ME 461 Laboratory #1
Digital I/O

Goals:

1. Learn the basics of controlling digital input and output port pins.
2. Use digital input port pins to read the state of switches or interrupt the processor when the switches are pressed.
3. Use digital output port pins to turn on and off LEDs.
4. Control the timing of turning on and off LEDs to create a pattern.

NOTE: When configuring the port pins for digital I/O, follow this progression.

Inputs:

1. Set PxSEL.
2. Set PxDIR.
3. Set PxREN (if pull-up/down resistor is required).
4. Set PxOUT (if pull-up/down resistor is enabled).
5. Read PxIN to find the status of the pins setup as inputs.

Note: items 6-8 are required only when pin interrupts are desired.

6. Set PxIES.
7. Clear PxIFG.
8. Set PxIE.

Outputs:

1. Set PxSEL.
2. Set PxDIR.
3. Set PxOUT.

Also, when setting register contents, use bitwise AND & OR assignments to ensure only the intended bits are modified. See the document "C Review" on the course website for more information.

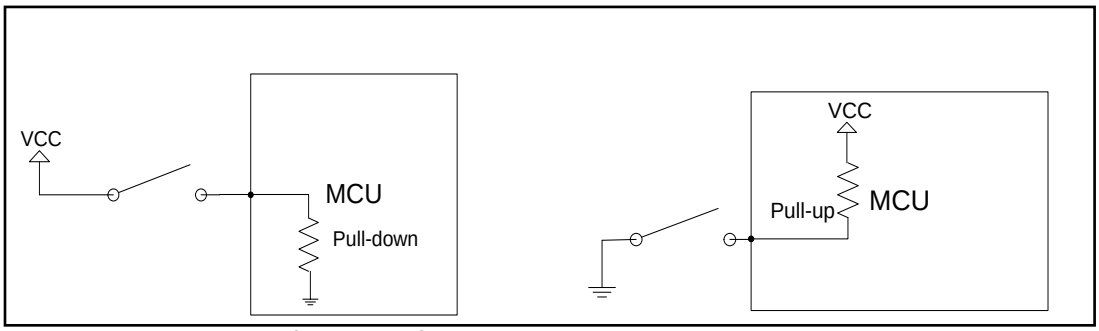
Exercise 1:

Write a function to return the state of the four switches you wired in Lab 0. Your function should be passed no parameters, and it should return a **char** between 0-15 that represents each of the sixteen possible states. You will need to configure a number of digital I/O control registers (e.g., PxSEL and PxDIR). You will also need to enable the pull-up/down resistors on the input pins (using PxREN and PxOUT) you wired to the switches (Why?).

Note that the things you learned by completing the Prelab assignment should significantly help you to complete this exercise. As with the Prelab assignment, you will find the lecture notes, the MSP430 User's Guide (especially the section about Digital I/O, Chapter 8,) and the MSP430 2272 Datasheet (pages 58-77) helpful. Also remember that many of the control register names, along with things such

as predefined constants with (relatively) easy to remember names, are declared in the "msp430x22x2.h" header file. You are strongly encouraged to become familiar with the contents of this file.

You should be able to tell from the diagrams below which type (pull-up or pull-down) you need. Pay close attention to the location of the switch in each diagram and where (electrically) you soldered the switch to your board.



Pull-up and pull-down resistor configurations for a switch.

Call the function inside your `while` loop in the `main` function every 0.5s. Blink a different LED for each state at a rate of 1Hz (on-off takes 1s). Don't forget to configure the pins connected to the LEDs as digital outputs. *Hint:* this is the default rate in the project creator. Also, print the switch state to the serial port as discussed in Lab 0. **Demonstrate your working application to the TA.**

Exercise 2:

Now, instead of calling your read digital inputs function from Exercise 1, take advantage of the interrupt capability of the MSP430 digital I/O pins. Setup the four digital input pins so that an interrupt function is call immediately after any one of the buttons are pressed.

Add a port 2 interrupt service routine to your program by copying the code below and pasting it into your program at the function definition (global) level.

```
// Port 2 interrupt service routine
#pragma vector=PORT2_VECTOR
__interrupt void Port_2(void)
{

}
```

This code was taken from one of the TI example C files found on the course website. These examples are a great resource for completing lab exercises.

Use the four switches wired to port 2 pins to trigger hardware interrupts. You will have to enable the corresponding pin interrupts using the `P2IE` register and select the trigger edge using the `P2IES` register. Two of your switches should trigger interrupts on a high-low edge, and two should trigger interrupts on a low-high edge. Inside the interrupt service routine, check to see which switch triggered

the interrupt by reading the P2IFG register and increment one of four counters depending on which switch was pressed. Don't forget to clear the interrupt flag for the pin that triggered the interrupt.

Print the values of these four counters to the terminal window every .5 seconds and **demonstrate the application to your TA.** When you press your switches you may receive multiple interrupt calls. This is due to "bounce" of the switch. If this happens quite a bit ask your TA how to solder an additional capacitor on each switch.

Exercise 3:

In this exercise I want you to have some fun playing a bit more with turning on and off the circle of 8 LEDs. Start with the code you developed in exercise 2 and add code to make the LEDs blink in a creative pattern. An example would be a five point star, but you can be more creative than that. Only one LED should be on at any point in the pattern. Put this new code in Timer_A's interrupt function outside of the "if (fastcnt == 500)" instruction. Remember that this function is called every 1ms.
Pick a default timing (how long a single LED stays on) that will start your pattern sequence. Then using two of the switches, make one of them your "speed up" button and the other a "slow down" button. When the "speed up" button is pressed, decrease the "on" time of the LEDs as they perform the pattern. When the "slow down" button is pressed, increase the "on" time of the LEDs.
Show your working application to your TA.