Microcontrollers

Serial Communication with PIC16F877A

Dr. Jafar Jallad

Palestine Technical University – Kadoorie

Second semester

2019-2020

Objective

- Explain serial communication protocol
- Describe data transfer rate and bps rate
- Interface the PIC16 with an RS232 connector
- Describe the main registers used by serial communication of the PIC16
- Program the PIC16 serial port in Assembly

WHY DO WE NEED FAST INTERFACES?

- Microcontrollers need fast ways of communication to the outside world for:

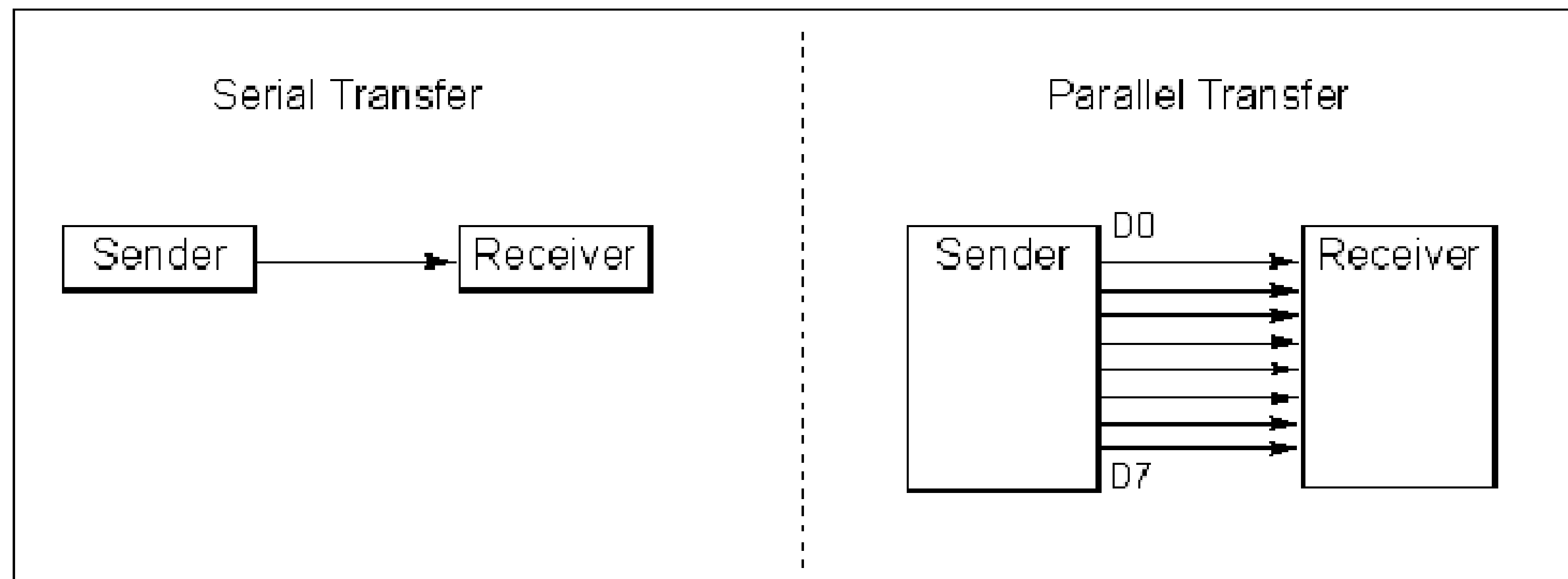
1- Communicating with other microcontrollers, DSPs or even FPGAs. (ex. SRIO, PCIe, I2C)

2- Capturing input from user and displaying outputs.

3- Communicating with other microcontrollers on different boards for applications with network of microcontrollers. (ex. CAN and LIN)

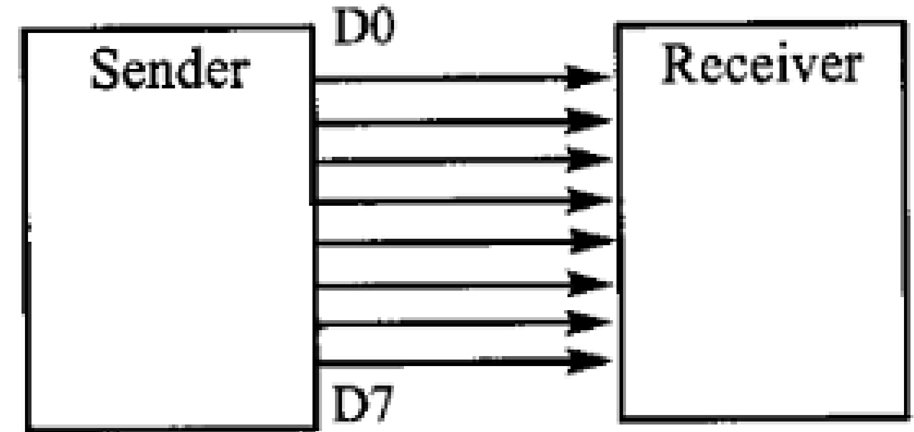
Basics of Serial Communication

- The byte of data must be converted to serial bits using a parallel-in-serial-out shift register



PARALLEL COMMUNICATION

- The process of sending several bits as a whole, on a link with several parallel channels.



- It requires a separate channel for each bit to be transmitted

- A parallel link use simpler hardware as there is no need for a serializer/deserializer.

- Usually used for very short distances.

