Lab 7: Serial Port

Introduction

In this lab, you will modify the lab from last week so that you can send temperature data to a serial port for display on a terminal. The terminal should display “T=XX.XC” where XX.X is the current temperature in Celsius. You will use the terminal keyboard input to control the state transitions instead of the ENTER button on the board.

Terminal Display

Use the hyperterminal program on Windows to connect to the board. You can start hyperterminal, by going to Start -> Run and typing hypertrm. You can create a new connection with a 19200 baud rate and 8-bit data, no parity, and 1 stop bit.

To enable display of data on the terminal, you will need to provide three new routines: SerialChar(char c), SerialString(const char *s), and SerialDecimal(unsigned short d). These routines will be very similar to the LCD display routines except that they will write the characters to the serial port as discussed below. In your DisplayTemp routine(), you can insert these calls in the appropriate place to display the temperature on the terminal. You will also need to make sure that you reset the cursor to the beginning of the line, before you write out the temperature. This can be accomplished by sending a carriage return character to the terminal by calling SerialChar(‘\r’). \r is the way you represent the carriage return character.

Terminal Input

In the previous lab, you transitioned states by checking for the ENTER button press. In this lab, you will replace that code instead to check if a terminal key was pressed. This can be done by checking if a character arrived on the serial port as discussed below.

Serial Port

The serial port is controlled by a device called the USART (Universal Synchronous Asynchronous Receiver Transmitter). This sometimes known as a UART for non-synchronous serial communication or SCI (Serial Communication Interface). In most computer systems the USART is a separate module on the motherboard, but most microcontrollers will include at least one and sometimes several USARTs on the microcontroller itself. The PIC16F874 that we are using has a single built-in USART with the transmit and receive ports connected to pins RC6 and RC7 respectively. As shown below, these pins are connected the MAX232 RS-232 driver which drives the actual cable.
The USART is controlled by two main registers: TXSTA and RCSTA. The TXSTA register controls the transmission and the RCSTA register controls the reception. The baud rate is controlled by the BRGH bit in the TXSTA register and the SPBRG register. The baud rate is determined by the following formula:

$$\text{Baud Rate} = \frac{F_{\text{osc}}}{(64 - 48 \cdot \text{BRGH})(\text{SPBRG} + 1)}$$

To set the baud rate to 19200 with a 7.37 MHz clock, set BRGH to 1 and SPBRG to 23.

To enable transmission and receiving, we have to set the TXEN bit in the TXSTA register and the SPEN and CREN bits in the RCSTA register. We also need to set the appropriate bits in the TRISC register to make sure that the input/output pins are set correctly.

Therefore, the initialization code for the serial port looks as follows:

```c
do void InitSerial()
{
    TXSTA = 0b00100100; // Enable TXEN and BRGH
    SPBRG = 23; // 19200 Baud for 7.37MHz
    TRISC7 = 1; // Set C7 (RX) as input
    TRISC6 = 0; // Set C6 (TX) as output
    RP0 = 0; // Force bank 0
    RCSTA = 0b10010000; // Enable SPEN and CREN
}
In order to send a character over the serial port, you should check if the previous transmission is complete by checking the TRMT bit. When the TRMT bit is ‘1’, then you can send the character by writing the character to the TXREG register. Setting the TXREG register clears the TRMT bit.

In order to receive a character, you check if the RCIF bit is set. If it is set, you can then read the character from the RCREG register. Reading the RCREG register will clear the RCIF bit.

**Discussion and Conclusions**

This is a very important lab because it introduces the use of a serial port, the most common way to get data from a microcontroller.

Keep all other functions operational. Thus BarChart, BarIntensity, DisplayV, DisplayTemp should continue to operate. Depending on your code, you may run into difficulties because the code may no longer fit in memory. If that is happening, you can disable the BarChart and BarIntensity functions. If your code exhibits difficulties, set up a watch window and use the ICD to debug.

Demonstrate your project and hand in your well-commented C code by the end of class.