I. Introduction:

The objective of this laboratory session is to acquaint students with the EasyPIC 7 Demonstration Board and the fundamental software tools required for programming in the course. The compact four-digit multiplexed seven-segment display is an electronic device utilized for visually representing numerical information. Its configuration enables the display of multiple digits with fewer connections, reducing resource requirements and simplifying circuitry. This type of display is frequently employed in digital applications such as timekeeping devices, timers, and temperature displays due to its efficient design, intuitive interface, and compatibility with microcontrollers. During the lab, our initial assignment is to program a microcontroller for a 3-digit ascending counter (ranging from 000 to 255) using a multiplexed display operating at a frequency of 1Hz. Subsequently, we are instructed to repeat the first part of the assignment to demonstrate a 4-digit ascending counter (ranging from 0000 to 9999) at the same frequency.

II. Procedure:

- ***Part A (3-digit up counter):***

In this section, we are tasked with developing a program for a microcontroller to control a 3-digit ascending counter that increments from 000 to 255 using a multiplexed display. The counter should advance at a rate of 1Hz, with the display update managed by a recurring interrupt that occurs every 4 milliseconds. The objective is to achieve seamless functionality and continuous refresh of counter values on the display.

A. MPLAB Code:

```
#include <p18cxxx.h>   // PIC18 Microcontroller family library

// 7-segment display codes for digits 0 to 9
char SScodes[] = {0x3F, 0x06, 0x5B, 0x4F, 0x66, 0x6D, 0x7D, 0x07, 0x7F, 0x6F};

// Variables for controlling interrupts and display states
unsigned char intcount, Qstates, i;
unsigned char counter;      // Counter variable to keep track of displayed value
char digits[3];   // Array to store BCD digits (hundreds, tens, units)

// Function prototypes
void Setup(void);

void main(void){
   Setup();     // Initialize microcontroller and peripherals
   while(1);    // Infinite loop (program logic will be handled in ISR)
}

void Setup(void){

   TRISD = 0X00;          // Set PORTD as output (for 7-segment display)
   ANSELD = 0X00;         // Disable analog functionality on PORTD

   TRISA &= 0xF8;         //RA 0,1,2 OUPUT
   ANSELA &= 0XF8;        //RA 0,1,2 DIGITAL

   INTCONbits.GIE = 1;    // Global Interrupt Enable
   INTCONbits.T0IE = 1;   // Timer0 Interrupt Enable

   i = 0;                 // Reset digit index
   intcount = 250;        // Initialize interrupt counter for timing
   T0CON = 0b11010100;    // Configure Timer0: 8-bit, prescaler 1:32
   TMR0L = 256 - 125;     // Load Timer0 with initial value (for 1ms timing)

   Qstates = 0b00000100;  // Start with the first digit

   // Convert the updated counter value to BCD for display
   digits[2] = counter%10;
   digits[1] = (counter/10)%10;
   digits[0] = (counter/100);
}

// Interrupt Service Routine (ISR) for Timer0
```

```
#pragma code ISR = 0x0008
#pragma interrupt ISR

void ISR(void){

  // Clear the Timer0 interrupt flag
  INTCONbits.T0IF = 0;
  TMR0L = 256 - 125;     // Reload Timer0 for next interrupt

  PORTD = 0x00;          // Turn off the display for refresh
  PORTA = Qstates;       // Activate the current 7-segment digit
  PORTD = SScodes[digits[i]]; // Display the appropriate digit on the 7-segment display

  Qstates >>= 1;         // Shift to the next digit (RA0 -> RA2)
  i++;                   // Move to the next BCD digit

  // If all 3 digits have been displayed, reset to the first one
  if(i == 3){
    i = 0;
    Qstates = 0b00000100; // Reset to display the first digit again
  }

  // Decrease intcount, reset when it reaches 0
  intcount--;
  if(intcount == 0)
  {
    intcount = 250;     // Reset the interrupt counter
    counter++;          // Increment the counter every 250ms

    // Convert the updated counter value to BCD for display
    digits[2] = counter%10;
    digits[1] = (counter/10)%10;
    digits[0] = (counter/100);

}
}
```

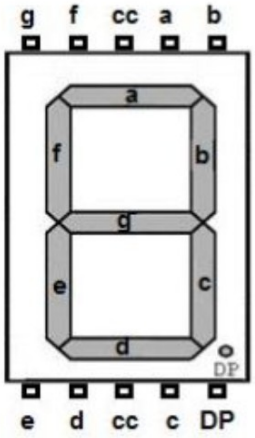
Explanation of the code:

1) *Libraries:*

- p18cxxx.h: This header file contains definitions specific to the PIC18 series microcontrollers.

