

Code Composer Studio

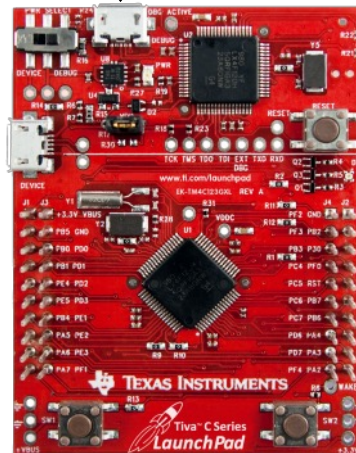
Objective

In this lab, we'll create a project that contains two source files, `main.c` and `startup_ccs.c`, which contain the code to blink an LED on your LaunchPad board. The purpose of this lab is to practice creating projects and getting to know the look and feel of Code Composer Studio. In later labs we'll examine the code in more detail. So far now, don't worry about the C code we'll be using in this lab.

Lab 2: Code Composer Studio



USB Emulation Connection



- ◆ Create a new project
- ◆ Experiment with some CCS features
- ◆ Use the LM Flash Programmer

Agenda ...

Lab 2 Procedure

Folder Structure for the Labs

1. Browse the directory structure for the workshop labs

- Using Windows Explorer, locate the following folder:

`C:\TM4C123G_LaunchPad_Workshop`

In this folder, you will find all the lab folders for the workshop. If you don't see this folder on your `c:\` drive, check to make sure you have installed the workshop lab files. Expand the `\lab2` folder and you'll notice that there are two sub-folders `\files` and `\project`. The `\files` folder will sometimes contain additional files for your reference. The `\project` folder will contain your project settings and files for both the projects that you create and the projects we created that you will import. It will also contain solution files saved as text files. You will be able to see these files in the Project Explorer and easily cut/paste the contents into your files if and when necessary.

Note: When you create a project, you have a choice to use the “default location” which is the CCS workspace or to select another location. In this workshop, we will not be using the workspace for the project files; rather, we'll use the folder where you installed the lab files, `C:\TM4C123G_LaunchPad_Workshop`.

The workspace will only contain CCS settings, and links to the projects we create or import.

Create a New CCS Project

2. Create a new project

- Launch CCS. When the “Select a workspace” dialog appears, ► browse to your My Documents folder:

(In WinXP) `C:\Documents and Settings\<user>\My Documents`

(In Win7) `C:\Users\<user>\My Documents`

Obviously, replace `<user>` with your own username. The name for your workspace isn't critical, but we suggest that you use `MyWorkspaceTM4C123G`. Do not check the “*Use this as the default and do not ask again*” checkbox. If at some point you accidentally check this box, it can be changed in CCS.

- Click OK.

3. Select a CCS License

If you haven't already licensed Code Composer, you may be asked to do so in the next few installation steps. You can do this step manually from the CCS Help menu.

- Click on *Help* → *Code Composer Studio Licensing Information*.

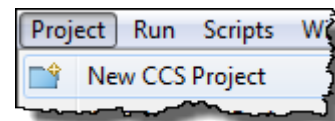
► Select the “*Upgrade*” tab, and then select the “*Free*” license. As long as your PC is connected to the LaunchPad board, CCS will have full functionality, free of charge.

4. Close TI Resource Explorer and/or Grace

When the “TI Resource Explorer” and/or “Grace” windows appear, close these windows using the “X” on the tab. At this time, these tools support other processor families, e.g. MSP430.

5. Create a New Project

To create a new project, ► select *Project* → *New CCS Project*:



► For the project name, type *lab2*

► Uncheck the box “*Use default location*” and click the *Browse...* button. Navigate to:

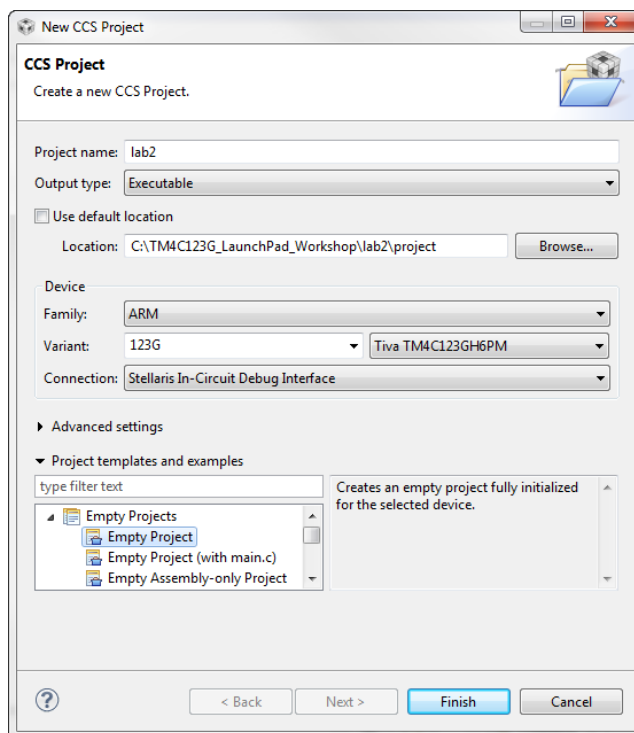
`C:\TM4C123G_LaunchPad_Workshop\lab2\project`

and click *OK*.