Basics of Serial Communication (cont'd)

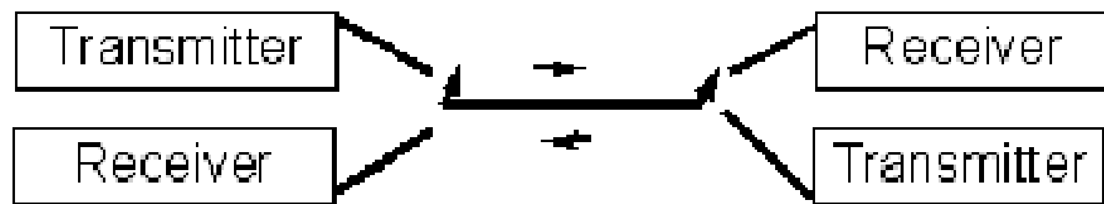
- The receiving end must be a serial-in-parallel-out shift register and pack them into a byte.
- Two methods of serial data communication: Asynchronous and Synchronous

Half-and Full-Duplex Transmission

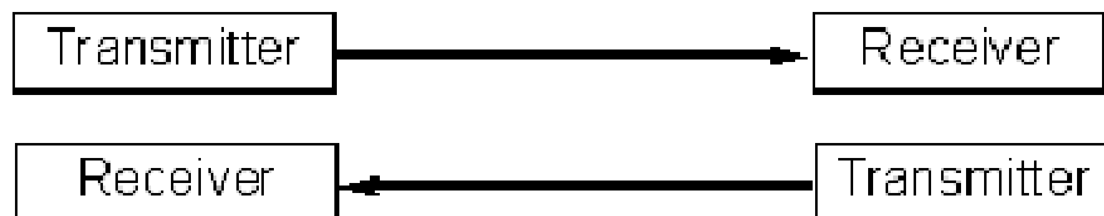
1. **Simplex:**
- Communication is possible in one direction only. Ex.TV
2. **Half duplex:**
- Communication is possible in both directions,
but only one TX and one RX at a time. Ex. Police radio
3. **Full duplex:**
- Communication is possible in both directions,
both sides can transmit and receive in the same time.

Simplex

Half Duplex



Full Duplex



Basics of serial communication

Bit rate:

- Number of bits sent every second (BPS)

Baud rate:

- Number of symbols sent every second, where every symbol can represent more than one bit.

Ex. high-speed modems which use phase shifts to make every data transition period represent more than one bit.

- For the PIC 16f877A's USART, with every clock tick one bit is sent, each symbol represents one bit.
- So, we can consider bit rate and baud rate the same thing.

BASICS OF SERIAL COMMUNICATION

- The sender and receiver must agree on a set of rules **(Protocol)** on :

1. When data transmission begins and ends.
2. The used bit rate and data packaging format.

- If there is no reference for the receiver to know when transmission begins or the used bit rate,

→ it'll sample the wrong values and data will be lost.

SYNCHRONOUS VS ASYNCHRONOUS

1. **Synchronous transmission:**

In synchronous transmission, a separate link is dedicated for the clock from one terminal (Master) to another (Slave).

1. **Asynchronous transmission:**

In asynchronous transmission, no link for the clock.
Synchronization is done → using a fixed baud rate and using start and stop bits.

How synchronization is done?

- **For synchronous transmission:**

- Synchronization is done using a clock line

- A clock line from one terminal (master) to the other terminal (slave) makes the synchronization.

- Another line is used for data transmission between master and slave(s).

- If the master communicates with many peripheral ICs using the same data and clock lines, a (slave select) line is used to determine which slave to communicate with.

HOW SYNCHRONIZATION IS DONE?

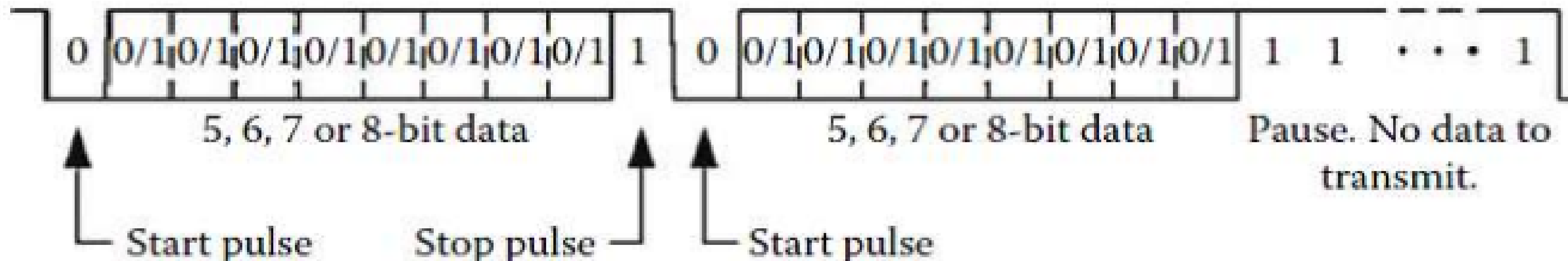
- **<u>For asynchronous transmission:</u>**

- <u>Synchronization is done every word</u>

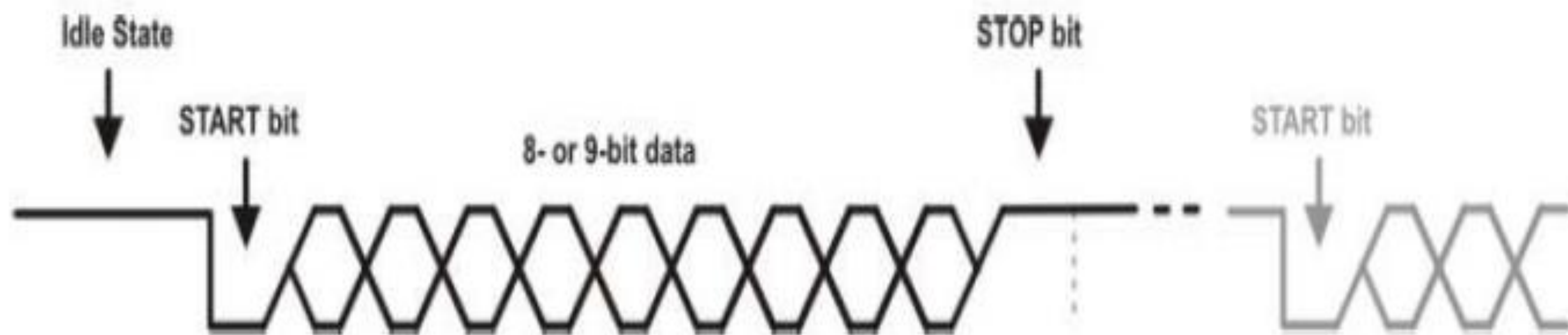
  → A *start bit* with the value 0 indicate the beginning of each word, then eight data bits are sent bit by bit, and finally a *stop bit* with the value 1 to indicate the end of the word.

