ICT 5307: Embedded Systems

Lecture 11
Serial Communication

Professor S.M. Lutful Kabir
BUET

Need for Serial Communication

- Computer transfers data in two ways: Parallel & Serial
- In parallel data transfer, eight or more lines (wire conductors) are used to transfer data to a device.
- Devices that uses parallel data transfer include printers, IDE Hard disk etc.
- Although a lot of data can be transferred in a short amount of time, by using many wires in parallel.
- To transfer many meters away, parallel method is costly, so serial method is used.

Serial Data Transfer

- In serial communication, the data is sent one bit at a time.

Parallel: expensive - short distance – fast

Serial :cheaper– long distance-slow

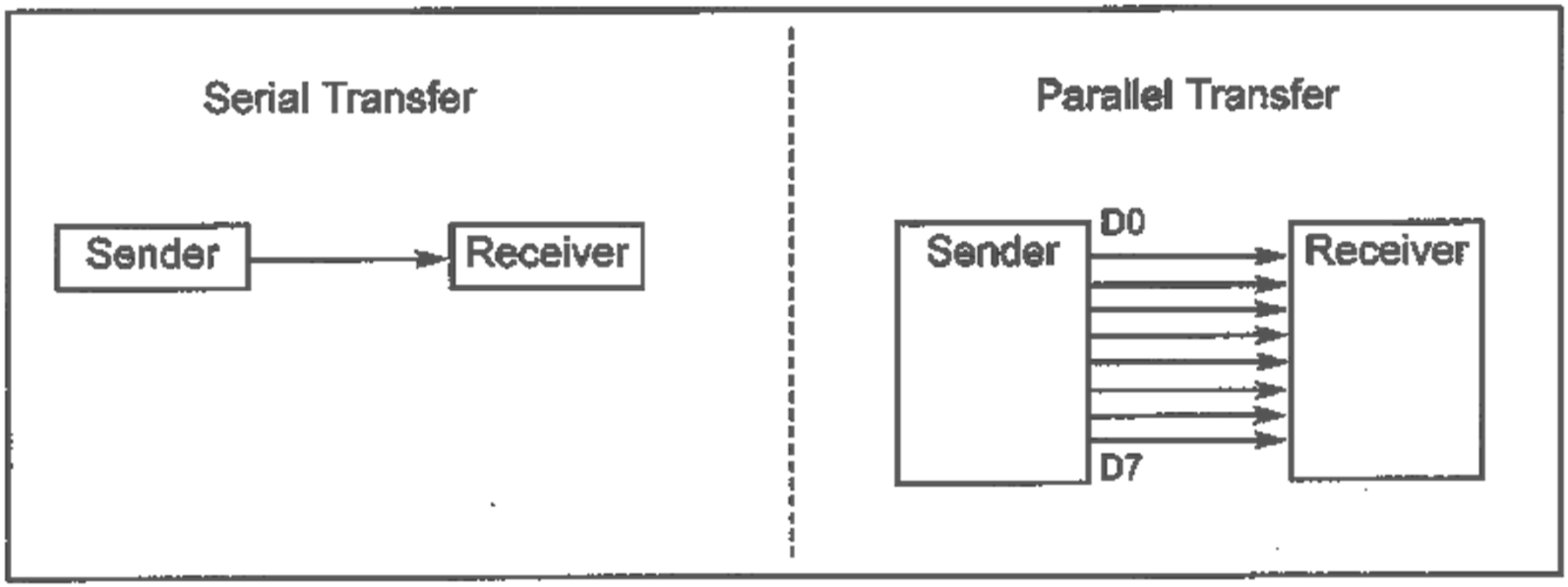


Figure 10-1. Serial versus Parallel Data Transfer

Serial Data Transfer

- For serial communication to work, the bytes of data must be converted to serial bits using a parallel-in-serial-out shift register.
- This also means that at the receiving end there must be a serial-in-parallel-out shift register in order to receive the serial data and pack them into a byte.
- For serial transfer, if the distance is short, the 1s and 0s are transferred directly from one device to other.
- For long distance the signal is modulated so that signal does not attenuate.
- This is done by a peripheral device called MODEM (stands for modulation/demodulation).

Methods of Serial Communication

- Serial Communication uses two methods, Synchronous and Asynchronous.
- The Synchronous method transfers a block of data (character) at a time.
- Whereas Asynchronous method transfers a single byte at a time.
- It is possible to write software to use either of the methods, the program can be tedious and long.
- That is why, special IC chips is made by many manufacturers for serial data communication.
- These chips are commonly referred to as Universal Asynchronous Transmitter/Receiver (UART) or Universal Synchronous- Asynchronous Transmitter/ Receiver (USART)
- AVR has built-in USART .