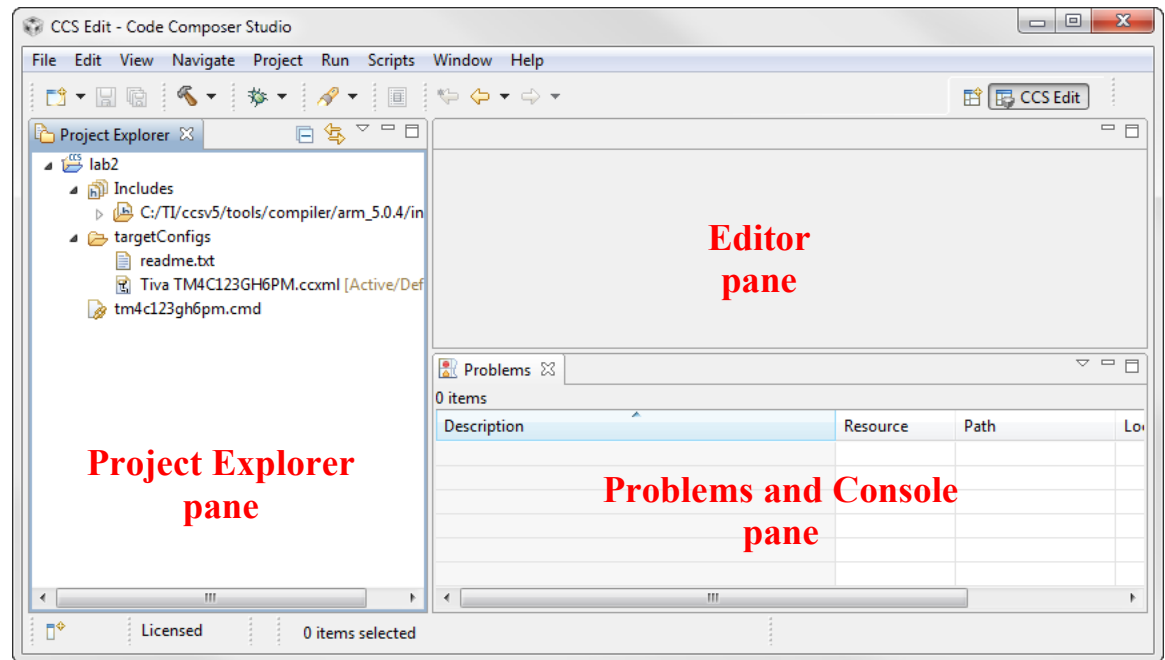
► Select Device family: *ARM*, for Variant, type *I23G* in the filter text field, then select *Tiva TM4C123GH6PM* in the drop-down box (typing *123G* narrows the list making it easier to find the exact part on the Tiva LaunchPad board.

► For Connection: choose *Stellaris In-Circuit Debug Interface*. This is the built-in emulator on the LaunchPad board.


► In the Project templates and examples box, choose *Empty Project* and then click *Finish*.



6. Review the CCS Editing GUI



Note the names of the Code Composer GUI panes above.

- In the Project Explorer pane on your desktop, click the  symbol next to *lab2*, *Includes* and *targetConfigs* to expand the project. Your project should look like the above.

Add Path and Build Variables

If you recall in the presentation, the path and build variables are used for:

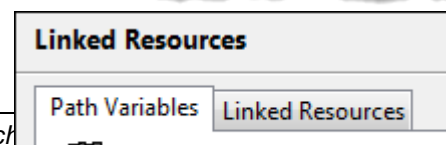
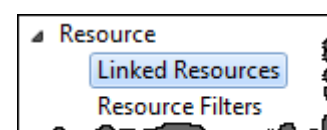
- Path variable – when you ADD (link) a file to your project, you can specify a “relative to” path. The default is *PROJECT_LOC* which means that your linked resource (like a *.lib* file) will be linked relative to your project directory.
- Build variable – used for items such as the search path for include files associated with a library – i.e. it is used when you build your project.

Variables can either have a *PROJECT* scope (that they only work for this project) or a *WORKSPACE* scope (that they work across all projects in the workspace).

In the next step, we need to add (link) a library file and then add a search path for include files. First, we’ll add these variables MANUALLY as *PROJECT* variables. Later, we will show you a quick and easy way to add these variables into your *WORKSPACE* so that any project in your workspace can use the variables.

