Lab 4

Formal Laboratory Report

EE202 – Embedded Systems

Student: Adelin Denis Diac

Student Number: 20337941

Date: 10/03/2022

Lecturer: Robert Sadleir

Faculty of Engineering & Computing
DCU Glasnevin Campus
Glasnevin
Dublin 9

Contents

Objective

The objectives of this lab session were as follows:

- To understand how to connect the physical PIC16F690 chip to the laptop via the PICKIT Debugger, allowing the program memory to be uploaded onto the chip.
- To understand the role of the three ports, and how certain pins relate to each port.
- Understanding how to initialise the pins as digital inputs or outputs to allow current to flow into/from certain pins.
- To understand how interrupts work and using them to cause an event to be triggered.

Procedure & Results

Part 1 – Initial Setup

For Part 1 of the lab session, it was required to construct the circuit, as per a given circuit diagram and upload the given code onto the PIC16F690.

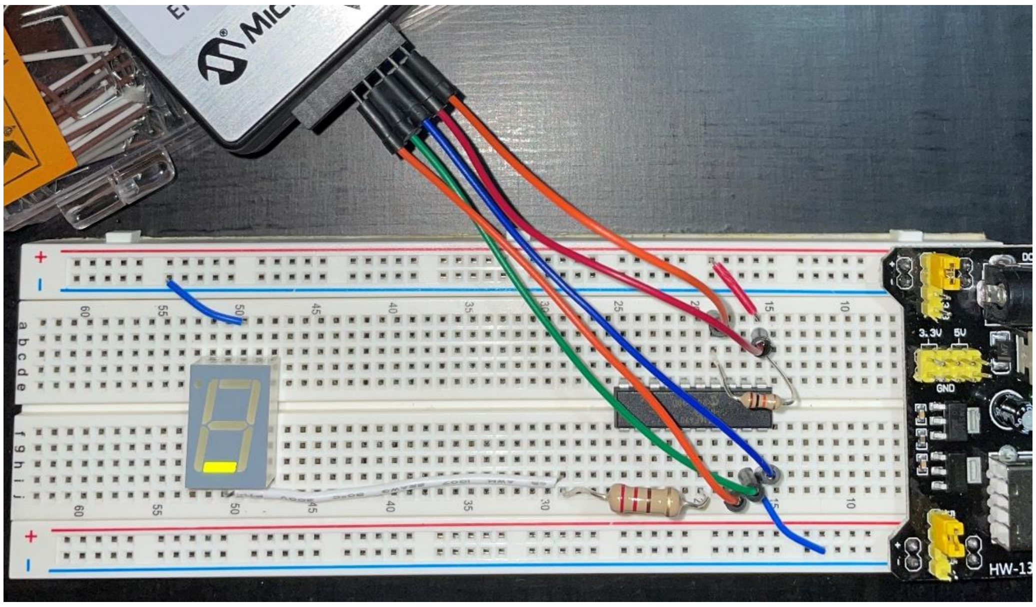


Figure 1 - Circuit Setup for part 1 - Blinking an LED

- Pin 1 of the PICKIT is connected to pin 4 of the IC, and a 10k resistor is then connected from pin 4 onto pin 1 of the IC. Pins 2 and 3 are connected to Vdd and Ground pins of the PIC16F690 respectfully, while the final pins from the PICKIT, pins 4 and 5 are connected onto pins 18 and 19. These connections are necessary to get the code from the C compiler, into a format understandable by the chip.
- The IC was then connected to the power rail and the ground rail.
- One of the LEDs was then connected to the RC0 pin (pin 16 of the IC) via a resistor. RC0 is represented by the 0 bit in the PORTC register.
- The code was then uploaded through the PICKIT.
- The code causes the LED, which is bright in Figure 1, to flash on and off for 500ms each. It does this by firstly, setting TRISC = 0x00. This indicates to the PIC that all the pins of port C are to be used as outputs. The ANSEL and ANSELH registers are also given the value 0x00 so that the output from the PIC is a digital output. This is required as the default setting is analogue.
- Inside a while loop, port C is then set to 0x01 for 500ms, and back to 0x00 for 500ms. The 1 in the first bit of port C indicates that RC0 is on, then it is turned off. This while loop continues indefinitely until the power supply is turned off.

