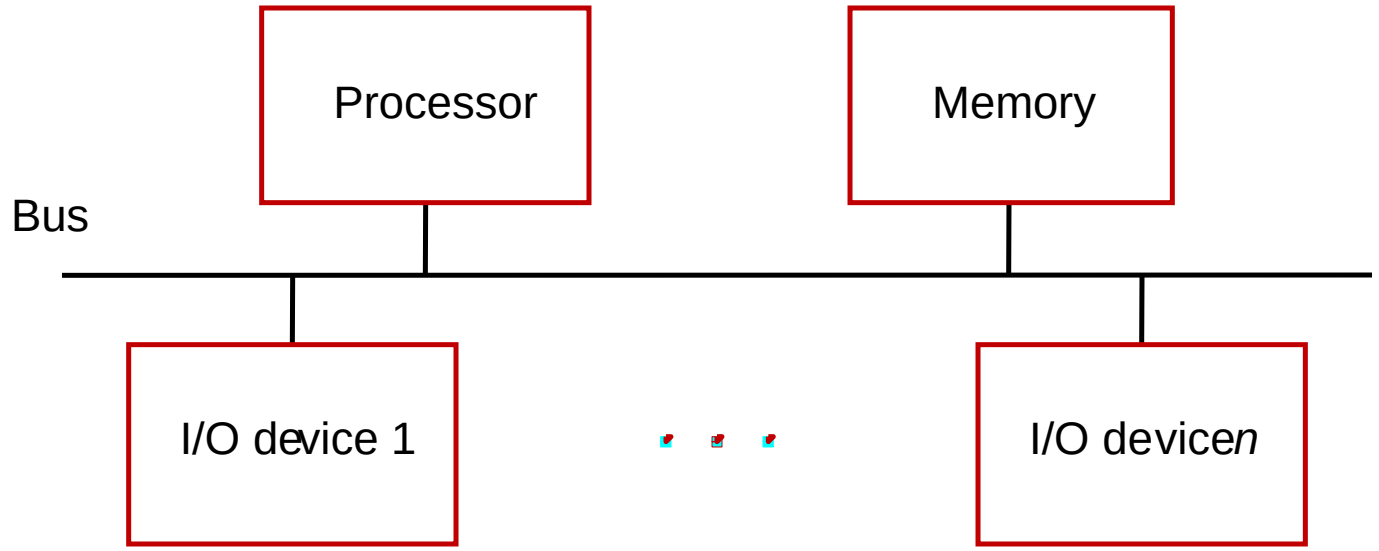
II B.Tech (CSE)
CO-Unit 4
INPUT/OUTPUT ORGANIZATION
Dt.30.01.18

Accessing I/O Devices

Accessing I/O devices



- Multiple I/O devices may be connected to the processor and the memory via a bus.
- Bus consists of three sets of lines to carry address, data and control signals.
- Each I/O device is assigned an unique address.
- To access an I/O device, the processor places the address on the address lines.
- The device recognizes the address, and responds to the control signals.

Accessing I/O Devices

Computers use single bus to connect I/O devices to CPU & Memory.
The bus enables all the devices connected to it to exchange information and it consists of 3 set of lines : *Address, Data, Control*

Address lines: Processor places a particular address (unique for an I/O Dev.) on address lines. Device which recognizes this address responds to the commands issued on the Control lines

Control lines :These lines are used to signal requests and acknowledgments, and to indicate what type of information is on the data lines.

The control lines are used to indicate what the bus contains.

These are used to implement the bus protocol.

Processor requests for either Read / Write

Data lines :These lines carry information between the source and the destination.

This information may consist of data, complex commands, or addresses.

Accessing I/O devices (contd..)

I/O devices and the memory may share the same address space:

- Memory-mapped I/O.
- Any machine instruction that can access memory can be used to transfer data to or from an I/O device.
- Simpler software.

I/O devices and the memory may have different address spaces (I/O mapped I/O)

- Special instructions to transfer data to and from I/O devices.
- I/O devices may have to deal with fewer address lines.
- I/O address lines need not be physically separate from memory address lines.
- In fact, address lines may be shared between I/O devices and memory, with a control signal to indicate whether it is a memory address or an I/O address.

I/O Interface

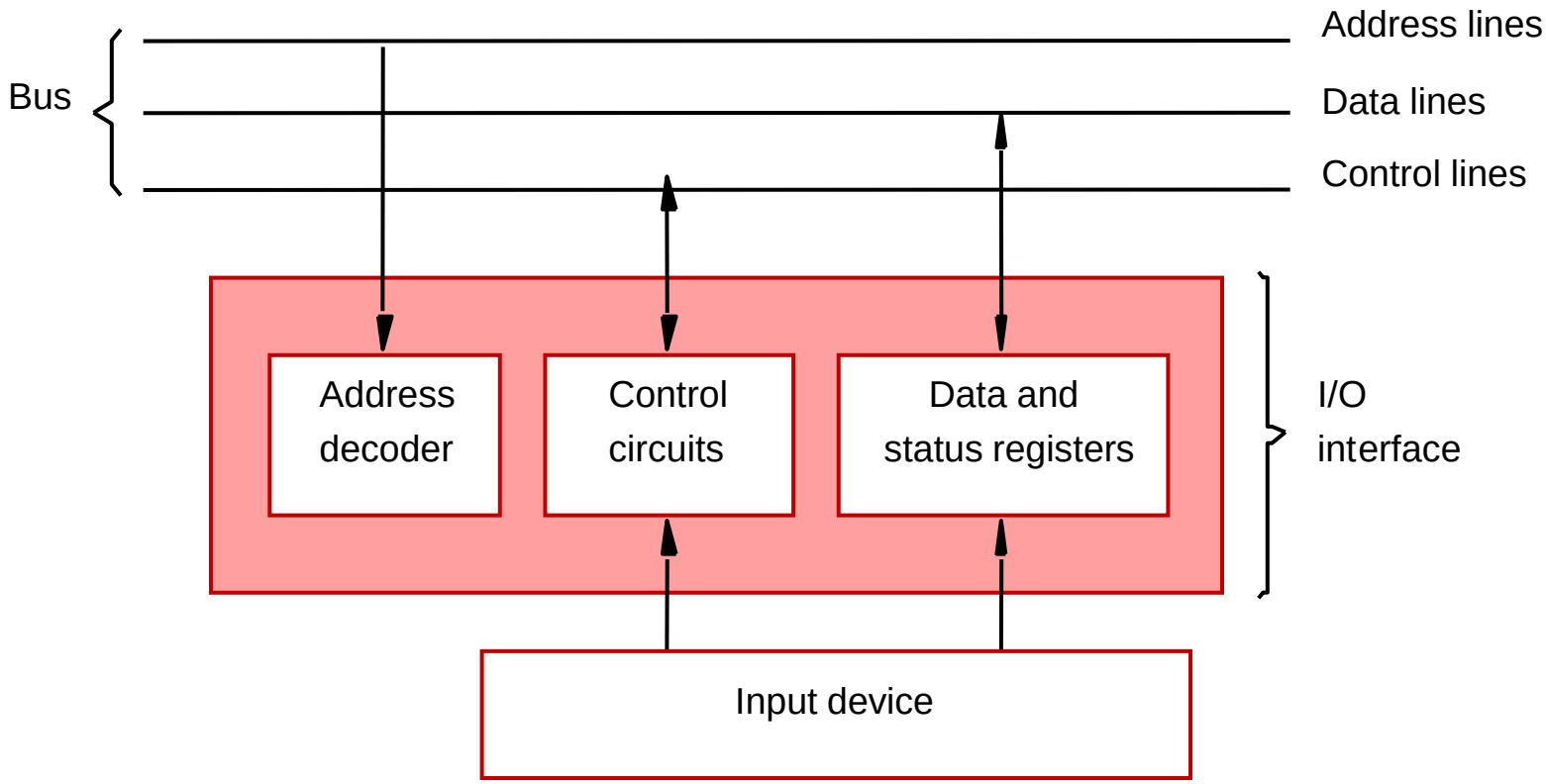


Fig. I/O interface circuit coordinates I/O transfers.

I/O Interface

- I/O device is connected to the bus using an I/O interface circuit.
- I/o interface has : Address decoder, control circuit, and data and status registers.
- **Address decoder** :
  decodes the address placed on the address lines thus enabling the device to recognize its address.
- **Data register:**
  holds the data being transferred to or from the processor.
- **Status register :**
  holds information necessary for the operation of the I/O device.
- **Data and status registers:**

Data registers stores the **data** being transferred to and from the immediate access storage

S***tatus register*** has bits read by the host to ascertain the status of the device, **such as** idle, ready for input, busy, error, transaction complete, etc.

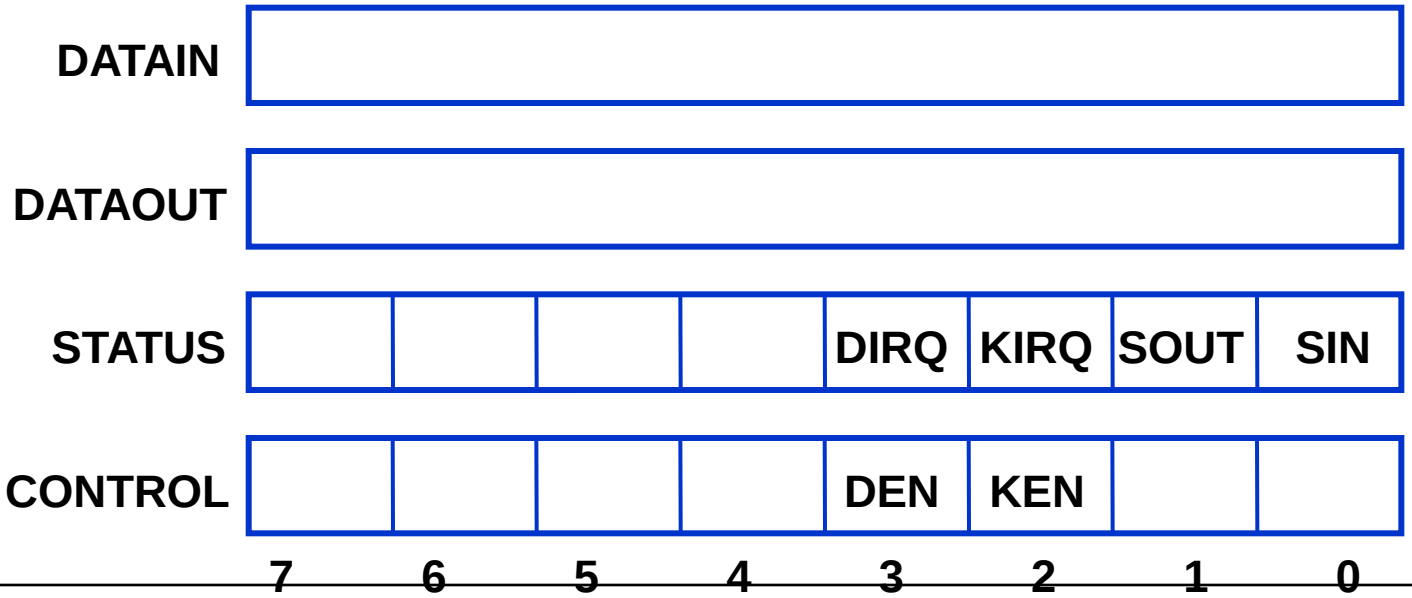
Program-Controlled I/O-Example

I/O operations using keyboard and a display device in a computer system.

The four registers shown below are used in the data transfer operations .

The two flags KIRQ and DIRQ in status in register are used in conjunction with interrupts.

Data from the keyboard are made available in the DATAIN register and data sent to display are stored in the DATAOUT register.



An Example

- A program that reads one line from the keyboard, stores it in the memory buffer and echoes it back to the display.

```
        Move        #LINE, R0       Initialize memory pointer
WAITK   TestBit     #0,STATUS       Test SIN
        Branch=0    WAITK           Wait for character to be entered
        Move        DATAIN,R1       Read character
WAITD   TestBit     #1,STATUS       Test SOUT
        Branch=0    WAITD           Wait for display to become ready
        Move        R1,DATAOUT      Send character to display
        Move        R1,(R0)+        Store character and advance pointer
        Compare     #$0D,R1         Check if Carriage Return
        Branch≠0    WAITK           If not, get another character
        Move        #$0A,DATAOUT    Otherwise, send Line Feed
        Call        PROCESS         Call a subroutine to process the input line
```

Accessing I/O devices (contd..)

Recall that the rate of transfer to and from I/O devices is slower than the speed of the processor. This creates the need for mechanisms to synchronize data transfers between them.

1.Program-controlled I/O:

- I/O devices operate at speeds that are very much different from that of the processor Ex. Key board
- Processor repeatedly monitors a status flag to achieve the necessary synchronization.
- Processor polls the I/O device.

Two other mechanisms used for synchronizing data transfers between the processor and memory:

2.Interrupts:

Synchronization is achieved by having the I/O device send a special signal over the bus whenever it is ready for a data transfer operation

3.Direct Memory Access:

It involves having the device interface transfer data directly to or from the memory

Interrupts

Interrupts

- In program-controlled I/O, when the processor continuously monitors the status of the device, it does not perform any useful tasks.
- However, in many situations other tasks can be performed while waiting for an I/O device to become ready.
- An alternate approach would be for the I/O device to alert the processor when it becomes ready.
  - Do so by sending a hardware signal called an interrupt to the processor.
  - At least one of the bus control lines, called an interrupt-request line is dedicated for this purpose.
- Processor can perform other useful tasks while it is waiting for the device to be ready.

Example

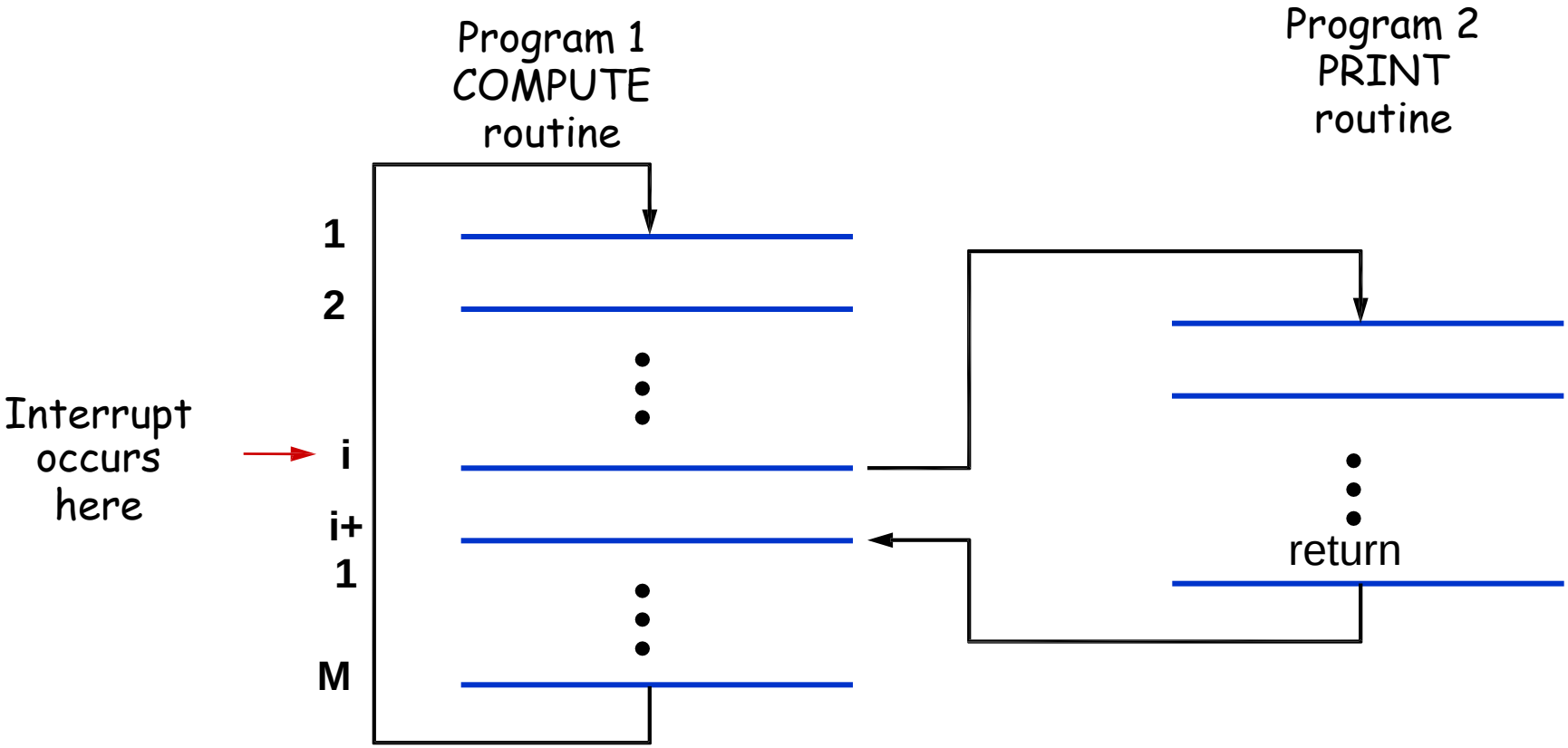


Fig. Transfer of control through the use of interrupts

Interrupts (contd..)

- Processor is executing the instruction located at address i when an interrupt occurs.
- Routine executed in response to an interrupt request is called the interrupt-service routine (ISR)
- When an interrupt occurs, control must be transferred to the interrupt service routine.
- But before transferring control, the current contents of the PC (i+1), must be saved in a known location.
- This enables the return-from-interrupt instruction to resume execution at (i+1).
- Return address, or the contents of the PC are usually stored on the processor stack.

Interrupt call vs subroutine

Treatment of an interrupt-service routine is very similar to that of a subroutine.

However there are significant differences:

- A subroutine performs a task that is required by the calling program.
- Interrupt-service routine may not have anything in common with the program it interrupts.
- Interrupt-service routine and the program that it interrupts may belong to different users.
- As a result, before branching to the interrupt-service routine, not only the PC, but other information such as condition code flags, and processor registers used by both the interrupted program and the interrupt service routine must be stored.
- This will enable the interrupted program to resume execution upon return from interrupt service routine.

Interrupt Latency

Saving and restoring information can be done automatically by the processor or explicitly by program instructions.

Saving and restoring registers involves memory transfers:

- Increases the total execution time.
- Increases the delay between the time an interrupt request is received, and the start of execution of the interrupt-service routine. This delay is called interrupt latency.

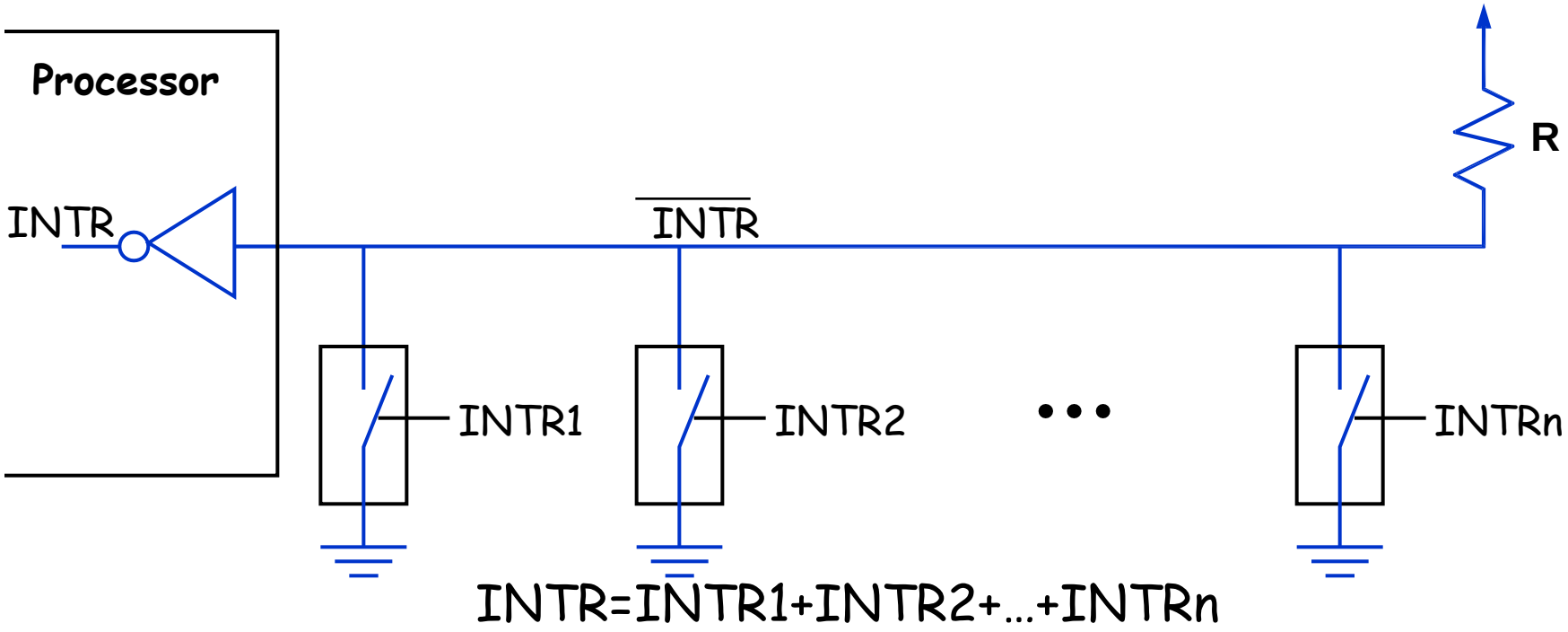
In order to reduce the interrupt latency, most processors save only the minimal amount of information:

- This minimal amount of information includes Program Counter and processor status registers.

Any additional information that must be saved, must be saved explicitly by the program instructions at the beginning of the interrupt service routine.

Interrupt Hardware

Fig. An equivalent circuit for a open –drain used to implement a common interrupt request line



Interrupts (contd..)

