

ECE2049 – Embedded Computing in Engineering Design

Lab 0 – Introduction to the MSP430F5529 Launchpad-based Lab Board and Code Composer Studio

In this lab, you will be introduced to the Code Composer Studio (CCS) development environment that we will be using to program and debug our custom MSP430F5529 Launchpad based lab board.

The computers in AK113 (and all of the other ECE labs) have fully licensed copies of Code Composer.

However, if you want, you can download a free code-size limited version of Code Composer to install on your own computer—see the course website for instructions on how to do this.

This lab is straightforward and tutorial in nature and does not have a pre-lab. You are expected to sign-off each section on the lab and to answer all the questions highlighted in yellow in your lab report.

Getting Started

1. Remove your board from its box by sliding the center portion of the box out of the cardboard sleeve. Place the board on the sleeve of the cardboard box or directly on the lab bench. When using the board, be careful that the bottom of the board does not touch a metal surface, which may cause a short circuit. For this reason, **DO NOT PLACE THE BOARD ON THE METAL COMPUTER CASE!**
2. Plug the USB cable into the board and the computer. If you do not have a USB cable packaged with your board for some reason, please ask the course staff for a replacement.
3. Once the board is connected, double-click on the desktop link to Code Composer Studio (CCS) or navigate to it through Start > All programs > Texas Instruments > Code Composer Studio 5.4.0. It is important that the cable be connected when you open CCS for the first time.
4. When CCS starts up, it will ask you to select a workspace. A **workspace** is CCS's location for storing all of your projects and configuration files. You should select a path on your network drive (**R:**) or a flash drive—do NOT select a path on the local computer. The local computers may be re-imaged at any time and you will lose your work! We recommend selecting a workspace path like **R:\ECE2049_Labs**, as shown in Figure 1.

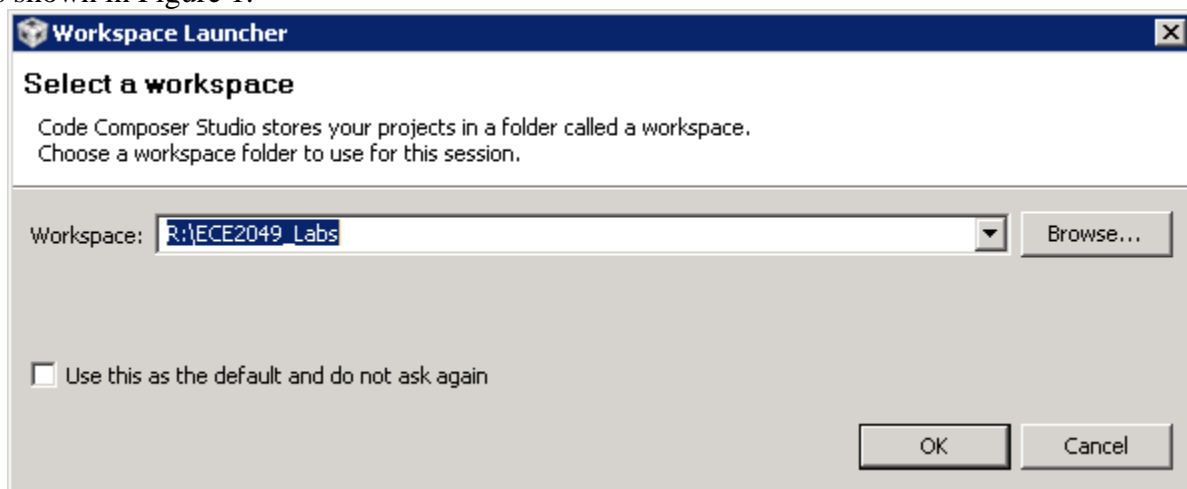


Figure 1: Selecting a Workspace

5. When CCS opens, it may prompt you to select a license file. If this happens, select the **Code Size Limited** license and click **Finish**.
6. When CCS starts for the first time, it may show a window labeled “TI Resource Explorer” that hides all of CCS’s features. To remove this, click the “X” on its tab to see the full view of CCS.
7. CCS groups windows into different “perspectives” for different tasks. At startup, CCS will be in the **Edit Perspective**, which is designed for editing and compiling your code. In the Edit perspective, CCS should look like Figure 2.

