TX Symbol Modulation, Filtering and PSD

Table of Contents

In this lab you will learn to:

- Create a sequence of QAM symbols
- Upsample and modulate the symbols with a raised cosine filter
- Measure the TX and RX PSD
- Continuously update the RX PSD to create a simple spectrum analyzer

Submissions: Run the lab with Pluto devices with either one or two hosts. Fill in all sections labeled `TODO`. Print and submit the PDF. Do not submit the source code.

While the lab is ideally run with one or more Pluto devices, I have pre-recorded data if you want to test your results.

```
% Set preRecorded = true if using without a Pluto and you want to run
% from pre-recorded data.  You can save new pre-recorded data with saveData
usePreRecorded = false;
saveData = false;

% Sample frequency
fsamp = 30.72e6;
```

Create Filtered QAM Symbols

First, create a set of QAM-symbols with random bits with the parameters below. Store the random bits in`b` and the QAM symbols in `s`.

```
nsym = 1024;        % number of symbols to transmit
nbitsPerSym = 4;    % number of bits per symbol
M = 2^nbitsPerSym;  % QAM order

% TODO:
%    nbits = ...
%    b = ...
%    s = qammod(b, ...);

% TODO:  Plot the TX constellation
```

Next, we will upsample the symbols so that that `nov=2` samples per symbol. Use a raised-cosine filter with the parameters below.

```
nov = 2;    % oversampling rate (i.e., number of samples / symbols)
nsamp = nsym*nov;  % number of samples
span = 8;          % filter length in number of symbols
beta = 0.25;       % filter parameter

% TODO:
%   ptx = rcosdesign(...)


% TODO:  Scale filter so it has unit norm
%   ptx = ...
```

Upsample and filter the symbols with the `filter` command to create the samples. Store the PSD in a vector `Ptx and frequency in a vector fx.`

```
% TODO:
%   s1 = upsample(...)
%   x = filter(...)
```

Next, use the `pwelch` command to compute the PSD of `x`. Store the PSD in a vector `Ptx and frequency in a vector fx.` Plot the PSD in dBW/Hz vs. frequency in MHz. Label the axes.

```
% TODO:
%    [Ptx,fx] = pwelch(...);
%    PtxdB = ...
%    plot(...);
```

Create the TX and RX objects

We will now transmit and receive the signals with the Pluto devi es. Following the initial lab, set the mode to the desired configuration (loopback, two hosts with single host, or two hosts with two hosts).

```
% TODO:  Set parameters
% Set to true for loopback, else set to false
loopback = true;

% Select to run TX and RX
runTx = true;
runRx = true;
```

Plug one or two Plutos into the USB ports of your computer. If it is correctly working, in each device the Ready light should be solid on and the LED1 light should be blinking. Next, run the following command that will detect the devices and create the TX and RX objects.

```
% clear previous instances
clear tx rx

% add path to the common directory where the function is found
addpath('..\common');
```

```
% Parameters
nsampsFrame = length(x);

% Run the creation function.  Skip this if we are using pre-recorded
% samples
if ~usePreRecorded
    [tx, rx] = plutoCreateTxRx(createTx = runTx, createRx = runRx, loopback = loopback, ...
        nsampsFrame = nsampsFrame, sampleRate = fsamp);
    centerFreq = tx.CenterFrequency;
end
```

TX and RX Data via the Pluto Device

Now TX the data x repreatedly through the Pluto device.

```
if runTx && ~usePreRecorded
    % TODO:  Use the tx.release and tx.transmitRelease commands to
    % continuously send x

end
% If not running the RX, stop the live script now
if ~runRx
    return;
end
```

On the RX Pluto device, receive the data

```
nbits = 12;               % number of ADC bits
fullScale = 2^(nbits-1);  % full scale

if ~usePreRecorded

    % TODO:  Capture data
    %   r = rx.capture(...)


    % TODO:  Scale to floating point
    %   r = ...


    % Save the data
    if saveData
        save symModSing r;
    end

else
    % Load pre-recorded data
    load symModSing;

end
```

Measuring the RX PSD

Using the `pwelch` command measure the PSD in `r`. Plot the PSD in dB/Hz vs. frequency in MHz.

```
% TODO:  Create an initla plot as before
%    [Prx,fx] = pwelch(...);
%    plot(...);
```

Create a Simple Spectrum Analyzer

A spectrum analyzer is a device that continuously measures the spectrum. We can create a simple spectrum analyzer by simply capturing the samples and updating the plot. We can do this in a MATLAB live script by:

- Initially plot the data with a command of the form `p = plot(...)`. This saves a handle to the plot `p`
- Set the axes of the plot with the `xlim` and `ylim` so they are fixed. This way the plot is not automatically rescaled on each new plot.
- Set pointers to the data with the `p.XDataSource` and `p.YDataSource` variables
- Get new data
- Updating the plot data and calling the `refreshdata` and `plotnow` commands.

Complete the following code to create a spectrum analyzer.

```
% TODO:  Create the inital plot as before
%    [Prx,fx] = pwelch(...);
%    fxMHz = fx/1e6;
%    p = plot(fxMHz, ...);

% Plot data initial time
p = plot(fxMHz, PrxdB);

% TODO:  Fix the limits to some reasonable values
%   xlim([...]);
%   ylim([...]);


% Set pointers to X and Y data
p.XDataSource = 'fxMHz';
p.YDataSource = 'PrxdB';

% Number of loops before the spectrum analyzer stops
ncaptures = 100;

% Load pre-recorded data if required
if usePreRecorded
    load symModMult;
    ncaptures = size(rdat,2);
else
    rdat = zeros(nsampsFrame, ncaptures);
end
```

```
for t = 1:ncaptures
    if ~usePreRecorded

        % TODO:  Capture data
        %    r = rx.capture(...);

        % Save data if required
        if saveData
            rdat(:,t) = r;
        end
    else
        % Load pre-recorded
        r = rdat(:,t);
    end


    % TODO:  Re-compute the PSD
    %    PrxdB = ...

    % Redraw plot
    refreshdata
    drawnow

end

% Save data
if saveData
    save symModMult rdat;
end
```