Data Streaming in MATLAB

This lab will focus on connecting MATALB to ADI transceiver platforms and streaming data. We will start will the basics of connecting to a device using System objects. Then we will perform some simple exercises with the radio and MATLAB signal processing libraries.

Checking Radio Connectivity

The first step before we can stream data to any of the ADI SDR devices is to check their connectivity. After connecting a PlutoSDR through USB to your PC, open MATLAB, and move to the folder containing the files necessary for Lab 1:

```
Command Window
>> cd C:\Seminar\lab1\
```

Next, in the command windows type the following and hit enter:

```
Command Window
>> plutoradiosetup
```

This will set up the MATLAB environment for PlutoSDR. Next in the command window type:

```
Command Window
fx >> findPlutoRadio
```

If connected, that command should return something similar to:

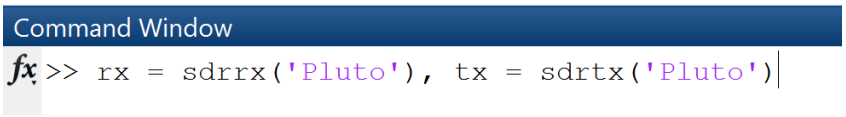
```
Command Window
>> findPlutoRadio

ans =

  struct with fields:

      RadioID: 'usb:0'
    SerialNum: '104473222a870017fdff070014ced4f484'

fx >>
```

Each Pluto device will have a unique Serial Number and a unique USB enumeration for a given machine. Multiple Plutos can be plugged into the same machine and their RadioIDs will enumerate automatically.

MATLAB System Objects for SDR Devices

Each ADI SDR device will have a unique associated System object. System objects are special classes inside MATLAB which share certain procedural methods and APIs, This makes them useful as an interface for hardware, or any data structure that requires state and has specific associated information.

In your MATLAB command line, type the following and hit enter:



This will initialize both a receive (rx) and transmit (tx) System object for a single PlutoSDR device.

This will output something like:

```
Command Window
>> rx = sdrrx('Pluto'),tx = sdrtx('Pluto')

rx =

  comm.SDRRxPluto with properties:

   Main
            DeviceName: 'Pluto'
               RadioID: 'usb:0'
        CenterFrequency: 2.4000e+09
             GainSource: 'AGC Slow Attack'
         ChannelMapping: 1
      BasebandSampleRate: 1000000
          OutputDataType: 'int16'
        SamplesPerFrame: 20000

  Show all properties


tx =

  comm.SDRTxPluto with properties:

   Main
            DeviceName: 'Pluto'
               RadioID: 'usb:0'
        CenterFrequency: 2.4000e+09
                   Gain: -10
         ChannelMapping: 1
      BasebandSampleRate: 1000000

  Show all properties

fx >>
```

Displayed are the parameters of the System objects, which control the radio's sample rate, LO frequency, gain settings, and other configurations. For all SDR devices provided by MathWorks, there will be independent System objects for transmit and receive capabilities.

These parameters can be modified in two ways in MATLAB: A parameter can be set during instantiation:

```
Command Window
fx >> rx = sdrrx('Pluto','CenterFrequency',5.2e9)
```

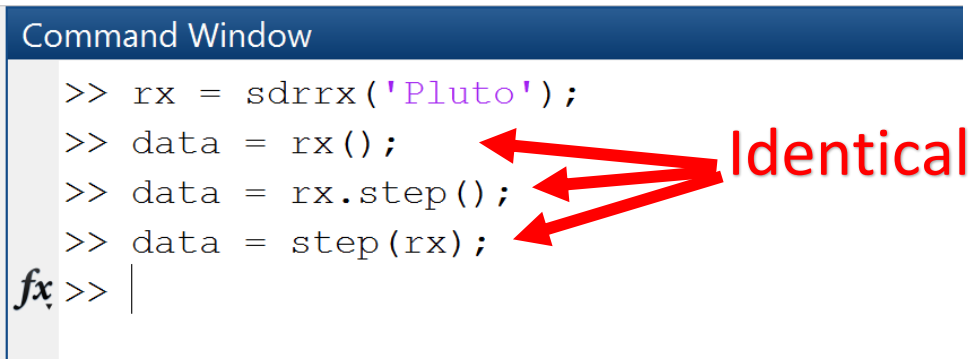
or, a parameter can be set through "dot notation" after instantiation:

```
Command Window
>> rx = sdrrx('Pluto');
>> rx.CenterFrequency = 5.2e9;
```

Parameters that require substantial configuration updates, like Sample Rate, may not be tunable in some cases. They can become locked after data is sent to or received by the radio. However, Center Frequency or Gain can be changed at any time.

