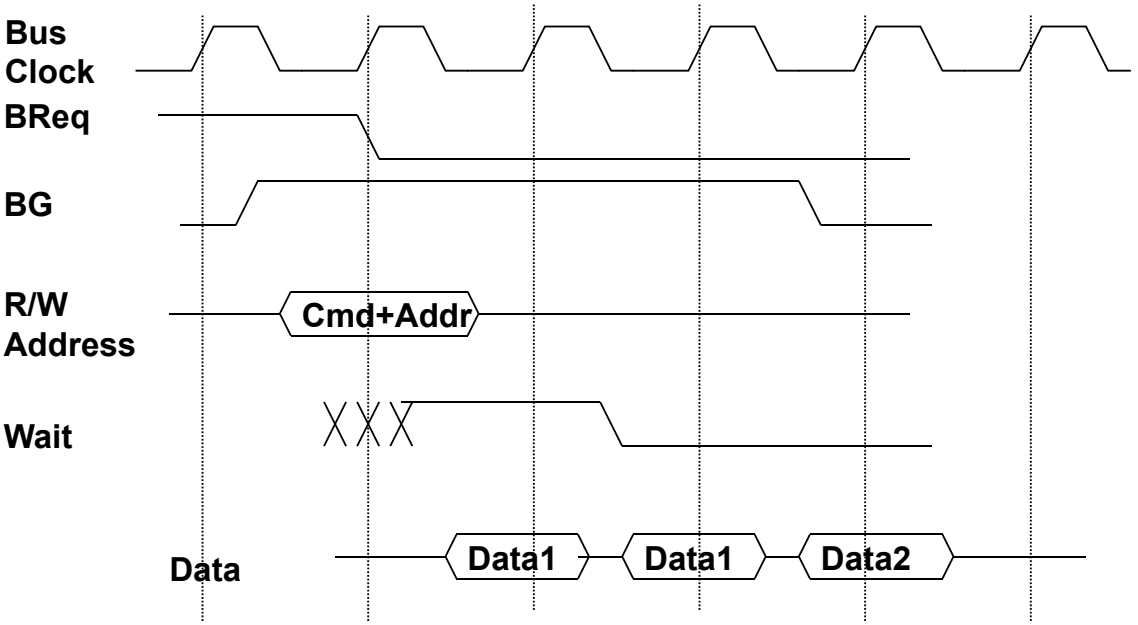
Peripherals (2/2)

Synchronous and Asynchronous Bus

- **Synchronous Bus:**
  - **Includes a clock in the control lines**
  - **All agents operate synchronously to the bus clock**
  - **A fixed protocol for communication that is relative to the clock**
  - **Advantage:**
    - Involves very little logic and can run very fast
  - **Disadvantages:**
    - Every device on the bus must run at the same clock rate
    - To avoid clock skew, they cannot be long if they are fast
- **Asynchronous Bus:**
  - **It is not clocked**
  - **It can accommodate a wide range of devices**
  - **It can be lengthened without worrying about clock skew**
  - **It requires a handshaking protocol**

Typical Synchronous Protocol



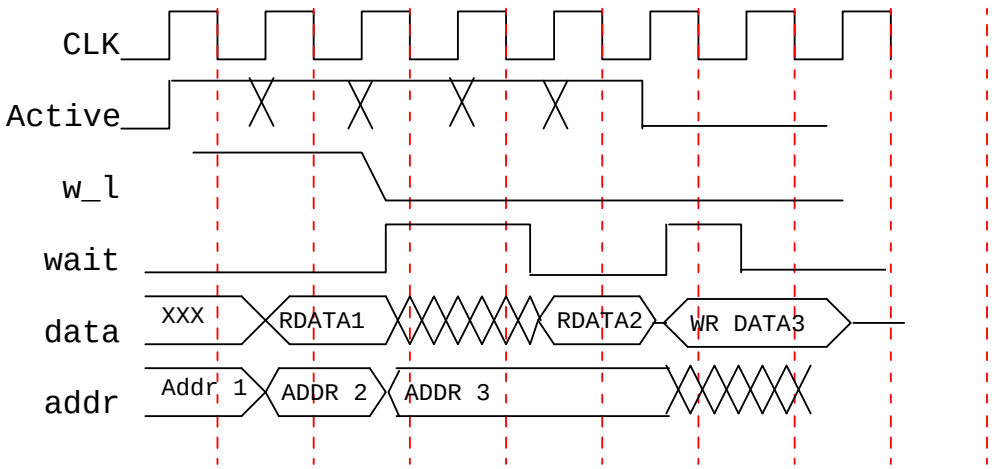
- Slave indicates when it is prepared for data xfer
- Actual transfer goes at bus rate

