TI ARM Lab 4
Digilent Orbit Board Introduction and LEDs

National Science Foundation
Funded in part, by a grant from the National Science Foundation
DUE 1068182

Acknowledgements
Developed by Craig Kief, Brian Zufelt, and Jacy Bitsoie at the Configurable Space Microsystems Innovations & Applications Center (COSMIAC). Co-Developers are Bassam Matar from Chandler-Gilbert and Karl Henry from Drake State. *Funded by the National Science Foundation (NSF).*

Lab Summary
This lab is very similar to the third lab but it is accomplished using the Digilent Corporation's Orbit daughter card. This card was built for academic purposes for learning how to do small projects (with a pathway to more complex projects) and can be found as a combination with the Stellaris board on the Digilent website.

Lab Goal
The goal of this lab is to continue to build upon the skills learned from previous labs. This lab helps the student to continue to gain new skills and insight on the C code syntax and how it is used in the TI implementation of the ARM processor with the addition of a daughter board. Each of these labs add upon the previous labs and it is the intention of the authors that they will build with each lab a better understanding of the ARM processor and basic C code. Even though these tutorials assume the student has not entered with a knowledge of C code, it is the desire that by the time the student completes the entire series of tutorials that they will have a sufficient knowledge of C code so as to be able to accomplish useful projects on the ARM processor.

Learning Objectives
The student should continue to become familiar with the complier and understand the use and modification of a "main.c" file. The student should also begin to be exposed to C "functions."

Grading Criteria
N/A

Time Required
Approximately one hour

Lab Preparation
It is highly recommended that the student read through this procedure once before actually using it was a tutorial. By understanding the final goal it will be easier to use this as a tutorial and as a learning guide. This lab creates the entire project from scratch. It assumes nothing. What the team found (after doing this tutorial) was an easier way to do this process that can be accomplished via an executable download file. However, the lab was left as it is to show the user (or those who can't make the executable file work) how to do these steps.

Equipment and Materials
Access to Stellaris LM4F120 LaunchPad software and evaluation kit (EK-LM4F-120XL) as well as the Digilent Orbit board. It is assumed that the student has already completed Lab 3 and the software is installed properly. Ensure that the boards are plugged in before start of the tutorial.

Software needed	Quantity
Download http://www.ti.com/tool/SW-EK-LM4F120XL Stellaris	1

LaunchPad™ software from the TI website. It will be best if it was possible to install the tutorial files from the COSMIAC installation link since it has a shell for all the labs. It can be found at http://www.cosmiac.org/Microcontrollers.html	
Hardware needed	Quantity
The hardware required is the TI Stellaris LaunchPad Kit and the Digilent Corporation's Orbit Daughter card	1

Additional References

Current TI ARM manuals found on TI web site: www.ti.com/lit/ug/spmu289a/spmu289a.pdf and the manuals for the Digilent website are located at www.digilentinc.com

Lab Procedure

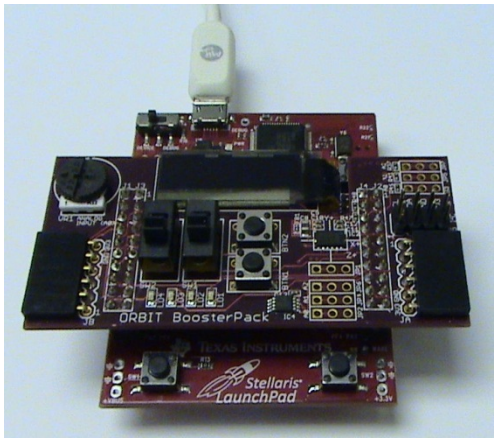


Figure 1. ARM Combination

This picture of the correct way to mate the Stellaris LaunchPad and the Digilent Orbit boards together. Please do so at this point and connect them as shown in Figure 1.

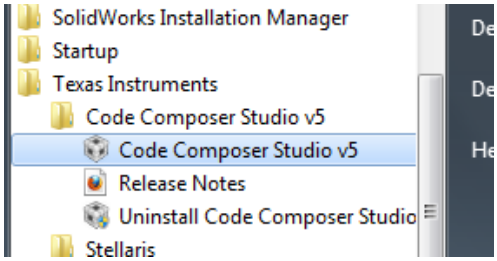


Figure 2. Code Composer

Launch Code Composer and where prompted, chose the workspaces location to store your project (as shown in Figure 3). We will later deviate from this location but for now it is acceptable. If you are asked about sending an exception report back to TI, chose either option. It isn't critical to this tutorial (or any other).

