

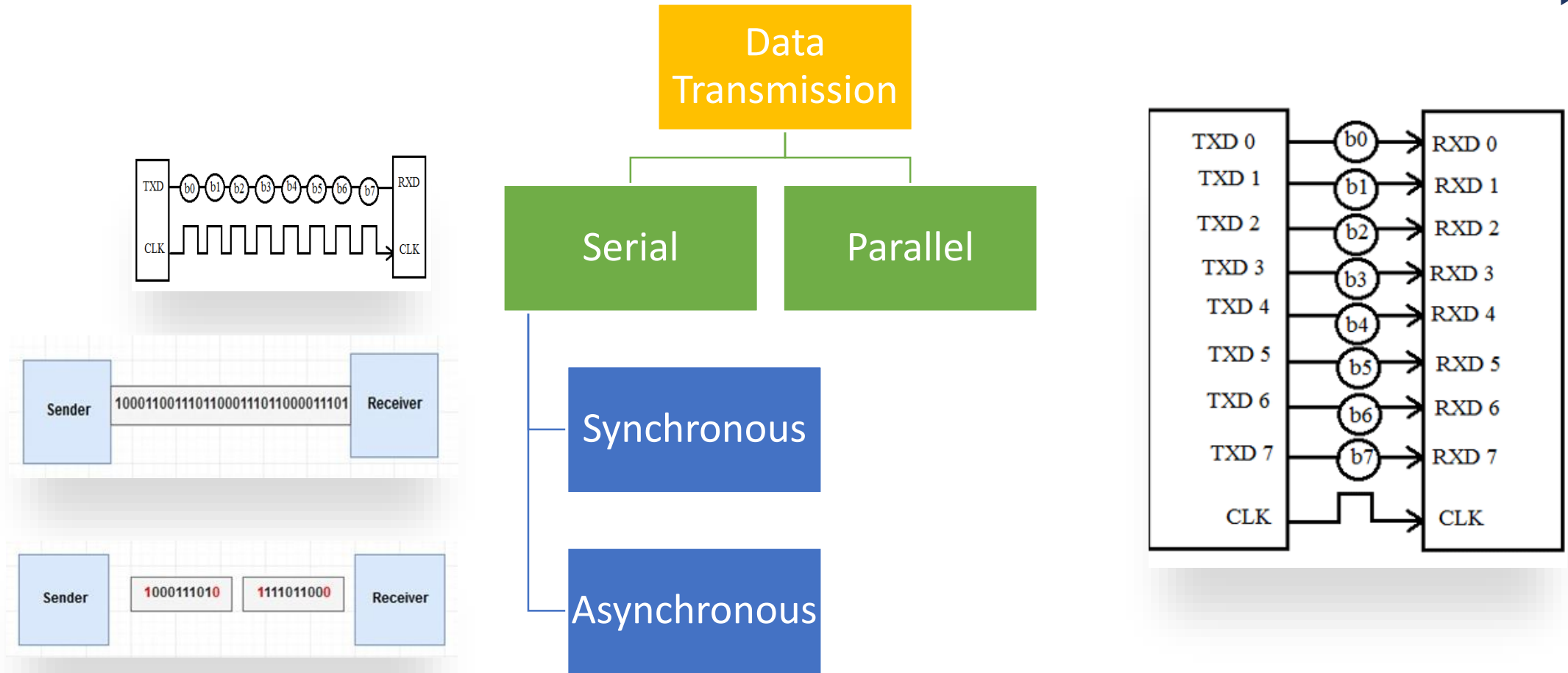
Mechatronics and Robotics

# RS-232C Serial Communication

Dr. Mangesh Deshpande

VIT Pune

# Data Transmission



# Data Communication

## Serial Communication

- Data is transmitted bit by bit
- Speed: Slow
- 1/2 data lines
- Cost Effective

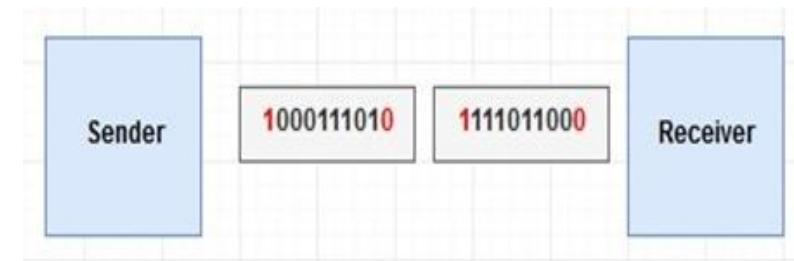
## Parallel Communication

- A byte/character is transmitted at a time
- Speed: Fast
- Multiple data lines
- Costly

# Modes of Data Transfer in Serial Communication

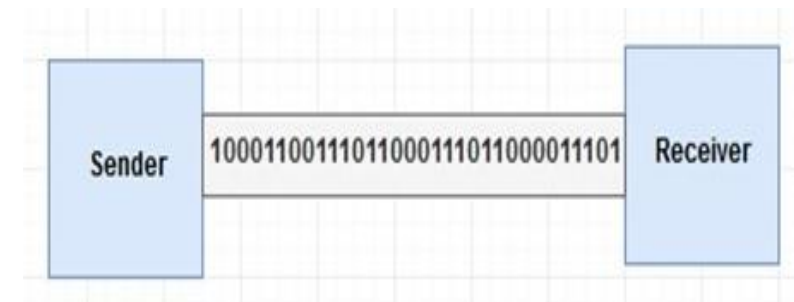
## Asynchronous Data Transfer

- Asynchronous Communication has **no timing signal or clock**. Instead, it inserts Start / Stop bits into each byte of data to "synchronize" the communication.
- E.g. RS232, RS422, RS485



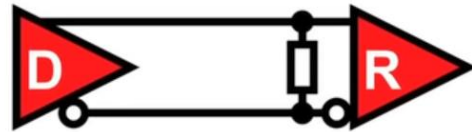
## Synchronous Data Transfer

- Synchronous Communication requires the sender and receiver to **share the same clock**.
- E.g. SPI (Serial Peripheral Interface), I2C (Inter-integrated Circuits)

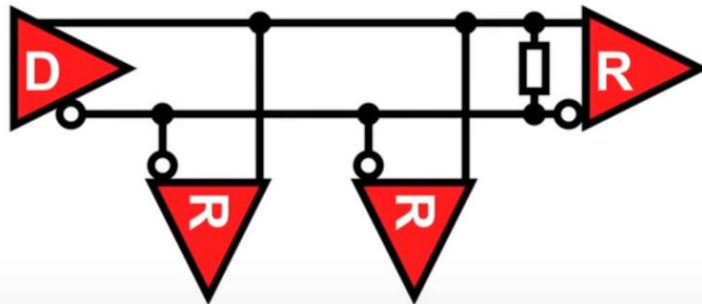


# Bus Topologies

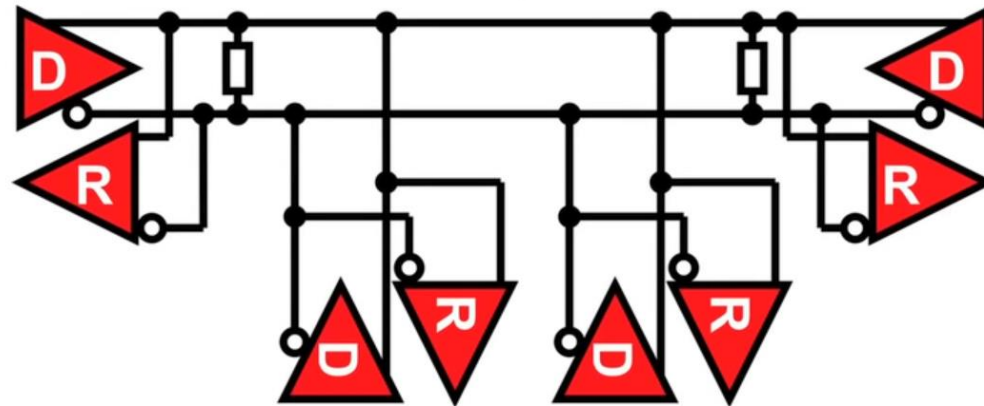
Point-to-Point



Multi-drop



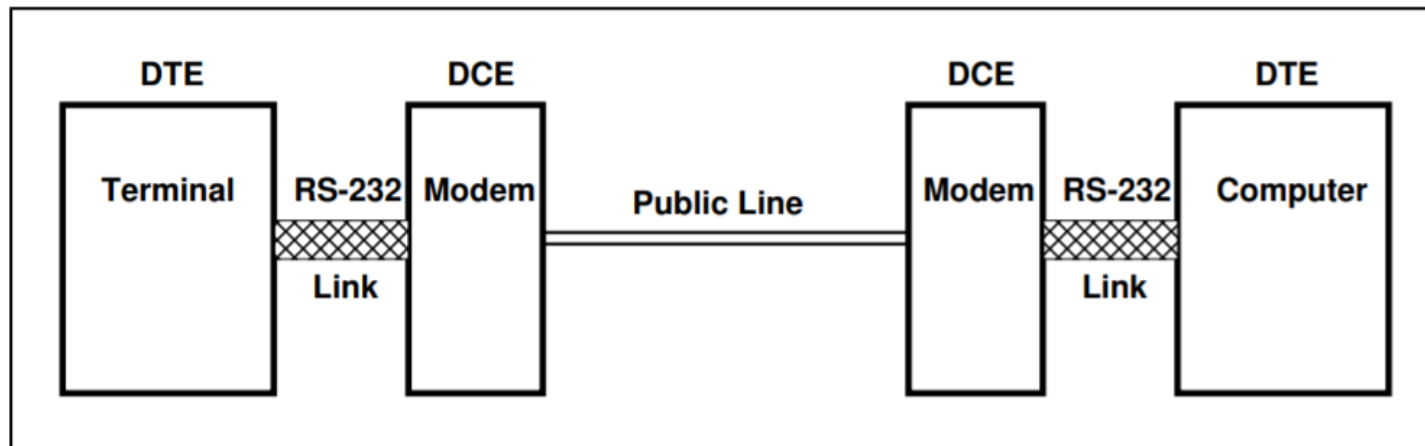
Multi-point



D: Driver  
R: Receiver

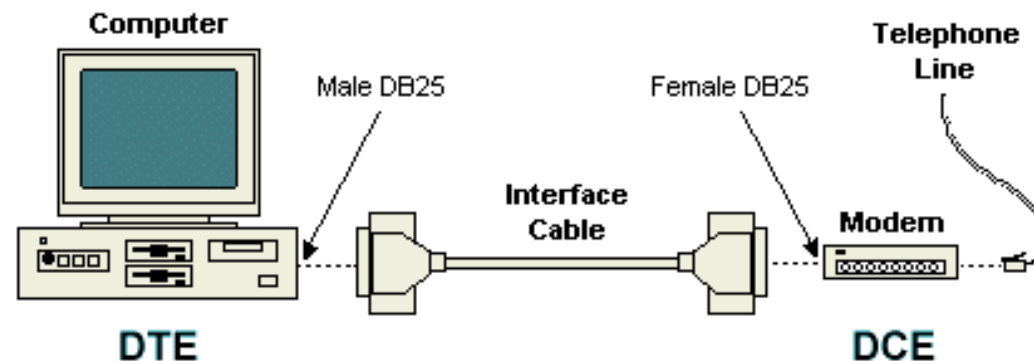
# RS-232C

- Developed by Electronic Industries Association (EIA) for asynchronous serial communications links
- The RS-232C is commonly abbreviated RS-232
- Its primary function application was in short-distance, low-bitrate links between devices (computers and modems) in clean-room environments.



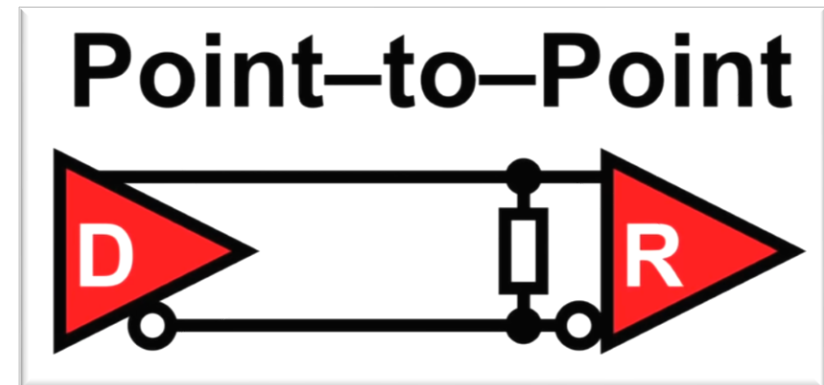
# RS232: Recommended Standard 232

- The RS-232 interface is the Electronic Industries Association (EIA) standard for the interchange of serial binary data between two devices.
- It is used in serial communication up to **50 feet** with the rate of **1.492kbps**.
- As EIA defines, the RS232 is used for connecting Data Transmission Equipment (DTE) and Data Communication Equipment (DCE).



