# Let's Get Started!

This handout will walk you through the steps to create your first program on the Texas Instruments MSP430FR6989 Microcontroller Launchpad.

Note, we go through a lot of details in these lab manuals, and at times, some students have thought we included too many steps. However, it is our intent to err on the side of caution and provide as much support for our students as possible. Thanks for understanding.

1.   Open the Launchpad development kit box and the plastic bag that contains the Launchpad board.
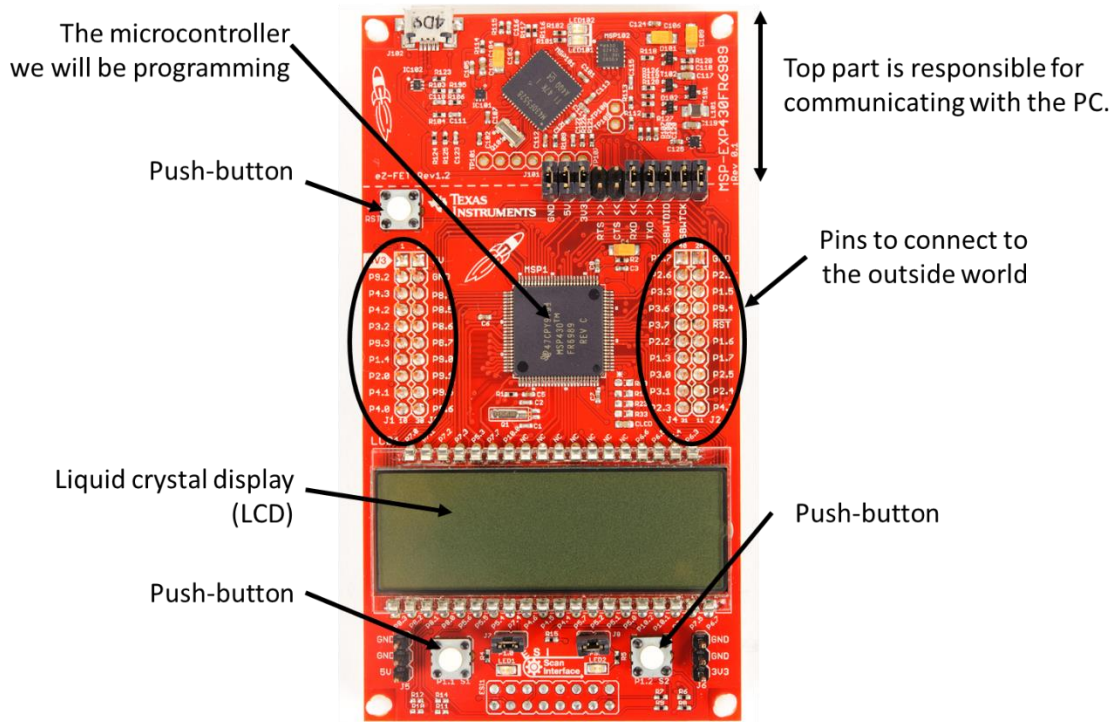

2.   Connect the Launchpad development board to the PC with the enclosed USB cable.
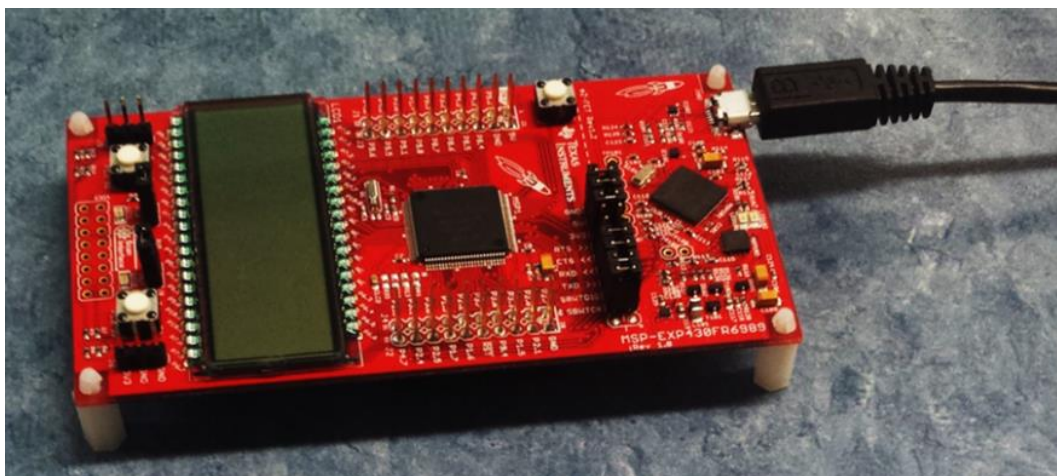
3. The board is divided into two parts. The top part (the side with the USB connector) is responsible for communicating with the PC make getting your program onto our MSP430FR6989 microcontroller.

The bottom part contains the MSP430FR6989 that we will be using in this class. In addition, the bottom part has the LCD screen, push-buttons, and a number of metal pins that can be used to connect the microcontroller to the outside world. Each of these will be discussed in more detail later in the course.

The microcontroller we will be programming

Push-button

Liquid crystal display (LCD)

Push-button

Top part is responsible for communicating with the PC.

Pins to connect to the outside world

Push-button

4. The power LED (**LED102**) should be on after you connect the board to the PC. The bottom two-thirds are for the microcontroller, two push-buttons, a couple LEDs, and holes for connecting your board to the rest of the world. Note, there might be slight differences in the printing on the board. As the board goes through revisions, the printing is occasionally changed.

Note, the font type for **LED102**. Periodically, you will see words and phrases using a similar format throughout our documentation. This will be your sign that this is either an important reference from the Texas Instruments documentation or hardware / software tools.

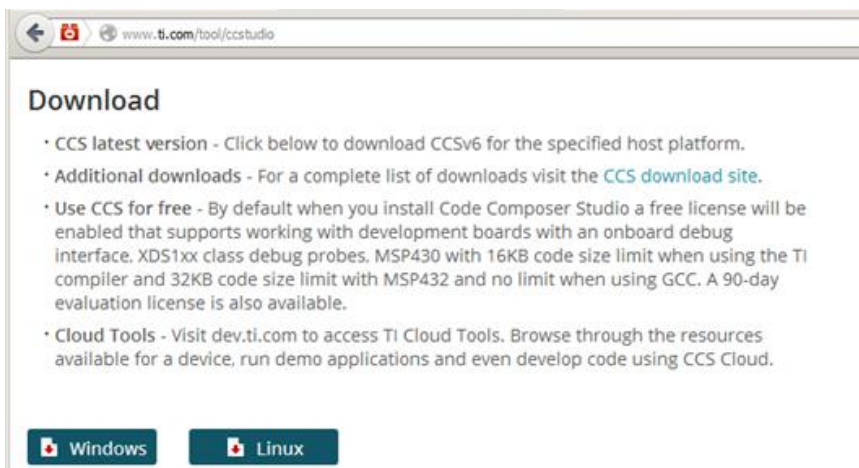**LED102** should turn on after you connect the board to the PC.
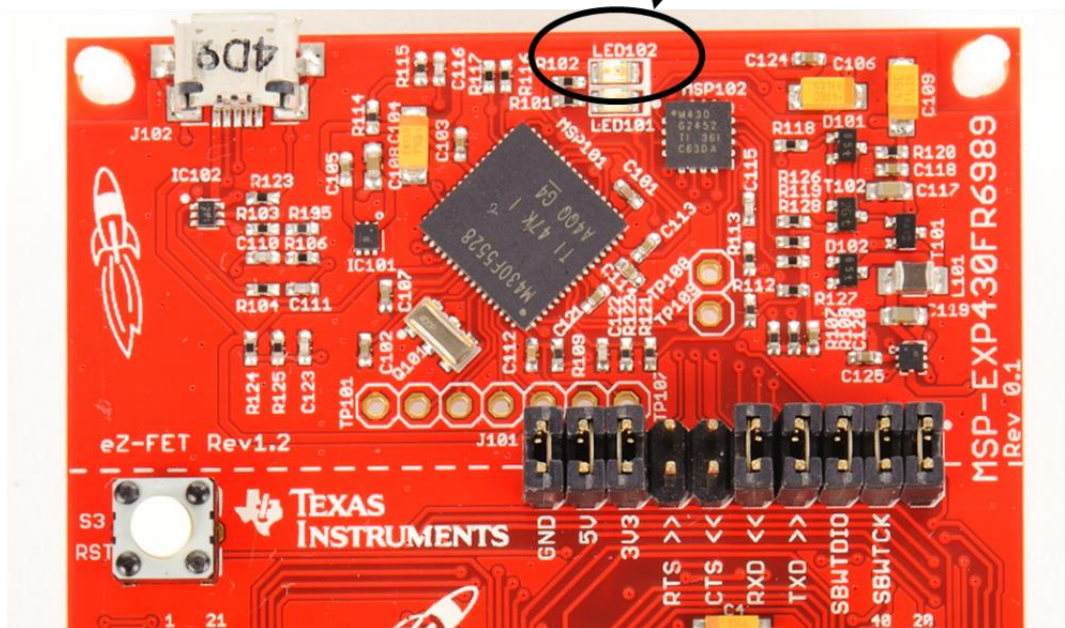
5. If you have not done so already, follow this link to download the **Code Composer Studio** (**CCS**) program.

