Lab 0: Introduction to TI MSP-EXP430F5438

ECE/CS

Due by Thursday Jan 20,2011 11:59 PM

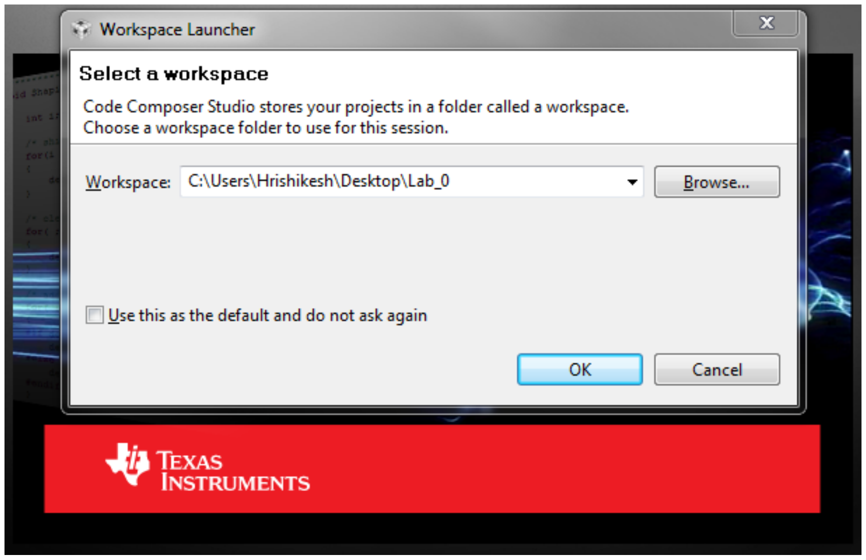
This document is divided into 2 main sections. The first one is a tutorial explaining how to work with the Code Composer Studio IDE. The second section pertains to the problem statement, questions and submission method.

1 Tutorial: Code Composer Studio (CCS)

The TI MSP-EXP430F5438 Experimenter Board will be used in the initial labs for this course. The aim of this tutorial is to take you through the entire flow of writing, building and debugging using the CCS. For this purpose, we will implement a simple LED blink program and then modify it. The CCS is a TI-custom extension of the Eclipse-IDE. For the lab, we will be using the MSP430 Data Sheet and User's Guide extensively.

1.1 Setting up the Project

Open the Code Composer Studio from Start Menu. You will be prompted to choose a workspace Figure 1.


Workspace Launcher

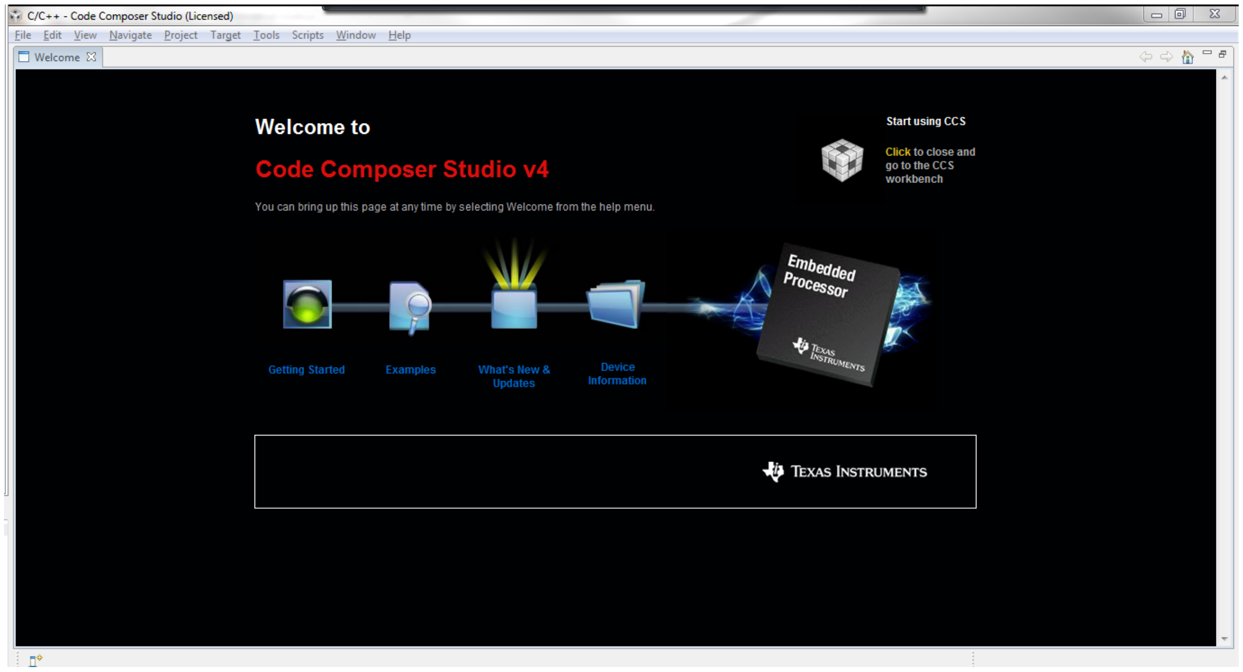
Select a workspace

Code Composer Studio stores your projects in a folder called a workspace.
Choose a workspace folder to use for this session.

Workspace: C:\Users\Hrishikesh\Desktop\Lab_0

Figure 1: Opening CCS

The next screen is the Welcome screen, Figure 2. You are encouraged to look through the options given in the welcome screen.



