Experiments in Communications

ECE316 - Spring 2009

Bruno Korst-Fagundes, P.Eng.

The Edward S. Rogers Sr.
Department of Electrical and Computer Engineering
University of Toronto

Contents

Chapter 1 - Introduction to the Working Environment

1.1 Introduction

Welcome to ECE316! In this first practice laboratory, you will experiment with the principles and tools which you will utilize extensively throughout the course. This outline will be longer than usual, since it will serve as a guide to `Simulink` and `Code Composer Studio`. It will deal with topics such as the Fourier Transform, frequency shift and noise in simulation and will show you how to run code on a DSP platform. You will find that these are topics which will be referred to multiple times in Communications and Signal Processing courses.

The software tools you will use here are standard in both the university environment and the industry. By working through all of the experiments, you will acquire enough practical experience to utilize state-of-the-art software tools as well as to program a DSP platform compatible with the entry-level expectations of the industry.

The experiments will always be made in groups of two students per workstation. There is no detriment to the learning experience if one works alone, but groups must not exceed two students. We encourage all students to work in groups of two, and no special consideration will be given to those working individually. All experiments are designed with that in mind.

The main tools to be used in this lab are:

- `Matlab`: a mathematics package which includes toolboxes for communications, signal processing, filter design and code generation for specific hardware targets, among other toolboxes;

- `Simulink`: a simulation package which is part of `Matlab` and includes sets of pre-defined blocks that will be used to design, simulate and download the systems required during each experiment;

- *Code Composer Studio*, a DSP software package which allows the user to program, debug and download software to the Texas Instruments DSP platform (TMS320C6713 DSK). This package provides the connection with the target DSP through which one can write and retrieve data during the experiments. It also provides a means to have `Simulink` download automatically generated code to the platform, if needed.

Laboratory skills for assembling and testing simple circuits are highly desired in this lab. In order for you to make the most out of the experiments through the semester, you should already be comfortable with the use of signal generators and oscilloscopes. Some specific details about the signal generators and oscilloscopes will be given to you in the Appendix. The instruments utilized in all experiments can be considered as standard in communications laboratories.

1.2 Background Reading and Preparation

A review of basic block diagrams for the systems involved in the topic of each experiment is essential. You must look at the outline for the experiment and view the introductory video prior to coming to the lab. Do not come to the lab late; if you arrive early, chances are you will be done well within the time allocated to do the lab.

You should prepare for this first experiment by reviewing the core `Matlab` tools that have already been utilized in other courses of your Engineering curriculum. `Matlab` will be the tool used to generate variables for the simulations, and to visualize results whenever necessary. You must be familiar with creating and manipulating vectors and matrices, basic signal processing opearations and different forms of plotting data, for instance. Whenever needed, specific commands related to the hardware will be provided. You **must** be familiar with time domain / frequency domain representation of signals. You should, therefore, review the concepts introduced in the Signals and Systems course.

Since `Simulink` is a simulation package based on block diagrams, prior knowledge of the diagrams involved in every experiment is advantageous. For this first practice, you will be presented with very simple input-output block diagrams to demonstrate the capabilities of `Simulink`.

The **lab preparation** is required and will be marked for this lab. You should have it ready prior to the beginning of the experiment.

1.3 Equipment and Software Tools

The following list of equipment and software tools will be utilized in all experiments.

Hardware:

1. One Signal Generator;
2. One Two-Channel Oscilloscope;
3. One Spectrum Analyzer;
4. One DSP development platform attached to a PC workstation.
5. Coaxial cables BNC-to-BNC.

Software:

1. `Matlab` 7.4 - Release 2007A;
2. `Simulink` with Signal Processing and Communications Toolbox and TI Developer's Toolbox;
3. `Code Composer Studio`, v.3.1.

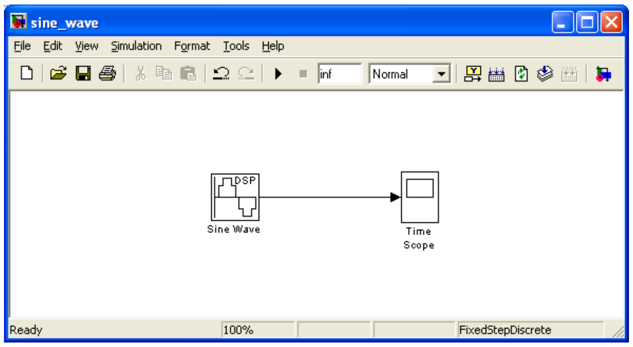
1.4 Introduction to `Matlab` and `Simulink`

Start up by running `Matlab`, and opening `Simulink`. You can open `Simulink` by typing "simulink" at the command line, or by clicking on the `Simulink` icon at the top menu. In the `Simulink` Library, you will find the blocks needed to perform all experiments. This first experiment will encompass the simulation of your workstation, including signal generators (sine and square wave) and oscilloscope, as well as the simulation of a communication channel. The latter will include, for now, the notions of attenuation, noise and band limitation. The basic `Simulink` tools will be reviewed when needed.

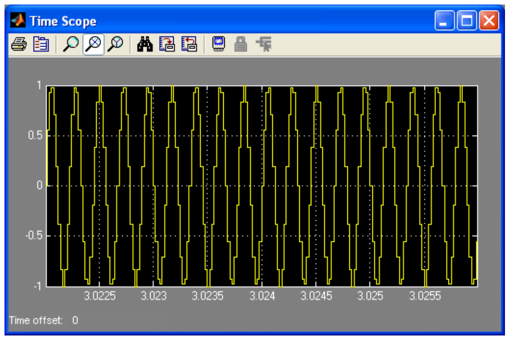
1.4.1 Simulating a Scope

Start by opening a new model (File/New, or click on the little white page on the top left corner). `Simulink` calls “sources” the blocks which simulate signal generators, and “sinks” the blocks which simulate any displaying device such as an oscilloscope or a spectrum analyzer. You will find most of the blocks you need in the `Signal Processing Blockset` within `Simulink`. Drag into the new model a sine wave generator block (`DSP Sine Wave`) and an oscilloscope block (`Time Scope`) from the `Simulink` library. Connect the two blocks. You should have a model looking like Figure 1.1(a).

Double click on the sine wave generator and set the parameters for amplitude 1 and frequency 1500, discrete-time. The amplitude set is a “peak” value for `Simulink`. When using sample-based blocks, you must specify a sampling period. In all experiments, the simulation portion should utilize parameters compatible with the DSP target hardware. For the hardware in the lab, you will utilize a sampling rate of either 96KHz or 48KHz (this is a 1/96000 or 1/48000 sampling period). You will be advised in the outline if the sampling rate is to be changed.



(a) Simulation Model



(b) Time Domain Display

Figure 1.1: Simulation and Time Domain Display for a Sinusoid

You will use discrete-time for all simulations in this course. The sampling period you choose **must be** consistent among all blocks in your system. In order to visualize if all blocks are operating on the same sampling rate, you can go under “Format - Port/Signal Displays” and select the “Sample Time Colours” option. Also, depending on the blocks you are using, you may find that the frequency of the signal you are generating with the block is specified in radians per second, in which case you must multiply the number you chose by 2π (or in `Matlab` code: `2*pi`).

Now run the simulation. You can do that from the menu (under Simulation/start), or by pressing Ctrl-t, or yet by clicking on the right-pointing arrow on the top panel. Intuitively you know that you should see some sort of display showing a sinusoid. If you double-click on the `Time Scope` this is indeed what you will see. `Matlab` will open up a small window resembling the display of an oscilloscope and the sinusoid will be plotted there.

It is likely that the first time you open it up, all you see is some yellow *smudge*. This corresponds to your sinusoid being displayed at the wrong horizontal sweep. Feel free to find the most convenient or meaningful display for you by adjusting the settings on the scope window. You can right-click on the scope display and set the properties and also set the axis properties from a button on the top panel of the new window. If you are changing parameters on other blocks and running your system repeatedly, you are better off finding a configuration for your scope and saving it (under “save current axis”). Your time-domain picture should look like Figure 1.1(b).

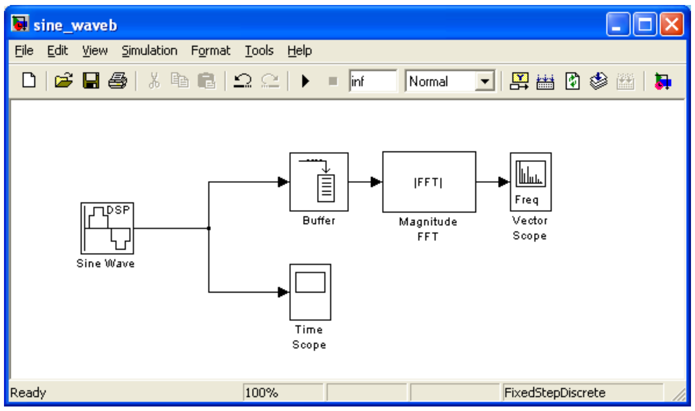
1.4.2 Simulating a Spectrum Analyzer

In every experiment you will visualize the signals in both time domain and frequency domain. The two real devices that will be mimicked by `Simulink` in this fashion are the oscilloscope and the spectrum analyzer. While `Simulink` provides a very good time domain oscilloscope, things are not as straight-forward with its `FFT-Spectrum Scope`. You may be better off creating your own. Below you will find the instructions to do that.

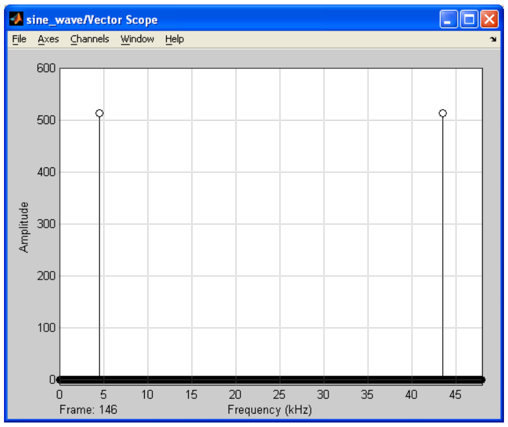
For reasons that you will learn later in your program, in order to perform an FFT you should "buffer" (that is, store) data prior to performing an FFT. The size should be at least as many samples as the number of points of the FFT. Your FFT length must be set to 1024 and you should try to display the output of the FFT from 0Hz (DC) to 48KHz (which is your sampling rate, Fs), realizing that this display is mirrored around Fs/2. That is what you need to know now.

Things are better learned when they are done. Now you will design your own spectrum analyzer in `Simulink`. You can achieve this by using three blocks: a `Buffer`, a `Magnitude FFT` and a `Vector Scope` block. These blocks are all found within the `Signal Processing Blockset`. The `Buffer` block is under `Signal Management - Buffers`. The `Magnitude FFT` is under `Transforms`, and the `Vector Scope` is under `Signal Processing Sinks`. These three will provide you with a good simulated spectrum analyzer. Drag these three into your model, so that you will have a sine wave generator connected to the time domain display as well as your version of a frequency domain display. These displays will be used extensively through the term.

You should now have a system resembling the one on Figure 1.2(a). Double-click on the blocks to adjust the parameters as follows. For your `Buffer` block use 1024 as buffer size, zero overlap and zero initial conditions. Select "Magnitude" as the output to your FFT bock and set 1024 as the FFT length. Finally, on the `Vector Scope`, select "Frequency" as your input domain, click on the tab "Axis Properties" and set the frequency range to [0...Fs] and the Amplitude Scaling to "Magnitude". Leave all other parameters as they are.



(a) Simulation Model



(b) Frequency Domain Display

Figure 1.2: A Model For Visualizing Time and Frequency Domain

Run the simulation again and you should see two "extra" windows: one representing your signal in the time domain (see Figure 1.1(b)) and another representing it in the frequency domain. You should see the picture as represented in Figure 1.2(b), for a 1500Hz, 1 V_p Sinusoid.

There are some things in `Simulink` which annoy you, the TA and everyone. One is the surreptitious data point. When the frequency domain display pops up, go under "channels – markers" and select the little circle. After you do that, two circles will appear on your display, representing data that you could swear it was never there before.

For your own entertainment and good learning, try changing the number of sample points buffered and FFT length. Use always powers of two (you will learn why in another course), and the same number for buffer and FFT length. Choose between 128 and 4096. Observe the changes. You can "run" and "pause" the simulation as you wish, and even save the generated waveforms as a file, if needed (look for the types of `Sinks` on the `Simulink` library). If you find that your simulation is not running long enough, go under "Simulation/Configuration Parameters" and change the number of simulation time units as you wish. The default number is 10.0, but most consider it too short. A better number is 1000 simulation time units, or you can also use `inf`. The idea is to make it run long enough so that you can change parameters and observe the results without having to stop the process.

Note that even though the simulation time is specified in seconds, these do not correspond to the seconds you would measure on your watch; they "emulate" seconds for the purpose of displaying the results. This is to say that the time units displayed on the x-axis on your `Time Scope` will *mean* seconds, even though your watch will tell you something else if you time the simulation. To verify this, measure the frequency of your signal based on the time axis of your scope, just like you would in a real scope (without the "measure" button, of course).