- When a processor receives an interrupt-request, it must branch to the interrupt service routine.
- It must also inform the device that it has recognized the interrupt request.
- This can be accomplished in two ways:
  - Some processors have an explicit interrupt-acknowledge control signal for this purpose.
  - In other cases, the data transfer that takes place between the device and the processor can be used to inform the device.

Interrupts (contd..)

Interrupt-requests interrupt the execution of a program, and may alter the intended sequence of events:

- Sometimes such alterations may be undesirable, and must not be allowed.
- For example, the processor may not want to be interrupted by the same device while executing its interrupt-service routine.

Processors generally provide the ability to enable and disable such interruptions as desired.

One simple way is to provide machine instructions such as **Interrupt-enable and Interrupt-disable** for this purpose.

- Let the interrupt be disabled/enabled in the interrupt-service routine.
- Let the processor automatically disable interrupts before starting the execution of the interrupt-service routine

i.e To avoid interruption by the same device during the execution of an interrupt service routine (ISR)

- First instruction of an ISR can be Interrupt-disable.
- Last instruction of an ISR can be Interrupt-enable.

Handling Multiple Devices

Multiple I/O devices may be connected to the processor and the memory via a bus. Some or all of these devices may be capable of generating interrupt requests.

- Each device operates independently, and hence no definite order can be imposed on how the devices generate interrupt requests?
- How does the processor know which device has generated an interrupt?
- How does the processor know which interrupt service routine needs to be executed? (Vectored interrupts)
- When the processor is executing an interrupt service routine for one device, can other device interrupt the processor? (Interrupt nesting)
- If two interrupt-requests are received simultaneously, then how to break the tie? (Daisy-chain)

Identification of interrupting device by Software Polling

Consider a simple arrangement where all devices send their interrupt-requests over a single control line in the bus.

When the processor receives an interrupt request over this control line, how does it know which device is requesting an interrupt?

This information is available in the status register of the device requesting an interrupt:

- The status register of each device has an *IRQ* bit which it sets to 1 when it requests an interrupt.

Interrupt service routine can poll the I/O devices connected to the bus. The first device with *IRQ* equal to 1 is the one that is serviced.

Polling mechanism is easy, but time consuming to query the status bits of all the I/O devices connected to the bus.

Vectored Interrupts

- The device requesting an interrupt may identify itself directly to the processor.
  - Device can do so by sending a special code (4 to 8 bits) the processor over the bus.
  - Code supplied by the device may represent a part of the starting address of the interrupt-service routine.
  - The remainder of the starting address is obtained by the processor based on other information such as the range of memory addresses where interrupt service routines are located.
- Usually the location pointed to by the interrupting device is used to store the starting address of the interrupt-service routine.
- Avoid bus collision

Interrupt Nesting

Multiple I/O devices may be connected to the processor and the memory via a bus. Some or all of these devices may be capable of generating interrupt requests.

- Each device operates independently, and hence no definite order can be imposed on how the devices generate interrupt requests?

How does the processor know which device has generated an interrupt?

How does the processor know which interrupt service routine needs to be executed?

When the processor is executing an interrupt service routine for one device, can other device interrupt the processor?

If two interrupt-requests are received simultaneously, then how to break the tie?

Interrupt Nesting (contd..)

Consider a simple arrangement where all devices send their interrupt-requests over a single control line in the bus.

When the processor receives an interrupt request over this control line, how does it know which device is requesting an interrupt?

This information is available in the status register of the device requesting an interrupt:

- The status register of each device has an *IRQ* bit which it sets to 1 when it requests an interrupt.

Interrupt service routine can poll the I/O devices connected to the bus. The first device with *IRQ* equal to 1 is the one that is serviced.

Polling mechanism is easy, but time consuming to query the status bits of all the I/O devices connected to the bus.

Vectored Interrupts

- The device requesting an interrupt may identify itself directly to the processor.
  - Device can do so by sending a special code (4 to 8 bits) the processor over the bus.
  - Code supplied by the device may represent a part of the starting address of the interrupt-service routine.
  - The remainder of the starting address is obtained by the processor based on other information such as the range of memory addresses where interrupt service routines are located.
- Usually the location pointed to by the interrupting device is used to store the starting address of the interrupt-service routine.

Interrupts (contd..)

Previously, before the processor started executing the interrupt service routine for a device, it disabled the interrupts from the device.

In general, same arrangement is used when multiple devices can send interrupt requests to the processor.

- During the execution of an interrupt service routine of device, the processor does not accept interrupt requests from any other device.
- Since the interrupt service routines are usually short, the delay that this causes is generally acceptable.

However, for certain devices this delay may not be acceptable.

- Which devices can be allowed to interrupt a processor when it is executing an interrupt service routine of another device?

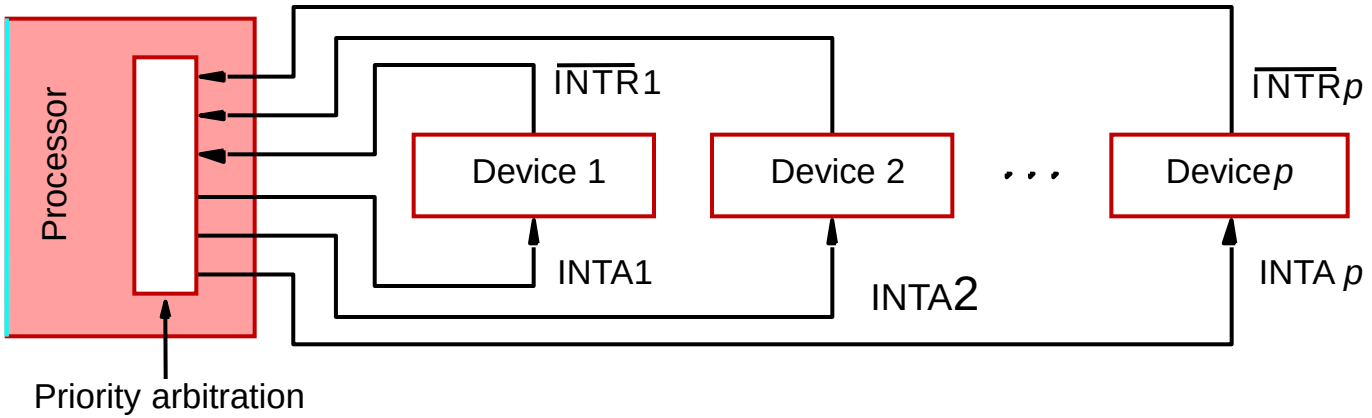
Interrupts –Priority levels -Interrupt Nesting

- I/O devices are organized in a priority structure:
  - An interrupt request from a high-priority device is accepted while the processor is executing the interrupt service routine of a low priority device.
- A priority level is assigned to a processor that can be changed under program control.
  - Priority level of a processor is the priority of the program that is currently being executed.
  - When the processor starts executing the interrupt service routine of a device, its priority is raised to that of the device.
  - If the device sending an interrupt request has a higher priority than the processor, the processor accepts the interrupt request.

Interrupts –priorities

- Processor's priority is encoded in a few bits of the processor status register.
  - Priority can be changed by instructions that write into the processor status register.
  - Usually, these are privileged instructions, or instructions that can be executed only in the supervisor mode.
  - Privileged instructions cannot be executed in the user mode.
  - Prevents a user program from accidentally or intentionally changing the priority of the processor.
- If there is an attempt to execute a privileged instruction in the user mode, it causes a special type of interrupt called as privilege exception.

Implementation of Interrupt Priority



- Each device has a separate interrupt-request and interrupt-acknowledge line.
- Each interrupt-request line is assigned a different priority level.
- Interrupt requests received over these lines are sent to a priority arbitration circuit in the processor.
- The interrupt request of a higher priority device is accepted.
- The Priority structure only accept one interrupt at a time, then disable all others.

Problem: some interrupts cannot be held too long.

Interrupts- Priority arbitration (contd..)

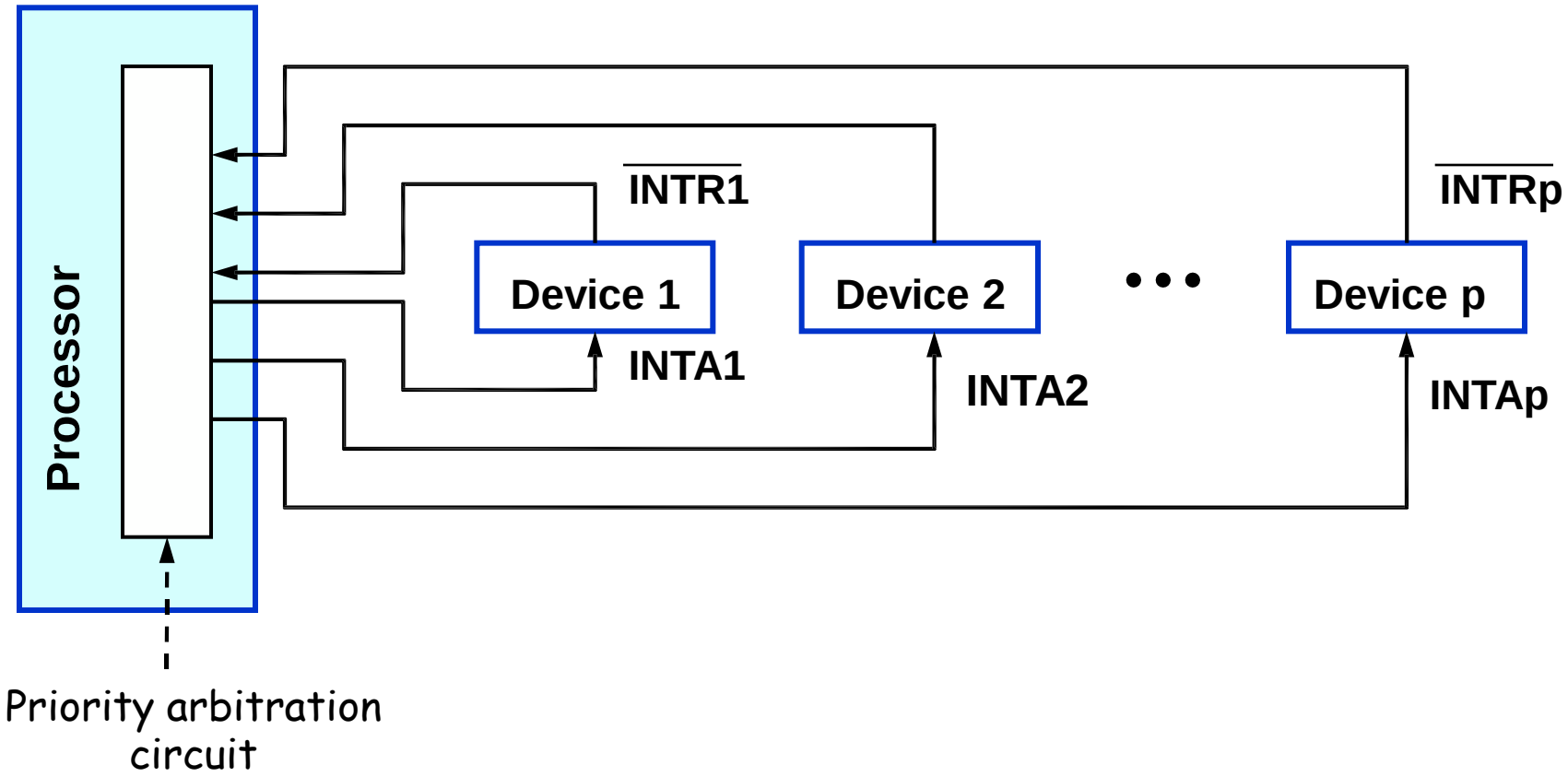
Which interrupt request does the processor accept if it receives interrupt requests from two or more devices simultaneously?.

If the I/O devices are organized in a priority structure, the processor accepts the interrupt request from a device with higher priority.

- Each device has its own interrupt request and interrupt acknowledge line.
- A different priority level is assigned to the interrupt request line of each device.

However, if the devices share an interrupt request line, then how does the processor decide which interrupt request to accept?

Implementation of multiple Interrupt Priority



Interrupt priority -Simultaneous Requests

Consider the problem of simultaneous arrivals of interrupt requests from two or more devices.

The processor must have some means of deciding which request to service first.

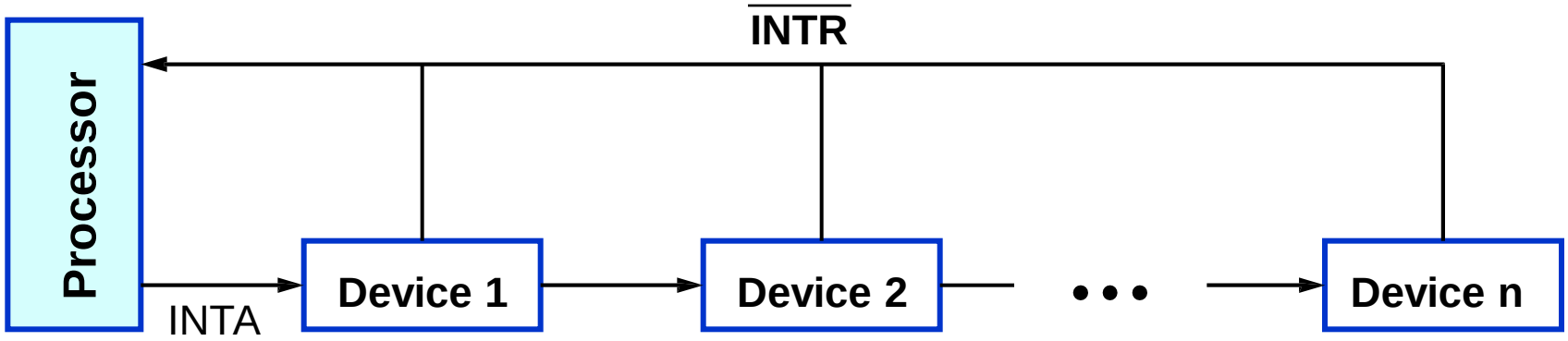


Fig. Interrupt priority with Daisy chain scheme

Interrupts - Simultaneous Requests

Polling scheme:

- If the processor uses a polling mechanism to poll the status registers of I/O devices to determine which device is requesting an interrupt.
- In this case the priority is determined by the order in which the devices are polled.
- The first device with status bit set to 1 is the device whose interrupt request is accepted first.

Daisy chainning

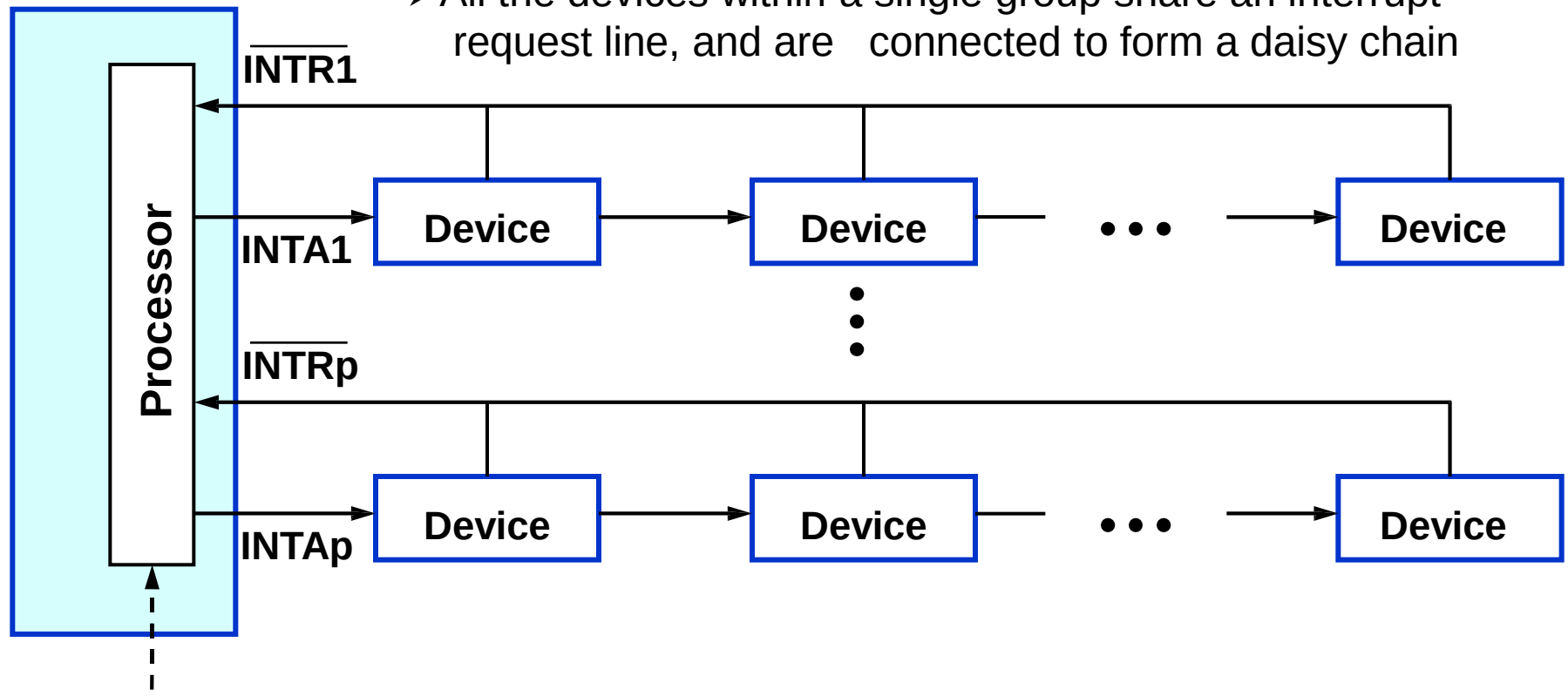
- Devices are connected to form a daisy chain.
- Devices share the interrupt-request line, and interrupt- acknowledge line is connected to form a daisy chain.
- When devices raise an interrupt request, the interrupt-request line is activated.
- The processor in response activates interrupt-acknowledge.
- Based on the request received by devices, if device 1 does not need service, it passes the INTA signal to device 2.
- Device that is electrically closest to the processor has the highest priority.

Interrupts - Simultaneous Requests

- When I/O devices were organized into a priority structure, each device had its own interrupt-request and interrupt-acknowledge line.
- When I/O devices were organized in a daisy chain fashion, the devices shared an interrupt-request line, and the interrupt-acknowledge propagated through the devices.
- A combination of priority structure and daisy chain scheme can also used.

Priority Group

- Devices are organized into groups.
- Each group is assigned a different priority level.
- All the devices within a single group share an interrupt-request line, and are connected to form a daisy chain



Interrupts - Controlling Device Requests

Only those devices that are being used in a program should be allowed to generate interrupt requests.

To control which devices are allowed to generate interrupt requests, the interface circuit of each I/O device has an interrupt-enable bit.

- If the interrupt-enable bit in the device interface is set to 1, then the device is allowed to generate an interrupt-request.

Interrupt-enable bit in the device's interface circuit determines whether the device is allowed to generate an interrupt request.

Interrupt-enable bit in the processor status register or the priority structure of the interrupts determines whether a given interrupt will be accepted.

Use of Interrupts in Operating Systems

- ✓ The OS and the application program pass control back and forth using software interrupts.
- ✓ Supervisor mode / user mode
- ✓ Multitasking (time-slicing)
- ✓ Process – running, runnable, blocked
- ✓ Program state

Exceptions

Interrupts caused by interrupt-requests sent by I/O devices.

Interrupts could be used in many other situations where the execution of one program needs to be suspended and execution of another program needs to be started.

In general, the term exception is used to refer to any event that causes an interruption.

- Interrupt-requests from I/O devices is one type of an exception.

Other types of exceptions are:

- Recovery from errors
- Debugging
  - Trace
  - Breakpoint
- Privilege exception

Exceptions (contd..)

Many sources of errors in a processor. For example:

- Error in the data stored.
- Error during the execution of an instruction.

When such errors are detected, exception processing is initiated.

- Processor takes the same steps as in the case of I/O interrupt-request.
- It suspends the execution of the current program, and starts executing an exception-service routine.

Difference between handling I/O interrupt-request and handling exceptions due to errors:

- In case of I/O interrupt-request, the processor usually completes the execution of an instruction in progress before branching to the interrupt-service routine.
- In case of exception processing however, the execution of an instruction in progress usually cannot be completed.

Exceptions (contd..)

- Debugging program is used as the exception-service routine i.e.Debugger uses exceptions to provide important features:
  - Trace,
  - Breakpoints.
- Trace mode:
  - Exception occurs after the execution of every instruction.
- Breakpoints:
  - Exception occurs only at specific points selected by the user.

Privileged Exceptions

- Certain instructions can be executed only when the processor is in the supervisor mode. These are called privileged instructions.
- If an attempt is made to execute a privileged instruction in the user mode, a privilege exception occurs.
- Privilege exception causes:
  - Processor to switch to the supervisor mode,
  - Execution of an appropriate exception-servicing routine.

Direct Memory Access

Direct Memory Access (DMA)

Think about the overhead in both polling and interrupting mechanisms when a large block of data need to be transferred between the processor and the I/O device.

- A special control unit may be provided to transfer a block of data directly between an I/O device and the main memory, without continuous intervention by the processor

DMA controller :

It is a special Control unit that performs these transfers as part of the I/O device's interface.

Functions of DMA :

- For each word, it provides the memory address and all the control signals.
- To transfer a block of data, it increments the memory addresses and keeps track of the number of transfers.

DMA Controller (contd..)

DMA controller can transfer a block of data from an external device to the processor, without any intervention from the processor.

- However, the operation of the DMA controller must be under the control of a program executed by the processor. That is, the processor must initiate the DMA transfer.

