Serial Communication

Serial port on the 8051

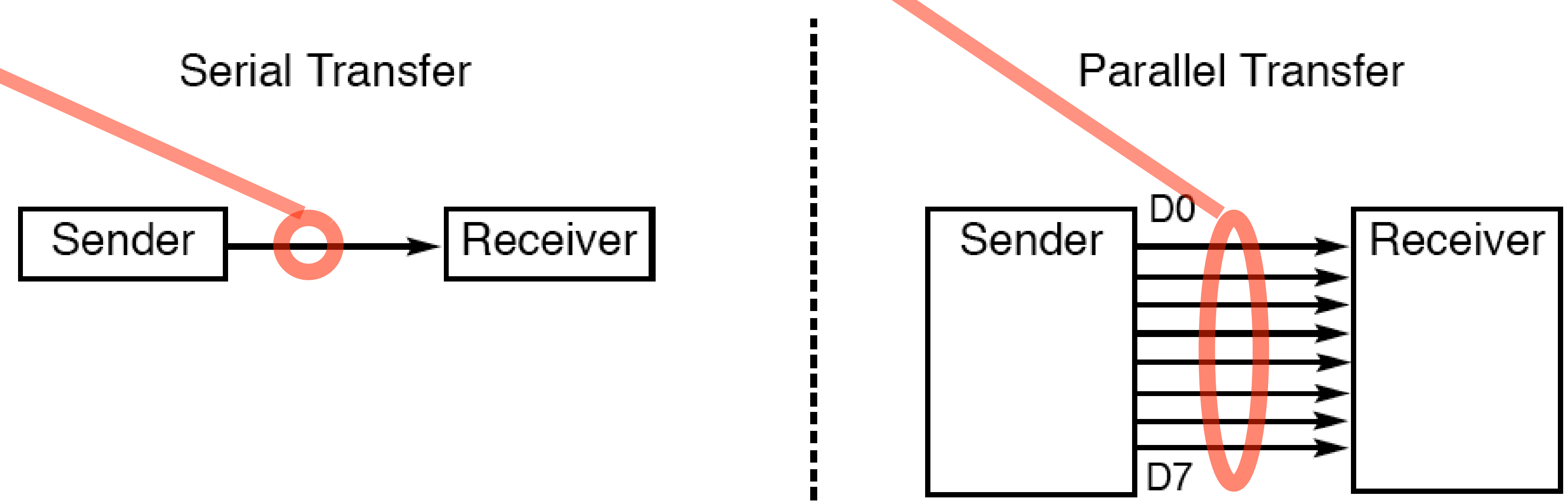
- Parallel vs. Serial
  - simplex, half-duplex, vs. full-duplex
  - Synchronous vs. asynchronous
- RS232 vs. UART
- SBUF, SCON
- See also
  - http://www.edsim51.com/8051Notes/8051/serial.html

Serial communication

- "Data-serial" communication
  - Data is serialized into single bit at a time
  - as opposed to "data-parallel" (mult. bits)
- Does not mean single-wire
  - Could be "differential pair" (voltage)
  - May have other control & power signals e.g., clock, flow control, power, ground, ..

Serial vs. Parallel

- Single bit in a given direction
  could actually use multiple wires,
  plus many other control signals!!!
- Could be 4-bit, 8-bit, 16-bit, ... plus all other control/clock signals (no strict definition)

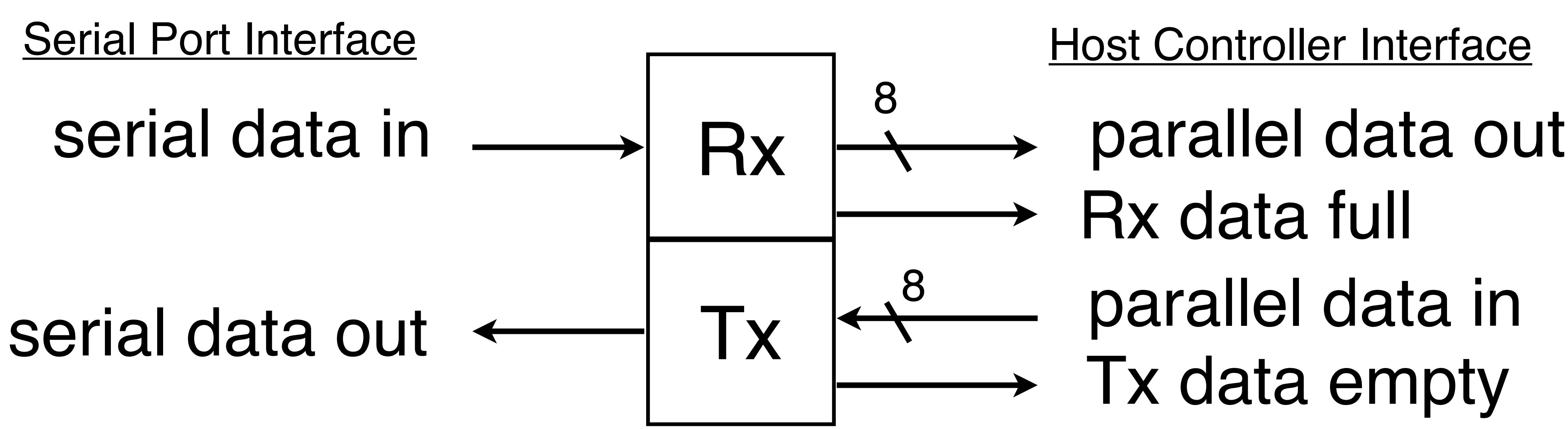


UART

- Universal Asynchronous Receiver Transmitter
  - Also known as ACIAs (Asynchronous Communication Interface Adapters)
- Points
  - Serial: data shifted in/out serially
  - Asynchronous: no clock; embedded in data
  - Both sides must run at the same baud rate