1.4.3 Multiplying Two Sinusoids

This simple simulation will give you the introduction to Amplitude Modulation, which you will later study in this course. From your system simulated above, add another `DSP Sine Wave` and a `Product` block. The latter is found under `Simulink -- Math Operations`. Connect the two sinusoids to the `Product` block and the output of the product to the scopes. Set one frequency to 1500Hz and the other to 12KHz. When you run your simulation, you should what is displayed on Figure 1.3. Remember, this plot is mirrored at its half point.

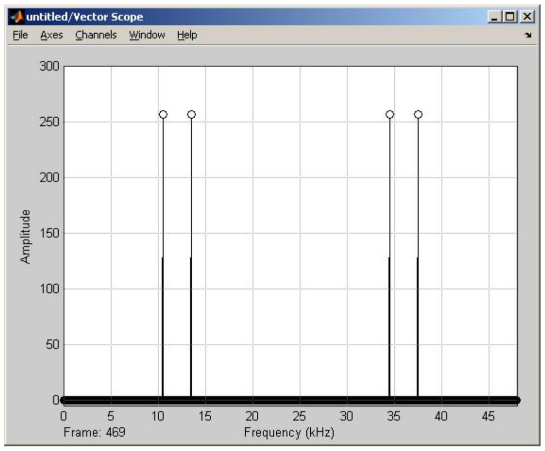


Figure 1.3: 1.5KHz multiplied by 12KHz

Now that you have thought about your high-school trigonometry, consider also the Frequency

Shift Property of the Fourier Transform (page 30 of your text). Now imagine that insteat of a 1.5KHz sinusoid you had a band limited signal. How would the frequency domain look?

1.4.4 Simulating a Square Wave

You will now create a zero-mean square wave generator. The `Pulse Generator` block is found under the `Simulink - Sources` library. The `Gain` and `Add` blocks are under `Simulink - Math Operations`. Finally, the `DSP Constant` block is under `Signal Processing Blockset -- Signal Processing Sources`.

This model is accomplished by setting up a 50% duty-cycle *pulse train* with the correct sampling rate and amplitude; then, you remove the DC component by means of a multiplication (gain) and the subtraction of a constant. This way, if you had a pulse swinging between 0 and $1V_p$, when you multiply it by 2 and subtract 1, your new signal is a square wave which will swing between -1 and 1 V_p. This is accomplished as presented in Figure 1.4. Note, however, that Figure 1.4 shows an "out1" block, which in your model must be substituted by the time and frequency scopes that you have already created.

The `Pulse Generator` parameters must be set. You will need a `Sample-Based`, 50% duty-cycle pulse. Since the sampling rate is 48KHz, a 50% duty-cycle is obtained by setting the pulse width as half of the period, both set in "number of samples". For instance, for a 1KHz square wave to be obtained using 48KHz sampling rate, one will have 48 samples per period and a pulse width of 24 samples.

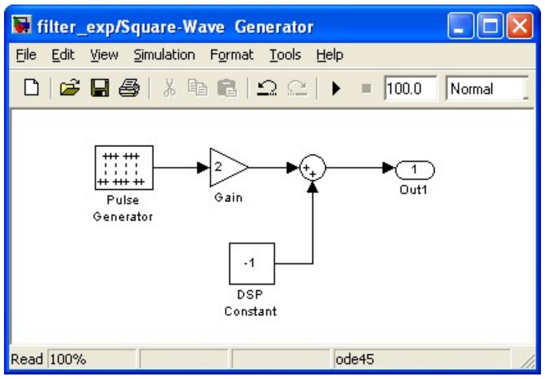

filter_exp/Square-Wave Generator
File Edit View Simulation Format Tools Help
100.0
Normal
2
Pulse Generator
Gain
1
Out1
-1
DSP Constant
Read 100%
ode45

Figure 1.4: Zero-mean square wave generator

Now look at the frequency domain display of the square wave you have just created and see if you can explain it. Look at the Fourier Series you - hopefully - wrote on your preparation. If you increase the frequency of this square wave, what happens to all components?

1.4.5 Signal and Noise

The noise source for this experiment is the `Gaussian Noise` block, found under `Communications Blockset -- Comm Sources -- Noise Generators`. You will include this noise block in your single sinusoid simulation, so that you will have noise added to your signal. You will need a `Gain` block and an `Add` block, both from the `Simulink -- Math Operations`. You will insert the gain on the path between the noise source and the adder, and set the gain to 0.1. The objective here is to look at the effects of noise on the signal, both in time domain and frequency domain. You will experiment with a similar situation when you run code in the next section. Your model should look like the one presented on Figure 1.5. Keep the frequency of the sinusoid at 1500Hz.

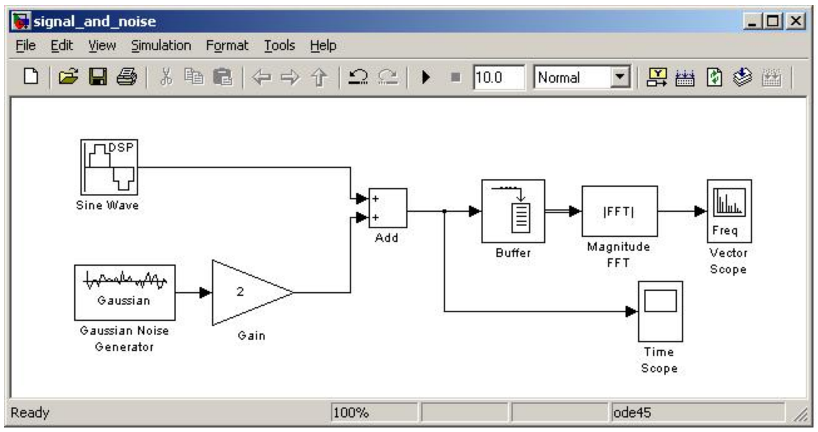


Figure 1.5: Noise added to signal and displayed in Time and Frequency

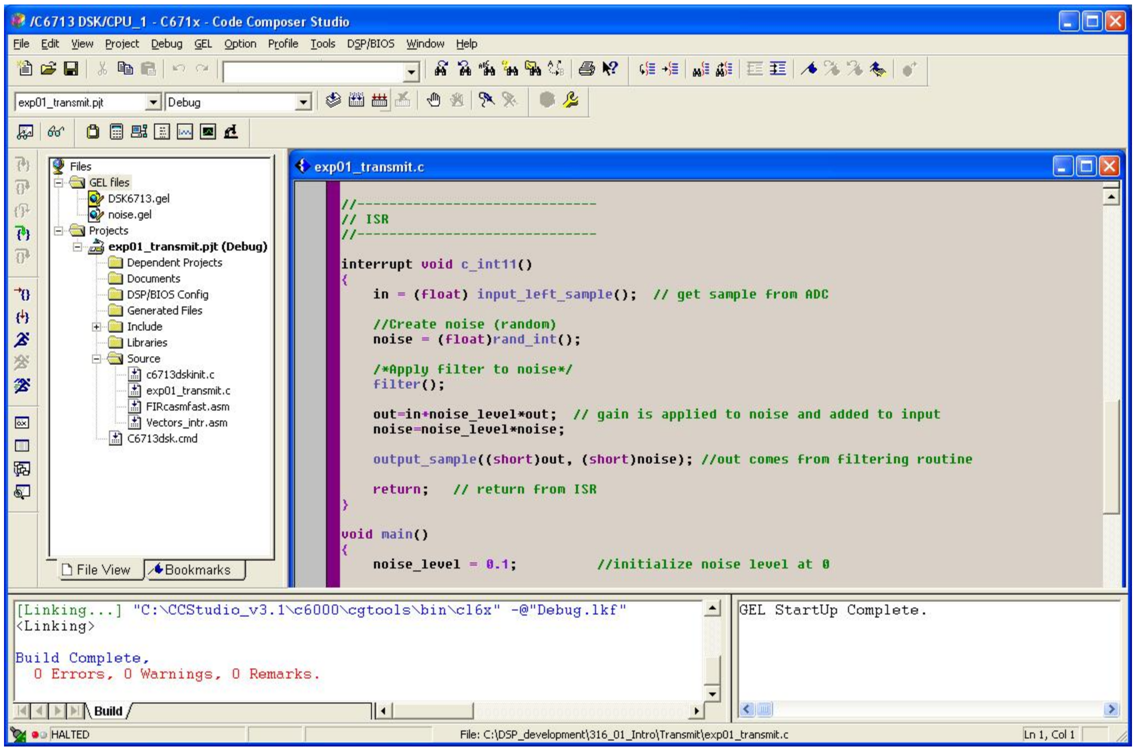
As you run the simulation, double-click on the `Gain` block and change the gain to 0.5. Click on `Apply`. You can change the gain as you wish and press `Apply` while the simulation runs. Try different gains up to a value of 5, and observe the effects on time domain and frequency domain. Make sure you obtain a meaningful display for both domains; there is no point in observing anything if your time domain plot just looks like a yellow blob.

1.5 Introduction to `Code Composer Studio` v3.1

Code Composer Studio is the programming environment supporting the TMS320 family of Texas Instruments Digital Signal Processors. The platform you will utilize in this laboratory (for any course) is the TMS320C6713 DSK. This is a development platform with an industry-grade DSP on it. If you are considering going into DSP in your professional practice, it is a good idea to become familiar with this programming environment. `Simulink` offers the capability of creating a project based on the block diagram. From within `Simulink` you can open `Code Composer Studio`, compile, load and run the automatically generated code. However, it is preferable that you become familiar with the programming environment and the type of code that will run on a floating-point DSP hardware. All code utilized in this laboratory is written in C and will be provided to you. You may be required to modify it when necessary. In this experiment, you will pair up with your neighbouring group to run two programs: one "transmitter" and one "receiver".

Upon opening `Code Composer Studio` (CCS), you should see the splash window appearing on your screen, and shortly after the main CCS window will open. First, you will need to connect the target. You do this by selecting `Debug` (on the top menu) and `connect`. Now you will check whether the target is ready, by selecting `GEL/Check DSK`. At the bottom at the CCS window you should see a message appear, saying *"Target is okay"*. The main CCS window looks like the one presented in Figure 1.6. It is very similar in look and feel to standard Windows-based programming platforms, such as Visual Basic, C++, etc.

You will also notice that one “GEL” file will be loaded automatically. This file will contain the commands that will be executed when you select any of the options under the menu `GEL`. Later on you will load another GEL file in addition to the existing one, in order to add noise in your system.



```
//------------------------------
// ISR
//------------------------------

interrupt void c_int11()
{
    in = (float) input_left_sample();  // get sample from ADC

    //Create noise (random)
    noise = (float)rand_int();

    /*Apply filter to noise*/
    filter();

    out=in+noise_level*out;  // gain is applied to noise and added to input
    noise=noise_level*noise;

    output_sample((short)out, (short)noise); //out comes from filtering routine

    return;   // return from ISR
}

void main()
{
    noise_level = 0.1;          //initialize noise level at 0
```

```
[Linking...] "C:\CCStudio_v3.1\c6000\cgtools\bin\cl6x" -@"Debug.lkf"
<Linking>

Build Complete,
  0 Errors, 0 Warnings, 0 Remarks.
```

```
GEL StartUp Complete.
```

Figure 1.6: Typical `Code Composer Studio` Window With a Project Opened

The CODEC on the c6713 DSK

On every experiment you will simulate your system first and then implement it using *real* signals coming from a signal generator. The CODEC will sample the input and transform it into numbers (i.e., quantize it) in order for the DSP to process them. The CODEC you will utilize is the TLV320AIC23B. It is programmable and offers sampling rates ranging from 8KHz to 96KHz. You will see from the code whether the program is utilizing 48KHz or 96KHz for the sampling rate. Since this is a *stereo* CODEC, each sample represents two channels in 32 bits; one channel on the most significant 16 bits and the other channel on the 16 least significant bits.

1.5.1 Running Code

Decide among the two groups who will run the transmitter and who will run the receiver. The software is found under `c:/ECE316/316_01_Intro/`. After you select the `Transmit` or `Receive` folder, you will find a project there with the same name as the folder. Open the **PROJECT** (that is a .pjt file) under `Project/Open`. You will notice that it will appear on the project tree display on the left-hand side of the CCS window.

The project has files written in C (that is .c and .h) as well as assembly (.asm). Feel free to take a look at these files. Some files are dedicated to configuring the hardware, its registers, memory and interrupts. Others pertain specifically to the program which implements that system you simulated. CCS will compile, build and create an executable (.out) which will run on the target hardware.

Now `Build` the executable code, by going under `Project/Build` or pressing F7. Your `Code Composer Studio` may already be configured to load the executable program automatically. In case it is not, you can do it manually going under `File/Load` (remember, you will load a .out file!). You will notice that a small window with a progress bar which indicates that the executable program is being loaded onto the platform. This small window will show which memory section within the platform is being filled as well. This last event will happen very fast, since the executable being loaded for this particular example is very small.

After the executable program is loaded, you must run it. You can do that by clicking on the "running man" icon on the left side of the `Code Composer Studio` window. You can also go under `Project/Run` or press F5. Now that your code is *supposedly* running, turn your attention to the instruments. If you are really not all that familiar with the instruments, it may be a good idea to read the document titled `On Instruments`, which has also been posted.

