ECE4703 B Term 2010 Laboratory Assignment 1

Project Code and Report Due by 3pm 04-Nov-2010

The goals of this laboratory assignment are:

- to familiarize you with the TMS320C6713 DSK hardware platform,
- to familiarize you with the process of creating, building, and testing some simple projects in the Code Composer Studio (CCS) integrated development environment (IDE),
- to familiarize you with some of the debugging, profiling, and visualization tools available within the CCS IDE, and
- to develop a code framework that you can re-use in the remaining laboratory assignments in ECE4703.

1 Preliminaries

This lab assignment is intended to familiarize you with most of the tools used to develop interesting real-time DSP systems in ECE4703. A portion of this assignment is self-study and is necessary to give you a better understanding of the capabilities and structure of the lab tools. The remainder of this assignment is a hands-on exploration of the lab tools where you will develop three small C programs that will run on the TMS320C6713, interface with the basic I/O on the DSK, and perform some simple signal processing functions.

2 Self-Study

Begin by familiarizing yourself with the TMS320C6713 DSK technical reference (available as a PDF file on the course web page). You do not need to memorize everything in this document but you should become familiar with the key features and basic operation of the DSK. In Chapter 2, focus on the AIC23 codec (Section 2.2) and the LEDs and DIP switches (Section 2.5). Also be sure to check out the board layout and connector information in Chapter 3. You should know the function of all connectors on the DSK and where the main components are located.

Next, find the help file for the TMS320C6713 DSK. This used to be available through the CCS help system but now you have to find it and open it manually. Search the `C:\CCStudio_v3.1\` directory for the file called "C6713DSK.HLP". Open this file and browse through the information. Much of what you will find here with respect to the hardware is repeated from the DSK technical reference but there is also additional information on the very important "Board Support Libraries" under the "Software" section. Focus your reading on the Software section (it is ok to skip the

DSP/BIOS subsection) and be sure to look at the Examples subsection. The information in this section may be very useful for you to develop the programs described later in this assignment.

At this point, you should be familiar with the DSK hardware and the Board Support Library function calls that you can use to interface to the hardware. The final self-study task is to familiarize yourself with the Code Composer Studio integrated development environment. Code Composer Studio (CCS) is a complicated program with a massive amount of functionality. To begin to understand how to use CCS, go to Help→Tutorial. Since ECE4703 will only rely on the basic capabilities of CCS, focus your reading on the chapter entitled "Code Composer Studio IDE" and specifically on the section entitled "Developing a Simple Program". Once you have completed this section, skim through the tutorials on "Using Debug Tools" and "Profiling Code Execution". You may want to take a peek at the other sections under "Code Composer Studio IDE" chapter but most of the material in these sections will not be used in ECE4703. Following the examples in key sections of this chapter will give you the basic tools needed to develop, debug, and characterize the performance of most programs in CCS.

One final note on the tutorials: The examples use the general runtime library file rts.lib. This is incorrect for the TMS320C6713 DSK. Instead, use the library file rts6700.lib.

3 Programming Assignments

Please set up separate Code Composer Studio projects for each of the following programs. Your code must also be liberally commented to receive full credit.

3.1 Program 1: Hello World

In this part of the assignment, you will write, compile, and run one of the simplest possible programs for the TMS320C6713 DSK. The goal is to simply generate the output "Hello World" on the CCS console. Here are the suggested steps:

1. Create a new project through Project→New. Call it whatever you want. Make sure you set the "Target" field to TMS320C67XX.

2. Create the C source code for your project through File→New→Source File.

   (a) You should put some header comments at the top of your file with your name, the date, and other information about the project. Comment lines begin with "//".

   (b) The code for this assignment is quite simple. After your header comments, just type

   ```
   // helloworld.c by A. Student and B. Student 1-Nov-2010
   // This code just prints ‘‘Hello World’’ to the stdout window in CCS
   #include <stdio.h>  // include the standard I/O functions
   void main()
   {
     printf("Hello World\n");
   }
   ```

3. Save your C source code to your project directory as a C file using File→Save. Make sure you set the file type as a C source file with .c extension and make sure it ends up in the correct project directory.

