosen for the radar system with a max signal of 10-11dBi from 2400 to 2450 MHz.

V. Testing results

In order to test the system and to make sure it would work properly, we tested our system in front of Kemper Hall before the competition day. On the day of the test we compared the difference between the Yagi antennas and the coffee cans we used as antennas in quarter one, the results are shown in Fig.11 and Fig.12. Analysis of the two results from coffee cans and Yagi antennas reveal that the signals from the Yagi antenna were not showing as obvious of an improvement as we expected. The reason for this was that we were directly holding the antennas, when ideally the antenna should be mounted to a frame since holding the antenna can affect the performance. Similar to the wood board of the Quarter 1 system, we mounted the Yagi antennas to a length of foam.

In order to optimize the signals and avoid interference from nearby structures, the testing location was at the Hutchison field to the south of the ARC Pavilion, as shown in Fig. 13. We were using on-board processing to obtain the data, as shown in Fig. 14, and the code for each of these parts can be seen in the Appendix of this report.

On the competition day, we did not get ideal results. The results we got were extremely noisy, and we could not differentiate the distance accurately. The reason for this was mainly due to the low resolution of the signals generated by Raspberry pi. After fixing the resolution problem of the Raspberry pi, we went to Hutchison and retested the results. However, the results, as shown in Fig. 15, did not present a result as we expected.

Realizing that we could not program Raspberry pi to optimize the signals in a limited time, we went back to using the Teensy instead and the result is shown in Fig. 16 and Fig. 17 show the results using coffee cans as quarter one and Fig. shows the one using Yagi antennas. The distance from the target and the antennas were about 7.3, 15.2, 21.3, 30.5, and 45.7 meters, which correspond well to our record. The longest distance of the testing was about 90 meters, as shown in Fig. 18.

VI. Conclusion

In this report, we present the components selections, PCB designs, testing and debugging process and the results of our radar systems. As the testing results show, we are able to get clear and good signals as well as on-board signal processing.

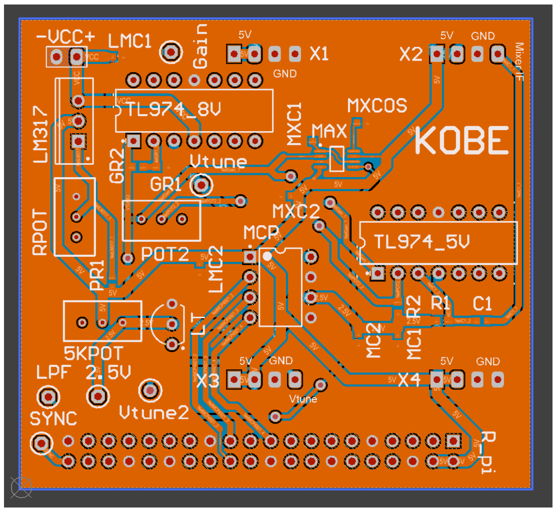


Fig. 8. RF layout.

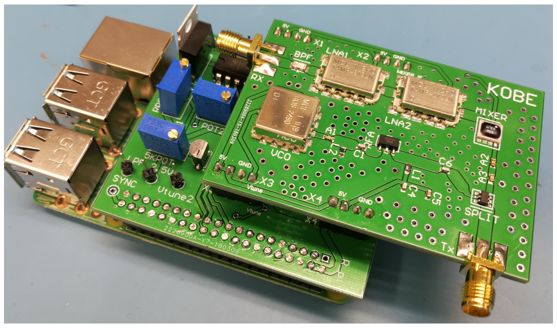


Fig. 9. Assembled PCBs

The largest distance we were able to achieve our system and the target was about 90 meters using the Teensy to generate the signals. Building a radar system during the two quarters includes the following concerns: budget control, component selection, PCB design and assembly, system testing, proper micro-controller programming, and a lot of teamwork.

Acknowledgment

The authors would like to thank Prof. Xiaoguang Liu, Songjie Bi, Mahmoud Nafe and Hind Reggad, for their instruction and guidance.