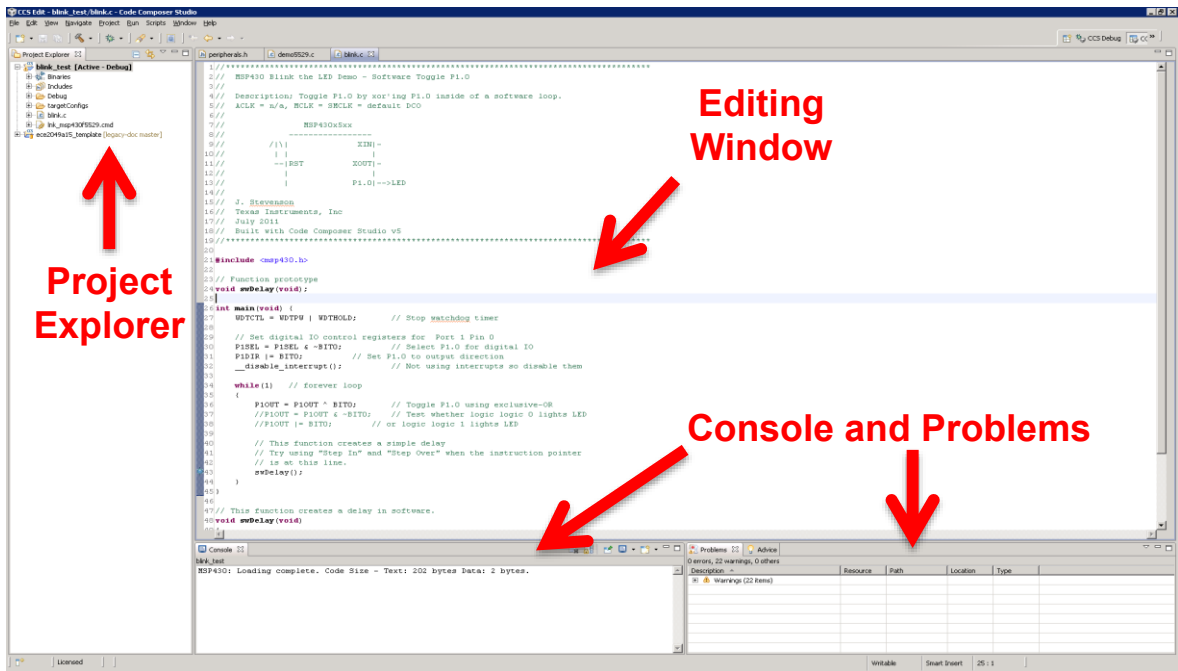


Figure 2: CCS Edit Perspective

In Edit perspective, CCS has the following panes:

- **Project Explorer:** This pane shows all of the projects that exist in the current workspace. A **project** is a set of code and configuration files that make up a single embedded application. (Your workspace likely doesn’t contain any projects yet.)
- **Editing Window:** This pane shows all of your open source files in different tabs.
- **Console and Problems Panes:** These two panes show the output produced by the compiler when building your code. The full build output is shown in the Console window; errors and warnings are highlighted in the Problems window.

Using these panes, you can perform most of the actions you need to find files in projects, edit code, and compiler it for the board. More useful features of these panes will be introduced later in this lab.

Building your first project – Don't blink!

We will start by building a simple project from a C file. This is a good way to make sure your CCS environment can program a board and demonstrate how the development process works. It will be over quickly, so don't blink!



8. Download the file **blink.c** from the course website. This file contains the source code for a simple MSP430 program, which just blinks an LED.
9. Back in CCS, go to the **Projects** menu at the top of the window and select **New CCS Project**. You should see a window that looks like Figure 3. Enter a project name and set the other fields as shown in the figure. Note that it is critical that you select the model of processor to match the one we use, the MSP430F5529, which tells the compiler how to compile and link the code for our chip. Select **Empty Project** and then click **Finish**.

