Bus Types

- Bus lines can be separated into two generic types: **dedicated** and **multiplexed**.
- A dedicated bus line is permanently assigned either to one function or to a physical subset of computer components.
  - —An example of functional dedication is the use of separate dedicated address and data lines, which is common on many buses.
- In a multiplexed bus, address and data information may be transmitted over the same set of lines using an *Address Valid* control line.

Bus Types

- At the beginning of a data transfer, the address is placed on the bus and the *Address Valid* line is activated.
- At this point, each module has a specified period of time to copy the address and determine if it is the addressed module. The address is then removed from the bus, and the same bus connections are used for the subsequent read or write data transfer.
- This method of using the same lines for multiple purposes is known as *time multiplexing.*
- The **advantage** of time multiplexing is the use of fewer lines, which saves space and, usually, cost.
- The **disadvantage** is that more complex circuitry is needed within each module. Also, there is a potential reduction in performance because certain events that share the same lines cannot take place in parallel.

Bus Arbitration

- In all but the simplest systems, more than one module may need control of the bus
  - —For example, an I/O module may need to read or write directly to memory, without sending the data to the processor.
- Because only one unit at a time can successfully transmit over the bus, some method of arbitration is needed.
- The various methods can be roughly classified as being either **centralized** or **distributed**.

Bus Arbitration

- In a centralized scheme, a single hardware device, referred to as a *bus controller or arbiter, is responsible for allocating time on the bus.*
- *The device may be* a separate module or part of the processor.
- In a distributed scheme, there is no central controller. Rather, each module contains access control logic and the modules act together to share the bus.

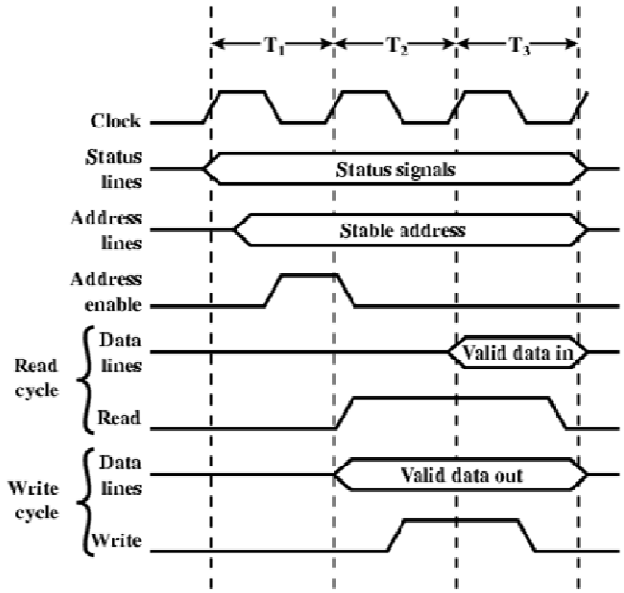
Bus Arbitration

- With both methods of arbitration, the purpose is to designate one device, either the processor or an I/O module, as master.
- The **master** may then initiate a data transfer (e.g., read or write) with some other device, which acts as **slave** for this particular exchange.

Timing

- *Timing refers to the way in which events are coordinated on the bus.*
- *Buses* use either **synchronous** timing or **asynchronous** timing.
- With synchronous timing, the occurrence of events on the bus is determined by a clock. The bus includes a clock line upon which a clock transmits a regular sequence of alternating 1s and 0s of equal duration.
- A single 1–0 transmission is referred to as a *clock cycle or bus cycle and defines a time slot.*
- *All other devices on* the bus can read the clock line, and all events start at the beginning of a clock cycle. The figure shows a typical, but simplified, timing diagram for synchronous read and write operations.
- Other bus signals may change at the leading edge of the clock signal (with a slight reaction delay).
- Most events occupy a single clock cycle.

