ELEC3042 Embedded Systems

Practical 1

Getting to know MPLAB X and the Arduino hardware

Laboratory Equipment
Personal Arduino Equipment Kit

Software
MPLAB X
Arduino IDE

Aims
The aims of this practical are:
- To configure MPLAB X to enable it to program the Arduino Uno.
- To use the simulator to follow the path of execution, and display the program memory and registers of the program.
- To write a first C program using the MPLAB X IDE and successfully program the Arduino with this program.

Pre-Work
- Install Microchip's MPLAB X IDE, as well as the **NON-App** version of the Arduino IDE. See the document *Configuring MPLAB X IDE to use Arduino UNO* for details.

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
- Ability to load a C program onto the Arduino Uno

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

Completed

Part 1	Done
Part 2	Done
Part 3	Done
Part 4	Done

Part 1: Flashing the inbuilt Arduino LED

Introduction
The Arduino UNO has an LED connected to I/O port PB5 (Arduino D13). It is customary for any new user of the Arduino ecosystem to write a first program that flashes this LED. We will do the same.

To achieve this, we first need to initialise the processor, to ensure that it is in a sane state for our program to execute. Then we need to configure the port as an output port, and give the output and alternating high and low voltage. Providing an alternating voltage implies we need to be able to provide a delay. Hence, the following program sections are needed.

Program Section	Purpose
Init	Initialise the processor, set up ports, put expected data values into variables.
Set Port Value	We need to be able to set either a high or a low on an output pin.
Delay	Busy wait for a defined period of time

Code to perform these actions is provided on iLearn and listed below.

We will use MPLAB X to compile this code and program the microcontroller with the resultant binary. If done correctly the LED will flash on the UNO's circuit board.

C source File: A print out of the C source file follows:

```c
/*
 * File:   ELEC3042_L1P1_blink.c
 * Author: rex
 *
 * Created on 21 December 2021, 9:58 AM
 */

#include <xc.h>

/**
 * The delay routine is calibrated for an Arduino Uno 16MHz
 *
 * @param num number of ms to delay
 */
void delay_ms(uint16_t 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 |= 0b00100000;     // PORTB pin 5 (D13) output
    PORTB |= 0b00100000;    // turn off LED
}

int main(void) {
    setup();    // set up the physical hardware
    uint16_t delay = 1000;
    while (1) {
        PORTB ^= 0b00100000;    // invert LED
        delay_ms(delay);
    }
}
```

The Arduino board's inbuilt programmer is not one of the officially supported MPLAB X programmers. Instead we will be using avrdude to copy the hex file produced during the compilation into the microcontroller's flash memory. We will now create a new project, and configure it to use the avrdude helper application from the Arduino IDE to program our arduino.

In MPLAB X create a new project, call the new project **ELEC3042_lab1_blink** (or a name of your choosing), add a new avr_main.c file (calling it blink.c) to the project, and copy the supplied contents to the to the file.
Change the build property as outlined in the ELEC3042 setup guide for MPLAB X to allow avrdude to be called, and compile the code.

If it was successful the inbuilt LED will flash. Change the delay from 1000 to 500 ms, and rebuild the project, ensuring the LED flashes faster this time.

Part 2: Turning on Multiple LEDs

Introduction

In Part 1 we made a single LED flash. Moving a value into the PORTB register would cause that value to be set onto the output pins of the microcontroller. In this part we want to display other values on the LEDs.

Examining the code in `ELEC3042_L1P2_count.c` we can see the following code snippet:

```
PORTB ^= 0b00100000;    // invert LED
delay_ms(delay);
```

We are changing the LED bit only. PORTB is 8 bits wide. Of these 8 bits the 2 most significant bits are used by the crystal oscillator and the remaining lower 6 bits, bits 0 to 5, are presented directly on Arduino output pins labelled D8 to D13 respectively. The HW-262 shield has LEDs connected to pins D10 to D13. In the Arduino IDE you can only change one bit at a time. We can change multiple bits simultaneously by writing a value to the PORT B register. We will experiment with this by using the four LEDs connected to the output pins and changing them simultaneously. We will use them as a 4-bit counter.

