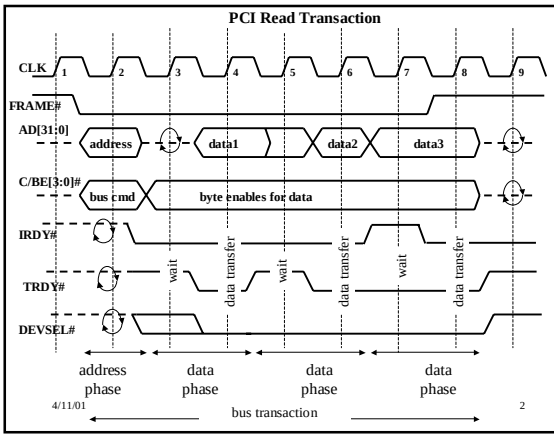
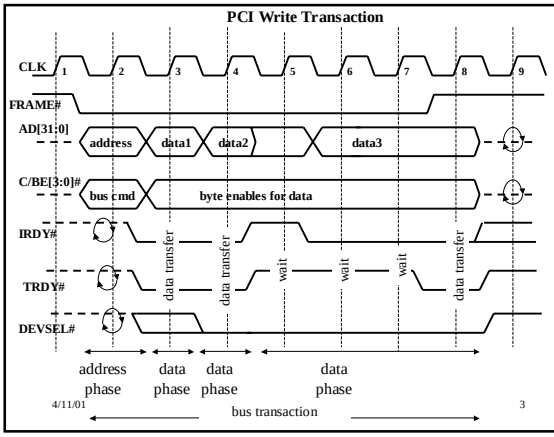
Solving Bus Contention between Multiple Devices

- Central Arbitration - arbiter decides which device gets the bus via Bus Request/Bus Grant pairs
- Single Bus Master – one device (the Root) initiates and controls all transfers (used by USB, IEEE Firewire)
- Time Division Multiplexing (TDM) – give each device a scheduled time period in which to access the bus
- Carrier Sense Multiple Access (CSMA)
  - Used by single-cable Ethernet
  - Each device listens to what it sends - if data is corrupted, then this means that a collision occurred (multiple devices tried to send)
  - On collision, wait a random amount of time ('backoff time'), then try again. On successive collisions, keep increasing the range of backoff time.

PCI Read Transaction



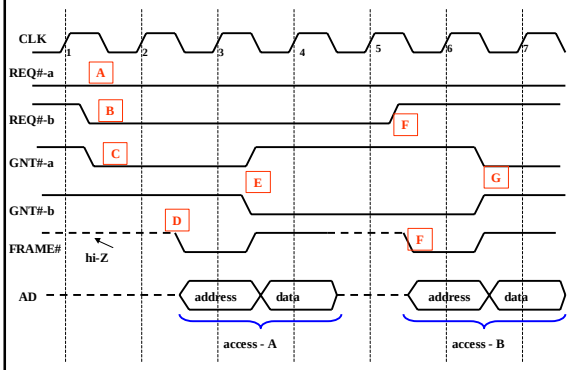
PCI Write Transaction



Read, Write Transactions

- Master drives FRAME#, Bus command, provides address, provides data (write only)
- Target (slave) decodes address, provides data (read), accepts data (write)
  - For READ, *turnaround* cycle needed on AD for target to drive Address/data lines with data
  - For READ, initial latency of data from address is at least 1 cycle (turnaround)
  - For WRITE, data can follow on cycle immediately after address
- Data can be transferred every clock. Wait cycles can be inserted by either Master or Slave
  - IRDY# driven by master to indicate it is ready to provide data
  - TRDY# driven by target to indicate it is ready to provide data
  - Both IRDY# and TRDY# must be asserted to perform a data transfer

PCI Basic Arbitration



Notes on Arbitration

A Device A has its request line asserted, requesting the bus.

B Device B asserts its request line, also requesting the bus.

C Arbiter grants Device A the bus since it had its request line asserted before Device B.