Increasing the Bus Bandwidth

- Separate versus multiplexed address and data lines:
  - Address and data can be transmitted in one bus cycle if separate address and data lines are available
  - Cost: (a) more bus lines, (b) increased complexity
- Data bus width:
  - By increasing the width of the data bus, transfers of multiple words require fewer bus cycles
  - Example: SPARCstation 20's memory bus is 128 bit wide
  - Cost: more bus lines
- Block transfers:
  - Allow the bus to transfer multiple words in back-to-back bus cycles
  - Only one address needs to be sent at the beginning
  - The bus is not released until the last word is transferred
  - Cost: (a) increased complexity
    (b) decreased response time for request

Pipelined Bus Protocols

- Attempt to initiate next address phase during current data phase

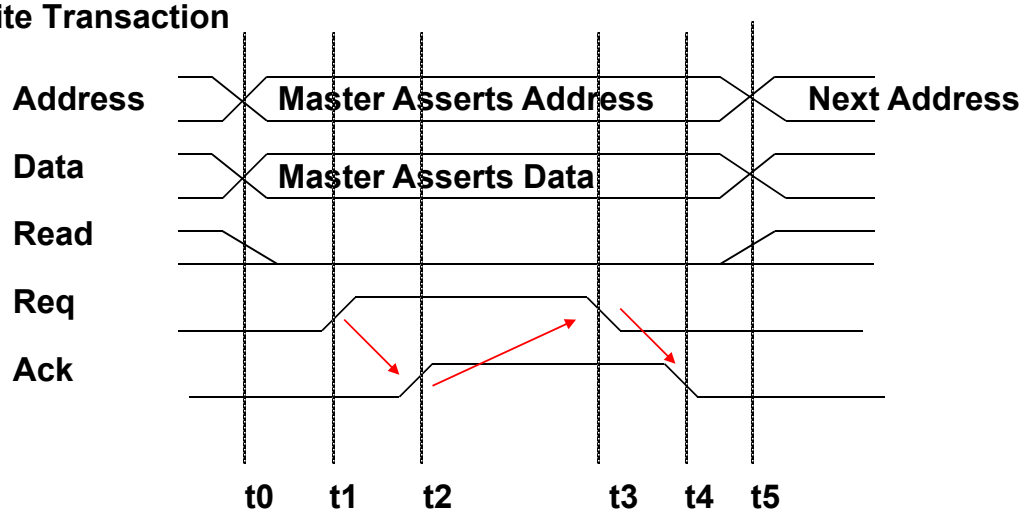


Increasing Transaction Rate on Multimaster Bus

- Overlapped arbitration
  - perform arbitration for next transaction during current transaction
- Bus parking
  - master can holds onto bus and performs multiple transactions as long as no other master makes request
- Overlapped address / data phases (prev. slide)
  - requires one of the above techniques
- Split-phase (or packet switched) bus
  - completely separate address and data phases
  - arbitrate separately for each
  - address phase yield a tag which is matched with data phase
- ”All of the above” in most modern mem busses

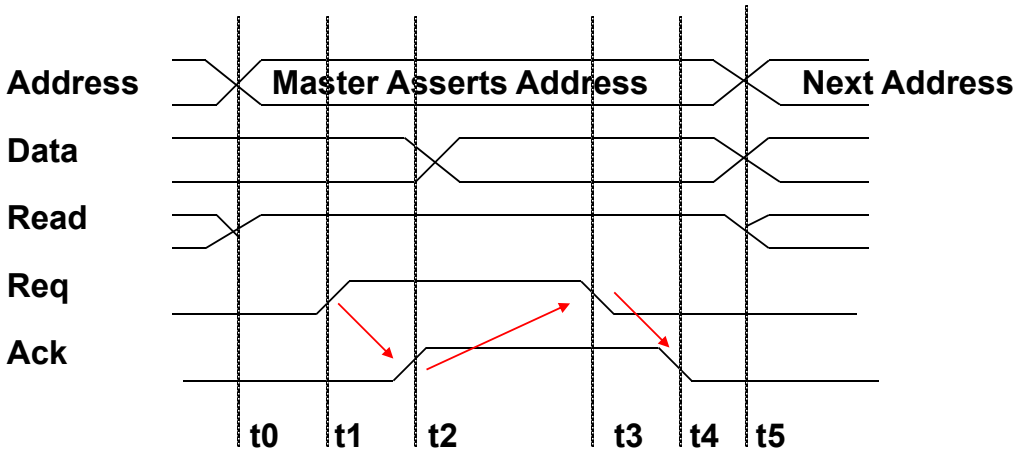
Asynchronous Handshake

- Write Transaction



- t0 : Master has obtained control and asserts address, direction, data
- Waits a specified amount of time for slaves to decode target
- t1: Master asserts request line
- t2: Slave asserts ack, indicating data received
- t3: Master releases req
- t4: Slave releases ack

