Using the RPI IOBoard (Red2) in MATLAB 7.4.0

This tutorial will describe the basics of how to communicate with the IOBoard through MATLAB. The example given shows how to generate simple waveforms with the IOBoard using the function generators/arbitrary waveform generators, and display them in MATLAB.

1. Start by opening MATLAB and creating a new M-File through File->New->M-File, or by pressing Ctrl+N. The M-File may open in a new window, depending on your preferences settings. The M-File is where we will be writing all of our commands to communicate with the IOBoard.

2. The first command that will always be needed is:

```
h = actxserver('RPI_IOBoard.IOBoard')
```

This command opens communications with the IOBoard through one of the computers COM ports.

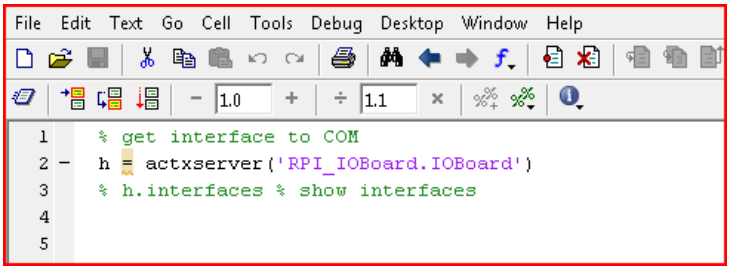

Figure 1 - Create COM Object

3. The command `h.interfaces` may be entered in the Command Window, and will show a list of possible interfaces that can be accessed on the IOBoard.

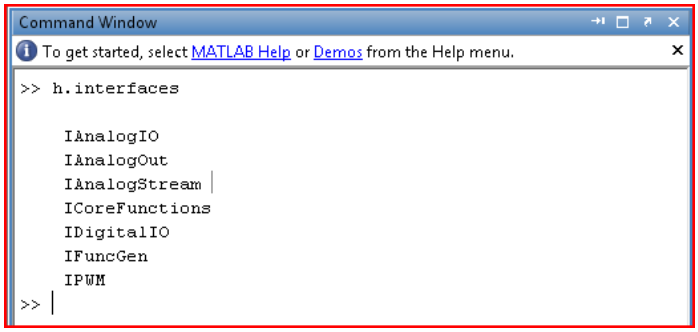

Figure 2 - IOBoard Interfaces

4. Next we need to create a variable which contains the “ICoreFunctions” interface of the IOBoard. `iob.invoke` and `iob.get` may be entered in the command window to show the interfaces and properties of ICoreFunctions.

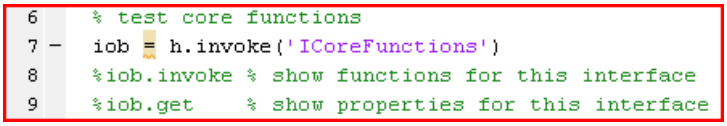

Figure 3 - Create Variable to hold ICoreFunctions

5. There should be a check to make sure that there is an IOBoard connected to the computer before going further, and if there is it should be opened (Figure 4)

```
num_devices = iob.getNumDevices()
if (num_devices < 1)
    error 'No boards connected'
    return
end

iob.openDevice(1)
```

Figure 4 - Opening IOBoard

6. Now we need to set up the function generators on the board to output a waveform. We'll start by getting the interface for the Function Generator with the line:

```
fg = h.invoke('IFuncGen');
```

`fg.invoke` may be entered in the command line to view functions available for use through the function generator interface.

```
Command Window
>> fg.invoke
    get_Frequency = double get_Frequency(handle, int32)
    set_Frequency = void set_Frequency(handle, int32, double)
    get_Phase = double get_Phase(handle, int32)
    set_Phase = void set_Phase(handle, int32, double)
    get_Waveform = string get_Waveform(handle, int32)
    set_Waveform = void set_Waveform(handle, int32, string)
    get_Offset = double get_Offset(handle, int32)
    set_Offset = void set_Offset(handle, int32, double)
    get_PkPk = double get_PkPk(handle, int32)
    set_PkPk = void set_PkPk(handle, int32, double)
    get_FrequencyResolution = double get_FrequencyResolution(handle, int32)
    Waveform_options = SafeArray(string) Waveform_options(handle)
>>
```

Figure 5 - Function Generator Interface

7. The next few lines configure function generator 1 to output a .5V, 1 kHz, Sine wave

```
fg_chan = 1;
fg.set_Waveform(fg_chan, 'Sine');
fg.set_Frequency(fg_chan, 1000);
fg.set_PkPk(fg_chan, 1.0);
fg.set_Offset(fg_chan, 0);
```

8. The function generator can only create the basic Sine, Square and Triangle waves. The red2 boards also have the ability to create any waveform through the Arbitrary Waveform Generator (AWG) which can be configured through the Analog Out interface.

```
ana_out = h.invoke('IAnalogOut');
```

Again, `ana_out.invoke` will display the available functions.

