

Using the DSK In CalPoly EE Courses - Dr Fred DePiero

The DSK by Texas Instruments is a development platform for DSP applications. The platform includes Code Composer Studio (CCS) with a high performance compiler, and an external processor board. The board has an A/D and D/A for audio signals and features a C5400 series DSP processor. The board supports a few sample rates. It also includes several LEDs and switches that are available for student projects.

This handout describes setup and use of the DSK board. Two projects are discussed: 'student_diffeq' and 'student_coef'. The student_coef project permits students to specify the filter coefficients for either an FIR or IIR filter, and to set the sample rate. Upon compilation and download students may then verify the filter response using test equipment in the lab. In the student_diffeq project students implement a difference equation in their own software. Students may also use switch and LED I/O during real-time operation. Example versions of the two software projects are included in this handout. This system has high performance, state-of-the-art, DSP processor which can implement 200th-order digital filters (and above) with a 48kHz sample rate.

Audio Connections, Sample Rate

Audio connections should be already established. If not, these should be made to the pigtail connections. Input: red leads, Output blue leads. Use the red (right) RCA jack for both input and output. Input signals should be limited to $V_{pp} = 2V$ MAXIMUM.

The sample rate for the A/D and D/A may be set at $S = 48, 24, 12$ and 8kHz. Sample rate is under software control – see example programs. The DSK board has an anti-aliasing filter with a cutoff that automatically adjusts to $\sim F_s/2$, when the sample rate is changed. The A/D input also has a DC blocking filter that cuts in at ~ 40 Hz.

Unfortunately, the native analog system does not possess a flat frequency response. This has been corrected via an inverse filter. However the inverse system is only setup for the $S = 48$ kHz case. With other sample rates the frequency response is not completely flat.

Edit/Compile/Execute Cycle in CCS

Always backup your project folders and other documents to a floppy (or USB drive). When the PCs are rebooted all your documents will be erased! Note: Never reboot the DSK Board while it is connected to the PC, and certainly not when any of the TI software is running. If you do this, the Earth will melt, and you will have to reboot both the PC and DSK as described under the section on troubleshooting.

To open you project, use the 'Project -> Open...' menu option and select the 'tone.pjt' in the appropriate folder, such as 'cpee_student_coef' or 'cpee_student_diffeq'. Access files to edit via the folder tree in the left hand window of CCS, expanding 'tone.pjt' and 'Source'. See Figure 1 and see comments provided in software.

After making the necessary modifications the .c file, compile and check for errors. To compile, use the Project >> Build menu option (or toolbar icon). If you get an error, you will have to modify your C-source file and compile again. However a project free from compiling errors does not necessarily mean your source code is correct...

To load your executable into the DSK, use the File >> Load Program menu option. Select the 'tone.out' file in the 'Debug' folder, within your current project folder. CCS

will often spew screens of assembly code. These may be closed without loss of functionality.

Run your program on the DSK using 'Debug>>Run' menu option, or via the (running man) toolbar icon.