D Device A asserts FRAME#, indicating that it is the current bus master. Device A sends address, data – transaction completed in cycle 4. Device negates FRAME#, releasing bus, but keeps its request asserted, indicating that it desires another transaction.

E Arbiter grants bus to Device B by asserting GNT#-b and negating GNT#-a. Device B must have higher priority than Device A.

F Device B only requires bus for one transaction, so negates its request while asserting FRAME#, indicating it is the bus master – the transaction is completed in cycle 6.

G Arbiter grants Device A the bus by asserting GNT#-a, and negating GNT#-a.

Latency vs. Throughput

- Latency is the time from when an operation is started to when the data is ready
- Throughput is operations per unit time
- Desire low latency and high throughput, but these are usually traded off against one another
- *Arbitration Latency* is time (in clocks) between a master's REQ# assertion and its GNT# assertion
  - depends on number of bus masters, priority scheme, and the length of each master's transactions
- Long transactions by bus masters increase arbitration latency for other bus masters since they have to wait for the transactions to complete
  - Arbiter needs some method of controlling the arbitration latency

PCI Latency Timer

- The Latency Timer is a programmable timer required in each bus master that will limit the amount of time a master can control the bus
  - used by OS to balance bus performance by controlling the arbitration latency

Data Phases	Bytes Transf.	Total Clks	Lat. Timer	Bandwidth (MB/s)	Latency (us)
8	32	16	14	67	0.48
16	64	24	22	89	0.72
32	128	40	38	107	1.20
64	256	72	70	119	2.16

Note that as throughput goes up (better bus utilization), latency increases. Numbers assume initial 8 clock latency from address and a 32-bit bus.

Theoretical Maximum (Peak) Bandwidth

- Peak Bandwidth of PCI is:
  bytes per clock * clk freq = MB/s
- 32 bit bus, 33 Mhz: 4 * 33 Mhz = 132 MB/s
- 64 bit bus, 66 Mhz: 8 * 66 Mhz = 528 MB/s
- PCI cannot achieve peak bandwidth. Many factors contribute to not reaching peak bandwidth
  - turnaround cycles on a bus are "dead" cycles (no data transferred)
  - handoff cycles from one bus master to another bus master
  - address phase of transaction does not transfer data
- Clocks for a PCI transaction is initial target latency (includes time for address phase) + data phases (assume 1 clk/data phase).

PCI Bus Example

Device	Bandwidth (MB/s)	Bytes/10 us	Time Used (us)	# of transactions per slice	Notes
Graphics	50	500	6.15	10	1
LAN	4	40	0.54	1	2
Disk	5	50	0.63	1	3
ISA Bridge	4	40	0.78	2	4
PCI to PCI bridge	9	90	1.17	2	5
Total	72	720	9.27	16	2.16

All transactions assume an 8 clock initial target latency.

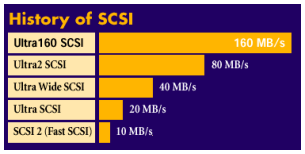
Note that bus utilization is 93%, but only 72/132 = 55% of peak bandwidth is achieved.

PCI Notes

- Supports 5V and 3.3V signaling
  - I/O buffers on board draw power from either 5.0V or 3.3V rail
- On backplane pinout, CLK is bracketed by two Ground pins to shield other signals from noise coupling
- CLK frequency is 33 Mhz or 66 Mhz.
  - Same protocol (all transfers on rising edge of clock)
  - All devices on 66 Mhz PCI bus must operate at 66 Mhz
- Split transactions is not supported on the PCI bus
  - Once the bus is released by the master, the transaction is over with
  - Can't send the address, then release the bus and come back and get the data later – must wait for the data during the same transaction
  - Because of this, devices on the PCI bus must be LOW LATENCY

Increasing Bus Bandwidth

The SCSI standard (Small Computer System Interface) provides a good example on how the bandwidth for a bus standard has been improved over time