Simplex and Duplex Communication

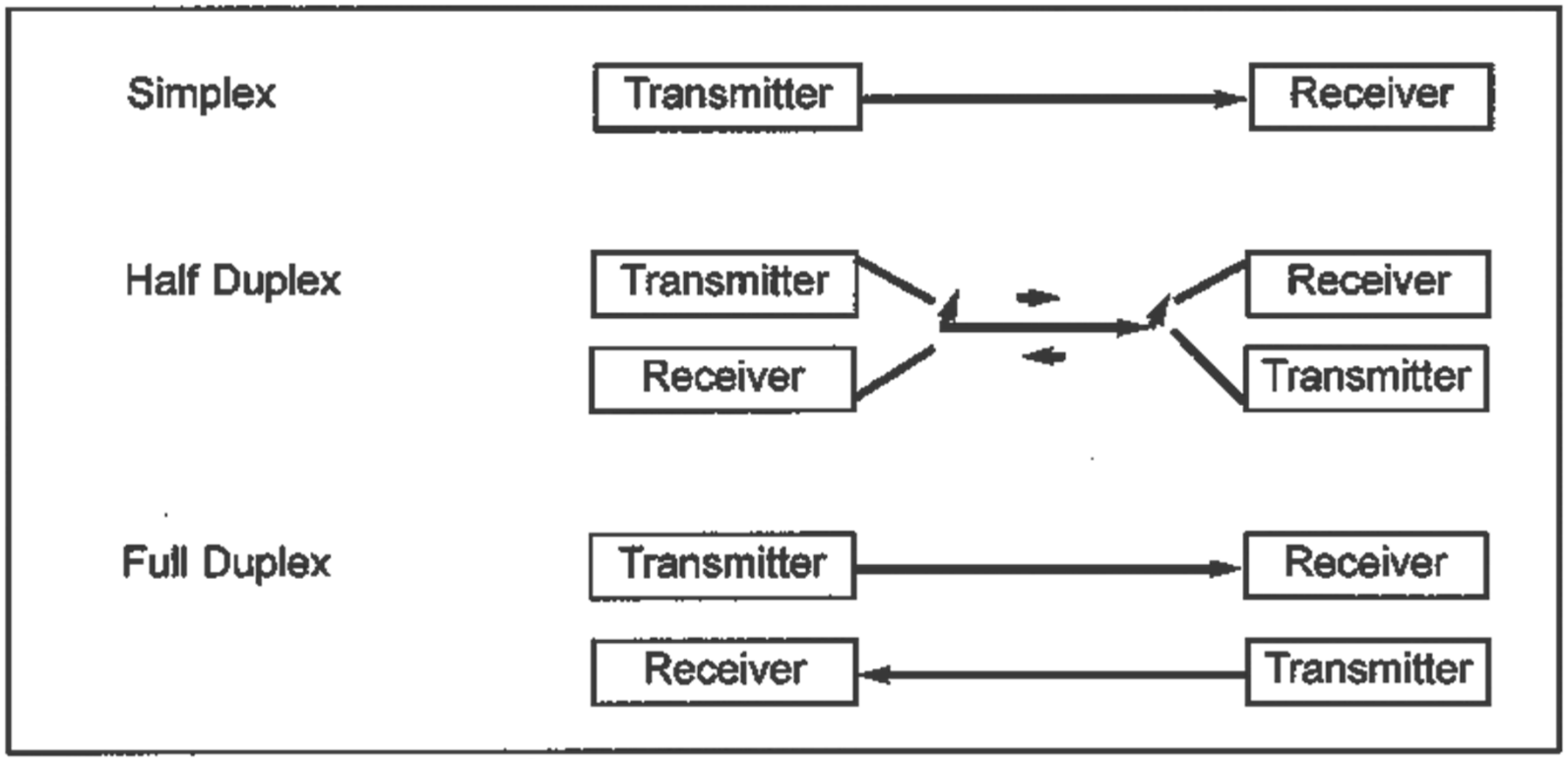


Figure 10-2. Simplex, Half-, and Full-Duplex Transfers

Protocol

- The data coming in at the receiving end of the data line in a serial data transfer is all 1s and 0s.
- So, it is difficult to make sense of the data unless the sender and the receiver agree on a set of rules, called Protocol, on
  - how the data is packed,
  - how many bits constitute a character, and
  - how the data begins and ends.

Asynchronous Communication and Data Framing

- When there is no transfer the signal is high
- Transmission begins with a start (low) bit
- LSB first
- Finally 1 stop bit (high)
- Data transfer rate (baud rate) is stated in bps

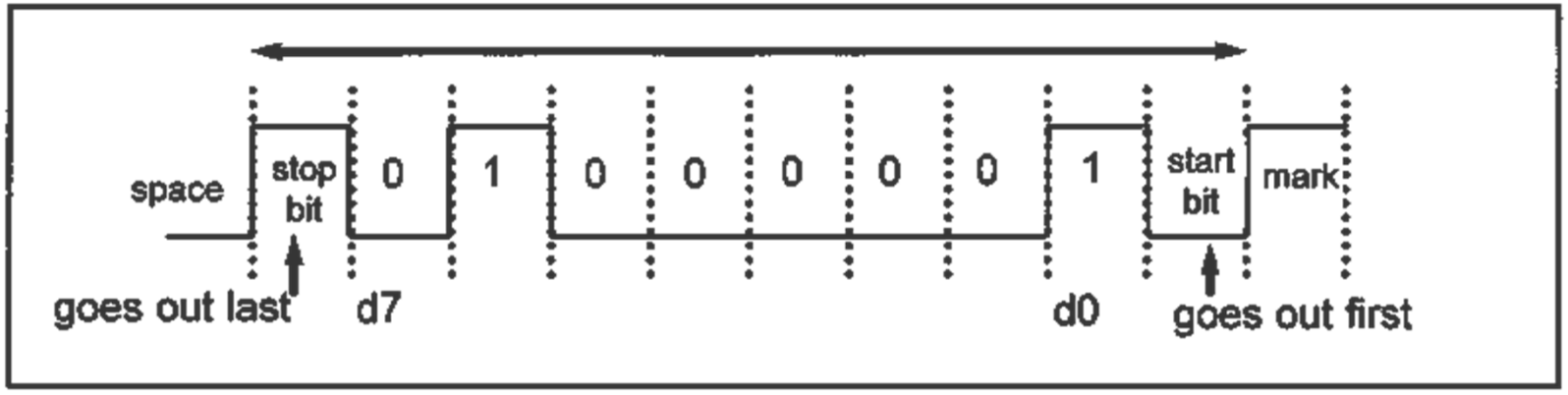


Figure 10-3. Framing ASCII "A" (41H)

Data Frame

- In Asynchronous serial communication, peripheral chips or modem can be programmed for data that is 7 bit or 8 bit wide.
- This is in addition of stop bits, 1 or 2.
- In present days, 10 bits for each character is transmitted: 8 bits for each ASCII code, and 1 bit for each start and stop bits.
- Therefore each 8 bit requires 2 extra bits, which gives 25% overhead.

Parity Bit

- In some systems, the parity bit of the data byte is included in the data frame in order to maintain data integrity.
- This parity bit is odd or even.
- In the case of odd parity number of 1s in the data byte including the parity bit is odd.
- Similarly for even parity, number of 1s in the data byte including the parity bit is even.
- UART/USART chips can be programmed for odd, even or no-parity options.

Data Transfer Rate

- The rate of data transfer in serial communication is called bits per second (bps).
- Another widely used terminology is called baud rate.
- However, baud rate and bps rates are not necessarily equal.
- This is because baud rate is used in modem and is defined as the number of signal changes per second.
- A single change of signal may transfer several bits of data
- As far as conductor wire is concerned, the baud rate and bps are the same and for this reason we shall use them interchangeably.

