Lab 2: Simple LEDs and 7-Segment Displays

Introduction

Single Sentence Overview

Learn how to program your microcontroller to use simple LED lights and more complex 7-segment displays.

Overview

This two-part lab is designed to introduce the student to a debugging-centric method for developing a program on a microcontroller.

1. First, the student will manipulate registers using a short program in C to see immediate changes in the state of the microcontroller and the hardware (LED lights) connected to it.
2. Second, the student will write a program to control the LED light output of a more complex 7-segment display.

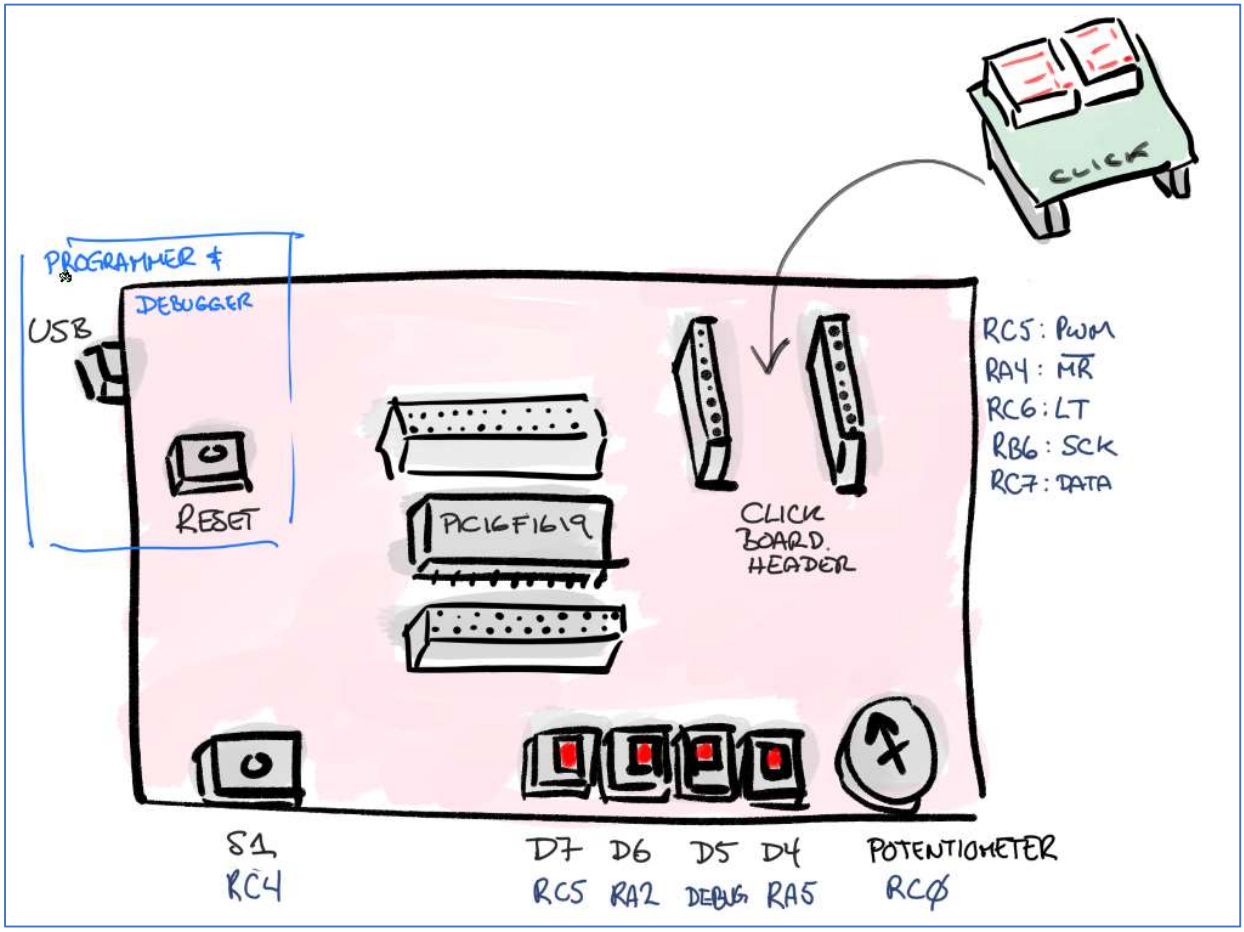

Figure 1 Use the LEDs (D7, D6 and D4), the Switch (S1) and the Click board to learn how to write simple programs for your PIC16.

