Experiment # 1

Introduction to `Matlab`, `Simulink`, Code Composer Studio and Spectrum Analyzers

1 Purpose

The purpose of this first experiment is to familiarize you with the laboratory tools needed throughout the semester. The tools you will utilize here are standard in both the university environment and the industry. Throughout the programme, the communications laboratory experiments will provide you with enough practical experience to utilize state-of-the-art software tools and a DSP platform at a level compatible with the current expectations of the industry.

Each work group should be comprised of two students per workstation, and there is no detriment to the learning experience if a student works alone. The main tools that the students will use are:

- `Matlab`, a mathematics package which includes toolboxes for communications, signal processing, filter design and code generation for specific hardware targets;
- `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 DSK platform, based on the TMS320C6711 DSP. This package also provides the channel used to write data to and retrieve data from `Matlab` during the experiments, and the means through which the simulated systems are downloaded from `Simulink` onto the DSP platform.

Though only some experiments will make use of extra circuitry, laboratory skills for assembling and testing simple circuits are desired. The students must be already comfortable with the use of signal generators and oscilloscopes and will be introduced to the use of spectrum analyzers, which are standard measurement instruments in all communications laboratories. It must be noted that the training provided is general enough so that it can be applied to the use of any spectrum analyzer.

2 Background Reading and Preparation

You should prepare for this introductory section by reviewing the core `Matlab` tools that have already been utilized in other courses of the ECE curriculum. `Matlab` will be utilized for three main purposes: to read data from a DSP platform and display the data appropriately; to generate variables that will modify - in real time - the parameters of a system running on the DSP platform; and to provide the means through which the systems designed with `Simulink` are simulated and downloaded onto the DSP platform. Specific commands for reading and writing data will be provided in the experiment outlines, but you must be familiar with generating variables (creating and manipulating vectors and matrices), basic signal processing opearations and different forms of plotting data, for instance.

A review of basic block diagrams for the systems involved in the topic of each experiment is essential. Since `Simulink` is a simulation package based on block diagrams, the prior knowledge of the diagrams involved in every experiment is advantageous. For this first experiment, you will be presented with very simple input-output block diagrams to demonstrate the capabilities of `Simulink`. For all future experiments, a **lab preparation** is to be completed individually, and from the preparation the students will know the necessary `Simulink` blocks to review and use during the experiment.

Finally, you **must** be familiar with time domain / frequency domain representation of signals. You should, therefore, review the concepts introduced on the Signals and Systems course (or a corresponding 3rd year course) of the ECE curriculum.

This lab session will not require a report; its sole purpose is to provide you with an opportunity to become familiar with the tools that will be used throughout the course.

3 Equipment

Hardware:

1. One Signal Generator;
2. One Two-Channel Oscilloscope;
3. One TMS320C6711DSK Module (with audio daughtercard) attached to a workstation;
4. Coaxial cables BNC-to-BNC.

Software:

1. `Matlab` Release 13
2. `Simulink` with ECE416 Toolbox
3. Code Composer Studio, v.2.1

4 Experiment

4.1 Introduction to `Matlab` and `Simulink`

Upon running `Matlab`, click on the `Simulink` icon. This will open the `Simulink` Library. In this library, you will find the **ECE416 blockset** and all the blocks needed to perform the experiments. In this experiment you will build four block diagrams in `Simulink`, simulate them and test the respective systems as they are running on the DSP target platform.

Start by opening a new model (File/New, or click on the little white page on the top left corner). Drag into it a sine wave generator block (`DSP Sine Wave`) and an oscilloscope block (`Time Scope`) from the ECE416 blockset. Connect the two blocks. Double click on the sine wave generator and set the parameters for amplitude 1 and frequency 1000, discrete-time. The amplitude defined represents a "peak" value for `Simulink`. When using sample-based blocks, you must specify a sampling period (i.e. 1/48000 for a 48KHz sampling frequency). This sampling period **must be** consistent with all other blocks in your system that operate in discrete-time (i.e., that require a sampling period). Also, depending on the block 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 click on "run" to start the simulation. Intuitively, you would know that what you want to see on the screen is some sort of display showing a sinusoid. This is indeed what you must see. `Matlab` will open up a small window resembling the display of an oscilloscope and the sinusoid will be plotted there. You are to find the most convenient display for you by adjusting the settings on the scope window. Now stop the simulation and connect an "FFT-based" oscilloscope (`Spectrum Scope`) to the line connecting the sine wave generator to the time scope. Double click on the FFT-based scope and adjust the parameters on it to utilize a 1024 point FFT and a buffered input, with buffer size of 1024. The FFT operation is done in blocks and the length of these is always a power of 2. The greater the number of points on the FFT, the more information you will obtain about the frequency contents of the signal. However, a longer FFT will require more computation time. Since the FFT is a block operation, it makes no sense to attempt to perform a 1024 point FFT on one single sample. Use always a buffer that is at least the same size of the FFT length, or an integer multiple of it. You will use the `DSP Sine Wave`, the `Time Scope` and the `Spectrum Scope` extensively throughout the course, so explore them to the best possible extent. You must know what are their limitations, so you can utilize them effectively in future experiments.