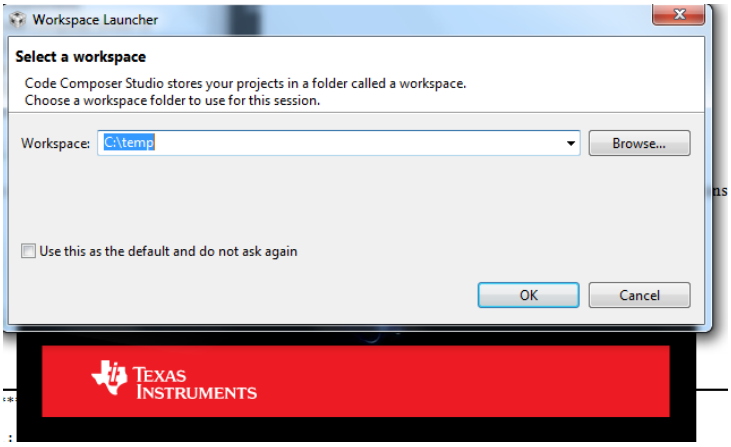


Figure 3. Workspace Selection

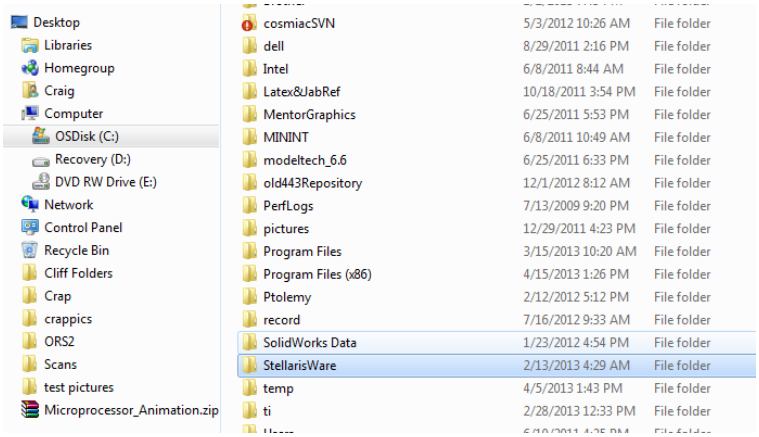


Figure 4. Software File

Open the StellarisWare folder using Microsoft Explorer as shown in Figure 4. The plan in the next several steps is to create a series of folder and then to drip in a startup file that contains all the functions we will be using.

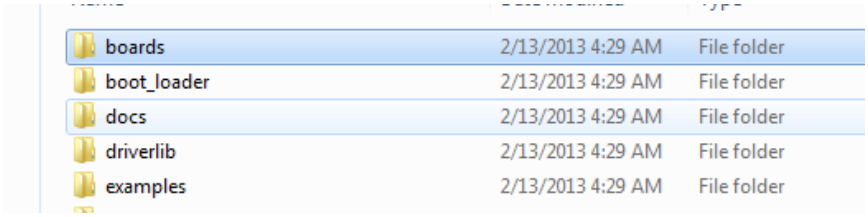


Figure 5. Project Folder Location

Open the “boards” folder is shown in Figure 5.

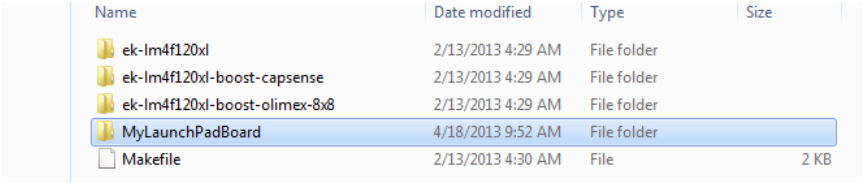


Figure 6. Project Folder Location

Within the boards folder, create a folder and call at My LaunchPad Board. As all future projects are created, they can go into subdirectories of this directory.

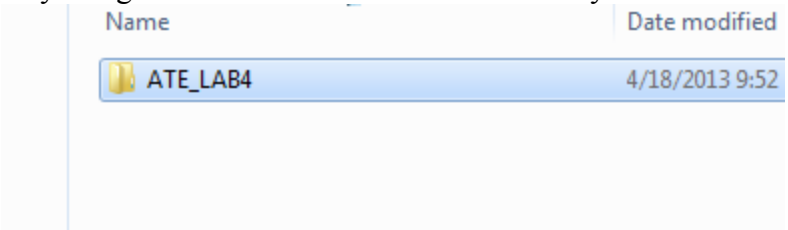


Figure 7. ATE Project Folder Location

Within the folder created, create a folder as shown in Figure 7.

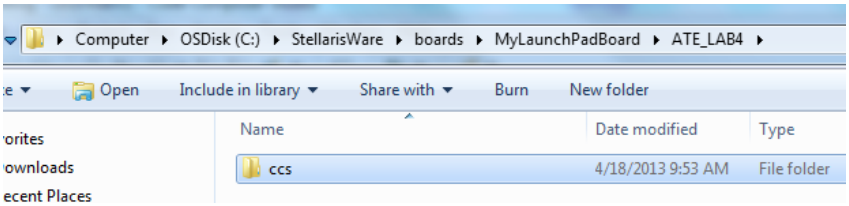


Figure 8. CCS Project Folder Creation

Finally create the last folder as shown in Figure 8. This ccs folder is where the startup file will be placed.

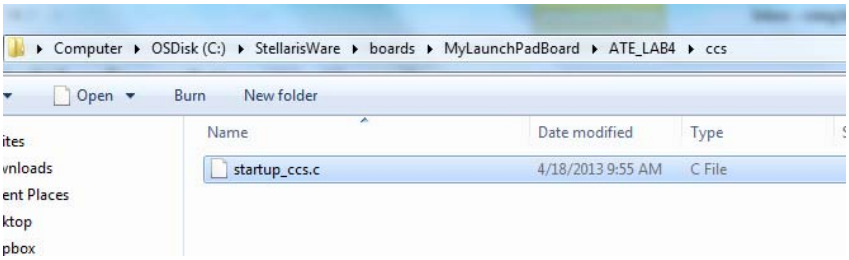


Figure 9. CCS Folder Location

Paste in the startup_css.c source file here. This file is on the website as a zip file and is also pasted at the end of the tutorial.

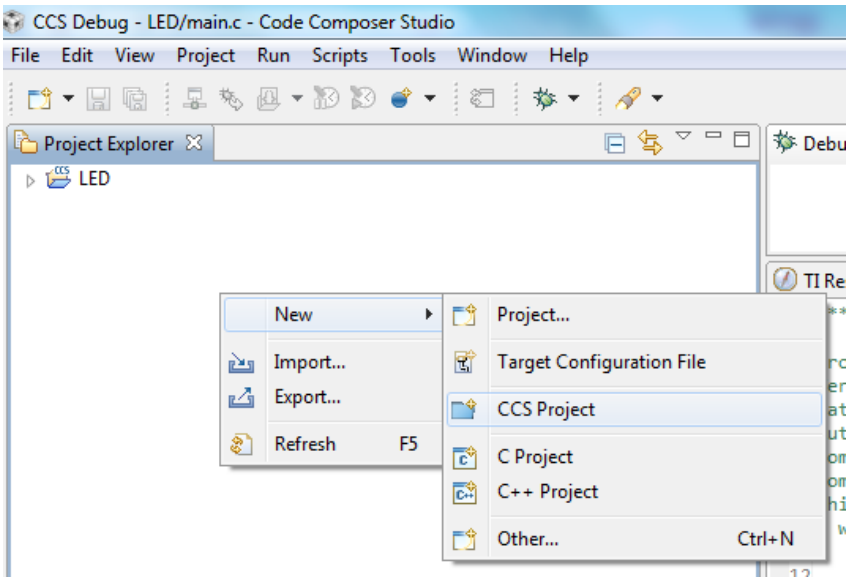


Figure 10. Create CCS Project

Return to Code Composer. For the next several steps, proceed slowly. It is easy to make simple mistakes. Right click into the Project Explorer section. Choose New → CCS Project.