UART - functional diagram

- Rx and Tx are independent controllers
  - either one may be missing
  - Sender/receiver need to have Rx Tx lines crossed



Simplex, Half-duplex, full-duplex

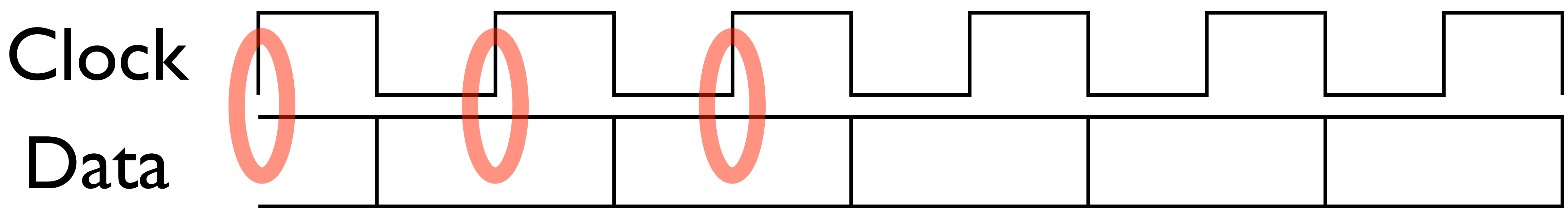
- Simplex: **one-way** data transfer
- Duplex: **two-way** data transfer
  - Half-duplex: one-way **at a time** (may share same data wire)
  - Full-duplex: two-way **simultaneously** (might need one wire each way)

Synchronous vs. Asynchronous

- These words are totally overloaded!! ("overloaded" => multiple meanings)
  - They mean opposite things in hw vs. sw
  - It is unrelated to block of chars vs. byte at a time
- Fundamental question: synchronous to what event?

Synchronous hardware

- Hardware: needs a "clock"
  - clock itself can pulse at different rates
  - data bit is qualified by the clock (edge)
- Example protocols: SPI, I2C, GPIB, ...



Software communication/call

- In software,
  - Synchronous means "blocking call"
    Asynchronous means "non-blocking"
- Blocking:
  wait for a call to finish before continuing
  - e.g., send() or receive() calls
    synchronous => don't return until finish
- UART: asynchronous comm (no clock across nodes) + locally synchronous hw

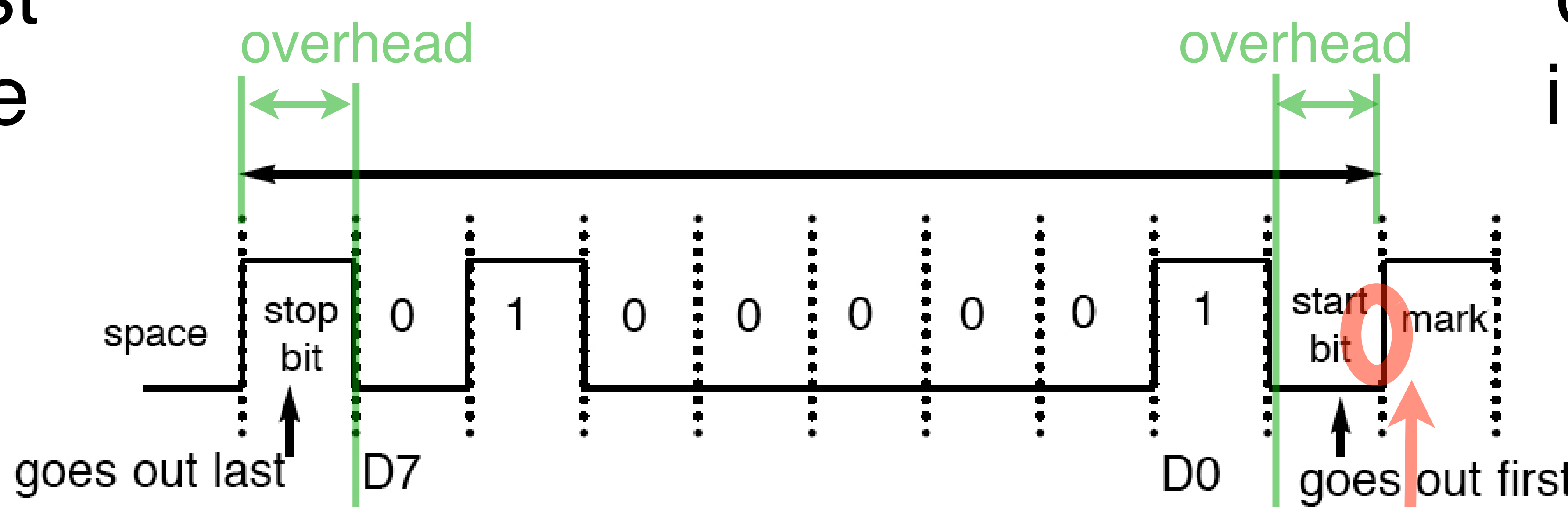
UART protocol

- During no signal: kept high
  To start: "start bit" goes low
- Each side locally generates its own clock
  - both sides must agree on clock rate
    => synchronous as hardware during data
    transfer phase, no acknowledgment!
    => sender doesn't know if receiver got it
- 1 or 2 Stop bits (high), space (low)

UART: waveform

newest
in time

oldest
in time



i.e. "Baud rate"
on both ends
must match!