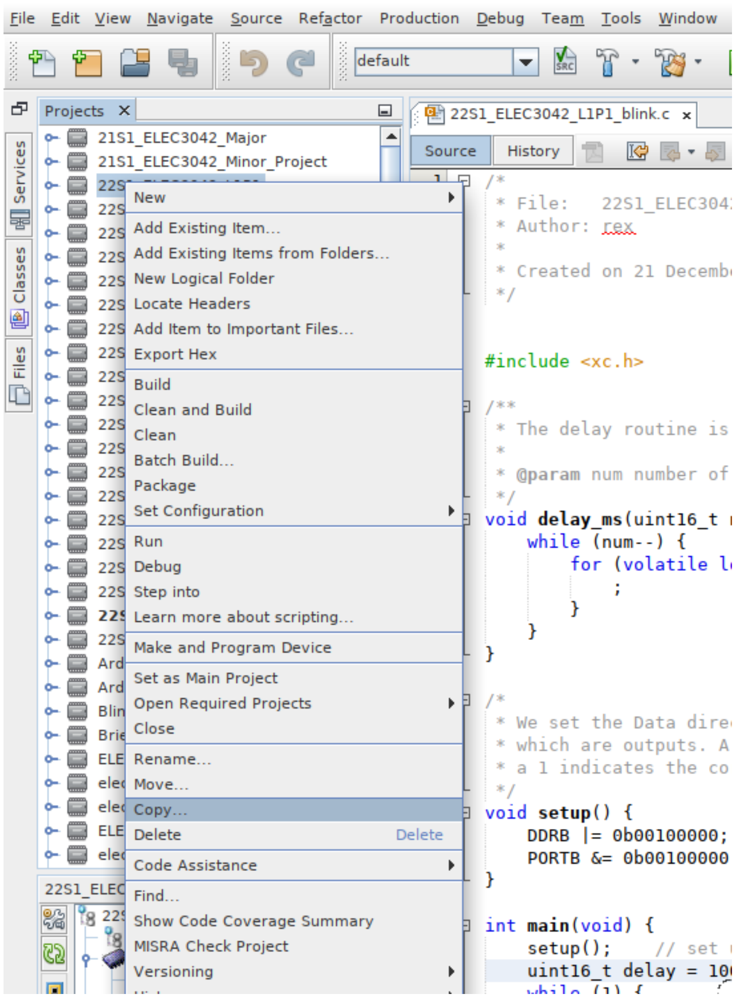
Procedure:

Note: We need to create a new project for this lab.

Select the project you created in Part 1. right click and select **Copy**. You will be prompted to change the project name to a new, unique, name. You can then rename the source files and edit them or replace them for the new project.

This process copies the build configuration, including the line we added in Part 1. if you created a new project you would have needed to insert a new build configuration line as we did in Part 1.

Right-click on the new project and choose “Set as Main Project”. This will ensure that the main function in this project is built and downloaded to the Arduino.



```
/*
 * File:   22S1_ELEC304
 * Author: rex
 *
 * Created on 21 Decemb
 */

#include <xc.h>

/**
 * The delay routine is
 *
 * @param num number of
 */
void delay_ms(uint16_t
    while (num--) {
        for (volatile l
            ;
        }
    }
}

/*
 * We set the Data dire
 * which are outputs. A
 * a 1 indicates the co
 */
void setup() {
    DDRB |= 0b00100000;
    PORTB &= 0b00100000
}

int main(void) {
    setup();    // set
    uint16_t delay = 10
```

We will replace the code snippet with a counter. We will use the Port B register to hold the current value, and also present it on the output pins. Each loop we will increment the visible counter and delay for a second. This will give the output as a counter that is increasing by one each second. Because our counter is 4 bits in size it will count from 0 to 15 and then repeat.

Disconnect the Arduino from power and add the HW-262 shield. Be careful to make sure the display pins do not short on the USB case. Find the LEDs connected to D10-D13 (Port B bits 2 to 5). These will be our display. We will use Port B bit 2 as the lowest bit (Bit 0) of our output, Port B bit 3 as output bit 1, Port B bit 4 as output bit 2, and Port B bit 5 as output bit 3. Change the code snippet to add 0b100 to PORTB each time through the loop. Example code is shown below. This should make the LEDs count 0000 through to 1111.

