Scott Richards
EEC 134 Team "TEAM"
#999135303

App Note: Signal Processing and Code Implementation

Introduction:

My main contribution to our design project was writing and implementing the code used in our system's signal generating and processing. In addition, I was the group's purchaser, and helped with testing.

Signal Generator:

For our system, we used the *Teensy 3.1* micro-controller and the MCP4921 12-bit DAC microchip.

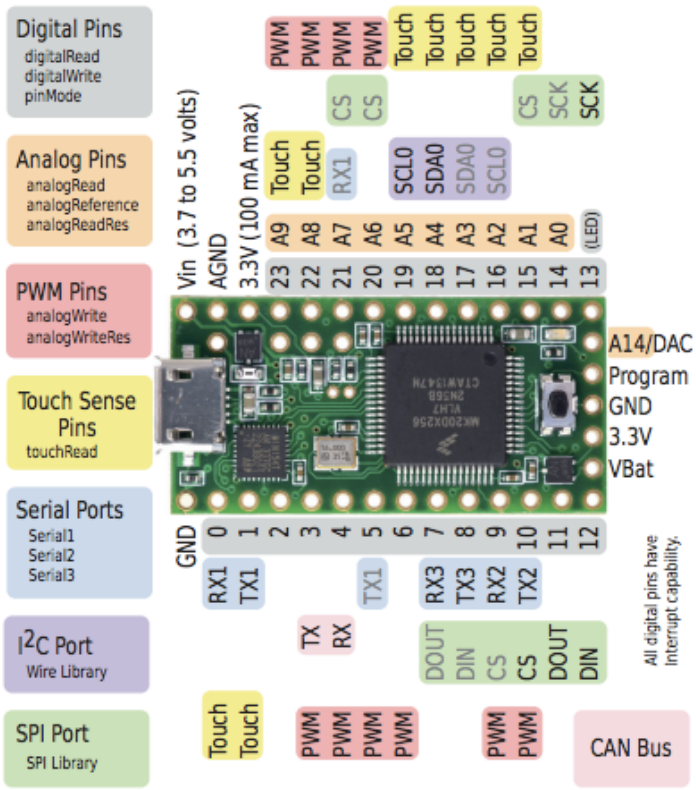
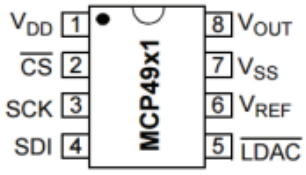


Figure 1 Teensy 3.1 Pin Layout



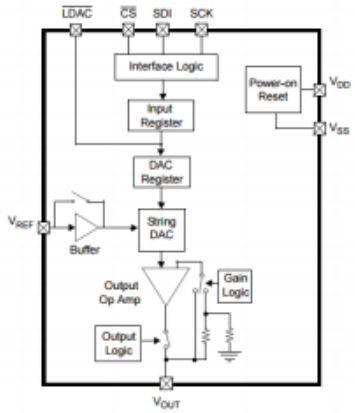


Figure 2 MCP Pin Layout

The *Teensy 3.1* is a micro-controller that we programmed using the Arduino IDE. The Teensy's output was then connected to our designated DAC, the MCP4921, in order to produce our desired triangle wave output.