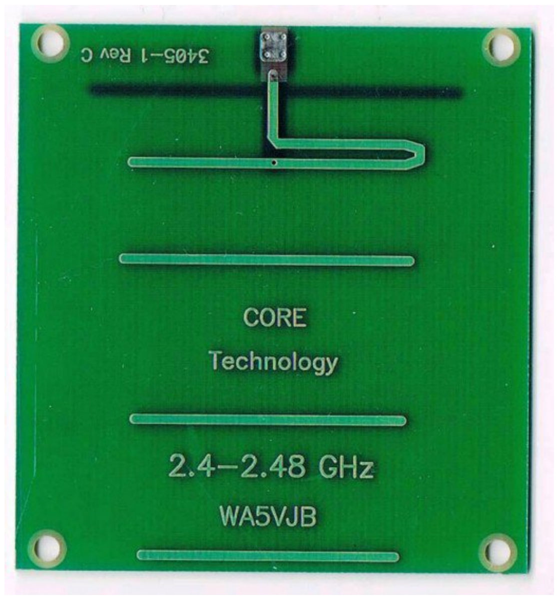


Fig. 10. A Yagi antenna from Kent Electronics at wa5vjb.com

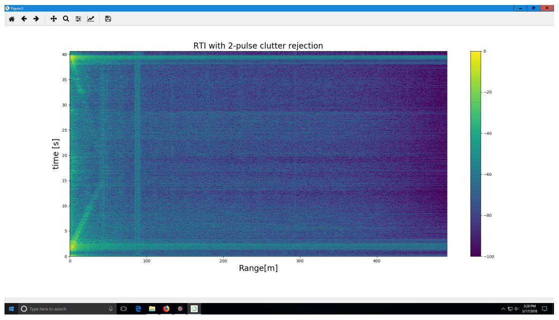


Fig. 11. Results of using coffee can antennas tested at Kemper

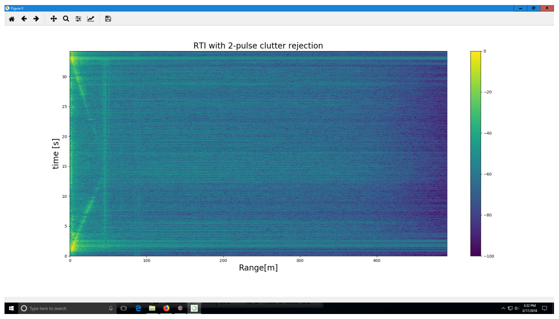


Fig. 12. Results of using Yagi antennas tested at Kemper

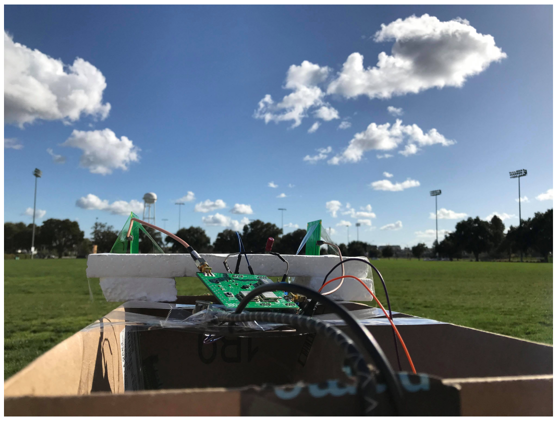


Fig. 13. Testing setup on the Hutchison field.