Learning Outcomes

The student will know how to

1. Write a program to flash LEDs that are directly connected to a microcontroller's output pins.
2. Write a more complex program to control 16 LEDs in a pair of seven-segment displays via a hardware shift register.

Success Criteria

Review the grading rubric and evaluate how it relates to you

1. Demonstrating during the TP lab session that you can toggle the states of two simple LEDs on the Curiosity Board
2. Demonstrating during the TP lab session that you can control the output of two seven segment displays using a pair of shift registers.

Grading Rubric

During the TP lab session make sure to conduct all the required demonstrations to the lab instructor. Also, submit a document with the answers to the "Stimulate" and "Explore" questions.

- *Part 1:*
  - *Stimulate questions: 0 both answers incorrect; 1 if one answer is correct; 2 if both answers are correct.*
  - *Explore questions: 0 both answers incorrect; 1 if one answer is correct; 2 if both answers are correct.*
  - *Demonstration 1: 0 if not attempted; 2.5 points if partially attempted; 5 points working demonstration*
  - *Demonstration 2: 0 if not attempted; 2.5 points if partially attempted; 5 points working demonstration*
- *Part 2:*
  - *Question: 0 if incorrect; 1 if correct.*
  - *Demonstration 1: 0 if not attempted; 5 points if partially attempted; 10 points working demonstration*

Prerequisites

Watch a video on the bit-shifting process and the 7-seg Click Board: https://youtu.be/DtNEfVMrxNI

Figure 2 Watch the video on the 7-segment display. https://youtu.be/DtNEfVMrxNI

More Resources and Information

- Curiosity board getting started PDF: [https://bit.ly/2PYjtzE]
    - The LED D4 on RA5 needs to be configured using configuration bits. See this blog post: [*https://bit.ly/2DwjZla]*
- PIC16F1619 datasheet [https://bit.ly/2OsQtQ1]
- MikroE 7-segment Click Board datasheet: [https://bit.ly/2MY0LGx]
- Using a 7-segment display with the Curiosity board:[https://youtu.be/DtNEfVMrxNI]
- New logic operators (ISO646):[http://www.cplusplus.com/reference/ciso646/]

Part 1: Turn on two LEDs

Single Sentence Overview

Write a program that turns on LED D4 (RA5) and then D6 (RA2) *including bit shifting*.

Background material

To get started writing a program for any microcontroller, you should generally look at two documents:

1. The board's schematic (e.g. https://bit.ly/2PYjtzE)
2. The chip's datasheet (e.g. https://bit.ly/2OsQtQ1)

The board schematic helps narrow down the chip's pins that are of direct importance to you. You then use the labels to search through the chip datasheet for hints as to which registers need direct manipulation by the debugger. Usually, these are:

1. Port input value register (e.g. PORTA, PORTB or PORTC)
2. Data Direction register (e.g. "Tristate": TRISA, TRISB or TRISB)
3. Function selection (e.g. "Analogue Select": ANSELA, ANSELB or ANSELC)
4. Port digital output value register (e.g. "Latch": LATA, LATB or LATC)

You typically initialize by setting up (1) to (3) first, and then change values using the register in (4). You can perform all of these steps in Assembler programming, C programming, direct debugger register manipulation, or a combination of any of these three.

In Part 1 of this lab, you will write a C program, execute it and monitor as required using the debugger.

Figure 3 The Curiosity board has four LEDs that can be turned on and off. While D5, the debug LED, is not available to you, you can modify the state of the other three using the Latch registers for Ports A and C (LATA and LATC). ***Warning****: LED D4, on RA5, cannot be used unless you change the "configuration bits" as described at https://bit.ly/2DwjZla and listed in Lab 2's Appendix.*

Bit shifting is the processes of moving shifting is an important operation that can be done either in C or in Assembler, as shown in the table below.

Table 1 Bit shifting in Assembler and C.

