**Construction and Performance of an S-Band Laptop Radar**

Malbu

This document details the construction and testing of a FMCW (Frequency Modulated Continuous Wave) radar capable of range and Doppler imaging. The design is a variation of a radar built by Dr. Gregory Charvat for an IAP course at MIT and consists of readily available parts for a budget of $317. The price and availability of these components makes this project ideal for introducing basic radar principles to students as part of a course or independent study. A block diagram of the system, along with the measured power outputs of each element, can be seen in the next figure.

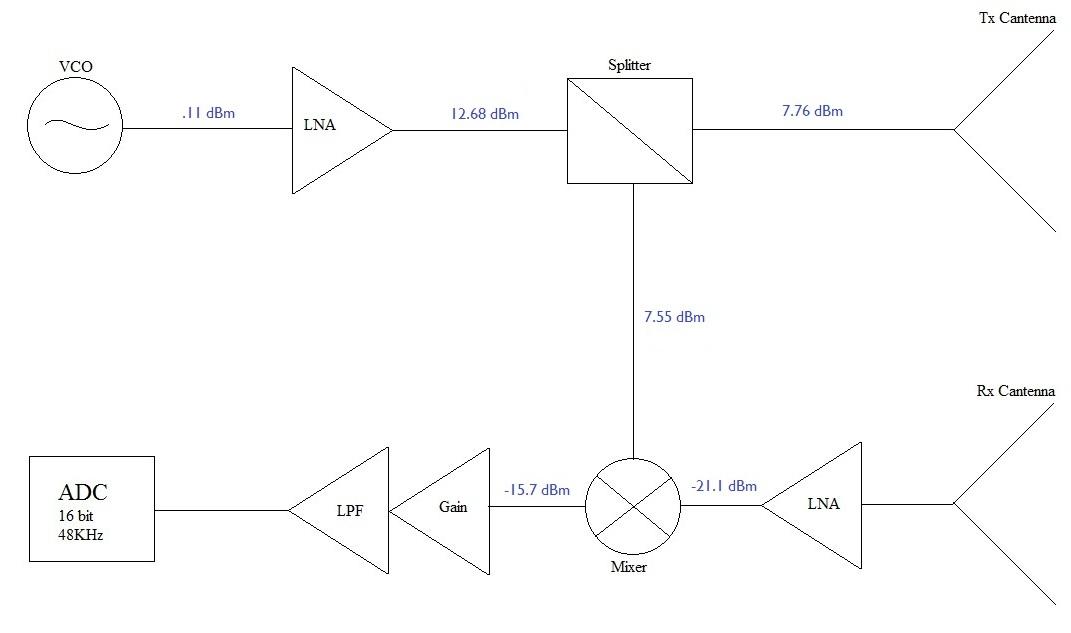


Figure 1. System diagram with measured power values.

A voltage controlled oscillator (VCO) can be DC tuned to produce a stationary sinusoid at a center frequency of 3.395GHz for doppler imaging, or modulated to produce a linear chirp that can be used for ranging. The signal is amplified and enters a splitter which transmits half of the signal via the Tx antenna. A frequency mixer is used to combine the signal received from the Rx antenna and the original transmitted signal from the splitter to reveal the phase difference to the target. This output signal is then amplified, and passed through a low pass filter . The transmit power is approximately 6mW.

**Part 2. Analog Circuit Description and Operation**

The two main sub-circuits are an op-amp gain and LPF circuit and a ramp generator circuit. The gain and LPF circuit is seen below.

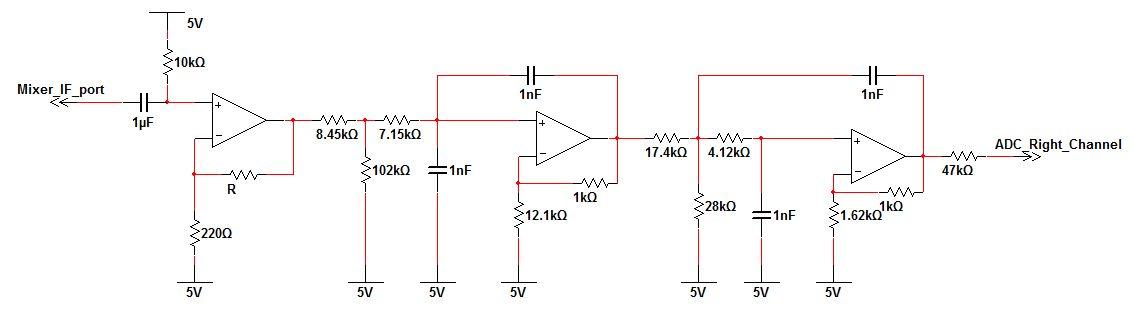


Figure 2. Analog gain and low pass filter.

The filter is a 4th order 15 kHz filter which prevents aliasing at the input port of the ADC. The response of the circuit with respect to frequency was measured and is shown on the next page.

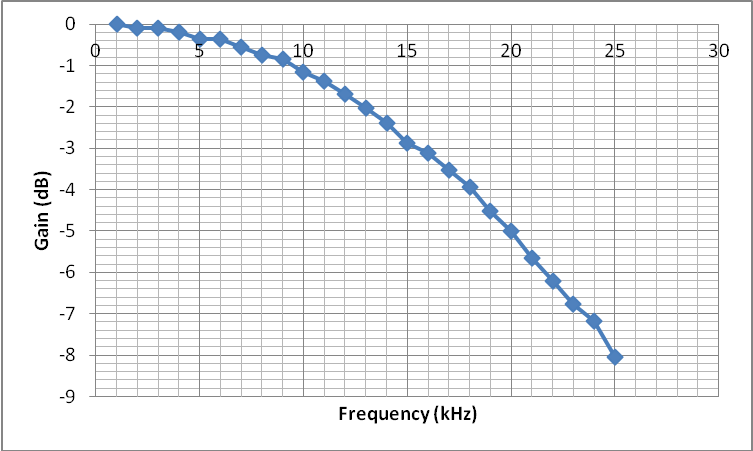


Figure 3. Gain of low pass filter.

As desired, the attenuation at 15 kHz is close to 3dB and increases beyond this frequency.

The second sub-circuit, is a function generator capable of producing a ramp to modulate the VCO and produce a chirp. The main component is a Exar monolithic IC wired as shown in Figure 4.

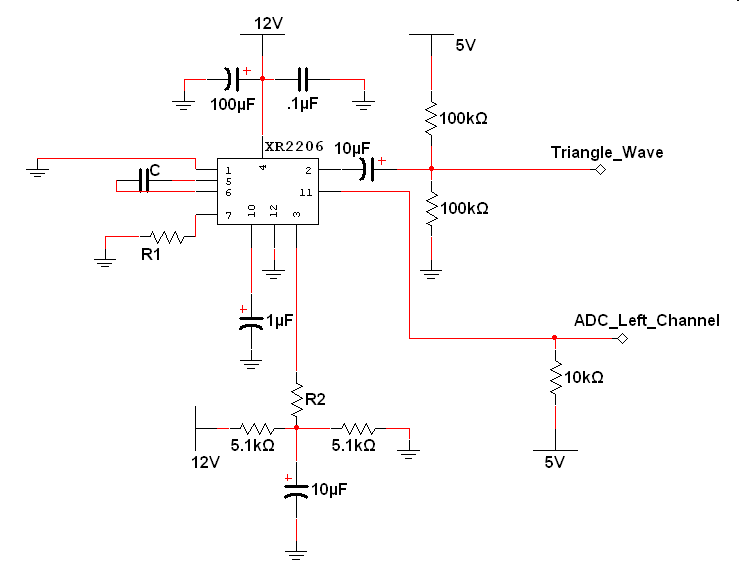


Figure 4. Function generator subcircuit.

The generated waveforms are shown on the next page.

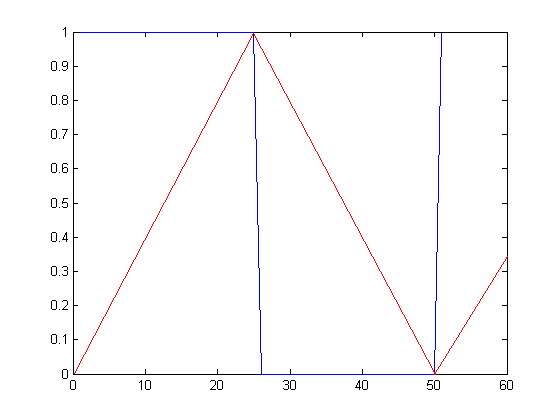
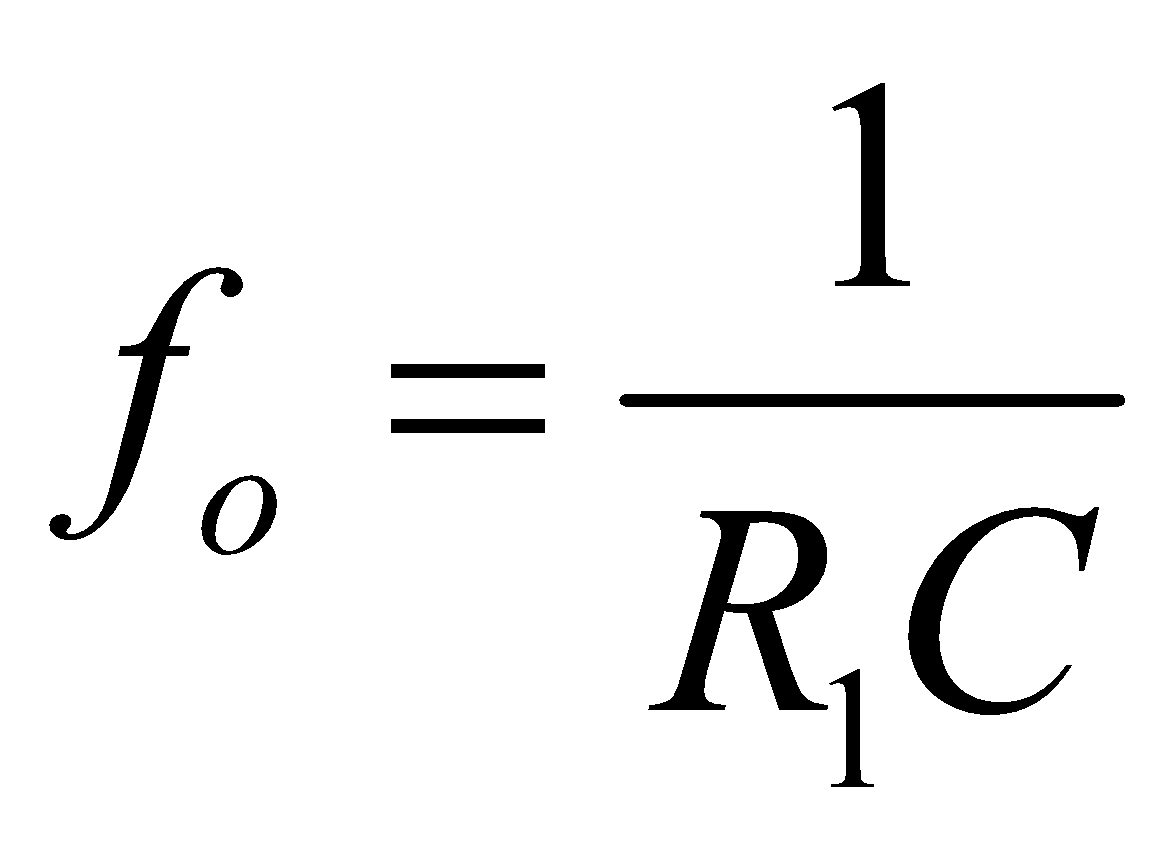
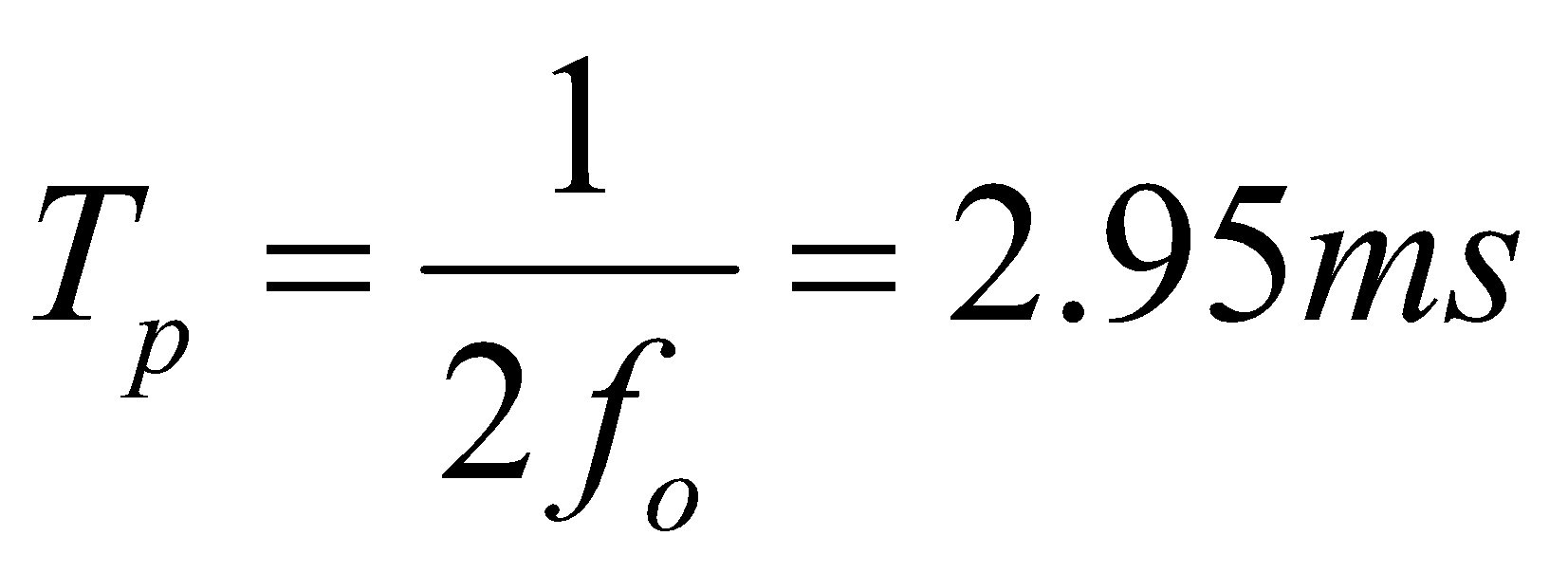


Figure 5. Output of function generator.

