Final Group Report

EEC 134: RF System

Team AAA

Chenxi Liu
Dongsheng Xue
Du Chen
Tracy He

March 27th, 2018

Table Of Content

Abstract

In fall quarter, a 2.4GHz Frequency Modulated Continuous Wave (FMCW) radar system was built using breadboard and given RF components to perform range and Doppler measurements. To improved the accuracy, and minimize power consumption and overall weight, we implemented the major part of the system with PCBs and lightweight antenna.

Overall Design

In the winter quarter, we would design a new radar system that based on the principles provided last quarter and focus on improving the accuracy and simplicity of our system. We choose accuracy as our top priority during the design and simplicity to be the second. We chose the most specific components through multiple choices for our system based on accurate calculation to insure it has satisfied accuracy, weight and power consumption. As the back up plan, we kept our quarter one system just in case any emergent malfunction of our design. We started with the calculation of power consumption of each component and came up with following block diagram.

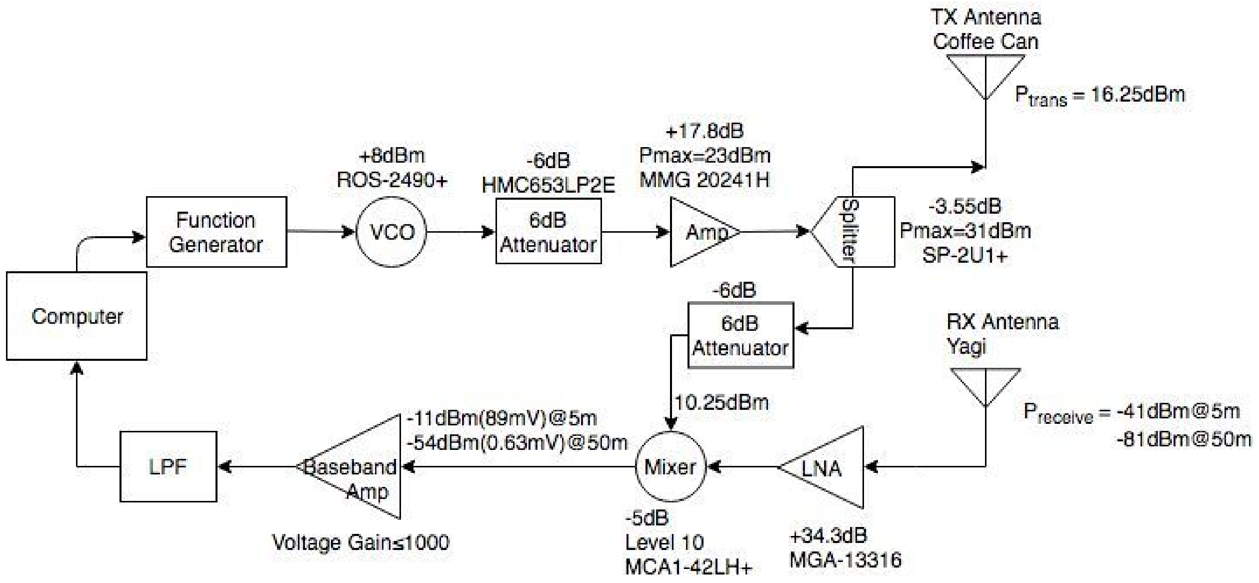

Fig 1. Block Diagram

After we finalized our design and placed our order, we were told that the VCO ROS-2536C-1194 was not available and we have done designing our RF PCB, so the only alternative was ROS-2490+. However, ROS 2490+ requires a tuning voltage of 5.5 V to have a 2.4 GHz frequency and the max output of the teensy triangle wave is 5 V. In order to raise the tuning voltage, we designed a voltage divider and connected it between the Teensy output and the VCO input, so we finally were able to get the teensy triangle wave from 1.8V to 6.7V, which covers the tuning range of the VCO for 2.4 GHz.

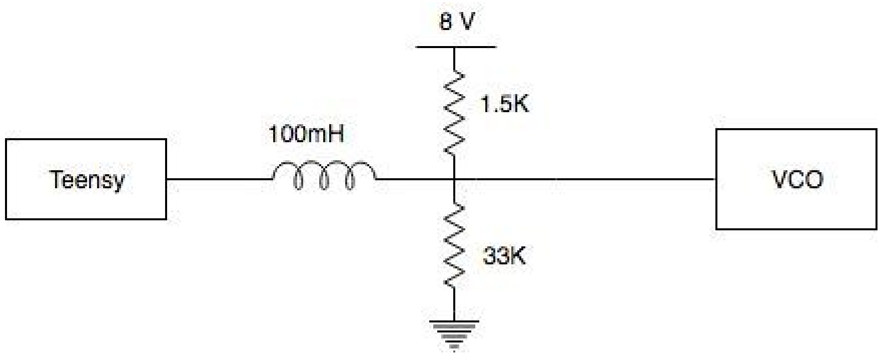

Fig 2. Voltage Divider

Table 1. Component List

Components	**Model#**
VCO	**ROS-2490+**
Power Amplifier	**MMG20241HT1CT-ND**
Low Noise Amplifier	**MGA-13316**
Splitter	**Mini SP-2U1+**
Mixer	**MCA1-42LH+**
Attenuator	**HMC653LP2E**
Antenna	**2.4GHz Yagi PCB antenna**
Gain Stage and Low Pass Filter	**OPA227PA-ND/OPA2227PA-ND**

Table 1 shows the key components we chose based on the power calculation in the block diagram. Other electronic components like resistors, conductors and inductors for our RF and baseband PCB. We purchased three for each component just in case any burning or malfunction during the soldering. The budget is also controlled under $300. All the components we bought are SMD components that would be soldered on our PCBs in order to reduce the weight of the whole system.