The Transmitter

For the group running the transmitter, you will also need a "GEL" file, which will allow you to dynamically change the noise level. This file, named `noise.gel` is loaded by going under `File/Load GEL`. After you load this type of file, another item appears under the menu `GEL` at the top of the window. In this case, the new item is called noise. If you go there and select it, a slider window will appear.

After you open the `transmit.pjt` and load the GEL, you must compile the project, load the executable file unto the platform and connect inputs to the target hardware. Do all that now. The transmitter will then output the sum between the input signal and noise. You will determine the amount of noise by using the slider which you have just opened.

Observe the output signal, using both time domain and frequency domain displays on the oscilloscope. By now, you should be familiar with the displays, since you have simulated both domains above. For the input, use a sinusoid with 0.5 Vpp and 1 KHz. Vary the frequency up to 10KHz and observe the output. After you take a good look at the output, you will disconnect it from the instrument and connect it to the input of the receiver (which is running on the board of the neighbouring group).

The Receiver

Load the receiver project, compile and run it. The receiver will utilize a filter to "cut out" the noise which was being added during the transmission. You can test your receiver prior to plugging in the signal coming from the transmitter, by using as input a 1 Vpp, 500Hz sine wave and varying its frequency up to 10KHz. By measuring the output at different frequency points between 500Hz and 10KHz, you will be able to plot the frequency response of the filter. Try it.

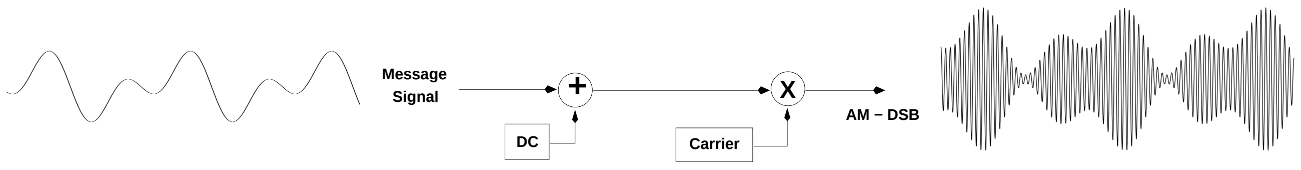
After you connect the transmitter to the receiver, you should be able to see that the transmitter output signal is noisy and the receiver output signal is "clean", i.e., without noise. You may observe that relative to the original signal (that which was input to the transmitter), the signal output by the receiver is attenuated. What would you suggest as a solution to this?

1.6 Going The Extra Mile

At this point, try to explore the files within the project you are running. You may have noticed that the core function is called from within an interrupt service routine. Why is that? How exactly is this code running? What is interrupting the process? How are the samples handled? How are the two channels separated? A good understanding of these simple steps will give you a better idea of what is being done in every experiment, as well as a good background to delve into real-time DSP programming.

Chapter 2

Amplitude Modulation



2.1 Introduction

Amplitude modulation (AM), still widely used in commercial radio today, is one of the simplest ways that a message signal can modulate a sinusoidal carrier wave. The purpose of this lab is for you to gain familiarity with the concepts of amplitude modulation and demodulation. This will be done in three main steps:

- First, an amplitude modulation system will be created and simulated using `Simulink`.
- Second, you will implement an AM modulator and an AM demodulator on a TMS320C6713 DSP platform, using a sinudoid as your input.
- Third, you will use voice as input to your modulator and another group will demodulate it also using the DSP platform.

If you so desire, you may also implement an AM demodulator based on an envelope detector. This can be done with discrete components on a prototype board in order to illustrate the most basic method for demodulating AM signals. No marks are associated with the "analog" implementation.

At each of these steps, you may be required to modify various system parameters and observe the resulting changes.

2.2 Background Reading and Preparation

Amplitude modulation is one type of continuous-wave modulation, covered in [5]. *Before* coming to the lab, you must complete the preparation sheet. Use any resource available to you to solve the preparation sheet.

The preparation work should lead you to identify the equation for an AM-modulated signal, to understand the concept of overmodulation, to understand the time domain and frequency domain representation of an AM-modulated signal with (AM-DSB) and without (AM-DSB-SC) and to have a clear high-level picture of an AM modulator and demodulator.

2.3 Experiment

The experiment is divided in three main parts, and possibly one extra optional part, as described above. For the implementation of a complete system, two groups must be paired up. You are expected to know where the necessary blocks are within `Simulink`. **All results that will be marked are to be reported in the proper spaces provided in this outline.**

2.3.1 Designing and Simulating an AM Modulator

Based on the block diagram that you have reviewed for an AM modulator, build your model using the blocks found in `Simulink`. Your AM Modulator should look like Figure 2.1 below.

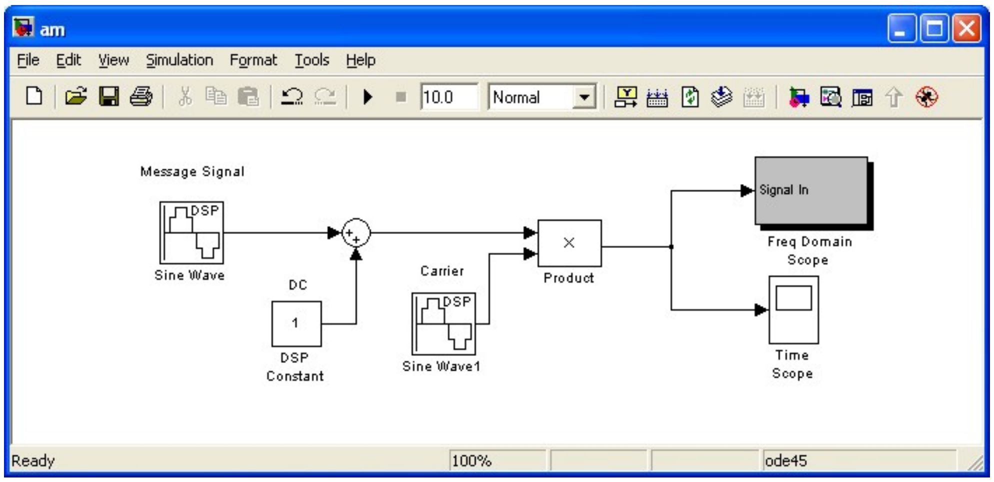


Figure 2.1: Amplitude Modulator

For the initial simulation, use a `DSP Sine` generator as input, and for output use one time scope and put together your own `Frequency Domain Scope`, as you did in the previous experiment. For your **message** signal, use a 1.5KHz, 0.5 peak Voltage (that is $1V_{pp}$) sinusoid. Use 48KHz as your sampling rate.

As the block diagram on your preparation should indicate, the input signal must be added to a DC component and the signal resulting from this addition will be multiplied by the carrier frequency. Use a DC component of 1. The carrier frequency is generated by another discrete-time sine wave generator, with an amplitude of 1 (peak value) and 12KHz. You may choose to add gain blocks at different locations in the signal paths. The gains are not a necessary part of the experiment, but they will allow you to control the values at different parts of the signal path while the simulation runs. Also, make sure you define the sampling rate for all the blocks that need it. You are simulating what will happen in the target hardware.

Run your simulation, and observe the result on the `Simulink` scopes. Adjust the setting of your scopes appropriately (i.e, should you really start displaying your signal at -150dB?). At this point, the scopes should allow you to observe an AM signal similar to the one you drafted on your lab preparation sheet.

- *The picture below shows the time-domain amplitude modulated signal that you should observe. Explain the resulting voltage values you obtained on your simulation.* **(0.5)**

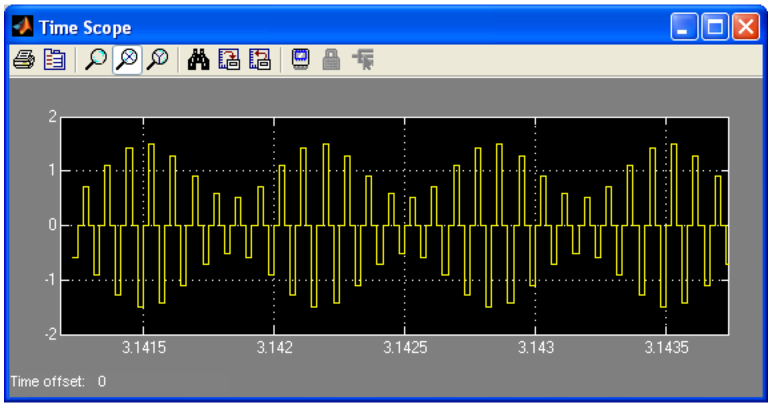


Figure 2.2: Amplitude Modulation, Time Domain

In all experimental reports from this point on, you may be asked to draw the waveforms you obtain through simulation or while running the actual system. This seemingly tedious exercise has a purpose: to create the habit on the Engineer (that is you) to report accurately the forms and numbers obtained, so that they can be reproduced at a later date if necessary. Your results *must be reported clearly*. For instance, if you are reporting a waveform in the time domain, you will need:

1. Two axes with labels indicating what they represent;
2. The unit in which the vertical axis is measured;
3. The unit in which the horizontal axis is measured;
4. The values found for peaks and valleys;
5. The location of the zero crossings, if they exist;
6. The measured frequency of your signal.

Now you can move on to the next questions, which will require that you record your values.

- *Figure 2.3 shows the frequency domain plot that your simulation should produce. Indicate what are the expected frequency values to each component, based on the values you have used in your simulation.* **(0.5)**

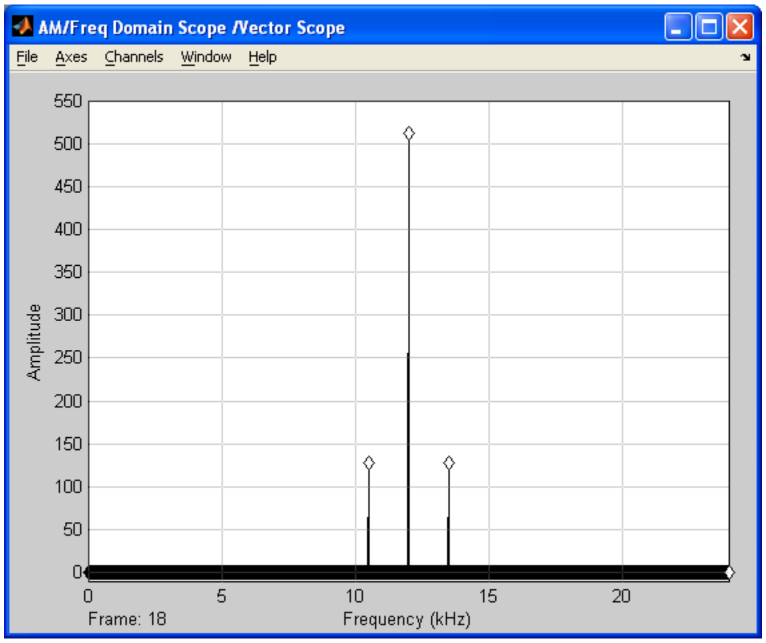


Figure 2.3: Amplitude Modulation, Frequency Domain

- *Modify your system to obtain a modulation index of 100%. Draw the time-domain plot. Be careful how you report your waveform.* **(0.25)**

- *Now modify your system to obtain a modulating index greater than 1 (i.e., greater than 100%), and sketch the resulting signal. Assuming you only have an envelope detector available for receiving your signal, what is the consequence of overmodulation?* **(0.5)**

- *Design an AM-DSB-SC system (sinusoidal input). Draw below the block diagram, the time-domain and frequency domain results. Would this method be preferable over AM-DSB? Why?* **(1.0)**

- *Suppose, hypothetically, that you have a signal made of three sinusoidal components - 1KHz, 3KHz and 5KHz - to transmit using AM. Which of the methods would you prefer to use: AM-DSB, AM-DSB-SC or AM-SSB? Why? Remember: bandwidth is at a premium, always.* **(0.25)**

2.3.2 Designing and Simulating an AM Demodulator

Your modulator runs fine, but your intention is to transmit the desired signal and receive it after it goes through modulator, amplifiers, antenna, channel and receiving antenna. You must demodulate the received signal to extract the **modulating** signal, which is the signal containing the information (in your case, this information is a 1KHz, $1V_{pp}$ sine wave, or tone).

In this part of the experiment, you will utilize your previously tested AM modulator and implement a demodulator by adding a filter to the signal path, after the multiplication with the carrier. As you have seen in the theory, by multiplying the incoming (received) signal by the carrier frequency, one will have as a result the desired baseband signal as well as DC and other higher frequency components. If you remove the higher frequency components with a low-pass filter, you retrieve the original message signal which was transmitted, plus a DC component. Since you do not have a capacitor to put in series and remove the DC, your option in software (firmware) is to create a bandpass filter. If you feel like simulating it, your system should look like Figure 2.4 below. The labels in the figure were changed to clarify all parts of the system for you.