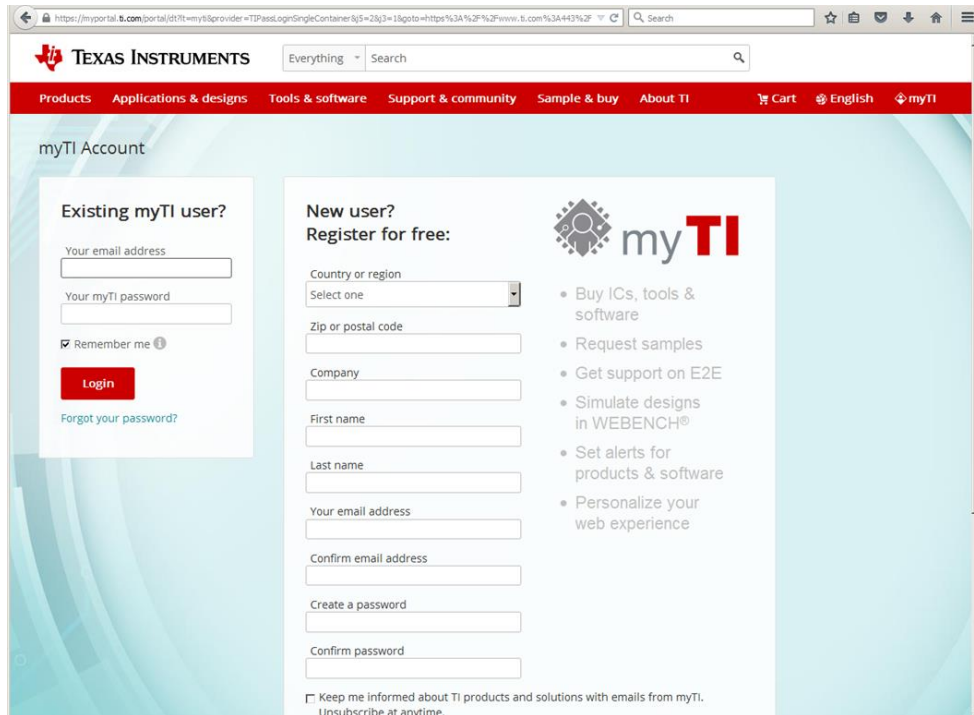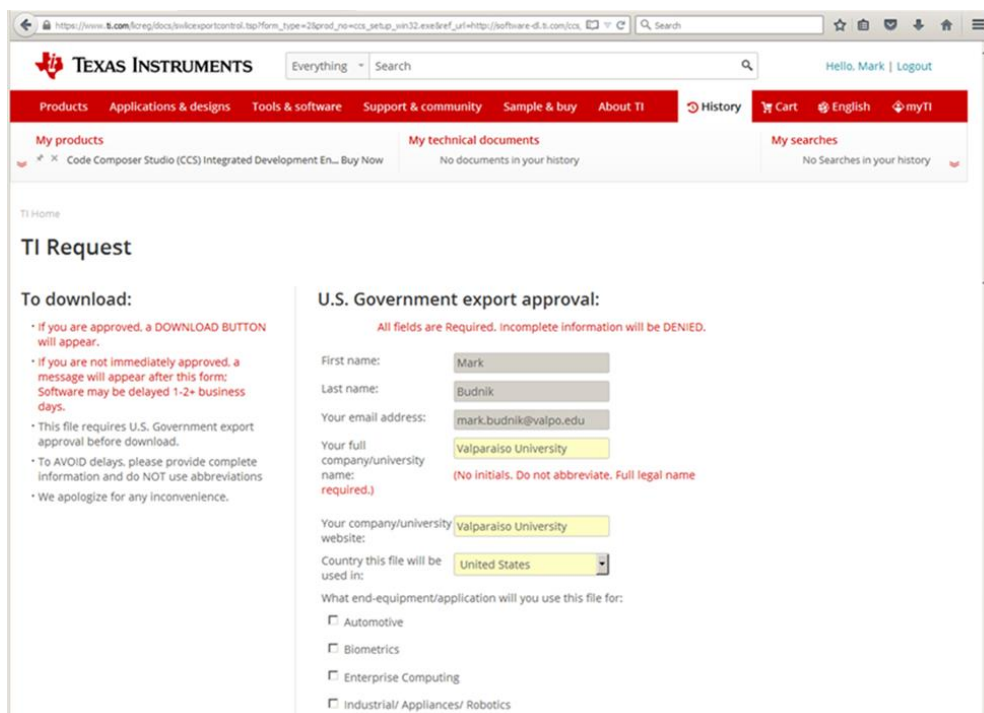   http://www.ti.com/tool/ccstudio
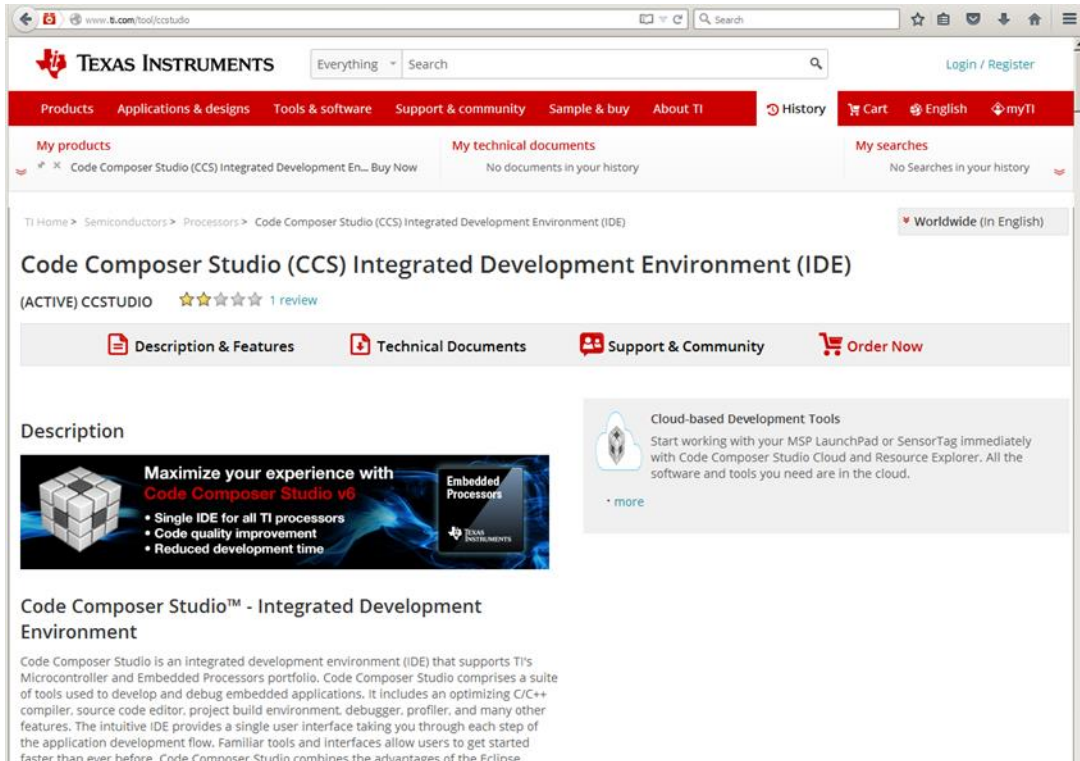
6. Scroll down the page a little and you will see a link to download **CCS**. We will be using the Windows version of the program. Go ahead and click the **Windows** button.

7. At this time, you will need to either create your Texas Instruments account or, if you have an account, enter your **email address** and **myTI** password.
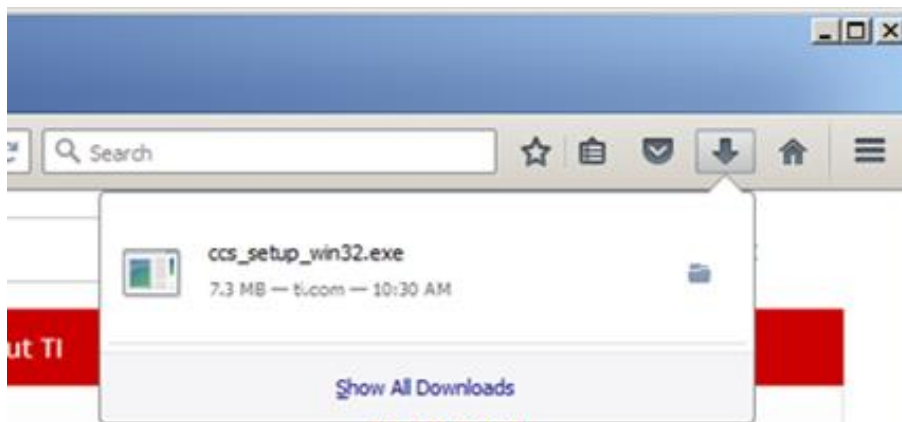

8. After logging in, you will need to complete the U.S. Government export approval form. Enter your information, and click **Submit** at the bottom of the form.
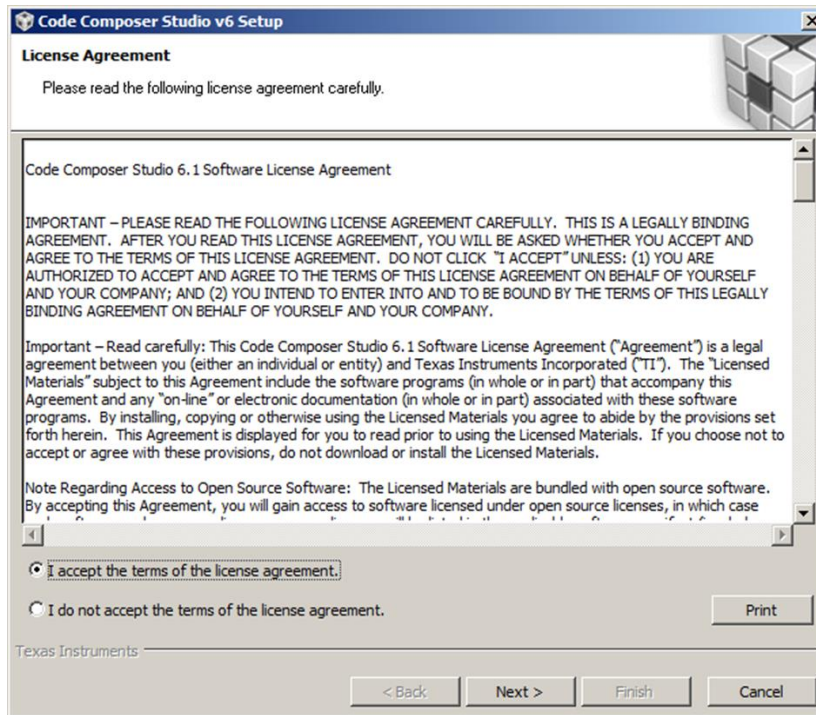
9. After a few moments, you will be taken to the download page. Click **Download** to start the process.
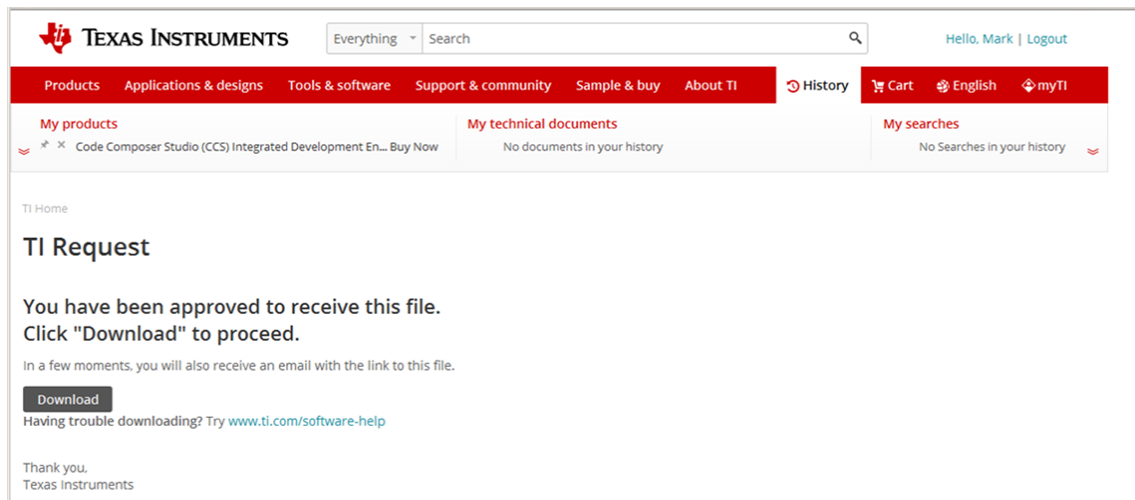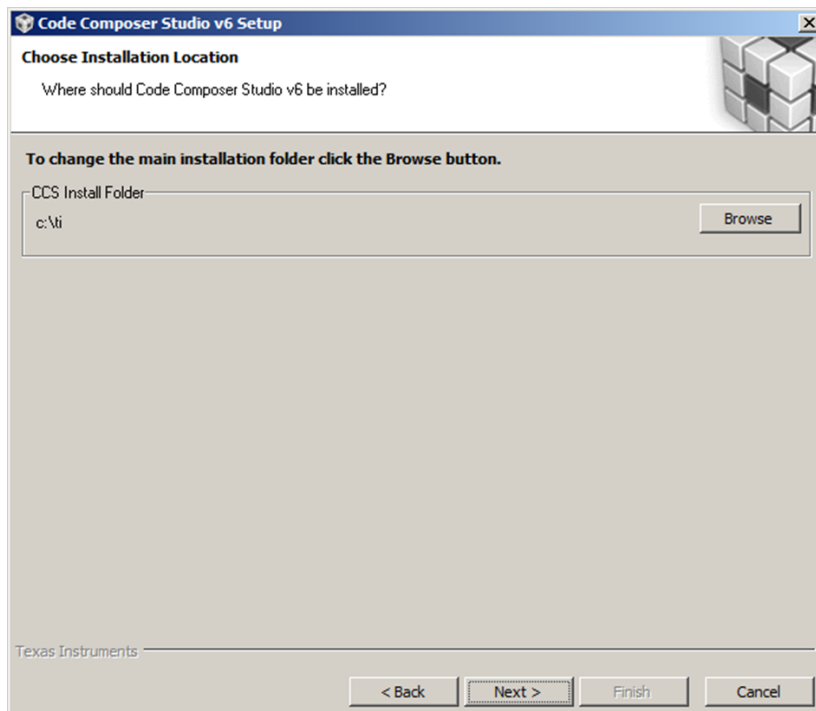

10. When complete, you will have a 7.3MB file entitled **ccs_setup_win32.exe**. Go ahead and open the file.

11.	Next, you will need to accept the terms of the license agreement and click **Next**.

12.	You will be asked to specify the location for installation. **Browse** to your preferred location or accept the default and click **Next**.