4. Add your source code to the project through Project→Add Files to Project.

5. Add a linker command (.cmd) file to the project. A good linker command file can be found in the C6713 files in the Kehtarnavaz textbook CD. We will discuss this more later, but the linker command file essentially sets up the memory map for the CCS linker to make sure the code and data are loaded to the desired locations.

6. Add the run-time support library functions file rts6700.lib to the project. This can be found in the directory c6000\cgtools\lib. Do this with Project→Add Files to Project. The rts6700 library is necessary for even the simplest programs.

7. Set up the target version by going to Project→Build Options.

   (a) Go to the Compiler tab.

   (b) Click on the Basic category.

   (c) Select "C671x" in the "Target Version".

8. Try building the project at this point through Project→Build. Fix any errors that occur at this point. You may also see 2 cryptic warnings about default sizes for the ".stack" and ".sysmem" sections. These warnings are caused by a lack of explicit directions in the Kehtarnavaz linker command file for the size of the ".stack" and ".sysmem" sections. Usually, the default values chosen by the compiler are fine. To use the default values and suppress the warnings, go to Project→Build Options.

   (a) Go to the Linker tab.

   (b) Click on the Advanced category.

   (c) Uncheck "Warn about output sections".

9. Try building the project now. You should have "0 Errors, 0 Warnings, 0 Remarks". There may be one final warning about a missing newline at the end of your source code. You can get rid of this warning by putting one blank line at the end of your C code. If you are still getting errors or warnings, go back through these steps and correct any mistakes.

10. Once you have no errors or warnings, it is time to load your code to the DSK. First, connect to the DSK by pressing "Alt+C". Then load your program with File→Load Program. Go into the debug directory and select the .out file. Hit ok.

11. Now do Debug→Run and you should see "Hello World" appear in the stdout pane at the bottom of the CCS window. To run the program again, go to Debug→Restart and then Debug→Run.

12. Play around with your program a bit to see what else you can do. Try printing "Hello World" 20 times in a loop. Try setting a breakpoint. Try profiling the execution time of the printf statement. Note that you may find the printf function handy in the future for debugging your code.

3.2 Program 2: LEDs and DIP switches

This program should check the status of all four DIP switches and light up the corresponding LED for each DIP switch in the on (up) position. Here are some hints on how to achieve this functionality:

1. You will need to include the library dsk6713bsl.lib. This library contains all of the special functions for interfacing with the components on the DSK. You should already be familiar with these functions and where to find information on them if you completed the self-study portion of this assignment.

2. You will also need the library csl6713.lib which contains chip specific information for the C6713.

3. In your main C code, you should initialize the DSK before you do anything else. See the BSL documentation in the C6713DSK.HLP file for information on how to do this.

4. You should initialize the LEDs and DIP switches before you access them. See the BSL documentation in the C6713DSK.HLP file for information on how to do this.

5. Once you have initialized everything, you should enter an infinite loop that checks the DIP switches and changes the state of the LEDs appropriately.

6. Your code should compile without any errors or warnings. Load the code to the DSK and run it. Your code should work correctly should have a liberal number of comments.

3.3 Program 3: AIC23 Stereo Codec

The AIC23 stereo codec is used for analog to digital as well as digital to analog conversion on the DSK. It is optimized for audio applications and is highly configurable. In this program, you will read both channels of the codec, buffer these channels to an array of data, and then output the channels to the codec after squaring and scaling the left channel. Here are some suggestions on how to achieve this functionality.

1. Use Kehtarnavaz's Lab 2 (see pages 106-107 of Chapter 5 of your textbook) as a framework for this project. You can either start with the code you developed for the LEDs and DIP switches project (make sure you start a new project, however) and add functionality from Kehtarnavaz's Lab 2 or, vice-versa. Make sure you understand each step in Kehtarnavaz's Lab 2 code.