Fig. 14. On board testing using raspberry pi

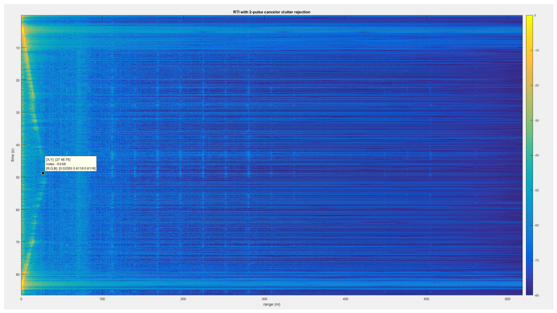


Fig. 15. On board testing using raspberry pi

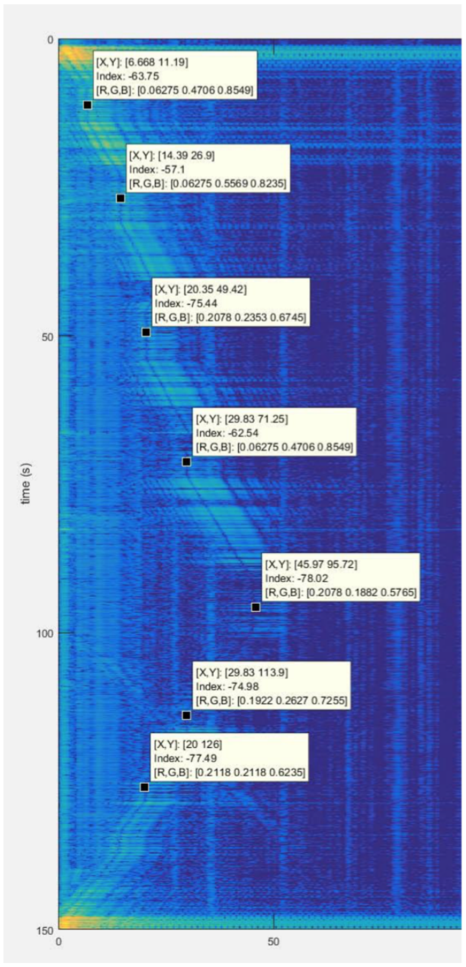


Fig. 16. Testing results of the radar system using Yagi antennas

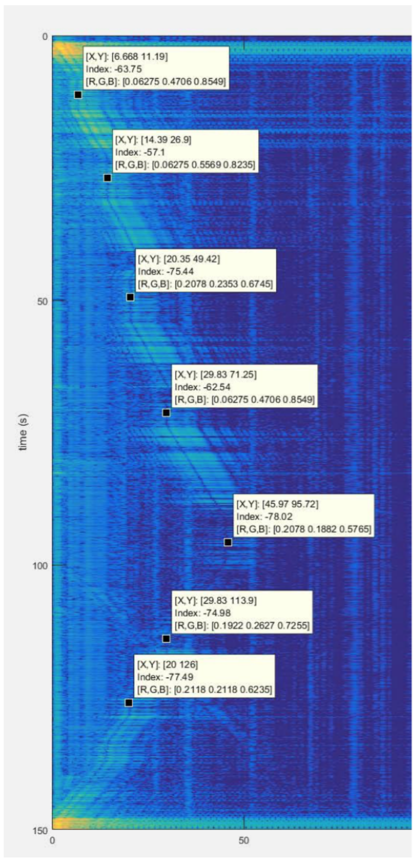


Fig. 17. Testing results of the radar system using coffee can

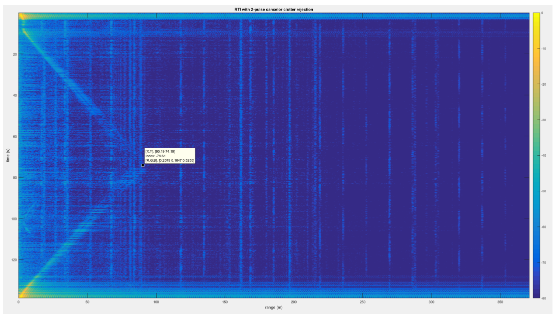


Fig. 18. Maximum distance of the testing results of the radar system using Yagi antennas

Appendix

-*- 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 = 2350E6 (Hz) LFM start frequency fstop = 2470E6 (Hz) LFM stop frequency BW = fstop-fstart (Hz) transmit bandwidth trnc$_{t}ime = 0numberofsecondstodiscardatthebeginingofthewavfile$

window = False whether to apply a Hammng window.

for debugging purposes log file logfile = 'log$_{n}ew.txt'logfh = open(logfile,' w')logfh.write('start')$

read the raw data .wave file here get path to the .wav file filename = os.getcwd() + 'runningoutside20ms.wav' filename = os.getcwd() + '/test.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$_{w}idth = wavefile.getsampwidth()$

sampling rate Fs = wavefile.getframerate() trncsmp = int(trnctime*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 - trncsmp]) s stores the sampled radar return signal in the .wav file s = np.zeros([rows,N]) s = np.zeros([numframes - trncsmp]) v stores ifft(s) v = np.zeros([rows,N]) v = np.zeros([numframes - trncsmp])

read data from wav file

data = wavefile.readframes(numframes)

for j in range(trncsmp,numframes): get the left (SYNC) channe 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$_{s}mp] = l/32768.0s[j - trnc_{s}mp] = r/32768.0$

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 maxrange =c*Fs*Tp/4/BW maximum time max$_{t}ime = Tp * rows$

plt.figure(0) plt.imshow(grid, extent=[0,max$_{r}ange, 0, max_{t}ime], aspect =' auto', cmap = plt.getcmap('gray'))plt.imshow(grid, extent = [0, maxrange, 0, maxtime], aspect =' auto')plt.colorbar()plt.clim(0, -100)plt.xlabel('Range[m]', 'fontsize'$ $pulseclutterrejection', 'fontsize' : 20)plt.tight_{l}ayout()plt.show()$