- Both the transmitter and receiver use the same baud rate.

- When the transmitter pauses because it does not have data to transmit (*idle state*), it keeps a sequence of stop bits (logic high) in its output.



Universal Synchronous Asynchronous Receiver Transmitter (USART)



USART

- The USART module is a full duplex, serial I/O communication peripheral.

- It contains all shift registers, clock generators and data buffers needed for serial communication.

- It can work in synchronous mode, or in asynchronous mode.

- The USART uses two I/O pins to transmit and receive serial data. Both transmission and reception can occur at the same time i.e. ‘full duplex’ operation.

USART

- To send a byte, the application writes the byte to the transmit buffer.
- The UART then sends the data, bit by bit in the requested format, adding Stop, Start, and parity bits as needed.

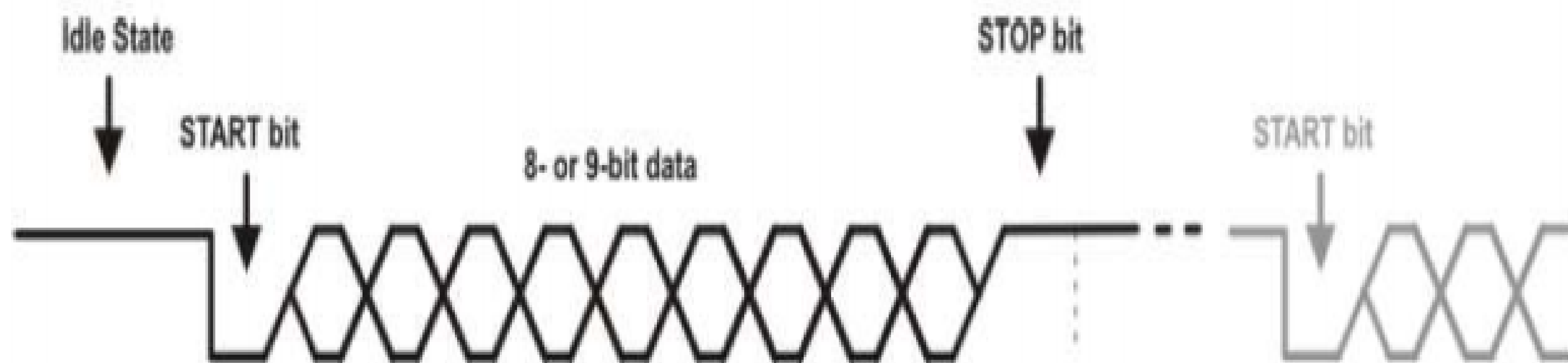
- In a similar way, the UART stores received bytes in a buffer.
- Then the UART can generate an interrupt to notify the application or software can poll the port to find out if data has arrived.

USART

- **Asynchronous Mode:**

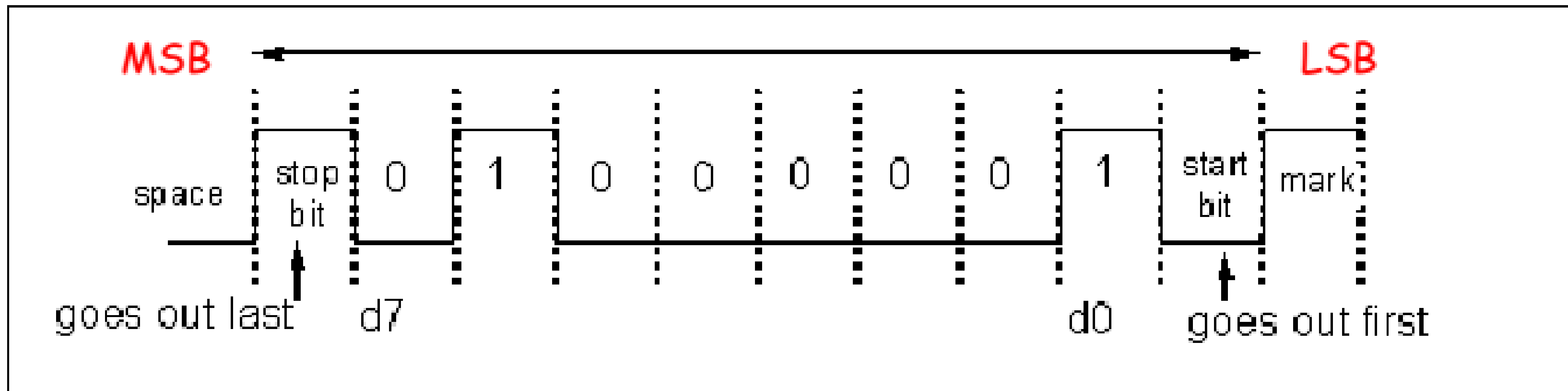
Data transfer happens in the following way:

1. In idle state, data line has logic high (1).
2. Data transfer starts with a start bit, which is always a zero.
3. Data word is transferred (8 or 9 bit), LSB is sent first.
4. Each word ends with a stop bit, which is always high (1).
5. Another byte can be sent directly after, and will start also with a start bit befor data.