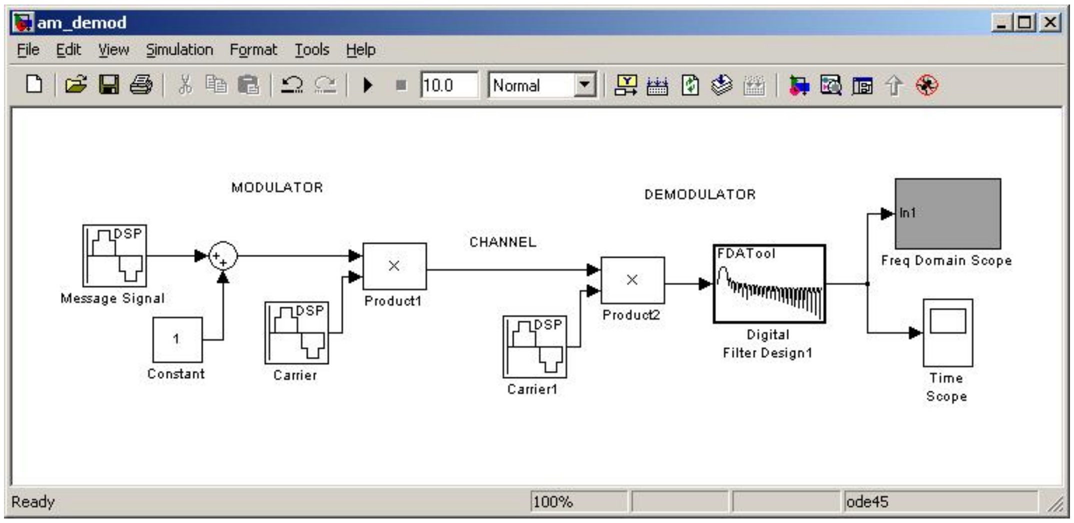


Figure 2.4: Simulating Modulation and Demodulation

- *Explain the demodulation process with a simple frequency domain plot. Report your numbers properly.* **(0.5)**

- *Have the TA initial here if your demodulator works.* **(0.5)**

2.3.3 Implementing an AM Modulator and Demodulator

Modulator

You will now program the DSP platform to implement an AM modulator. The code to achieve that is given to you, and can be found at `c:/ECE316/Exp02/AM_mod`. Open `Code Composer Studio`(CCS). Go under `Project/Open` and select `AM_mod.pjt`. All files related to this project should appear on a project tree at the left side of CCS. Inspect the files, identifying the most relevant routines, such as carrier generation, sample handling, adding DC, etc. Make sure the target is "connected" (go under Debug/Connect). Compile the project, load the executable file andn run it.

Answer the questions below, pertaining to the system you are running. Use a 3KHz, $1V_{pp}$ sine wave as input. Make sure your signal generators are adjusted to see a high-impedance load at the output. Also, take a good look at the code and make sure you understand what is going on. You will be required to change it.

One interesting point: when you run your system on the target hardware, you will notice that the **modulated** signal will **not** have the expected amplitude. This is due to the fact that there is an analog filter at the output of the CODEC, which starts to roll off at approximately 10KHz. Since your modulated signal presents a carrier at 24KHz, you can expect your output to be off by a certain margin (question for thinking only: how would you determine this margin?).

1. Time Domain Results

- *Sketch the time domain results. Yes, these should be similar to the first ones you got in your simulation. As you sketch your output, report the values that you believe are relevant for someone who reads it to understand what you have done. Why is the amplitude different than what you expected? (Read the paragraph above and remember it for future experiments).* **(0.5)**

- *Use the XY display on your scope (that is known as the Lissajous Figure) to determine at what point you have overmodulation. Explore the oscilloscope to find out where the XY display is. Show the TA and have the TA sign off the box.* **(0.5)**

- *Change your DC value to zero on the code (remember the block diagram? Find the line that adds the DC). If you change your DC value to zero, what is the advantage, if any? Does it make any difference for demodulating the signal?* **(0.5)**

2. Frequency Domain Results

- *Use the FFT capability of your scope to obtain the Frequency Domain display. Sketch the Frequency Domain results both with and without DC. Identify the missing component.***(0.5)**

Make sure your code is doing AM-DSB-SC, and use now a 1KHz, $1V_{pp}$ square wave as input. Sketch the results. Now when you look at your results, think of it this way: you have **mixed** a carrier with a stream of bits. Your 1KHz square wave mimmics alternating bits at 2Kbps. Note: in communications, "mixing" is the multiplication between two signals; in audio it is their addition!

- *How wide is the bandwidth needed to transmit this signal? If you decrease your bit rate, what happens to the bandwidth? What if you increase the rate?* **(0.5)**

Demodulator

You may leave the modulator project open when you open the demodulator project. Make sure that the demodulator will be your *active* project. The project to be compiled can be found at `c:/ECE316/Exp02/AM_demod`. Open `Code Composer Studio`, go under `Project/Open` and select `AM_demod`. Build the project and run it.

Remember that now you are implementing a **demodulator**, which means that you will need an amplitude modulated signal to serve as input to your system. You will test your demodulator first by using an AM signal coming from the signal generator. Assuming you have now a sinusoid on your signal generator, you will proceed as follows:

- Set the existing sinusoid to 24KHz, $3V_{pp}$. This will be your sinusoidal carrier.
- Press *Mod* (Agilent) or *Modulation* (Tektronix) on your signal generator.
- Set the message, or modulat *ing* signal to be a 3KHz sinusoid. You want sine modulating sine.
- Set the modulation index (or *depth*) to be less than 100% (say, 80%).

Test to see which of the two input channels is the input to the demodulator. After you have an output, answer the questions.

- *Have the TA sign if your demodulator works.* **(0.5)**
- *Change your message signal to a triangle or a square wave. The received signal is different from the "perfect" triangle or square that was your message. What does this (however small) change in their shape indicate?* **(0.5)**

2.3.4 The Full System: Sine and Voice

Now that you have tested both modulator and demodulator, pair up with the group next to you and have one group run the modulator and the other run the demodulator. Initially, your input to the modulator will be a 3KHz, $2V_{pp}$ sine. Set the system up, connect everything appropriately and run it, making sure it works for a sinusoid.

Now for the group running the **modulator**, go under `c:/speech_samples` and double-click on one of the voice samples. This will bring up Windows Media Player. Connect one of the coax cables to the output (i.e., headphone jack) of the PC workstation and to the oscilloscope. Adjust the volume level to give you a voice signal of around (eyeball it) $2V_{pp}$. Connect the output of the PC to the input to your modulator. Now on the demodulator you should see the voice being demodulated after the AM transmission. If you are curious as to how it may sound at the receiving end, ask for some loudspeakers.

- *Have the TA sign if your whole system works for both sinusoid and voice.* **(0.5)**

2.3.5 If Time Allows: Demodulating the AM Signal With an Envelope Detector

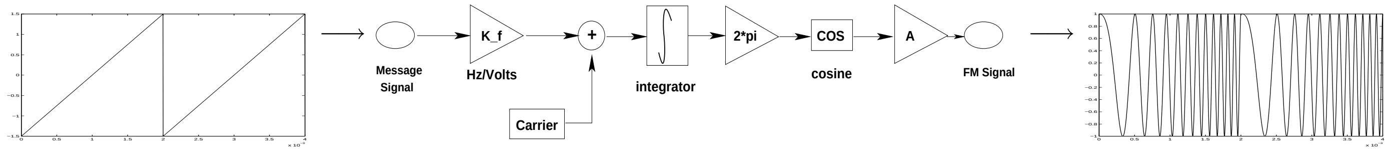
You can build and test a simple AM demodulator based on the envelope detector you have designed in your preparation sheet. If you calculated the component values for an existing AM station in your area, then you can try to pick that up. Otherwise, if you want to detect the AM coming from the DSP platform, you will need to recalculate the numbers. If you feel like it, give it a try. Your task is to "transmit" the output signal of the modulator running on the DSP target onto the input of the envelope detector you have assembled on a prototype board, and display the output of your envelope detector to the oscilloscope. If you would like to do this, ask the TAs for the material and give it a try.

2.4 Accomplishments

In this experiment, you were presented with the key issues involved in designing, simulating and implementing an AM modulator and demodulator. This was done by using a simulation model, a DSP platform in which modulator and demodulator were implemented. You tried a software-defined and possibly an envelope detector as well for the demodulation of the AM signal. The experiment intended to guide you through the steps necessary to achieve a practical understanding of the concepts studied in the theory of Amplitude Modulation.

Chapter 3

Frequency Modulation



3.1 Introduction

In Experiment # 2, you designed, simulated and built an AM modulator, and hopefully had the opportunity also to see how to demodulate an AM signal. In this experiment, another widely used modulation technique will be introduced: Frequency Modulation (FM). As the amplitude of the sinusoidal carrier wave was modulated in AM, this time the instantaneous **frequency** of a sinusoidal carrier wave will be modified proportionally to the variation of amplitude of the message signal. You can also see this process as a voltage-controlled oscillation: as the voltage of the input varies, so does the sinusoidal frequency of the output. FM is widely used in commercial radio, spanning be RF band between 88MHz and 108MHz. In commercial radio, a regulatory body (the CRTC in Canada, the FCC in the United States) allocates 200KHz of bandwidth for each station, or channel.

A three-step process will allow you to become more familiar with FM:

- First, a frequency modulation system will be created and simulated using `Simulink`.
- Second, an FM demodulator based on a Phase Locked Loop (PLL) will be designed and simulated, using the previous modulator as the “signal source”.
- Third, an FM modulator and PLL-based demodulator will be implemented on a TMS320C6713 DSP platform. The input (message) signal to the modulator will come from the signal generator.

You will be required to change various system parameters and observe the consequences of the changes.

3.2 Background Reading and Preparation

Frequency Modulation (FM), as well as Phase Modulation (PM), are types of Angle (Exponential) Modulation. These are covered in more detail in [5], pp. 166-169.Take a look at the theory behind Phase-Locked Loops (PLL), as in pp. 178-182 of [5].

Also *before* coming to the lab, you should complete the **lab preparation** and hand it to the T.A..

3.3 Experiment

As it was mentioned above, this experiment is divided in three parts: the design and simulation of an FM modulator in `Simulink`; the design and simulation of a PLL-based FM demodulator; and the implementation of both modulator and demodulator on the DSP platform. For this third step, the incoming signal will be produced by the signal generator.

All results are to be reported in the spaces provided in this outline.

3.3.1 Designing and Simulating an FM Modulator

Based on the FM signal equation and the block diagram for an FM modulator that you have submitted with your **lab preparation** (AHA!), build your model in `Simulink`. Follow similar procedures for input sources and output sinks to what you have used in previous experiments. In particular, do not forget the following details:

- The input to an FFT-based scope must be buffered, with a buffer size that is an integer multiple of the FFT size (could be the same size). It makes no sense to attempt to perform an FFT on a single sample, or buffer a smaller number of samples than the length of the FFT;
- A greater FFT length means a better resolution for the frequency domain representation of the signal (take 1024 points as a reasonable length). Frequencies which fall at an integer multiple of f_s/N will be resolved exactly. At this point you should be able to tell when the results are producing relevant information or insufficient information;
- All blocks that require a sampling frequency (or sampling period) to be defined **must have** the same sampling frequency. Since the daughtercard utilized in the lab makes use of a 48KHz sampling frequency, you should preferably use this number throughout your simulation exercise;
- The frequency of a commercial FM carrier wave in practice is much higher than the one utilized in this experiment. Your favourite FM radio station operates between 88MHz and 108MHz. One must keep in mind, however, that in the lab a limitation is imposed by the sampling frequency utilized by the CODEC daughtercard. The carrier frequencies for this experiment will be compatible with the hardware capabilities.

Try to run the `Simulink` block diagram you had in your lab preparation. Do not put your hand up just yet. Now let us assume, hypothetically, that the straight-forward approach to "build the model from the equation on the book" did not work. If it did, make sure you show the TA and get some bonus marks. Now, if it did not, you should re-work the equation. Maybe bringing some elements "into the integral" will simplify your block diagram. Do not despise the factor 2π on the constant k_f if you are working with ω (in *rad/s*). Textbooks may create some confusion for not explicitly showing the 2π factor.

- *Rewrite your FM equation (write it in the box below).* **(0.5)**

- *Draw the new block diagram for the FM modulator.* **(0.5)**

Use as input signal a sine wave with amplitude 1.0 (this would mean a 2Vpp simulated signal) and 1200Hz, a carrier frequency of 16000Hz and a sensitivity factor of 3600Hz/V (that's the k_f constant on [6]). You should observe a time domain output signal similar to the one you drafted on your lab preparation sheet. Include in your model a `Freq Domain Scope`. In this lab you will concentrate on understanding what goes on in the frequency domain. Make sure your frequency domain display gives you a good picture.

So your model should look like Figure 3.1 below. This is a *suggested* model, so your numbers and blocks will be different. For instance, you should try to use a `Discrete-Time Integrator` rather than the digital filter shown below (search for that block within `Simulink`.