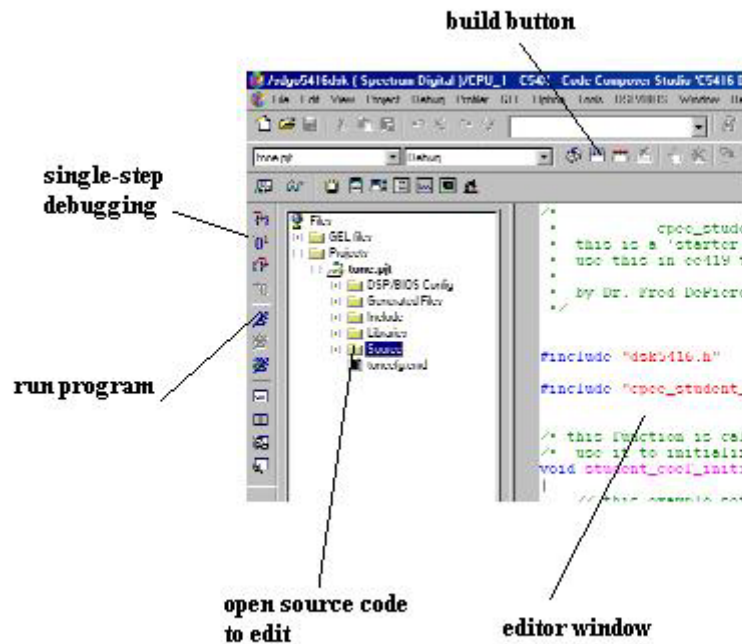


Figure 1. CCS Development Environment.

If the four LEDs all flash at a 1 Hz rate, this indicates a problem with the use of the supplied software routines. The LEDs will all flash on, and then some turn off – revealing a binary-coded error number. The value is read with lowest bit corresponding to LED 'D9'. See board. Error code values are listed in the comments with the supplied example programs.

Debugging Programs: Checking Connections and Verifying Real-Time Operation

Four LEDs and four DIP switches are located on one corner of the DSK board. Some of these have dedicated debugging purposes and others are available for use in student projects.

DIP switches #1 and #2 have been setup for special debugging purposes. Turning on switch #1 (down position) causes the board to output $y(n) = x(n)$, irrespective of any difference equation that may be established. This feature is convenient for checking signal connections. Note that even though $y(n) = x(n)$, the analog signals won't have the same phase due to the analog filters associated with the A/D and D/A. Turning on switch #2 (just #2, not both) causes the board to output a 1kHz sawtooth waveform (independent of sample rate or difference equation). This is useful for checking the output connections and is also quite helpful for verifying real-time operation.

'Real-time' operation of the DSK refers to the board's ability to complete all arithmetic and other calculations needed for the digital filter, before the next input sample

becomes ready. In cases with high order filters (above 200) and high sample rates, the board may not be able to keep up. To verify real-time operation throw switch #2 and verify that the output waveform has a frequency of 1kHz. (The sawtooth frequency has some nominal variation of ~5%. Non-real-time operation is characterized by a shift of 30-50%).

Debugging Programs: Using Single Step Debugger

This type of debugging would normally only be done by students working with the 'student_diffeq' project, who write their own software to evaluate difference equations. The single step debugger permits execution to precede one line (of C code) at a time. The value of variable may be examined by highlighting the variable name in the CCS edit window. The current line to be executed is indicated by a yellow arrow.

To begin single-step debugging, set a breakpoint in the edit window by right-clicking on a line of code in your program. Select 'Toggle Breakpoint'. Then run the program.

A Few More Notes

- Use a peak-to-peak input voltage of **V_{pp} = 2V MAXIMUM**.
- The nominal gain of the board, with $y(n) = x(n)$ is ~1.4.
- If a high-frequency oscillation is present in the output, switch to a 50 Ohm impedance for the output voltage of the function generator
- Some distortion in the output may be observed, near the zero-crossings of the output signal. Don't let this bother you.
- It is best to operate the board with a signal source connected to the input.

Troubleshooting: Rebooting

Regrettably, these boards and development tools can lock up at times. A fresh reboot and USB reconnect fixes a wide range of ailments!

1. Backup your work before rebooting!

2. Disconnect USB cable to DSK Board
3. Reboot PC and reboot DSK (cycle power off and on)
4. Wait for PC and DSK to restart
5. Reconnect USB Cable
6. Run TI Diagnostic Utility (via desktop icon, and 'Start'), then quit diagnostic tool. If the diagnostic tool fails to connect to the board (at 'DSK STARTUP') then disconnect the USB, reboot DSK, and go to step 4.
7. Run CCS (via desktop icon) and good luck...
8. Before attempting to run your program on the DSK, ensure that an appropriate signal is applied to the input.

