USB/FireWire Simulation

- The goal of this simulation is to write a VHDL model for a system that is somewhat similar in nature to the new serial standards USB (Universal Serial Bus) and IEEE Firewire
- Simulation aspects
  - Totally event driven – no global clock
  - Communication is bidirectional, half-duplex over a single wire
  - Data type is a resolved data type using a record structure
  - Network structure is a tree structure that consists of a root node, hubs, and endpoints

Universal Serial Bus

- Universal Serial Bus is a new synchronous serial protocol for low to medium speed data transmission
- Full speed signaling 12 Mbs
- Low Speed signaling 1.5 Mbs
- Intended devices are keyboards, mice, joysticks, speakers; other low to medium speed IO devices

PERFORMANCE	APPLICATIONS	ATTRIBUTES
LOW SPEED •Interactive Devices •10-100 Kb/s	Keyboard, Mouse Stylus Game peripherals Virtual Reality peripherals Monitor Configuration	Lower cost Hot plug-unplug Ease of use Multiple peripherals
MEDIUM SPEED •Phone, Audio, Compressed Video 500Kb/s - 10Mbp/s	ISDN PBX POTS Audio	Low cost Ease of use Guaranteed latency Guaranteed Bandwidth Dynamic Attach- Detach Multiple devices
HIGH SPEED •Video, Disk •25-500 Mb/s	Video Disk	High Bandwidth Guaranteed latency Ease of use

Figure 3-1. Application Space Taxonomy

Physical Topology is point-to-point tree.

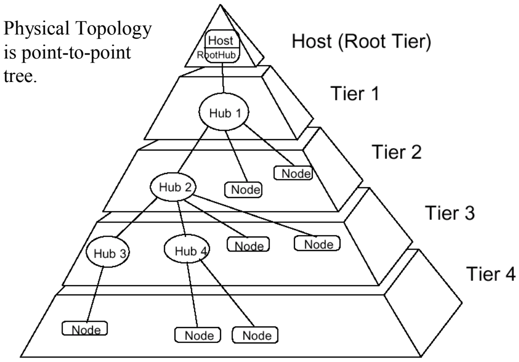


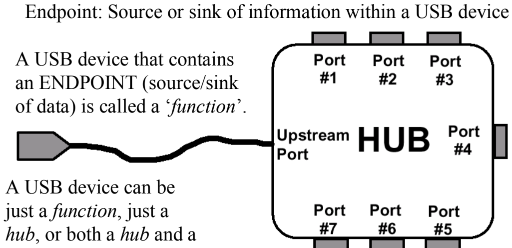
Figure 4-1. Bus Topology

Root: primary contoller

Hub: allows the connection of multiple USB devices

Endpoint: Source or sink of information within a USB device

A USB device that contains an ENDPOINT (source/sink of data) is called a '*function*'.



A USB device can be just a *function*, just a *hub*, or both a *hub* and a *function*.

Figure 4-3. A Typical Hub

Figure 4-4 illustrates how hubs provide connectivity in a desktop computer environment.

