ELEC3042 Embedded Systems

Practical 2

Hardware Interaction and Interrupts

Laboratory Equipment
Personal Equipment Kit

Software
MPLAB X

Aims
The aims of this practical are:
- To understand limitations of hardware in the real world.
- To utilise hardware functional units on the Atmel ATmega328P.

Pre-Work
- Complete the self-assessment quiz

Outcomes
After completing this practical, you will have the following knowledge and skills:
- Knowledge of the architecture and functionality of the Atmel ATmega328
- Knowledge of the C programming language
- Beginning understanding of interrupts

There are three parts to this practical. All three parts should be completed.

Completed

Part 1	Done
Part 2	Done
Part 3	

Part 1: Switch Debouncing

Introduction

Switches are an example of a real-world interaction. In an ideal world, the switch will change from open to closed and back again cleanly without any noise or other problems.

Unfortunately, the real world is not neat, nor tidy. A switch in the real world will not transition cleanly. Because it is a mechanical system there is play, spring, bending, momentum and other forces at play. This has the result that the opening or closing of a switch may generate multiple transitions between high and low.

The micrcontroller is fast enough that it can detect these transitions. This gives the appearance to your code of the switch being changed multiple times. Switches are designed to minimise this problem, but it can still occur.

The code shown on the next page, and provided in ELEC3042_L2P1_switch_count.c increments a binary counter displayed on the LEDs on the HW-262 board when Switch S1 is pressed. If you press the switch occasionally you may see the count increase by more than one for every button press.

We need to eliminate these extra counts. To do this, we can wait for the switch to change, and then check it is still changed 10 ms later. The bounces generally stop well within the 10 ms time, so a delay of 10 ms and an extra check should eliminate these bounces.

The code snippet below demonstrates a framework for this change:

```
for(;;) {
   while ((PINC & 0b00000010) != 0) {
      ;  // wait until the push button is pressed
   }
   delay_ms(10);
   if ((PINC & 0b00000010) != 0) {
      // no longer pressed, go around again
      continue;
   }
   break;
}
```

The outer loop is now only exited when the inner loop finishes (the switch has changed), a delay of 10 ms has passed, and the button is still pressed.

Make the equivalent change to the button release code and verify it works.

Congratulations, you have made a user-controlled state machine. Your system implements a 32-state machine, where each press-and-release is a separate state. This is represented by the count variable and the loop that is currently active.

```c
/*
 * File:   ELEC3042_L2P1_switch_count.c
 * Author: rex
 *
 * Created on 3 January 2022, 6:43 AM
 */


#include <xc.h>

/**
 * The delay routine is calibrated for an Arduino Uno 16MHz
 *
 * @param num number of ms to delay
 */
void delay_ms(long num) {
   while (num--) {
      for (volatile long x = 0; x < 468; x++) {
         ;
      }
   }
}

/*
 * We set the Data direction register to specify which pins are inputs and
 * which are outputs. A 0 bit indicates the corresponding pin is an input,
 * a 1 indicates the corresponding pin is an output
 */
void setup() {
   DDRB |= 0b00111100;     // PORTB pin 2-5 (D10-D13) output
   DDRC  &= 0b11110001;    // PORTC pin 1-3 (A1-3) input
   PORTB |= 0b00111100;    // turn off LEDs
   PORTC |= 0b00001110;    // turn on internal pullup for Port C pins
}

int count = 0;

int main(void) {
   setup();    // set up the physical hardware
   while (1) {
      while ((PINC & 0b00000010) != 0) {
         ;   // wait until the push button is pressed
      }
      PORTB = ~((count++)<<2);    // show count on LEDs
      while ((PINC & 0b00000010) == 0) {
         ;   // wait until the push button is released
      }
   }
}
```

Part 2: Interrupts (I/O Pins)

Introduction
Interrupts are a powerful technique to allow an important event to occur asynchronously to the currently running code.

For very simple examples, such as what we have been using so far, interrupts are typically overkill. However, when we progress to state machines, and then to complex state machines, interrupts provide a valuable tool to simplify programming.

Every I/O pin on the ATmega328P can generate an interrupt on input change. We can also have interrupts generated when internal hardware events occur. It is important to be able to capture these interrupts. This code is a demonstration piece of code that uses an interrupt to capture the current button state. So, instead of looking at PINC, we only have to look at the current button state.

This removes the direct interaction with hardware from the main code base, and moves it to a self-contained piece of code. This is an example of Object-Oriented code, as we could then change the physical interface in the self-contained code, and that change would be reflected everywhere in our code.

A framework has been provided for this practical on ilearn in the file
`ELEC3042_L2P2_switch_interrupt.c`.

You need to create the project, add the source file, and add code to configure the interrupt for the appropriate input pin. The framework has lines marked as "**TODO**" that you will need to complete.

You will need to set up for switch S1 on the HW-262 to cause a pin-change interrupt on the ATMega328P. You can do this by:

- Looking at the schematic for the HW-262 to determine which PCx pin of the ATMega328P that S1 is connected to.
- Looking at the 28 SPDIP diagram on Page 12 of the ATMega328P datasheet or the pinout diagram of the Arduino UNO, to determine the pin-change interrupt bit value (PCINTn) that corresponds to that PCx pin.
- In your code, set the relevant bit in the Pin Change Mask Register (PCMSKx) corresponding to the selected PCINTn. (Refer section 12 and 13 of the ATMega328P datasheet)
- In your code, set the relevant bit in the Interrupt Control Register (PCICR) that enables interrupts for the PCIx group that corresponds to the PCINTn. (Refer section 12 and 13 of the ATMega328P datasheet)
- Finally, enable interrupts globally in the Global Status Register (using the sei() call).

As an aside, this sample code introduces an *enum*, this is a user defined data type that can only take on a range of values, which are fully specified.

Also, note that `button_state` is defined as `volatile`. This tells the compiler it can change because of an interrupt. If we don't do this the compiler may change our code assuming it won't change.

Part 3: Exercise

Create a program such that each time you press A2, it shifts a single lit LED from D1 to D2 to D3 to D4 to D1, etc.