ECGR 4101/5101, Fall 2015: Lab 1

Baby's First Embedded Systems Project

Version 1.2 – 8/25/2015

Learning Objectives:

This lab will introduce basic embedded systems programming concepts by familiarizing the user with an embedded programming IDE (Integrated Development Environment), writing C code for embedded devices, compiling and uploading code, and maybe even a little debugging.

The Development Board: TI MSP430 Launchpad



The board we will be using for this lab is TI's MSP430 Launchpad. The Launchpad is the name for a line of development boards produced by TI; we are specifically using the MSP-EXP430G2. The MSP-EXP430G2 is also compatible with multiple microcontrollers, but for this lab, we'll be using the MSP430-G2553. That's a lot of information, but it's necessary to know so the programming software knows how the code is being uploaded (through the programmer on the MSP-EXP430G2) and for which processor the code is being compiled (The MSP430-G2553).

One of the main features of a microcontroller vs. a general purpose computer is the addition of peripherals. Peripherals are additional circuitry provided to the microprocessor to allow it to interact with the "outside" world. Peripherals can include anything from analogue to digital converters and timers to serial ports and internal temperature sensors. These peripherals are accessed through special locations in memory. These registers in memory allow the microcontroller to configure (such as set a comparison value in a timer), write (send a byte through a serial port), or read (obtain a voltage from an analogue to digital controller) from a peripheral.

The MSP430-G2553 is a small, 16 bit microcontroller, operating at 3.3 Volts. It has a few peripherals including 2 16-bit timers, a 10-bit ADC (analogue/digital converter), and a serial communication interface. There are other peripherals available; to find out what, you only need to consult the datasheet! The datasheet will tell you what peripherals your microcontroller is packing and how to interface with them (However, because there might be multiple types of compilers, it is unlikely the datasheet has any example code). A link to the MSP430-G2553 datasheet can be found here: http://www.ti.com/lit/ds/symlink/msp430g2553.pdf

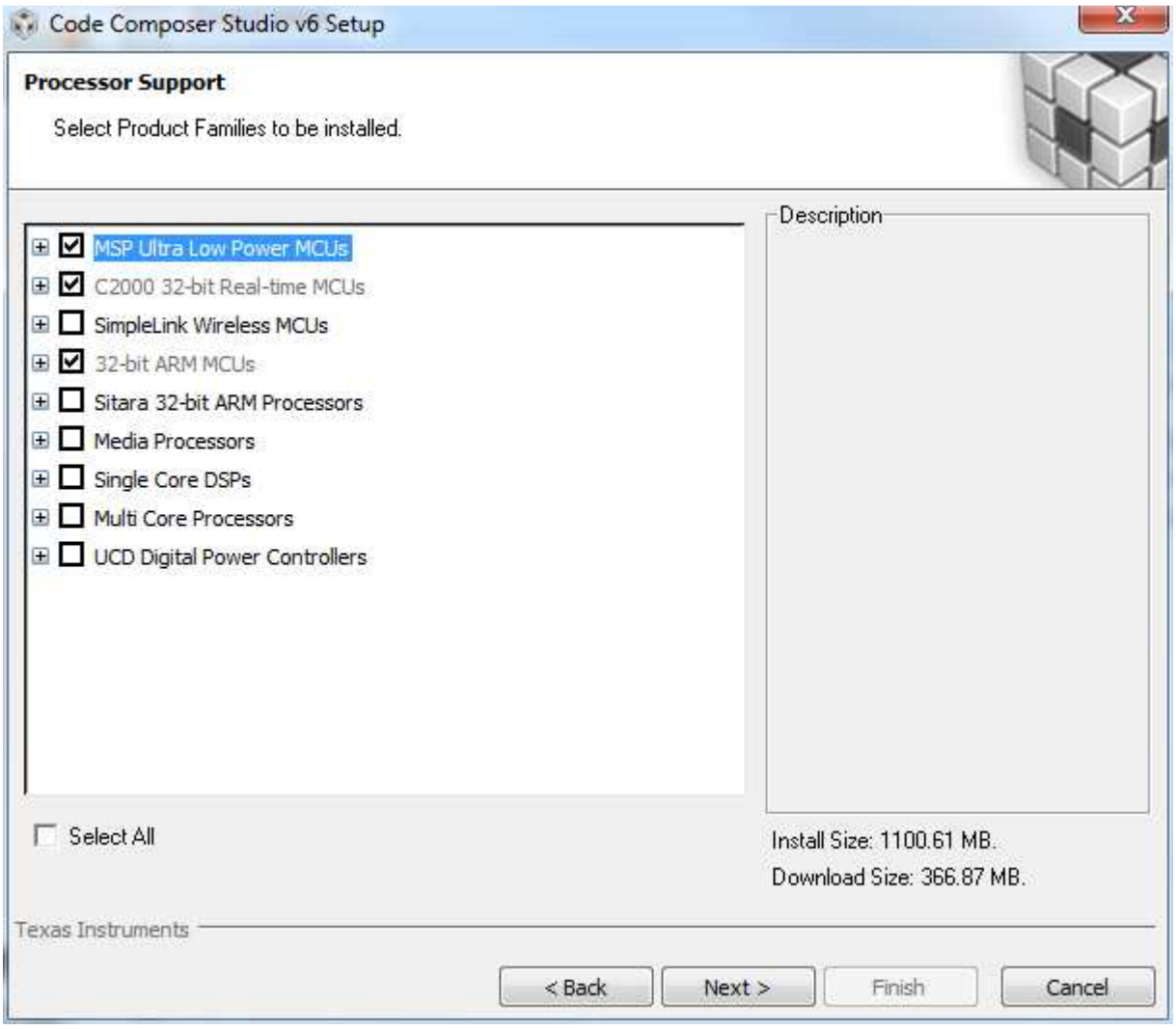
Installing the IDE: Code Composer Studio

