CPE 325: Embedded Systems Laboratory
Laboratory #3 Tutorial
Digital I/O on Experimenter's Board: LEDs and Switches

Aleksandar Milenković
Email: milenka@uah.edu
Web: http://www.ece.uah.edu/~milenka

Objective:

This tutorial will help introduce MSP430 parallel ports and how they are used for interfacing LEDs and switches. Specifically, you will learn the following topics:

Hardware development platform MSP-EXP430F5539LP Experimenter Board
I/O Interfacing Using Parallel Ports (LEDs and switches)
Software delays (time estimation)

Notes:

All previous tutorials are required for successful completion of this lab, especially, the tutorials introducing the TI Experimenter's board and the Code Composer Studio software development environment.
Recommended reading: Introduction to the TI's MSP-EXP430F5529 LaunchPad Experimenter's Board User Guide http://www.ti.com/lit/ug/slau533d/slau533d.pdf

Contents:

1 Digital Input/Output Introduction

A microcontroller interacts in many ways with the system in which it is embedded. It may receive inputs from a human through switches, buttons, sensors, etc. In the opposite direction, the microcontroller may control external devices such as, light-emitting diodes (LEDs), seven-segment displays, liquid-crystal displays (LCDs), or actuators (e.g., motors). The MSP430 microcontrollers can drive these external devices directly if they work from the same voltage and draw a sufficiently small current. Heavier loads require dedicated circuitry to drive them.

The most straightforward form of input/output is through the digital input/output ports using binary values (0 or 1). The primary way how the MSP430 interfaces the rest of the world is through 8-bit parallel ports. The actual number of physical parallel ports varies with device type and can range from two to ten 8-bit parallel ports, typically marked P1-P10. The MSP430 parallel ports are bit-configurable and can work as standard digital input/outputs or as special-function ports (e.g., serve as analog inputs for an analog-to-digital converter or timer output). The parallel ports are directly connected to the chip pins. A parallel port encompasses multiple registers, including:

- PxIN - input register, reading it returns the logical values on the pins (determined by the external signals);
- PxOUT - output register, writing it sends the value to the pins;
- PxDIR - direction register, configures pins as inputs or outputs (e.g., P2DIR.BIT1=0 configures bit 1 of port P2 as an input pin; P2DIR.BIT2=1 configures bit 2 of port P2 as an output pin).
- PxSEL - selection register, this allows the user to change the register from the standard digital I/O to a special function. On the hardware diagram, when multiple symbols are seen on a pin, this will select between those functions. The default setting is the digital input (all direction bits are initially cleared).
- PxREN – Enables the pull-up or pull-down resistor connected. (e.g, P2REN.BIT1 = 1 enables the pull-up resistor that is connected to P2.1 SW1 in MSP-EXP430F5529LP board.)
- Ports P1 and P2 also have ability to serve as sources of interrupts and several registers are associated with this function. These are: PxIE – Port x Interrupt Enable register for enabling/disabling interrupts, PxIFG – Port x Interrupt Flag register for tracking pending requests, and PxIES – Port x Interrupt Edge Select register for selecting type of event that triggers an interrupt – rising edge at the port input (0 -> 1) or falling edge (1 -> 0).

Our development platform includes two LEDs (LED1-LED2) and switches (SW1 and SW2). The LEDs can be turned on and off by writing digital 1 or 0 to the appropriate output port registers. Therefore, in order to turn a LED on, first the I/O port should be set to output direction, and then either a 0 or a 1 should be written to the output register. In this lab, LEDs are turned on and off with specific frequencies. Let us develop a program to blink a LED.

2 Turning on a LED Project Using C Language

This section defines the problem that will be solved by the "Turn on a LED" application. Your task is to write a C program that will turn on the LED1 on the TI's MSP430F5529 Experimenter Board.

Step 1. Analyze the assignment.

In order to better understand the problem, we will first study schematics of the MSP430F5529LP Experimenter Board. This board includes TI's MSP430 microcontroller (MSP430F5529), 2 leds (LED1-LED2), 2 switches (SW1 and SW2), and several extension slots that allow an easy access to all microcontroller ports. A detailed schematic of the board is provided in the following document: http://www.ti.com/lit/ug/slau533d/slau533d.pdf. Go to page 55 and locate images with headings “User Buttons” and “User LEDs”.In the schematic see how the ports are actually connected to physical LEDs (a diode through a resistor Figure 1).

