Lab # 6

**Serial communications & EEPROM**

Objectives

- To be familiar with the UART (RS-232) protocol.
- To be familiar with one type of internal storage system in PIC (EEPROM).

Serial Communications

Serial communications is the process of sending data one bit at one time, sequentially, over a communications channel or computer bus. This is in contrast to parallel communications, where all the bits of each symbol are sent together. Serial communications is communications and most computer networks, where the cost of cable and synchronization difficulties makes parallel communications impractical.

Serial computer buses are becoming more common as improved technology enables them to transfer data at higher speeds. Examples of serial communication architectures includes: RS-232, RS-485, Universal Serial Bus (USB), SPI, I2C and others.

Serial interfaces have certain advantages over parallel interfaces. The most significant advantage is simpler wiring. In addition, serial interface cables can be longer than parallel interface cables, because there is much less interaction (crosstalk) among the conductors in the cable.
USART

USART stands for the *Universal Synchronous/Asynchronous Receiver/Transmitter*.

- **Universal** means that it can be used with a wide scope of devices
- **Synchronous** devices that communicate with each other require an external synchronization line (the clock).
- **Asynchronous** The Asynchronous mode (without the common clock) is easier to implement, although it is generally slower than the synchronous. It is also the older way – older versions of PIC did not have the possibility of working in synchronous mode, therefore the devices they had were more appropriately named as UART (without S)
- **Receiver/Transmitter** means that this device can receive and transmit (send) data simultaneously. It is also called the two-way or full duplex communication.

UART Asynchronous Mode

The UART transmits and receives data using standard non-return-to-zero (NRZ) format. As seen in figure below, this mode does not use clock signal, while the data format being transferred is very simple:

![UART Asynchronous Mode Diagram](image)

Briefly, each data is transferred in the following way:
- In idle state, data line has high logic level (1);
- Each data transmission starts with START bit which is always a zero (0);
- Each data is 8- or 9-bit wide (LSB bit is first transferred); and
- Each data transmission ends with STOP bit which always has logic level which is always a one (1).
Each UART contains a **shift register** which is the fundamental method of conversion between serial and parallel forms. After waiting a further bit time, the state of the line is sampled and the resulting level clocked into a shift register. After the required number of bit periods for the character length (8, 9 bits typically) have elapsed, the contents of the shift register is made available.

**RS-232**

The RS-232 standard defines the voltage levels that correspond to logical one and logical zero levels for the data transmission and the control signal lines. Valid signals are plus or minus 3 to 15 volts - the range near zero volts is not a valid RS-232 level.

For example Data is $\text{H’90’} = \text{B’01011010’}$ (LSB bit is first transferred)
Beside voltage level, RS-232 also has a few extra pins:

<table>
<thead>
<tr>
<th>Pin</th>
<th>Signal abbreviation</th>
<th>Signal Name</th>
<th>DTE (PC)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>DCD</td>
<td>Data Carrier Detect</td>
<td>In</td>
</tr>
<tr>
<td>2</td>
<td>RXD</td>
<td>Receive Data</td>
<td>In</td>
</tr>
<tr>
<td>3</td>
<td>TXD</td>
<td>Transmit Data</td>
<td>Out</td>
</tr>
<tr>
<td>4</td>
<td>DTR</td>
<td>Data Terminal Ready</td>
<td>out</td>
</tr>
<tr>
<td>5</td>
<td>GND</td>
<td>Signal Ground</td>
<td>-</td>
</tr>
<tr>
<td>6</td>
<td>DSR</td>
<td>Data Set Ready</td>
<td>in</td>
</tr>
<tr>
<td>7</td>
<td>RTS</td>
<td>Request to Send</td>
<td>out</td>
</tr>
<tr>
<td>8</td>
<td>CTS</td>
<td>Clear to Send</td>
<td>In</td>
</tr>
<tr>
<td>9</td>
<td>RI</td>
<td>Ring Indicator</td>
<td>In</td>
</tr>
</tbody>
</table>

![Serial RS-232 connector diagram]

**Figure 1: Serial Communication Between 2 devices**
RS-232 HW Connection

Concerning with voltage levels, all that is required (at the hardware) is an external level shifter to translate TTL signals from PIC to RS232 levels, and vice-versa. This can be achieved by using a MAX232 chip.
Connection Establishment

In order to establish the communication, PC must have communication software installed. One such communication terminal is part of mikroBasic IDE. It can be accessed by clicking Tools > USART Terminal from the drop-down menu. Terminal allows you to monitor transfer and to set all the necessary transfer settings. First of all, we need to set the transfer rate to match the microcontroller’s rate. Then, select the appropriate communication port and finally press Connect button.
UART Library Functions

- UART1_Init

Initializes hardware USART module with the desired baud rate.