# RS-232

- **Single ended point-to-point** bus topology
- Logic levels
  - Driver End: Logic 0: Voltage between +5 V to +15 V
  - Driver End : Logic 1: Voltage between -5 V to -15 V
  - Receiver End: Voltage between +3V to +15V is recognized as logic 0
  - Receiver End: Voltage between -3 V to -15 V is recognized as logic 1
  - Voltage between -3 V to +3 V is undefined



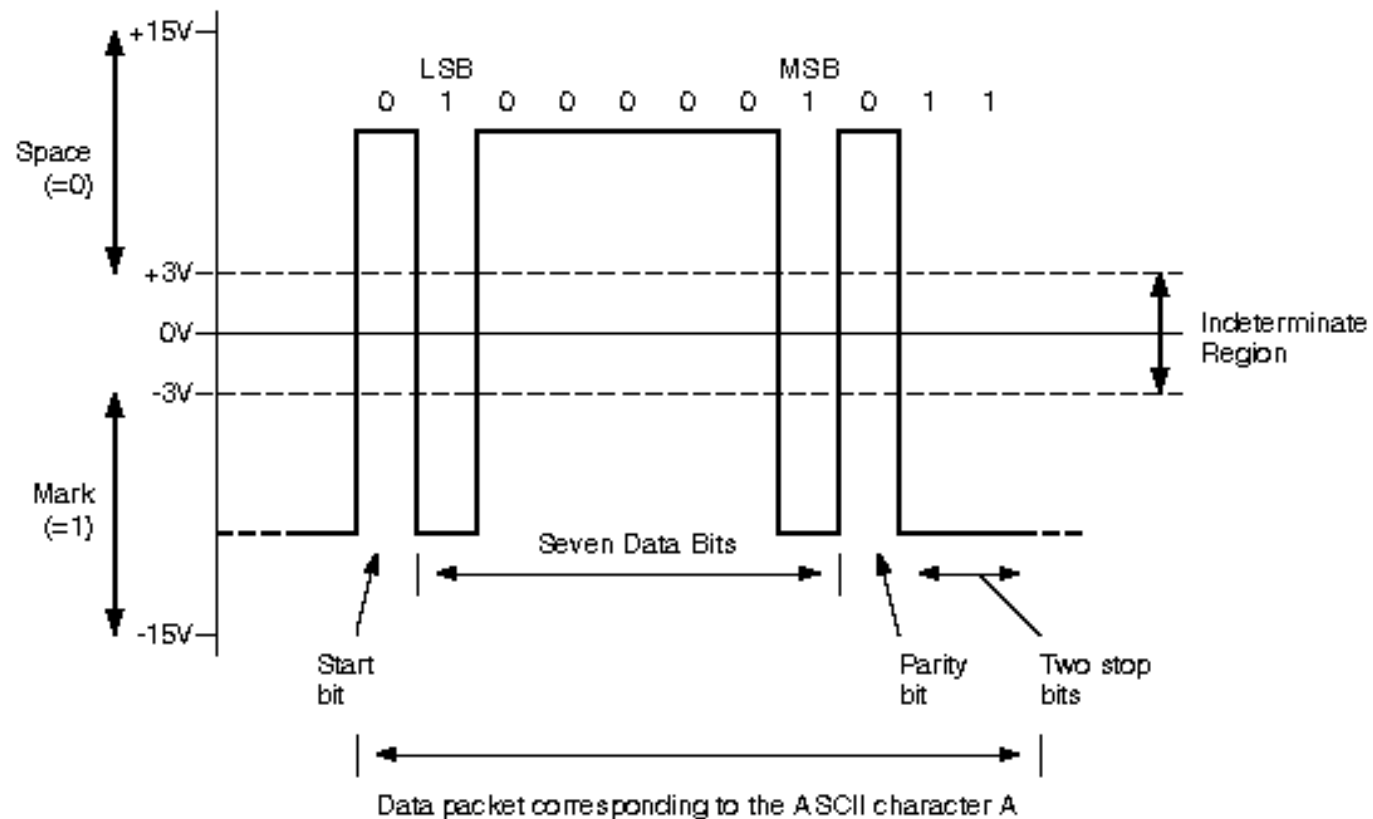


# RS-232: Electrical Characteristics

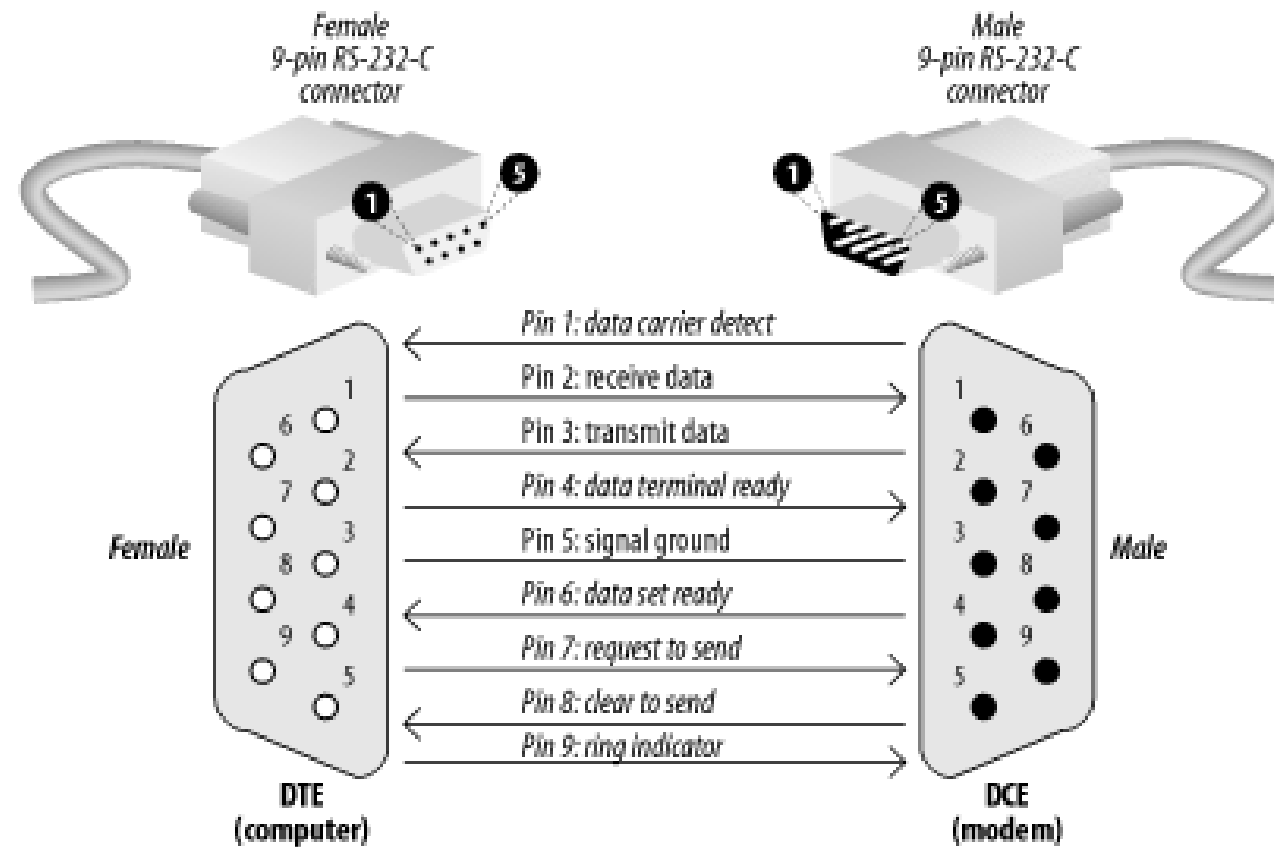
- **Line Impedances:** The impedance of wires is up to 3 ohms to 7 ohms and the maximum cable length are 15 meters, but new maximum length in terms of capacitance per unit length.
- **Slew Rate:** The rate of change of signal levels is termed as Slew Rate. Its slew rate is up to 30 V/microsecond and the maximum bitrate will be 20 kbps.
- **Baud rate:** It is used to measure the speed of transmission. It is described as the number of bits passing in one second.

For example, if the baud rate is 200 then 200 bits per Sec passed. In telephone lines, the baud rates are 14400, 28800 and 33600.

# A byte of Asynchronous Data

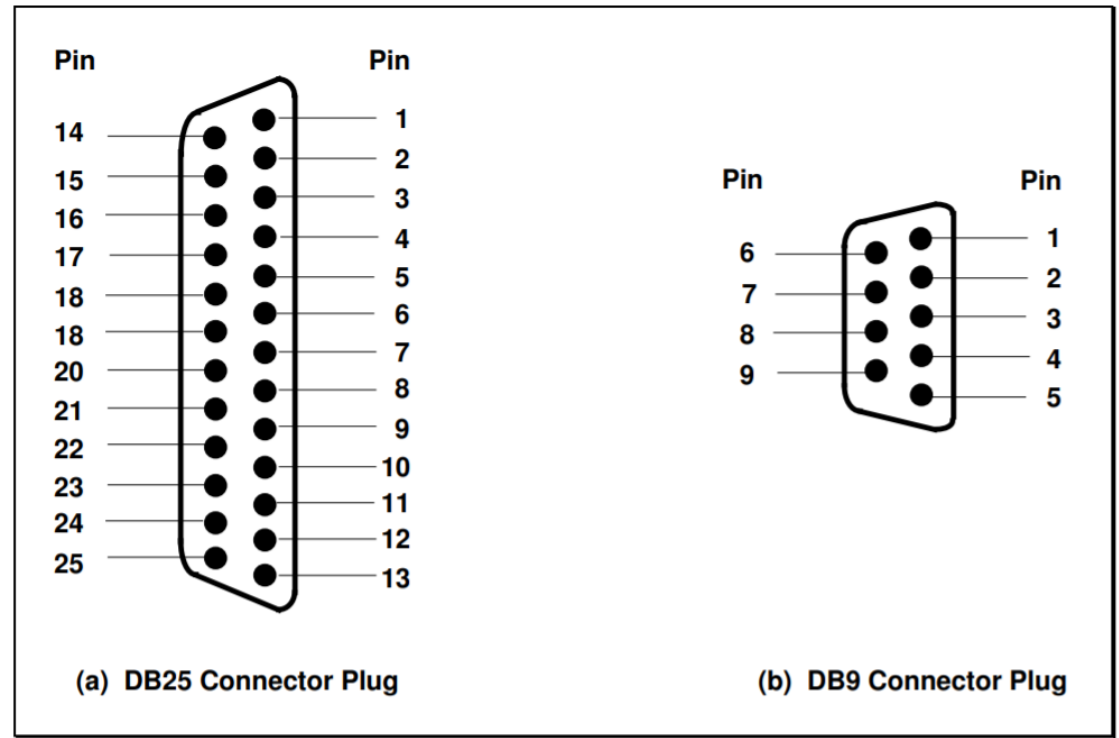


# DET DCE Interface



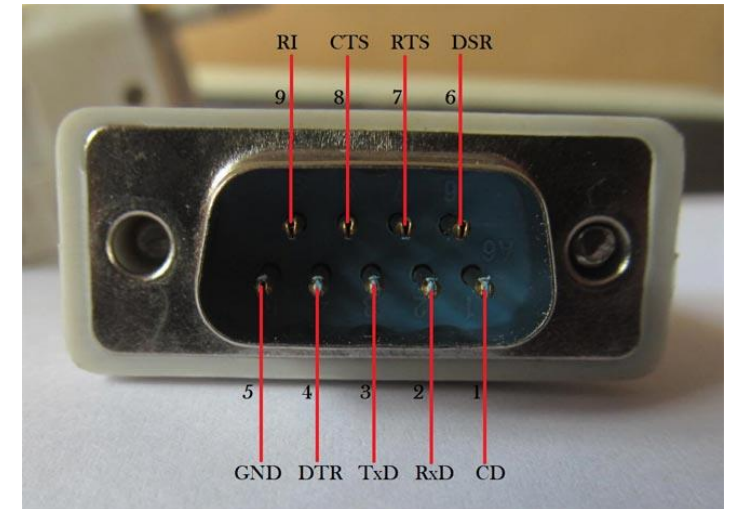
# The RS-232C Connectors

- There are two, very common connectors used for RS-232C communications
- These are "D" shaped connectors that come in either a 25 pin (called DB25) or 9 pin (called DB9), male or female form
- Modern RS-232 applications require only the 9 pins of the DB9 connector