Getting Data from SDR Devices

Data is transferred to and from the SDR device in buffers which are also called frames in MATLAB. Using the System object operator, we can send or receive data from the device. Internally, this is called the *step* method and the following calls are equivalent:



Run one of these three lines above in your command window and hit enter. Inspecting the output vector "data" and the object attributes as we do in Figure 1.

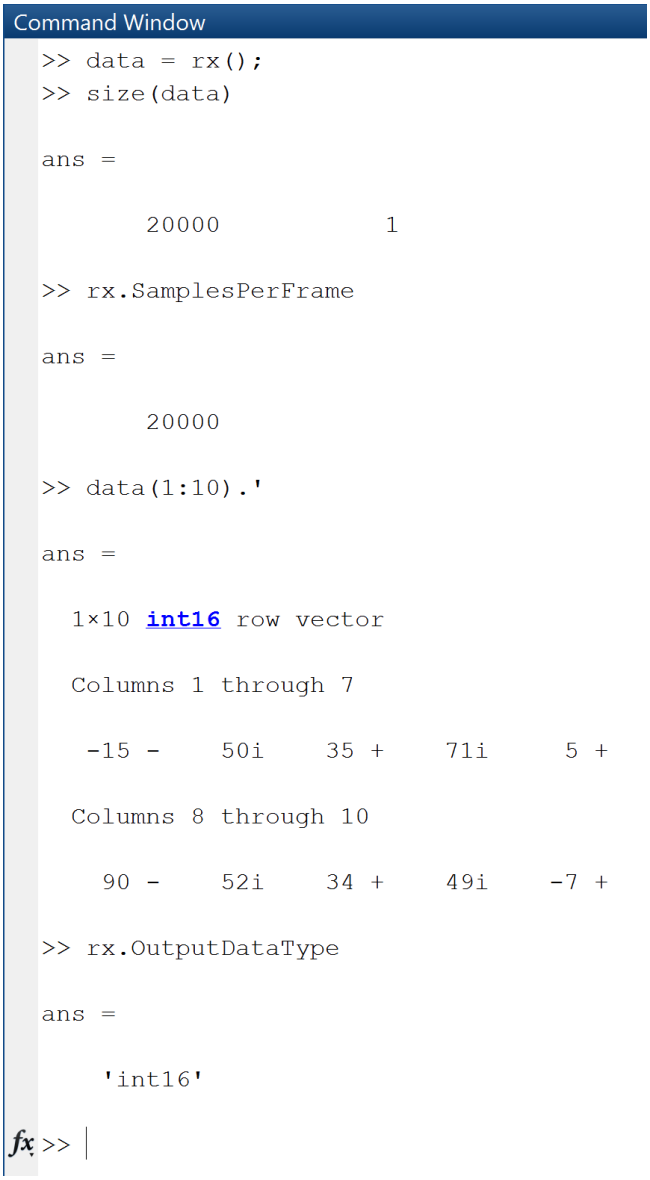
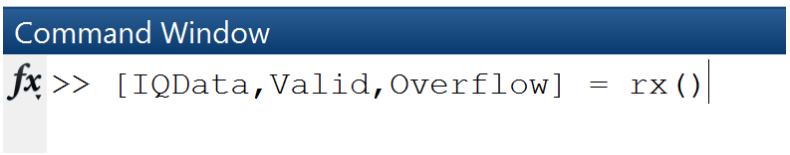