What microcontroller port pins are connected to LED1 and LED2? A LED is on when the current is flowing through it and it is off when there is no current. How should we drive the corresponding ports to have the current flow?

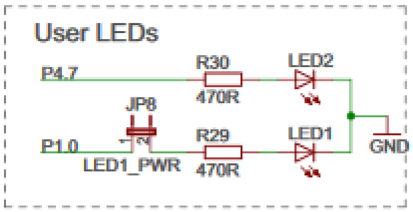

Figure 1. LED1 and LED2 connections on the TI's experimenter's board.

Step 2. Develop a plan.

From the schematic we see that if we want LED1 on, we should provide a logical '1' at the output port of the microcontroller (port P1.0), and a logical '0' if we want LED1 to be off. We could take several approaches to solving this problem. Figure 2 illustrates one such approach - after initializing the port P1.0 as output (P1DIR=00000001), setting P1.0 to logic '1', the program will spend all its time in an infinite loop (Figure 2).

```
/*------------------------------------------------------------------------------
 * File:        Lab3_D1.c (CPE 325 Lab3 Demo code)
 * Function:    Turning on LED1(MPS430F5529)
 * Description: This C program turns on LED1 connected to P1.0 by writing 1
 *              (P1.0 = 1).
 * Clocks:      ACLK = 32.768kHz, MCLK = SMCLK = default DCO (~1 MHz)
 *
 *                          MSP430F552x
 *                      -----------------
 *                 /|\|                 |
 *                  | |                 |
 *                    --|RST             |
 *                      |            P1.0|-->LED1(RED)
 *                      |                |
 * Input:       None
 * Output:      LED1 is turned on
 * Author:      Aleksandar Milenkovic, milenkovic@computer.org
 *              Prawar Poudel
 *------------------------------------------------------------------------------*/
#include  <msp430.h>

void main(void)
{
    WDTCTL = WDTPW + WDTHOLD;  // Stop watchdog timer
    P1DIR |= 0x01;              // Set P1.0 to output direction (0000_0001)
    P1OUT |= 0x01;              // Set P1OUT to 0000_0001b (LED1 is ON)
    for (;;);                   // Infinite loop so that the LED stays ON forever
}
```

Figure 2. Turn-on an LED Using C Code (Lab3_D1.c)

3 Blinking LEDs Using C Language

This section defines the problem that will be solved by the "Blink the LEDs" application. Your task is to write C program that will alternately blink the LED1 and LED2 on the TI's MSP430F5529 Experimenter Board with 1 Hz frequency, i.e., the LED1/LED2 will be on/ off for about 0.5 sec.

We could take several approaches to blink the LEDs. The simplest one is to toggle the port P1.0 and P4.7 and have 0.5 seconds delay in software as shown in Figure 3 (Lab3_D2.c). After initializing the microcontroller, our program will spend all its time in an infinite loop (LED1 and LED2 should be repeatedly blinking alternatively). Inside a loop we will toggle the ports P2.1 and P2.2 and then wait for approximately 0.5s. The port toggling can be done using an XOR operation of the current value of the port (P1OUT) and the constant 0x01, i.e., (P1OUT=P1OUT xor 0x01). Software delay of 0.5s can be implemented using an empty loop with a certain number of iterations (see the for loop in Figure 3).

To exactly calculate the software delay we need to know the number of clock cycles to execute one iteration of the for loop and the clock cycle time. The total number of clock cycles taken to execute the entire loop can be calculated by multiplying the number of clock cycles taken for one iteration of the loop with the number of iterations. In Figure 3, one iteration of the for loop takes 10 clock cycles and the loop counter is 50,000. Thus, it takes 10*50,000 = 500,000 clock cycles to execute the for loop (Note: when the counter is initialized in the for loop, first iteration takes some extra clock cycles. Since this is only for one time, it is not considered in our calculation for simplicity). Make sure you have your optimization turned-off. Sometimes the delay may differ because of optimizations and you may not observe the desired delay.