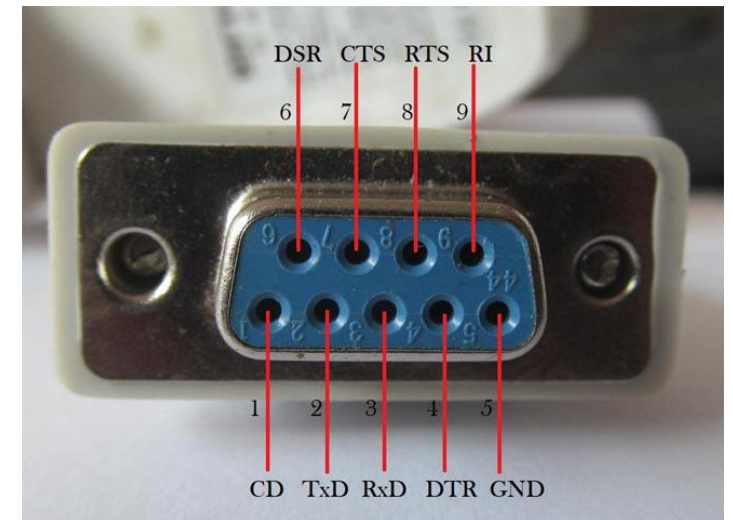


# DB-9 Connectors

- These are of two types: **Male Connector (DTE)** and **Female Connector (DCE)**. There are 5 pins on the top row and 4 pins in the bottom row. It is often called **DE-9** or **D-type connector**
- **DB-9** pin connector is used for connection between microcontrollers and connector



DB-9 Male Connector



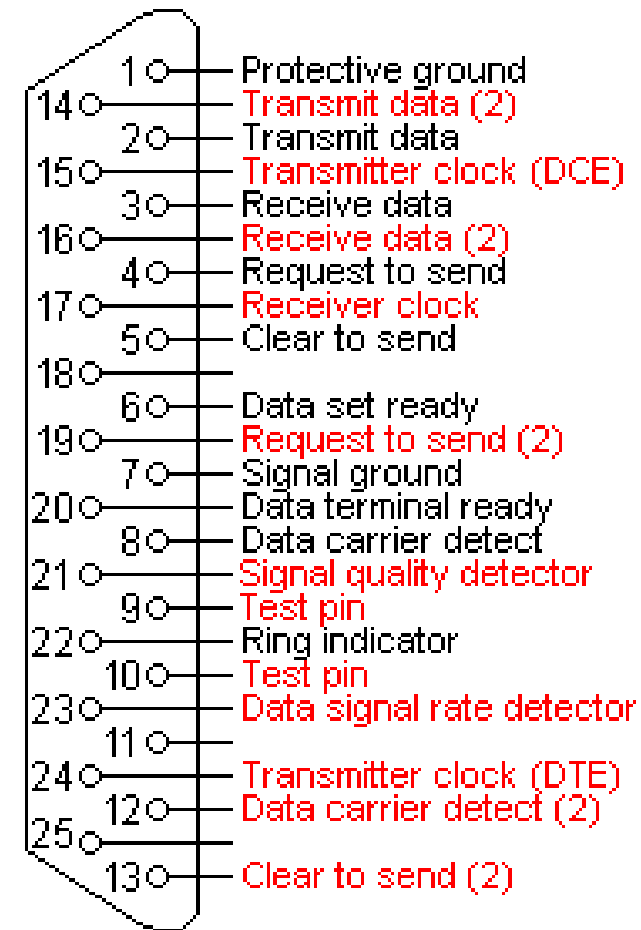
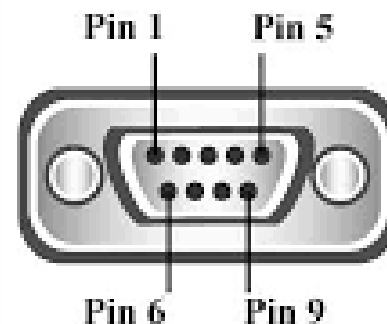
DB-9 Female Connector

# DB 9, DB 25 Connector: Pin Description

## Powered RS232

|       |             |
|-------|-------------|
| Pin 1 | DCD/12V/GND |
| Pin 2 | RXD         |
| Pin 3 | TXD         |
| Pin 4 | DTR         |
| Pin 5 | GND         |
| Pin 6 | DSR         |
| Pin 7 | RTS         |
| Pin 8 | CTS         |
| Pin 9 | RI/12V/5V   |

Powered RS232  
Pinout (9 Pin Male)



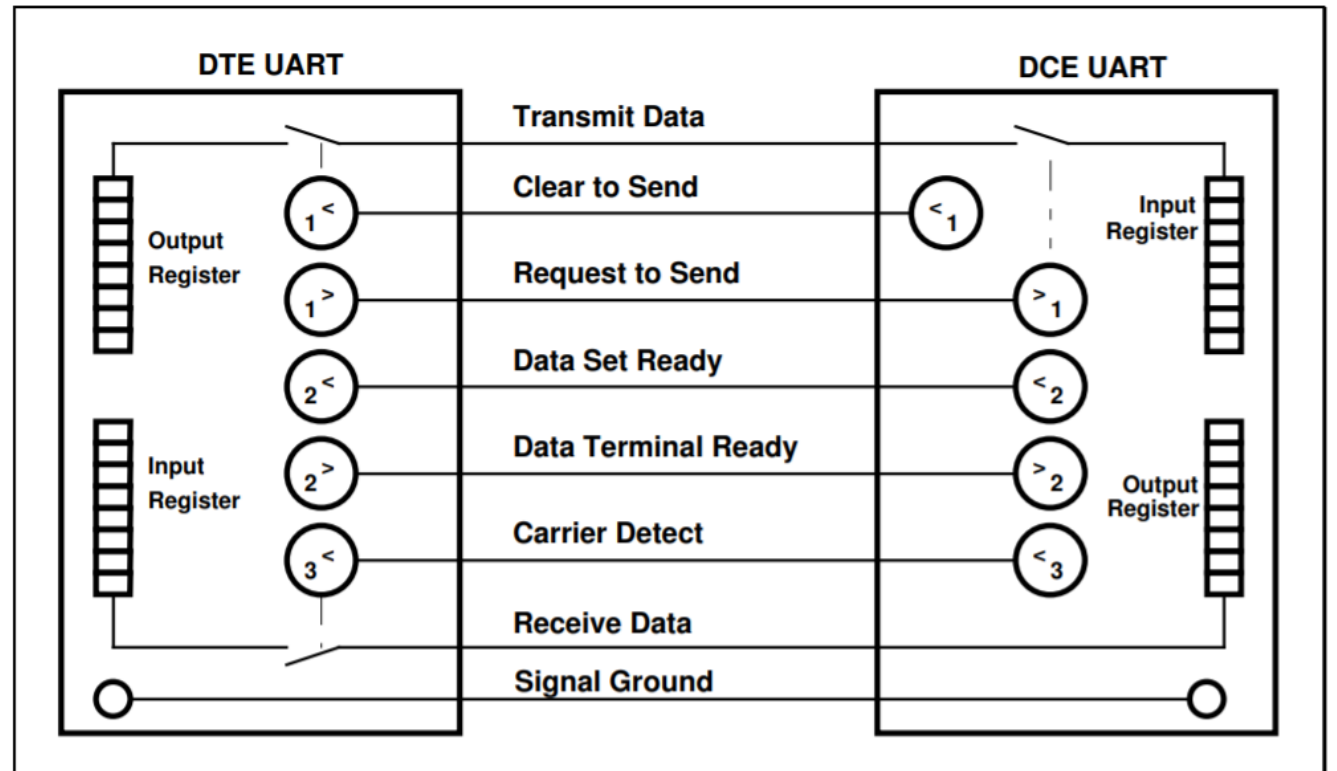
# Handshaking (Flow Control)

- Handshaking is required
  - To manage the flow of data across equipment using a serial connection
  - To prevent receiver overloading
- By using Handshaking signals, receivers will be able to tell the sending device to pause data transmission if the receiver is overloaded
- Handshake lines ensure a computer won't transmit data if the receiving computer is not ready.



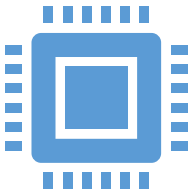
# DTE, DCE UART and Handshaking

- The RS-232 port on each of the devices is driven by a Universal Asynchronous Receiver Transmitter (UART)
- The UART performs conversion of outgoing data from parallel form to serial form and incoming data from serial to parallel form
- Some special purpose inputs and outputs are referred to as the hardware handshaking lines





# Handshaking Types



## No Handshaking

Assumes that receiver reads the data before arrival of next character



## Software Flow Control

Using control Characters:  
XON and XOFF

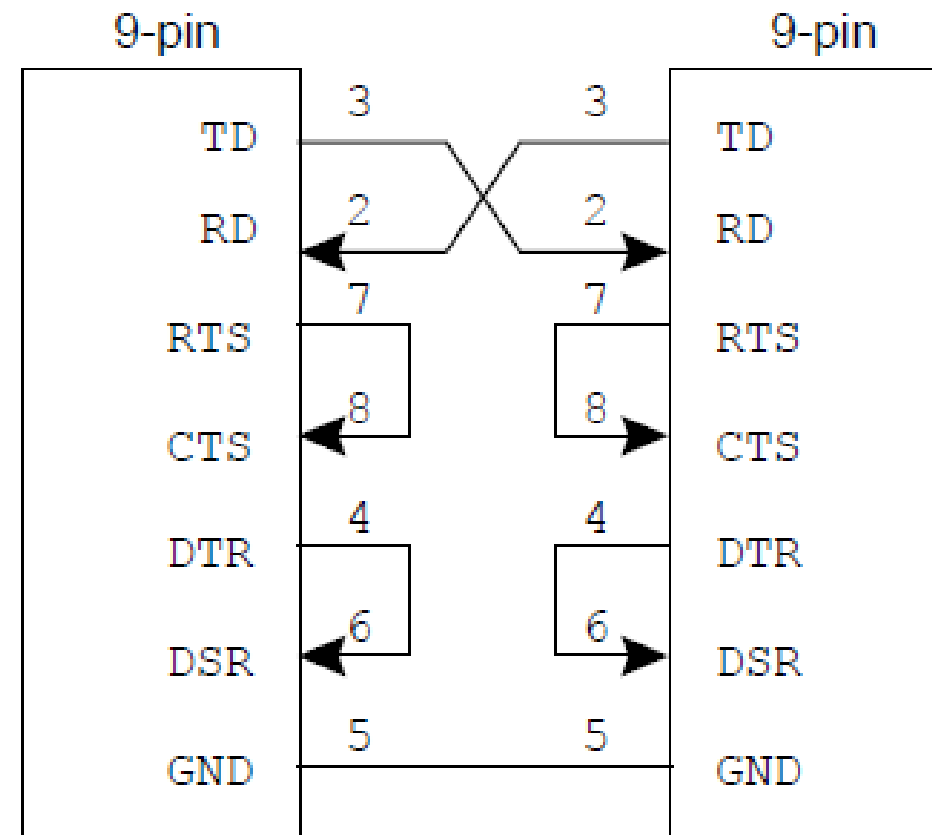


## Hardware Flow Control

Makes use of actual hardware lines, such as RTS / CTS, DTR / DSR

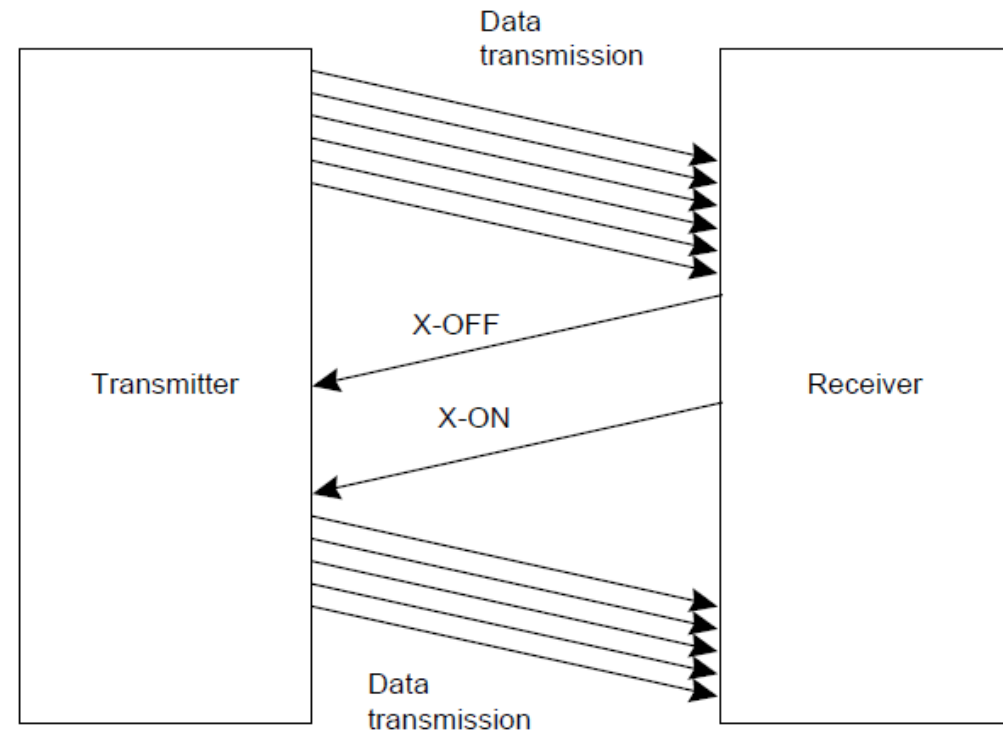
# No Handshaking

- Assumes that receiver reads the data from the receiver buffer before arrival of next character
- TxD pin of transmitter is connected to RxD pin of receiver
- It is also called **Null Modem**