13. You will then be asked to select which microcontrollers and microprocessors you would like to use with **CCS**. For now, just select the **MSP Ultra Lower Power MCUs** as shown below and click **Next**.
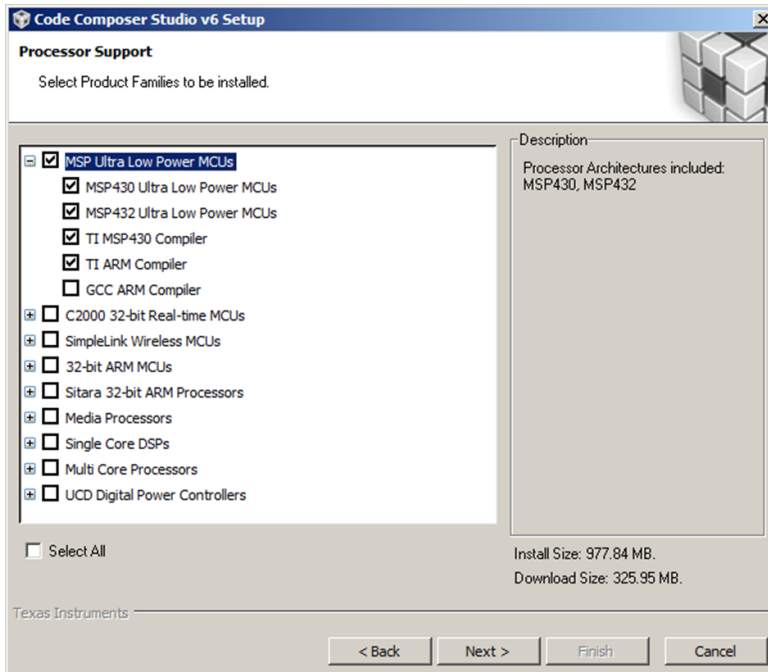

14. The next window asks you to **Select Debug Probes**. Select the options shown below and click **Next**.
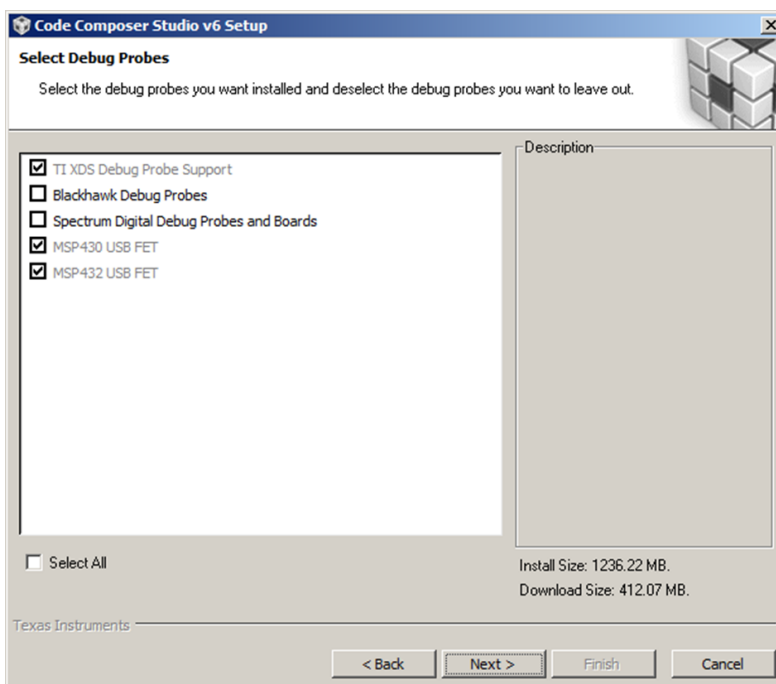
15.   Ok, we are almost done. The last window asks if you want to install any Texas Instrument apps for **CCS**. Do not select any of these options and click **Finish** to begin the installation.
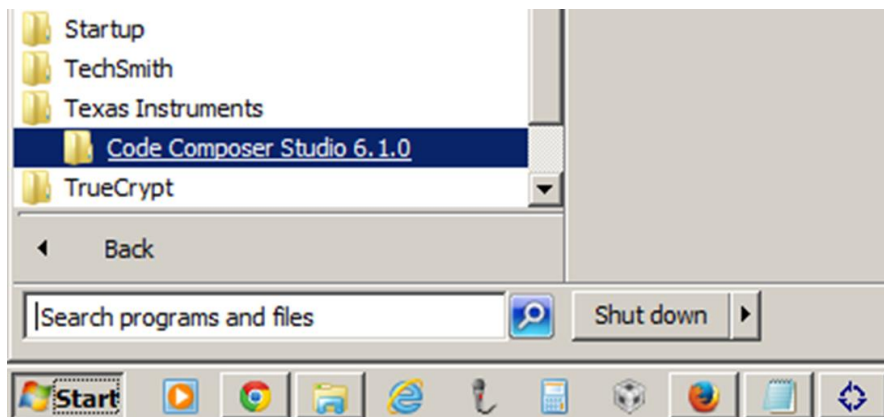

16.   Windows will show you the progress of the download. Note, the download and installation may take a couple minutes.
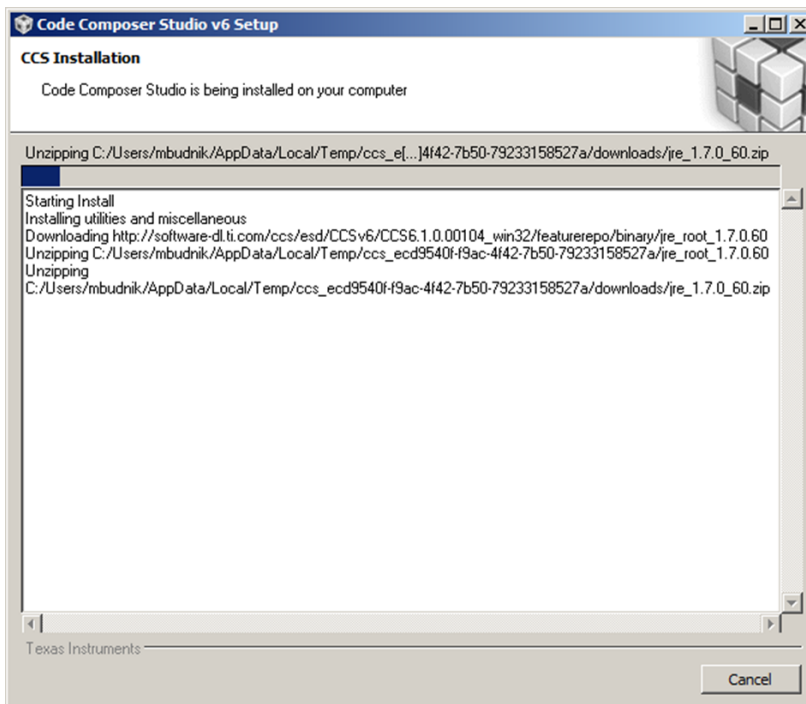
17.   When the installation is complete, you will have the option of immediately launching **Code Composer Studio**.
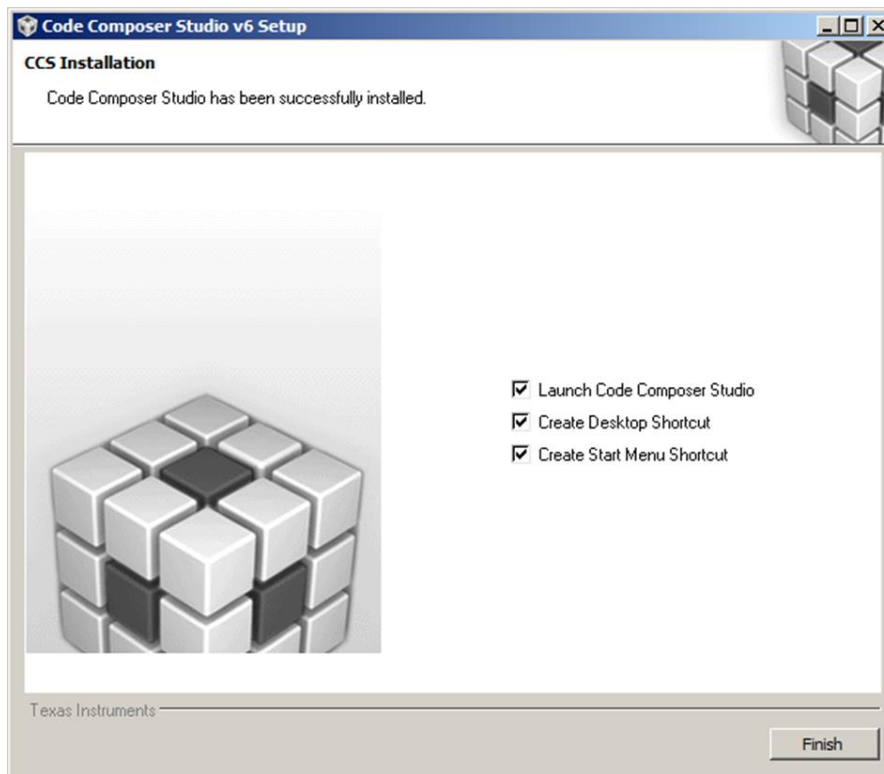
18.   In the future, you can open **CCS** from the start menu.

19.   Note, there is also an off-line installer option for `Code Composer Studio`.

http://processors.wiki.ti.com/index.php/Download_CCS

20.   When **CCS** opens, you will first be greeted by a splash screen.

21. You will then be asked to select a **Workspace**. Click **Browse** to specify a folder where you
     would like to store your **CCS** projects.

     Note, make sure you do NOT check the **Use this as the default…** option.

