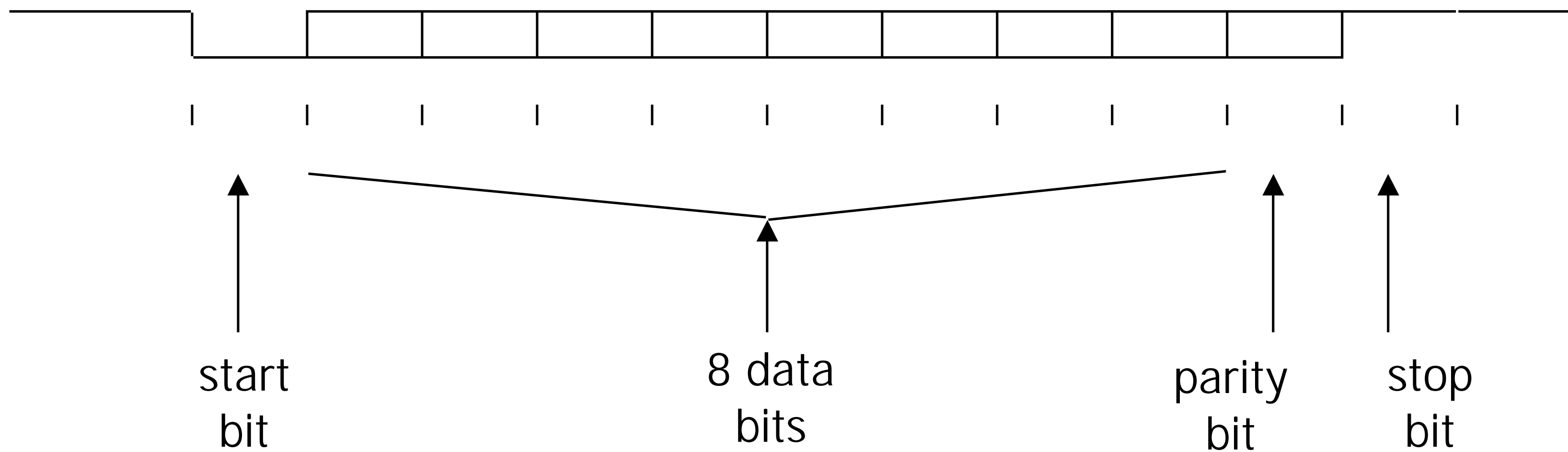
Serial Communication

- RS-232 (standard serial line)
  - Point-to-point, full-duplex
  - Synchronous or asynchronous
  - Flow control
  - Variable baud (bit) rates
  - Cheap connections (low-quality and few wires)

Serial data format

- Variations: parity bit; 1, 1.5, or 2 stop bits



RS-232 wires

- TxD – transmit data
- TxC – transmit clock
- RTS – request to send: Handshake
- CTS – clear to send : Handshake

- RxD – receive data
- RxC – receive clock
- DSR – data set ready: Handshake
- DTR – data terminal ready: Handshake

- Ground

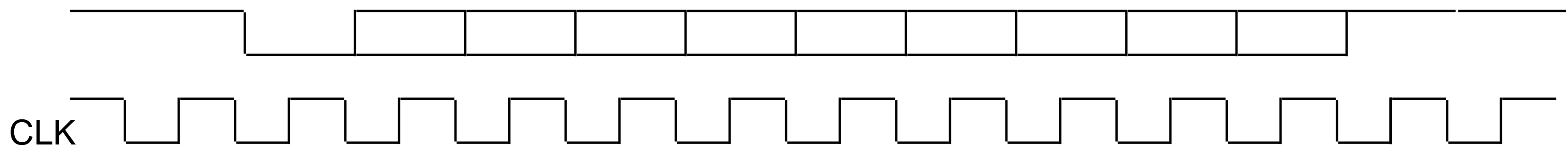
all wires active low

"0" = -12v, "1" = 12v

special driver chips that generate ±12v from 5v

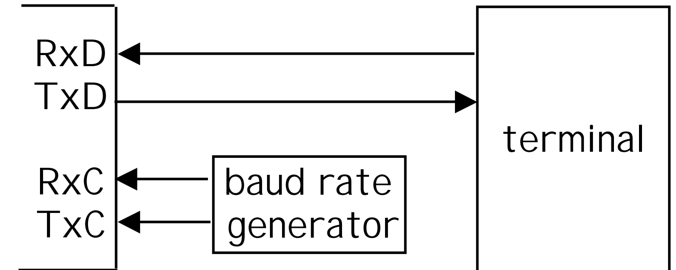
Transfer modes

- Synchronous
  - clock signal wire is used by both receiver and sender to sample data
- Asynchronous
  - no clock signal in common
  - data must be oversampled (16x is typical) to find bit boundaries
- Flow control
  - handshaking signals to control rate of transfer