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 your knowledge acquired on the Signals and Systems course. You can "run" and "pause" the simulation as you wish, and even save the displayed waves onto a file, if needed. If you find that your simulation is not running long enough, go under "Simulation/Simulation Parameters" and change the number of simulation time units as you wish. The default number is 10, but most consider it too short. A better number is 1000 simulation time units, since you can change parameters as the simulation runs 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 be in 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.

Now stop this simulation and add a gain block inbetween the input (sinusoid, $0.5V_p$, 1KHz) and the output (`Time Scope`). Drag the gain block (it is a sideways triangle, as amplifiers are commonly represented in block diagrams) onto the project, and drop it in the middle of the connection between the `DSP Sine Wave` and the `Time Scope`. The connections should happen automatically, but in case they do not, make sure the input of one block is connected to the output of the preceeding block. Double-click on the gain block and set the gain to be of a factor of 2. Now run your simulation and see if the result is what you expected.

Next, insert a two-input adder after the gain block. Drag onto your project a Gaussian noise generator (found under the Communications Blockset/Sources/Noise Generators), and connect its output to the second input of the adder. Double-click on the Gaussian Noise block, and you will notice that it too requires a sampling time to be defined. Set it to 1/48000, to be consistent with all other discrete-time blocks. Based on your knowledge of probability density functions, try to interpret the parameters of that block (the "help" button will bring up a page with limited information on each of the parameters). Now run the simulation and observe the effects on both the time domain scope and the FFT-based scope. Try to explain what is happening based on your prior knowledge of Signals and Systems.

Finally, move the gain to be placed after the two-input adder (be sure that all sampling periods are the same, or an error message will appear). Run the simulation and observe the results, both in time and frequency. Does the new location of the gain cause a difference on your output signal?

Move on to the next section, but if time allows at the end of the lab, you are encouraged to experiment with Simulink as much as you can. You will find that being fluent in designing block diagrams will be very useful for the future experiments.

4.2 Introduction to Code Composer Studio v2.1

The system we are interested in testing first is a "straight-through" system, which will take two inputs, pass them through the DSP and send them back out in two outputs. Throughout this course, you will use this model as a base on which to build all of your other models. You can do it once and use it for your next experiments, adding on to it. Such system is provided to you (if you would like to use it) inside the `work/template` directory in `matlab6p5`; the file name is `direct.mdl`. It is better, however, if you try to build your own model. You may feel that this is lengthy and troublesome, but it is a *one time* procedure.

Open a new model in `Simulink`, and bring into it the following blocks, connecting them in this order: the ADC block, the Channel Separator block, the Channel Combiner block and the DAC block. The two outputs of the separator go into the two inputs of the combiner.You must pay attention to one detail: the default configuration of the ADC block utilizes frame-based data handling, and the default frame size must be changed to **1**. To change that, double-click on the ADC block and write **1** in the `number of samples per frame` field. This necessary change is due to the utilization of another codec with the audio daughtercard which passes the data on to the DSP sample by sample, instead of frame by frame.

Since you will be dealing with real electric signals now (as opposed to running a simulation), connect each channel of the oscilloscope to the output ports on your target hardware and a signal generator to the input ports. Use a T-connector for the inputs. The blocks that you will use on yor `Simulink` model will provide you with two channels from input to output. You can choose which of the channels you want to operate on, and you can attach the two output ports to the oscilloscope. By doing this, you will have more flexibility to monitor what is happening within your system. Use as a standard input signal a $1V_{pp}$, 1KHz sine wave.