Welcome to

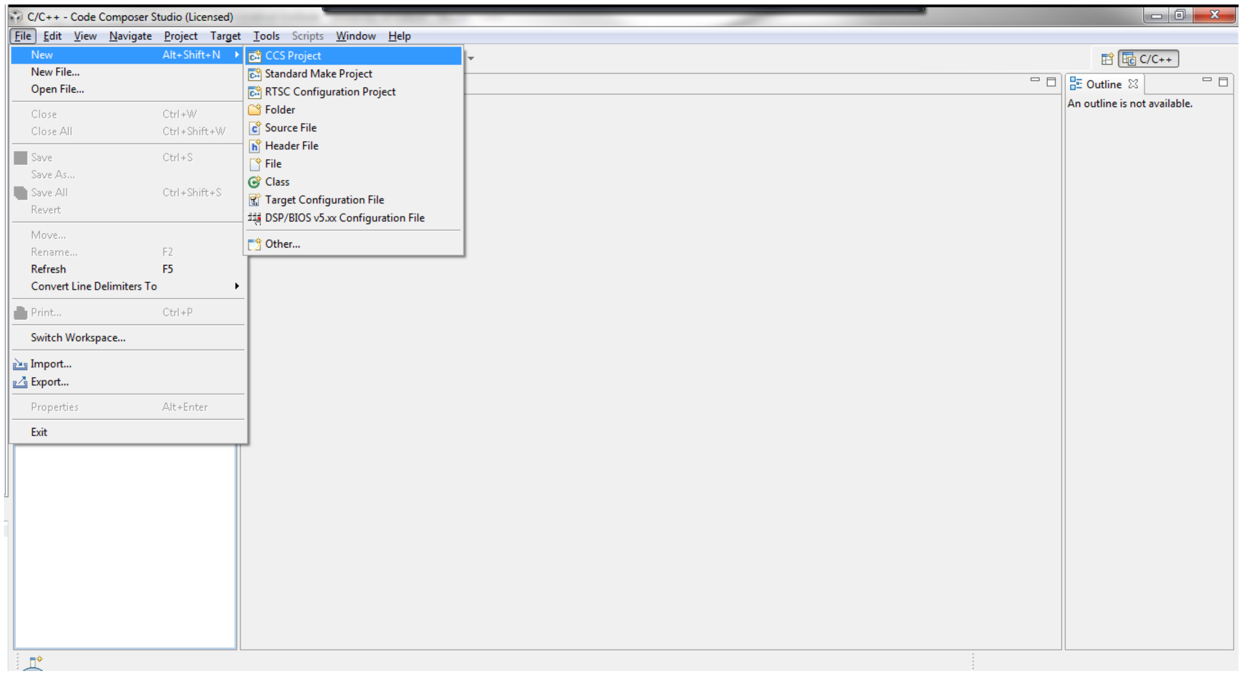
Code Composer Studio v4

You can bring up this page at any time by selecting Welcome from the help menu.

Click to close and go to the CCS workbench

Figure 2: Welcome Screen

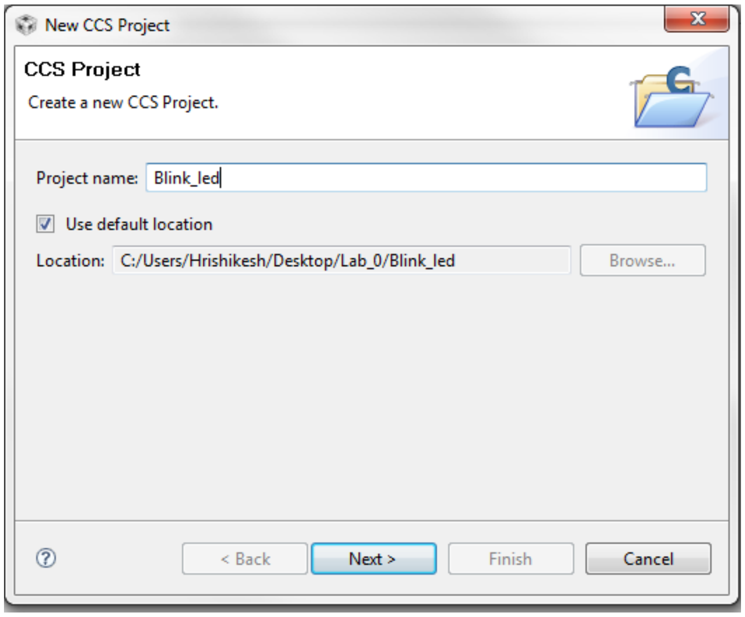
Once you close the Welcome screen, you will be taken to an empty project screen. From *File* → *New*, choose *CCS Project* as shown in Figure 3.



An outline is not available.

Figure 3: Create New Project

Enter the project name on the subsequent prompt and click *Next*.
Choose MSP430 if already not chosen at the next screen and proceed.