     When you are ready, click **OK** to continue.
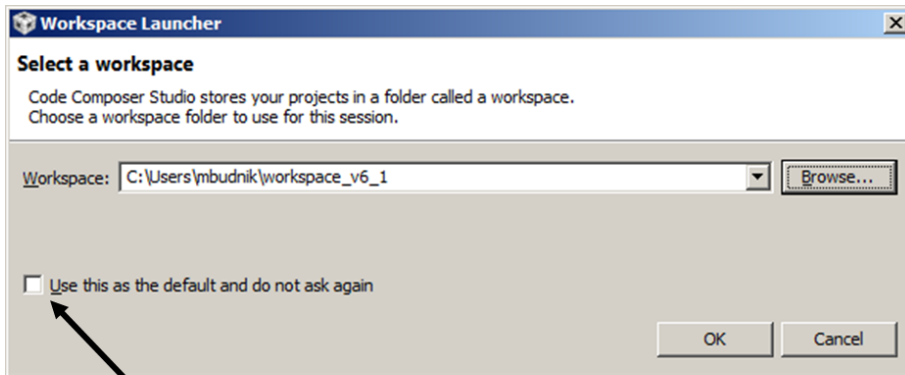
Do NOT check

22. When opened, the **CCS** program may have a **Getting Started** window open like this. If it
     does, go ahead and close the tab.
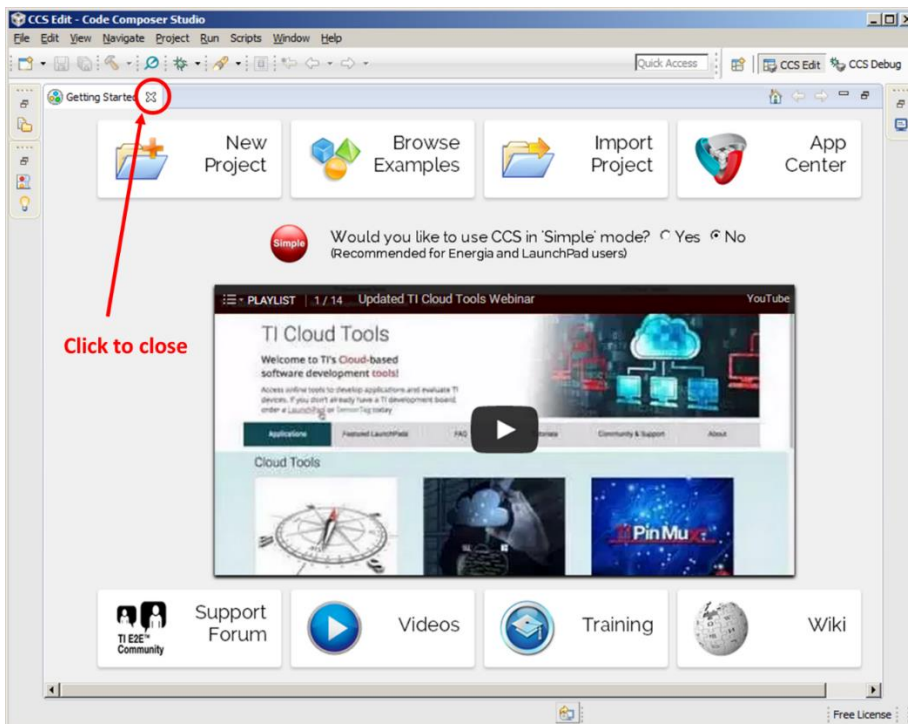
23.   When the **Getting Started** tab is closed, **CCS** will look like this. This is the default
       configuration that you will be using for almost all of your code creation.
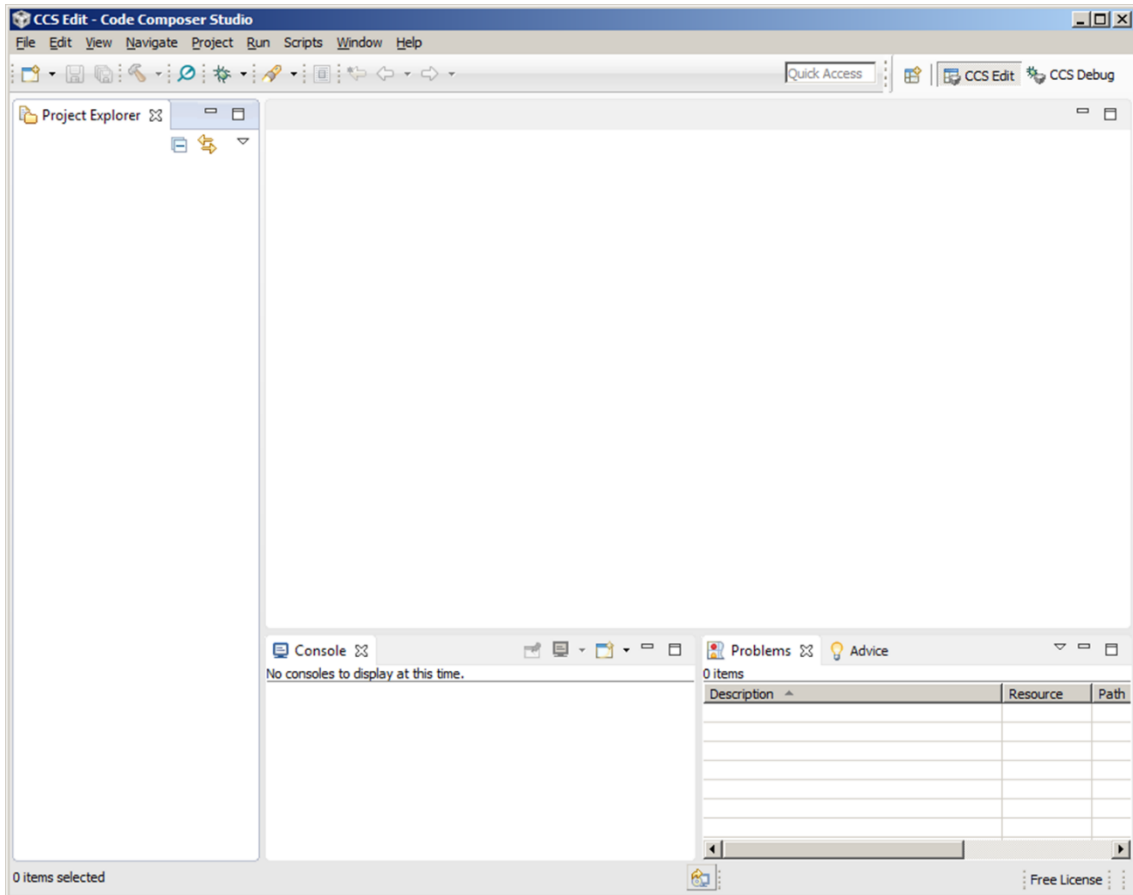
24.   Under **File**, select **New**, then select **CCS Project.**

25.   This will open the **New CCS Project** window. It is REALLY important to make sure you get the next several steps right, so please be careful.

For **Target**, select **MSP430FRxxx Family** from the pull-down menu.

a)   In the box to the right of **Target**, select **MSP430FR6989** from the pull-down menu. (Note, it is very close to the bottom of the list. This is the microcontroller in the **MSP430FRxxx Family** that we will be using in this class.

b)   For Connection, you should accept the default, **TI MSP430 USB1 [Default]**.

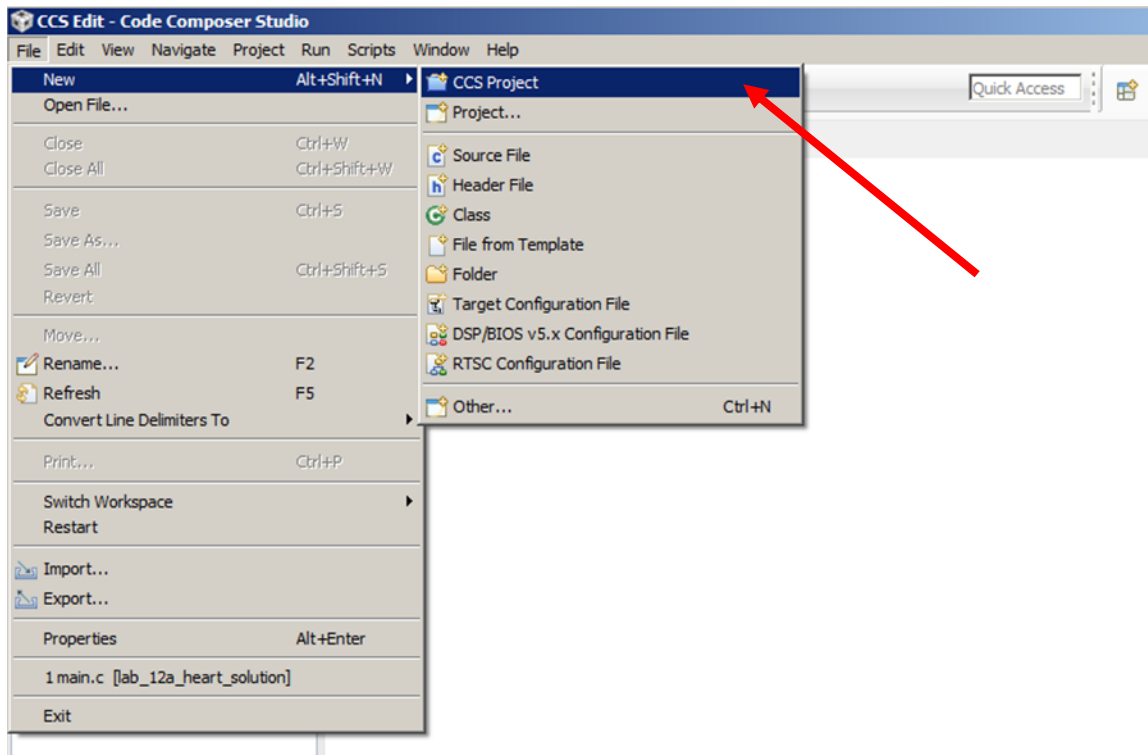c)   For **Project name**, enter **Blink**.

d)   Under **Project name**, check the **Use Default Location** box.

e)   Use the default **Compiler version**.

f)   Finally, toward the bottom of the window, under **Project Templates** and examples, select the **Empty Project (with main.c)**.

