
# University of Balamand

## Faculty of Engineering

## Microprocessors Final Project Report

**Submitted to:** Dr. Nicholas Haddad

**Submitted by:** Hanna Daoud and Michel Sfeir

**CPEN213: Microprocessors**

December 17, 2018

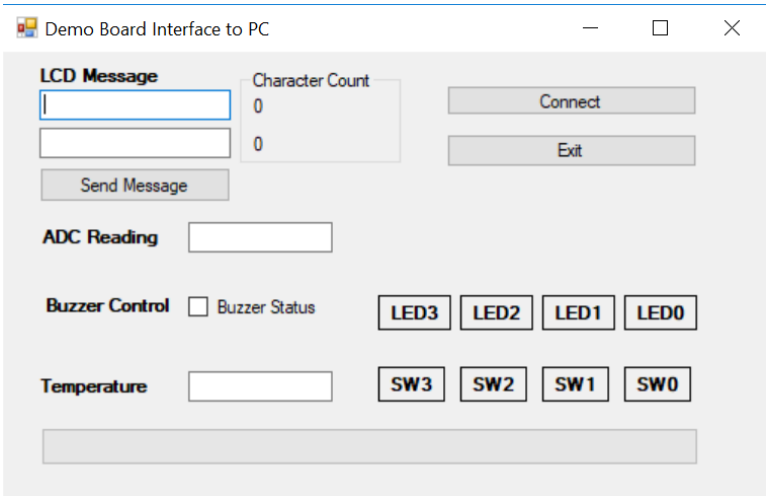
## Introduction:

The purpose of this project is to use C18 and C Sharp to communicate between the microcontroller and the personal computer. Through the use of a Graphical User Interphase (GUI) designed using C#, the user is given the ability to manipulate and monitor several of the microcontroller's components, such as:

- Writing a message and displaying it on the LCD.
- Toggling of LEDs connected to Port D.
- Turning a Buzzer on and off.

In addition, the GUI also displays information about components connected to the board, such as:

- Status of several switches connected to the PIC.
- 10-bit A/D result in decimal format.
- 10-bit A/D result in graphic format.
- Temperature in Celsius read via the LM35 temperature sensor.




*The GUI used to control and monitor the PIC and breadboard.*

## Objective:

Implementation and documentation of the above features, with respect to the C# / C18 solution we were presented with. In this report we will go through them one by one.

## Writing a Message on the LED:

To write a message on the LCD we simply need to receive the data serially from the PC and then use the built in functions of hyperterm.h to send it to the screen. The following code does the trick:

```
case GetLCDdata:
        //Receive LCD data and print it
        RxPacket(rxPacket, 32); // 32 because 16 characters can be printed on two lines
```

```
        DispVarStr(&rxPacket[0], Ln1Ch0, 16);
        DispVarStr(&rxPacket[16], Ln2Ch0, 16);

        break;
```

## Toggling of the LEDs:

Since only 4 LEDs need to be turned on, the information can all be stored and transmitted as one char variable. Hence the code:

```
case GetLEDs:
        RxPacket(rxPacket, 1);
        PORTC = rxPacket[0];

        break;
```

## Turning on the Buzzer:

In order for the buzzer to "sing", a square wave is required to be outputted on pin RB3. We want the square wave to have a frequency of 500 Hz and a 50% duty cycle. The square wave was created via periodic interrupts toggling the value of the RB3 pin. As a result, the following calculations were made to determine the number of cycles and prescaler to be used:

$$F = 500Hz$$

$$T = \frac{1}{500} = 0.02\ s = 2000\ \mu s$$

$$\frac{T}{2} = \frac{2000}{2} = 1000\ \mu s$$

As a result, we used a prescaler of 1:8 for Timer 0 in 8-bit mode.

$$\frac{2000\ \mu s}{8\ \mu s} = 125\ cycles$$

Knowing that the square wave on RB3 was created via periodic interrupts, the following additions were made to the program's setup:

```
void Setup(void) {

    InitUART();

    InitLCD();


  ANSELC = 0x00; // PortC is a digital I/O
    TRISCbits.RC0 = 0x00;
    TRISCbits.RC1 = 0x00;
    TRISCbits.RC2 = 0x00;
    TRISCbits.RC3 = 0x00;
```

```
    TRISCbits.RC4 = 0x00;
    TRISCbits.RC5 = 0x00;
    TRISCbits.RC6 = 0x00;
    TRISCbits.TRISC7 = 0x01;
    ANSELB = 0x00;
    TRISA = 0x01;

    VREFCON0 = 0b10010000;

    ANSELBbits.ANSB3 = 0; // RB3 is a digital i/o

    TRISCBits.TRISB3 = 0; // RB3 is a digital output pin


    T0CON = 0b11010010; // Run internal clock with prescaler 1:8


    INTCONbits.TMR0IE = 0; // Initially the timer interrupt is not
                              enabled

    INTCONbits.GIE = 1; // Global Interrupt Enable on

}
```

In addition, the following code was added to the "BuzzerOnOff" case statement:

```
case BuzzerOnOff:

 INTCONbits.TMR0IE = !INTCONbits.TMR0IE; // Toggle timer interrupt
                                            every time the checkbox is
                                            pressed

break;
```

Also, the following code was used to perform the periodic interrupt that causes the production of the square wave on RC2:

```
#pragma code ISR = 0x0008

#pragma interrupt ISR


void ISR(void){


    TMR0L = 256 - 125; // f = 1/8 MHz, T = 8 us; T * 125 = 1000 us
                       //f = 1000 Hz/2 = 500 Hz


    INTCONbits.T0IF = 0; // Acknowledge interrupt
```

```
    PORTBbits.RB3 = !PORTBbits.RB3; // Toggle RB3 to create a square
                                       wave with 50% duty cycle

}
```

When the TIMER0 interrupt is enabled, the program goes to the ISR each time TMR0L reaches its maximum. There, the TMR0L starting value is reset to count 125 cycles again (1000 us = T/2), the interrupt flag is reset, and RB3 is toggled, causing the production of a square wave with a period of 2000 us and 50% duty cycle.

