**Final Design Lab**

**ECE 3720-005**

**27 April 2018**

**Abstract**

For our final design lab we utilized an RS-232 cable in order to interact with the PIC32 microcontroller. This integration allowed for communication between a Visual Basic application and hard-wired LEDs and motors.

**Introduction**

Visual Basic programming is a combination of visually arranging components on a display and giving specific assignments and actions to them. This allows for creation of both simple and complex applications. In our case, clicking “Turn On Motor” does what one might think, turns on the motor as well as an LED indicating its movement. Following in similar fashion, the button below it does the same as found in **Figure 1.4**.

The device used in order to interface between our visual basic code and our microcontroller was an RS-232 converter. This piece of hardware is standard for serial communication which defines signals used, their electrical properties, and their pin output positions in connectors.

**Experimental Procedures**

This project was split into hardware, and software. The hardware dealt with the physical aspect of the project, while the software was mostly various types of coding.

**Hardware**

Using the wiring diagram in **Figure 1.1** we hooked up the microcontroller. Most recent computers have been phasing out the RS-232 COM port in favor of USB ports so we had to use the USB to RS 232 cable in **Figure 1.3**. We connected this cable into the RS-232 to TTL converter found in **Figure 1.2**.  The RS-232 TTL converter was needed because COM ports communicate using 3-15V for logic 0, and -3V to -15V for logic 1. The magnitudes of these voltages are too high for the microcontroller to handle so the converter converts these to a more acceptable level. Specifically it changes it to Transistor-Transistor logic which uses 3.3V or 5V as a 1, and 0V as a 0.

Once the computer was connected to the converter we needed to connect the TX (transmit) pin of the converter to the RX (Receive) pin of the microcontroller. In this lab we used Pin 5 as the receive pin. With the communication setup we now had to connect the motor and the lights. The lights were simply wired to two output pins. The motor needed to go through the L293DNE chip. This chip converts the microcontroller’s PWM (Pulse width modulation) signals to a voltage which runs the motor. After all of the hardware was hooked up we needed to program the Visual Basic application, and the microcontroller.

**Software**

This project had two major software aspects. The first was a visual basic program which communicated with the microcontroller using a COM port, and UART communication. The application window can be seen in **Figure 1.4**. The application had multiple dropdown lists for the different UART settings. This allowed the user to connect to many different devices without needing to change the application. The drop down boxes for the COM port reads through every available COM port on a PC, and lets the user select which one to communicate with. There are also two buttons on the applications which send two different signals. All of this allows the computer to send a UART waveform to the microcontroller.

Before the microcontroller can receive the UART waveform it must be properly set up. The first step in setting up the connection was to set the BRGH, and BRG registers to set the baud rate. Next we set the parity, and stop bits through the PDSEL, and STSEL registers. Finally we enabled the receiver. Now that the microcontroller was setup to receive UART signal we had to use PPSOutput to set a receiving pin.

Now that the microcontroller could receive the computer’s signal we needed to set up what the microcontroller did with that signal. First we set up the LEDs, and motor to operate properly. We used a switch statement to set up two cases. The first case is for the signal that the computer sends when the turn motor on button is pressed. To turn on the motor the microcontroller sets the OC1RS to 100, and turns on the first light. The second case turns the motor to off by setting OC1RS to 1, and lighting up the second light. When either light is turned on the other one is turned off.

**Results**

Our circuit ran as designed. As intended, when clicking each specified button in the Visual Basic application, our microcontroller properly communicated to our circuit LEDs and motor turning on and off. As we clicked turn on motor button, the motor turned on as well as the light and stayed on until we clicked turn off motor. The microcontroller successfully read both of the signals that the computer sent out. In some cases it might misread the signal, but the majority of the time it communicated properly. The visual basic coding had some small issues with connecting to the COM port, but this was fixed by changing how the parity bits, and stop bits were read from the drop down boxes.