Read Transaction

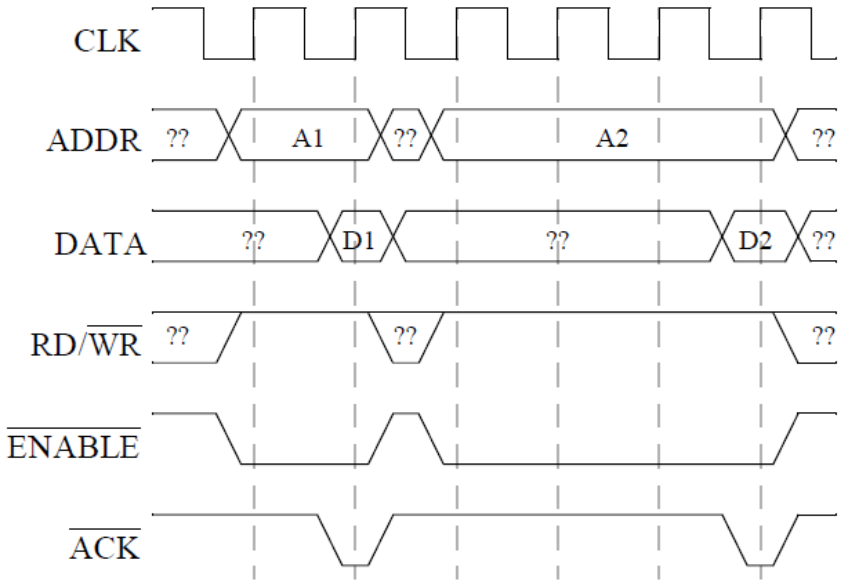


- t0 : Master has obtained control and asserts address, direction, data
- Waits a specified amount of time for slaves to decode target\
- t1: Master asserts request line
- t2: Slave asserts ack, indicating ready to transmit data
- t3: Master releases req, data received
- t4: Slave releases ack

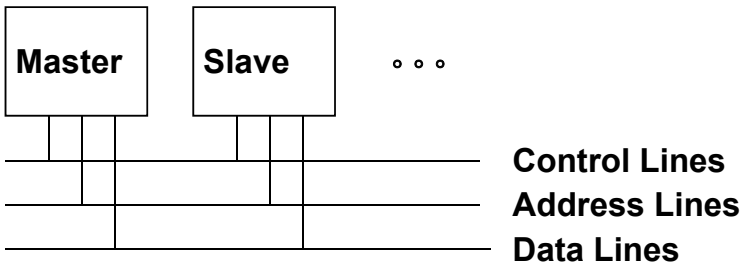
Semi-Synchronous Bus

- Everything synchronized to bus clock, but transactions take variable number of cycles
- Slave asserts ACK to indicate that data is valid at clock edge (must be synchronized to clock)
- Control signal edges convey no timing information
- Minimum transaction usually multiple cycles
- Extra clock cycles between start and end (due to late ACK) are called wait states
  - Variation: replace ACK with WAIT; assume minimum transaction time unless device asserts WAIT
- ACK or WAIT can be driven by:
  - device itself (if it provides such a thing)
  - "wait state generator": dedicated counter/FSM logic, delays ENABLE by fixed number of cycles to generate ACK (based on device response time)

Simi-Synchronous Bus Example



Busses so far



- Bus Master: has ability to control the bus, initiates transaction
- Bus Slave: module activated by the transaction
- Bus Communication Protocol: specification of sequence of events and timing requirements in transferring information.