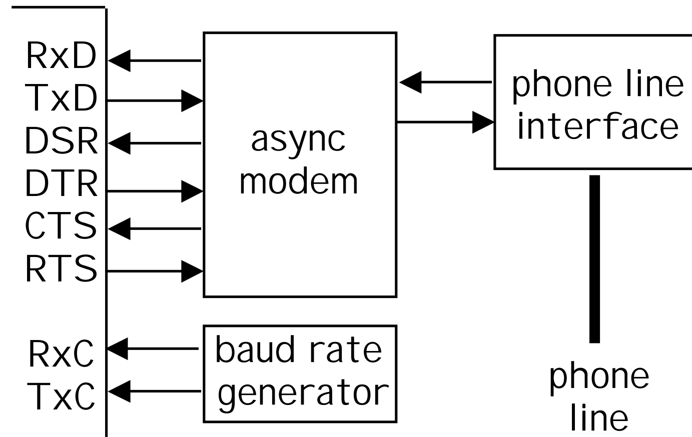


Typical connections

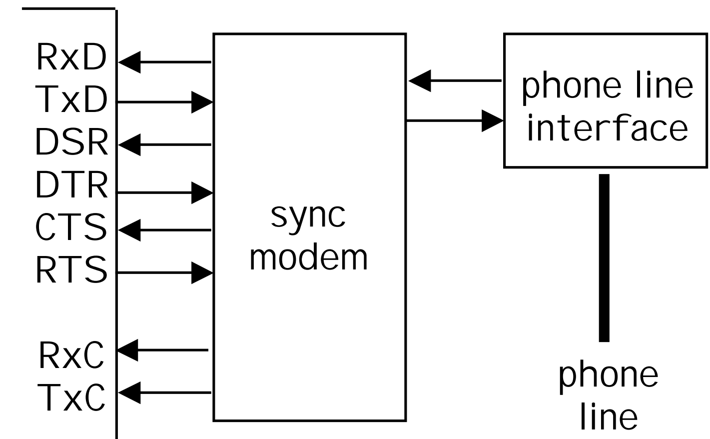
- Terminal



- Asynchronous modem



- Synchronous modem



8051 Serial Interface

- TxD: Port 3, pin 1
  - Transmit data shifted out
- RxD: Port 3, pin 0
  - Receive data shifted in
- Full duplex: both operate in parallel
- We will use Mode 1 only
  - asynchronous
  - 10 bit transfer: 1 start, 8 data, 1 stop
  - Look at documentation for other modes
- Clock for serial shift provided by timer 1
  - i.e. programmable baud rate
  - takes away a timer from other uses

Serial Port Control Register (SCON)

- Configures the serial interface

MSB							LSB
SM0	SM1	SM2	REN	TB8	RB8	TI	RI

Where SM0, SM1 specify the serial port mode, as follows:

SM0	SM1	Mode	Description	Baud Rate
0	0	0	shift register	$f_{OSC}/12$
0	1	1	8-bit UART	variable
1	0	2	9-bit UART	$f_{OSC}/64$ or $f_{OSC}/32$
1	1	3	9-bit UART	variable

SM2 Enables the multiprocessor communication feature in Modes 2 and 3. In Mode 2 or 3, if SM2 is set to 1, then RI will not be activated if the received 9th data bit (RB8) is 0. In Mode 1, if SM2=1 then RI will not be activated if a valid stop bit was not received. In Mode 0, SM2 should be 0.

REN Enables serial reception. Set by software to enable reception. Clear by software to disable reception.

TB8 The 9th data bit that will be transmitted in Modes 2 and 3. Set or clear by software as desired.

RB8 In Modes 2 and 3, is the 9th data bit that was received. In Mode 1, it SM2=0, RB8 is the stop bit that was received. In Mode 0, RB8 is not used.

TI Transmit interrupt flag. Set by hardware at the end of the 8th bit time in Mode 0, or at the beginning of the stop bit in the other modes, in any serial transmission. Must be cleared by software.