	Assembler	*C*	*Explanation*	*Ex. Assembler*	*Ex. C*
Shift Left (Rotate Left)	*RLF*	*<<*	*Assign a value (0b00110000) to LATC, then shift it left by two bits to result in 0b11000000.*	* /* 1: Assign value */ asm("BANKSEL LATC"); asm("MOVLW 0b00110000"); asm("MOVWF LATC"); /* 2: shift twice */ asm("RLF LATC, f"); asm("RLF LATC, f");*	*/* 1: Assign value */ LATC = 0b00110000; /* 2: shift twice */ LATC = LATC << 2;*
Shift Right (Rotate Right)	*RRF*	*>>*	*Assign a value (0b00110000) to LATC, then shift it right by three bits to result in 0b00000110.*	*/* Step 1: Assign value */ asm("BANKSEL LATC"); asm("MOVLW 0b00110000"); asm("MOVWF LATC"); /* Step 1: shift 3 times */ asm("RRF LATC, f"); asm("RRF LATC, f"); asm("RRF LATC, f");*	*/* 1: Assign value */ LATC = 0b00110000; /* 2: shift 3 times */ LATC = LATC >> 3;*

While you can assign a single bit in an 8-bit word using the PIC16 Assembler mnemonic BSF (bit set) or BCF (bit clear) to make a bit 1 or 0, respectively, C does not provide that functionality natively. Instead, we must assign entire 8-bit words at a time. Best practice is to read the existing values in an 8-bit register and to use a logic function against the existing value in a way that only changes the single bit we are interested in. Similar to painting a wall in a house where we use masking tape to cover things we don't want to paint, we call the bit-hiding "masking" as it hides the uninteresting bits and only exposes the interesting one.

While shifting a bit can be a quick way to move data (or to divide or multiply by 2) there are four practical operations in C that use shifting:

Table 2 Common C bitwise operations[1], with equivalent modern alternate operator spelling (ISO646 variant).[2]

	Major Bitwise Logic Operation	**Generic form** *(Replace myvar or mybit with your own variable or register and n with the bit number)*	**Explanation of Example**	**Typical** *(explicit)*	**Typical** *(compact with shifting)*	**"Alt ISO646"** *(explicit)*	**"Alt ISO646"** *(compact with shifting)*
***Set**: Assign a bit to 1*	*Or*	`myvar |= (1 << n);`	*Make Bit 6 a "1".*	`myvar = myvar | 0b01000000;`	`myvar |= (1 << 6);`	`myvar = myvar bitor 0b01000000;`	`myvar or_eq (1<<3);`
***Clear**: Assign a bit to 0*	*And, Complement*	`myvar &= ~(1 << n);`	*Make Bit 5 a "0".*	`myvar = myvar & 0b11011111;`	`myvar &= ~(1 << 5);`	`myvar = myvar bitand 0b10111111;`	`myvar and_eq compl(1<<5);`
***Toggle**: Invert a bit*	*Xor*	`myvar ^= (1<<n);`	*Change Bit 4 to its opposite.*	`myvar = myvar ^ 0b01000000;`	`myvar ^= (1 << 4);`	`myvar xor_eq 0b00100000;`	`myvar xor_eq (1UL << 5);`
*Examine **status** of a bit*	*And*	`mybit = (number >> n) & 1;`	*What is the value of bit 5?*	-	`mybit = (myvar >> 5) & 1;`	-	`bit = (myvar >> 5) bitand 1;`

[1] Standard bitwise operations are explained here: https://en.wikipedia.org/wiki/Bitwise_operations_in_C

[2] The alternate operator spelling is common in modern C++. To use the alternate operator spelling in C make sure to use **#include <iso646.h>** in your code.

Stimulate

Answer the following questions in your lab report.

1. What PIC16 Port is LED D4 connected to?
2. What PIC16 port is LED D7 connected to?

Explore

Answer the following questions in your lab report.

1. What specific port pin is LED D4 connected to?
2. What is main power voltage (Vdd) of the Curiosity board ?

Problem

In Lab 1 you used the SFR view to modify the configuration of the pins and then modify the voltage output of the pins so that the LEDs turn on and off. You also wrote a simple program in C that could turn on LEDs. Here, you will repeat that second task, but you will use bit-shifting to control the contents of the Latch Register, LATA. The first time you run the program you should verify the contents of the latch register using the SFR view in a debugging session.