New CCS Project

CCS Project

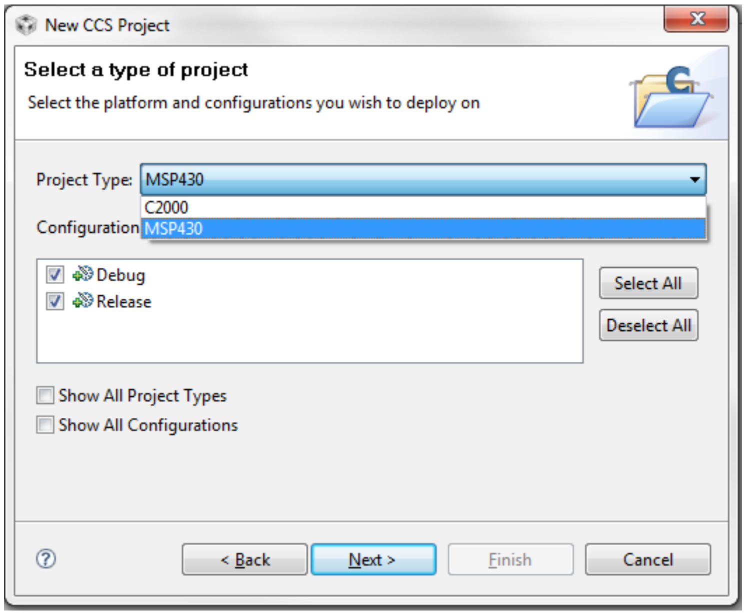
Create a new CCS Project.

Project name: Blink_led

[✓] Use default location

Location: C:/Users/Hrishikesh/Desktop/Lab_0/Blink_led

Figure 4: Create New Project - Project Name



New CCS Project

Select a type of project

Select the platform and configurations you wish to deploy on

Project Type: MSP430

C2000
MSP430

Configuration

[✓] Debug
[✓] Release

[ ] Show All Project Types
[ ] Show All Configurations

Figure 5: Create New Project - Choose the Configuration

Leave the next screen as it is. As this is our first project, we will not be referencing any other projects and use the full C/C++ indexer. Click *Next* and proceed to *CCS Project Settings*.

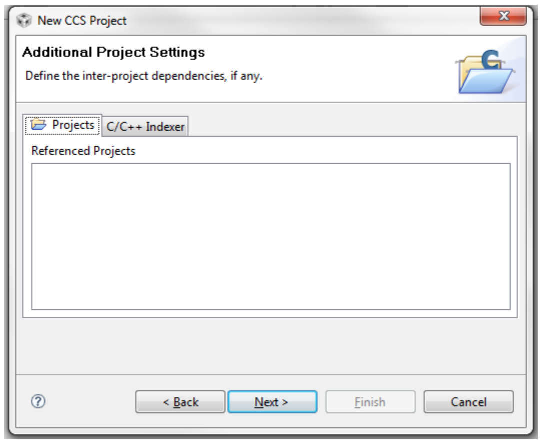
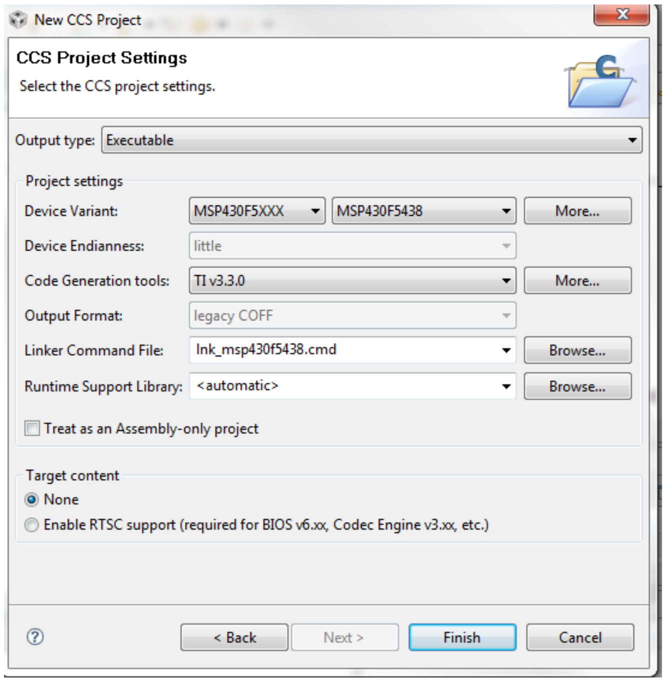


Figure 6: Create New Project - Project Reference

Choose the Micro-controller variant. The MCU we will be using is the *MSP430F5438*. Leave the rest as defaults. Your settings should match the values in Figure 7. Hit *Finish*. The project environment is up.


New CCS Project

CCS Project Settings

Select the CCS project settings.

Output type: Executable

Project settings

Device Variant: MSP430F5XXX MSP430F5438

Device Endianness: little

Code Generation tools: TI v3.3.0

Output Format: legacy COFF

Linker Command File: lnk_msp430f5438.cmd

Runtime Support Library: <automatic>

Treat as an Assembly-only project

Target content

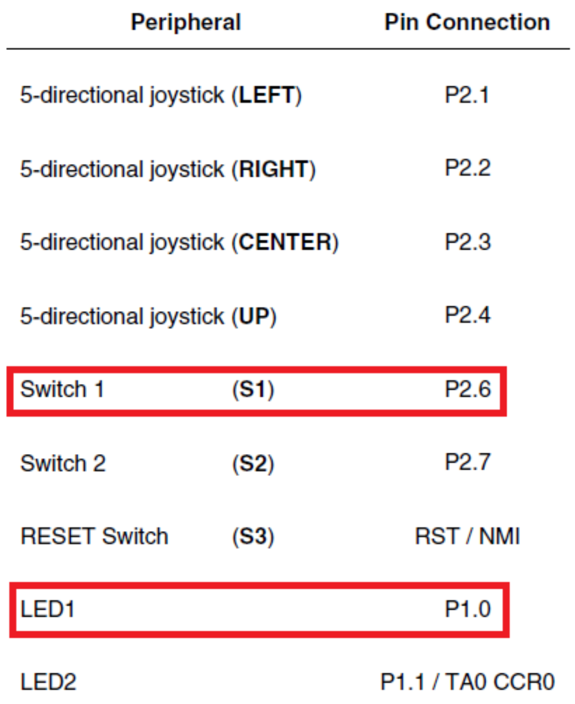
None

Enable RTSC support (required for BIOS v6.xx, Codec Engine v3.xx, etc.)