# Software Handshaking

- It uses two control characters: **XON and XOFF**
- XON is decimal 17 and XOFF is decimal 19 in the ASCII
- The receiver sends these control characters to pause transmitter during communication
- If sending of characters must be postponed, the character XOFF is sent on the line, to restart the communication again XON is used
- Limitation: these two control characters can not be used in data



# Hardware Handshaking



Hardware flow control is superior compared to software flow control



Extra lines are necessary in the communication cable to carry the handshaking signals



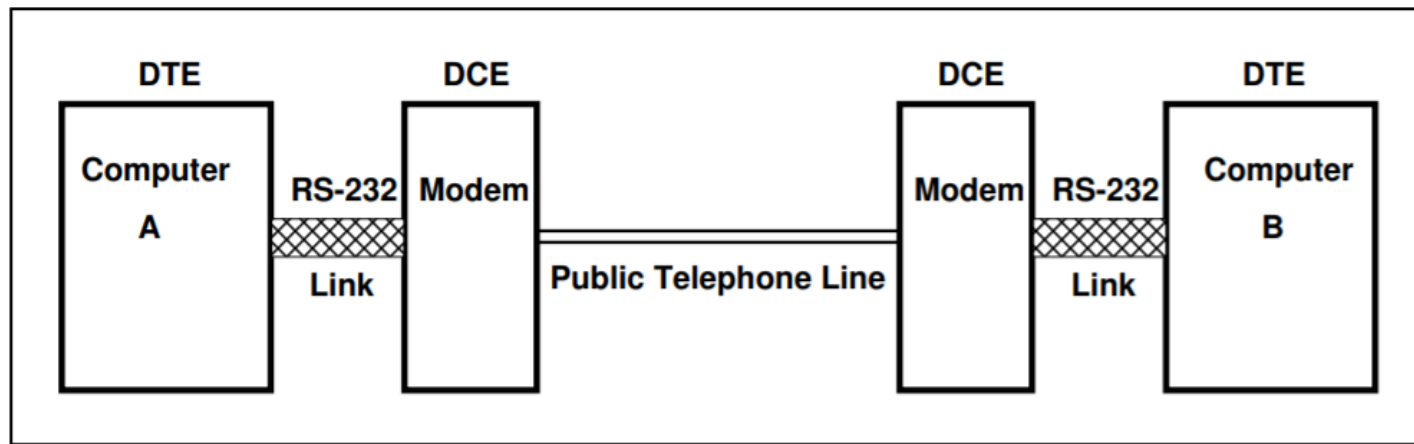
"Request to Send", "Clear to Send" and "Carrier Detect" lines are directly responsible for switching data transmission ON and OFF



The "Data Set Ready" and "Data Terminal Ready" lines are used to indicate whether the DCE and DTE are powered up and ready for communication

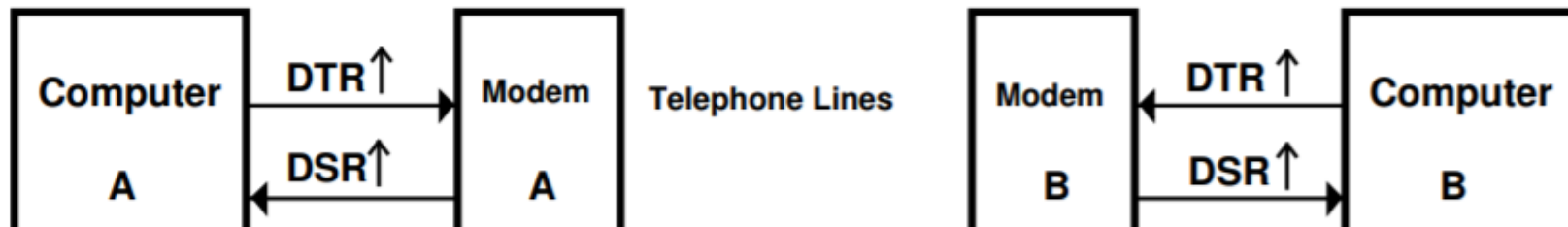
# Hardware Handshaking

- Assume that Computer A wishes to access Computer B by dialing it on the telephone network
- Once Computer A dials, Computer B should respond with an answering tone



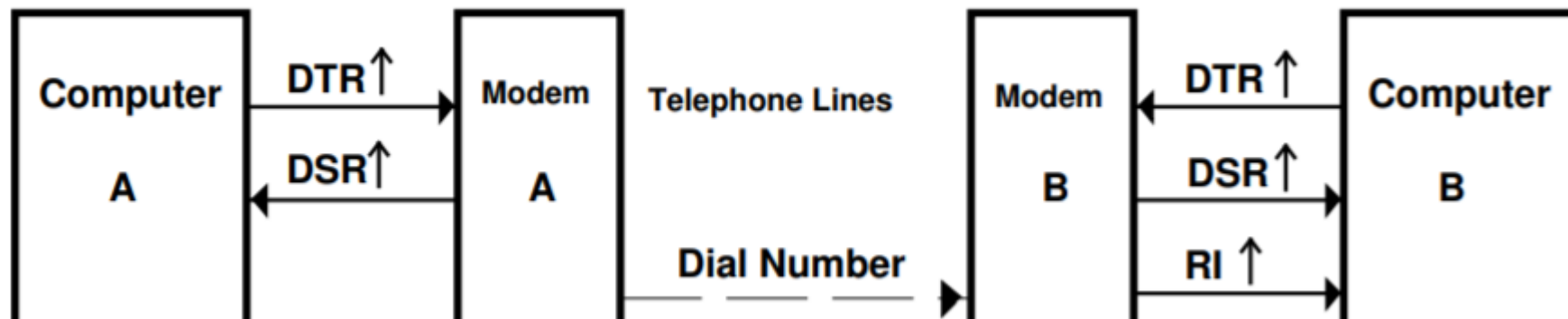
# Hardware hand-shaking sequence

- If Computers A and B are both ready for communication, then they each enable their Data Terminal Ready (DTR) lines.
- If Modems A and B are both ready for communication, then they each enable their Data Set Ready (DSR) lines.



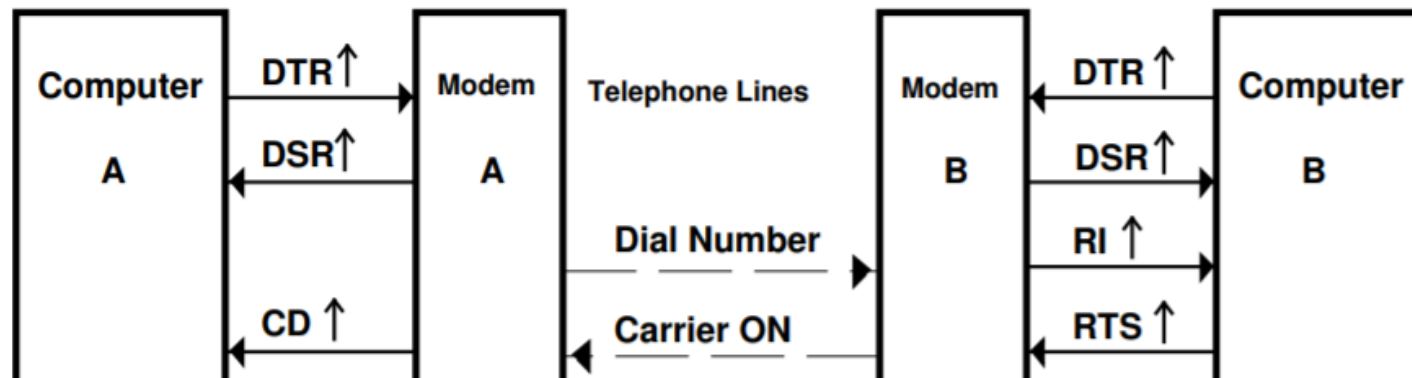
# Hardware hand-shaking sequence

- After Computer A has dialed the number, Modem B enables its Ring Indicator (RI) to tell Computer B that it has been called



# Hardware hand-shaking sequence

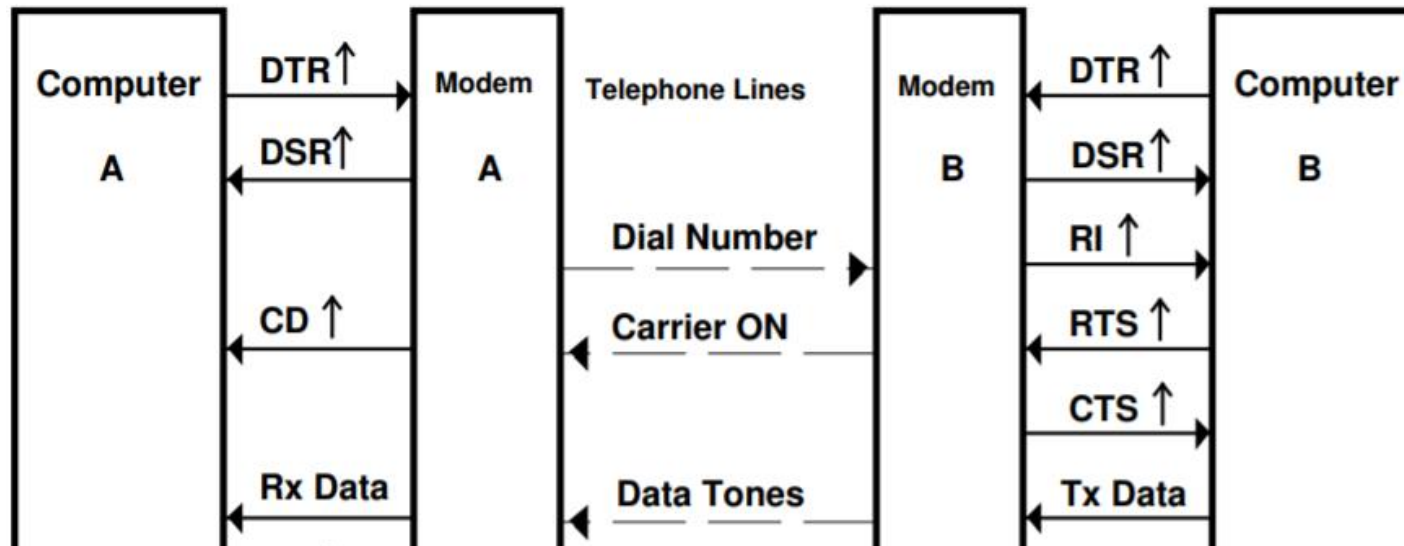
- Computer B responds to the Ring Indicator by enabling its Request to Send (RTS) line
- When Modem B receives an RTS from Computer B it transmits a carrier tone across the phone line to Modem A
- When Modem A receives the carrier signal from Modem B, it enables its Carrier Detect (CD) line. an enabled CD tells Computer A that the link is active





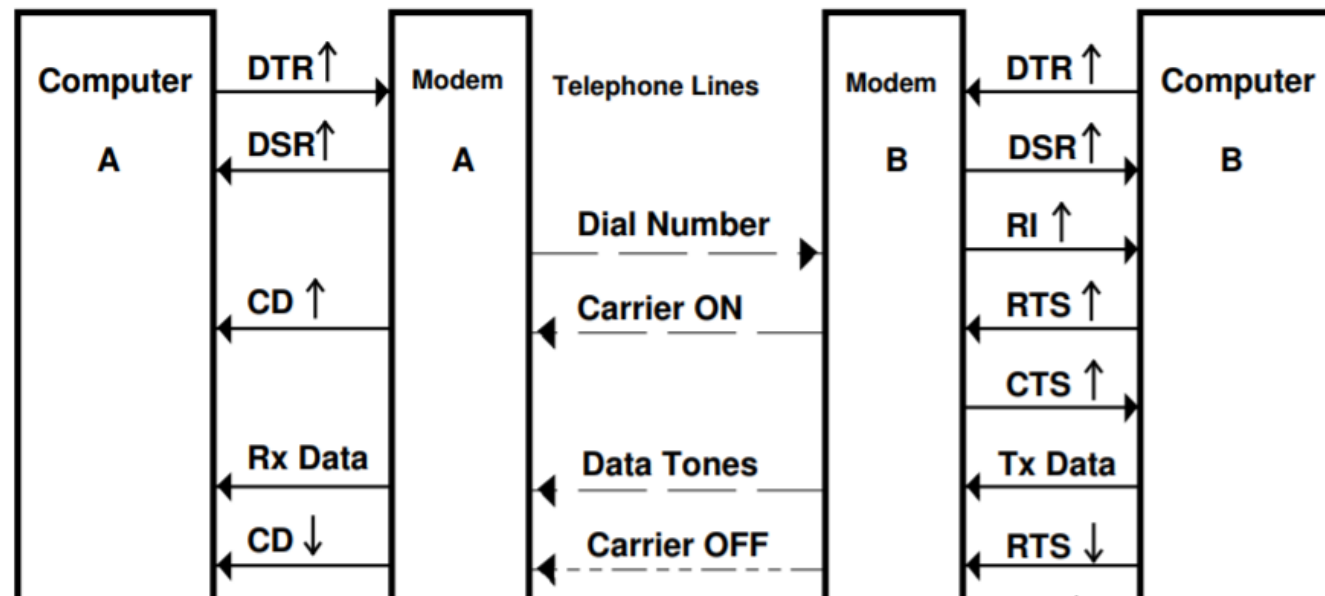
# Hardware hand-shaking sequence

- After a short delay, Modem B gives Computer B a Clear to Send (CTS) signal indicating that it should now proceed with data transmission
- Computer B transmits its message to Modem B, which then modulates the binary information into data tones