After double-checking everything, click **Finish** when you are ready to go on.
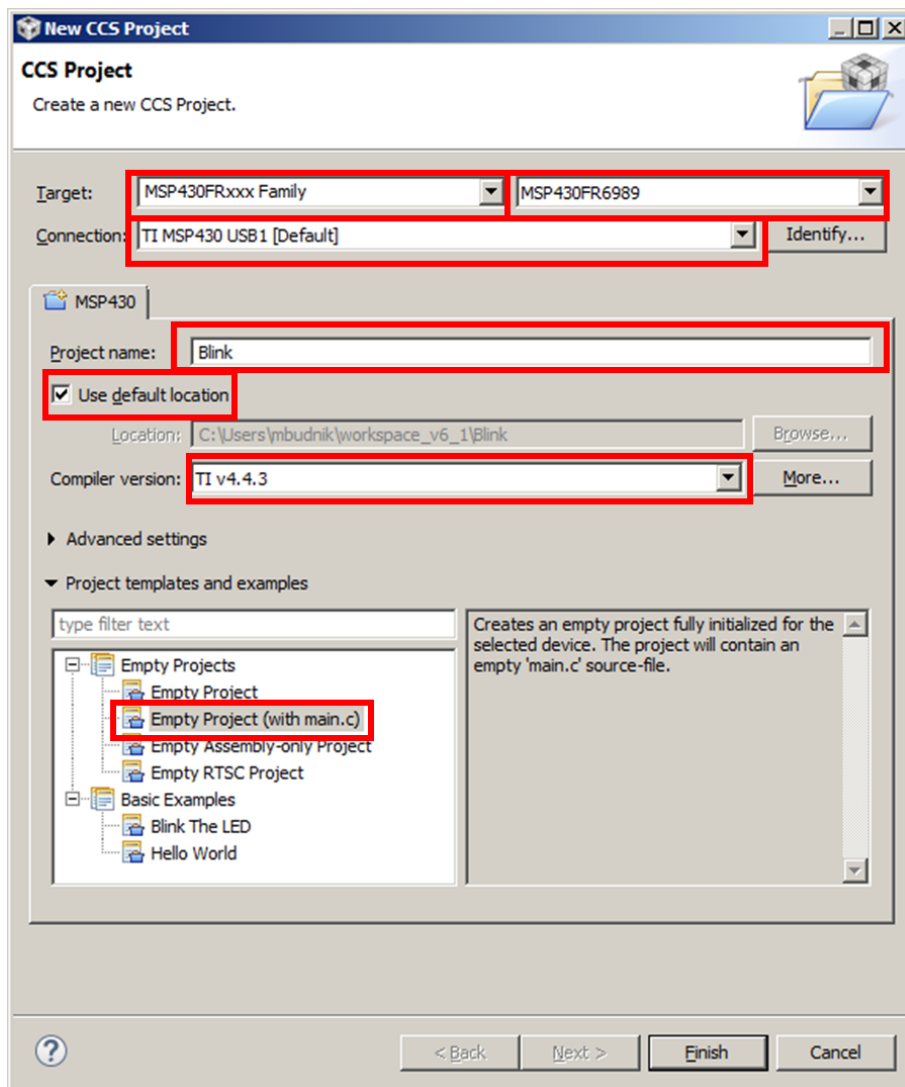
26.   After a few seconds, the new project will be added to the **Project Explorer** window and the "empty" **main.c** file is also shown.
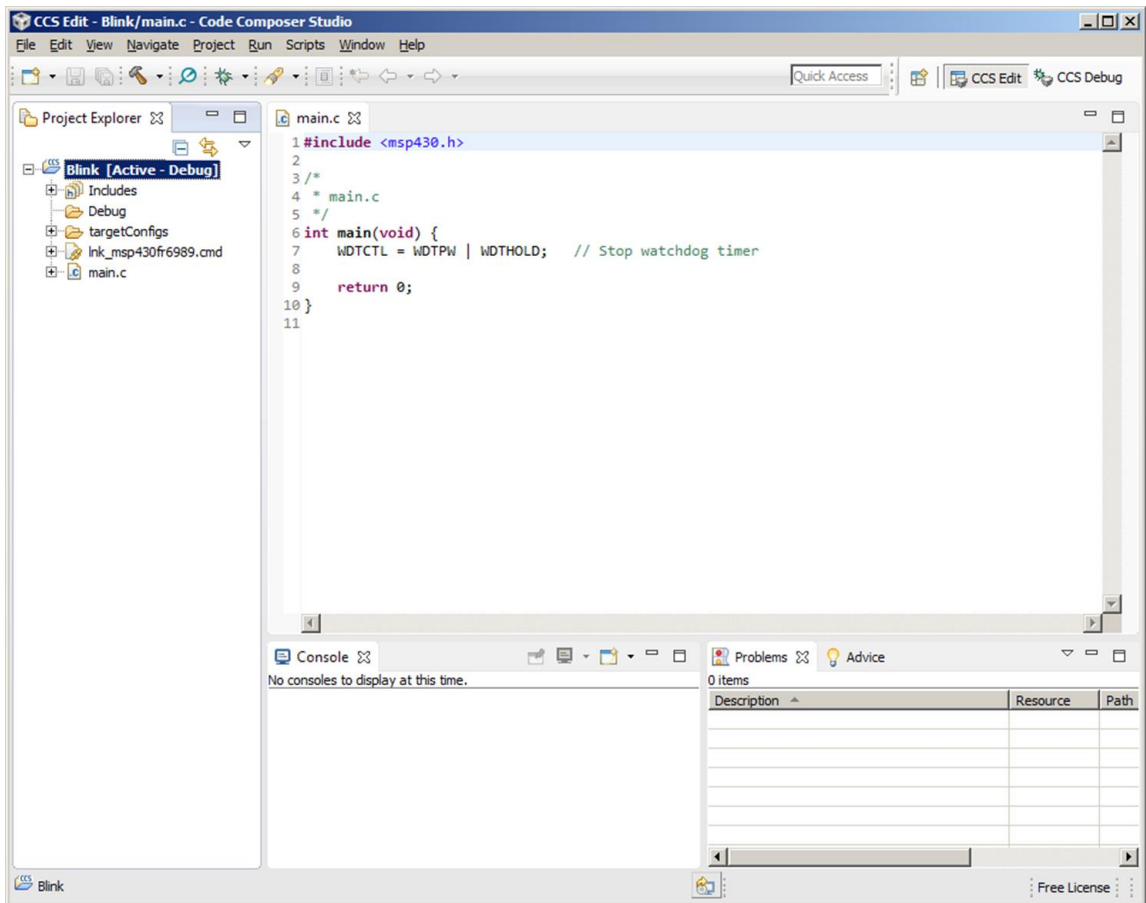
27.   If you the main.c tab does not open automatically, you can open it anytime you want. First, if the **Blink** folder is collapsed like this, click on the small icon in front of its name.
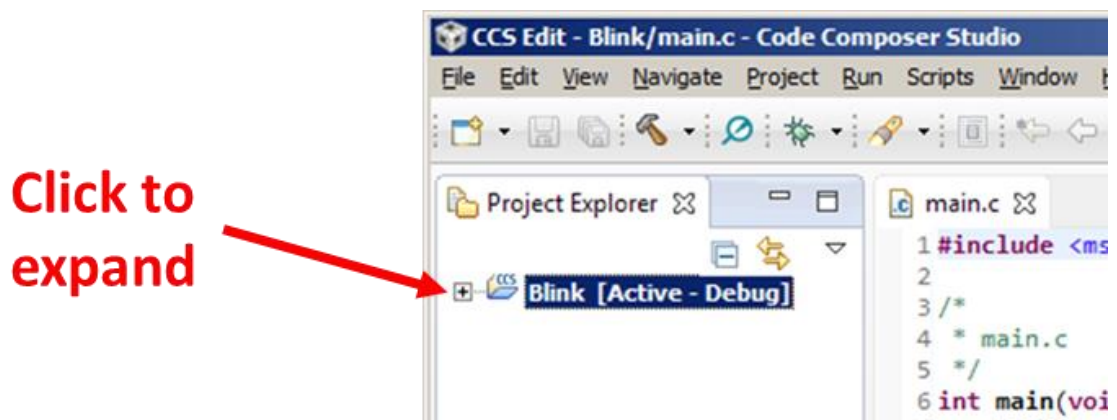
28. This will expand the **Blink** folder to and show its various components. Go ahead and double-click on the **main.c** file to open the tab.
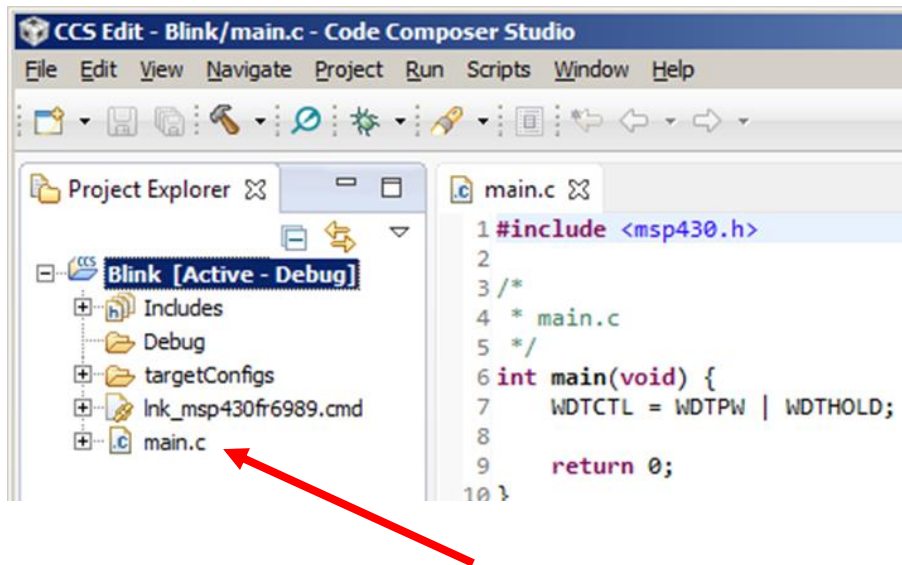
29. Make sure your **CCS** window looks like this before proceeding to the next step. If not, please go back and double-check your work.
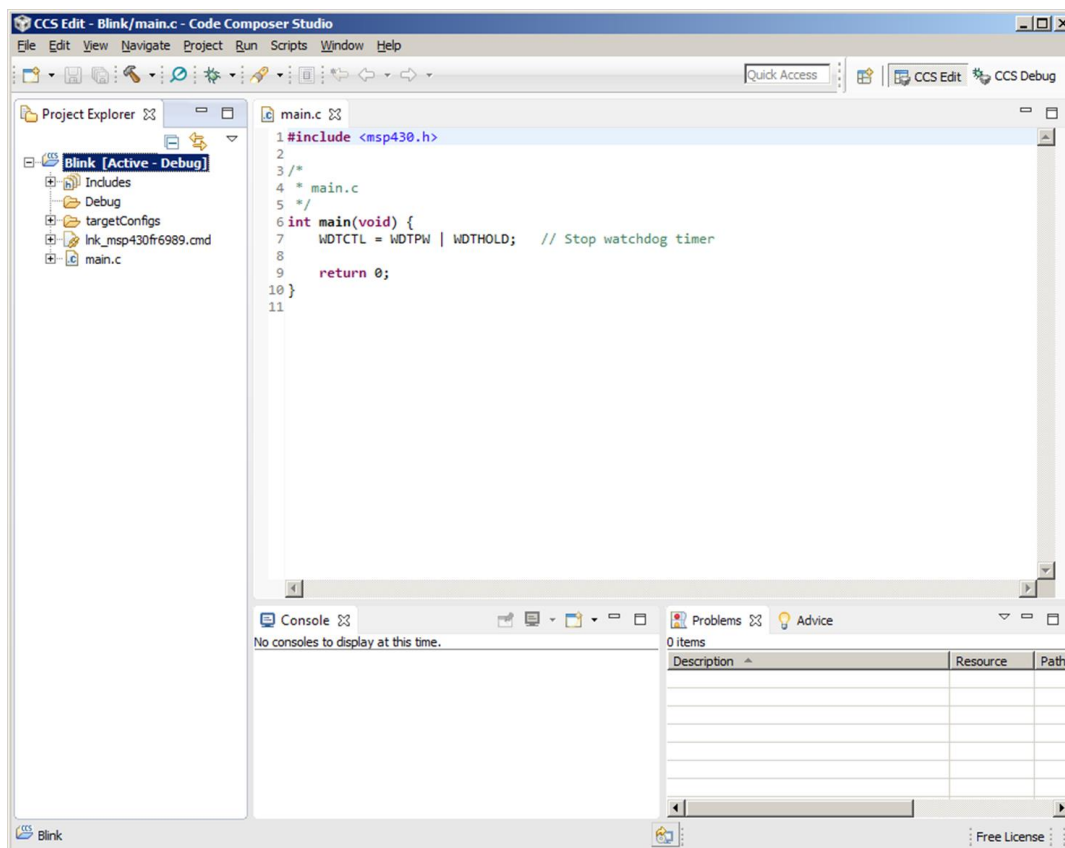
30.   For this handout, the program we will be using is shown below. We will take a look at the various lines of the program in the following steps. These will be covered in a lot more detail later in the course, but for now, our quick explanation should help.

```c
// This program will blink a red LED

#include <msp430.h>                      // Allows us to use "short-cut" names
                                         // to make our code easier to read

#define   RED          0x0001            // Specifies the red LED light for us

#define   RED_OFF      0x00FE            // Used to turn the red LED off

#define   ENABLE_RED   0xFFFE            // Used to enable microcontroller's pins

#define   DEVELOPMENT  0x5A80            // Used to disable some of the security
                                         // features while we are still learning


main()                                   // All C programs have a main function
{
    WDTCTL  = DEVELOPMENT;               // Disables some security features

    PM5CTL0 = ENABLE_RED;                // Enables the pins to the outside world

    P1DIR   = RED;                       // Make a pin an output

    long x  = 0;                         // Will be used to slow down blinking


    while(1)                             // Continuously repeat everything below
    {
        for(x=0 ; x < 30000 ; x=x+1);    // Count from 0 to 30,000 for a delay

        P1OUT = RED;                     // Turn red LED light on

        for(x=0 ; x < 30000 ; x=x+1);    // Count from 0 to 30,000 for a delay

        P1OUT = RED_OFF;                 // Turn off the red LED light
    }

}
```

31. The first line is a comment that briefly describes what your program will do. To designate a comment, use two backslash characters. **CCS** will recognize the comment and color it green to make it easier to tell apart from the rest of your code.

```
// This program will blink a red LED
```

32. The **#include** `<msp430.h>` command is used to tell **CCS** you want to use, or include, another file when your program is ready. The `msp430.h` file contains various names of parts of the microcontroller (that is, registers like **P1DIR** and **P1OUT**) which can be used to make your program more readable.