Data Communication Classification

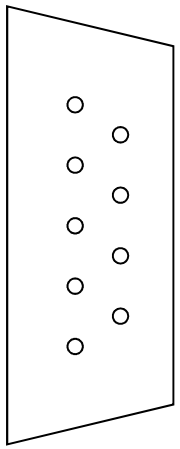
- Data communication equipments are classified as
  - Data Terminal Equipment (DTE)
  - Data Communication Equipment (DCE)
- DTE refers to terminals and computers that send and receive data
- DCE refers to communication equipments such as modem that are responsible for transferring data.

RS232 Standard

- In order to allow compatibility among data communication equipment made by various manufacturer, an interfacing standard called RS232 was set by Electronic Industries Association (EIA) in 1960.
- RS232 is one of the most widely used serial I/O interfacing standards.
- This standard is used by PCs and numerous types of equipment.

RS 232 Standard

- Logic 1 : -3 to -25 volt
- Logic 0 : 3 to 25 volt
- To Connect TXD to RXD and RXD to TXD from pc to AVR you must use MAX232 IC (or equivalent IC) to convert signal from TTL level to RS232 level
- The baud rate of the AVR must match the baud rate of the pc
- Standard baud rates are
  - 2400-4800-9600-14400-19200-28800-33600-57600



1	DCD
2	RD
3	TD
4	DTR
5	GND
6	DSR
7	RTS
8	CTS
9	RI

DB-9 Port

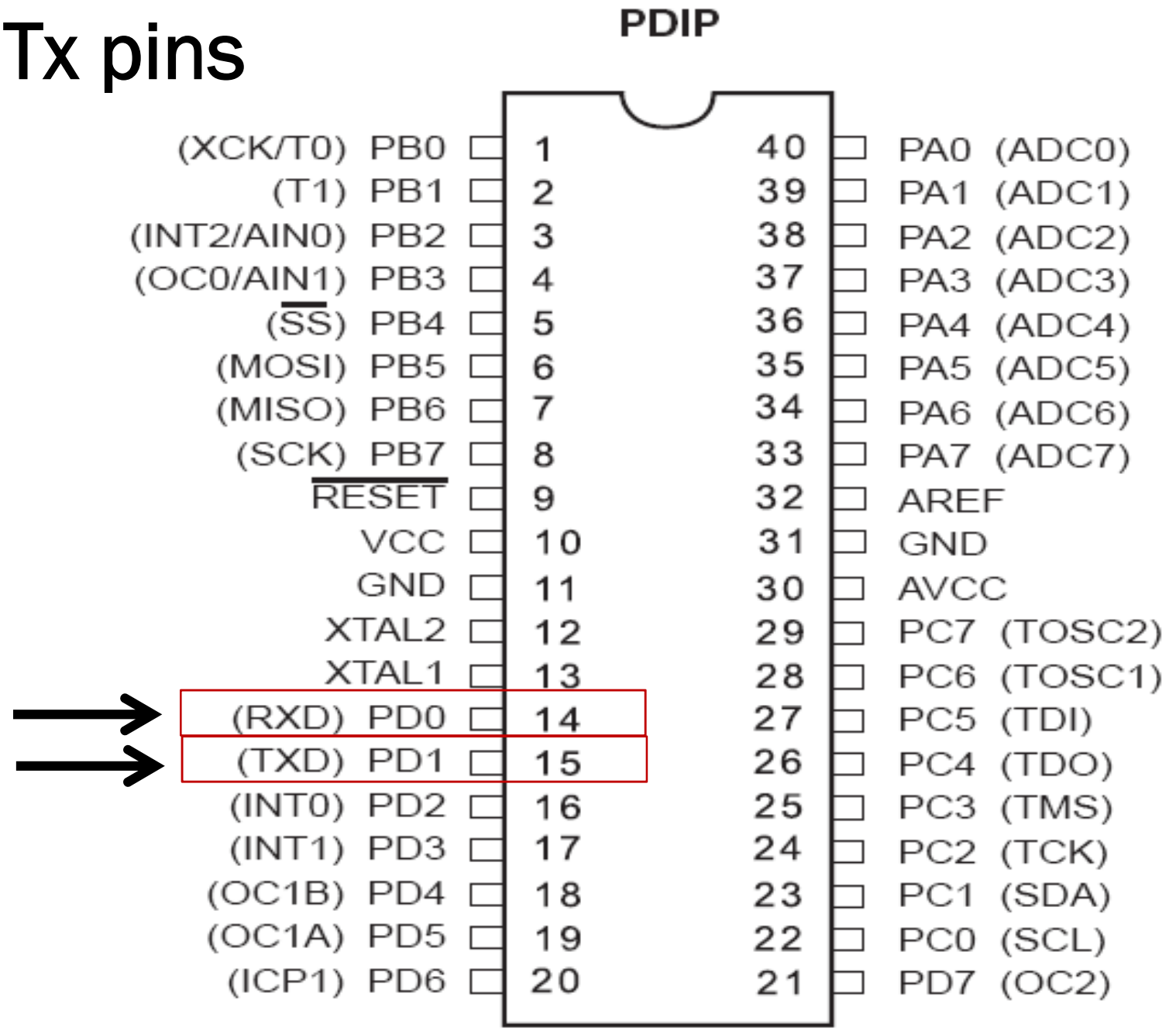
Table 11-2: IBM PC DB-9 Signals

Pin	Description
1	Data carrier detect ($\overline{\text{DCD}}$)
2	Received data (RxD)
3	Transmitted data (TxD)
4	Data terminal ready (DTR)
5	Signal ground (GND)
6	Data set ready ($\overline{\text{DSR}}$)
7	Request to send ($\overline{\text{RTS}}$)
8	Clear to send ($\overline{\text{CTS}}$)
9	Ring indicator (RI)

Rx and Tx Pins in ATmega32 uC

- The ATmega32 has 2 pins that are used specially for transferring and receiving data.
- These pins are called Rx and Tx pins and part of group portD (PD0 and PD1) of the 40-pin package.
- Pin 15 is assigned to Tx and Pin 14 as Rx.
- These Pins are TTL compatible that is why they require a line driver to make them RS232 compatible.

