ECE4305: Software-Defined Radio Systems and Analysis

Getting Started with MATLAB, Simulink, USRP2 Hardware, and USRP2 Blocks

Contents

1 MATLAB Refresher and Simulink Introduction

You will be using MATLAB and Simulink for the ECE 4305 laboratory experiments, as well as for the course design project. This section serves as a brief refresher of MATLAB, since you have undoubtedly used it in previous courses. However, you probably have very little experience (if any) with Simulink, so this section also provides a crash course in Simulink.

1.1 What is MATLAB?

MATLAB is widely used in all areas of applied mathematics, in education and research at universities, and in industry. MATLAB stands for MATrix LABoratory and the software is built up around vectors and matrices. Consequently, this makes the software particularly useful for solving problems in linear algebra, but also for solving algebraic and differential equations as well as numerical integration. MATLAB possesses a collection of graphic tools capable of producing advanced GUI and data plots in both 2D and 3D. MATLAB also has several toolboxes useful for performing signal processing, image processing, optimization, and other specialized operations.

1.2 How to Edit and Run a Program in MATLAB

When writing programs, you will need to do this in a separate window, called the *editor*. To open the editor, go to the “File” menu and choose either the “New...M-file” (if you want to create a new program) or “Open” (to open an old document) option. In the editor, you can now type in your code in much the same way that you would use a text editor or a word processor. There are menus for editing the text that are also similar to any word processor. While typing your code in the editor, no commands will be performed. Consequently, in order to run a program, you will need to do the following:

1. Save your code as *<filename>.m*, where *<filename>* is anything you wish to name the file. **It is important to add “.m” at the end of your filename. Otherwise MATLAB may not understand your program.**

2. Go to the command window. If necessary, change directories to the directory containing your file. This can be accomplished using the `cd` command common to UNIX and DOS. Or, alternatively, you can select the button labeled “...” on the “Current Directory” window, and change directories graphically. You can view your current directory by typing the `pwd` command.

3. In order to run the program, type the name of the file containing your program at the prompt. **When typing the filename in the command window do not include “.m”.** By pressing enter, MATLAB will run your program and perform all the commands given in your file.

In case your code has errors, MATLAB will complain when you try to run the program in the command window. When this happens, try to interpret the error message and make necessary changes to you code in the editor. The error that is reported by MATLAB is hyperlinked to the line in the file that caused the problem. Using your mouse, you can jump directly to the line in your program that has caused the error. **After you have made the changes, make sure you save your file before trying to run the program again in the command window.**

1.3 Useful Tools

This section introduces general techniques for finding errors, as well as using automatic code analysis functions in order to detect possible areas for improvement within the MATLAB code. In particular, the MATLAB debugger features located within the Editor, as well as equivalent Command Window debugging functions, will be covered.

Debugging is the process by which you isolate and fix problems with your code. Debugging helps to correct two kinds of errors:

- Syntax errors: For example, misspelling a function name or omitting a parenthesis.
- Run-time errors: These errors are usually algorithmic in nature. For example, you might modify the wrong variable or code a calculation incorrectly. Run-time errors are usually apparent when an M-file produces unexpected results. Run-time errors are difficult to track down because the function's local workspace is lost when the error forces a return to the MATLAB base workspace.

1.3.1 Code Analysis and M-Lint Messages

MATLAB can check your code for problems and recommend modifications to maximize the performance and maintainability though messages, sometimes referred to as M-Lint messages. The term "lint" is the name given to similar tools used with other programming languages such as C. In MATLAB, the M-Lint tool displays a message for each line of an M-file it determines possesses the potential to be improved. For example, a common M-Lint message is that a variable is defined but never used in the M-file.

You can check for coding problems using three different ways, all of which report the same messages:

- Continuously check code in the Editor while you work. View M-Lint messages and modify your file based on the messages. The messages update automatically and continuously so you can see if your changes addressed the issues noted in the messages. Some messages offer extended information, automatic code correction, or both.
- Run a report for an existing MATLAB code file: From a file in the Editor, select **Tools** > **Code Analyzer** > **Show Code Analyzer Report**.
- Run a report for all files in a folder: In the Current Folder browser, click the Actions button, then select **Reports** > **Code Analyzer Report**.

For each message, review the message and the associated code in order to make changes to the code itself based on the message via the following process:

- Click the line number to open the M-file in the Editor/Debugger at that line.
- Review the M-Lint message in the report and change the code in the M-file, based on the message.
- Note that in some cases, you should not make any changes based on the M-Lint messages because the M-Lint messages do not apply to that specific situation. M-Lint does not provide perfect information about every situation.

- Save the M-file. Consider saving the file to a different name if you made significant changes that might introduce errors. Then you can refer to the original file as you resolve problems with the updated file.

- If you are not sure what a message means or what to change in the code as a result, use the Help browser to look for related topics.

You can also get M-Lint messages using the `mlint` function. For more information about this function, you can type `help mlint` in the Command Window. Read the online documentation Code Analysis Options for more information about this tool.

1.3.2 Debugger

The MATLAB Editor, graphical debugger, and MATLAB debugging functions are useful for correcting run-time problems. They enable access to function workspaces and examine or change the values they contain. You can also set and clear *breakpoints*, which are indicators that temporarily halt execution in a file. While stopped at a breakpoint, you can change the workspace contexts, view the function call stack, and execute the lines in a file one by one.

There are two important techniques in debugging: one is the *breakpoint* while the other is the *step*. Setting *breakpoints* to pause the execution of a function enables you to examine values where you think the problem might be located. While debugging, you can also *step* through an M-file, pausing at points where you want to examine values.

There are three basic types of breakpoints that you can set in the M-files, namely:

- A standard breakpoint, which stops at a specified line in an M-file.

- A conditional breakpoint, which stops at a specified line in an M-file only under specified conditions.

- An error breakpoint that stops in any M-file when it produces the specified type of warning, error, or NaN or infinite value.

You cannot set breakpoints while MATLAB is busy, e.g., running an M-file, unless that M-file is paused at a breakpoint. While the program is paused, you can view the value of any variable currently in the workspace, thus allowing you to examine values when you want to see whether a line of code has produced the expected result or not. If the result is as expected, continue running or step to the next line. If the result is not as expected, then that line, or a previous line, contains an error.

While debugging, you can change the value of a variable in the current workspace to see if the new value produces expected results. While the program is paused, assign a new value to the variable in the Command Window, Workspace browser, or Array Editor. Then continue running or stepping through the program. If the new value does not produce the expected results, the program has a different or another problem.

As mentioned before, you can debug MATLAB files using the Editor, which is a graphical user interface, as well as by using debugging functions from the Command Window. You can use both methods interchangeably. Read the online documentation Debugging Process and Features for more information about this tool.

1.3.3 Profiler

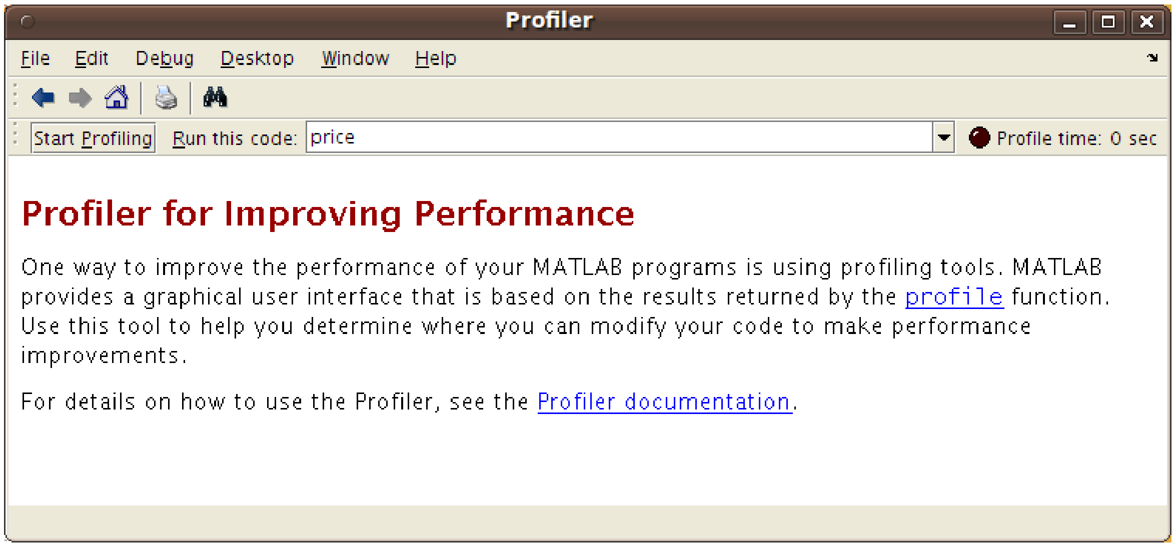
Profiling is a way to measure the amount of time a program spends on performing various functions. Using the MATLAB Profiler, you can identify which functions in your code consume the most time. You can then determine why you are calling them and look for ways to minimize their use. It is often helpful to decide whether the number of times a particular function is called is reasonable. Since programs often have several layers, your code may not explicitly call the most time-consuming functions. Rather, functions within your code might be calling other time-consuming functions that can be several layers down into the code. In this case, it is important to determine which of your functions are responsible for such calls.