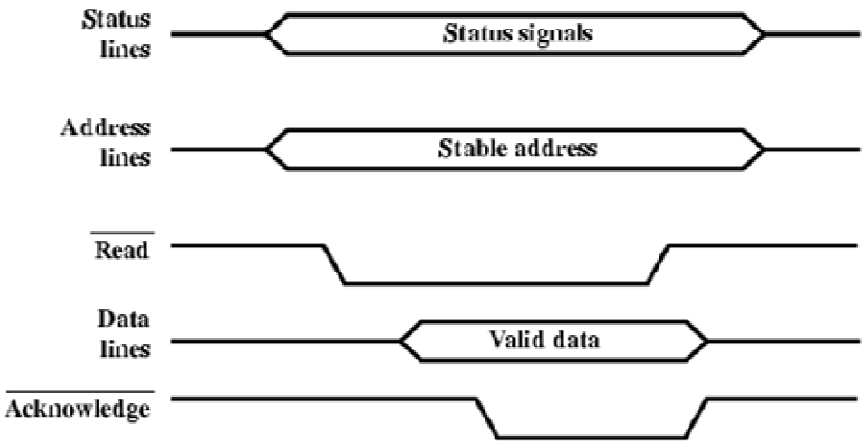
Synchronous Timing Diagram



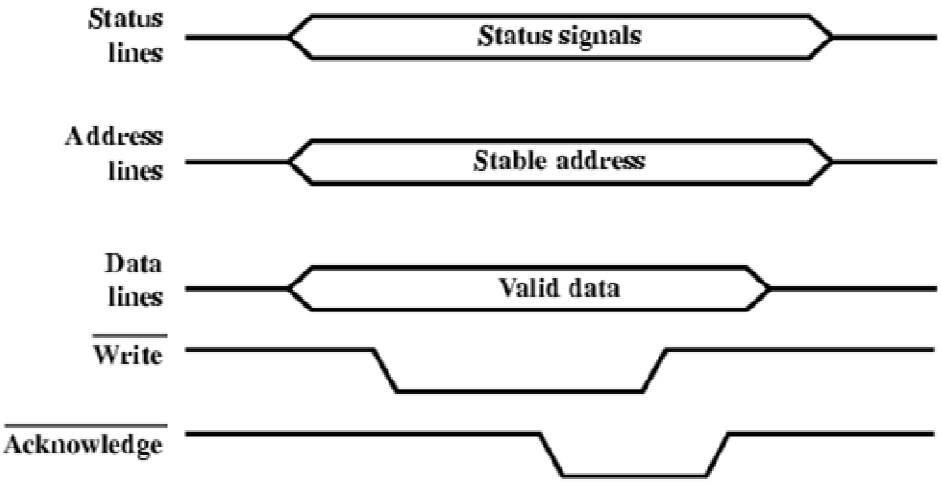
Synchronous Timing Explained

- In this simple example, the processor places a memory address on the address lines during the first clock cycle and may assert various status lines.
- Once the address lines have stabilized, the processor issues an address enable signal.
- For a read operation, the processor issues a read command at the start of the second cycle.
- A memory module recognizes the address and, after a delay of one cycle, places the data on the data lines.
- The processor reads the data from the data lines and drops the read signal.
- For a write operation, the processor puts the data on the data lines at the start of the second cycle, and issues a write command after the data lines have stabilized.
- The memory module copies the information from the data lines during the third clock cycle.

Asynchronous Timing – Read Diagram



Asynchronous Timing – Write Diagram



Asynchronous Timing Explained

- With **asynchronous timing**, the occurrence of one event on a bus follows, and depends on, the occurrence of a previous event.
- In a read operation, the processor places address and status signals on the bus.
- After pausing for these signals to stabilize, it issues a read command, indicating the presence of valid address and control signals.
- The appropriate memory decodes the address and responds by placing the data on the data line.
- Once the data lines have stabilized, the memory module asserts the acknowledged line to signal to the processor that the data are available.
- Once the master has read the data from the data lines, it de-asserts the read signal. This causes the memory module to drop the data and acknowledge lines.
- Finally, once the acknowledge line is dropped, the master removes the address information.
- In an asynchronous write operation. The master places the data on the data line at the same time that is puts signals on the status and address lines.
- The memory module responds to the write command by copying the data from the data lines and then asserting the acknowledge line.
- The master then drops the write signal and the memory module drops the acknowledge signal.

PCI Bus

- Peripheral Component Interconnection
- Intel released Specification to public domain around 1990
- May be configured as 32 or 64 bit
- Synchronous Timing
- Centralized Arbitration
- 49 Mandatory signal lines arranged in functional groups:
  - Systems lines
    - Including clock and reset
  - Address & Data
    - 32 time mux lines for address/data
    - Interrupt & validate lines
  - Interface Control
  - Arbitration
    - Not shared
    - Direct connection to PCI bus arbiter
  - Error lines

