Lab 3: Simulink Libraries and Amplitude Modulation

Objectives:

1) To learn to build libraries in Simulink.
2) To simulate Amplitude Modulation (AM) and demodulation.

Introduction:

A useful feature in Simulink is the ability to create user-made libraries. For example, if you implement a certain modulation scheme using various blocks and wish to use this scheme in several different models down the road, you can create a library with all of the blocks you need, then open it later and pull out the blocks you need just as you do with the built-in Simulink libraries.

Modulation, by definition, is a process by which a certain characteristic of a fixed carrier wave is varied in accordance with an information-bearing signal. The primary motivation for modulation is to facilitate transmission of the information-bearing signal over a communication channel with a specific frequency range. For example, a human voice may have frequencies between 100 and 3000 Hz, but if this signal were to be transmitted via RF directly at those frequencies, only one voice could be transmitted at a time without interference (and it would require an antenna the length of the United States). By modulating and transmitting the signal at a much higher frequency, we can communicate over a much wider variety of channels, and also share these channels with other signals.

Amplitude modulation refers to the manipulation of the amplitude of a carrier wave in some manner to transmit the required information. In the simplest case, the information signal is multiplied by the carrier wave, creating a new signal that has frequencies above and below the carrier wave, but none at the original frequencies of the message signal. This is very useful for a wide array of applications, since we can effective choose which frequency range our message gets transmitted over.

Preliminary: Assume your message is a sine wave and AM modulation index is $\alpha = 0.5$. Find the modulation efficiency of the AM scheme.

Procedure:

Part A – Designing a Simulink Library

To begin, open a new library by choosing *New -> Library* from the *File* menu. This top-level window is the area where our final user-made blocks will reside, ready to be taken and used in another program. The basic principle is that, just as we did in Lab 2, we will build a system out of the default blocks and then enclose it all in a subsystem to create a user-made block. The main difference is that, in a library, it is only a collection of these blocks. You cannot simulate anything within a library file; you must take the blocks out and place them into a model file if you wish to simulate anything.

For this lab, we will be constructing a block that performs DSB-AM modulation on a given signal. The defining equation for AM modulation in this manner is:

$$x(t) = A(1 + \alpha m(t))cos(2\pi f_c t), \quad (1)$$

Where A is the amplitude of the carrier, f_c is the frequency of the carrier, α is the modulation index, and *m(t)* is the *normalized* message signal (i.e. it is modified to occupy the range between -1 and +1). This subsystem will consist of one input, one output, and three user-changeable parameters. In order to create this input and output, search for the blocks *In1* and *Out1* in the Library browser and add them to your library. You may rename these components m(t) and x(t), respectively, for clarity.

The first part to build in this model is the normalizer, which takes the input message signal m(t) and constrains it to fall between -1 and 1. Ordinarily, normalization would entail dividing every sample of the signal by the absolute maximum value the signal would take. However, this is somewhat tricky to implement in Simulink, and it runs into trouble where noisy signals are concerned. For the purposes of this lab, we will assume that you have normalized the input message signal to avoid over modulation. We will however *clip* the signal in case it falls outside the range of -1 to +1. This can be accomplished with the *Saturation* block in the library. Add this block to your model and set its limits accordingly.

After clipping, you must implement the above equation using blocks from Simulink. You should have the necessary knowledge to build a system that computes Equation (1). When defining the parameters for the blocks, we may allow the user to change three of them: the amplitude *A*, the carrier frequency *fc*, and the modulation index *alpha*. Make sure to define these parameters as variable names rather than constants. When you are finished, create a subsystem out of these

blocks and define a mask that will allow the user to change the three parameters that were defined by variables. Your finished block should appear like Figure 1.