Profiling helps to uncover performance problems that you can solve by:

- Avoiding unnecessary computation, which can arise from oversight.
- Changing your algorithm to avoid costly functions.
- Avoiding recomputation by storing results for future use.

When you reach the point where most of the time is spent on calls to a small number of built-in functions, you have probably optimized the code as much as you can expect. You can use any of the following methods to open the Profiler:

- Select **Desktop** $\rightarrow$ **Profiler** from the MATLAB desktop.
- Select **Tools** $\rightarrow$ **Open Profiler** from the menu in the MATLAB Editor/Debugger.
- Select one or more statements in the Command History window, right-click to view the context menu, and choose **Profile Code**.
- Enter the following function in the Command Window: `profile viewer`.



Profiler for Improving Performance

One way to improve the performance of your MATLAB programs is using profiling tools. MATLAB provides a graphical user interface that is based on the results returned by the profile function. Use this tool to help you determine where you can modify your code to make performance improvements.

For details on how to use the Profiler, see the Profiler documentation.

Figure 1: Profiler

To profile an M-file or a line of code, follow these steps after you open the Profiler, as shown in Fig. 1:

1. In the **Run this code** field in the Profiler, type the statement you want to run.
2. Click **Start Profiling** (or press **Enter** after typing the statement).
3. When profiling is complete, the **Profile Summary** report appears in the Profiler window.

Read the online documentation Profiling for Improving Performance for more information about this tool.

1.4 Simulink Introduction

Simulink (**Simu**lation and **Link**) is an extension of MATLAB created by Mathworks Inc. It works with MATLAB to offer modeling, simulation, and analysis of dynamical systems within a graphical user interface (GUI) environment. The construction of a model is simplified with click-and-drag mouse operations. Simulink includes a comprehensive block library of toolboxes for both linear and nonlinear analyses. Models are hierarchical, which allow using both top-down and bottom-up approaches. As Simulink is an integral part of MATLAB, it is easy to switch back and forth during the analysis process and thus, the user may take full advantage of features offered in both environments. This section presents the basic features of Simulink and is focused on Communications System Toolbox.

1.5 Getting Started in Simulink

1.5.1 Start a Simulink Session

To start a Simulink session, you'd need to bring up MATLAB program first.

From MATLAB command window, enter:

```
>> simulink
```

Alternately, you may click on the Simulink icon located on the toolbar as shown:

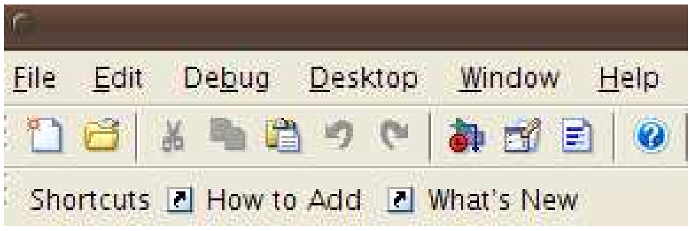


Figure 2: Start a Simulink session by clicking on the Simulink icon.

Simulink's library browser window, as shown in Fig. 3 will pop up presenting the block set for model construction.

To see the content of the blockset, click on the "+" sign at the beginning of each toolbox.

1.5.2 Start a Simulink Model

To start a model click on the *new file* icon as shown in Fig. 3. Alternately, you may use keystrokes **CTRL+N**.

A new window will appear on the screen. You will be constructing your model in this window. Also in this window the constructed model is simulated. A screenshot of a typical working (model) window is shown in Fig. 4.

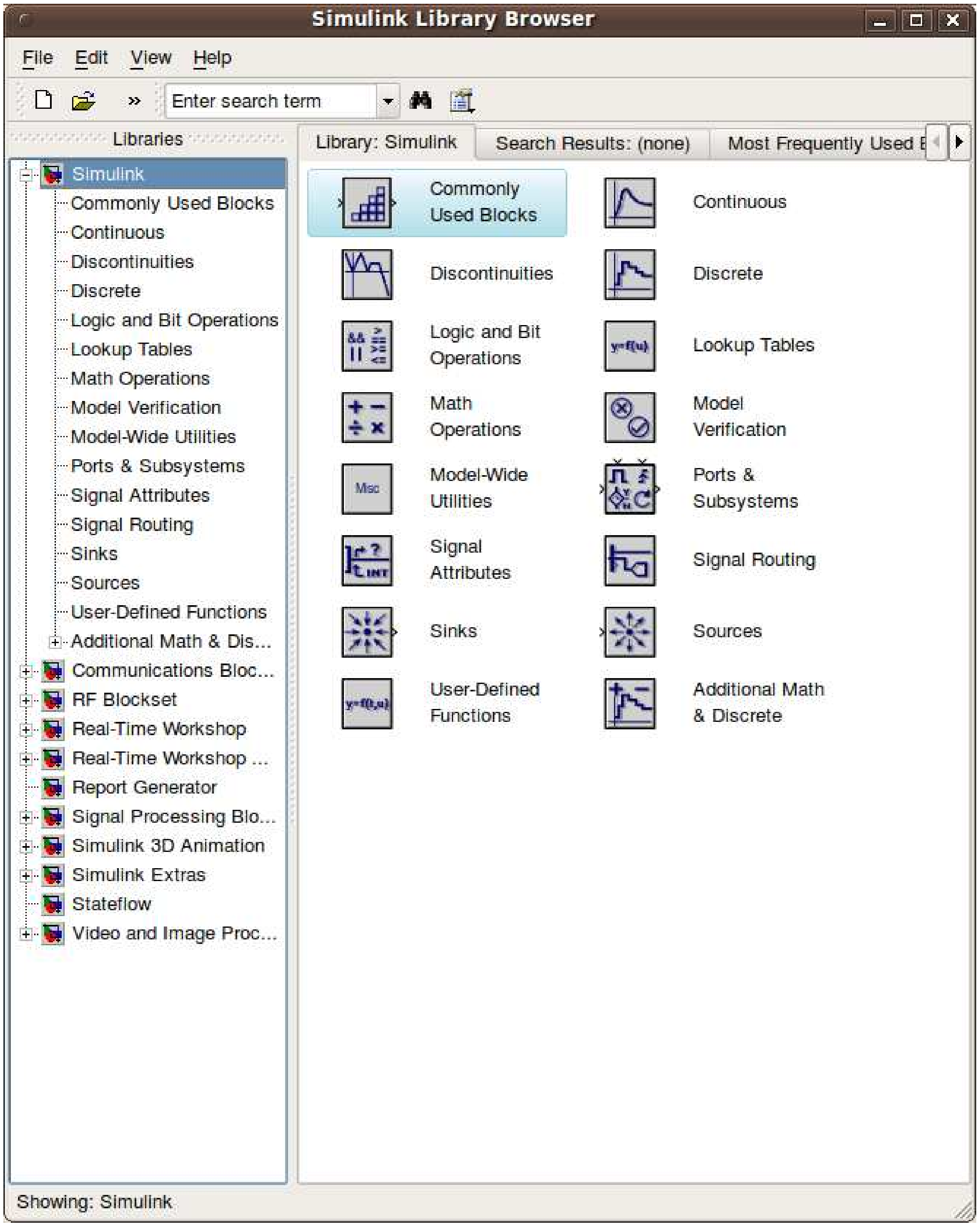


Figure 3: Simulink library browser.