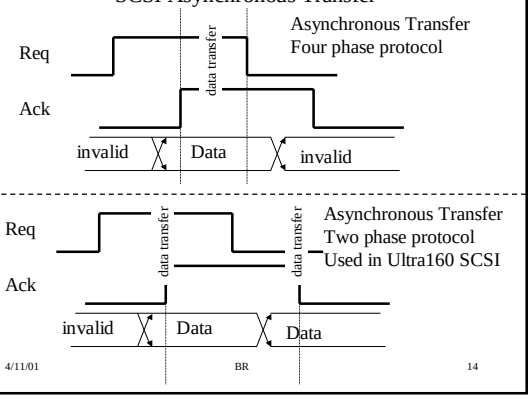
From Adaptec

SCSI Evolution

- SCSI-1 was 8 bits, 5 MB/sec.
- SCSI-2 doubled bandwidth by doubling bus speed
- Ultra SCSI doubled bandwidth by doubling bus speed
- Ultra Wide SCSI doubled bandwidth by increasing data size from 8 bits to 16 bits
- Ultra2 SCSI doubled bandwidth by going to differential signaling with reduced voltage swing
- Ultra160 SCSI doubles bandwidth by transferring data on each transition

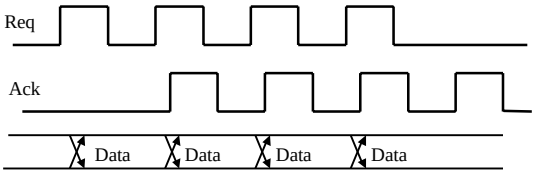
To increase bandwidth: increase bus speed, change physical signaling method, increase data width, transfer data on each clock edge

SCSI Asynchronous Transfer



SCSI Synchronous Transfer

SCSI supports a synchronous transfer mode in which the ACK lags the REQ.

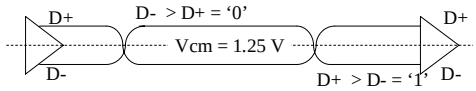


Ack pulses can lag behind Req by several pulses. Same number of Ack pulses must be sent as Req pulses.

Advanced Electrical Signaling
LVDS – Low Voltage Differential Signal (LVDS)

LVDS used on Ultra2 SCSI - differential signaling of 300 mv about Vcm = 1.25 V.

Smaller voltage swing means faster signaling!
Differential voltage signaling also rejects cabling noise.



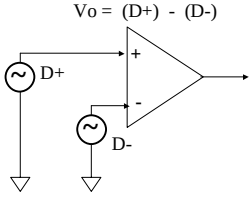
Resistor termination needed to prevent reflections.

Differential signal requires 2X wires!

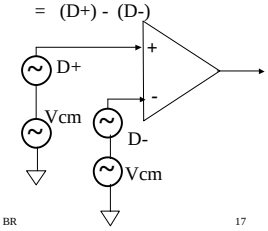
Differential Signaling: Noise Rejection

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-)



Other Standards Use Advanced Signaling Techniques

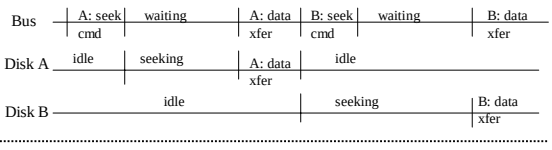
- USB uses differential signaling, but is not low voltage swing
  - Voltage swing can be > 2 volts
  - Lower data rates - peak is 12 Mb/s
- IEEE Firewire uses differential low voltage signaling, but a different standard than LVDS
  - Data rates 100 Mb/s to 400 Mb/s
  - Differential voltage swing < 300 mV
- Rambus uses limited voltage swing signaling (+/- 200 mV) about a fixed voltage reference. Differential pairs NOT used.
  - Voltage Ref = 1.0 V.
  - Limited swing signaling gives high speed signaling (400 Mhz clock, data transfer on each clock edge)
  - Two clock signals used which are complements of each other – data latched on clock crossing.
  - Only used for Processor/Memory buses – distance is very short, so noise coupling is not as much of a problem.