Software Installation

The EE Technician Staff would normally perform this installation. Should you attempt this on your own, install the TI CCS version 2 using all default settings. Then copy DePiero's software into the default project folder (C:\ti\myprojects). The folders 'cpee_student_coef' and 'cpee_student_diffeq' should appear under 'myprojects'.

It will be helpful to have a shortcut to both the TI diagnostic program and CCS on the desktop. (These should appear by default via the TI install). Also recommended is a desktop shortcut to the TI project folder, which should be located at C:\ti\myprojects.

The 'cpee_student_diffeq' project

In this project students not only setup the filter coefficients and sample rate, but also implement the difference equation, in C. This project would normally be used in EE419. Students edit two subroutines, shown below. As described in the comments, the first routine is called *once* at the start of execution, and the other is called to process each input sample.

```
/*
 * ===== cpee_student_diffeq.c =====
 * by Dr. Fred DePiero
 */

/* declare any global variables here... */

/* this function is called ONCE, at the start of execution */
/* it is useful for initializing filter coefficients, or other variables... */
void student_diffeq_initialize()
{
    /* initialize global variables as desired... */

    // adjust sample rate, if desired
    // acceptable values: 48000, 24000, 12000, 8000
    diffeq_set_sample_rate(48000);

    return;
}

/* this function is called once for EVERY SAMPLE */
/* it should be used to evaluate your difference equation, or other processing */
/* input: xin, this is the current input sample, from the A/D */
/* output: yout, (returned from the function) it is sent to the D/A */
/* input and output values are 16-bit signed integers */
long int student_diffeq_evaluate(long int xin)
{
    long int yout;

    /* evaluate difference equation here... */
    yout = xin;

    /* return current output */
    return yout;
}
```

Note there are additional functions listed in the `cpee_student_diffeq.h` file that permit user programs to test the status of the 4th switch and to control the 3rd and 4th LEDs.

The 'cpee_student_coef' project

This project allows students to edit filter coefficients (a_k , b_k) and to set the system sample rate (F_s). This project would normally be used in either EE328 or in EE419. Students edit a single subroutine, shown below. This routine is called *once* at the start of execution. See comments.

```
/*
 * ===== cpee_student_coef.c =====
 * by Dr. Fred DePiero
 */

/* this function is called ONCE, at the start of execution */
/* use it to initialize filter coefficients, or set sample rate */
void student_coef_initialize()
{
    // this example sets up the difference equation:
    //  $y(n) = b_0 * x(n) + b_1 * x(n-1) + b_2 * x(n-2) - a_1 * y(n-1)$ 
    // with  $b_0 = b_1 = b_2 = a_1 = 0.25$ 

    // allocate and initialize filter
    // specify integer values for M,N (respectively)
    // M and N are defined as the maximum delays, plus 1,
    // associated with  $b_k$  and  $a_k$  coefficients
    // possible error codes: 1,2,3
    diffeq_allocate_iir(3,2);

    // set  $a_k$  coefficients of an IIR difference equation
    // recommended magnitudes: 10.0 to 0.001
    // the coefficient  $a_0$  *CAN NOT* be set, its always 1.0!
    // the parameters below are  $k$ ,  $a_k$ , where  $1 \leq k < N$ 
    // possible error codes: 5,6,7
    diffeq_set_a_coef(1, 0.25);

    // set  $b_k$  coefficients of an IIR difference equation
    // recommended magnitudes: 10.0 to 0.001
    // the parameters below are  $k$ ,  $b_k$ , where  $0 \leq k < M$ 
    // possible error codes: 6,7
    diffeq_set_b_coef(0, 0.25);
    diffeq_set_b_coef(1, 0.25);
    diffeq_set_b_coef(2, 0.25);

    // NOTE ON FIR:
    // replace the above _allocate_ function with
    // diffeq_allocate_fir(3), specifying M only
    // also, only use diffeq_set_b_coef() function calls

    // adjust sample rate, if desired
    // acceptable values: 48000, 24000, 12000, 8000
    // possible error code: 8
    diffeq_set_sample_rate(48000);

    return;
}
```