R1 and C were adjusted according to  to produce a ramp frequency of 169Hz. The duration for the positive slope of the ramp, and the positive portion of the square wave, is then . For ranging, the square wave is used to mark the start and end of a positive chirp slope and parse out the corresponding mixer data. This will be discussed further in section 5.

**Part 3. Antenna Design**

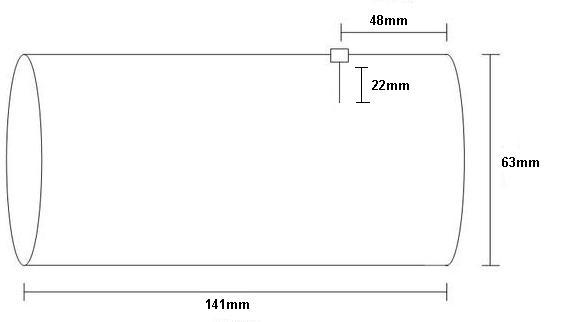
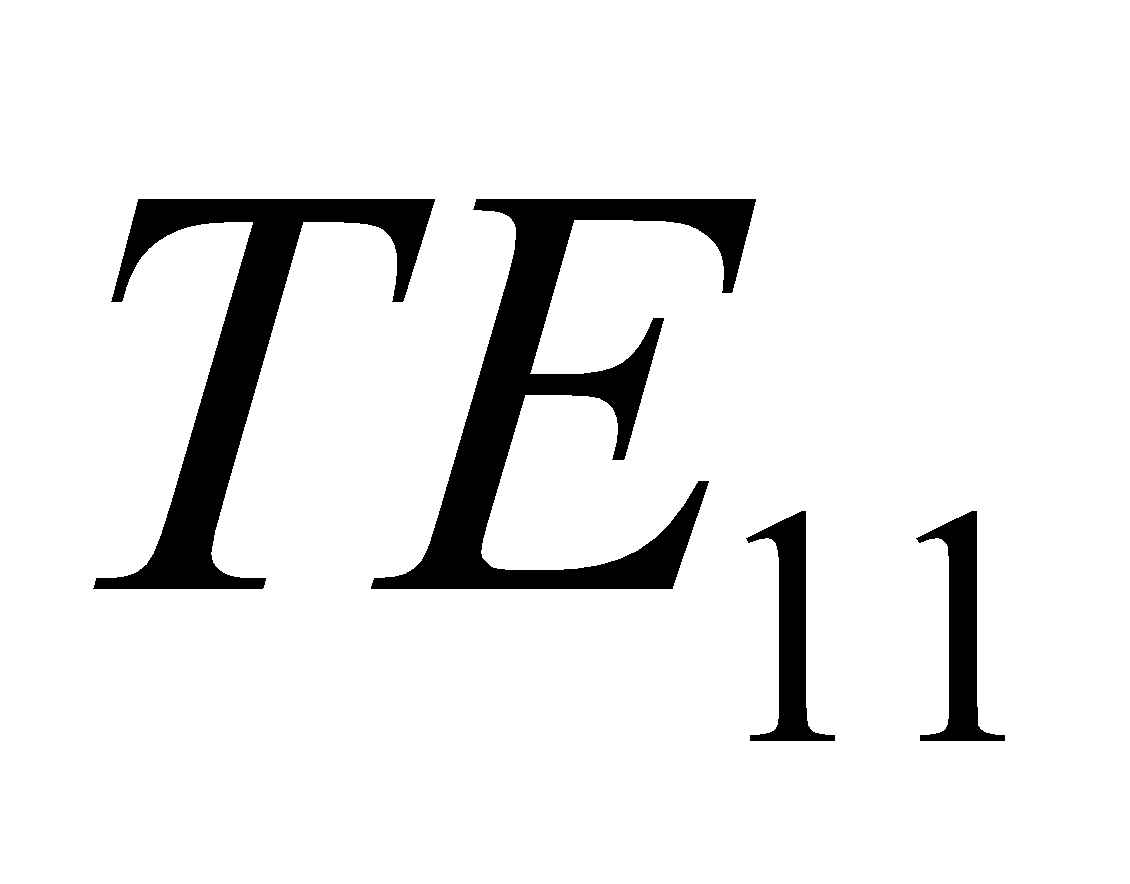
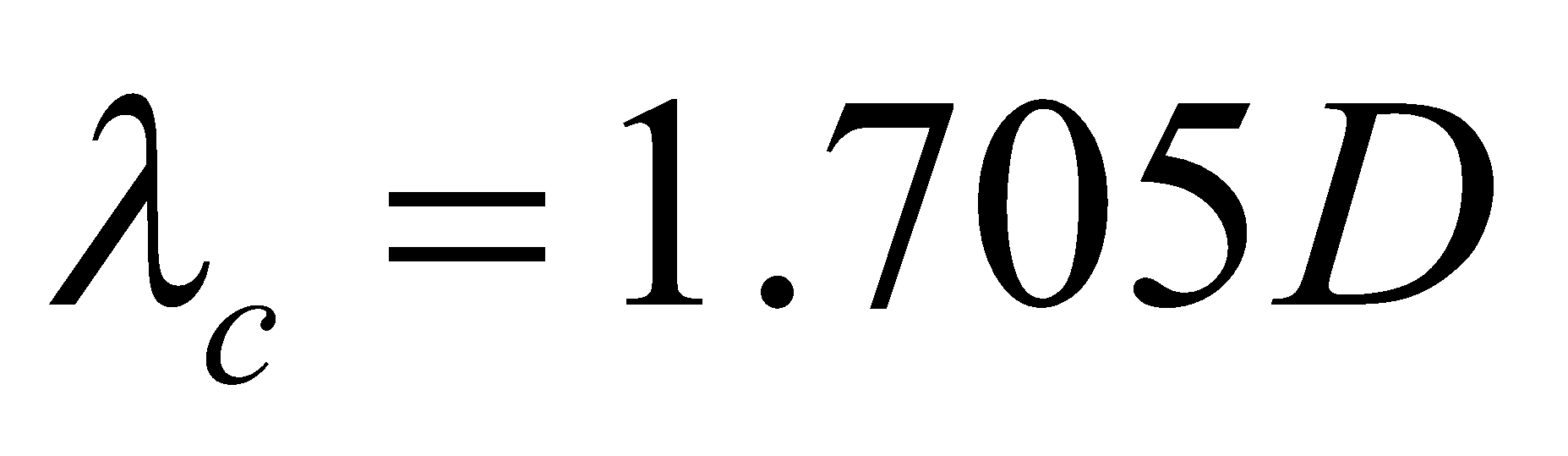
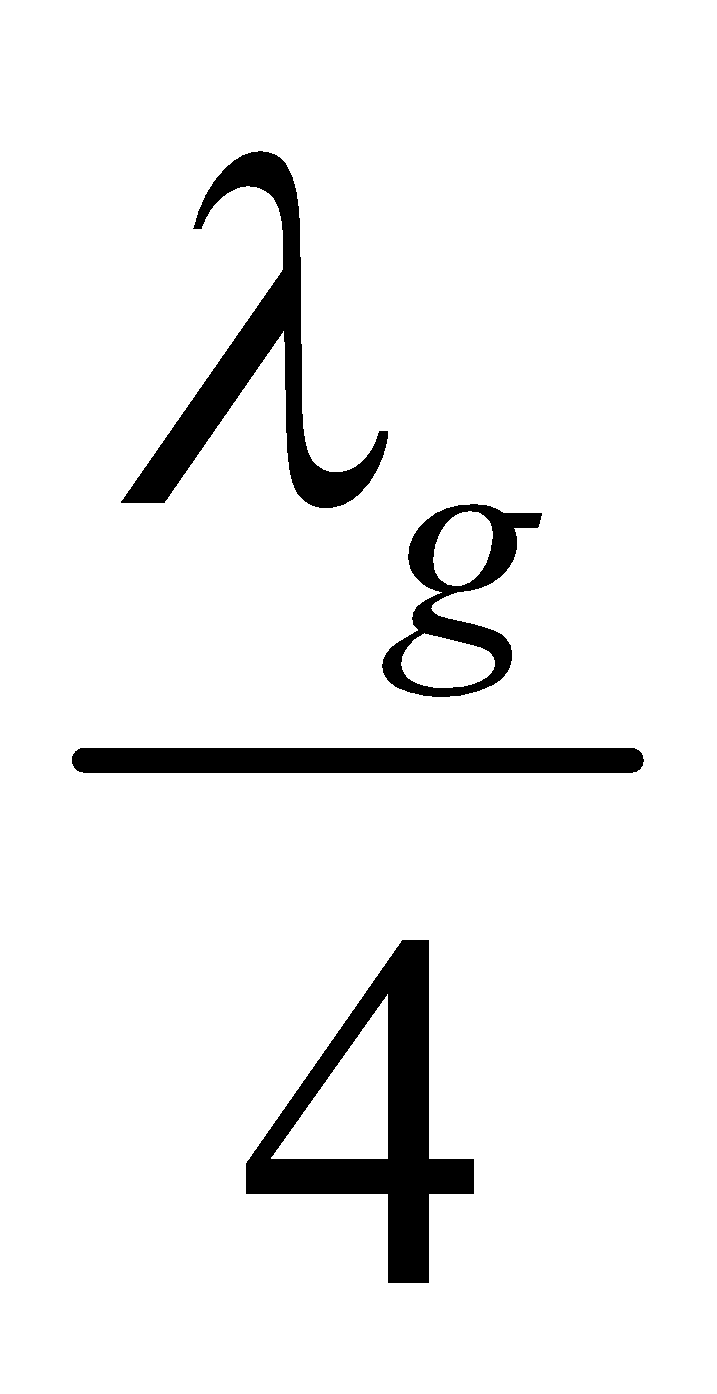
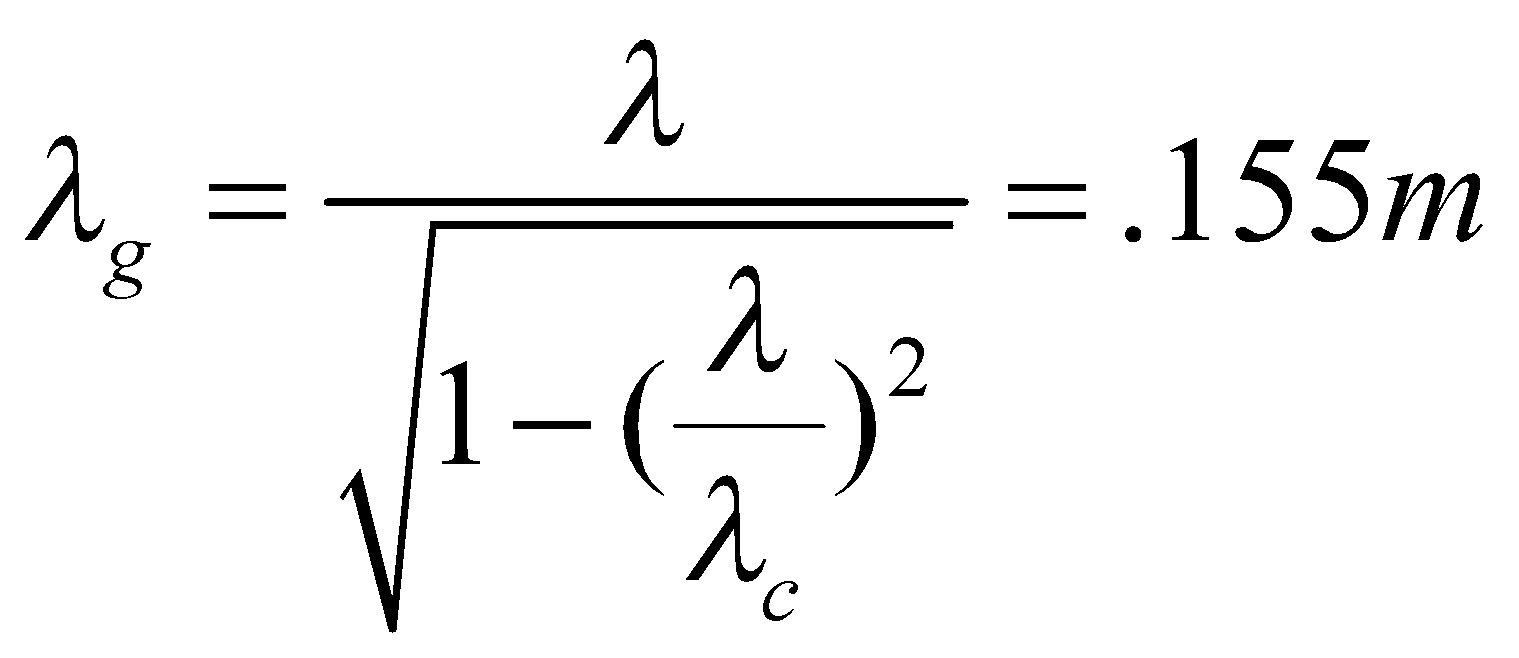
The Tx and Rx antennas are circular waveguides constructed from recycled cans and fed with a quarter wavelength monopole as seen in Figure 6. 

