

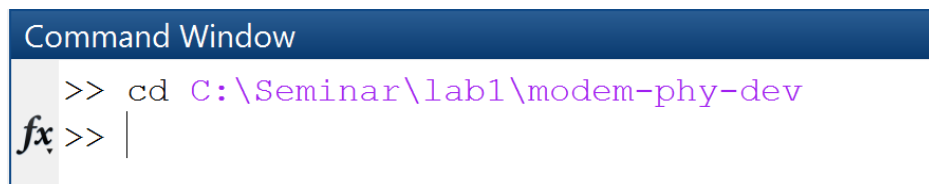
## Testing Harness

This lab will focus on the testing harness which was developed to validate modem design requirements. This test harness was design to integrate with all MATLAB reference designs and Simulink models.

### Overview: What is Where

The testing harness is built with the MATLAB Unittest Framework, which is similar to x-Unit Style tests. This allowed us to design our tests with the necessary test fixtures, logging, and validation processes, without much work on our end to design the infrastructure. These tests apply to all Simulink model designs and MATLAB reference designs in both Floating Point and Fixed Point implementations. For validation purposes the harness built here can both utilize Simulation and radio interfacing to check receiver performance. Most recently HDL tests were integrated which can deploy designs and do programmatic evaluation of those implementations on real hardware.

Let us get started by accessing the modem development folder for this lab:



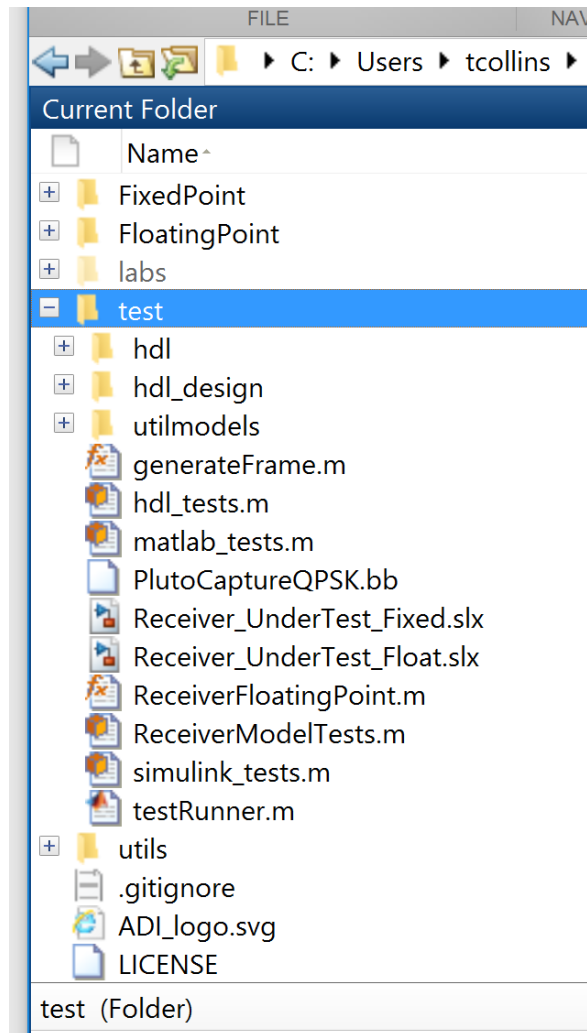
```
Command Window
>> cd C:\Seminar\lab1\modem-phy-dev
fx >> |
```

Next, we need to initialize the environment. To do this run the “startup.m” script:



```
Command Window
>> startup
fx >> |
```

From the main repository all tests and test infrastructure is based in the test folder:

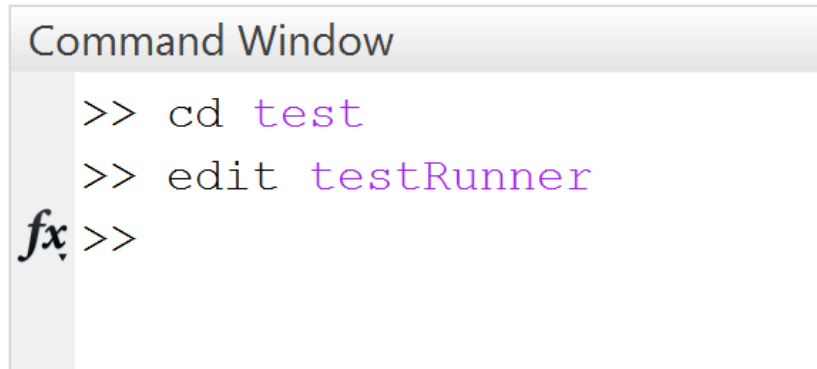


These test files are structured starting with the main top level “testRunner.m” file which controls the entire harness and allows us to pick specific tests. Individual test cases are maintained in matlab\_tests.m, Simulink\_tests.m, and hdl\_tests.m. The main scaffolding is in ReceiverModelTests.m, which runs the wrapper models Receiver\_UnderTest\_Fixed.slx , Receiver\_UnderTest\_Float.slx, and ReceiverFloatingPoint.m .