Start and Stop Bits

- In the asynchronous method, each character is placed between start and stop bits (framing)



Framing ASCII 'A' (41H)

Data Transfer Rate

- Rate of data transfer: *bps* (bits per second)
- Another widely used terminology for bps is baud rate
- For Asynchronous serial data communication, the baud rate is generally limited to 100,000bps

RS232 Standard

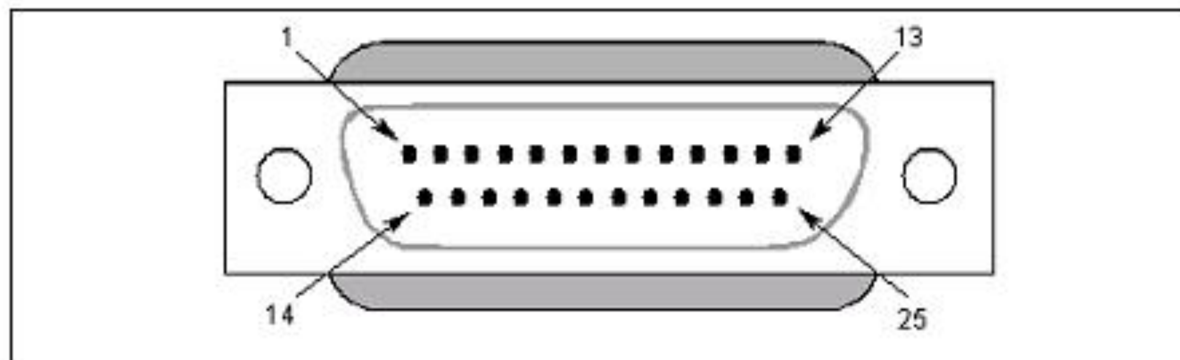
- Standard for serial comm (COM port)
  1: -3V to -25V;
  0: +3V to +25V
  - Reason: for long distance wired line
- Input-output voltage are not TTL compatible
- So, we need MAX232/233 for voltage converter. Commonly known as line drivers

RS232 Pins

Connectors:

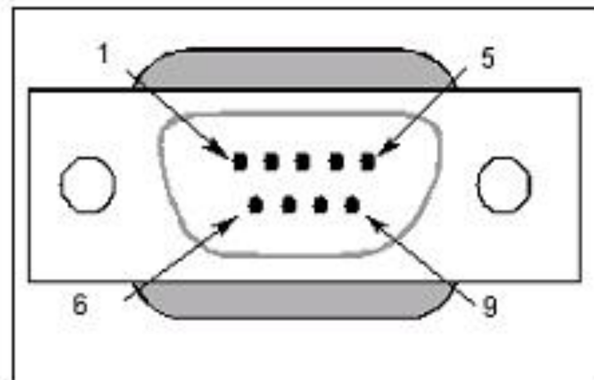
Minimally, 3 wires: RxD, TxD, GND

Could have 9-pin or 25-pin



DB-25

25-Pin Connector



DB-9

9-Pin Connector

Universal Synchronous Asynchronous Receiver Transmitter (USART)

PIC microcontrollers, obviously, can do more than just light up LEDs or reading button states.

Microcontrollers can also communicate with another microcontroller or with other devices like sensors, memory cards, etc.

Often the communication is done serially, where data bits are sent one at a time.

In this presentation, we will look at how to implement serial communication with PICs in assembly language

While you can implement serial communications through “bit-banging”, i.e., setting a pin high or low in specific time intervals (also known as software serial), using the hardware USART module is a much more reliable and easier approach.

Software serial offers the advantage of assigning transmit and receive pins it to any output pin. This is useful when you ran out of pins and need to communicate to multiple devices.

In contrast, hardware USART exclusively uses the pins RC6 (TX) and RC7 (RX).

To configure the PIC’s hardware USART, we need three registers: TXSTA, RCSTA and SPBRG.

The SPBRG is used to calculate the baud rate of the transmissions.

The TXSTA and RCSTA registers are shown Next Slides:

TXSTA: TRANSMIT STATUS AND CONTROL REGISTER (ADDRESS 98h)

R/W-0	R/W-0	R/W-0	R/W-0	U-0	R/W-0	R-1	R/W-0
CSRC	TX9	TXEN	SYNC	—	BRGH	TRMT	TX9D
bit 7							bit 0

bit 7 **CSRC:** Clock Source Select bit

Asynchronous mode:
Don't care.

Synchronous mode:
1 = Master mode (clock generated internally from BRG)
0 = Slave mode (clock from external source)

bit 6 **TX9**: 9-bit Transmit Enable bit

1 = Selects 9-bit transmission
0 = Selects 8-bit transmission

bit 5 **TXEN**: Transmit Enable bit

1 = Transmit enabled
0 = Transmit disabled

Note: SREN/CREN overrides TXEN in Sync mode.

bit 4 **SYNC**: USART Mode Select bit

1 = Synchronous mode
0 = Asynchronous mode

bit 3 **Unimplemented:** Read as '0'

bit 2 **BRGH**: High Baud Rate Select bit

Asynchronous mode:
1 = High speed
0 = Low speed

Synchronous mode:
Unused in this mode.

bit 1 **TRMT**: Transmit Shift Register Status bit

1 = TSR empty
0 = TSR full

bit 0 **TX9D:** 9th bit of Transmit Data, can be Parity bit

RCSTA: RECEIVE STATUS AND CONTROL REGISTER (ADDRESS 18h)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R-0	R-0	R-x
SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D
bit 7							bit 0

bit 7 **SPEN:** Serial Port Enable bit

1 = Serial port enabled (configures RC7/RX/DT and RC6/TX/CK pins as serial port pins)
0 = Serial port disabled

bit 6 **RX9**: 9-bit Receive Enable bit

1 = Selects 9-bit reception
0 = Selects 8-bit reception

bit 5 **SREN**: Single Receive Enable bit

Asynchronous mode:
Don't care.

Synchronous mode – Master:
1 = Enables single receive
0 = Disables single receive
This bit is cleared after reception is complete.