7. Adding a Path Variable

To add a path variable, ► Right-click on your project and select *Properties*. ► Expand the *Resource* list in the upper left-hand corner as shown and click on *Linked Resources*:



You will see two tabs on the right side – *Path Variables* and *Linked Resources*:

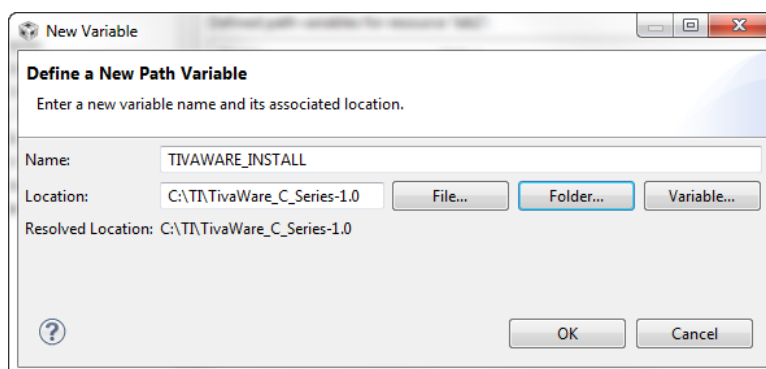
In the Path Variables tab, notice that *PROJECT_LOC* is listed and will display as the default path variable for linked resources in your project.

We want to add a *New* variable to specify exactly where you installed TivaWare.

► Click *New*

► When the New Variable dialog appears, type *TIVAWARE_INSTALL* for the *name*.

► For the *Location*, click the *Folder...* button and navigate to your TivaWare installation. Click on the folder name and then click OK.



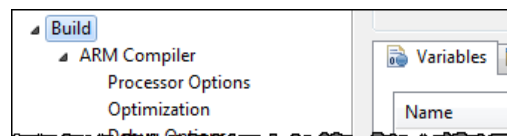
► Click OK. You should see your new path variable listed in the Path Variables list.

8. Adding a Build Variable

Now let's add a build variable that we will use in the include search path for the INCLUDE files associated with the TivaWare driver libraries.

► Click on *Build* and then the *Variables* tab:

► Click the *Add* button. When the *Define a New Build Variable* dialog appears, specify the same variable name as before, *TIVAWARE_INSTALL*



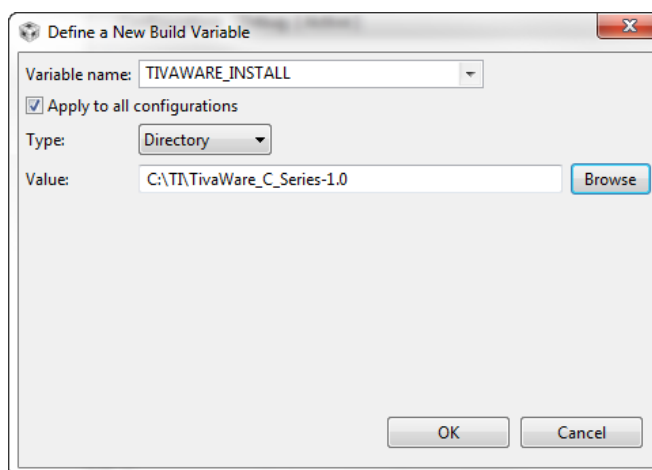
► Select Type: *Directory* so that the Browse button pops up. Then browse to your correct installation directory as shown.

► Make sure that the *Apply to all configurations* checkbox is checked (that way, if you change the build configuration from *Debug* to *Release*, you can still use this variable).

► Click Ok.

You should now see your new build variable in the list.

► Click Ok.



Add files to your project

We need to add two files `main.c` and `startup_ccs.c` to the project. We also need to add the TivaWare `driverlib.lib` object library. The C files should be copied to the project, the `driverlib` file should be linked.