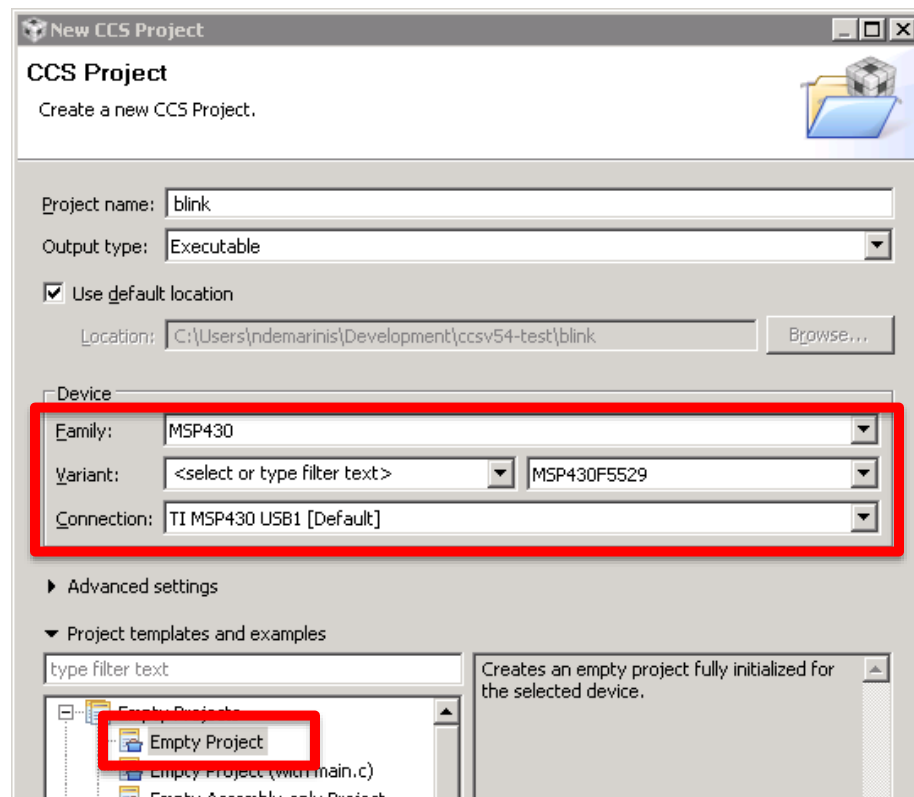


Figure 3: New CCS Project Dialog

Clicking Finish should return you to Edit perspective, where you should see an entry for the project you just created in the Projects Explorer.

10. Click on the project and expand it—you should see a few files that configure the project to use our board. **If a skeleton .c file was created in your project, you should delete it.** Note that clicking on your project sets is to the *Active Project*, as indicated next to the project’s name. This is important for some operations we will perform later in the lab.
11. Now we need to add our .c file to the project. Right-click on the name of your project to open the context menu shown in Figure 4. This is a very useful menu—it contains functionality for performing operations specific to your project. To add blink.c, select **Add Files** from the menu and browse to the file you downloaded. When prompted, select “Copy Files” to copy the new file into your project. You should now see blink.c listed in the tree of files for your project. If the file does not open in your editor window, double-click on the file in the Project Explorer and it should open.

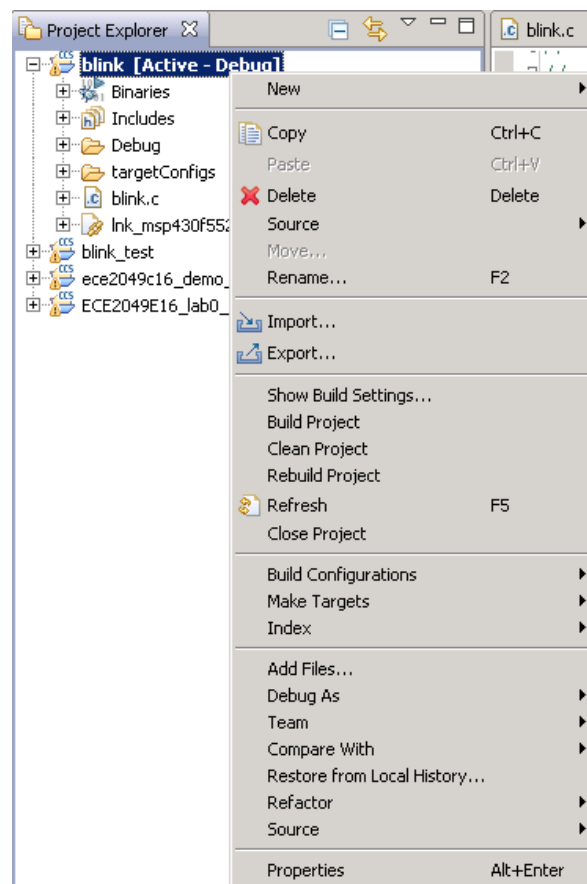




Figure 4: Project configuration menu

12. Now it’s time to build and run your code. Click on the build icon () from the toolbar at the top of the window. This will invoke the compiler to build your project. If this is the first time you’ve run these steps, CCS may take a long time (several minutes) to build your project; this is normal—CCS needs to build some libraries and drivers the first time it runs. When the build completes, you should see “Build finished” in the console and no errors in the console or problems windows. If you receive any errors, ask the course staff for help!

