CPE 325: Intro to Embedded Computer System

Lab03

<u>Digital I/O on Experimenter's Board: LEDs and Switches</u>

Submitted by: <u>Dan Otieno</u>

Date of Experiment: 01/28/2022.

Report Deadline: 01/31/2022.

Demonstration Deadline: 02/03/2022.

Introduction

This lab's purpose is to learn, understand and demonstrate how we can use programming language to design software logic that interacts with hardware. In the case of the MSP430, we define interface instructions for I/O switches and pins using "and" with "or" logic. We set up these instructions to define LED light outputs based on the switch inputs, using two switches (S1 and S2) and two output LEDs (LED1 and LED2) on the MSP430 board.

Theory

The topics for theory, debouncing and software delay, have been explained below while answering questions about my approach to completing this assignment.

Results & Observation

Program 1:

Program Description:

Because this is a project that requires programming to define instructions that are outputted on a piece of hardware. I decided to follow the method provided in our lab demo codes to design my software. The first step was to define the switches (S1 and S2) globally. By setting global definitions for S1 (Parallel Port 2, Pin 1) and S2 (Parallel Port 1, Pin 1), we make it easier to reference them when we need to as we set loop conditions to determine whether a switch is pressed or not. Next, going into the main loop, I set the direction registers (PxDIR) for the input and output pins, and then set the output registers (PxOUT), which would correspond to the LED port pins (keeping in mind that the inputs, the switches, are define globally). Next, we define the instructions to enable pull-up resistors using the PxREN register and then writing the outputs on another set of PxOUT registers (pins with 0 bit value will produce low output voltage and bit value of 1 will produce high output voltage).

The input commands are then defined in a series of embedded loops and if-else statements. We need to define a counter "i" to associate with our debounce (20ms = 20,000 clock cycles). And then, I initialized an infinite for-loop, inside which the if-else statements are embedded. The approach was simple, an if-statement that sets the input conditions (i.e., S1 is pressed, S2 is not, and vice-versa, both S1 and S2 are pressed, and finally, an initial state where none of the switches are pressed). And then inside the if statements, I set up other for-loops to provide outputs within the 20ms debounce period and do-while loops that included on/off/blink instructions within the frequencies that were asked in the assignment (2Hz, 5Hz, 8Hz). Here is how I converted frequencies to clock cycles, (also commented in the code):

```
2Hz = 1/2(secs) = 0.5 secs. To determine delay parameter, we
first divide the total time by half, for ON and OFF. In this case
0.5/2 = 0.25s ON/0.25s OFF. This is equivalent to (0.25*10^6)
clock cycles = 250000 cycles.

5hz = 1/5(secs) = 0.2secs. 0.2/2 = 0.1 ON/0.1 OFF. Clock cycles
delay parameter = (0.1*10^6) cycles = 100000.
```

```
8hz = 1/8(secs) = 0.125secs. 0.125/2 = 0.0625 ON/0.0625 OFF.
Clock cycles delay parameter = (0.0625*10^6) cycles = 62500.
```

Finally, the default condition is defined at the end, inside an else statement, i.e., both switches are not held. NOTE: The frequency clock cycles calculated above are passed into the _delay_cycles function inside the do-while loops to set delay times.

Program Output:

N/A, output is to be observed on MSP430 hardware.

Report Questions:

1. **How do you handle debouncing?**
   I handled debouncing by adding software delay inside my for-loops.

2. **How do you create the required delay?**
   I created the delay by setting up the do-while instructions to execute my I/O commands inside for-loops with 20milliseconds iterations, that translates to (20ms*10^6) clock cycles, or 20,000 clock cycles.

3. **How does your code handle both the switches?**
   Due to the software delay defined in the code, when the switches were pressed or released, they were detected properly.

Program Flowchart:

FLOWCHART FOR MSP430 LED I/O PROGRAM.

