Lab 2: Hierarchical Design and Linear Systems

Objectives:

1) To learn how to build and mask subsystems in Simulink.
2) To simulate linear systems and observe their properties.

Introduction:

In circuit modeling, a network of elements such as resistors, capacitors, and inductors compose a system with the properties of linearity and time-invariance. This lab is concerned with the property of linearity, which states that a weighted sum of any number of inputs, when sent through the system, will produce an output exactly the same as if each individual input was sent through the system one-at-a-time and the outputs added together with the same weights. In mathematical terms, if input $x_1(t)$ results in output $y_1(t)$, and input $x_2(t)$ results in output $y_2(t)$, the output due to $x(t) = a_1x_1(t)+ a_2x_2(t)$ (where a_1 and a_2 are scalar constants) will be:

$$y(t) = a_1y_1(t)+ a_2y_2(t)$$

A key feature of Simulink is that it allows us to make hierarchical designs. This means that we can compose a complicated system that consists of many individual "subsystems", The main reason for doing this is that it encapsulates the underlying details of each subsystem so that only the top level of the design is visible to the user. In doing this, the model window becomes less cluttered, and the end-user is protected against accidentally making unwanted changes to the system inside.

Preliminary:

1) Consider the input $x_1(t) = 2[u(t) - u(t-3)]$
   Find $y_1(t)$ for $h_1(t) = 3\, e^{-2t}\, u(t)$
2) Consider the input $x_2(t) = 4\, u(t)$
   Find $y_2(t)$ for $h_2(t) = 5\, e^{-3t}\, u(t) - 2\, e^{-5t}\, u(t)$

(Hint: Use Laplace transforms to convert x(t) and h(t) to the s-domain, then multiply.)

Post-Lab Questions:

Q1. What happens if the given code for the popup dialog is not entered at all?

Q2. What problems will occur if the signal names in the popup dialog do not match those defined in the callback code?

Q3. What happens if the order of the parameters is changed to be different from Figure 3? (You may test this by using the "move up" and "move down" options).

Q4. Define a possible function for h(t) that would NOT be linear.

Procedure:

Part A – Designing a Hierarchical System

To begin, download the file "*Lab2.mdl"* from Blackboard and open it. The model shown in Figure 1 will open:

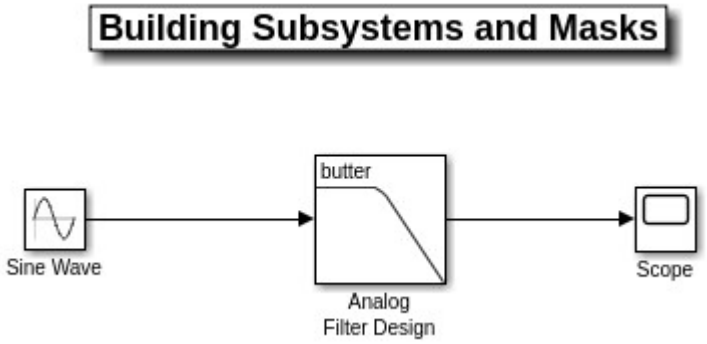

Figure 1 - Initial model

Simulate the given model and observe the output on the scope. It is important to note that, unlike the previous lab, this lab uses *continuous-time* solvers. In this mode, Simulink solves the system by solving a set of differential equations symbolically, as opposed to approximating the system using discrete methods. This allows us to simulate analog systems to a high degree of accuracy. Sample times do not exist in this mode, and any block that requires discrete-time operation will not work properly.

(If you try to connect continuous blocks directly to discrete ones, Simulink will automatically change the solver and the whole system will become discrete.)

To start, we want to add some more signal sources to the model. Open the Simulink libraries and search for a *Chirp Signal*, a *Step* source, and a *Random Number* source, then add all of these to your project. Next, find and insert a *Multiport Switch* bloctransfer fcnk and a *Constant* block into your model. The subsystem will consist of these various sources as inputs, with the switch and constant for selecting between them. Connect all inputs to the multiport switch and the constant to the selector input of the switch as shown in Figure 2. In order to make a subsystem, select all of these blocks (all blocks except the filter and the scope), then right click and choose *Create Subsystem from selection.* All selected blocks will be replaced with a single block containing all the functionality of the original blocks.