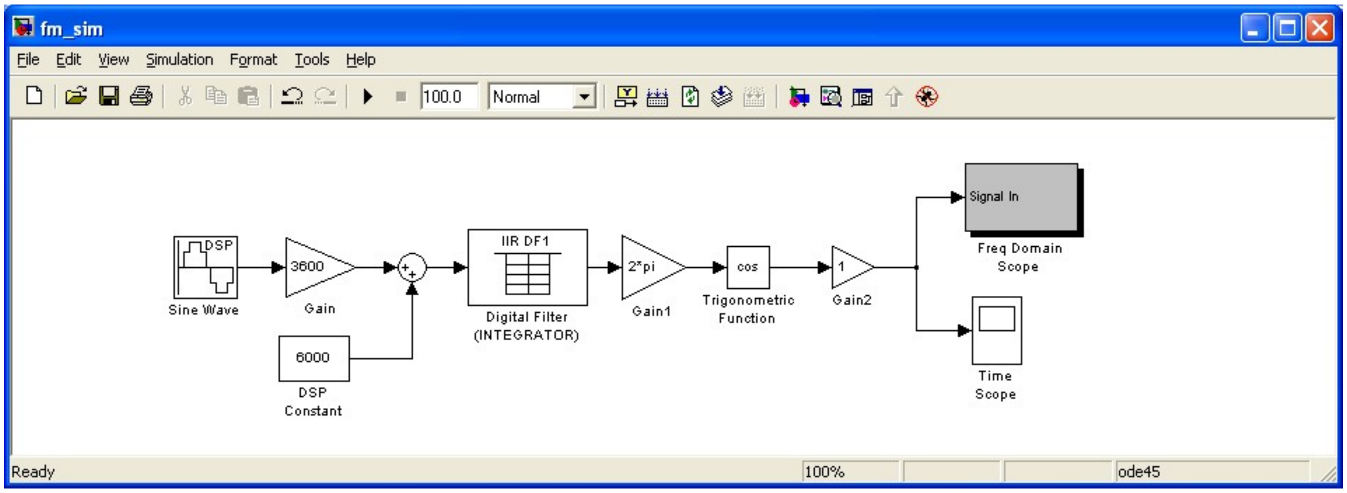
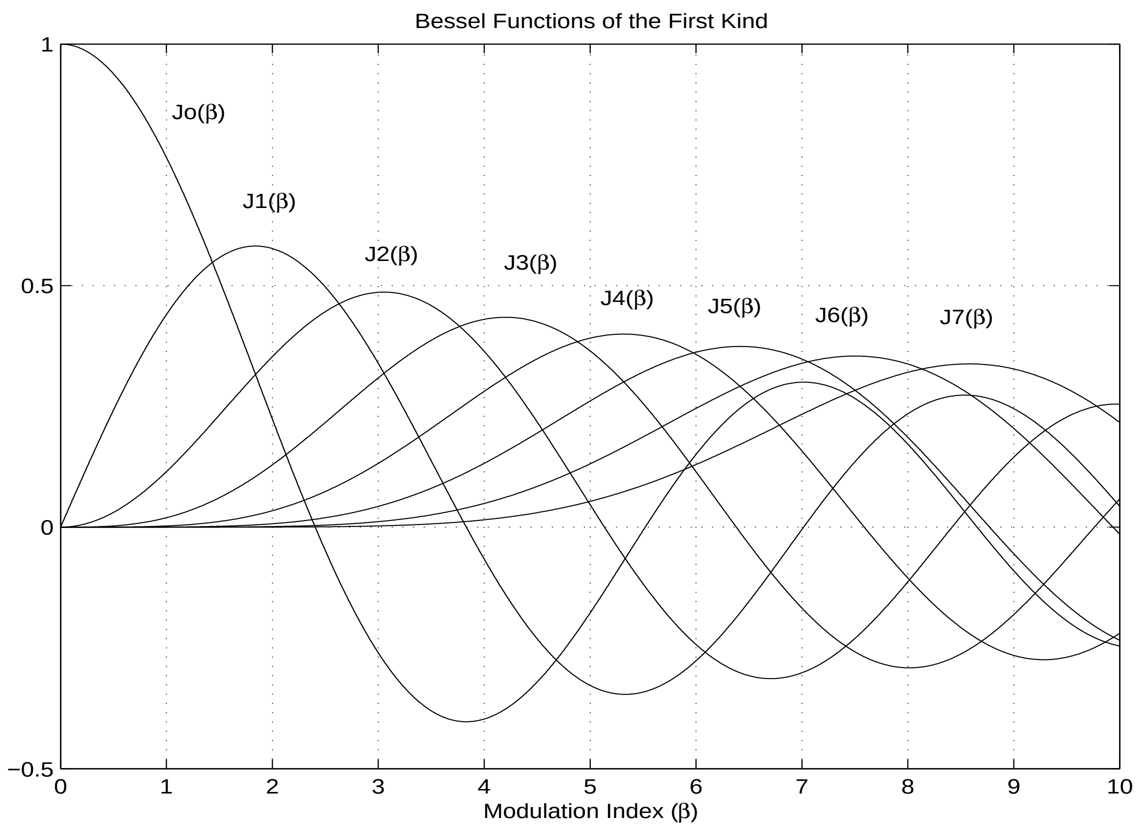


Figure 3.1: Suggested FM Modulator for Simulation

For the next questions, use the graph presented below, when necessary.



- *The output signal observed in the time and frequency domains should look similar to the pictures below. Present below the* **calculation** *for the deviation ratio β, the values for the carrier and frequency components on each side of the carrier, and the bandwidth according to Carson's Rule. Identify the carrier component on the frequency domain picture below* **(0.5)**.

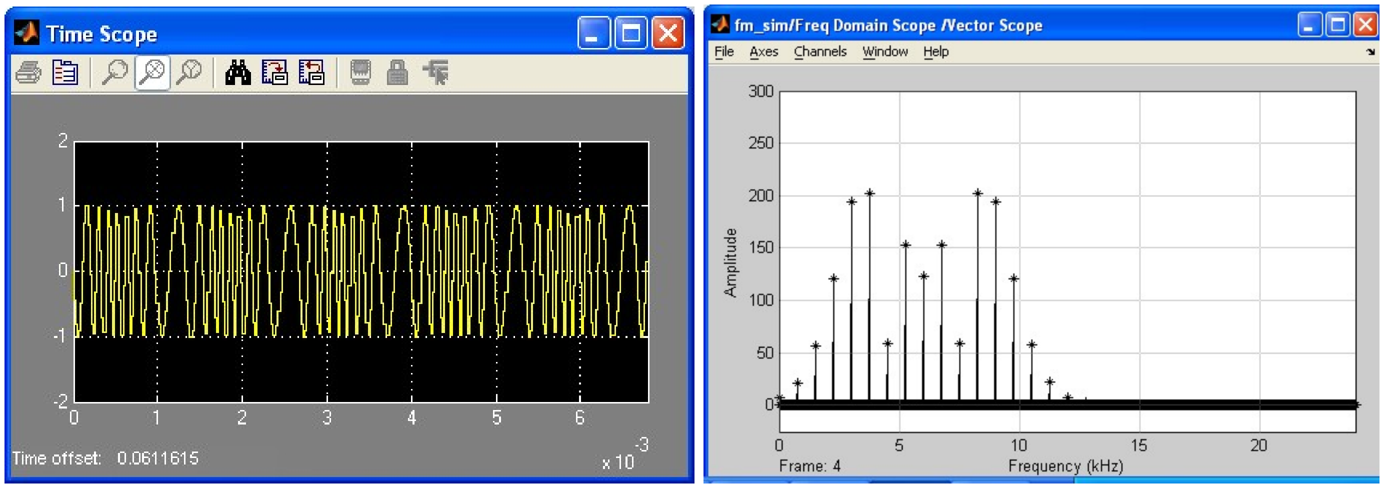


Figure 3.2: FM in Time Domain and in Frequency Domain

Deviation ratio calculation. **(1.0)**

Report the frequency values for the carrier and two frequency components on each side of the carrier, as well as the $J_n(\beta)$ associated with those frequency components. **(1.0)**

Bandwidth following Carson's Rule. **(1.0)**

- *Change the amplitude of the message signal to 0.8 (that is 1.6Vpp). Draw what you observe in the frequency domain. For crying outloud, what happened to your carrier? Explain.* **(1.0)**

- *Your input signal is a 1 V_p sinusoid. Is there any other input signal frequency for which there will* **not** *be a carrier component in the frequency domain? Present some values for those frequencies (use the Bessel Function graph given and test them using the simulation).* **(1.0)**

- *If you set your β to be $<< 1$, what will be the bandwidth? How about for $\beta >> 1$?* **(0.5)**

3.3.2 Designing and Simulating an FM Demodulator

Following the block diagram of the PLL-based FM demodulator you have drawn on your preparation sheet, build a new model in `Simulink`. The input signal for the demodulator simulation will be the FM-modulated signal from your modulator. Your demodulator should be made of a PLL and, if necessary, an output low-pass filter. The PLL, as you have drawn on your lab preparation, is made of a voltage-controlled oscillator, a multiplier and a low-pass filter. It is not always necessary to have a second filter after the PLL. You may be able to avoid the use of a second filter by using a loop filter with a sharper roll-off at the cutoff frequency of interest.

You will need to make a change to the VCO on the feedback path, by setting the trigonometric function block to `sine`. For those who were attentive to this detail while reviewing the theory, the PLL design calls for a 90 degree shift between the incoming FM-modulated signal and the signal output from the VCO on the feedback loop. Check out [5], p. 178 and 179.

To make your life easier, you can create a "subsystem" with your modulator (VCO), and utilize this new block to implement your demodulator. To do this, select the blocks on yor model that implement the VCO, and click on `edit/create subsystem`. The constants and gains that you have to adjust for the VCO are the same as the ones used on the modulator (the VCO **is** a frequency modulator). The forward path filter will be designed using the FDA-tool block. Make the filter a low-pass FIR design using `windowing` (this is set by the drop-down menu found beside `FIR` on the low left corner of the FDA-tool). Set the order to 30, select a Kaiser window and check the `scale passband` option. The cutoff frequency is the frequency of your **modulating** signal.

Run your modulator-demodulator system. If your output is still not looking right, try increasing the amplitude of your **message** signal to 2Vp. Your system should resemble the figure below.

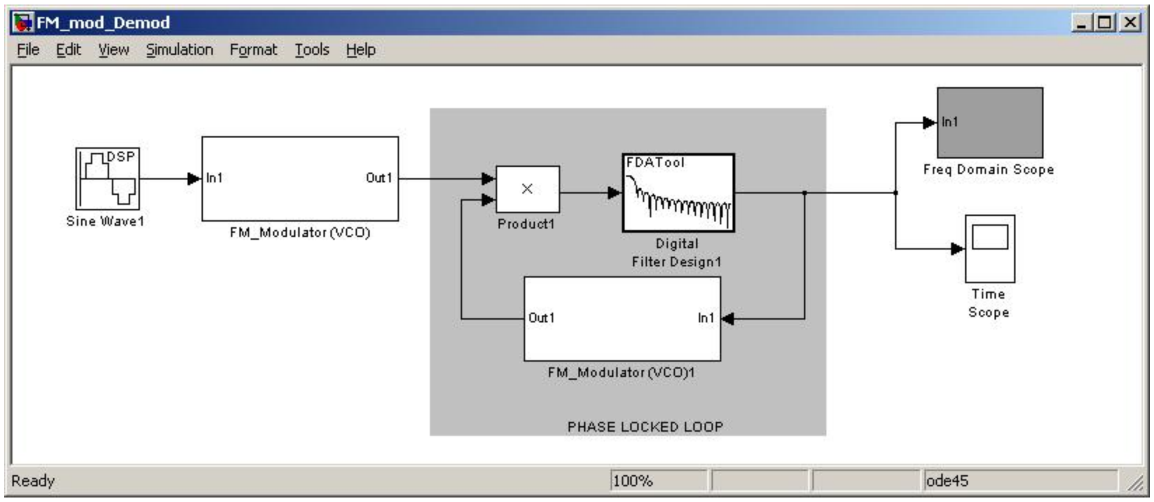


Figure 3.3: Suggested FM DEmodulator for Simulation

The loop filter will separate the "detected" signal from all other frequency components, wich arise from the multiplication of the incoming (FM) signal with the one output by the VCO. The parameters for the filter you will design using the FDA Tool are: Lowpass; FIR – select Window; Order 70; Kaiser window (leave the other parameter at 0.5); fs = 96000 Hz and fc=1200Hz. Those last two are the sampling frequency and the cutoff frequency.

You can observe this by attaching `Freq.Domain Scopes` to your demodulator at appropriate probing points. Also, attach a time-based scope to the output. Now run your system and observe the output scopes.

Make it work first, and show the T.A.. Make sure the sinusoid you see on the output of the demodulator has the same frequency as the one input to the modulator. Vary the frequency of the message signal up and down by about 1KHz. Does it still work? Explain why to the TA and have the TA sign the box. **(0.5)**

3.3.3 Running and Testing Your Modulator and Demodulator

Two working projects are given to you under `C:/ECE316/Exp03`. Open the **modulator** first, which is found on `FM_mod/FM_mod.pjt`. Remember, go under `Project - Open`, not under `File - Open`.

FM Modulator

After opening your FM modulator project, compile, download and run it, and make sure it works. The input to your modulator will be a 100 Hz, 1Vpp sinusoid from the signal generator. Make sure you adjust your signal generator to use the `High Z Load` setting, and that the output button is on. One channel will display the input (or message) signal and the other the FM signal modulated with the input.