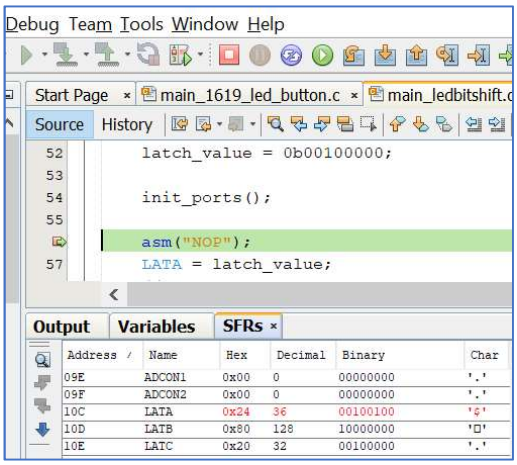

Figure 4 The Special Function Register (SFR) View during debugging is helpful for viewing current state and changes to hardware registers like LATA.

First, initialize the PORT, TRIS and ANSEL registers. While you can set all the bits to 0 in all three registers, it's best to note that:

- RA2, RA5 and RC5 are outputs to the LEDs
  - RA5 needs to be specially *configured*, as described in https://bit.ly/2DwjZla and in the Lab 2 Appendix.
- RC4 is an input from the switch S1.
- RC5, RA4, RC6, RB6 and RC7 are all outputs to the MikroE Click board.

The PORT bits for all of these should be set to 0. The ANSEL bits for all of these should be set to 0 (making them digital). The TRIS bits for output pins should be 0, while the TRIS bits for input pins should be 1.

How does this work in C in MPLAB X? For an *imaginary* set of registers of an *imaginary* 8-bit PORT D that was completely digital, with bits 0 to 3 as inputs and 4 to 7 as outputs, we would set:

```
PORTD = 0b00000000;    // assign zeros to all 8 bits
ANSELD = 0b00000000;   // All 8 bits are digital
TRISD = 0b00001111;    // bits 7-4: output; 3-1 input.
```

You can see an Assembler equivalent in Example 12-1 in the PIC16F1619 datasheet (**https://bit.ly/2OsQtQ1**). Next, you're going to turn LED D4 and D6 on and off. These are connected to Port A Bit 5 ("RA5") and Port A Bit 2 ("RA2), respectively. To modify the voltage on RA2 and RA5 then change Bit 2 and Bit 5 of LATA ("Latch Port A").

Again, for an imaginary Port D, if we want to set Bits 7 and 5 to "high voltage" and 6 and 4 to "ground" then we would assign the following command to the latch:

```
LATD = 0b10100000;     // bits 7 & 5 are high. Others low.
```

In Lab 1, you showed how you could write a simple program to change the LEDs on the Curiosity board. Now, do the same thing, but by creating an 8-bit "char" variable and then assigning it to the latch. Near the top of your main function create the variable and assign it a value such that RA2 will be able to light up. For instance, if I wanted to light up bit 7 of Port D:

```
char myvariable;            // create an 8bit variable
myvariable = 0b10000000;    // bit 7 is 1; others 0.
```

and the I would assign that value to Port D's Latch:

```
LATD = myvariable;          // give Latch D a value.
```

Write a program that uses the method above to assign values to either RA2 or RA5 to light up an LED on the Curiosity board. Assign one value that lights up RA2's, then change the value in `myvariable` and light up RA5's LED. Use the debugger to step through your program so that you can verify that the variable is changing value and that you can see the LEDs changing. Because you are using the debugger to halt the program you do not need to use a loop or delays in your code.

Note that RA5 (LED D4) is special & needs to be configured properly. Refer to https://bit.ly/2DwjZla & the Lab 2 Appendix about the configuration bits.

- ***All groups**: demonstrate to the lab instructor that you can step through your code and turn on and off the D6 and D4 LEDs as described.*

Next, let's look at bit-shifting. First assign a value to `myvariable` such that the LED attached to RA5 lights up. Then, shift the contents of `myvariable` to the left three times using a bit shift operator.