RI Receive interrupt flag. Set by hardware at the end of the 8th bit time in Mode 0, or halfway through the stop bit time in the other modes, in any serial reception (except see SM2). Must be cleared by software.

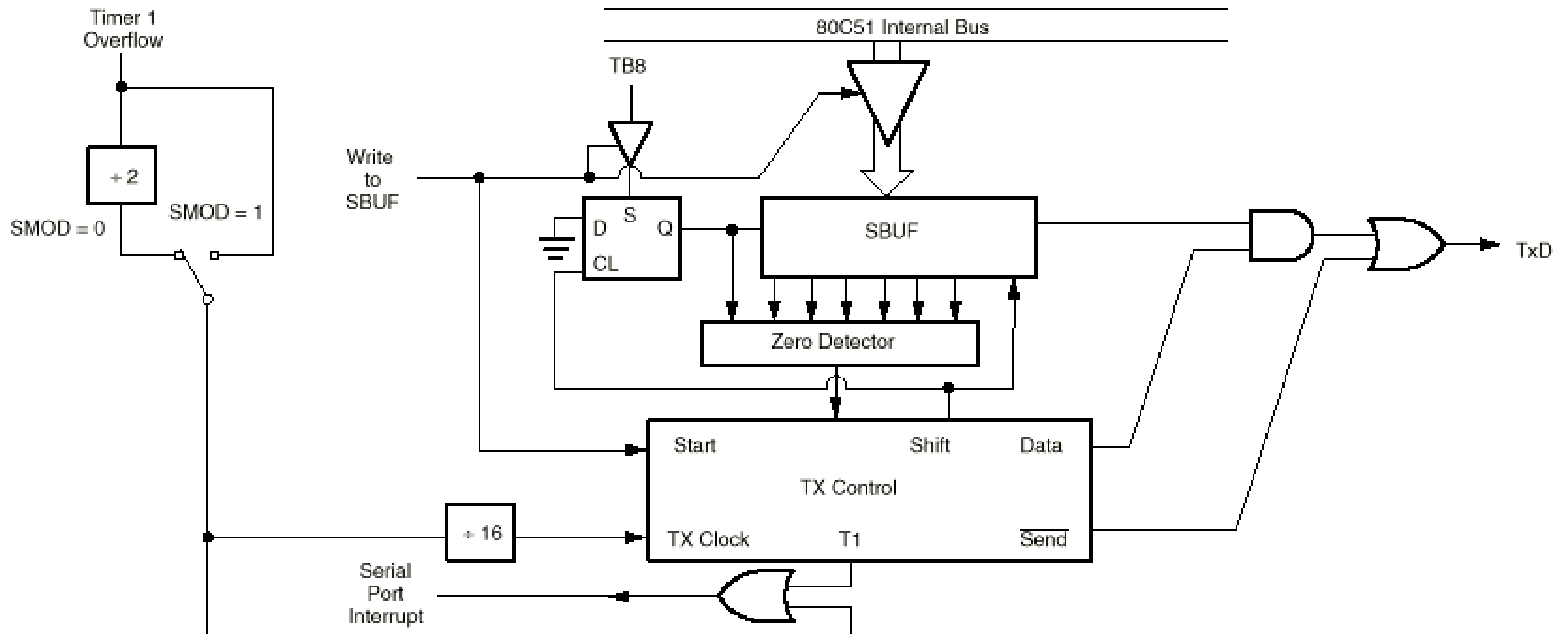
SU00120

Figure 11. Serial Port Control (SCON) Register

Baud Rate Generator

- Use timer 1 overflow to generate serial data clock
  - serial clock is 16x oversampled, i.e. baud rate x16
  - SMOD bit (PCON register)
    - 0: divides baud rate by 2
- Typical timer 1 setup
  - auto-reload timer
  - reload value determines overflow clock rate
- Baud rate calculation
  - Clocks between overflows = clocks
  - Overflow frequency =
  - Baud rate (assuming SMOD = 1)
  - Baud rate =
  - Max Baud rate =
  - TH1 value for 9600 baud =