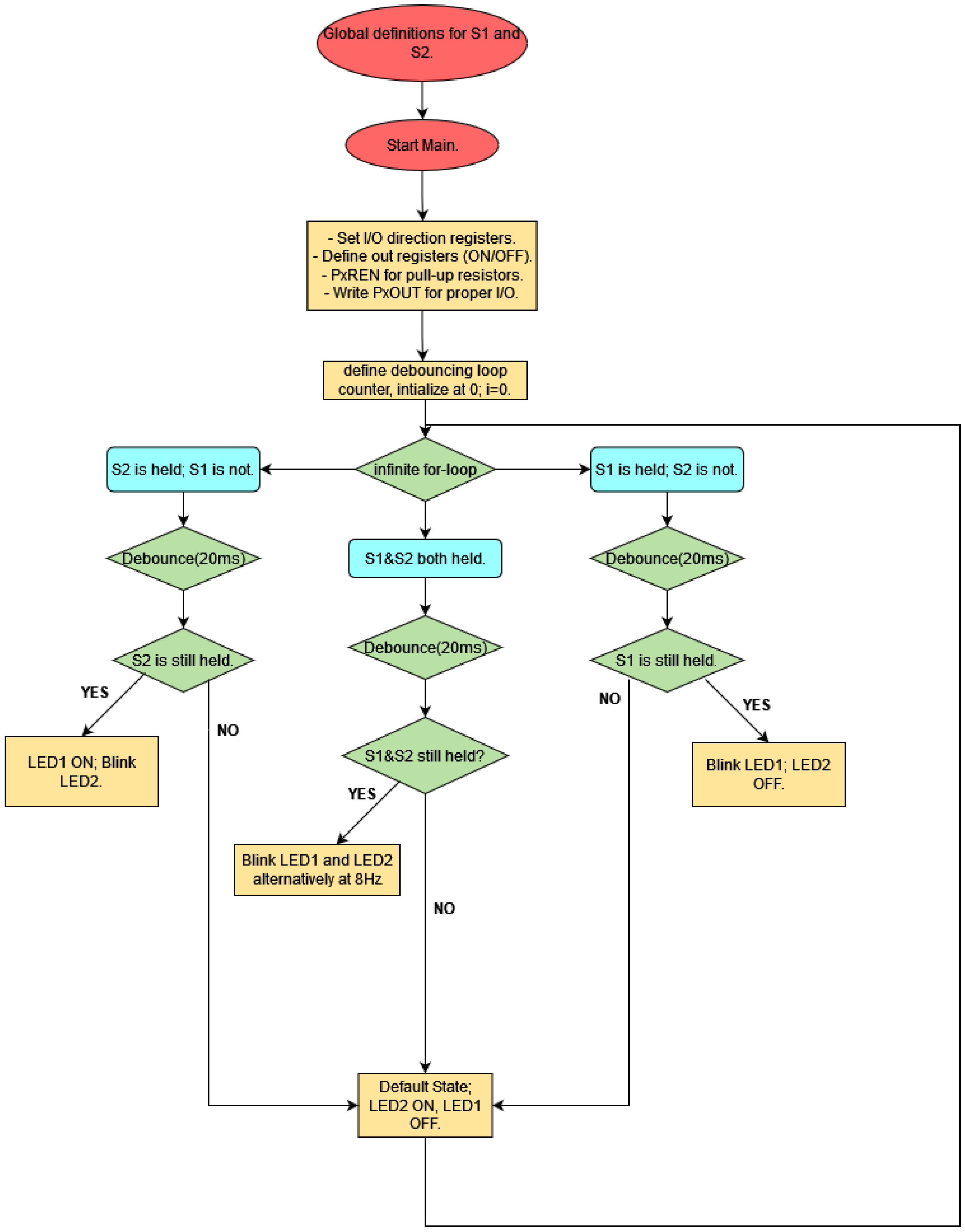
Global definitions for S1 and S2.
Start Main.
- Set I/O direction registers.
- Define out registers (ON/OFF).
- PxREN for pull-up resistors.
- Write PxOUT for proper I/O.
define debouncing loop counter, intialize at 0; i=0.
infinite for-loop
S2 is held; S1 is not.
S1 is held; S2 is not.
Debounce(20ms)
S1&S2 both held.
Debounce(20ms)
S2 is still held.
Debounce(20ms)
S1 is still held.
YES
NO
LED1 ON; Blink LED2.
S1&S2 still held?
YES
Blink LED1 and LED2 alternatively at 8Hz
NO
NO
YES
Blink LED1; LED2 OFF.
Default State; LED2 ON, LED1 OFF.

Figure 01: Program Flowchart.

Conclusion

In conclusion, this was an interesting program to work on, I think the MSP430's architecture makes this a comfortable learning process. I did not notice any odd program behavior and the logic was straightforward. I noticed that when I moved my default state instructions right below the infinite for loop, but above the if statements, and then removed the if statements and wrote instructions for S1, S2, S1&S2 in individual for loops (with do-while, of course), both lights just kept blinking!