2) Global variables:

- SScodes[]: This array stores the 7-segment display codes for each digit from 0 to 9. Here, we need to represent small integer values, so we selected the type of this variable as "char" since it typically represents 1 byte (8 bits) of data storage, which is sufficient to cover the required values.



Number	g f e d c b a	Hex Code
0	0111111	3F
1	0000110	06
2	1011011	5B
3	1001111	4F
4	1100110	66
5	1101101	6D
6	1111101	7D
7	0000111	07
8	1111111	7F
9	1001111	4F

Figure 1: Display numbers on a 7-segment display in common cathode configuration

- i, Qstate, IntCount, counter: These are global variables utilized for controlling the program flow, counting interrupts, and managing display states. The data type "unsigned char" is selected for these variables because it can represent values ranging from 0 to 255, leveraging all 8 bits to represent positive integers, which is suitable for counting or storing state information that does not involve negative values.
- digits[3]: An array to hold the digits to be displayed.

3) Calculations:

a) The programming code sets up a recurring interrupt every 4 milliseconds to refresh the display in a way that is imperceptible to the user. To accomplish this, 4 milliseconds is converted to 4000 microseconds. It is found that the minimum number of cycles is attained when a prescaler of 32 is used, resulting in 4000/32 = 125 cycles. As a result, the 1:32 prescaler option is selected. Since 125 is less than 255, the 8-bit option for TMR0 can be utilized, and accordingly, T0CON is set to "0b11010100".

b) To achieve the objective of incrementing the counter every 1 second, an alternative approach involves utilizing TMR0L, which is already being used for screen refreshing. Given the desired time interval of 1 second (T = 1000 ms) and the fact that a periodic interrupt occurs every 4 milliseconds, the number of interrupts required is calculated: 1000 ms divided by 4 ms equals 250 interrupts. Therefore, every 250 interrupts can be considered equivalent to 1 second, providing a straightforward method for incrementing the counter.

$$T = 1\,s = 1000\,ms$$

$$4\ ms \rightarrow 1 \text{ interrupt}$$

$$\rightarrow 1000ms/4ms = 250 \text{ interrupts}$$

4) Functions:

I. Setup Function:

a) Configure PORTD as digital outputs for connection to the segments of the 7-segment display:

‘TRISD = 0x00’ and **‘ANSELD = 0x00’**

b) Configure bits 0, 1, and 2 of PORTA as digital outputs connected to LEDs, while maintaining the initial state for other bits:

‘TRISA &= 0xF8’ and **‘ANSEL &= 0xF8’**

c) Enable global interrupts and Timer0 interrupt:

‘INTCONbits.GIE = 1’ and **‘INTCONbits.T0IE = 1’**

d) Configure Timer0 (TMR0) with the binary value **‘0b11010100’** to ensure effective timing for interrupts (4 *ms*).

e) Set the initial value for Timer0 to achieve a 4 ms interval:

‘TMR0L = 256 – 125’ (calculated as 4000/32).

f) Set **‘IntCount’** to 250.

g) Enable the most significant digit (MSD) of the 3-digit counter for the multiplexed display:

‘Qstate = 0b00000100’

II. The Interrupt Service Routine (ISR) function:

a) Memory Location and Interrupt Handling:

The Interrupt Service Routine (ISR) is situated at memory address 0x0008, and its primary function is to handle Timer0 overflow interrupts. Upon entering the ISR, the first step is to clear the Timer0 interrupt flag. By performing this flag-clearing operation, the ISR ensures that each interrupt is processed only once, thereby preventing repeated interruptions and preserving the priority of interrupts.

b) Display Update Preparation:

Prior to updating the display, PORTD is initialized to 0x00, effectively clearing any existing display data and preparing it for the presentation of the new digit. Following this, the corresponding 7-segment code for the current digit is assigned to PORTD. These 7-segment codes are retrieved from the SScodes[] array based on the current digit index 'i'. The routine then progresses through the digits by incrementing the index 'i' and shifting 'Qstate', which governs multiplexing, to select the next digit.

c) Digit Cycling and Reset:

When all digits have been displayed, i.e., when the index 'i' reaches 3, it is reset to '0' to commence the next cycle of digit display. Furthermore, 'Qstate' is reset to its initial state "0b00000100" to guarantee that the next cycle starts from the most significant digit, thus maintaining the correct sequence of digit display.

d) Counter Incrementation:

Once 1 second has elapsed, as indicated by 250 cycles having passed (IntCounter reaching 0), the counter is incremented by 1. Immediately after, IntCounter is reset to 250, preparing it for the next counting interval, thereby maintaining the continuous incrementation of the counter at a rate of 1 second.

e) Binary to BCD Conversion:

- digits[2] = counter % 10; This retrieves the ones digit (right-most digit) of ‘counter’ since ‘% 10’ gives the remainder when ‘counter’ is divided by 10.

- digits[1] = (counter / 10) % 10; Here, ‘counter’ is first divided by 10 (which removes the ones digit), and then ‘% 10’ extracts the new right-most digit—this becomes the tens digit of the original number.

- digits[0] = (counter / 100); Dividing ‘counter’ by 100 removes the last two digits, leaving the hundreds digit (or more if the number is greater than 999, but the code implicitly expects a 3-digit number).