2. Modify the Kehtarnavaz's Lab 2 code to set the sampling frequency to 8kHz for this assignment.

3. Note that the Kehtarnavaz code uses an interrupt interface for dealing with the AIC23 codec. Some comments:

   - The code "hooks" the interrupt to the interrupt service routine (ISR) called "serialPortRcvISR". This is done in the function "hookint" and also in the file "vectors.asm". The hookint function maps a physical interrupt event (`IRQ_EVT_RINT2`) to an interrupt number (INT15) and enables interrupts. The file vectors.asm has some assembly language code that causes a branch to serialPortRcvISR to occur whenever INT15 is detected.

- Note that the DSP is sitting in an infinite loop and basically doing nothing unless it gets an INT15 signal.
- In the ISR, the codec is accessed through the `MCBSP_read` and `MCBSP_write` functions.

4. The tricky part here is how to interpret the results returned by the codec. The variable "temp" is a Uint32 data type and contains two 16-bit samples corresponding to the left and right channels. More specifically, the first 16 bits of "temp" are a signed integer representing the sample of one channel and the second 16 are a signed integer for the other channel. A convenient way to deal with this is to replace `temp` with a "union" in C, e.g.

```
union {Uint32 combo; short channel[2];} data;
```

   With this structure, you can access the individual channel samples via data.channel[0] or data.channel[1] and you can access the entire 32 bit chunk containing both channels with data.combo. Which channel is the left and which is the right? Can you experimentally verify this?

5. Test your code up to this point. Make sure that you are correctly reading the left and right channels before proceeding. You may want to generate a test signal in Matlab with different signals in the left and right channels (and listen to this test signal with headphones to ensure that the channels are correct) to confirm that your code is working correctly.

6. Declare arrays to hold the last 128 samples of the left and right channels. Increment a pointer through the array (taking care of the wraparound issues) and put the samples into the arrays as they are read. You can look at the contents of these arrays in the watch window or (even better) try using the visualization tools in CCS to plot the contents of the arrays. You may see a large discontinuity in the signal when you plot the array in CCS. Can you explain the source of this discontinuity?

7. Finally, implement the "signal-squarer" as follows:

   (a) Square the samples on the left input channel before sending them to the left output channel. Pass the right input channel samples directly to the right output channel. Note that the samples in each channel are 16-bit signed integers, hence the range of the samples is between -32768 to +32767. Make sure you declare an appropriate datatype to hold the result of the squaring operation without overflow *and* make sure you scale the result prior to sending it to the output to avoid overflow. What happens in the case of overflow?

   (b) Your code should compile without any errors or warnings. Load the code to the DSK and run it. Your code should work correctly and should be liberally commented.

   (c) Test your "signal-squarer" by sending a 1 kHz sinusoidal signal to the DSK and recording the output for analysis in Matlab. Note that there may be some strange things about the recorded signal. Is the recorded signal always positive (as you would expect from a squared quantity)? Does the recorded signal match perfectly with a squared sinusoid in Matlab? You may want to think about the squaring operation in the frequency domain to explain what is going on. Careful analysis of the recorded signal will reveal some interesting properties of the AIC23 codec and its circuitry on the DSK.

4 Additional Exploration

1. Even if you get your code working perfectly, it is important that you explore the debugging and code profiling features of CCS. The number one reason for difficulty in ECE4703 is an inability to effectively debug problems with your code. Make sure that you can:

   - Set breakpoints: It is very easy to set breakpoints in CCS. Just highlight the line where you want your code to stop and double click on it. The breakpoint is indicated by a solid red circle on the left margin. You can clear the breakpoint in the same way.
   - Watch variables: This is also fairly straightforward. Use the CCS menu to select "View → Watch Window". The watch window will appear (usually somewhere near the bottom of the CCS environment). Select the "Watch 1" tab. To add a variable, double click under the "Name" column and type the variable name. You can also specify how you want CCS to display the variable, e.g. decimal, hex, etc.

   Play around with CCS and read the tutorials to see the full range of debugging features.