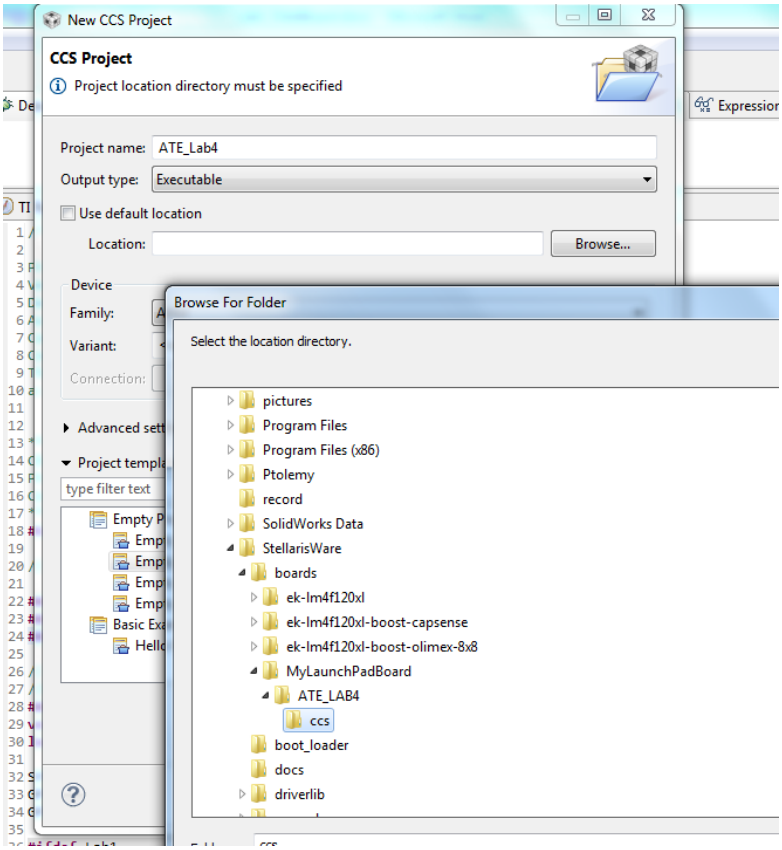


Figure 11. Modified Project Settings

The next several figures are related to the choices that need to be made on the new project. The first is to make the folder where the project will reside to be the folder that was just created and not c:\temp as was done in the previous tutorials. Set the settings as shown in Figure 11 and 12. In Figure 11, make sure the output type is executable. Deselect the default location choice and point he location to the ccs directory where the startup file was placed.

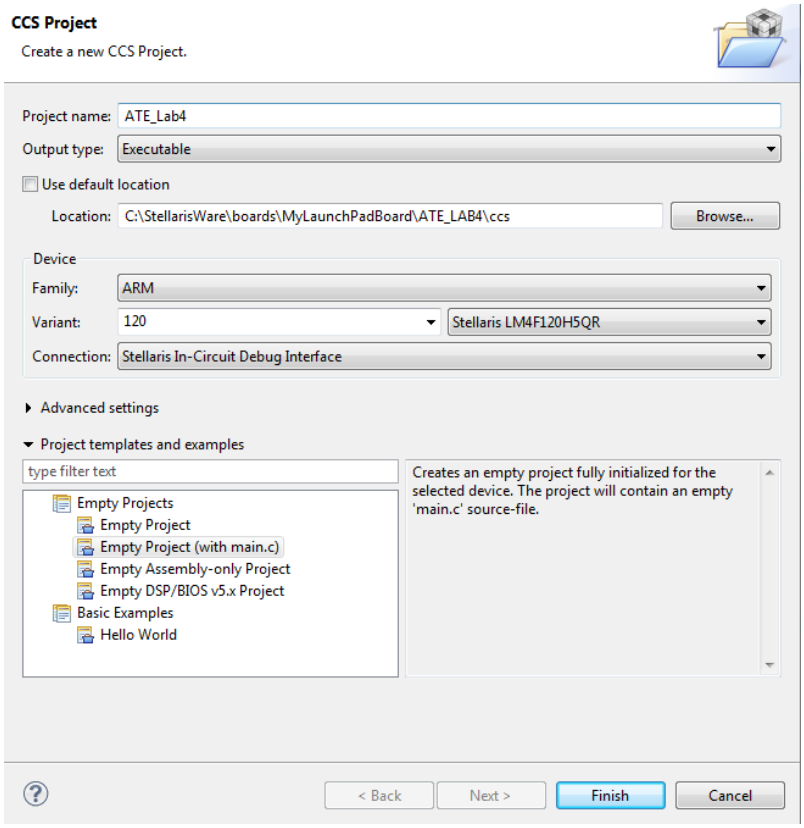
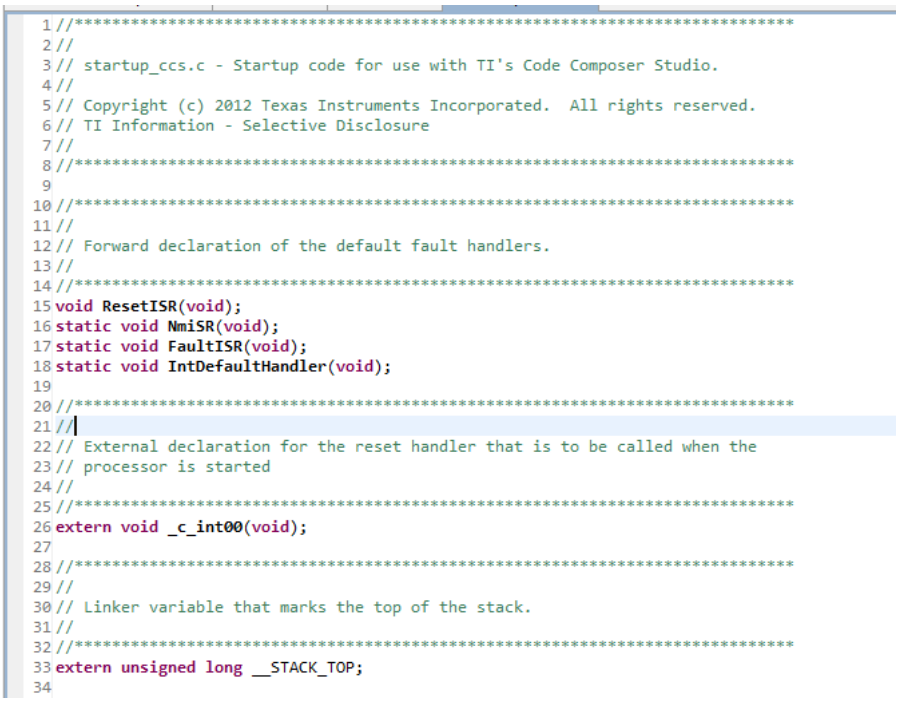