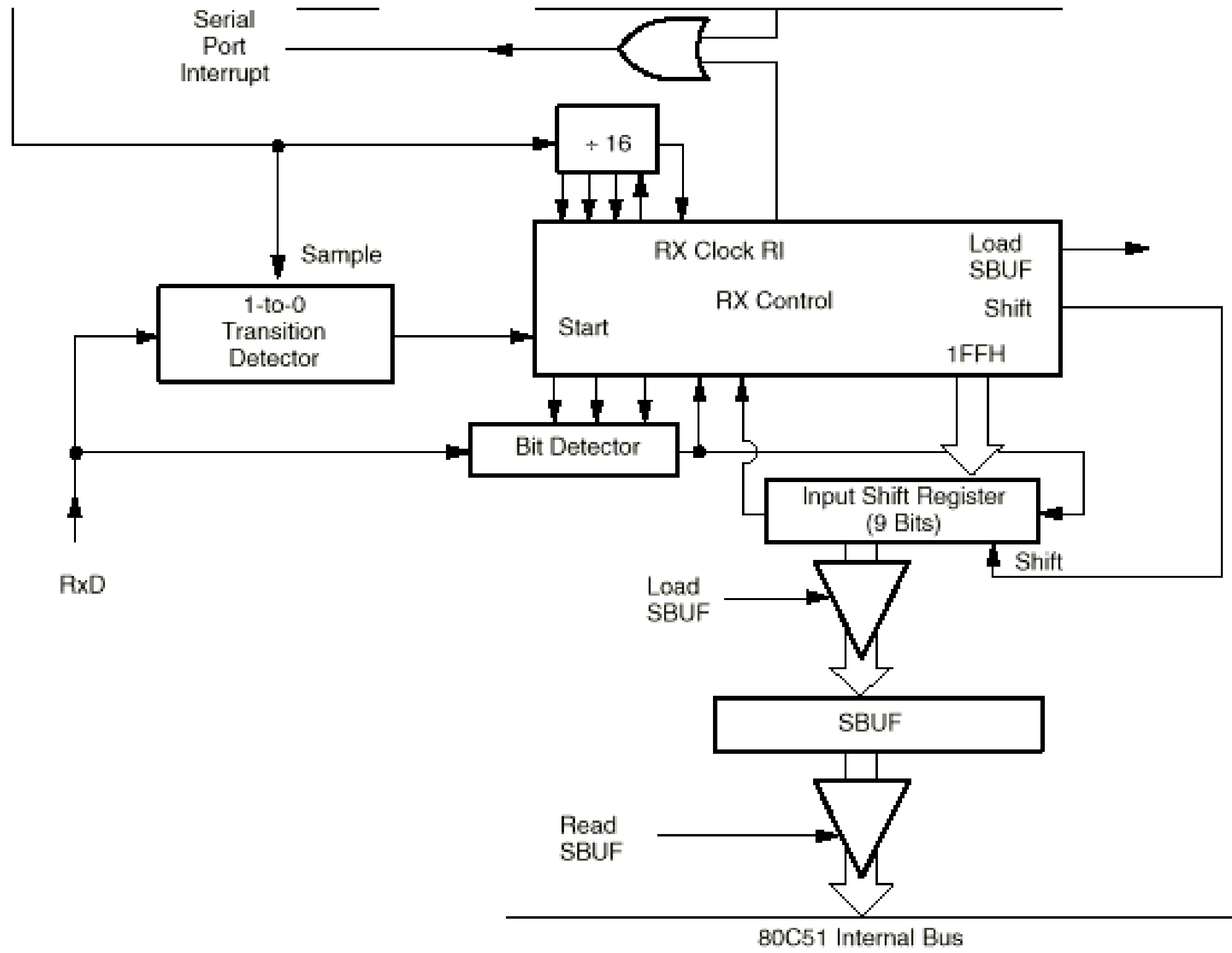
8051 Serial Interface Transmitter



Sending Serial Data

- Transmission is initiated by a write to SBUF
  - start, data and stop bits shifted out automatically
  - TI (transmit interrupt) set when stop bit goes
    - indicates that interface is ready for next character
    - TI can be polled, or used to interrupt
    - must reset it in the software

8051 Serial Receiver Interface



Receiving Serial Data

- Reception is initiated by a 1-0 transition - a start bit
  - data is sampled and shifted in automatically
  - on the stop bit, the 8 data bits are loaded into SBUF
    - same address, but different register and sending SBUF
  - RI (receive interrupt) set when SBUF is loaded
    - indicates a character is ready
      - next character can start entering before SBUF is read
      - must read SBUF before next character arrives
    - RI can be polled, or used to interrupt
    - must be reset in the software

Serial Interface Interrupts

- RI and TI share the same interrupt
  - Interrupt #4
- Interrupt routine must look at RI and TI to see which caused the interrupt
- Routine must reset RI or TI before returning
  - If both RI and TI are on, another interrupt will happen right away
  - Which bit do you check first?