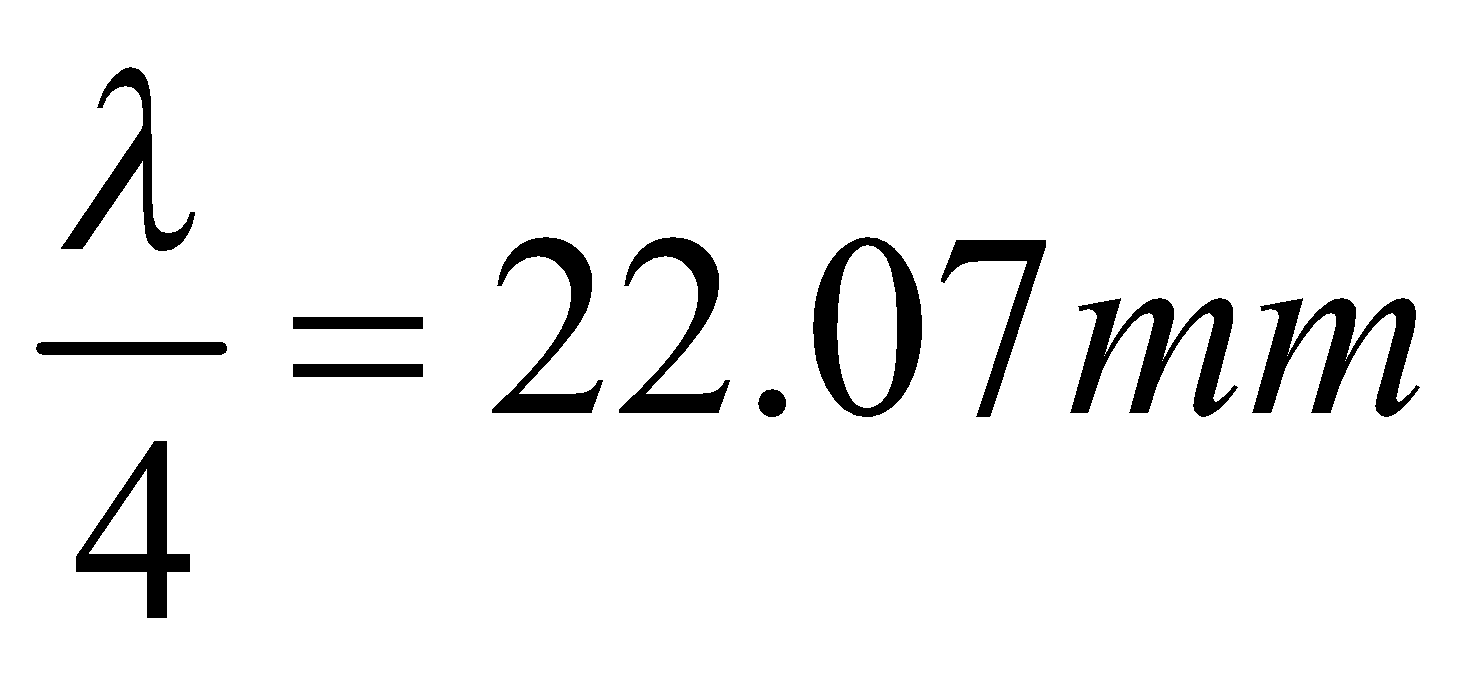
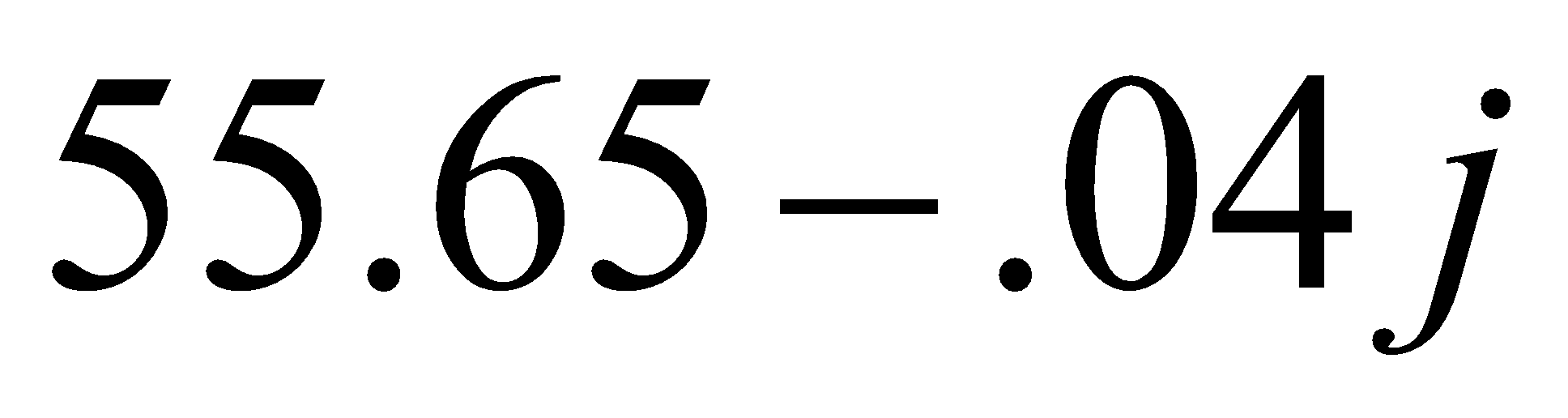
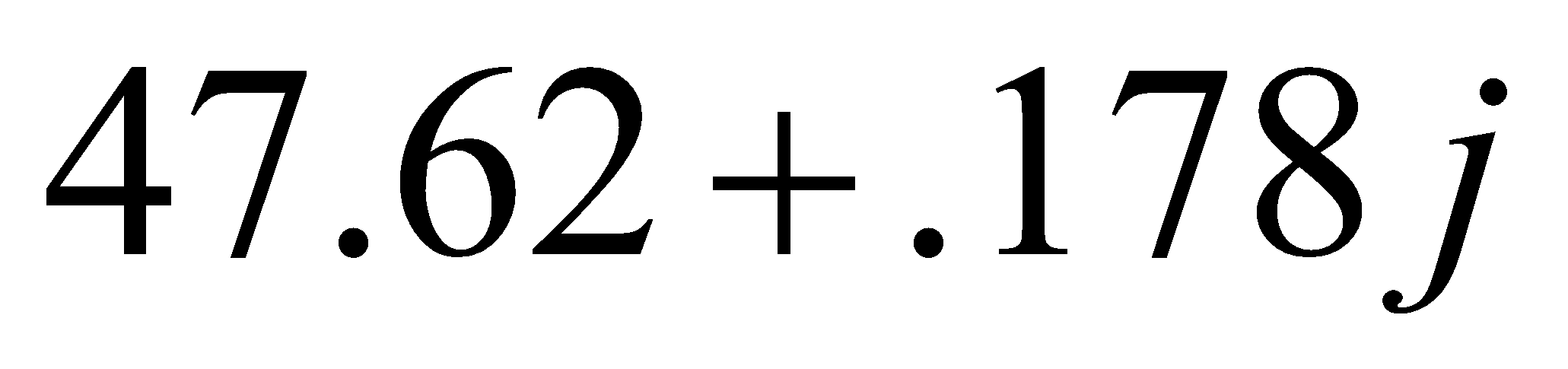
Figure 6. Final dimensions of “Cantenna”.

After a search of the supermarket aisles, an asparagus can with a diameter of 63mm was chosen. The cutoff wavelength of the dominant was calculated to be:

= 107.4mm

This yields a cutoff frequency of 2.8GHz. The position of the probe from the back wall of the guide was initially calculated to be  =26.85mm where:



As previously mentioned, the probe is a quarter wavelength long or . An HFSS model was constructed with the calculated dimensions. A parametric analysis was run to observe the effect of probe position on the input impedance and radiation pattern of the waveguide. The figures on the following pages show the simulated input impedance and radiation pattern when the probe position was changed from the calculated value to 48mm from the closed end. With the new probe position, the input impedance changed from 94.97+38.72 to . The measured input impedance of the fabricated antenna was . The VSWR and return loss can be seen in figures 7 and 8, respectively.

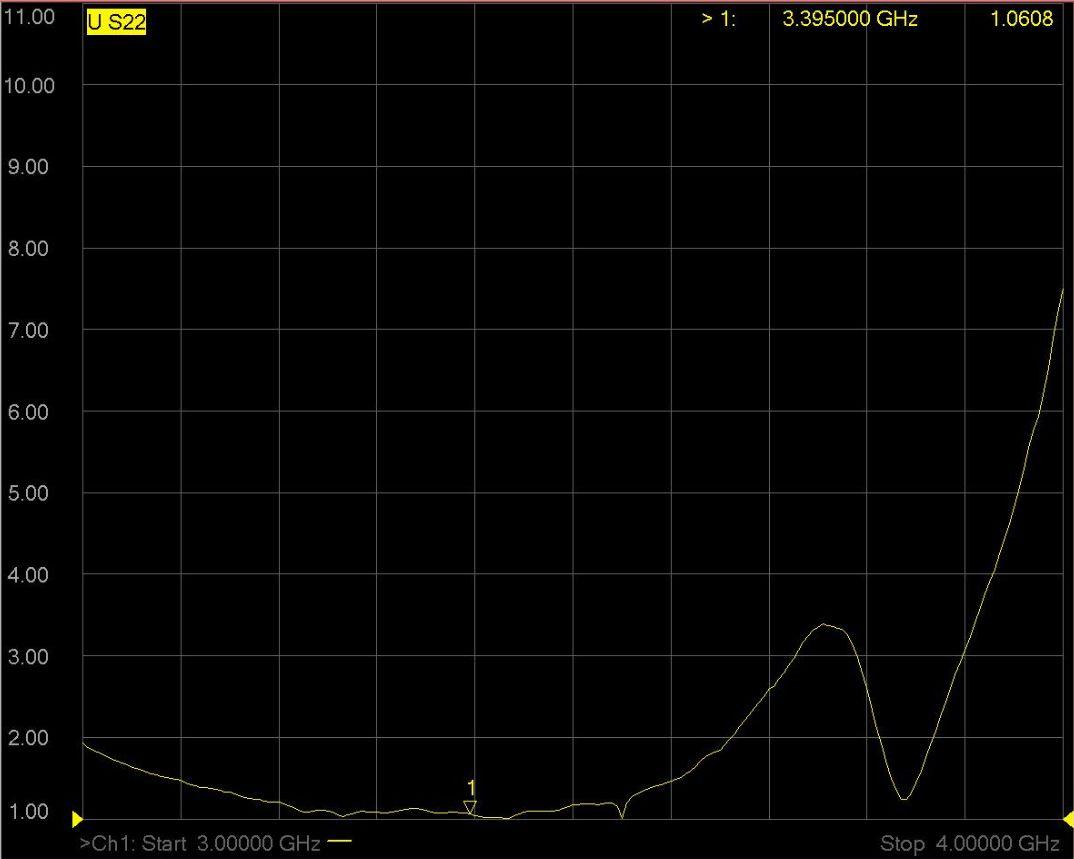


Figure 7. VSWR of mounted Cantenna

As seen from Figure 7, the bandwidth (VSWR≤2) for the match is large at 659MHz which is desirable since the frequency will be swept when ranging.



Figure 8. Return loss of mounted Cantenna.

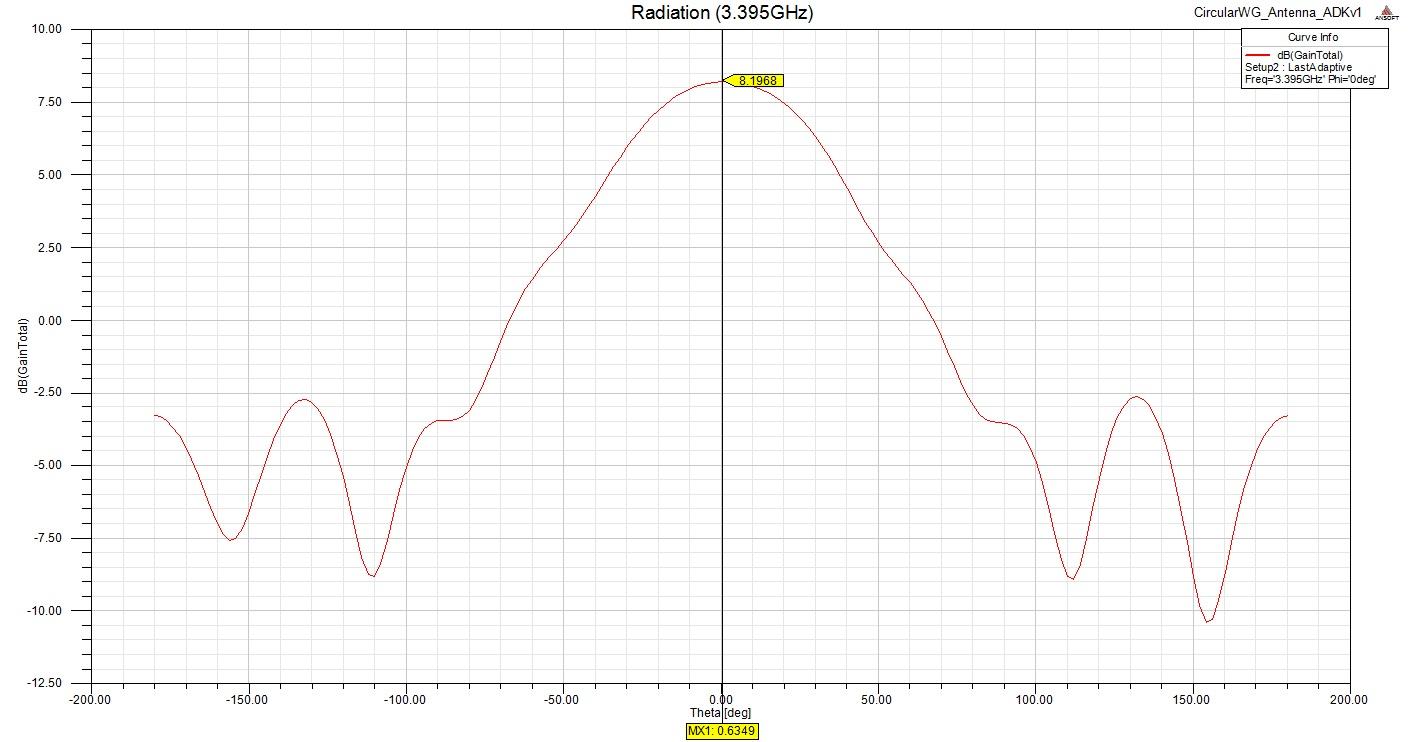


Figure 9. Simulated radiation pattern of Cantenna in free space.

The calculated gain is 8.1968 dB with a HPBW of 72 degrees. Ideally, the HPBW should be narrower. This can be accomplished by increasing the length of the guide.

**Part 4. Data Acquisition and Real-Time Gui**

Initially, the ADC to be used was the standard laptop sound card. This proved to be inadequate as two independent channels were needed for range imaging, one channel for the radar data, and a second for the square wave marking the chirp boundaries. The laptop used for testing and data acquisition had a sound card with only a single channel wired to both channel inputs. The alternate low cost solution ($11) is an Inport Deluxe USB device. This device has two channels, a maximum sampling rate of 48kHz, and is recognized as a sound card by the MATLAB Data Acquisition Toolbox.

There are two possible option for data acquisition. The first option is recording the radar data as a WAV file and using the MATLAB wavread function to extract this data into an array to be processed. For his original design, Dr. Charvat has provided MATLAB programs for both doppler and range processing. The second option consists of a MATLAB program, daqradar.m, written to accomplish the same goal in real-time. This program can be seen in its entirety in the Appendix I. The two methods will be compared in the next sections.

The gui for the real time daqradar.m is shown on the next page.