Synchronous mode – Slave:
Don't care.

bit 4 **CREN**: Continuous Receive Enable bit

Asynchronous mode:
1 = Enables continuous receive
0 = Disables continuous receive

Synchronous mode:
1 = Enables continuous receive until enable bit CREN is cleared (CREN overrides SREN)
0 = Disables continuous receive

bit 3 **ADDEN:** Address Detect Enable bit

Asynchronous mode 9-bit (RX9 = 1):
1 = Enables address detection, enables interrupt and load of the receive buffer when RSR<8> is set
0 = Disables address detection, all bytes are received and ninth bit can be used as parity bit

bit 2 **FERR**: Framing Error bit

1 = Framing error (can be updated by reading RCREG register and receive next valid byte)
0 = No framing error

bit 1 **OERR**: Overrun Error bit

1 = Overrun error (can be cleared by clearing bit CREN)
0 = No overrun error

bit 0 **RX9D:** 9th bit of Received Data (can be parity bit but must be calculated by user firmware)

That’s a lot of bits

Four our purpose, we will only look at four bits from TXSTA and 1 bit on RCSTA.

TXSTA: TRANSMIT STATUS AND CONTROL REGISTER (ADDRESS 98h)

R/W-0	R/W-0	R/W-0	R/W-0	U-0	R/W-0	R-1	R/W-0
CSRC	TX9	TXEN	SYNC	—	BRGH	TRMT	TX9D
bit 7							bit 0

TXEN (bit 5) on TXSTA enables or disables transmission,

SYNC (bit 4) sets asynchronous or synchronous mode

BRGH (bit 2) sets high speed or low speed mode.

TRMT (bit 1) is a flag that sets if the data has been sent.

The formula used to compute for the baud rate is different in high speed or low speed mode:

SYNC	BRGH = 0 (Low Speed)	BRGH = 1 (High Speed)
0	(Asynchronous) Baud Rate = Fosc/(64 (X + 1))	Baud Rate = Fosc/(16 (X + 1))
1	(Synchronous) Baud Rate = Fosc/(4 (X + 1))	N/A

Here, X is the contents of the register SPBRG. So, for example, a baud rate of 9600 using a 4 MHz oscillator at high speed and asynchronous mode will have SPBRG = 25 as shown:

$$X = \frac{4,000,000}{9600 * 16} - 1$$

$$X = 25$$

Baud rate generator

- From the last two equations, we can determine the value of the SPBRG register and the BRGH bit according to the required baud rate, and the used oscillator.
- Another form:

→If BRGH=0 (low speed):

$$SPBRG = \frac{Fosc}{64 \times Baudrate} - 1$$

→If BRGH=1 (high speed):

$$SPBRG = \frac{Fosc}{16 \times Baudrate} - 1$$

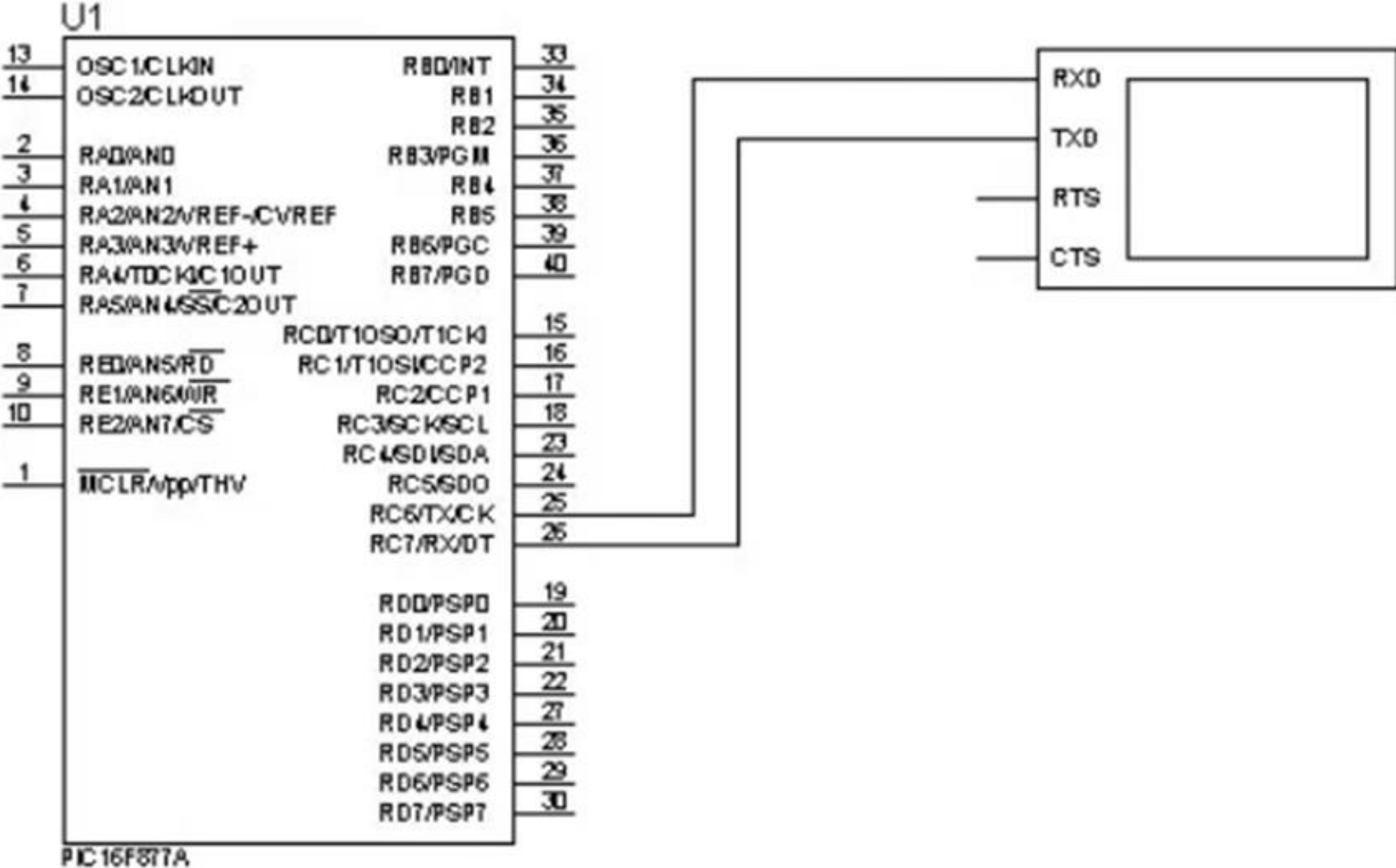
RCSTA: RECEIVE STATUS AND CONTROL REGISTER (ADDRESS 18h)