Figure 12. Modified Project Settings

Double check the settings for locations and hardware as shown in Figure 11 and 12. When you type the Variant value of 120 into the box, the specific processor option will become available. For Empty Project, choose the option that adds main.c. Even though we will not use the auto generated content of the main.c file, it is convenient to have the shell created through this process.



```
//*****************************************************************************
//
// startup_ccs.c - Startup code for use with TI's Code Composer Studio.
//
// Copyright (c) 2012 Texas Instruments Incorporated.  All rights reserved.
// TI Information - Selective Disclosure
//
//*****************************************************************************

//*****************************************************************************
//
// Forward declaration of the default fault handlers.
//
//*****************************************************************************
void ResetISR(void);
static void NmiSR(void);
static void FaultISR(void);
static void IntDefaultHandler(void);

//*****************************************************************************
//
// External declaration for the reset handler that is to be called when the
// processor is started
//
//*****************************************************************************
extern void _c_int00(void);

//*****************************************************************************
//
// Linker variable that marks the top of the stack.
//
//*****************************************************************************
extern unsigned long __STACK_TOP;

```

Figure 13. The Startup File

Open the startup_ccs.c file (also found in Attachment 1). Take a few minutes to look through this file. It wasn't written by the tutorial creators but is used to find and process certain functions.

Now, open the main.c file and delete the contents. Paste in all the code from main.c file (also found in Attachment 2)

Now it is time to introduce come new concepts in C code syntax. Here are three rules that the authors always follow. If they are memorized, it will make it easier to quickly work through new code when it is presented.

- Any time you see something where all letters are capitalized the authors will use this to identify a "define" statement for a preprocessor.
- If the first letter only is capitalized, it is what is known as a function cell.
- If all of the letters are lower case, it is known as a variable.

```
Core Clock frequency    : 80.000000 MHz
*******************************************************/

#include "inc/hw_ints.h"
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/sysctl.h"
#include "driverlib/interrupt.h"
#include "driverlib/gpio.h"
#include "driverlib/timer.h"


void main(void) {

    // Setting the internal clock
    SysCtlClockSet(SYSCTL_SYSDIV_5|SYSCTL_USE_PLL|SYSCTL_XTAL_16MHZ|SYSCTL_OSC_MAIN);

    // Enable Peripheral ports for input/ output
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOC);  //PORTC
    GPIOPinTypeGPIOOutput(GPIO_PORTC_BASE, GPIO_PIN_6|GPIO_PIN_7); // LED 1 LED 2
```

Figure 14. Source File Changes

Open the main.c file that has the newly pasted code into it. What is immediately seen as that many of the included statements have question marks. This means these “h” or header files can’t be found. Generally, a header file notifies the computer of certain things so that the complier can correctly build a single translation unit.