Appendix

Table 01: Program 1 source code.

```
/*------------------------------------------------------------------------------------------
-----------
 * File:        Lab03_LEDs.c
 * Function:    This C code will interact with the MSP430 hardware by controlling LED
I/Os.
 * Description: This program will interface LED 1 and LED 2 on the MSP430 board by
 *              defining input and output instructions, and using LED lights to
demonstrate results.
 * Input:       Defined in code, corresponding to hardware input pins.
 * Output:      LED light states (on or off).
 * Author(s):   Dan Otieno, dpo0002@uah.edu
 * Date:        January 28th, 2022.
 * ----------------------------------------------------------------------------------------
-----------*/
#include <msp430.h>
#define S1 P2IN&BIT1
#define S2 P1IN&BIT1

int main(void)
{
    WDTCTL = WDTPW | WDTHOLD;   // stop watchdog timer

    // INTERFACING DIRECTION REGISTERS FOR INPUT PINS;
    P2DIR &= ~BIT1;         // Input: S1 - P2.1;
    P1DIR &= ~BIT1;         // Input: S2 - P1.1;

    // INTERFACING DIRECTION REGISTERS FOR OUTPUT PINS;
    P1DIR |= BIT0;          // Output: LED1 - P1.0;
    P4DIR |= BIT7;          // Output: LED2 - P4.7;

    // ON/OFF STATE;
    P4OUT |= BIT7;          // LED2 is ON;
    P1OUT &= ~BIT0;         // LED1 is OFF;

    // ENABLE PULL-UP RESISTOR/SET PROPER IN/OUT;
    P2REN |= BIT1;          // Enable pull-up resistor at P2.1;
    P2OUT |= BIT1;          // Required for proper IO;
    P1REN |= BIT1;          // Enable pull-up resistor at P1.1;
    P1OUT |= BIT1;          // Required for proper IO;

    unsigned int i = 0;
    for (;;)
    {
        if((S1)==0 && (S2)!=0)          // When S1 is pressed, S2 is NOT pressed;
        {
            for (i=20000; i>0; i--);    // Debounce: 20 ms = 0.02secs = (0.02*10^6)
cycles = 20000 cycles;
            {
                do
                {
                    P4OUT &= ~BIT7;         // Turn off LED2;
                    P1OUT ^= BIT0;          //  Blink LED1;

                    /* 2Hz = 1/2(secs) = 0.5 secs. To determine delay parameter,
                    - we first  divide the total time by half, for ON and OFF . In this
case 0.5/2 = 0.25s ON/0.25s OFF.
                    - This is equivalent to (0.25*10^6) clock cycles = 250000 cycles.*/

                    __delay_cycles(250000);

                }while((S1)==0 && (S2)!=0); // ....S1 is still pressed but S2 is not!;
            }
        }

        else if((S2)==0 && (S1)!=0)     // S2 is pressed, S1 is not pressed;
        {
            for(i=20000; i>0; i--);     // Debounce: 20ms/20000 clock cycles;
            {
                do
                {
                    P1OUT |= BIT0;      // Turn on LED1;
                    P4OUT ^= BIT7;      // Blink LED2;

                    /* 5hz = 1/5(secs) = 0.2secs.
                    - 0.2/2 = 0.1 ON/0.1 OFF.
                    - clock cycles delay parameter = (0.1*10^6) cycles = 100000.*/

                    __delay_cycles(100000);
                }while ((S2)==0 && (S1)!=0); //....S2 is still pressed but S1 is not;
            }
        }

        //(Bonus) S1 and S2 are both held, blink LED1 and LED2 alternatively;
        else if((S1)==0 && (S2)==0)     //Both switches are pressed;
        {
            for(i=20000; i>0; i--);     // Debounce: 20ms/20000 clock cycles;
            {
                do
                {
                    /* 8hz = 1/8(secs) = 0.125secs.
                    - 0.125/2 = 0.0625 ON/0.0625 OFF.
                    - clock cycles delay parameter = (0.0625*10^6) cycles = 62500.*/

                    P1OUT |= BIT0;      // Blink LED1;
                    P4OUT &= ~BIT7;     // Turn off LED2;
                    __delay_cycles(62500);

                    P1OUT &= ~BIT0;     //Turn off LED1;
                    P4OUT |= BIT7;      //Blink LED2;
                    __delay_cycles(62500);

                }while((S1)==0 && (S2)==0);
            }
        }

        else        // No switches held, back to original state;
        {
            P4OUT |= BIT7;      // Turn on LED2;
            P1OUT &= ~BIT0;     // Turn off LED1;
        }
    }

    return 0;
}
```