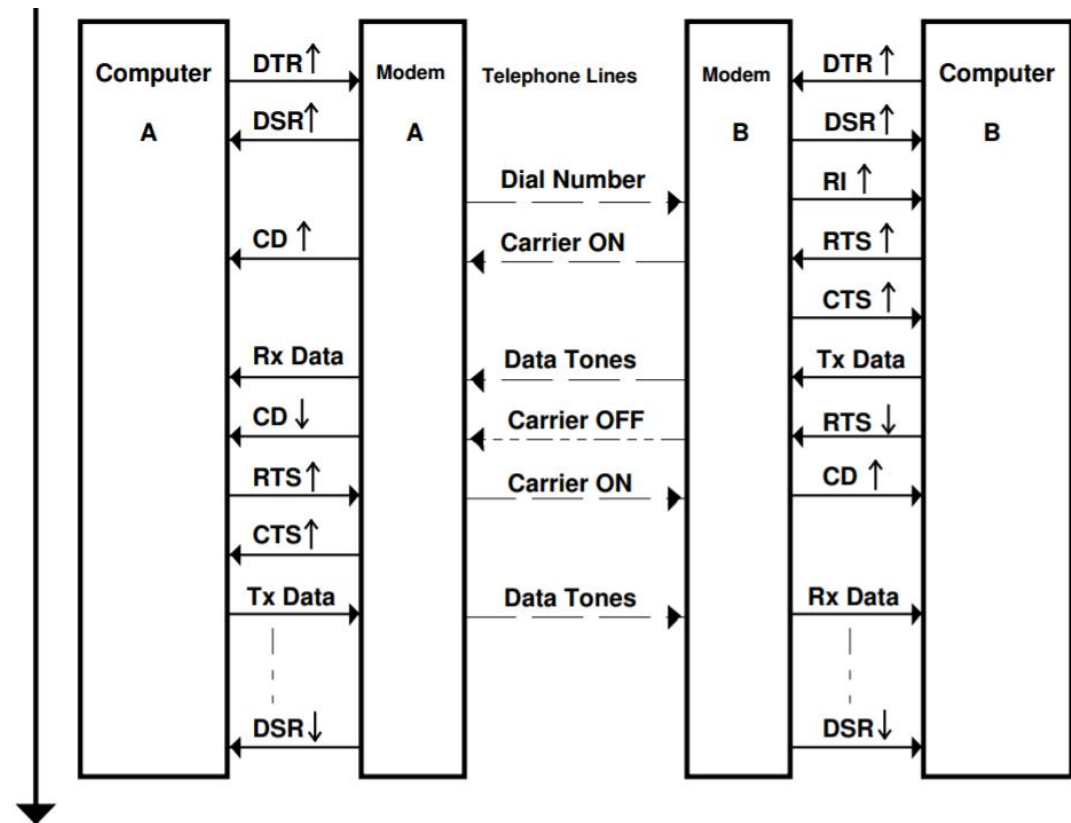
# Hardware hand-shaking sequence

- When Computer B has finished transmission it disables its RTS line
- When Modem B notes that the RTS line is disabled, it stops transmitting the carrier to Modem A
- Modem A responds by disabling its Carrier Detect



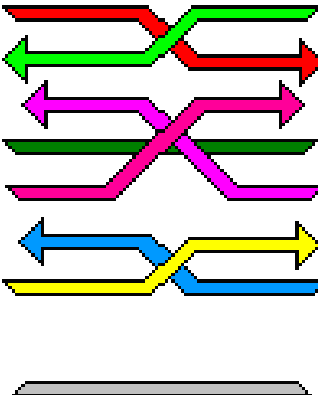
# Hardware hand-shaking sequence

- When Computer A notes that the Carrier Detect is disabled, it enables its Request to Send line in order to send a response message
- After a short delay, Modem A responds to the Request to Send signal from Computer A with a Clear to Send signal. Computer A can then transmit its data



# Modem to Modem cable

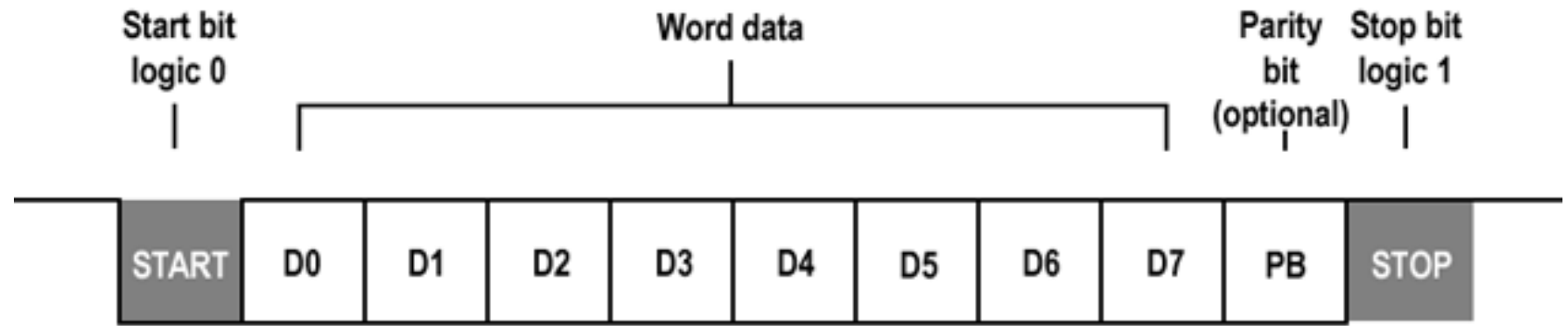
## Modem to Modem Cable - Crossover Cable DB9 to DB9

| DCE Device (Modem)             |                           | DB9                 | DCE to DCE<br>Connections                                                            | DCE Device (Modem)             |                           | DB9                 |
|--------------------------------|---------------------------|---------------------|--------------------------------------------------------------------------------------|--------------------------------|---------------------------|---------------------|
| Pin#                           | DB9                       | RS-232 Signal Names | Signal Direction                                                                     | Pin#                           | DB9                       | RS-232 Signal Names |
| #1                             | Carrier Detector (DCD)    | CD                  |  | #1                             | Carrier Detector (DCD)    | CD                  |
| #2                             | Receive Data (Rx)         | RD                  |                                                                                      | #2                             | Receive Data (Rx)         | RD                  |
| #3                             | Transmit Data (Tx)        | TD                  |                                                                                      | #3                             | Transmit Data (Tx)        | TD                  |
| #4                             | Data Terminal Ready       | DTR                 |                                                                                      | #4                             | Data Terminal Ready       | DTR                 |
| #5                             | Signal Ground/Common (SG) | GND                 |                                                                                      | #5                             | Signal Ground/Common (SG) | GND                 |
| #6                             | Data Set Ready            | DSR                 |                                                                                      | #6                             | Data Set Ready            | DSR                 |
| #7                             | Request to Send           | RTS                 |                                                                                      | #7                             | Request to Send           | RTS                 |
| #8                             | Clear to Send             | CTS                 |                                                                                      | #8                             | Clear to Send             | CTS                 |
| #9                             | Ring Indicator            | RI                  |                                                                                      | #9                             | Ring Indicator            | RI                  |
| Soldered to DB9 Metal - Shield |                           | FGND                |                                                                                      | Soldered to DB9 Metal - Shield |                           | FGND                |

Note: Signal directions reversed if devices are DTE to DTE - "Null Modem" cable for DTE devices also connects pins #1 & #6 on each side to simulate Carrier (CD) which is required by some Terminal program software.

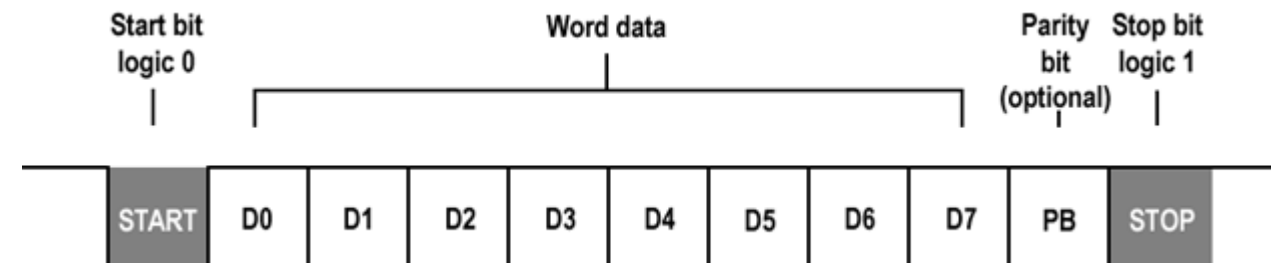
# Frame Format

- Data format:
  - 1 Start bit,
  - 8 Data bits, (LSB sent first and MSB sent last)
  - No Parity,
  - 1 Stop bit.



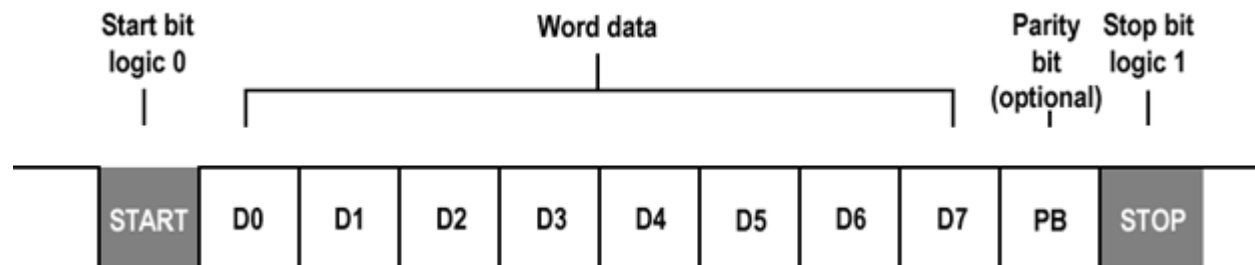
# Data Frame: Start Bit

- It is a synchronization bit added before the actual data.
- Start bit marks the beginning of the data packet.
- Usually, an idle data line i.e. when the data transmission line is not transmitting any data, it is held at a high voltage level (1)
- In order to start the data transfer, the transmitting UART pulls the data line from high voltage level to low voltage level (from 1 to 0).
- The receiving UART detects this change from high to low on the data line and begins reading the actual data.
- Usually, there is only one start bit.



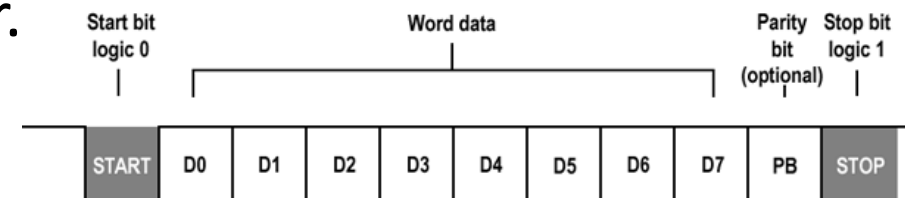
# Data Frame: Data Bits

- The actual data being transmitted from sender to receiver.
- The length of the data frame can be anywhere between 5 and 9 (9 bits if parity is not used and only 8 bits if parity is used).
- Usually, the LSB is the first bit of data to be transmitted



# Data Frame: Parity Bit

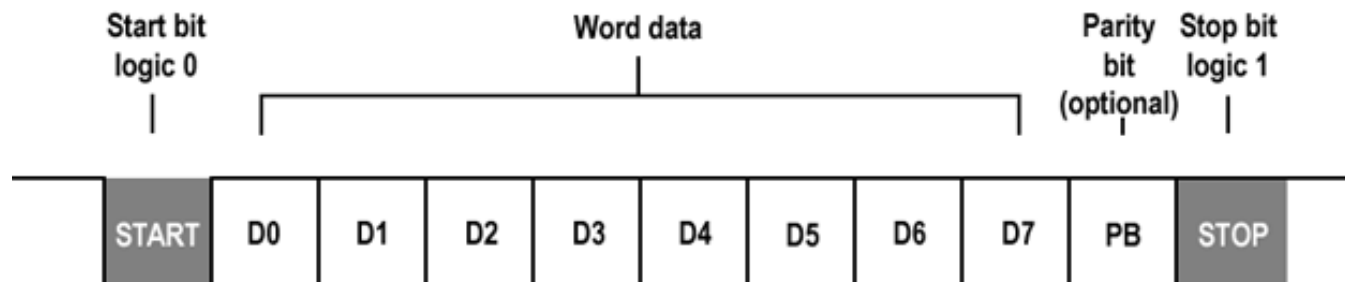
- It allows the receiver to check whether the received data is correct or not.
- A parity bit is added to transmitted data to make the number of 1s sent either even (even parity) or odd (odd parity).
- A single parity bit can only detect an odd number of errors, i.e., 1, 3, 5, and so on.
- If there is an even number of bits in error then the parity bit will be correct and no error will be detected.
- This type of error coding is not normally used on its own where there is the possibility of several bits being in error.