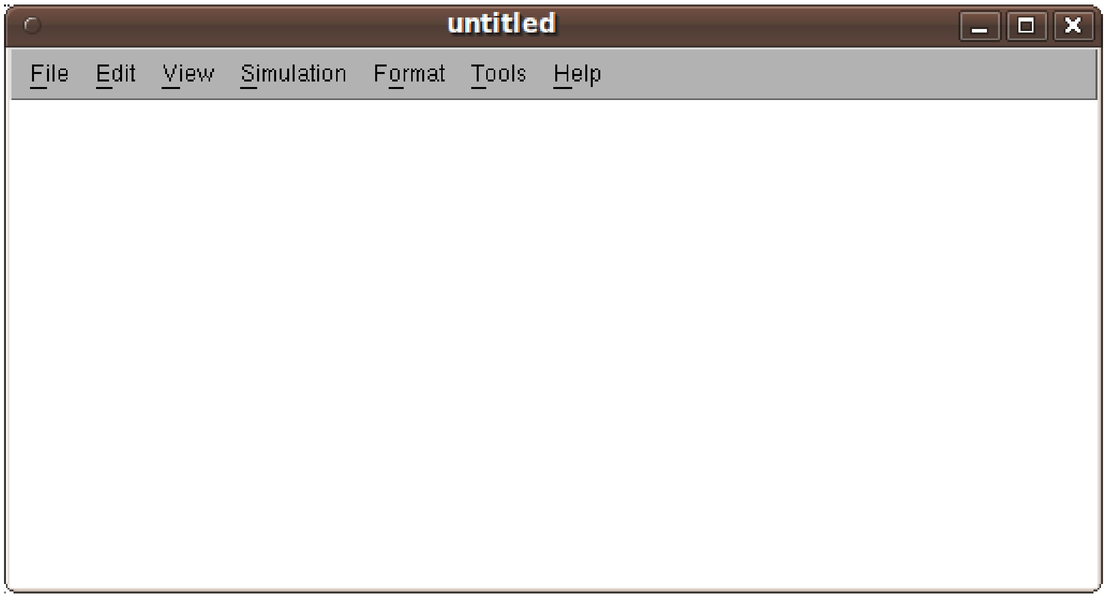


Figure 4: A screenshot of a typical Simulink model window.

1.5.3 Simulink Model Settings

In this course, most of the simulations and implementations are designed for communication systems, so a proper Simulink setting is very helpful.

`commstartup` changes the default Simulink model settings to values more appropriate for the simulation of communication systems. The changes apply to new models that you create later in the MATLAB session, but not to previously created models.

To install the communications-related model settings each time you start MATLAB, invoke `commstartup` from your `startup.m` file. To be more specific, the settings in `commstartup` cause models to:

- Use the variable-step discrete solver in single-tasking mode.
- Use starting and ending times of 0 and Inf, respectively.
- Avoid producing a warning or error message for inherited sample times in source blocks.
- Set the Simulink Boolean logic signals parameter to Off.
- Avoid saving output or time information to the workspace.
- Produce an error upon detecting an algebraic loop.
- Inline parameters if you use the Model Reference feature of Simulink.

If your communications model does not work well with these default settings, you can change each of the individual settings as the model requires.

To become familiar with the structure and the environment of Simulink, you are encouraged to explore the toolboxes and scan their contents. You may not know what they are all about at first, but perhaps you could catch on the organization of these toolboxes according to their categories. For instance, you may see that the Communications Blockset consists of Comm Sources, Channels, Comm

Sinks and so on. A good way to learn Simulink (or any computer program in general) is to practice and explore.

1.6 Building a System

To demonstrate how a system is represented using Simulink, we will build the block diagram for a simple model consisting of a sinusoidal input multiplied by a constant gain, which is shown in Fig. 5.

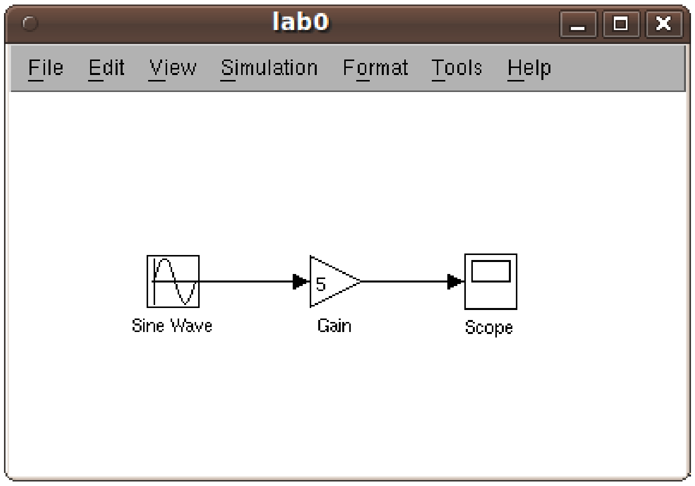


Figure 5: A simple example of building a Simulink model.

This model will consist of three blocks: Sine Wave, Gain, and Scope. The Sine Wave is a **Source Block** from which a sinusoidal input signal originates. This signal is transferred through a **line** in the direction indicated by the arrow to the Gain **Math Block**. The Gain block modifies its input signal (multiplies it by a constant value) and outputs a new signal through a **line** to the Scope block. The Scope is a **Sink Block** used to display a signal (much like an oscilloscope).

Building the system model is accomplished through a series of steps:

1. The necessary blocks are gathered from the Library Browser and placed in the model window.
2. The parameters of the blocks are then modified to correspond with the system we are modeling.
3. Finally, the blocks are connected with lines to complete the model.

Each of these steps will be explained in detail using our example system. Once a system is built, simulations are run to analyze its behavior.

1.6.1 Gathering Blocks

Each of the blocks we will use in our example model will be taken from the Simulink Library Browser. To place the Sine Wave block into the model window, follow these steps:

1. Click on the "+" in front of "Simulink" ("Sources" is a subfolder beneath the "Simulink" folder).
2. Scroll down until you see the "Sources" folder. Click on "Sources", the right window will display the various source blocks available for us to use and Sine Wave is one of them, as shown in Fig. 6.

3. To insert a Sine Wave block into your model window, click on it in the Library Browser and drag the block into your workspace.

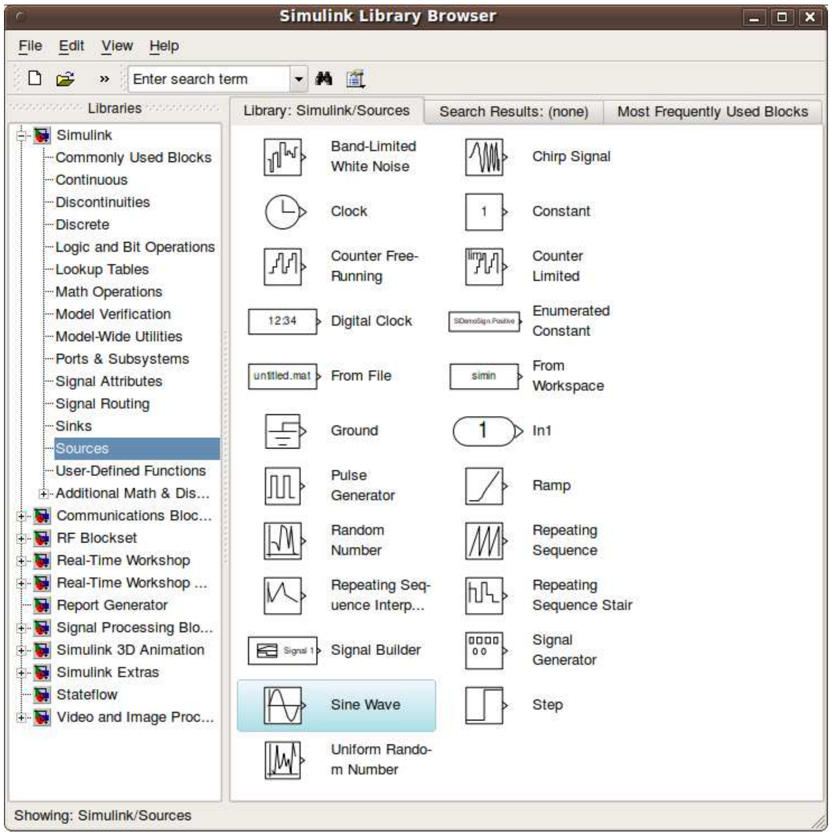


Figure 6: Gather Sine Wave block from Simulink Library Browser.