Figure 1

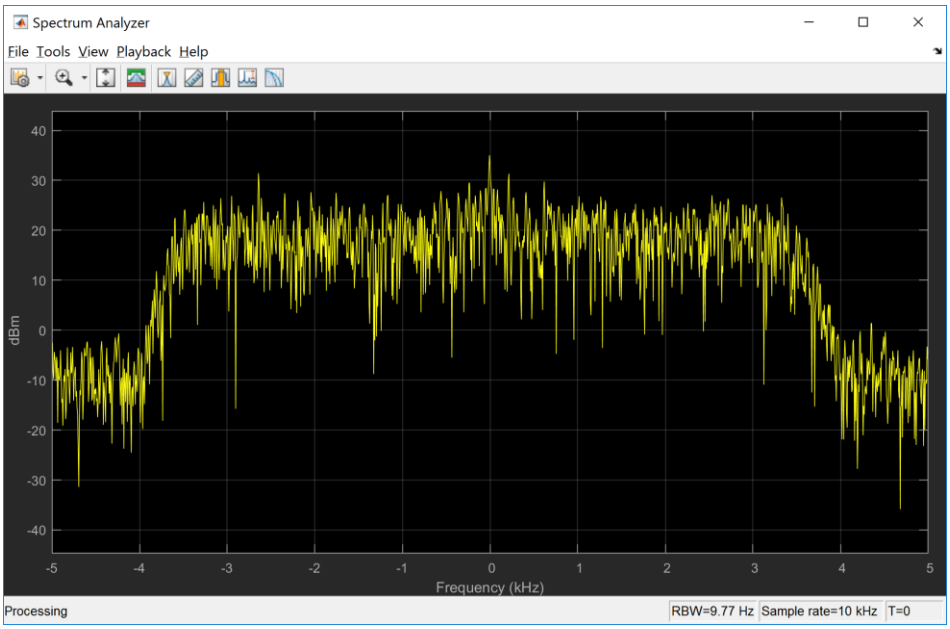
The receiver object can return three different parameters during reception: IQ data, valid out, and an overflow condition.



Let us first look at the IQ data from the SDR, which by default will be an int16 data type, but can be cast to double. ***Run the following commands which will collect data from the receiver and display it in a spectrum scope:***

```
Command Window
>> sa = dsp.SpectrumAnalyzer();
>> rx = sdrrx('Pluto', 'SamplesPerFrame',2^14);
>> sa( rx() );
```

You should be able to view a similar plot to the one shown below:

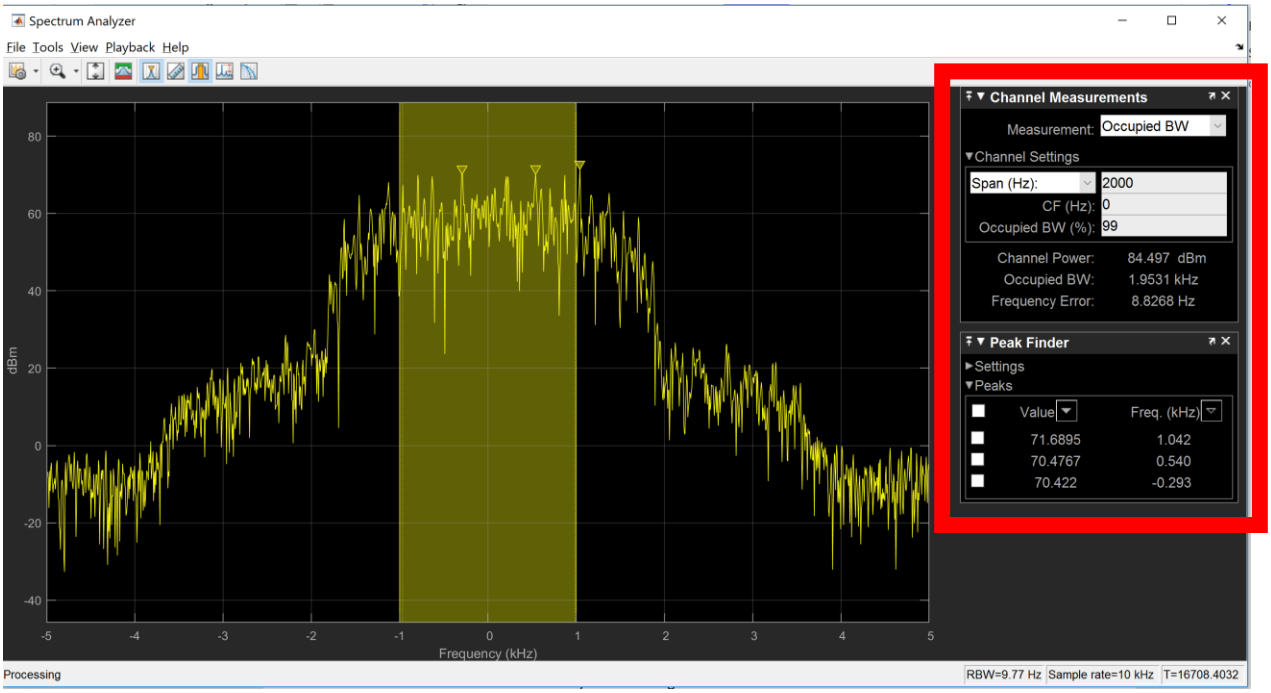


It can be useful to scale the axes using the  button on the Spectrum Analyzer scope.

Now we can repeatedly pull data (stream) from the radio and view it in the Spectrum Analyzer ***by running the following command which uses the objects we just created:***

```
Command Window
>> for k=1:1e4,sa(rx());end
```

While we are streaming data, we can enable measurements on the Spectrum Analyzer. ***Push the Peak Finder button***  ***and the Channel Measurements button***  ***on the Spectrum Analyzer.*** These will open measurement panels in the spectrum analyzer:



If you know a nearby FM station (89.9 MHz is used below for example), update the Center Frequency of the receiver to that frequency and inspect the receive power of that station. The center frequency can be updated on the System object with the MATLAB command:

Command Window

```
>> rx.CenterFrequency = 89.9e6;
>> for k=1:1e4,sa(rx());end
```

Transmitter Functionality and AGC Operation

MATLAB is a sequential language and cannot perform multiple tasks simultaneously. For example, triggering transmit and receive events simultaneously cannot be done in a single MATLAB script. To enable simultaneous transmit and receive, a method exists in the transmitter System Object called "transmitRepeat" which can be used to continuously transmit a signal. Below is the API to do so but is provided as just an example. **Do not run the code below**

```
% Set up TX
Tx = sdrtx('Pluto');
Tx.transmitRepeat(<waveform>);
% Now the transmitter will continuously repeat
% the vector "waveform" without gaps
```

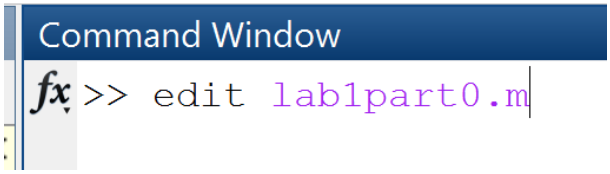
Another feature of AD936x family of transceivers is that all contain internal Automatic Gain Control functionality (AGC). By default, they can be configured in the following modes:

- Manual

- Fast Attack
- Slow Attack

These modes are configurable, and their operation can be explored in the AD9361 simulation model.

We will use the transmit repeat functionality to see how the AGC reacts over time in each of its modes. ***Start by opening script lab1part0.m from the command line as:***



In the second block (starting at line 7) of this script, we generate a complex sinusoid for testing, as shown below:

```
%% Generate test sinewave
amplitude = 1; frequency = 1e4;
swv1 = dsp.SineWave(amplitude, frequency);
swv1.SampleRate = SampleRate;
swv1.SamplesPerFrame = SamplesPerFrame;
swv1.ComplexOutput = true;
y = swv1();
```

Next a single radio is configured, with the receiver in “Slow Attack Mode”, in which is set in line 17:

```
15    %% Show  AGC changing
16    rx = sdrrx('Pluto','SamplesPerFrame',SamplesPerFrame);
17    rx.GainSource = 'AGC Slow Attack';
19    tx.transmitRepeat(y);
```

When the script is run, the transmitter is initialized with an attenuation of -20dB. Then, after half the frames are received, the transmitter is reconfigured with a gain of 0dB

```
for k=1:Frames
    z(k,:) = rx();
    % Update gain halfway through
    if k==floor(Frames/2)
        tx.Gain = 0;
        tx.transmitRepeat(y);
        disp('Gain Change');
    end
end
```

Run this script to by typing:

```
Command Window
fx >> lab1part0
```

The plot observed should be similar to Figure 2. The AGC initially converges after the first 1,000 samples. At ~1.75e5 samples, the transmitter is reconfigured with a new gain. This new signal begins at ~2.2e5 samples, where it saturates the receiver and slowly ramps down to the desired output level.