In order for the buzzer to be toggled on and off through the GUI, TMR0IE (Timer0 Interrupt Enable) is initially disabled in the setup, and thus the buzzer is initially off due to the absence of the square wave output on pin RB3. When the Buzzer ON/OFF Checkbox UI is pressed, this toggles the TMR0IE and, as a result, if the buzzer is off, it will turn on and vice versa.

## Displaying Data on the GUI:

In general, for all the data that we want to display the same principle applies. We simply use TxPacket instead of RxPacket to transmit data from the microcontroller to the PC. There, the C# coding handles the changing of the displays. All that the microcontroller needs to do is get the accurate measurements and transmit them properly. The full code for this is attached at the end of this report.

## Full Code:

```
#include    <p18cxxx.h>
#include    "LCD4lib.h"
#include    "Hyperterm.h"

void Setup(void);
void ReadPTC(void);
void ReadAnalogPot(void);

#define sw3    PORTBbits.RB7    // switch 3 pin
#define sw2    PORTBbits.RB6    // switch 2 pin
#define sw1    PORTBbits.RB5    // switch 1 pin
#define sw0    PORTBbits.RB4    // switch 0 pin
#define buzzer LATBbits.LATB3
char txPacket[5];
char rxPacket[1 + 32];
int i;
```

```
enum cmds {
    SendData, GetLEDs, GetLCDdata, BuzzerOnOff,
DipSwitchesStatus
};
char T, V;

void main(void) {
    Setup();

    while (1) {
        while (!PIR1bits.RCIF);

        switch (RCREG) {

            case SendData:
                //Send data to the PC
                ReadAnalogPot();
                txPacket[0] = ADRESH;
                txPacket[1] = ADRESL;
                ReadPTC();
                txPacket[2] = ADRESH;
                txPacket[3] = ADRESL;
                txPacket[4] = PORTB >> 4;
                TxPacket(txPacket,5);

                break;

            case GetLEDs:
                RxPacket(rxPacket, 1);
                PORTC = rxPacket[0];

                break;

                break;
            case BuzzerOnOff:
                buzzer = ~buzzer;
                INTCONbits.TMR0IE = !INTCONbits.TMR0IE;
// Toggle timer interrupt every time the checkbox is
pressed

                break;
```

```
            case GetLCDdata:
                //Receive LCD data and print it
                RxPacket(rxPacket, 32);
                DispVarStr(&rxPacket[0], Ln1Ch0, 16);
                DispVarStr(&rxPacket[16], Ln2Ch0, 16);

                break;
        }
    }
}

void Setup(void) {
 InitUART();
    InitLCD();
    ANSELC = 0x00; // PortC is a digital I/O
    TRISCbits.RC0 = 0x00;
    TRISCbits.RC1 = 0x00;
    TRISCbits.RC2 = 0x00;
    TRISCbits.RC3 = 0x00;
    TRISCbits.RC4 = 0x00;
    TRISCbits.RC5 = 0x00;
    TRISCbits.RC6 = 0x00;
    TRISCbits.TRISC7 = 0x01;
    ANSELB = 0x00;
   // ANSELA = 0x01;
    TRISA = 0x01;

    //ANSELBbits.ANSB3 = 0;
    TRISBbits.TRISB3 = 0;
        ANSELAbits.ANSA0 = 0x01;
        ANSELAbits.ANSA1 = 0x01;
    // INTCON2bits.RBPU = 0;

    TRISBbits.TRISB4 = 0x01;
    TRISBbits.TRISB5 = 0x01;
    TRISBbits.TRISB6 = 0x01;
    TRISBbits.TRISB7 = 0x01;


    T0CON = 0b11010010; // Run internal clock with
prescaler 1:8
```

```
    INTCONbits.TMR0IE = 0; // Initially the timer
interrupt is not enabled
    INTCONbits.GIE = 1; // Global Interrupt Enable on

    VREFCON0 = 0b10010000;

}

void ReadPTC(void) {
       ADCON0bits.ADON = 1; // turn on A/D, AN0:
default channel
    ADCON0 = 0b00011001;
    Delay10KTCYx(5);
    ADCON1 = 0b00001000;
    ADCON2 = 0b10100101;

   ADCON0bits.GO = 1;
   while (ADCON0bits.GO);
}

void ReadAnalogPot(void) {
     ADCON0bits.ADON = 1; // turn on A/D, AN0: default
channel
    ADCON0 = 0b00001001;
 //   Delay10KTCYx(5);
   ADCON1 = 0b00001000;
    ADCON2 = 0b10100101;

    ADCON0bits.GO = 1;
    while (ADCON0bits.GO);
}

#pragma code ISR = 0x0008
#pragma interrupt ISR

void ISR(void) {
    TMR0L = 256 - 125; // f = 1/8 MHz, T = 8 us; T *
125 = 1000 us; f = 1000 Hz/2 = 500 Hz
    INTCONbits.T0IF = 0; // Acknowledge interrupt
    PORTBbits.RB3 = !PORTBbits.RB2; // Toggle RC2 to
create a square wave with 50% duty cycle
}
```