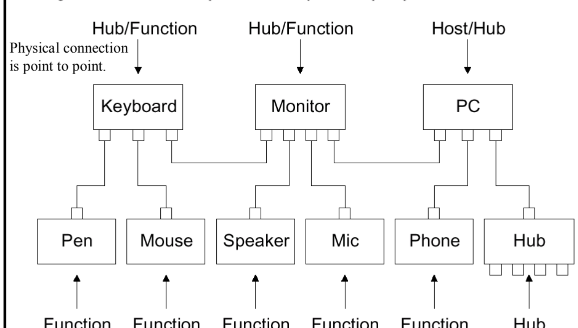


Figure 4-4. Hubs in a Desktop Computer Environment

Physical Interface

Differential Signaling,   Half duplex

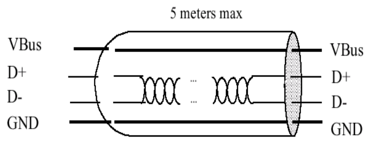


Figure 4-2. USB Cable

Full Duplex: data transmission can occur in both directions at the same time

Half Duplex: data transmission can go in only one direction at a time

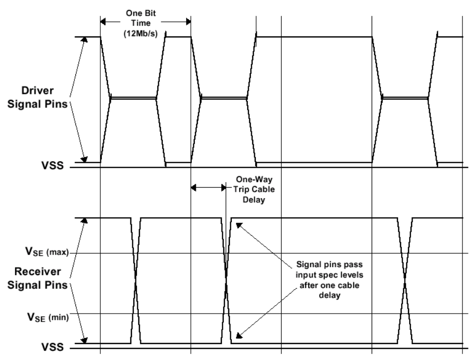


Figure 7-2. Full Speed Driver Signal Waveforms

Table 7-1. Signaling Levels

Bus State	Signaling Levels	
	From Originating Driver	At Receiver
Differential "1"	(D+) - (D-) > 200 mV and D+ or D- > V_{SE} (min.)	
Differential "0"	(D+) - (D-) < -200 mV and D+ or D- > V_{SE} (min.)	

Input Levels:					
Differential Input Sensitivity	VDI	|(D+)-(D-)|, and Figure 7-4	0.2		V
Differential Common Mode Range	VCM	Includes VDI range	0.8	2.5	V
Single Ended Receiver Threshold	VSE		0.8	2.0	V
Output Levels:					
Static Output Low	VOL	RL of 1.5 kΩ to 3.6 V		0.3	V
Static Output High	VOH	RL of 15 kΩ to GND	2.8	3.6	V

Vse = Voltage Single Ended threshold

On disconnect, D+, D- become same voltage value (Vss). Condition is known as a Single-Ended 0.

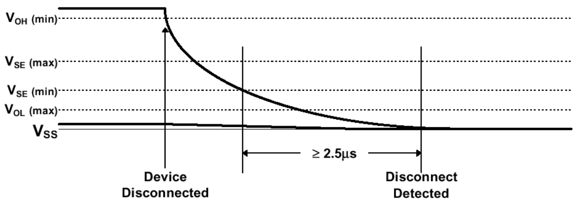


Figure 7-7. Disconnect Detection

On connection of a high-speed device, D+ > D-.
Idle state is D+ > D-, so idle state is a differential '1'.

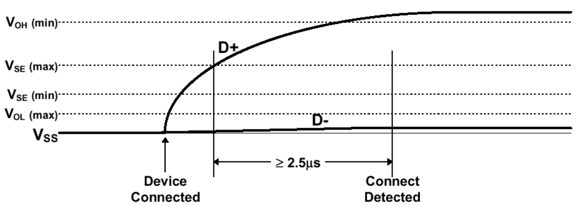


Figure 7-8. Full Speed Device Connect Detection

On connection of a low-speed device, D- > D+.
Idle state is D- > D+, so idle state is a differential '0'.

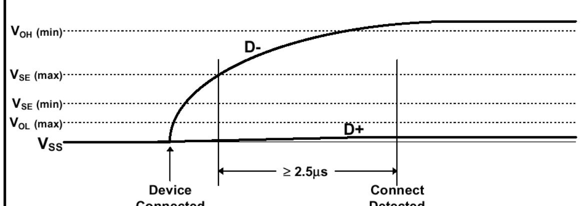
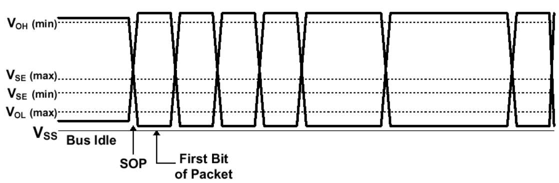


Figure 7-9. Low Speed Device Connect Detection

Data transmission



Idle State: D+, D- outside of range of VSE, at either a differential ‘1’ (high speed) or ‘0’ (low speed).