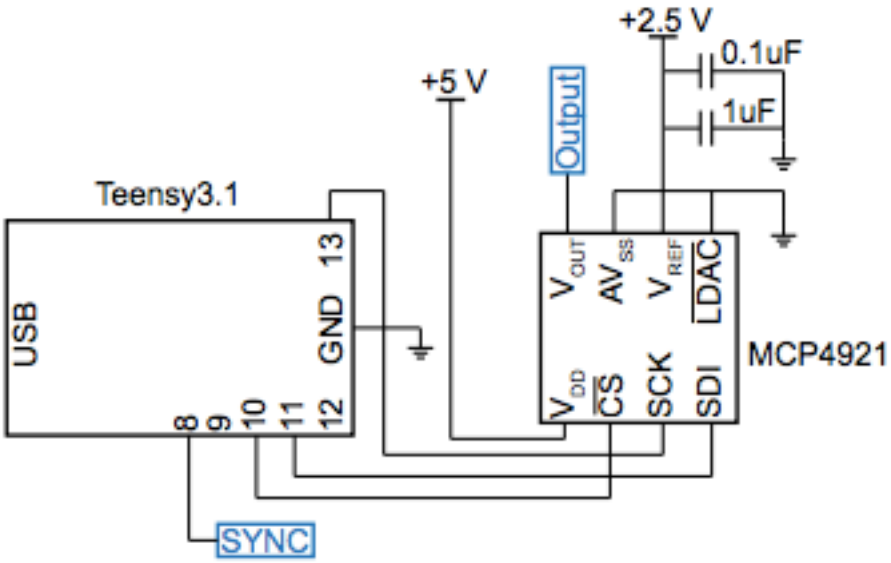


Figure 3 Teensy-MCP connection

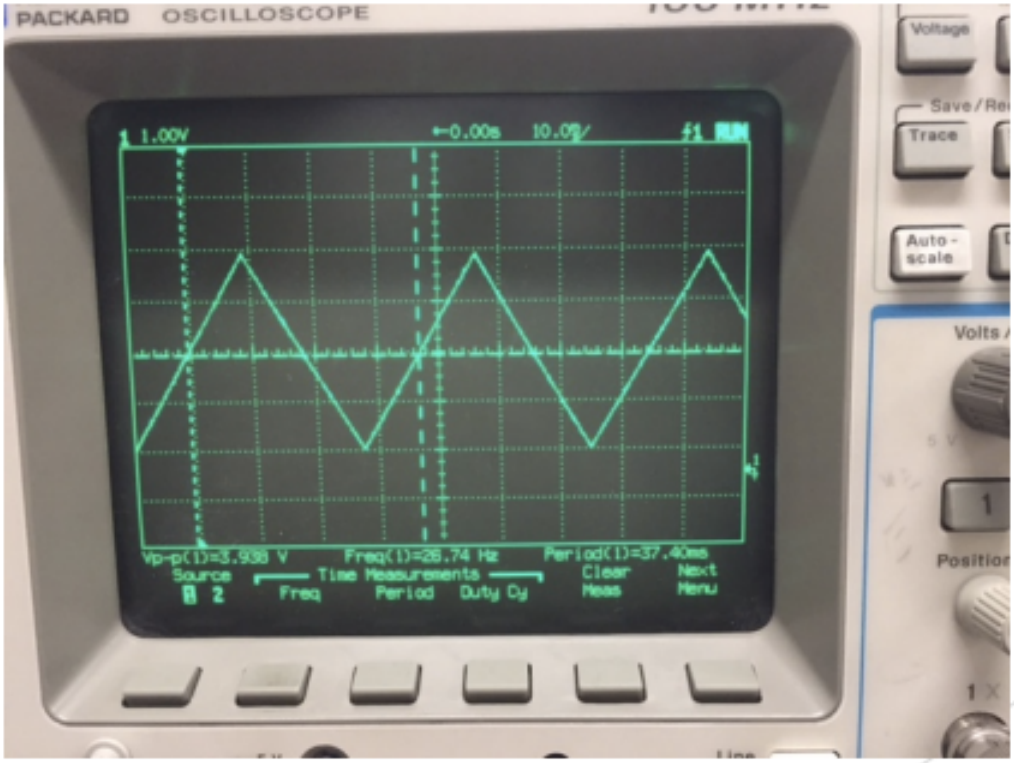


Figure 4 Triangle Wave output

To produce a triangle wave of the proper gain and period, I needed to modify the MCP code given to us in lab 1.

```
ICP_Code
clude <SPI.h> // Include the SPI library

d outputValue = 412;// A word is a 16-bit number
 incr = 4;

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

d 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(800000, MSBFIRST, SPI_MODE0));  // Set up the SPI transaction; this is not very elegant as there is never a close transaction action.


d loop()



 if (outputValue >= 3600 || outputValue <= 410){
   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
```

ure 5 Arduino Code

Here is a list of the changes I made:

1. Changed the starting value of outputValue to 412 (~0.5V) and the conditions of the loop to contain outputValue within 3600 and 410 (0.5V – 4.5V).
2. Changed the SPISettings value to 80000 Hz for a ~40ms period.
3. Changed the 4th bit of the HighByte variable from 0 to 1 for a $2V_{p\text{-}p}$ wave.

The resulting triangle wave was used as V_{TUNE} for our VCO for the proper frequency range.

Signal Processing:

For processing the final signal from our LPF, we decided to use the MATLAB code from lab 6.

// radar_range.m

```
%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('yagi_and_coffee_can.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 = 2260E6; %(Hz) LFM start frequency
fstop = 2590E6; %(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');
```

Here's a short function needed for the radar_range.m file.

```
// dbv.m
function out = dbv(in)
out = 20 * log10(abs(in));
```

At first, I tried to use the python script, range.py, to generate our RTI graphs. However, our results were quite lackluster, so I tried the MATLAB code and found much better results. (We later found out the python code had some bugs that were later fixed by Professor Leo.)

Conclusion:

The MCP was a very solid DAC, but it required a 2.5V V_{ref}. This does consume additional power. I tried using the Teensy's built in DAC, but we could not get a proper signal.

After Professor Leo updated the python script, I tried it with a wave file taken from our field test. The python script generated a black and white figure with varying hues of gray to specify power. This was very difficult to see outside, but produced somewhat similar results. We decided to use the MATLAB script in the end.