You may use the PCs in EPIC 2130 or 2148 or your own PC to do this lab experiment. If you want to work on lab assignments on your own PC, then you will need to install the IDE to write, compile, and upload code. What's an IDE? IDE stands for Integrated Development Environment. Without an IDE, you would have to write the code for your board, run it through a separate compilation program with parameters for your target microcontroller, and finally send it through a program to upload through the

programmer hardware. An IDE is a single program that does all of that for you, plus provide syntax highlighting, automatic linking of software libraries, and even provide debugging options!

The MSP430 Launchpad can be programmed through a few different IDEs. We'll be using TI's code composer studio. The link to download it can be found here: http://processors.wiki.ti.com/index.php/Download_CCS

After registering with TI and answering a bunch of obnoxious questions, you will be able to download CCS (Code Composer Studio). When installing, most default configurations will be fine, but be sure to select the MSP Ultra Low Power MCUs option under "Processor Support". This will ensure that the compiler for the MSP430-G2553 is installed.

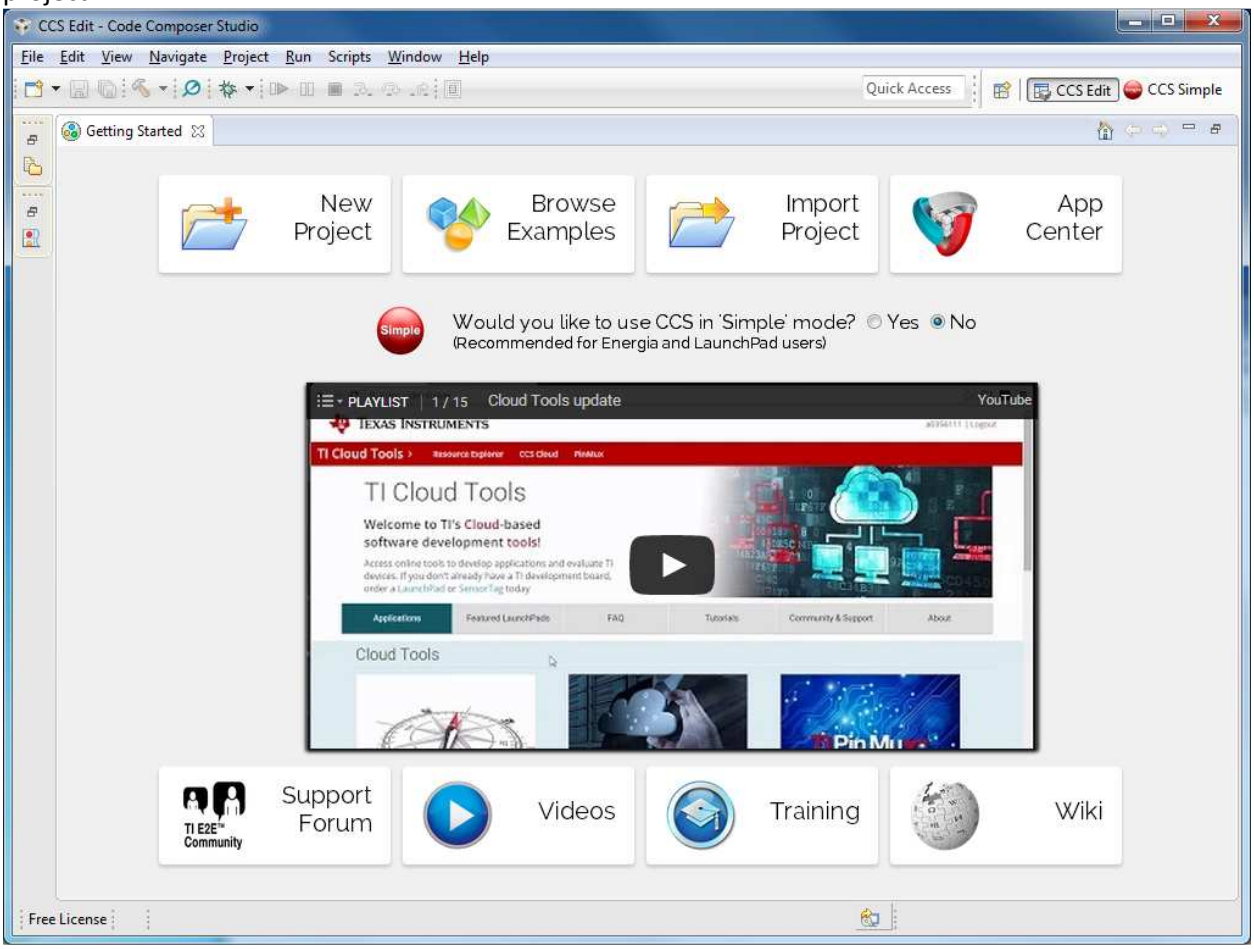


Note: I checked a few other things, but you will only need the MSP

Default settings for the rest of the installation process should be fine.