Lines like this are not part of the actual program and are often called "preprocessor statements". They are used strictly by the user to specify to **CCS** how the program should be used.

33. The next lines are the **#define** statements. These allow you to associate numerical values with names that you can use later inside of your program. Again, this will make your code easier to read.

34. The next line signifies the beginning of your actual program. Every C program has a **main()** function. When you run your program, the microcontroller will first look for **main()** and perform everything contained inside of its curly braces **{ }** .

```
main()
```

35. In the next line, our program puts the microcontroller into a development or "practice" mode by disabling a security system called the watchdog timer. Essentially, the watchdog timer can be used to restart your program if something goes wrong. This is very important in some more critical programs, but not in simple ones like this.

We will learn more about watchdog timers later, but for now, it simplifies our program to disable it.

Notice also that semicolon (**;**) at the end of the instruction. You will quickly learn that the C programming language needs to see these semicolons appropriately or your program will not work.

```
WDTCTL = DEVELOPMENT;
```

36.   The next line of the program is used to enable the microcontroller pins to connect to the outside world. As we go through this course, you may find it amusing to see how many different features of a microcontroller have to be enabled or disabled. However, today's microcontrollers have many, many different features that you could use. By requiring a user to enable the features you want to use, it makes it easier to avoid some mistakes.

For now, just remember that including this line of code in our program allows us to connect the microcontroller to the "outside" world, or in this case, the red LED light.

```
PM5CTL0 = ENABLE_RED;
```

37.   The next instruction tells the microcontroller to use its pin connected to the red LED light as an output that it can turn on and off.

```
P1DIR = RED;
```

38.   The next line creates a variable called **x**. This variable will be used to implement a delay so the red LED light will turn on and off slowly enough that you can see it.

Later in the class, we will introduce the term **long** and explain how similar terms can be used to designate how much memory a variable uses in a microcontroller. For now, just recognize that **x** can hold values up to about 2,000,000,000.

```
long x = 0;
```

39.   The next line of the program starts an execution loop.

```
while(1)
```

Loops will be discussed in more detail in a later section, but for now know that this loop will keep executing the instructions inside of its curly braces **{ }** indefinitely. Without this loop, the program would only turn the red LED light on and off one time.

40.   Next we have our first delay implemented with another loop called a **for** loop.

```
for(x=0 ; x < 30000 ; x=x+1);
```

For loops will also be discussed in more detail in a later section. For our purposes, it creates a delay in our program (set by the 30,000 number). Later in this handout, you can play with increasing/decreasing the length of the delay by changing this number. (But, do not do it yet!)

41.   The next instruction turns on the red LED light.

```
P1OUT = RED;
```

42.   We then implement a second delay with another for loop.
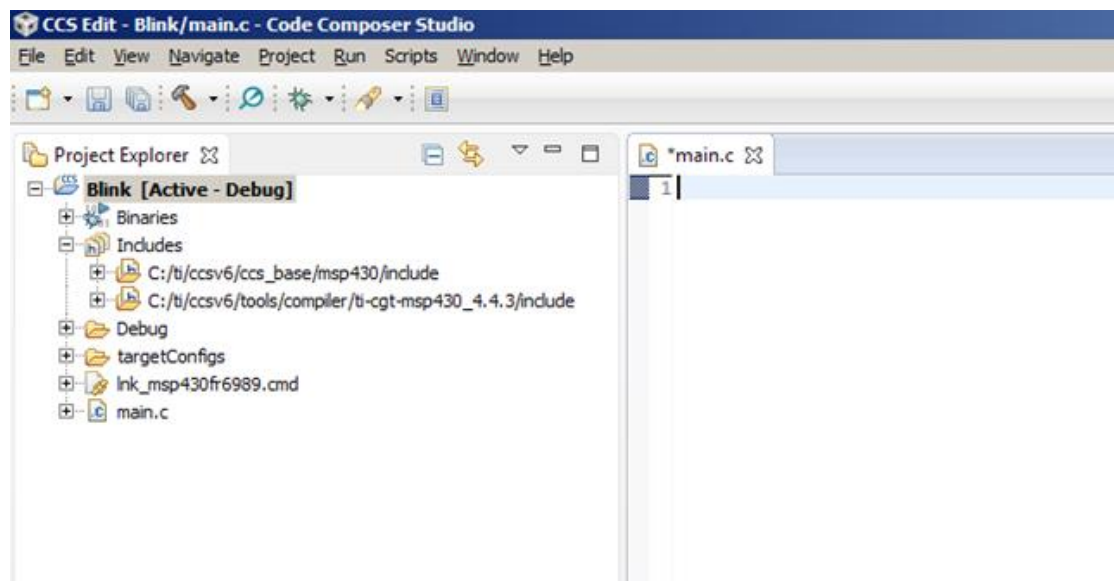
```
for(x=0 ; x < 30000 ; x=x+1);
```

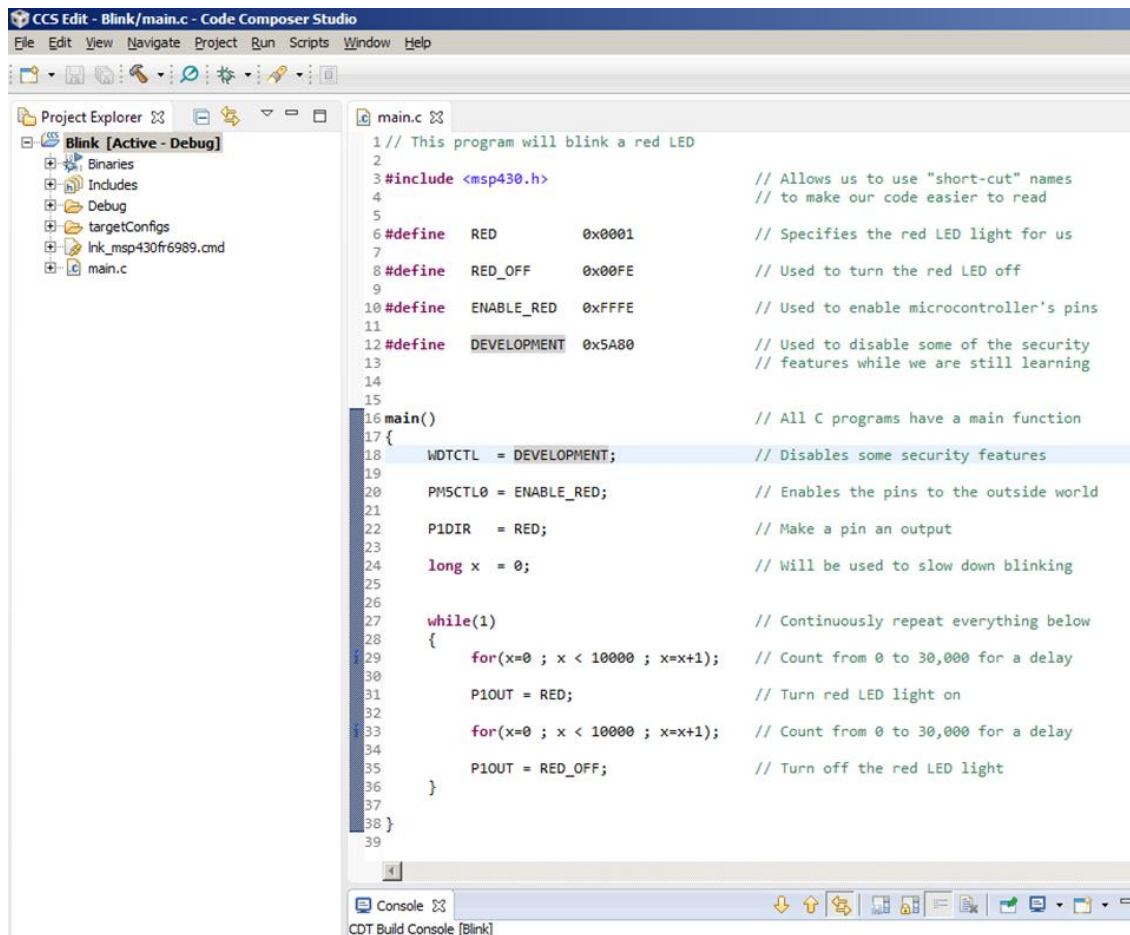43.   After the second delay, we turn the red LED light off.

```
P1OUT = RED_OFF;
```

44.   At this point in the program, the while loop returns us back to the previous delay statement to turn the red LED light on and then off again indefinitely.

45.   Now that we have looked at the program in more detail, let's get it running on your microcontroller. In **CCS**, highlight the program presently displayed in the **main.c** tab. After highlighting it, hit the **[Backspace]** key to delete it. CCS should then look like this:

46. Highlight and **Copy** the original, complete program listing from above and **Paste** it into the **main.c** tab.

47. Look carefully at the **main.c** tab. Do you see the asterisk (*****) in front of its name? This asterisk will appear whenever you have unsaved changes in your program.

48.	To **Save** your changes, you can select **Save** from the **File** menu.
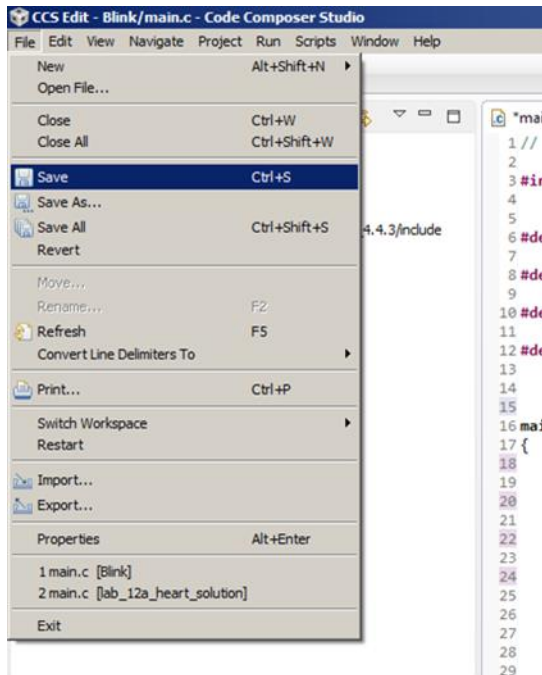
49.	Or, you can simply click one of the **Save** shortcut buttons. The first saves only the file (**main.c**) that is presently opened. The second saves any and all files related to your project (some really big programs may be spread across dozens of files). If you hover your mouse over the **Save** icons, their names will appear.