Now you must modify the `Simulink` build option parameters to make your model compatible with the target hardware. The parameters to be set are found under `tools/real-time workshop/options`, found on the command bar of your `Simulink` model. You can do it once in this first experiment, and use the model created as a template for future experiments, so you do not have to redefine all the parameters again. By selecting "options", a window will appear to you. Follow this procedure:

- Select the "Solver" tab (top left on the window). Click on "Solver Options" and select type "Fixed-Step";
- Next, select the "Real-Time Workshop" tab. Click on the "Browse" button on the Configuration area. A new window will appear, with a list of "`.tlc`" files. Select the "`ti_c6000.tlc`" file and click on the OK button.
- Under the same "Real-Time Worshop" tab, go under Category and look under "TI C6000 Target Selection". You should have the C6711DSK selected, and must not change any other selections.
- Under the "TI C6000 Code Generation", the DSP/BIOS option should be disabled.
- Under the "TI C6000 Linker options", the option "user defined linker file" should be selected and the default user linker file should be $c:\backslash comm\_lab\backslash my\_file.cmd$.
- Under the "TI C6000 Runtime Options", the option "Create_CCS_Project" should be selected.
- Click on OK and you are ready to download your model onto the DSP platform.

You are ready to "build" the model. However, **save it** first, and use a name different than the default. There are three ways to build your model: by going under `Tools/Real-Time Workshop/Build`, by pressing the "build all" button on the top panel (centre) or by pressing `Ctrl-B`. At this stage, progress messages will appear on the `Matlab` Command Window. If any error occurs, you will be showed a new window with the specific details of the error. Try to resolve it, and if you cannot proceed, ask for assistance from your laboratory T.A.. Since you have a lot of parameters to set prior to building the model, errors should not come as a surprise.

After the code for your model has been generated, `Matlab` will load it onto Texas Instruments' DSP programming environment, called *Code Composer Studio*(CCS). This program will be opened in a new window and will indicate to you that your model has been turned into a "project". You need to compile and build the project from the files generated. You can compile the project now, by clicking on the icon "build all"(you are no longer dealing with `Simulink`; this build process is within CCS). If there is any compilation error, they will be listed on the "stdio" window at the bottom of the CCS environment.

Now that the project is visible in Code Composer Studio, explore the files avaliable on the project tree (left-hand side). Take a careful look at the code, in both .c files and .h files. From this brief survey, you should have a picture of how the project was generated. Load the compiled program (`.out` format) by selecting `File/Load Program`, and run it. Here you may run into a problem. Sometimes the executable does not load properly into the target hardware. You may have to load it **again** to get it to run properly. Having done that, press "run" (that's the little guy running on the left-hand side of the CCS window). At this point, after about a second, you should see your sinusoids (two channels) on the oscilloscope. Some oscilloscopes do not refresh automatically when a new signal is input to them. This is to say that after you make your system run, you should press "autoscale" prior to thinking that there is something wrong with it.

While it is running, change the frequency on your signal generator and observe the result. What happens when you use a 25KHz sinusoid as an input? You can run or halt the program as you wish. Try to identify how the code is generated to allow for the handling of input and output; that might come in handy in some future experiments.

You can manipulate the channels independently, by adding blocks to the corresponding path. For instance, you can insert a gain block on each of the paths, and assign different values to them. Now build (in CCS!), download and run your system and observe the output. If you would like to change one of the gains, you don't need to go back to your block diagram and repeat the lengthy procedure. You can go straight into the code and change it. Parameters utilized by the blocks are *u*sually stored at the $< project\_name > \_data.c$ file as structs in C. After you find the parameter and change it, you must compile, build, download and run your system directly from Code Composer Studio. Did it work?

At this point, you can design a more complex system. Add to the signal path the adder and the noise generator that you have used before in this experiment. In order to see the noise generated, you will need a gain block just after the generator. You must multiply the output of the noise generator by a gain (which you will determine) in order to see the effects of the noise on your input signal. This is done to adjust the numbers generated to be placed at the right position within the word passed between the CODEC and the DSP. Feel free to add gain blocks as you wish to play with the signal-to-noise ratio (SNR) and observe the output wave form on the oscilloscope.

