**Mei Yang**

**4/24/2018**

**AMRUPT, Spring 2018**

# Goals

Waveform reconstruction continued, test hardware, Matlab to GNU radio transfer (?) and focus on the simulation aspect in particular

# Problem

Part of the simulation is to simulate the RF signal and the AOA calculation. Therefore, it’s important to simulate the hardware of the system. The entire simulation process will generate RF sine waves, down converts them to IF, quantizes the signals to simulate the ADC stage, and then uses the DF algorithm to calculate the AOA. Last week, I tried to install the Hardware Support Package with Matlab and failed in the end. I also focused on writing code this week because I didn’t have my laptop to verify my code in lab. Therefore, I wrote untested code for signal generation, down-converting, and quantization.

# General approach

The signal generation code I have right now just sets frequency as an user input parameter so that once we’ve decided on a frequency value, I can easily set it. I will also add a plotting function to it so that the signal generated can be visualized. Matlab provides a very detailed explanation and tootlbox on implementing a Digital Down Converter, and the sample code is provided below. For the quantizer, for the time being, I have used a partition length of 20 intervals for simplicity. As I test the program this following week, I will adjust the parameters as I go.

# Code-level problems and solutions, and empirical testing

# Singal Generation

function test = FirstTestScript(fs,T,edge,type,f,opt)

%fs sampling frequency

%T total sig length

%edge decay parameter

%f modulation frequency

%amplitude is 1

if nargin==4

f=0;

opt=[16\*edge/(5\*T),64\*edge/(5\*T)];

elseif nargin==5

opt=[16\*edge/(5\*T),64\*edge/(5\*T)];

end

t=-T/2:1/fs:T/2;

%specifies signal length

sig=(T/8/edge)^2;

switch type

case {'guassian'} % guassian

y=exp(-(t).^2/sig);

case {'square'} % square

y=ones(size(t));

case {'triangle'} % triangle

y=(t+T/2).\*(t<0)-(t-T/2).\*(t>=0);

case {'monocycle'} % gaussian monocycle

y=2\*t./sig.\*exp(-(t).^2/sig);

case {'biexponential'} % biexponential

y=exp(-abs(t)\*8\*edge/T);

case {'mexican hat'} % gaussian second deriviative

z=t./sqrt(0.75\*sig);

y=sqrt(1/2\*pi).\*(1-z.^2).\*exp(-z.^2/2);

case {'sinc'} % sinc function

y=sinc(2\*pi\*edge\*16.\*t/(5\*T));

case {'double sinc'} % bandlimited function from two sinc functions

y=opt(1)\*sinc(2\*opt(1).\*t)-opt(2)\*sinc(2\*opt(2).\*t);

case {'sinc squared'} % sinc squared function

y=sinc(2\*pi\*edge\*16.\*t/(5\*T)).^2;

case {'sweep'} % frequency sweep

theta=(opt(1)+(opt(2)-opt(1))/T).\*(t+T/2);

y=real(exp(j\*(theta.\*(t+T/2)-pi/2)));

case {'windowed sweep'} % windows frequency sweep

theta=(opt(1)+(opt(2)-opt(1))/T).\*(t+T/2);

y=real(exp(j\*(theta.\*(t+T/2)-pi/2)));

c=length(y);

edge=min(1,edge);

edge=max(0,edge);

w=hamming(ceil(c\*(1-edge)));

w2=[w(1:ceil(length(w)/2));ones(c-length(w),1);w(ceil(length(w)/2)+1:end)]';

y=w2.\*y;

otherwise

error('invalid pulse type');

end

if f~=0

y = y.\*cos(2\*pi\*t\*f);

end

%modulation

test=y./max(abs(y));

Down-converting

Fs = 2.4e6; FrameSize = 768;

hsine = dsp.SineWave( ...

'Frequency', 2.4e6\*5/24, ...

'SampleRate', Fs, ...

'Method', 'Trigonometric function', ...

'SamplesPerFrame', FrameSize);

hnco = dsp.NCO( ...

'PhaseIncrementSource', 'Property', ...

'PhaseIncrement', int32((5/24) \*2^32), ...

