ECE 320
Lab 0: Hardware Introduction

Michael Kramer and Daniel Sachs - January 13, 2002

1 Introduction

The purpose of this lab is to introduce you to the hardware and software used in the course. By the end of this lab, you should be comfortable with the basics of testing a simple real-time DSP system with the debugging environment we will be using throughout the semester. The lab consists of first connecting the lab equipment and testing a real-time DSP system with pre-written code to implement an eight-tap (eight coefficient) FIR filter. With a working system available, you will then begin to explore the debugging software used for downloading, modifying, and testing your code. Finally, some exercises are included to refresh your familiarity with MATLAB.

2 Lab Equipment

Each lab station is equipped with a Texas Instruments TMS320C549 digital signal processor chip mounted on a Spectrum Digital TMS320LC54x evaluation board. The DSP evaluation module is connected to the PC running Windows 2000 and is controlled using the PC application "Code Composer Studio," a debugger and development environment. Mounted on top of each DSP evaluation board is a Spectrum Digital surround-sound module employing a Crystal Semiconductor CS4226 codec. This board provides two analog input channels and six analog output channels at the CD sample rate of 44.1 KHz. The DSP board can also communicate with user code or a terminal emulator running on the PC via a serial data interface.

In addition to the DSP board and PC, each lab station is equipped with a function generator to provide test signals and an oscilloscope to display the processed waveforms. Other equipment in the lab includes a DVD (digital video disc) player; a communications receiver capable of pulling in a wide range of broadcast frequencies; a microphone, mixer and speakers for audio and speech projects; and a vector signal analyzer for demodulating and displaying digital communications signals.

Step 1: Connect cables

Use the provided BNC cables to connect the output of the function generator to input channel 1 on the DSP evaluation board and to connect output channels 1 and 2 of the board to channels 1 and 2 on the oscilloscope. The input and output connections for the DSP board are shown in Figure 1.

Note that with this configuration, you will have only one signal going into the DSP board and two signals coming out, allowing you to view the raw input and filtered output simultaneously on the oscilloscope. If the function generator and/or the oscilloscope are not on, turn them on now.

Step 2: Log in

Use the network ID and password provided to log into the PC at your lab station. If you are taking another lab course in the ECE department, your network ID and password for ECE 320 will be the same.

When you log in, two shared networked drives should be mapped to the computer: the `W:` drive, which contains your own private work directory, and `V:` drive, where the necessary files for ECE 320 are stored. Be sure to save any files that you use for the course on the `W:` drive. Temporary files may be stored in the `C:\TEMP` directory; however, since files stored on the `C:` drive are accessible to any user, are local to each computer, and may be erased at any time, do not store course files in these directories. On the `V:` drive, the directories `v:\ece320\54x\dsplib\`

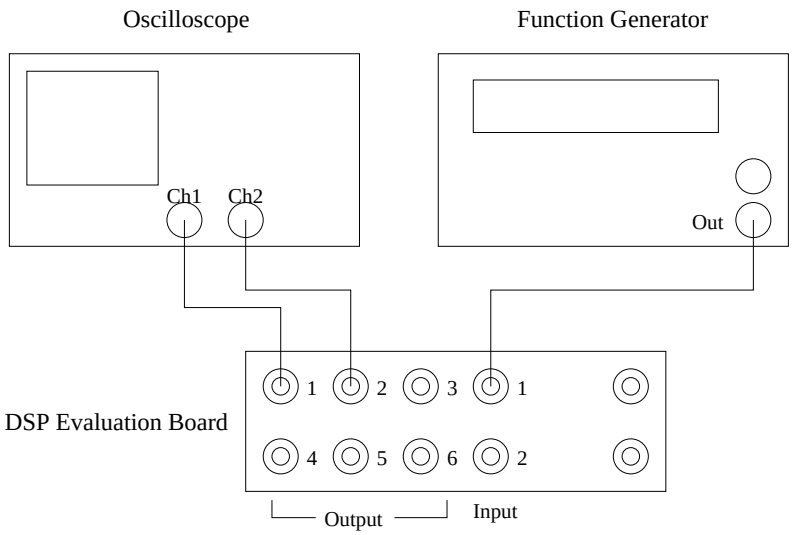

Figure 1: Example Hardware Setup

and `v:\ece320\54x\dsptools\` contain the files necessary to assemble and test code on the TI DSP evaluation boards in the lab.