Part 2 – Display Single Digit

For the second part, the circuit from part 1 was built upon so that now it displays a digit from 0-9, depending on the value of a variable inside the code. Bits 0-6 of port C were used to power the LEDs of the 7-segement display.

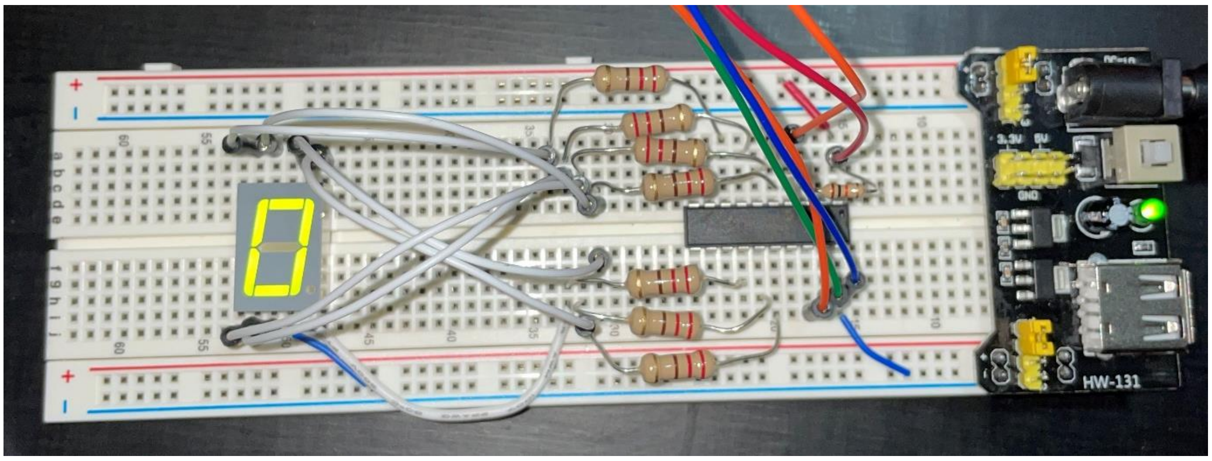


Figure 2 - Circuit Setup for part 2 showing the digit zero on the display

- The 7-segment display used in this lab session was a common cathode, so one of the middle pins had to be connected to the ground rail.
- The code which was implemented for this part was given in the lab sheet. It consisted of a function called displayUnits which would affect the value of the bits of the PORTC file register. It involved a switch case which would go to a given value depending on the integer passed into it. It would then set the PORTC value to the one which would turn on the corresponding LEDs required to display the number passed into the integer. For example, in Figure 2 above, the LEDs A, B, C, D, E, F are lit up, while G is off. LED G corresponds to the bit 6 of PORTC so this was kept at 0, while the others were given a value of 1. The value of PORTC in this example is then 0b00111111.
- A value of 1 on a given pin meant that it was set to high (5V), while a value of 0 meant low (0V).
- The second function inside the code was called displayNumber. This would check which number was displayed in the Tens display (used in Part 3) and what value was displayed by the Units display. It works by taking in an integer and checking how many times 10 divides into it (Ten's value) and would verify the remainder when dividing by ten (Unit's value). Using the value of the remainder, it then called the displayUnits function and passed in that value as an argument.
- It was necessary to have the resistors connected, otherwise it would be very possible to burn the LEDs since the PIC supplies up to 25mA of current.

Part 3 – Display Double Digit

For the third part of the lab session, a second 7-segment display was added to the circuit similarly to the first one, except in this case, there was no other port with at least 7 pins, so it was required to use various bits from each port. The pins used were RB7, RB6, RB5, RB4, RC7, RA5 and RA4. It was then required to come up with a function which would affect the values of these pins to display the integer value (between 0-9) on this display.