PCB Design

After finishing the system block diagram and selecting the components, we began the PCB design on KiCad. We took a look at some designs of groups from previous terms, and we were inspired by Team RadioFreqs from year 2016-2017. Our system consists of two PCBs: baseband and RF. We use pin headers to stack two boards together for mechanical support. In design process, we carefully align the position of each pin-header connector by utilizing the "grid" in KiCad to make sure the successful stacking. Schematics and layouts of the baseband and the RF are shown below.

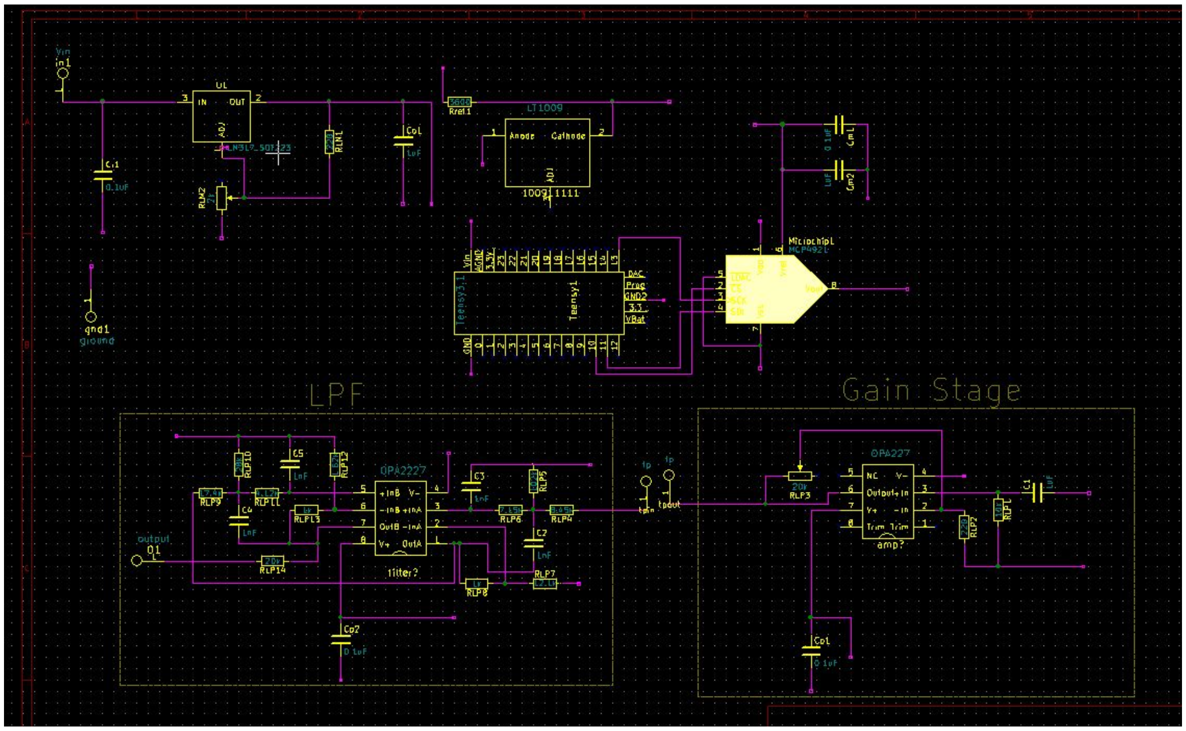

Fig.3 Baseband schematic

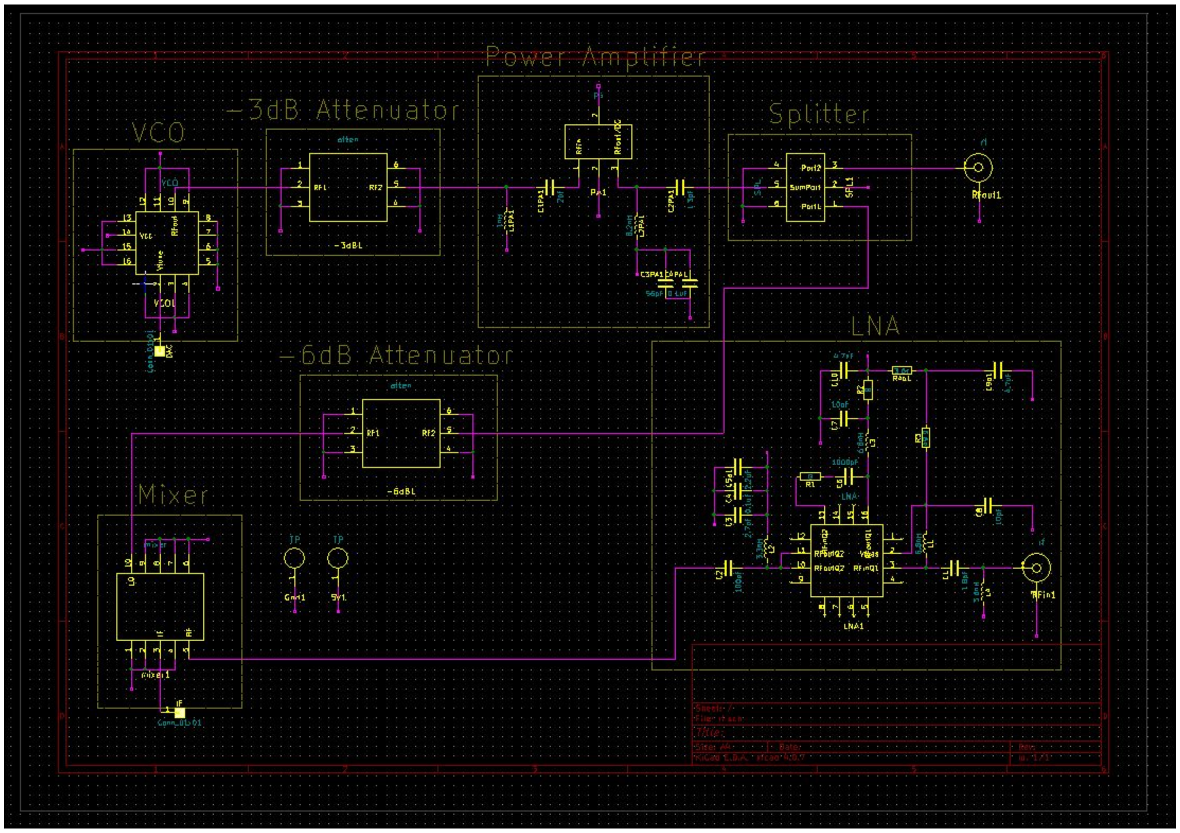

Fig.4 RF schematic

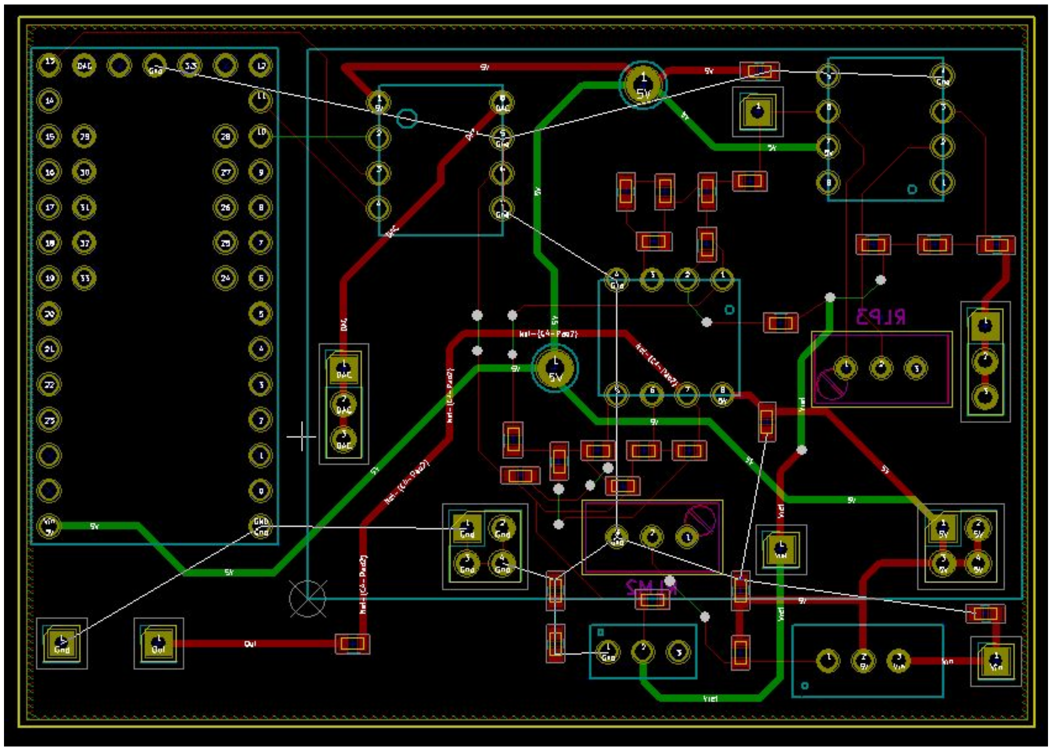

Fig.5 Baseband layout

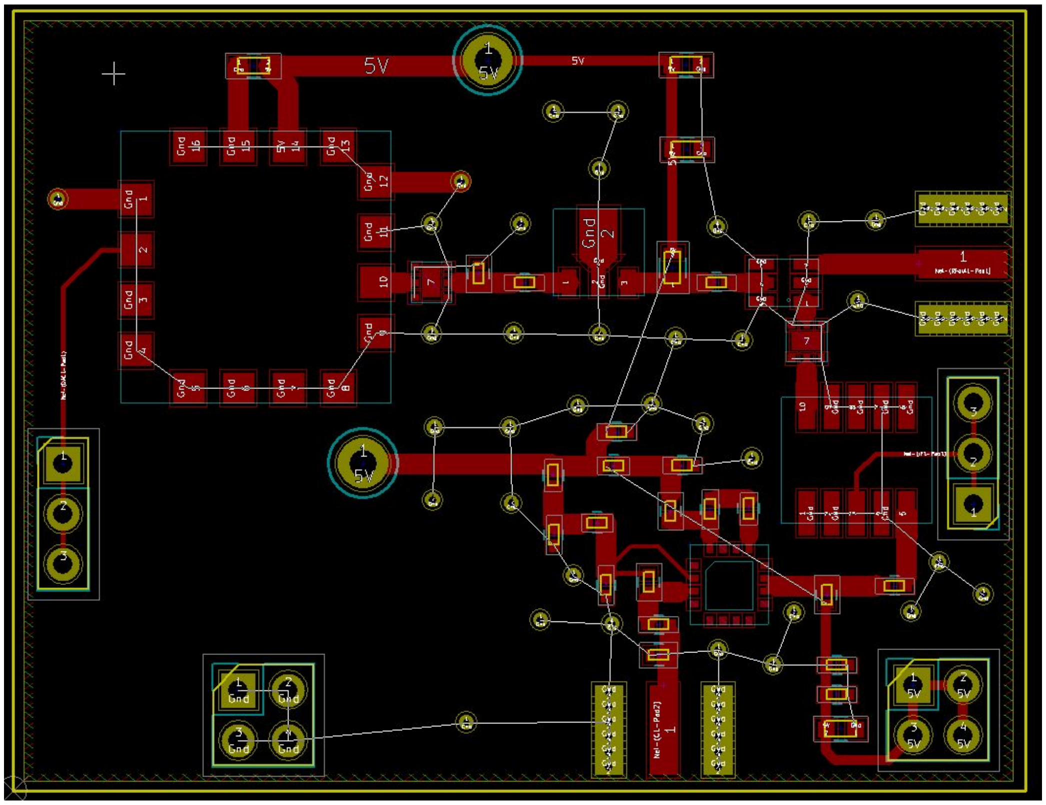
Fig.6 RF layout

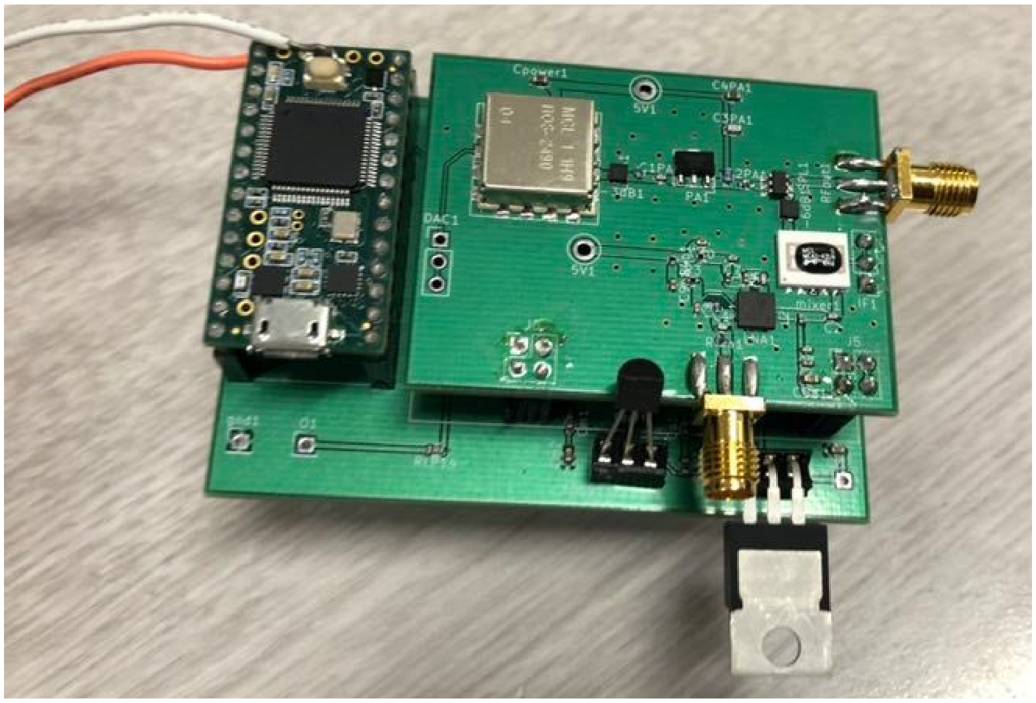
Fig.7 stacked-up system (front view)