13. Now that the code has been compiled, it's time to load it onto your board to run it. Make sure your board is connected to the computer and click on the debug icon () in the toolbar to enter the debugger. This will make CCS connect to the board, load your program onto the MSP430, and start the debugger at the beginning of the program. If this is the first time starting the debugger, this may also take a while. During the process, if you get a pop-up window about with MSP430 Ultra Low Power (ULP) warnings, you can dismiss it. If the process is successful, CCS should switch to *Debug Perspective*, which looks like Figure 5. If you encountered any errors in the process, ask the course staff for help.

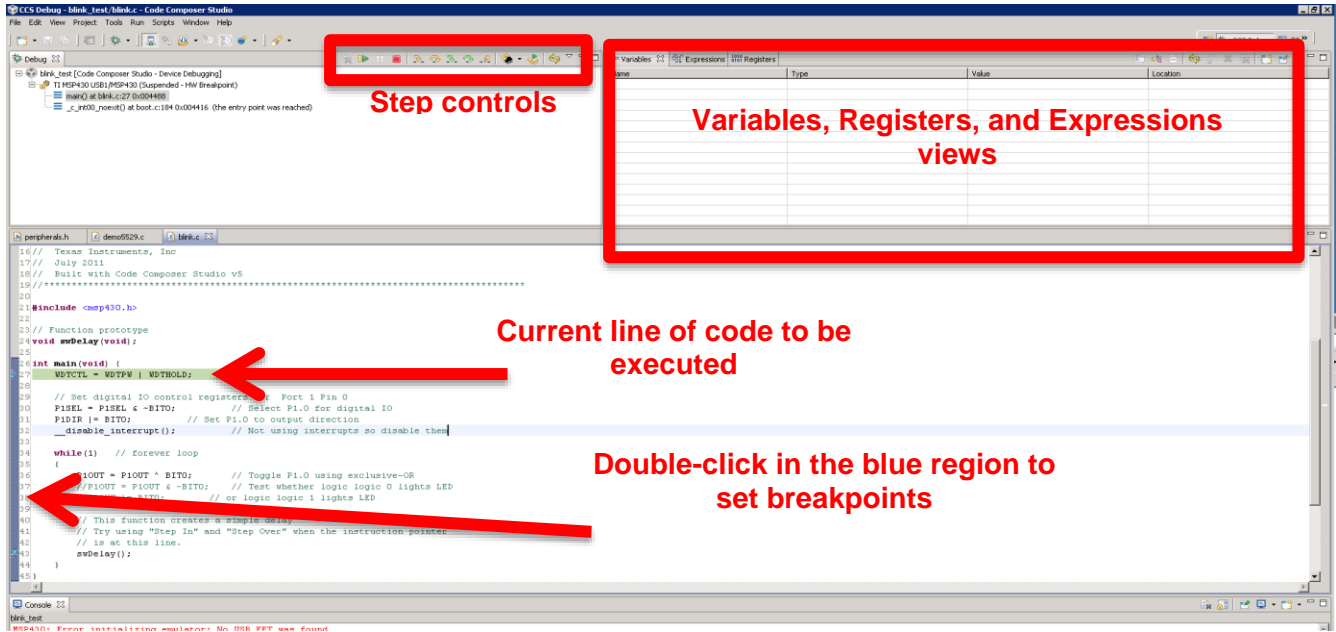



Figure 5: CCS Debug Perspective

The first line of your program should be highlighted in green. The highlighted line indicates that the position of the *program counter*, a pointer to the next line of the program that will execute. In this state, the debugger has paused the execution of your program *before* running this line and is waiting for you to let it continue.

14. The Debug perspective has many useful and important features for observing and controlling how your program executes. However, at present, we don't want to use any of these—we just want to run the program! In the toolbar labeled “Step controls” in Figure 5, click the green arrow () to tell your program to run. The Red led on the small board should be blinking! Blinking lights are cool. **Demonstrate your first embedded programming success to the course staff for signoff.** Optionally, take a video and send it to your mom to show her what you do at school.

Using the Debugger

Before continuing, let's examine the step controls more closely, shown in Figure 6.