B. Proteus simulation results:

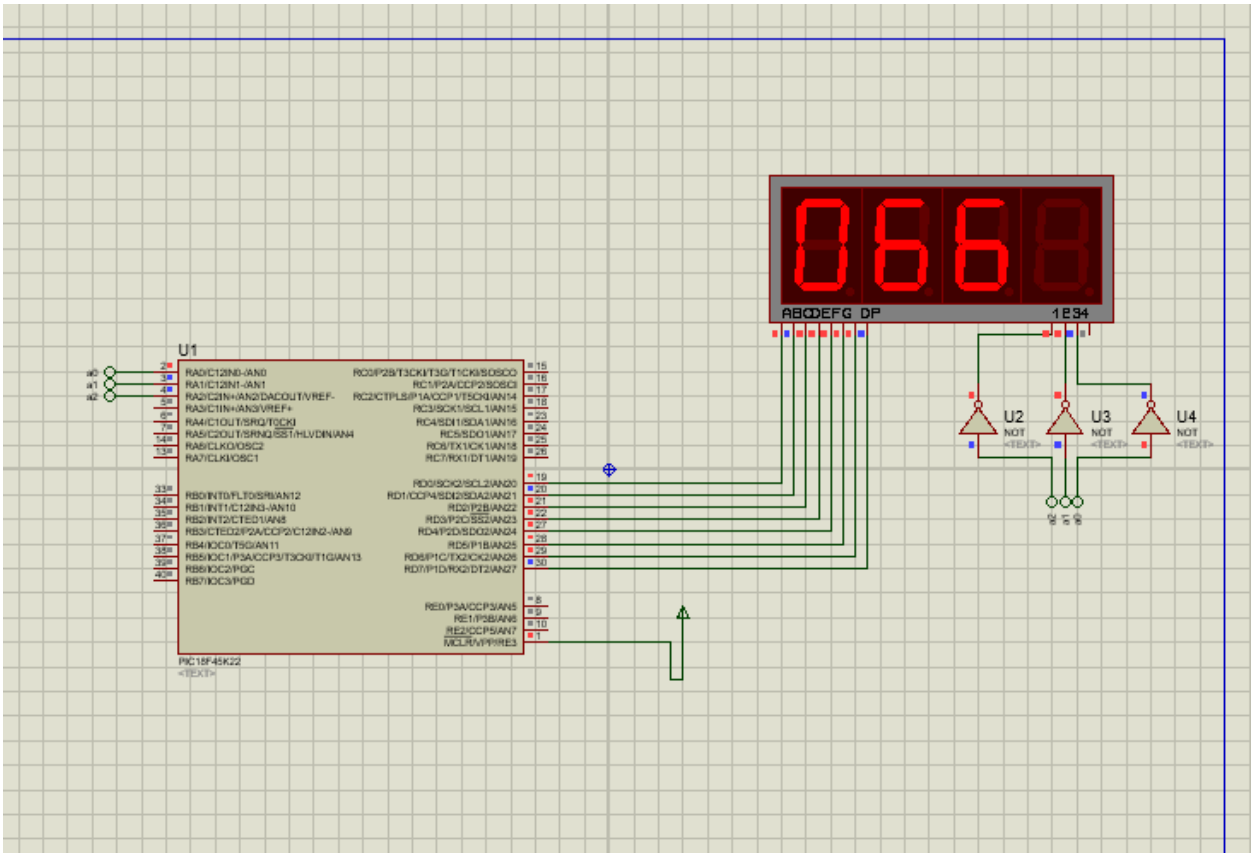


Figure 2: Simulation of the 3-digit up counter on Proteus

Since this multiplexed display is a common cathode display, we need to put an inverter for the output signal to work correctly.

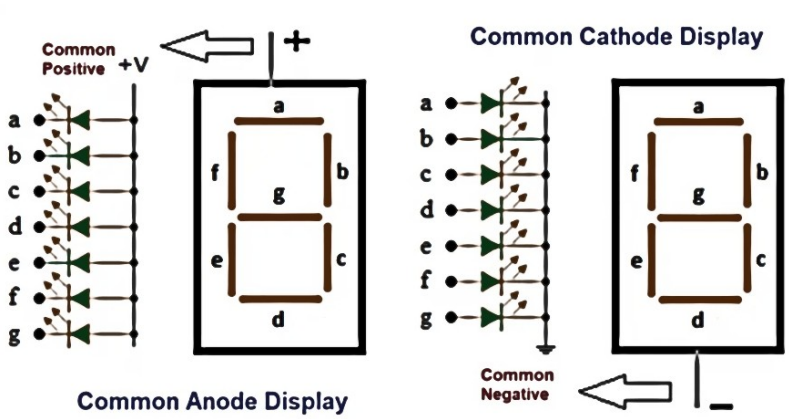
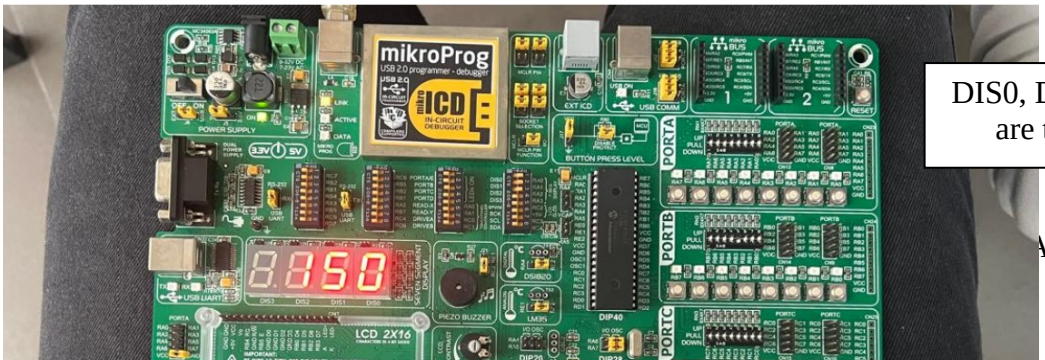


Figure 3: Common Cathode/Anode Display

C. Demo Board Implementation:

For this part, the EasyPIC v7 Demo Board was utilized. To program the code onto the board, the necessary software, mikroProg Suite for PIC, was employed. On the demo board, it is necessary to manually enable the transistors for the desired digit of the 7-segment decoder. In this case, DIS0, DIS1, and DIS2 were turned on, which correspond to RA0, RA1, and RA2, respectively.Once the code is written on the board, the display starts showing the counted-up values from 0 until it reaches 255 and starts over.



DIS0, DIS1 and DIS2 are turned **ON**.

Atallah

Figure 4: Displaying the number 150 on the 7-segment display (testing the 3-digit up counter)

- ***Part B (4-digit up counter):***

In this section, the task is to design a 4-digit ascending counter (counting from 0000 to 9999) utilizing a multiplexed display. The program should operate at a frequency of 1Hz, with the display refresh managed by a periodic interrupt that occurs every 4 milliseconds. The objective is to achieve smooth and real-time display of continuously updating counter values, ensuring a seamless and uninterrupted visualization of the counting process.