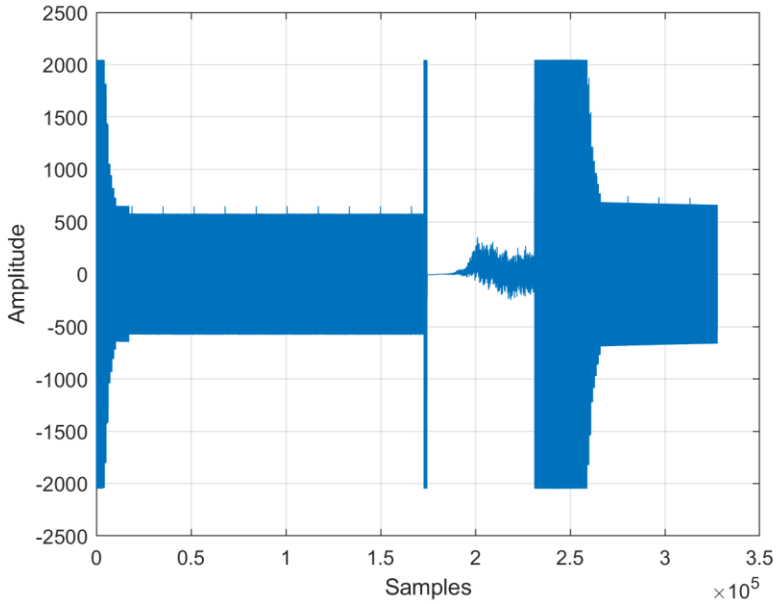


Figure 2 Received amplitude of sinusoid from lab1part0.m script

Now we can observe the "Fast Attack" mode by modifying line 17 to change the AGC mode:

```
15      %% Show  AGC changing
16 -    rx = sdrrx('Pluto','SamplesPerFrame',SamplesPerFrame);
17 -    rx.GainSource = 'AGC Fast Attack';
18 -    tx = sdrtx('Pluto','Gain',-20);
19 -    tx.transmitRepeat(y);
```

Now rerun the script with command

```
Command Window
fx >> lab1part0
```

and the resulting plot should look similar to Figure 3. In this case, the signal converges to the target power level much faster.

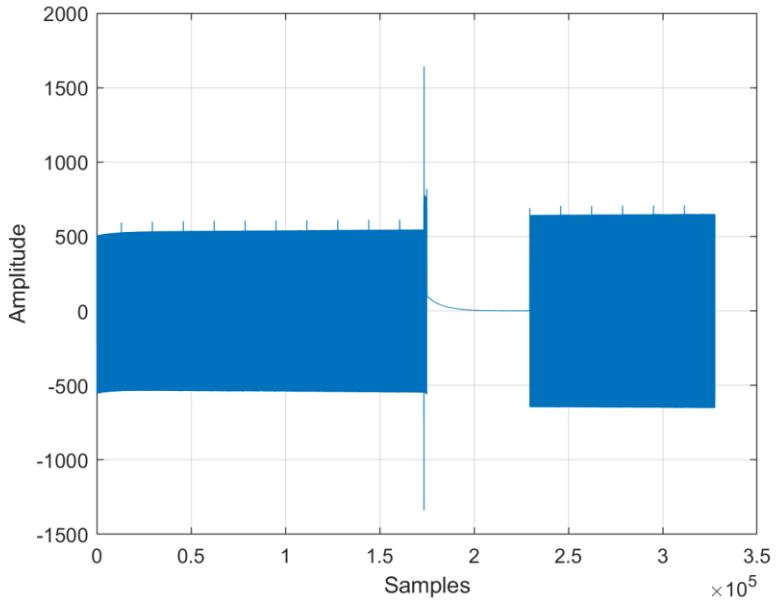


Figure 3 Received amplitude of sinusoid from lab1part0.m script in Fast-Attack mode