- ***All groups**: demonstrate to the lab instructor that you can step through your code and turn on and off the D6 and D4 LEDs as described.* This time use ***bit shifting*** to change the value of the variable prior to assigning it to the latch.

Part 2: Lighting up the 7-segment displays

Single Sentence Overview

Write a program in C that will shift in 16 bit sequences to the MikroE 7-segment board for display on a pair of LED-based 7-segment displays.

Background material

The PIC Curiosity board has an interface for the MikroE Click series of boards. We will use a 7-segment display board from MikroE in this part of the lab, as illustrated in the figure below and in this video:

- Video: https://youtu.be/DtNEfVMrxNI

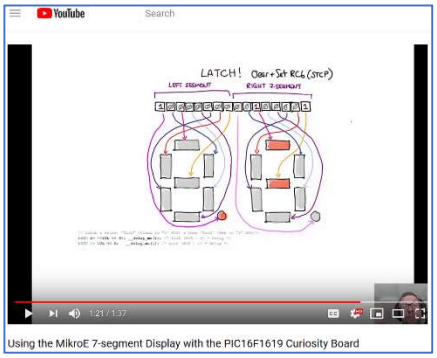

Figure 5 Watch this video: https://youtu.be/DtNEfVMrxNI

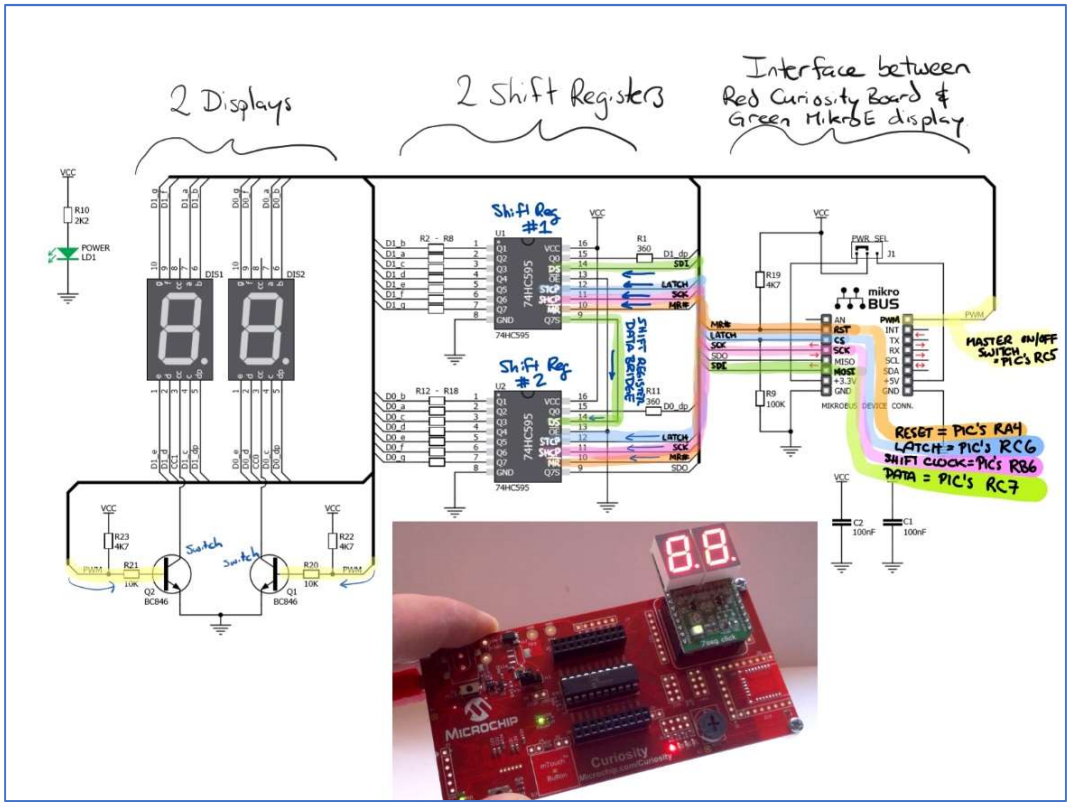

Figure 6 The two 7-segment displays on the green MikroE 7-seg Click board interfaces to the PIC on the red Curiosity board via the PIC's RA4, RB6, RC5, RC6 and RC7 pins. A video illustrates its usage: https://youtu.be/DtNEfVMrxNI. Schematic courtesy of Mikroelectronika.

