Experiment 01: Tools Of The Trade

Bruno Korst - bkf@ece.utoronto.ca

Abstract
In this first laboratory experiment, you will experiment with basic principles for this course and will familiarize yourself with the tools which you will utilize extensively throughout the course. You will simulate and later run a project using a DSP platform. This outline will also serve as a guide to `Simulink` and `Code Composer Studio` if you need one in the future.

Keywords
Fourier Series — Sinusoid — Square Wave — `Simulink` — Time Domain Scope — Frequency Domain Scope — Code Composer Studio

Contents

Introduction

This is your first marked experiment. In it, you will run a series of exercises that are meant to prepare you for future experiments. All skills learned and used here will be necessary for the experimental portion of the course. It is assumed that at this point you are already familiar with the oscilloscope and the signal generators.

The software packages that you will use, `Simulink` and `Code Composer Studio` are tools widely used in both academia and the industry. The latter is used in this lab to program an TMDXLCDK138, platform, based on a dual-core ARM+DSP processor. This is one of the latest family of processors from Texas Instruments.

As a future *Engineer*, you should be able to report your results in an intelligible manner. In every experiment from now on, you will write your answers in the reporting document that is posted for the experiment, along with your preparation. Your ability to report results intelligibly will be taken into account in the marking. Prior to the experiment, it is a good idea to read the outline and become familiar with the diagrams and individual blocks, and to do the preparation for the experiment.

1. Suggested Reading

You should prepare for this first experiment by reviewing Fourier Series from your Signals and Systems course. You should be familiar with time domain / frequency domain representation of signals. You can review, for instance, [1] or [2]. For this practical session, 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.

2. Equipment and Software Tools

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

Hardware:

- One Signal Generator;
- One Two-Channel Oscilloscope;
- One TI LCDK DSP development platform attached to a PC workstation.
- Coaxial cables BNC-to-BNC and a T connector.

Software:

- `Matlab` - Release 2016A;
- `Simulink` with DSP and Signal Processing Toolbox;
- `Code Composer Studio (CCS)`, v.7.4

3. 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 `+New - Simulink - Simulink Model` icon at the top menu. When the blank `Simulink` model opens, you must open the `Library Browser`, which is the coloured icon with four small squares on the top menu. In the `Simulink` Library, you will find the blocks needed to perform all experiments in this course. This first experiment will involve a number of simulations with different signals displayed in the time and frequency domains. Towards the end of the experiment you will implement and test a system on the DSP platform using a real signal generator and scope.