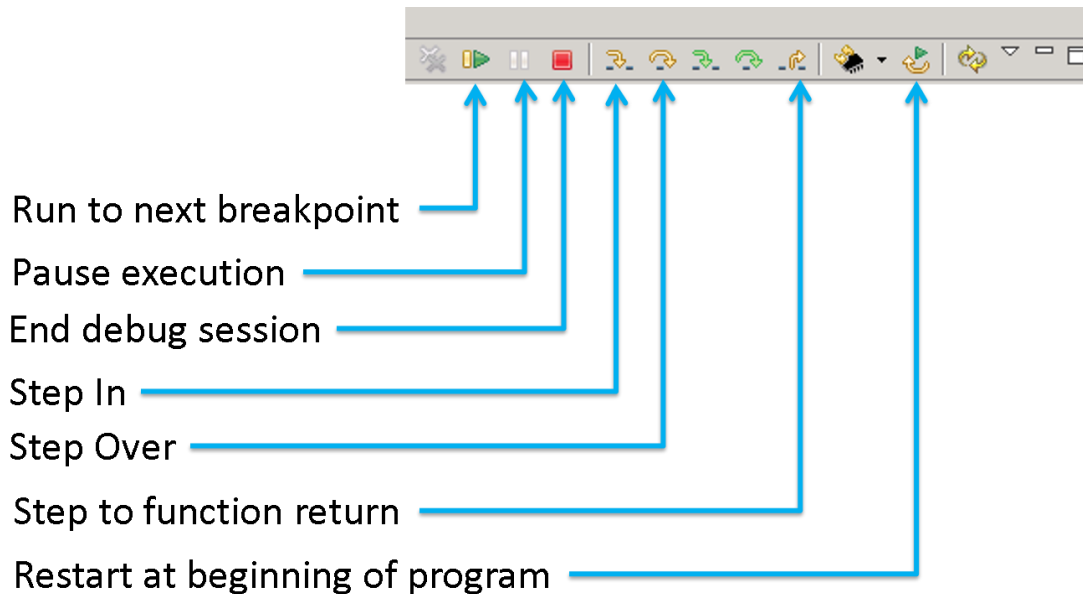


Figure 6: Debugger controls

The debugger controls allow you to control how your program executes. For the remainder of this portion of the lab, you will try out the step controls to see how they work. The key features of the step controls, are defined below:

- **Run to next breakpoint** (also simply called “Go”): This control runs your program until it encounters a *breakpoint*, or a marker that indicates that the debugger should stop running the program so you can examine it.
- **Pause execution:** Stops the program at its current position. You can use this button to see what your program is doing at a certain time.
- **Terminate/End debug session:** Stops the debugger and detaches the debugger connection from the MSP430. The MSP430 will continue running your program, but the debugger can no longer control it. Use this button when you are done debugging and want to go back to Edit perspective.
- **Step In/Over/Return:** Runs a single line of your program and halts execution again. You can use these buttons to run your program step-by-step to see how it executes. We will use these buttons later in the lab.
- **Restart:** This control resets the program counter to the beginning of the program, allowing you to start it again as if you had just started debugging. Note that this button does **not** reload any changes you have made to your program since you started debugging, it just starts the program again from the beginning.

15. Now that you are familiar with the step controls, hit the Pause button and restart your program at the beginning. Use the **Step Over** button to step through the setup portion of the code, then step through the while loop a few times. You should see the LED change state for each iteration of the loop.

16. Often, it is tedious to manually step through your all of code to a point you want to observe. As an alternative, you can set a **breakpoint**, which denotes a point in the code where the debugger should pause the program. To set a breakpoint on a specific line of code, double-click the blue margin next to the line numbers—you should see a blue circle appear where you clicked, as in Figure 7. To test this, set a breakpoint on the call to the `swDelay` function. Then, click the **Run** button a few times—each time, your program will run for a moment and pause when the line is reached again in the next iteration of the loop. Like in the last example, you should see the LED change state with each iteration.

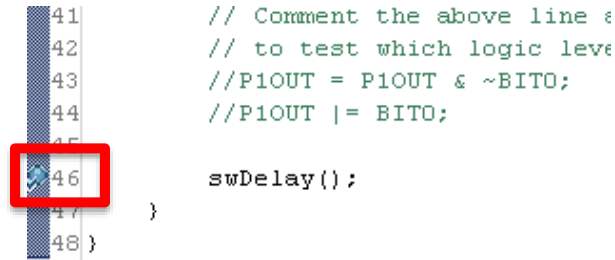


Figure 7: An example breakpoint

We can examine how the LED changes state in more detail. As we will discuss later in class, the LED is connected to a *Digital I/O* port, which enables us to set a specific pin on the microcontroller to a logic 0 or logic 1. On the MSP430F5529 Launchpad, the Red LED is connected to Digital I/O port 1, pin 0 (P1.0), which itself is wired to a physical pin on the chip and then connected to the LED. The LED will turn on or off depending on whether a logic 1 or logic 0 is present on that pin.