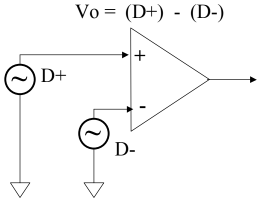
Active State: D+, D- transition to signal a 1 or 0

Data J State:		
Low Speed	Differential “0”	
Full Speed	Differential “1”	
Data K State:		
Low Speed	Differential “1”	
Full Speed	Differential “0”	
Idle State:		
Low Speed	Differential “0” and D- > V_{SE} (max.) and D+ < V_{SE} (min.)	
Full Speed	Differential “1” and D+ > V_{SE} (max.) and D- < V_{SE} (min.)	
Resume State:		
Low Speed	Differential “1” and D+ > V_{SE} (max.) and D- < V_{SE} (min.)	
Full Speed	Differential “0” and D- > V_{SE} (max.) and D+ < V_{SE} (min.)	
Start of Packet (SOP)	Data lines switch from Idle to K State	
End of Packet (EOP)	D+ and D- < V_{SE} (min) for 2 bit times[1] followed by an Idle for 1 bit time	D+ and D- < V_{SE}(min) for ≥ 1 bit time[2] followed by a J State

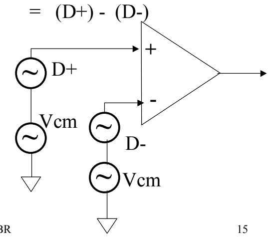
Why differential signaling??

Differential signaling very good at rejecting common-mode noise. If noise is coupled into a cable, then usually it is coupled into all wires in the cable. This ‘common-mode’ noise (Vcm) can be rejected by input amplifier.

Vo = (D+) - (D-)



Vo = (Vcm + D+) - (Vcm+ D-)
= (D+) - (D-)



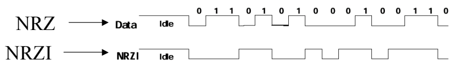


Figure 7-11. NRZI Data Encoding

Non-return to zero (NRZ) - normal data transitions.

NRZ – Inverted (not a good description, is not inverse of NRZ). A transition for every zero bit.

Strings of zeros means lots of transitions. Strings of ‘1’s means steady line.

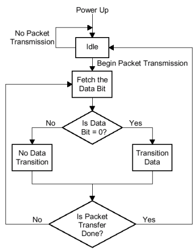


Figure 7-12. Flow Diagram for NRZI

Bit Stuffing – a ‘0’ is inserted after every six consecutive ‘1’s in order to ensure a signal transition so that receiver clock can remain synchronized to the bit stream.

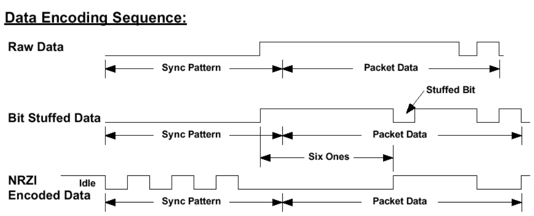


Figure 7-13. Bit Stuffing

Bit stuffing done automatically by sending logic. Sync pattern starts data transmission and is seven ‘0’s followed by a ‘1’.

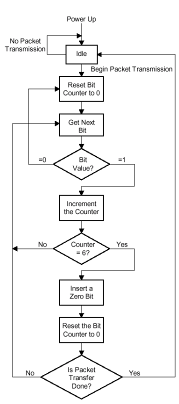


Figure 7-14. Flow Diagram for Bit Stuffing

Receiver/Xmitter logic uses a 48 Mhz internal local clock.

48Mhz/ 12Mbs = 4 clocks per bit time for high speed signaling.

48Mhz/1.5 Mbs = 32 clocks per bit time for low speed signaling.

A guaranteed transition every 7 bit times allows local clock synchronization to the serial data stream. Sync pattern allows clock sync at beginning of packet.