### Top Level Control and Running Individual Tests

All testing is controlled through the testRunner.m file, utilizes a “Runner” based API in MATLAB speak.

***We will start by moving to the test directory, then opening and inspecting this file with the following commands:***



```
Command Window
>> cd test
>> edit testRunner
fx >>
```

The first thing you will notice is all the import commands at the top which look like:

```
% Import necessary infrastructure
import matlab.unittest.TestRunner;
import matlab.unittest.TestSuite;
import matlab.unittest.selectors.HasTag
import matlab.unittest.plugins.TestRunProgressPlugin
import matlab.unittest.plugins.LoggingPlugin
import matlab.unittest.plugins.DiagnosticsRecordingPlugin;
```

What this code does is import the necessary libraries and add them to our namespace. Allowing us to use short names such as “LoggingPlugin” which is short for “matlab.unittest.plugins.LoggingPlugin”. This is common in the MATLAB the Unittest framework since much of it is based from Java.

Since the harness is a large list of tests it is useful to use to only use the tests that we want. This is done through Tags which are given to the test cases. The default Tag filter in the repository is set on line 15:

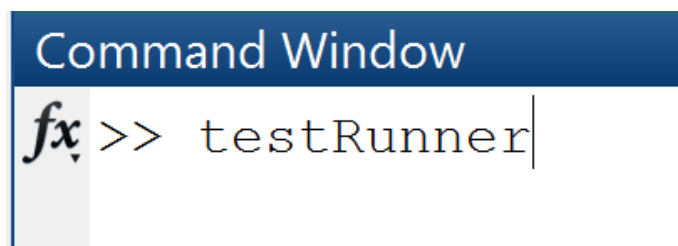
```
Tags = {'Simulation'};
```

***However, since we do not want to run Simulink tests at this time we will update line 15 to the following:***

```
Tags = {'Simulation', 'MATLAB'};
```

This will test the test runner to only run tests that are Simulation based (No radio) and MATLAB only. The remaining code in this file will attach some debugging and logging plugins to the Runner so we know what is going on when the tests run.

***With this new configuration of only running Simulation MATLAB tests run the test harness by executing the script:***



```
Command Window
fx >> testRunner
```

You should see an output similar to:

```
>> testRunner
Running Tests
1: matlab_tests/testPacketSizesSimulationFloatingPointSim
2: matlab_tests/testPacketGapsSimulationFloatingPointSim
3: matlab_tests/testFrequencyOffsetsFloatingPointSim
-----
Running matlab_tests
  Setting up matlab_tests
    Evaluating TestClassSetup: DisableWarnings
Loading
  Evaluating TestClassSetup: findRadio
Done setting up matlab_tests in 0.37485 seconds
Running matlab_tests/testPacketSizesSimulationFloatingPointSim
  Evaluating Test: testPacketSizesSimulationFloatingPointSim
Done matlab_tests/testPacketSizesSimulationFloatingPointSim in 1.2742 second
Running matlab_tests/testPacketGapsSimulationFloatingPointSim
  Evaluating Test: testPacketGapsSimulationFloatingPointSim
```

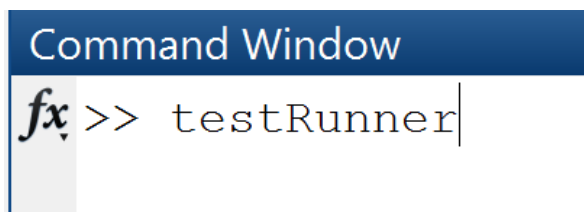
This contains the most basic output of the harness and at the end will print a table of the passed and failed tests. Sometimes we wish to view more information while a test is running. This can easily be done by increasing the verbosity level which is defined on line 35:

```
p = LoggingPlugin.withVerbosity(1);
```

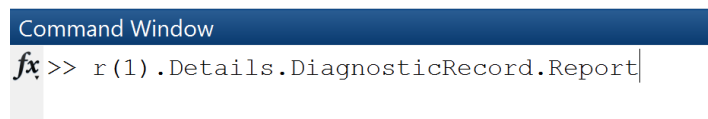
**Update this line (35) to increase the verbosity level to 4 to match:**

```
p = LoggingPlugin.withVerbosity(4);
```

**Run the test harness again by executing the script:**



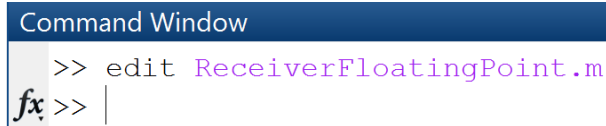
In this configuration this will be a large amount of information printed to the console, which is all the data logged by the MATLAB Reference Design during runtime. This data is also logged in the Diagnostic reports produced by the Runner at the end of the tests. **We can view the log of the first test by running the following commands:**



