EECS 363 Digital Filtering, Winter 2017, Lab 1.   Due Jan. 13

CCC Studio v7.x Tutorial on Developing a Simple Program

Developing a Simple Program

This tutorial is adapted from the "CCStudio IDE Introduction" bundled with CCS v3.1, and originally used a C54xx processor. It shows you the basics of putting together a project and how to use common features like build options, Breakpoints, Variables windows, and Expressions windows. It is assumed that the board's USB port is connected to the PC. Though headphones are irrelevant in this exercise, in general you should have headphones connected to the low impedance "STEREO OUT" port at all times, directly or via the Y connector.

Learning Objectives:

- Create a simple program using Code Composer Studio™ tools
- Use basic debug techniques to analyze the program
- Use the tools to facilitate development of programs

Application Objective:

This lesson uses the `volume1` example to develop and run a simple program. You will create a project from scratch, add files to it, and review the code. After you build and run the program, you will change build options using the build options dialog, and fix syntax errors using the editor. Finally, you will utilize basic debug techniques like breakpoints and watch windows.

As you do this exercise you should bear in mind that your program is being executed in the C5535 processor on the eZdsp5535 board, not in your PC's microprocessor.

Creating the New Project

After opening CCS and choosing the workspace D:CCS_2016v7, create new project `smith_lab1` (or whatever your name is) in the same way you created `Hello` during the installation process. Now copy all the files in

C:\ Code Composer v7\ccsv7\tools\compiler\c5500\tutorials\volume1\

into your project folder `smith_lab1`. There should be an automatic attempt to rebuild, but there will be errors indicated, because `main` has been redefined within `volume.c`. Delete the original `main` and there should be a successful build. The appearance of an .out file in “Binaries” means that a valid project has been created.

It is okay to add to a project files that were not created in CCS, but never try to change a file, that has figured in a CCS build, outside of CCS.

In its original form this tutorial had the program read from the file `sine.dat` but that feature has been dropped here.

Connect the board via the USB cable, as earlier, and launch the configuration file created for project `smith_lab1`. Connect the target via the menu icon.

Reviewing the Source Code

View the volume.c file by double-clicking on it in the C/C++ Projects window.

Note the following functions within the code:

> After the main function prints a message, it enters an infinite loop. Within this loop, it calls the `dataIO` and processing functions.
>
> The processing function multiplies each value in the input buffer by the gain and puts the resulting values into the output buffer. It also calls the assembly language load routine, which consumes instruction cycles based on the processingLoad value passed to the routine.
>
> The dataIO function in this example does not perform any actions other than return. Rather than using C code to perform I/O, we will use a breakpoint to read data from a file on the host into the inp_buffer location.

Building and Running the Program

There should have been an automatic Build showing no Problems; if not click the relevant icon in the menu bar. There may be an automatic Load; if not do it via the icon in the menu bar.

Now Run (green icon in the menu). You get hung up with a message saying "volume example started", because the portion of the program enclosed by the preprocessor commands (#ifdef and #endif) did not run because FILEIO was undefined.

Changing Program Options and Fixing Syntax Errors

In this section, you set a preprocessor option. You also find and correct a syntax error. Open the View>Problems window, which should appear next to the Console Window.

Right click on the project folder name `smith_lab1` and choose Properties> Build>C5500 Compiler/>Advanced Options>Predefined Symbols.

In the Pre-define NAME window enter **`FILEIO`** (click on the green plus sign). Finish with OK. The code after the `#ifdef FILEIO` statement in the program is now included when you recompile the program.

After the Build the Problems window should show one error and three warnings. The error says ";" was expected on line 66. Look at line 66 in the source code. As often happens in C, the actual error is in a preceding executable statement, there was no semicolon after the right parenthesis) . Fix the syntax error and save the source code. There should be a successful Build followed by an offer to Load, which you should accept.

Click the green Resume arrow; you go into an infinite loop repeatedly saying "begin processing". Now Suspend (to right of the green arrow). Resume, Suspend, Resume etc..