The layout of the gui is as follows:

A. Mixer output display; ADC right channel.

B. Square wave diplay; ADC left channel.

C. Data extracted from positive slope of chirp.

D. Difference signal of two consecutive frames.

E. FFT of chosen signal. Depends on which processing method is chosen.

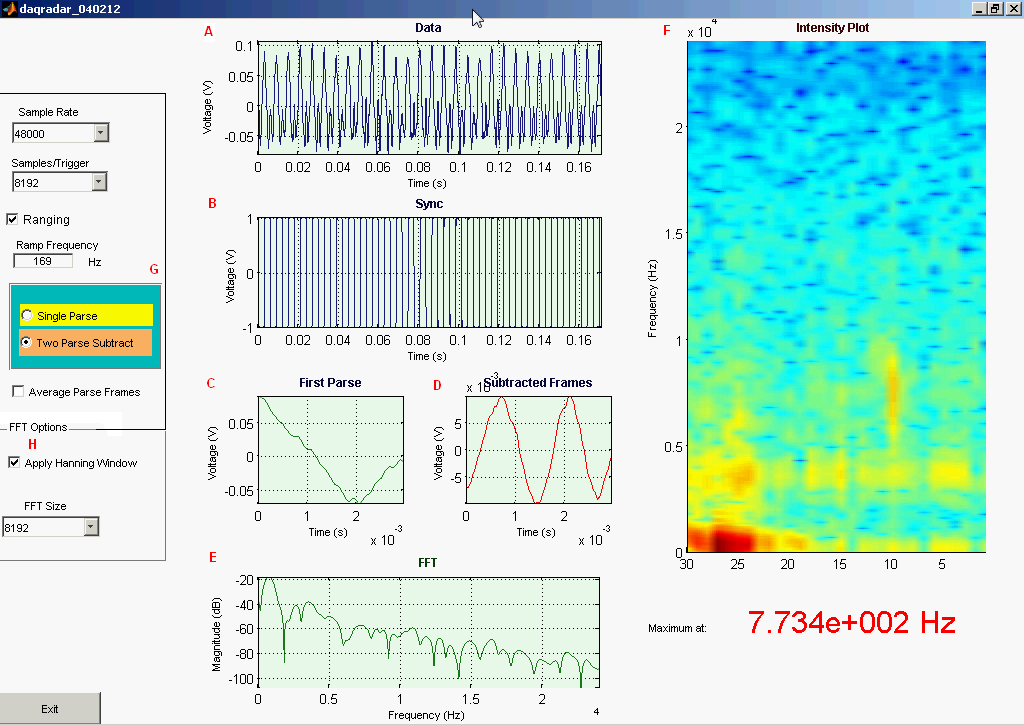


Figure 10. Daqradar.m gui.

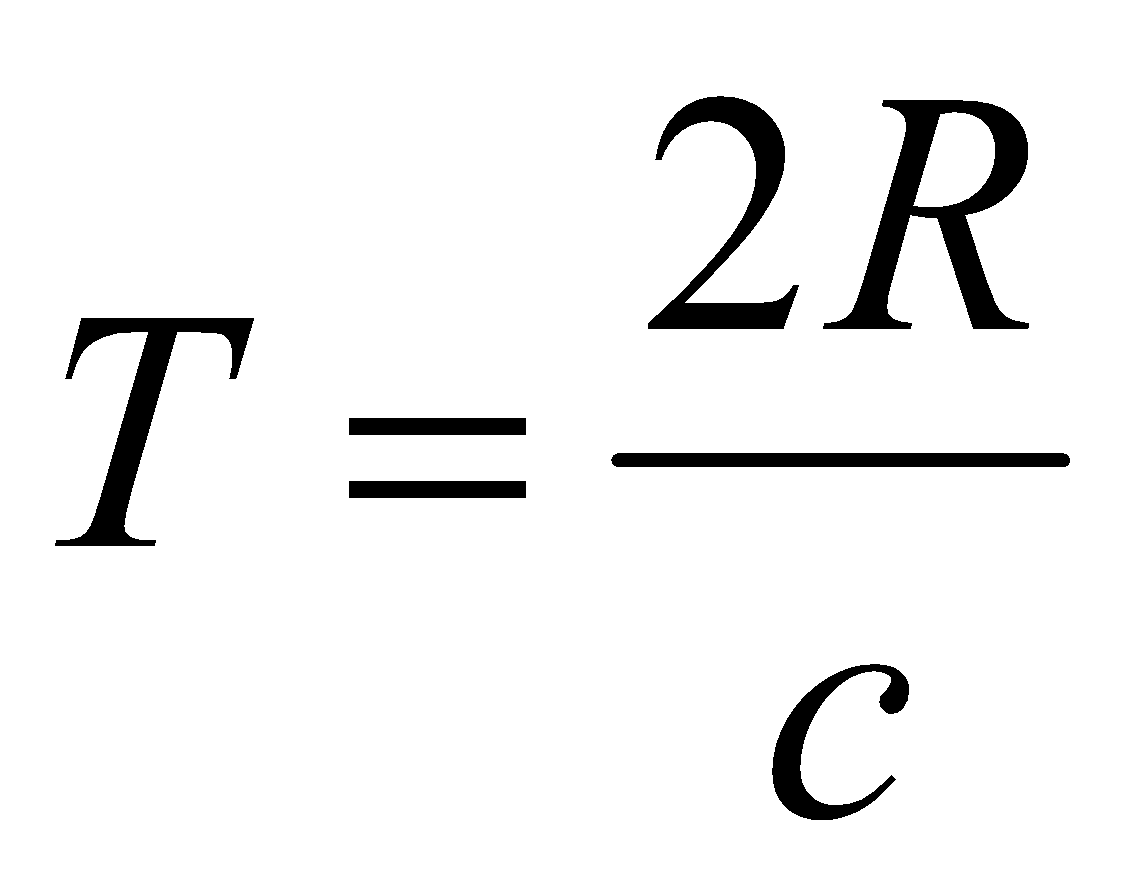
F. Spectrogram composed of binned FFTs.

G. Two processing methods available: Single frame vs. two frame clutter rejection

Functions C through G are discussed in section 5.

H. Apply Hanning Window before taking FFT.

**Part 5. Range Imaging**

FMCW ranging utilizes frequency shifting to create a time marker for the transmitted pulse. In this case, the transmit pulse is a sinusoid modulated by a triangle function. The transmit frequency of the resulting signal increases and decreases linearly as seen in the figure below. The frequency difference between the transmitted and received pulse is proportional to the round trip time T and also to the range of the target since  **.**

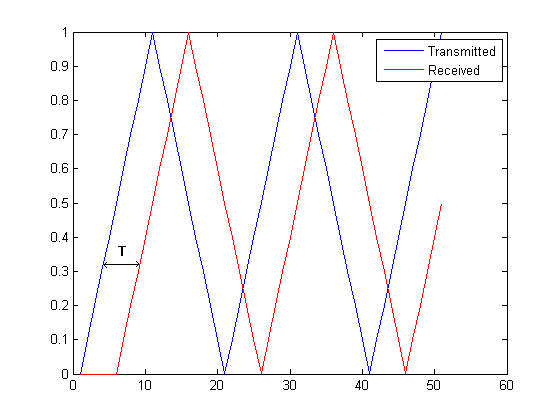
****

Figure 11. The mixer output is the product of received and transmitted chirps.

In a clutter-free environment, the output of the mixer is expected to be a clean sinusoid. To test this claim the radar assembly was rewired as seen in Figure 12. A length of cable L is used to simulate the target at a range of L/2 from the radar. Figure 13 shows the effect of different lengths L on the beat frequency.

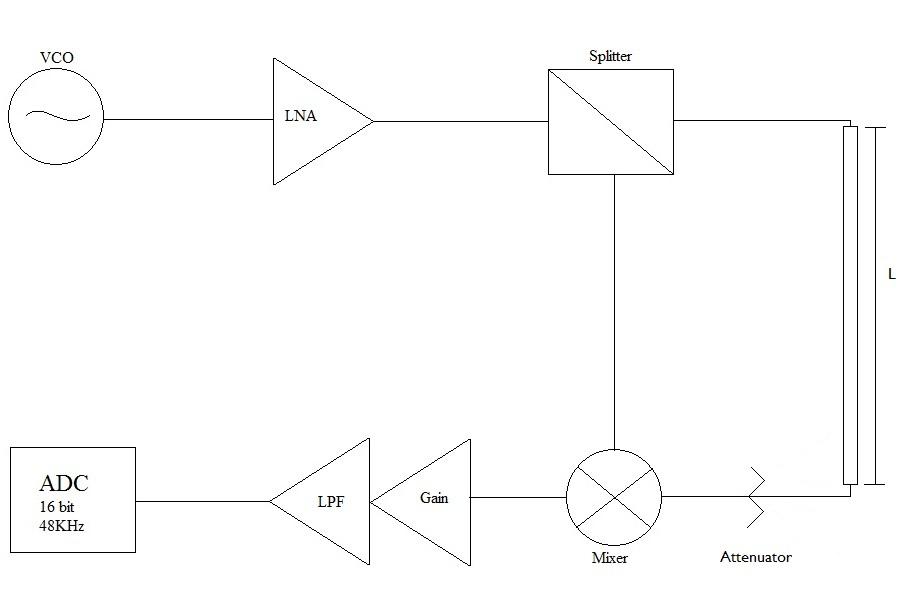


Figure 12. Test setup to eliminate unwanted reflection.

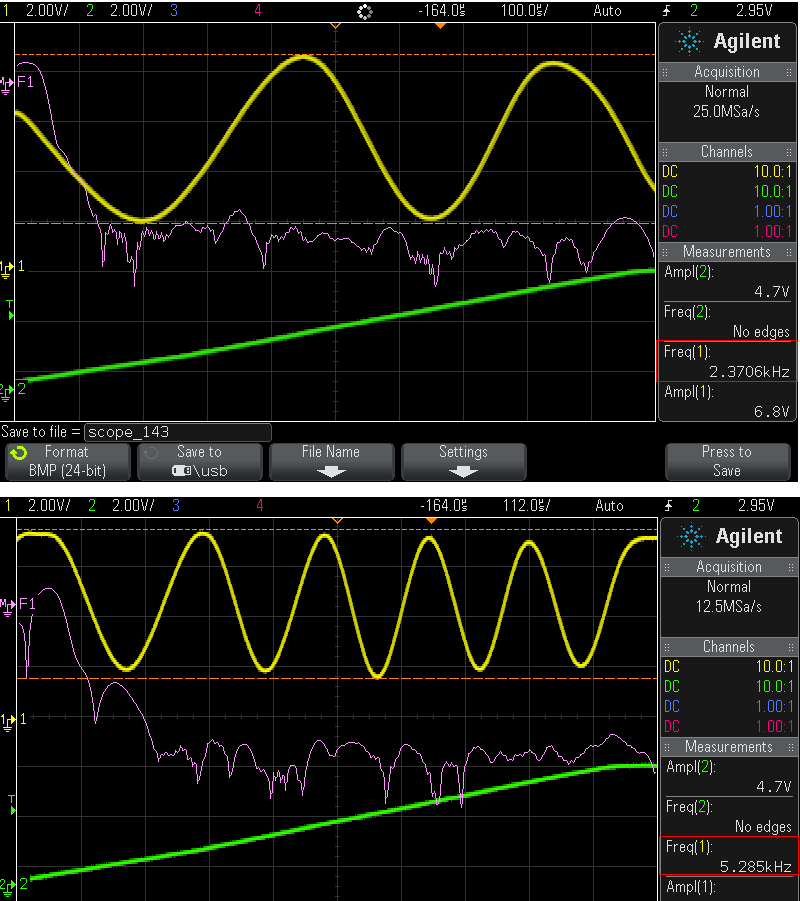


Figure 13. The beat frequencies (boxed in red) increase proportionally with L. The FFT of the sinusoid is shown in purple and the ramp signal in green.

The radar signal for a target in a real environment is not as simple. This is due to the added reflections of unintended targets (trees, houses,etc). An example of a realistic signal can be seen in Figure 14i. The superimposed square wave marks the edges of the chirp. We wish to take the fourier transform of the radar data on the positive slope of the chirp. In order to do this, the hardware generated square wave is smoothed by “thresholding”. This discards everything below a set value and retains the rest. The resulting waveform is shown in Figure 14ii. By experimentation, this threshold value was set to .8 (the magnitude of the generated square wave is 1). While a higher threshold value might yield a more accurate edge position, it also risks finding false edges. Notice how, in Figure 14i, a rising edge will have a ripple that dampens before the following falling edge and vice versa. To determine whether a found edge is rising or falling, the magnitudes of the points immediately before and after an edge location are compared. Figure 14iii marks the falling and rising edge locations and Figure 14v shows a single extracted frame.

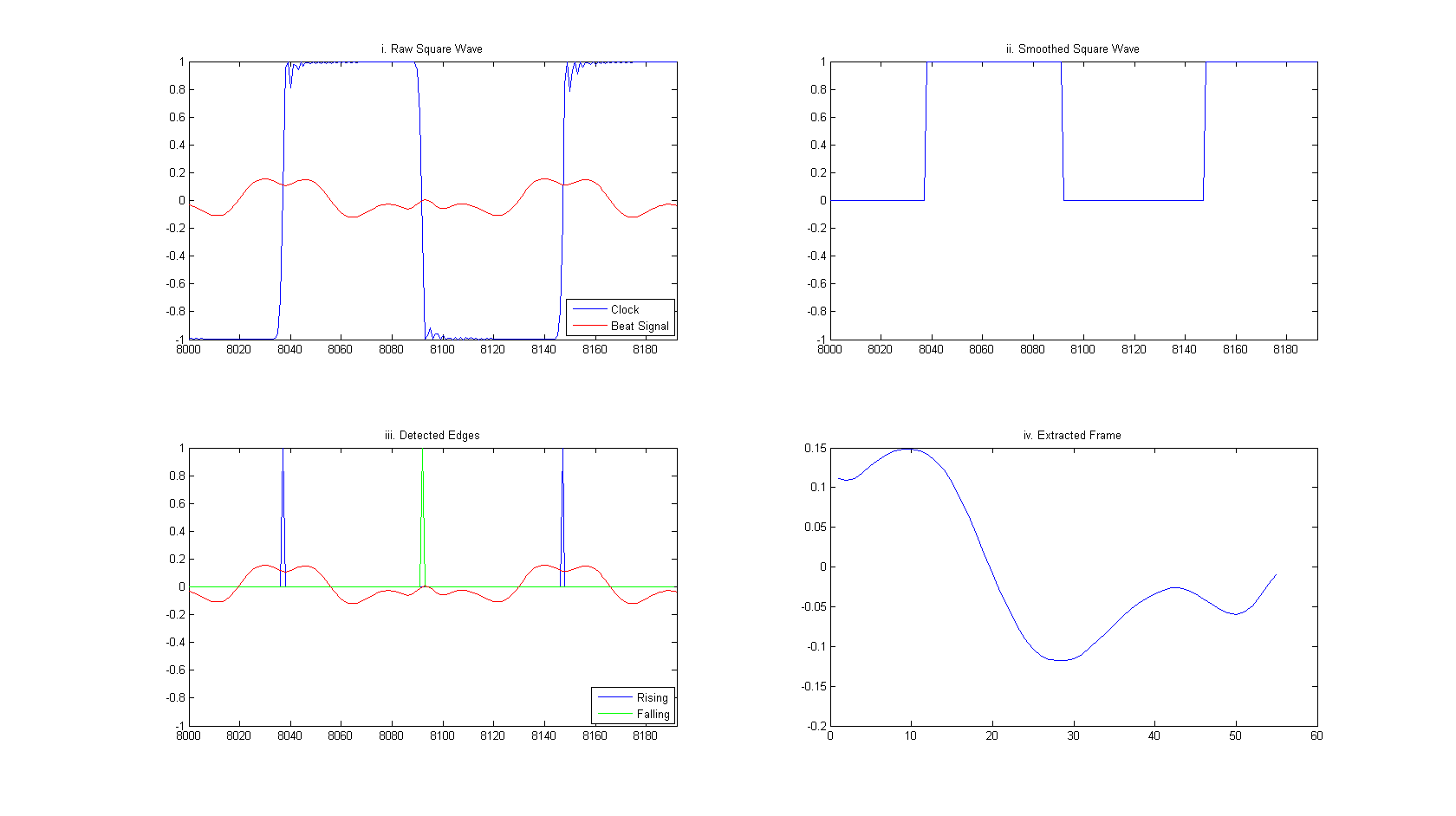
****

Figure 14. Edge processing and data extraction.