Sender & Receiver are
assumed to write/read
bits at the same rate
during this time
as triggered by
their local clocks

This falling edge
can come at
any time
(asynchronous)

Asynchronous vs. Synchronous serial

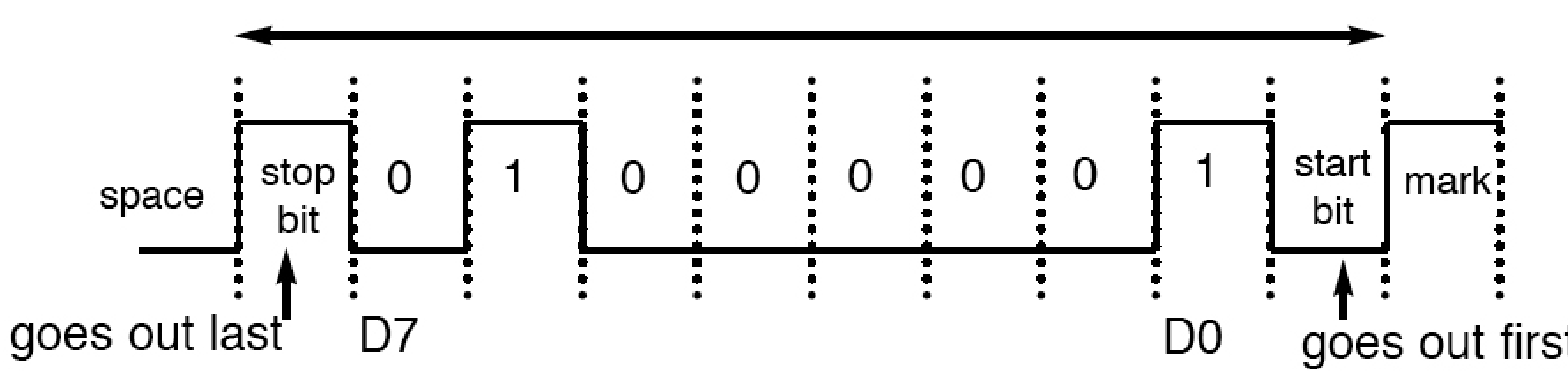
- Asynchronous (e.g. UART)
  - start bit marks the start, end bits the end however, each bit is transferred w/out ack
  - pay time overhead, save wire
- Synchronous (e.g., SPI, I2C, etc)
  - lower runtime overhead
  - extra clock wire; may need to negotiate clock

UART vs. USART

- UART: Universal Asynchronous Receiver/Transmitter
  - Controller for the asynchronous serial protocol
- USART: Universal Synchronous/ Asynchronous Receiver/Transmitter
  - UART + controller for synchronous protocol(s) (SPI, maybe I2C)

The asynchronous serial protocol

- When idle, signal (=1) (indicates online)
- Serialized bit shift, at
  - start bit establishes timing (speed)
    7 or 8 data bit, in LSB ... MSB order
    Optional Parity bit
- 1 or 2 Stop bits (=1), then space (=0)

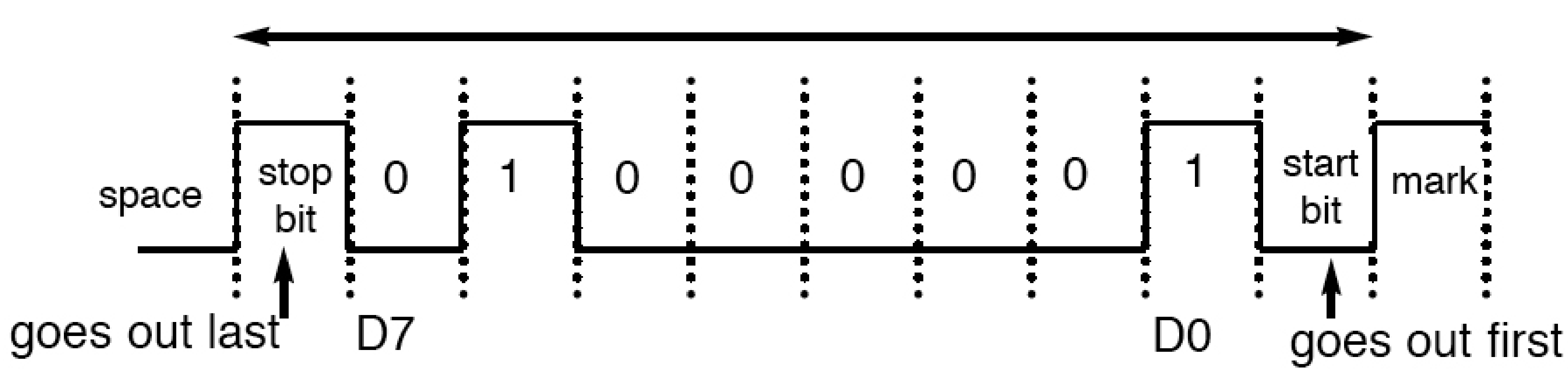


Parity bit

- *Even parity* or *Odd parity*
  - Extra bit to make the total number of 1 bits in (byte + parity bit) even or odd
- Example,
  if data=11100101 (five 1s), even parity=>1
  to make total of six (even) 1's
- Odd parity is more common, because it forces some zeros and ones

Data transfer rate

- *Bits per second* (bps)
  - Payload / actual bps (excl. start/stop bits)
  - Raw bps (treating overhead as data bits)
- *Baud rate*: number of "symbols" per second
  => not necessarily same as bps, often misused!!
  - baud may be 1-bit or multi-bit symbols
    (voltage levels)

