Abstract

This exercise introduces the hardware and software used in the course. By the end of this module, you should be comfortable with the basics of testing a simple real-time DSP system with Code Composer Studio, the debugging environment we will be using throughout the semester. First you will connect the laboratory equipment and test a real-time DSP system with provided code to implement an eight-tap (eight coefficient) finite impulse response (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, you will create a filter in MATLAB and use test vectors to verify the DSP’s output.

1 Introduction

This exercise introduces the hardware and software used in testing a simple DSP system. When you complete it, you should be comfortable with the basics of testing a simple real-time DSP system with the debugging environment you will use throughout the course. First, you will connect the laboratory equipment and test a real-time DSP system with pre-written code to implement an eight-tap (eight coefficient) finite impulse response (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, exercises are included to refresh your familiarity with MATLAB.

2 Lab Equipment

This exercise assumes you have access to a laboratory station equipped with a Texas Instruments TMS320C5510A-200 digital signal processor chip mounted on a Spectrum Digital TMS320VC5510 evaluation board. The DUAL3006, a daughterboard produced by Educational DSP, is mounted on the external peripheral interface of the board to enable four-input/four-output capability. The evaluation module should
be connected to a PC running Windows and will be controlled using the PC application Code Composer Studio, a debugger and development environment. We will be using a 48kHz sample rate. The DSP board can also communicate with user code or a terminal emulator running on the PC via a USB interface.

In addition to the DSP board and PC, each laboratory station should also be equipped with a function generator to provide test signals and an oscilloscope to display the processed waveforms.

2.1 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. Connect output channels 1 and 2 of the board to channels 1 and 2 of the oscilloscope. The input and output connections for the DSP board are shown in Figure 1 (Example Hardware Setup). The figure may not be up to date, so ask a TA if you need help.

![Example Hardware Setup](http://cnx.org/content/m13811/1.14/)
2.2 Step 2: Log in

Use the network ID and password provided to log into the PC at your laboratory station.

When you log in, two shared networked drives should be mapped to the computer: the \textit{W}: drive, which contains your own private network work directory, and the \textit{V}: drive, where the necessary files for ECE 420 are stored. Be sure to save any files that you use for the course to the \textit{W}: drive. Temporary files may be stored in the \textit{C}: \texttt{\textbackslash TEMP} directory; however, since files stored on the \textit{C}: drive are accessible to any user, are local to each computer, and may be erased at any time, do not store course files on the \textit{C}: drive. On the \textit{V}: drive, the directory \texttt{\textbackslash ece420\textbackslash 55x} contains the files necessary to assemble and test code on the TI DSP evaluation boards.

Although you may want to work exclusively in one or the other of lab-partners’ network account, you should be sure that both partners have copies of the lab assignment assembly code.

\textbf{WARNING:} Not having the assembly code during a quiz because "it’s on my partner’s account" is \textit{NOT} a valid excuse!

For copying between partners’ directory on \textit{W}: or for working outside the lab, SFTP access to your files is available at \url{sftp://ece-serv-05.ece.uiuc.edu}.

3 The Development Environment

The evaluation board is controlled by the PC through the JTAG interface 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 using the provided FIR filter code (Steps 3, 4 and 5), then verify the filter’s frequency response with the subsequent MATLAB exercises (Steps 6 and 7).

3.1 Step 3: Assembly filter code

Before you can execute and test the provided FIR filter code, you must assemble the source file. First, bring up a \texttt{DOS} prompt window and create a new directory to hold the files, and then copy them into your directory:

\begin{itemize}
\item \textit{w}:
\begin{itemize}
\item mkdir ece420
\item cd ece420
\item copy v:\texttt{\textbackslash ece420\textbackslash 55x\textbackslash filter} .
\end{itemize}
\end{itemize}

Next, open up Code Composer Studio 3.1. A new window should pop out. From the 'Project' menu, select 'Open...' and choose the \texttt{filter.pjt} file in the directory you just copied.

There are two methods of assembling your code. The first method requires use of the command line but compiles faster since files that we use often are not recompiled. The second method only requires a click of the button but will recompile all the files used in the lab, even if no changes have been made. Use whichever method during your own debugging time, but be prepared to demonstrate both methods. Please be aware that the two methods will place the .out files in different directories.

3.1.1 Method 1: Command-line

Assemble the filter code by typing \texttt{casm55x filtercode.asm main.c} at the DOS prompt. The assembling process first includes the FIR filter coefficients (stored in \texttt{coef.asm}) into the assembly file \texttt{filtercode.asm}, then compiles the result to produce an output file containing the executable binary code, \texttt{filtercode.out}.

\url{http://cnx.org/content/m13811/1.14/}
Remember to look through the text output to verify there are no errors. Ignore the "bash.exe: warning: could not find ..." as well as the five remarks. All errors must be fixed to successfully assemble the code. Remember to re-compile after fixing any errors!