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])

```
!/usr/bin/python
#-------------------------------------
-----------------------------------------
# Name:        MCP4911_sawtooth.py
# Purpose:     Output a sawtooth waveform
#
# Author:      paulv
#
# Created:     18-09-2015
# Copyright:   (c) paulv 2015
# Licence:     <your licence>
#-----------------------------------------
-------------------------------------

import spidev
from time import sleep

DEBUG = False
spi_max_speed = 4 * 100000 # 4 MHz
V_Ref = 2500 # 3V3 in mV
Resolution = 2**12 # 10 bits for the MCP 4911
CE = 0 # CE0 or CE1, select SPI device on bus

# setup and open an SPI channel
spi = spidev.SpiDev()
spi.open(0,CE)
spi.max_speed_hz = spi_max_speed


def setOutput(val):
# lowbyte has 8 data bits
# B7, B6, B5, B4, B3, B2, B1, B0
# D7, D6, D5, D4, D3, D2, D1, D0
lowByte = val & 0b11111111
# highbyte has control and 4 data bits
# control bits are:
# B7, B6,   B5, B4,     B3,  B2,  B1,  B0
# W  ,BUF, !GA, !SHDN,  D9,  D8,  D7,  D6
# B7=0:write to DAC, B6=0:unbuffered,
B5=1:Gain=1X, B4=1:Output is active
highByte = ((val >> 6) & 0xff)
| 0b0 << 7 |
0b0 << 6 | 0b1 << 5 | 0b1 << 4
 #
# by using spi.xfer2(), the CS
is released after
each block, transferring the
# value to the output pin.
if DEBUG :
print("Highbyte = {0:8b}".
format(highByte))
print("Lowbyte =  {0:8b}".
format(lowByte))
spi.xfer2([highByte, lowByte])


try:
    while(True):
```

```
# create a sawtooth ramp starting from 0 to V-ref
for step in range(1024):
if DEBUG :
print("Step = {0}".format(step))
print("Output level should be :
{0} mV".format(step * V_Ref / Resolution))
setOutput(step)
for step in range(1024,0,-1):
if DEBUG :
print("Step = {0}".format(step))
print("Output level should be :
{0} mV".format(step * V_Ref / Resolution))
setOutput(step)

except (KeyboardInterrupt, Exception) as e:
print(e)
print("Closing SPI channel")
spi.close()

def main():
  pass

if __name__ == '__main__':
  main()

%MIT IAP Radar Course 20112.5
%Resource: Build a Small Radar System
Capable of Sensing Range, Doppler,
%and Synthetic Aperture Radar Imaging
%
%Gregory L. Charvat

%Process Range vs. Time Intensity (RTI) plot

clear all;
close all;

% read the raw data .wav file here
% replace with your own .wav file
[Y,FS,NBITS] = wavread('testtest5.wav');

%constants
c = 3E8; %(m/s) speed of light

%radar parameters
Tp = 20E-3; %(s) pulse time
N = Tp*FS; %# of samples per pulse
fstart = 2325E6; %(Hz) LFM start frequency
fstop = 2427E6; %(Hz) LFM stop frequency
BW = fstop-fstart; %(Hz) transmti bandwidth
f = linspace(fstart, fstop, N/2);
%instantaneous
transmit frequency

%range resolution
rr = c/(2*BW);
max_range = rr*N/2;
```

```
%the input appears to be inverted
trig = -1*Y(:,1);
s = -1*Y(:,2);
clear Y;

%parse the data here by triggering off rising edge of sync pulse
count = 0;
thresh = 0;
start = (trig > thresh);
for ii = 100:(size(start,1)-N)
if start(ii) == 1 & mean(start(ii-11:ii-1)) == 0
%start2(ii) = 1;
count = count + 1;
sif(count,:) = s(ii:ii+N-1);
time(count) = ii*1/FS;
end
end
%check to see if triggering works
% plot(trig,'.b');
% hold on;si
% plot(start2,'.r');
% hold off;
% grid on;

%subtract the average
ave = mean(sif,1);
for ii = 1:size(sif,1);
    sif(ii,:) = sif(ii,:) - ave;
end

zpad = 8*N/2;

%RTI plot
figure(10);
v = dbv(ifft(sif,zpad,2));
S = v(:,1:size(v,2)/2);
m = max(max(v));
imagesc(linspace(0,max_range,zpad),time,S-m,[-80, 0]);
colorbar;
ylabel('time (s)');
xlabel('range (m)');
title('RTI without clutter rejection');

%2 pulse cancelor RTI plot
figure(20);
sif2 = sif(2:size(sif,1),:)-sif(1:size(sif,1)-1,:);
v = ifft(sif2,zpad,2);
S=v;
R = linspace(0,max_range,zpad);
for ii = 1:size(S,1)
    %S(ii,:) = S(ii,:).*R.^(3/2); %Optional: magnitude scale to range
end
S = dbv(S(:,1:size(v,2)/2));
m = max(max(S));
imagesc(R,time,S-m,[-80, 0]);
colorbar;
ylabel('time (s)');
xlabel('range (m)');
title('RTI with 2-pulse cancelor clutter
rejection');
```