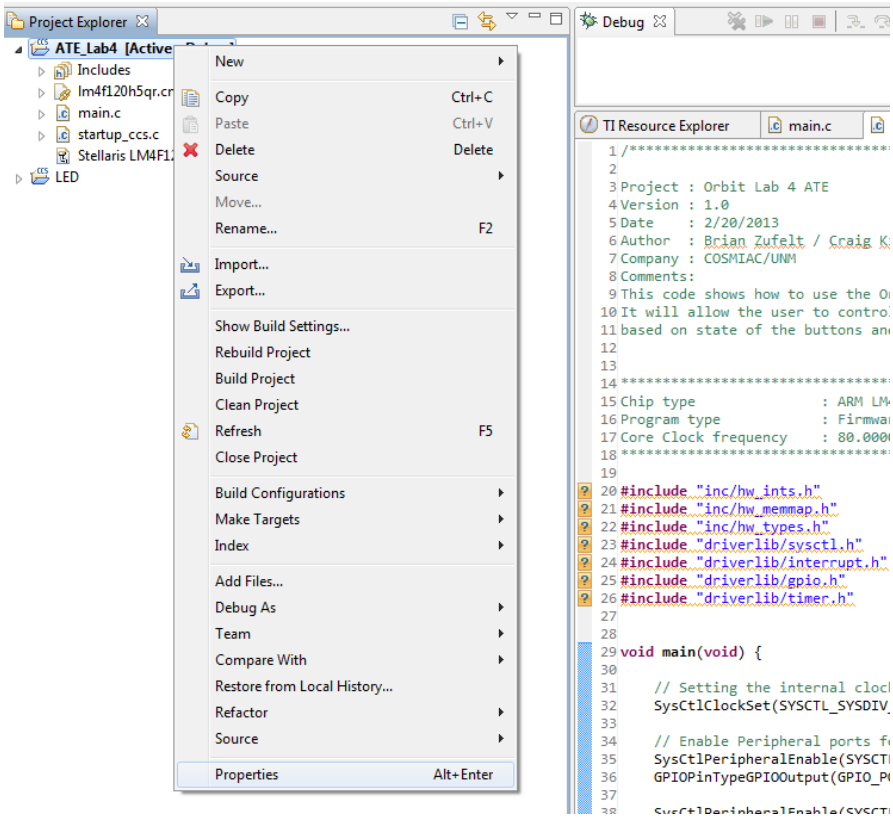


Figure 15. Project Explorer Select Properties

To clear the question marks, it is necessary to tell the system where to look for these source files. In the Project Explorer pane, right click on the bold project name and at the very bottom of the pop up screen, choose "properties." These next two options can also be found in Attachment 3 at the end of the tutorial.

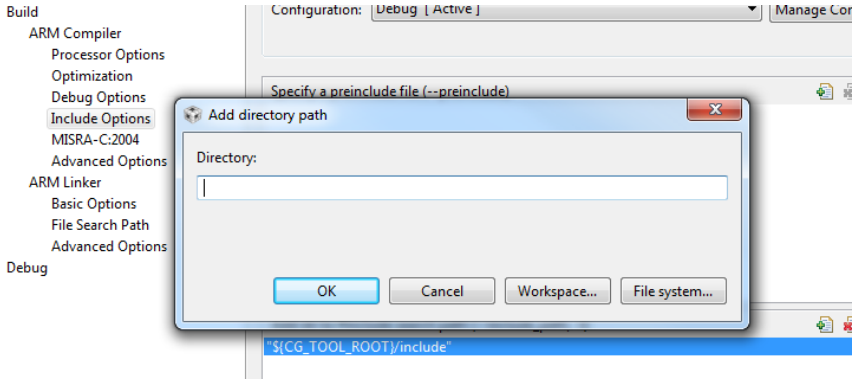


Figure 16. Select ARM Complier Directory Path

Highlight the ARM Compiler and then "Include Options." In the bottom window (where it says "add dir to…") click on the green "+" symbol. Here is the ARM Compiler Option that should be pasted into the "Add directory path":
"${PROJECT_ROOT}/../../../.."

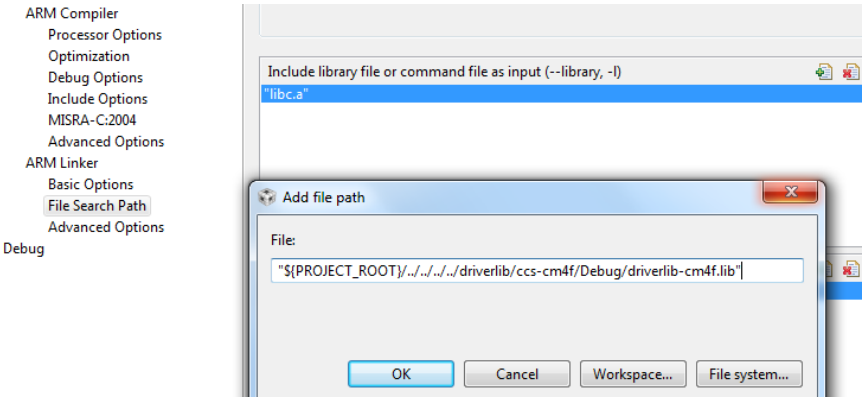


Figure 17. Add a New File Path for the ARM Linker

Next, it is necessary to add a location for the file search path to the ARM linker. Click on the ARM Linker and then on the File Search Path. In the upper window (under Include library file…) click on the green "+" symbol. Here is the ARM Linker file search path, this should be pasted into the "Add file path":
"${PROJECT_ROOT}/../../../../driverlib/ccs-cm4f/Debug/driverlib-cm4f.lib"

At the end, there should be two entries in the upper right window and two entries in the lower right window. Click OK. All the question marks next to the included statements in main.c should be gone. Now it is possible to compile the program, create the executable and run the program.

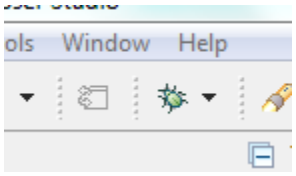


Figure 18. Debug Icon

Click on the debug icon. Save changes to main.c if prompted. The program will then compile the code, create and executable file and place it into the microcontroller.

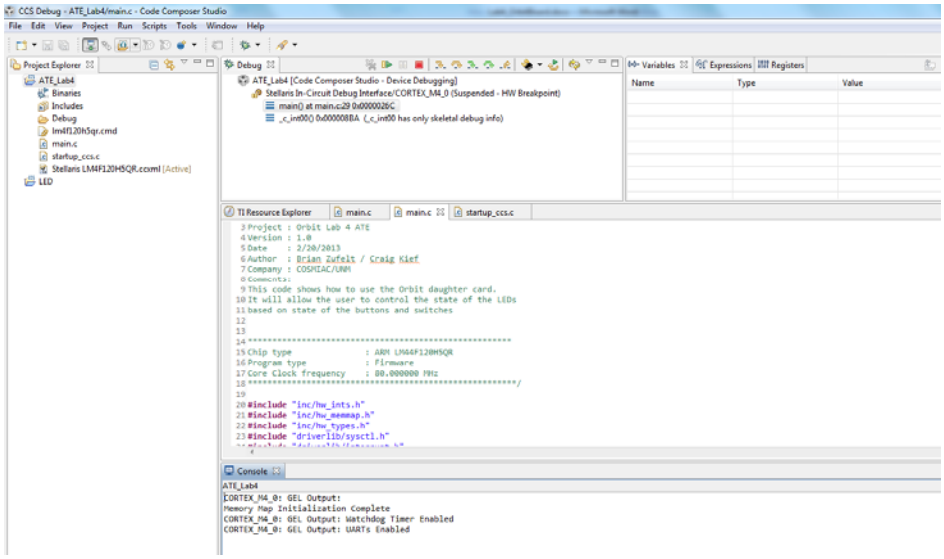


Figure 19. Screen Rearrange

What has now happened is that the executable file has been loaded into the microcontroller and has halted the program at void main (void) which is the start of your program. Using the mouse, rearrange the desktop to look like Figure 19. This will make it easier to see all the components as the program executes.