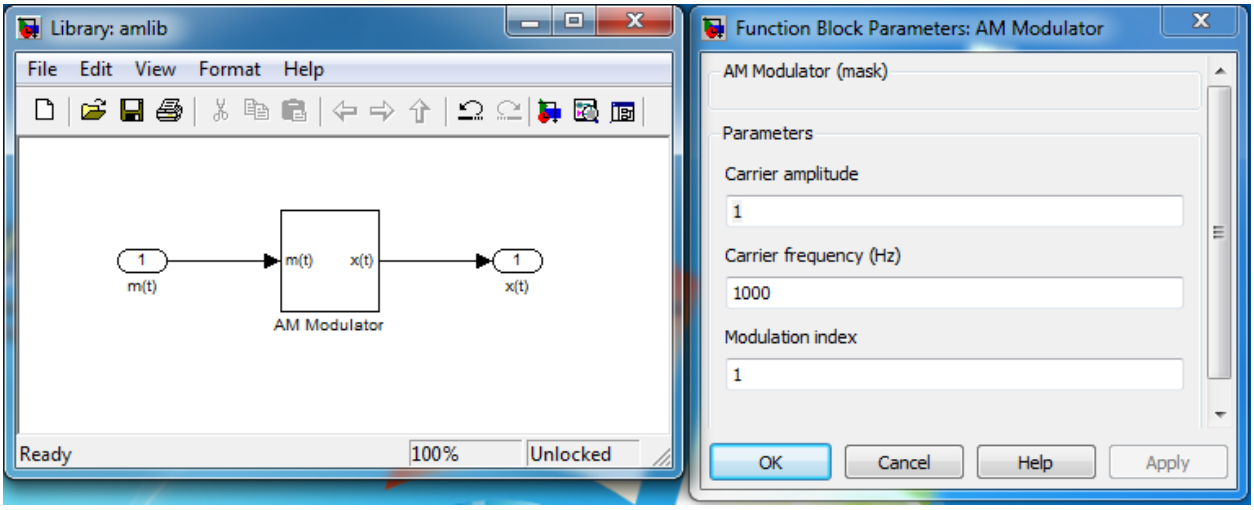

Figure 1 – Finished AM modulator block

Part B – Using a Simulink Library

After creating the above block, save the library as "am_lib.mdl". Now, open a new model file, set the solver to discrete time, and save it as "am_sim.mdl". This model file will be where we test the modulator built in the previous step. Drag the modulator block you just designed into the new model. In this way, user-made libraries are much like the ones that are built into Simulink; you just drag them into model files as needed.

To test this model, create a sine wave block and connect it to the input of the modulator. Set its amplitude to 1 and its frequency to 1 Hz. Now, in the modulator parameters, set the carrier amplitude to 1 and the modulation index to 0.5. Set the carrier's frequency to around 20 times the frequency you set for the source sine wave. Connect a scope to the output, then set it to accept two inputs so that you can view the original source sine wave on it at the same time. Set the max step size to 0.001 and the simulation run time to 2 seconds, then run it. After autoscaling the axes, you should get an output that looks like Figure 2.

Save a screenshot of this scope output, then change the modulation index to 0.8 and run it again. Also save screenshots of the scope outputs for modulation indices of 1 and 1.5. Comment on the effects the modulation index has on the modulated signal. Finally, set the modulation index back to 1, but set your sine wave source amplitude to 2. You should notice the clipping effect on the final output.

For extra credit, you may connect an FFT block to the output of the modulator and observe the resulting spectrum. There should be a large impulse at the carrier frequency, along with two smaller ones on either side which correspond to the message signal.