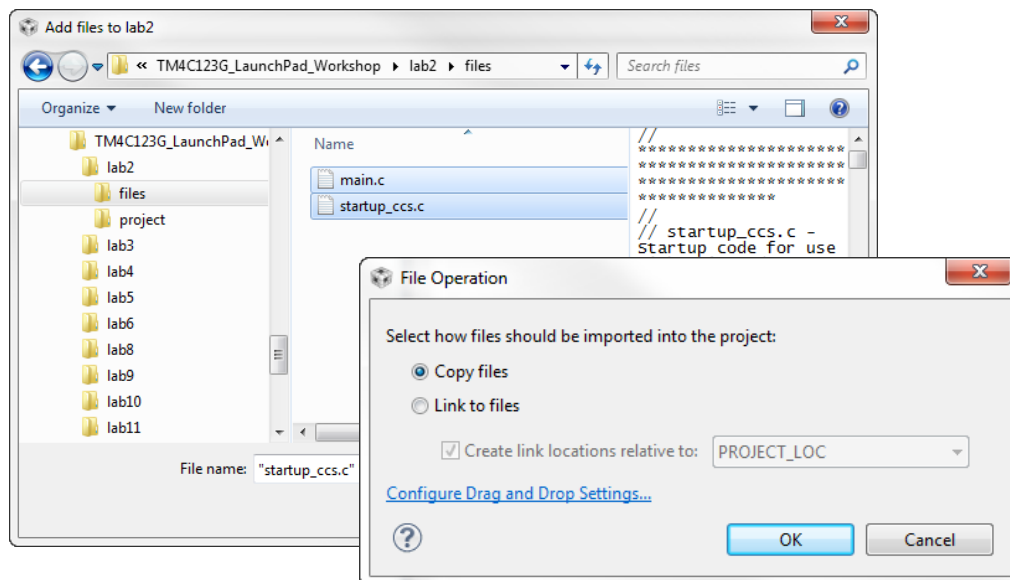
9. Add (copy) the C files

► Select *Project* → *Add Files...* ► Navigate to the folder:

`C:\TM4C123G_LaunchPad_Workshop\lab2\files`

Select the `main.c` and `startup_ccs.c` files and click *Open*.

Then select *Copy Files* and click *OK*.



10. Link the TivaWare `driverlib.lib` file to your project

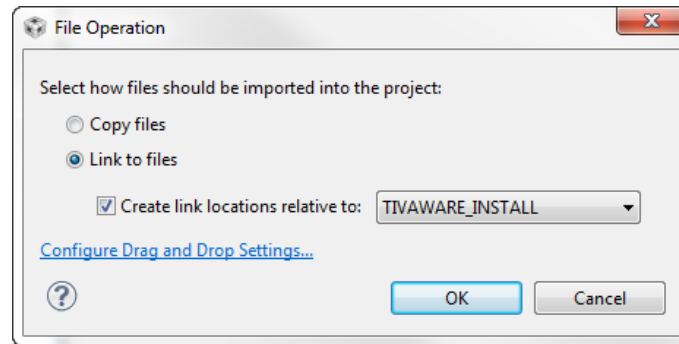
► Select *Project* → *Add Files...* Navigate to:

`C:\TI\TivaWare_C_Series-1.0\driverlib\ccs\Debug\driverlib.lib`

... and ► click *Open*. The *File Operation* dialog will open ...

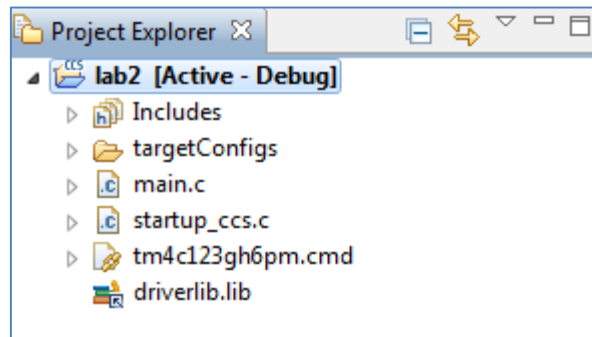
Use the `TIVAWARE_INSTALL` path variable you created earlier. This means that the `LINK` (or reference to the library) file will be `RELATIVE` to the location of the TivaWare installation. If you hand this project to someone else, they can install the project anywhere in the file system and this link will still work. If you choose `PROJECT_LOC`, you would get a path that is relative to the location of your project and it would require the project to be installed at the same “level” in the directory structure. Another advantage of this approach is

that if you wanted to link to a new version, say TivaWare_C_Series-1.1, all you have to do is modify the variable to the new folder name.



- Make the selections shown and click OK.

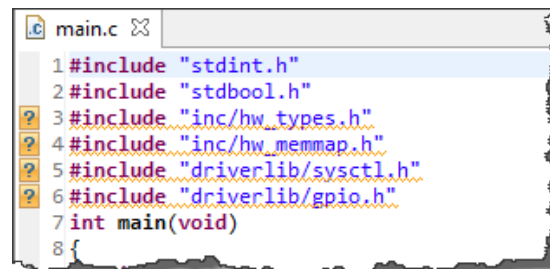
Your project should now look something like this:



Add the INCLUDE search paths for the header files

- Open your `main.c` file by double-clicking on the filename. You should see “?” warnings in the left margin which indicate “unresolved inclusion”.

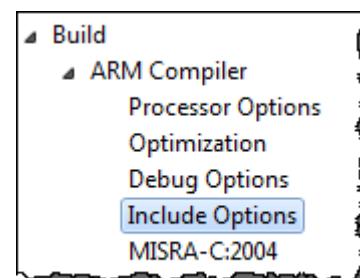
Until now, you haven’t told the project where to find these header files.



