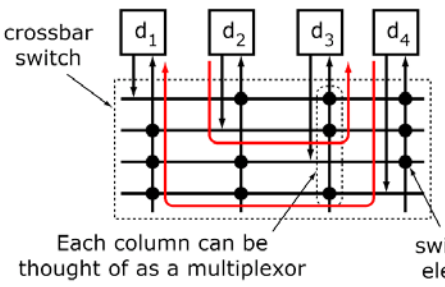
How Do We Connect Multiple Devices?

- A common need in a computer system is for multiple devices to communicate with each other
- Two extremes are a *Crosspoint* Switch and a *Bus*
- A Crosspoint switch is very expensive in terms of hardware
  - Allows multiple devices to communicate at the same time (parallelism)
  - High Data Bandwidth
- A bus is very cheap in terms of hardware, but only two devices can communicate at time

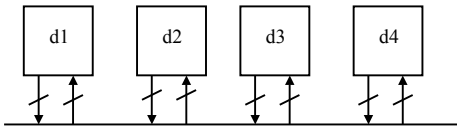
Crossbar Switch



Adding more devices adds more switches (grows as N^2)

Multiple devices can talk at a time.

A Bus



- Only one device can talk at a time
  - All devices on bus can listen (broadcast)
  - If all devices can listen while one device talks, then the connection is called a BUS despite the number of signals used, physical signaling, etc.
- When a device is not talking on the bus, its output signals are disabled (tri-stated)

Bus Classifications

- One way to classify busses is to divide them into System, Backplane, and Peripheral busses
  - This classification is based on what types of devices are connected to the devices
- Another classification is Parallel versus Serial data transfer
- Yet another classification is Asynchronous versus Synchronous
  - Asynchronous – data transfer without a clock

System Bus

- The *System bus* is the bus that connects directly to the pins of the processor. Also known as the processor-memory bus.
  - These days the system bus only has cache memory connected to it and the 'system chipset'
  - Multiple processors may connect to the system bus
  - The system chipset will transfer the data between the system bus and any other busses that are in the system (will act as a *bridge* between the system bus and other busses).
- A System bus is proprietary, and is dependent upon the processor type
  - AMD, Intel have different system busses
  - A Pentium III and a Pentium IV have different system busses

Backplane Bus

- A *backplane bus* is a bus that stays 'inside the box' (cards will plug into bus via slots on motherboard) and forms a high speed path between memory and the peripherals
  - Will be a high bandwidth bus (wide and fast --- parallel bus clocked at high rate)
  - lines run a short distance
  - The PCI (Peripheral Component Interconnect) bus is a local bus
  - System chipset connects the PCI bus to the procesor (system) bus.
- A backplane bus is usually standardized – the PCI bus is found in systems using Intel, AMD processors.

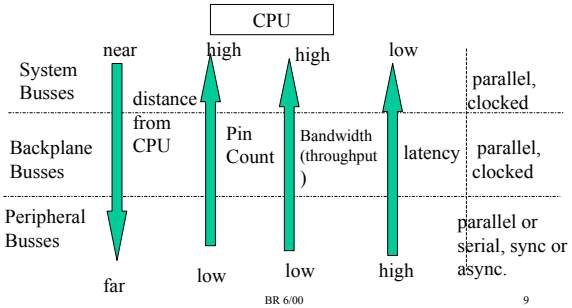
Latency versus Throughput

- *Latency* is the time from when an operation is started to when the data is ready
- *Throughput* is operations per unit time
- Best case is low latency and high throughput, but these are usually traded off against one another for a FIXED bus clock speed.
- Backplane busses are intended to provide low latency first, then as much Throughput as possible
  - When an IO device connected to a backplane bus needs servicing, don't want it to wait very long.
- Achieve low latency by limiting the amount of time one device can hold the bus for transfering data.

Peripheral Busses

- A *Peripheral bus* is a bus that goes 'outside the box' to connect peripherals to memory.
  - Will NOT be as high bandwidth as system bus
  - Will support a wide range of speeds
  - Can be either serial (Firewire, USB) or parallel (SCSI)
  - lines can run a long distance compared to system busses
  - Firewire, USB, SCSI (disks, scanners, CDROMs), EIDE (disks, CDROMs) are peripheral busses.
- Peripheral busses are meant to deal with devices with widely varying latency

Characteristics of System, Backplane, Peripheral Busses



Direct Memory Access (DMA)

- Direct Memory Access (DMA) is when a device other than the processor controls the transfer of data between memory and an IO device
- A DMA controller is specialized logic that is optimized for this task
- A DMA channel is the set of control lines that a DMA controller uses to perform the transfer
  - DMA controller can have multiple channels so that DMA can be performed for multiple devices

DMA (cont).