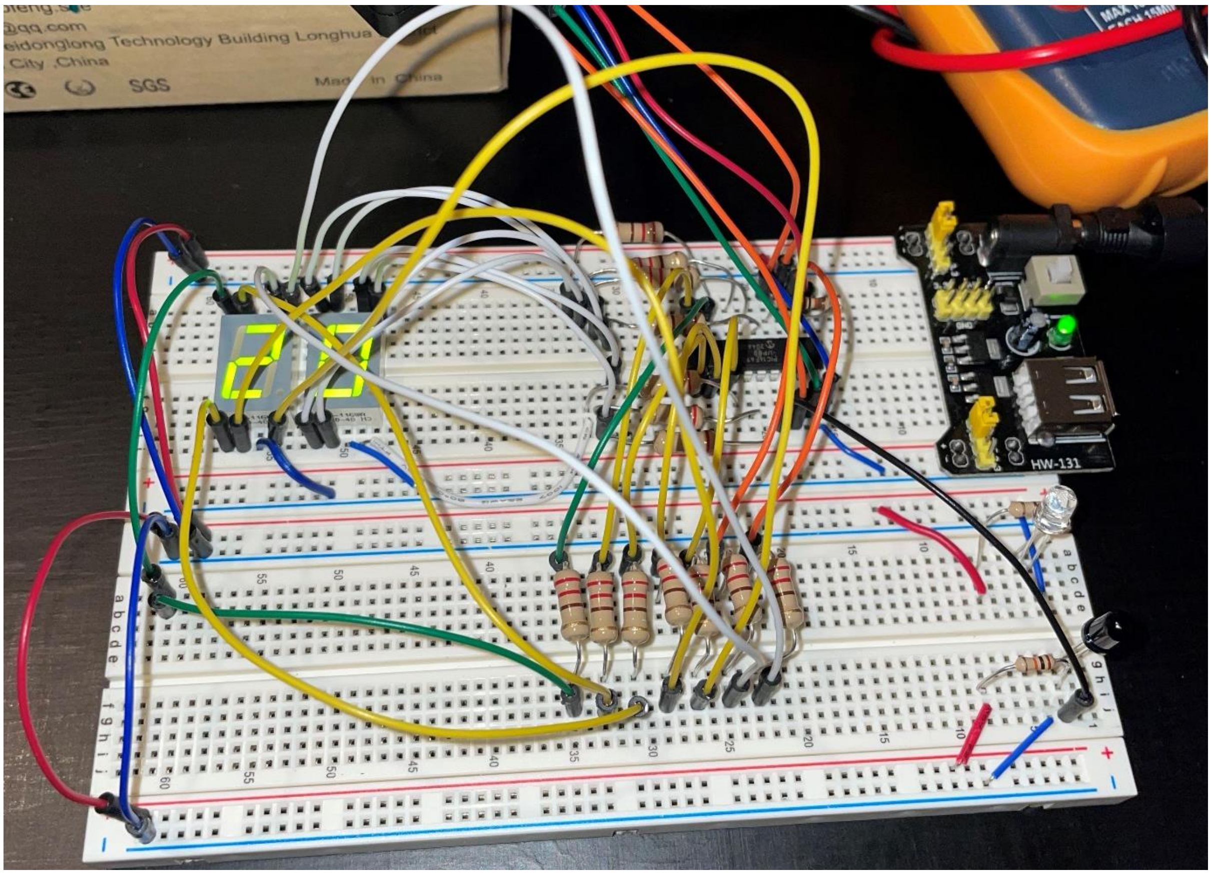
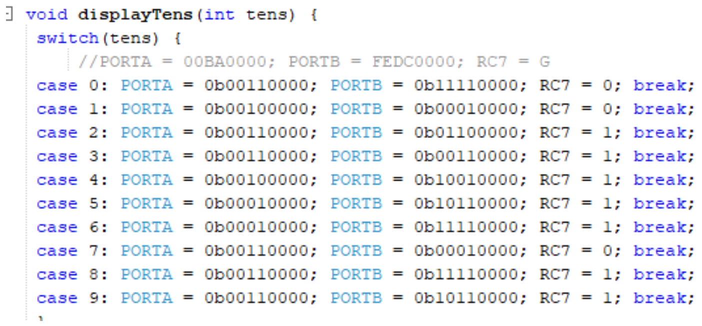