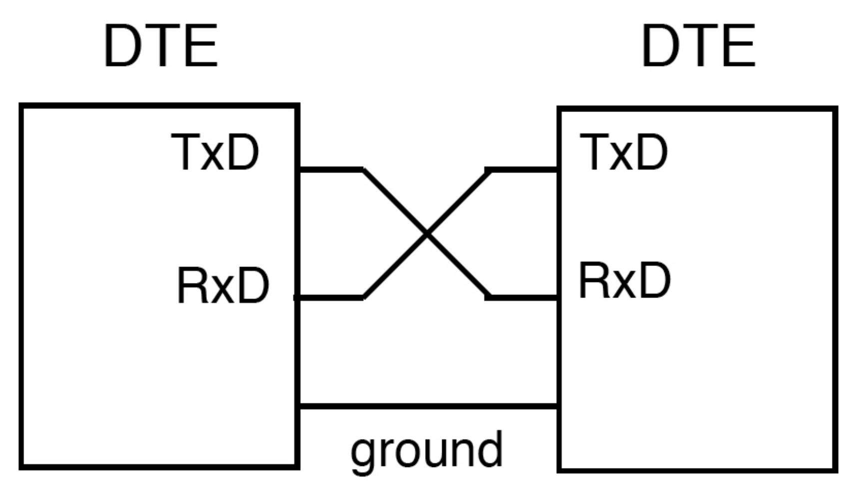


RS-232 standards

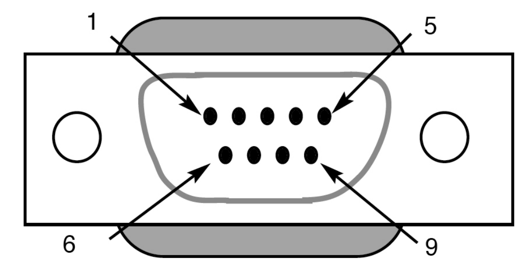
- Standard for serial comm. (COM port)
  - 1: -3V to -25V;   0: +3V to +25V
  - Reason: for long distance wired line
- Connectors
  - Minimally, 3 wires: RxD, TxD, GND
  - Could have 9-pin or 25-pin

Minimal serial connection

- 3 wires:
  - TxD (transmitted data)
  - RxD (received data)
  - GND (ground)
- Crossover
  - TxD connected to RxD of the other, and vice versa



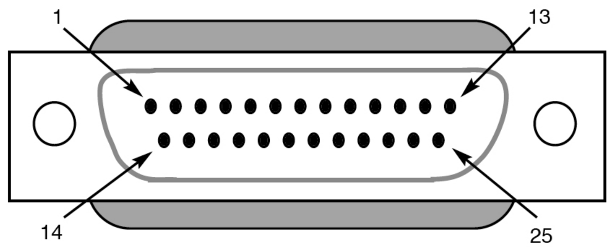
DB-9 connector



- includes *RxD*, *TxD*, *GND*
- 6 more signals for control

signal	meaning	direction
DTR	data terminal	terminal out
DSR	data set ready	peripheral out
RTS	ready-to-send	from PC,terminal
CTS	clear-to send	to PC, terminal
DCD	data carrier detect	from modem
RI	ring indicator	from modem

DB-25 connector



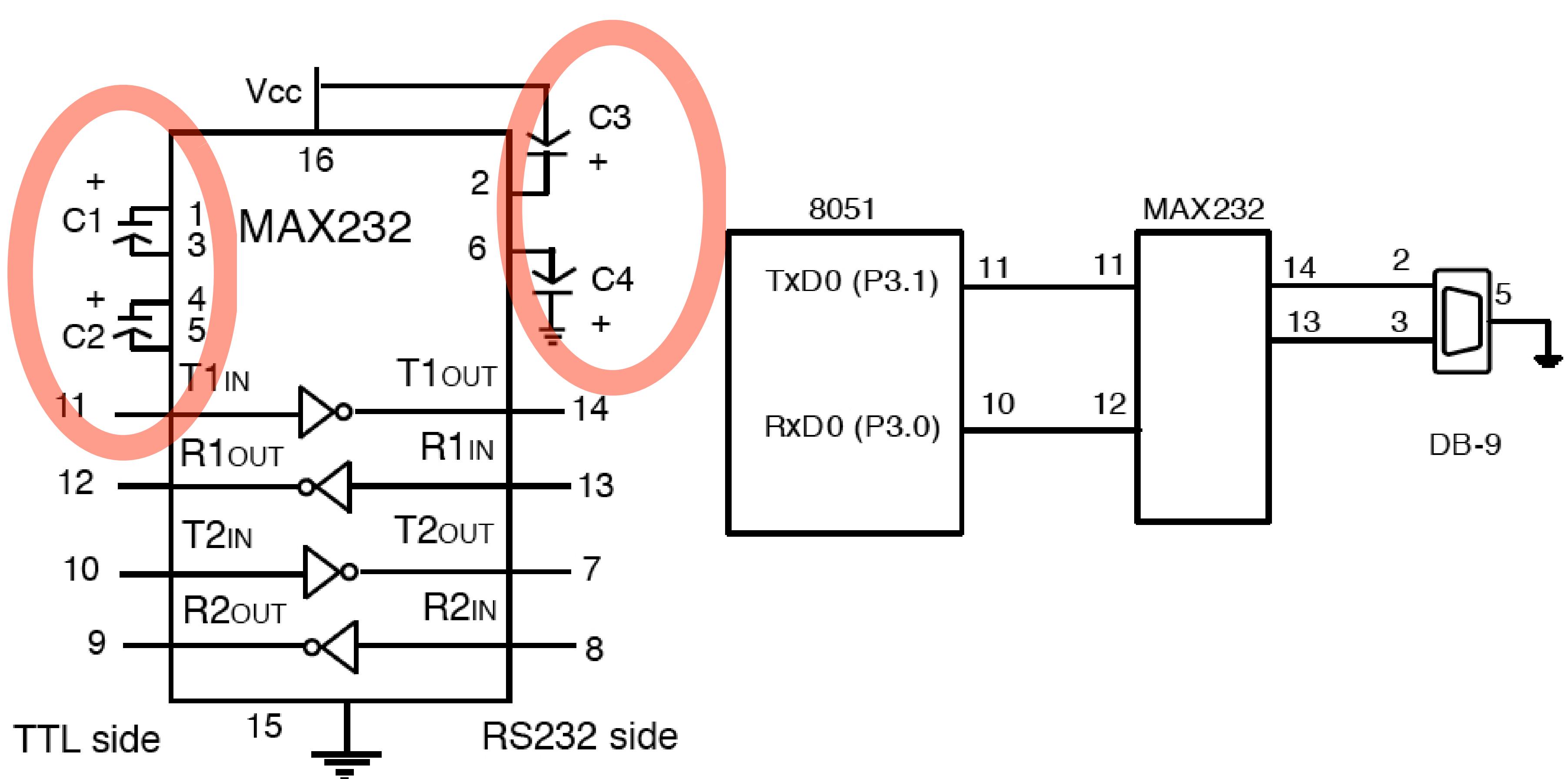
- Includes all of the DB-9 ones
- Additional signals:
  - Protective Ground
    different from Signal Ground
  - Reserved for testing
  - Secondary DCD, CTS, TxD, timing, RTS,