The FFT of multiple frames are then binned together to form an image. Figures 15 shows the result of this process. For this example, the target was walking at the constant velocity toward and away from the radar. Elapsed time is marked on the vertical axis and beat frequency on the horizontal. While the motion of the target can be made out, it takes a bit of imagination. The program provided by Dr. Charvat produces the image seen in Figure 16 for a target moving in a similar fashion.

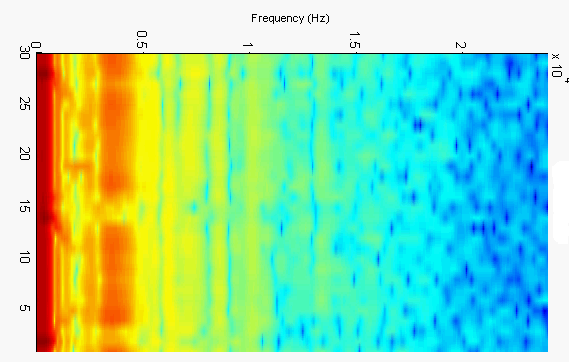
****

Figure 15. Image generated in real time.

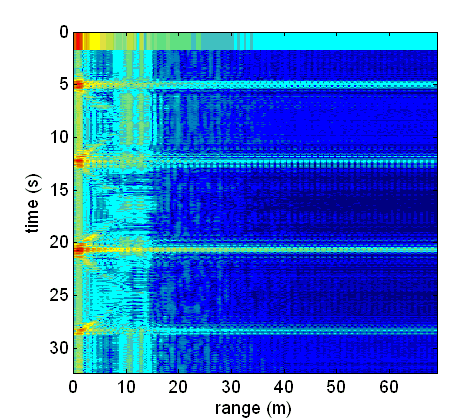
****

Figure 16. Generated from recorded wav file.

An effective method of minimizing the weight of static clutter while ranging moving targets involves taking the difference of successive frames. Two consecutive frames are seen in Figure 17i and their difference signal in 17ii.

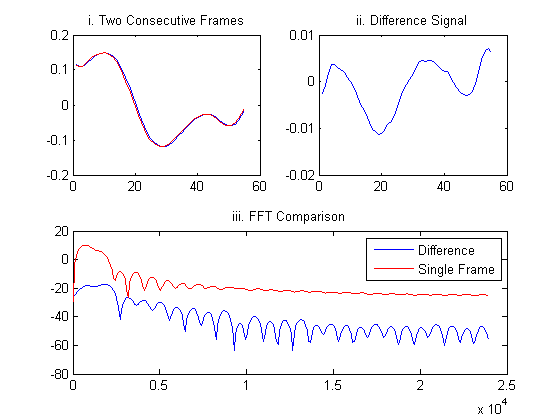


Figure 17. Two frame clutter rejection.

The FFTs of the difference signal and a single frame are shown for comparison in Figure 17iii. The improved images produced with this method are shown on the following page. The motion of the target in these cases is much more readily apparent. Another example that compares these two processing methods are seen in Figures 20 and 21.

Some additional traffic snapshots are seen in Figures 22, 23 and 24.

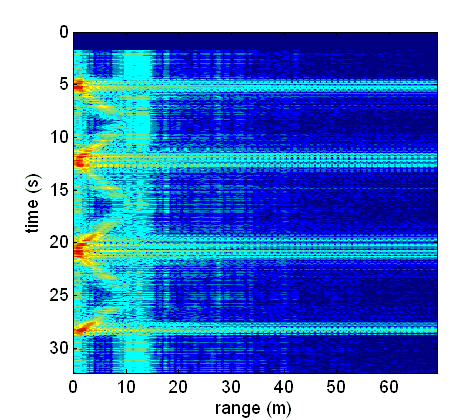
****

Figure 18. Generated by differencing with Dr. Charvats program.

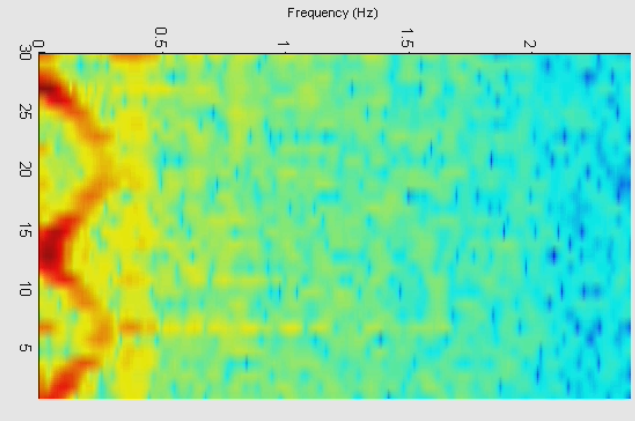
****

Figure 19. Generated by differencing with daqradar.m

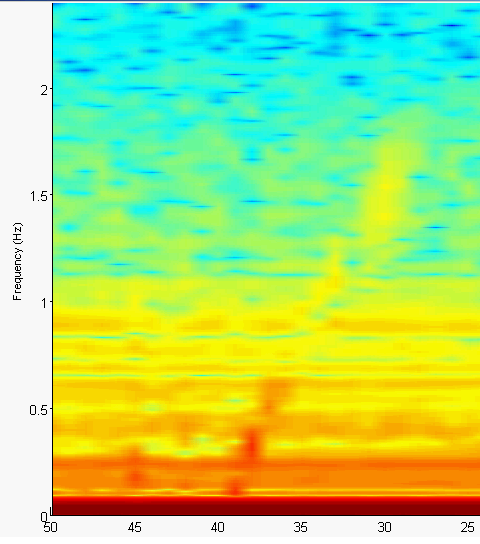
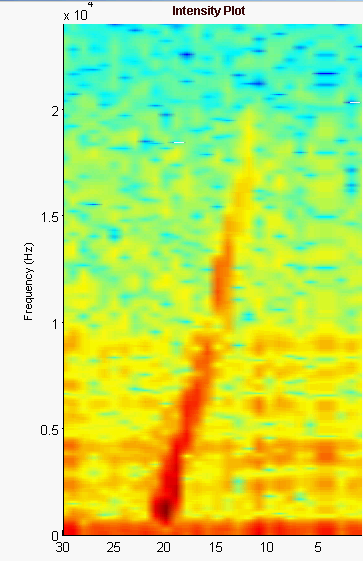
** **

Figure 20. Large target (truck). Single frame processing. Figure 21. Similar target processed with

with differencing method.

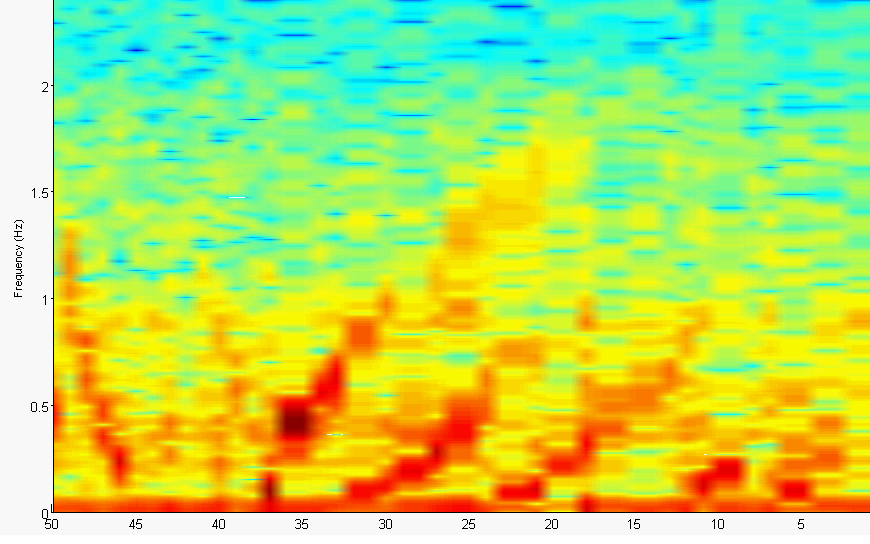
****

Figure 22. Passing traffic. Processed by using differencing method.

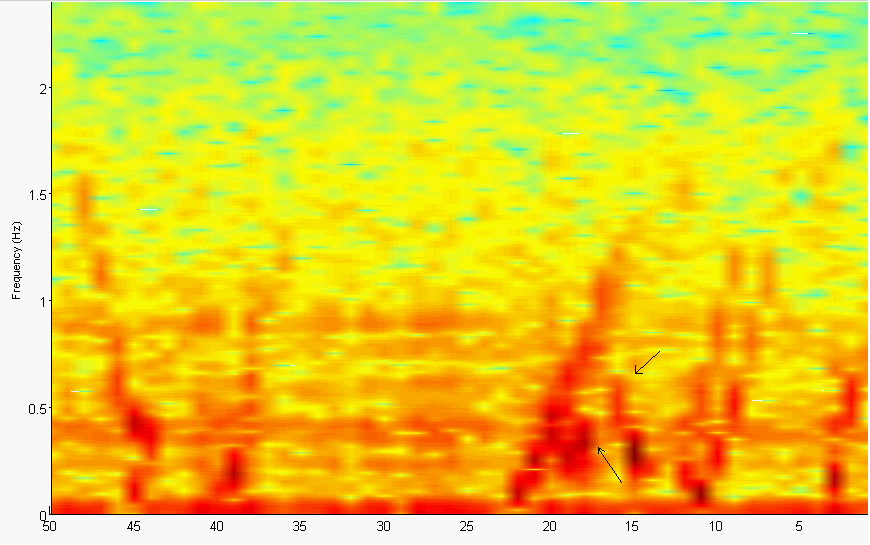
****

Figure 23. Traffic on Rt 40. The arrows mark eclipsing vehicles.