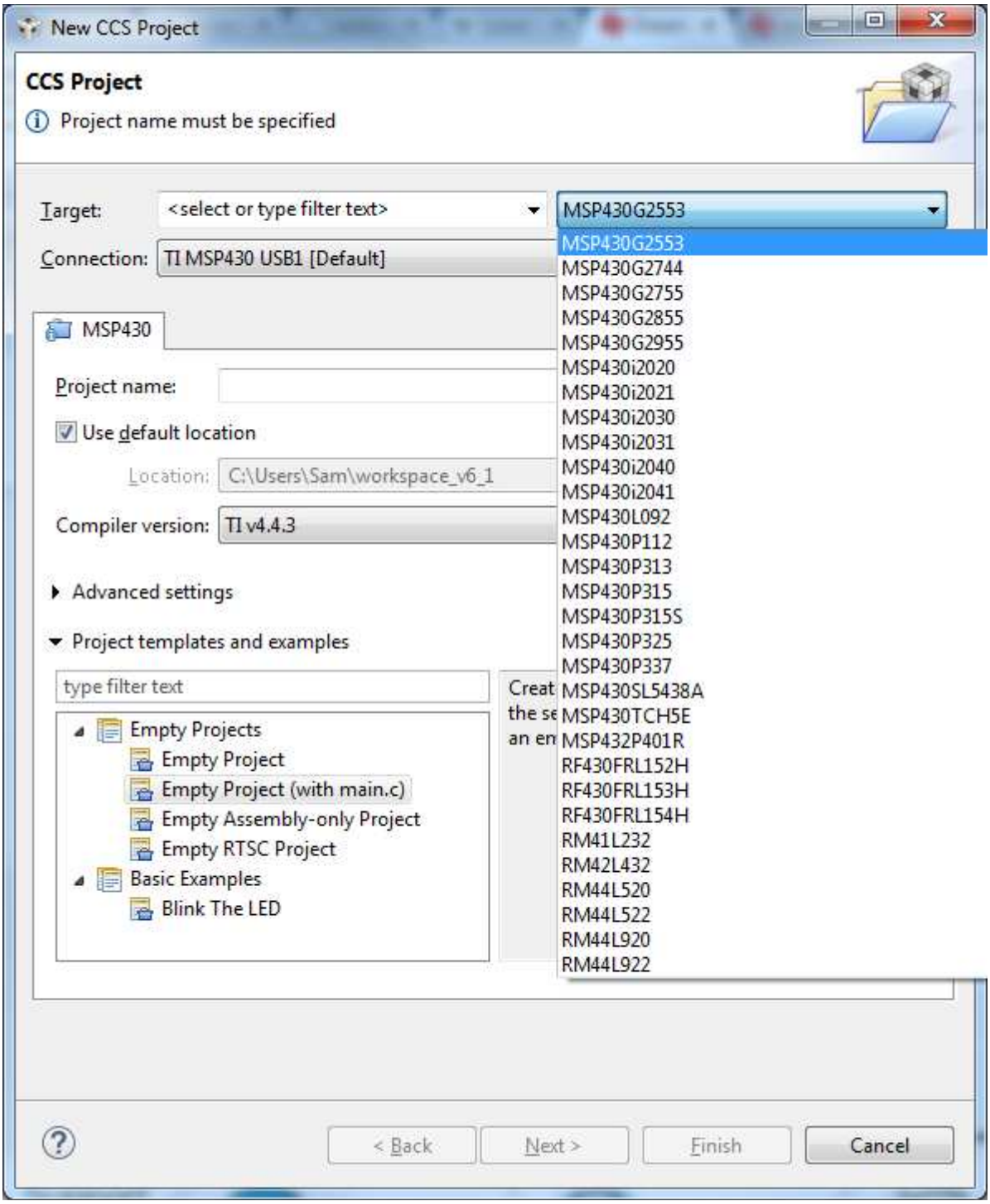
Creating a New Project: CCS

After finishing the installation process, launch CCS. From this screen we will begin by creating a new project.