COM port on PC

- Two COM ports: COM1 and COM2
  - some use COM1 for mouse
    COM2 for modem (not universally)
  - COM port = UART + RS232 level conv.
- Issue: voltage bridging
  - chip: TTL level, 0-5V
    RS-232: [-25 ~ -3V] to [+3 ~ +25V]

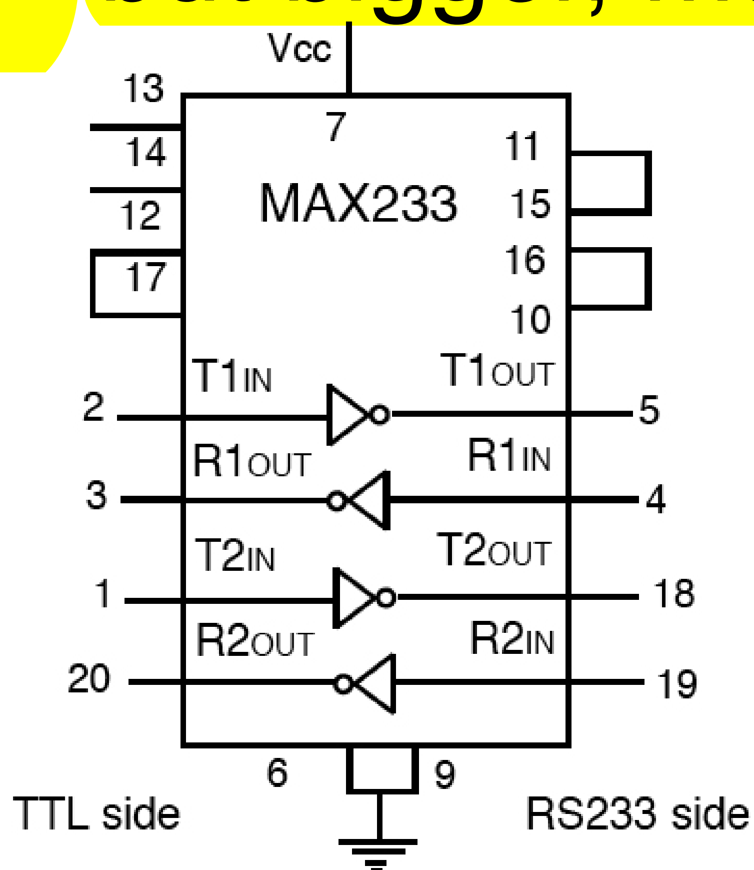
Hardware Connection

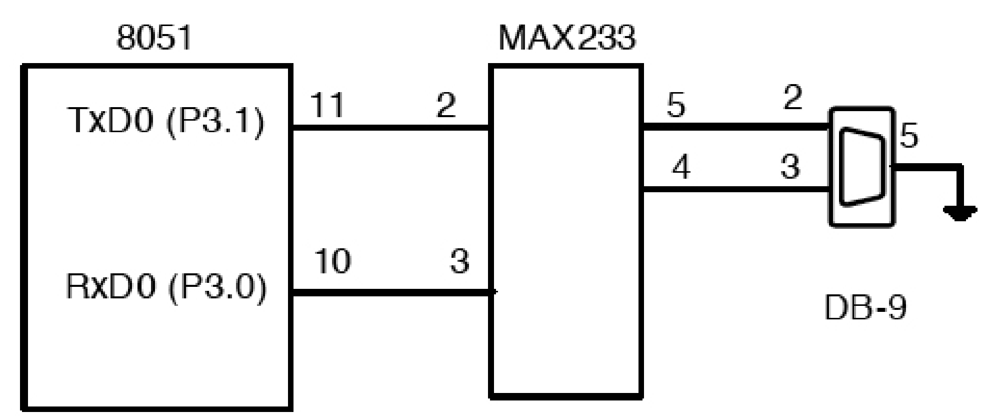
- 8051: TxD (same pin as P3.1), RxD (P3.0)
- install capacitors as indicated



Alternative hardware connection

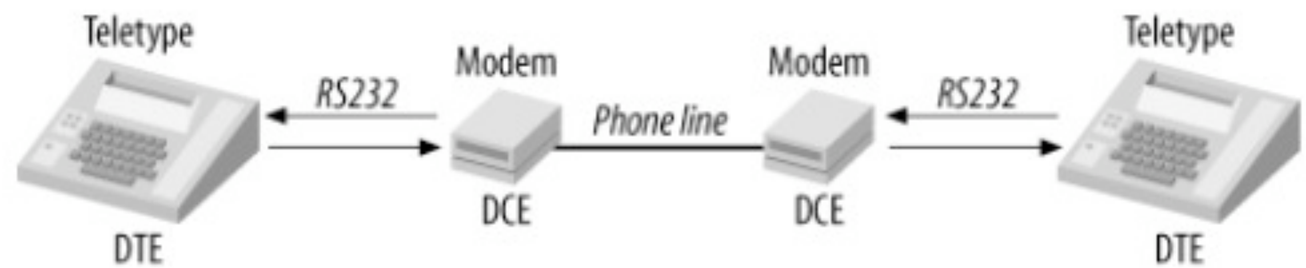
- Use MAX 233 instead of MAX232
  - Eliminates external capacitors
  - but bigger, more expensive