In the code, the output value of P1.0 is controlled by bit 0 of the register P1OUT; setting this bit to 0 will output a logic 0 on the pin, while setting it to a 1 will output a logic 1. By changing the value of this single bit in P1OUT, `blink.c` can turn the LED on and off.

17. We can observe the output register P1OUT changing state in the debugger. In the top-right corner of the Debug perspective (see Figure 5), we can view the state of variables and registers in our program. In `blink.c`, there are no variables, so the variables pane is pretty boring. Instead, switch to the **Registers** tab and find the register P1OUT listed under the header “Port1_2”, as shown in X. You can see the value of the entire register next to P1OUT, or expand it to view the values of each bit (eg. P1OUT5 shows the value of bit 5 of P1OUT).

If your breakpoint from the previous step is still set, click the **Go** button again to let the program run through another iteration of the loop. When the program stops, you should see the value of P1OUT change and the state of the LED change accordingly.

Name	Value	Description
Port_1_2		
P1IN	0xBE	Port 1 Input [Memory Mapped]
P1OUT	0x01	Port 1 Output [Memory Mapped]
P1OUT7	0	P1OUT7
P1OUT6	0	P1OUT6
P1OUT5	0	P1OUT5
P1OUT4	0	P1OUT4
P1OUT3	0	P1OUT3
P1OUT2	0	P1OUT2
P1OUT1	0	P1OUT1
P1OUT0	1	P1OUT0
P1DIR	0x00	Port 1 Direction [Memory Mapped]

**Figure 8: Viewing register P1OUT
(Here P1.0 has a value of 1)**

18. We will now examine how the code actually changes the value of the LED. In blink.c, P1.0 is toggled by XOR'ing (^) the current value of the output register P1OUT with the constant BIT0, as shown in the line below:

```
P1OUT = P1OUT ^ BIT0;
```

The name BIT0 is a constant defined in msp430.h. To see its value, hold down the **Ctrl** key and click on the name BIT0 in blink.c in CCS. This should open up the relevant portion of msp430.h to show you that BIT0 is defined as 0x0001 (you can say no to any pop-up messages about scalability features). Congratulations, you just used CCS to find the definition of a variable—this is one of its most useful features!

Back to the point: we now know that this line of code toggles the LED by XOR'ing the current value of the output register with 0x0001. **How does this toggle the state of the LED? Explain how this works in your report.**

19. Which logic level (0 or 1) is responsible for turning on the LED? You may be able to see the answer from examining the register values, but we can also find out by modifying the code. In blink.c, there are two commented lines of code below the XOR statement, one which always sets P1.0 to a logic 1, and one which always sets it to a logic 0. **How do each of these lines work? Explain how they always set the LED to a constant value in your report.**

Comment out the XOR statement, uncomment one of these, and rebuild your program by clicking the Debug button again. **Which logic level turns on the LED? Document this in your report.**

(continued on the next page)

Importing and Building the Demo Project

We are now ready to build the demo project, which is a much larger project that contains libraries for interacting with all of the peripherals on our development board. Rather than making you set up this project yourself, the course website provides an archive of the project that you just need to import into CCS. In the remainder of the lab, you will import this project and play around with the debugger to answer some questions about it.

Importing the Demo Project

20. Download the **Lab 0 template** from the course website (ece2049e17_demo_template.zip). Back in CCS, if you still have a debug session running, return to Edit perspective by clicking the **Stop** button in the step controls. Then, in the **Project** menu, select **Import Existing CCS Eclipse Project**, which will open the dialog shown in Figure 9.