3.1.2 Method 2: Code Composer GUI

Select Rebuild All under the Project menu. In a successful build, there will be 0 errors and maybe a few warnings and remarks. The output file will be placed in a Debug folder within the project's directory. In this case, the executable binary code will be in filter.out.

3.2 Step 4: Verify filter execution

Before loading your code, you must connect to the DSP board. To connect the board, select the Connect option from the Debug menu in Code Composer.

Select the Registers option from the View menu, then select CPU Register. This will open a sub-window at the bottom of the Code Composer application window that displays several of the DSP registers. This display will be useful when debugging your code in the later labs.

Now, load your assembled filter file (filter.out or filtercode.out) onto the DSP by selecting Load Program from the File menu. Finally, execute the code by selecting Run from the Debug menu. On subsequent runs, selecting Reload Program can be used to reload the same output file without requiring user selection of the output file.

The program 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). Note that the "raw input" on output channel 2 may differ from the actual input on input channel 1, because of distortions introduced in converting the analog input to a digital signal and then back to an analog signal. The A/D and D/A converters on the six-channel surround board operate at a sample rate of 48 kHz and have an anti-aliasing filter and an anti-imaging filter, respectively, that in the ideal case would eliminate frequency content above 24 kHz. On the basis of this information, what differences do you expect to see between the signals at input channel 1 and at output channel 2? The converters on the board are also AC coupled and cannot pass DC signals.

Set the amplitude on the function generator to 1.0 V peak-to-peak and the pulse shape to sinusoidal. Adjust the function generator so that it expects a high impedance load. The sequence of button presses to accomplish this on the function generator in the lab is Shift -> Enter -> Right -> Right -> Right -> Down -> Down -> Right -> Enter. Observe the frequency response of the filter by sweeping the input signal through the relevant frequency range. Make sure the oscilloscope is set to 1M impedance. This can be accomplished by pressing channel 1 or 2 and then selecting 1M Ohm from the Imped menu. What is the relevant frequency range for a DSP system with a sample rate of 48 kHz?

Based on the frequency response you observe, characterize the filter in terms of its type (e.g., low-pass, high-pass, band-pass) and 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.

3.3 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 with the following:

- copy coef.asm coef1.asm
- copy v:\ece420\55x\filter\coef2.asm coef.asm

You can now repeat the assembly and testing process with the new filter by repeating the steps required to assemble and execute the code discussed in Step 4 (Section 3.2: Step 4: Verify filter execution).

Just as you did in Step 4 (Section 3.2: Step 4: Verify filter execution), determine the type of filter you are running and the filter's -6 dB point by testing the system at various frequencies.

http://cnx.org/content/m13811/1.14/
3.4 Step 6: Check filter response in MATLAB

In this step, you will use MATLAB to verify the frequency response of your filter by copying the coefficients from the DSP 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. The location in memory can be found by typing the name of the variable. In this case, the coefficients are stored at `coef1`. View the filtercode.asm to make sure you understand why. To view the filter coefficients as signed integers, select the Memory option from the View menu to bring up a Memory Window Options box. In the appropriate fields, set the starting address to `coef1` and the format to 16-Bit Signed Int. Click "OK" to open a memory window displaying the contents of the specified memory locations. The numbers along the left-hand 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 `coef1` and the first coefficient, \(h[0]\), is stored at `coef1+7`.

Now that you can find the coefficients in memory, you are ready to use the MATLAB command `freqz` to view the filter's response. You must create a vector in MATLAB with the filter coefficients to use the `freqz` command. For example, if you want to view the response of the three-tap filter with coefficients -10, 20, -10 you can use the following commands in MATLAB:

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

Note that you will have to enter eight values, the contents of memory locations `coef1` through `coef1+7`, into the coefficient vector, \(h\). **IMPORTANT:** Remember to divide all of the coefficients by the correct scaling factor!

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

3.5 Step 7: Create new filter in MATLAB and verify

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). These scripts may already be installed on your computer; otherwise, download the files from the links as they are introduced.

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 by typing `save_coef('coef.asm',fliplr(h));` Make sure you save the file in your own directory. (The scripts that perform these functions are available as `gen_filt.m` and `save_coef.m`. They are also available at `v:/ece420/55x/m_files`)

The `save_coef` MATLAB script will save the coefficients of the vector \(h\) into the named file, which in this case is `coef.asm`. Note that the coefficient vector is "flipped" prior to being saved; this is to make the coefficients in \(h\) fill DSP memory-locations `coef1` through `coef1+7` in reverse order, as before.

You may now re-assemble and re-run your new filter code as you did in Step 5 (Section 3.3: Step 5: Re-assemble and re-run with new filter).

Notice when you load your new filter that the contents of memory locations `coef1` through `coef1+7` update accordingly.