- Bus Transaction
  - Arbitration
  - Request
  - Action

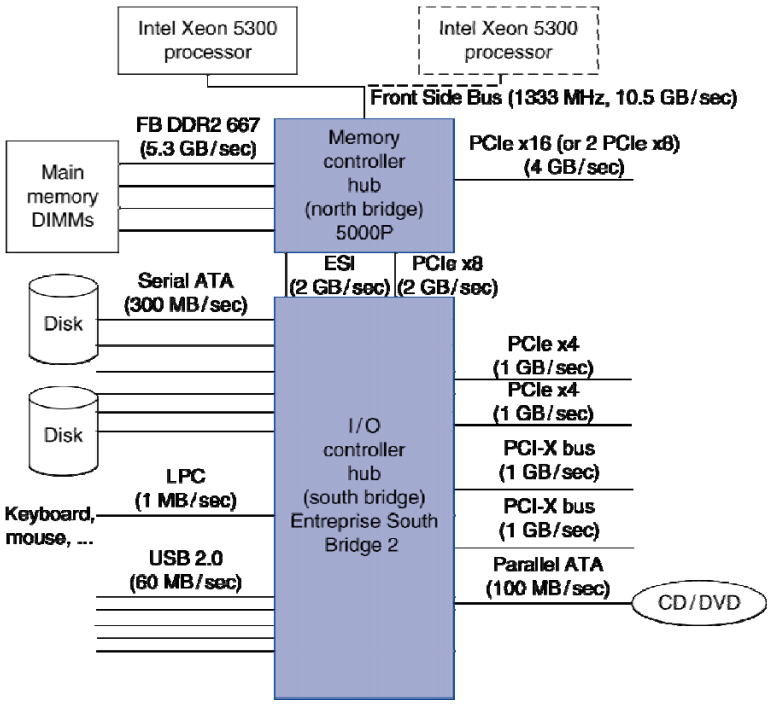
Summary of Buses

- Shared communication link (one or more wires)
- Difficult design:
  - — may be bottleneck
  - — length of the bus
  - — number of devices
  - — tradeoffs (buffers for higher bandwidth increases latency)
  - — support for many different devices
  - — cost
- Types of buses:
  - — processor-memory (short high speed, custom design)
  - — backplane (high speed, often standardized, e.g., PCI)
  - — I/O (lengthy, different devices, standardized, e.g., SCSI)
- Synchronous vs. Asynchronous
  - — use a clock and a synchronous protocol, fast and small
    but every device must operate at same rate and
    clock skew requires the bus to be short
  - — don't use a clock and instead use handshaking

I/O Bus Examples

	Firewire	USB 2.0	PCI Express	Serial ATA	Serial Attached SCSI
Intended use	External	External	Internal	Internal	External
Devices per channel	63	127	1	1	4
Data width	4	2	2/lane	4	4
Peak bandwidth	50MB/s or 100MB/s	0.2MB/s, 1.5MB/s, or 60MB/s	250MB/s/lane 1×, 2×, 4×, 8×, 16×, 32×	300MB/s	300MB/s
Hot pluggable	Yes	Yes	Depends	Yes	Yes
Max length	4.5m	5m	0.5m	1m	8m
Standard	IEEE 1394	USB Implementers Forum	PCI-SIG	SATA-IO	INCITS TC T10

Typical x86 PC I/O System



I/O Management

- **I/O is mediated by the OS**
  - **Multiple programs share I/O resources**
    - Need protection and scheduling
  - **I/O causes asynchronous interrupts**
    - Same mechanism as exceptions
  - **I/O programming is fiddly**
    - OS provides abstractions to programs

I/O Commands

- **I/O devices are managed by I/O controller hardware**
  - **Transfers data to/from device**
  - **Synchronizes operations with software**
- **Command registers**
  - **Cause device to do something**
- **Status registers**
  - **Indicate what the device is doing and occurrence of errors**
- **Data registers**
  - **Write: transfer data to a device**
  - **Read: transfer data from a device**

Communicating with an I/O device

- The CPU sends commands to the I/O device over the memory-I/O bus and reads status back
- Approach 1: memory-mapped I/O
  - A portion of the memory address space is reserved for sending commands (as store instructions) and reading status (as load instructions)
  - The reserved pages are designated as non-cacheable so that they bypass the caches
  - The memory addresses are ignored by the main memory
  - Particular addresses are reserved for particular devices
  - User programs cannot use these instructions, only the OS
  - Used by MIPS and many others
- Approach 2: special privileged I/O instructions are defined in the ISA
  - Separate instructions to access I/O registers
  - Can only be executed in kernel mode
  - Used by X86

Polling

- Periodically check I/O status register
  - If device ready, do operation
  - If error, take action
- Common in small or low-performance real-time embedded systems
  - Predictable timing
  - Low hardware cost
- In other systems, wastes CPU time