Rx and Tx pins



MAX 232 or 233

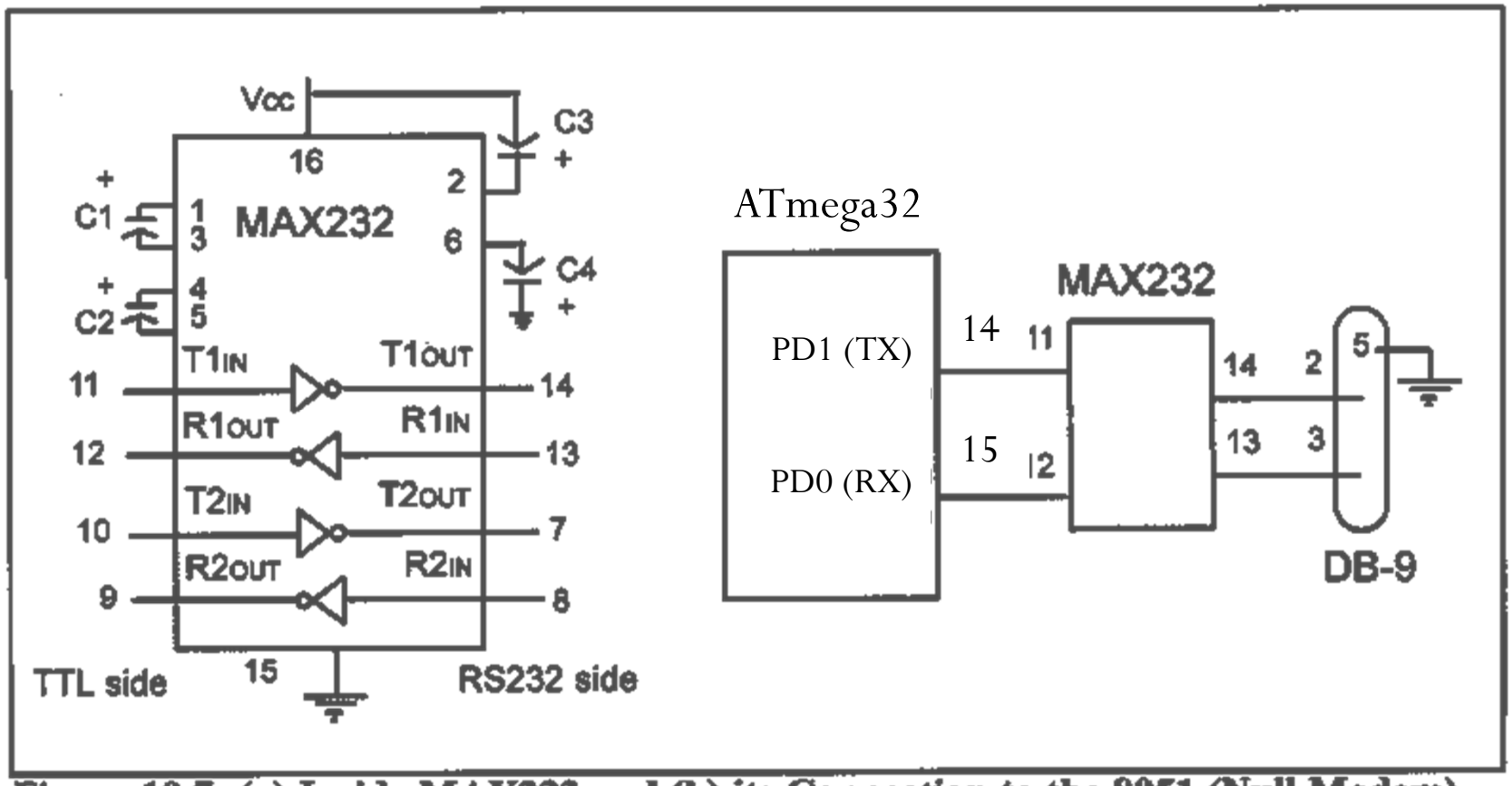


Figure 10.7: (a) Inside MAX232 and (b) its Connection to the 8051 (Null Modem)

USART modes in AVR

- In four mode:
  - Normal asynchronous
  - Double-speed asynchronous
  - Master synchronous
  - Slave synchronous

Registers used for Serial Communication

- UDR (USART Data Register)
- UCSRx (USART Control and Status Register);

  [x=A, B or C]
- UBRR (USART Baud Rate Register)

UBRR Register and Baud Rate in AVR

- Some of the baud rates supported by PC are 1200, 2400, 4800, 9600, 19200, 38400, 57600, and 115200
- AVR transfers and receives data in different baud rate and its baud rate is programmable.
- This is done by **U**SART **B**aud **R**ate **R**egister (UBRR).
- For a given crystal frequency, the value of UBRR decides the baud rate.

The Relationship Between Baud Rate and the Value in UBRR register

- The relationship between the baud rate and the UBRR register value is given by:

$$Desired\ Baud\ Rate = \frac{F_{osc}}{16\,(X+1)}$$

Where, X=Value in UBRR Register

- In order to get the values of 'X' for different baud rate we can solve the above equation, it gives,

