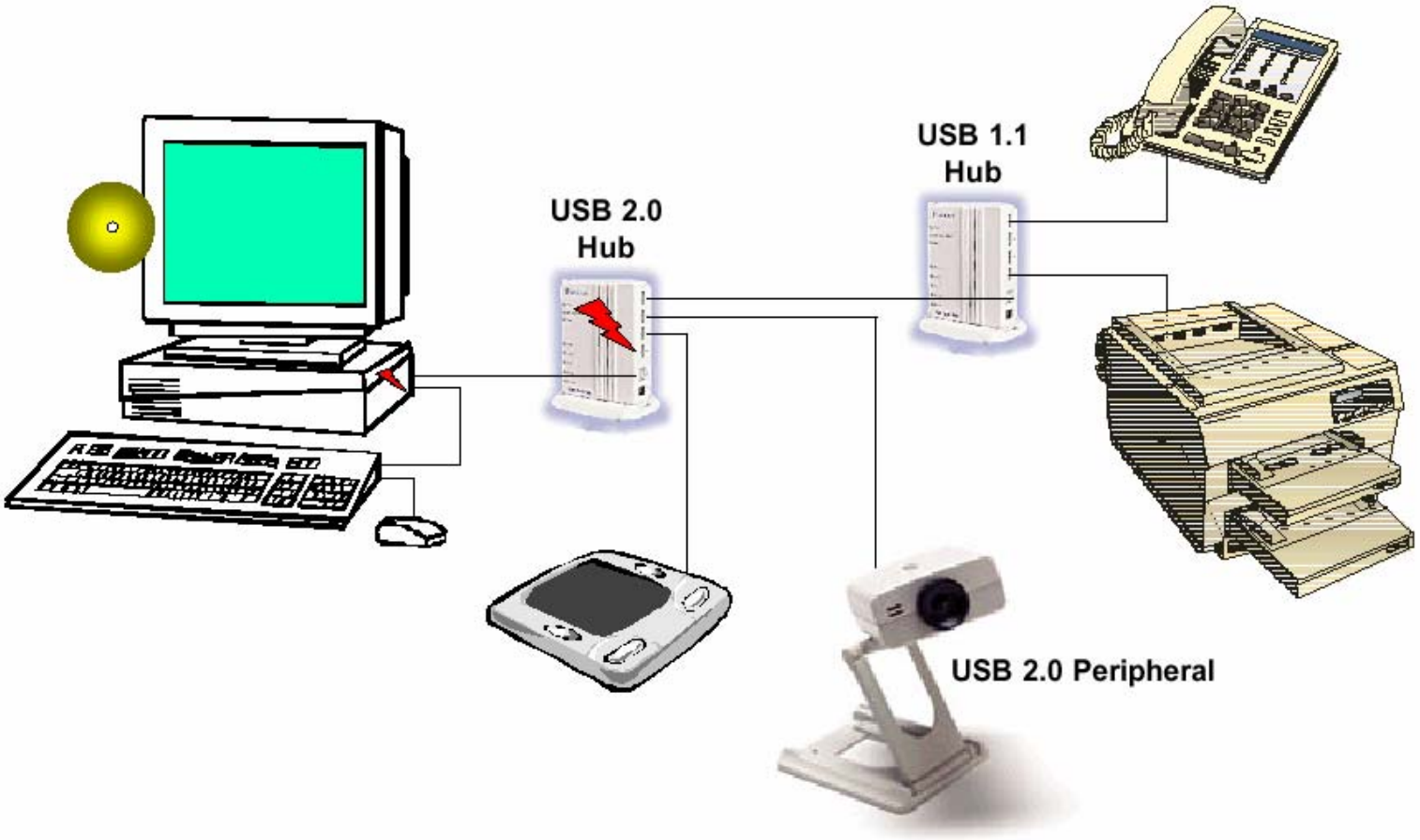
Overview

Serial communication is economical and widely used:

- modem
- mouse
- keyboard
- Printer
- Scanner
- Digital camera
- …

Serial Communication



Serial I/O Basics

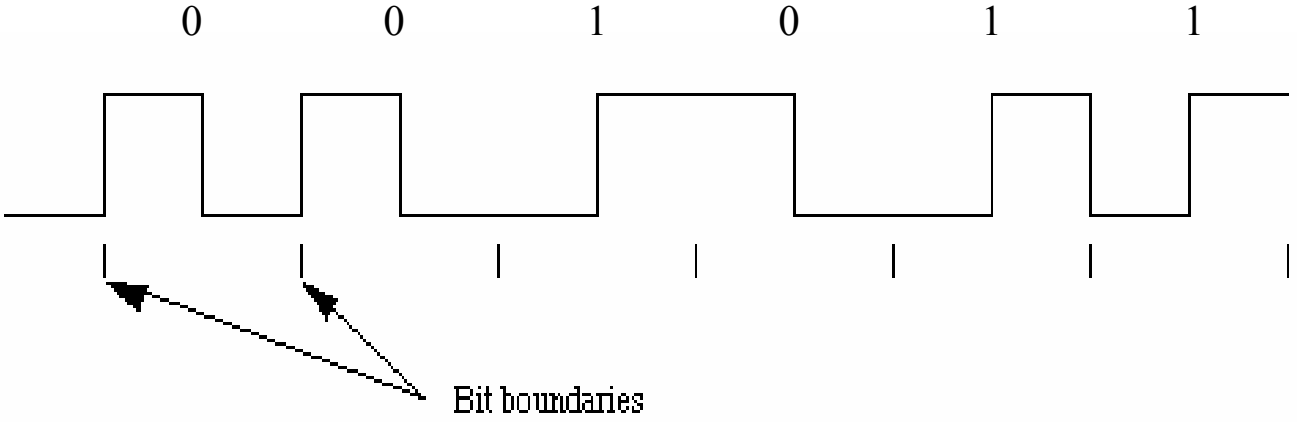
- Coding
  - Bit rate
  - Clock synchronization