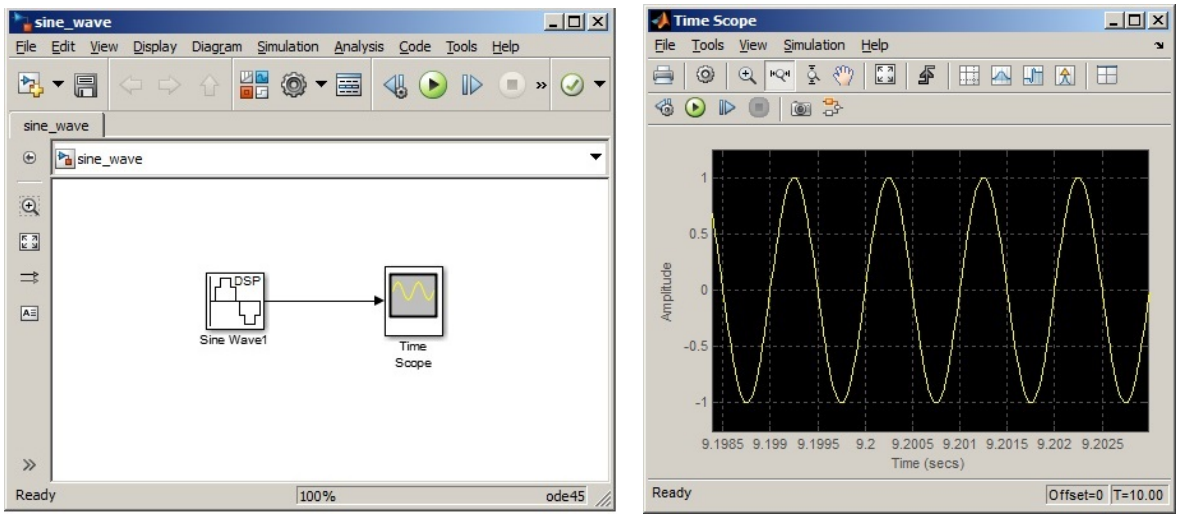
3.1 Simulating a Scope

Start by opening a new model. `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 `DSP System Toolbox` 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(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.

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 use the "Sample Time Colours" option (search for it).

Now run the simulation. You can do that by clicking on the green, right-pointing arrow on the top panel. Intuitively you know that you should see some sort of display showing a sinusoid. The simulation will run and a display will open, showing some moving yellow *smudge*. This strange signal is your sinusoid being displayed at the wrong horizontal sweep. Your job is to find the most convenient or meaningful display for you by adjusting the zoom settings on the scope window. You can set the display and axis properties from the *gear* icon on the scope window menu. After you set everything properly, your time-domain picture should look like Figure 1(b).



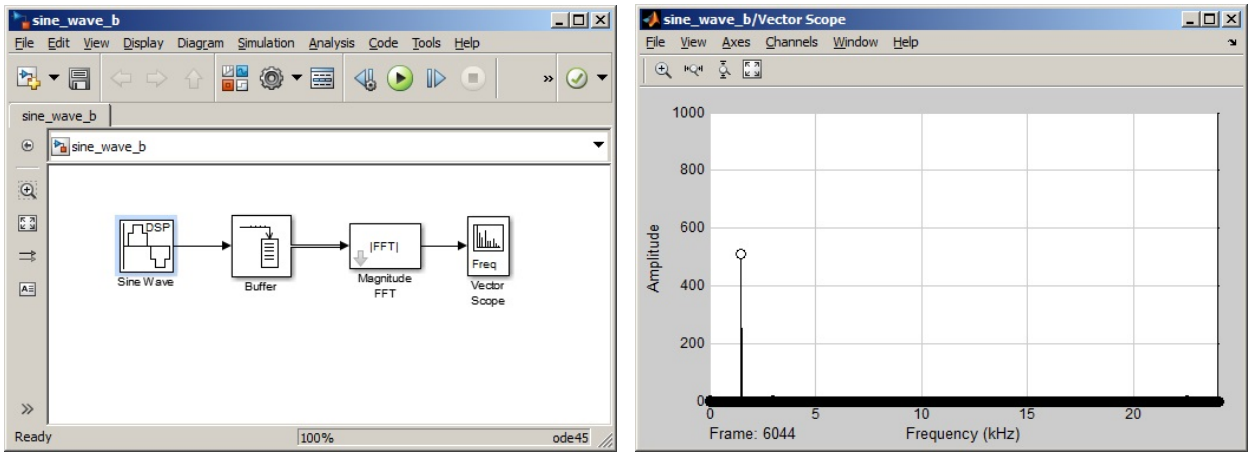
(a) Simulation Model (b) Time Domain Display

Figure 1. Simulation and Time Domain Display for a Sinusoid

3.2 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. We will use the stock time domain scope, but will create our own frequency domain one. `Simulink` does offer an `FFT-Spectrum Scope`, and you are welcome to use it. However, putting one together from simpler blocks will allow you to understand all the parts in case you need some adjustments.

A simple spectrum analyzer can be simulated using three blocks: a `Buffer`, a `Magnitude FFT` and a `Vector Scope` block. These blocks are all found within the `DSP System Toolbox`. The `Buffer` block is under `Signal Management – Buffers`. The `Magnitude FFT` is under `Transforms`, and the `Vector Scope` is under `Signal Processing Sinks`. Drag these three into a new model, along with a `DSP Sine Wave` block and connect them.