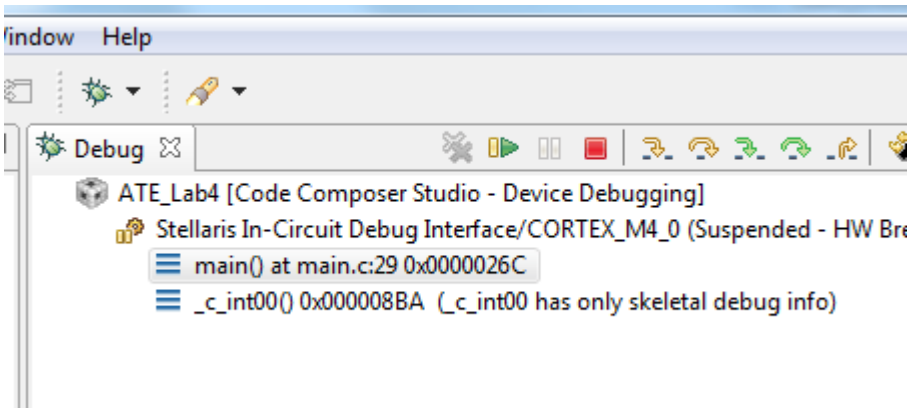


Figure 20. Run Executable

There are now several options. The designer can step line by line through the program by clicking on the yellow “step over” icon. When ready to continuously run the program, click on the resume icon (or hit F8) in the Debug pane to allow the program to continuously run.

Challenge: Remove the if/else statement, see results
Challenge: Change from button 1 to button 2

To accomplish this, you would need to refer to the LaunchPad Evaluation Board and the Orbit Schematics. There is a quick cheat sheet at the end of this tutorial that will provide ports and pins.