Data Formatting

- Data sent in packets
- Packets will have:
  - Start of Packet Sync Pattern (8 bits, 7 zeros + 1 one)
  - Packet ID (PID) – identifies type of packet. 8 bits total, but only 4 unique bits
  - Address field - 11 bits. 7 bits for USB device (so 128 possible USB devices on bus, host is always address 0), 4 bits for internal use by USB device .
  - Frame number field (11 bits) – incremented by host
  - Data Payload (up to 1023 bytes for high-speed connection)
  - CRC bits - 5 bits for address field, and 16 bits for data field
  - EOP strobe – single ended 0 (160ns-175 ns for high speed, 1.25 us to 1.75 us for high speed)
- Not all packets sent over USB bus have all of these fields (always have SOP, EOP and PID). Packet without data field is a token packet.

Packet Types

Table 8-1. PID Types

PID Type	PID Name	PID[3:0]	Description
Token	OUT	b0001	Address + endpoint number in host -> function transaction
	IN	b1001	Address + endpoint number in function -> host transaction
	SOF	b0101	Start of frame marker and frame number
	SETUP	b1101	Address + endpoint number in host -> function transaction for setup to a control endpoint
Data	DATA0	b0011	Data packet PID even
	DATA1	b1011	Data packet PID odd
Handshake	ACK	b0010	Receiver accepts error free data packet
	NAK	b1010	Rx device cannot accept data or Tx device cannot send data
	STALL	b1110	Endpoint is stalled
Special	PRE	b1100	Host-issued preamble. Enables downstream bus traffic to low speed devices.

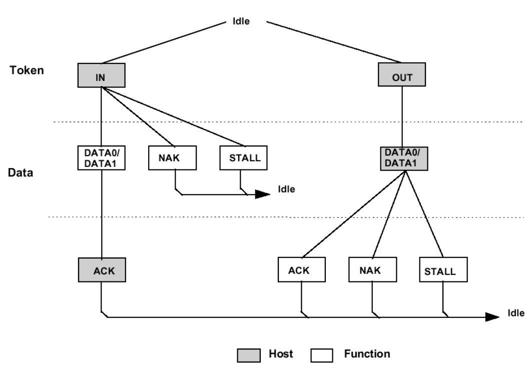


Figure 8-9. Bulk Transaction Format

VHDL Model

- Our VHDL Model will present a VERY abstract view of a USB network
- Models
  - root.vhd -- models the root node
  - hub.vhd -- models a hub, has an upstream port and 2 downstream ports
  - endpoint.vhd – models an endpoint
- Data packet will contain a packet ID, an address (destination), and a payload
  - Payload is an 80 character string
  - Packet types of POUT, PIN, PACK, ERR, NONE

root.vhd

```
library ieee;
use ieee.std_logic_1164.all;
library work;
use work.usbpkg.all;

entity root is
       generic (
         ADDR : natural := 0
       );

       port (
         signal dport : inout pkt
       );
end root;
```

address of root is always '0'.

Only one port on root.

hub.vhd

```
library ieee;
use ieee.std_logic_1164.all;
library work;
use work.usbpkg.all;

entity hub is
       generic (
         HUBDELAY : time := 5 ns
       );

       port (
         signal downstrm_a,downstrm_b : inout pkt;
         signal upstrm : inout pkt
       );
end hub;
```

One upstream port,

two downstream ports

endpoint.vhd

```
library ieee;
use ieee.std_logic_1164.all;
library work;
use work.usbpkg.all;
entity endpoint is
       generic (
         MANUF: string := "FooBar Enterprises";
        ADDR : natural := 0
       );
       port (
        signal dport : inout pkt
       );
end endpoint;
```

Data stored at endpoint

Address of endpoint, cannot be 0.

usbpkg.vhd

Package that defines 'pkt' type

```
PACKAGE usbpkg IS
 constant PTIME: time :=  1 us;  -- packet time
 constant RTIME: time :=  70 ns; -- turn around time
 constant MAXENDPT: natural := 32;  -- maximum # of endpoints
 type ptype is (NONE, POUT, PIN, PACK,ERR);
 type upkt is    RECORD
    id:  ptype;
    dest : integer ;
    data: string(1 to 80);
   END RECORD;
 type upkt_vector is array (natural range <>) of upkt;
 function resolve_upkt (s : upkt_vector) return upkt;
 subtype pkt is resolve_upkt upkt;
END usbpkg;
```