Interrupts

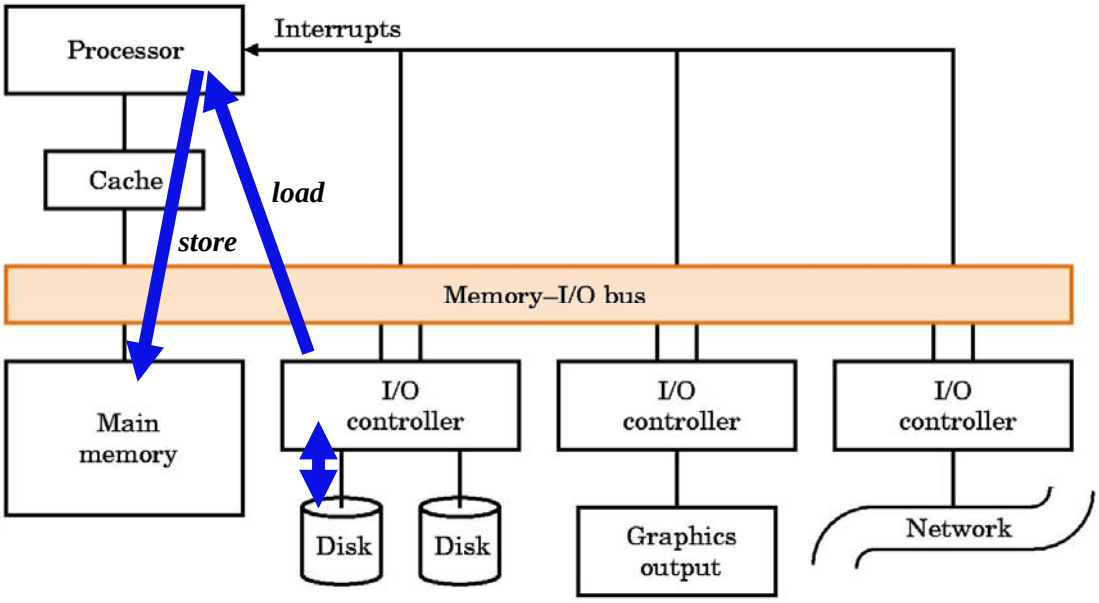
- When a device is ready or error occurs
  - Controller interrupts CPU
- Interrupt is like an exception
  - But not synchronized to instruction execution
  - Can invoke handler between instructions
  - Cause information often identifies the interrupting device
- Priority interrupts
  - Devices needing more urgent attention get higher priority
  - Can interrupt handler for a lower priority interrupt

I/O Data Transfer

- Polling and interrupt-driven I/O
  - CPU transfers data between memory and I/O data registers
  - Time consuming for high-speed devices
- Direct memory access (DMA)
  - OS provides starting address in memory
  - I/O controller transfers to/from memory autonomously
  - Controller interrupts on completion or error

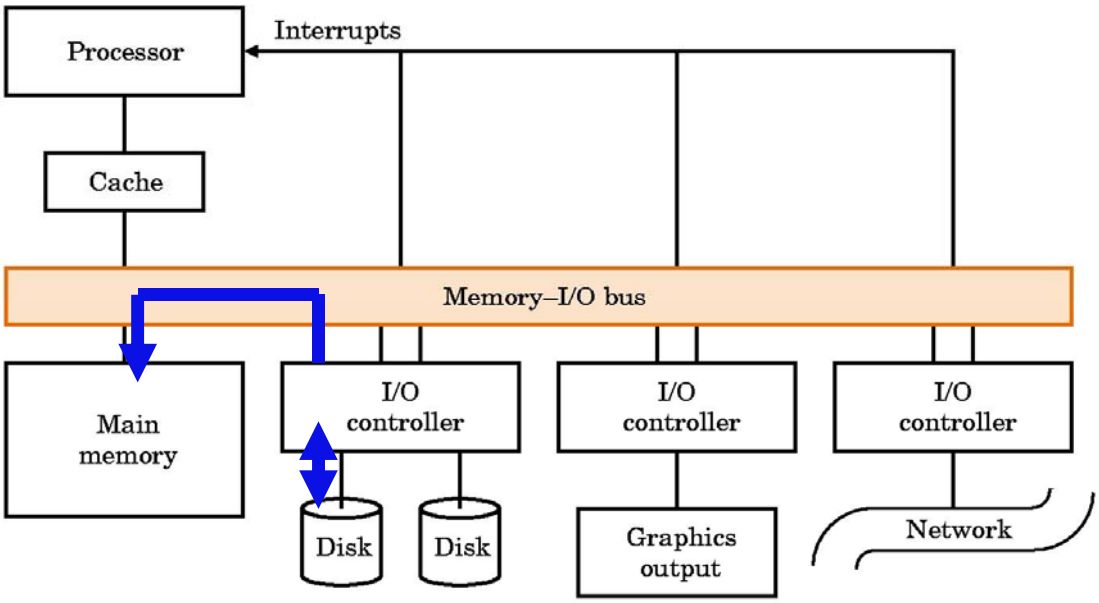
Data transfer between I/O and memory

- Simplest approach is for the CPU to perform the data transfer through load and store operations (assuming memory-mapped I/O)
  - CPU determines either through polling or interrupts that the transfer is done and another can occur