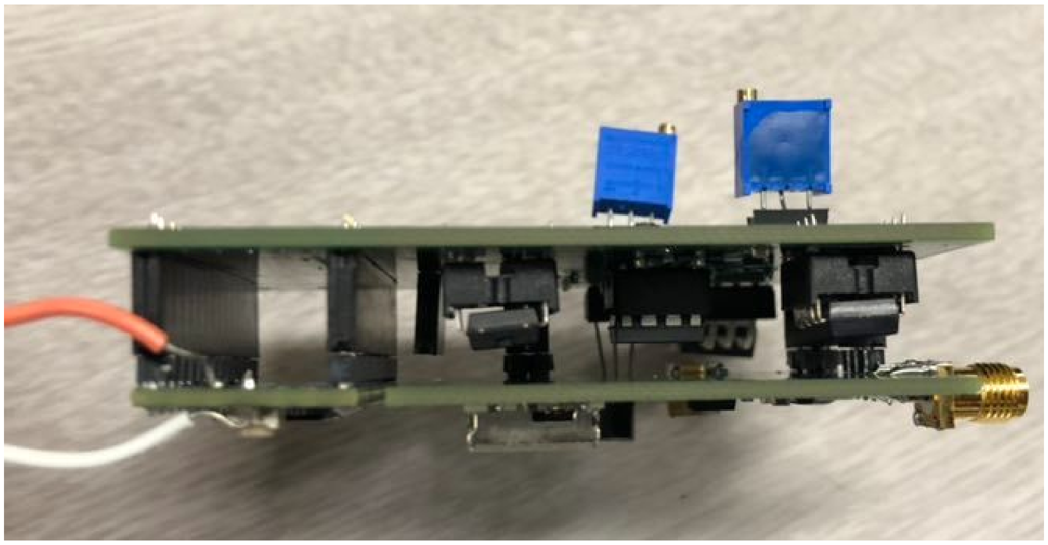
Fig.8 stacked-up system (side view)

Design Testing

After soldering both Baseband and RF PCBs, we first performed various tests to ensure that each individual PCB is working as expected. Then, we assembly the whole system to do further testing.

<u>Baseband PCB</u>

1. Connect teensy to the computer using a USB cable, and program it with the triangle wave code. Check the waveform at the output of the DAC (which is MCP4921 in our case).