Dealing with Varying Latency: Split Transactions

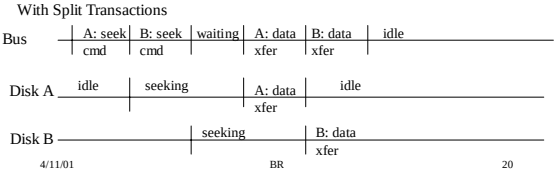
- SCSI is intended for devices with widely varying I/O characteristics.
  - This means the I/O latency can be highly variable
- For high latency devices, do not want to wait for data to become available after initial command.
  - Would like to issue commands to other devices, then come back later and finish the transaction.
- Split Transactions adds parallelism to I/O operations

Split Transaction Example

Without Split Transactions



With Split Transactions



Maintaining Data Synchronization

- When sending data between devices, must delineate data boundaries
- Asynchronous protocols use handshaking signals to do this
- Synchronous protocols use a time reference (a clock signal)
- Methods for specifying the time reference in synchronous transfers
  - The clock signal can be part of the bus specification (i.e. PCI)
  - Data encoding allows clocks at receiver and transmitter to remain synchronized
  - The clock signal is encoded as part of the data

Data Synchronization: USB

USB – Universal Serial Bus – serial data transfer.

Transfer is half duplex using bidirectional differential signaling

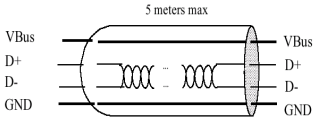


Figure 4-2. USB Cable

Data speeds are 1.5 Mb/s, 12 Mb/s.

Data encoding allows Sending/Receiving clocks to remain synchronized.

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 transistion for every zero bit.

Strings of zeros means lots of transitions. Strings of '1's means steady line.

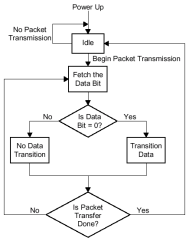


Figure 7-12. Flow Diagram for NRZI

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.

Data Encoding Sequence:

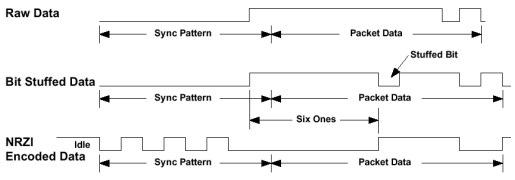


Figure 7-13. Bit Stuffing

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

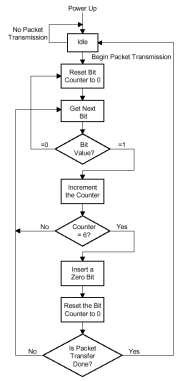


Figure 7-14. Flow Diagram for Bit Stuffing

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 Synchronization: IEEE Firewire

IEEE 1394 (Fire Wire) – serial data transfer.

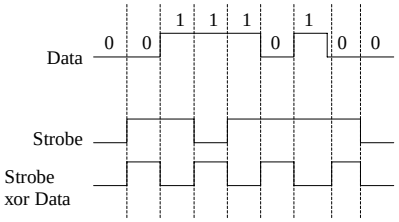
Transfer is half duplex using bi-directional differential signaling

- Signaling uses *Data Strobe* encoding – requires two binary signals to send one bit, each binary signal is represented by a differential pair of signals (D+, D-, S+, S-) Cable also has VDD, GND signals for 6 wires total (USB has 4 wires total).

Data encoding sends the clock encoded in the data signal. The Receiver can *EXTRACT* the clock from the data.

Data Strobe Signaling

Serial Encoding method first used in a multicomputer called the *Transputer*, invented by SGS-Thompson



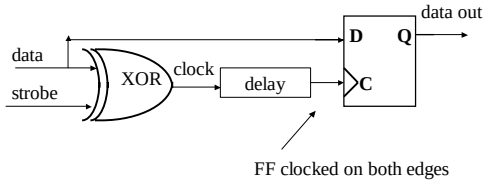
Strobe changes when Data does not.

Extract clock from data and strobe as:

Clock = Data XOR Strobe ; Data clocked on both edges

Extracting the clock from data/strobe, latching the data. Data stream is 'self-clocking'. Can vary speed of data stream and circuit will still work.

No bit stuffing needed.



FF clocked on both edges