2. You should explore the graphing capabilities of CCS. CCS allows you to plot signals generated and processed by your program. To do this, use the CCS menu to select "View → Graph → Time/Frequency". This will bring up a graph property dialog box where you can set various parameters. Select the type of graph you want (you can get time or frequency domain plots) and set the start address to your output buffer. Set the acquisition buffer size to the length of your output buffer (probably 1024), set the display data size to the same number and set the DSP data type to 16-bit signed integer. Finally, set the sampling rate to your actual sampling rate and hit the OK button. You should see nice plot of your data. Play around with various plot types and other parameters to become familiar with this convenient tool. You can use this tool for generating plots that can be used in your lab reports. See the CCS tutorial link on the course web page for more details on the graphing capabilities of CCS (including how to animate plots to show the data in "real-time").

3. Make sure you are able to profile your code. This skill will be very important in later laboratory assignments when you need to ensure that your more complicated DSP algorithms are running in real-time. An interesting profiling experiment is to edit your linker command file to put all code and data for Program 3 in SRAM (your linker command file is probably already set up this way). Profile your ISR to determine the average number of cycles it takes for your code to complete. Then edit your linker command file to put all code and data for Program 3 in SDRAM. Profile your ISR to determine the average number of cycles it takes for your code to complete. Is there any difference? Can you explain why or why not?

4. Finally, try out various signals in your "signal-squarer" including sinusoids with higher and lower frequencies, square waves, and even music or speech (you can use the computer's sound card output or bring in a portable audio player). Try listening to the output with a pair of headphones.

5 In Lab

Teams of two are permitted. In the case of an odd number of students in the course, one team of one/three will be formed with permission of the instructor. You will keep these lab partner(s) for all of the laboratory assignments in this course. You and your lab partner(s) will submit joint project code and lab reports that receive a single grade.

6 Specific Items to Discuss in Your Report

Please refer to the general report guidelines provided on the course web page for an overview of the ECE4703 report format. Since the first two programs in this assignment were quite simple, you do not need to discuss them in the report. Your report should focus on your methods, solutions, and results obtained with Program 3. To help understand what is going on with the "signal-squarer", you should also perform a few simple experiments with a function generator and an oscilloscope and report on the following characteristics of the AIC23 codec:

1. What is the maximum peak-to-peak input voltage (at the "line in" input) that the AIC23 codec can accept before clipping occurs?

2. What is the maximum peak-to-peak output voltage (at the "line out" output) the AIC23 codec can generate?

3. What happens when you try to sample a sinusoid with frequency higher than the Nyquist rate? To test this, set the sampling frequency of the codec to 44.1kHz and use the function generator to provide a 25kHz sinusoidal input to the codec.

4. What happens when you sample a signal with DC offset? To test this, use the function generator and apply a one volt offset to a 1kHz sinusoidal input with a one volt peak-to-peak voltage. Plot the input buffer using Code Composer Studio's graphing functions. What does the DC offset look like after sampling? Can you explain what is going on here?

5. What happens when you try to output a signal with a DC offset? Try generating an output with a DC offset by adding a constant to each sample before writing the sample to the AIC23 codec. Look at the result on the scope. What happens?

6. Discuss your SRAM/SDRAM ISR profiling results.

7. Finally, determine where the codec configuration is written in your code and take a look at the AIC23 codec datasheet (a link is provided on the course web site). Notice that, among other configuration options, the codec has registers that allow for left/right line input channel volume control in 1.5dB steps. In Sections 3.2.1 and 3.2.3 of the datasheet, it is stated that the ADC and DAC each have a full-scale range of 1.0 VRMS. Using what you know about the codec configuration, can you explain how these specifications agree or disagree with the first and second results that you obtained regarding the maximum peak-to-peak input and output voltage?

Where appropriate, include plots generated in Code Composer Studio and/or screenshots from the oscilloscope.

7 Final Remarks

Please be aware that each of the laboratory assignments in ECE4703 will require a significant investment in time and preparation if you expect to have a working system by the signoff period on the assignment's due date. This course is run in "open lab" mode where it is not expected that you will be able to complete the laboratory in the scheduled official lab time. It is in your best interest to plan ahead so that you can use the TA and instrudctor's office hours most efficiently.