Just like the system you have simulated, your FM signal will vary 3600 Hertz per Volt of the input signal, having 12000Hz (carrier) as a centre frequency. Remember, FM changes the **instantaneous frequency** of the carrier, not its amplitude.

- *On one channel you have the message signal and on the other channel you have the FM signal. Set your oscilloscope to perform an FFT on the FM-modulated signal. Input a 2V_{pp}, 1.2KHz sine wave. Use the cursors on the scope to identify the components. Draw below the spectrum of your FM signal. Label the components and verify if they match those you found during the simulation.* **(1.0)**

FM Demodulator

Load the demodulator project from `C:/ECE316/Exp03/FM_Demod/Fm_demod.pjt`. This project implements an FM demodulator. Look at the code and see if you can understand what is taking place. As input to the demodulator, set your signal generator to generate FM, 4000Hz deviation, 12KHz carrier and 1KHz sinusoidal message signal. Please, check on the oscilloscope if your input is correct prior to calling a TA. Compile, download and run the project. Can you demodulate the signal? If you can, have the TA sign the box. **(0.5)**

3.4 The Extra Mile

If you are brave enough, try out viewing your favourite radio station on the Spectrum Analyzer. You will need to know the frequency of its carrier (in MHz) and a make-shift antenna. Set the spectrum analyzer to start at frequency 88.1MHz and end at 108MHz. By doing this you will be able to see all FM radio stations operating in Toronto. Choose one, and center the reading on the spectrum analyzer on it. Since by regulation every FM radio station has about 200MHz for its operation, you can change the span on the spectrum analyzer to be 200MHz. If you'd like to listen to what is being displayed on your spectrum analyzer, turn on an FM radio and listen to it.

3.5 Accomplishments

In this experiment, you were required to design and simulate an FM modulator and demodulator, and to implement these on a DSP platform using a given program. As a result, it is hoped that you became familiar with the theory behind Angle Modulation, and with some practical details pertaining to the modulation and demodulation as they were implemented. All of these concepts are widely used in the study of communication systems.

Chapter 4 - Noise in AM and FM

4.1 Introduction

In this experiment you will verify the relation between input signal-to-noise ratio (SNR) and output SNR. This will be done for three cases, namely:

- Amplitude Modulation with Envelope Detector receiver;
- Double-Sideband, Suppressed Carrier with Coherent Detector receiver;
- Frequency Modulation with Phase-Locked Loop receiver.

4.2 Experiment

4.2.1 Common Procedure To All Cases

In all cases, you will have to load the specific project, compile and load it onto the DSP platform. At the top of the project tree (left-hand panel in `Code Composer Studio`), you will find a `Gel Files` directory. You will need to right-click on that to load a GEL file and right-click on `Projects` to load a project. Each of the three cases below has its own project and GEL file. You will need to load both to make your measurements.

Your measurements will be done on four variables: the noiseless modulated signal input to the demodulator (Si), the channel noise (Ni), the demodulated signal without noise (So) and the noise level measured at the output (No). You will monitor these variables using `Code Composer Studio` and will record their values for different noise scenarios. Your objective is to populate the SNR input/output table given and use the values you calculate to draw a graph. The procedure described below is to be followed in all three configurations after you have loaded the right project and the corresponding GEL file.

The procedure to add the variable to the `Watch Window` is to put the program to run, highlight the desired variable, right-click and select `Add to Watch Window`. The variable will appear in the new window, with the value it had when the window was open. In order to see the value changing, you must refresh the window. This is done by right-clicking and selecting `refresh`. Your other option is to click on the other tab found on the window that just opened and click back on the tab pertaining to the `Watch Window`. This will cause it to automatically refresh. Throughout the experiment you will be able to go over these motions almost automatically.

1. Go under the top menu item called GEL (not the one on the project tree), select SNR Control and open all sliders under that menu item: Noise, Si, Ni, So and No. You should now have five sliders appearing within `Code Composer Studio`. The Noise slider has many levels and all other sliders are “on-off” type switches.

2. Place the noise at the lowest level (0) and all sliders in the *off* position (that is position 0).

3. Turn the input signal on (from the signal generator or from a neighbouring board) and **make sure you see a meaningful demodulated signal on the oscilloscope**. Leave the program running throughout the measurement session.

4. Switch the control slider for variable Si on and refresh the `Watch Window`. The input signal, measured after a band-pass filter on the receiver, is your *modulated* signal. The number you will read is your input signal power. Since you are interested in the *ratio* between signal and noise, the unit here may be left out.

5. Having read the input signal power, you will now measure the different levels of noise power at the input. Start with the noise slider at the lowest level. Switch off the control slider for variable Si and switch on the one for variable Ni. Make sure that the slider is set to 1. If it is not, switch it off and on again to ensure a value of 1. Variable Ni is the noise level coming into the demodulator from the channel. It is the *channel noise.*

6. With the slider controlling Ni set to 1, refresh the `Watch Window`. Read and record the value of Ni for the lowest position of the Noise slider. Ensure that Si, So and No are zero during your measurement of Ni. This will prevent possible errors and single out the variable to be read from the `Watch Window`.

7. Bring the slider up a notch. Switch the Ni control off and on again. Refresh the `Watch Window`. Record the value of Ni. Repeat this procedure for eight levels of noise.

8. Switch the noise off (lowest level = 0), switch the control for Ni off and the control for So on. Ensure that the values for off (0) and on (1) for those switches are correct. Sometimes the sliders will not switch the values even though the slider has been moved. Ensure that you have the demodulated signal on the scope without noise. Refresh the `Watch Window`. Read the value for So and record it.

9. With the noise at the lowest level (off, or zero), switch the control for So off (0) and the control for No on (1). Refresh the `Watch Window`, read the value for No and record it. Bring the noise level up a notch, switch the control slider for No off and on again and refresh the `Watch Window`. Read the value and record it. Repeat this step for eight noise values.

10. Calculate the SNR tables in dB. Plot the curve.

11. When you are done with the measurements for one type of system, close the `Watch Window`. You will have to select the variables to be watched again when a new project is loaded.

4.2.2 First Things First

This experiment will deal with Signal-to-Noise Ratio (SNR). The plots you will generate based on your calculations will be in decibels (dB). Let us, then, make sure that those two concepts are clear.

- *In words, explain Signal-to-Noise Ratio.* **(0.25)**

- *Is "decibel" a unit? If your calculation of SNR results in 0dB, what does that mean?* **(0.25)**

This is just a warning. For the second scenario (DSB-SC with Coherent Detection) you will need the help from a neighbouring group, since your signal generator cannot generate DSB-SC. You have already done that in the second experiment, and that makes you an expert in DSB-SC. As a reward to your colleagues' services, you will graciously allow them to peruse the results that you read and record on your data table. The drawback is that if the reader/recorder screws up, four people take the hit. Feel the pressure? Remember: the SNR results you record and report will be used in an important meeting tomorrow with an investor/customer which will potentially put Can$ 1B into the company's – i.e., your employer's – bank account. Fool yourself not; the marketing guy will take the credit for it if it works, but you may get the boot if your numbers are fluff.

4.2.3 AM with Envelope Detector

For this part of the experiment you will use the signal generator to generate AM, and use the DSP board to demodulate that signal. The AM parameters are a carrier frequency of 12KHz, 1Vpp, a modulation index of 80% and a 1KHz sinusoid as message signal. Load the project from `c:/ECE316/Exp04/AM_demod_envdet_noise/AM_demod_envdet_noise.pjt`. Compile and load it onto the platform. Load the GEL named `AM_snr_control.gel`.

Be attentive to the order of events while you go through the procedure. You will create tables based on your results and then draw plots based on those tables. Any mistake in the procedure will render your results useless. The power measurement is done in software, by means of an integration of the signal over one second. The result of that measurement is stored in a variable which must be monitored using the `Watch Window` found in `Code Composer Studio`. The variables to be watched are listed, for your convenience, right at the beginning of the interrupt service routine in code. Locate them now.

Run the program, and add the four variables to the `Watch Window`. Go under the GEL top menu item, and open all the necessary sliders. Follow the *common procedure* given in the previous section to answer the items below.

- *Measure the input signal power (Si)* **(0.25)**

Input Signal Power (measured after BPF)	

- *Measure the output signal power (So)* **(0.25)**

Output Signal Power (after demodulation)	

- *Populate the noise values below and calculate the SNR for each case.* **(1.0)**

Noise Level	Input Noise Power	10*log10(Si/Ni)	Output Noise Power	10*log10(So/No)
1				
2				
3				
4				
5				
6				
7				
8				

- *Draw the plot using SNR input on the X axis and SNR output on the Y axis.* **(1.0)**

4.2.4 DSB-SC with Coherent Detection

For this part of the experiment you will make use of the kind help from the neighbouring group. They will provide you with the DSB-SC signal, since your signal generator cannot generate it. Your DSP platform will then demodulate that signal using Coherent Detection.

The transmitter group will run the project `c:/ECE316/Exp04/AM_mod_DSB_SC/AM_mod_dsb_sc.pjt`.

The receiver is to be run on a separate DSP platform, by loading the project found at `c:/ECE316/Exp04/AM_demod_coher_noise/AM_coher_demod_noise.pjt`. Compile and load it onto the platform. Run it. Load also the GEL named `DSB_SC_snr_control.gel`. **Make sure that when you run the receiver, one channel displays DSB-SC and the other displays the demodulated message signal (otherwise, your inputs are reversed)**.

- *Measure the input signal power (Si)* **(0.25)**

Input Signal Power (measured after BPF)	

- *Measure the output signal power (So)* **(0.25)**

Output Signal Power (after demodulation)	

- *Populate the noise values below and calculate the SNR for each case.* **(1.0)**

Noise Level	Input Noise Power	10*log10(Si/Ni)	Output Noise Power	10*log10(So/No)
1				
2				
3				
4				
5				
6				
7				
8				

- *Draw the plot using SNR input on the X axis and SNR output on the Y axis.* **(1.0)**

4.2.5 FM

In this part you will use your signal generator again to generate FM. The parameters are: carrier frequency of 10KHz, 1Vpp; $\Delta f = 500$ Hz for a 1KHz sinusoidal message.

Load the project from `c:/ECE316/Exp04/FM_demod_NB_noise/FM_demod_NB_noise.pjt`. Compile and load it onto the platform. Load the GEL named `FM_snr_control.gel`.

- *Measure the input signal power (Si)* **(0.25)**

Input Signal Power (measured after BPF)	

- *Measure the output signal power (So)* **(0.25)**

Output Signal Power (after demodulation)	

- *Populate the noise values below and calculate the SNR for each case.* **(1.0)**

Noise Level	Input Noise Power	10*log10(Si/Ni)	Output Noise Power	10*log10(So/No)
1				
2				
3				
4				
5				
6				
7				
8				

- *Draw the plot using SNR input on the X axis and SNR output on the Y axis.* **(1.0)**

4.3 Conclusion

In this experiment, have verified the role of noise in basic communication systems. This has hopefully given you an appreciation for the restrictions imposed by noise on AM and FM systems, and for the relative differences in performance between three different systems.

Appendix A: Common Lab Instruments

1.1 Purpose

This appendix will guide you to explore further the instruments in your laboratory. You will explore the three types of instruments commonly used in Communications courses:

- An Arbitrary Signal Generator;
- An Oscilloscope;
- A Spectrum Analyzer.

1.2 Suggested Reading

There is no standard background reading to prepare you to use the instruments. Your best option is to explore the instruments either during the experiment or at the laboratory in your free time. A good reference is [1], in particular the second half of the book which is dedicated to explaining some of the instruments found in a teaching lab. The web-sites for manufacturers such as Agilent and Tektronix do offer literature about their instruments, which may be helpful. Manuals for the instruments in the lab [2] [3] [4] are available both at the web-sites and at the lab.

1.3 The Arbitrary Signal Generator

There are two types of Arbitrary Signal Generators in use at the Communications Lab: a Tektronix AFG3021 and an Agilent 33220A. They are called *arbitrary* because they allow for waveforms to be designed and edited either on the instrument or on a computer, and then loaded onto the instrument. Some of your experiments will utilize arbitrary (i.e., made up) waveforms.

It is not the intent of this book to provide explanations on how the instruments work. It is assumed that you have already seen some or all of these instruments in laboratories for different courses. This will only deal with common problematic details.

1.3.1 Setting the Output Impedance

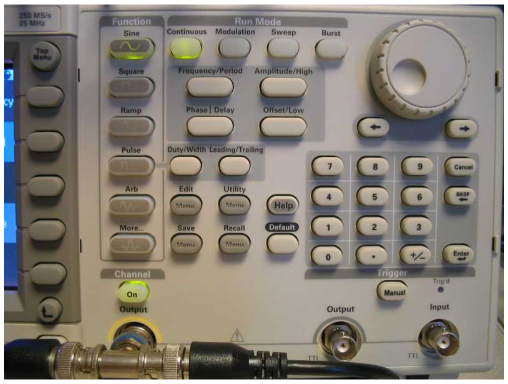
Since you will implement your systems on a DSP platform, all your input signals will be presented to an analog-to-digital converter prior to being processed by the DSP. At the input of the converter there is an operational amplifier, which has an input impedance of around 1 MΩ. By default, your signal generators assume a 50Ω load at their output, which results in having a 50Ω output impedance in series with a 50Ω load. The display of the signal generator is then preset to assume such load and display *half* of what it is actually producing. You must set it to display the proper value for a *High Z* load. If your signal generator does not allow for such adjustment, keep this difference in mind when you make your measurements.