Figure 7: Create New Project - Choose Device

1.2 Programming the MSP-EXP430F5438 MCU

Now we are ready to start coding. For our program, a LED should glow when a switch is pressed and should stop glowing when the switch is released. You will notice that there are 2 LEDs and 3 switches on the board. To begin with, we need to know which GPIOs of the micro-controller control the LEDs and switches. A GPIO is a General Purpose In/Out Pin. The board comes with peripherals connected to some GPIO of the MCU. Check chapter 4 of the MSP-EXP430F5438 Experimenter Board User's Guide. It details the connections of the peripherals to the ports of the MCU. For this lab, we want to use a switch and a light emitting diode.



Peripheral		Pin Connection
5-directional joystick (**LEFT**)		P2.1
5-directional joystick (**RIGHT**)		P2.2
5-directional joystick (**CENTER**)		P2.3
5-directional joystick (**UP**)		P2.4
Switch 1	(**S1**)	P2.6
Switch 2	(**S2**)	P2.7
RESET Switch	(**S3**)	RST / NMI
LED1		P1.0
LED2		P1.1 / TA0 CCR0

Figure 8: Pin Map for Switches and LEDs - Experimenter Board (taken from Board User's Guide)

In Figure 8, we can see that the switch S1 is connected to the GPIO 2.6 and the LED is connected to P1.0. Therefore, the idea is to sample the signal from the switch through port 1.0 and depending upon the value of the signal, control the value which is written out from port 2.6. That seems very straightforward. But, GPIOs are "general purpose". Therefore, they may have multiple functions multiplexed onto the same pin. This in turn,requires the programmer to configure the direction, the pull-up/pull-down resistors etc. depending upon the choice of the function. In other words, we have to *configure* the MCU GPIOs in a particular way to complete the connection with the LEDs or any other peripheral device on the experimenter board.

This subsection is Optional

Look at the MSP430F5438 Data Sheet. It is a 99-page document. We are only interested in the functions multiplexed onto the GPIOs of the MCU. One can infer this information from either

1. Pin Designation diagram (For people familiar with MCU programming and data sheets).
2. Terminal Functions Table

I prefer the Terminal Functions Table. In the table, search for the ports P2.6 and P1.0 which are of immediate interest to us. You will find that the port P2.6 also serves as an Auxiliary Clock (ACLK). The choice of functionality on this GPIO, therefore depends upon some *configuration*. Similarly for P1.0.

Required: Configuration Registers

The way that we *configure* a GPIO to a certain functionality is through *configuration registers*. Open up the MSP430 MCU User's Guide. The User's Guide is what you need to write software for the MCU and thus for the board. We will be using this document extensively for the labs to come. Go to Chapter 10: Digital I/O. This chapter provides all the information that you require to use a GPIO. Read Section 10.2. It is highly recommended that you peruse through the entire chapter before continuing with this document.

Section 10.4 describes the registers used for configuring different GPIO ports and their initial states.

Programming

Ok, so let us start writing our program. Open a new C source file as shown in Figure 9.