Packet type

Packet destination

Packet payload

Resolved data type

Protocol

- Root initiates all transactions
- Root will either send a PIN or POUT packet with an destination (address) field set
  - At endpoint, if destination field matches endpoint address, process packet else ignore packet
  - if POUT packet, endpoint responds with ACK packet and places local data (initially set to MANUF string) in ACK packet
  - if PIN packet, endpoint copies packet payload ('data' field) into local data, and responds with ACK packet – the data field of this ACK packet is a don't care

Releasing the Line

- To simulate 'releasing' the line, after either a POUT, PIN, or PACK packet is sent, send a packet of type NONE
- A signal between a hub and an endpoint/root will only have 2 drivers
  - To resolve the two drivers, look at the packet type
  - A packet type of NONE resolved with POUT/PIN/PACK will return POUT/PIN/PACK
  - A packet type of NONE resolved with NONE will return NONE
  - A packet type of POUT/PIN/PACK resolved with POUT/PIN/PACK will return a packet of type ERR (this should not happen – if it does, then you have a packet collision which should never happen).
- root_a.vhd illustrates how to send/receive packets

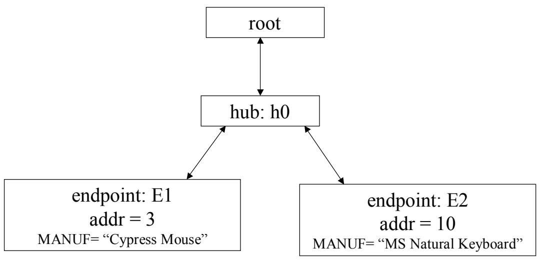
HUB operation

- On downstream ports, any packets of type PACK or NONE should be echoed to upstream port
  - PACK packet can only come from an endpoint
- On upstream port, any packet that is not a PACK packet should be echoed to both downstream ports
- Use HUBDELAY generic for delay time through hub

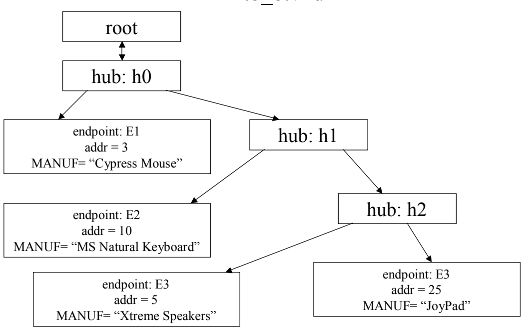
What do you have do?

- Complete the resolution function for *pkt* data type
- Complete architectures for ENDPOINT and HUB
- Test your design with 'tb_a.vhd' and 'tb_b'.vhd
  - I may test your code with other configurations!!!!
- The *root_a.vhd* code does the following:
  - Loops sending POUT packets to addresses 1 to 32. If a PACK is received, know that there is an ENDPOINT at that address
  - Prints out data from ACK packet to screen
  - Sends a PIN packet to the endpoint with the data from the PACK packet modified
  - Send a POUT packet to the endpoint to verify that the endpoint stored the new data - wait for the PACK response and print returned data to console

tb_a.vhd



tb_b.vhd



Sample Run with *tb_b.vhd*

```
> qhsim –lib usb tb_b –c –do “run 150 us;quit”

# Found Endpoint 3, data is Cypress Mouse
# Modifying Endpoint
# Modified Endpoint data: Cypress MouseFOUND
# Found Endpoint 5, data is Xtreme Speakers
# Modifying Endpoint
# Modified Endpoint data: Xtreme SpeakersFOUND
# Found Endpoint 10, data is MS Natural Keyboard
# Modifying Endpoint
# Modified Endpoint data: MS Natural KeyboardFOUND
# Found Endpoint 25, data is JoyPad
# Modifying Endpoint
# Modified Endpoint data: JoyPadFOUND
# Finished Scan for Endpoints
```