The same method can be used to place the Gain and Scope blocks in the model window. The “Gain” block can be found in the “Math Operations” subfolder and the “Scope” block is located in the “Sink” subfolder. Arrange the three blocks in the workspace (done by selecting and dragging an individual block to a new location) so that they look similar to Fig. 7.

1.6.2 Modifying the Blocks

Simulink allows us to modify the blocks in our model so that they accurately reflect the characteristics of the system we are analyzing. For example, we can modify the Sine Wave block by double-clicking on it. Doing so will cause the following window to appear, as shown in Fig. 8.

This window allows us to adjust the amplitude, bias, frequency, and phase shift of the sinusoidal input. The “Sample time” value indicates the time interval between successive readings of the signal. Setting this value to 0 indicates the signal is sampled continuously.

Let us assume that our system’s sinusoidal input has:

- Amplitude = 2
- Bias = 0
- Frequency = pi

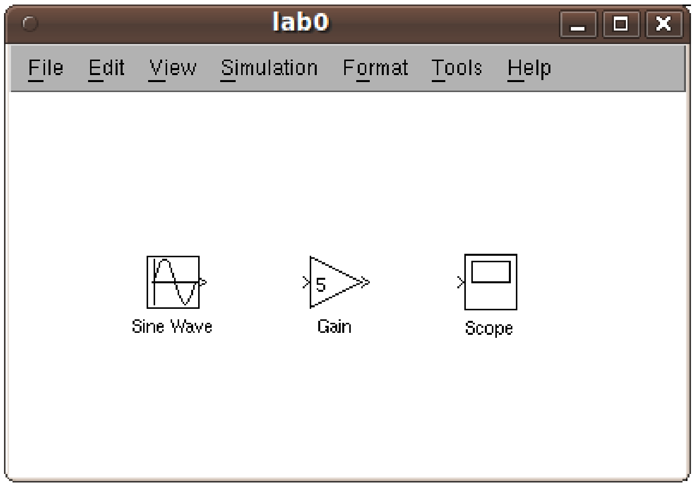


Figure 7: Gathering all the necessary blocks for the model.

- Phase = pi/2

Enter these values into the appropriate fields (leave the "Sample time" set to 0) and click "OK" to accept them and exit the window. Note that the frequency and phase for our system contain 'pi'. These values can be entered into Simulink just as they have been shown. Next, we modify the Gain block by double-clicking on it in the model window. For our system, we will let $k = 5$. Enter this value in the "Gain" field, and click "OK" to close the window.

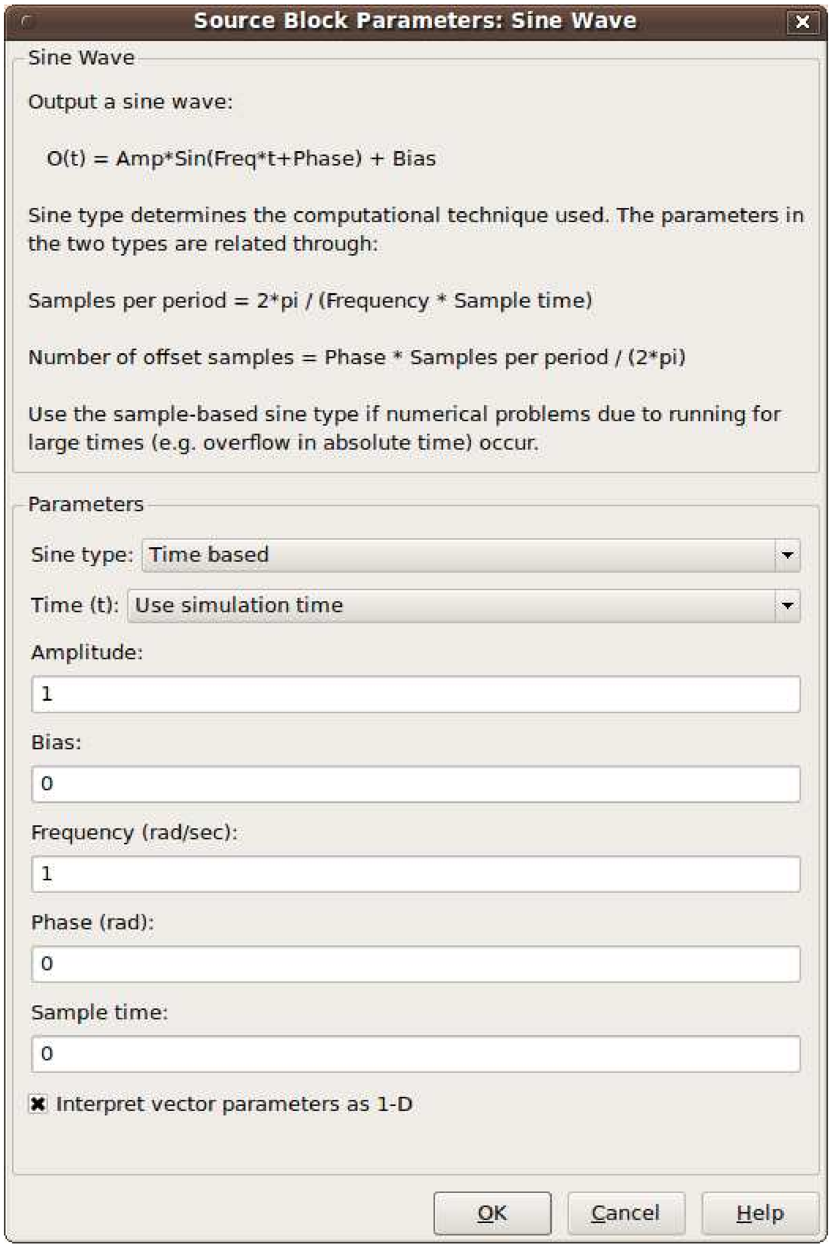
Note that Simulink gives a brief explanation of the block's function in the top portion of this window. In the case of the Gain block, the signal input to the block (u) is multiplied by a constant (K) to create the block's output signal (y). Changing the "Gain" parameter in this window changes the value of K.

The Scope block simply plots its input signal as a function of time, and thus there are no system parameters that we can change for it. We will look at the Scope block in more detail after we have run our simulation.

1.6.3 Connecting the Blocks

For a block diagram to accurately reflect the system we are modeling, the Simulink blocks must be properly connected. In our example system, the signal output by the Sine Wave block is transmitted to the Gain block. The Gain block amplifies this signal and outputs its new value to the Scope block, which graphs the signal as a function of time. Thus, we need to draw lines from the output of the Sine Wave block to the input of the Gain block, and from the output of the Gain block to the input of the Scope block.

Lines are drawn by dragging the mouse from where a signal starts (output terminal of a block) to where it ends (input terminal of another block). When drawing lines, it is important to make sure that the signal reaches each of its intended terminals. Simulink will turn the mouse pointer into a crosshair when it is close enough to an output terminal to begin drawing a line, and the pointer will change into a double crosshair when it is close enough to snap to an input terminal. A signal is properly connected if its arrowhead is filled in, as shown in Fig. 9(a). If the arrowhead is open, as shown in Fig. 9(b), it means the signal is not connected to both blocks. To fix an open signal, you can treat the open arrowhead as an output terminal and continue drawing the line to an input terminal in the same manner as explained before.