- At a minimum, a DMA operation needs the channel number (which IO device that requires the DMA), a start address in memory for the transfer, the number of bytes to transfer, and the direction of the transfer (from IO to memory, or vice-versa)
- The processor kicks off the DMA by a write to a control register in the DMA controller
- When DMA is finished, the DMA controller interrupts the processor.
- DMA is more efficient at transferring block data between IO device and memory than the CPU -- also, CPU is freed to perform other tasks.

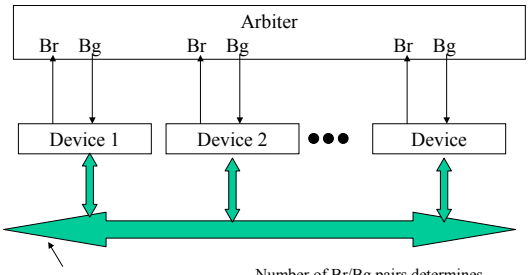
Bus Mastership

- DMA is an example of where the processor turns control of the bus over to another device
- When a device has control of a bus, it is known as the *Bus Master*
- In early system busses, only the processor and DMA controller could have ownership of the bus (become Bus Master)
- Modern system busses (e.g. PCI) allow any IO device to become bus master – an IO device that can become bus master can performs DMA to memory itself instead of the DMA controller
  - If all IO cards can become bus master, then don't need a separate DMA controller.

Bus Arbitration

- A piece of control logic known as the 'arbiter' must decide which device gets ownership of the bus. This control logic resides in the system *chipset*.
- Two lines are used for each peripheral that can assume bus mastership
  - Bus Request – used by peripheral to ask for control of a bus (input to arbiter)
  - Bus Grant – used by arbiter to grant the bus to the peripheral (output to arbiter)
- A peripheral typically has control of the bus for maximum number number of clock cycles (typically 32) before it has to release control of the bus back to the arbiter.

Bus Arbiter



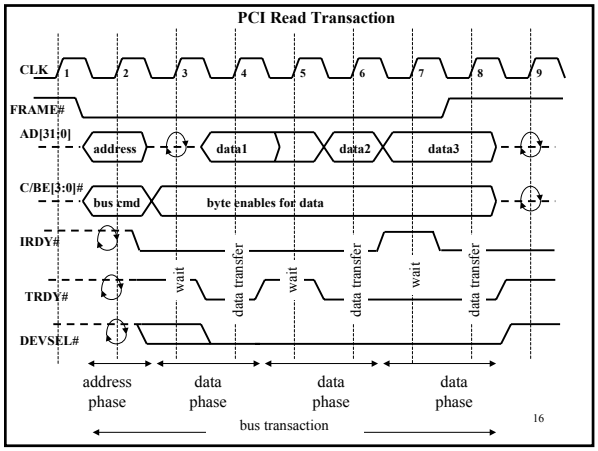
Backplane Bus, e.g PCI

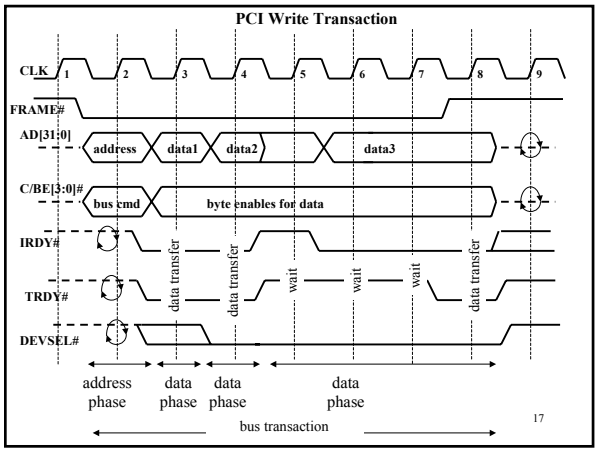
Br: Bus Request, Bg: Bus Grant

Number of Br/Bg pairs determines maximum number of devices on bus (i.e. 'slots' on bus)

Bus Signals

- Signals on System Busses, Backplane busses can be divided into three categories
  - Address, Data, Control
- Address, Data pins can be multiplexed to reduce pin count
  - Multiplexing address/data usually reduces bandwidth
- Typical Control Signals
  - Clock
  - Interrupt Request/Acknowledge
  - Bus Arbitration
  - Transfer control (Read/Write, Command bus that identifies type of transfer, burst mode, single word, etc).