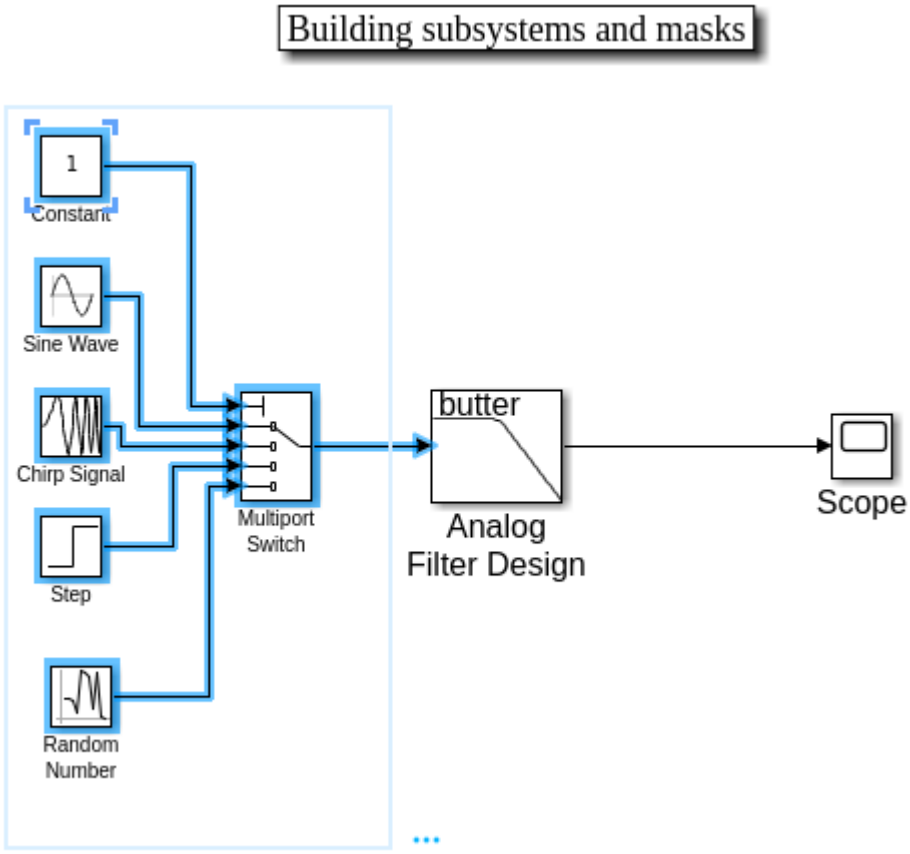

Figure 2 - Select the create subsystem option

Rename the resulting subsystem as *"Input"* by clicking the text box beneath the block. You can reveal and change the original components of this new block by double-clicking it. However, it is much more elegant to create a user interface to control the various settings than to open the block and change the parameters inside every time. For example, one way to change the selected source is to go into the block and change the constant C manually, but this can be time-

consuming for frequently-used blocks, and it also brings the hazard of accidentally changing something you didn't want to change. The solution to this problem is called *masking* the subsystem, which creates a handy user interface much like the ones you see when you open the pre-made blocks to change the parameters.

In order to create a mask for the subsystem, right-click on the subsystem block and select *Mask > Create Mask*. A window will open that will allow you to specify all of the details necessary to fully define a custom user interface for your new subsystem.

1) The first tab in this window defines the behavior of the block icons and the input-output ports. Under *Icon Drawing Commands*, we can represent the block graphically using common MATLAB functions such as *disp*. Use a simple command such as:

   ```
   disp('Input Signal');
   ```

   to draw the name of the subsystem on the body of the block.

2) The next tab, *Parameters & Dialog*, is where we define the parameters of the system that can be changed dynamically by the user. Obviously, we require an option for the user to select which of the input sources to use, but we also require many other parameters related to the source selected. For example, the sine wave requires an amplitude, frequency, and phase to be defined, whereas the random source requires a mean and variance. We shall make the mask dynamic such that, for each source selected, only parameters related to that source will appear to the user for modification. To accomplish this, first we edit the window to look as given in Figure 3.