Source Block Parameters: Sine Wave

Sine Wave

Output a sine wave:

O(t) = Amp*Sin(Freq*t+Phase) + Bias

Sine type determines the computational technique used. The parameters in the two types are related through:

Samples per period = 2*pi / (Frequency * Sample time)

Number of offset samples = Phase * Samples per period / (2*pi)

Use the sample-based sine type if numerical problems due to running for large times (e.g. overflow in absolute time) occur.

Parameters

Sine type: Time based

Time (t): Use simulation time

Amplitude: 1

Bias: 0

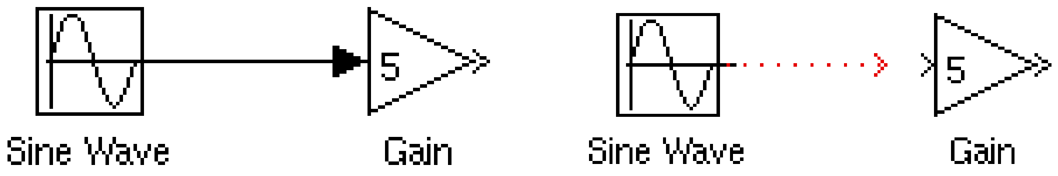
Frequency (rad/sec): 1

Phase (rad): 0

Sample time: 0

☒ Interpret vector parameters as 1-D

Figure 8: Setting parameters of Sine Wave block.



(a) Properly connected signal. (b) Open signal.

Figure 9: Connecting the blocks.

When drawing lines, you do not need to worry about the path you follow. The lines will route themselves automatically. Once blocks are connected, they can be repositioned for a neater appearance. This is done by clicking on and dragging each block to its desired location (signals will stay properly connected and will re-route themselves). After drawing in the lines and repositioning the blocks, the example system model should look like Fig. 5.

In some models, it will be necessary to branch a signal so that it is transmitted to two or more different input terminals. This is done by first placing the mouse cursor at the location where the signal is to branch. Then, using either the CTRL key in conjunction with the left mouse button or just the right mouse button, drag the new line to its intended destination. The routing of lines and the location of branches can be changed by dragging them to their desired new position. To delete an incorrectly drawn line, simply click on it to select it, and hit the DELETE key.

Note that it is recommended that you save your model at some point early on so that if your PC crashes you wouldn't lose too much time reconstructing your model.

1.7 Running Simulations

Now that our model has been constructed, we are ready to simulate the system. To do this, go to the **Simulation** menu and click on **Start**, or just click on the "Start/Pause Simulation" button in the model window toolbar (looks like the "Play" button on a VCR). Since our example is a relatively simple model, its simulation runs almost instantaneously. With more complicated systems, however, you will be able to see the progress of the simulation by observing its running time in the the lower box of the model window. Double-click the Scope block to view the output of the Gain block for the simulation as a function of time. Once the Scope window appears, click the "Autoscale" button in its toolbar (looks like a pair of binoculars) to scale the graph to better fit the window. Having done this, you should see Fig. 10.

Note that the output of our system appears as a cosine curve with a period of 2 seconds and amplitude equal to 10. Does this result agree with the system parameters we set? Its amplitude makes sense when we consider that the amplitude of the input signal was 2 and the constant gain of the system was 5 ($2 \times 5 = 10$). The output's period should be the same as that of the input signal, and this value is a function of the frequency we entered for the Sine Wave block (which was set equal to pi). Finally, the output's shape as a cosine curve is due to the phase value of pi/2 we set for the input (sine and cosine graphs differ by a phase shift of pi/2).

What if we were to modify the gain of the system to be 0.5? How would this affect the output of the Gain block as observed by the Scope? Make this change by double-clicking on the Gain block

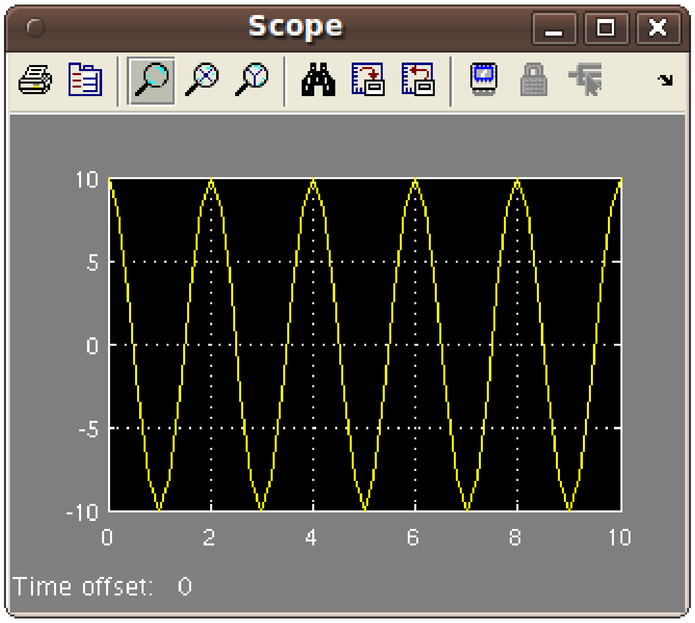


Figure 10: The scope display of the model.

and changing the gain value to 0.5. Then, re-run the simulation and view the Scope (the Scope graph will not change unless the simulation is re-run, even though the gain value has been modified). The Scope graph should now look like Fig. 11.

Notice that the only difference between this output and the one from our original system is the amplitude of the cosine curve. In the second case, the amplitude is equal to 1, or 1/10th of 10, which is a result of the gain value being 1/10th as large as it originally was.

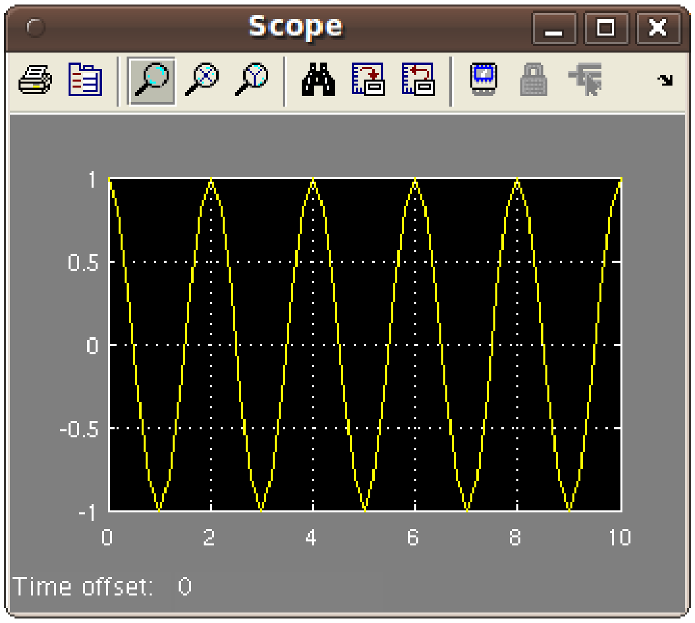


Figure 11: The scope display of the model with gain=0.5.

2 USRP2 Hardware

For all the laboratory experiments to be conducted in this course, you will be using the USRP2 hardware platform and USRP2 blocks in Simulink to implement real SDR systems. The USRP hardware was developed and built by Ettus Research LLC [1], but the design is open-source and the schematics are freely available [2]. The baseband signal processing is performed on the host computer workstation and the USRP contains the electronics (RF-frontend, ADCs, DACs, FPGA, etc.) required to upconvert or downconvert to and from RF. There are two versions of the USRP: the USRP1 (which communicates with the PC via a USB 2.0 interface) and the USRP2 (which is more powerful and uses a gigabit Ethernet interface). In these laboratory experiments, you will be using the USRP2 platform.

In order to support many different RF frequency bands, the RF frontend electronics are contained on a removeable RF daughtercards. Ettus Research sells a variety of different daughtercards for the USRP. In these labs, you will use the XCVR2450 daughtercard, which supports 2.4GHz and 5.0GHz frequency bands. The schematics for the daughtercards are also freely available [2].

Figure 12 provides a diagram of how the entire system goes together. The USRP2 platform and its XCVR2450 daughtercard is shown on the left. The USRP2 contains a Spartan-3 FPGA (similar to one you use in ECE 3801 and ECE 3810). The Spartan-3 interfaces with an ADC chip and a DAC chip at 100 MHz (though you sometimes see “400 MSps” written for the DAC because it automatically interpolates by a factor of four) [3]). The USRP2 communicates with the PC via its gigabit Ethernet controller. The Linux PC is shown on the right-hand side.