One simple test to see if you are working with the right voltage is to connect the signal generator directly to the scope and measure the amplitude of your signal with the scope. The scope has an input impedance of about 1 MΩ. If you have *not* set your signal generator properly, you will see that the measured value is actually twice the voltage displayed by the signal generator.

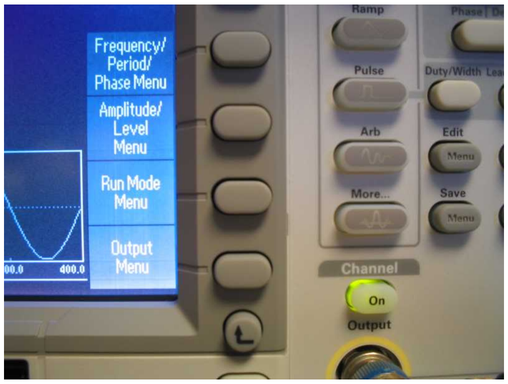
Follow the procedure below to set the output impedance for the two types of instruments listed below.

Tektronix AFG 3021

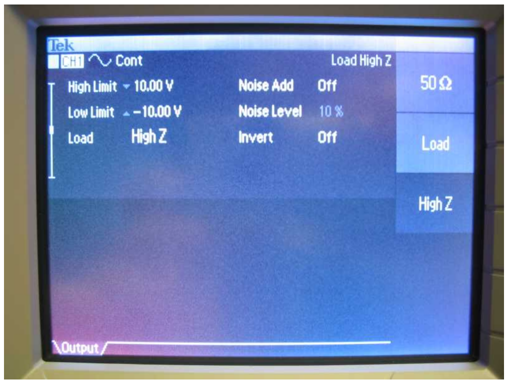
- Turn the instrument on [1]. From the front panel, press Sine. You will see on the bottom right corner of your screen a menu labeled *Output Menu.* Press on the corresponding button. Another menu will appear, in which you will have three options. Press on *Load Impedance*, then select *High Z.* From this point on, your instrument is set to display the values assuming a high impedance load on its output. To return to the main screen, just press on Sine again. Snapshots of the screen are presented below.



(a) Pressing Sine



(b) Finding Output Menu


Tek
CH1 Cont Load High Z
High Limit 10.00 V Noise Add Off
Low Limit −10.00 V Noise Level 10 %
Load High Z Invert Off

(c) Selecting High Z

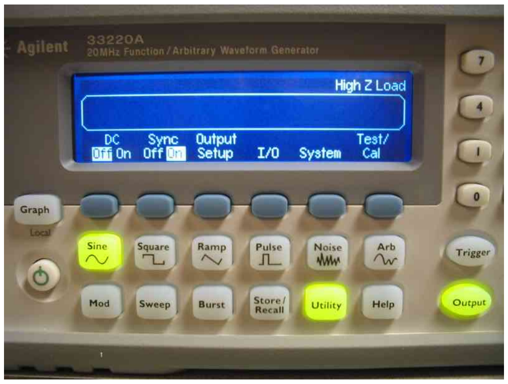
Figure 1.1: Tektronix AFG 3021

- Make sure that the **output ON** button is lit (if it is not, press it). It is located right above the BNC output connector. Otherwise you will not have a signal.

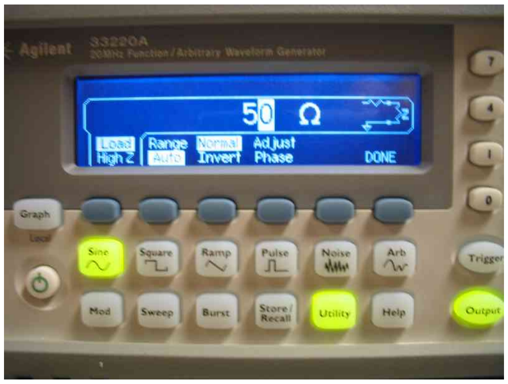
[1] yes, people do forget to do it.

Agilent 33220A or 33250A

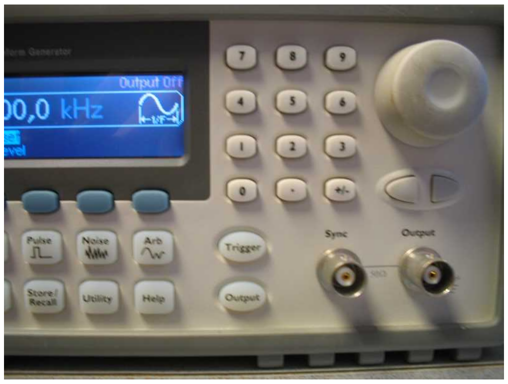
- Turn the instrument on [2]. From the front panel, press Sine. Press the *Utility* button. On the screen, you will see some options. Press *Output Setup*. Select *High Z*. **Do not forget to press "Done"**.



(a) Pressing Utility



(b) After Pressing Output Setup



(c) Where is the output button?

Figure 1.2: Agilent 33220A/33250A

- Ensure that the **Output** button is lit (if it is not, press it). Otherwise you will not have a signal. The output button (not lit) is shown on Figure 1.2(c) below.

1.4 The Tektronix TDS 2012 Oscilloscope

This section will describe the capabilities that will potentially cause you to waste precious time during the experiments. These are: the MATH/FFT function and the use of cursors for measuring voltage, time, amplitude and frequency. Whenever you are in the lab, try very hard to avoid pressing the *Autoset* button. Yes, it is convenient and very popular, but you will not learn as much about the instrument if you habitually use it. Sometimes students are lead to believe that whatever is displayed after the button is pressed must be the truth. Always read the numbers and see if they make sense. The trick is to *know what to expect* from the instrument.

Also, the *Measure* button is very convenient, but you must check if the numbers make sense. If your DSP board is putting out a 20 V_{pp} sine wave, it makes absolutely no sense. Not only its

[2] it happens to the best of families.

power source only feeds it with 12 V_{DC}, but the output of the codec cannot handle signals greater than 3.5 V_{pp}. In that case, consider that maybe your 10x probe option is on...

The Math/FFT Function

This scope allows you to see your signal in frequency domain by means of an FFT operation. You can select the Math mode by pressing a big fat red button right in the middle of the instrument.

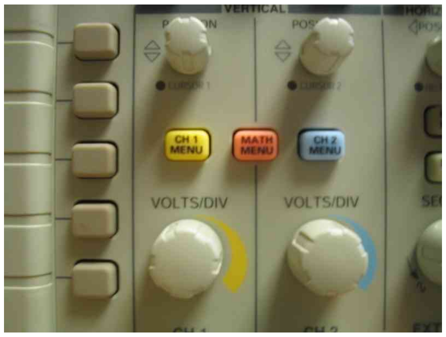
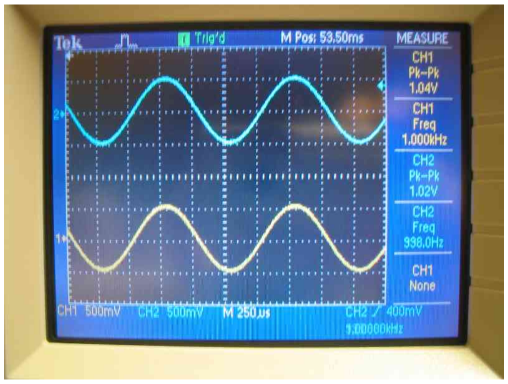
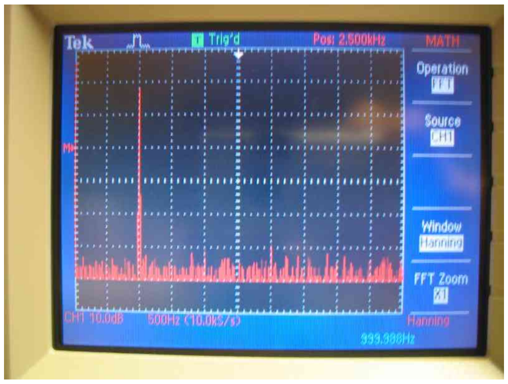


Figure 1.3: Big Fat Red Button – Math Mode

Figure 1.4 below shows you the time domain for two input channels and the corresponding frequency domain for one of those channels. Both inputs are a 1KHz, $1V_{pp}$ sine wave.



(a) Time Domain: Two channels



(b) Frequency Domain: One Channel

Figure 1.4: Tektronix TDS 2012 Oscilloscope

After pressing the Math mode button, you must select what type of operation you want the instrument to perform. For the case of an FFT, you also must select the channel (*Source*)that you want to display. Figure 1.4(b) is showing the FFT mode for Channel 1. On FFT mode, the horizontal control works “opposite” to its time domain operation. This is to say that if you want to increase the Hertz per Division reading you should turn the horizontal scale knob *clockwise*. Note that at the bottom of the screen the number displayed represents the horizontal scale units of Hz per Division. Figure 1.4(b) shows the frequency component at 1KHz, as expected.

Using Cursors

In order for you to perform specific or relative measurements between points of a signal, the instrument allows you to place cursors on the desired points, and presents you with the values at each cursor and the calculation for the difference between the cursors. Cursors can measure differences in time (vertical cursors) and amplitude (horizontal cursors), or in Math mode they can measure differences in magnitude and frequency. After pressing the *cursors* button on the top part of the front panel, you will be presented with a menu to select the signal source (channel 1, 2 or math) and the type of measurement you want. Try a simple experiment by using a direct input from the signal generator on both channels of the scope. Look at the time domain and frequency domain and use cursors to measure different points. A good choice would be a 1 V_{pp}, 1KHz square wave. What do you expect to see in the frequency domain?

1.5 The Spectrum Analyzer

The spectrum analyzers found in the Communications Lab are a standard model which for many years has been used in the industry. Rather than detailing the capabilities of the instrument, the exercise below is meant to facilitate your practical understanding of it. There are some copies of the manual in the lab, so feel free to refer to it as well if you need to explore further.

Your objective here is to visualize the FM broadcast spectrum. FM broadcast radio in North America is allotted frequencies in the VHF band between 88 and 108 MHz (100 channels). This is between the low and high TV VHF bands. Standard AM broadcast radio is allotted frequencies between 535 and 1605 KHz (107 channels).

Follow the procedure below:

- Connect a non-terminated coaxial cable (BNC-alligator) to the input of the Spectrum Analyzer. This cable will serve as an antenna (use an antenna if you have access to one.) If you are located in a central position within a building, it is unlikely that your signal strength will be satisfactory, so you need some sort of antenna.

- Press the BW button and set the resolution bandwidth to AUTO. Press FREQUENCY and set the start frequency to 100KHz, the stop frequency to 250MHz. Press AMPLITUDE and set the reference level to 0dBm. If this reference value is too high, modify it so you see some signal groups.

- Observe the display. Using the marker (MKR button) identify the low TV VHF band (54-88 MHz; channels 2-6). You can move the marker by rotating the round knob. Channel 5 on open TV is the CBC station, likely a strong signal to see. Now identify the FM broadcast radio band (88-108MHz), and the high TV VHF band (176-216MHz; channels 7-13). Channel 9 on open TV is the CTV station, likely also a strong signal.

- Set the start frequency now to 87MHz and the stop frequency to 109MHz in order to observe the FM broadcast radio band.

- Using the PEAK SEARCH function, determine the frequency of the strongest signal. Use the NEXT PK function (on the side of the display, a softkey function) to count the number of signals that one can receive in this area with fairly strong signal and little noise. If your reception is bad, you may have to move the marker by hand to identify the peaks. If you have a frequency list for the FM stations in your area, check if the frequencies you read from the instrument match those listed.

- Using the marker, identify the signals at the folloing stations: CIUT 89.5MHz (gotta have this one), Jazz FM 91.1MHz, CBC Radio 1 99.1MHz, CKFM Mix 99.9MHz and CHUM FM

104.5MHz. What is the amplitude of each signal? Any reasons why there are some stronger signals than others?

- Set the start frequency to 500KHz and the stop frequency to 1650 KHz. This encompasses the AM band. Set the reference level to -40dBm in order to observe the AM broadcast radio band (you may adjust this level if the signal strength is weak). Press the bandwidth control (BW) button and look at the menu on the right side of the screen (softkey). The VBW/RBW ratio is the ratio between the video and resolution bandwidths. If signal responses near the noise level are visually masked by the noise, the ratio should be set to less than 1 to smooth this noise. Therefore, adjust the noise to a value which gives you a good display.
- Determine the frequencies of the two strongest signals. Do you recognize these as carriers of a known station?
- Using the marker, identify some AM stations in your area. What is the amplitude of each signal?

Appendix B: On Simulink and the FFT

1.1 Introduction

This appendix provides a brief explanation on how to display and interpret the frequency domain representation of a signal using `Simulink` blocks. It is not meant as a theoretical explanation on the Fast Fourier Transform.