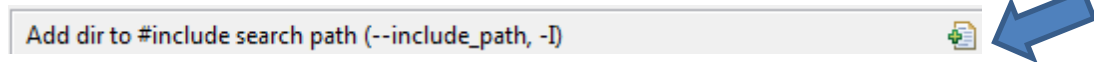
- Right-click on your lab2 project in the Project Explorer pane and select *Properties*.

- Click on *Build* → *ARM Compiler* → *Include Options* (as shown):

Which variable contains the location of your include files?
The build variable does.



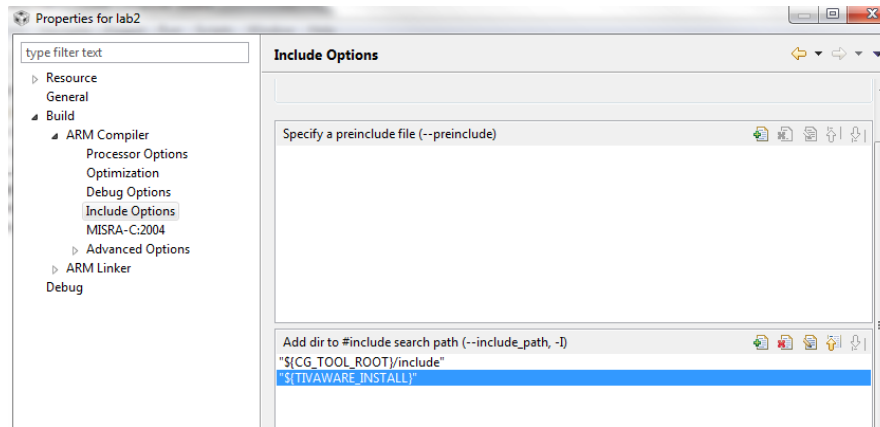
- In the lower-right panel, click the “+” sign next to *Add dir to #include search path*



and add the following path using the build variable you created earlier. Place the variable name inside braces, after the \$ as shown:

```
{TIVAWARE_INSTALL}
```

- Click OK.



- Click OK, and now you should see those “?” in `main.c` disappear.

Problem solved.

Examine your Project files using Windows Explorer

- Using Windows Explorer, locate your lab2 project folder:

```
C:\TM4C123G_LaunchPad_Workshop\lab2\project
```

Do you see `main.c`? It should be there because you copied it there. Do you see the `driverlib.lib` file? This file should NOT be there because it’s only linked in your project. Notice the other folders in the `\project` folder – these contain your CCS project-specific settings. Close Windows Explorer.

11. Examine the properties of your new project

- In CCS, right-click on your project and select *Properties*. Click on each of the sections below:

Resource: This will show you the path of your current project and the resolved path if it is linked into the workspace. Click on “*Linked Resources*” and both tabs associated with this.

What is the PROJECT_LOC path? _____

Are there any linked resources? If so, what file(s)? _____

General: shows the main project settings. Notice you can change almost every field here AFTER the project was created.

Build → ARM Compiler: These are the basic compiler settings along with every compiler setting for your project.

Other: feel free to click on a few more settings, but don't change any of them.

► Click *Cancel*.

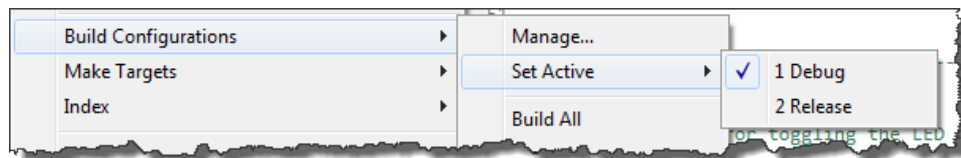
Explore Build Configurations

CCS ships with two default build configurations – *Debug* and *Release*. These are just containers for build options (compiler and linker). You can change the settings of the default configurations and you can create your own.

12. The *Debug* configuration turns on symbolic debug and turns the optimizer off. These options are ideal when you want to debug your program's logic and be able to single step your code, but they use up valuable flash memory with symbol tables.

The *Release* configuration typically turns off symbolic debug and turns on a medium level of optimization. This configuration usually provides better performance but it is more difficult (if not impossible) to single step your code because you only have function-level visibility.

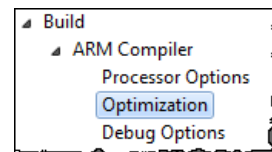
► In the Project Explorer pane, right-click on lab2 and select:



Make sure the configuration is set to *Debug*.

► Right-click on the project and select *Properties*.

► Under *Build → ARM Compiler*, click on *Optimization*:



What optimization level is used (-O)? _____

► Click on *Debug Options*:

Which debugging model is used? _____

► Click *Cancel*

► Switch the build configuration to *Release*.

Opt level (-O)? _____


Debugging model? _____


- Click *Cancel* and switch the build configuration back to *Debug*.