Although you may want to work exclusively in one or the other of the lab-partners' network accounts, you should be sure that both partners have copies of the lab assignment assembly code. For copying between partners' directories on `W:` or for working outside the lab, FTP access to your files is available at `ftp://elalpha.ece.uiuc.edu`.

3 The Development Environment

The evaluation board is controlled by the PC through the JTAG interface (XDS510PP) using the application Code Composer Studio. This development environment allows the user to download, run, and debug code assembled on the PC. Work through the steps below to familiarize yourself with the debugging environment and real-time system with a pre-coded FIR filter (Steps 3, 4 and 5), then verify the filter's frequency response with the subsequent MATLAB exercises (Steps 6 and 7).

Step 3: Assemble filter code

Before you can execute and test the pre-written FIR filter code, you must assemble the source file. First bring up a `DOS` prompt window and type the following commands at the `DOS` prompt to create a new directory to hold the files and to copy the filter source code from the shared network drive.

- `w:`
- `mkdir lab0`
- `cd lab0`
- `copy v:\ece320\54x\dsplib\filter.asm .`
- `copy v:\ece320\54x\dsplib\coef.asm .`

Make sure you include the periods at the end of the last two commands. Next assemble the filter code by typing

•`asm filter`

at the `DOS` prompt. The assembling process first includes the FIR filter coefficients (stored in `coef.asm`) into the assembly file `filter.asm`, then compiles the result to produce an output file containing the executable binary code, `filter.out`.

Step 4: Verify filter execution

With your filter code assembled, double-click on the Code Composer icon (labeled "CCS Compile C5000 1.20") to bring up the debugging environment. Before loading your code, you must first reset the DSP board and initialize the processor mode status register (PMST). To reset the board, select the `Reset` option from the `Debug` menu on the Code Composer application. Once the board is reset, select the `CPU Registers` option from the `View` menu, then select `CPU Register`. This will bring up a sub-window at the bottom of the Code Composer application that displays several of the DSP registers. Look for the `PMST` register; it should be set to the hexadecimal value `FFE0` to have the DSP evaluation board work correctly. If it is not set correctly, change the value of the `PMST` register by double-clicking on the value and make the appropriate change in the `Edit Register` window that comes up.

Now, load your assembled filter file onto the DSP by selecting `Load Program` from the `File` menu. (Recall your filter code is located on the `W:` drive at `W:\lab0\filter.out`.) Finally, reset the DSP again and execute the code by selecting `Run` from the `Debug` menu.

Each of the the A/D and D/A converters on the six-channel surround board operate at a sampling rate of 44.1 KHz and have an anti-aliasing filter and an anti-imaging filter, respectively, that should ideally eliminate frequencies above 22.05KHz. Also note that the converters on the 6-channel board are AC-coupled and cannot pass DC signals. The filter you are running accepts input from input channel 1 and sends output waveforms to output channels 1 and 2 (the filtered signal and raw input, respectively).

Set the amplitude on the function generator to 1.0 V peak-to-peak and the pulse shape to sinusoidal. The pulse shape is set by pressing the waveform buttons (marked with a sine, square, triangle, and sawtooth pulses). The frequency is set by pressing the `Freq` button and then turning the knob to the right of the display; the "`<<`" and "`>>`" buttons can be used to choose which digit changes as you turn the knob. Likewise, the amplitude can be set by pressing the `Amp` button and changing the displayed value with the dial.

Observe the frequency response of the filter by sweeping the input signal through the relevant frequency range. (What is the relevant frequency range for a DSP system with at 44.1kHz sampling rate?) Based on the frequency response you observe, characterize the filter as low-pass, high-pass, band-pass, or band-stop, and find its -6 dB (half-amplitude) cutoff frequency (or frequencies). It may help to set the trigger on channel 2 of the oscilloscope since the signal on channel 1 may go to zero.

Step 5: Re-assemble and re-run with new filter

Once you have determined the type of filter the DSP is implementing, you are ready to repeat the process with a different filter by including different coefficients during the assembly process. Copy a second set of FIR coefficients over to your working directory by typing

•`copy v:\ece320\54x\dsplib\coef2.asm coef.asm`

at the `DOS` prompt. This will copy over a second set of coefficients into your directory and rename them appropriately.

You can now repeat the assembly and testing process with the new filter using the `asm` instruction at the `DOS` prompt and repeating the steps required to execute the code discussed in Step 4.