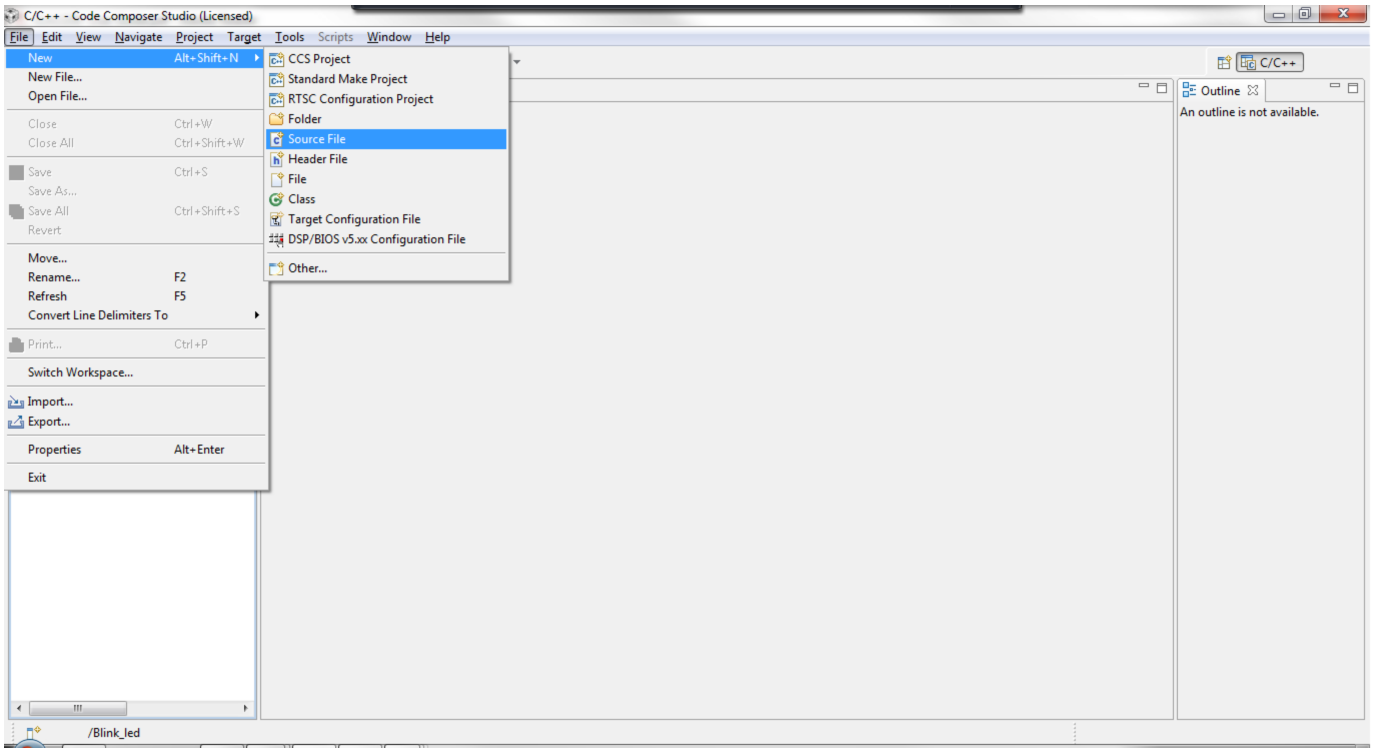
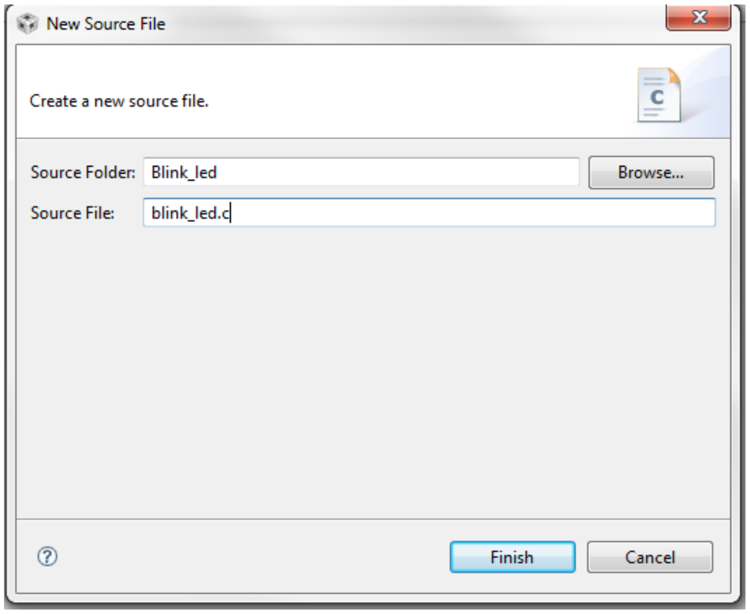


Figure 9: Open new C source file

Give a file name with the .c extension as in Figure 10.

Include the header file `msp430f5438.h`. The header file contains all the registers in our MCU which may be used in the program. The following program is for demonstration of the flow. But it is not the solution. We shall build it, run it on the board and then proceed to debug it. This document will only outline some of the methods you may use to debug using the CCS and will not provide the full solution! Copy the following code into your `blink_led.c` file.

```
#include "msp430f5438.h"

void main()
{
  //Switch S1
  P2SEL &= ~0x40; //I/O Function
  P2DIR &= ~0x40; //P2.6 as input
  P2REN |=  0x40; //Enable pull resistor
  P2OUT |=  0x40; //Enable Pull-Up resistor

  //LED 1
  P1SEL &= ~0x01; //I/O Function
  P1DIR |=  0x01; //P1.0 as output
  P1DS  |=  0x01; //Enable Full drive strength

  if(P2IN & 0x40) //If P2.6 is high
    P1OUT |= 0x01;
  else //If P2.6 is low
    P1OUT |= 0x00;
}
```



Create a new source file.

Source Folder: Blink_led

Source File: blink_led.c

Figure 10: Name C file

Save the file. To build the code go to *Project→Build All* or hit *Ctrl+B*. Your CCS window should look something like Figure 11.