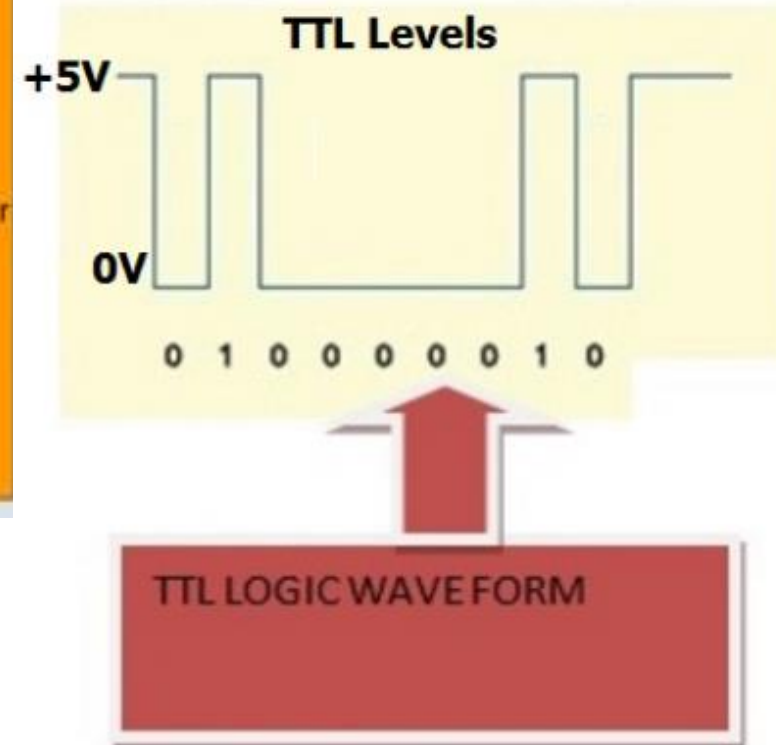
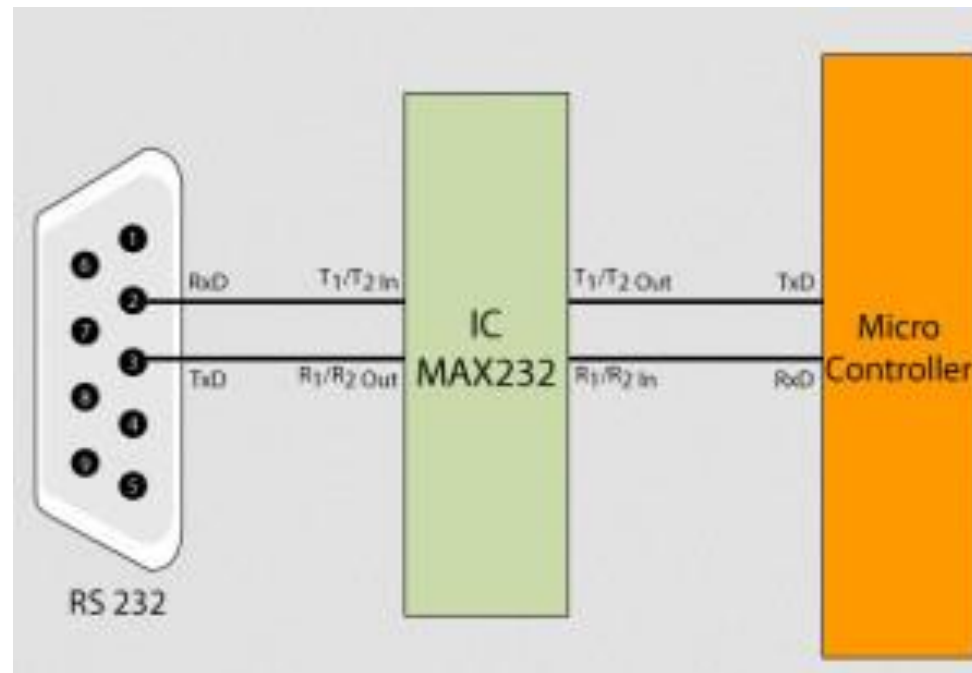
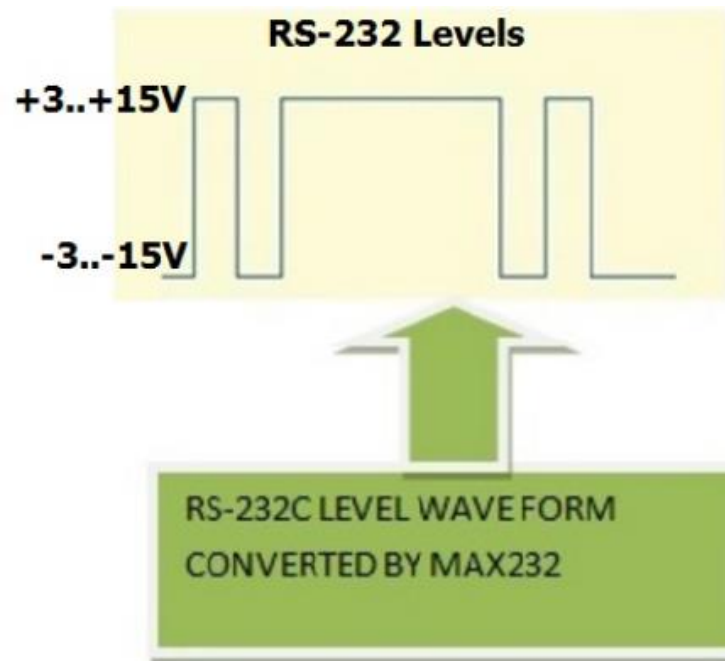


# Data Frame: Stop Bit

- It marks the end of the data packet
- It is usually two bits long (Can be set to 1, 1.5 or 2 bits) but often only one bit is used
- In order to end the transmission, the UART maintains the data line at high voltage (1)



# Interfacing to Microcontroller



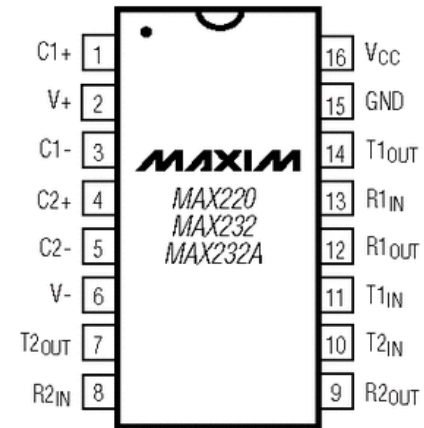
# Necessity of MAX 232 Transceiver

- Microcontrollers have a build in UART which is compatible with TTL level
- Standard PC(Personal Computers) serial port (COM Port) works on RS-232 level
- To transfer data from microcontroller to PC, it is necessary to convert data from TTL to RS-232 level and to send data from PC to micro-controller, it is necessary to convert data from Rs-232 to TTL
- MAX232 converts TTL logic level signal into its equivalent RS-232 level signal and Rs-232 level to its equivalent TTL level signal



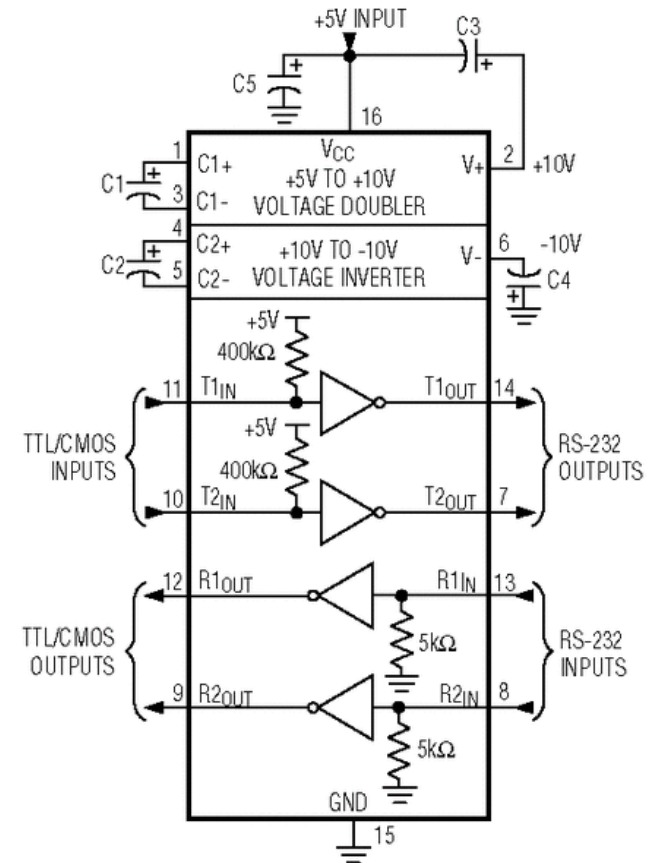
# MAX 232 Pinout

Source From [pilgoo.blogspot.com](http://pilgoo.blogspot.com)



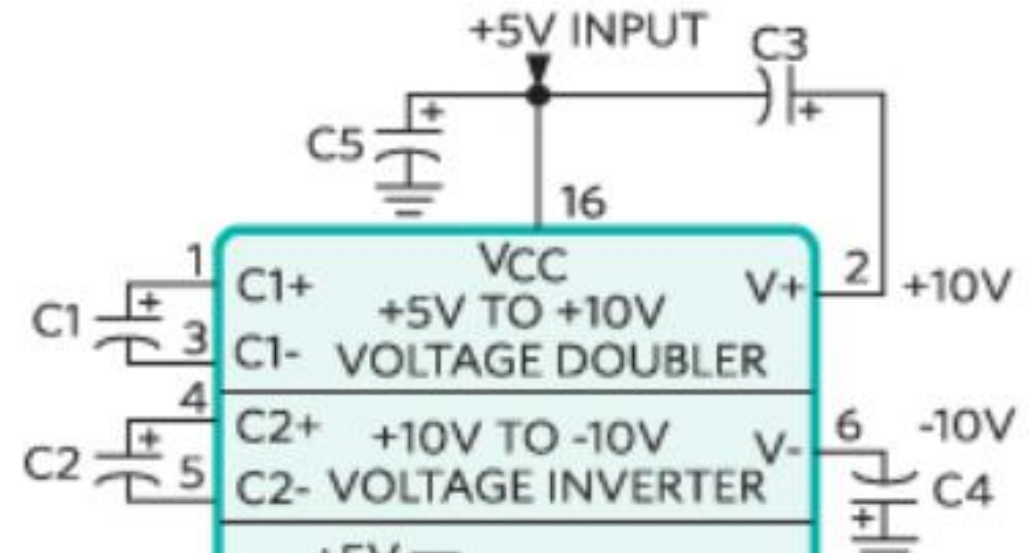
DIP/SO

| CAPACITANCE ( $\mu\text{F}$ ) |       |      |      |      |      |
|-------------------------------|-------|------|------|------|------|
| DEVICE                        | C1    | C2   | C3   | C4   | C5   |
| MAX220                        | 0.047 | 0.33 | 0.33 | 0.33 | 0.33 |
| MAX232                        | 1.0   | 1.0  | 1.0  | 1.0  | 1.0  |
| MAX232A                       | 0.1   | 0.1  | 0.1  | 0.1  | 0.1  |