(a) Simulation Model (b) Frequency Domain Display

Figure 2. A Model For Visualizing Time and Frequency Domain

You should now have a system resembling the one on Figure 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/2] and the Amplitude Scaling to “Magnitude”. Leave all other parameters as they are.

Run the simulation again and you should see the picture as represented in Figure 2(b), for a 1500Hz, 1 V_p Sinusoid.

There are some things in `Simulink` which annoy everyone. One such thing is the elusive data point. When you run the model and the frequency domain appears, go under "channels – markers" and select the little circle. After you do that, round markers will appear on data points, showing stuff 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 [1]. Observe the changes. You can run and pause the simulation as you wish.

You may find that you would like to run the simulation longer while you analyze the signal. This is done by increasing the simulation time, found in a field on the top menu in the model window. 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 (that is, without the "measure" button, of course).

3.3 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 3. Note, however, that Figure 3 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.

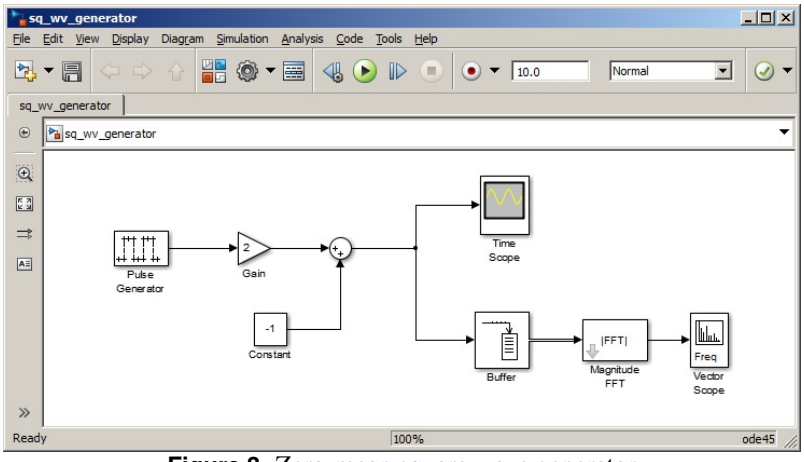


Figure 3. Zero-mean square wave generator

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, note the amplitude and frequency values and see if they make sense. Think now, how much bandwidth would you need to "transmit" a square wave? Should you attempt to do that?

[1] for more details on buffering and FFT length, refer to the supplement document on the FFT

3.4 Multiplying Two Sinusoids

This simple simulation will give you the introduction to Amplitude Modulation, which you will later study in this course. From the one sinusoid system you simulated first, 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 see what is displayed on Figure 4. Remember, this plot is mirrored at its half point because we are looking at a sampled signal.

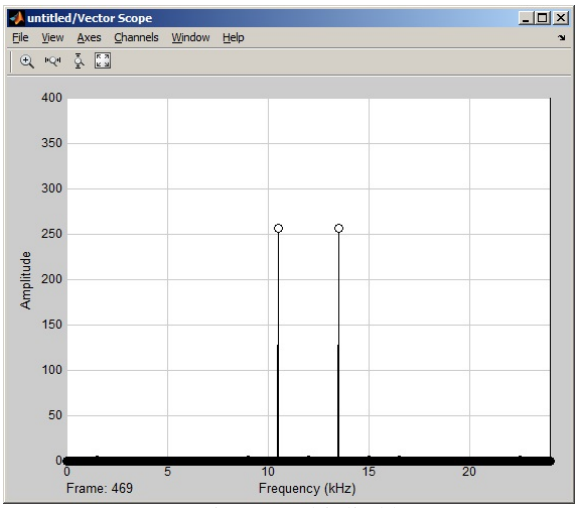


Figure 4. 1.5KHz multiplied by 12KHz

Think about your high-school trigonometry for the product$(\cos(a)\cos(b))$, and now consider also the Frequency Shift Property of the Fourier Transform (page 30 of your text).

Later on you will see that the terminology in communications calls the highest of these frequencies the *carrier*. This carrier brings the modulated signal along with itself from baseband (i.e., around DC) to a particular location in the spectrum. This location is determined by the type of system or service, such as AM, FM, TV, satellite, etc. The simplest example would be for services offering voice over the air, such as AM radio.