- Major concern (Protocol)
  - How to split the incoming data stream into individual units, bits, bytes and packets
  - How does the receiver know where a data word starts and where it stops?
  - How to configure and control different devices and their configurations?

Manchester Code (IEEE Definition)

- Two conflicting requirements
  - Synchronization of sender's and receiver's clocks (more High-Low Transitions help)
  - Bit rates. Cables have capacitance. The more is the transition, the higher is the frequencies and the more is the demand on the quality of cables. (Less transitions are better. Why?)

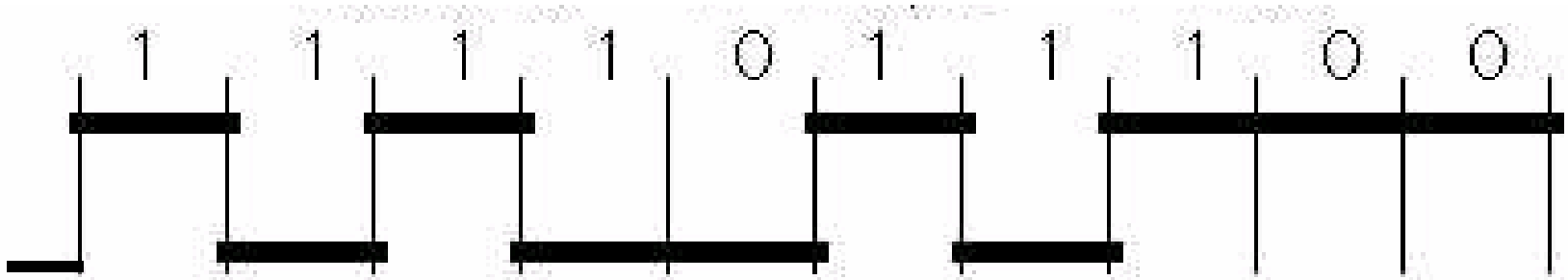
- Manchester code is "clock friendly". It is used in 10 Mbit Ethernet. It always has a transition in the middle. One to zero transition is 0. Zero to one transition is 1. So what is the code here?



Non Return to Zero Inverted (NRZI) Code

- However, Manchester code is not bit rate friendly. It has one transition per bit, so that high speed USB and 100 Mbit dump it and use non zero return invert code.

- If there is a transition at the beginning, it is 1. Otherwise, it is zero. What is this code under NRZI?



Asynchronous vs. Synchronous

- Asynchronous (e.g., UART, character by character)
  - Transmitter's and receiver's clocks are not synchronized.
  - Very old (slower) transmission system (telephone,Morse code, etc.)
  - Time interval between successive data may vary.
  - Character oriented (ASCII: 7 bit, Unicode: 16bit)
  - Receiver's clock is typically 16x of bit rate

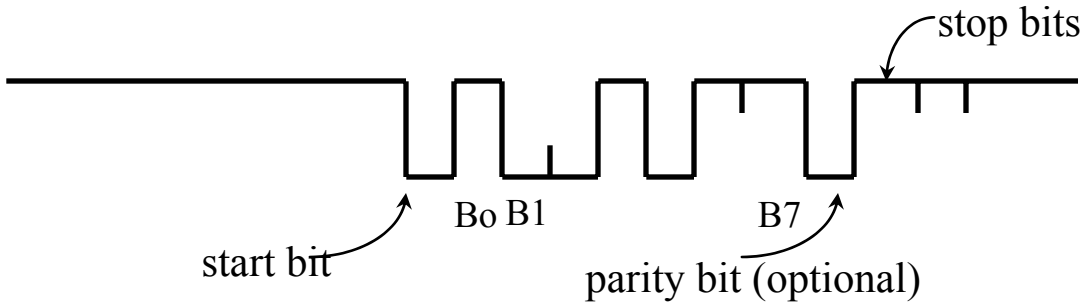
- Synchronous (e.g., USB, frame by frame)
  - More efficient (w/o start and stop bits)
  - Clocks between transmitter and receive needs to be synchronized periodically, e.g., USB synchronizes clocks packet by packet

UART

- From the viewpoint of the computer, UART is a pair of I/O ports
- UART clock: 8x, 16x, 64x times the bit rate. After a frame starts, it is used count and find the approximate CENTER of a bit for sampling.

