

Complete by:
Thursday January 29th for an A+

References:
Section 5.6 T2.c template
Section 6.7 ASCII4 function from Measure.c

Overview for P4A.c

For this part, you are to strip the code of T2.c of almost everything except what is needed to update the LCD every sixth loop time (i.e., every $16 \times 6 = 96$ milliseconds) with the decimal expression of the potentiometer output, ranging from 0000 to 1023 (or thereabouts), centered on the LCD. Your code should be stripped down to consist of only the Initial, BlinkAlive, ReadPot (from your P3.c file), ASCII4 (from Measure.c), and Display functions. Modify ReadPot to return the right-justified ten-bit value of the potentiometer output in the *int* register, **ADRES**. Copy this value into the *int* variable, **BIGNUM**, for use by the ASCII4 function. Use the output of the ASCII4 function to update **LCDSTRING**.

Initial function

In your Initial function from T2.c, removed the crossed out lines shown below. In the space marked < Insert Part P4A or Part P4B code here > add the six lines shown here:

```
PORTBbits.RB4 = 1;           // Measure execution time of LoadLCDSTRING + Display
LoadLCDSTRING("POT.VALUE"); // Put constant string into LCDSTRING
Display();                   // Display POT.VALUE
PORTBbits.RB4 = 0;           // Finish measurement
Delay(50000);                 // Pause for
Delay(50000);                 //           one second

void Initial()
{
  OSCCON = 0b01100010;        // Use Fosc = 4 MHz (Fcpu = 1 MHz)
  SSPSTAT = 0b00000000;      // Set up SPI for output to LCD
  SSPCON1 = 0b00110000;
  ADCON1 = 0b00001011;       // RA0,RA1,RA2,RA3 pins analog; others
                              // digital
  TRISA = 0b00001111;        // Set I/O for PORTA
  TRISB = 0b01000100;        // Set I/O for PORTB
  TRISC = 0b10000000;        // Set I/O for PORTC
  TRISD = 0b10000000;        // Set I/O for PORTD
  TRISE = 0b00000010;        // Set I/O for PORTE
  PORTA = 0;                  // Set initial state for all outputs low
  PORTB = 0;
  PORTC = 0;
  PORTD = 0b00100000;        // except R05 that drives LCD interrupt
  PORTE = 0;
  SSPBUF = ' ';              // Send a blank to initialize state of UART
  Delay(50000);              // Pause for half a second
  RCONbits.SBOREN = 0;       // Now disable brown-out reset
  PBFLAG = 0;                // Clear flag until pushbutton is first pressed
  LCDFLAG = 0;              // Flag to signal LCD update is initially off
  TIMECNT = 0;              // Reset TIMECNT
  TENS = '5';                // Initialize to 50 so first display = 00
  ONES = '0';                // Initialize count of pushbutton presses
  PBTENS = '0';              // Initialize count of pushbutton presses
  PBONES = '1';              // Initialize count of pushbutton presses
  OLDPB = 0;                // Initialize to unpressed pushbutton state
  ALIVECNT = 247;            // Blink immediately

  < Insert Part P4A or Part P4B code here >

  WDTCONbits.SWDTEN = 1;     // Enable watchdog timer
  Display();                  // Display initial "PRESS PB" message
  LoadLCDSTRING("00 01. "); // Reinitialize LCDSTRING
}
```

Overview for P4B.c

For this part, you are to introduce a new DisplayC function that is called with a line like

```
DisplayC(PotStr);
```

to display a constant string created in the program area (rather than formed in the RAM area) with

```
const char PotStr[] = { "POT.VALUE" };
```

I believe that this will store an even number of bytes (ten, including an "end of string" terminator character of 0x00) in program memory, of which only the first nine will be used. Please note that the compiler used to malfunction with constant strings having an odd number of bytes. Check this by compiling and running both the above definition of PotStr and then putting an extra blank character at the end of the string ("POT.VALUE ") and compiling and running this modification.

Declare the DisplayC function prototype with

```
void DisplayC(const char *);
```

and the function definition itself with

```
void DisplayC(const char * TempPtr)
{
    static char i;

    <etc.>

    SSPBUF = *TempPtr;    // Use this to send an ASCII-coded character from the string
    TempPtr++;           // Use this to increment the pointer to the next character

    <etc.>
}
```

In the space marked < Insert Part P4A or PartP4B code here > in the Initial function, insert the six lines shown here in place of those added in Part P4A:

```
PORTBbits.RB4 = 1;           // Measure execution time of DisplayC function
DisplayC(PotStr);           // Display POT.VALUE
PORTBbits.RB4 = 0;           // Finish measurement
Delay(50000);                // Pause for
Delay(50000);                //           one second
```

Reformatting P4A.c and P4B.c

David Bauer has developed a utility that will reformat your source file, so that C control constructs are consistently indented three spaces and so that the comment fields all line up. The usage is

```
C:\Work> QwikIndent P4A.c    and    C:\Work> QwikIndent P4B.c
```

Use these on the source files before you turn them in to the TA.