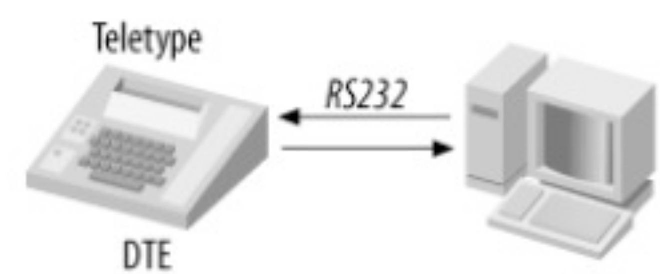


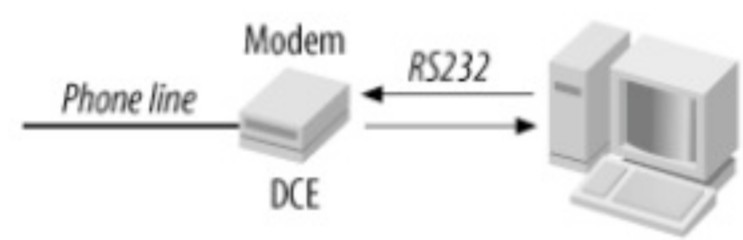


RS 232 Devices

- Teletype to modem
  - keyboard and printer or screen
  - connect to modem
  - phone line
- TTY to computer (no phone line)
- Computer to Modem







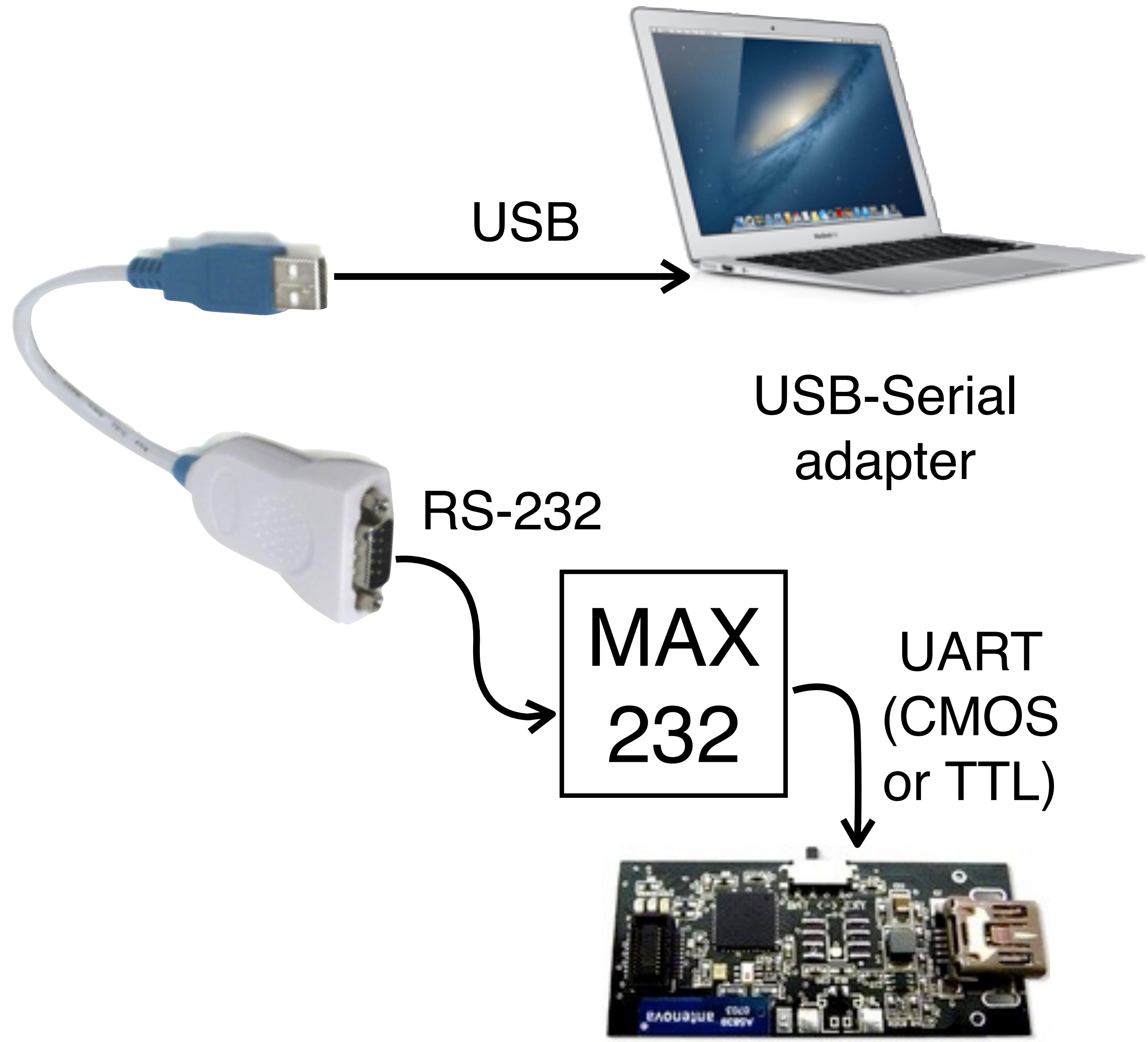
But today's computers have no RS-232 ports...