Build, Load, Run

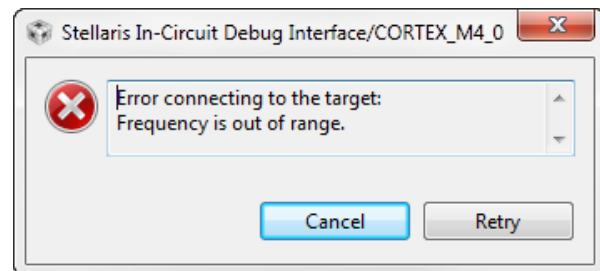
13. Build your project and fix any errors

- Build and load your project to the TM4C123GH6PM flash memory by clicking the Debug

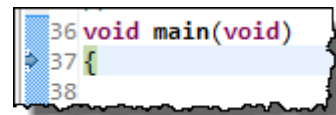
button . If you ever want to build the project without loading it, click the HAMMER

(Build) button: 

- Fix any errors that occur. If you encounter this error, either your board is disconnected or your power switch is in the wrong position.

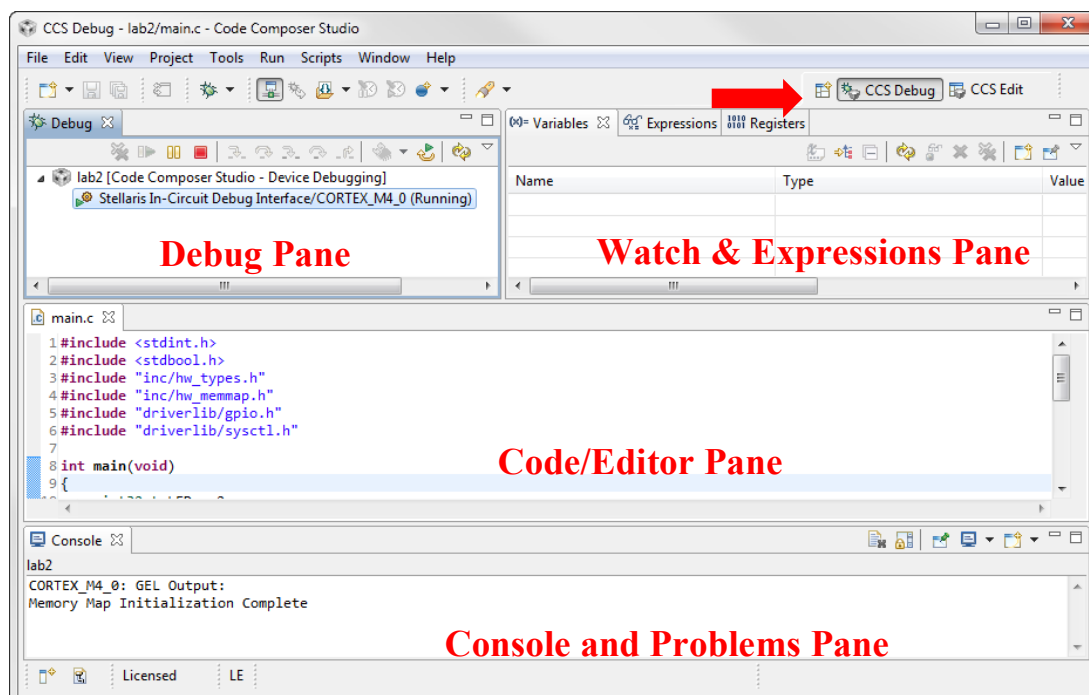


The program counter will run to `main()` and stop as shown:



```
36 void main(void)
37 {
38
```

14. Getting to know the CCS Debug GUI



Note the names of the Code Composer panes above. There are two pre-defined perspectives in Code Composer; CCS Edit and CCS Debug. ► Click and drag the tabs (at the arrow above) to the left so you can see both. Perspectives are only a “view” of the available data ... you can edit your code here without changing perspectives. And you can modify these or create as many additional perspectives as you like. More on that in a moment.

15. Run your program.

► Click the Resume button or press the F8 key on your keyboard:



The tri-color LED on your target board should blink showing the three colors in sequence. If not, attempt to solve the problem yourself for a few minutes, and then ask your instructor for help.

To stop your program running, ► click the Suspend button:



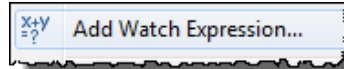
If the code stops with a “No source available ...” indication, click on the `main.c` tab. Most of the time in the `while()` loop is spent inside the delay function. That source file is not linked into this project.

16. Set a Breakpoint

In the code window in the middle of your screen, double-click in the blue area to the left of the line number of the `GPIOPinWrite()` instruction. This will set a breakpoint (it will look like this:). Click the Resume button to restart the code. The program will stop at the breakpoint and you will see an arrow on the left of the line number, indicating that the program counter has stopped on this line of code. **Note that the current ICDI driver does not support adding or removing breakpoints while the processor is running.** Click the Resume button a few times or press the F8 key to run the code. Observe the LED on the LaunchPad board as you do this.