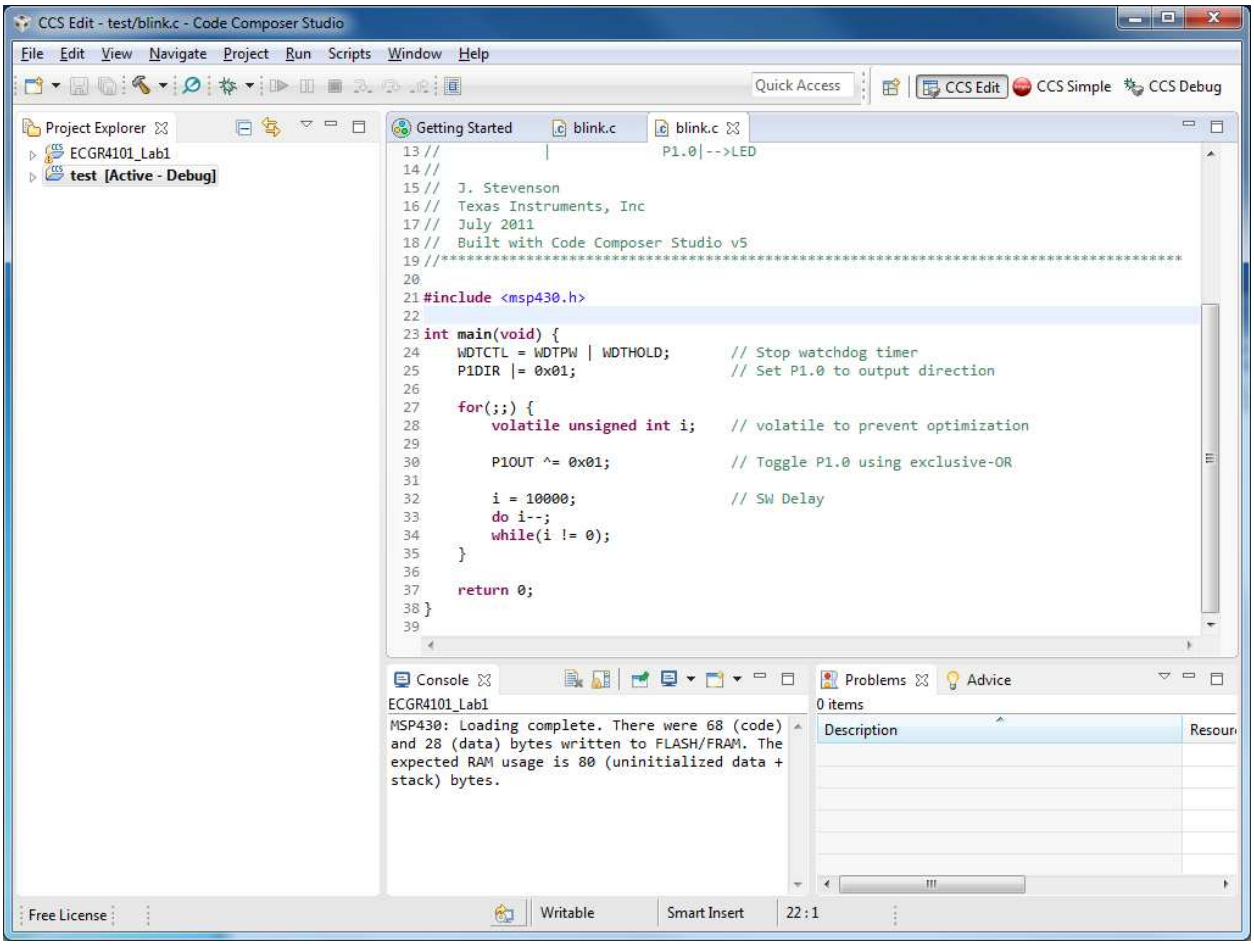
When you click “New Project” a configuration window will appear. Be sure to select your microcontroller from the drop down menu as shown below. This sets the compiler to build code for the selected microcontroller’s architecture! In an IDE the software library associated with that microcontroller is imported as well. Each microcontroller has a different memory map, and the software library allows us to call memory locations with a name rather than a confusing address number.

After you have selected your microcontroller, you should plug up your Launchpad board via USB cable and test the connection. Give your computer a moment to install the driver, then press the “Identify” button to confirm your board is connected. This communicates with the circuity at the top section of the Launchpad above the “Emulation” line which controls uploading code and emulation for software debugging. Give your project and a name, select the “Blinking LED” example from the templates and examples section and hit “Finish”.



Analyzing the Example: Blinking LED Example

As “Hello World” is to beginner programming tutorials, blinking an LED is to embedded systems tutorials. This program will toggle one of the two on board LEDs on and off as the program runs. Let’s break down the example code.




```
#include <msp430.h>
```

This is the first line of code. There are other lines before it in green, but they are only comments, and are ignored by the compilation process. They do not contribute to the function or size of the code in any way, they are only there to leave notes behind for yourself and others to explain what's happening in your code. This line of code however also doesn't go through the compiler. This line is a pre-processor command, which is denoted by the # sign in front of the line. The preprocessing phase happens before the compiler runs, and in this case is being used to include a library. This is the library that provides interfaces to all of the peripherals and features of the MSP430-G2553!

```
int main(void) {
```