For most of this class, we will only be using a single **main.c** file, so you can use either short-cut.
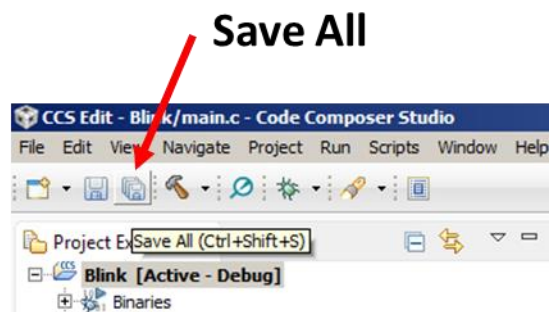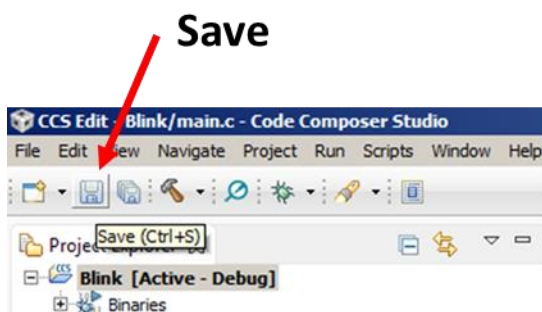
50. After saving your program, you need to **Build** your project by clicking the hammer icon on the top of your screen. (Get it? You can build things with a hammer?)
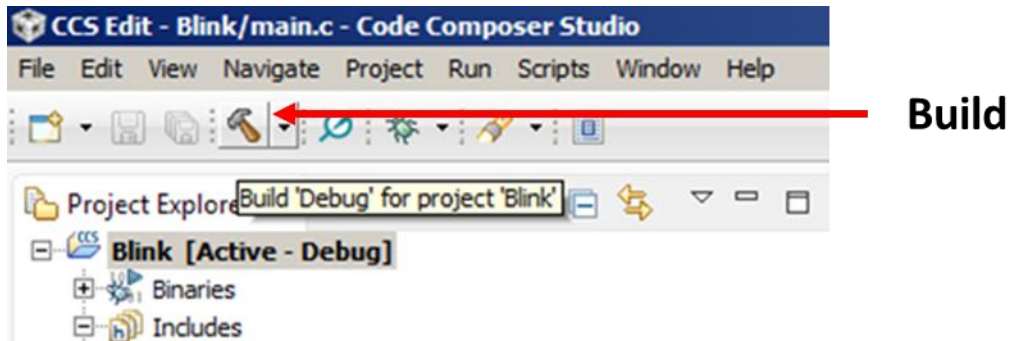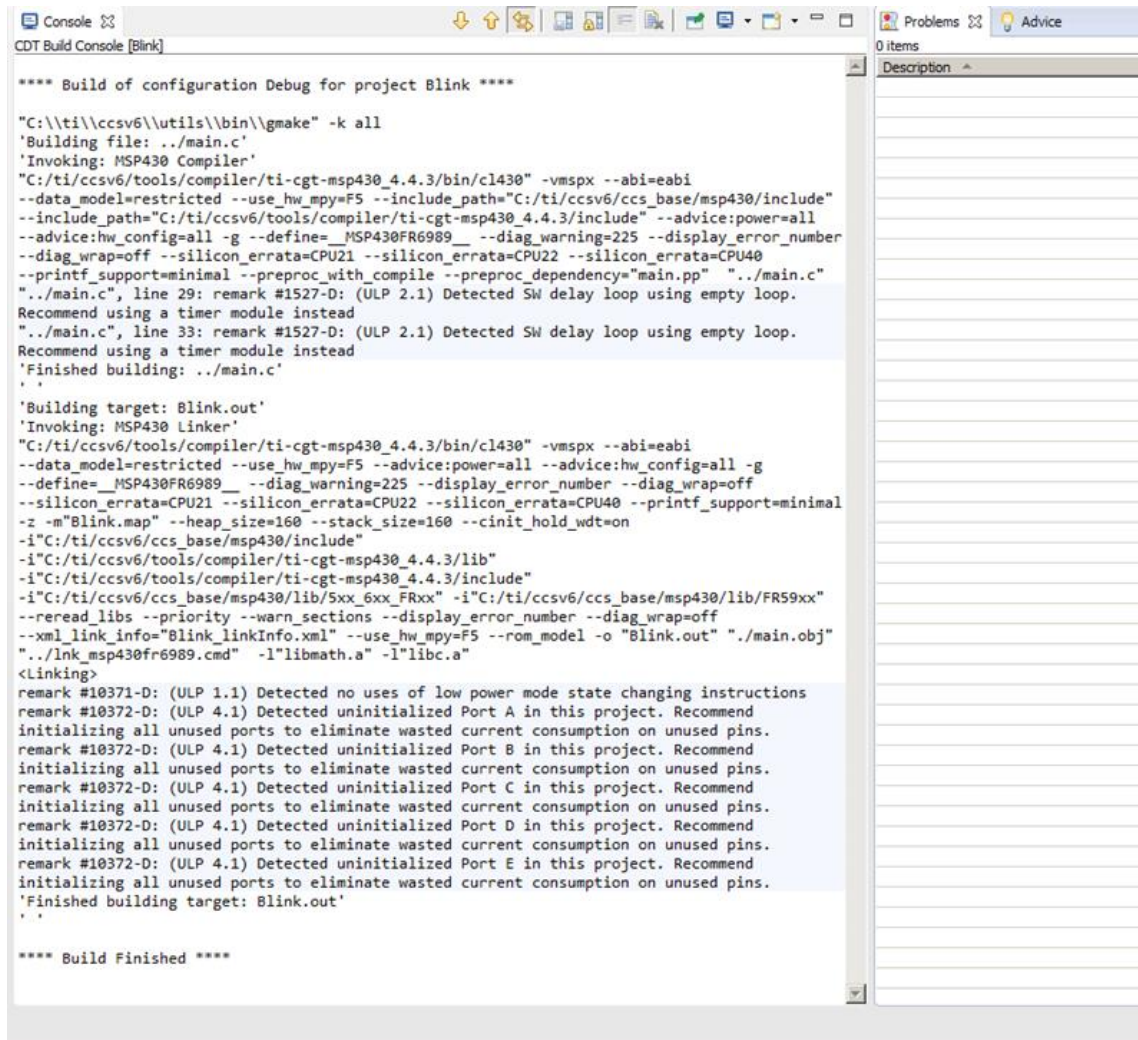
51. After you click **Build**, **CCS** translates your text **main.c** file into the **0**'s and **1**'s that your microcontroller can understand.
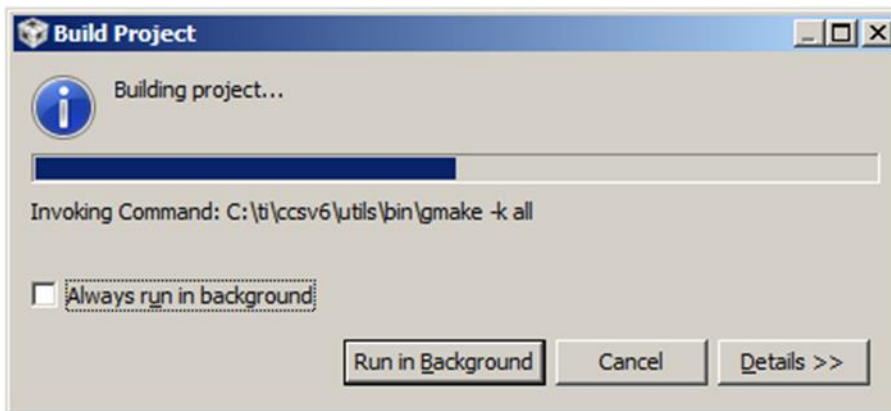
52. The **Build** results will be shown in the **Console** and **Problems** windows.at the bottom of the CCS window.

If there are any errors, you may have accidentally changed some of the instructions in the program. Go back and correct the errors. If CCS reports and warnings, you can disregard them at this time.

53. Once you successfully **Build** your program, you are ready to load it onto the microcontroller. To put your program onto the microcontroller, click the **Debug** button
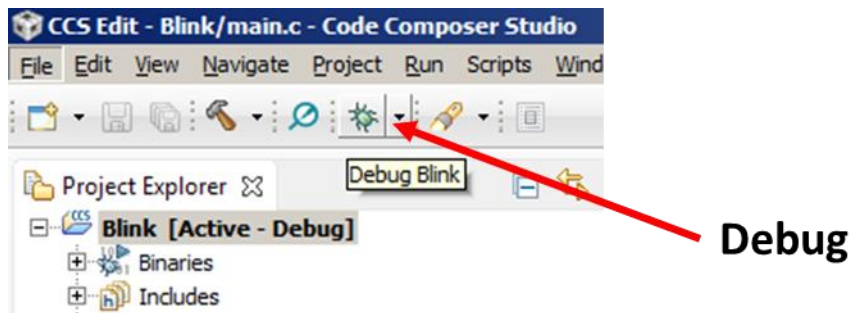
54. Sometimes when you click the **Debug** button, you'll get an error like this.

If this occurs, make sure your Launchpad is plugged in properly and click **Retry**.

If the error persists try unplugging and then replugging your Launchpad USB cable and then rebuilding your program.
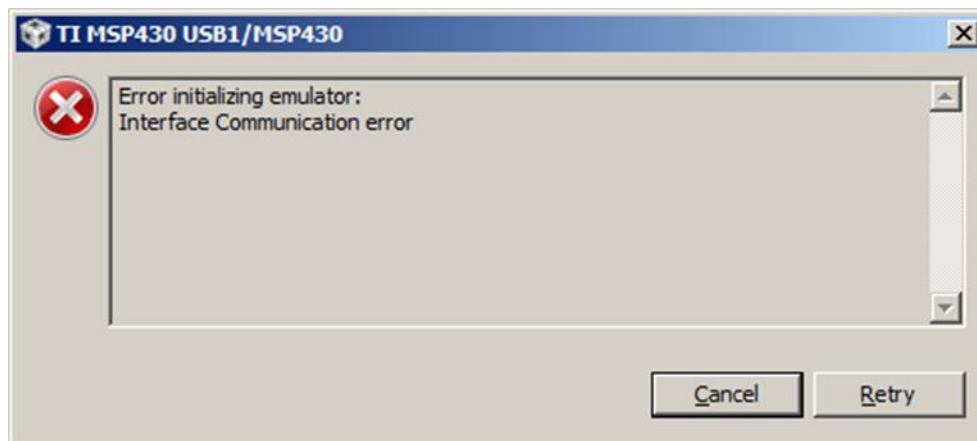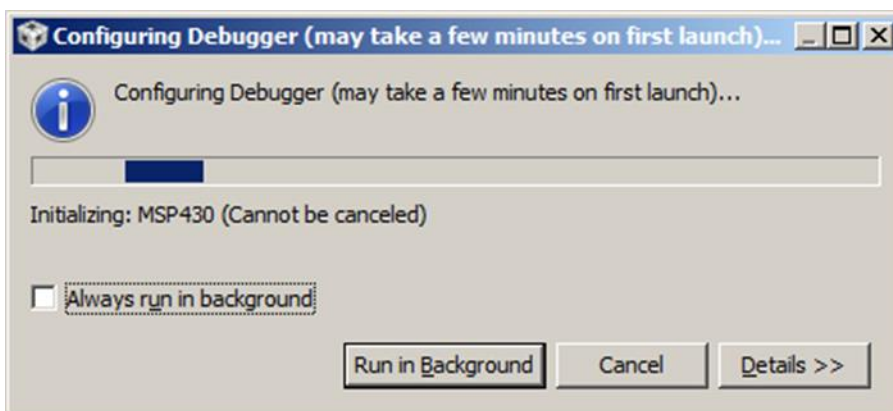
55.   You may see the next window pop up. It is notifying you of a new feature in **CCS** that can help you use best practices for ultra-low power applications.