Attachment 1: Startup_ccs.c file

```
//*****************************************************************************
//
// startup_ccs.c - Startup code for use with TI's Code Composer Studio.
//
// Copyright (c) 2012 Texas Instruments Incorporated.  All rights reserved.
// TI Information - Selective Disclosure
//
//*****************************************************************************

//*****************************************************************************
//
// Forward declaration of the default fault handlers.
//
//*****************************************************************************
void ResetISR(void);
static void NmiSR(void);
static void FaultISR(void);
static void IntDefaultHandler(void);

//*****************************************************************************
//
// External declaration for the reset handler that is to be called when the
// processor is started
//
//*****************************************************************************
extern void _c_int00(void);

//*****************************************************************************
//
// Linker variable that marks the top of the stack.
//
//*****************************************************************************
extern unsigned long __STACK_TOP;

//*****************************************************************************
//
// The vector table.  Note that the proper constructs must be placed on this to
// ensure that it ends up at physical address 0x0000.0000 or at the start of
// the program if located at a start address other than 0.
//
//*****************************************************************************
#pragma DATA_SECTION(g_pfnVectors, ".intvecs")
void (* const g_pfnVectors[])(void) =
{
    (void (*)(void))((unsigned long)&__STACK_TOP),
                                            // The initial stack pointer
    ResetISR,                               // The reset handler
    NmiSR,                                  // The NMI handler
    FaultISR,                               // The hard fault handler
    IntDefaultHandler,                      // The MPU fault handler
    IntDefaultHandler,                      // The bus fault handler
```

```
    IntDefaultHandler,              // The usage fault handler
    0,                              // Reserved
    0,                              // Reserved
    0,                              // Reserved
    0,                              // Reserved
    IntDefaultHandler,              // SVCall handler
    IntDefaultHandler,              // Debug monitor handler
    0,                              // Reserved
    IntDefaultHandler,              // The PendSV handler
    IntDefaultHandler,              // The SysTick handler
    IntDefaultHandler,              // GPIO Port A
    IntDefaultHandler,              // GPIO Port B
    IntDefaultHandler,              // GPIO Port C
    IntDefaultHandler,              // GPIO Port D
    IntDefaultHandler,              // GPIO Port E
    IntDefaultHandler,              // UART0 Rx and Tx
    IntDefaultHandler,              // UART1 Rx and Tx
    IntDefaultHandler,              // SSI0 Rx and Tx
    IntDefaultHandler,              // I2C0 Master and Slave
    IntDefaultHandler,              // PWM Fault
    IntDefaultHandler,              // PWM Generator 0
    IntDefaultHandler,              // PWM Generator 1
    IntDefaultHandler,              // PWM Generator 2
    IntDefaultHandler,              // Quadrature Encoder 0
    IntDefaultHandler,              // ADC Sequence 0
    IntDefaultHandler,              // ADC Sequence 1
    IntDefaultHandler,              // ADC Sequence 2
    IntDefaultHandler,              // ADC Sequence 3
    IntDefaultHandler,              // Watchdog timer
    IntDefaultHandler,              // Timer 0 subtimer A
    IntDefaultHandler,              // Timer 0 subtimer B
    IntDefaultHandler,              // Timer 1 subtimer A
    IntDefaultHandler,              // Timer 1 subtimer B
    IntDefaultHandler,              // Timer 2 subtimer A
    IntDefaultHandler,              // Timer 2 subtimer B
    IntDefaultHandler,              // Analog Comparator 0
    IntDefaultHandler,              // Analog Comparator 1
    IntDefaultHandler,              // Analog Comparator 2
    IntDefaultHandler,              // System Control (PLL, OSC, BO)
    IntDefaultHandler,              // FLASH Control
    IntDefaultHandler,              // GPIO Port F
    IntDefaultHandler,              // GPIO Port G
    IntDefaultHandler,              // GPIO Port H
    IntDefaultHandler,              // UART2 Rx and Tx
    IntDefaultHandler,              // SSI1 Rx and Tx
    IntDefaultHandler,              // Timer 3 subtimer A
    IntDefaultHandler,              // Timer 3 subtimer B
    IntDefaultHandler,              // I2C1 Master and Slave
    IntDefaultHandler,              // Quadrature Encoder 1
    IntDefaultHandler,              // CAN0
    IntDefaultHandler,              // CAN1
    IntDefaultHandler,              // CAN2
    IntDefaultHandler,              // Ethernet
```

```
    IntDefaultHandler,                      // Hibernate
    IntDefaultHandler,                      // USB0
    IntDefaultHandler,                      // PWM Generator 3
    IntDefaultHandler,                      // uDMA Software Transfer
    IntDefaultHandler,                      // uDMA Error
    IntDefaultHandler,                      // ADC1 Sequence 0
    IntDefaultHandler,                      // ADC1 Sequence 1
    IntDefaultHandler,                      // ADC1 Sequence 2
    IntDefaultHandler,                      // ADC1 Sequence 3
    IntDefaultHandler,                      // I2S0
    IntDefaultHandler,                      // External Bus Interface 0
    IntDefaultHandler,                      // GPIO Port J
    IntDefaultHandler,                      // GPIO Port K
    IntDefaultHandler,                      // GPIO Port L
    IntDefaultHandler,                      // SSI2 Rx and Tx
    IntDefaultHandler,                      // SSI3 Rx and Tx
    IntDefaultHandler,                      // UART3 Rx and Tx
    IntDefaultHandler,                      // UART4 Rx and Tx
    IntDefaultHandler,                      // UART5 Rx and Tx
    IntDefaultHandler,                      // UART6 Rx and Tx
    IntDefaultHandler,                      // UART7 Rx and Tx
    0,                                      // Reserved
    0,                                      // Reserved
    0,                                      // Reserved
    0,                                      // Reserved
    IntDefaultHandler,                      // I2C2 Master and Slave
    IntDefaultHandler,                      // I2C3 Master and Slave
    IntDefaultHandler,                      // Timer 4 subtimer A
    IntDefaultHandler,                      // Timer 4 subtimer B
    0,                                      // Reserved
    0,                                      // Reserved
    0,                                      // Reserved
    0,                                      // Reserved
    0,                                      // Reserved
    0,                                      // Reserved
    0,                                      // Reserved
    0,                                      // Reserved
    0,                                      // Reserved
    0,                                      // Reserved
    0,                                      // Reserved
    0,                                      // Reserved
    0,                                      // Reserved
    0,                                      // Reserved
    0,                                      // Reserved
    0,                                      // Reserved
    0,                                      // Reserved
    0,                                      // Reserved
    0,                                      // Reserved
    IntDefaultHandler,                      // Timer 5 subtimer A
    IntDefaultHandler,                      // Timer 5 subtimer B
    IntDefaultHandler,                      // Wide Timer 0 subtimer A
    IntDefaultHandler,                      // Wide Timer 0 subtimer B
```

```
    IntDefaultHandler,                      // Wide Timer 1 subtimer A
    IntDefaultHandler,                      // Wide Timer 1 subtimer B
    IntDefaultHandler,                      // Wide Timer 2 subtimer A
    IntDefaultHandler,                      // Wide Timer 2 subtimer B
    IntDefaultHandler,                      // Wide Timer 3 subtimer A
    IntDefaultHandler,                      // Wide Timer 3 subtimer B
    IntDefaultHandler,                      // Wide Timer 4 subtimer A
    IntDefaultHandler,                      // Wide Timer 4 subtimer B
    IntDefaultHandler,                      // Wide Timer 5 subtimer A
    IntDefaultHandler,                      // Wide Timer 5 subtimer B
    IntDefaultHandler,                      // FPU
    IntDefaultHandler,                      // PECI 0
    IntDefaultHandler,                      // LPC 0
    IntDefaultHandler,                      // I2C4 Master and Slave
    IntDefaultHandler,                      // I2C5 Master and Slave
    IntDefaultHandler,                      // GPIO Port M
    IntDefaultHandler,                      // GPIO Port N
    IntDefaultHandler,                      // Quadrature Encoder 2
    IntDefaultHandler,                      // Fan 0
    0,                                      // Reserved
    IntDefaultHandler,                      // GPIO Port P (Summary or P0)
    IntDefaultHandler,                      // GPIO Port P1
    IntDefaultHandler,                      // GPIO Port P2
    IntDefaultHandler,                      // GPIO Port P3
    IntDefaultHandler,                      // GPIO Port P4
    IntDefaultHandler,                      // GPIO Port P5
    IntDefaultHandler,                      // GPIO Port P6
    IntDefaultHandler,                      // GPIO Port P7
    IntDefaultHandler,                      // GPIO Port Q (Summary or Q0)
    IntDefaultHandler,                      // GPIO Port Q1
    IntDefaultHandler,                      // GPIO Port Q2
    IntDefaultHandler,                      // GPIO Port Q3
    IntDefaultHandler,                      // GPIO Port Q4
    IntDefaultHandler,                      // GPIO Port Q5
    IntDefaultHandler,                      // GPIO Port Q6
    IntDefaultHandler,                      // GPIO Port Q7
    IntDefaultHandler,                      // GPIO Port R
    IntDefaultHandler,                      // GPIO Port S
    IntDefaultHandler,                      // PWM 1 Generator 0
    IntDefaultHandler,                      // PWM 1 Generator 1
    IntDefaultHandler,                      // PWM 1 Generator 2
    IntDefaultHandler,                      // PWM 1 Generator 3
    IntDefaultHandler                       // PWM 1 Fault
};

//*****************************************************************************
//
// This is the code that gets called when the processor first starts execution
// following a reset event.  Only the absolutely necessary set is performed,
// after which the application supplied entry() routine is called.  Any fancy
// actions (such as making decisions based on the reset cause register, and
// resetting the bits in that register) are left solely in the hands of the
// application.
```

```
//
//*****************************************************************************
void
ResetISR(void)
{
    //
    // Jump to the CCS C initialization routine.  This will enable the
    // floating-point unit as well, so that does not need to be done here.
    //
    __asm("    .global _c_int00\n"
          "    b.w     _c_int00");
}

//*****************************************************************************
//
// This is the code that gets called when the processor receives a NMI.  This
// simply enters an infinite loop, preserving the system state for examination
// by a debugger.
//
//*****************************************************************************
static void
NmiSR(void)
{
    //
    // Enter an infinite loop.
    //
    while(1)
    {
    }
}

//*****************************************************************************
//
// This is the code that gets called when the processor receives a fault
// interrupt.  This simply enters an infinite loop, preserving the system state
// for examination by a debugger.
//
//*****************************************************************************
static void
FaultISR(void)
{
    //
    // Enter an infinite loop.
    //
    while(1)
    {
    }
}

//*****************************************************************************
//
// This is the code that gets called when the processor receives an unexpected
// interrupt.  This simply enters an infinite loop, preserving the system state
```

```
// for examination by a debugger.
//
//*****************************************************************************
static void
IntDefaultHandler(void)
{
    //
    // Go into an infinite loop.
    //
    while(1)
    {
    }
}
```