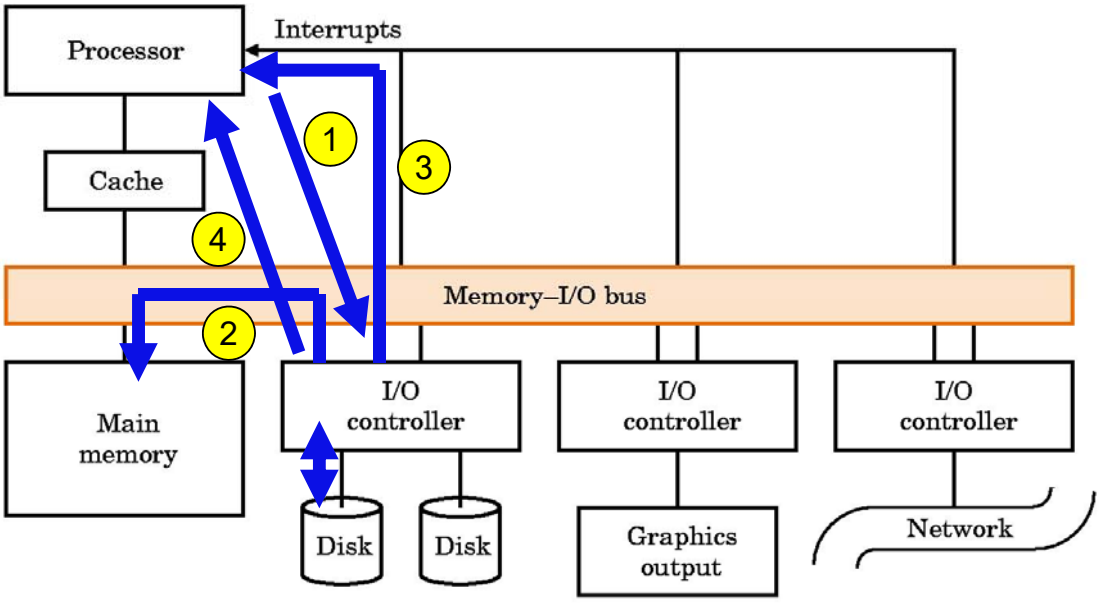
Data transfer between I/O and memory

- A common higher performance alternative is *direct memory access (DMA)* in which the device directly transfers data to/from memory without CPU intervention



DMA operation

- **Processor provides to the DMA device the number of bytes to transfer, the memory address, and the operation to be performed**



Modes of DMA operations

- **Burst Mode**
  - Once the DMA controller is granted access to the system bus by the CPU, it transfers all bytes of data in the data block before releasing control of the system buses back to the CPU
- **Cycle Stealing Mode**
  - In the cycle stealing mode, the DMA controller obtains access to the system bus the same way as in burst mode. However, in cycle stealing mode, after one byte of data transfer, the control of the system bus is de-asserted to the CPU via BG.
- **Transparent Mode**
  - The DMA controller only transfers data when the CPU is performing operations that do not use the system buses

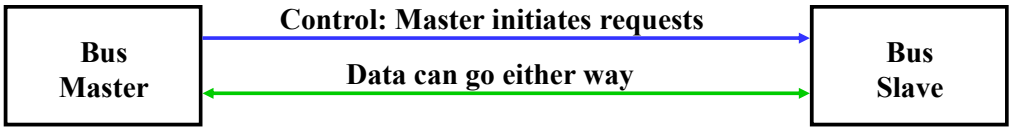
DMA Interaction W/ Other Components

- **DMA/Cache Interaction**
  - **If DMA writes to a memory block that is cached**
    - Cached copy becomes stale
  - **If write-back cache has dirty block, and DMA reads memory block**
    - Reads stale data
  - **Need to ensure cache coherence**
    - Flush blocks from cache if they will be used for DMA
    - Or use non-cacheable memory locations for I/O
- **DMA/VM Interaction**
  - **OS uses virtual addresses for memory**
    - DMA blocks may not be contiguous in physical memory
  - **Should DMA use virtual addresses?**
    - Would require controller to do translation
  - **If DMA uses physical addresses**
    - May need to break transfers into page-sized chunks
    - Or chain multiple transfers
    - Or allocate contiguous physical pages for DMA