Baud Rate Generator

- Use timer 1 overflow to generate serial data clock
  - serial clock is 16x oversampled, i.e. baud rate x16
  - SMOD bit (PCON register)
    - 0: divides baud rate by 2
- Typical timer 1 setup
  - auto-reload timer
  - reload value determines overflow clock rate
- Baud rate calculation
  - Clocks between overflows = 12 x (256-TH1) clocks
  - Overflow frequency = Fclk/Clocks-between-overflows
  - Baud rate (assuming SMOD = 1)
    - 1/16 x overflow-frequency
  - Baud rate = 24MHz / (16 x 12 x (256-TH1))
  - Max Baud rate = 125KHz
  - TH1 value for 9600 baud = 13

getchar() / putchar()

- c = getchar()
  - returns the character in the buffer, if there is one
  - returns NULL otherwise
  - could check for error (character overrun)
- r = putchar(c)
  - sends the character to the serial port, if it is not busy
  - returns c for normal operation, NULL if port was busy
- Simple operation, no need for interrupts

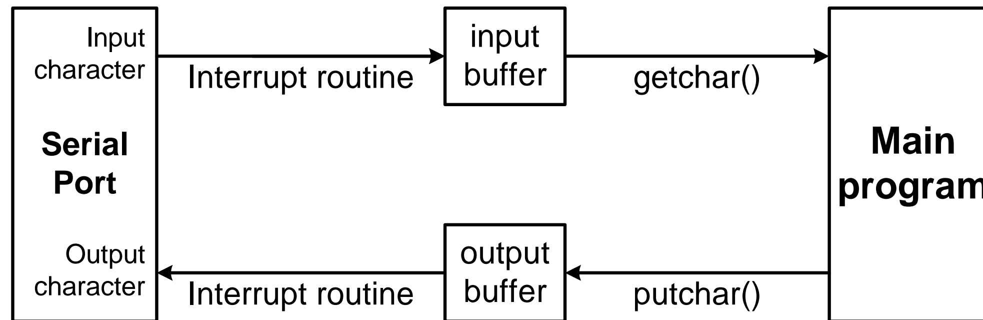
```
while ((c = getchar) == NULL) { };

while (putchar(c) == NULL) { };
```

- Polling doesn't allow us to do anything else
- If we are busy, we might miss a character

getchar() / putchar() (Part 2)

- We'll add a 1-character buffer for both input and output
- getchar()
  - interrupt when a new character arrives
  - if the buffer is empty, place character in buffer
  - otherwise, set error flag (new function to check for errors)
  - getchar() now looks at the buffer for a character
  - otherwise the same as before
- putchar()
  - interrupt when a character has been sent
  - if the buffer has a character, send it to the serial port
  - putchar() now puts the character into the buffer
  - otherwise the same as before
  - what if the buffer is empty when interrupt occurs?
    - new character to buffer will not be sent
- Complication: one interrupt routine for both input and output

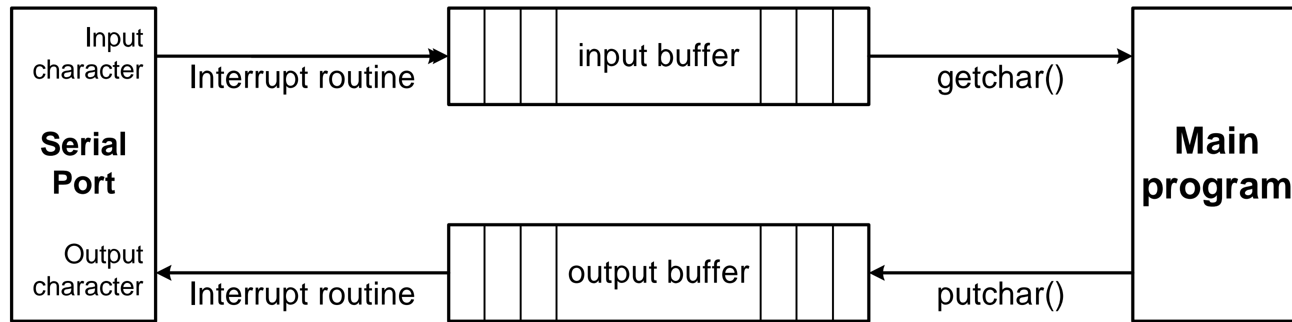
getchar() / putchar() (Part 2)



getchar() / putchar() (Part 3)