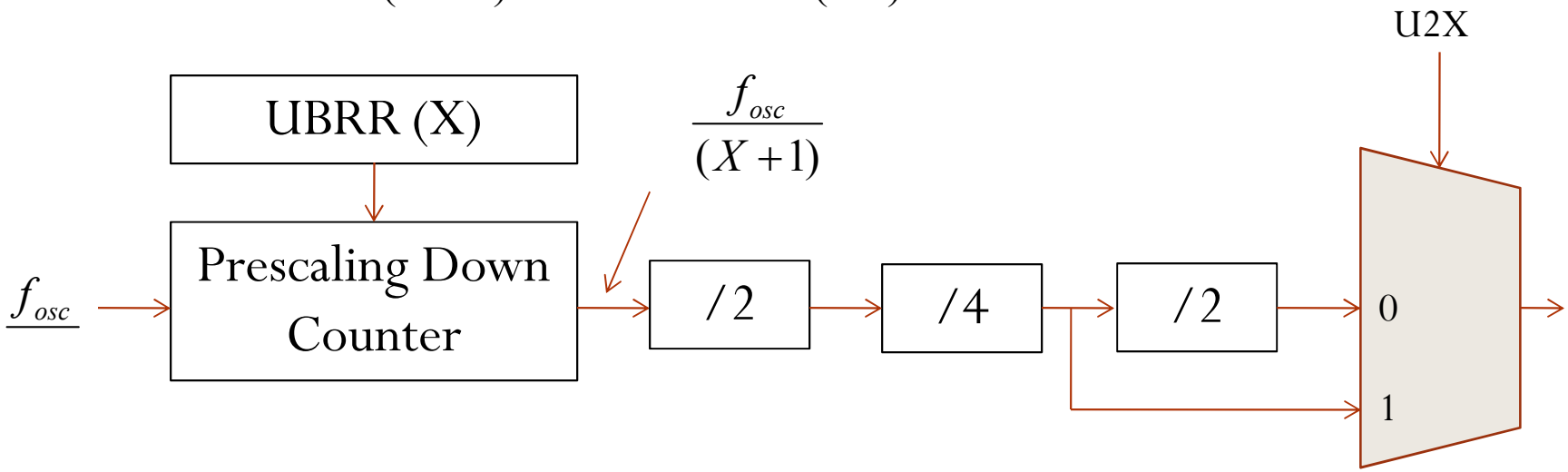
$$X = \frac{F_{osc}}{16\,(Desired\ Baud\ Rate)} - 1$$

UBRR Values corresponding to Baud Rate (F_{osc}=16MHz)

Baud Rate	UBRR (Decimal Value)	UBRR (Hex Value)
38000	25	19
19200	51	33
9600	103	67
4800	207	CF
2400	415	19F
1200	831	33F

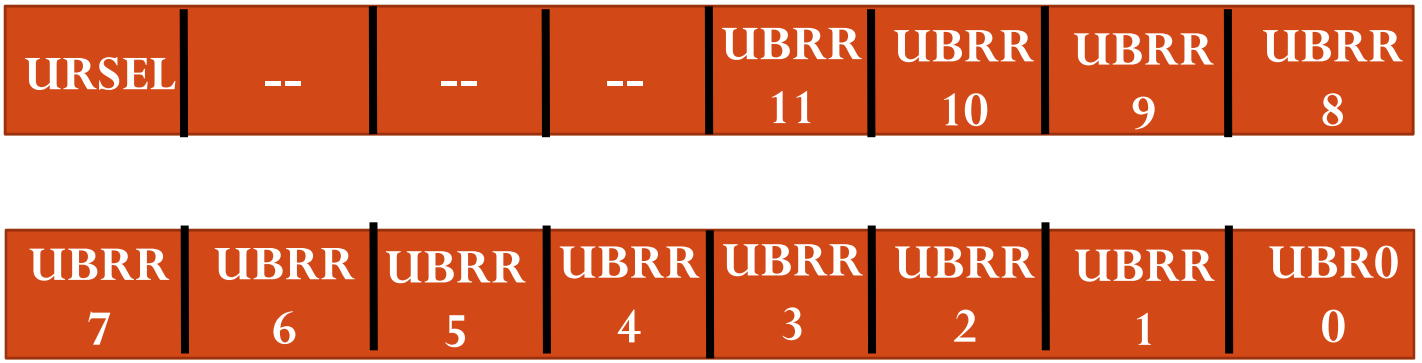
UBRR Register

$$BR = \frac{f_{osc}}{16\,(X+1)} \qquad X = \frac{f_{osc}}{16\,(BR)} - 1$$



With the selection of U2X bit (of UCSRA register - will be discussed later) the last block can be bypassed. That means the baud rate can be doubled.

UBRR Register



- UBRR is a 16-bit register but only 12 bits are used to set baud rate of USART of AVR
- Bit 15 is URSEL used for selection between UBRRH and UCSRC;