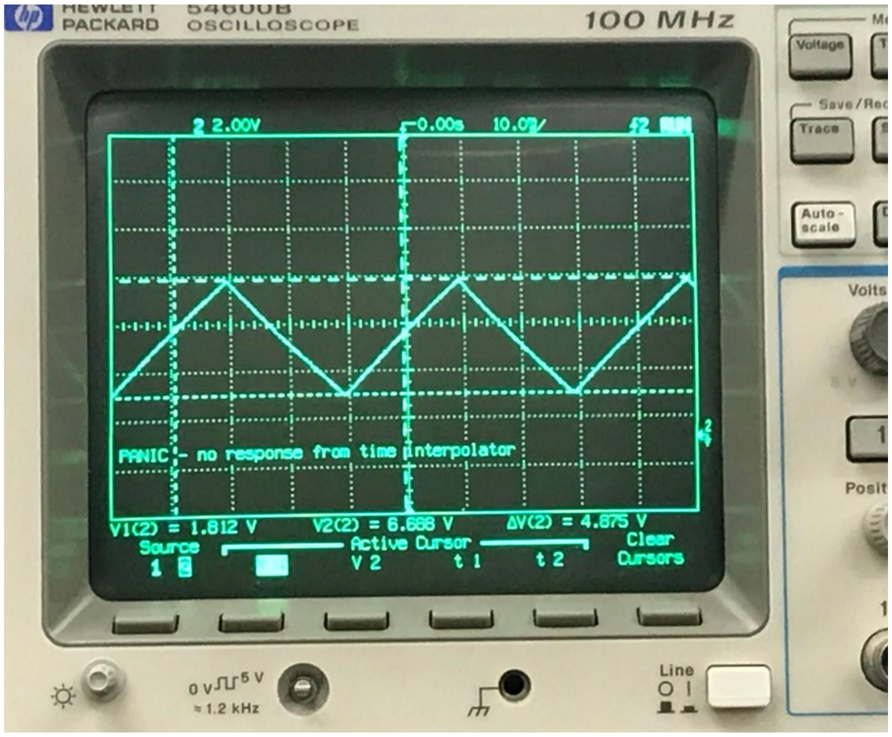

Fig.9 Output of Function Generator

2. Input a signal to the LPF using a function generation, and observed the output of the gain stage. Adjust the potentiometer until it meet the deserved gain. Also, we test the cutoff frequency of the LPF, which is about 20kHz.

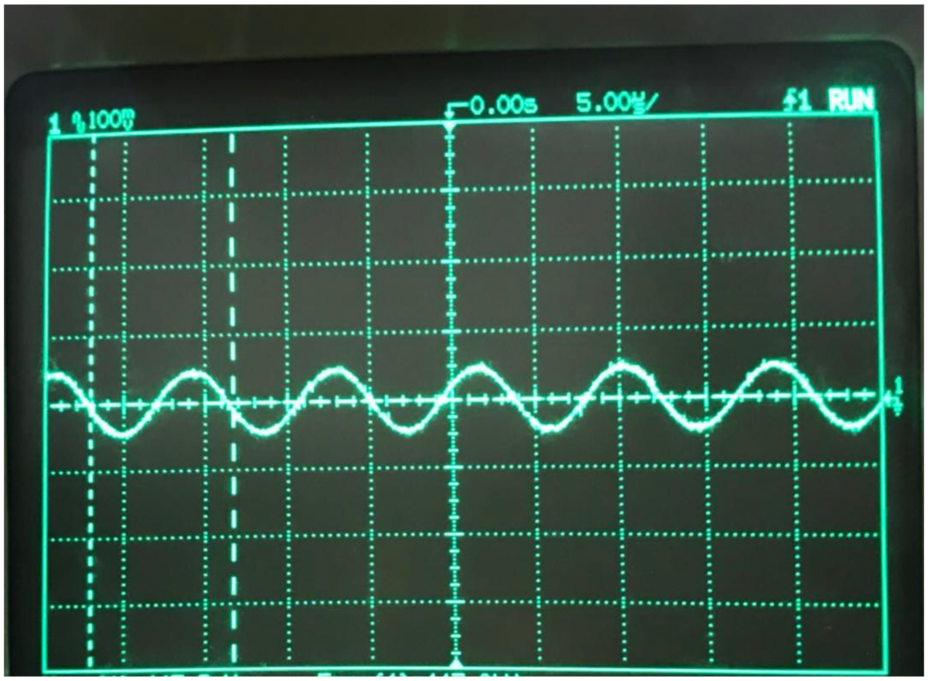

Figure 10. Output of the gain stage

<u>RF PCB</u>

1. Check the transmitting side of the RF board. According to our design, we expect to see a 16.25dBm signal at the transmitting antenna. We observe a 14.7dBm signal, and believe this is pretty good because there are some reading error in the spectrum analyzer.

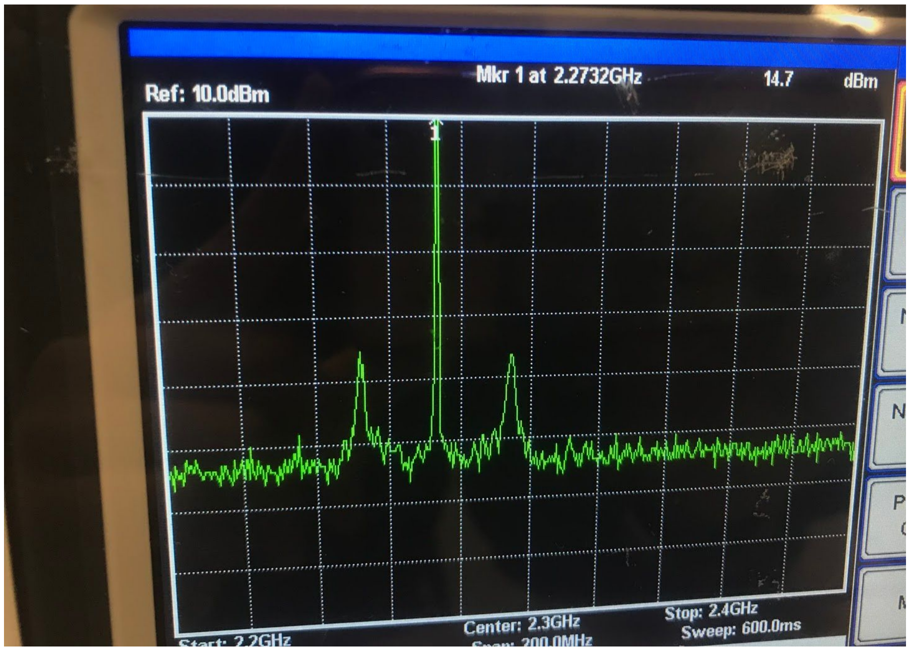

Fig. 11 Transmitting power at start frequency

2. Check the Receiving side of the of RF board.
- After confirming that the transmitting side has enough power to power the mixer, we then use a probe make sure there is signal going into the mixer.
- Compare the signal coming in and out of the LNA, to ensure the LNA is working properly.
- Finally, check the signal at the mixer output. However, there is no signal except some noise in our system. So, we come to conclusion that our mixer is not working. To fix this error, we replace the mixer with another new mixer (same model). However, we encounter the same issue again, with no signal coming out of the mixer. We believe the mixers have defects.