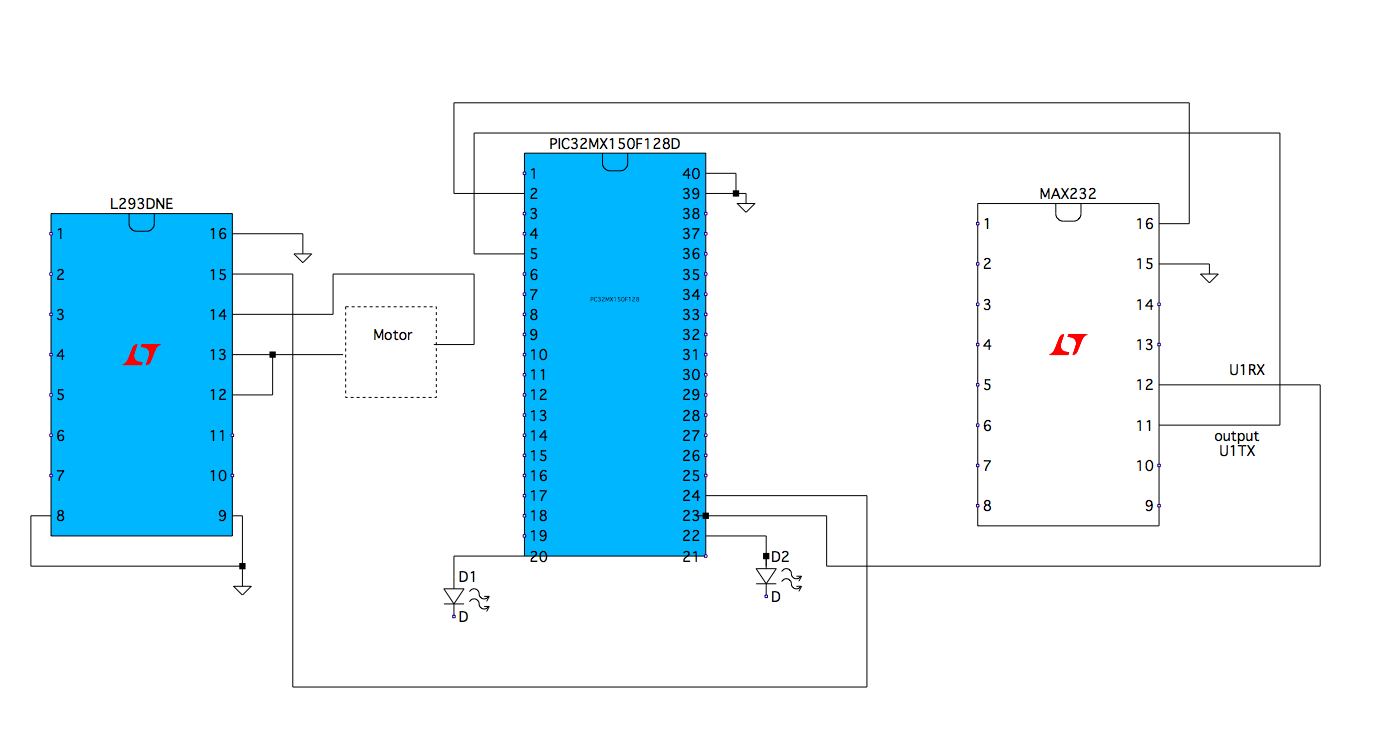
**Discussion**

The biggest hurdle we faced throughout building and testing our project was determining the correct baud rate to set in our separate codes. When initially tested with our microcontroller’s UxBRG set to 25, a baud rate of 9600 bps, and visual basic at 9600, the signals didn’t seem to match up correctly. However, once slowing down visual basic’s baud rate to 10% (960), our desired results began to show. In addition, the signals crossed over three times before producing a result. In order to combat this, we created a while loop to capture input three times in visual basic before outputting to the microcontroller, and finally our LED and motor. This allowed the microcontroller more chances to catch the signal that the computer sent.

**Conclusions**

In conclusion, the project was successfully done. The buttons on the visual basic application causes the motor to turn on and off. The indicator lights also worked successfully. In a manufacturing environment this could be used to increase the safety of the workers by informing them when the machine is on. The use of a visual basic application would allow even untrained personnel to manage a machine, with little oversight needed. Overall the lab was a success, and was a good final project.