Figure 3 - Circuit for Part 3, Displaying Tens, and Units

- Firstly, the circuit was set up as per the given circuit diagram, which resulted in the circuit in Figure 3. The connections out of each port were colour coordinated so that it would be easier to see which pin is connected. The green cable going from the PIC into the 220Ω resistor is the one connected from RC7, the yellow ones were coming from port B, while the orange ones were coming from the two pins in port A. The same colour coordination was made when connecting the resistor to the LED, however, due to a shortage in orange cables, the connection from the resistor to the LED had to be made with a white cable for the port A pins.
- Once this was complete, it was required to write the diplayTens function which can be seen below in Figure 4.



```
void displayTens(int tens) {
 switch(tens) {
     //PORTA = 00BA0000; PORTB = FEDC0000; RC7 = G
 case 0: PORTA = 0b00110000; PORTB = 0b11110000; RC7 = 0; break;
 case 1: PORTA = 0b00100000; PORTB = 0b00010000; RC7 = 0; break;
 case 2: PORTA = 0b00110000; PORTB = 0b01100000; RC7 = 1; break;
 case 3: PORTA = 0b00110000; PORTB = 0b00110000; RC7 = 1; break;
 case 4: PORTA = 0b00100000; PORTB = 0b10010000; RC7 = 1; break;
 case 5: PORTA = 0b00010000; PORTB = 0b10110000; RC7 = 1; break;
 case 6: PORTA = 0b00010000; PORTB = 0b11110000; RC7 = 1; break;
 case 7: PORTA = 0b00110000; PORTB = 0b00010000; RC7 = 0; break;
 case 8: PORTA = 0b00110000; PORTB = 0b11110000; RC7 = 1; break;
 case 9: PORTA = 0b00110000; PORTB = 0b10110000; RC7 = 1; break;
```

Figure 4 - displayTens function used to set values of the pins connected to the second 7-segment display

- A comment was added at the beginning of the function which showed which bit of each port corresponded to a certain LED. This way it would be easier to know which ones had to be high or low for a certain number.
- Port A and B were given a value directly, however for the pin connected to Port C, only the required bit was changed using the RC7 variable declared in the xc.h file. This was done so that the value of the unit's display isn't affected since it is connected to Port C.
- A loop was then added to the code to ensure that all numbers were displayed correctly. This explains why there is a two on the Ten's display. The loop was as follows:
```
	for(int i=0; i<10; i++)
		displayTens(i);
```

Part 4 – Implement Event Counter

For the final part of this lab session, it was required to add in one final section onto the circuit, along side with some additional code. This would be the Interrupt Service Routine, which would cause the PIC to do something if an interrupt occurred. The additional circuitry can be seen on the bottom right of Figure 5 below. It consisted of a photodiode and an IR LED.