Overall System Testing

When we found out that the mixer is not working, there was no time for us to order a new mixer and redesign our PCB. Therefore, we decided to continue use the transmitting side of the RF PCB, but on the receiving side, we used the system from Quarter one.

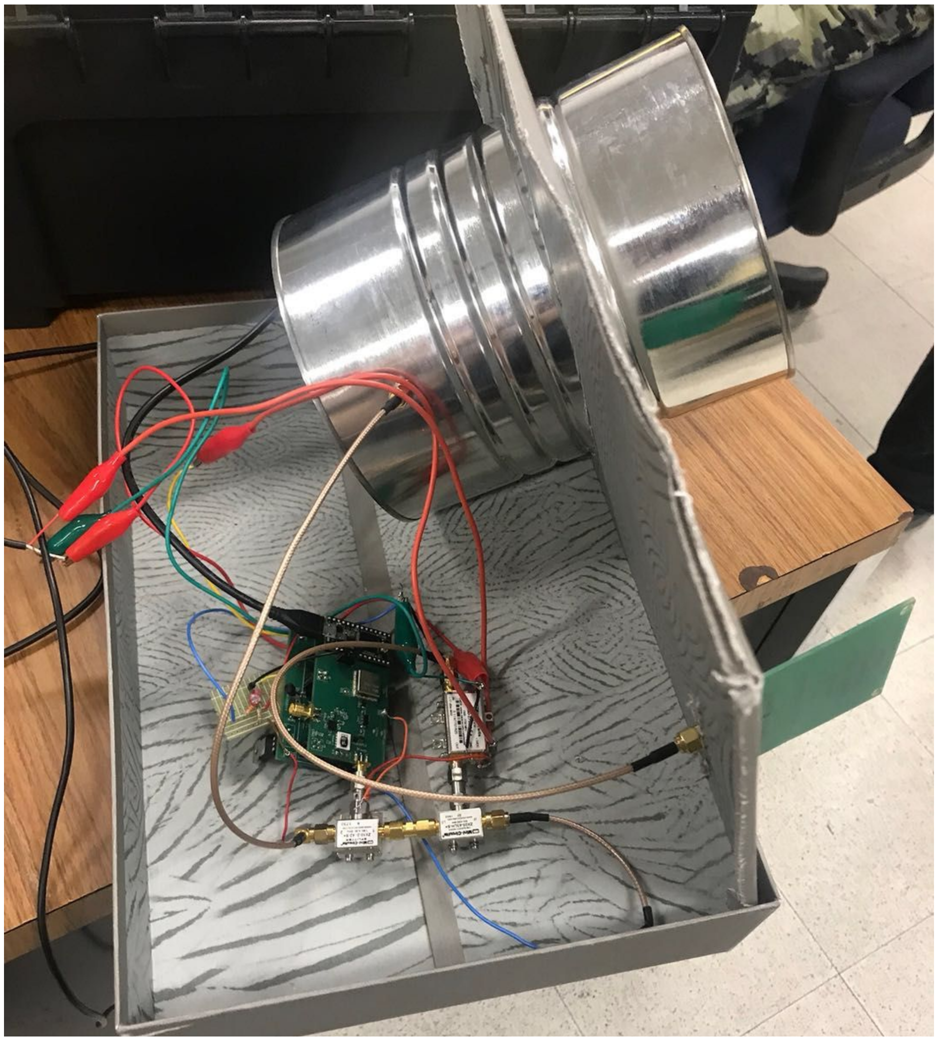
Fig.12 Assembled system

We tried different combination of attenuate, such as using both coffee can for transmitting and receiving, using both yagi antennas, and using one coffee can and one yagi. We found that using the coffee can for transmitting signal and yagi antenna for receiving signal give the us the most clear result and furthest measurement range. According to our design when the object is 50 meter away if we want the output signal to be 1 V, which will give us the clearest resolution, we need to amplified our signal with a gain of 1000, which means we need to adjust our pot to about 200K ohms. However, during our testing, we found that a 20K ohm (gain of 200) is sufficient for 50 meter range measurement.

First Field Test:

In the figure below, we can tell that the range goes up to about 70m, but we believe if we go further the system will still be able to detect it. However, due to the fact that the test environment didn't allow us to go further so we didn't continue testing.