UART1_Init(9600)

- UART1_Data_Ready

Function returns 1 if data is ready or 0 if there is no data.

if UART1_Data_Ready() then \ or Usart_Data_Ready ()=1

'read

end if

- UART1_Read()

Returns the received byte. If byte is not received, returns 0.

if UART1_Data_Ready () then

portb = UART1_Read()

end if

- UART1_Read_Text

Reads characters received via USART until the delimiter sequence is detected (here:"ok"). The read sequence is stored in the parameter output (here:txt).

UART1_Read_Text(txt, “ok”) <<"hello ok"____txt:hello

- UART1_Write

Function transmits a byte (data) via USART.

UART1_Write(portb)

- UART1_Write_Text

Sends text (parameter uart_text) via USART. Text should be zero terminated.

UART1_Write_Text(“hello”)
Explore the UART Library following functions:

- UART1_Init
- UART1_Data_Ready
- UART1_Read_Text
- UART1_Write_Text

Simulate and connect the circuit.

Hints:
- Code examples in the library are very useful.
- MAX232 IC is already connected to the board.
- Note that RX and TX are on pins C6 and C7.
- For simulation you will need to place a virtual terminal.
And connect it like this

<< MAX232 IC is already connected to the board >>

For all parts, Don’t forget to change the internal clock frequency of the PIC to 8MHz

Mikrobasic

```
program Terminal

' Declarations section

main:
  ' Main program
  ADCON1 = ADCON1 or 0x0f
  uart1_init(9600)
  UART1_Write_Text("Welcome in serial communication lab 6")
end.
```
After run, black window appears with the result, if you close it you can get it back by:

**Result:**

![Virtual Terminal Image]

---

**Lab Work 2**

Now we will receive data from PC to PIC and view it on LCD

- Simulate your work on Proteus.
- **Save your code and a snapshot of your Proteus connection. (this must be included in your report!)**
Mikrobasic

program Terminal

' Declarations section
  dim LCD_RS as sbit at RB1_bit
  LCD_EN as sbit at RB2_bit
  LCD_D4 as sbit at RB4_bit
  LCD_D5 as sbit at RB5_bit
  LCD_D6 as sbit at RB6_bit
  LCD_D7 as sbit at RB7_bit

  LCD_RS_Direction as sbit at TRISB1_bit
  LCD_EN_Direction as sbit at TRISB2_bit
  LCD_D4_Direction as sbit at TRISB4_bit
  LCD_D5_Direction as sbit at TRISB5_bit
  LCD_D6_Direction as sbit at TRISB6_bit
  LCD_D7_Direction as sbit at TRISB7_bit
  dim msg as string[100]

main:
  ' Main program
  ADCON1 = ADCON1 or 0x0f
  Lcd_Init()
  uart1_init(9600)
  while true
    if UART_Data_Ready() then
      UART1_Read_Text(msg,"$",10)
      Lcd_Out(1,1,msg)
    end if
  wend
end.
Lab Work 3

1. Write a basic program that sends the state of switch from master PIC to slave PIC (leds).
2. Simulate your work on Proteus.

*Save your code and a snapshot of your Proteus connection. (this must be included in your report!)*
Remember *
TX <> RX , RX <> TX
Mikrobasic

Master

```program Master_SerialCom
  ' Declarations section
  main:
    ' Main program
    adcon1=adcon1 or 0x0f
    trisc=1
    trisc=0xff
    uart1_init(9600)
    while true
      UART1_Write(portb)
    end
  end.
```

Slave

```program Slave_SerialCom
  ' Declarations section
  main:
    ' Main program
    adcon1=adcon1 or 0x0f
    trisc=0
    uart1_init(9600)
    while true
      if UART_Data_Ready then
        portb=UART_Read()
      end if
    end
    end.
```
Connect Proteus Virtual Serial Port with Windows Terminal

To connect Proteus virtual terminal with windows terminal, use compim DB-9 connector.