- Use USB-to-Serial Adapter
- USB on PC side
  - Shows up as "virtual COM port"
- RS-232 on device side
  - could be MCU, after max-232 conversion

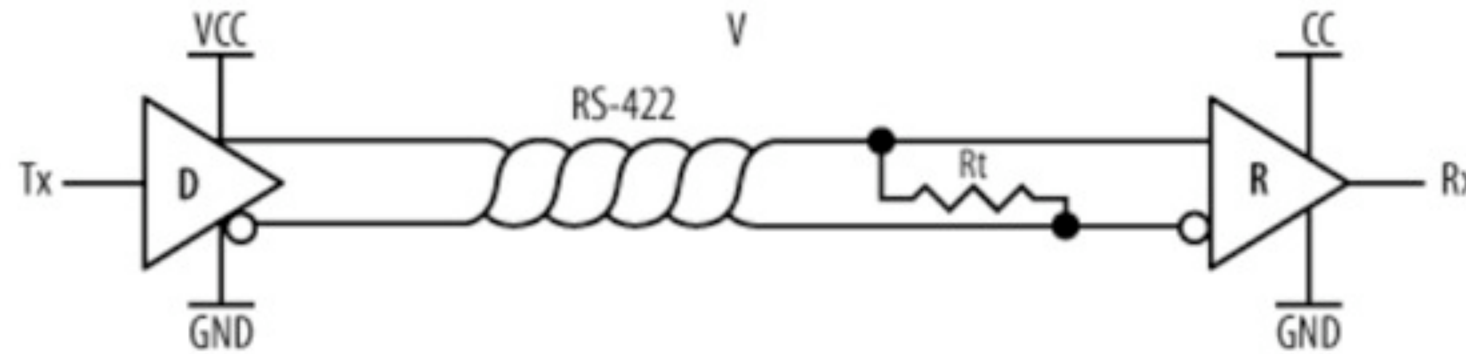
USB-Serial adapter

Possible way to connect PC to MCU

- PC side: a "serial terminal" program
  - e.g., hyperterminal
  - or Python pyserial library
- USB-serial adapter
- RS-232 to (CMOS, TTL) level converter



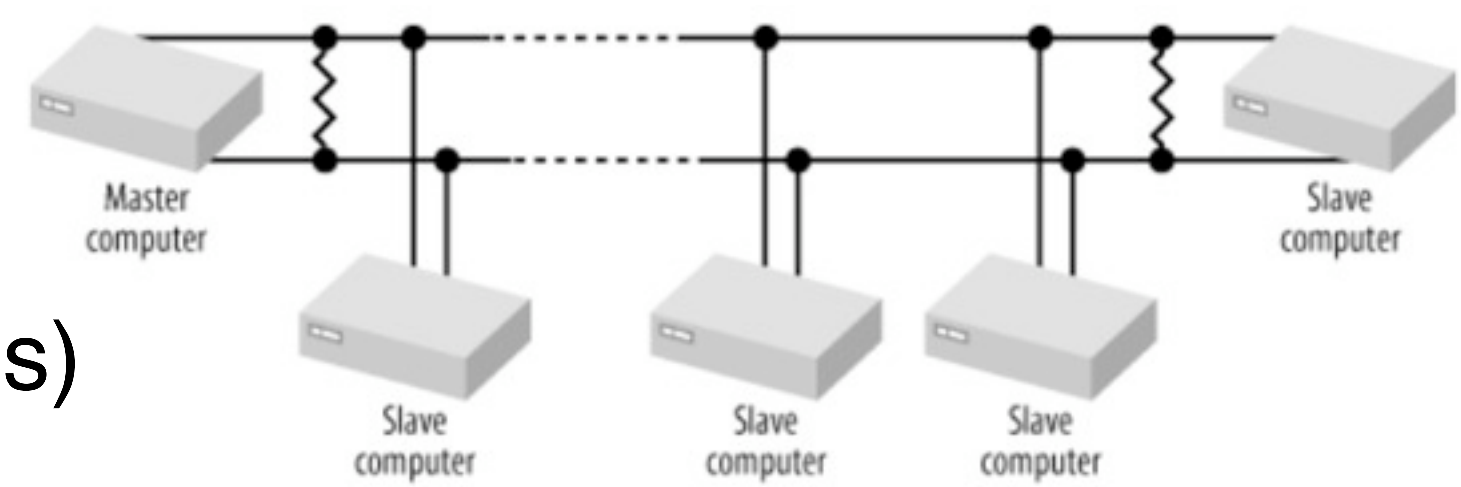
RS-422



- Differential pair
  - whereas RS-232C is referenced to local GND
- Advantages
  - more robust: higher immunity to noise
  - longer transmission distance: 1200 meters
  - higher speed
- compatible with RS-232C