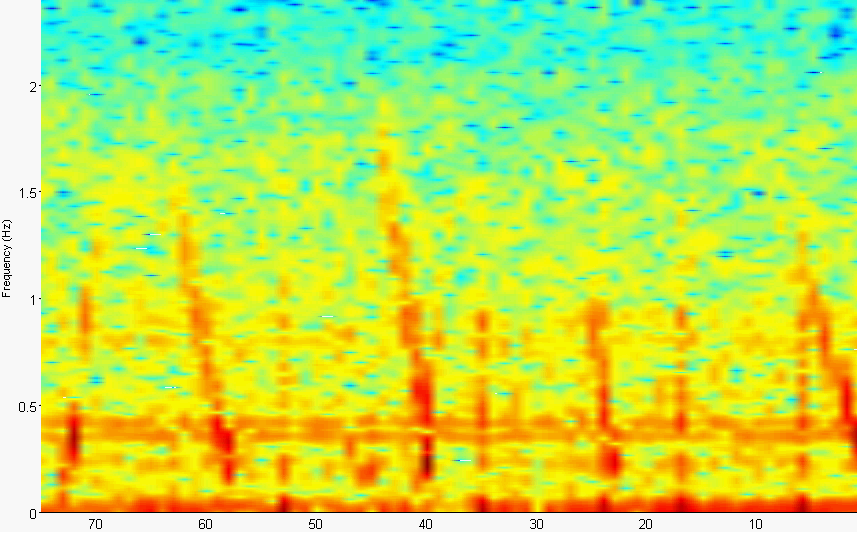
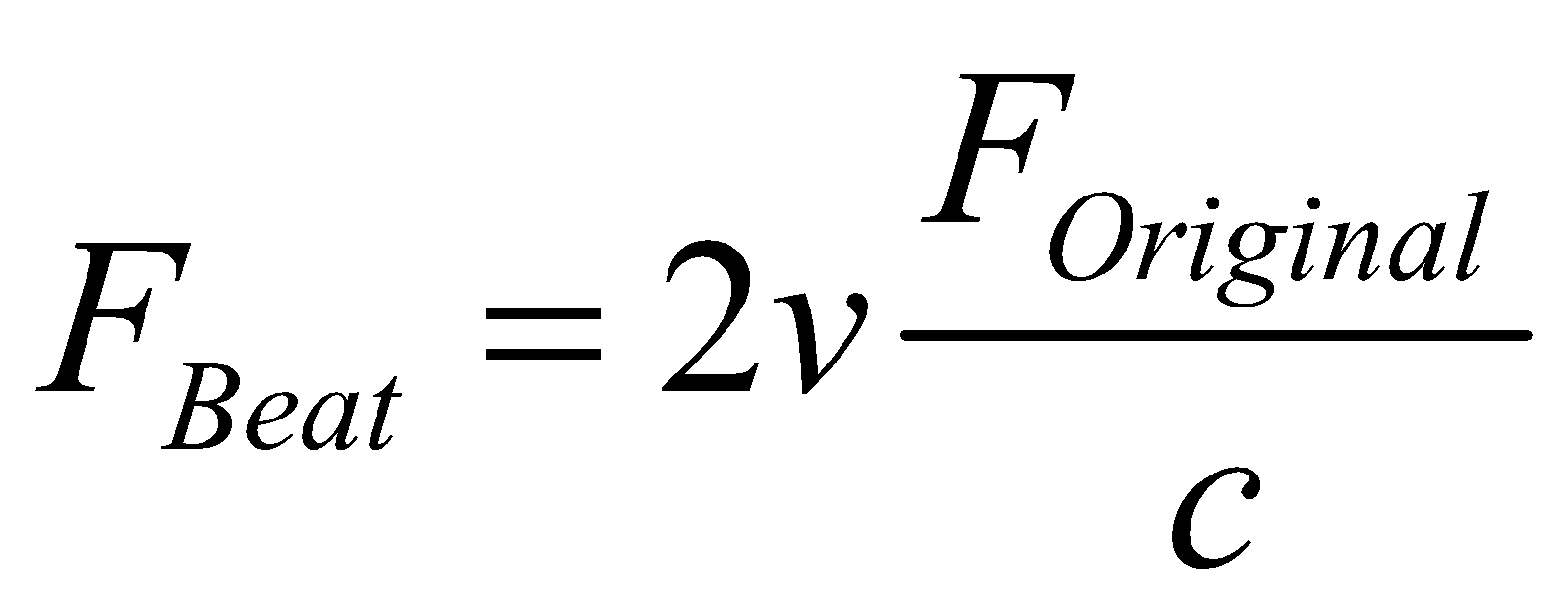
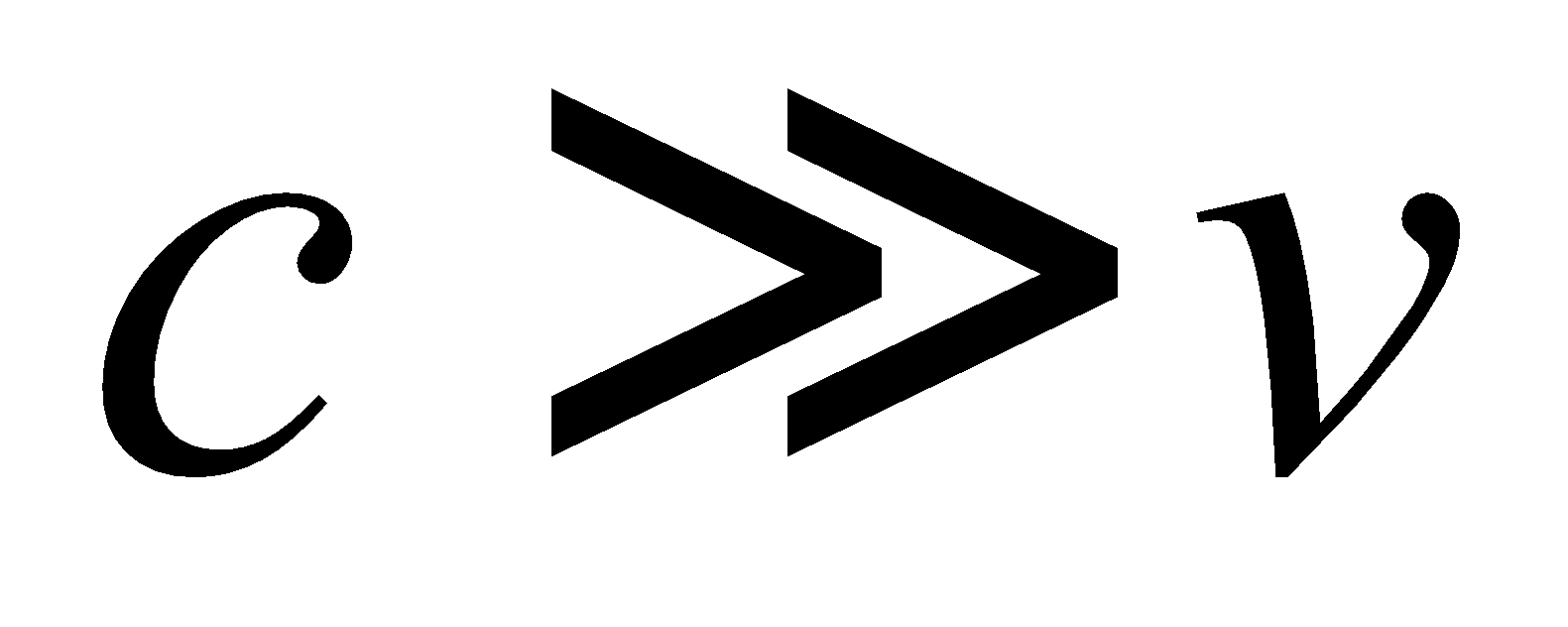
****

Figure 24. Another portrait of multiple targets on Rt. 40

**Part 6. Doppler Imaging**

For Doppler imaging, the VCO is DC tuned to 3.395GHz. The transmitted signal will then be a simple sinusoid with a constant frequency. A non-stationary target will reflect the transmitted signal with a shifted frequency proportional to the velocity of the target with respect to the observer. The difference of these two frequencies is called the beat or Doppler frequency and can be calculated from:

 where 

In this mode, the functionality of this radar is similar to handheld radars used by police. The main advantage of extracting velocity information using this method is that it is not as susceptible to static clutter from the environment since it is only sensitive to targets that are not stationary. Therefore, good snapshots can be obtained with very little processing. However, due to the simplicity of the transmit waveform, only the magnitude of the velocity can be observed and not the position and direction of motion. Some results obtained from the real time program can be seen in the figures on the following pages.

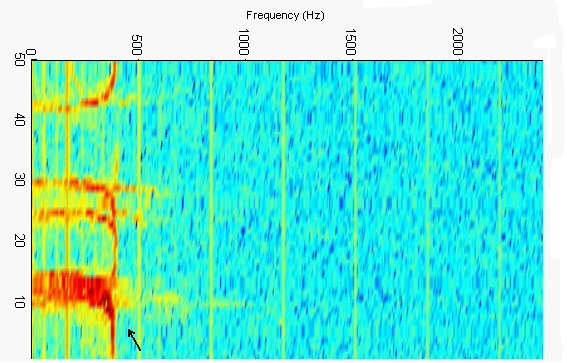


Figure 25. Vehicles traveling down Rt. 40. Note the larger signature (marked with black arrow) of a semi-trailer truck.

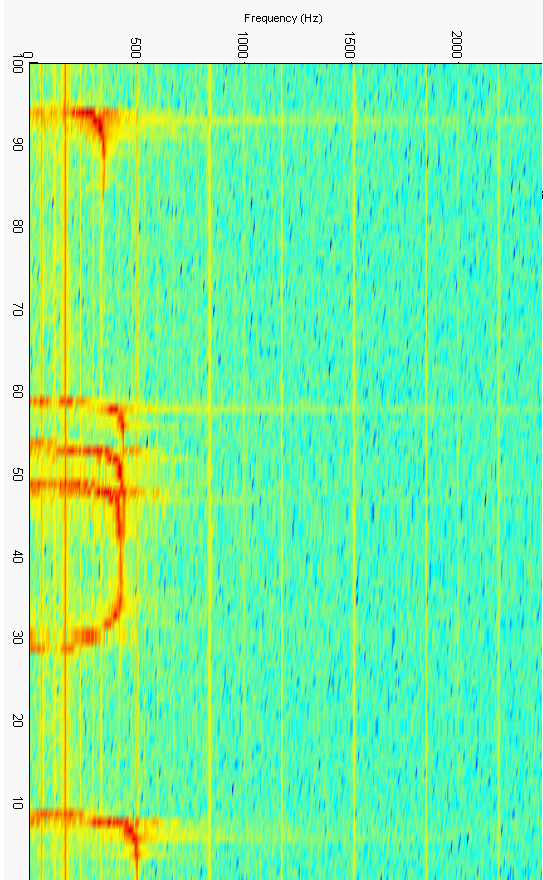


Figure 26. The maximum frequency on this graph is 474 Hz which corresponds to a velocity of 18.1 m/s or 40.5 mph. The speed limit is 35 mph.

The following images were obtained by using the MATLAB program provided by Dr. Charvat. They are presented for comparison with the results shown on the previous page.

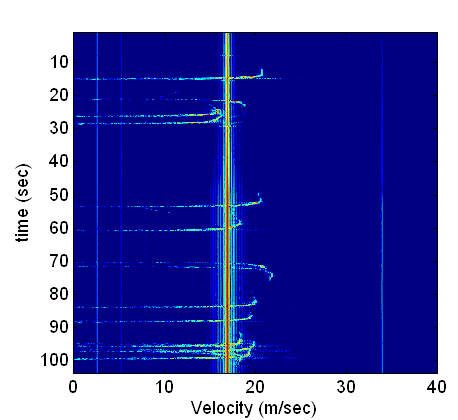
****

Figure 27. More vehicles on Rt. 40. Note the prominent vertical striation. This artifact was minimized by increasing the gain of the op-amp in Figure 2. The gain was initially set in the lab to prevent over-volting the ADC. Inport lists a maximum permissible input voltage of 6V.

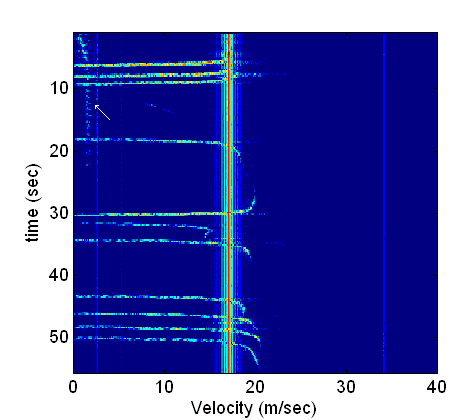
****

Figure 28. An interesting Rt. 40 scene. The velocity of a pedestrian walking in front of the detector is marked by a white arrow. The remaining signals are vehicles.

**Part 7. Further Consideration and Possible Improvements**

-Increasing directivity of antenna by increasing can length or welding a second can to current configuration.

-Less loose wires that can radiate.

-Physically separate the function generator and op-amp used to for gain and LPF to further lower the noise floor.

**REFERENCES**

Greg, Charvat, Building a Small Radar System Capable of Sensing Range, Doppler and Synthetic Aperture Radar Imaging 2011. http://ocw.mit.edu/resources/res-ll-003-build-a-small-radar-system-capable-of-sensing-range-doppler-and-synthetic-aperture-radar-imaging-january-iap-2011/

N. Marcuvitz, *Waveguide Handbook*, MIT Radiation Laboratory Series, New York, 1951

Bassem R. Mahafza, Radar Signal Analysis and Processing using MATLAB, CRC Press, Florida 2009

**APPENDIX 1**

%daqradar.m

% realtime acquisition for laptop radar

function varargout = daqradar\_040212(varargin)

gui\_Singleton = 1;

gui\_State = struct('gui\_Name', mfilename, ...

'gui\_Singleton', gui\_Singleton, ...

'gui\_OpeningFcn', @daqradar\_040212\_OpeningFcn, ...

'gui\_OutputFcn', @daqradar\_040212\_OutputFcn, ...

'gui\_LayoutFcn', [] , ...

'gui\_Callback', []);

if nargin && ischar(varargin{1})

gui\_State.gui\_Callback = str2func(varargin{1});

end

if nargout

[varargout{1:nargout}] = gui\_mainfcn(gui\_State, varargin{:});

else

gui\_mainfcn(gui\_State, varargin{:});

end

% End initialization code

% --- Executes just before daqradar is made visible.

function daqradar\_040212\_OpeningFcn(hObject, eventdata, handles, varargin)

% This function has no output args, see OutputFcn.

% hObject handle to figure

% eventdata

% handles structure with handles and user data (see GUIDATA)

% varargin command line arguments to daqradar (see VARARGIN)

% Choose default command line output for daqradar

handles.output = hObject;

% I N P U T S

%Selecting a source.

handles.adaptor = 'winsound';

handles.id = 0;

handles.chanup = 1;

handles.chandown = 2;

handles.samplesPerTrigger = 8192;

handles.FFTlength=8192;

handles.samplesPerChirp=56;

handles.sampleRate = 48000;

handles.numTraces = 30; % number of traces to show in the intensity plot.

handles.HanningToggle = 0; %Hanning window off by default

handles.RangingToggle =0; %Not ranging by default

handles.ChirpToggle=0; %Faster trigger for edge extraction

handles.ParseNumber=1; %One parse display by default

handles.AverageParses=0; %Averaging off by default

% F I G U R E

%set(handles.figure1,'Color'.get

%Time Domain Axis

axes(handles.axes1);

handles.hLine1 = plot(zeros(1,handles.samplesPerTrigger)');

set(handles.hLine1,'Color', [.1 .1 0.5]);

set(handles.axes1,'Color',[235/255 255/255 235/255])

set(handles.axes1,'XGrid','on','YGrid','on')

t=title('Data','Color',[.05 .05 .25],'FontWeight','Bold','FontSize',9);

xlabel('Time (s)','FontSize',8);

ylabel('Voltage (V)','FontSize',8);

axes(handles.axes5);

handles.hLine5 = plot(zeros(1,handles.samplesPerTrigger)');

set(handles.hLine5,'Color', [.1 .1 0.5]);

set(handles.axes5,'Color',[235/255 255/255 235/255])

set(handles.axes5,'XGrid','on','YGrid','on')

t=title('Sync','Color',[.05 .05 .25],'FontWeight','Bold','FontSize',9);

xlabel('Time (s)','FontSize',8);

ylabel('Voltage (V)','FontSize',8);

%Frequency Domain Axis

axes(handles.axes2);

if(handles.RangingToggle==1)

set(handles.axes2,'Visible','off');

set(handles.hLine2,'Visible','off');

end

handles.hLine2 = plot(zeros(1,handles.samplesPerTrigger/2)');

set(handles.hLine2,'Color', [.1 0.5 .1]);

set(handles.axes2,'Color',[235/255 255/255 235/255])

set(handles.axes2,'XGrid','on','YGrid','on')

t=title('Frequency Domain','Color',[.05 0.25 .05],'FontWeight','Bold','FontSize',9);

xlabel('Frequency (Hz)','FontSize',8);

ylabel('Magnitude (dB)','FontSize',8);

axes(handles.axes3);

set(handles.axes3,'View',[90 -90]);

set(handles.axes3,'Color',[255/255 255/255 255/255]);

grid(handles.axes3,'on');

h = get(handles.axes3,'title');

set(h,'string','Intensity Plot','FontWeight','Bold','Color',[.25 .05 .05],'FontSize',9);

h = get(handles.axes3,'ylabel');

set(h,'string','Frequency (Hz)','FontSize',8);

h = get(handles.axes3,'zlabel');

set(h,'string','Magnitude (dB)','FontSize',8);

axes(handles.axes6);

handles.hLine6 = plot(zeros(1,handles.samplesPerTrigger)');