---

1. [http://cnx.org/content/m13811/latest/gen_filt.m](http://cnx.org/content/m13811/latest/gen_filt.m)
2. [http://cnx.org/content/m13811/latest/save_coef.m](http://cnx.org/content/m13811/latest/save_coef.m)


3.6 Step 8: Modify filter coefficients in memory

Not only can you view the contents of memory on the DSP using the debugger, you can change the contents at any memory location simply by double-clicking on the location and making the desired change in the pop-up window.

Change the contents of memory locations coef1 through coef1+7 such that the coefficients implement a scale and delay filter with impulse response:

\[
 h[n] = 8192 \delta[n - 4] \tag{1}
\]

Note that the DSP interprets the integer value of 8192 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 \(\frac{1}{4}\). More information on the DSP’s interpretation of numbers appears in Two’s Complement and Fractional Arithmetic for 16-bit Processors\(^3\).

\textbf{NOTE:} A clear and complete understanding of how the DSP interprets numbers is absolutely necessary to effectively write programs for the DSP. Save yourself time later by learning this material now!

After you have made the changes to all eight coefficients, run your new filter and use the oscilloscope to 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 long is a single-sample delay? Six-sample delay?

3.7 Step 9: Test-vector simulation

As a final exercise, you will find the output of the DSP for an input specified by a test vector. Then you will compare that output with the output of a MATLAB simulation of the same filter processing the same input; if the DSP implementation is correct, the two outputs should be almost identical. To do this, you will generate a 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 with a MATLAB simulation of the filter.

The first step in using test vectors is to generate an appropriate input signal. One way to do this is to use the MATLAB function to generate a sinusoid that sweeps across a range of frequencies. The MATLAB function \texttt{save_test_vector} (available as \texttt{save_test_vector.m}\(^4\)) can then save the sinusoidal sweep to a file you will later include in the DSP code.

Generate a sinusoidal sweep using \texttt{swEEP.m}\(^5\) and save it to a DSP test-vector file using the following MATLAB commands:

\begin{verbatim}
>> 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
\end{verbatim}

Next, use the MATLAB \texttt{conv} command to generate a simulated response by filtering the sweep with the filter \(h\) you generated using \texttt{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 vector.

\(^3\)“Two’s Complement and Fractional Arithmetic for 16-bit Processors” \(<\text{http://cnx.org/content/m10608/latest/}\>

\(^4\)\text{http://cnx.org/content/m13811/latest/save_test_vector.m}

\(^5\)\text{http://cnx.org/content/m13811/latest/swEEP.m}
The main.c file needs to be told to take input from memory on the DSP. Fortunately, the changes have already been made in the files. The test vector is stored in a block of memory on the DSP just like other variables. 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 all four channels of input and from four channels of output.

To run your program with test vectors, you will need to modify main.c as well as filtercode.asm. Both are simply text files and can be edited using the editor of your preference, including WordPad, Emacs, and VI. (The changes have already been made, but please visually verify the changes are there.) Within main.c, uncomment the #define FILE_INPUT line so that your program will rewrite input from the A/D with the test vector you specified and then save the output into a block of memory.

In filtercode.asm, add the following at the end of the .data section. Depending on where you saved testvect.asm, the reference may have to be a direct reference. If you saved it in the folder of the source code we are working with, then the following should work.

```
.copy "testvect.asm"
```

Note that, as usual, the whitespace in front of the .copy directive is required.

These changes will copy in the test vector. After modifying your 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). How many seconds do you think it should take?

Next, save the test output file and load it back into MATLAB. This can be done by first saving 2000 memory elements (4 channels times 500 samples) starting with location tv_outbuf in data memory. Do this by choosing File->Data->Save... in Code Composer Studio, then entering the filename output.dat and pressing Enter. Next, enter tv_outbuf in the Address field of the dialog box that pops up, 2000 in the Length field, and choose Data from the drop-down menu next to Page. Always make sure that you use the correct length (four times the length of the test vector) when you save your results.

Last, use the read_vector (available as read_vector.m⁶) function to read the saved result into MATLAB. Do this using the following MATLAB command:

```
≫ [ch1, ch2, ch3, ch4] = 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 to the test vector you generated, as it was passed from the DSP system’s input to its output unchanged.

**NOTE:** Because of quantization error introduced in saving the test vector for the 16-bit memory of the DSP, the vector ch2 will not be identical to the MATLAB generated test vector.

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 on the same axes; for example:

---

⁶ [http://cnx.org/content/m13811/latest/read_vector.m](http://cnx.org/content/m13811/latest/read_vector.m)

---

http://cnx.org/content/m13811/1.14/
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

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(1:length(ch1))-ch1); % Plot error signal

You will observe that the two sequences are not exactly the same; this is due to the fact that the DSP computes its response to 16 bits precision, while MATLAB uses 64-bit floating point numbers for its arithmetic. Blocks of output samples may also be missing from the test vector output due to a bug in the test vector core. Nonetheless, the test vector environment allows one to run repeatable experiments using the same known test input for debugging.