'PhaseOffset', int16(0), ...

'NumDitherBits', 14, ...

'NumQuantizerAccumulatorBits', 18, ...

'Waveform', 'Complex exponential', ...

'SamplesPerFrame', FrameSize, ...

'AccumulatorDataType', 'Custom', ...

'CustomAccumulatorDataType', numerictype([],32), ...

'OutputDataType', 'Custom', ...

'CustomOutputDataType', numerictype([],20,18));

M1 = 64;

hcicdec = dsp.CICDecimator( ...

'DecimationFactor', M1, ...

'NumSections', 5, ...

'FixedPointDataType', 'Minimum section word lengths', ...

'OutputWordLength', 20);

gsmcoeffs; % Read the CFIR and PFIR coeffs

M2 = 2;

hcfir = dsp.FIRDecimator(M2, cfir, ...

'CoefficientsDataType', 'Custom', ...

'CustomCoefficientsDataType', numerictype([],16), ...

'FullPrecisionOverride', false,...

'OutputDataType', 'Custom', ...

'CustomOutputDataType', numerictype([],20,-12));

M3 = 2;

hpfir = dsp.FIRDecimator(M3, pfir, ...

'CoefficientsDataType', 'Custom', ...

'CustomCoefficientsDataType', numerictype([],16), ...

'FullPrecisionOverride',false, ...

'OutputDataType', 'Custom', ...

'CustomOutputDataType', numerictype([],20,-12));

hfirsrc = dsp.FIRRateConverter(4, 3, fir1(31,0.25),...

'CoefficientsDataType', 'Custom', ...

'CustomCoefficientsDataType', numerictype([],12), ...

'FullPrecisionOverride',false, ...

'OutputDataType', 'Custom', ...

'CustomOutputDataType', numerictype([],24,-12));

% Create a fi object of specified numerictype to act as a data type

% conversion for the sine output.

gsmsig = fi(zeros(768,1),true,14,13);

%%

% Create a fi object of specified numerictype to store the fixed-point

% mixer output.

mixsig = fi(zeros(768,1),true,20,18);

for ii = 1:100

gsmsig(:) = step(hsine); % GSM signal

ncosig = step(hnco); % NCO signal

mixsig(:) = gsmsig.\*ncosig; % Digital mixer

% CIC filtering and compensation

ycic = step(hcfir, step(hcicdec, mixsig));

% Programmable FIR and sample-rate conversion

yrcout = step(hfirsrc, step(hpfir, ycic));

% Frequency and time-domain plots

step(hts1, real(yrcout));

step(hts2, imag(yrcout));

step(hss1, ncosig);

step(hss2, ycic);

end

Quantizer

t = [0:.1:2\*pi]; % Times at which to sample the signal

sig = sin(t); % Original signal, a sine wave; later replaced with the generated signals when testing

partition = [-1:.1:1]; % Length 11, to represent 12 intervals

codebook = [-1:.1:1]; % Length 12, one entry for each interval

[index,quants] = quantiz(sig,partition,codebook); % Quantize.

plot(t,sig,'x',t,quants,'.')

legend('Original signal','Quantized signal');

axis([-.2 7 -1.2 1.2])

# Planned Course of Action

# Is it possible for me to get a separate SDR-RTL for the Matlab package? I don’t want to crash our existing hardware setup because I think the installation process is prune to a lot problems and I don’t want to ruin the setup we have now;

# Re-try to do the setup with a new SDR-RTL

# Test the existing code blocks and continue with the DF algorithm part of the simulation (this part includes signal detection, frequency calculation, phase calculation, AOA calculation)

# Resources and relevant Forum Posts

# Matlab Simulation Model: <https://pdfs.semanticscholar.org/66f7/d0d7e1bed27acc37b5721f4abe649f9a053e.pdf>

# Matlab Digital Down Converter:

# <https://www.mathworks.com/help/dsp/examples/design-and-analysis-of-a-digital-down-converter.html>

# RTL-SDR Support Package Hardware Setup: <http://www.mathworks.com/help/supportpkg/rtlsdrradio/ug/support-package-hardware-setup.html#bunsvm7-34>