%handles.hLine6b = plot(zeros(1,handles.samplesPerTrigger)');

if(handles.RangingToggle==0)

set(handles.axes6,'Visible','off');

set(handles.hLine6,'Visible','off');

%set(handles.hLine6b,'Visible','off');

end

set(handles.hLine6,'Color', [.1 0.5 .1]);

%set(handles.hLine6b,'Color', [0 0 0]);

set(handles.axes6,'Color',[235/255 255/255 235/255])

set(handles.axes6,'XGrid','on','YGrid','on')

t=title('First Parse','Color',[.05 .05 .25],'FontWeight','Bold','FontSize',9);

xlabel('Time (s)','FontSize',8);

ylabel('Voltage (V)','FontSize',8);

axes(handles.axes7);

handles.hLine7 = plot(zeros(1,handles.samplesPerTrigger)');

if(handles.RangingToggle==0)

set(handles.axes7,'Visible','off');

set(handles.hLine7,'Visible','off');

end

set(handles.hLine7,'Color', [1 0 0]);

set(handles.axes7,'Color',[235/255 255/255 235/255])

set(handles.axes7,'XGrid','on','YGrid','on')

t=title('Subtracted Frames','Color',[.05 .05 .25],'FontWeight','Bold','FontSize',9);

xlabel('Time (s)','FontSize',8);

ylabel('Voltage (V)','FontSize',8);

axes(handles.axes8);

handles.hLine8 = plot(zeros(1,handles.samplesPerTrigger/2)');

if(handles.RangingToggle==0)

set(handles.axes8,'Visible','off');

set(handles.hLine8,'Visible','off');

end

set(handles.hLine8,'Color', [.1 0.5 .1]);

set(handles.axes8,'Color',[235/255 255/255 235/255])

set(handles.axes8,'XGrid','on','YGrid','on')

t=title('FFT','Color',[.05 0.25 .05],'FontWeight','Bold','FontSize',9);

xlabel('Frequency (Hz)','FontSize',8);

ylabel('Magnitude (dB)','FontSize',8);

set(hObject,'RendererMode','Manual') % If you don't do this, the surface plot

set(hObject,'Renderer','OpenGL') % will draw VERY slowly.

set(handles.poSampleRate,'String',[{'48000'},{'44100'},{'22000'},{'8000'}]);

set(handles.poBufferSize,'String',[{'8192'},{'4096'},{'1024'},{'512'}]);

set(handles.poFFTSize,'String',[{'8192'},{'4096'},{'1024'},{'512'}]);

set(handles.rbOnePulse,'Enable','off');

set(handles.rbTwoPulse,'Enable','off');

set(handles.cbAverageParses,'Enable','off');

ai=localSetupAI(handles);

handles.ai = ai;

% Update handles structure

guidata(hObject, handles);

localStartAI(ai);

function localStartAI(ai)

% S T A R T A I

start(ai);

trigger(ai);

%end localStartAI

function localStopAI(ai)

% S T O P A I

stop(ai);

delete(ai);

% end localStopAI

function ai=localSetupAI(handles)

% S E T U P T H E A N A L O G I N P U T

% Define object and add channels

ai = analoginput(handles.adaptor, handles.id);

addchannel(ai, handles.chanup);

addchannel(ai, handles.chandown);

% Configure the callback to update the display.

set(ai, 'TimerFcn', @localfftShowData);

% Configure the analog input object.

set(ai, 'SampleRate', handles.sampleRate);

% Configure the analog input object to trigger manually twice.

% We do this because we are using peekdata to acquire the data in

% a timer callback function.

% The first trigger will fill the buffer with handles.samplesPerTrigger

% number of samples. We'll know we have enough samples to start

% processing data when the analog input object's SamplesAvailable property

% is equal to handles.samplesPerTrigger.

% The analog input object will then wait for

% another manual trigger, and while it is waiting the object will still be

% in its running state, which means the timer event will run. To keep the

% object in the running state, we need only never manually trigger this

% second trigger.

% Had we set the TriggerRepeat to 0, the analog input object would stop

% after the first trigger and the timer functions would stop running.

% From waterfall.m by J. Lee

set(ai, 'SamplesPerTrigger', handles.samplesPerTrigger);

disp(handles.samplesPerTrigger);

set(ai, 'TriggerRepeat', 1);

set(ai, 'TriggerType', 'manual');

% Initialize callback parameters. The TimerAction is initialized

% after figure has been created.

set(ai, 'TimerPeriod', 0.01);

set(ai, 'BufferingConfig',[handles.samplesPerTrigger\*2,20]);

% Initialize time and frequency plots with lines of y=0

d=zeros(1,handles.samplesPerTrigger);

time = 1:handles.samplesPerTrigger;

f=1:handles.samplesPerTrigger/2;

mag=zeros(1,handles.samplesPerTrigger/2);

% Store state information in the analog input objects UserData area.

data.storedFFTsIndex = 1;

data.plotSurf = 0;

data.ai = ai;

data.getdata = [d time];

data.daqfft = [f mag];

data.handle = [];

data.figureHandles = handles;

%data.view = [103 10];

%data.rotateStep = 4;

data.counter = 0;

% Set the object's UserData to data.

set(data.ai, 'UserData', data);

%end localSetupAI(handles)

% S E T U P T H E A N A L O G I N P U T

% --- Outputs from this function are returned to the command line.

function varargout = daqradar\_040212\_OutputFcn(hObject, eventdata, handles)

% varargout cell array for returning output args (see VARARGOUT);

% hObject handle to figure

% eventdata reserved - to be defined in a future version of MATLAB

% handles structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure

varargout{1} = handles.output;

% --- Executes on button press in pbExit or when you press

% the figure close 'X' button (I set this function to

% the figures CloseRequestFcn in GUIDE)

function pbExit\_Callback(hObject, eventdata, handles)

% hObject handle to pbExit (see GCBO)

% eventdata reserved - to be defined in a future version of MATLAB

% handles structure with handles and user data (see GUIDATA)

localStopAI(handles.ai);

closereq;

% --- Executes during object creation, after setting all properties.

function poSampleRate\_CreateFcn(hObject, eventdata, handles)

% hObject handle to poSampleRate (see GCBO)

% eventdata reserved - to be defined in a future version of MATLAB

% handles empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.

% See ISPC and COMPUTER.

if ispc

set(hObject,'BackgroundColor','white');

else

set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));

end

% C H A N G E T H E S A M P L E R A T E

% --- Executes on selection change in poSampleRate.

function poSampleRate\_Callback(hObject, eventdata, handles)

% hObject handle to poSampleRate

% eventdata

% handles structure with handles and user data

% First, stop and delete the current analog input object

localStopAI(handles.ai);

% Extract the new samplerate.

v=get(handles.poSampleRate,'Value');

s=get(handles.poSampleRate,'String');

handles.sampleRate = str2num(s{v});

% Create a new analog input with the new sample rate.

handles.ai = localSetupAI(handles);

% Update handles structure

guidata(hObject, handles);

% Restart the analog input

localStartAI(handles.ai);

% end poSampleRate\_Callback

% C H A N G E T H E SAMPLES/TRIGGER

% --- Executes on selection change in poBufferSize.

function poBufferSize\_Callback(hObject, eventdata, handles)

% First, stop and delete the current analog input object

localStopAI(handles.ai);

% Extract the new size.

v=get(handles.poBufferSize,'Value');

s=get(handles.poBufferSize,'String');

handles.samplesPerTrigger = str2num(s{v});

% Create a new analog input with the new size.

handles.ai = localSetupAI(handles);

% Update handles structure

guidata(hObject, handles);

% Restart the analog input

localStartAI(handles.ai);

% end poSampleRate\_Callback

% C H A N G E T H E FFT size

% --- Executes on selection change in poFFTSize.

function poFFTSize\_Callback(hObject, eventdata, handles)

% hObject handle to poFFTSize (see GCBO)

% eventdata reserved - to be defined in a future version of MATLAB

% handles structure with handles and user data (see GUIDATA)

% First, stop and delete the current analog input object

localStopAI(handles.ai);

% Extract the new size.

v=get(handles.poBufferSize,'Value');

s=get(handles.poBufferSize,'String');

handles.FFTlength = str2num(s{v});

% Create a new analog input with the new size.

handles.ai = localSetupAI(handles);

% Update handles structure

guidata(hObject, handles);

% Restart the analog input

localStartAI(handles.ai);

% end poFFTSize\_Callback

% --- Executes during object creation, after setting all properties.

function poFFTSize\_CreateFcn(hObject, eventdata, handles)

% hObject handle to poFFTSize (see GCBO)

% eventdata reserved - to be defined in a future version of MATLAB

% handles empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.

% See ISPC and COMPUTER.

if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))

set(hObject,'BackgroundColor','white');

end

% \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

% Calculate the fft of the data. (Copied from demoai\_fft.m)

function [f, mag] = localDaqfft(data,Fs,blockSize)

% Calculate the fft of the data.

%xFFT = fft(data(:,2)); %channel 2

%xFFT = fft(data(:,1)); %channel 1

xFFT = fft(data);

xfft = abs(xFFT);

% Avoid taking the log of 0.

index = find(xfft == 0);

xfft(index) = 1e-17;

mag = 20\*log10(xfft);

mag = mag(1:floor(blockSize/2));

f = (0:length(mag)-1)\*Fs/blockSize;

f = f(:);

% \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

% Update the plot. This routine is a Timer callback, it is called

% automatically at a preset time interval. See line 144 for where

% this routine is assigned as a callback

function localfftShowData(obj,event)

if (get(obj,'SamplesAvailable') >= obj.SamplesPerTrigger)

% Get the handles.

data = obj.UserData;

global x;

global f;

global mag;

global parse;

global parsediff;

global ind;

handles = data.figureHandles;

% Execute a peekdata.

x = peekdata(obj, obj.SamplesPerTrigger);

% Applying Hanning window if checked

if(handles.HanningToggle==1 && handles.ChirpToggle==0)

win=hanning(get(obj, 'SamplesPerTrigger'));

x(:,2)=win(:).\*x(:,2); %chandown

%disp('Hello')

end

Fs = obj.SampleRate;

blockSize = obj.SamplesPerTrigger;

% disp(sprintf('FS: %d',Fs));

%disp(sprintf('Block Size: %d',blockSize));

%disp(sprintf('Chirp Size: %d',handles.samplesPerChirp));

% Edge Detection for Ranging

%disp(sprintf('Ranging Toggle: %d',handles.RangingToggle));

if(handles.ChirpToggle==1)

global signal;

sync=x(:,1);

signal=x(:,2);

%h= fdesign.lowpass('N,Fst,Ast', 10, 500,80, Fs);

%Hd=design(h,'cheby2');

%signal=filter(Hd,signal);

tp=.5; %clock is good enough for ~.8. Anything highers risks false edges

tn=-.5;

atp=sync(:,1)>tp;

atn=sync(:,1)<tn;

atp=atp';

atn=atn';

global risingedge;

global fallingedge;

risingedge=diff(atp)>0;

risingedge=risingedge';

fallingedge=diff(atn)>0;

fallingedge=fallingedge';

global indrising;

global indfalling;

if(risingedge==[0]) %prevent initialization break because of empty array

return;

end

indrising=find(risingedge);

indfalling=find(fallingedge);

% index position of start

% end

global fsize;

fsize=zeros(1,handles.samplesPerChirp+1);

if(indrising(1)<indfalling(1)) %first edge is rising

for k=1:1:length(indfalling)-1

fsize(:,k)=indfalling(k)-indrising(k);

end

elseif(indrising(1)>indfalling(1)) %first edge is falling

for k=2:1:length(indfalling)

fsize(:,k-1)=indfalling(k)-indrising(k-1);

end

end

fsize=fsize(find(fsize));

global chirpsize;

chirpsize=mode(fsize);

%disp(sprintf('Index: %d',length(ind)));

parse=zeros(length(fsize),mode(fsize));

global j;

for j=1:1:length(fsize)-1; %parsing data from positive slope

if(fsize(:,j)==fsize(:,j+1) && fsize(:,j)==chirpsize)

parse(j,:)=signal(indrising(j,:):indrising(j,:)+fsize(:,j)-1);

end

end

global bprune;

bprune=parse;

parse(all(parse==0,2),:)=[]; %deleting zero rows

if(size(parse,1)<2) %need at least 2 same size consecutive frames

return;

end

parse=parse';

[nrows ncols]=size(parse);

splitpoint=floor(ncols/2);

global parse1;

global parse2;

parse1=parse(:,1:splitpoint);

parse2=parse(:,splitpoint+1:2\*splitpoint);

parse1=mean(parse1,2);

parse2=mean(parse2,2);

%subtracting consecutive frames

global start;

global stop;

global pad;

global startz;

global stopz;

global padded2;

global padded1;

global time\_padded1;

global time\_padded2;

global padded\_diff;

if(handles.AverageParses==0) %no average

time\_padded1=parse(:,1)';

time\_padded2=parse(:,2)';

elseif(handles.AverageParses==1)

time\_padded1=parse1';

time\_padded2=parse2';

end

%h= fdesign.lowpass('N,F3dB', 10, 5000, Fs);

%Hd=design(h,'butter');

%time\_padded1=filter(Hd,time\_padded1);

%time\_padded2=filter(Hd,time\_padded2);

if(handles.HanningToggle==1)

win2=hanning(chirpsize)';

time\_padded1=win2.\*time\_padded1;

time\_padded2=win2.\*time\_padded2;

end

%Zero-Padding

pad=handles.FFTlength-chirpsize;

start=floor(pad/2);

stop=handles.FFTlength-(pad-start);

%stop=pad-start;

parsediff=parse(:,2)-parse(:,1);

startz=zeros(1,start);

stopz=zeros(1,handles.FFTlength-stop);

padded1=[startz time\_padded1(1:end) stopz];

padded2=[startz time\_padded2(1:end) stopz];

padded\_diff=padded2-padded1;

%disp(sprintf('Parse Ranging: %d',length(parse)));

%disp(sprintf('F Ranging: %d',length(f)));

% disp(sprintf('Mag Ranging: %d',length(mag)));

[f,mag] = localDaqfft(x(:,2),Fs,blockSize);

global f1;

global mag1;

global f2;

global mag2;

global f3;

global mag3;

global f4;

global mag4;

global f5;

global mag5;

global f6;

global mag6;