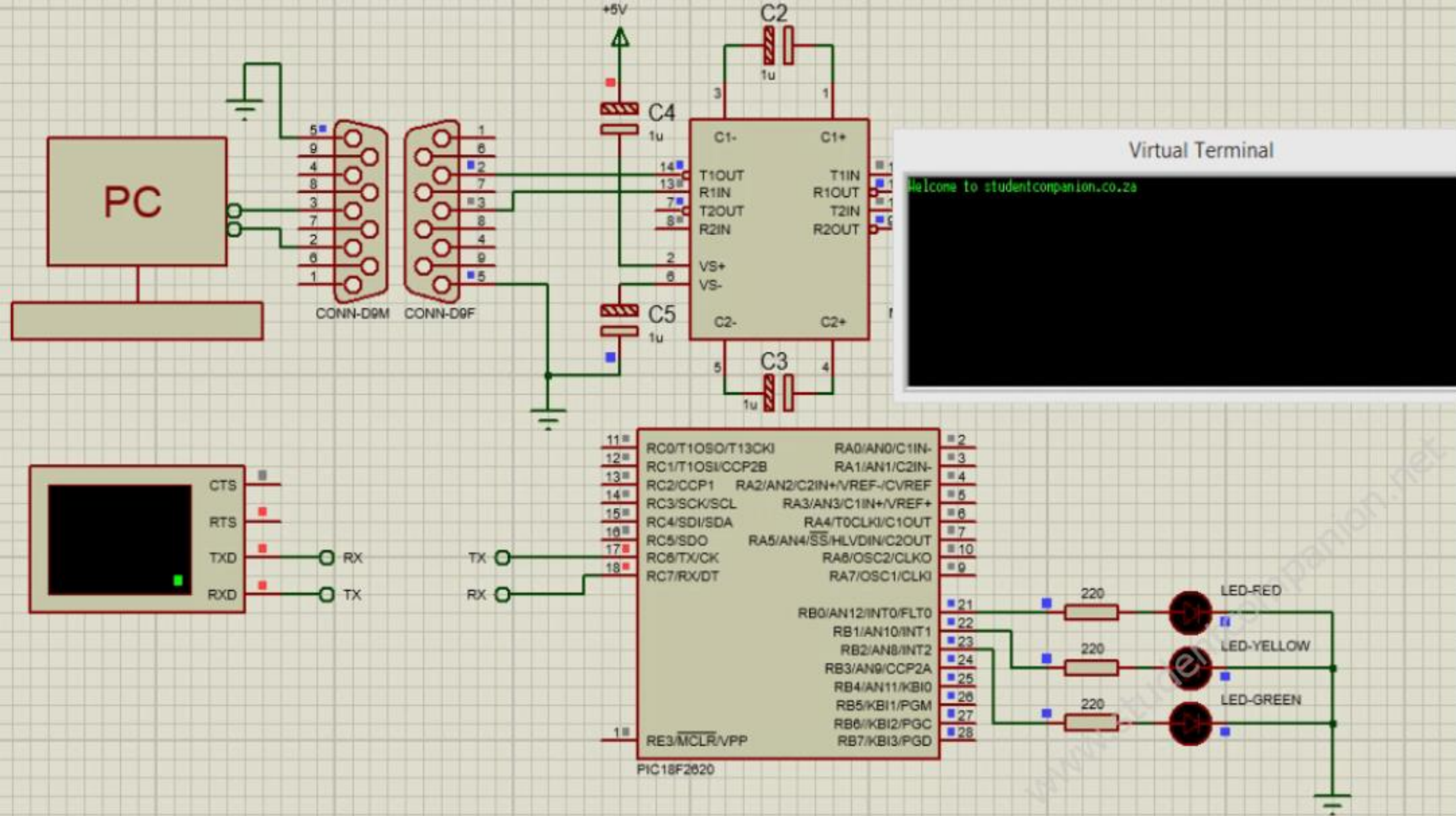
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R-0	R-0	R-x
SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D
bit 7							bit 0

SPEN (bit 7) of the RCSTA register enables RC6 and RC7 as serial port pins. This is the bit you need to set to enable serial communication.