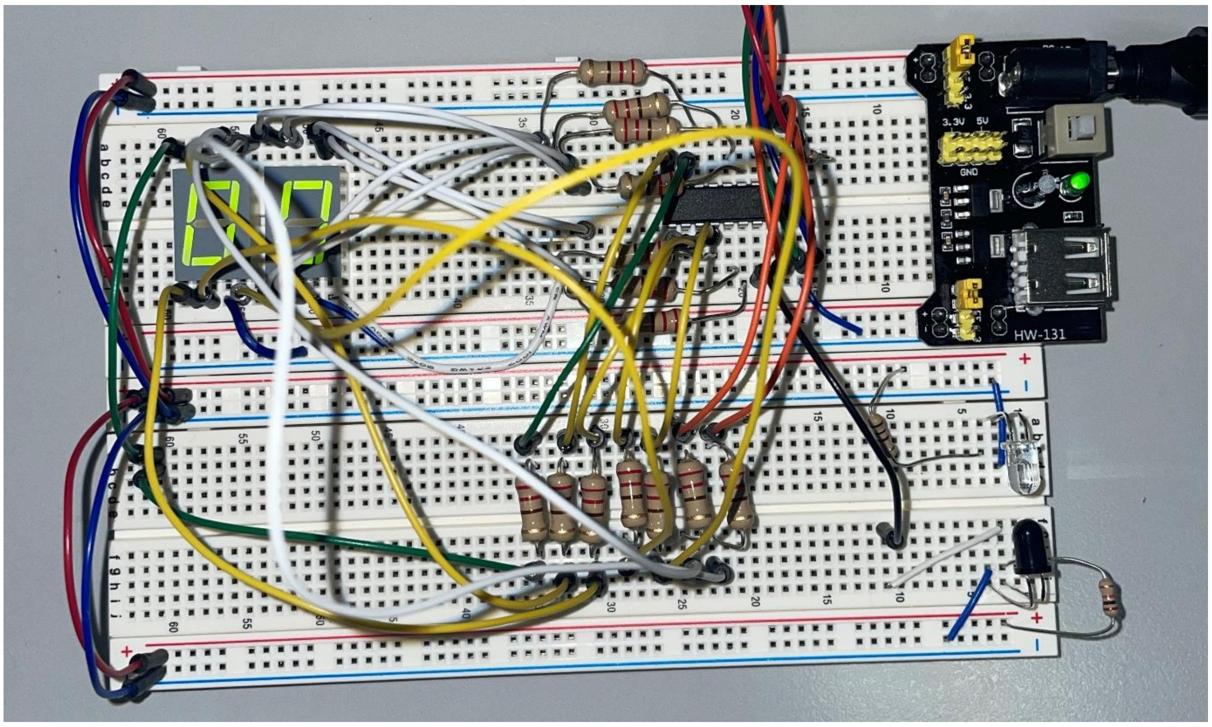
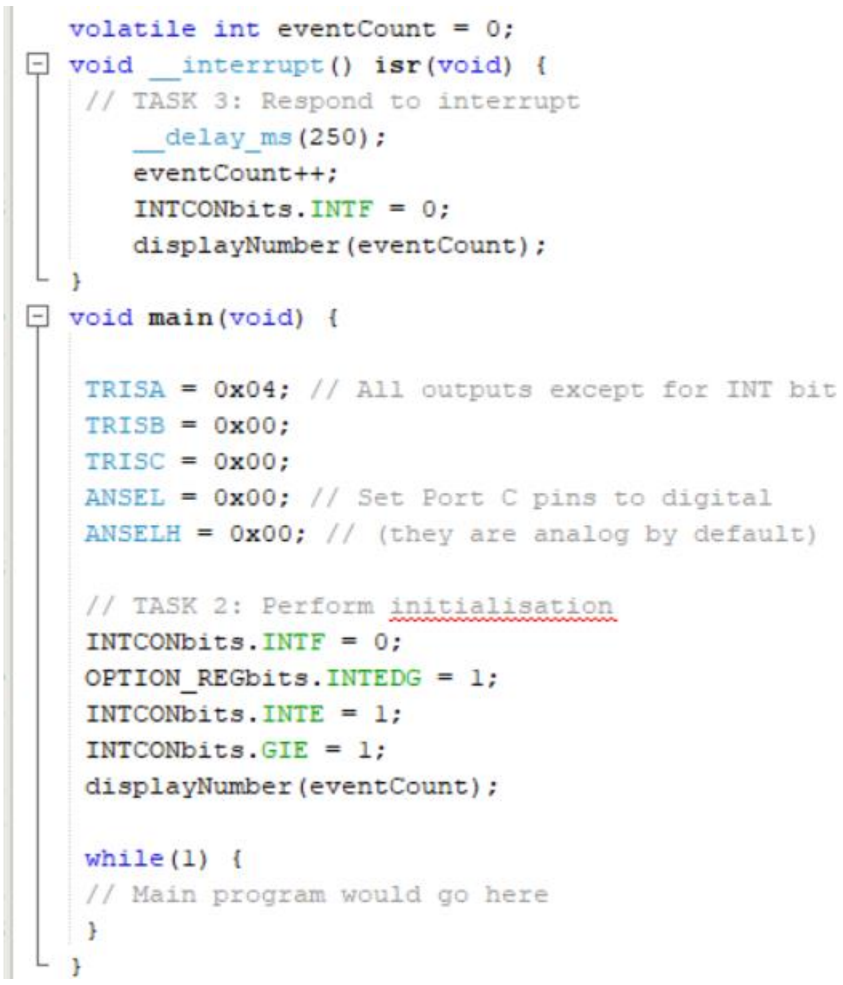