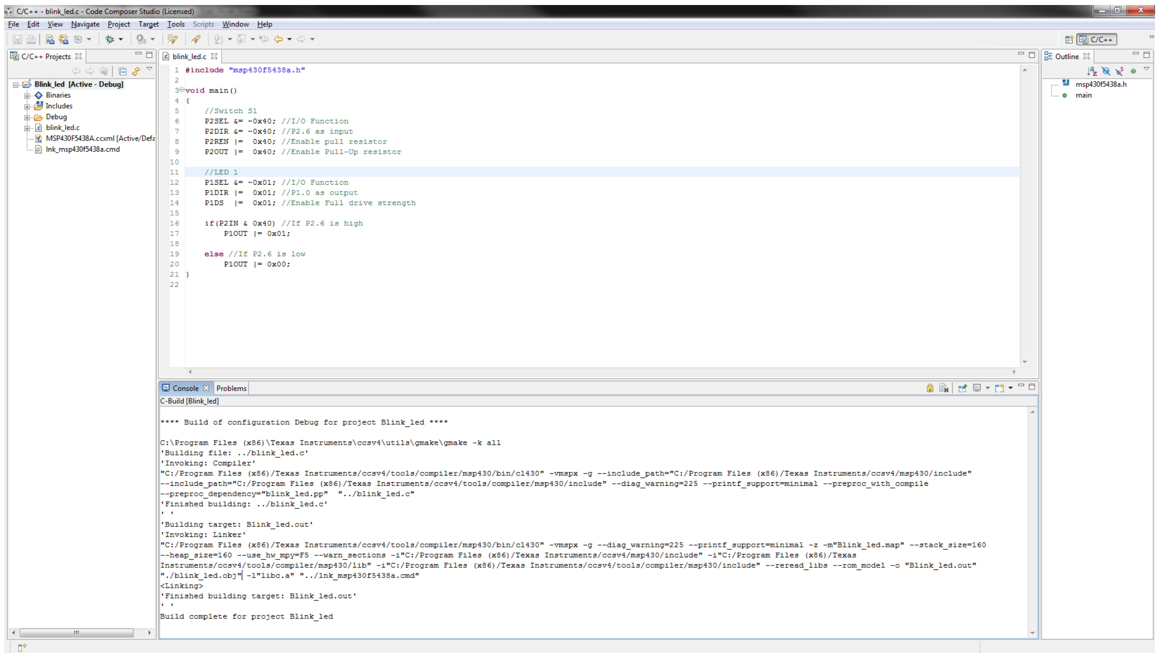
Debug

Now we have to run it on the board. Look at Figure 12 to run the debug session on the board.

You can see the program being loaded on a new pop-up window and subsequently the CCS switches into Perspective Window Mode. Go ahead and hit the play button on the debug panel, Figure 13, to see what the program does on the board. Try pressing the switch S1.

You would have seen that the LED is glowing all the time. There are multiple mistakes in the program. Pause the run and hit the *restart* button on the debug panel. As promised, the program contains bugs and is not the solution. Let us insert some break-points and start to debug. Insert break points at the lines 17, 20, and 21 - also shown in Figure 14 by double clicking on the vertical bar to the left of the code. Now, we would like to view the values of the registers we are using. Enable the register view by going to *Window→Show View→Registers*. I rearranged my Perspective View so that it becomes easy for me to view the code, registers and memory values. You can rearrange your window to suit your needs. Open the tree for *Port_1_2*.

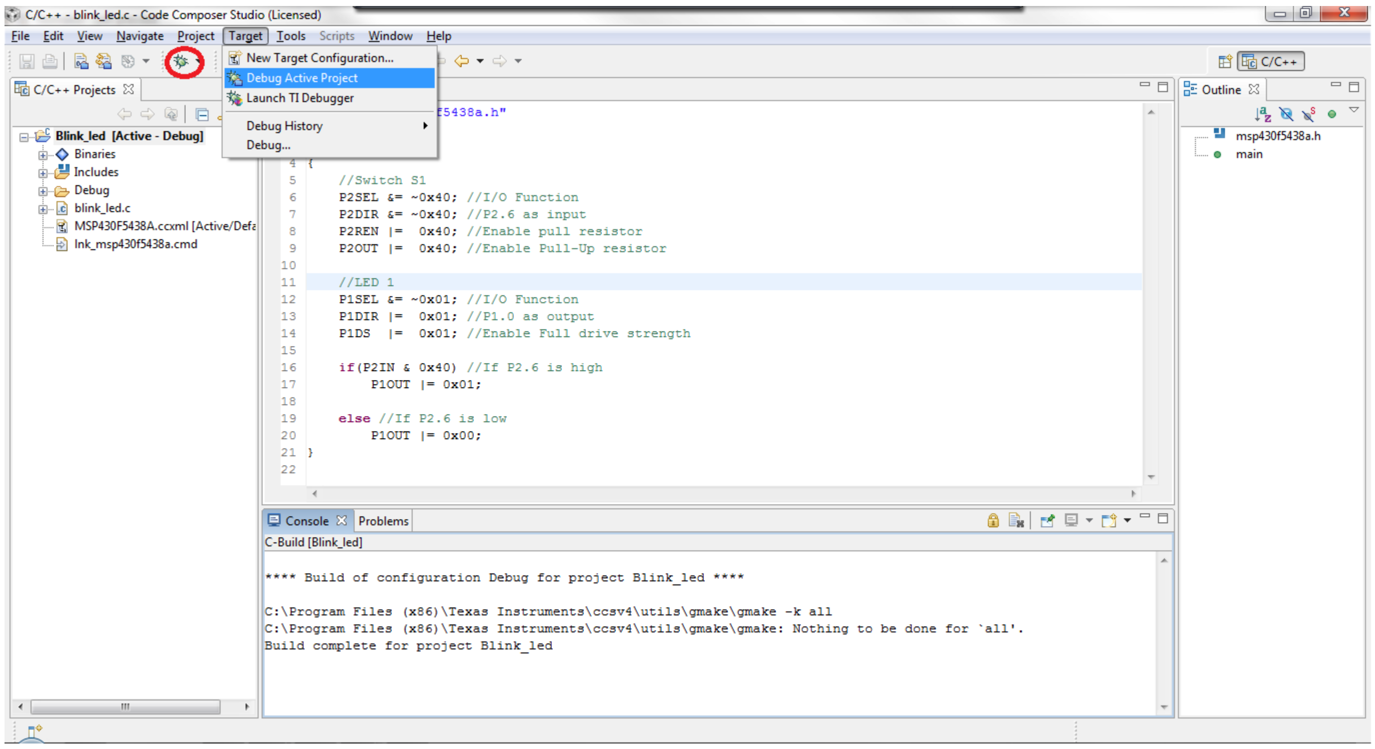
Now do a step by step debug and watch the values of registers. You will hit the end of the code pretty soon which leads us to the most obvious error. You may also want to see the memory space if you are using any variables inside the code. You can open up the Memory View in the same way as the Register View. The example program is a simple one and thus does not use any variables. There are some redundant lines in the given program which you can remove after understanding section 10.2 and going through 10.4.



```
#include "msp430f5438a.h"

void main()
{
    //Switch S1
    P2SEL &= ~0x40; //I/O Function
    P2DIR &= ~0x40; //P2.6 as input
    P2REN |=  0x40; //Enable pull resistor
    P2OUT |=  0x40; //Enable Pull-Up resistor

    //LED 1
    P1SEL &= ~0x01; //I/O Function
    P1DIR |=  0x01; //P1.0 as output
    P1DS  |=  0x01; //Enable Full drive strength

    if(P2IN & 0x40) //If P2.6 is high
        P1OUT |= 0x01;

    else //If P2.6 is low
        P1OUT |= 0x00;
}
```