Note that to get the program working for this portion of the lab you will need to control the following bits:

- RC5, RA4, RC6, RB6 and RC7 are all outputs to the MikroE Click board.

As per the video, https://youtu.be/DtNEfVMrxNI, You will need to "shift" sixteen values into the "7Seg" board's shift registers by

1. First, **loading one bit of data** for the 7-segment shift register
    a. Make RC7 a 1 ("set") or 0 ("clear") on the RC7 "data" wire.
2. Then, making the **clock signal** for the 7-segment shift register
    a. Make R**B**6 a 0 ("clear"), *pause*, then make R**B**6 a 1 ("set"), then *pause*.
    b. Use the __delay_ms() function to pause.
3. ***Repeat*** Steps 1-3 *until* you've sent 16 bits.
4. Once all 16 bits are sent then engage the **Latch Clock** on R**C**6:
    a. Make R**C**6 a 0 ("clear"), *pause*, then make RC6 a 1 ("set"), *pause*

Note that the **Reset pin** (**RA4**) needs to be latched to a logic value of 1 while you are assigning values into the shift register. Also note that if nothing is displaying at all then ensure that **RC5** is a logic 1, too. Otherwise the transistor switches controlling all current to the LEDs (via the "PWM" line) might be off.

Try to do this initially by sending the following 16 bit pattern: 0b1000000010000000 where the 1 is the last bit sent and 0 is the first.

What does this display?

Stimulate

Answer the following questions in your lab report.

1. How many LED segments are on the "7seg Click" board?

Problem

In your main function write a for loop that never ends. Inside of this loop write a sequence of instructions to the PIC16 chip that examines the state of a push-button on the board and then shows a pair of digits on the 7-segment display.

- *Section GE4 Group 1*: Display 15 and then 11 on the 7-segment display with a 500 millisecond delay between the two.
- *Section GE4 Group* 2: Display 32 and then 24 on the 7-segment display with a 1000 millisecond delay between the two.
- *Section GE4 Group 3*: Display 45 and then 33 on the 7-segment display with a 2000 millisecond delay between the two. Repeat infinitely.
- *Section MIQ4 Group 1*: Display 16 and then 99 on the 7-segment display with a 3000 millisecond delay between the two.
- *Section MIQ4 Group 2*: Display 96 and then 42 on the 7-segment display with a 4000 millisecond delay between the two.

Part 4: Wrap-up

Reflection on Learning

Consider how you could make the sending of data more efficient. Would creating dedicated functions for different steps be good? Could you use an array of data? Would pointers help? Discuss with your partner and others in the lab.

Communication – Reporting

Review the "Grading Rubric" at the beginning of this document. Make sure that you have demonstrated everything written there. Then, write all of the answers to the "Stimulate" and "Explore" questions. Submit the answers to your lab instructor.