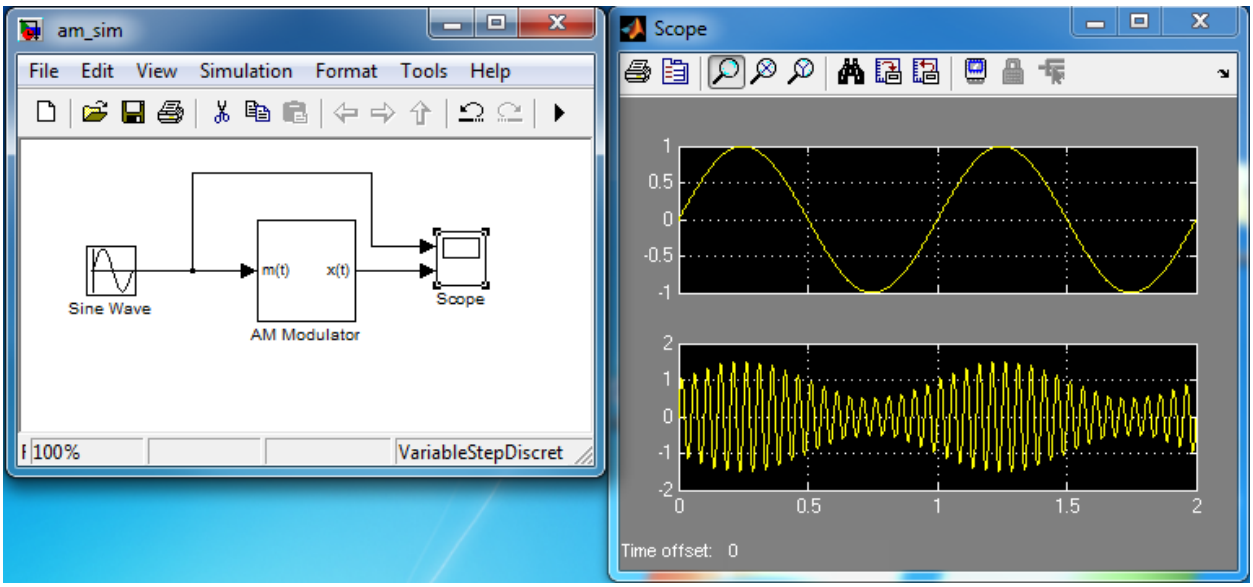

Figure 2 – Output of AM modulator with α=0.5

Part C – AM Demodulation

Return to the library file you created in part A. We are now going to create a demodulator block and keep it in the same library file. It is important to note that, if you closed the library and then reopened it, it is now *locked*, meaning it won't let you make changes to it. In order to change it again, you must choose *Edit –> Unlock Library* from the top menu.

An AM demodulator can be constructed by multiplying the output of the modulator by another cosine wave equal to the frequency of the original carrier, then low-pass filtering the result. This is known as a *product detector*. To begin, place the blocks seen in Figure 3 in your library window, then configure the parameters as shown and create a subsystem out of them.

After doing this, create a mask for the subsystem. There will be three user-modifiable parameters: the carrier frequency *fc*, the filter sampling frequency *Fs*, and the filter order *n*. Set these parameters as shown in Figure 4, and set the sampling time of the zero-order hold as 1/Fs, then save the mask. You now have a fully-functional AM demodulator block. Save the library, and then reopen your simulation model file.
(Note: the LP filter coefficients are designed at runtime. Make sure this line of code is entered properly, or the filter will not work.)

```
firpm(n,[0 (1/2*fc)/(Fs/2) fc/(Fs/2) 1],[1 1 0 0])
```