To initiate the DMA transfer, the processor informs the DMA controller of:

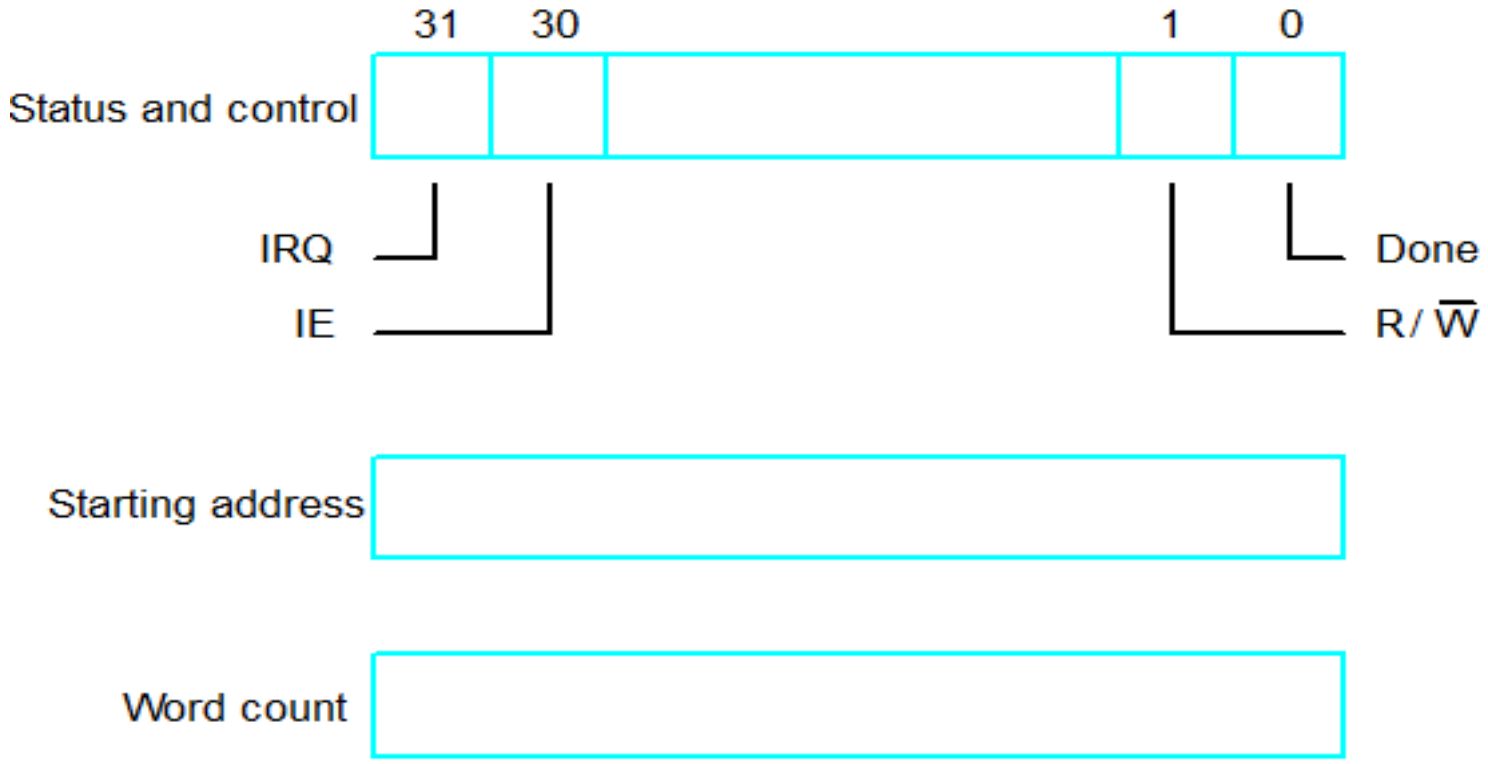
- Starting address,
- Number of words in the block.
- Direction of transfer (I/O device to the memory, or memory to the I/O device).

Once the DMA controller completes the DMA transfer, it informs the processor by raising an interrupt signal.

During DMA transfer, the program that requested the transfer is blocked by OS and the processor can be used to execute another program.

After the DMA transfer is completed, the suspended program is put into the *Runnable state*.

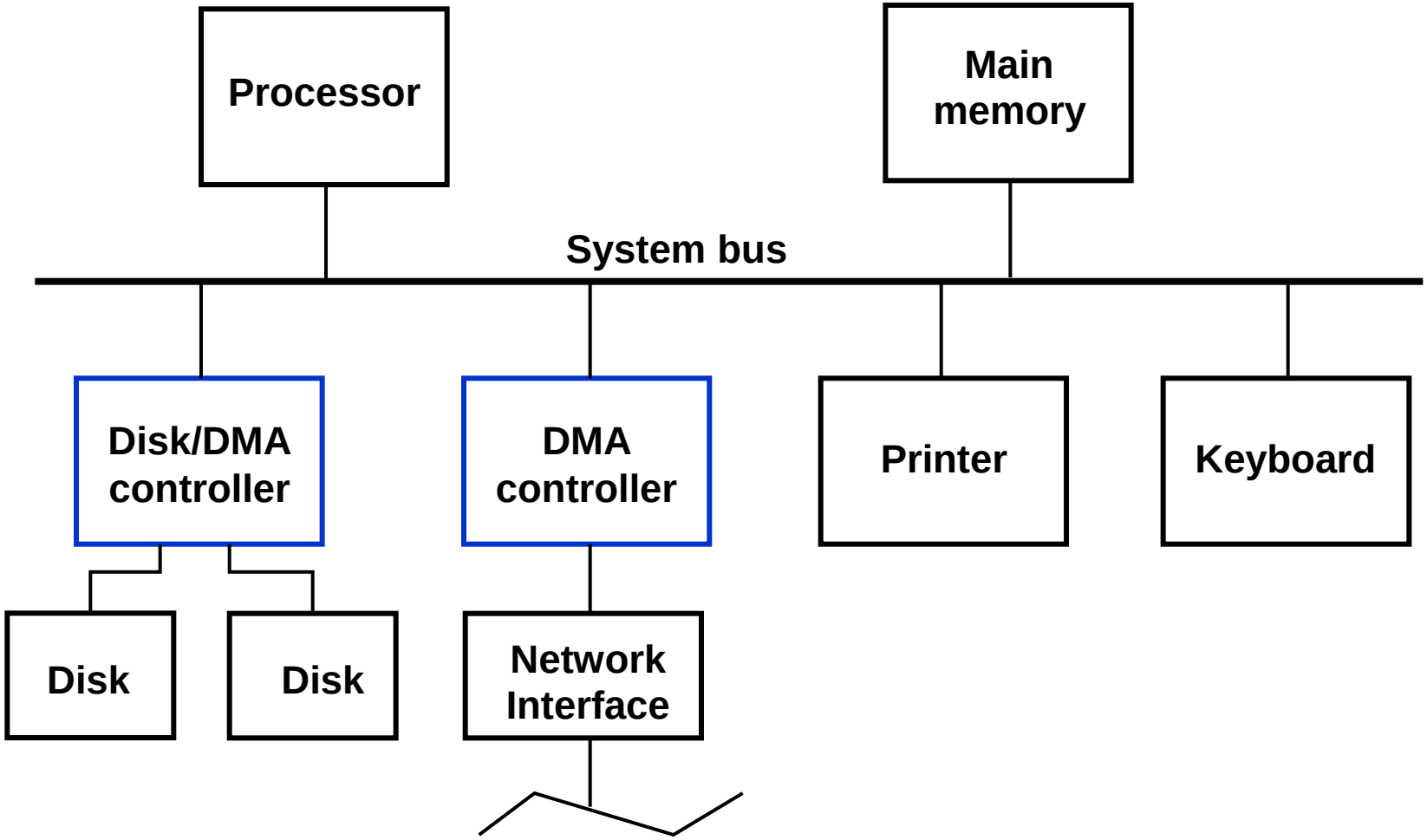
DMA Registers



DMA Controller in Computer System

- DMA controller connects a high-speed network to the computer bus.
- Disk controller, which controls two disks also has DMA capability. It provides two DMA channels.
- It can perform two independent DMA operations, as if each disk has its own DMA controller.
- The registers to store the memory address, word count and status and control information are duplicated.

DMA controller in Computer System



DMA controller

Processor and DMA controllers have to use the bus in an interwoven fashion to access the memory.

- DMA devices are given higher priority than the processor to access the bus.
- Among different DMA devices, high priority is given to high-speed peripherals such as a disk or a graphics display device.

Processor originates most memory access cycles on the bus.

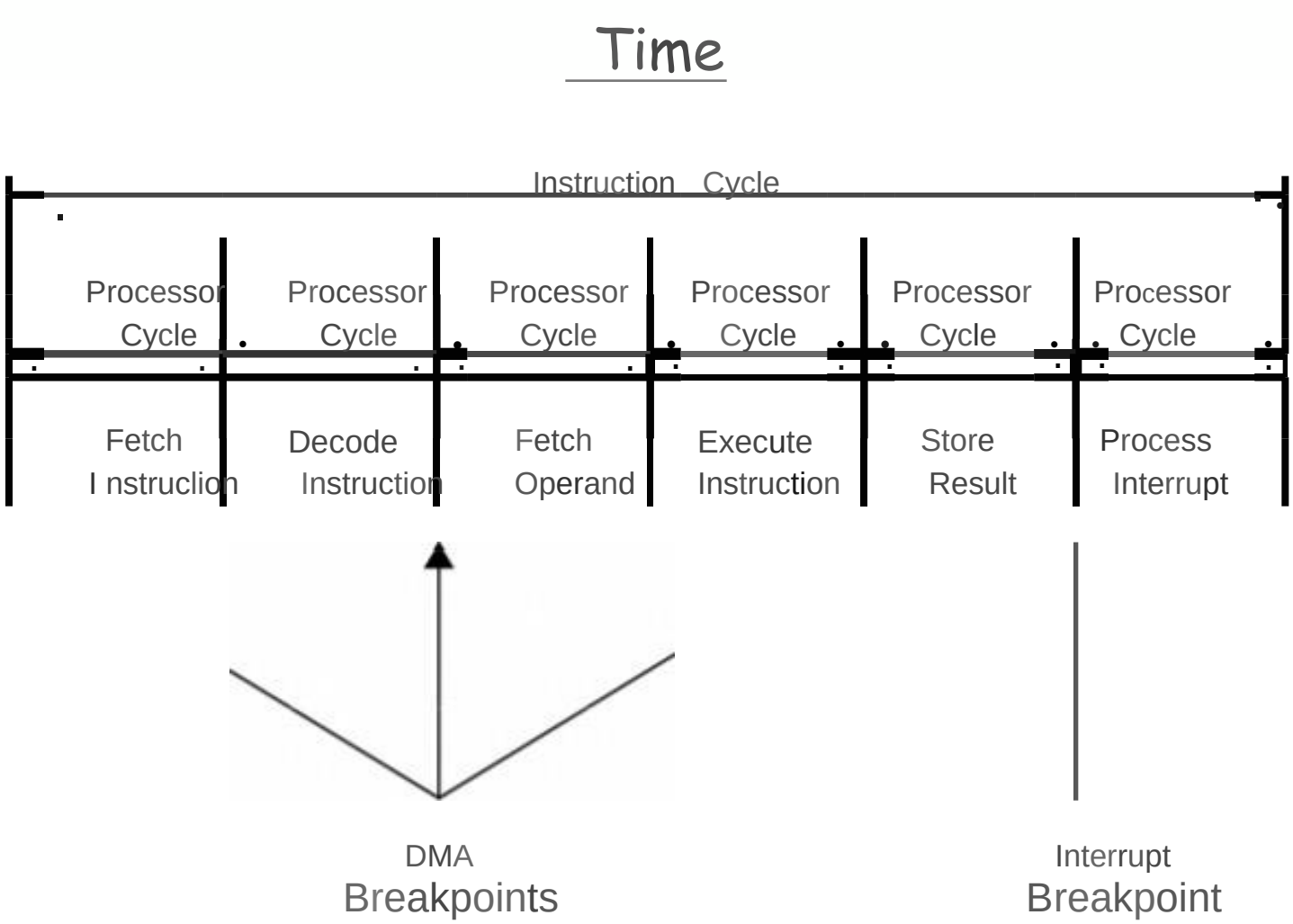
- The modes of DMA operations. Cycle stealing and Block (burst) mode
  - Cycle Stealing : DMA controller can be said to "steal" memory access cycles from the bus.
  - Block or Burst mode: DMA controller can have capability to initiate transfers on the bus, and hence exclusive access to the main memory.

In burst mode, the data in the buffer are transmitted over the network at the speed of the network.

There is a **scope for conflicts** for the control of the bus at the same time and needs coordination to access the main memory

i) Between Processor and DMA and ii) Two DMA controllers

DMA and Interrupt Breakpoints During an Instruction Cycle



Bus Arbitration

Bus Arbitration

Processor and DMA controllers both need to initiate data transfers on the bus and access main memory.

The device that is allowed to initiate transfers on the bus at any given time is called the **bus master**.

When the current bus master relinquishes its status as the bus master, another device can acquire this status.

Multiple devices may need to use the bus at the same time so must have a way to arbitrate multiple requests.

- Bus Arbitration is the process by which the next device to become the bus master is selected and bus mastership is transferred to it.

Centralized arbitration:

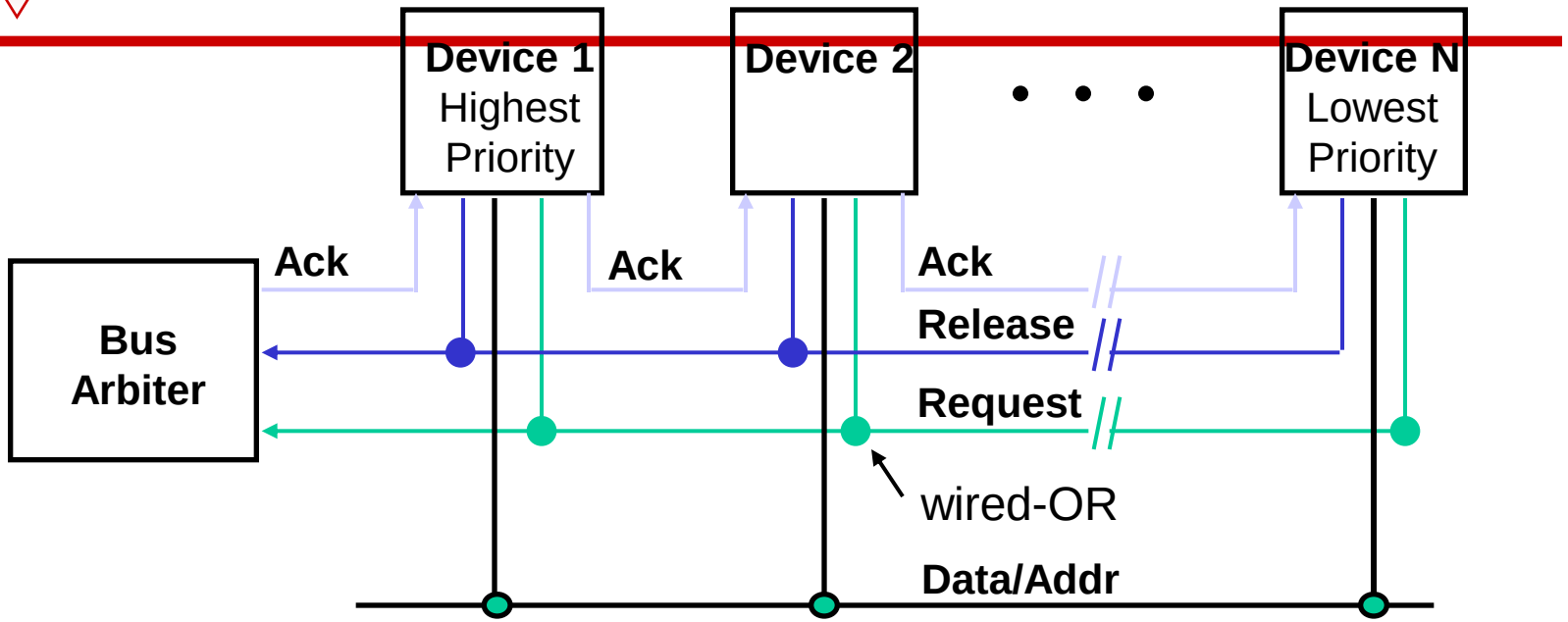
- A single bus arbiter performs the arbitration.

Distributed arbitration:

- All devices participate in the selection of the next bus master.

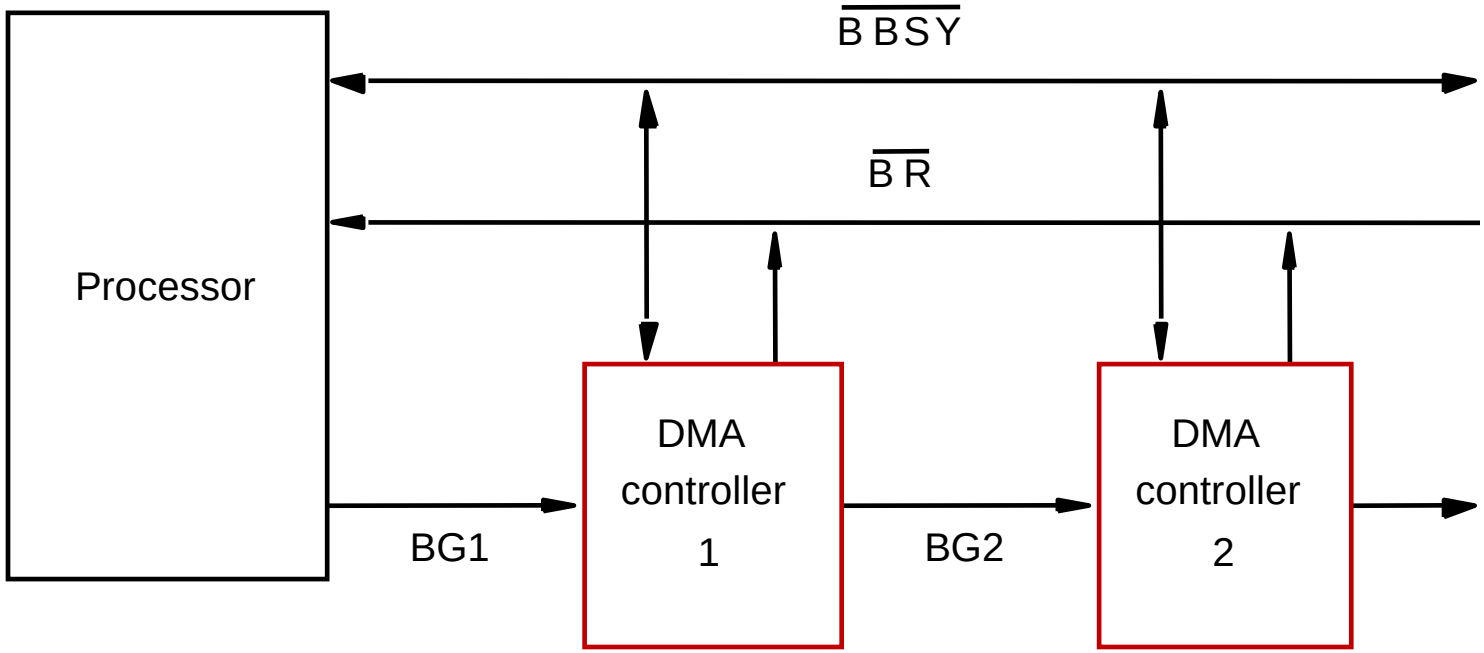
- Bus arbitration schemes can be divided into **four classes**
  - Daisy chain arbitration
  - Centralized, parallel arbitration
  - Distributed arbitration by self-selection – each device wanting the bus places a code indicating its identity on the bus
  - Distributed arbitration by collision detection – device uses the bus when its not busy and if a collision happens (because some other device also decides to use the bus) then the device tries again later (Ethernet)

Daisy Chain Bus Arbitration



- Advantage: simple
- Disadvantages:
  - Cannot assure fairness – a low-priority device may be locked out indefinitely
  - Slower – the daisy chain grant signal limits the bus speed

Bus Arbitration using Daisy Chain



Bus Arbitration using Daisy Chain

Centralized Bus Arbitration (cont.,)

Bus arbiter may be the processor or a separate unit connected to the bus.

Normally, the processor is the bus master, unless it grants bus membership to one of the DMA controllers.

DMA controller requests the control of the bus by asserting the Bus Request (BR) line.

BR is a open drain line - the signal on this line is a logical OR of the bus requests from all the DMA devices .

In response, the processor activates the Bus-Grant1 (BG1) line, in responsive to BR ,indicating that the controller may use the bus when it is free.

$\overline{BG1}$ signal is connected to all DMA controllers in a daisy $\overline{chain}$ fashion.

BBSY (bus busy) line - open collector line.

BBSY signal is 0, it indicates that the bus is busy. When BBSY becomes 1, the DMA controller which asserted BR can acquire control of the bus.

Centralized arbitration (contd..)