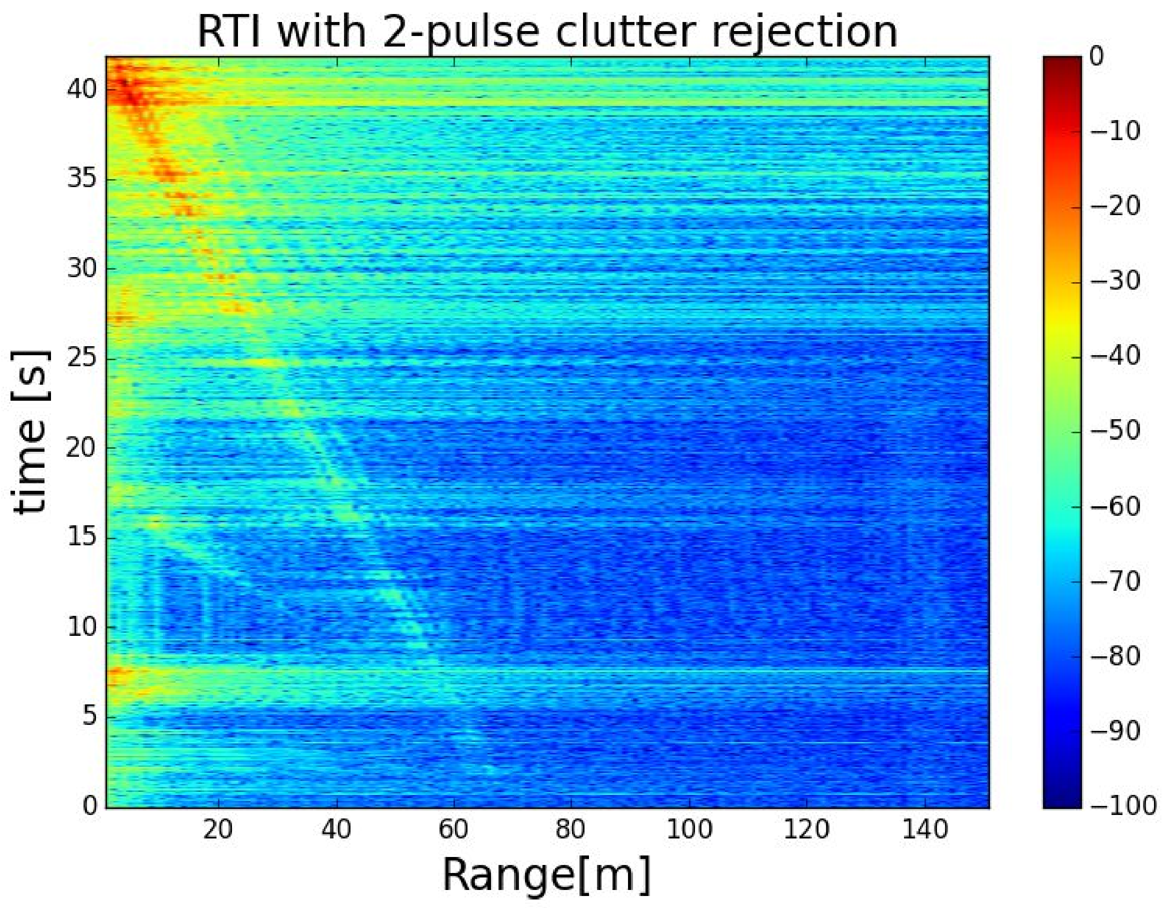

Fig.13 First filed test result

Result

On Completion Day

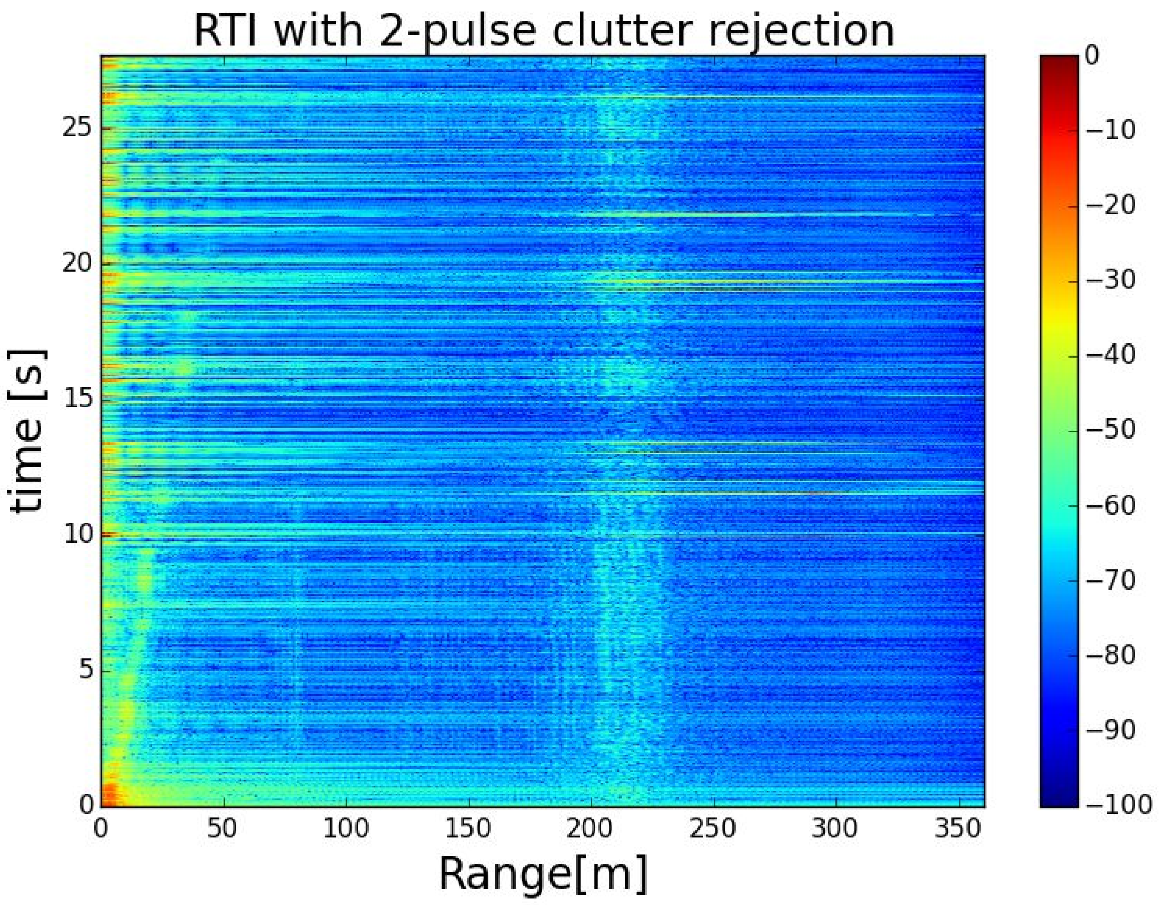

Fig.14 Competition trial 1

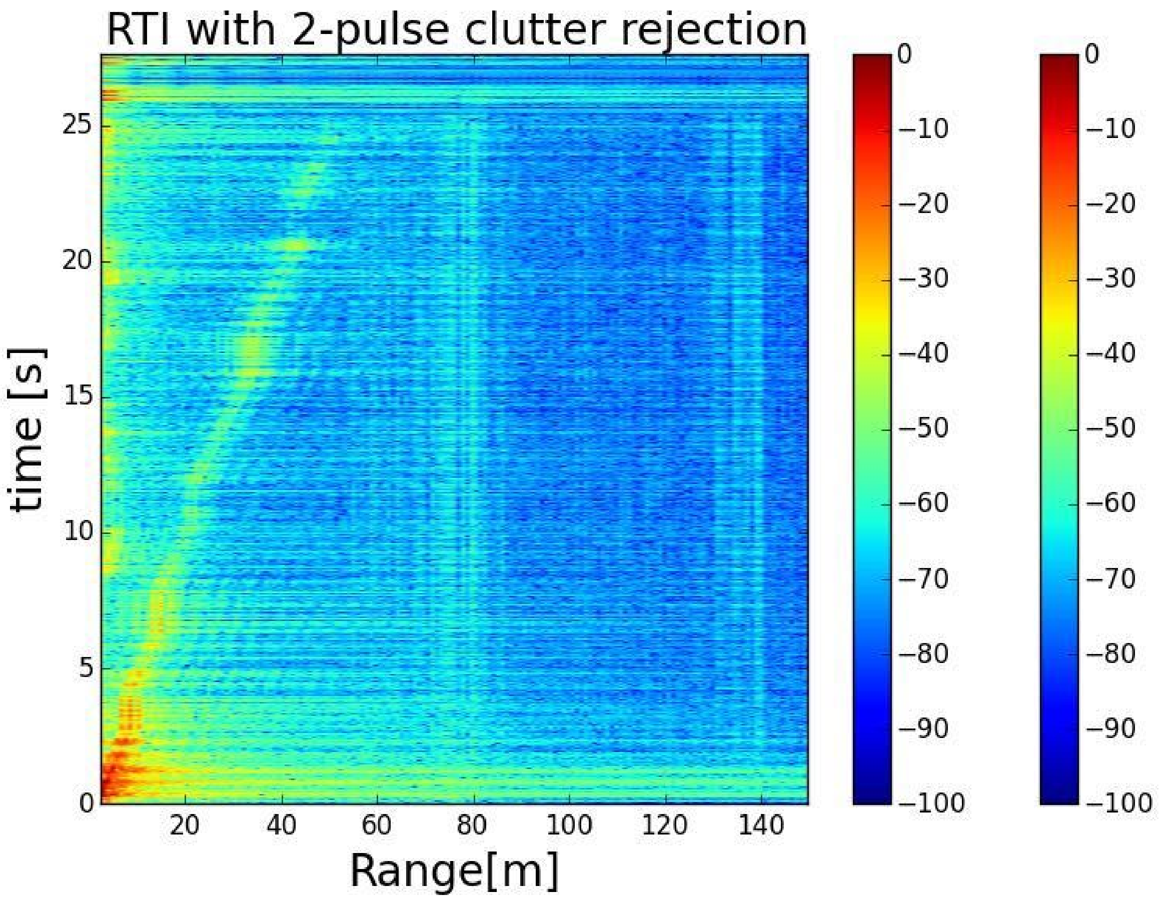

Fig.15 Competition Trial 2

The whole system weighs 600g and the power consumption is 1.92W. It was raining during the first trial and we believed that the rain caused a lot of noise and made our result very ambiguous. We waited until the rain stopped and did the second trial and we got a perfect diagram that clearly showed the 5 stops made during the test. However, all the five results we reported were 2 meters smaller than the actual value. We thought the problem was that we did not test the precision of our system and calibrate out the constant error.

Conclusion

This is a practical and comprehensive project that involves designing, debugging, and dealing with unpredictable real-life challenges. From the out-of-stock VCO to the defect mixer, we experienced many possible challenges we may face in actual production. Fortunately, we successfully solved these problems and had a fully worked radar system. However, the competition result could be better if we tested the precision of our radar system and calibrate out its inherent error. Moreover, the radar system could be more functional if we had on-board processing to see the real-time displacement.

Acknowledgements
Professor Xiaoguang “Leo” Liu
TAs: Songjie Bi, Mahmoud Nafe, Hind Reggad
Team RadioFreqs (2016-2017)

Appendix A: Teensy Code

```
/*

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

word outputValue = 4;// A word is a 16-bit number

int incr = 1;

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

const int SYNC = 8;  //set the SYNC output pin number



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 (outputValue == 4092 || outputValue == 0){

     incr = -incr;

     digitalWrite(SYNC, !digitalRead(SYNC));

   }



   outputValue = outputValue + incr;

   byte HighByte =highByte(outputValue);    // 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);     // 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

}
```

Appendix B: Python Code

```
#
-*-
cod
ing:
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 = 19E-3  #(s) pulse duration T/2, single
frequency sweep period of our system.
fstart = 2273E6 #(Hz) LFM start frequency of
our system
fstop = 2438E6 #(Hz) LFM stop frequency of
our system
#Remember to change the Tp, fstart, and fstop
value according to your system.

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.imshow(grid,
extent=[0,max_range,0,max_time],
aspect='auto')
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])
```