17. View/Watch memory and variables.

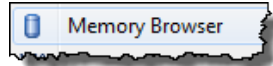
- ▶ Click on the Expressions tab in the Watch and Expressions pane.
- ▶ Double-click on the *LED* variable anywhere in `main()`.
- ▶ Right-click on *LED* and select:



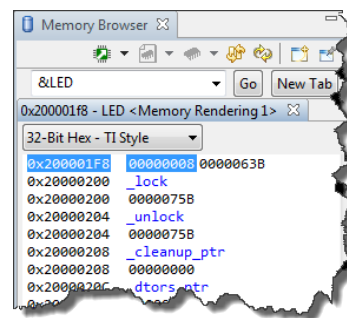
- ▶ Click Ok. Do you see *LED* in the list? What is the value? _____

Does *LED* live somewhere in memory? Of course it does. You can see the actual address in the expressions view. But let's go see it in memory.

- ▶ Select *View* → *Memory Browser*:

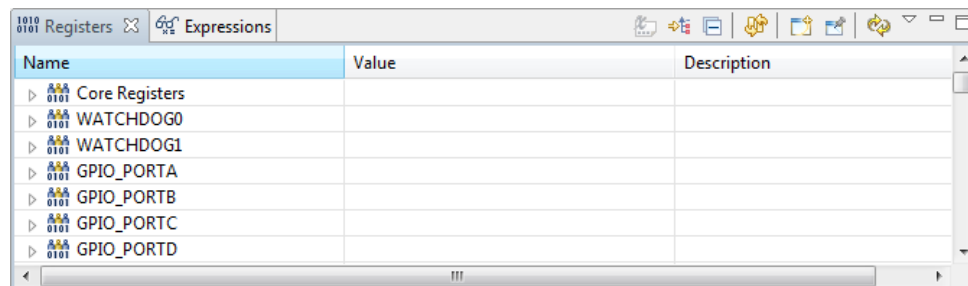


- ▶ Type “&LED” into the memory window to display *LED* in memory:



18. View Registers

- ▶ Select *View* → *Registers* and notice that you can see the contents of all of the registers in your target's architecture. This is very handy for debugging purposes.



- ▶ Click on the arrow on the left to expand the register view. Note that non-system peripherals that have not been enabled cannot be read. In this project you can view Core Registers, GPIO_PORTA (where the UART pins are), GPIO_PORTF (where the LEDs and pushbuttons are located), HIB, FLASH_CTRL, SYSCTL and NVIC.

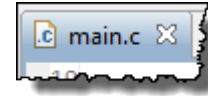
Perspectives

CCS perspectives are quite flexible. You can customize the perspective(s) and save them as your own custom views if you like. It's easy to resize, maximize, open different views, close views, and occasionally, you might wonder “How do I get things back to normal?”

19. Let's move some windows around and then reset the perspective.

- ▶ Right-click on the *Problems* window tab and select “*Detached*”. You can now move this window around wherever you want.
- ▶ Right click again and select “*Detached*” to re-attach it.


In the editing pane, ► double-click on the tab showing `main.c`:



Notice that the editor window maximizes to full screen. Double-click on the tab again to restore it.

► Move some windows around on your desktop by clicking-and-holding on the tabs.

Whenever you get lost or some windows seem to have disappeared in either the CCS Edit, CCS Debug or your own perspectives, you can restore the window arrangement back to the default.

► Find and click the Restore button  on the left or right of your display. If you want to reset the view to the factory default you can also choose *Window → Reset Perspective*:

20. Remove all breakpoints

► Click *Run → Remove All Breakpoints* from the menu bar or double-click on the breakpoint symbol in the editor pane. Again, breakpoints can only be removed when the processor is not running.

Terminate the debug session.

► Switch to the Edit Perspective. Click the red Terminate button:



LM Flash Programmer

LM Flash Programmer is a standalone programming GUI that allows you to program the flash of a Tiva C Series device through multiple ports. Creating the files required for this is a separate build step in Code Composer that is shown on the next page. If you have not done so already, install the LM Flash Programmer onto your PC.

Make sure that Code Composer Studio is not actively running code in the CCS Debug perspective... otherwise CCS and the Flash Programmer may conflict for control of the USB port.