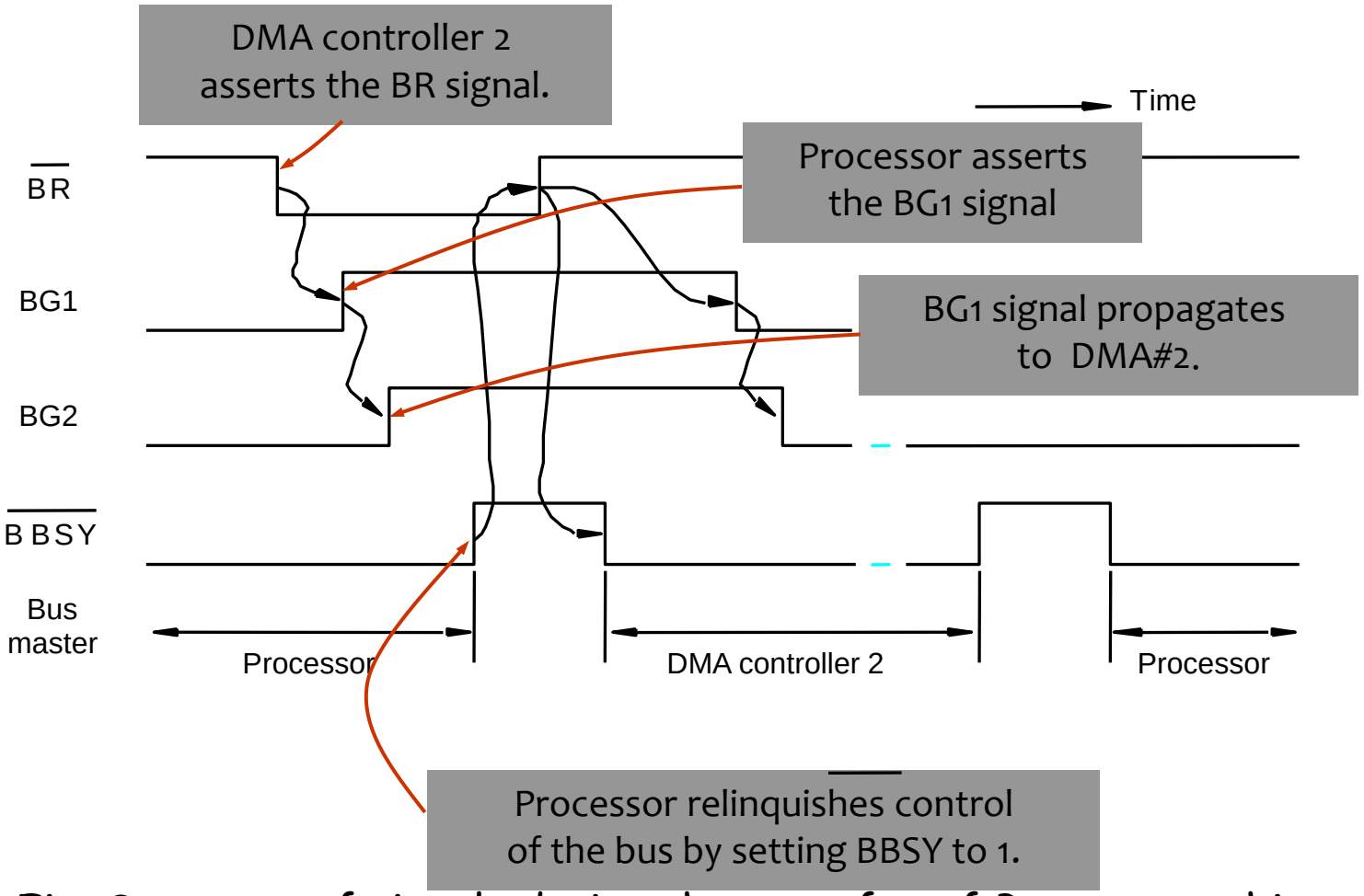


Fig. Sequence of signals during the transfer of Bus mastership.

Centralized Arbitration

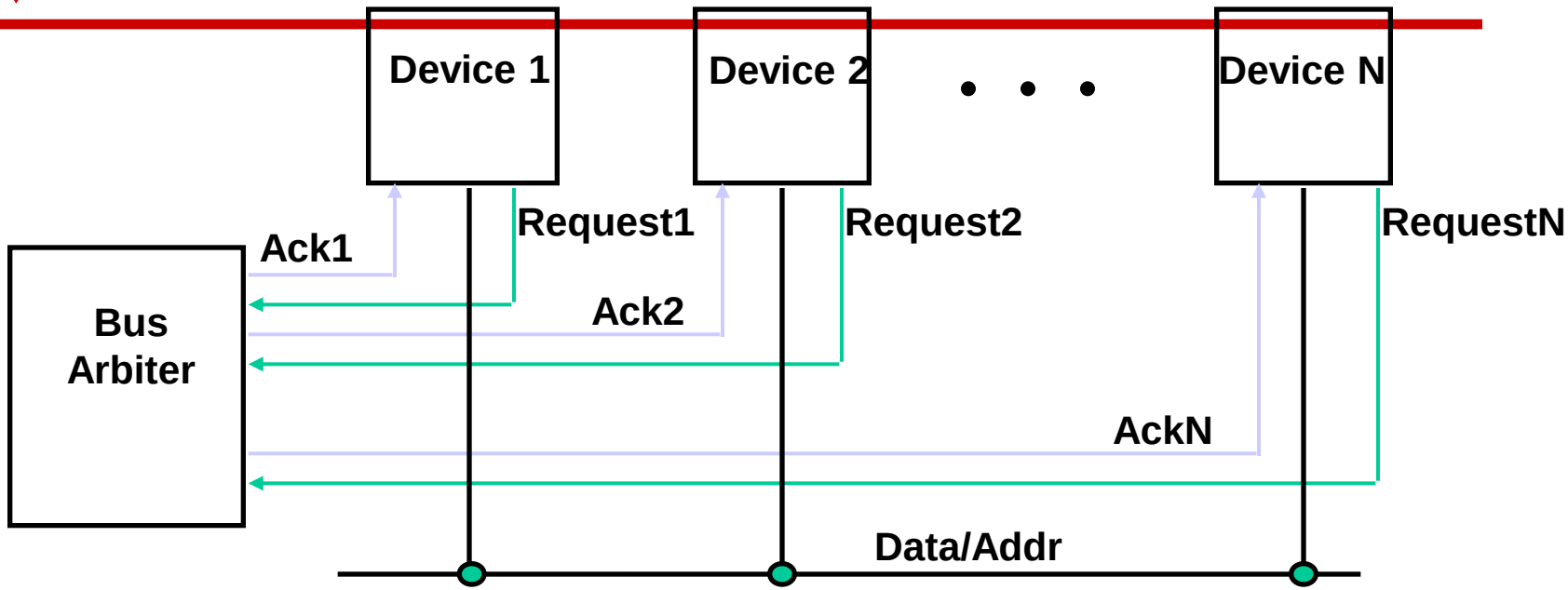
Separate unit (bus arbitration circuitry) connected to the bus.
Processor is normally the bus master, unless it grants bus mastership to DMA
For the timing/control, in previous slide:

DMA controller 2 requests and acquires bus mastership and later releases the bus.

During its tenure as the bus master, it may perform one or more data transfer operations, depending on whether it is operating in the cycle stealing or block mode.

After it releases the bus, the processor resumes bus mastership.

Centralized Parallel Arbitration



- Advantages: flexible, can assure fairness
- Disadvantages: more complicated arbiter hardware
- Used in essentially all processor-memory buses and in high-speed I/O buses

Distributed arbitration

All devices waiting to use the bus share the responsibility of carrying out the arbitration process.

- Arbitration process does not depend on a central arbiter and hence distributed arbitration has higher reliability.

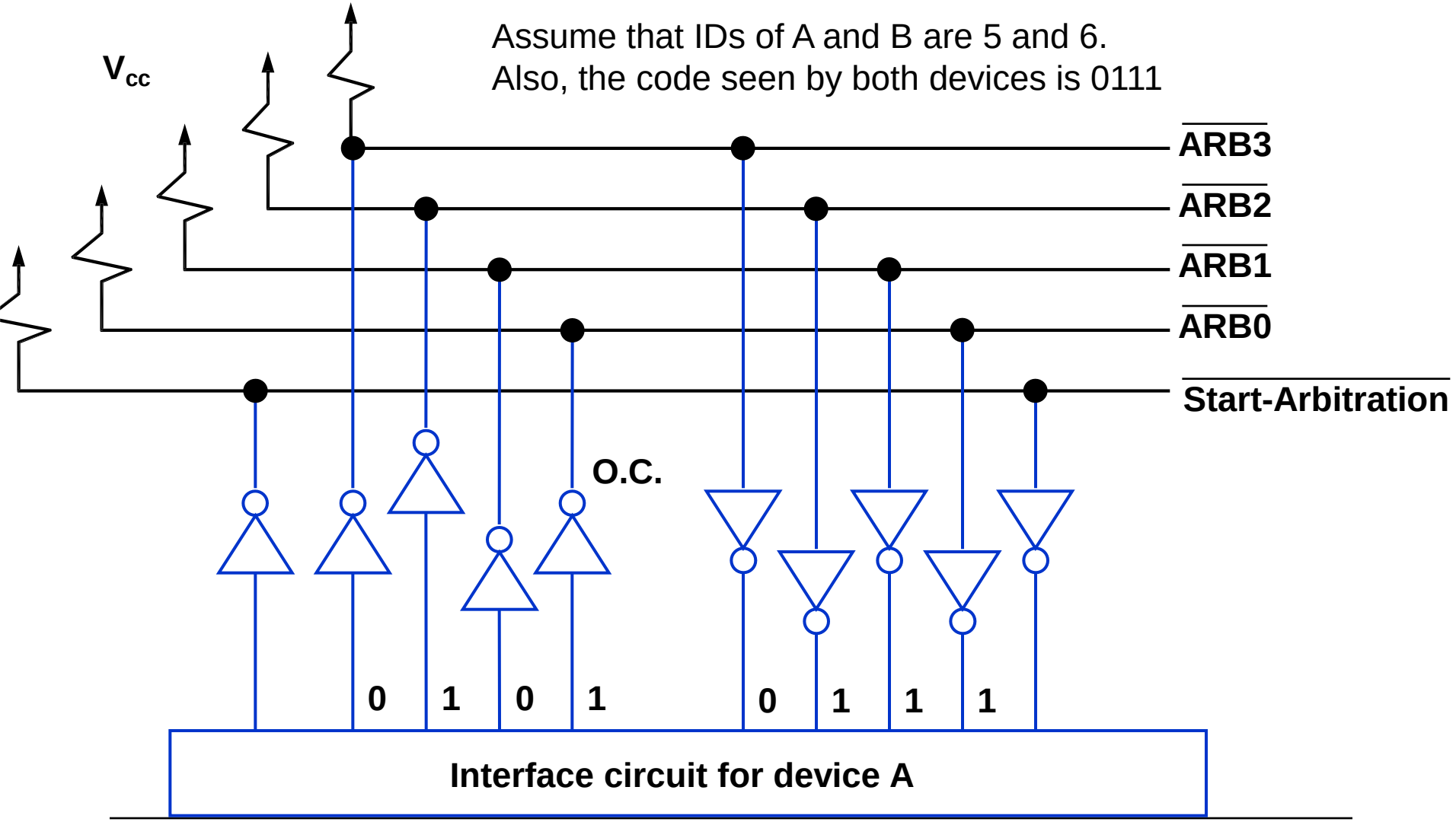
Each device is assigned a 4-bit ID number.

All the devices are connected using 5 lines, 4 arbitration lines to transmit the ID, and one line for the Start-Arbitration signal.

To request the bus a device:

- The devices request for the bus Assert the Start-Arbitration signal.
- They Places their 4-bit ID number on the arbitration lines ($\overline{ARB_0}$-$\overline{ARB3}$)
- ARB0 through ARB3 are the four open collector lines
- One among the four is selected using the code on the lines and one with the highest ID number

Distributed Arbitration by self-selection



Distributed Arbitration(Contd.,)

- Arbitration process:
  - Each device compares the pattern that appears on the arbitration lines to its own ID, starting with MSB.
  - If it detects a difference, it transmits 0s on the arbitration lines for that and all lower bit positions.
  - The pattern that appears on the arbitration lines is the logical-OR of all the 4-bit device IDs placed on the arbitration lines.
  - Decentralized arbitration offers higher reliability.

Distributed arbitration -by self-selection (contd)

- Device A has the ID 5 and wants to request the bus:
  - Transmits the pattern 0101 on the arbitration lines.
- Device B has the ID 6 and wants to request the bus:
  - Transmits the pattern 0110 on the arbitration lines.
- Pattern that appears on the arbitration lines is the logical OR of the patterns: The code seen by both the devices is 0111.

Arbitration process:

✓Each device compares the pattern that appears on the arbitration lines to its own ID, starting with MSB.

✓If it detects a difference at any bit position, it disables its drivers at that bit position and for all lower –order bits.

✓It transmits 0s at the input of these drivers.

✓In the example, Device A detects a difference on **$\overline{\text{ARB1}}$.**

(compares its ID 5 with a pattern 0101 to pattern 0111)

Hence, it disables its drivers on lines **$\overline{\text{ARB1}}$ and $\overline{\text{ARB0}}$.**

✓This causes the pattern on the arbitration lines to change to 0110,which means that B has won the contention.

Note: The code on the arbitration lines is 0111 for a short period .

The device B is temporarily disable its driver on the $\overline{\text{ARB0}}$.

However ,it will enable this driver again once it sees a 0 on line $\overline{\text{ARB1}}$ resulting from action A.

Buses

Buses

- Bus provides a communication path for the transfer of data.
  - Bus also includes lines to support interrupts and arbitration.
- A bus protocol is the set of rules that govern the behavior of various devices connected to the bus, as to when to place information on the bus, when to assert control signals, etc.
- Buses are traditionally classified as processor-memory buses or I/O buses or special purposed buses (Graphics, etc.).
- Processor memory buses are short, generally high speed, and matched to the memory system so as to maximize memory- processor bandwidth.
- I/O buses, by contrast, can be lengthy, can have many types of devices connected to them, and often have a wide range in the data bandwidth of the devices connected to them.
- I/O buses do not typically interface directly to the memory but use either a processor-memory or a backplane bus to connect to memory.
- Bus master (initiator) / slave (target)

Bus Design

Reason why bus design is so difficult :

- The maximum bus speed is largely limited by physical factors like: the length of the bus and the number of devices. These physical limits prevent us from running the bus arbitrarily fast.
- In addition, the need to support a range of devices with widely varying latencies and data transfer rates also makes bus design challenging.
- It becomes difficult to run many parallel wires at high speed due to clock skew and reflection.

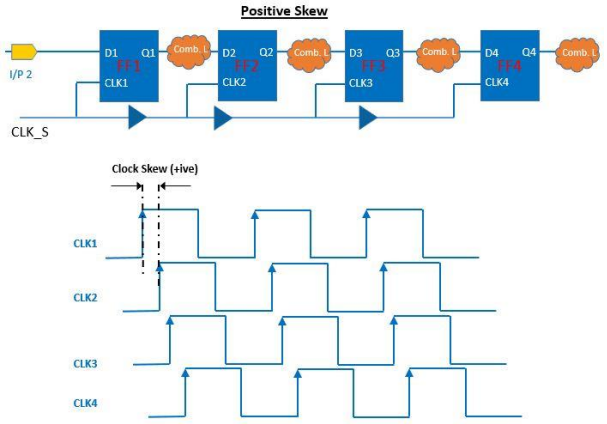
Bus is a communication channel shared by many devices and hence rules need to be established in order for the communication to happen correctly. These rules are called Bus protocols.

i)The two basic schemes for timing of data transfers over a bus can be classified into: Synchronous and Asynchronous

ii) Depending on whether the data bits are sent on parallel wires or multiplexed onto one single wire ,there are parallel (PCI) and serial (USB) buses.

Positive skew/Terminators

skew



Terminator

A device connected to one end of a bus or cable that absorbs signals. Terminators prevent signal reflection, which can produce interference that causes signal loss. Most communication systems such as networks and computer buses require some form of termination at the ends of the data path, although this is often provided internally by the devices at the ends of the data path

How Terminator Works

In a bus-based system, a single wire or series of wire segments connects network components in a chain formation. If the ends of the cable are not terminated, a signal placed on the wire by one component will bounce back and forth between the ends of the cable, hogging the cable and preventing other components from signaling. Terminators eliminate this signal bounce by absorbing the signal after each component has seen it once, allowing other components to place their signals on the cable.

Ex.Coaxial cabling,SCSI Adapter ,Stackable hubs (Free connectors on both ends of hubs)

Synchronous bus

- If a bus is synchronous (e.g. Processor-memory), it includes a clock in the control lines and a fixed protocol for communicating that is relative to the clock.
- *Every thing is synchronized to bus clock and every transaction takes one clock cycle.*
- *All master outputs valid on the rising edge of the CLK , stay valid through to the falling edge of the CLK.*
- *Slave output (for read) valid by falling edge of the CLK.*
- *Setup and hold times part of bus specification.*
- This type of protocol can be implemented easily for short buses.

Because the protocol is predetermined and involves little logic, the bus can run very fast and the interface logic will be small.

Synchronous buses have two major disadvantages:

First, every device on the bus must run at the same clock rate.

Second, because of clock skew problems, synchronous buses cannot be long if they are fast.

Synchronous bus

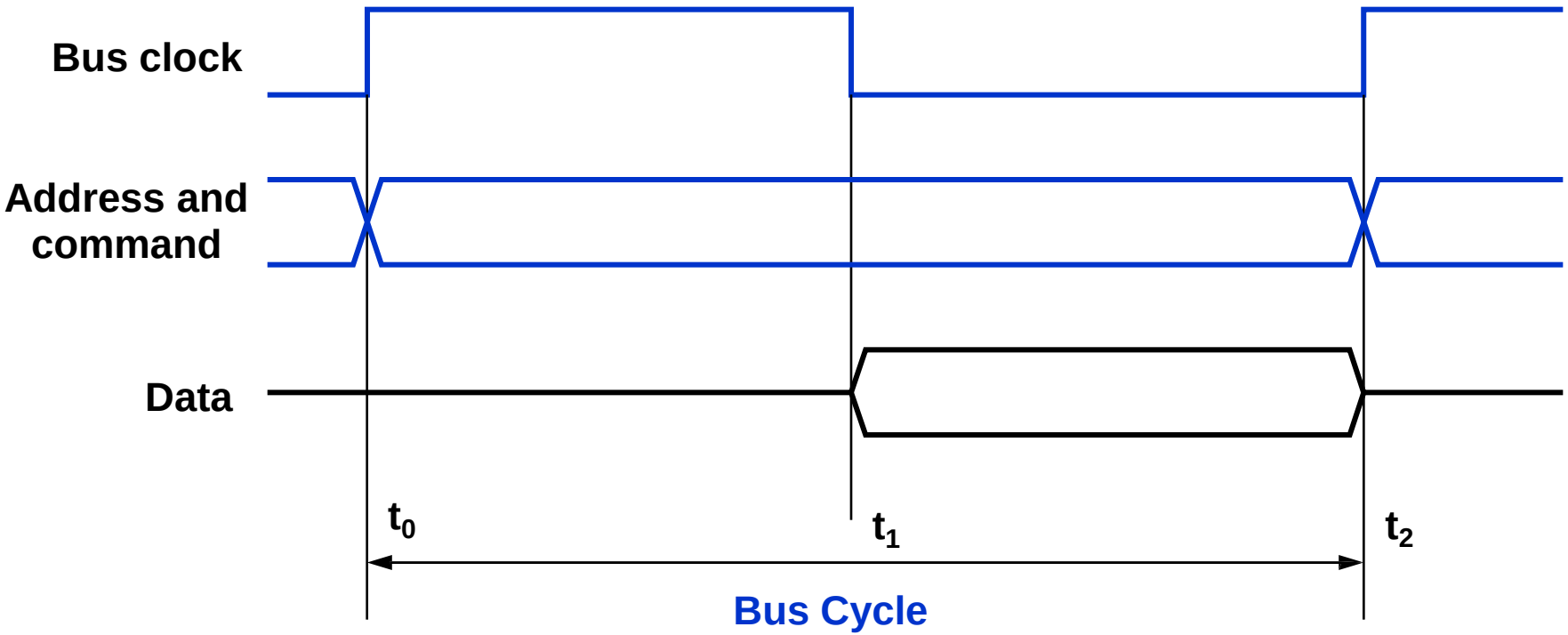
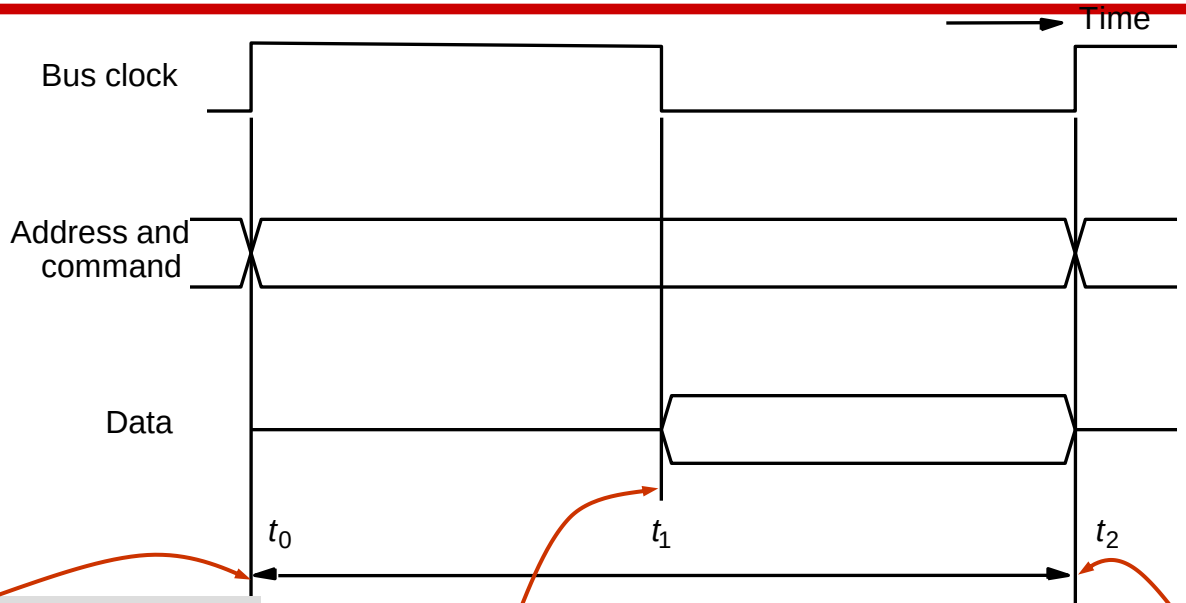


Fig.Timings of an input transfer on a synchronous bus

Synchronous Bus Timing



Master places the device address and command on the bus, and indicates that it is a Read operation.

