Supplement to Experiment # 1

Simulink and Code Composer Studio v2.1

1 Purpose

This supplement will provide you with the practice necessary to get acquainted with building and loading `Simulink` systems onto the DSP target hardware. On every experiment there will be one portion dedicated to building and testing a system that you have designed. Therefore, it is valuable for you to get familiar with all facets of this process. This is not part of an experiment; it is an extra-class activity that you come to the lab and do if you will. Some students find that during Experiment #1 there is not enough time to practice with all the tools that will be used throughout the semester.

`Simulink` is to interface seamlessly with Code Composer Studio v2.1. However, "seamlessly" is a marketing idiom to say that "as long as you don't modify anything, it will run well". It does run well, but it is extremely limited when the designer does not utilize its default values. In this lab, the intention is to modify things to see how they work. This is to say that occasinally we may have to face a new error. Feel free to explore the tools.

1.1 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 *o*ne 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". 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. You may have to load it **twice** to get it to run properly. Having loaded it again, 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, 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 (watch out for the 2-element paremeters). 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.