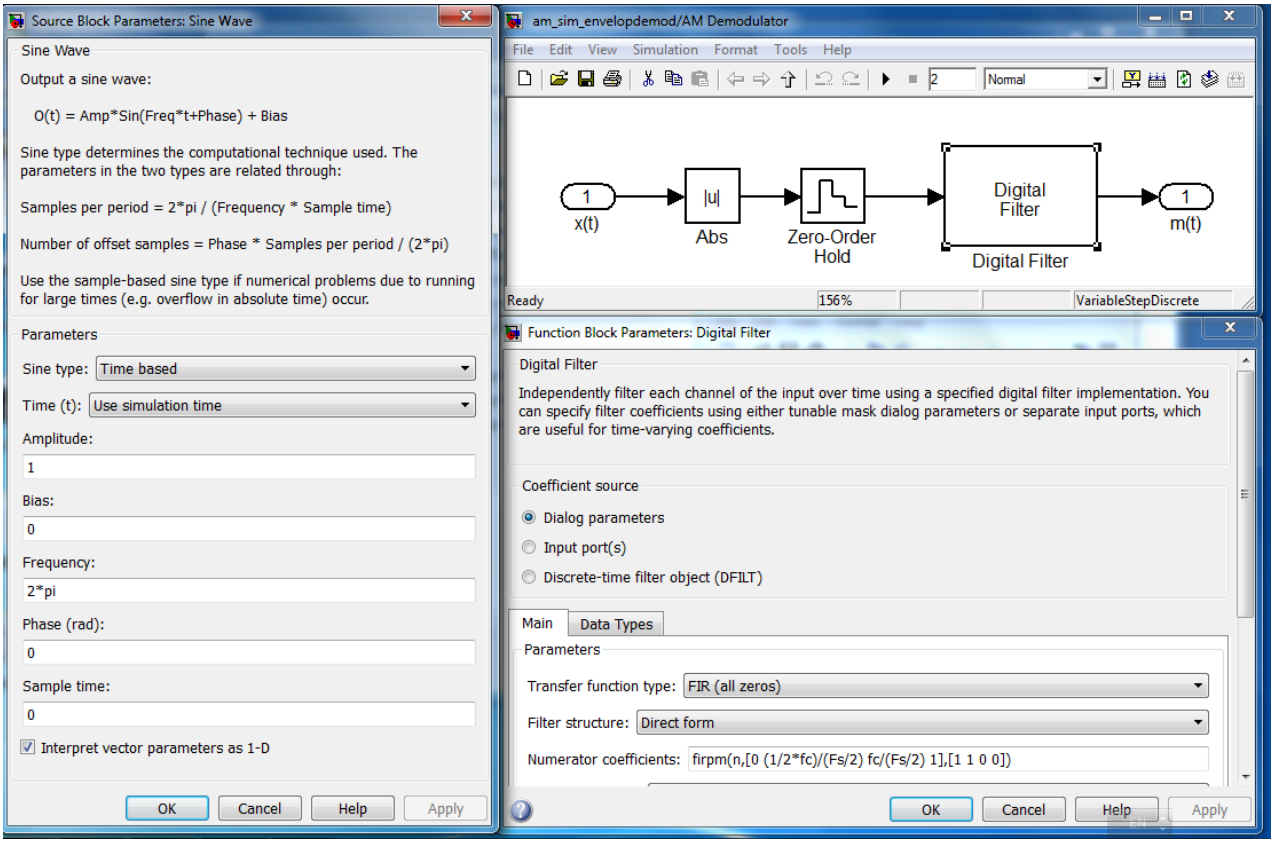

Figure 3 – AM demodulator model with block parameters

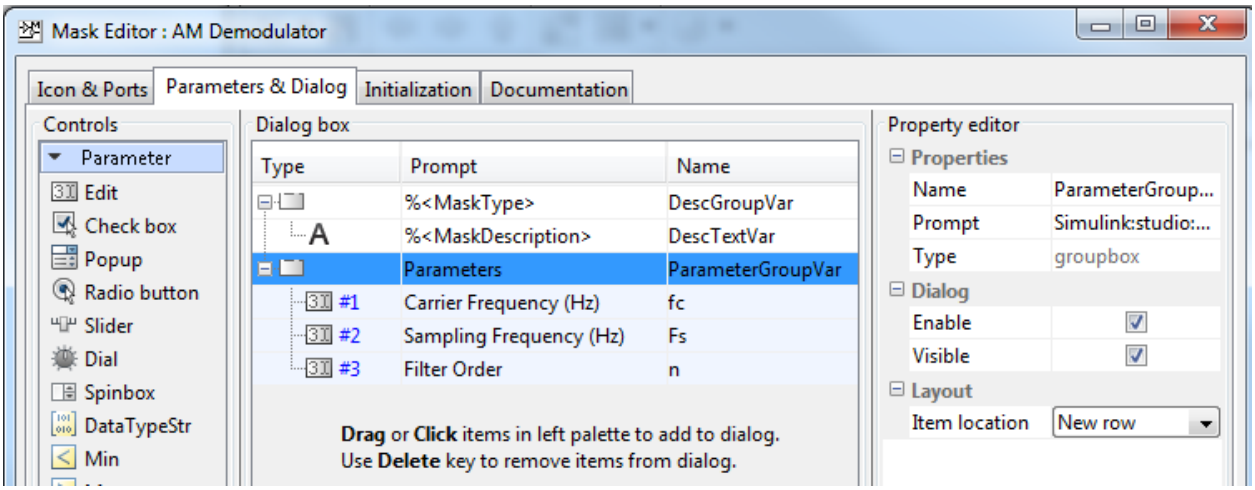

Figure 4 – Mask parameters for AM demodulator

Essentially, this block multiplies the modulated signal by a cosine wave at the carrier frequency, then prepares to filter the result by sampling at regular intervals specified by Fs by using a zero-order hold block. A digital low-pass filter is created using the MATLAB function "firpm", which takes a given set of specifications (in our case, the cutoff frequencies) and attempts to create an

optimal FIR filter of the given order. Just as Simulink can evaluate block parameters that include variables (such as 1/Fs), it can also evaluate functions whenever the model is simulated for the first time. In our case, it was not possible to use the analog Butterworth design block because our model is being executed in discrete time.