[f1,mag1]=localDaqfft(parse(:,1),Fs,chirpsize); % FFT of first parsed frame

[f2,mag2]=localDaqfft(parse(:,2),Fs,chirpsize); % FFT of second parsed frame

[f3,mag3]=localDaqfft(parsediff,Fs,chirpsize); %FFT of difference

[f4,mag4]=localDaqfft(padded1,Fs,handles.FFTlength); %padded FFT 1

[f5,mag5]=localDaqfft(padded2,Fs,handles.FFTlength); %padded FFT 2

[f6,mag6]=localDaqfft(padded\_diff,Fs,handles.FFTlength); %FFT of padded diff

if(handles.ParseNumber==1)

f6=f4;

mag6=mag4;

end

else %not ranging

[f,mag] = localDaqfft(x(:,2),Fs,blockSize);

%disp(size(f))

disp(sprintf('F Regular: %d',length(f)));

disp(sprintf('Mag Regular: %d',length(mag)));

end

%Dynamically modify Analog Chan 1 (chanup=sync)

maxX=max(x(:,1));

minX=min(x(:,1));

yax3=get(handles.axes5,'YLim');

yax3(1)=minX-.0001;

yax3(2)=maxX+.0001;

set(handles.axes5,'YLim',yax3)

set(handles.axes5,'XLim',[0 (obj.SamplesPerTrigger-1)/obj.SampleRate])

% Dynamically modify Analog Chan 2 (chandown=data)

maxX2=max(x(:,2));

minX2=min(x(:,2));

yax1=get(handles.axes1,'YLim');

yax1(1)=minX2 - .0001; % need to subtract a value to make sure yax(1) never equals yax(2)

yax1(2)=maxX2 + .0001;

set(handles.axes1,'YLim',yax1)

set(handles.axes1,'XLim',[0 (obj.SamplesPerTrigger-1)/obj.SampleRate])

% Dynamically modify Frequency axis as we go.

maxF=max(f);

minF=min(f);

xax=get(handles.axes2,'XLim');

xax(1)=minF;

xax(2)=maxF;

set(handles.axes2,'XLim',xax)

% Dynamically modify Magnitude axis as we go.

maxM=max(mag);

minM=min(mag);

yax2=get(handles.axes2,'YLim');

yax2(1)=minM - .0001;

yax2(2)=maxM + .0001;

set(handles.axes2,'YLim',yax2)

if(handles.ChirpToggle==1)

%Dynamically modify Parse Frame

maxX3=max(parse(:,1));

minX3=min(parse(:,1));

yax4=get(handles.axes6,'YLim');

yax4(1)=minX3-.0001;

yax4(2)=maxX3+.0001;

set(handles.axes6,'YLim',yax4)

set(handles.axes6,'XLim',[0 (chirpsize-1)/obj.SampleRate])

%disp(obj.SamplesPerTrigger);

%Dynamically modify Difference Frame

maxX4=max(parsediff);

minX4=min(parsediff);

yax5=get(handles.axes7,'YLim');

yax5(1)=minX4-.0001;

yax5(2)=maxX4+.0001;

set(handles.axes7,'YLim',yax5)

set(handles.axes7,'XLim',[0 (chirpsize-1)/obj.SampleRate])

% Dynamically modify Frequency axis DifferenceFFT

maxF3=max(f6);

minF3=min(f6);

xax2=get(handles.axes8,'XLim');

xax2(1)=minF3;

xax2(2)=maxF3;

set(handles.axes8,'XLim',xax2)

% Dynamically modify Mag axis Difference FFT

maxM3=max(mag6);

minM3=min(mag6);

yax3=get(handles.axes8,'YLim');

yax3(1)=minM3 - .0001;

yax3(2)=maxM3 + .0001;

set(handles.axes8,'YLim',yax3)

end

% Update the line plots.

set(handles.hLine1, 'XData', [0:(obj.SamplesPerTrigger-1)]/obj.SampleRate, 'YData', x(:,2));

set(handles.hLine5, 'XData', [0:(obj.SamplesPerTrigger-1)]/obj.SampleRate, 'YData', x(:,1));

set(handles.hLine2, 'XData', f(:,1), 'YData', mag(:,1));

if(handles.ChirpToggle==1)

set(handles.hLine6, 'XData', [0:(chirpsize-1)]/obj.SampleRate, 'YData', parse(:,1));

set(handles.hLine7, 'XData', [0:(chirpsize-1)]/obj.SampleRate, 'YData', parsediff);

set(handles.hLine8, 'XData', f6, 'YData', mag6);

%set(handles.hLine6b, 'XData', [0:(handles.samplesPerChirp-1)]/obj.SampleRate, 'YData', parse(:,2));

end

% Find the frequency at which the max signal strength is at.

%Store FFT into array for intensity plot

if(handles.ChirpToggle==1)

[yrange,rangeindex] = max(mag6);

set(handles.tFreq,'String',sprintf('%4.3d Hz',f6(rangeindex)));

data.storedFFTs(data.storedFFTsIndex,:) = mag6';

elseif (handles.RangingToggle==0)

[ymax,maxindex] = max(mag);

set(handles.tFreq,'String',sprintf('%4.3d Hz',f(maxindex)));

data.storedFFTs(data.storedFFTsIndex,:) = mag';

end

% This circular shift is used so that when we display the 3D plot, the

% newest FFT will appear in 'front' and the oldest in 'back'.

% To understand this, note how the plotting routines are using this fftOrder

% array to reorder the FFTs stored in data.storedFFTs and also note

% how data.storedFFTsIndex is used to store FFTs in data.storedFFTs.

%

fftOrder = 1:handles.numTraces;

fftOrder = circshift(fftOrder,[ 1 -data.storedFFTsIndex ]);

data.storedFFTsIndex = data.storedFFTsIndex + 1;

if (data.storedFFTsIndex > handles.numTraces)

data.storedFFTsIndex = 1;

data.plotSurf = 1; % Indicates a full history is stored.

end

% Update the surface plot if we have a full history.

if(data.plotSurf)

cla(handles.axes3);

data.view = [180 -90];

if(handles.RangingToggle==0)

data=localClassic(handles,data,f,fftOrder);

elseif(handles.ChirpToggle==1)

data=localClassic(handles,data,f6,fftOrder);

end

end

set(data.ai, 'UserData', data);

drawnow;

end

% I N T E N S I T Y G R A P H

function data=localClassic(handles,data,f,fftOrder)

[X,Y] = meshgrid(1:handles.numTraces,f(1:end));

surf(X,Y,data.storedFFTs(fftOrder,:)','parent',handles.axes3,'EdgeColor','none');

set(handles.axes3,'XLim',[1 handles.numTraces],'YLim',[0 f(end)])

shading(handles.axes3,'interp');

set(handles.axes3,'View',data.view)

% --- Executes during object creation, after setting all properties.

function poPlotType\_CreateFcn(hObject, eventdata, handles)

% hObject handle to poPlotType (see GCBO)

% eventdata reserved - to be defined in a future version of MATLAB

% handles empty - handles not created until after all CreateFcns called

if ispc

set(hObject,'BackgroundColor','white');

else

set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));

end

% --- Executes during object creation, after setting all properties.

function poBufferSize\_CreateFcn(hObject, eventdata, handles)

% hObject handle to poBufferSize (see GCBO)

% eventdata reserved - to be defined in a future version of MATLAB

% handles empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.

% See ISPC and COMPUTER.

if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))

set(hObject,'BackgroundColor','white');

end

% --- Executes on button press in cbHanningToggle.

function cbHanningToggle\_Callback(hObject, eventdata, handles)

% hObject handle to cbHanningToggle (see GCBO)

% eventdata reserved - to be defined in a future version of MATLAB

% handles structure with handles and user data (see GUIDATA)

% hObject handle to pbHanningToggle (see GCBO)

% eventdata reserved - to be defined in a future version of MATLAB

% handles structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of pbHanningToggle

% First, stop and delete the current analog input object

localStopAI(handles.ai);

%

toggle=get(handles.cbHanningToggle,'Value');

%disp(toggle)

handles.HanningToggle = toggle;

% Create a new analog input with the new state.

handles.ai = localSetupAI(handles);

% Update handles structure

guidata(hObject, handles);

% Restart the analog input

localStartAI(handles.ai);

% end cbHanningToggle\_Callback

% --- Executes on button press in cb\_ranging.

function cb\_ranging\_Callback(hObject, eventdata, handles)

% hObject handle to cb\_ranging (see GCBO)

% eventdata reserved - to be defined in a future version of MATLAB

% handles structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of cb\_ranging

localStopAI(handles.ai);

ranging=get(handles.cb\_ranging,'Value');

handles.RangingToggle=ranging;

if(ranging==1)

set(handles.text\_chirpf,'Enable','on'); %enable text Box

set(handles.rbOnePulse,'Enable','on');

set(handles.rbTwoPulse,'Enable','on');

set(handles.cbAverageParses,'Enable','on');

set(handles.axes6,'Visible','on');

set(handles.axes7,'Visible','on');

set(handles.hLine6,'Visible','on');

set(handles.hLine7,'Visible','on');

set(handles.axes8,'Visible','on');

set(handles.hLine8,'Visible','on');

%set(handles.hLine6b,'Visible','on');

set(handles.axes2,'Visible','off');

set(handles.hLine2,'Visible','off');

else

set(handles.text\_chirpf,'Enable','off');

set(handles.rbOnePulse,'Enable','off');

set(handles.rbTwoPulse,'Enable','off');

set(handles.cbAverageParses,'Enable','off');

set(handles.axes6,'Visible','off');

set(handles.axes7,'Visible','off');

set(handles.hLine6,'Visible','off');

set(handles.hLine7,'Visible','off');

%set(handles.hLine6b,'Visible','off');

set(handles.axes2,'Visible','on');

set(handles.hLine2,'Visible','on');

set(handles.axes8,'Visible','off');

set(handles.hLine8,'Visible','off');

handles.ChirpToggle=0;

end

handles.ai = localSetupAI(handles);

% Update handles structure

guidata(hObject, handles);

% Restart the analog input

localStartAI(handles.ai);

% end cb\_ranging\_Callback

function text\_chirpf\_Callback(hObject, eventdata, handles)

% hObject handle to text\_chirpf (see GCBO)

% eventdata reserved - to be defined in a future version of MATLAB

% handles structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of text\_chirpf as text

% str2double(get(hObject,'String')) returns contents of text\_chirpf as a double

sample\_rate=handles.sampleRate;

%disp(sample\_rate);

localStopAI(handles.ai);

chirp\_freq=str2double(get(handles.text\_chirpf,'String'));

chirpSamples=round(1/(2\*chirp\_freq)\*sample\_rate);

%num\_samples=chirpSamples\*6; %new window size want at least 2 positive edge for difference

%disp(num\_samples)

%handles.samplesPerTrigger =num\_samples ;

handles.samplesPerChirp=chirpSamples;

handles.ChirpToggle=1; %faster trigger than RangingToggle

disp(sprintf('ChirpToggle inside text\_chripf: %d',handles.ChirpToggle));

% Create a new analog input with the new size.

handles.ai = localSetupAI(handles);

% Update handles structure

guidata(hObject, handles);

% Restart the analog input

localStartAI(handles.ai);

% end text\_chirpf\_Callback

% --- Executes during object creation, after setting all properties.

function text\_chirpf\_CreateFcn(hObject, eventdata, handles)

% hObject handle to text\_chirpf (see GCBO)

% eventdata reserved - to be defined in a future version of MATLAB

% handles empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.

% See ISPC and COMPUTER.

if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))

set(hObject,'BackgroundColor','white');

end

% --- Executes on button press in cbAverageParses.

function cbAverageParses\_Callback(hObject, eventdata, handles)

% hObject handle to cbAverageParses (see GCBO)

% eventdata reserved - to be defined in a future version of MATLAB

% handles structure with handles and user data (see GUIDATA)

localStopAI(handles.ai);

average=get(handles.cbAverageParses,'Value');

handles.AverageParses=average;

% Create a new analog input with the new state.

handles.ai = localSetupAI(handles);

% Update handles structure

guidata(hObject, handles);

% Restart the analog input

localStartAI(handles.ai);

% --- Executes when selected object is changed in uipanel6.

function uipanel6\_SelectionChangeFcn(hObject, eventdata, handles)

% hObject handle to the selected object in uipanel6

% eventdata structure with the following fields (see UIBUTTONGROUP)

% EventName: string 'SelectionChanged' (read only)

% OldValue: handle of the previously selected object or empty if none was selected

% NewValue: handle of the currently selected object

% handles structure with handles and user data (see GUIDATA)

localStopAI(handles.ai);

%

parsenumber=get(eventdata.NewValue,'tag');

switch parsenumber

case 'rbOnePulse'

handles.ParseNumber = 1;

case 'rbTwoPulse'

handles.ParseNumber = 0;

end

% Create a new analog input with the new state.

handles.ai = localSetupAI(handles);

% Update handles structure

guidata(hObject, handles);

% Restart the analog input

localStartAI(handles.ai);

**APPENDIX 2.**

**Bill Of Material for Microwave Components, ADC, and ICs**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Description | Supplier Part # | Manufacturer | Quantity | Unit Price | Subtotal | Supplier |
| Monolithic Function Generator | [XR2206CP](http://www.jameco.com/webapp/wcs/stores/servlet/Product_10001_10001_34972_-1) | Exar | 4 | 6.49 | 25.96 | Jameco |
| Voltage Controlled Oscillator | Zx95-3490C+ | Mini-circuits | 1 | 40.95 | 40.95 | Mini-circuits |
| Quad-Op Amp | MAX414CPD+-ND | Maxim | 1 | 5.86 | 5.86 | Digikey |
| 3dB SMA Attenuator | VAT-3+ | Mini-circuits | 1 | 11.95 | 11.95 | Mini-circuits |
| Frequency Mixer | zx05-43MH-S+ | Mini-circuits | 1 | 46.45 | 46.45 | Mini-circuits |
| SMA M to SMA M | SM-SM50+ | Mini-circuits | 4 | 8.95 | 35.8 | Mini-circuits |
| SMA M-M 6" Cable | 086-12SM+ | Mini-circuits | 3 | 9.65 | 28.95 | Mini-circuits |
| LNA | zx60-362LN-S+ | Mini-circuits | 2 | 44.95 | 89.9 | Mini-Circuits |
| Power Splitter | ZX10-2-722-S+ | Mini-circuits | 1 | 31.95 | 31.95 | Mini-Circuits |
| ADC Inport Deluxe | INPDEL-I1-X1 | Xitel | 1 | 11.89 | 11.89 | Amazon |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  | Total | 317.66 |  |