Here are a few miscellaneous things to keep in mind about the USRP2 hardware platform:

- The XCVR2450 daughtercard supports two frequency bands (2.4 to 2.5 GHz and 4.9 to 5.9 GHz).
- The XCVR2450 daughtercard hardware does not support bidirectional communication; in other words, you can either transmit or receive but not both simultaneously.

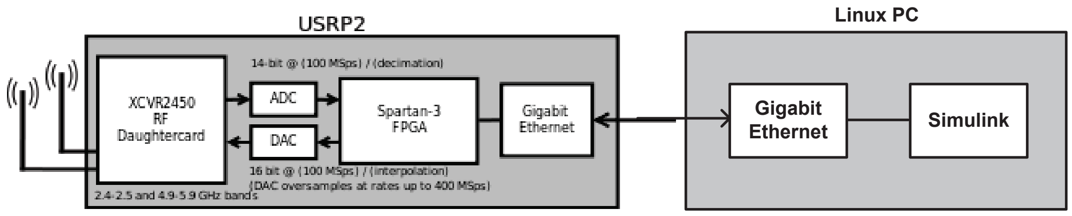

USRP2
14-bit @ (100 MSps) / (decimation)
XCVR2450
RF
Daughtercard
ADC
DAC
Spartan-3
FPGA
Gigabit
Ethernet
16 bit @ (100 MSps) / (interpolation)
(DAC oversamples at rates up to 400 MSps)
2.4-2.5 and 4.9-5.9 GHz bands
Linux PC
Gigabit
Ethernet
Simulink

Figure 12: System Block Diagram

- The XCVR2450 daughtercard always uses its 'J1' connector for transmitting and 'J2' for receiving. At present time, this behavior seems to be hard-wired in the FPGA code, so you cannot change the transmit/receive antenna channels in the software.
- If you look at the front of the USRP2 box, you will see that there are four SMA connectors, two of which are respectively labeled 'RF1' and 'RF2.' If you take a look inside the USRP2, you will see that two cables run from these connectors to the XCVR2450 daughtercard. Take a look at the screen-printed labels 'J1' and 'J2' on the daughtercard. Notice that the left-hand connector on the daughtercard is 'J2' and the right-hand connector is 'J1' (assuming that you orient the USRP2 so that it is facing towards you). Thus, if the two cables in the USRP2 run parallel (without crossing), 'RF1' will be connected to 'J2' and 'RF2' will be connected 'J1'. You need to be aware of this. This may sound trivial or obvious, but it is very easy to overlook!
- Each of the antennas has 50 Ohm input impedance.
- Never connect a cable between two XCVR2450 daughtercards cards. The maximum transmit power (+20dBm = 100mW) exceeds the maximum receive power rating (+10dBm), and you would risk damaging the receiver if you were not careful. You shouldn't need to connect them in these labs because the wireless transmission works just fine.

3 Experimental Preparations

There are a few things you need to do before you can actually use your USRP2 hardware with the Simulink software, introduced as following:

3.1 Setting Up Your Hardware

Connect the USRP2 board to your computer using a gigabit Ethernet cable. For optimal results, attach the USRP2 directly to the computer, on a private network (i.e., one without any other computers, routers, or switches). Connecting the USRP2 board directly to your computer requires the following:

- A version of USRP2 firmware which supports UDP. Links to this firmware are provided in [5].

- A gigabit Ethernet card, which supports flow control and has a static IP address. For more information, follow these guidelines below.

3.2 Burning the Firmware to an SD Card

If you are using the Ubuntu operating system, you need to be a root user:

1. Insert the SD memory card into the USB burner.
2. From the terminal, change to the directory containing the USRP2 SD Card Burner GUI using 'cd' command.
3. Launch the USRP2 SD Card Burner GUI by typing 'sudo python usrp2_card_burner_gui.py' at the terminal.
4. Select the drive for the memory card device. For example, if the memory card device is on the sdd drive, select /dev/sdd.
5. Click the Firmware Image button and navigate to the firmware image file.
6. Click the FPGA Image button and navigate to the image file.
7. Click the Burn SD Card button.
8. If a prompt indicating that burning the firmware failed appears, confirm that it states the verification information passed and click OK.
9. Insert the SD memory card into the USRP2 radio and power on.

3.3 Configure the Ethernet Card

If you are using the Ubuntu operating system, to configure the Ethernet card for your USRP2 hardware, perform the following tasks:

1. Select System → Preferences → Network Connections
2. Select the network card to which the USRP2 will be attached. If you are not sure which network card the USRP2 is attached to, you can use `ifconfig` command to find it out.
3. Click the "Edit" button
4. Go to the "IPv4 Settings" tab
5. Select "Manual" for "Method", and click the "Add" button
6. Use the following addresses:
   - Address: 192.168.10.1
   - Netmask: 255.255.255.0
   - Leave Gateway blank

3.4 Modify the Iptables

Iptables is a firewall, installed by default on all official Ubuntu distributions (Ubuntu, Kubuntu, Xubuntu). When you install Ubuntu, iptables is there. However, in order to use the USRP2 Transmitter and Receiver Blocks, you need to modify the Iptables to allow all traffic. You can access the Iptables on your computer by typing the following commands on the terminal:

- sudo bash
- cd /root/scripts/
- emacs iptables.sh

Now, you should have the Iptables open in front of you. You only need to modify two lines concerning the default firewall policies:

- IPTABLES_COM -P INPUT **ACCEPT**
- IPTABLES_COM -P FORWARD **ACCEPT**

When this modification is done, you should save the Iptables and execute it by typing the following command on the terminal:

- ./iptables.sh

Having done the operations in Section 3.1 through Section 3.4, Simulink should be able to recognize your USRP2 board. In order to verify, open a USRP2 Transmitter Block or USRP2 Receiver Block from Simulink Library Browser and see whether the information of your USRP2 board appears in Hardware part, as shown in Fig. 14 and Fig. 16. If Simulink is not able to recognize your USRP2 board, it will show that "No USRP2 found at the specified IP address."

4 Interaction between Simulink and USRP2 Hardware

The USRP2 blocks for the Communications Blockset allow you to interact with the USRP2 Software Defined Radio peripheral from within a Simulink model. This capability enables you to model and deploy your communications algorithms within the context of Simulink, with the benefit of real-time signal I/O for Software Defined Radio (SDR). The USRP2 Transmitter and Receiver blocks support communication between Simulink and a USRP2 board, allowing simulation and development of various software-defined radio applications. These two blocks enable communication with a USRP2 board on the same Ethernet subnetwork. The following block diagram illustrates how Simulink, USRP2 blocks, and USRP2 hardware interface.

4.1 USRP2 Transmitter Block

This block accepts a column vector input signal from Simulink and transmits signal and control data to a USRP2 board using User Datagram Protocol (UDP) packets. Although the USRP2 Transmitter block sends data to a USRP2 board, the block acts as a Simulink sink [7].

For your convenience, the mask parameters of the block are listed below from [7]:

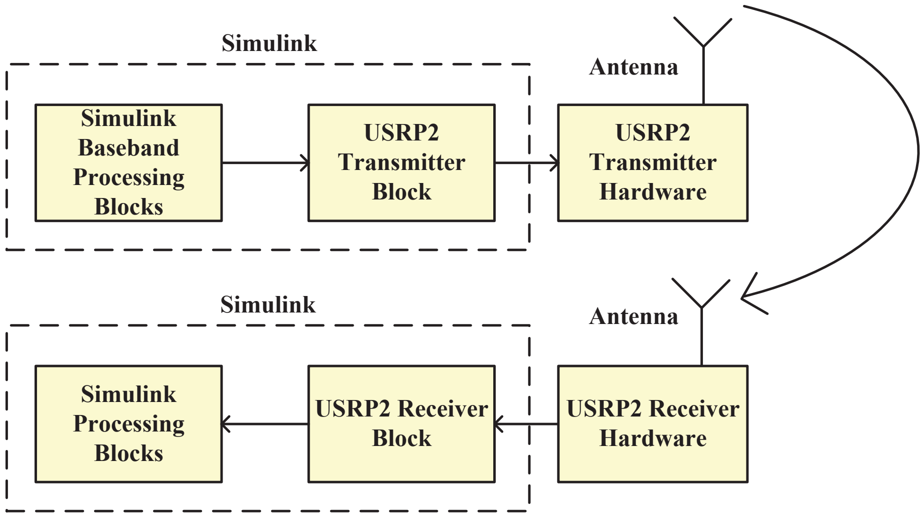


Figure 13: Simulink, USRP2 blocks, and USRP2 hardware interface

- **USRP2 IP address**: This parameter contains a string that specifies the logical network location of the USRP2 board. This field displays a list of subnets corresponding to each network interface. Alternatively, you can specify a known IP address by entering it directly into this field.