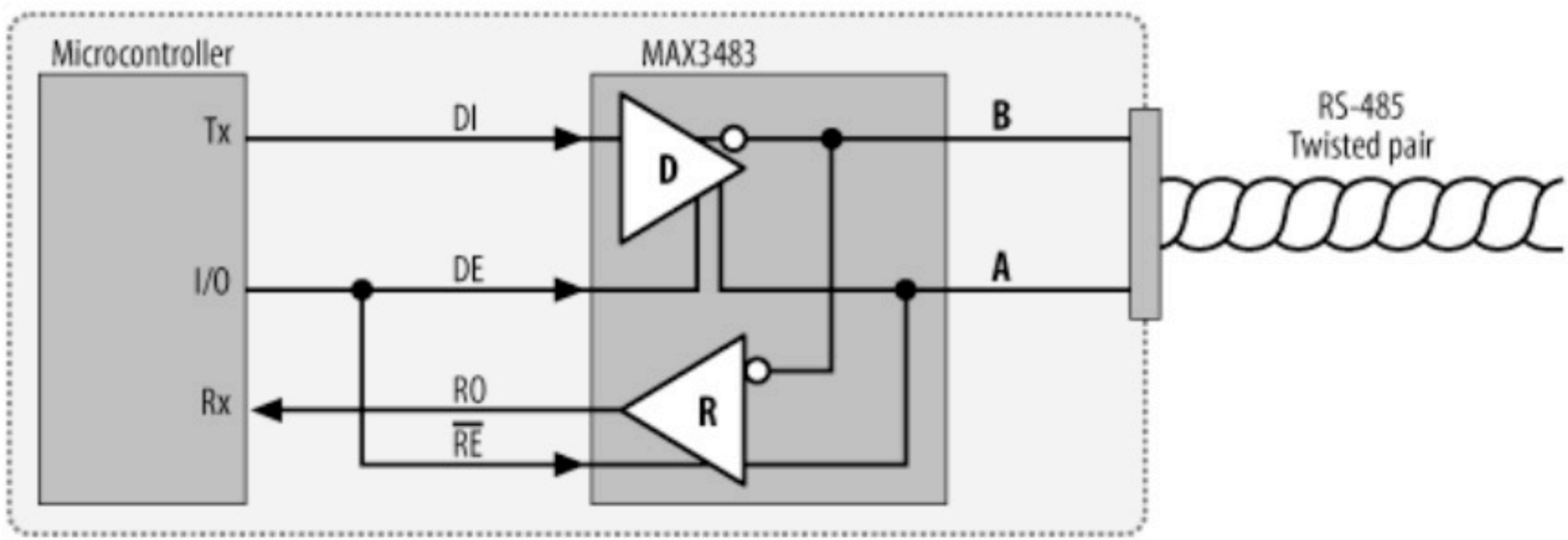
RS-485

- Master-slave architecture based on RS-422
  - initiated by master
- Uses interface chip with enable
  - Data Enable
  - /Receive Enable
  - Network size limited by chip drive (32 nodes)



RS-485

- Normally: DE and /RE low => listening
- When DE and /RE high => transmit
- Half duplex or full duplex



Accessing UART on MCU

- Configuration
  - Set up a timer with auto-reload
  - Rx/Tx enable bits
  - Timer used to generate timing for the bits
- Access
  - Reading/Writing register SBUF; could be interrupt driven

Serial port programming on the 8051

- Easy part: send/receive
  - MOV SBUF, data ;; to send
    MOV dest, SBUF ;; to receive
  - SCON (SFR) register for configuration
- Tricky part: configuring the speed for locally generated clock!
  - PC/COM-port need to set a speed, 8051 needs timer to match the speed

SCON: Serial Control register (8051 SFR)

- 8-bit reg. for serial port control
  - SM0..SM2: serial port mode
  - REN: Receive-enable
  - TB8, RB8 (Tx/Rx bit 8) (normally=0)
  - TI, RI: Tx/Rx interrupt bit (flags)

SM0	SM1	SM2	REN	TB8	RB8	TI	RI

Serial Port Modes

SM 0	SM 1	Serial Mode	Meaning	Baud rate
0	0	0	8-bit shift register	Osc / 12
0	1	1	8-bit UART	set by timer 1
1	0	2	9-bit UART	Osc / 64
1	1	3	9-bit UART	set by timer 1

if PCON.7 is set => doubles UART baud rate

SM0	SM1	SM2	REN	TB8	RB8	TI	RI

SM2: multiprocessor communication

- '0': normal receive-interrupt (RI) flag
- '1': sets RI flag only if received bit 9 = '1'
  - does not set RI flag if received bit 9 = '0'
- Why? interrupt only to notify receiver; no interrupt to transfer

SM0	SM1	SM2	REN	TB8	RB8	TI	RI

TB8, RB8

- Used in 9-bit UART mode
  - 8 bits are in SBUF
  - bit-9 transmitted (TB8) or received (RB8)
- How to use it
  - set or clear TB8, then write SBUF
  - on receive, read SBUF and then RB8

SM0	SM1	SM2	REN	TB8	RB8	TI	RI

Timer 1 and UART

- Timer cycle time = 12x crystal cycle time
- UART clock time = 32x Timer cycle time
- Standard baud rates:
  1200, 2400, 4800, 9600, 14.4k, 19.2k, 28.8k..
- Use 11.0592MHz crystal,
  11.0592MHz / 12 / 32 = 28.8 KHz (exactly)

Timer in reload mode

- How to determine the timer reload value:
  - 28800 / 9600 = 3
    28800 / 4800 = 6
  - assume PCON.7='0'
- Double the baud rate by setting PCON.7='1'

Baud	TH1 (dec)	TH1 (hex)
9600	-3	FD
4800	-6	FA
2400	-12	F4
1200	-24	E8

Steps in serial transfer

- TMOD=20H (timer 1 mode 2 auto reload)
  Load TH1 to match baud rate
- SCON = 50H for serial mode 1:
  8-bit, start/stop bits
- Start timer TR1
- CLR TI to clear interrupt flag, poll TI
- on flag, read from SBUF

Example: Receive and put in P1

```
       MOV   TMOD, #20H    ;; timer 1 mode 2
       MOV   TH1, #-6      ;; 4800 baud
       MOV   SCON, #50H    ;; 8-bit 1 stop REN
       SETB  TR1           ;; start timer 1
HERE:  JNB   RI, HERE      ;; wait to receive
       MOV   A, SBUF       ;; read in the char
       MOV   P1, A         ;; write to port
       CLR   RI            ;; clear
       SJMP  HERE          ;; repeat
```

Run in EdSim @11.0592MHz, 4800 baud