Figure 5 - Circuit for part 4

- The photodiode is the dark component in the bottom right of the circuit, while the clear component is the IR LED. The photodiode is connected in reverse-bias, and there is a connection made from its cathode to the INT pin of the PIC16F690 (pin 17 – RA2). This would be the signal for the PIC to enter its interrupt routine.
- The code for this part was just a continuation of code in part 3, with one function added; the interrupt routine, and some initialisation in the main function which would occur when the PIC turns on. The code can be seen in Figure 6 on the next page.
- The initialisations that had to be done was that, first, the RA2 pin, which was the interrupt had to be set as an input pin, as opposed to all other pins in port A which were outputs. This was done by setting the value of all bits, except bit index 2, to 0. Therefore, the value of TRISA would be 0x04. The other ports were both set to 0x00 as their pins were only used as outputs.
- The pins were then set as digital outputs using the ANSEL and ANSELH registers.



```
volatile int eventCount = 0;
void __interrupt() isr(void) {
 // TASK 3: Respond to interrupt
    __delay_ms(250);
    eventCount++;
    INTCONbits.INTF = 0;
    displayNumber(eventCount);
}
void main(void) {

 TRISA = 0x04; // All outputs except for INT bit
 TRISB = 0x00;
 TRISC = 0x00;
 ANSEL = 0x00; // Set Port C pins to digital
 ANSELH = 0x00; // (they are analog by default)

 // TASK 2: Perform initialisation
 INTCONbits.INTF = 0;
 OPTION_REGbits.INTEDG = 1;
 INTCONbits.INTE = 1;
 INTCONbits.GIE = 1;
 displayNumber(eventCount);

 while(1) {
 // Main program would go here
 }
}
```

Figure 6 - Final code for the interrupt service routine

- The final initialisations that had to be done were to do with the interrupt routine. Firstly, the INTF bit (bit 1) of the INTCON register was set to 0. This bit gets set when the interrupt occurs, so it had to be initialised as zero so that the interrupt function doesn't get called right as the PIC is turned on.
- Next, the INTEDG bit (bit 6) of the OPTION_REG register was set to 1. This indicated that an interrupt should occur on a rising edge, and not a falling edge at the interrupt pin.
- After this, the interrupt pin had to be enabled so that the PIC knows it is being used. This was done by setting the value of the INTE bit of the OPTION_REG as 1.
- The final initialisation done was the GIE bit of the OPTION_REG was also set as 1. This enabled all unmasked interrupts.
- Once all initialisations were done, it was required to write the interrupt function. This had two underscores before the interrupt word as this is the function called when the interrupt pin (RA2) is set high. Once this function was called, a delay of 250ms was added to allow for any noise to settle down and stabilise. The evenCount variable was then increased. This variable showed how many times the interrupt occurred and displayed it on the 7-segment displays via the displayNumber function from previous parts. Lastly, the Interrupt flag bit (INTF) inside the INTCON register was set to 0 so that the function doesn't loop indefinitely since the PIC doesn't automatically do this when an interrupt occurs.

Once the circuit was all set up and the program installed into the PIC, the interrupt routine was tested by putting a card between the IR LED and the photodiode. When this occurred, the number on the 7-segment display should be increased by 1, however when testing this was not happening. This was, in part, due to the positioning of the two components. It was required for them to be facing each other. This was fixed; however, the interrupt was still not working correctly, it only increased the count when measuring the voltage across the photodiode.

The solution for this was found by accident. When testing it one more time, the card accidently hit the IR LED and caused it to push back, and the count increased. It was concluded that the IR LED was not inserted into the circuit properly, and it was removed and inserted again, this time ensuring a proper connection. When this was complete, the circuit worked as it should. The final positioning of the IR LED and the photodiode can be seen in figure 7 below.