9. Next we need to configure the AWG to use channel two of the Mobile Studio Card. The following lines select a rate from a list of rates and sets channel two to that rate:

```
awg_chan = 2;
% this gives the list of rates available in Mobile Studio, but other
% (though not all) rates are also possible
rates = ana_out.get_AvailableSampleRates(awg_chan);
% get 3rd entry from list and convert to string
rate3 = cell2mat(rates(3));
ana_out.set_SampleRateString(awg_chan, rate3);
```

10. Now we will use MATLAB to create a function that will be output through the AWG. We will choose to output the sum of two sine waves.

```
theta = 0:0.1:2*pi;
data = sin(theta) + sin(2*theta);
```

Now that we have data we need to set a sample rate that is appropriate for the size of the data we are using, and write the data to channel two.

```
ana_out.set_SampleRate(2, 440*length(data));
ana_out.get_SampleRateString(2)  % displays sample rate
ana_out.StopAllChannelsSynchronously()
ana_out.WriteData(2, data')  % always stop AWGs before writing data!
ana_out.StartAllChannelsSynchronously()
```

Now that we have two signals being generated, let's take a look at them.

11. To generate our "oscilloscope" we will need the analog stream input interfaece.

```
ana_str = h.invoke('IAnalogStream');
```

Once again `ana_str.invoke` will show a list of available functions, and `ana_str.get` will show the current properties of the analog stream (Figure 6).

12. Next let's set write some code to make sure the analog stream is configured properly. Before we can do that however we need to add a simple line to ensure everything works properly.

```
feature('COM_SafeArraySingleDim', 1);
```

Without this line, the code will not run

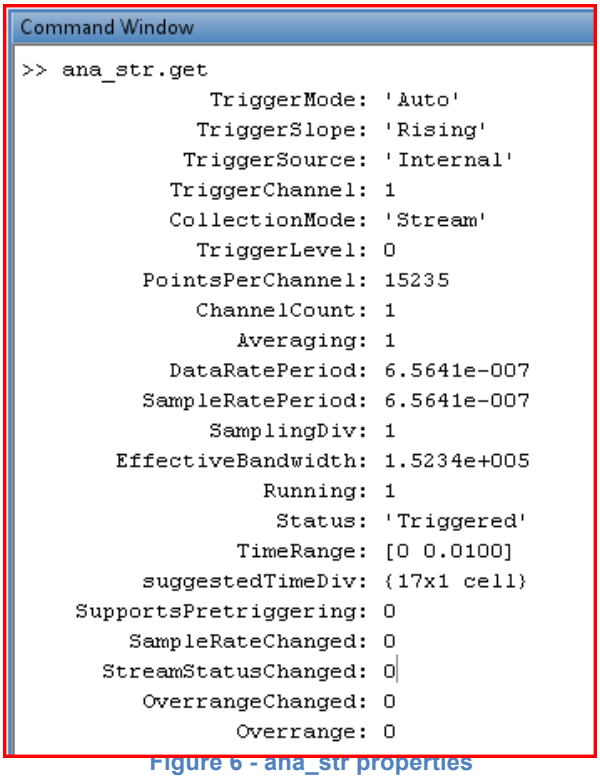

Figure 6 - ana_str properties

properly. Now that that's taken care of we can really begin with the configuration of the input stream.

```
ana_str.Averaging = 1;
ana_str.SamplingDiv = 1;
ana_str.TimeRange = [0.0; 0.01];
ana_str.TriggerChannel = 1;
ana_str.TriggerLevel = 0.0;
ana_str.TriggerMode = 'Auto';
ana_str.TriggerSlope = 'Rising';
ana_str.TriggerSource = 'Internal';
% both channels must be initialized, even if only reading from one of
% them
ana_str.set_Coupling(1, 'DC');
ana_str.set_Coupling(2, 'DC');
ana_str.set_DisplayVerticalRange(1, [-4.0; 4.0]);
ana_str.set_DisplayVerticalRange(2, [-4.0; 4.0]);
ana_str.set_InputChannel(1, AWG1);
ana_str.set_InputChannel(2, AWG2);
ana_str.Start()
```

If you are familiar with an oscilloscope, then most of these lines should look familiar. This is similar to setting an oscilloscope to 1 volt per division, setting the trigger to automatically trigger at zero volts on the rising edge of the signal on channel one, with both channels DC coupled.