This line is the beginning of the main function. As the name seems to imply, this is the function that is called first in a C program; where the execution begins after everything else is loaded. Functions always have a return type, sometimes have parameters passed to them, and have their contents contained by a set of curly braces. In this case, "void" is passed to the pain function (which is nothing), and "int" is returned. In the context of imbedded systems, returning a value from main is arbitrary, as the computer shuts off afterwards. A return value from and a parameter being passed to main() really only is functional when it is running on top of an operating system, such as Windows or Linux.

```
WDTCTL = WDTPW | WDTHOLD;        // Stop watchdog timer
                                 // Set P1.0 to output direction
```

This probably looks a tad confusing to a new embedded systems programmer, and would look even crazier without the <msp430.h> library! This line of code disables the watchdog timer, a peripheral used to reset the microcontroller when the code gets stuck. A useful feature, but not something we want to deal with when we're simply blinking an LED. Each of these ALL CAPS words are defined in the library and write a certain bit combination to the register in memory which hold the watchdog timer's settings.

Also, notice that lines of C code are terminated in a semicolon.

```
P1DIR |= 0x01;                                  // Set P1.0 to output direction
```

As you can see from the comment, this sets Port 1 Pin 0 on our board as an output. This is one of the first peripherals we will be using on the MSP430. It's part of the general purpose input/output, or GPIO. Most microcontrollers have GPIO, and most GPIO have 4 associated registers. One register for data direction (input or output), a register for writing to the pin, a register for reading to a pin, and a register for enabling pull-up resistors. P1DIR is the named shortcut for the memory location mapped to the data direction register of Port 1. Each bit in the byte is attached to an individual pin on the microcontroller. As you can see on the side of the board, each pin has a label next to it, indicating the port number, and the pin, or bit, number in the port's byte.

You will also notice the "|=" notation followed by the "0x01;". "|=" is an or equals operator. It is the equivalent of "P1DIR = P1DIR | (or 'OR') 0x01; ". The 0x01 sets the first bit in the P1DIR register to 1, which sets bit 0 as an output. A 0 is an input, and a 1 is an output. It is good practice to use |= when setting a particular bit in a register because it does not disturb the other bits. This technique is known as "bit masking". Think about it, if the register already contained 0x20, using only "=" would overwrite the entire register as 0x01. By using "|=", we take the current contents, 0x20 and OR it with 0x01, giving us 0x21;

```
for(;;) {
```

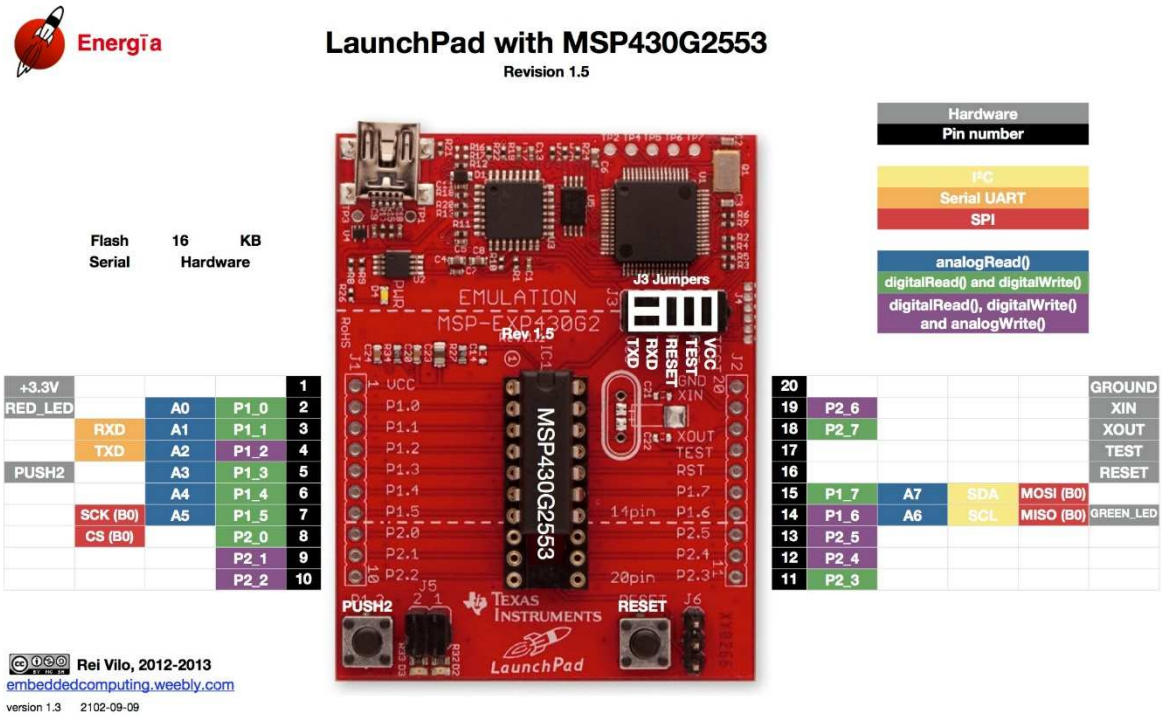
This for loop with no contents is an infinite loop. In embedded systems, we rarely want our running program to terminate. The primary function of our code will go inside of this loop to be repeated as long as the device remains powered. More commonly, a while(1) loop will be used, but it makes no difference.

```
volatile unsigned int i;  // volatile to prevent optimization
```