**Figures and Tabl**es

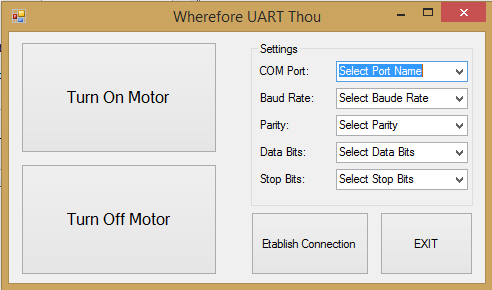


**Figure 1.1: Circuit Wiring Diagram**

**  
Figure 1.2: RS232 to TTL Converter**

****

**Figure 1.3: USB to RS232 Converter Cable**

  
**Figure 1.4: Visual Basic Application Screen**

**Visual Basic Code:**

'Imports the IO Ports for the computer to view.

Imports System.IO.Ports

Public Class Form1

   'Creates Virtual Serial Port to use

   Private ComPort As New SerialPort

   Private Sub ComSetup()

       'Changes the Comport settings to the ones chosen on the page

       With ComPort

           .PortName = Combox.SelectedItem

           .BaudRate = Convert.ToDecimal(BaudBox.SelectedItem)

           .DataBits = Convert.ToDecimal(DataBox.SelectedItem)

           If ParityBox.SelectedItem = "None" Then

               .Parity = Parity.None

           ElseIf ParityBox.SelectedItem = "Even" Then

               .Parity = Parity.Even

           ElseIf ParityBox.SelectedItem = "Odd" Then

               .Parity = Parity.Odd

           End If

           If StopBox.SelectedItem = "1" Then

               .StopBits = StopBits.One

           ElseIf StopBox.SelectedItem = "1.5" Then

               .StopBits = StopBits.OnePointFive

           ElseIf StopBox.SelectedItem = "2" Then

               .StopBits = StopBits.Two

           End If

       End With

   End Sub

   Private Sub ExitButt\_Click(sender As Object, e As EventArgs) Handles ExitButt.Click

       'Stops the user from exiting the program if they are still connected to the Com Port

       If ComPort.IsOpen Then

           MsgBox("Please disconnect From the Com Port")

       Else

           Close()

       End If

   End Sub

   Private Sub LightButt\_Click(sender As Object, e As EventArgs) Handles LightButt.Click

       Dim buffer As Char()

       Dim offset As Integer

       Dim count As Integer

       Dim counter As Integer

       buffer = "1"

       offset = 0

       count = 1

       counter = 0

       'Writes the character ASCI 1 to the Com port. with an offset of zero and counting 1 byte of data. repeats 3 times to...

       'ensure data is recieved

       While counter < 3

           ComPort.Write(buffer, offset, count)

           counter = counter + 1

       End While

       falsebutt.Visible = True

       falsebutt.Enabled = True

   End Sub

   'Fills in Com Port box with possible com ports on computer when application loads

   Private Sub Form1\_Load(sender As Object, e As EventArgs) Handles MyBase.Load

       'Adds all avalible com ports to the Com ports drop down box

       For Each pname As String In My.Computer.Ports.SerialPortNames

           Combox.Items.Add(pname)

           LightButt.Enabled = False

       Next

   End Sub

   Private Sub ConButt\_Click(sender As Object, e As EventArgs) Handles ConButt.Click

       'when the connection button is clicked, checks to see if comport is already opened. if it isn't then it connects

       If (ComPort.IsOpen) Then

           disconnect()

       Else

           Connect()

       End If

   End Sub

   Private Sub disconnect()

       ComPort.Close() 'Closes open connection (should only be open  if button has been pressed once

       ConButt.Text = "Connect" 'Changes button to a connect button

       LightButt.Enabled = False 'Disables the light button

   End Sub

   Private Sub Connect()

       'If every settings box has had a number chosen Then Set up the com port With chosen values