PCI Mandatory Pins

Designation	Type	Description
System Pins		
CLK	in	Provides timing for all transactions and is sampled by all inputs on the rising edge. Clock rates up to 33 MHz are supported.
RST#	in	Forces all PCI-specific registers, sequencers, and signals to an initialized state.
Address and Data Pins		
AD[31:0]	t/s	Multiplexed lines used for address and data.
C/BE[3:0]#	t/s	Multiplexed bus command and byte enable signals. During the data phase, the lines indicate which of the four byte lines carry meaningful data.
PAR	t/s	Provides even parity across AD and C/BE lines one clock cycle later. The master drives PAR for address and write data phases; the target drive PAR for read data phases.
Interface Control Pins		
FRAME#	s/t/s	Driven by current master to indicate the start and duration of a transaction. It is asserted at the start and deasserted when the initiator is ready to begin the final data phase.
IRDY#	s/t/s	Initiator Ready. Driven by current bus master (initiator of transaction). During a read, indicates that the master is prepared to accept data; during a write, indicates that valid data are present on AD.
TRDY#	s/t/s	Target Ready. Driven by the target (selected device). During a read, indicates that valid data are present on AD; during a write, indicates that target is ready to accept data.
STOP# IDSEL	s/t/s in	Indicates that current target wishes the initiator to stop the current transaction. Initialization Device Select. Used as a chip select during configuration read and write transactions.
DEVSEL#	in	Device Select. Asserted by target when it has recognized its address. Indicates to current initiator whether any device has been selected.
Arbitration Pins		
REQ#	t/s	Indicates to the arbiter that this device requires use of the bus. This is a device-specific point-to-point line.
GNT#	t/s	Indicates to the device that the arbiter has granted bus access. This is a device-specific point-to-point line.
Error Reporting Pins		
PERR#	s/t/s	Parity Error. Indicates a data parity error is detected by a target during a write data phase or by an initiator during a read data phase.
SERR#	o/d	System Error. May be pulsed by any device to report address parity errors and critical errors other than parity.

PCI Bus

- 51 optional signal lines arranged in functional groups
  - —Interrupt pins
  - —Cache support pins
  - —64-bit bus extension pins
  - —JTAG/boundary scan pins

PCI Optional Pins

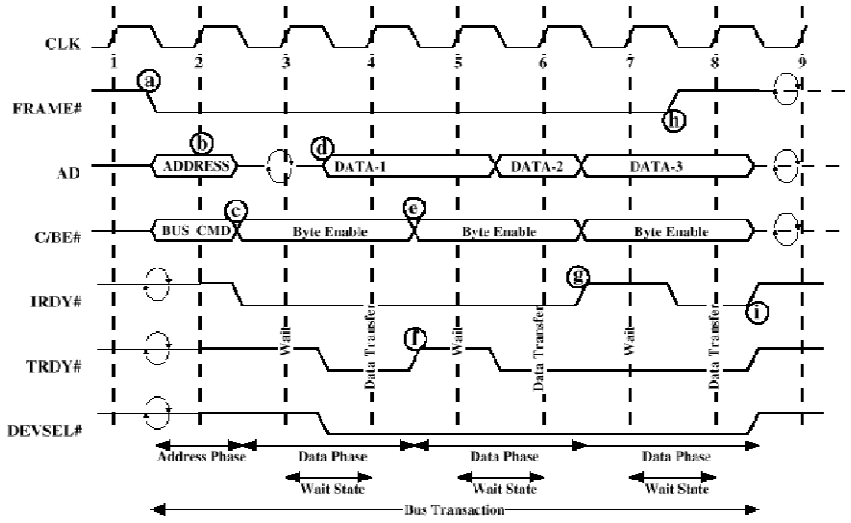
Designation	Type	Description
Interrupt Pins		
INTA#	o/d	Used to request an interrupt.
INTB#	o/d	Used to request an interrupt; only has meaning on a multifunction device.
INTC#	o/d	Used to request an interrupt; only has meaning on a multifunction device.
INTD#	o/d	Used to request an interrupt; only has meaning on a multifunction device.
Cache Support Pins		
SBO#	in/out	Snoop Backoff. Indicates a hit to a modified line.
SDONE	in/out	Snoop Done. Indicates the status of the snoop for the current access. Asserted when snoop has been completed.
64-Bit Bus Extension Pins		
AD[63:32]	t/s	Multiplexed lines used for address and data to extend bus to 64 bits.
C/BE[7:4]#	t/s	Multiplexed bus command and byte enable signals. During the address phase, the lines provide additional bus commands. During the data phase, the lines indicate which of the four extended byte lanes carry meaningful data.
REQ64# ACK64#	s/t/s s/t/s	Used to request 64-bit transfer. Indicates target is willing to perform 64 bit transfer.
PAR64	t/s	Provides even parity across extended AD and C/BE lines one clock cycle later.
JTAG/Boundary Scan Pins		
TCK	in	Test clock. Used to clock state information and test data into and out of the device during boundary scan.
TDI	in	Test input. Used to serially shift test data and instructions into the device.
TDO	out	Test output. Used to serially shift test data and instructions out of the device.
TMS	in	Test mode Select. Used to control state of test access port controller.
TRST#	in	Test reset. Used to initialize test access port controller.