This line declares a variable i. It is a volatile unsigned int i, which means it cannot be negative, has to be an integer, and volatile. Volatile is a data type used to prevent the compiler optimization from "optimizing". Since we are using it in a delay, compiler optimization may remove this line as it seemingly does nothing. It's not terribly important at this point in time, but for further reading, see this webpage.

```
P1OUT ^= 0x01;                          // Toggle P1.0 using exclusive-OR
```

Similar to the data direction line, here we are using another bit masking shortcut to write to a peripheral register. P1OUT is the GPIO Port 1 Output register. A 1 here writes a "high" signal to the corresponding pin, and a 0 writes a "low" signal. The voltages depend on the operating voltage of your microcontroller, but in this case a high is 3.3 volts and a low is 0 volts.



The "^=" operator is an XOR equals. This is the same as "P1OUT = P1OUT ^ (or 'XOR') 0x01;" XOR is used for toggling a bit. Each time XOR equal is called, the 1's in the byte will be toggled. As you can see at the bottom of the Launchpad, LED1 is attached to Port 1 Pin 0.

```
i = 10000;          // SW Delay
do i--;
while(i != 0);
```

These lines of code are used to create a small delay, so we can see the toggling of the led. The i variable is assigned 10,000, and the do/while statement will decrement i until i == 0.

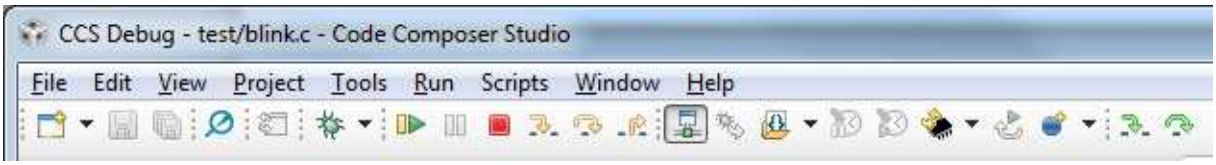
```
        }

        return 0;
}
```

These lines of code close up our infinite loop and main function, and the return statement is necessary for any function which does not have the "void" type. Again, main doesn't return anything anywhere, so it's just there to satisfy the syntax requirements of the compiler. This line is also outside of the infinite loop, so it will never be reached (The compiler will throw a warning about this, but you can ignore it).