Appendix

```
// PIC16F1619 Configuration Bit Settings
// 'C' source line config statements

// CONFIG1
#pragma config FOSC = INTOSC    // Oscillator Selection Bits (INTOSC oscillator: I/O function on CLKIN pin)
#pragma config PWRTE = OFF      // Power-up Timer Enable (PWRT disabled)
#pragma config MCLRE = ON       // MCLR Pin Function Select (MCLR/VPP pin function is MCLR)
#pragma config CP = OFF         // Flash Program Memory Code Protection (Program memory code protection is disabled)
#pragma config BOREN = ON       // Brown-out Reset Enable (Brown-out Reset enabled)
#pragma config CLKOUTEN = OFF   // Clock Out Enable (CLKOUT function is disabled. I/O or oscillator function on the
CLKOUT pin)
#pragma config IESO = ON        // Internal/External Switch Over (Internal External Switch Over mode is enabled)
#pragma config FCMEN = ON       // Fail-Safe Clock Monitor Enable (Fail-Safe Clock Monitor is enabled)

// CONFIG2
#pragma config WRT = OFF        // Flash Memory Self-Write Protection (Write protection off)
#pragma config PPS1WAY = ON     // Peripheral Pin Select one-way control (The PPSLOCK bit cannot be cleared once it is
set by software)
#pragma config ZCD = OFF        // Zero Cross Detect Disable Bit (ZCD disable.  ZCD can be enabled by setting the ZCDSEN
bit of ZCDCON)
#pragma config PLLEN = ON       // PLL Enable Bit (4x PLL is always enabled)
#pragma config STVREN = ON      // Stack Overflow/Underflow Reset Enable (Stack Overflow or Underflow will cause a Reset)
#pragma config BORV = LO        // Brown-out Reset Voltage Selection (Brown-out Reset Voltage (Vbor), low trip point
selected.)
#pragma config LPBOR = OFF      // Low-Power Brown Out Reset (Low-Power BOR is disabled)
#pragma config LVP = ON         // Low-Voltage Programming Enable (Low-voltage programming enabled)

// CONFIG3
#pragma config WDTCPS = WDTCPS1F// WDT Period Select (Software Control (WDTPS))
#pragma config WDTE = OFF       // Watchdog Timer Enable (WDT disabled)
#pragma config WDTCWS = WDTCWSSW// WDT Window Select (Software WDT window size control (WDTWS bits))
#pragma config WDTCCS = SWC     // WDT Input Clock Selector (Software control, controlled by WDTCS bits)

#define _XTAL_FREQ 500000     // Define PIC16F1619 clock freq - 500 khz is default with internal clock.

#include <xc.h>

/* function prototypes */
void init_ports(void);


int main(void) {

    char latch_value;   /* 8 bit word */
    latch_value = 0b00100000;

    init_ports(); // initialize the ports

    while(1)
    {
        asm("NOP");
        latch_value = latch_value + 1;
        LATA = latch_value;               // assign value to the LATA register
        __delay_ms(100);                  // 100 ms delay
    }

    return 0;
}
```

Above, we have an example for integrating the configuration bits statements with regular code. The init_ports() function is not shown. More details here: http://drsmith.blog.yorku.ca/2018/11/the-led-on-my-pic16-board-wont-light-up/

Below, we have the **Configuration setup** for the PIC16F1619 on the Curiosity Board. **You should put this at the top of your .c file.**

```
#include <xc.h>

#define _XTAL_FREQ 500000    // Define PIC16F1619 clock freq - 500 kHz

#pragma config FOSC = INTOSC    // Oscillator Selection Bits
#pragma config PWRTE = OFF      // Power-up Timer Enable (PWRT disabled)
#pragma config MCLRE = ON       // MCLR Pin Function Select
#pragma config CP = OFF        // Flash Program Memory Code Protection
#pragma config BOREN = ON       // Brown-out Reset Enable
#pragma config CLKOUTEN = OFF   // Clock Out Enable
#pragma config IESO = ON        // Internal/External Switch Over
#pragma config FCMEN = ON       // Fail-Safe Clock Monitor Enable
// CONFIG2
#pragma config WRT = OFF        // Flash Memory Self-Write Protection
#pragma config PPS1WAY = ON     // Peripheral Pin Select one-way control
#pragma config ZCD = OFF        // Zero Cross Detect Disable Bit
#pragma config PLLEN = ON       // PLL Enable Bit (4x PLL is always enabled)
#pragma config STVREN = ON      // Stack Overflow/Underflow Reset Enable
#pragma config BORV = LO        // Brown-out Reset Voltage Selection
#pragma config LPBOR = OFF      // Low-Power Brown Out Reset
#pragma config LVP = ON         // Low-Voltage Programming Enable
// CONFIG3
#pragma config WDTCPS = WDTCPS1F// WDT Period Select (Software Control
#pragma config WDTE = OFF       // Watchdog Timer Enable (WDT disabled)
#pragma config WDTCWS = WDTCWSSW// WDT Window Select
#pragma config WDTCCS = SWC     // WDT Input Clock Selector
```

You can also regenerate all of the "#pragma" lines in the code above using the MPLAB X "set configuration bits" wizard. See this blog post for details: http://drsmith.blog.yorku.ca/2018/11/the-led-on-my-pic16-board-wont-light-up/