Addressed slave places data on the data lines

Master "strobes" the data on the data lines into its input buffer, for a Read operation.

- In case of a Write operation, the master places the data on the bus along with the address and commands at time t_0.
- The slave strobes the data into its input buffer at time t_2.

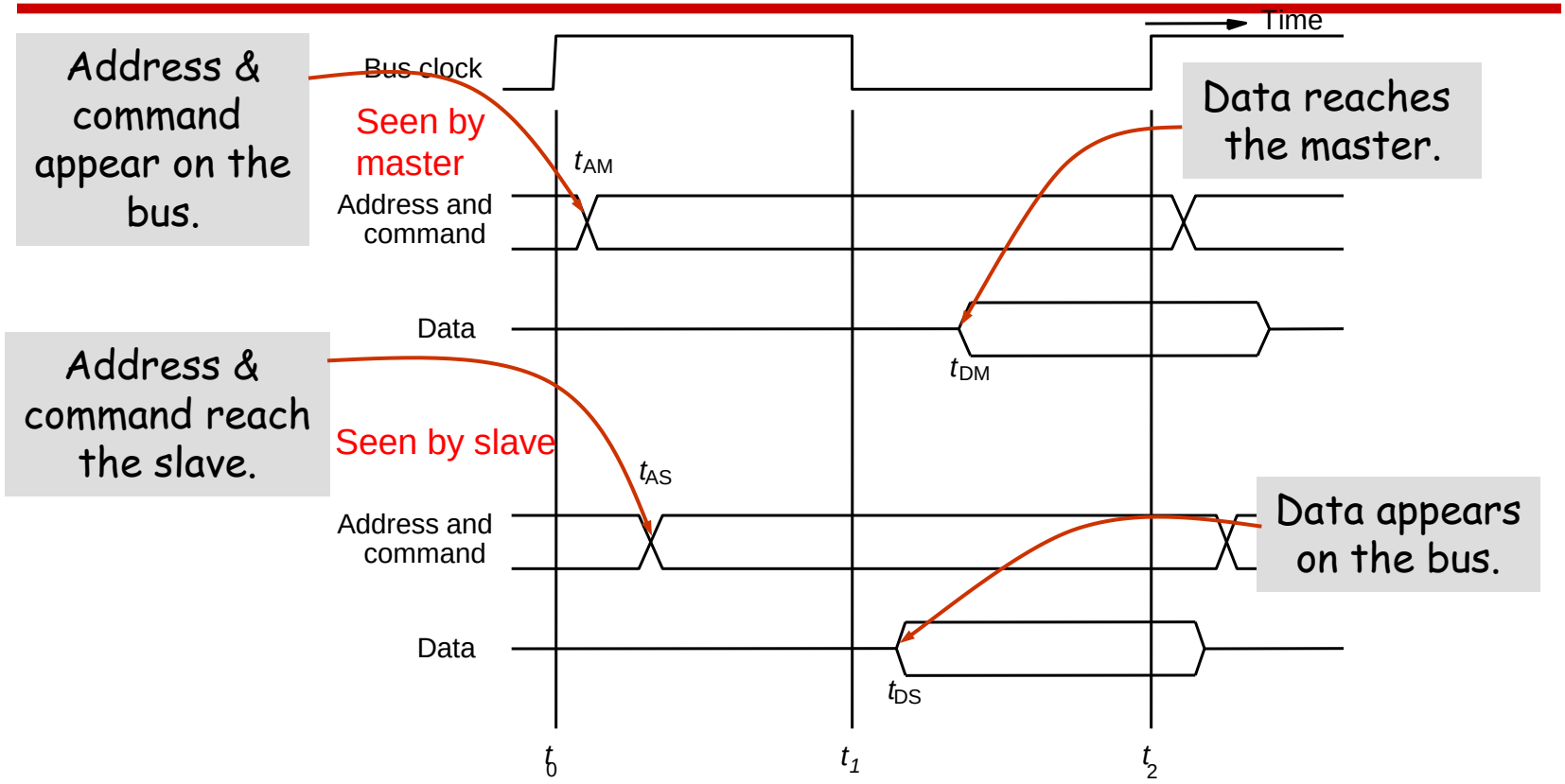
Synchronous bus (contd..)

- Once the master places the device address and command on the bus, it takes time for this information to propagate to the devices:
  - This time depends on the physical and electrical characteristics of the bus.
- Also, all the devices have to be given enough time to decode the address and control signals, so that the addressed slave can place data on the bus.
- Width of the pulse t_1 - t_0 depends on:
  - Maximum propagation delay between two devices connected to the bus.
  - Time taken by all the devices to decode the address and control signals, so that the addressed slave can respond at time t_1.

Synchronous bus (contd..)

- At the end of the clock cycle, at time t_2, the master strobes the data on the data lines into its input buffer if it's a Read operation.
  - "Strobe" means to capture the values of the data and store them into a buffer.
- When data are to be loaded into a storage buffer register, the data should be available for a period longer than the setup time of the device.
- Width of the pulse $t_2 - t_1$ should be longer than:
  - Maximum propagation time of the bus plus
  - Set up time of the input buffer register of the master.

Synchronous Bus Detailed Timing



- Signals do not appear on the bus as soon as they are placed on the bus, due to the propagation delay in the interface circuits.
- Signals reach the devices after a propagation delay which depends on the characteristics of the bus.
- Data must remain on the bus for some time after t_2 equal to the hold time of the buffer.

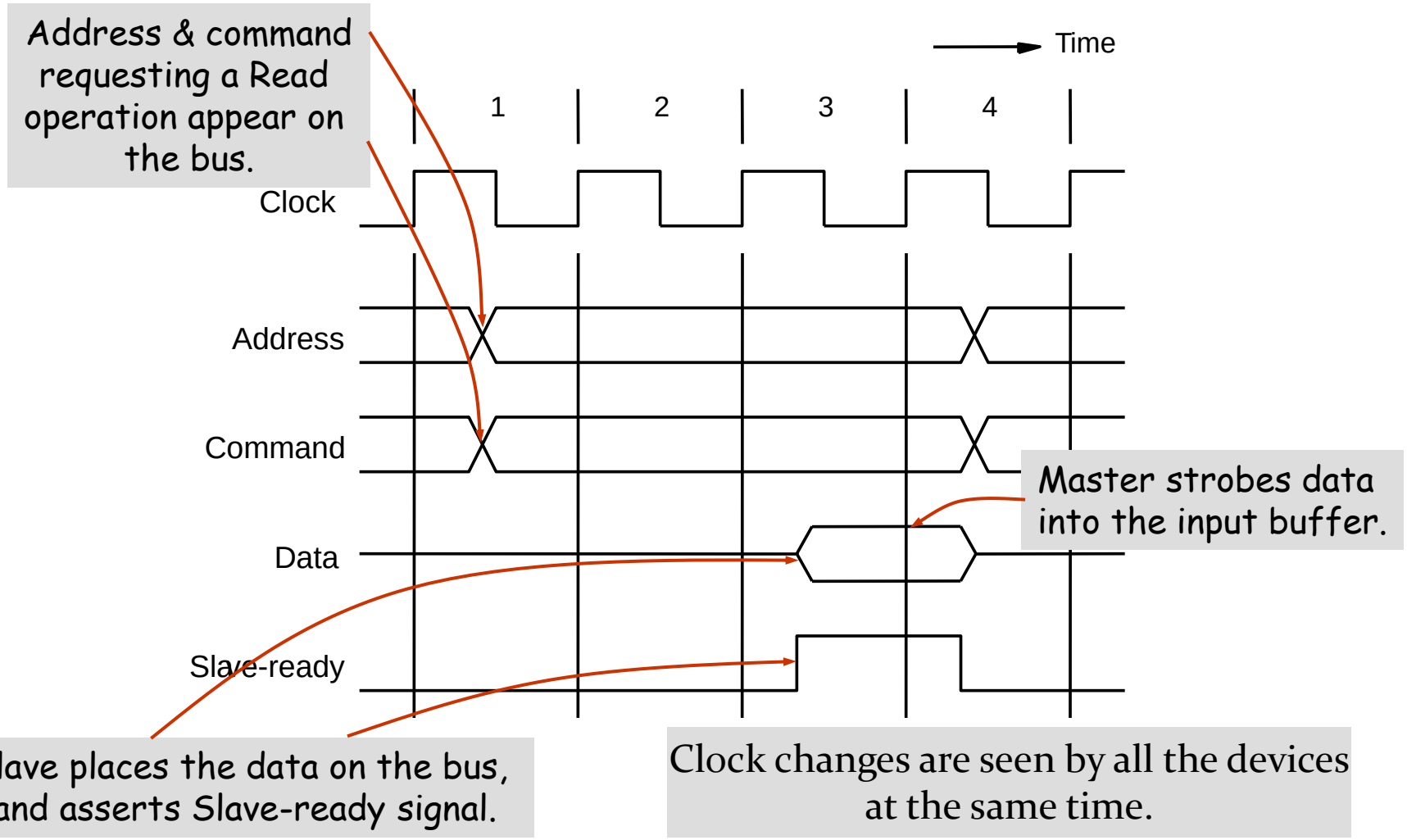
Synchronous bus (contd..)

- Data transfer has to be completed within one clock cycle.
  - Clock period $t_2 - t_0$ must be such that the longest propagation delay on the bus and the slowest device interface must be accommodated.
  - Forces all the devices to operate at the speed of the slowest device.
- Processor just assumes that the data are available at t2 in case of a Read operation, or are read by the device in case of a Write operation.
  - What if the device is actually failed, and never really responded?

Synchronous bus (contd..)

- Most buses have control signals to represent a response from the slave.
- Control signals serve two purposes:
  - Inform the master that the slave has recognized the address, and is ready to participate in a data transfer operation.
  - Enable to adjust the duration of the data transfer operation based on the speed of the participating slaves.
- High-frequency bus clock is used:
  - Data transfer spans several clock cycles instead of just one clock cycle as in the earlier case.

Synchronous bus- Multiple-Cycle Transfers



Asynchronous bus

i) Bus is not clocked.

All timings based on control signal edges.

Transmission is event driven.

ii) It can accommodate a wide variety of devices

iii) The bus can be lengthened without worrying about clock skew or synchronization problems.

iv) Data transfers on the bus is controlled by a handshake between the master and the slave.

Common clock in the synchronous bus case is replaced by two timing control lines:

- Master-ready (Strobe) and Slave-ready (Acknowledge)

Master-ready signal is asserted by the master to indicate to the slave that it is ready to participate in a data transfer.

Slave-ready signal is asserted by the slave in response to the master- ready from the master, and it indicates to the master that the slave is ready to participate in a data transfer.

v) Communication is fully interlocked; No device timing dependencies

Asynchronous bus (contd..)

Data transfer using the handshake protocol:

- Master places the address and command information on the bus.
- Asserts the Master-ready signal to indicate to the slaves that the address and command information has been placed on the bus.
- This causes all devices on the bus to decode the address.
- The selected slave performs the required operation, and informs the processor it has done so by asserting the Slave-ready signal.
- The master waits for slave ready to become asserted before it removes all its signals from the bus.
- If the operation is a Read , the Master also strobes the data into its input buffer.

Asynchronous Bus – Handshaking Protocol for Input Operation

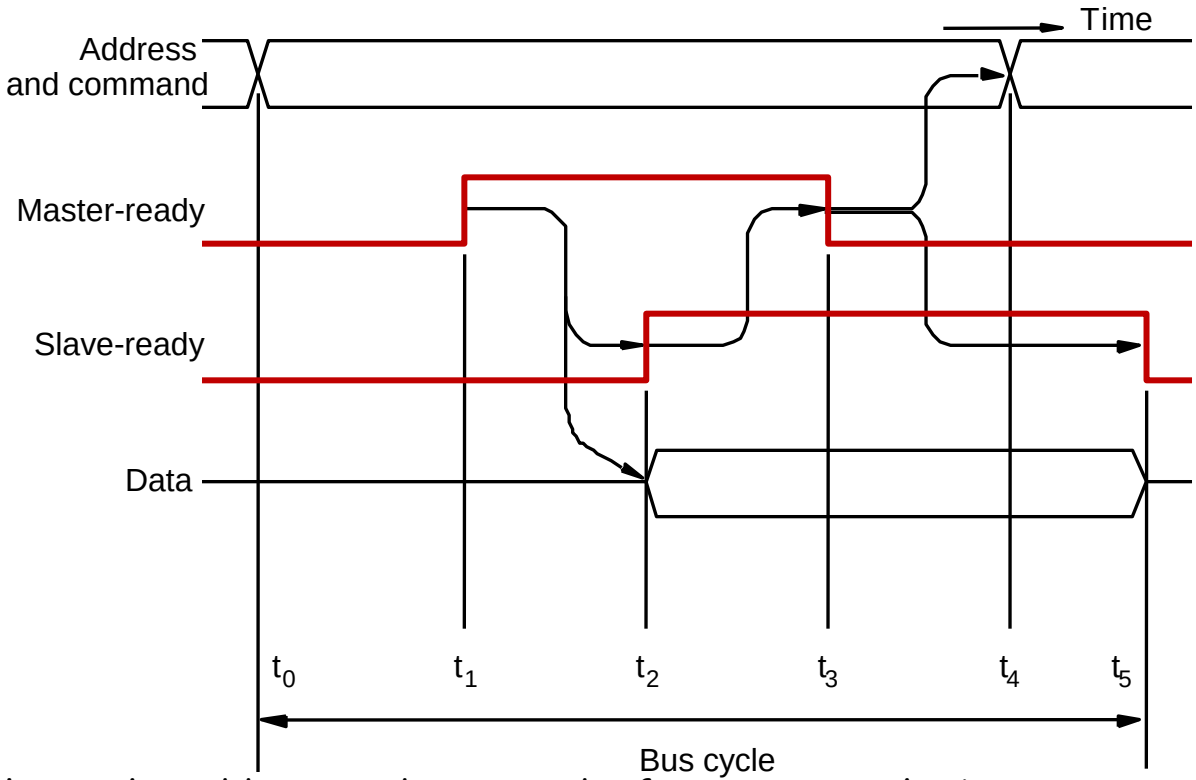
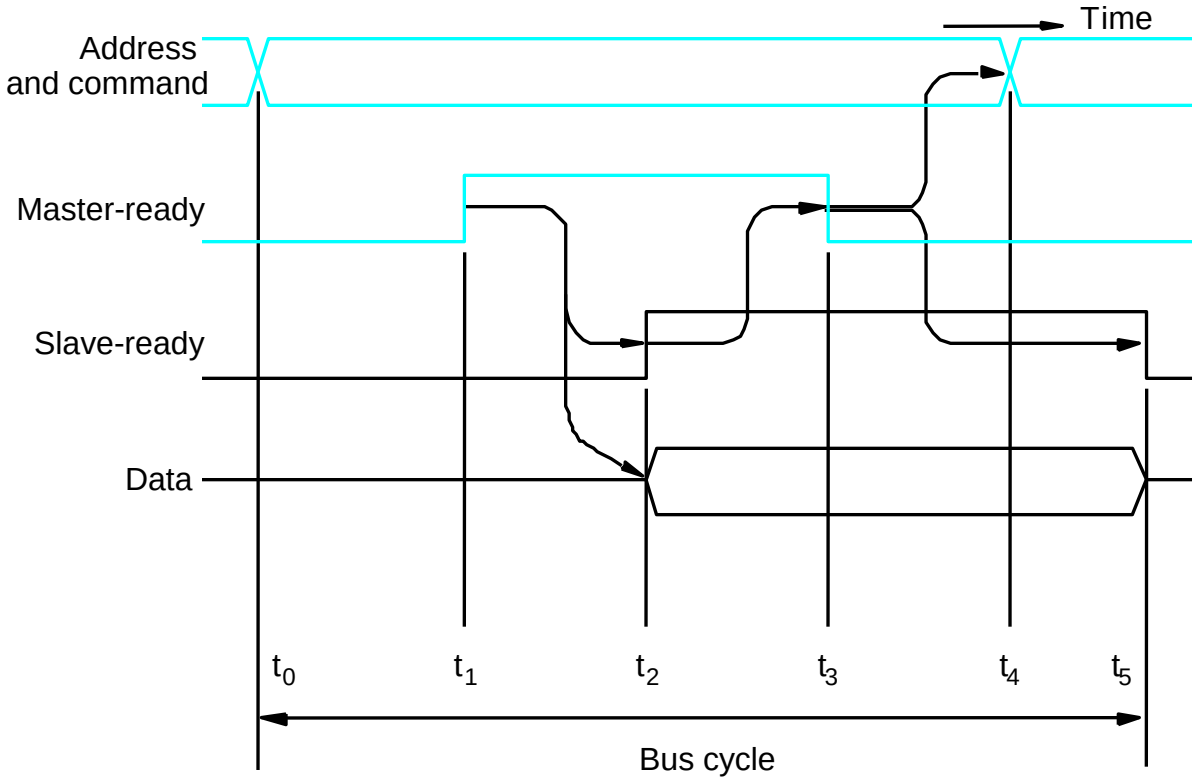


Fig. Handshake control of data transfer during an input operation.

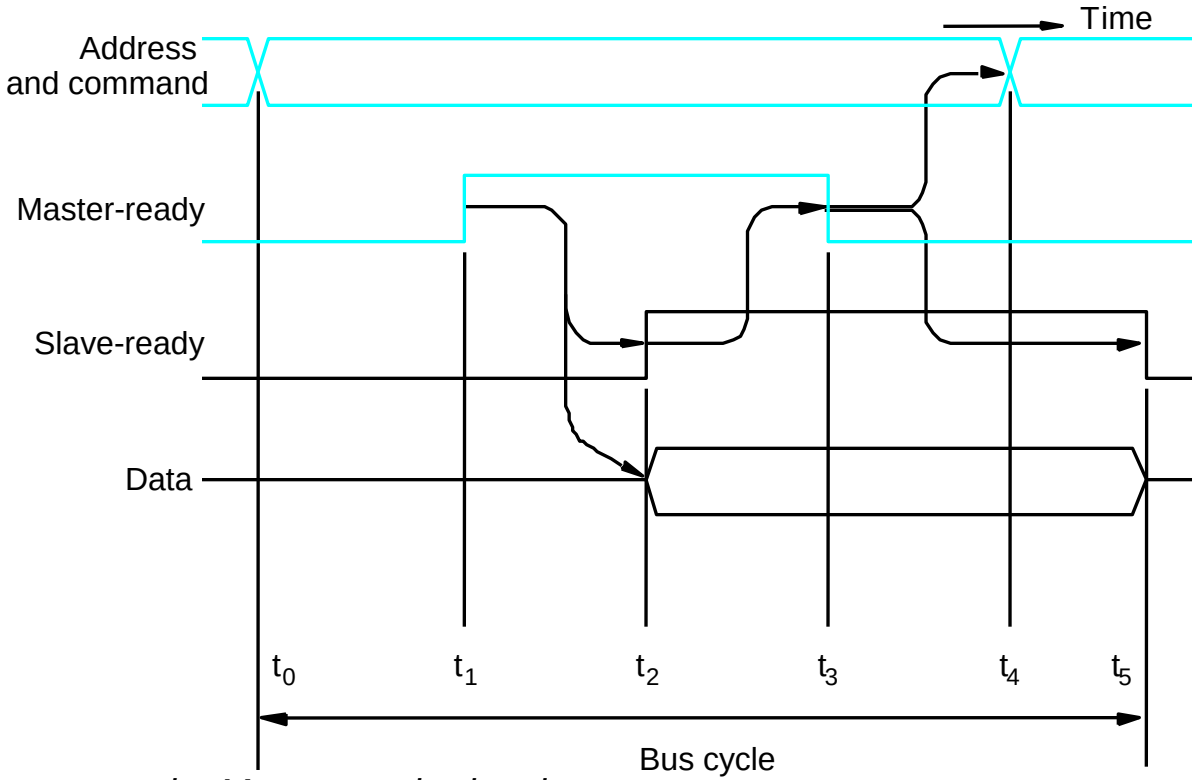
t_0 - Master places the address and command information on the bus.
t_1 - Master asserts the Master-ready
t_2 - Addressed slave places the data on the bus and asserts the Slave-ready signal.
t_3 - Slave-ready signal arrives at the master.
t_4 - Master removes the address and command information.
t_5 - Slave receives the transition of the Master-ready signal. It removes the data and the Slave-ready signal from the bus.

Asynchronous bus (contd..)



t_0 : Master places the address and command information on the bus. Command indicates that it is a Read operation (the data transfer from the device to the memory)

Asynchronous bus (contd..)



t_1 :Master asserts the Master-ready signal.