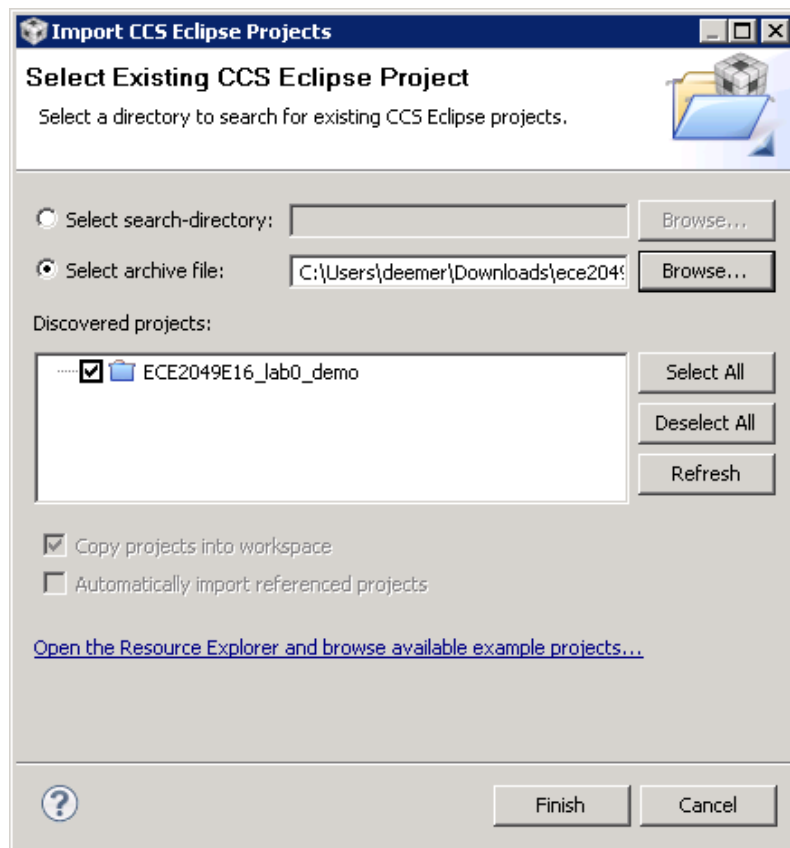


Figure 9: Importing the demo project

21. As shown in Figure 9, choose “Select archive file” and browse to the zip file for the demo project you just downloaded. CCS should find the project in the archive file and display it in the pane. Click **Finish** to import the project.
If the import process fails at the last second with an error about the project metadata, repeat the process and it should succeed this time. If you have issues importing the project, please ask the course staff for help.

22. Once the demo project has been imported, it should appear in your Projects Explorer. Click on the demo project to make it the active project, and then build it. You should see some warnings about unused variables—these are safe to ignore for now. Enter the debugger and run the project. You should see some text display on the LCD, and you can play with the buttons to make things happen. **Demonstrate your working demo project to the course staff for signoff.**

Exploring the Demo Project

23. When you are done playing with the demo project, stop the debugger and examine main.c in the editor window. You will notice that this project is quite a bit more complex than blink.c—this is because it includes libraries required to use the LCD screen, and some library functions that utilize the buttons and LEDs. Since all of our labs will use the LCD in some way, we will start our labs by importing a variant of this project. To implement the labs, you will use the graphics library functions to use the LCD and expand on some of the other library functions to add new capabilities.
24. Starting with example code and modifying it to perform desired tasks is a typical strategy in embedded systems. Experiment with a modify main.c in the demo project to do at least the following tasks:
1. Play with the LCD write commands (GrStringDrawCentered) to move the text to different positions on the screen. **Explain how to control the position of the text on the screen in your report.**
 2. Use the Ctrl+Click method described in step 17 of the lab to find the definition of any function in main(). This is a very useful feature of CCS—you will use it a lot in your labs!
 3. Declare an array of type char and initialize it to contain your name—use it to write your name to the LCD.
 4. Step through the code until you get to the assignment for variables flt, X, and tst. **Answer the questions asked in the code in your report using the Variables window in CCS.** Note that you can change the way CCS displays a variable in the variables window by right-clicking on it and selecting “Number Format”, as shown in Figure 10.

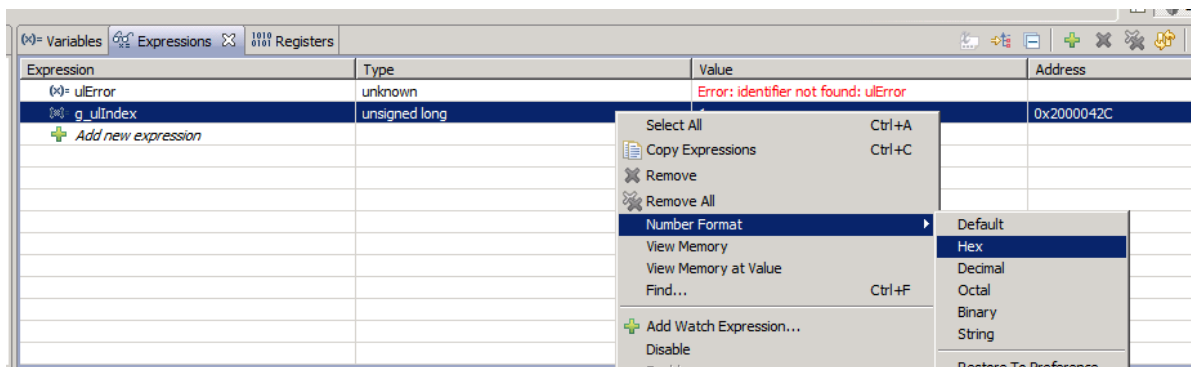


Figure 10: Changing number format in the variables window

(continued on the next page)