Determining clock cycle time requires in-depth understanding of the FLL-Clock module of the MSP430 which is beyond the scope of this tutorial. We note that the processor clock frequency is approximately 1 MHz for our configuration, so the clock cycle time is 1µs. The total delay is thus 500,000*1µs=0.5s.

```
/*------------------------------------------------------------------------------
 * File:        Lab3_D2.c (CPE 325 Lab3 Demo code)
 * Function:    Blinking LED1 and LED2 (MPS430F5529)
 * Description: This C program toggle LED1 and LED2 at 1Hz by xoring P1.0 and
 *              P4.7 inside a loop. The LEDs are on when P1.0=1 and P4.7=1.
 *              The LED1 is initialized to be off and LED2 to be on.
 * Clocks:      ACLK = 32.768kHz, MCLK = SMCLK = default DCO (~1 MHz)
 *
 *                           MSP430F552x
 *                       -----------------
 *                   /|\|                 |
 *                    | |                 |
 *                    --|RST              |
 *                      |             P1.0|-->LED1(RED)
 *                      |             P4.7|-->LED2(GREEN)
 *                      |                 |
 * Input:       None
 * Output:      LED1 and LED2 blinks alternately at 1Hz frequency
 * Author:      Aleksandar Milenkovic, milenkovic@computer.org
 *              Prawar Poudel
 *------------------------------------------------------------------------------*/
#include <msp430.h>

void main(void)
{
    WDTCTL = WDTPW + WDTHOLD;    // Stop watchdog timer
    P1DIR |= BIT0;               // Set P1.0 to output direction
    P4DIR |= BIT7;               // Set P4.7 to output direction
    P1OUT &= ~BIT0;              // LED1 is OFF
    P4OUT |= BIT7;               // LED2 is ON
    unsigned int i = 0;
    while(1){                    // Infinite loop
        for (i = 0; i < 50000; i++); // Delay 0.5s
                                     // 0.5s on, 0.5s off => 1/(1s) = 1Hz
        P1OUT ^= BIT0;              // Toggle LED1
        P4OUT ^= BIT7;              // Toggle LED2
    }
}

```

Figure 3. Blinking the LEDs Every Second (Lab3_D2.c)

We can step through the program using the Code Composer Studio debugger as in the previous labs. As you step through the program observe the Disassembly, Register View, and Memory View windows, and answer the following questions:

What is the starting address of the program?

How many clock cycles does each line of code take to execute?

Observe the contents of memory location and registers as you step through the program. What is the content of the memory location at the address 0xFFFE? What are addresses of the special-purpose registers P1DIR and P1OUT? Monitor the contents of these locations as you walk through your program. Set breakpoints to move easier through your program.

4 Interfacing Buttons (Switches)

Often, we would like to trigger a certain task in embedded systems by pressing a button or switch. Here we will learn how to interface a switch, how to detect that it is pressed and how to detect that it is released. First, let us look at the TI Experimenter's board development platform schematic (Figure 4) that illustrates how the buttons are connected to the MSP430. We can see two switches S1 and S2. The lines SW1 and SW2 that are connected to the MSP430’s pins P2.1 and P1.1, respectively (see Figure 4). When the switches are not pressed the inputs SW1 and SW2 are at logic 1 level (VCC) – there is an open circuit and the voltage level at SW1 and SW2 is equal to DVCC, which is power supply of the board. When the switches are pressed, SW1 and SW2 are connected to the ground and port inputs are at logic 0 (GND). This may seem counterintuitive, but it should reinforce the habit of becoming familiar with your hardware schematic before programming.

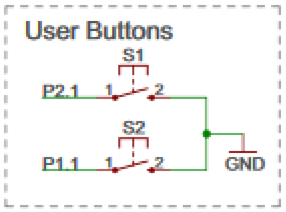

Figure 4. MSP430 Experimenter Board Switch Schematic

When interfacing switches we must take care that we properly detect whether a switch is pressed. Typically, we need (i) to detect that a switch has been pressed, (ii) to debounce it (apply software delay to ensure that is indeed pressed, rather than a faulty detection caused by noise), and (iii) to detect that it has been released. An example of the noise created by the action of a switch pressing can be seen in Figure 6 (note that in this example, pressed switch generates logic 1). If not programmed correctly, our application could think that each spike was an individual press of the switch, which could be detrimental to the functionality of the application.