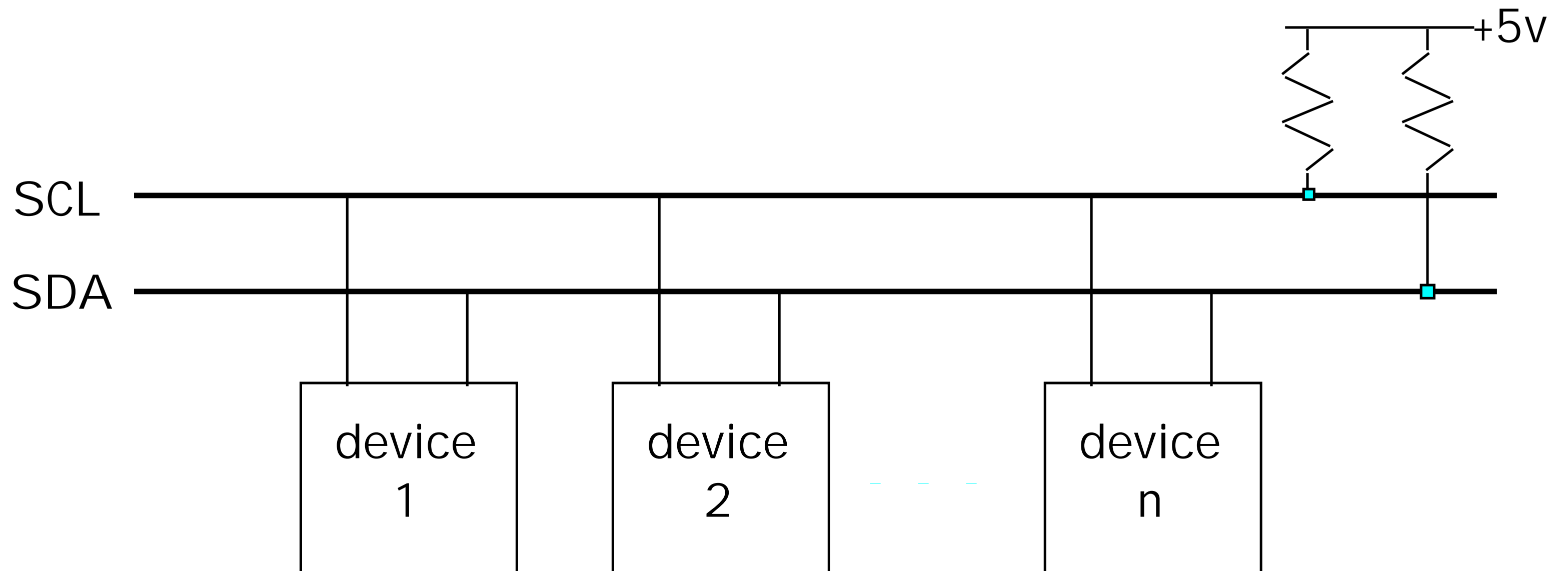
- The 1-character buffer gives us some time to read/write
  - but not a lot
- Extend the 1-character buffers to 32 characters buffers
  - now we can go away for a long time and not miss incoming characters
  - we can write out lots of characters and not wait for them all to go
- Each buffer now becomes a queue
  - standard circular queue
    - 33 character vector (why 33?)
    - head, tail pointers
  - initialize on startup
- getchar()
  - interrupt routine writes characters to buffer, getchar() reads
- putchar()
  - putchar() writes characters to buffer, getchar() reads

getchar() / putchar() (Part 3)