For now, check the **Do not show this message again** box and click **Proceed**.
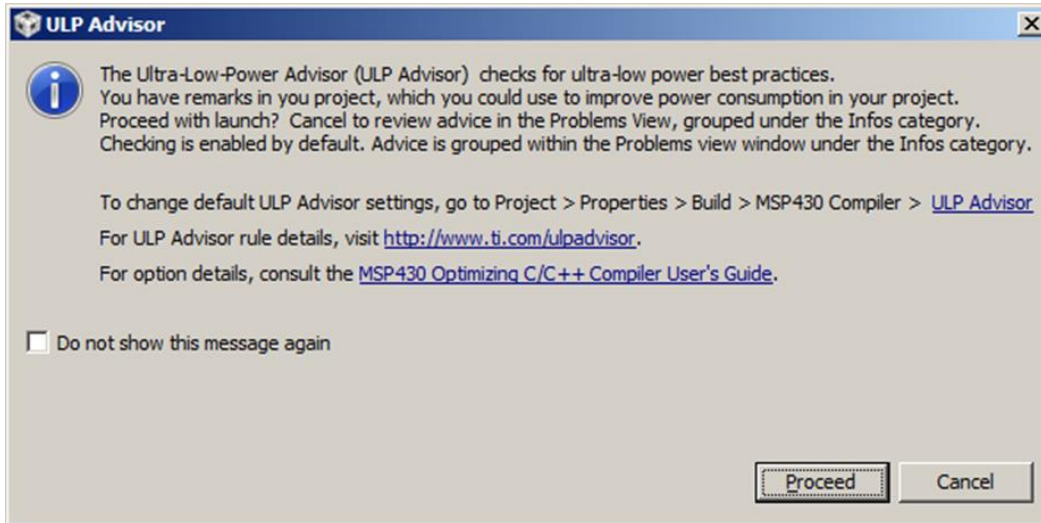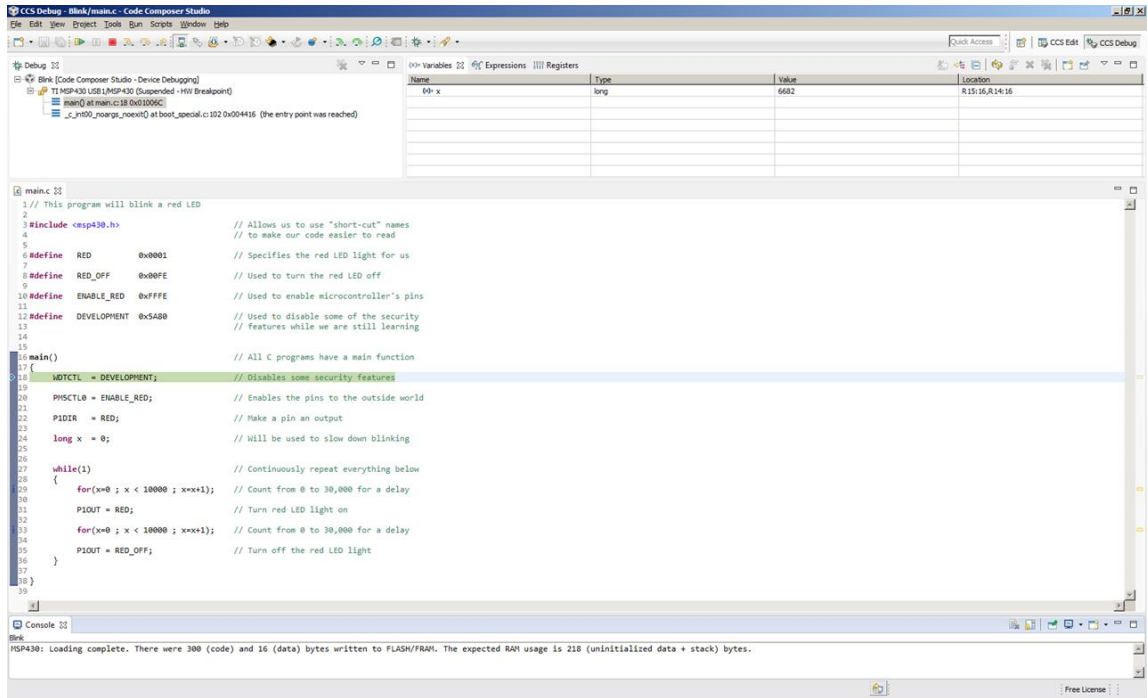

56.   The **CCS Debug** interface (or **Debugger**) will begin loading.

57.   When it is loaded, you should see the **CCS Debug** interface. Previously we have been in the **CCS Edit** interface.
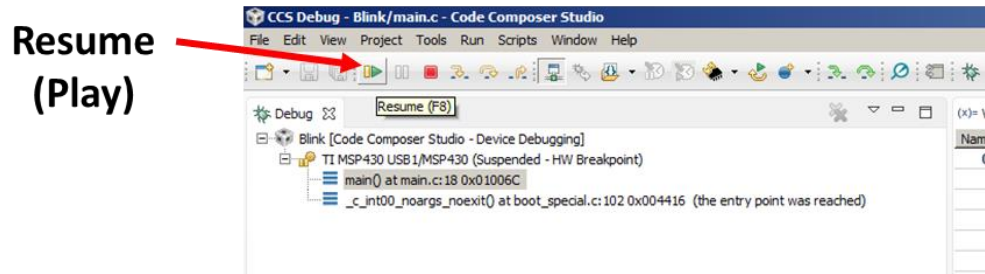
As you might expect, you create and edit your programs from the **CCS Edit** interface, and you run your programs from the **CCS Debug** interface.
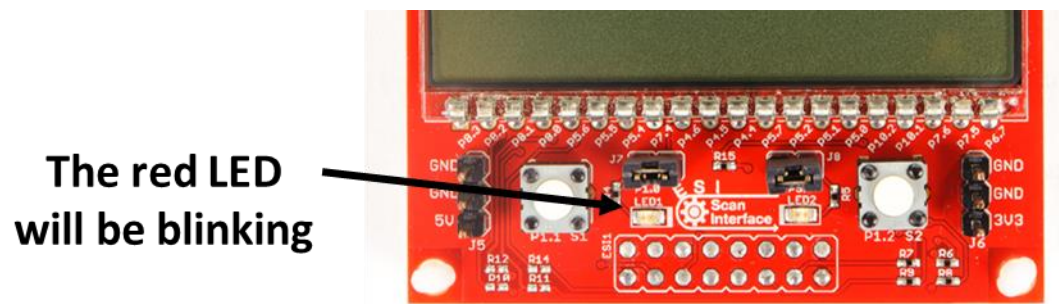

58.   The **Console** panel at the bottom will tell you how big your program is. In this example, the program was **300** + **16** = **316** bytes long.
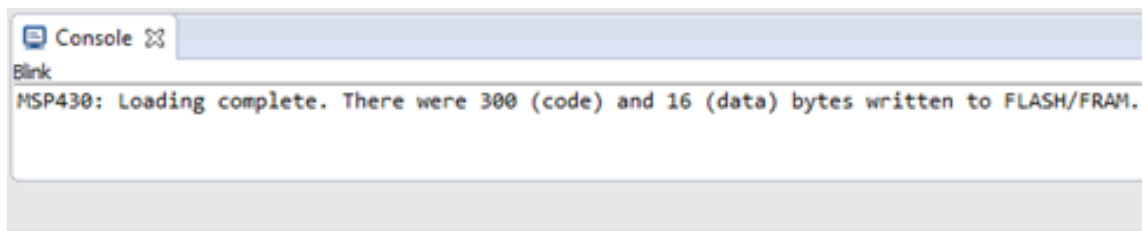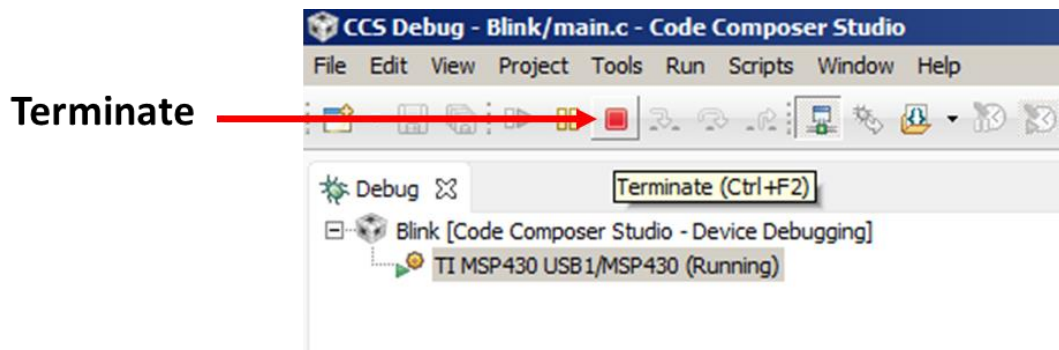
59. To start your program running, click the **Resume** (play) short-cut button on the toolbar.
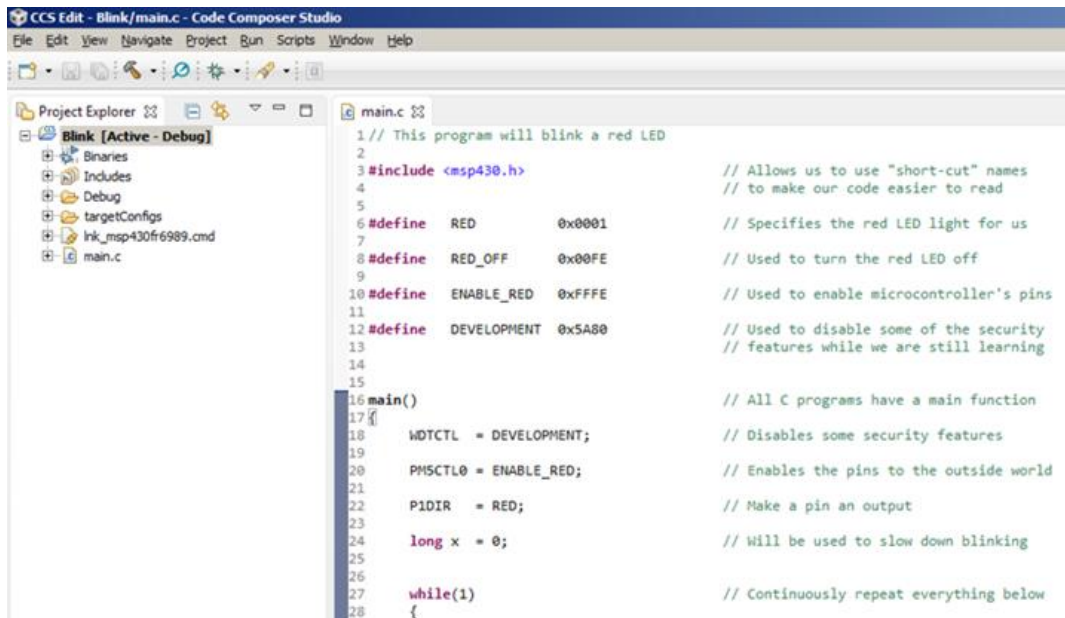

60. Your red LED light should now be blinking! Congratulations, you have completed your first microcontroller program.

61. Let's leave the **CCS Debugger** and go back to the **Editor**. To do this, click on the **Terminate** short-cut button.

62.   This closes the **Debugger** and opens the **Editor**.

63.   However, if you look at your Launchpad, you will see that the red LED is still blinking.

       Terminate refers to terminating (ending) the **Debugger**. Your program will continue to run on the Launchpad.

64. Try changing the **30000** values in the **for** loops to implement different delay lengths. Try changing both values to **10000**. (Note, do not use 10,000 or 10.000 in **CCS**. Just **10000**.)

```
16 main()                                      // All C programs have a main function
17 {
18      WDTCTL  = DEVELOPMENT;                  // Disables some security features
19
20      PM5CTL0 = ENABLE_RED;                   // Enables the pins to the outside world
21
22      P1DIR   = RED;                          // Make a pin an output
23
24      long x  = 0;                            // Will be used to slow down blinking
25
26
27      while(1)                                // Continuously repeat everything below
28      {
29          for(x=0 ; x < 10000 ; x=x+1);       // Count from 0 to 30,000 for a delay
30
31          P1OUT = RED;                        // Turn red LED light on
32
33          for(x=0 ; x < 10000 ; x=x+1);       // Count from 0 to 30,000 for a delay
34
35          P1OUT = RED_OFF;                    // Turn off the red LED light
36      }
37
38 }
```

65. **Save** and **Build** your program. When complete, launch the **Debugger**.

66. Click **Resume** to start your new program. Notice that the red LED is blinking faster (3 times faster).

67. After trying out the **10000** value, you can **Terminate** the **Debugger** and try different values with the **Editor**. Just remember to **Save** and **Build** your program after you make any changes before launching the **Debugger**.

68. This concludes our **Let's Get Started** handout. Congratulations!