       If Combox.SelectedIndex <> -1 And BaudBox.SelectedIndex <> -1 And ParityBox.SelectedIndex <> -1 And DataBox.SelectedIndex <> -1 And StopBox.SelectedIndex <> -1 Then

           ComSetup() 'Sets up the Com Port Settings

           Try

               ComPort.Open()

           Catch ex As Exception

               MessageBox.Show(ex.Message)

           End Try

       Else

           'Displays a message box if a setting Is Not selected

           MsgBox("Please Select a Value for all settings")

       End If

       'Tries to connect to the comport. displays an error if it is unable to connect

       'Changes the Connect Button to a Disconnect button

       If ComPort.IsOpen Then

           ConButt.Text = "DISCONNECT"

           LightButt.Enabled = True

       End If

   End Sub

   Private Sub falsebutt\_Click(sender As Object, e As EventArgs) Handles falsebutt.Click

       Dim buffer As Char()

       Dim offset As Integer

       Dim count As Integer

       Dim counter As Integer

       buffer = "0"

       offset = 0

       count = 1

       counter = 0

       'Writes the character ASCI 0 to the Com port. with an offset of zero and counting 1 byte of data. repeats 3 times to...

       'ensure data is recieved

       While counter < 3

           ComPort.Write(buffer, offset, count)

           counter = counter + 1

       End While

   End Sub

   Private Sub Form1\_FormClosing(sender As Object, e As FormClosingEventArgs) Handles Me.FormClosing

       If ComPort.IsOpen Then ComPort.Close() 'Makes sure that the Comport is closed when closing the form

   End Sub

End Class

**Microcontroller Code:**

#include <plib.h>

void main(void)

{

   // Motor

   ANSELCbits.ANSC0 = 0;

   TRISCbits.TRISC0 = 0;   // pin 24

   //Turns on Timer and sets PR3 = 5

   T3CONbits.TCS = 0;

   T3CONbits.TCKPS = 0;

   T3CONbits.TGATE = 0;

   PR3 = 100;

   T3CONbits.ON = 1;

   OC1CONbits.ON = 0;

   OC1RS = 0;

   OC1CONbits.OCM = 6;

   OC1CONbits.OCTSEL = 1;

   OC1CONbits.ON = 1;

   PPSOutput(1, RPC0, OC1);

   // LEDs

   TRISBbits.TRISB0 = 0;   // pin 20

   TRISBbits.TRISB2 = 0;   // pin 22

   TRISCbits.TRISC6 = 1;   // pin 5

   PPSOutput(1, RPB3, U1TX);

   PPSInput(3, U1RX, RPC6);

   OSCCONbits.PBDIV = 3;

   U1STA = 0;        // clear status

   U1MODE = 0;    // clear mode

// UART Transmitter Setup

   // 1. Initialize the UxBRG register for the appropriate baud rate.

   U1MODEbits.BRGH = 0;

   //baud rate 9600

   U1BRG = 25;

   // 2. Set the number of Data and Stop bits and Parity selection by writing to the PDSEL and STSEL in UxMODE.

   U1MODEbits.PDSEL = 0;    // 8-bit, no parity

   U1MODEbits.SIDL = 0;    // discontinue in IDLE

   U1MODEbits.STSEL = 0;   //1 Stop Bit

   // 3. If interrupts are desired, set UxTXIE in the correct IECx register and specify the priority and subpriority in UxIP and UxIS control bits in the correct IPCx register. Then select the Transmit Interrupt mode in UTXISEL of UxSTA and clear UxTXIF.

   // 4. Enable transmission by setting UTXEN in UxSTA.

   U1STAbits.URXEN = 1;    // enable reciever

   U1STAbits.UTXEN = 1;    // enable transmitter

   // 5. Enable the UART module by setting the ON bit in UxMODE.

   U1MODEbits.ON = 1;    // enable UART

   while(1){

       switch(U1RXREG){

           case '1': LATBbits.LATB0 = 1;

               LATBbits.LATB2 = 0;

               OC1RS = 100;

               break;

           case '0': LATBbits.LATB2 = 1;

               LATBbits.LATB0 = 0;

               OC1RS = 1;

               break;

             }

    }

}

}