Just as you did in Step 4, determine the type of filter you are running and the filter's -6 dB point by testing the system at various frequencies.

Step 6: Check filter response in MATLAB

You can use MATLAB to verify the frequency response of your filter by copying the coefficients from the DSP over to MATLAB and displaying the magnitude of the frequency response using the MATLAB command `freqz`.

The FIR filter coefficients included in the file `coef.asm` are stored in memory on the DSP starting at location `0x1000` [1], and each filter you have assembled and run has eight coefficients. To view the filter coefficients as signed integers, select the `Memory` option from the `View` menu in Code Composer to bring up a `Memory Window Options` box. In the appropriate fields, set the starting address to `0x1000` and the format to `16-Bit Signed Int`. After closing this window you will notice a memory window open displaying the contents of the specified memory locations (the numbers along the left side indicate the memory locations).

In this example, the filter coefficients are placed in memory in decreasing order; that is, the last coefficient, $h[7]$, is at location `0x1000` and the first coefficient, $h[0]$, is stored at `0x1007`

Now that you can find the coefficients in memory, bring up a MATLAB window by double-clicking on the MATLAB icon on the desktop. You can now use the MATLAB command `freqz` to view the filters response. To do this you must first create a vector in MATLAB with the filter coefficients, then use the `freqz` command to display the filter's response. For example, if we want to view the response of the three-tap filter with coefficients $-10, 20, -10$, we can use the following commands in MATLAB

- `h = [-10, 20, -10];`
- `plot(abs(freqz(h)))`

Note that you will have to enter 8 values into the coefficient vector, `h`, by changing the contents of memory locations `0x1000` through `0x1007`.

How does the MATLAB response compare with your experimental results? What might account for any differences?

Step 7: Create new filter in MATLAB and verify

During the semester MATLAB scripts will be made available to you to aid in code development. For example, one of these scripts allows you to save filter coefficients created in MATLAB in a form that can be included as part of the assembly process without having to type them in by hand (a very useful tool for long filters).

First, have MATLAB generate a "random" eight-tap filter by typing

- `h = gen_filt;`

at a MATLAB prompt. Then save this vector of filter coefficients so that you can test your new random filter using the `save_coef` MATLAB script. Do this by typing the following at the MATLAB prompt:

- `cd w:\lab0`
- `save_coef('coef.asm',flipud(h));`

The `save_coef` MATLAB script saves the vector of the coefficients `h` into the named file, in this case `coef.asm`. Note that the coefficient vector is "flipped" prior to being saved; this will make sure that the first coefficient in `h` is saved last and the last is saved first.

You may now assemble and run your new filter code as you did in Step 5.

You should notice when you load in your filter with new coefficients that the contents of memory locations

[1] The "0x" part of "0x1000" is a prefix signifying that "1000" should be understood as a hexadecimal number.

`0x1000` through `0x1007` update accordingly.

Step 8: Modify filter coefficients in memory

In addition to being able to view the contents of memory on the DSP, you can change the contents simply by double-clicking on a particular memory location and making the desired change in the pop-up window.

Change the contents of memory locations `0x1000` through `0x1007` such that the coefficients correspond to a scaled and delayed version of the input; that is, change the coefficients to

$$h(n) = 8192\,\delta(n-4)$$

Note that although you may enter the integer value of 8192, the DSP interprets this as a fractional number by dividing the integer by $32,768$ (the largest integer possible in a 16-bit two's complement register). The result is an output that is delayed by four samples and scaled by a factor of $\frac{1}{4}$. See "Introduction to TI DSP Assembly," available on the course web page, for more information about numeric representation on the DSP.

After you have made the appropriate changes (remember to change all eight coefficients), run your new "filter" and measure the delay between the raw (input) and "filtered" (delayed) waveforms. What happens to the output if you change either the scaling factor or the delay value? How many seconds does a 6-sample delay correspond to?

Step 9: Test-vector simulation

As a final exercise, you will generate a test vector in MATLAB and verify that the output of the DSP given an input specified by the test vector matches MATLAB simulations of the filter. To do this, you will generate a test waveform in MATLAB and save it as a test vector. You will then run your DSP filter using the test-vector as input and import the results back into MATLAB for comparison.

You will be expected to generate appropriate test vectors for all of the labs in this course and use these test vectors to verify your DSP assembly code. You will also be expected to use MATLAB to simulate your system's response to these test vectors. We strongly recommend that you create these MATLAB simulations before working on any assigned lab exercise, as they will help you to design and test your DSP assembly code.