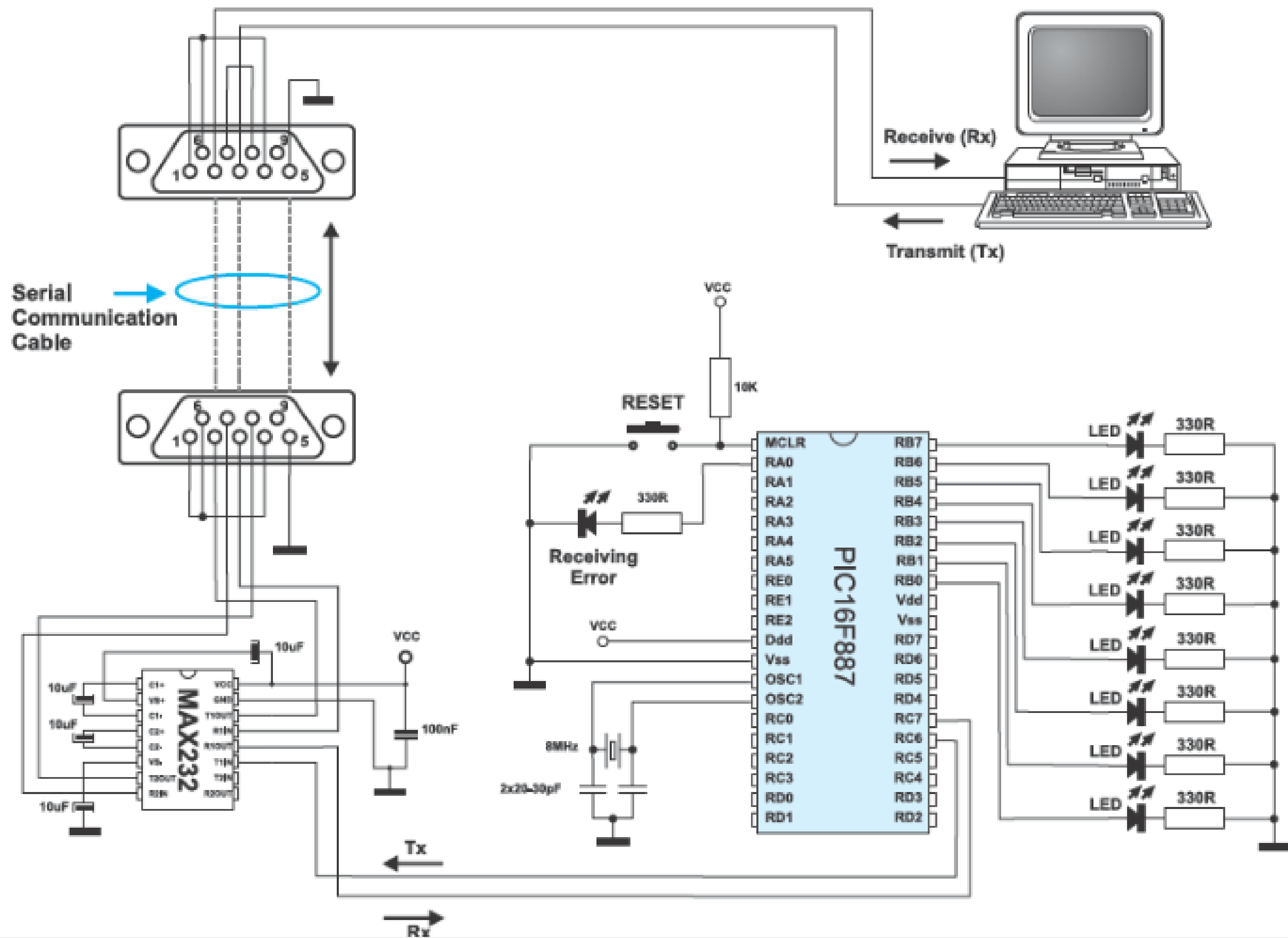
The data to be transmitted must be placed inside the TXREG register while the data received is placed inside the RCREG register.

We can simulate the serial communication using Proteus ISIS.



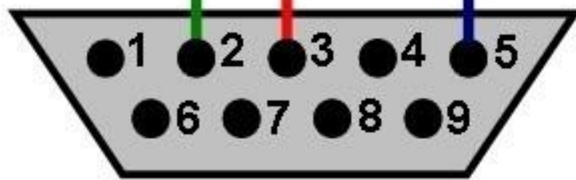


PC
CONN-D9M
CONN-D9F
+5V
C2
1u
C4
1u
C5
1u
C3
1u
C1-
C1+
T1OUT
R1IN
T2OUT
R2IN
T1IN
R1OUT
T2IN
R2OUT
VS+
VS-
C2-
C2+
Virtual Terminal
Welcome to studentcompanion.co.za
CTS
RTS
TXD
RXD
RX
TX
RC0/T1OSO/T13CKI
RC1/T1OSI/CCP2B
RC2/CCP1
RC3/SCK/SCL
RC4/SDI/SDA
RC5/SDO
RC6/TX/CK
RC7/RX/DT
RE3/MCLR/VPP
RA0/AN0/C1IN-
RA1/AN1/C2IN-
RA2/AN2/C2IN+/VREF-/CVREF
RA3/AN3/C1IN+/VREF+
RA4/T0CLKI/C1OUT
RA5/AN4/SS/HLVDIN/C2OUT
RA6/OSC2/CLKO
RA7/OSC1/CLKI
RB0/AN12/INT0/FLT0
RB1/AN10/INT1
RB2/AN8/INT2
RB3/AN9/CCP2A
RB4/AN11/KBI0
RB5/KBI1/PGM
RB6/KBI2/PGC
RB7/KBI3/PGD
PIC18F2620
220
LED-RED
LED-YELLOW
LED-GREEN

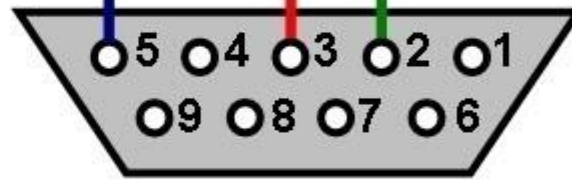


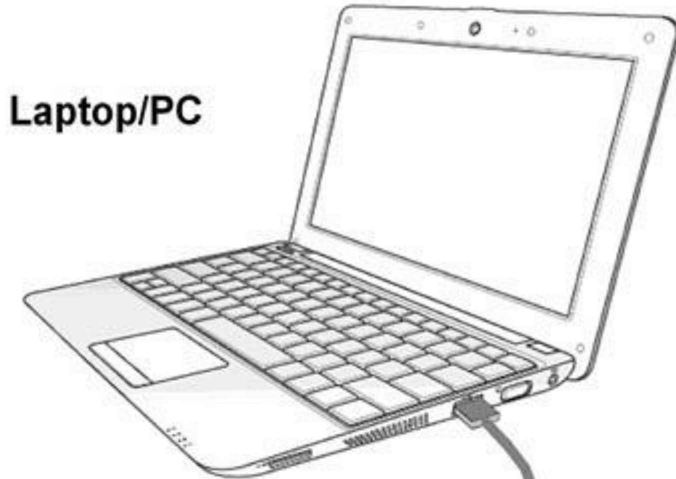
Receive (Rx)
Transmit (Tx)
Serial
Communication
Cable
VCC
10K
RESET
MCLR
RA0
RA1
RA2
RA3
RA4
RA5
RE0
RE1
RE2
Ddd
Vss
OSC1
OSC2
RC0
RC1
RC2
RC3
RD0
RD1
RB7
RB6
RB5
RB4
RB3
RB2
RB1
RB0
Vdd
Vss
RD7
RD6
RD5
RD4
RC7
RC6
RC5
RC4
RD3
RD2
PIC16F887
LED
330R
Receiving
Error
8MHz
2x20-30pF
10uF
100nF
MAX232
Tx
Rx