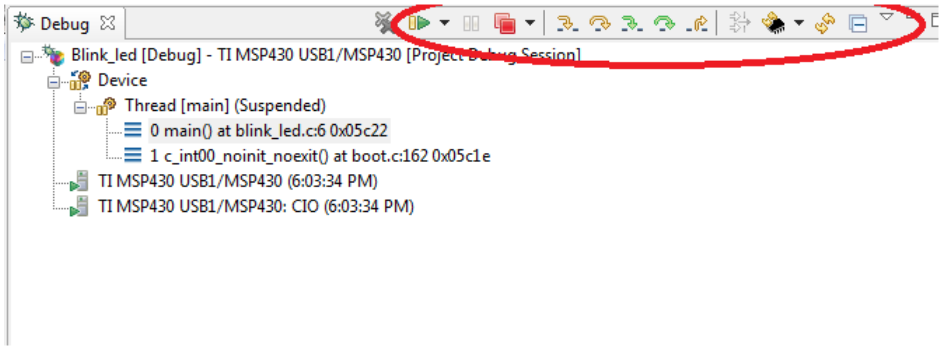
```
**** Build of configuration Debug for project Blink_led ****

C:\Program Files (x86)\Texas Instruments\ccsv4\utils\gmake\gmake -k all
'Building file: ../blink_led.c'
'Invoking: Compiler'
"C:/Program Files (x86)/Texas Instruments/ccsv4/tools/compiler/msp430/bin/cl430" -vmspx -g --include_path="C:/Program Files (x86)/Texas Instruments/ccsv4/msp430/include"
--include_path="C:/Program Files (x86)/Texas Instruments/ccsv4/tools/compiler/msp430/include" --diag_warning=225 --printf_support=minimal --preproc_with_compile
--preproc_dependency="blink_led.pp"  "../blink_led.c"
'Finished building: ../blink_led.c'
' '
'Building target: Blink_led.out'
'Invoking: Linker'
"C:/Program Files (x86)/Texas Instruments/ccsv4/tools/compiler/msp430/bin/cl430" -vmspx -g --diag_warning=225 --printf_support=minimal -z -m"Blink_led.map" --stack_size=160
--heap_size=160 --use_hw_mpy=F5 --warn_sections -i"C:/Program Files (x86)/Texas Instruments/ccsv4/msp430/include" -i"C:/Program Files (x86)/Texas
Instruments/ccsv4/tools/compiler/msp430/lib" -i"C:/Program Files (x86)/Texas Instruments/ccsv4/tools/compiler/msp430/include" --reread_libs --rom_model -o "Blink_led.out"
"./blink_led.obj" -l"libc.a" "../lnk_msp430f5438a.cmd"
<Linking>
'Finished building target: Blink_led.out'
' '
Build complete for project Blink_led
```

Figure 11: Build completed


C/C++ - blink_led.c - Code Composer Studio (Licensed)
File Edit View Navigate Project Target Tools Scripts Window Help
New Target Configuration...
Debug Active Project
Launch TI Debugger
Debug History
Debug...
C/C++ Projects
Blink_led [Active - Debug]
Binaries
Includes
Debug
blink_led.c
MSP430F5438A.ccxml [Active/Defa
lnk_msp430f5438a.cmd
Outline
msp430f5438a.h
main
Console
Problems
C-Build [Blink_led]