5 Spectrum Analyzers

Up to this point in the experiment, you have become familiar with two forms of visually representing the signal you measure from the output of the system: the time-domain scope and the "FFT-based" scope. You are also very familiar with oscilloscopes, signal generators and multimeters from laboratory experiments in other courses. On this part of the experiment, you will be introduced to the use of the spectrum analyzer, which at this point is added to the set of main measurement instruments that you will use in communications. Should you go to work in the design of communication systems, most certainly you will make use of a spectrum analyzer. As the oscilloscope displays signals in the time domain, the spectrum analyzer does it in the frequency domain. You will not be using the DSP platform in this section of the experiment; your signal will come directly from the signal generator into the spectrum analyzer.

There are three main control buttons that you will be using to measure your incoming signal on

the spectrum analyzer: **frequency**, **span** and **amplitude**. The frequency button you will find useful to determine the centre frequency for your display, for instance. You can press on it, type in the desired frequency and its unit and press the "centre" option. Span will determine the width of the spectrum that will be displayed on your screen (having the centre frequency as reference). For instance, it makes not much sense to try to see a 100KHz sinusoid using a span of 50MHz (assuming your centre is at 100KHz). In case you do that, you will likely see the negative representation of that 100KHz, as well as DC (which for our spectrum analyzers is anything below 9KHz). Amplitude will determine the vertical axis of your display, in units that you can assign (dB, dBm, etc). Another useful feature is the **marker**, which is a cursor that travels along the signal on the screen and displays individual values for the points. You are free to find the best setting for you to read your signal.

Connect the output of the signal generator to the input (50 Ohm) of the spectrum analyzer. From the signal generator, output a sine wave of 100KHz, $1V_{pp}$. Your objective is to have on the screen of your spectrum analyzer the frequency domain representation of the 100KHz sine wave. Similarly to the oscilloscope, you will have to set the axis parameters, which in this case will be frequency (bandwidth) on the x-axis and amplitude on the y-axis. After you set the desired parameters, try to explain what you see. Vary the frequency and amplitude of the sine wave and verify the changes on the screen.

Now change the input wave to a square wave (same frequency, same amplitude). Re-set the bandwidth (*span*) on the spectrum analyzer. Try to explain what you see in light of what you have learned in Signals and Systems (*Fourier Series* in particular). Modify the frequency and amplitude of the square wave and verify the changes to the displayed result on the spectrum analyzer.

Since through the semester you will study modulation in amplitude and frequency, you should try them out with the signal generator to see the corresponding frequency-domain representation with the spectrum analyzer. Use a T-connector to see the same signal in time-domain on the oscilloscope.

Set the signal generator for amplitude modulation. Set the carrier frequency for a 100KHz (sinusoid) and the modulating signal to a 20KHz. Note that some of the signal generators in the lab do not allow you to set the amplitude for the modulating signal (your sinusoid); you will have to use the modulation index setting (a percentage) to achieve the desired signal. Observe the output displayed on the spectrum analyzer. Keep in mind that you will be looking for a similar display when you do the amplitude modulation experiment (that is Experiment #3). Change the modulating signal to be a square wave, with 20KHz. Observe the resulting display on the spectrum analyzer. Vary the frequency of the carrier and observe the result.

Now set the signal generator for frequency modulation, with the carrier signal as a 100KHz, 2Vpp sinusoid, and the modulating signal as a 20KHz, sine wave. Vary the frequency of the carrier and observe the result; now vary the frequency of the modulating signal and observe the results. The explanations for these changes will become clear in Experiment #4. For now, your task is to find the best way to display them.

6 Going The Extra Mile

In this experiment, a simple extra step is to generate the code for the straight-through block diagram (i.e. ADC connected directly to DAC) and add a gain (say, multiply by a factor of 2) directly on the C code. Make sure you save your model with a "name"; the packages we utilize in the lab may refuse a model with the default name assigned by `Simulink`. Do not forget to recompile and rebuild the project before running it from `Code Composer Studio`. As the course progresses, you will feel more comfortable with the way the code is generated and will be able to modify it more easily.

Another extra step you may want to try (with more time) is to modify the C code of the generated project and make your own project.

References

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

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

[3] The Mathworks, Inc., *Using Matlab*.

[4] Tme Mathworks, Inc., *Using Matlab Graphics*

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

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