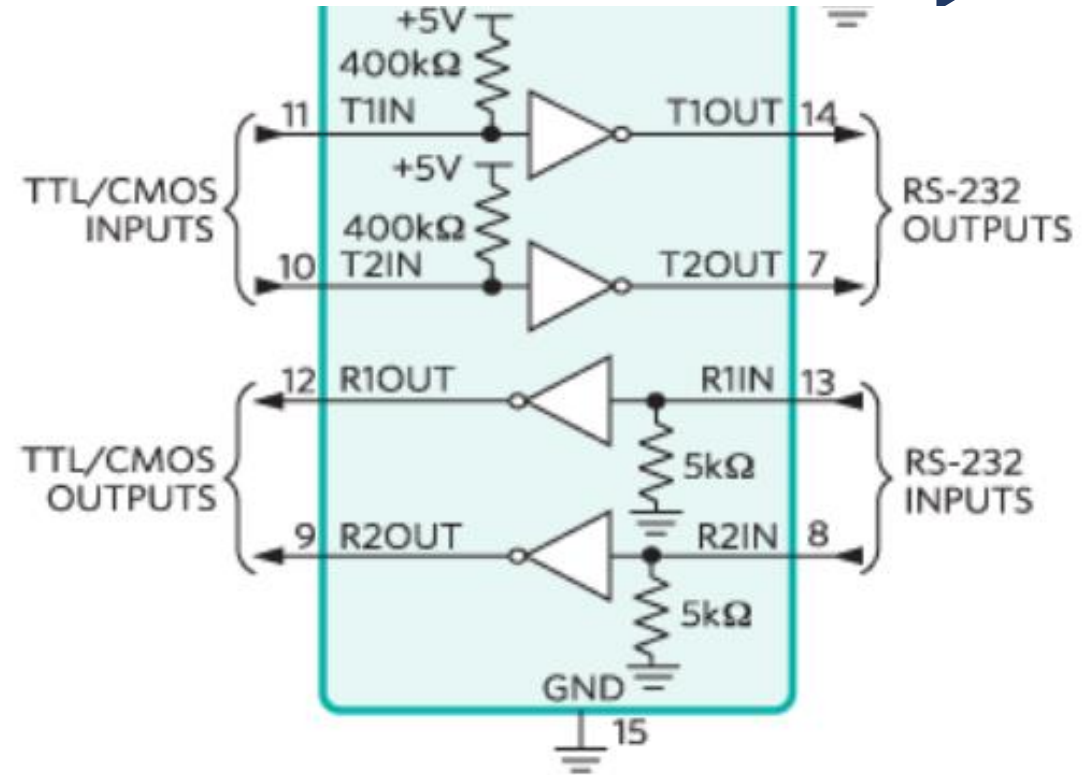
# MAX 232 Pin Configuration

- It requires four external capacitors. Capacitors can range between 8  $\mu\text{F}$  to 10  $\mu\text{F}$  and are of up to 16 volts
- The doubler doubles the voltage level to produce +10 V
- The inverter produces the negative voltage supply of -10 V
- C1 and C3 are part of the voltage doubler circuit
- capacitors C2 and C4 are part of the voltage inverter circuit



# MAX 232 Pin Configuration

- **RS232 Driver:** Two drivers interface standard logic level to RS232 levels. Internal pull up resistors on TIN inputs ensures a high input when the line is high impedance.
- **RS232 Receiver:** Two receivers interface RS232 levels to standard logic levels. An open input will result in a high output on ROUT.



# MAX 232 Driver Receivers

**Function Table Each Driver**

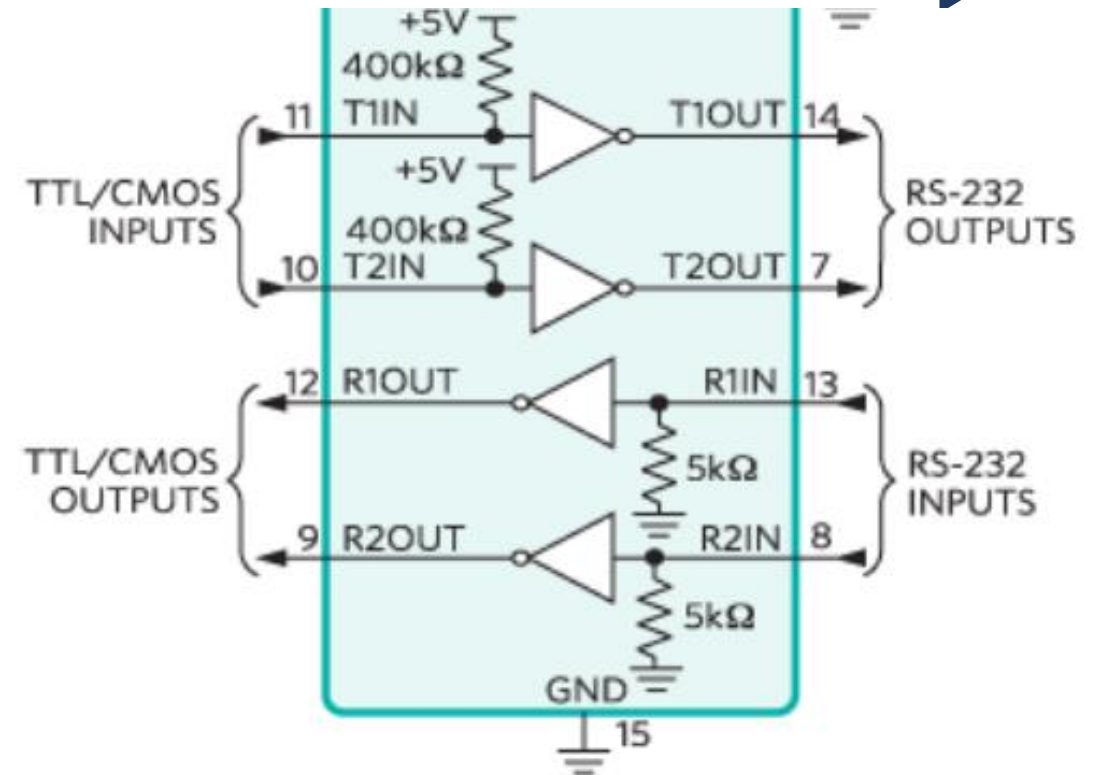
| INPUT<br>T <sub>IN</sub> | OUTPUT<br>T <sub>OUT</sub> |
|--------------------------|----------------------------|
| L                        | H                          |
| H                        | L                          |

H = high level, L = low level, X = irrelevant, Z = high impedance

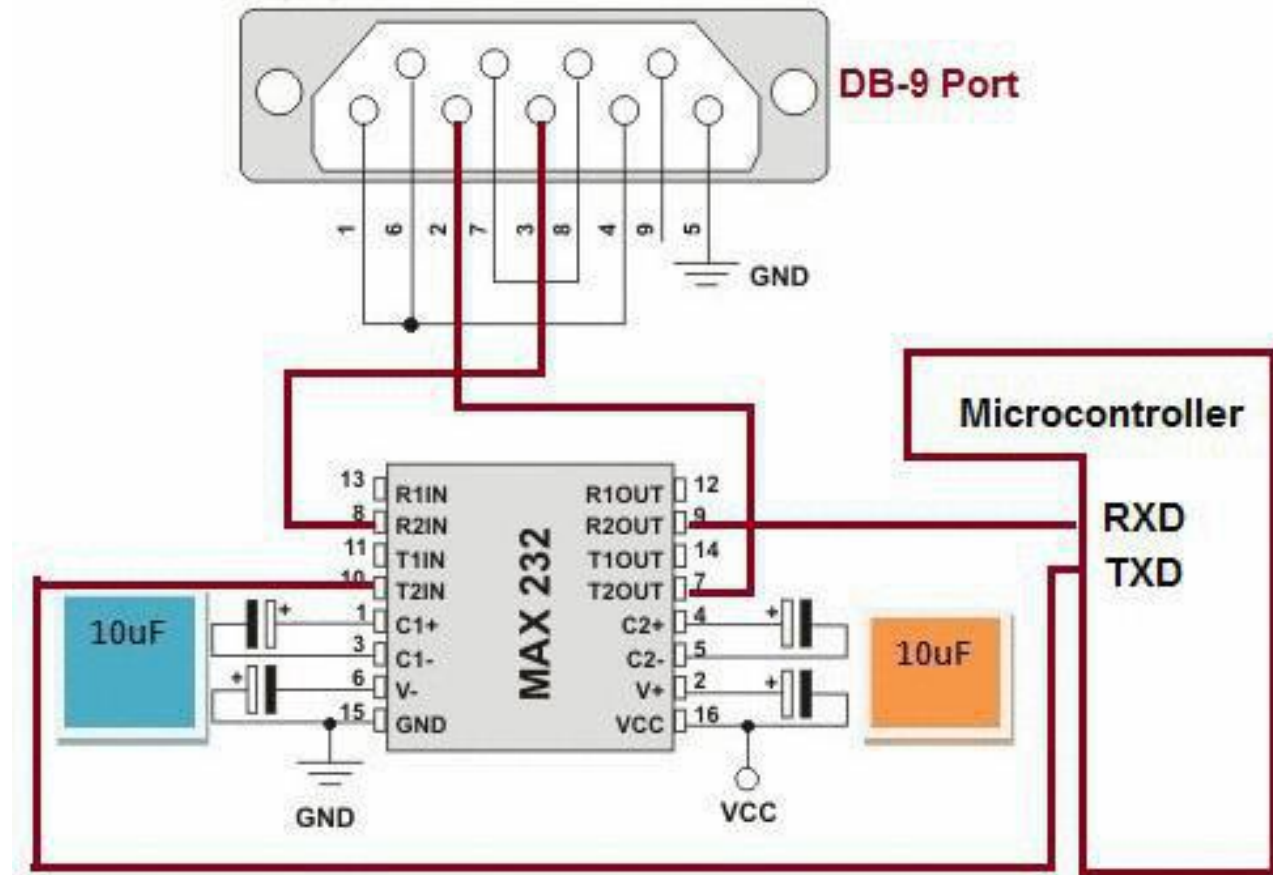
**Function Table Each Receiver**

| INPUTS<br>R <sub>IN</sub> | OUTPUT<br>R <sub>OUT</sub> |
|---------------------------|----------------------------|
| L                         | H                          |
| H                         | L                          |
| Open                      | H                          |

H = high level, L = low level, X = irrelevant, Z = high impedance (off),



# Interfacing to Microcontroller



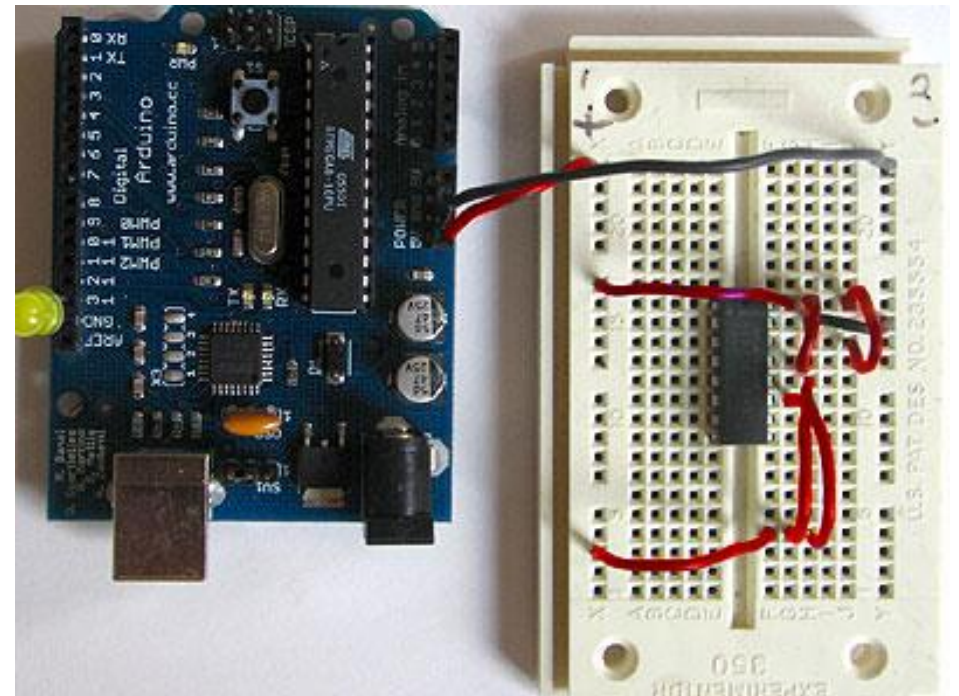


# Interfacing RS232 to Arduino

- Tutorial to communicate with a computer using a MAX3323 single channel RS-232 driver/receiver and a software serial connection on the Arduino
- Components required:
  - Computer with a terminal program installed (i.e. HyperTerminal)
  - Breadboard
  - MAX3323 chip (or similar)
  - Four 1 $\mu$ F capacitors
  - Wires
  - Arduino Microcontroller Module
  - Light emitting Diode (LED), for debugging