A. MPLAB Code:

```
#include <p18cxxx.h>   // PIC18 Microcontroller family library


// 7-segment display codes for digits 0 to 9
char SScodes[] = {0x3F, 0x06, 0x5B, 0x4F, 0x66, 0x6D, 0x7D, 0x07, 0x7F, 0x6F};

// Variables for controlling interrupts and display states
unsigned char intcount, Qstates, i;
unsigned int counter;     // Counter variable to keep track of displayed value
char digits[4];   // Array to store BCD digits (thousand, hundreds, tens, units)

// Function prototypes
void Setup(void);
```

```
void main(void){
   Setup();    // Initialize microcontroller and peripherals
   while(1);   // Infinite loop (program logic will be handled in ISR)
}

void Setup(void){

   TRISD = 0X00;          // Set PORTD as output (for 7-segment display)
   ANSELD = 0X00;         // Disable analog functionality on PORTD

   TRISA &= 0xF0;         //RA 0,1,2,3 OUPUT
   ANSELA &= 0XF0;        //RA 0,1,2,3 DIGITAL

   INTCONbits.GIE = 1;    // Global Interrupt Enable
   INTCONbits.T0IE = 1;   // Timer0 Interrupt Enable

   i = 0;                 // Reset digit index
   intcount = 250;        // Initialize interrupt counter for timing
   T0CON = 0b11010100;    // Configure Timer0: 8-bit, prescaler 1:32
   TMR0L = 256 - 125;     // Load Timer0 with initial value (for 1ms timing)

   Qstates = 0b00001000;  // Start with the first digit

      // Convert the updated counter value to BCD for display
      digits[3] = counter%10;
      digits[2] = (counter/10)%10;
      digits[1] = (counter/100)%10;
      digits[0] = (counter/1000);
}

// Interrupt Service Routine (ISR) for Timer0
#pragma code ISR = 0x0008
#pragma interrupt ISR

void ISR(void){

   // Clear the Timer0 interrupt flag
   INTCONbits.T0IF = 0;
   TMR0L = 256 - 125;     // Reload Timer0 for next interrupt

   PORTD = 0x00;          // Turn off the display for refresh
   PORTA = Qstates;       // Activate the current 7-segment digit
   PORTD = SScodes[digits[i]]; // Display the appropriate digit on the 7-segment display
```

```
  Qstates >>= 1;        // Shift to the next digit (RA0 -> RA3)
  i++;                  // Move to the next BCD digit

  // If all 3 digits have been displayed, reset to the first one
  if(i == 4){
     i = 0;
     Qstates = 0b00001000; // Reset to display the first digit again
  }

  // Decrease intcount, reset when it reaches 0
  intcount--;
  if(intcount == 0)
  {
     if(counter==9999){   // Segment explained below
     counter = 65535;
     }

     intcount = 250;     // Reset the interrupt counter
     counter++;          // Increment the counter every 250ms

     // Convert the updated counter value to BCD for display
     digits[3] = counter%10;
     digits[2] = (counter/10)%10;
     digits[1] = (counter/100)%10;
     digits[0] = (counter/1000);

}
}
```

B. The modifications:

1. `unsigned int counter;`

   Since the variable 'counter' requires storing values from 0 to 9999, the data type "unsigned char" is no longer suitable because it has a limited range of 0 to 255 (decimal) on the C18 compiler. Therefore, "unsigned int" is a suitable choice as its range spans from 0 to 65535 (decimal), and 9999 falls well within this range, making it an appropriate data type for the 'counter' variable.

2.
```
char digits[4];
```

We needed an additional bit to display the 4th bit, so we changed the declaration from digits[3] to digits[4] so we now have 4 elements instead of 3.

3.
```
  TRISA &= 0xF0;          //RA0, 1, 2, 3 OUTPUT
  ANSELA &= 0xF0;         //RA0, 1, 2, 3 DIGITAL
```

These lines configure the first 4 pins of PORTA as outputs and digital pins.