ECE2049: Lab 0

For reference, the questions asked in the code regarding these variables are as follows:

```
// What size does the Code Composer MSP430 Compiler use for the
// following variable types? A float, an int, a long integer and a char?

// What value stored in myGrade (i.e. what's the ASCII code for "A")?
// What is the new value of tst? Explain?
```

5. When does the buzzer sound? More importantly, what turns it off?
6. What is the relationship between the 4 buttons and the 4 colored LEDs?
7. If your board does not have a keypad, ask the course staff for one and put it in the P6 header on the board. You may need to hold it in place to get good contact.
What does the keypad do?

Once you have modified the demo project to display your name and thought about these questions (you don't need to have them completely answered yet!), ask the course staff for a signoff. If you have any questions about the lab, feel free to ask the course staff before you write your report.

25. You're done! Review the guidelines below for how to write your report and submit your code.

(continued on the next page)

Writing your Report

Since this lab was mainly a tutorial, the report does not need to be substantial—however, we are asking you to write one as practice for future reports. Your lab should be written in a professional style. It should be an electronically-prepared technical document like what you would submit to a fellow engineer or your boss.

Only one report is required per lab team. The report should include:

- **Introduction** (1-2 paragraphs max): Succinctly state the objectives of the lab and give an overview of what you accomplished.
- **Discussion and results**: Discuss what you did in each part of the lab and how you solved any problems. For this lab, you can briefly describe what you did and **be sure to thoroughly answer and explain the questions asked in the lab assignment**. In general, this section should be as long as necessary to say what you need—no padding or fluff!
- **Summary and Conclusion** (1-2 paragraphs max): Summarize what you accomplished in the lab and what you learned. This should be a “bookend” to the introduction.
- **Appendices**: You should not need any in this lab. **DO NOT PASTE YOUR CODE INTO THE END OF THE LAB REPORT!** Instead, your code will be submitted as an archive file alongside your report, which is a lot cleaner!

Lab reports are important. In industry, the FIRST view of YOUR work by *anybody* other than your immediate supervisor will see will probably be in WRITING!

Learning to be an effective communicator of technical information is probably THE MOST IMPORTANT job skill you can have.

Thus, we care about lab reports. We read them. Really.

Submitting your Work

When you are done with your report, you will submit it and your code on Canvas for grading. In order to receive a grade, you must submit **both** your code and your report online—even though you did not write much code for this lab, we will start the submission process now. Only one member of your team needs to submit files for your lab.

In addition, you must turn in your signoff sheet to the course staff—usually, you will do this when receiving your last signoff. If not, you can turn it in by placing it in the box in the ECE office, or handing it to a member of the course staff.

To submit your code for grading, you will need to create a zip file of your CCS project so that the course staff can build it. You can also use this method to create a complete backup copy of your project (perhaps to send to your partner or save for later). To do this:

1. Right click on your project and select "**Rename...**"
2. If you are submitting your project, enter a name in the following format: **ece2049e17_lab0_username1_username2**, where username1 and username2 are the user names of you and your partner. (**NOTE**: Failure to follow this step will result in points deducted from your lab grade! If you don't do it, it makes a **lot** of extra work for the graders!)
3. Click **OK** and wait for CCS to rename your project.

4. Right click on your project again and select "**Export...**" then select "General" and "Archive file" from the list and click **Next**.
5. In the next window, you should see the project you want to export selected in the left pane and all of the files in your project selected in the right pane. Select all. You should not need to change which files are selected.
6. Click the "**Browse**" button, find a location to save the archive (like your R drive) and type in a file name using the EXACT SAME NAME used in Step (2).
7. Click "Finish". CCS should now create a zip file in the directory you specified.
8. Go to the Assignments page on the class Canvas website. Click on the assignment for Lab 0 and attach the archive file of your project that you just created and your report. When you are ready, hit the Submit button. Only one code and report submission is required per team.