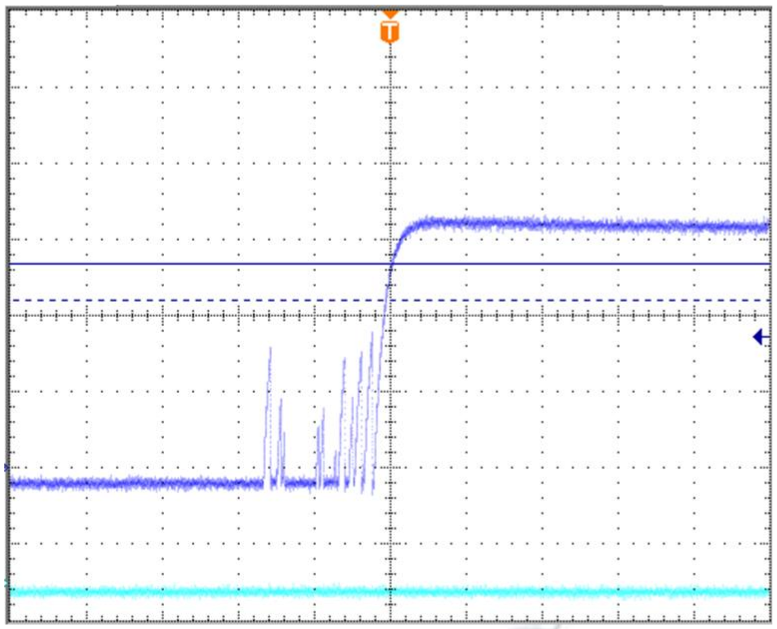
Figure 5. Typical Oscilloscope Reading of Switch Input

We can individually examine the P2IN bit 1 status to see if SW1 has been pressed. If it has indeed been pressed (bit 1 of P2IN is 0), we execute a loop to wait for a short period of time (~20 ms) to avoid faulty detections that may be caused by electrical noise. After the software delay, we validate that the input port pin is still at 0; if yes, that means that the switch is indeed pressed. We may keep the switch pressed for a longer period of time, and we often want to ensure that a switch is released before we go to process an event that may be triggered by this detection.
Figure 7 shows a program that turns LED1 on when the switch SW1 is pressed and it is keeps it on as long as SW1 is pressed. When the switch is released, LED1 is turned off.

```
/*------------------------------------------------------------------------------
 * File:        Lab3_D3.c (CPE 325 Lab3 Demo code)
 * Function:    Turning on LED1 when SW1 is pressed (MPS430F5529)
 * Description: This C program turns on LED1 connected to P1.0 when the SW1 is
 *              pressed. SW1 is connected to P2.1 and when the switch is pressed
 *              it is logic 0 (check the schematic). To avoid faulty detection
 *              of switch press delay of 20ms is added before turning on the LED1.
 * Clocks:      ACLK = 32.768kHz, MCLK = SMCLK = default DCO (~1 MHz)
 *
 *                          MSP430F552x
 *                       -----------------
 *                   /|\|                 |
 *                    | |                 |
 *                    --|RST              |
 *                      |             P1.0|-->LED1(RED)
 *                      |             P2.1|<-- SW1
 *                      |                 |
 * Input:       Press SW1
 * Output:      LED1 is turned on when SW1 is pressed
 * Authors:     Aleksandar Milenkovic, milenkovic@computer.org
 *              Prawar Poudel
 *--------------------------------------------------------------------------------*/
#include <msp430.h>

#define SW1 P2IN&BIT1

void main(void)
{
    WDTCTL = WDTPW + WDTHOLD;              // Stop watchdog timer
    P1DIR |= BIT0;                         // Set P1.0 to output direction (0000_0001)
    P1OUT &= ~BIT0;                        // LED1 is OFF

    P2DIR &= ~BIT1;                        // Set P2.1 as input for SW1 input
    P2REN |= BIT1;                         // Enable the Pull-up resistor at P2.1
    P2OUT |= BIT1;                         // Required for proper IO

    unsigned int i = 0;
    for (;;) {                             // Infinite loop
        if ((SW1) == 0) {                  // If SW1 is pressed
            for (i = 2000; i > 0; i--);    // Debounce ~20 ms
            if ((SW1) == 0)
            {
                P1OUT |= BIT0;             // SW1 pressed, turn LED1 on
            }
            while((SW1)==0);               // Hang-on as long as SW1 pressed
        }else
            P1OUT &= ~BIT0;
    }
}
```

Figure 6. Example of Turning on LED1 when SW1 is Pressed (Lab3_D3.c)

5 References

It is crucial that you become familiar with the basics of how digital ports work - how to set their output direction, read from or write to the ports, set interrupts, and set up their special functions. We will be using these features to control hardware and communication between devices throughout this class. Please reference the following material to gain more insight:

- The MSP430F5529LP Experimenter's Board hardware schematic (http://www.ti.com/lit/ug/slau533d/slau533d.pdf)
- Chapter 7 in the John H. Davies' MSP430 Microcontroller Basics