The first step in using test vectors is to generate an appropriate input signal. One way to do this is to use the provided MATLAB function `sweep` to generate a sinusoid that sweeps across a range of frequencies. Type `help sweep` to learn about the parameters of the `sweep` function, then generate and save a sinusoidal-sweep signal to a DSP test vector file using the following MATLAB commands:

```
>> t=sweep(0.1*pi,0.9*pi,0.25,500);     % Generate a frequency sweep
>> save_test_vector('testvect.asm',t);  % Save the test vector
```

Next, use the MATLAB `conv` command to generate an simulated response by filtering the sweep with the filter `h` you generated using `gen_filt` above. Note that this operation will yield a vector of length 507 (which is $n+m-1$, where n is the length of the filter and m is the length of the input). You should keep only the first 500 elements of the resulting sequence.

```
>> out=conv(h,t);            % filter t with FIR filter h
>> out=out(1:500);           % Keep first 500 elements of out
```

Now, modify the `filter.asm` assembly file to use the alternate "test vector" core file. Rather than accepting input from the A/D converters and sending output to the D/A, this core file takes its input from, and saves its output to, memory on the DSP. The test vector is stored in a block of memory on the DSP evaluation board that

will not interfere with your program code or data.[2] The memory block that holds the test vector is large enough to hold a vector up to 4,000 elements long. The test vector stores data for both channels of input and from all six channels of output.

The required modification can be done by changing the first two lines of the `filter.asm` file. The assembly source file is simply a text file and can be edited using any editor of your preference such as Wordpad, Emacs, or VI. Change the first line of the file:

```
.copy  ``v:\ece320\54x\dsplib\core.asm''
```

to:

```
.copy  ``testvect.asm''
.copy  ``v:\ece320\54x\dsplib\vectcore.asm''
```

Note that the whitespace in front of the `.copy` directive is required.

These changes in the DSP assembly code copy in the test vector you created, and use an alternate core file to run the filter on the test vector. After modifying your `filter.asm` code, assemble it then load and run the file using Code Composer as before. After a few seconds, halt the DSP (using the `Halt` command under the `Debug` menu) and verify that the DSP has halted at a branch statement that branches to itself. In the disassembly window, the following line should be highlighted:

```
0000:611F F073       B     611fh.
```

Next, save the test output file and load it back into MATLAB. This can be done by first saving 3,000 memory elements (6 channels times 500 samples) starting with location `0x8000` in program memory. Do this by choosing `File->Data->Save...` in Code Composer Studio, then entering the filename `output.dat` and pressing `Enter`. Next, enter `0x8000` in the Address field of the dialog box that pops up, `3000` in the Length field, and choosing `Program` from the drop-down menu next to `Page`. (Always make sure that you use the correct length — 6 times the length of the test vector — when you save your results.)

Last, use the `read_vector` function to read the saved result into MATLAB. Do this using the following MATLAB command:

```
>> [ch1, ch2] = read_vector('output.dat');
```

Now, the MATLAB vector `ch1` corresponds to the filtered version of the test signal you generated. The MATLAB vector `ch2` should be nearly identical[3] to the test vector you generated, as it was passed from the DSP system's input to its output unchanged.

After loading the output of the filter into MATLAB, compare the expected output (calculated as `out` above) and the output of the filter (in `ch1` from above). This can be done graphically by simply plotting the two curves against each other on the same axis, for example:

```
>> plot(out,'r');   % Plot the expected curve in red
>> hold on          % Plot the next plot on top of this one
>> plot(ch1,'g');   % Plot the expected curve in green
>> hold off
```

[2] It is stored in the ".etext" section. See the Core File Documentation handout for more information on the DSP memory sections, including a memory map.

[3] Note that with quantization error due to the 16-bit memory on the DSP, the channel 2 vector will be slightly different from the MATLAB generated test vector.

You should also ensure that the difference between the two outputs is near zero. This can be done by plotting the difference between the two vectors:

```
plot(out-ch1);      % Plot error signal
```

You will observe that the two sequences aren't exactly the same; this is because the DSP computes its response to 16 bits precision, while MATLAB uses 64-bit floating point numbers for its arithmetic.

Note that to compare to vectors in this way, the two vectors must be exactly the same length, which is ensured after using the MATLAB command `out=out(1:500)` above.