After designing the block, drag it into your main model file and connect it to the output of your modulator. Add another input to the scope, then connect it and run the simulation. Using a sampling frequency of 500 Hz and a filter order of 100, you should get results that look similar to Figure 5.

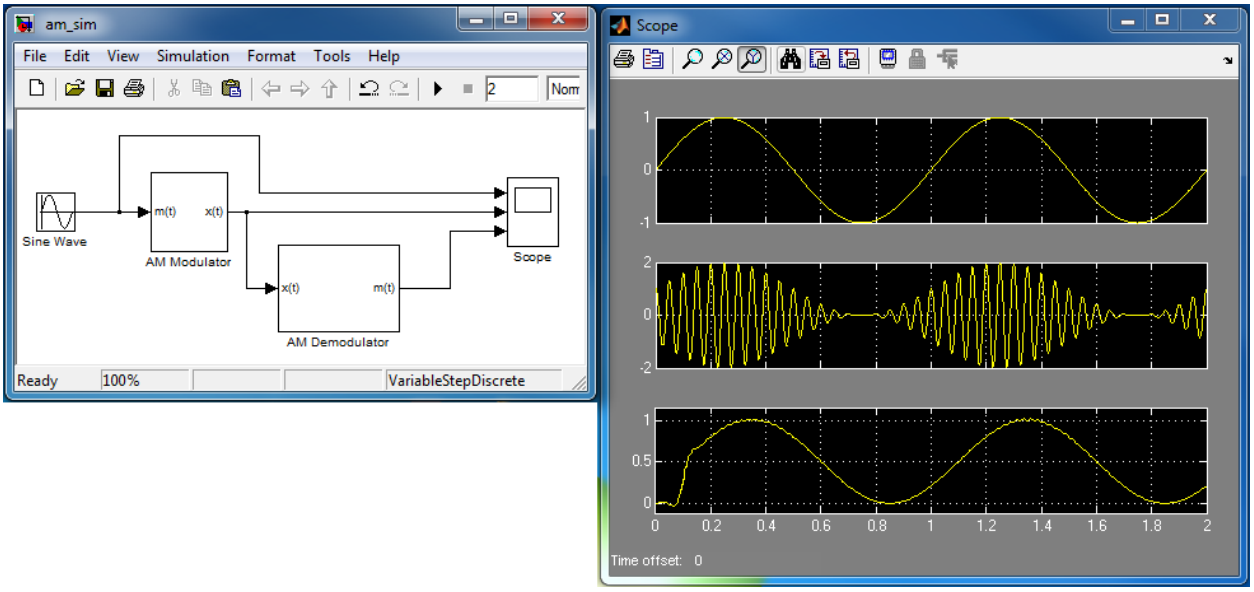

Figure 5 – Final results for AM modulation

The output from the demodulator is going to be distorted at the beginning and slightly delayed when compared to the original input. The reason for this is that an FIR filter consists of a series of shift registers (in our case, 100 of them) that all begin at 0. It takes some time for these registers to be filled, so the filter output will not be correct until a certain amount of time has passed. The higher the filter order, the longer this will take.

After completing the above steps, save your model files and scope outputs and submit them in your report. For extra credit, use the blocks *From Wave File* and *To Wave File* to run a sound file (use "nineoneone.wav" from Blackboard) through your system and record the output. The final output should sound similar to the original input.

Post-Lab Questions (justify your answers by your simulation results):

Q1. In part B, change the carrier signal to a square wave. How does the modulated output change?

Q2. In part B, what are the consequences of setting the modulation index above 1? Will this signal be demodulated properly, or will there be problems?

Q3. What will happen if the modulation index is set too *low*? How will this affect the transmission efficiency in a real system?

Q4. In this lab, we defined the carrier frequency to be around 20 times the message frequency. What will happen if the carrier frequency is not that much higher than the message? Will this cause problems in demodulation?

Q5. Adjust the FIR filter order in part C to different values. What happens to the output if the filter order is set very low? What if it is set very high?