in	Input only signal
out	Output-only signal
t/s	Bidirectional, tri-state, I/O signal
s/t/s	Sustained tri-state signal driven by only one owner at a time
o/d	Open drain: allows multiple devices to share as a wire-OR
#	Signal's active state occurs at low voltage

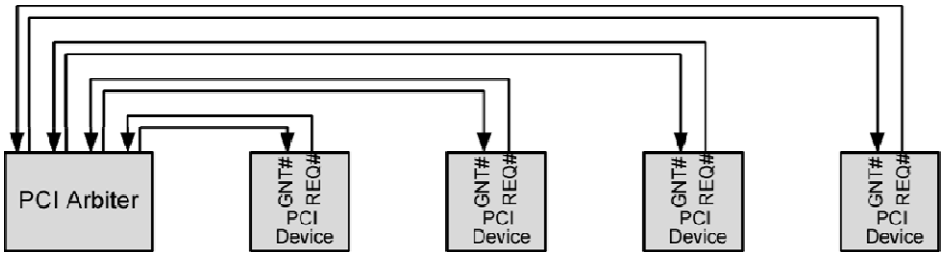
PCI Bus Data Transfers

- Every data transfer on the PCI bus is a single transaction consisting of one address phase and one or more data phases.
- Transaction between initiator (master) and target (slave)
- Master claims bus
- Determine type of transaction
  - —e.g. I/O read/write
- Address phase
- One or more data phases

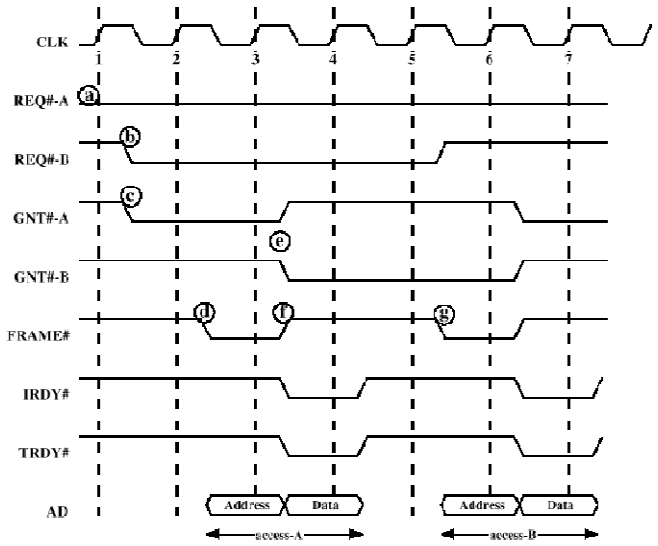
PCI Read Timing Diagram



PCI Bus Arbiter



PCI Bus Arbitration



Arbitration Example

- a. At some point prior to the start of Clock 1, A has asserted its REQ signal. The arbiter samples this signal at the beginning of Clock Cycle 1.
- b. During Clock Cycle 1, B requests use of the bus by asserting its REQ signal.
- c. At the same time, the arbiter asserts GNT-A to grant bus access to A.
- d. Bus master A samples GNT-A at the beginning of clock 2 and learns that it has been granted bus access. It also finds IRDY and TRDY de-asserted, indicating that the bus is idle. Accordingly, it asserts FRAME and places the address information on the address bus and the command on the C/BE bus (not shown). It also continues to assert REQ-A, because it has a second transaction to perform after this one.
- e. The bus arbiter samples all REQ lines at the beginning of clock 3 and makes an arbitration decision to grant the bus to B for the next transaction. It then asserts GNT-B and de-asserts GNT-A. B will not be able to use the bus until it returns to an idle state.
- f. A de-asserts FRAME to indicate that the last (and only) data transfer is in progress. It puts the data on the data bus and signals the target with IRDY. The target reads the data at the beginning of the next clock cycle.
- g. At the beginning of clock 5, B finds IRDY and FRAME de-asserted and so is able to take control of the bus by asserting FRAME. It also de-asserts its REQ line, because it only wants to perform one transaction.

Subsequently, master A is granted access to the bus for its next transaction.

Notice that arbitration can take place at the same time that the current bus master is performing a data transfer. Therefore, no bus cycles are lost in performing arbitration. This is referred to as ***hidden arbitration***.