13. Finally, we are ready to create a display in MATLAB to view the signals. First we need a variable that will represent our time axis.

```
x = ana_str.TimeRange(1):ana_str.DataRatePeriod:ana_str.TimeRange(2);
```

Remember that we set `TimeRange` in the previous step to contain 0.0 and 0.01, and the step size, `DataRatePeriod` (6.5641e-007), is preset. Now we're going to create an infinite loop to collect and display the data from the two channels.

```
disp 'press <control>-c to break out of loop'
while(1)
    analogData1 = ana_str.GetData(1);
    analogData2 = ana_str.GetData(2);
    if (~isempty(analogData1) && ~isempty(analogData2))
        plot(x, analogData1, x, analogData2)
        drawnow
    end
end
```

The if-statement simply checks to make sure that data was collected before plotting the graph.

14. To run the m-file and see if everything is working either click Debug->Save and Run, press F5, or click the button in the editor tool bar circled in Figure 7.

Figure 7 - Save and Run

15. The command window will display several variables and their contents before displaying a window that displays the two signals that should look similar to those seen in Figure 8.

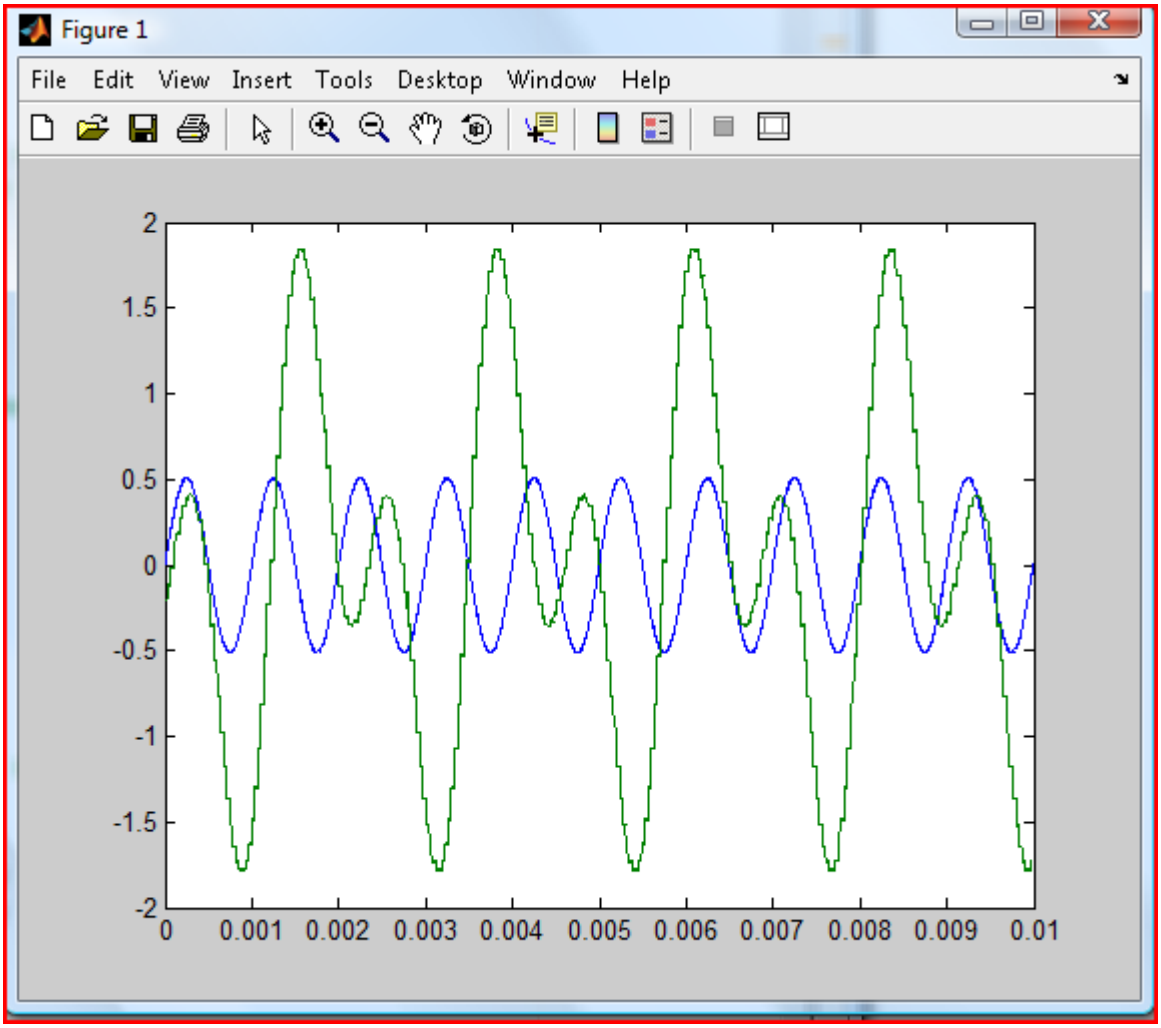

Figure 8 - Display of the two signals

To stop the signal from being displayed, press Ctrl+C while in the command window.