Other important issues

- **Bus Arbitration:**
  - **— daisy chain arbitration (not very fair)**
  - **— centralized arbitration (requires an arbiter), e.g., PCI**
  - **— self selection (many req lines), e.g., NuBus used in Macintosh**
  - **— collision detection, e.g., Ethernet**
- **Operating system:**
  - **— polling**
  - **— interrupts**
  - **— DMA**
- **Performance Analysis techniques:**
  - **— queuing theory**
  - **— simulation**
  - **— analysis, i.e., find the weakest link (see "I/O System Design")**
- **Many new developments**

Arbitration: Obtaining Access to the Bus

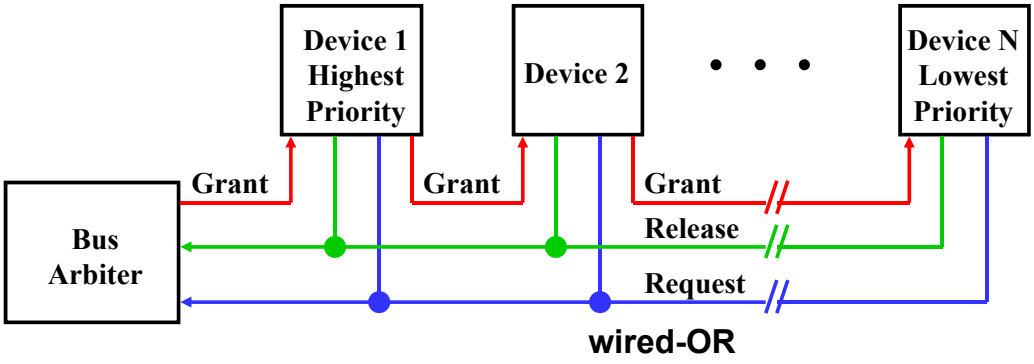


- One of the most important issues in bus design:
  - How is the bus reserved by a devices that wishes to use it?
- Chaos is avoided by a master-slave arrangement:
  - Only the bus master can control access to the bus:
    It initiates and controls all bus requests
  - A slave responds to read and write requests
- The simplest system:
  - Processor is the only bus master
  - All bus requests must be controlled by the processor
  - Major drawback: the processor is involved in every transaction

Multiple Potential Bus Masters: Arbitration

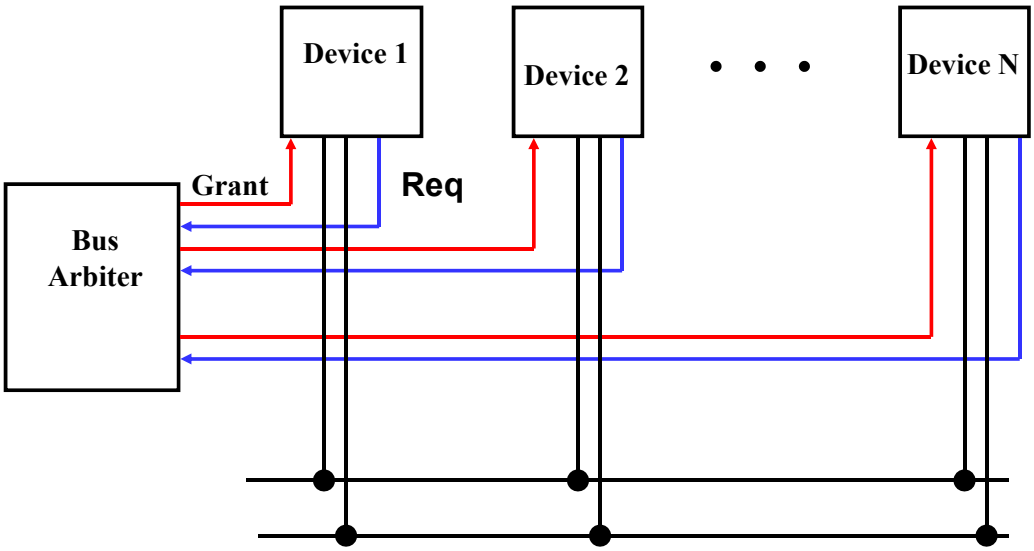
- Bus arbitration scheme:
  - A bus master wanting to use the bus asserts the bus request
  - A bus master cannot use the bus until its request is granted
  - A bus master must signal to the arbiter after finish using the bus
- Bus arbitration schemes usually try to balance two factors:
  - Bus priority: the highest priority device should be serviced first
  - Fairness: Even the lowest priority device should never be completely locked out from the bus
- Bus arbitration schemes can be divided into four broad classes:
  - Daisy chain arbitration: single device with all request lines.
  - Centralized, parallel arbitration: see next-next slide
  - Distributed arbitration by self-selection: each device wanting the bus places a code indicating its identity on the bus, and competes the bus. (ex. NuBus used in Macintosh)
  - Distributed arbitration by collision detection: Ethernet uses this.