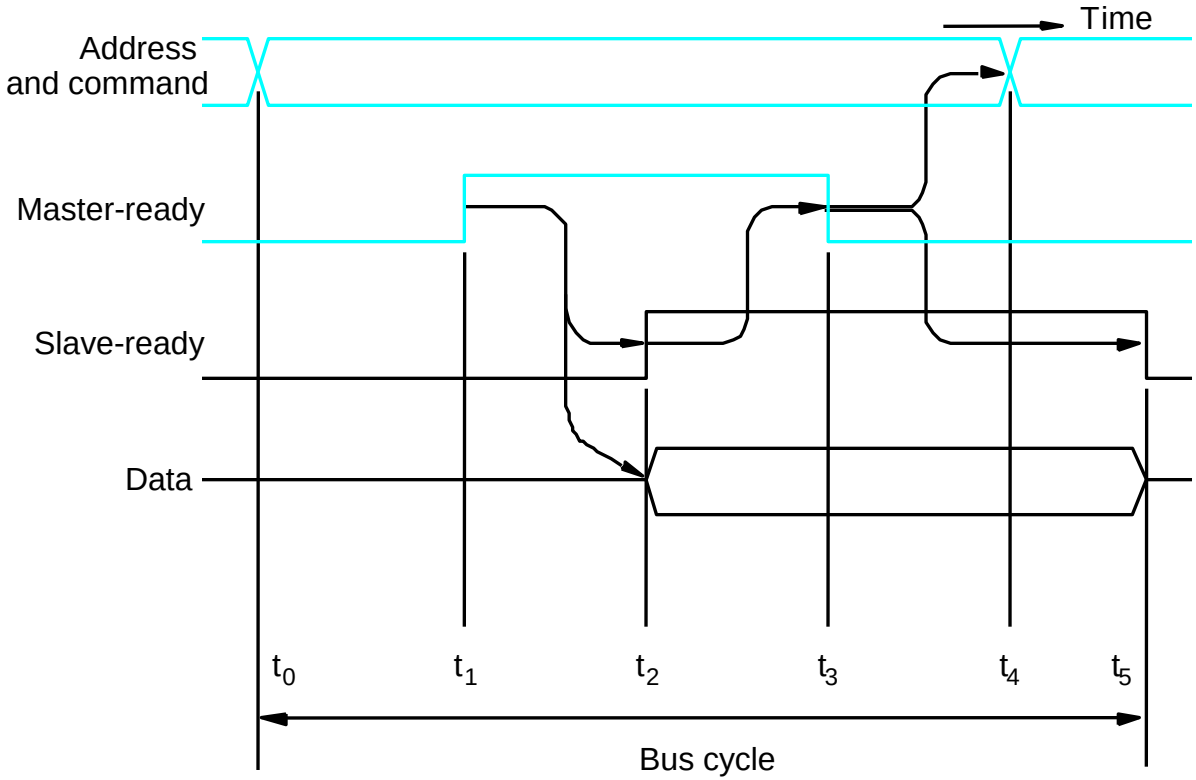
Master-ready signal is asserted at t_1 instead of t_0 to allow for bus skew. Bus skew occurs when two signals transmitted simultaneously reach the destination at different times).

This may occur because different bus lines may have different speeds. To guarantee that the master ready signal does not arrive at any device ahead of address and command information.

t_1-t_0 should be greater than maximum skew.

t_1-t_0 should also include the time taken to decode the address information.

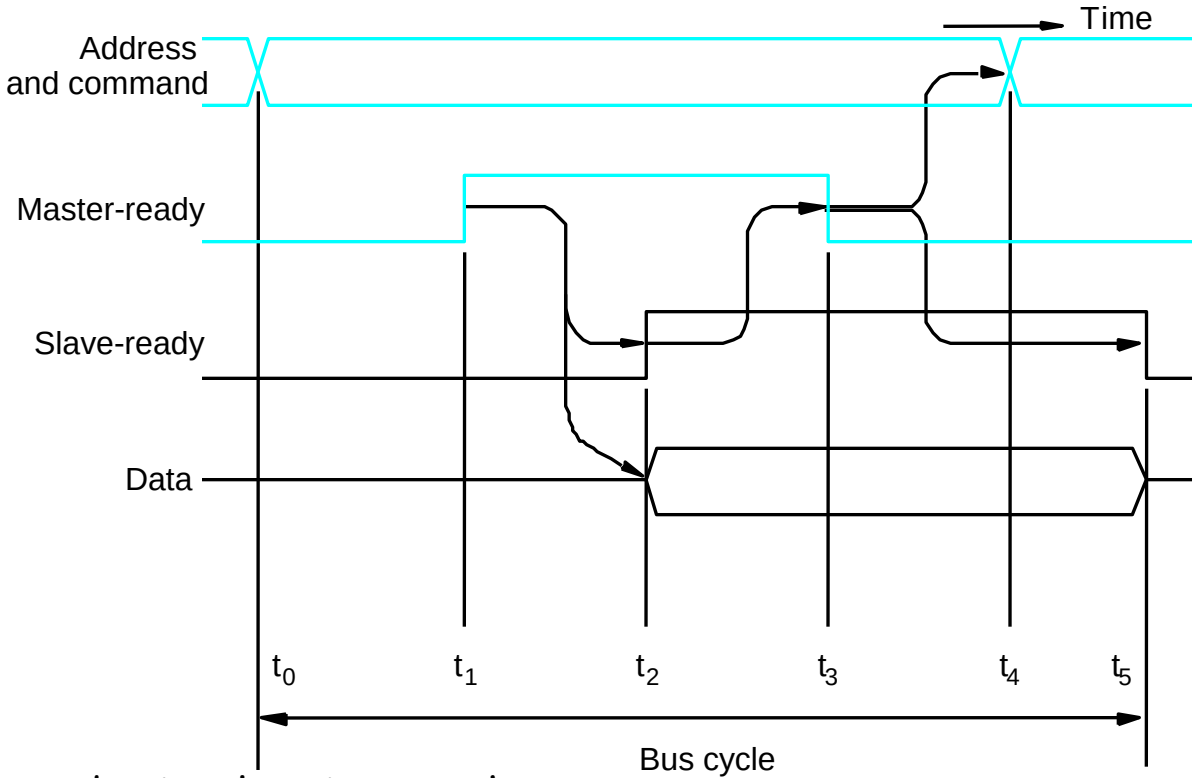
Asynchronous bus (contd..)



T_2 :The selected slave, having decoded the address and command info.performs the required input operation places the data from its data Register on to the data Bus and asserts the Slave-ready signal.

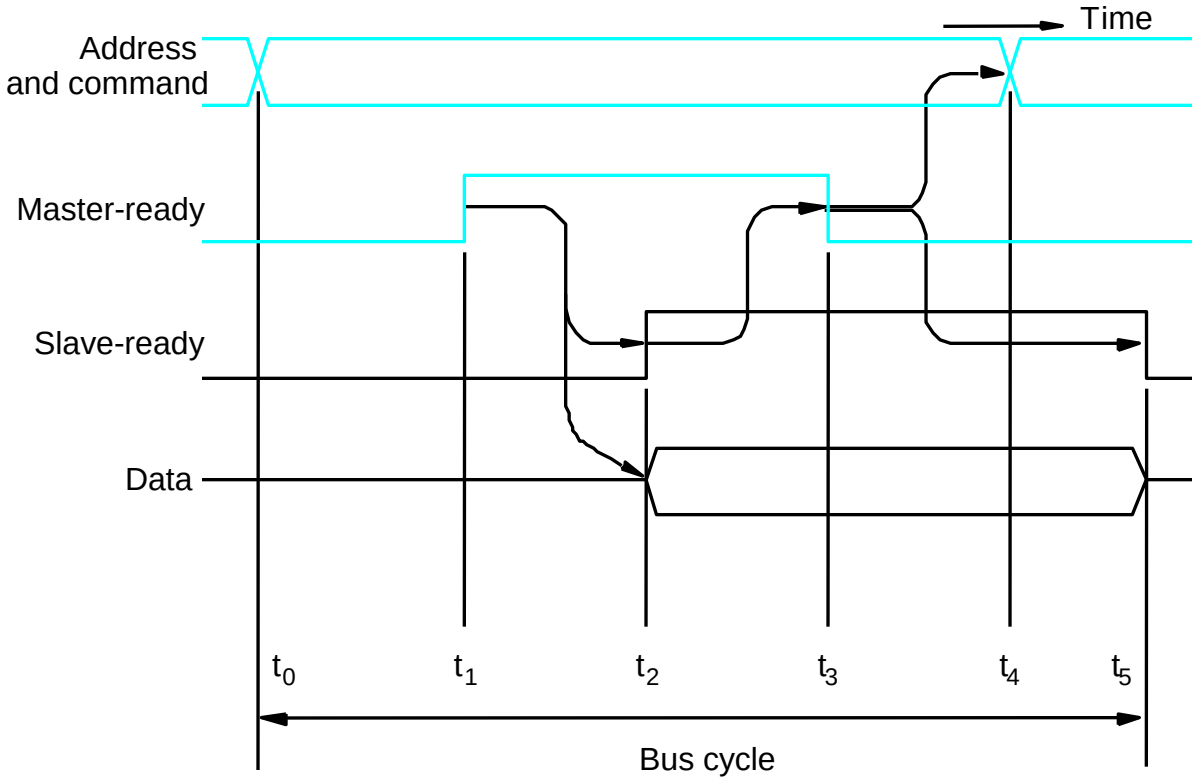
The period t_2-t_1, depends on the propagation delay between the master and the slave, and the delay in the slave's interface circuit.

Asynchronous bus (contd..)



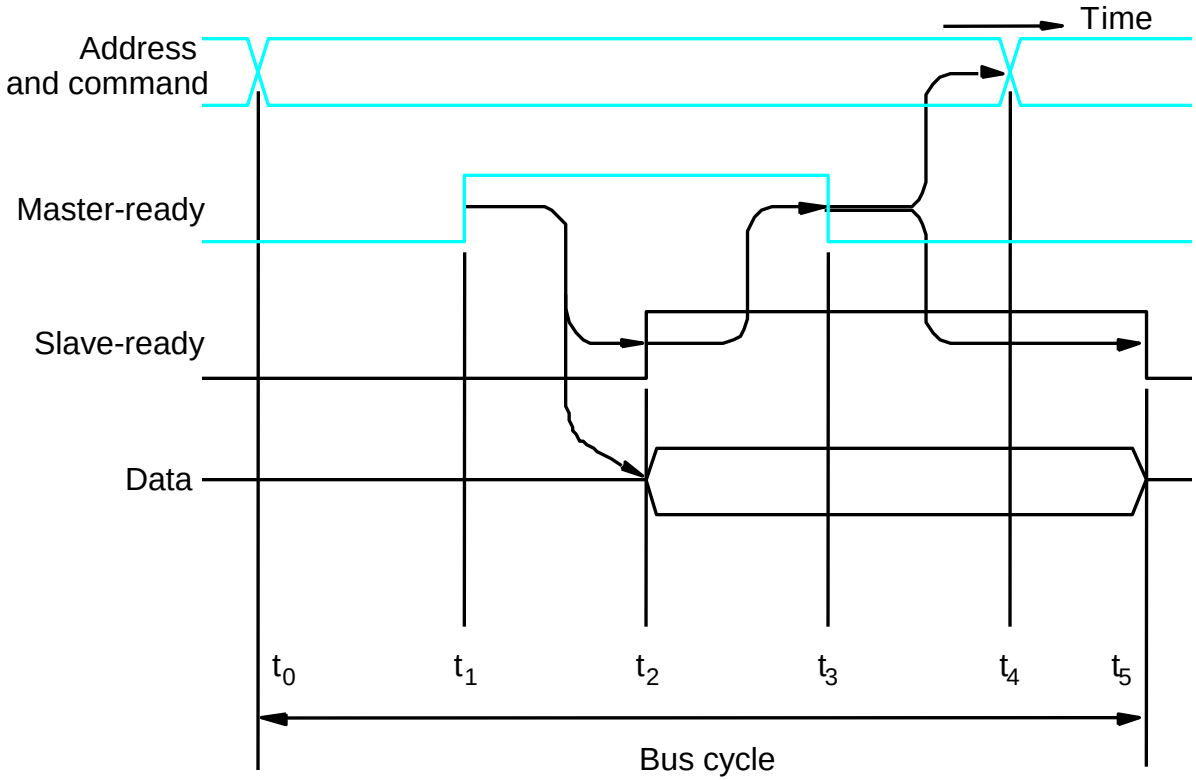
t_3 :Slave-ready signal arrives at the master.
Slave-ready signal was placed on the bus at the same time that data were placed on the bus. As a result, the data may also experience bus skew.
The master should wait for maximum bus skew plus the setup time of its input buffer and then strobes the data.
It also deactivates the Master-ready signal to indicate that it has received the data.

Asynchronous bus (contd..)



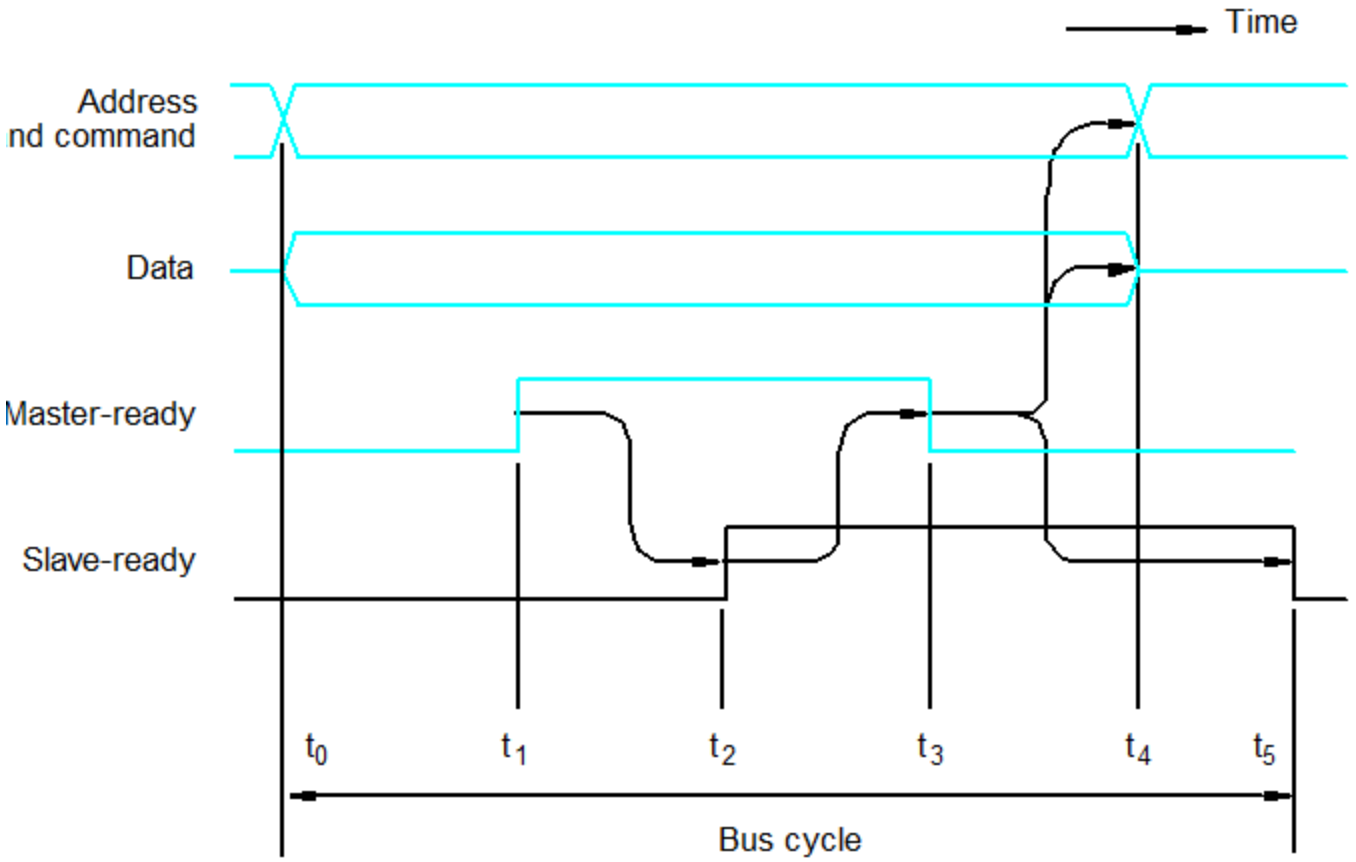
t_4 :Master removes the address and command information.
t_4-t_3 : allows for bus skew.
Once Master-ready signal is set to 0, it should reach all the devices before the address and command information is removed from the bus.

Asynchronous bus (contd..)



t_5 :Slave receives the transition of the Master-ready signal from 1 to 0. It removes the data and the Slave-ready signal from the bus.

Asynchronous Bus Handshaking Protocol for Output Operation



Discussion

The choice of a particular design involves trade-offs among factors such as

- Simplicity of the device interface
- Ability to accommodate device interfaces that introduce different amounts of delay
- Total time required for bus transfer
- Ability to detect errors results from addressing a nonexistent device or from an interface malfunction

Asynchronous bus

The handshake process eliminates the need for synchronization of the sender and receiver clock, thus simplifying timing design

Synchronous bus

Clock circuitry must be designed carefully to ensure proper synchronization, and delays must be kept within strict bounds

Asynchronous vs. Synchronous bus

- Advantages of asynchronous bus:
  - Eliminates the need for synchronization between the sender and the receiver.
  - Can accommodate varying delays automatically, using the Slave-ready signal.
  - Ability to detect errors resulting from addressing a nonexistent device or from an interface malfunction
  - Allows the devices to transfer at any speed by allowing each device to signal the end of a unit of transfer
- Disadvantages of asynchronous bus:
  - Slower data transfer rate-transfer rate with full handshake is limited by two-round trip delays.
  - Data transfers using a synchronous bus involves only one round trip delay, and hence a synchronous bus can achieve faster rates.

Interface Circuits

Function of I/O Interface

- Provide a storage buffer for at least one word of data;
- Contain status flags that can be accessed by the processor to determine whether the buffer is full or empty;
- Contain address-decoding circuitry to determine when it is being addressed by the processor;
- Generate the appropriate timing signals required by the bus control scheme;
- Perform any format conversion that may be necessary to transfer data between the bus and the I/O device.

Interface circuits

I/O interface consists of the circuitry required to connect an I/O device to a computer bus.

Side of the interface which connects to the computer has bus signals for: (input Interface)

- Address,
- Data
- Control

Side of the interface which connects to the I/O device has: (Output Interface)

- Datapath and associated controls to transfer data between the interface and the I/O device.
- This side is called as a "port".

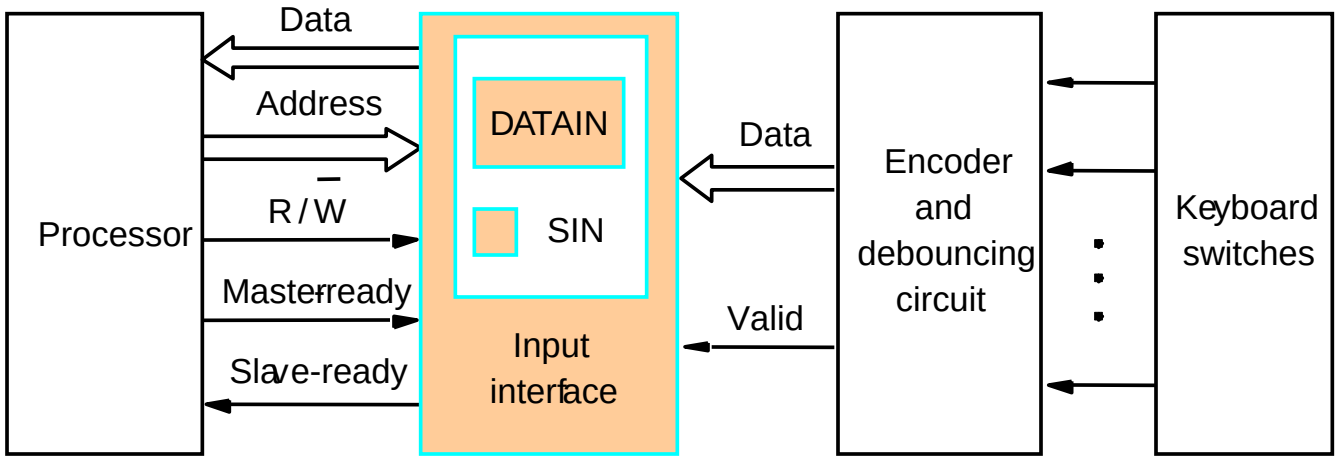
Ports can be classified into two:

- Parallel port,
- Serial port.

Interface circuits (contd..)

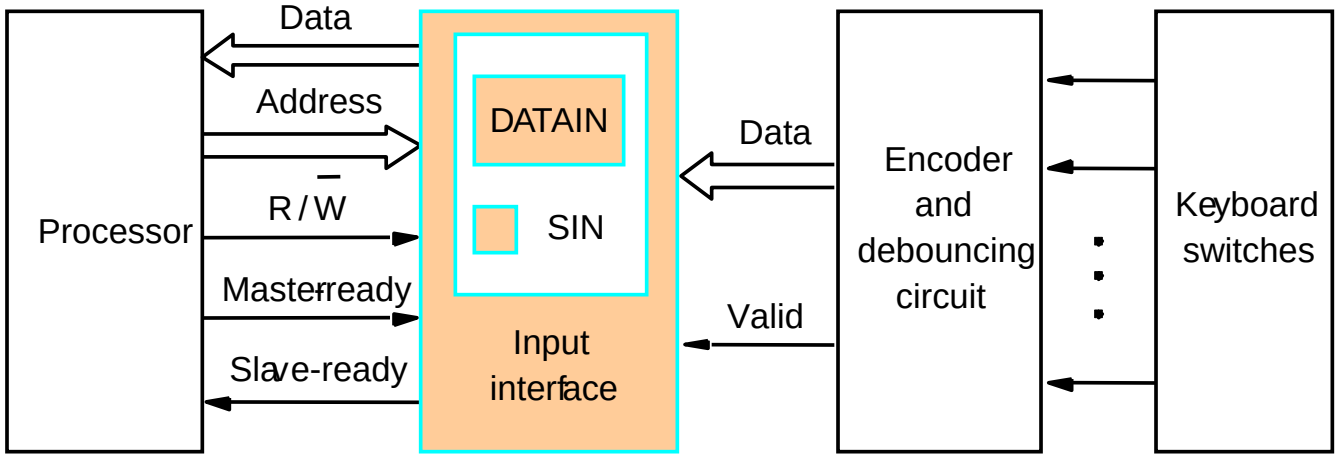
- Parallel port transfers data in the form of a number of bits, normally 8 or 16 to or from the device.
- Serial port transfers and receives data one bit at a time.
- Processor communicates with the bus in the same way, whether it is a parallel port or a serial port.
  - Conversion from the parallel to serial and vice versa takes place inside the interface circuit.