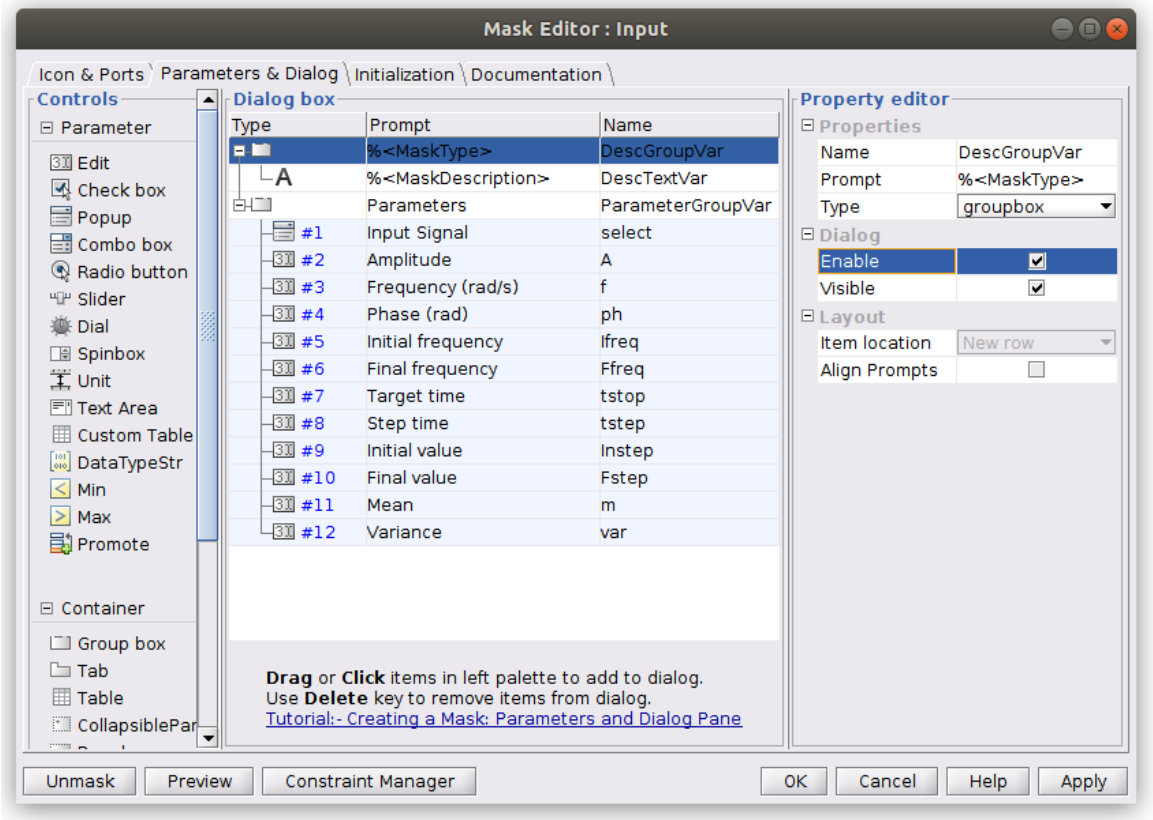

Figure 3 - Parameter pane in the mask window.

All parameters except for the input signal are considered “edit boxes”, which mean that the user simply enters whatever value they want for these variables when they open the settings panel. For the choice of input source, however, we want a popup menu that lists the various sources. To enter the possible choices, type the following lines into the *Popup options* under *Properties*:

Sine Wave
Chirp Signal
Step Input
Random Number

3) Besides the input signal, each of these parameters relates to only one source. We do not want parameters to appear that are unrelated to the source that is selected. For example, if the chirp signal is selected, the user should not be able to see and modify the mean or variance variables. In order to control which variables are visible for each source, we require a bit of MATLAB code that will execute whenever the input source is changed. Highlight the Input Signal parameter, then open the *callback editor box* under *Dialog*, enter the following code:

```
c=get_param(gcb, 'MaskValues'); % Get mask parameter values

% Check the value selected in the pop-up menu and change visible
parameters
if strcmp(c{1},'Sine Wave')
      set_param(gcb,'MaskVisibilities',
      {'on';'on';'on';'on';'off';'off';'off';'off';'off';'off';'off';'of
      f'})
end
if strcmp(c{1},'Chirp Signal')
      set_param(gcb,'MaskVisibilities',
      {'on';'off';'off';'off';'on';'on';'on';'off';'off';'off';'off';'of
      f'})
end
if strcmp(c{1},'Step Input')
      set_param(gcb,'MaskVisibilities',
      {'on';'off';'off';'off';'off';'off';'off';'on';'on';'on';'off';'of
      f'})
end
if strcmp(c{1},'Random Number')
      set_param(gcb,'MaskVisibilities',
      {'on';'off';'off';'off';'off';'off';'off';'off';'off';'off';'on';'
      on'})
end
```

(Note: The names in the code must match the names in the popups tab EXACTLY)

4) The third tab, *Initialization*, is where we can define the initial values for any the parameters in the mask. However, the catch is that this code executes every time you run the simulation, which means that any values you chose for the parameters in the dialog box prior to running the model will be overwritten by this code. For that reason, leave this blank for now.

5) The last tab, *Documentation*, contains areas to enter helpful information about your mask for any user who might use it in the future. Since this is a tutorial lab, you may leave this blank for now. Click OK to save the changes to the mask. With the mask completely defined, our last step is to make it so that the variables actually correspond to parameters in the constituent blocks. Right-click on the subsystem and select *Look Under Mask* to see the original blocks making it up. Inside each source, we want to make the value of each parameter dependent on the variables we defined. To do this, simply replace the value in each parameter with the corresponding variable. For example, in case of the sine wave you would have replaced the amplitude value with ‘A’, the frequency with ‘f’ and the phase with ‘ph’. The value of the constant that switches between sources will become ‘select’.

To test your subsystem, double click the block and look at the parameters that are visible to change. Depending on which source you select, the available parameters should change. Experiment with different sources and different values for the parameters, then take a screenshot of the scope output for each source (pick any parameters you want) and include them in your report. Comment on how the low-pass filter affects each type of source.

Part B – Simulating Linear Systems

Next, we are going to simulate the linear systems given in the preliminary. Start a new, blank model and find the *Signal Builder* block. When you add the block and open the settings panel, a screen will open which will allow you to create a time-domain signal from points. Use this tool to create the input signal $x_1(t)$ from the preliminary. Next, find the *Transfer Fcn* block in the libraries and place it. This block is used to filter a system given its transfer function in the s-domain. During the preliminary, you should have found the Laplace transforms of h_1 and h_2. Enter the expression for $H_1(s)$ in this block, and connect it to the input source. Finally, attach a scope to the output. Include a printout of the scope output in your report, and observe how close the output comes to your calculated value of $y_1(t)$ in the preliminary exercise.

To demonstrate the properties of linearity, we are now going to modify this model. Replace the signal builder source with the two separate step sources that comprise it. Duplicate the transfer function block, then connect each step function to a separate instance of $H_1(s)$. Subtract the two outputs using a *Sum* block, then connect the final result to the scope. If the system is linear, the final result should be exactly the same as before. Include a printout of the scope output in your report, then observe whether or not it is identical to the output in the previous step.

Finally, use the knowledge from the previous steps to construct $x_2(t)$ and run it through the system $H_2(s)$. Include the final output in your lab report, and observe how closely it matches the theoretical output for $y_2(t)$ you calculated in the preliminary. For extra credit, you may make a masked subsystem containing x_1, x_2, h_1, and h_2, with the entire block connected to the scope. In the settings for the block, allow the user to choose between either input, and between either system. The block should output the result of the chosen input filtered by the chosen system. If you choose to do this, include printouts of the final model, the contents of the subsystem, and the mask parameters window in your lab report.