Using Breakpoints and the Variables and Expressions Windows

When you are developing and testing programs, you often need to check the value of a variable during program execution. In this section, you use breakpoints and the Variables and Expressions Windows to view such values. You also use the Step commands after reaching the breakpoint. There are both software and hardware breakpoints, but this tutorial only uses software breakpoints. Hardware breakpoints are implemented by the hardware internally, and can be set in any memory type. They are typically used when software breakpoints will not work, such as in memory that cannot be written to, like ROM memory. Software breakpoints are implemented as an opcode replacement, and there is no limit to the number of software breakpoints that can be set.

The program should be loaded into the chip. Double-click on the volume.c file in the C/C++ Projects View. Put your cursor on the line number 61 (your line number may vary) in the main function to the left of the statement that says:

```
dataIO();
```

and double-click on that number. A breakpoint is established there. You can also right-click on that line and specify Breakpoint.

View>Breakpoints brings up the breakpoints window. You will see your new breakpoint listed there. Click the green Resume arrow and you should cycle through the "begin processing" loop only once. Click the green Resume arrow again and the same happens.

If you are ever forced to interrupt your work and want to keep breakpoints, then when you reopen CCS you will probably see the breakpoint icon dimmed in the edit window. It will be reactivated if, after loading the program's .out file, you just check it in the Breakpoints window. Note especially that the program must be loaded in order for breakpoints to be active.

Using the Variables Window

Choose View>Variables. The Variables window opens if it was not already there. Choose View>Expressions. The Expressions window opens if it was not already there, with the "Add new expression" tab showing.

The Variables window is used to monitor local variables and the Expressions window is used to monitor global variables, expressions, and structures. Each window has four headed columns, but you may have to use the horizontal scroll bar to see them all. At the outset you will probably see only the pointer variables `input` and `output`, because they are the only variables declared in `main`; `inp_buffer` and `out_buffer` are global, declared outside `main`. By clicking to the left of input and output you can see the values pointed to.

Choose Resume (Green arrow or F8). You go through one instance of the infinite loop and stop because of the Breakpoint; the Action for this breakpoint is "Remain halted". Choosing F6 or clicking on the Step Over icon steps you through main statements in the `while` loop that `dataIO` is in without entering functions called in that loop. Since you are still in `main` the Variables window does not change. Choosing F5 or Step Into in the toolbar makes you enter the functions called in the `while` loop and the Variables window will reflect the fact, but `gain` will not show up, because that is a global variable. No Variables will show when you are in `data` but when you are in `processing` its `size` variable will show, and change every time you Step Into (F5). When you see a value of size < 100 hit Alt-Print Screen to copy a picture of the CCS window to the clipboard, which can then be pasted into a Word document. Describe what the displayed variables do as you do the various Steps, and why. My underlining means this is something to hand in.

Using the Expressions Window with Structures

In addition to watching the value of a simple variable, you can watch the values of the elements of a structure.

1. In the Expressions window click on the words "Add new expression" and substitute **`str`**.
2. Return to save the change. The value should immediately appear but **`str`** is a structure and has no specific value.
3. Recall from "Reviewing the Source Code" that a structure of type `PARMS` was declared globally and initialized in `volume.c`. The structure type is defined in `volume.h`.
4. Click once on the arrow sign next to **`str`** (you may have to scroll left)**,** to expand this line to list all the elements of the structure and their values. The values (`Beta` etc.) may vary but should initially agree with the declarations in `volume.c` and `volume.h` of **`str`** as a global structure with certain names and values.
5. Probably `str.EchoPower = str.Ratio` = `9432`. Scroll down to Add new expression and enter as a New expression `str.EchoPower - str.Ratio`, click on the white space and you should get value `0`. Now change the value of `str.Ratio` to 9431 and click on the white space, getting the new difference 1, indicating it has just changed. Although of course you have not changed the source code, `str.Ratio` has the new value as far as execution of the program goes. The Expressions window is a convenient tool in debugging a program.

Hit Alt-Print Screen to copy a picture of the CCS window to the clipboard, which can then be pasted into a Word document.. My underlining means this is something to hand in.