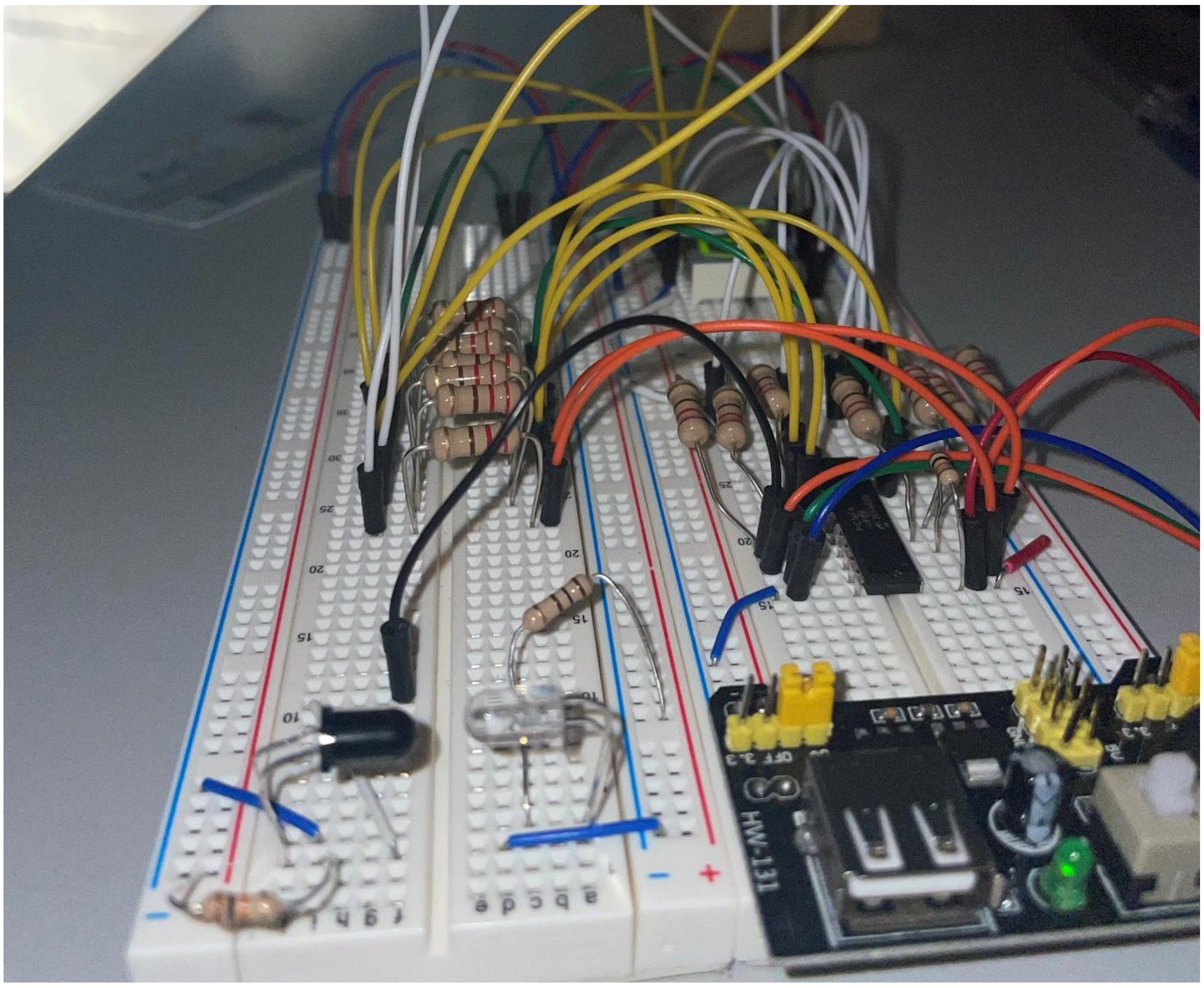


Figure 7 - Side view of the final circuit for part 4

Conclusion

From completion of this lab session, a series of new skills and lessons were learned.

- Learned how to use the PICKIT to transfer code onto the PIC16F690.
- Learned how to connect the software and hardware to make a small Embedded System.
- Learned how to use the three ports of the PIC16F690 to allow for digital inputs, or outputs.
- Used C language to program the circuit as it is closer to human language, and to shorten the number of instructions needed to be typed into the IDE.
- Learned how to implement interrupts into the system to allow for external sources to cause a change in the PIC.