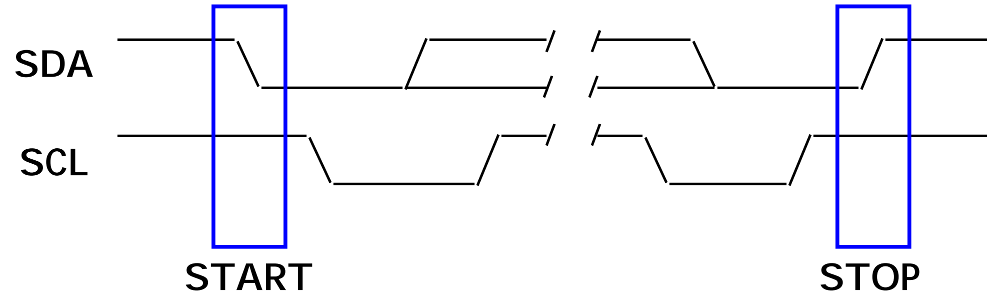
Inter-Integrated Circuit Bus (I2C)

- Modular connections on a printed circuit board
- Multi-point connections (needs addressing)
- Synchronous transfer (but adapts to slowest device)
- Similar to Controller Area Network (CAN) protocol used in automotive applications



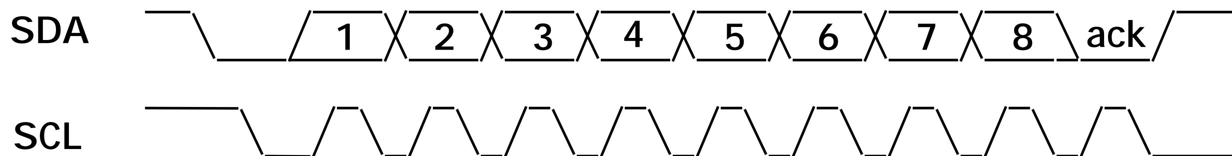
Serial data format

- SDA going low while SCL high signals start of data
- SDA going high while SCL high signals end of data
- SDA can change when SCL low
- SCL high (after start and before end) signals that a data bit can be read



Byte transfer

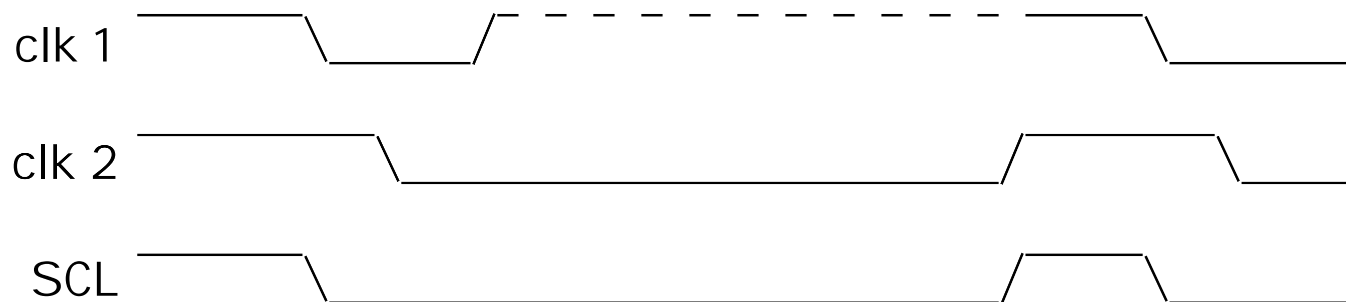
- Byte followed by a 1 bit acknowledge from receiver
- Open-collector wires
  - sender allows SDA to rise
  - receiver pulls low to acknowledge after 8 bits



- Multi-byte transfers
  - first byte contains address of receiver
  - all devices check address to determine if following data is for them
  - second byte usually contains address of sender

Clock synchronization

- Synchronous data transfer with variable speed devices
  - go as fast as the slowest device involved in transfer
- Each device looks at the SCL line as an input as well as driving it
  - if clock stays low even when being driven high then another device needs more time, so wait for it to finish before continuing
  - rising clock edges are synchronized



Arbitration

- Devices can start transmitting at any time
  - wait until lines are both high for some minimum time
  - multiple devices may start together - clocks will be synchronized
- All senders will think they are sending data
  - possibly slowed down by receiver (or another sender)
  - each sender keeps watching SDA - if ever different (driving high, but its really low) then there is another driver
  - sender that detects difference gets off the bus and aborts message
- Device priority given to devices with early 0s in their address

Inter-Integrated Circuit Bus (I2C)

- Supports data transfers from 0 to 400KHz
- Philips (and others) provide many devices
  - microcontrollers with built-in interface
  - A/D and D/A converters
  - parallel I/O ports
  - memory modules
  - LCD drivers
  - real-time clock/calendars
  - DTMF decoders
  - frequency synthesizers
  - video/audio processors