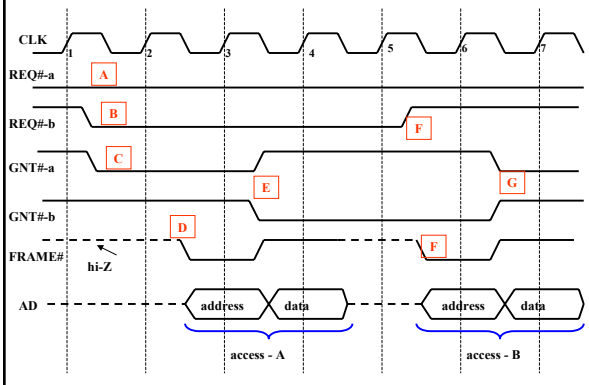




Read, Write Transactions

- 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



hi-Z BR 6/00 active drive 19

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.

BR 6/00 20

Theoretical Maximum (Peak) Bandwidth

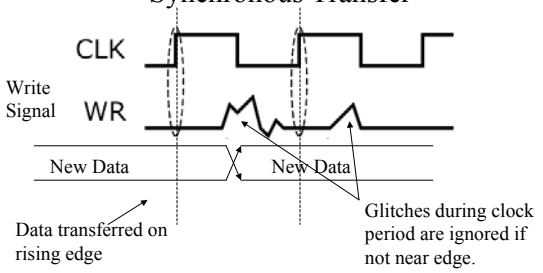
- Peak Bandwidth of PCI is:
  bytes per clock * clk freq = MB/s
- 32 bit bus, 33 Mhz: 4 * 33 Mhz = 132 MB/s
- 64 bit bus, 66 Mhz: 8 * 66 Mhz = 528 MB/s
- PCI cannot achieve peak bandwidth. Many factors contribute to not reaching peak bandwidth
  - turnaround cycles on a bus are “dead” cycles (no data 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).

BR 6/00 21

Synchronous vs. Asynchronous

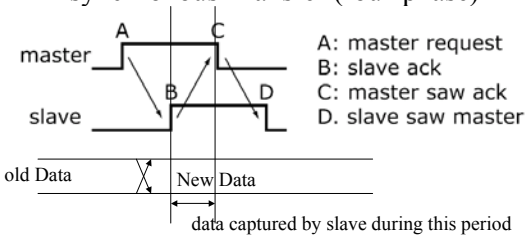
- Synchronous Transfer
  - Clock signal included in bus. Transfer can occur on a single edge (either rising or falling) or on both edges (double-pumped)
  - Clock encoded with data (i.e, Data strobe signaling via IEEE Firewire)
- Asynchronous Transfer
  - Handshaking lines controls data transfer
- Synchronous bus usually higher bandwidth than asynchronous bus

Synchronous Transfer



A problem with clocked busses is that the slowest device sets the clock frequency! All devices must be able to communicate at the bus speed.

Asynchronous Transfer (four-phase)

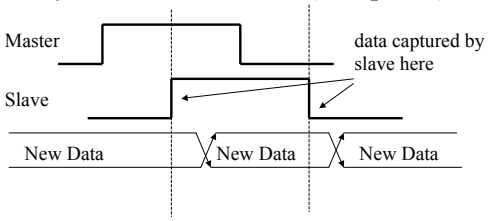


Different speed devices easily supported.

Noise on handshaking lines can be a problem

The above protocol is called "four-phase" handshaking. All signals start a '0' and end at '0' after data transfer.

Asynchronous Transfer (two-phase)



Two-phase transfer – data transfer occurs on each transition of handshaking signals.

Methods for Increasing Bus Transfer Rates

- Make Data bus wider
  - Increases pin count, cost
  - Increases noise – more signal transitions, more current draw, more noise
- Increase clock speed
  - Not all devices on bus may be able to talk at new clock rate
- Clock data on both edges
  - Good solution for bus standards that already have a fixed clock rate – need a configuration line that determines if this is a single or double edge clocked device

Methods for Increasing Bus Transfer Rates (cont).

- Split Transactions
  - A data transaction is composed of two parts: sending the address, then sending/receiving the data
  - Once an address is received, the I/O device may require some time to locate the data
  - A split transaction allows the address to be sent over the bus, then transactions to other I/O devices can be done while the first device is locating the data. The transaction is finished at a later time when the I/O device has retrieved the data.
  - Split transactions help eliminate waiting time
- Burst mode transfers – more efficient than single mode transfers
- Advanced electrical signaling
  - RAMBUS limited voltage swing signaling