- **Host data port**: Specify the UDP port of the host data connection. This parameter defaults to 30000. The range of valid values on a Windows system is 1 to 65535. The range of valid values for a Linux system is 1025 to 65535. When you use the USRP2 Transmitter and Receiver blocks in a model with one USRP2 device, the values identifying host data ports on each block must match.

- **Host control port**: Specify the UDP port of the host control connection. This parameter defaults to 30001. The range of valid values on a Windows system is 1 to 65535. The range of valid values for a Linux system is 1025 to 65535. When you use the USRP2 Transmitter and Receiver blocks in a model with one USRP2 device, the values identifying host control ports on each block must be different.

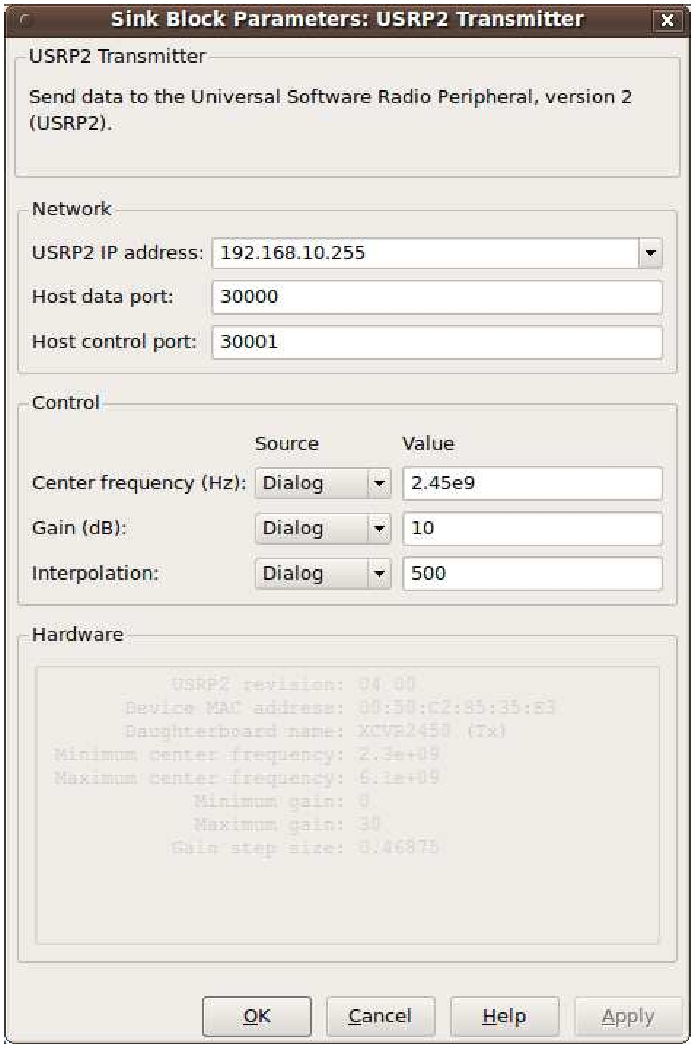
- **Center frequency (Hz)**: Specifies the center frequency of the signal out of the USRP2 hardware RF front end. You can specify this parameter using the block dialog mask or a block input port.

- **Gain (dB)**: Specifies the gain that the block applies to the USRP2 hardware RF front end.

- **Interpolation**: Specifies the interpolation factor for the USRP2 receiver. Select an integer value within the following ranges:

  1. 1 to 128
  2. 128 to 256 (values in this range must be even)
  3. 256 to 512 (values in this range must be evenly divisible by 4)

For an example of a model that uses this block, enter `commusrp2_tx` at the MATLAB command line. You will see the built-in Simple Tx model as shown in Fig. 15.


Sink Block Parameters: USRP2 Transmitter

USRP2 Transmitter

Send data to the Universal Software Radio Peripheral, version 2 (USRP2).

Network

USRP2 IP address: 192.168.10.255

Host data port: 30000

Host control port: 30001

Control

	Source	Value
Center frequency (Hz):	Dialog	2.45e9
Gain (dB):	Dialog	10
Interpolation:	Dialog	500

Hardware

```
            USRP2 revision: 04 00
        Device MAC address: 00:50:C2:85:35:E3
        Daughterboard name: XCVR2450 (Tx)
  Minimum center frequency: 2.3e+09
  Maximum center frequency: 6.1e+09
              Minimum gain: 0
              Maximum gain: 30
            Gain step size: 0.46875
```

Figure 14: USRP2 Transmitter Block

4.2 USRP2 Receiver Block

This block receives signal and control data from a USRP2 board using User Datagram Protocol (UDP) packets. Although the USRP2 Receiver block receives data from a USRP2 board, the block acts as a Simulink source that outputs a column vector signal of fixed length [6].

In addition to the parameters introduced in Section 4.1, there are several other mask parameters you need to know about USRP2 Receiver Block:

- **Decimation**: Specifies the decimation factor for the USRP2 receiver. The USRP2 hardware uses this information to downconvert the Intermediate Frequency (IF) signal to complex baseband. Select an integer value within the following ranges:
  1. 1 to 128
  2. 128 to 256 (values in this range must be even)
  3. 256 to 512 (values in this range must be evenly divisible by 4)
- **Enable overrun output port**: Select this check box to display the overrun port. This port indicates if one or more packets is dropped during the USRP2 transmission to the host.

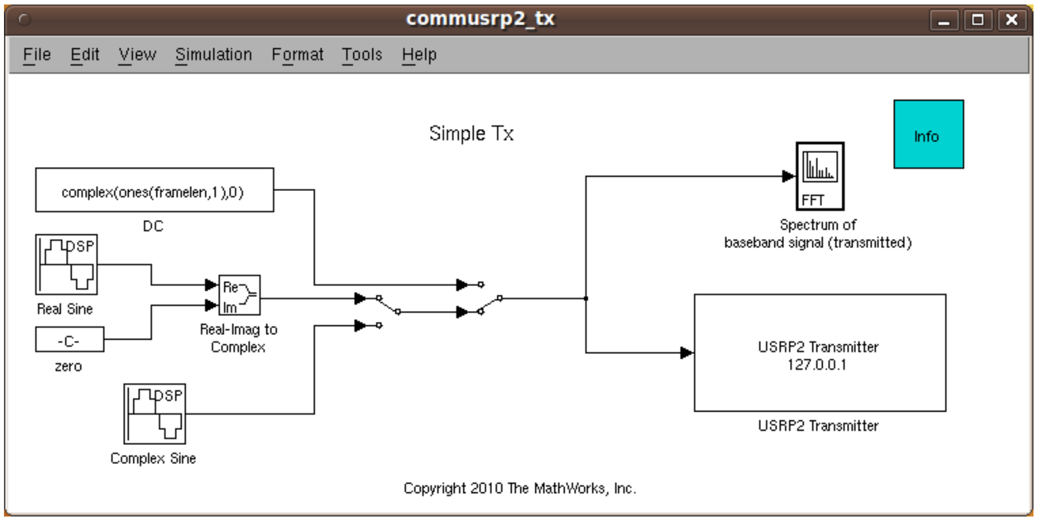

commusrp2_tx
File Edit View Simulation Format Tools Help
Simple Tx
Info
complex(ones(framelen,1),0)
DC
Real Sine
-C-
zero
Re
Im
Real-Imag to Complex
Complex Sine
FFT
Spectrum of baseband signal (transmitted)
USRP2 Transmitter 127.0.0.1
USRP2 Transmitter
Copyright 2010 The MathWorks, Inc.