Compiling, Uploading, and Running

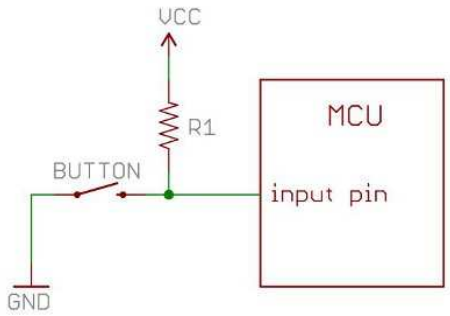
Now that we understand what the example code is doing, it's time to compile and upload. To compile, click the "hammer" icon on the toolbar or under "Project" select "Build All". Once the code is built and you've received your obligatory warning about the return statement being unreachable, you can run the code. In CCS, code is ran by clicking the little "bug" icon on the toolbar. This begins the debugging process. Here, you can step through your code, view register contents, and other useful things. Right now, we just want to run the code, so click the green triangle button to run. You may get a few pop up indicating some background processes are running, so don't freak out, it's normal.



Taking It Too Far: Your Assignment

For this lab assignment, we are going to modify our blinking LED code. We are going to change the code such that LED2 blinks only when the button labeled S2 is held down. However, if you are unfortunate enough to be a graduate student, you will have a different set of requirements. Instead, graduate students are required to pause the blinking when the button is quickly pressed, and change the blinking LED from LED1 to LED2 when the button is held down for 1 to 2 seconds.

The only peripheral you will need to use is the GPIO discussed above. To read input from the button, you must enable the pull up resistors for the pin connected to the button. When a pull up resistor is enabled, the following circuit is created:



This means the input pin will read 1 when the button is not pressed because of the high impedance input of the microcontroller causing a very, very low voltage drop across the pull-up resistor, and 0 when the button is pressed because the button is grounded. To get the names of the registers for the GPIO, you can consult the datasheet. For convenience, I the table below has the information from the datasheet:

Table 15. Peripherals With Byte Access (continued)

MODULE	REGISTER DESCRIPTION	REGISTER NAME	OFFSET
Port P1	Port P1 selection 2	P1SEL2	041h
	Port P1 resistor enable	P1REN	027h
	Port P1 selection	P1SEL	026h
	Port P1 interrupt enable	P1IE	025h
	Port P1 interrupt edge select	P1IES	024h
	Port P1 interrupt flag	P1IFG	023h
	Port P1 direction	P1DIR	022h
	Port P1 output	P1OUT	021h
	Port P1 input	P1IN	020h

BUG ALERT! Sometimes the pull up enable register is cleared after reading the status of pin. For safety's sake, just be sure to re-write the pull up enable register after each read.

Requirements for Undergraduates:

- LED1 only blinks when button S2 is pressed.
- LED1 blinks at a frequency easily observable by the human eye
- Code must be written in C on CCS

Requirements for Graduates:

- LED1 blinks by default
- Blinking is paused when button S1 is quickly pressed (quickly means less than 1 second)
- Blinking is resumed when button S1 is quickly pressed when paused
- When button S1 is held down for 1 to 2 seconds LED1 stops blinking and LED2 begins blinking. The configuration is reversed when pressed for 1 to 2 seconds again. The blinking should not be paused after the long button press.
- LED1 and LED2 blink at a frequency easily observable by the human eye
- Code must be written in C on CCS

To Demo and Submit:

To submit, have the demonstration sheet below printed off. Demonstrate your working (or partially working) code to the TA. After grading, the TA will take your demonstration sheet and save for grading.

Embedded Systems Lab Demonstration Validation Sheet

This sheet should be modified by the student to reflect the current lab assignment being demonstrated

Lab Number:	Lab 1 – Blinking LEDs
Team Members	Team Member 1: Team Member 2:
Date:	

Lab Requirements

REQ Number	Objective - Undergrads	Self-Review	TA Review
1	The LED Properly Blinks at an observable rate		
2	The LED only blinks when button S1 is held down		

REQ Number	Objective - Grads	Self-Review	TA Review
1	LED 1 Blinks at an observable rate		
2	Blinking pauses/unpauses when button is quickly pressed		
3	LED 1 and LED 2 alternate blinking when button S1 is held down for longer than 1~2 seconds.		