3.5 Adding Multiple Sinusoids

In this part of the experiment, you will try to work out the **synthesis** of a square wave from multiple sinusoids. You can use as many as you want - be careful with your time - to verify the validity of what you wrote in your preparation. At this point you should be familiar with most necessary blocks in `Simulink`, except for the addition (hint: it's called `Add`). Look for it, build your system, run it and have it display your results as you have just learned.

4. Introduction to `Code Composer Studio` v7.4

Code Composer Studio (CCS) is an IDE released by Texas Instruments to program all TI-related platforms. The development platform you will utilize in this laboratory is the TI OMAP-L138 LCDK. It features a dual-core DSP/ARM processor and multiple input/output options. In this course, you will use the audio interface. The CCS programming environment is based on Eclipse, which is widely used as an IDE for developing applications and supports multiple programming languages. For all experiments in this course, you will be given the core code – in C – and will be required to make some modifications on the code to reach your experimental objective. The focus of all courses here is not to teach programming (you have already done APS105), so the working program is given to you. You will modify the program as needed to achieve what is needed in the experiment. Below is a simple guide to get your first program running.

Upon opening `Code Composer Studio`, you should see the splash window appearing on your screen, and shortly after the main CCS window will open. All programs that run on the platform (for all courses using this lab) are listed on the left pane. You will choose the program according to the course and experiment that you are performing. All programs have been tested; the lab outlines will lead you to make them run. Figure 5 shows you a picture of what you should see.

When you click on a project title, the project becomes *active*. For this experiment, you will choose `ECE316_Exp01_INTRO`. After you click on it, you will see the message `[Active - Debug]` written next to the project name. For the code to run on

the target hardware, you will need to compile the given code, create an executable and load it onto the hardware. This is done by right-clicking on its name and selecting `Build Project`. A small progress window will appear and messages will run through the `Console` area of CCS, until the message `**** Build Finished ****` appears.

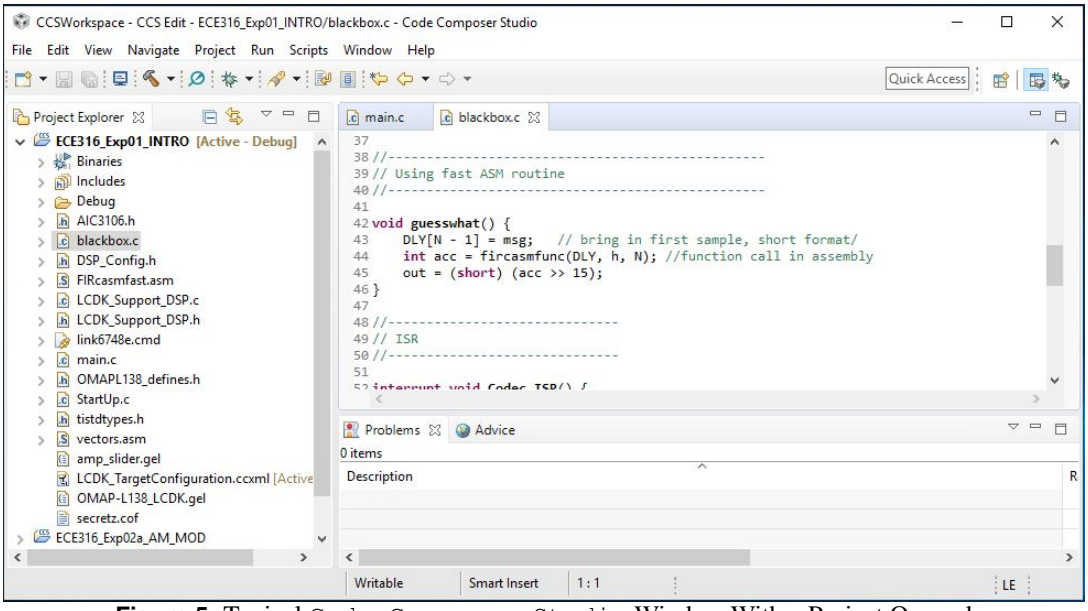


Figure 5. Typical `Code Composer Studio` Window With a Project Opened

You can compile, build *and* load the project in one shot by clicking on the *green bug* located right under and at the centre of the top menu of the CCS window. It looks like this:

Figure 6. Bug

After you compile, build and load a project, the CCS view will change from `CCS Edit` to `CCS Debug`, as indicated by the tabs on the top right corner of the CCS window. You will also notice that a `Debug` tab will appear on the left, with your project name and a message saying `Texas Instruments XDS100v2 USB Emulator` under it. All you need to do now is to click on the right-pointing green arrow (a "play button", really) that is located within the `Debug` pane.

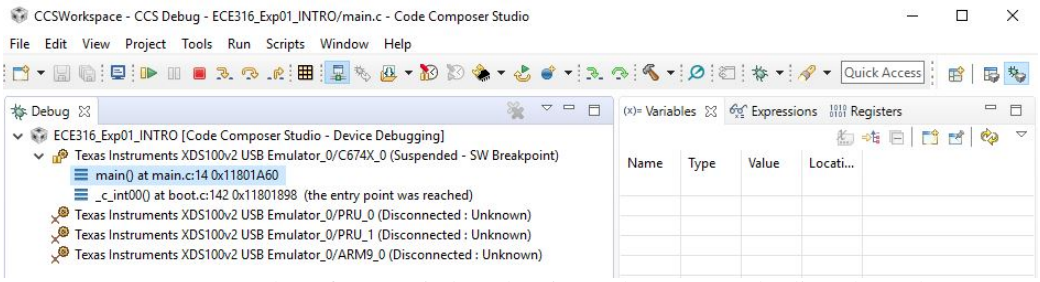


Figure 7. Snapshot of CCS Window showing Debug pane and Edit/Debug tabs

The project you are using today has files written in C (that is .c and .h) as well as assembly (.asm). Feel free to take a look at what is in these files. This is done by moving from the `CCS Debug` to the `CCS Edit` environment. You can select the desired

environment at the top right corner of the IDE. Some files are dedicated to configuring the hardware, its registers, memory and interrupts. Others pertain specifically to the program which implements the *secret* system you are to analyze in this experiment.

4.1 The Hardware

The DSP platform you are using will take an analog input in two channels from the signal generator, pass it through an analog-to-digital converter (A/D), process the input as dictated by your code, and send the output to a digital-to-analog (D/A) converter. This will then produce the analog signals that you will display on the oscilloscope. The A/D and D/A converters are found in one device, commonly known as a CODEC.

The CODEC will sample the input and transform it into numbers (i.e., quantize it) in order for the DSP to process the samples as numbers. The CODEC you will utilize is the TVL320AIC3106. It is programmable and offers sampling rates ranging from 8KHz to 96KHz. You will see from the code (check it out now, if you can find it) 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.

You should now connect the signal generator and the oscilloscope to the board. Always use two coaxial cables for the input and two for the output. We will always deal with two channels in this course. In case you need, the outline presented in the previous experiment – On Instruments – should give you more information on how to make the instruments work.

5. Conclusion

The experiment today lead you to explore the instruments that you will use in the lab. It is hoped that this was no more than a refresh practice on topics that you have already seen in other labs. The experiment also required you to explore very basic notions of time domain and frequency domain for two types of signals which are widely found in the theory.

Your next experiment will be a formal introduction to the workstation, which will encompass an introduction to MATLAB, Simulink and a DSP-based platform as your main hardware.

Acknowledgments

Thanks for all the students who have provided input on the previous versions of this experiment.

References

[1] Schafer R. W. Yoder M.A. McClellan, J.H. *Signal Processing First*. Pearson, 2003.

[2] Schafer R. W. Oppenheim A. V. *Discrete-Time Signal Processing, 3rd. Ed.* Prentice-Hall, 2009.