Figure 15: Simple Tx model.

- **Sample time**: The standard sample time that shows on all Simulink sources.
- **Output data type**: Select the data type of the output signal. This block supports the following complex output data types:
  1. Double-precision floating point
  2. Single-precision floating point
  3. 16-bit signed integers

For an example of a model that uses this block, enter `commusrp2` at the MATLAB command line. You will see the built-in Simple Rx model as shown in Fig. 17.

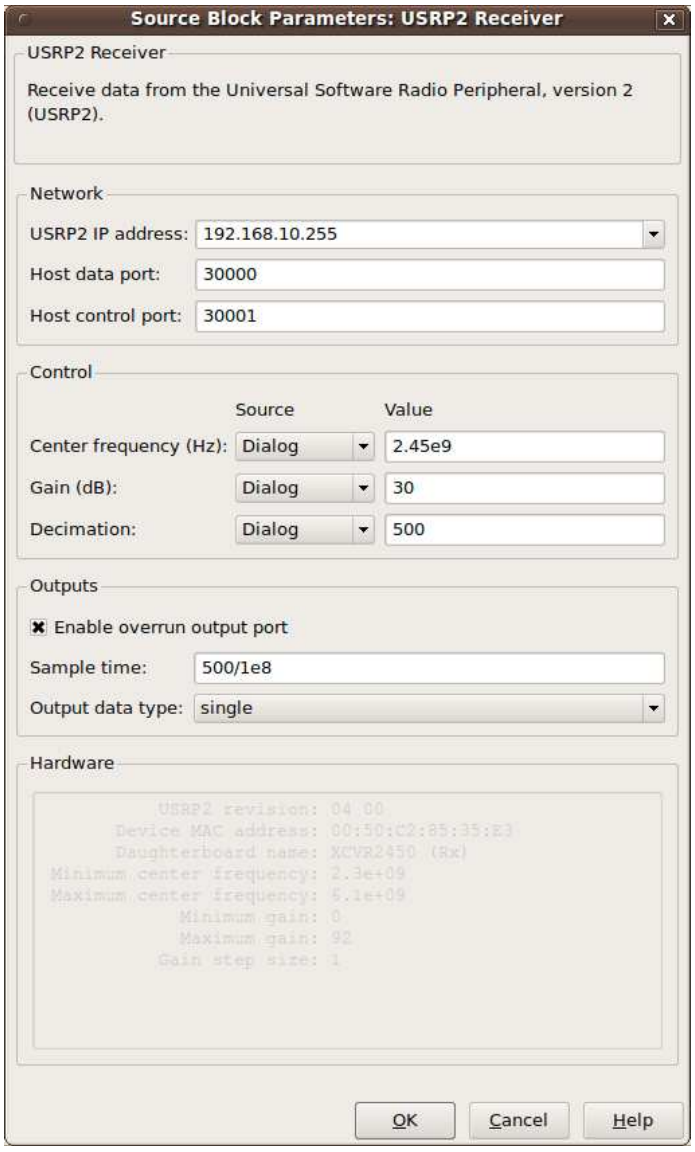
4.3 Frame Size and Dropped Packets

The Simulink USRP2 Receiver block has a fixed output frame size of 358. It has three outputs: Data, Data Length, and Overrun. The data reads from the USRP are non-blocking. The block will try to read data from the socket on every sample time hit. If there is less than 358 samples of data ready, it will output a Data Length of 0. If there are 358 samples or more, it will output 358 samples on Data and set Data Length to 358. If Simulink is running faster than real time, then the pattern for Data Length should be something like: 358, 0, 0, 0, 0, 0, 0, 358, 0, 0, 0, 0, 0, ...

If Simulink is running slower than real time, then the pattern for Data Length will be: 358, 358, 358, 358, 358, ... When whatever socket buffering exists becomes exhausted, then packets will end up being dropped. Whenever one or more packets are dropped, the Overrun output will be 1. You can always attach a Scope to the Overrun output to observe whether there is any dropped packets during your simulation, as shown in the `commusrp2` demo.

5 Conclusion

By introducing MATLAB, Simulink, USRP2 hardware and USRP2 blocks, this tutorial has provided you with all the necessary tools you will be using in the next five laboratory experiments. Specifically,


Source Block Parameters: USRP2 Receiver

USRP2 Receiver

Receive data from the Universal Software Radio Peripheral, version 2 (USRP2).

Network

USRP2 IP address:	192.168.10.255
Host data port:	30000
Host control port:	30001

Control

	Source	Value
Center frequency (Hz):	Dialog	2.45e9
Gain (dB):	Dialog	30
Decimation:	Dialog	500

Outputs

[✓] Enable overrun output port

Sample time:	500/1e8
Output data type:	single

Hardware

```
         USRP2 revision: 04 00
     Device MAC address: 00:50:C2:85:35:E3
     Daughterboard name: XCVR2450 (Rx)
Minimum center frequency: 2.3e+09
Maximum center frequency: 5.1e+09
           Minimum gain: 0
           Maximum gain: 92
         Gain step size: 1
```

Figure 16: USRP2 Recevier Block

you should now be familiar with the following items:

1. Fundamental knowledge and useful tools of MATLAB.
2. Build and simulate a model in Simulink.
3. Important components of USRP2 hardware, especially FPGA.
4. The function and parameters of USRP2 Transmitter and Receiver Blocks.
5. The interaction between Simulink and USRP2 hardware.

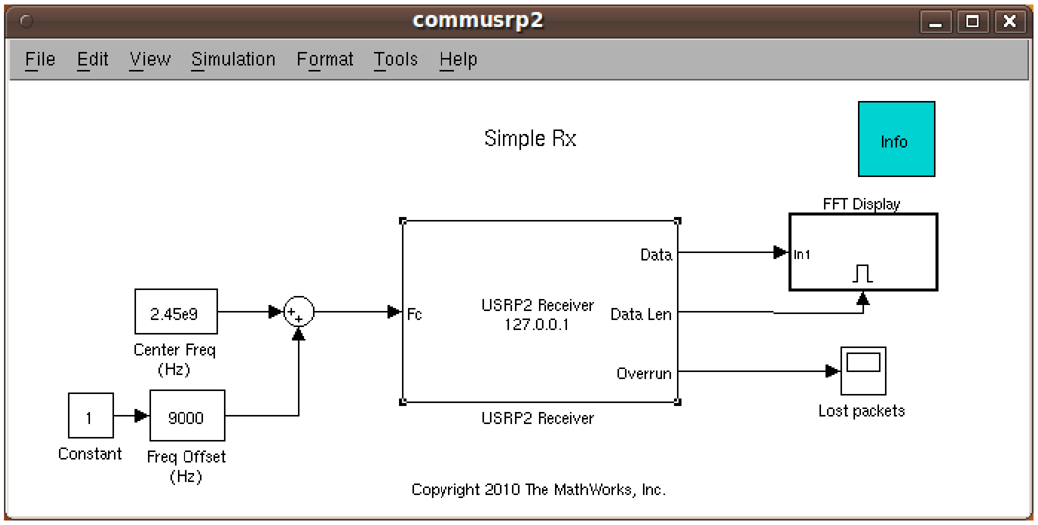


Figure 17: Simple Rx model.

References

[1] Ettus Research Website. `http://www.ettus.com`.

[2] Ettus Technical Documentation and Schematics. `http://code.ettus.com/redmine/ettus/projects/public/documents`.

[3] `http://gnuradio.org/redmine/wiki/gnuradio/USRP2GenFAQ`.

[4] `http://zone.ni.com/devzone/cda/tut/p/id/3983`.

[5] `http://www.mathworks.com/support/solutions/en/data/1-CUN7JZ`.

[6] `http://www.mathworks.com/help/toolbox/commblks/ref/usrp2receiver.html`.

[7] `http://www.mathworks.com/help/toolbox/commblks/ref/usrp2transmitter.html`.