The Daisy Chain Bus Arbitrations Scheme



- Advantage: simple
- Disadvantages:
  - Cannot assure fairness:
    A low-priority device may be locked out indefinitely
  - The use of the daisy chain grant signal also limits the bus speed

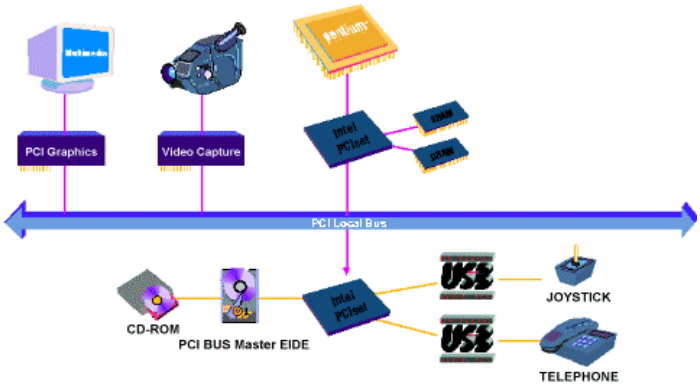
Centralized Parallel Arbitration



- Used in essentially all processor-memory busses and in high-speed I/O busses

High Speed I/O Bus

- Examples
  - graphics
  - fast networks
- Limited number of devices
- Data transfer bursts at full rate
- DMA transfers important
  - small controller spools stream of bytes to or from memory
- Either side may need to squelch transfer
  - buffers fill up



PCI Read/Write Transactions

- All signals sampled on rising edge
- Centralized Parallel Arbitration
  - overlapped with previous transaction
- All transfers are (unlimited) bursts
- Address phase starts by asserting FRAME#
- Next cycle “initiator” asserts cmd and address
- Data transfers happen on when
  - IRDY# asserted by master when ready to transfer data
  - TRDY# asserted by target when ready to transfer data
  - transfer when both asserted on rising edge
- FRAME# deasserted when master intends to complete only one more data transfer

PCI Optimizations

- Push bus efficiency toward 100% under common simple usage
- Bus Parking
  - retain bus grant for previous master until another makes request
  - granted master can start next transfer without arbitration
- Arbitrary Burst length
  - intiator and target can exert flow control with xRDY
  - target can disconnect request with STOP (abort or retry)
  - master can disconnect by deasserting FRAME
  - arbiter can disconnect by deasserting GNT
- Delayed (pended, split-phase) transactions
  - free the bus after request to slow device
- Other issues
  - Interrupts: support for controlling I/O devices
  - Cache coherency to support for I/O and multiprocessors
  - Locks: support timesharing, I/O, and MPs
  - Configuration Address Space

Summary of Bus Options

Option	*High performance*	*Low cost*
Bus width	Separate address & data lines	Multiplex address & data lines
Data width	Wider is faster (e.g., 32 bits)	Narrower is cheaper (e.g., 8 bits)
Transfer size	Multiple words has less bus overhead	Single-word transfer is simpler
Bus masters	Multiple (requires arbitration)	Single master (no arbitration)
Clocking	Synchronous	Asynchronous
Protocol	Pipelined	Serial

The OS and I/O

- **User programs request I/O services through *system call* exceptions to the OS**
- **The OS provides the low-level software routines (*drivers*) that send *commands* to the I/O device and read *status* back**
- **Multiple programs need to share I/O services and need them to be protected from each other**

I/O vs. CPU Performance

- **Amdahl's Law**
  - **Don't neglect I/O performance as parallelism increases compute performance**
- **Example**
  - **Benchmark takes 90s CPU time, 10s I/O time**
  - **Double the number of CPUs/2 years**
    - I/O unchanged

Year	CPU time	I/O time	Elapsed time	% I/O time
now	90s	10s	100s	10%
+2	45s	10s	55s	18%
+4	23s	10s	33s	31%
+6	11s	10s	21s	47%