1.2 Building your own FFT display

FFT stands for Fast Fourier Transform. As the name says, it is an algorithm which computes the Fourier Transform (i.e., a transformation from time domain to frequency domain) very fast. It is by nature a block computation, meaning that it is performed on groups, or frames, of data. Therefore, it makes no sense to try to compute the FFT of a single sample. The input to an FFT is a group of samples and the output from the FFT is also a block of data representing the frequency domain in complex format.

You must gather a number of samples to input to an `FFT Simulink` block. This is done by "buffering" the samples, or "storing" them somewhere in memory. You must first determine which frequency resolution you will require and from that you will decide how many samples to store or buffer in [3]. If for some reason you buffer in a smaller number of samples than the size of the FFT you are about to perform, the data will be padded with zeros prior to the FFT operation, i.e., `Matlab` will pad the buffered data with zeros to extend the buffer up to the number of points you have chosen for the FFT. When the FFT is taken with this padding, it results in an output that was computed with a "partially complete" input. This is to say that your best output will be when the signal without padding is fed to the FFT, and no portion of the input signal is followed by zeroes. The number of FFT points (or FFT length) which you will use for all lab exercises is 1024, unless otherwise specified.

The output from the FFT will be a block, or frame, of 1024 complex values. These will yield magnitude and phase. Remember that now you are in the frequency domain; the first output point represents DC (or 0 Hz) and the last point represents the sampling frequency, which in the lab exercises will likely be 48KHz. This is to say that the frequency interval, or the output resolution obtained is 48,000 divided by 1024. For discrete-time (sampled) signals, the output of an FFT is made of two halves, mirrored at half of the sampling frequency. Frequencies which do not fall on integer multiples of this interval will not be represented precisely, resulting in "spectral leakage".

How about the amplitude of the output? For the 1024 point FFT of a single sinusoid, whose frequency you will make fall exactly on a multiple of the frequency interval, the amplitude of the

[3]you can think of frequency resolution as how *fine* you want your output to be. The more points you use to calculate an FFT, the finer your resultion will be.

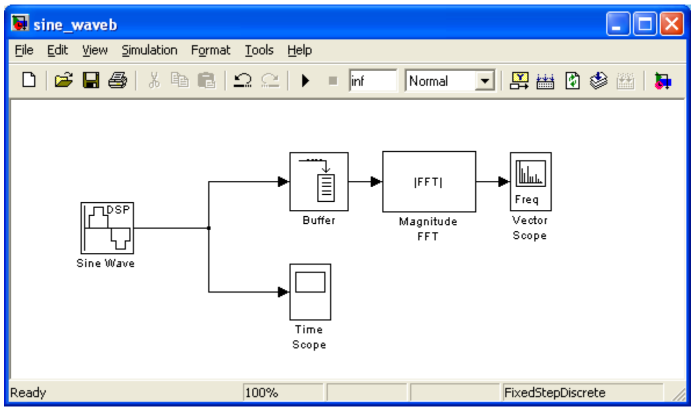
two [4] components is the amplitude peak of the sinusoid, multiplied by the number of FFT points divided by two.

To summarize now, you know you should buffer data prior to performing an FFT with at least as many samples as the number of points of the FFT, you know that 1024 points will be your FFT length and that you should try to display the output of the FFT from 0Hz (DC) to 48KHz (Fs), realizing that this display is mirrored around Fs/2. For all lab exercises, you can use these numbers and they will yield a good picture.

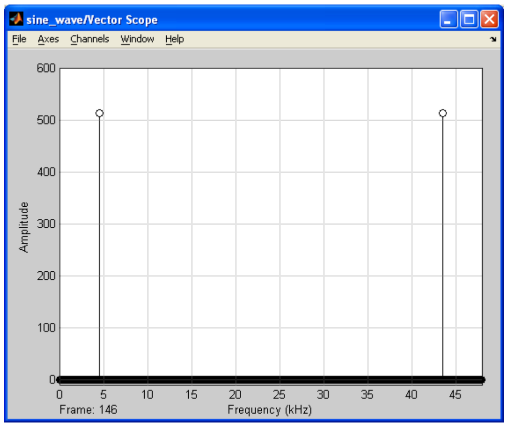
Things are better learned when they are done. Now you will design your own spectrum analyzer in `Simulink`. You can achieve this by using three blocks: a `Buffer`, a `Magnitude FFT` and a `Vector Scope` block. These blocks are all found within the `Signal Processing Blockset`. The `Buffer` block is under `Signal Management - Buffers`. The `Magnitude FFT` is under `Transforms`, and the `Vector Scope` is under `Signal Processing Sinks`. These three will provide you with a good simulated spectrum analyzer. Drag these three into your model, so that you will have a sine wave generator connected to the time domain display as well as your version of a frequency domain display. These displays will be used extensively through the exercises.

You should now have a system resembling the one on Figure 1.5(a). Double-click on the blocks to adjust the parameters as follows. For your `Buffer` block use 1024 as buffer size, zero overlap and zero initial conditions. Select "Magnitude" as the output to your FFT bock and set 1024 as the FFT length. Finally, on the `Vector Scope`, select "Frequency" as your input domain, click on the tab "Axis Properties" and set the frequency range to [0...Fs] and the Amplitude Scaling to "Magnitude". Leave all other parameters as they are.

Run the simulation again and you should see two "extra" windows: one representing your signal in the time domain and another representing it in the frequency domain. Try to explain what you see in the frequency domain, based on what you learned in the above paragraphs. You should see the picture as represented in Figure 1.5(b), for a 1500Hz, 1 V_p Sinusoid.



(a) Simulation Model



(b) Frequency Domain Display

Figure 1.5: Visualizing Time and Frequency Domain

There are some things which annoy everyone, including the TAs. One is the surreptitious data point. When the frequency domain display pops up, go under "channels – markers" and select the

[4]remember, the output is made of two mirror images. For a single frequency you will see an impulse between DC and Fs/2 and another between Fs/2 and Fs

little circle. After you do that, two circles will appear on your display, representing data that you could swear it was never there before.

So here is a question: why are the sine waves at 1KHz and 1.5KHz displayed differently in the frequency domain if both have 1 as peak value, you have sampled both at 48KHz and used the same FFT length of 1024? Try it out and see if you can explain it. This is an example of the spectral leakage briefly explained above, when the frequency of the input sinusoid does not fall on an integer multiple of the desired resolution (in this case 48,000/1024).

Appendix C

Preparation for Experiments

1.1 Introductory Experiment

• *Name:*	*Student No.:*
• *Name:*	*Student No.:*

1. Write the compact trigonometric Fourier Series for a square wave with and without a DC component (up to 4 elements in the series). Identify the fundamental and its harmonics in the equation.

2. Draw the time domain and frequency domain representation of the square wave above, identifying the fundamental and its harmonics. Assume the fundamental frequency of your square wave to be 1KHz. (Review Fourier Transform) Plot the vertical axis of the frequency domain plot as *magnitude squared*.

3. Now suppose that instead of a square wave you have a pulse with a 1% duty cycle, and a period of 1ms. Draw the time domain and frequency domain representation of that signal. What is different from the previous case?

1.2 Amplitude Modulation Experiment

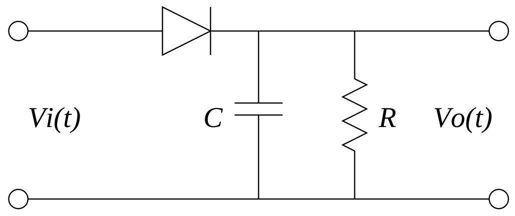
- *Name:* *Student No.:*
- *Name:* *Student No.:*

1. Write an equation for an AM-modulated signal (AM-DSB), and explain the significance of all constants in the equation. Present a `Simulink`-style block diagram to implement an AM modulator. **0.6pt**

2. Explain overmodulation. **0.2pt**

3. Sketch the AM modulated signal (assume sinusoidal signal modulating a sinusoidal carrier) in both time and frequency domain. **0.4pt**

4. Calculate the values for R and C on the envelope detector given below for a local radio station. Make sure to use values which are commercially available for the components (otherwise, you will have designed something you cannot build!). Explain the role of the diode, resistor and capacitor on the demodulation of the AM signal. **0.4pt**



Envelope Detector

5. Suppose the carrier is not transmitted (AM-DSB SC). Explain (words or block diagram) how the message signal could be recovered. **0.4pt**

1.3 Frequency Modulation Experiment

• *Name:*	*Student No.:*
• *Name:*	*Student No.:*

1. Write an equation for an FM-modulated signal, and explain the significance of all elements in the equation.

2. Explain Carson's Rule and how it is expressed for the wide-band and narrow-band cases. Why is there a difference?

3. Based on the FM equation you wrote on 1, draw a `Simulink`-style block diagram to implement an FM modulator. Use the direct method (you're looking for a Voltage-Controlled Oscillator).

4. Sketch the FM modulated signal (assume a sinusoidal signal modulating a sinusoidal carrier) in both time and frequency domain. How will the bandwidth be limited in the frequency domain? What is the spacing between the frequencies represented in the frequency domain? Is there any instance in which the carrier frequency will not appear in your frequency domain plot? (Refer to [**?**], pp.114 and 115)

5. Draw a `Simulink` block diagram for an FM demodulator based on a Phase-locked loop (PLL). You can find a simplified figure on pg. 178 of [5]. Explain the principle behind the demodulation with a PLL.

1.4 Noise in AM and FM

• *Name:*	*Student No.:*
• *Name:*	*Student No.:*

1. The time domain output of an AM modulator with a single tone message signal shows the carrier component at 3dB (power) higher than the message. The carrier frequency is 12KHz and the message frequency is 1KHz. Plot what you would expect to see in the frequency domain. Show how to obtain k_a from that plot. The AM signal is measured at $2.6V_{pp}$. Calculate the average power of the full AM signal.

2. Assume the bandwidth for this AM system is 5KHz (voice). Also, assume that your input signal-to-noise ratio to the demodulator is 30dB. What is the input average noise power?

3. Calculate the expected output signal-to-noise ratio if the receiver were an envelope detector.

4. For the same input average noise power and modulated signal ($s(t)$) average power as above, would you expect the SNR at the output of the receiver to improve or worsen if you used DSB-SC and coherent detection? Why?

5. Draw a graph for SNR-input to the demodulator (X-axis) versus SNR-output (Y-axis) for DSB-SC.

6. Explain the threshold effect observed in FM. Draw a graph comparing SNR-input to the demodulator versus SNR-output indicating how the plot would change for different β values. Refer to pg. 387 of [5].

Bibliography

[1] P. Kumar, *Digital Signal Processing Laboratory*, CRC Press, 2005

[2] Tektronix, *TDS1000 and TDS2000 Series Digital Storage Oscilloscope - User Manual*

[3] Agilent Technologies, *Agilent 33220A - 20MHz Function Arbitrary Waveform Generator - User's Guide* Agilent Technologies, Inc. 2003

[4] Tektronix, *AFG3000 Series Arbitrary/Function Generators - Quick Start User Manual*

[5] S. Haykin *Communication Systems*, 4th Edition. Toronto: John Wiley & Sons, Inc., 2001.

[6] B. P. Lathi, *Modern Digital and Analog Communication Systems*, 3rd Edition. New York: Oxford University Press, 1998.

[7] I. Glover and P. Grant *Digital Communications*, Prentice Hall, 1998

[8] A. Bateman, *Digital Communications - Design for the Real World*, Addison-Wesley

[9] W.Tomasi, *Electronic Communication Systems, Fundamentals Through Advanced*, 2nd Ed., Prentice Hall

[10] Oppenheim, Willsky, with Young, *Signals and Systems*, 2nd Edition, Prentice Hall

[11] S. Orfanidis, *Introduction to Signal Processing*, Prentice Hall

[12] E. Ifeachor and B. Jervis, *Digital Signal Processing - A Practical Approach*, Addison Wesley

[13] J. Stein, *Digital Signal Processing - A Computer Science Perspective*, Wiley Interscience 2000

[14] A. Bateman, I. Paterson-Stephens, *The DSP Handbook - Algorithms, Applications and Design Techniques*, Prentice Hall, 2004

[15] R. Chassaing, *DSP Applications Using C and the TMS320c6x DSK*, Wiley, 2002

[16] R. Chassaing, *Digital Signal Processing and Applications with the c6713 and c6416 DSK*, Wiley, 2004

[17] N. Ketharnavaz, *Real-Time Digital Signal Processing*, Elsevier, 2004

[18] S. M. Kuo and B. H. Lee *Real-Time Digital Signal Processing - Implementations, Applications and Experiments with the TMS320c55x*, Wiley, 2001

[19] P. Embree, *C Algorithms for Real-Time DSP*, Prentice Hall, 1995

[20] S. A. Tretter, *Communication System Design Using DSP Algorithms - With Laboratory Experiments for the TMS320c6701 and TMS320c6711*, Kluwer Academic, 2003

[21] N. Dahnoun, *Digital Signal Processing Implementation - Using the TMS320c6000 DSP Platform* Prentice Hall, 2000

[22] J. B. Dabney and T. L. Harman, *Mastering* `Simulink`, Pearson Prentice Hall, 2004

[23] The Mathworks, Inc., *Using Simulink.*

[24] The Mathworks, Inc., *Real-Time Workshop.*

Index