Attachment 2: Main.c file

```
/*****************************************************

Project : Orbit Lab 4 ATE
Version : 1.0
Date    : 2/20/2013
Author  : Brian Zufelt / Craig Kief
Company : COSMIAC/UNM
Comments:
This code shows how to use the Orbit daughter card.
It will allow the user to control the state of the LEDs
based on state of the buttons and switches


*****************************************************
Chip type               : ARM LM44F120H5QR
Program type            : Firmware
Core Clock frequency  : 80.000000 MHz
*****************************************************/

#include "inc/hw_ints.h"
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/sysctl.h"
#include "driverlib/interrupt.h"
#include "driverlib/gpio.h"
#include "driverlib/timer.h"


void main(void) {

        // Setting the internal clock
        SysCtlClockSet(SYSCTL_SYSDIV_5|SYSCTL_USE_PLL|SYSCTL_XTAL_16MHZ|SYSCTL_OSC_MAIN);

        // Enable Peripheral ports for input/ output
        SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOC);  //PORTC
        GPIOPinTypeGPIOOutput(GPIO_PORTC_BASE, GPIO_PIN_6|GPIO_PIN_7); // LED 1 LED 2

        SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);  //PORTB
        GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_5); // LED 4

        SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOD);  //PORT D
        GPIOPinTypeGPIOOutput(GPIO_PORTD_BASE, GPIO_PIN_6); // LED 3
        GPIOPinTypeGPIOInput(GPIO_PORTD_BASE, GPIO_PIN_2); // BTN 1

        SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);  //PORT F
        GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3); // RGB LED on Launchpad

        SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);  //PORT A
        GPIOPinTypeGPIOInput(GPIO_PORTA_BASE, GPIO_PIN_6); // Switch 2

        while(1)
        {
                if(GPIOPinRead(GPIO_PORTA_BASE, GPIO_PIN_6)){    // Listen for the switch

                        GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3, 4); // LED Green on
Launchpad
```

```
                // Cycle through the LEDs on the Orbit board
                GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_6|GPIO_PIN_7, 0x40); // LED 1 on LED 2 Off
                GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_6, 0x00); // LED 3 off, Note different PORT
                GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_5, 0x00); // LED 4 off

                SysCtlDelay(2000000); // Delay, Replaces for LOOP

                GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_6|GPIO_PIN_7, 0x80); // LED 1 OFF, LED 2 on

                SysCtlDelay(2000000); // Delay

                GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_6|GPIO_PIN_7, 0x00); //LED 1 off LED 2 off
                GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_6, 0x40); // LED 3 on

                SysCtlDelay(2000000); //delay

                GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_6, 0x00); // LED 3 off
                GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_5, 0x20); // LED 4 on

                SysCtlDelay(2000000); // delay


        }

        else {

                GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3, 8); // LED Blue on
Stellaris

                // Cycle through the LEDs on the Orbit board
                GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_6|GPIO_PIN_7, 0x00); // LED 1 and 2, off
                GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_6, 0x00); // LED 3 off
                GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_5, 0x20); // LED 5 on

                SysCtlDelay(2000000); // delay

                GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_5, 0x00); // LED 4 off
                GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_6, 0x40); // LED 3 on

                SysCtlDelay(2000000); //delay

                GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_6|GPIO_PIN_7, 0x80); // LED 1 off, LED 2 on
                GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_6, 0x00);  //LED 3 off

                SysCtlDelay(2000000); //delay

                GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_6|GPIO_PIN_7, 0x40); //LED 1 on, LED 2 off

                SysCtlDelay(2000000); //delay

        }

        while(GPIOPinRead(GPIO_PORTD_BASE, GPIO_PIN_2)){     // Listen for BTN 1;  However, this
will not happen until finish the middle of the if, else loop

                // turn all the leds on
                GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_6|GPIO_PIN_7, 0xC0); // LED 1 and 2 on;
Could have been 0xFF by bit banding
                GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_6, 0x40); //LED 3 on
```

```
                    GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_5, 0x20); //LED 4 on
                    GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3, 0xC2); // LED on
stellaris to white

                    SysCtlDelay(2000000);

                    // turn off all the leds
                    GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_6|GPIO_PIN_7, 0x00); // LED 1 and 2 off
                    GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_6, 0x00); // LED 3 off
                    GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_5, 0x00); // LED 4 off
                    GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3, 0x00); // LED on
stellaris to off

                    SysCtlDelay(2000000);
                }

        }

}
```

Attachment 3: Two items to paste in:

Arm Compiler include options

"${PROJECT_ROOT}/../../../.."

ARM linker file search path

"${PROJECT_ROOT}/../../../../driverlib/ccs-cm4f/Debug/driverlib-cm4f.lib"

Attachment 3: Block Diagram of the Pins Used in Projects