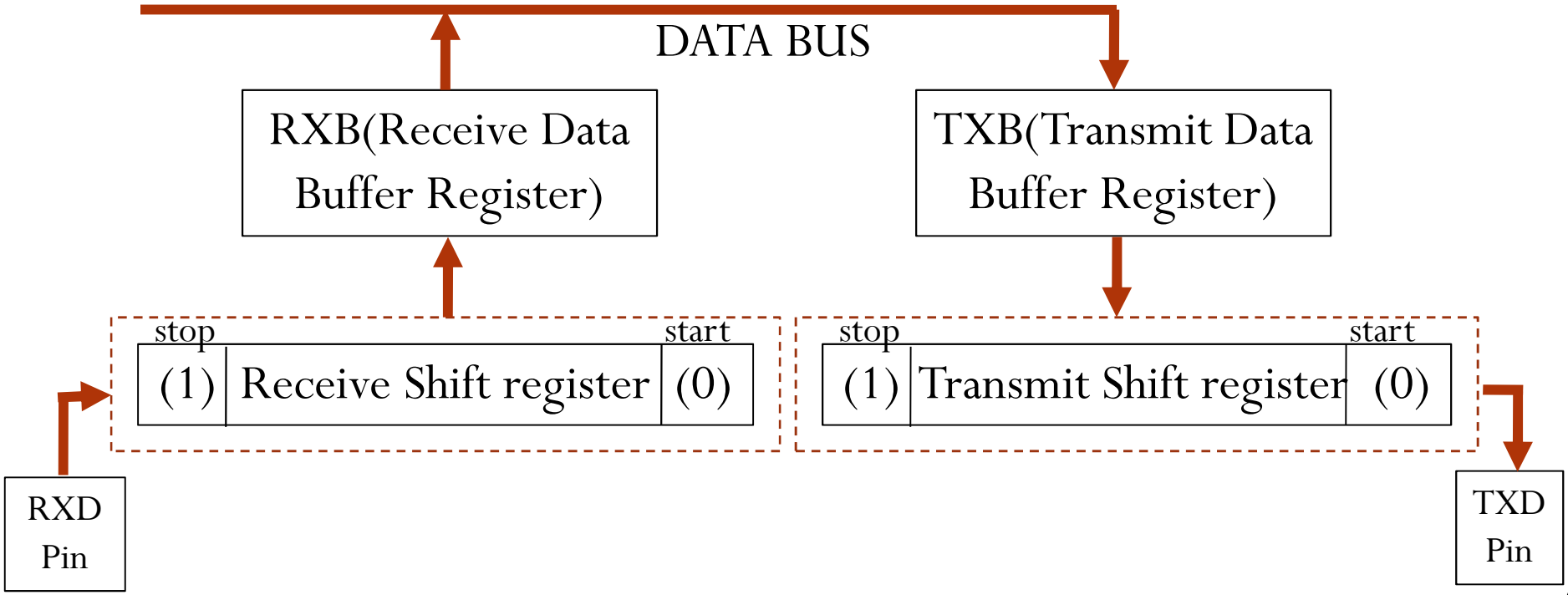
  0-UBRRH & 1-UCSRC

UDR Registers and USART Data I/O in AVR

- In AVR, in order to provide a full duplex serial communication, there are two shift registers referred to as *Transmit Shift Register* and *Receive Shift Register*
- Each shift register has a buffer that is connected to it directly. They are called –
- Transmit Data Buffer Register and Receive Data Buffer Register
- USART Transmit Data Buffer Register and USART Receive Data Buffer Register share the I/O memory, which is called USART Data Register or UDR

UDR Register

- When you write data to UDR, it will be transferred to Transmit Data Buffer Register (TXB) and when you read data from UDR it is read from Receive Data Buffer Register (RXB).



UCSRx Registers

- UCSRx are 8-bit Control registers used for controlling serial communication in the AVR.
- There are three USART Control registers in the AVR.
- They are UCSRA, UCSRB and UCSRC.

UCSRA

RXC	TXC	UDRE	FE	DOR	PE	U2X	MPCM

USCRA Register

USCRA Register

RXC	TXC	UDRE	FE	DOR	PE	U2X	MPCM

- RXC – USART receive complete
- TXC – USART transfer complete
- UDRE : USART data register is empty.
- FE – Frame Error
- DOR – Data Overrun
- PE – Parity Error
- U2X – Double the USART transmission speed
- MPCM – Multi- Process Communication Mode

RXC and TXC bits in USCRA register

- **RXC – USART Receive Complete**
  - This flag is set when there are new data in the receive buffer that are not read yet.
  - It is cleared when the receive buffer is empty.
- **TXC – USART Transmit Complete**
  - This flag is set when entire frame in the transmit shift register has been transmitted and there are no new data available in the transmit data buffer register.
  - It can be cleared by writing a 1 in its bit location.
  - Also it is automatically cleared when a transmit complete interrupt is executed.

UDRE and FE bits in USCRA register

- **USRE – USART Data register empty**
  - This flag is set when the transmit data buffer is empty and it is ready to receive new data.
  - If this bit is cleared, you should not write to UDR because it overrides your last data.
  - This flag can generate a data register empty interrupt.
- **FE – Frame Error**
  - This bit is set if a frame error has occurred in receiving next character in the receive buffer.
  - A frame error is detected when the first stop bit of the next character in the receive buffer is zero.

DOR and PE bits of USCRA register

- **DOR – Data Overrun**
  - This bit is set if data overrun is detected.
  - A data overrun occurs when the receive data buffer and receive shift register are full and a new start bit is detected.
- **PE – Parity Error**
  - This bit sets if the parity checking was enabled (UPM1=1 in USCRC register) and the character in the receive buffer had a parity error when received

U2X and MPCM bits in USCRA register

- **U2X – Double the USART transmission speed**
  - Setting this bit will double the transfer rate in asynchronous communication.
- **MPCM – Multi-processor Communication mode**
  - This bit enables the multi-processor communication mode.
  - To understand this feature you may consult the data sheet.

UCSRB Register

UCSRB Register

RXCIE	TXCIE	UDRIE	RXEN	TXEN	UCSZ2	RXB8	TXB8

- RXCIE – Receive Complete Interrupt Enable
- TXCIE – Transfer Complete Interrupt Enable
- UDRIE : USART Data Register Empty interrupt.
- RXEN – Receive Enable
- TXEN – Transfer Enable
- UCSZ2 – Character Size
- RXB8 – Receive Data bit 8
- TXB8 – Transmit Data bit 8

RXCIE and TXCIE bits of USCRB Register

- **RXCIE – Receive Complete Interrupt Enable**
  - To enable the interrupt on the RXC flag in USCRA register you should set this bit to one.
- **TXCIE – Transmit complete Interrupt Enable**
  - To enable the interrupt on the TXC flag in USCRA register you should set this bit to one.

UDRIE, REN and TEN bits in USCRB Register

- **UDRIE – USART Data Register Empty Interrupt Enable**
  - To enable the interrupt on UDRE flag in USCRA register you should set bit to one.
- **RXEN – Receive Enable**
  - To enable USART Receiver enable you should set this bit to one.
- **TXEN – Transmit Enable**
  - To enable USART transmitter enable you should set this bit to one.

UCSZ2, RXB8 and TXB8 bits in USCRB Register

- **UCSZ2 – Character Size**
  - This bit combined with UCSZ0:1 bits in USCRC register sets the number of data bits (character size) in a frame.
- **RXB8 – Receive data bit 8**
  - This is the 9^{th} bit of received character when using serial frame with 9 data bits. This case will not be discussed in our course.
- **TXB8 – Transmit data bit 8**
  - This is the 9^{th} bit of the transmitted character when using serial frame with 9 data bits. This case will not be discussed in our course.

UCSRC Register

UCSRC Register

URSEL	UMSEL	UPM1	UPM0	USBS	UCSZ1	UCSZ0	UCPOL

- URSEL – Register Select
- UMSEL – USART mode Select
- UPM1:0 – Parity Mode
- USBS – Stop bit select
- UCSZ1:0– Character Size
- UCPOL – Clock Polarity

URSEL and UMSEL bits of USCRC Register

- **URSEL – Register Select**
  - This bit used to access either the USCRC register or UBRRH register.
- **UMSEL – USART Mode Select**
  - This bit selects either to operate in Asynchronous or in Synchronous mode.

    0 = Asynchronous and

    1 = Synchronous

UPM1:0 and USBS bits in USCRC Register

- **UPM 1:0 – Parity Mode**

  00 = Disabled

  01 = Reserved

  10 = Even parity

  11 = Odd parity

- **USBS – Stop bit select**

  0 = 1 bit

  1 = 2 bits

UCSZ1:0 and UCPOL bits in USCRC Register

- **UCSZ1:0 – Character Size**
  - These two bits along with UCSZ2 in USCRB register are used for setting the character size.