Parallel port
Input Interface (Keyboard to Processor Connection)



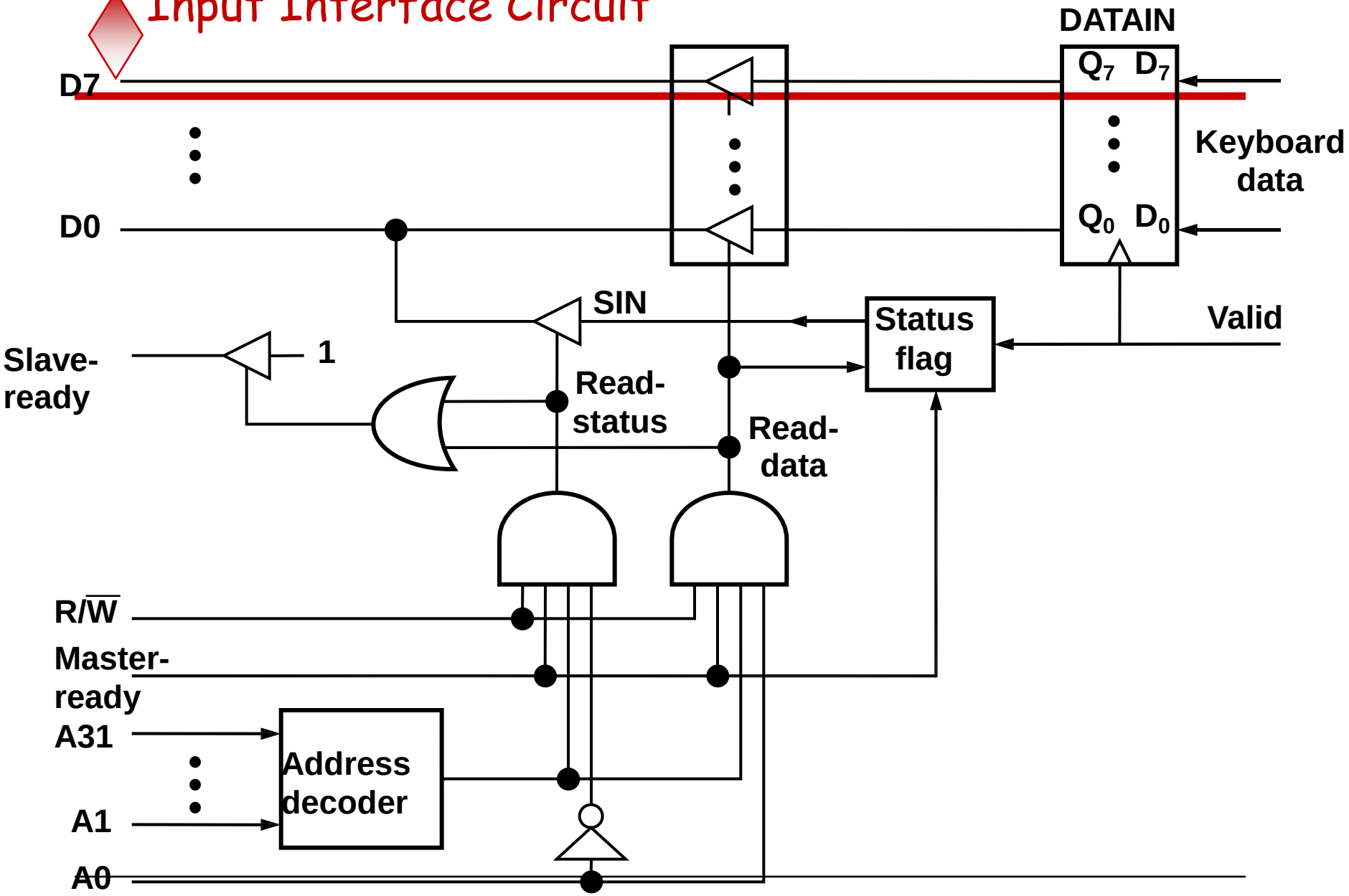
- Keyboard is connected to a processor using a parallel port.
- Processor is 32-bits and uses memory-mapped I/O and the asynchronous bus protocol.
- On the processor side of the interface we have:
  - Data lines.
  - Address lines
  - Control or R/W line.
  - Master-ready signal and
  - Slave-ready signal.

Parallel Port – Output Interface (Printer to Processor Connection)



- On the keyboard side of the interface:
  - Encoder circuit which generates a code for the key pressed.
  - Debouncing circuit which eliminates the effect of a key bounce (a single key stroke may appear as multiple events to a processor).
- Data lines contain the code for the key.
- Valid line changes from 0 to 1 when the key is pressed. This causes the code to be loaded into DATAIN and SIN to be set to 1.

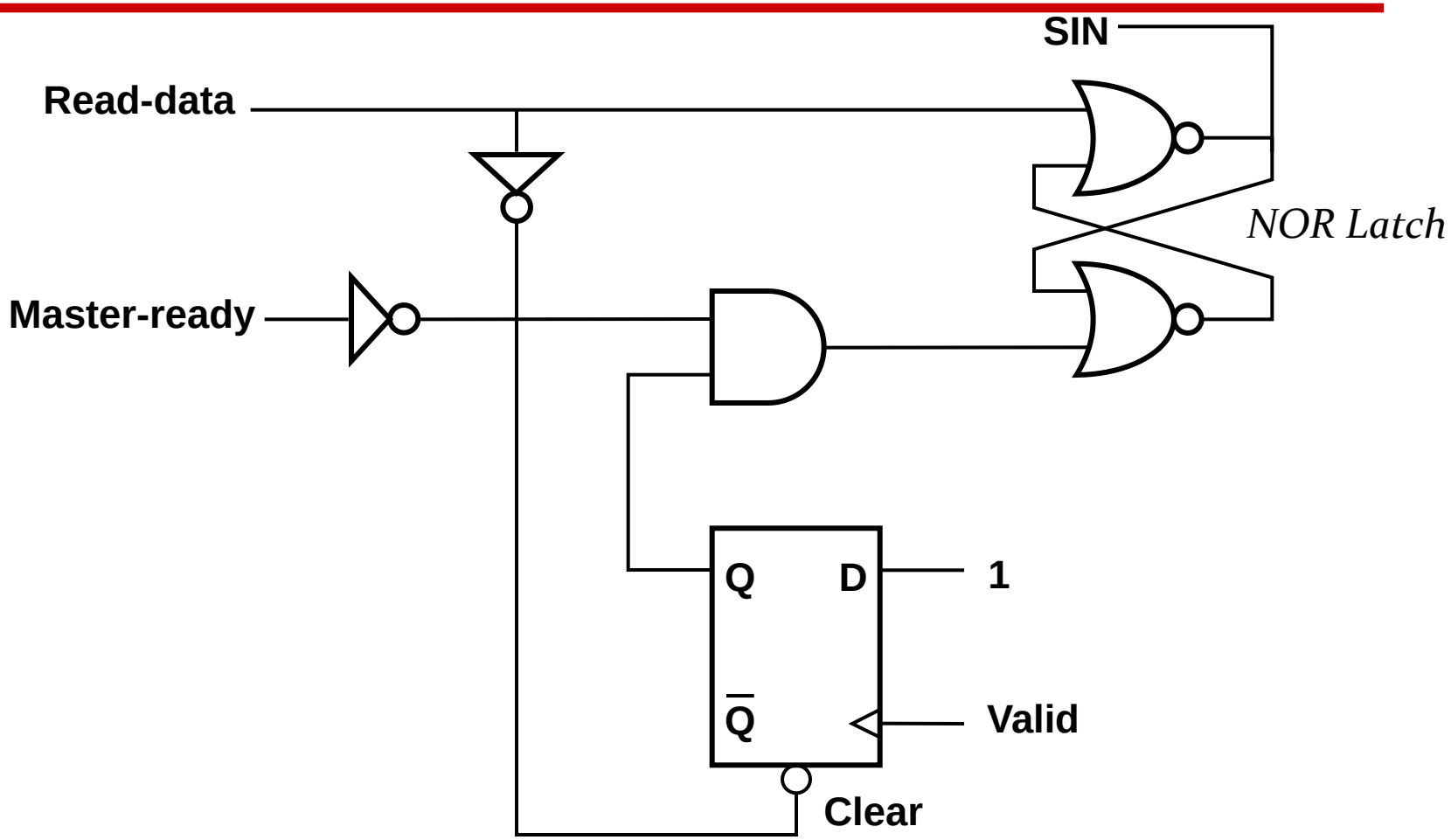
Input Interface Circuit



Input Interface Circuit

- Output lines of DATAIN are connected to the data lines of the bus by means of 3 state drivers
- Drivers are turned on when the processor issues a read signal and the address selects this register.
- SIN signal is generated using a status flag circuit.
- It is connected to line D_0 of the processor bus using a three-state driver.
- Address decoder selects the input interface based on bits A_1 through A_{31}.
- Bit A0 determines whether the status or data register is to be read, when Master-ready is active.
- In response, the processor activates the Slave-ready signal, when either the Read-status or Read-data is equal to 1, which depends on line A_0.

Circuit for the status flag block

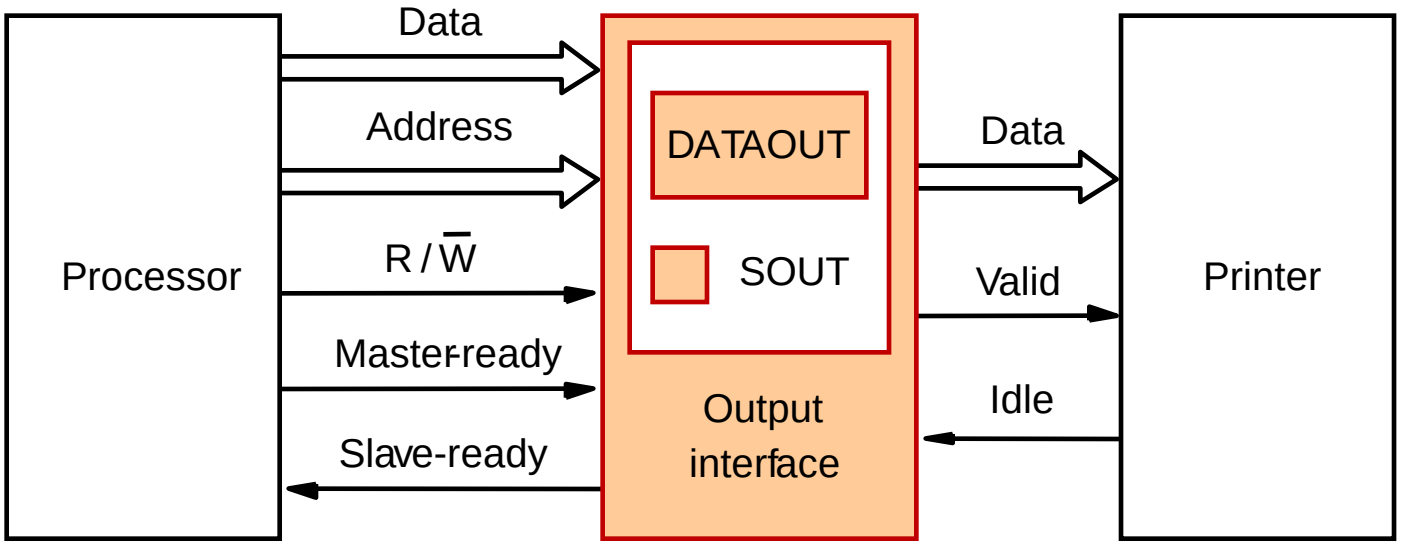


Parallel Port – Input Interface –status flag (Keyboard to Processor Connection)

Possible Implementation of Status Flag.

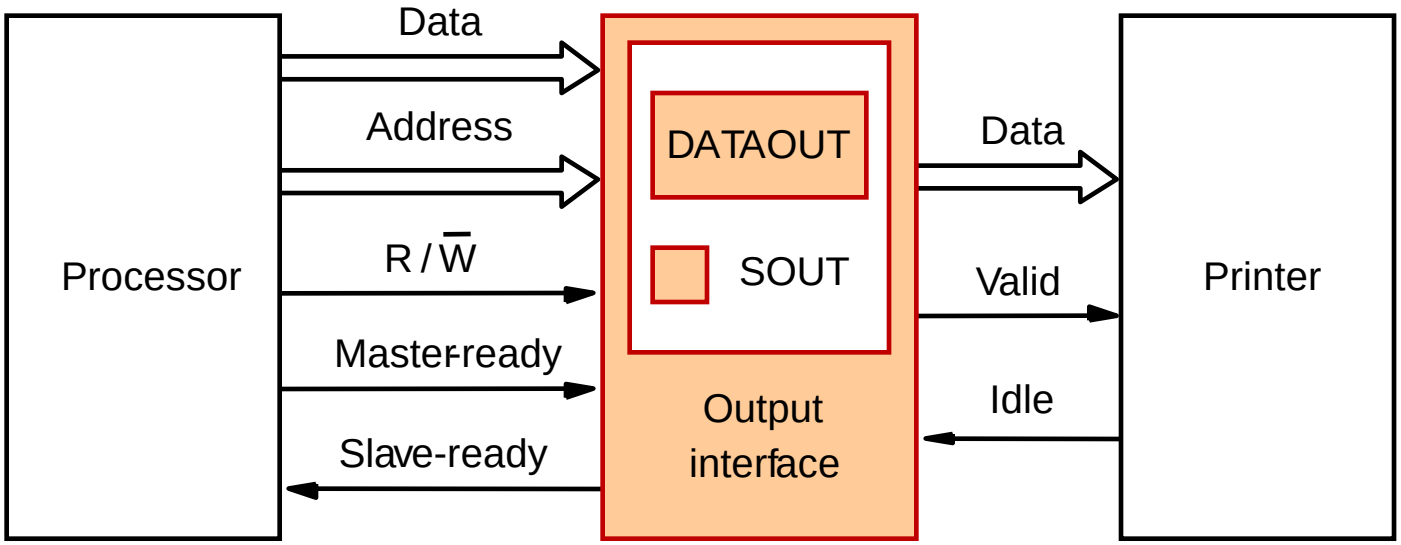
1. D flip-flop is set to 1 by a rising edge on valid signal line.
2. Then NOR latch SIN is set to 1.
3. As the state of SIN is to be 1 for the CPU to act, it ensure that SIN can be set only when Master –ready is equal to 1.

Parallel port- Printer to Processor Connection



- Printer is connected to a processor using a parallel port.
- Processor is 32 bits, uses memory-mapped I/O and asynchronous bus protocol.
- On the processor side:
  - Data lines.
  - Address lines
  - Control or R/W line.
  - Master-ready signal and
  - Slave-ready signal.

Parallel port -Printer to Processor Connection



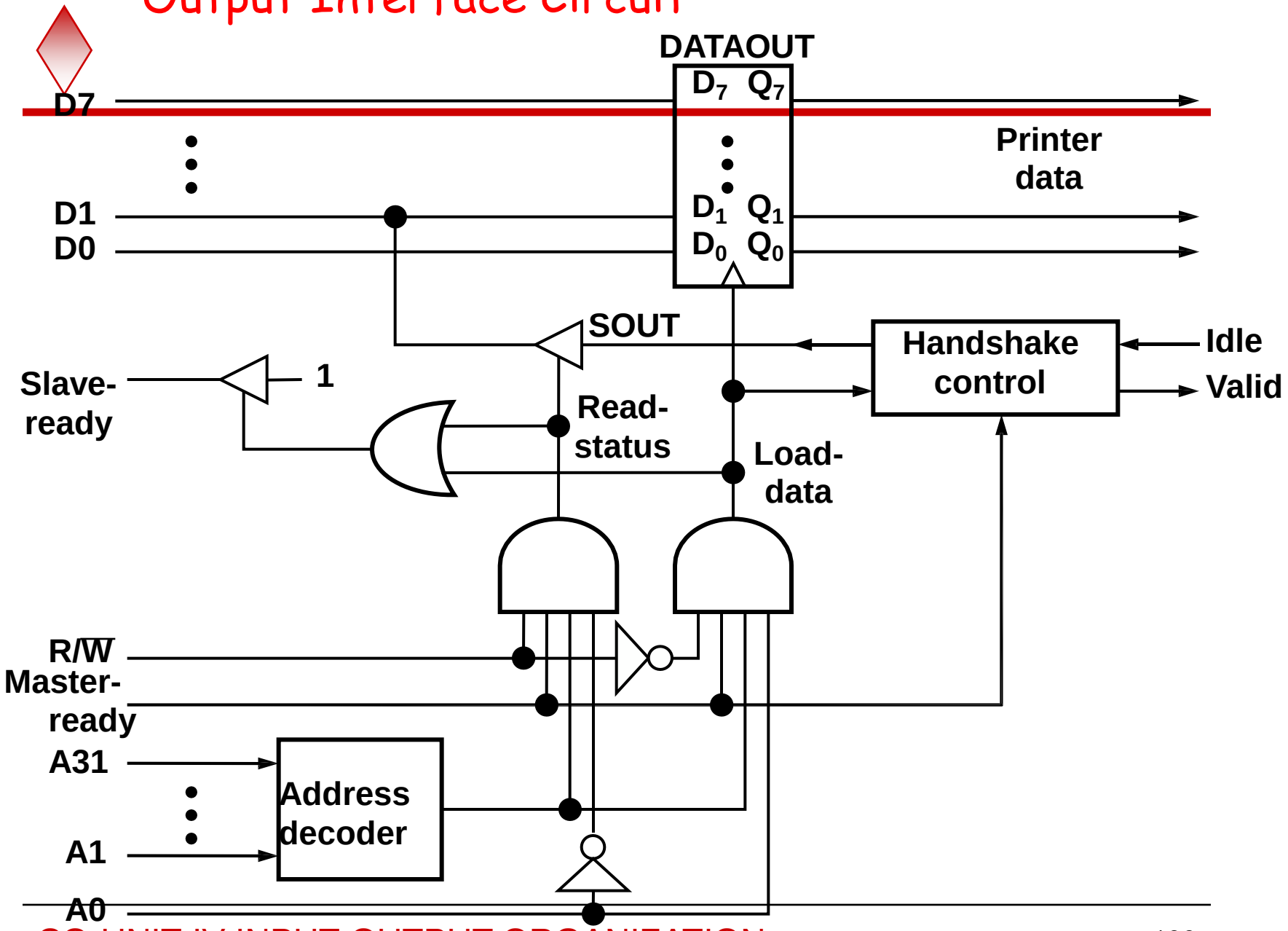
➢On the printer side:

- Idle signal line which the printer asserts when it is ready to accept a character.

This causes the SOUT flag to be set to 1.

- Processor places a new character into a DATAOUT register.
- Valid signal, asserted by the interface circuit when it places a new character on the data lines.

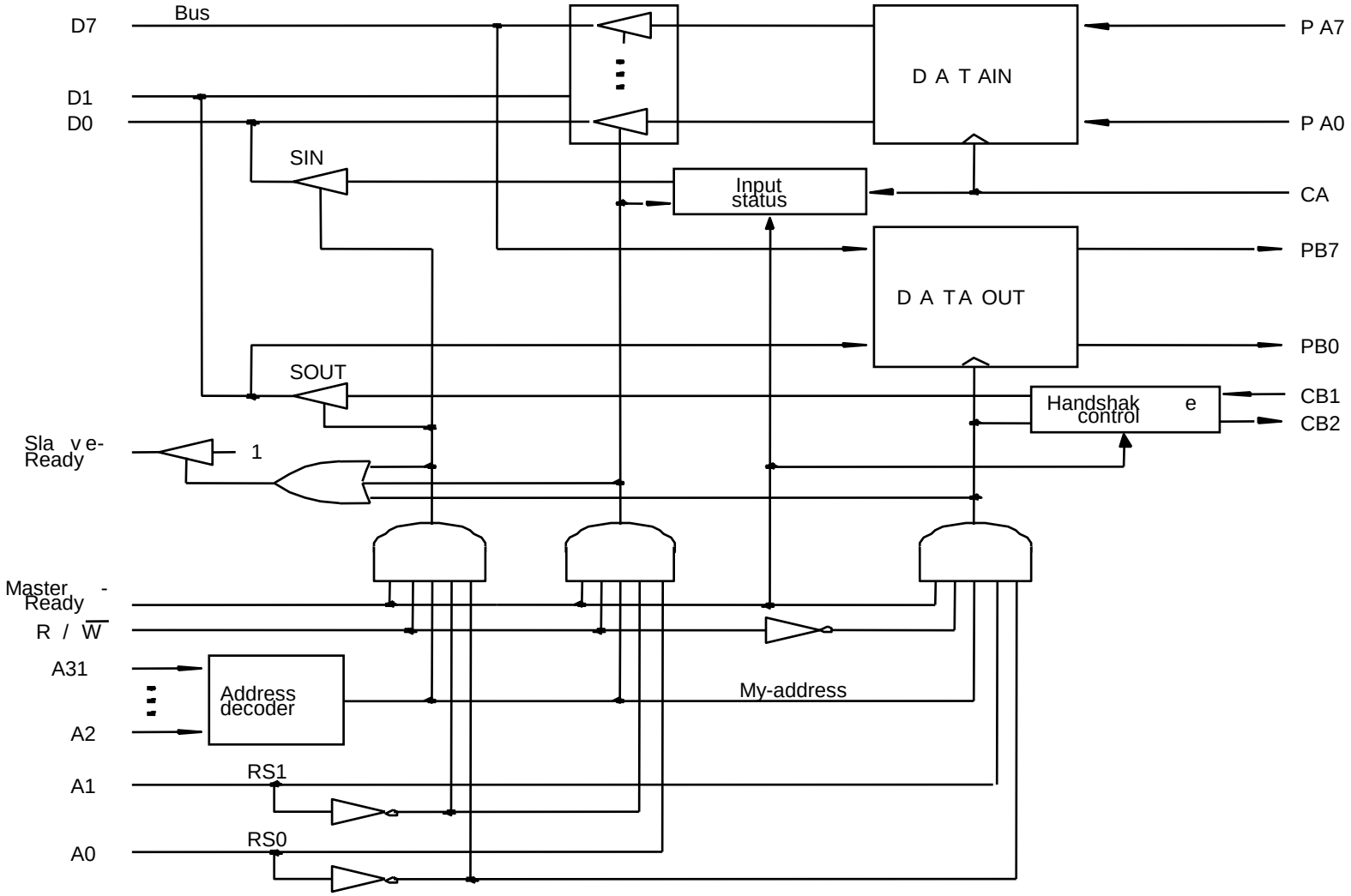
Output Interface Circuit



Output Interface Circuit

- Data lines of the processor bus are connected to the DATAOUT register of the interface.
- The status flag SOUT is connected to the data line D1 using a three-state driver.
- The three-state driver is turned on, when the control Read-status line is 1
- Address decoder selects the output interface using address lines A1 through A31.
- Address line A0 determines whether the data is to be loaded into the DATAOUT register or status flag is to be read.
- If the Load-data line is 1, then the Valid line is set to 1.
- If the Idle line is 1, then the status flag SOUT is set to 1.