# Prepare the Breadboard

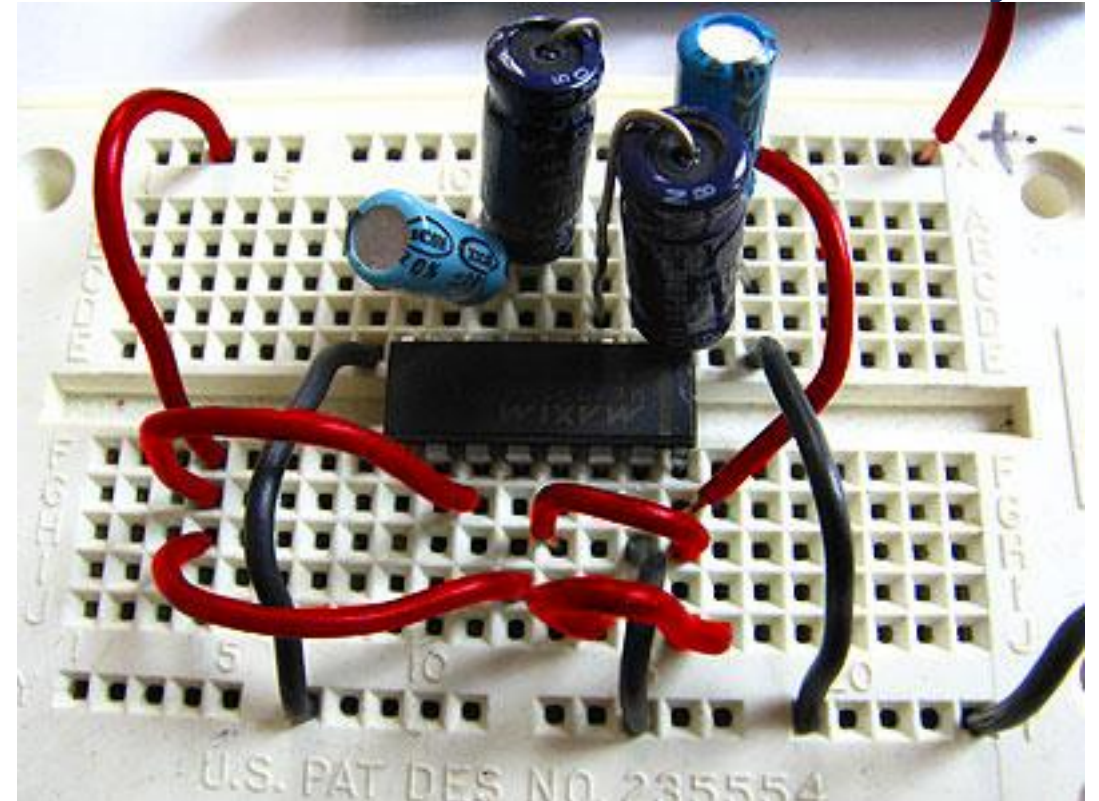
- Insert the MAX3323 chip in the breadboard.
- Connect 5V power and ground from the breadboard to 5V power and ground from the microcontroller.
- Connect pin 15 on the MAX3323 chip to ground and pins 16 and 14 - 11 to 5V.
- Connect LED between pin 13 and ground.



*+5v wires are red, GND wires are black*

# Prepare the Breadboard

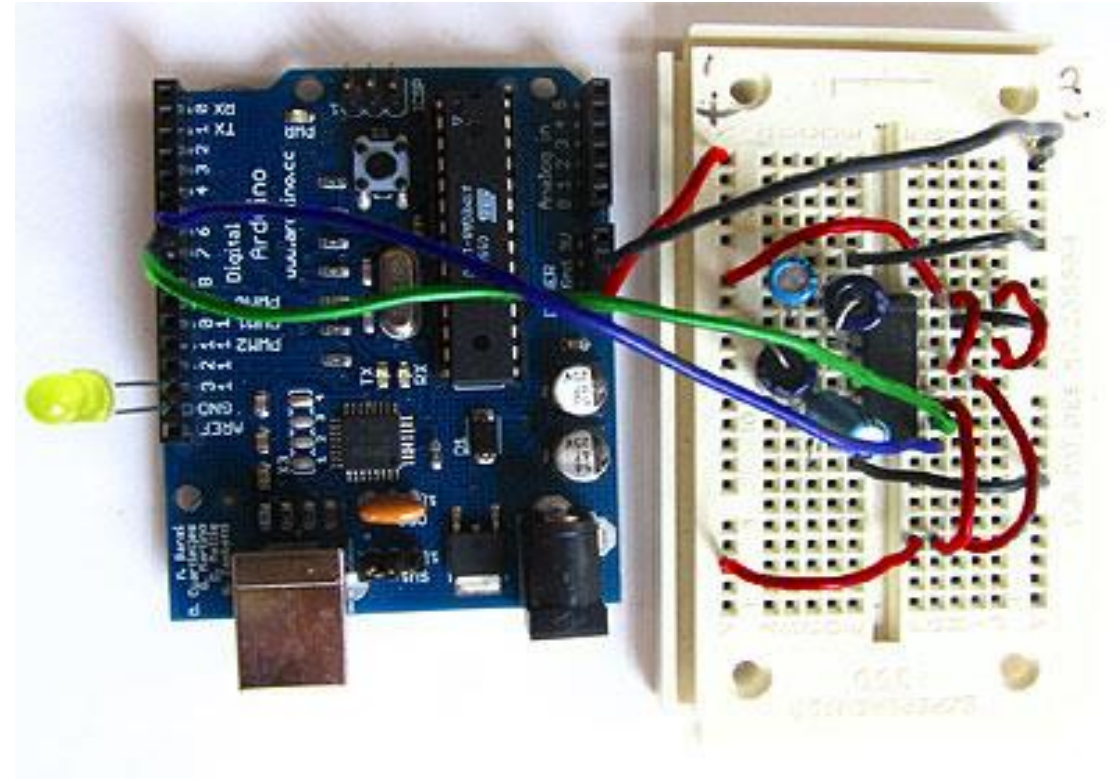
- Connect a  $1\mu\text{F}$  capacitor across pins 1 and 3, another across pins 4 and 5, another between pin 2 and ground, and the last between pin 6 and ground.
- If electrolytic capacitors are used, make sure the negative pins connect to the negative sides (pins 3 and 5 and ground).



*+5v wires are red, GND wires are black*

# Prepare the Breadboard

- Consider Arduino pin 6 for receiving and pin 7 for transmitting the data.
- Connect TX pin (7) to MAX3323 pin 10 (T1IN) and RX pin (6) to MAX3323 pin 9 (R1OUT).

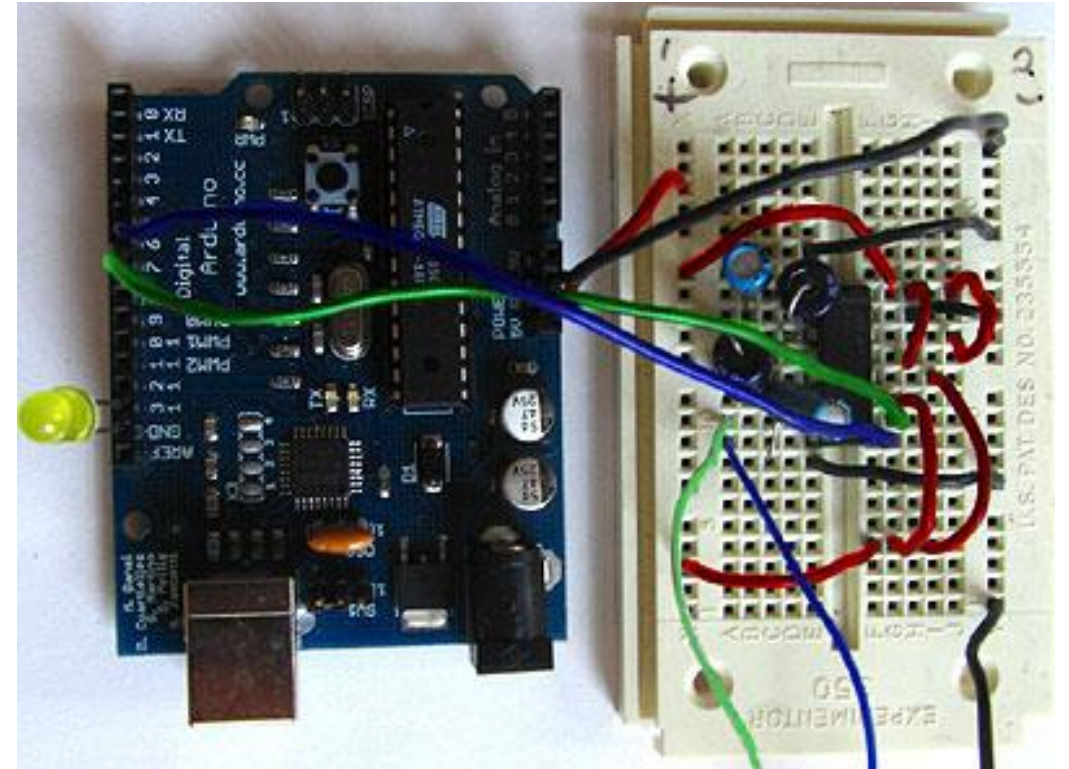


*TX wire Green, RX wire Blue, +5v wires are red, GND wires are black*



# Prepare the Breadboard

- Connect the TX line from computer to pin 8 (R1IN) on the MAX3323 and the RX line to pin 7 (T1OUT).
- Connect the ground line from computer to ground on the breadboard.



*TX wire Green, RX wire Blue, +5v wires are red, GND wires are black*

# Program the Arduino

Functions available:

- `SWread();`  
Returns a byte long integer value from the software serial connection  
Example:  
`byte RXval;`  
`RXval = SWread();`
- `SWprint();`  
Sends a byte long integer value out the software serial connection  
Example:  
`byte TXval = 'h';`  
`byte TXval2 = 126;`  
`SWprint(TXval);`  
`SWprint(TXval2);`

# Program the Arduino

Definitions Needed:

`#define bit9600Delay 84`

`#define halfBit9600Delay 42`

`#define bit4800Delay 188`

`#define halfBit4800Delay 94`

```
#include <ctype.h>
```

```
#define bit9600Delay 100
```

```
#define halfBit9600Delay 50
```

```
#define bit4800Delay 188
```

```
#define halfBit4800Delay 94
```

These definitions set the delays required for 9600 baud and 4800 baud software serial operation.

# Program the Arduino

## **Problem Statement:**

**Write a program to wait for a character to arrive in the serial receiving port and then write it back out in uppercase out the transmit port.**

```
#include <ctype.h>
```

First, include the file `ctype.h` in the application.

This gives us access to the `toupper()` function from the Character

This library is part of the Arduino install, no need to do anything other than type the `#include` line in order to use it.



# Program the Arduino

## **Code to enable serial data communication:**

This program will simply wait for a character to arrive in the serial receiving port and then write it back out in uppercase out the transmit port.

```
#include <ctype.h>
```

First, include the file ctype.h in the application.

This gives us access to the toupper() function from the Character

This library is part of the Arduino install, no need to do anything other than type the #include line in order to use it.

# Program the Arduino

- Set the transmit (tx) and receive (rx) pins.
- Change the pin numbers to suit the application.
- Allocate a variable to store the received data in, SWval.

```
#include <ctype.h>

#define bit9600Delay 100
#define halfBit9600Delay 50
#define bit4800Delay 188
#define halfBit4800Delay 94

byte rx = 6;
byte tx = 7;
byte SWval;
```

# Program the Arduino

- Initialize the lines, turn on the debugging LED and print a debugging message to confirm all is working as planned.
- Pass individual characters or numbers to the SWprint function.

```
void setup() {  
  pinMode(rx, INPUT);  
  pinMode(tx, OUTPUT);  
  digitalWrite(tx, HIGH);  
  delay(2);  
  digitalWrite(13, HIGH); //turn on debugging LED  
  SWprint('h'); //debugging hello  
  SWprint('i');  
  SWprint(10); //carriage return  
}
```

# Program the Arduino

- Transmit line is pulled low to signal a start bit.
- Then iterate through a bit mask and flip the output pin high or low 8 times for the 8 bits in the value to be transmitted.
- Finally, pull the line high again to signal a stop bit.
- For each bit transmitted, hold the line high or low for the specified delay. In this example, a 9600 baudrate is used.

```
void SWprint(int data)
{
    byte mask;
    //startbit
    digitalWrite(tx, LOW);
    delayMicroseconds(bit9600Delay);
    for (mask = 0x01; mask > 0; mask <= 1) {
        if (data & mask) { // choose bit
            digitalWrite(tx, HIGH); // send 1
        }
        else {
            digitalWrite(tx, LOW); // send 0
        }
        delayMicroseconds(bit9600Delay);
    }
    //stop bit
    digitalWrite(tx, HIGH);
    delayMicroseconds(bit9600Delay);
}
```

# Program the Arduino

- Wait for a byte to arrive on the receive pin and then return it to the allocated variable.
- First, wait for the receive line to be pulled low. Check after a half bit delay to make sure the line is still low.
- Then, iterate through a bit mask and shift 1s or 0s into our output byte based on what is received.
- Finally, allow a pause for the stop bit and then return the value.

```
int SWread()
{
    byte val = 0;
    while (digitalRead(rx));
    //wait for start bit
    if (digitalRead(rx) == LOW) {
        delayMicroseconds(halfBit9600Delay);
        for (int offset = 0; offset < 8; offset++) {
            delayMicroseconds(bit9600Delay);
            val |= digitalRead(rx) << offset;
        }
        //wait for stop bit + extra
        delayMicroseconds(bit9600Delay);
        delayMicroseconds(bit9600Delay);
        return val;
    }
}
```

# Program the Arduino

- Finally, implement the main program loop.
- In this program, wait for characters to arrive, change them to uppercase and send them back.

```
void loop()
{
    SWval = SWread();
    SWprint(toupper(SWval));
}
```