UCSZ2	UCSZ1	UCSZ0	SIZE
0	0	0	5
0	0	1	6
0	1	0	7
0	1	1	8
1	1	1	9

- **UCPOL – Clock Polarity**
  - This is for Synchronous mode only, will not be covered.

Summary of the Registers

Data

UDR7	UDR6	UDR5	UDR4	UDR3	UDR2	UDR1	UDR0

Baud Rate

URSEL	--	--	--	UBRR 11	UBRR 10	UBRR 9	UBRR 8
UBRR 7	UBRR 6	UBRR 5	UBRR 4	UBRR 3	UBRR 2	UBRR 1	UBR0 8

Control

USCRA Register

RXC	TXC	UDRE	FE	DOR	PE	U2X	MPCM

UCSRB Register

RXCIE	TXCIE	UDRIE	RXEN	TXEN	UCSZ2	RXB8	TXB8

UCSRC Register

URSEL	UMSEL	UPM1	UPM0	USBS	UCSZ1	UCSZ0	UCPOL

To Transfer Character Serially

- Step #1: The UCSRB register is loaded with 08H, enabling the USART transmitter. The transmitter will override normal port operation of the TXD pin when enabled.
- Step #2: The UCSRC register is loaded with 86H, indicating asynchronous mode with 8-bit data frame, no parity and one stop bit.

UCSRB

RXCIE	TXCIE	UDRIE	RXEN	TXEN	UCSZ2	RXB8	TXB8

UCSRC

URSEL	UMSEL	UPM1	UPM0	USBS	UCSZ1	UCSZ0	UCPOL

To Transfer Character Serially

- Step #3: The UBRR register should be loaded with one of values required to set a desired baud rate.
- Step #4: Monitor the UDRE bit of UCSRA register to make sure that UDR is ready for sending a byte.
- Step #5: The character byte to be transmitted serially should be written in UDR register
- Step#6: To transmit next character go to step #4

UCSRA

RXC	TXC	UDRE	FE	DOR	PE	U2X	MPCM

Programming the AVR to receive the data serially

- Step#1: The UCSRB register is loaded with 10H, enabling USART receiver. The receiver will override normal port operation for the RXD pin when enabled.
- Step#2: The UCSRC register is loaded with 86H, indicating asynchronous mode with 8-bit frame, no parity and one stop bit.
- Step #3: The UBRR is loaded with appropriate value for setting the baud rate

Programming the AVR to receive the data serially

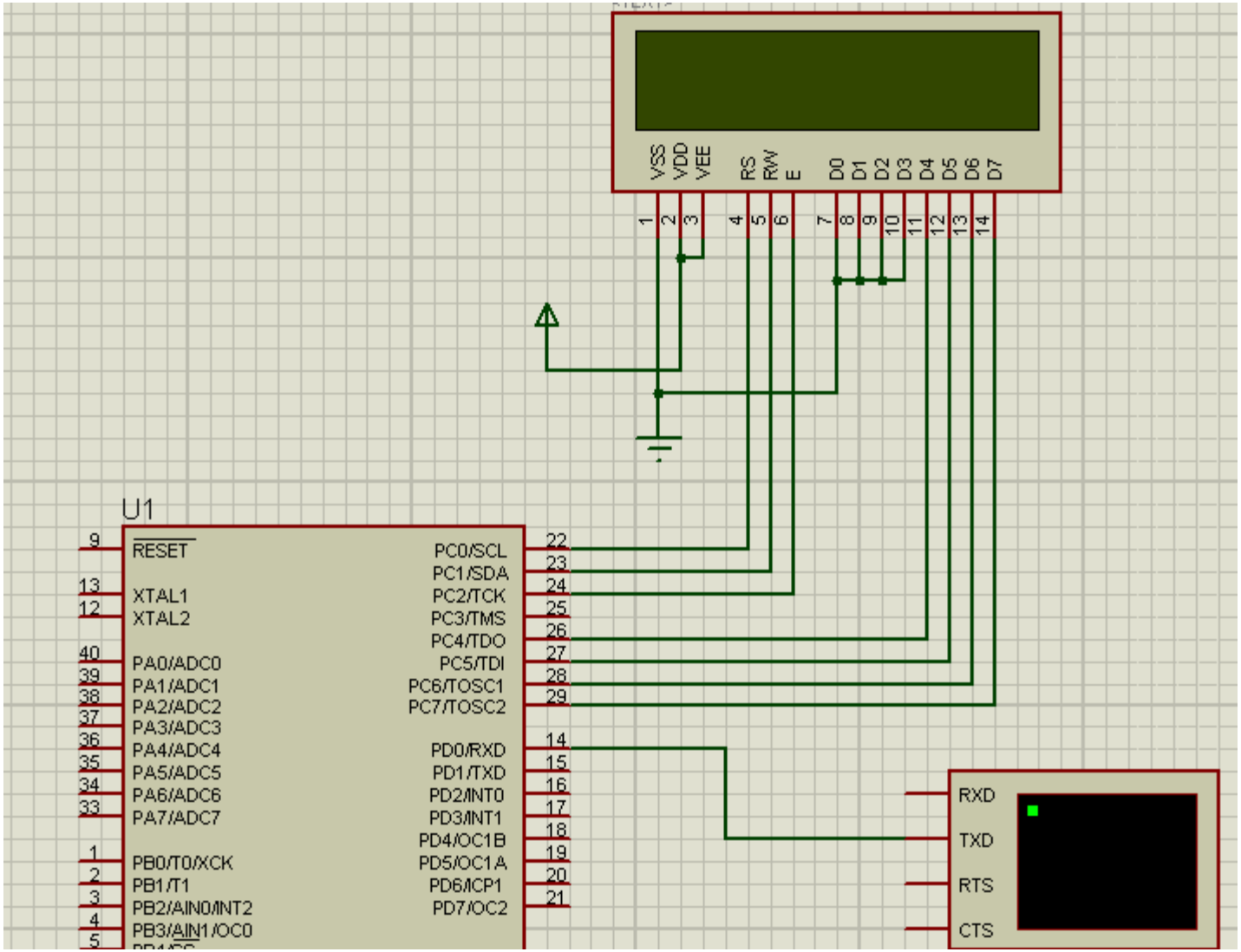
- Step#4: The RXC flag bit of UCSRC register is monitored for a HIGH to see if an entire character has been received yet.
- Step#5: When RXC is raised, the UDR register has the byte. Its content is read and saved.
- Step#6: To receive the next data go to step 4.