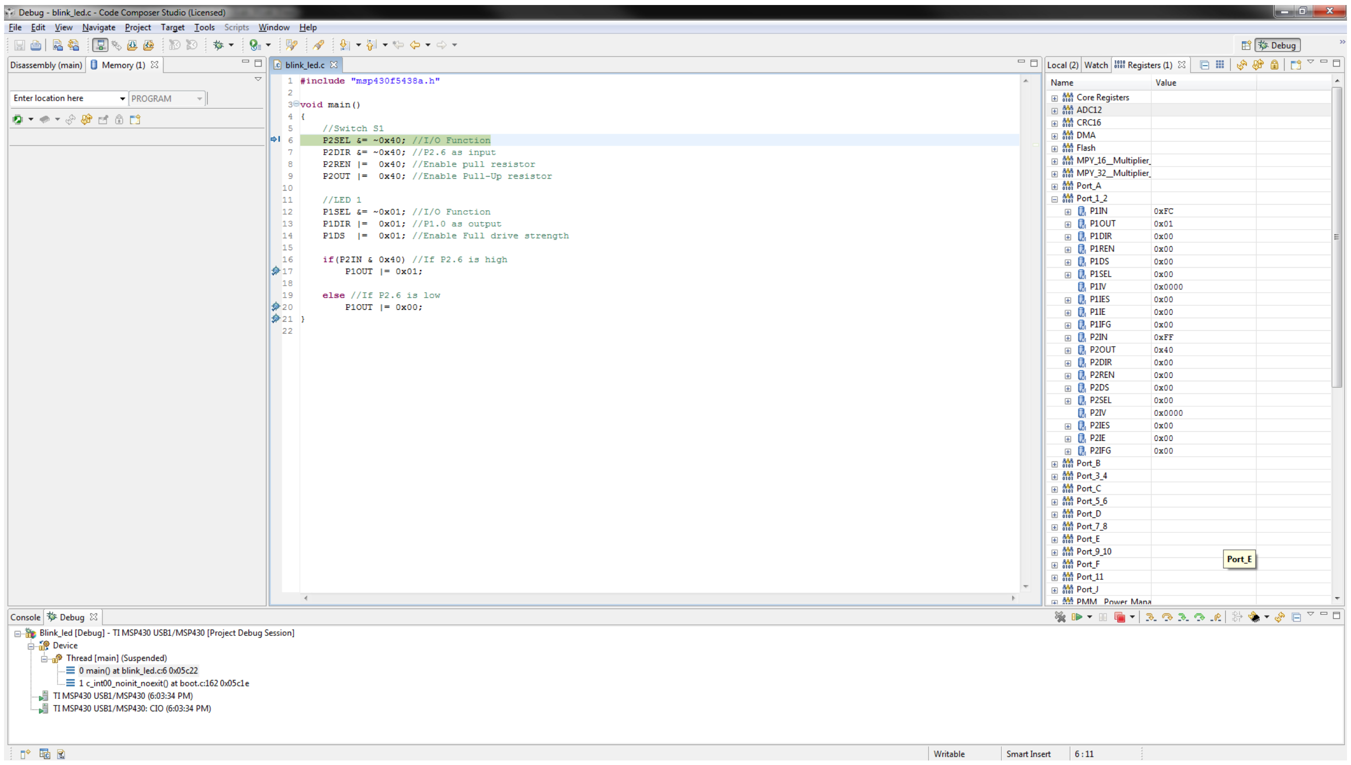
```
**** Build of configuration Debug for project Blink_led ****

C:\Program Files (x86)\Texas Instruments\ccsv4\utils\gmake\gmake -k all
C:\Program Files (x86)\Texas Instruments\ccsv4\utils\gmake\gmake: Nothing to be done for `all'.
Build complete for project Blink_led
```

Figure 12: Run the debug session on the board



```
Blink_led [Debug] - TI MSP430 USB1/MSP430 [Project Debug Session]
  Device
    Thread [main] (Suspended)
      0 main() at blink_led.c:6 0x05c22
      1 c_int00_noinit_noexit() at boot.c:162 0x05c1e
  TI MSP430 USB1/MSP430 (6:03:34 PM)
  TI MSP430 USB1/MSP430: CIO (6:03:34 PM)
```

Figure 13: Debug buttons



```c
#include "msp430f5438a.h"

void main()
{
    //Switch S1
    P2SEL &= ~0x40; //I/O Function
    P2DIR &= ~0x40; //P2.6 as input
    P2REN |=  0x40; //Enable pull resistor
    P2OUT |=  0x40; //Enable Pull-Up resistor

    //LED 1
    P1SEL &= ~0x01; //I/O Function
    P1DIR |=  0x01; //P1.0 as output
    P1DS  |=  0x01; //Enable Full drive strength

    if(P2IN & 0x40) //If P2.6 is high
        P1OUT |= 0x01;

    else //If P2.6 is low
        P1OUT |= 0x00;
}
```

Name	Value
Core Registers	
ADC12	
CRC16	
DMA	
Flash	
MPY_16__Multiplier_	
MPY_32__Multiplier_	
Port_A	
Port_1_2	
P1IN	0xFC
P1OUT	0x01
P1DIR	0x00
P1REN	0x00
P1DS	0x00
P1SEL	0x00
P1IV	0x0000
P1IES	0x00
P1IE	0x00
P1IFG	0x00
P2IN	0xFF
P2OUT	0x40
P2DIR	0x00
P2REN	0x00
P2DS	0x00
P2SEL	0x00
P2IV	0x0000
P2IES	0x00
P2IE	0x00
P2IFG	0x00
Port_B	
Port_3_4	
Port_C	
Port_5_6	
Port_D	
Port_7_8	
Port_E	
Port_9_10	
Port_F	
Port_11	
Port_J	
PMM_Power_Mana	

```
Blink_led [Debug] - TI MSP430 USB1/MSP430 [Project Debug Session]
  Device
    Thread [main] (Suspended)
      0 main() at blink_led.c:6 0x05c22
      1 c_int00_noinit_noexit() at boot.c:162 0x05c1e
  TI MSP430 USB1/MSP430 (6:03:34 PM)
  TI MSP430 USB1/MSP430: CIO (6:03:34 PM)
```

Figure 14: Adding Breakpoints, Registers and Memory

2 Problem Statement

Implement a simple XOR system which does the following for the TI MSP430F5438 Experimenter Board. Switches are considered to be On when they are pushed.

S1	S2	LED1	LED2
Off	Off	Off	Off
Off	On	Off	On
On	Off	On	Off
On	On	Off	Off

Table 1: LED Glowing Pattern.

2.1 Submission

We will use the Blackboard System (http://www.itap.purdue.edu/tlt/blackboard/index.cfm) for the submission throughout our course. Please zip everything before submission and only upload the package. Do not upload every single file separately. The content of the package should be properly organized. Particularly, you should put everything under a directory named *username*`_lab0`, where *username* is your Blackboard system login id. Under the directory, include

1. `blink_led.c`: the bug-free blink-LED program;
2. `pattern.c`: the implementation of the XOR system.