  (what is the advantage of having a faster clock rate?)
- UART control and status registers
  - number of data bits, number of stop bits
  - flow control - slow down the sender if it is too fast for the receiver
  - use parity bit or not, even or odd if used
  - 16, 32 or 64 byte FIFO data buffers are used to reduce the interrupts to the computer.
  - interrupt is normally enabled, telling the computer it has completed the sending or has received data. If interrupt is disabled, the UART has to be checked periodically.

Asynchronous Communication Basics



•Sender and receivers clocks runs at the SAME rate but not synchronized.

•The clocks run 8x, 16x, or 64x of the bit rate. The idle (normal) line logic is 1, no frame being transmitted.

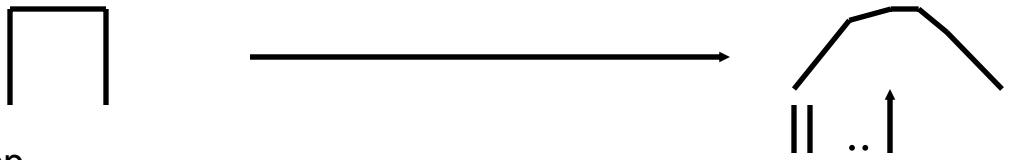
•when the line drops to zero after idle, it signals the start of a frame

•If 16x is used, after 7 (or 8 if parity is enabled) bits, sender goes to 1 for at least 1 bit (stop bit) to signal the end of a frame. (Is it necessary? Why?)

•Efficiency: 7/(7 + 3) = 70%

PseudoCode for Receiving Frames

//in real world, the pulse will not be rectangular after traveling a distance. The following pseudo code tries to sample the middle of the pulse.



- Loop
  - Wait for a start bit(high-low transition) ; //causes the UART to start //receiving the data
  - Set bit-counter to zero
  - While (bit-counter<8) //the frame consists of 7 bits
    - set tick-counter to count from 0 to 7 //the clock is 16x
    - sample the signal and store the bit //approx middle of bit time
    - set tick-counter to count from 0 to 7 again
    - increment bit-counter
  - EndWhile
  - Check stop bit ; stop bit causes UART to go back to Idle state
  - Report error if there is no stop bit
- EndLoop

Flow Control

- Software flow control (e.g. Xon - Xoff)
  - Xon: start to transfer, 11 H.
  - Xoff: stop transfer, 13H
  - Receiver sends Xon/Xoff. Transmitter acts accordingly to avoid overflow the receiver's buffer.

- Hardware flow control
  - uses RS-232's request-to-send (RTS) and clear-to-send signal (CTS) lines.
  - DTE (e.g. PC) asserts RTS when it wants to send data
  - DCE (e.g. modem) asserts CTS when it is ready to receive data from the PC.

RS-232 (on the way out)

- The comm. ports in PCs implement the RS232 standard
- DTE (data termination equipment) uses male connector
- DCE (data communication equipment) uses female connector
- RS-232 signals are divided into data, status, control and ground.

DB-9	DB-25	I/O	Comment
1	8	in	DCD data carrier detect
2	3	in	RX receive data
3	2	out	TX transmit data
4	20	out	DTR data terminal ready
5	7		GND ground
6	6	in	DSR data set ready
7	4	out	RTS request to send
8	5	in	CTS clear to send
9	22	in	RI ring indicator

Basic Ideas USB

- USB can thought as a simplified small “master – slave” LAN.
- Up to 127 devices, you may hot swap connections
- Standardized data format and control to support plug and play
- Use polling to check if a device is present or absent
- Use a sync byte to get clocks sync for each packet (DPLL)
- Use token packet to control and data packet for communication

- Packet IDentification (PID)code tells the nature of the packet, e.g.

E1H/69H	Out/IN	Token	Host to Device/Device to Host
D2H/5AH	ACK/NACK	Token	Ready to receive/Not Ready
2DH	SET UP	Token	Set up command
C3H/4BH	Data Packet Even/Odd	Data Up to 1023byte	Data packets

Digital Phase Lock Loop

- A phase lock loop (PLL) is an electronic circuit that adjusts a local oscillator so as to keep a constant phase angle relative to a reference signal

- In digital data communication, the signal are square waves (1's and 0's).A digital phase lock loop (DPLL) is a FSM that adjust the local clock (oscillator), so that the sending side's and receiving side's clocks are synchronized with respect to the reference signal.

Operation of DPLL

- Suppose that we use 8x sampling, Manchester code and that our synchronization patter is three 1's in a row, the following is what we've got in the buffer (NRZI is actually used in USB)
  - 000011110000111100001111 3 1's in a row. This is perfect
  - 11 000011110000111100011 missing the first 00011
- Identify if local clock is too fast or too slow
- Adjusted the local clock (count in the clock counter) accordingly

Summary

Today, we have studied

- How to send and receive frames
- How UARTs work
- How USB work