This output will be the same as what we view in the console. This verbosity can be set which as well is done on line 40:

```
runner.addPlugin(DiagnosticsRecordingPlugin(...
    'IncludingPassingDiagnostics',true,'Verbosity',4));
```

The statements that are printed are defined throughout the MATLAB Reference Design. We can look at an example by first opening the Receiver Design with the command:



```
Command Window
>> edit ReceiverFloatingPoint.m
fx>> |
```

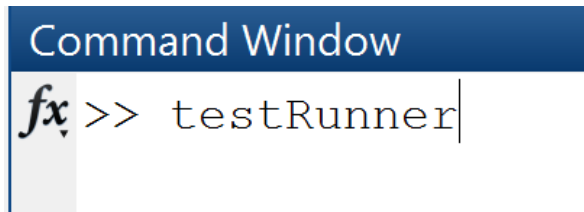
Scrolling to line 203 we can see an example which prints EVM information:

```
log(testCase,4,sprintf('Frame EVM: %f RMS (Max %f) \n',
rmsEVM,maxEVM) );
```

This statement is actually calling the log method of the testCase class with verbosity 4 and outputs EVM message. Next let us add an additional logging statement below for the mean EVM values. **Change the blank line 204 to the following:**

```
log(testCase,4,sprintf('Mean Frame EVM: %f RMS (Max
%f) \n',mean(rmsEVMs),mean(maxEVMs) ) );
```

**Run the test harness again by executing the script:**



```
Command Window
fx>> testRunner|
```

In the produced log you should observe the new statement which will look something like:

```
[Verbose] Diagnostic logged (2018-01-24T23:56:59): Mean Frame EVM: 5.961738 RMS (Max 21.644030)
```

### Debugging Failed Tests

Now tests are only useful if they provide useful information when a failure occurs. We have provided an additional case which will cause a failure mode. **To run this test again open testRunner.m and change line 15 to:**

```
Tags = {'Simulation','Failure'};
```

**Run the test harness again by executing the script:**

```
Command Window
fx >> testRunner
```

In the resulting report you will have a failure which looks like:

Name	Passed	Failed	Incomplete
'matlab_tests/testPacketSizesSimulationFloatingPointSim'	false	true	true

**Run the following command to look at the error exactly:**

```
>> r(1).Details.DiagnosticRecord.Report
```

Which will produce something similar to:

```
ans =
=====
Error occurred in matlab_tests/testPacketSizesSimulationFloatingPointSim and it did not run to completion.

-----
Error ID:
-----
''

-----
Error Details:
-----
Error using generateFrame (line 19)
Invalid payload size
```

Which tells us the exact line and function where the failure occurred. At this point we can set a breakpoint and debug as normal in MATLAB.

## Enable Visualization

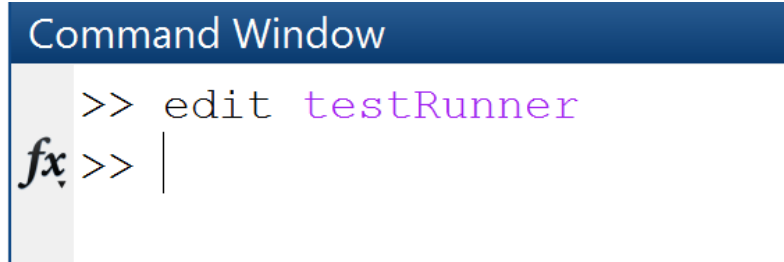
In this next part we will enable scopes to be displayed during test runtime, which can be helpful for debugging. To do so we need to enable a flag. Open the ReceiverModelTest.m main file:

```
Command Window
fx >> edit ReceiverModelTests.m
```

In this file, update line 17 to:

```
EnableVisuals = true;
```

to enable scopes. Next we need to update our original controlling script “testRunner.m” to run tests that do not fail. Reopen the “testRunner.m” script:



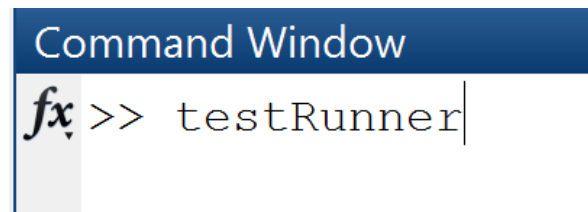
```
Command Window
fx >> edit testRunner
fx >> |
```

***However, since we do not want to run Simulink tests at this time we will update line 15 to the following:***

```
Tags = { 'Simulation' };
```

This will enable both Simulink and MATLAB tests, both which have visualizations of the receive chains.

***Run the test harness again by executing the script:***



```
Command Window
fx >> testRunner|
```

You should see additional scope appear and close as the tests run.