An Example Program for Receiving data (*using USART's Receive Interrupt*)

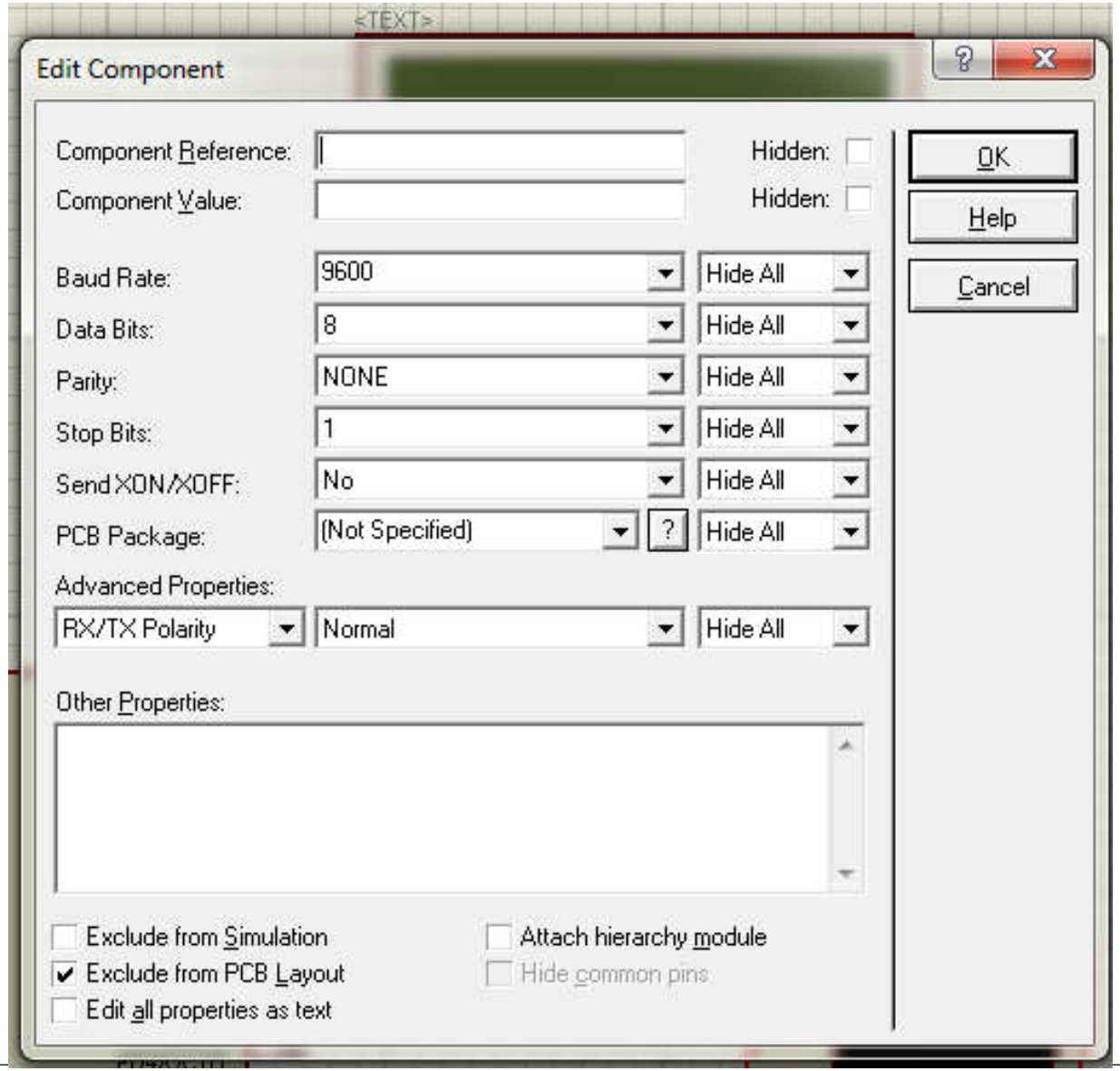
Write a C program for the AVR to receive data at baud rate of 9600 baud, continuously. Use 8-bit data, 1 stop bit and no parity. Assume XTAL=16 MHz.

- 8-bit data so, UCSZ2:0=001
- 1 stop bit, so, USBS1:0=00
- No parity. So, UPM1:0=00;
- Baud rate=9600, so, for f_{osc}=16MHz, X=0x0067; So, UBRRH=0x00 and UBRRL=0x67
- We have to enable Receive, so, RXE=1
- We have to enable Receive Interrupt, so, RXCIE=1

Proteus Drawing



Settings in the Virtual Terminal



The Code

```
#include <mega32.h>
#include <delay.h>
#include <alcd.h>
#include <stdio.h>
char mychar;
bit i;
// Interrupt Serial Receive
interrupt [USART_RXC] void
   usart_rxc_isr(void)
{
   mychar=getchar();   // get character
   from data stored in serial register
   (UDR)
   i=1;
}
```

```
void main(void)
{
DDRC=0xFF;
UCSRA=(0<<RXC) | (0<<TXC) |
   (0<<UDRE) | (0<<FE) |
   (0<<DOR) | (0<<UPE) |
   (0<<U2X) | (0<<MPCM);
UCSRB=(1<<RXCIE) |
   (0<<TXCIE) | (0<<UDRIE) |
   (1<<RXEN) | (0<<TXEN) |
   (0<<UCSZ2) | (0<<RXB8) |
   (0<<TXB8);
UCSRC=(1<<URSEL) |
   (0<<UMSEL) | (0<<UPM1) |
   (0<<UPM0) | (0<<USBS) |
   (1<<UCSZ1) | (1<<UCSZ0) |
   (0<<UCPOL);
UBRRH=0x00;
UBRRL=0x67;
```

Code (continued...)

```
# asm("sei");        //to enable Global Interrupt bit
lcd_init(16);
while (1)
      {
        if(i==1)
        {
          lcd_putchar(mychar);
          i=0;
        }
      }
}
```

Thanks