21. Open LM Flash Programmer

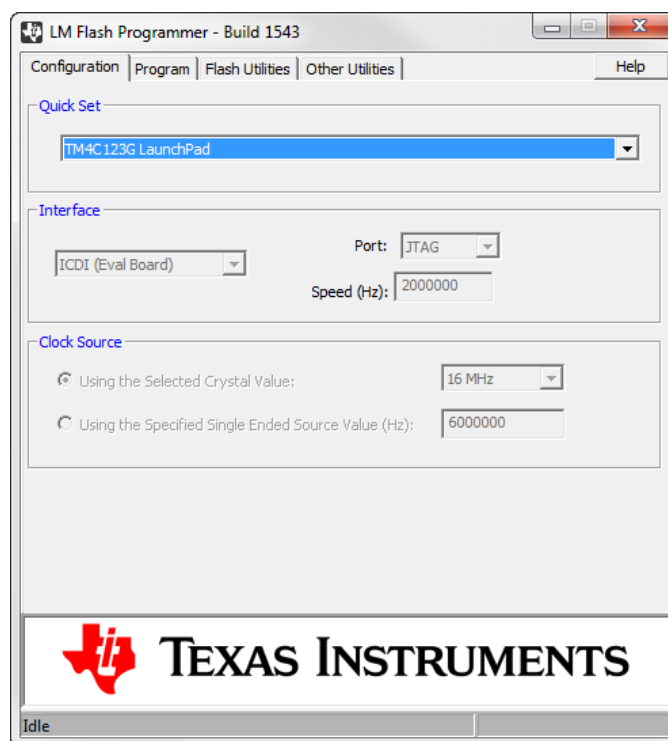
There should be a shortcut to the LM Flash Programmer on your desktop, double-click it to open the tool. If the shortcut does not appear, go to *Start* → *All Programs* → *Texas Instruments* → *Stellaris* → *LM Flash Programmer* and click on *LM Flash Programmer*.



Your evaluation board should currently be programmed with the lab2 application and it should be running. If the User LED isn't blinking, press the RESET button on the board.

We're going to program the original application back into the TM4C123GH6PM flash memory.

► Click the Configuration tab. Select the *TM4C123G LaunchPad* from the Quick Set pull-down menu under the Configuration tab. See the user's guide for information on how to manually configure the tool for targets that are not evaluation boards.



22. Click the Program Tab, then click the Browse button and navigate to:

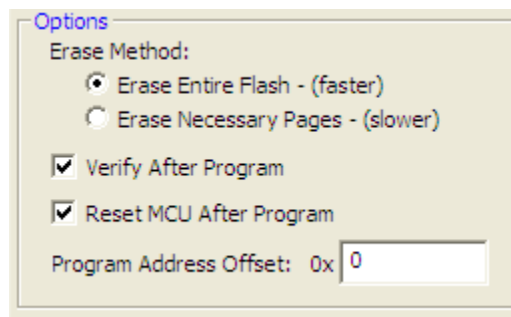
```
c:\TI\TivaWare_C_Series-1.0\examples\boards\ek-tm4c123gx1\
qs-rgb\ccs\Debug\qs-rgb.bin
```

and ► click Open. You may find that clicking on the ► symbol rather than the file name is easier to navigate.

qs-rgb is the application that was programmed into the flash memory of the TM4C123GH6PM when you removed it from the box.

Note that there are applications here which have been built with each supported IDE.

► Make sure that the following checkboxes are selected:

**23. Program**

► Click the Program button. You should see the programming and verification status at the bottom of the window. After these steps are complete, the quickstart application should be running on your LaunchPad.

24. Close the LM Flash Programmer

Creating a bin File for the Flash Programmer

If you want to create a .bin file for use by the stand-alone programmer in any of the labs in this workshop or in your own project, use these steps below. Remember that the project will have to be open before you can change its properties.

25. Set Post-Build step to call “tiobj2bin” utility

► In CCS Project Explorer, right-click on your project and select *Properties*. On the left, click Build and then the *Steps* tab. Paste the following commands into the *Post-build steps Command* box.

Note: The following four “lines” should be entered as a single line in the *Command* box. To make it easier, we included a text file that you can copy-paste. Navigate to C:\TM4C123G_LaunchPad_Workshop\postbuild.txt to find the complete command line.

```
"${CCS_INSTALL_ROOT}/utils/tiobj2bin/tiobj2bin"  
"${BuildArtifactFileName}" "${BuildArtifactFileName}.bin"  
"${CG_TOOL_ROOT}/bin/armofd" "${CG_TOOL_ROOT}/bin/armhex"  
"${CCS_INSTALL_ROOT}/utils/tiobj2bin/mkhex4bin"
```

PS: Make sure you don’t have any line breaks when inputting the above command

26. Rebuild your project

This post-build step will run after your project builds and the .bin file will be in the C:\TM4C123G_LaunchPad_Workshop\labx\project\debug folder. You can access this .bin in the CCS Project Explorer in your project by expanding the Debug folder.

If you try to re-build and you receive a message “gmake: Nothing to be done for ‘all’ .”, this indicates that no files have changed in your project since the last time you built it. You can force the project to build by first right-clicking the project and then select *Clean Project*. Now you should be able to re-build your project which will run the post-build step to create the .bin file.



You’re done.

Submissions

There is no submission required for this Lab.