4.
```
Qstate = 0b00001000;
```

The 'Qstate' represents the transistors, acting as an indicator of which segments should be ON to display a specific digit, with an initial value of "00001000" that activates the most significant digit, and we are utilizing only 4 bits from 'Qstate'.

5.
```
// Convert the updated counter value to BCD for display
     digits[3] = counter%10;
     digits[2] = (counter/10)%10;
     digits[1] = (counter/100)%10;
     digits[0] = (counter/1000);
```

The code change was made to accommodate a 4-digit number by updating the indices and division factors to correctly extract the individual digits from the counter variable, with each digit now being placed in the correct position in the digits array.

6.
```
// If all 3 digits have been displayed, reset to the first one
 if(i == 4){
     i = 0;
     Qstates = 0b00001000; // Reset to display the first digit again
  }
```

Once all digits have been displayed (i.e., when ‘i’ reaches 4), the index ‘i’ is reset to ‘0’ to initiate the next cycle of digit display. Additionally, ‘Qstate’ is reset to its initial state “0b00001000” to ensure that the next cycle begins from the most significant digit.

```
7.  if(counter==9999)
    counter = 65535;
```

The 4-digit up counter should only count till 9999 then resets to 0, even though the type ‘unsigned int’ variable has a wider range. This is done by adding this if block. The counter is not reset by writing counter = 0; but rather by writing counter = 0 is not used, instead counter = -1 is not used either, counter is reset by writing counter = 0 in a way that the next statement counter++ will make it 0 and then display 0 on the 7-segment display, so the actual reset is done by writing counter = 65535; This is done since the if block is followed by an increment statement counter++; If counter = 0; was used, the number ‘0’ won’t be displayed on the 7-segment display, because counter will be 0 and then immediately 1.

C. Proteus simulation results:

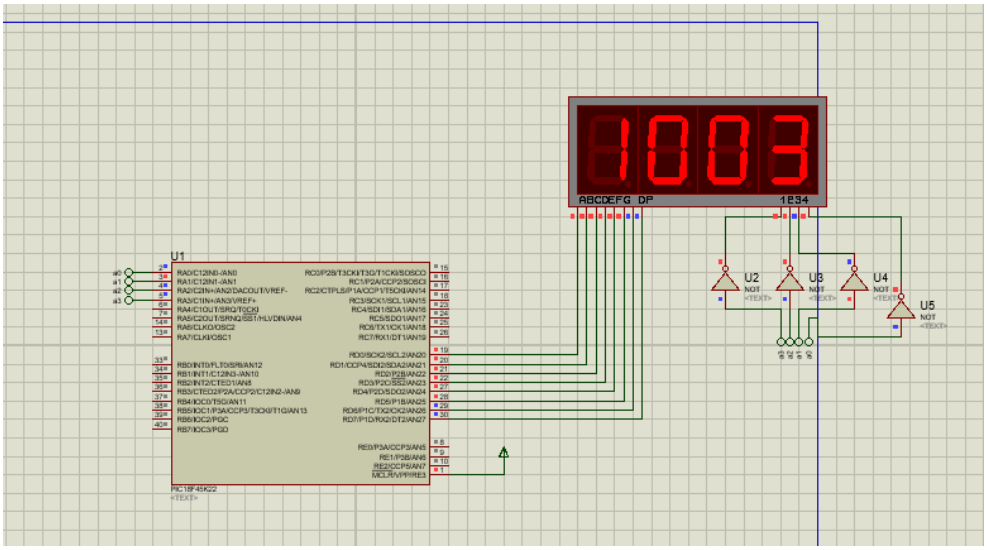


Figure 5: Simulation of the 4-digit up counter on Proteus

Knowing how the common cathode display works, an inverter was added on RA3, so the output signal is inverted to work correctly with the display.

D. Demo Board Implementation:

Our project utilizes the EasyPIC v7 Demo Board, requiring the mikroProg Suite™ for PIC® to program it. The board features a 7-segment display controlled by manually enabling specific transistors. In our case, transistors DIS0, DIS1, DIS2, and DIS3 **(corresponding to pins RA0, RA1, RA2, and RA3)** are used to activate individual digits. The written code is successfully implementing a 4-digit up counter. It also displays the counted-up values from 0 until it reaches 9999 and starts over.