Active High v. Active Low:
When we execute the code, we can see the LEDs count in a binary sequence. However, they are not counting up, they are counting down! Recall from ELEC2042, we can have digital lines that are active high, and have lines that are active low. The LEDs on the shield are active LOW, so we will need to invert them before we display them on Port B.

To do this change the assignment of value to Port B to the code below:

```
        PORTB = ~value;
```

Which when executed will have the LEDs counting correctly.

We can see that the output on the LEDs is any value written to the PORTB register. We can manipulate individual bits, or operate on them as a group. Any operation can be performed on the data that is written to PORTB.

Replace the `value += 0b100;` instruction with the following code:

```
        value = (!(value & 0x3c)?0b100:(value<<1));
```

What is the effect of this change? **The LED's now are all on, a low signal moves up through the LED's until it reaches the end and repeats.**

```
/*
 * File:   ELEC3042_L1P2_count.c
 * Author: rex
 *
 * Created on 21 December 2021, 13:17 PM
 */

#include <xc.h>

void delay_ms(uint16_t num) {
   while (num--) {
      for (volatile long x = 0; x < 468; x++) {
         ;
      }
   }
}

uint8_t value = 0;

void setup() {
   DDRB = 0b00111111;  // all pins are outputs
   PORTB = value;         // initial value
}

int main(void) {
   setup();    // set up the physical hardware
   uint16_t delay = 1000;
   while (1) {
      value += 0b100;    // increment value presented on Port B LEDs
      PORTB = value;
      delay_ms(delay);
   }
}
```

Part 3: Input

Introduction

We commonly want to control our program through the use of inputs. In this exercise we will provide a control over our blinking LED. We will organise our system so when a button is pressed the blinking of the LED will pause, and it will continue when the button is released.

As we will see in the lectorials, input lines are often Active Low, and this case is no exception. The HW-262 shield has three buttons. They are connected to Port C bits 1, 2 and 3. When pressed the input will be read as a 0.

Procedure:

We will monitor I/O port Port C bit 1 (A1) for an input. While this is high we will allow the loop to continue. If it is low we will busy wait until it returns to a high value. Note that in this case low means go and high means stop.

Ensure the shield is connected to the Arduino. The A1 button will pull the Port C bit 1 pin low when pressed and leave it to float when high. We will configure an internal pullup resistor so that the pin is registered as a high when not pulled low by the switch.

To configure the internal pullup resistors, add the following line to the setup function:

```
PORTC |= 0b00000010;    // turn on internal pullup for Port C pins
```

Compile and build the program. Observe that when you push the A1 button, the blinking will pause.

Study the code to ensure you understand how it is working.

```
/*
 * File:   ELEC3042_L1P3_blinkpause.c
 * Author: rex
 *
 * Created on 21 December 2021, 15:41 PM
 */

#include <xc.h>

void delay_ms(uint16_t num) {
    while (num--) {
        for (volatile long x = 0; x < 468; x++) {
            ;
        }
    }
}

void setup() {
    DDRB  |= 0b00100000;    // PORTB pin 5 (D13) output
    DDRC  &= 0b11111101;    // PORTC pin 1 (A1) input
    PORTB |= 0b00111100;    // turn off LEDs
}

int main(void) {
    setup();    // set up the physical hardware
    uint16_t delay = 1000;
    while (1) {
        while ((PINC & 0b00000010) == 0) {
            ; // busy_loop until button released
        }
        PORTB ^= 0b00100000;    // invert LED
        delay_ms(delay);
    }
}
```

Part 4: Exercise

The delay is fixed at 1000 ms. Use the Port C bit 3 (A3 switch) to allow it to be changed dynamically. When the switch is pressed use a delay of 500 ms. When the switch is not pressed use the default 1000 ms delay.

Make a copy of the Project from Part 3 and extend the blinkpause code with this extension. You will need to set Port C bit 3 as an input, enable it’s internal pullup resistor and insert an if-statement so that when A3 is pressed it blinks at 500 ms, else it should blink at 1000 ms.