RS232 Cable
2 = TX
3 = RX
5 = GND

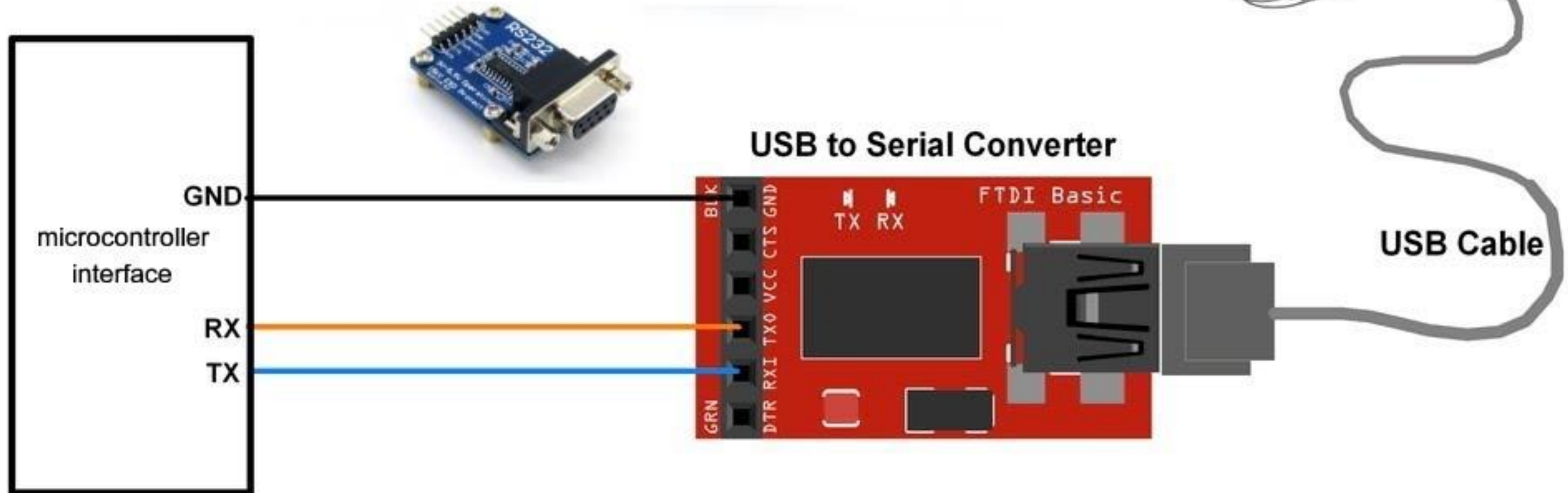

1 2 3 4 5
6 7 8 9
Microcontroller
(male Sub-D connector)

2 = RX
3 = TX
5 = GND


5 4 3 2 1
9 8 7 6
PC COM Port
(femal Sub-D connector)


Laptop/PC

1


GND
microcontroller
interface
RX
TX
RS232
USB to Serial Converter
FTDI Basic
TX RX
BLK
GND
CTS
VCC
TXO
RXI
DTR
GRN
USB Cable

Assembly Code for Transmitting Data

Here's a PIC ASM code that sends a character 'A' out of the serial port of the PIC16F877A at a baud rate of 9600.

```
ORG 0x00
goto init

init
   bsf STATUS, RP0   ;bank 1
 ;---CONFIGURE SPBRG FOR DESIRED BAUD RATE
 movlw D'25'   ;baud rate =  9600bps
 movwf SPBRG   ;at 4MHZ
;---CONFIGURE TXSTA
 movlw B'00100100'
 movwf TXSTA
;Configures TXSTA as 8 bit transmission, transmit enabled,
;async mode, high speed baud rate
 bcf STATUS, RP0   ;bank 0
 movlw B'10000000'
 movwf RCSTA   ;enable serial port receive

 movlw 0x41
 movwf char0   ;put A (ascii code 0x41) character to char0
;register

main movf char0, W
         movwf TXREG   ;place the A character to TXREG
         Call Test
         goto main

Test
                btfss TXSTA, TRMT ;check if TRMT is empty
                 goto Test   ;if not, check again
                bcf STATUS, RP0   ;bank 0, if TRMT is empty then the
;character has been sent
               return

 end
```

You should see a single ‘A’ on the serial monitor.

The first part of the code above configures the serial port by making SPBRG = 25 (as per the calculations on our example calculation) and enabling transmit (at high speed, async) and receive ports via TXTSA and RCSTA.

The character to be sent is inside char0 and then moved to TXREG. Then we looped the program using btfss (on TRMT bit) to check if the character has been sent out.

Receiving Data using Assembly

As mentioned, the received data through the serial port is stored in RCREG. When RCREG is read and emptied, a flag, RCIF is set.

RCIF is an interrupt flag that can be disabled using the RCIE bit in PIR1.

However, we don’t need to set interrupts for the RCIF to trigger!

Here's an assembly code that sets PORTB according to the value received from the serial port:

```
org 0x00
goto start

start bsf STATUS, RP0
        movlw .25
        movwf SPBRG
        movlw 0x24
        movwf TXSTA
        clrf TRISB
        bcf STATUS, RP0
       movlw 0x90
       movwf RCSTA

main  btfss PIR1, RCIF
          goto main
         movf RCREG, W
         movwf PORTB
         goto main
```

Thank you!