1. Connect Rx to Rx & Tx to Tx only. Don't make cross connection here. (Connector will be connected to the pic as the same side not opposite side)
2. Now get **Virtual Serial Port** Driver & install it in your system.
3. Then run this tool & create a virtual ports pair.
4. Choose COM3 and COM4 << add pair << pair will appear as virtual ports.

5. In order to be sure that you can work with these ports as a real prts on windows check this:

6. Our work will depend on implementing the **compim** connector (which is connected to PIC) as a COM3 then send and receive data for a pic using it.
**Connection Establishment**

In order to establish the communication, PC must have communication software installed. One such communication terminal is part of MikroBasic IDE. It can be accessed by **clicking Tools > USART Terminal from the drop-down menu.** Terminal allows you to monitor transfer and to set all the necessary transfer settings. First of all, we need to set the transfer rate to match the microcontroller's rate. Then, select the appropriate communication port and finally press Connect button.
Note:

**COM4:** USART Terminal Tool of Mikrobasic  
**COM3:** COMPIM that connected to PIC at Proteus

### Lab Work 4

1. Write a basic program that sends the text from terminal to the PIC side and view it on LCD.
2. Simulate your work on Proteus.

   **Save your code and a snapshot of your Proteus connection. (this must be included in your report!)**

### Mikrobasic

```plaintext
program LCD_SerialCom
'Declarations section
dim
    LCD_RS as sbit at RB1_bit
    LCD_EN as sbit at RB2_bit
    LCD_D7 as sbit at RB7_bit
    LCD_D6 as sbit at RB6_bit
    LCD_D5 as sbit at RB5_bit
    LCD_D4 as sbit at RB4_bit
    LCD_RS_Direction as sbit at TRISB1_bit
    LCD_EN_Direction as sbit at TRISB2_bit
    LCD_D7_Direction as sbit at TRISB7_bit
    LCD_D6_Direction as sbit at TRISB6_bit
    LCD_D5_Direction as sbit at TRISB5_bit
    LCD_D4_Direction as sbit at TRISB4_bit
dim msg as char
main:
' Main program
ADCON1 = ADCON1 or 0x0f
trisp=0
lcd_init()
uart1_init(9600)
while true
    if UART1_Data_Ready() then
        lcd_chr_cp(UART1_Read())
    end if
wend
end.
```
Lab Work 5

1. Write a basic program that sends the text from the PIC side to the terminal.
2. Simulate your work on Proteus.

Save your code and a snapshot of your Proteus connection. *(this must be included in your report!)*

Mikrobasic

```plaintext
program Terminal
  ' Declarations section
main:
    ' Main program
    ADCON1 = ADCON1 or 0x0f
    uart1_init(9600)
    UART1_Write_Text("Welcome from Com3 @")
end.
```
Result

Receive from Pic
EEPROM

**EEPROM** (also called an **E2PROM**) or Electrically Erasable Programmable Read-Only Memory is a non-volatile storage chip used in computers and other devices to store small amounts of volatile (configuration) data. Some microcontrollers use built-in (internal) EEPROM, while other microcontrollers don’t have built in ones. Otherwise, they use external ones connected via parallel or serial communication.

PIC18F4550 has 256 bytes of EEPROM memory locations on addresses from 00h to FFh those can be written to or read from.

**Characteristics of EEPROM**

The most important characteristic of this memory is that combines the characteristics ROM and RAM:

- It does not lose its contents during power supply turned off like ROM
- Data can be retained in EEPROM without power supply for up to 40 years
- Its contents can be modified during the execution of PIC program by PICs CPU like RAM
- Data can be written more than 1.000.000 times.
- Data can be loaded to the EEPROM during the programming operation for the PIC (by the programmer)

**Programming the internal EEPROM with mikrobasic**

Functions used: Eeprom_Read  Eeprom_Write
HomeWork

- Include all your work during the lab.
- Write a basic program that shines the led connected to PIC if the pc (V.T.) entered “hi” word.

- Write a basic program that send the pressed number on keyboard in the terminal

- Send your homework in my Email Write Subject ‘Embedded System’

😊😊😊😊Good luck😊😊😊😊