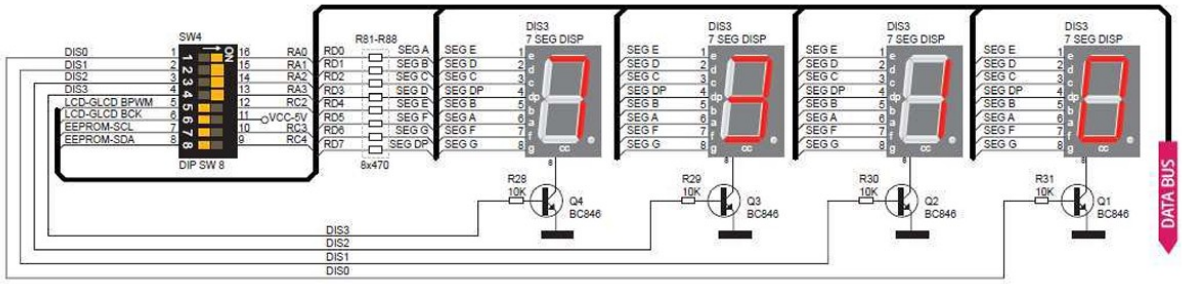
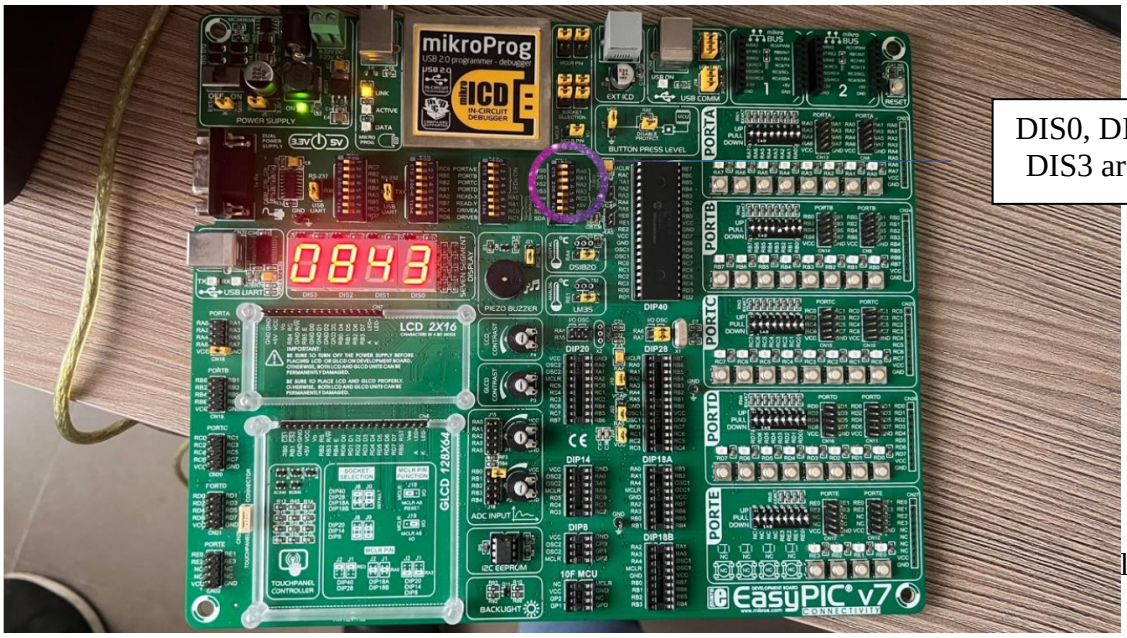


Figure 6: Multiplexed display on demo board.



DIS0, DIS1, DIS2 and DIS3 are turned **ON**.

lah

Figure 7: Displaying the number 0843 on the 7-segment display (testing the 4-digit up counter)

III. Conclusion:

In conclusion, two up-counters were successfully designed and implemented: a 3-digit up counter ranging from 0 to 255 and a 4-digit up counter ranging from 0 to 9999, thus achieving the objectives of the experiment, which involved initial code development and compilation in the MPLAB X IDE, hardware construction on PROTEUS, and final implementation on the EasyPIC 7 Demonstration Board, providing hands-on experience with its hardware components and their connections.