Combined input/output interface circuit

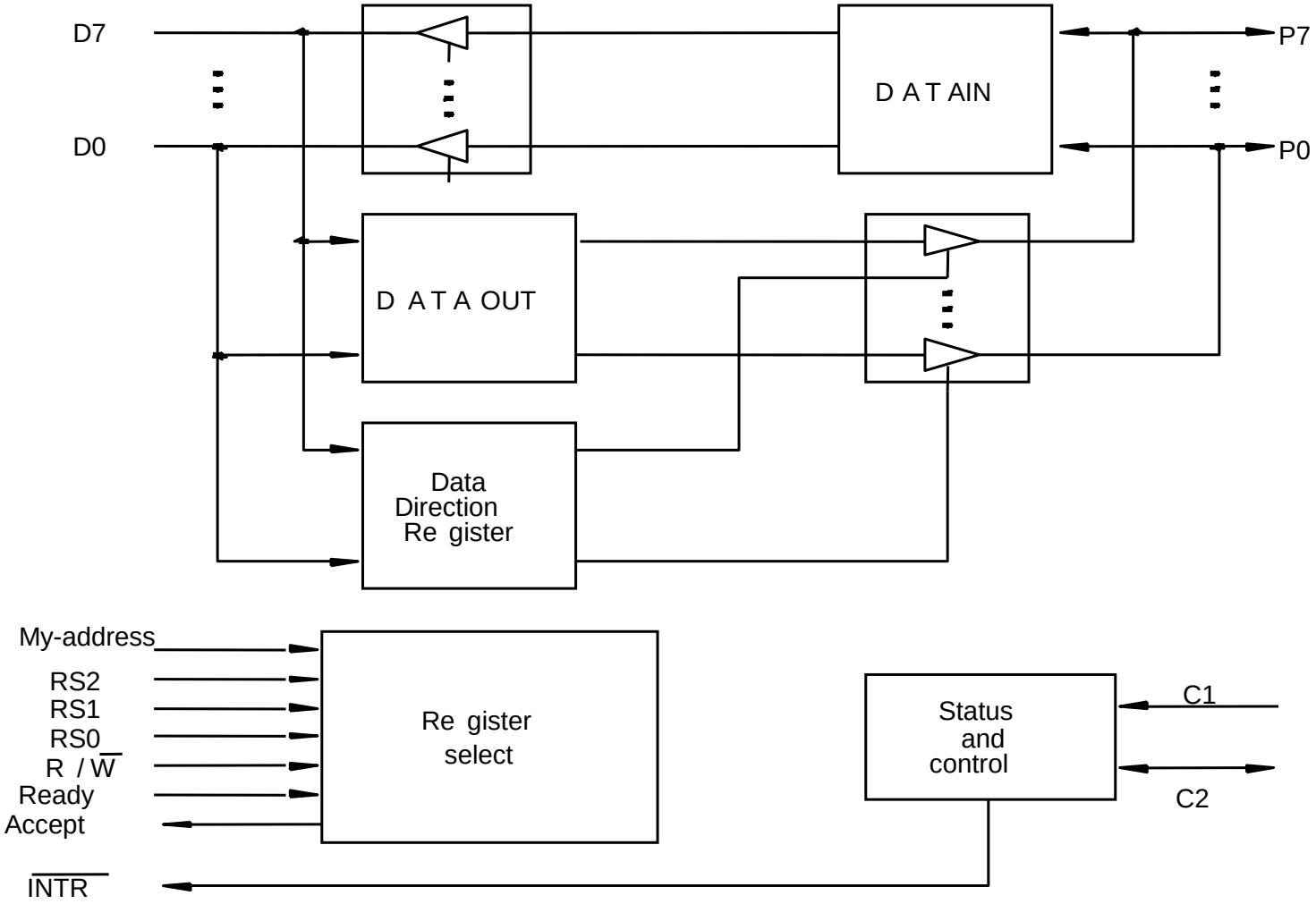


Combined input/output interface circuit

Combined I/O interface circuit.

- Address bits A2 through A31, that is 30 bits are used to select the overall interface.
- Address bits A1 through A0, that is, 2 bits select one of the three registers, namely, DATAIN, DATAOUT, and the status register.
- Status register contains the flags SIN and SOUT in bits 0 and 1.
- Data lines PA0 through PA7 connect the input device to the DATAIN register.
- DATAOUT register connects the data lines on the processor bus to lines PB0 through PB7 which connect to the output device.
- Separate input and output data lines for connection to an I/O device.

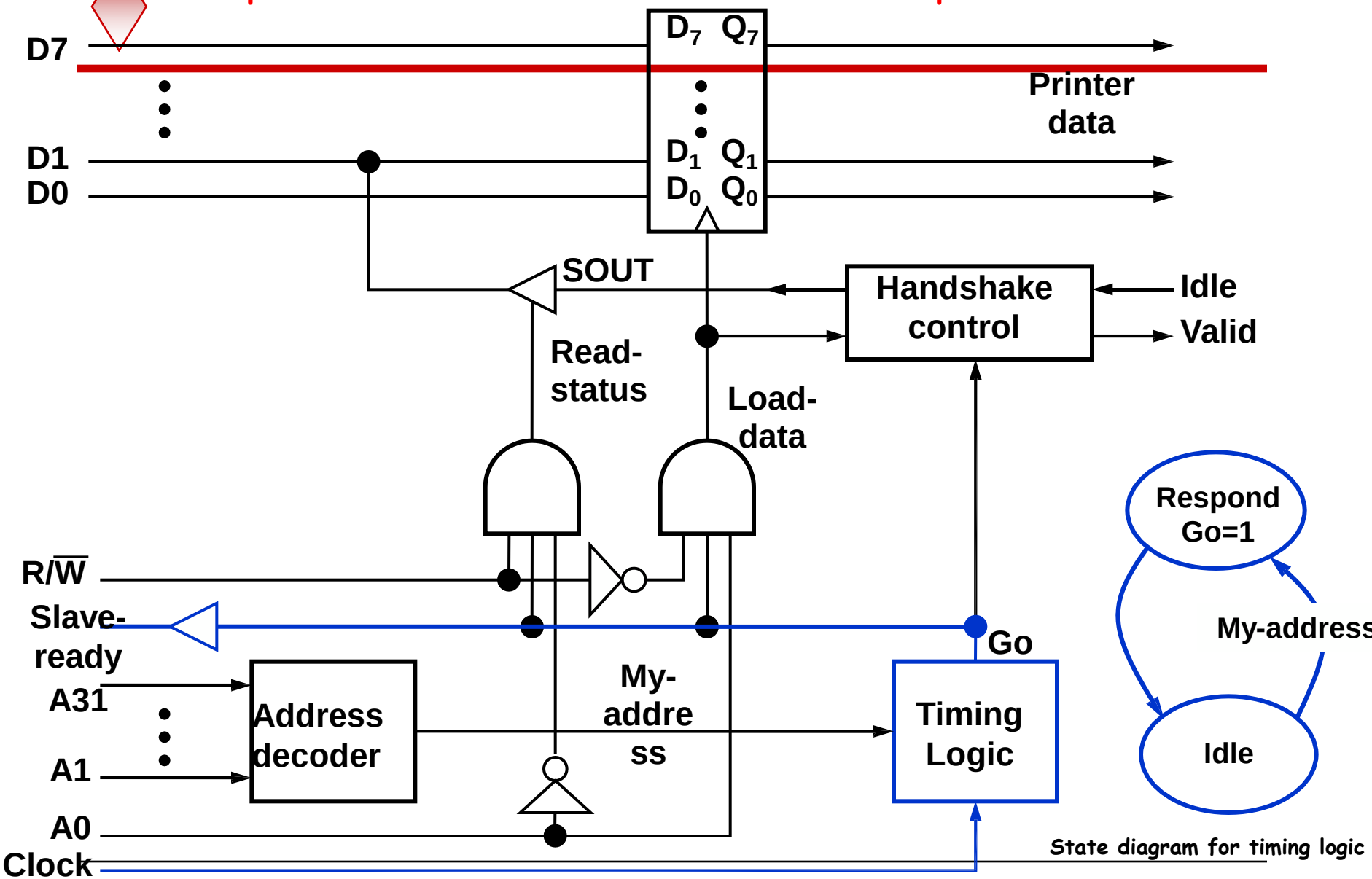
A General 8 Bit Parallel Interface



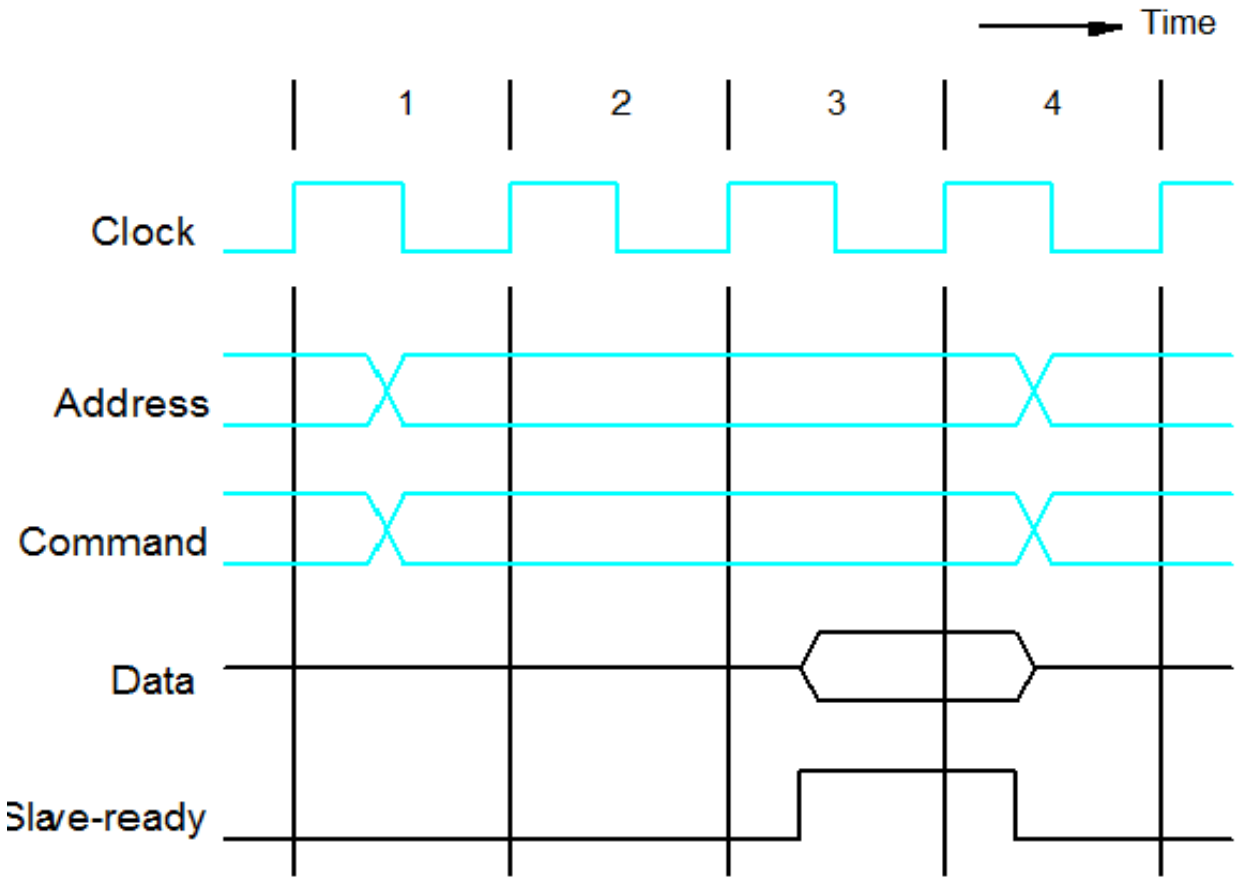
A General 8 Bit Parallel Interface

- Data lines to I/O device are bidirectional.
- Data lines P7 through P0 can be used for both input, and output.
- In fact, some lines can be used for input & some for output depending on the pattern in the Data Direction Register (DDR).
- Processor places an 8-bit pattern into a DDR.
- If a given bit position in the DDR is 1, the corresponding data line acts as an output line, otherwise it acts as an input line.
- C1 and C2 control the interaction between the interface circuit and the I/O devices.
- Ready and Accept lines are the handshake control lines on the processor bus side, and are connected to Master-ready & Slave-ready.
- Input signal My-address is connected to the output of an address decoder.
- Three register select lines that allow up to 8registers to be selected.

Output Interface Circuit for a Bus protocol

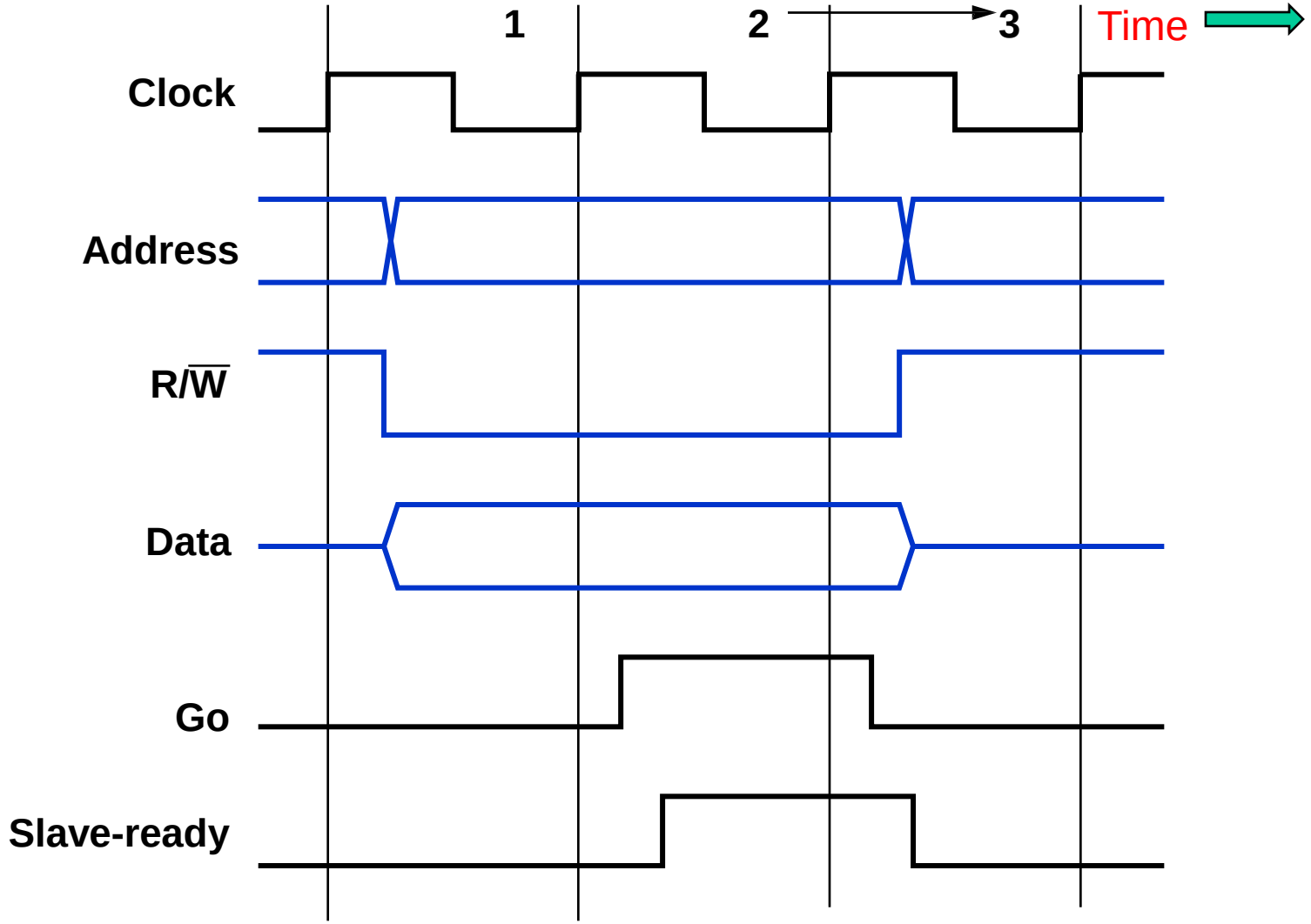


Input transfer using multiple clock cycles



An input transfer using multiple clock cycles.

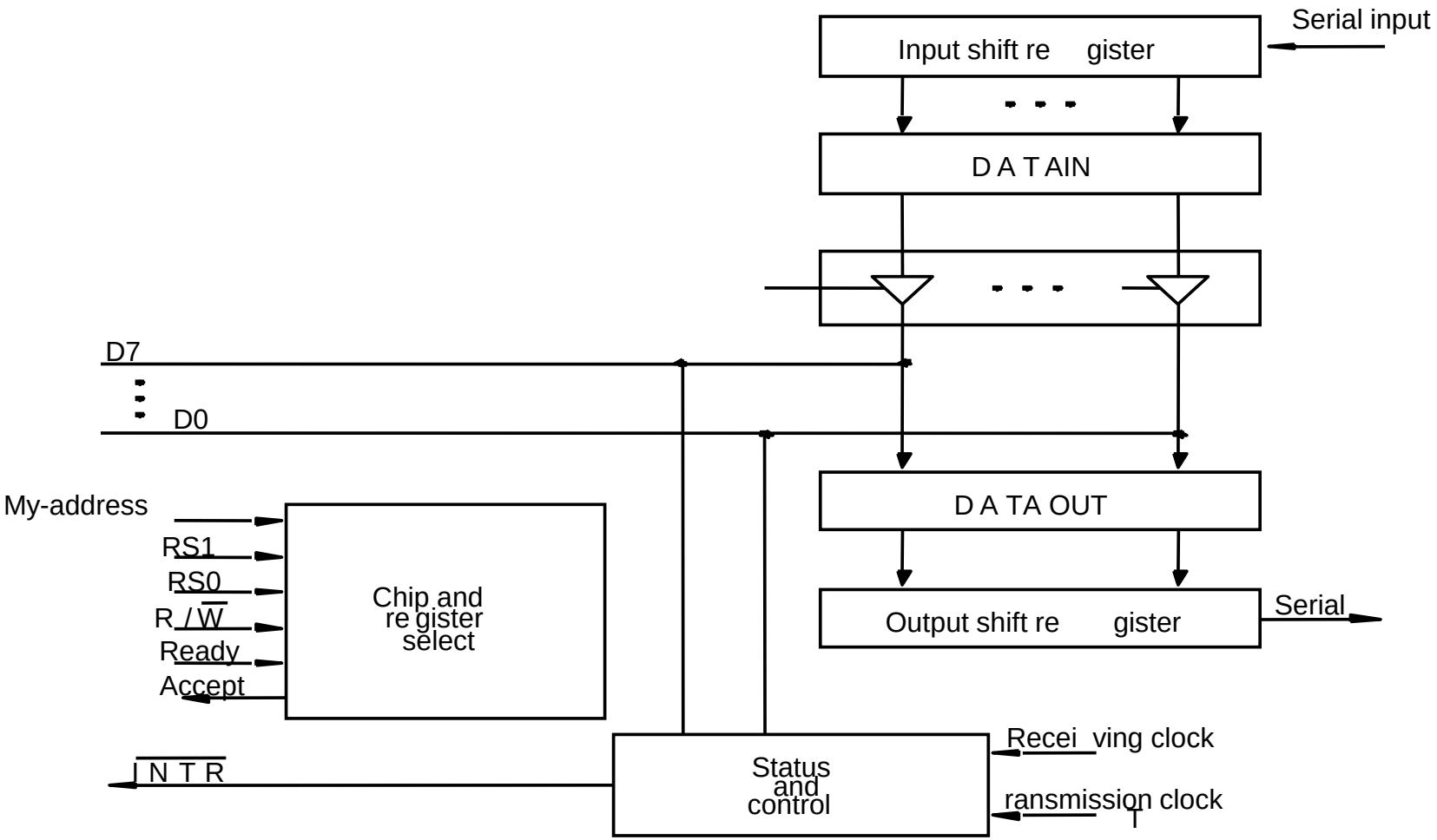
Recall the Timing Protocol for Output operation



Serial port

- Serial port is used to connect the processor to I/O devices that require transmission of data one bit at a time.
- Serial port communicates in a bit-serial fashion on the device side and bit parallel fashion on the bus side.
  - Transformation between the parallel and serial formats is achieved with shift registers that have parallel access capability.
  - Capable of longer distance communication than parallel transmission.

A Serial Interface



- Input shift register accepts input one bit at a time from the I/O device.
- Once all the 8 bits are received, the contents of the input shift register are loaded in parallel into DATAIN register.
- Output data in the DATAOUT register are loaded into the output shift register.
- Bits are shifted out of the output shift register and sent out to the I/O device one bit at a time.
- As soon as data from the input shift reg.are loaded into DATAIN, it can start accepting another 8 bits of data.
- Input shift register and DATAIN registers are both used at input so that the input shift register can start receiving an other set of 8 bits from the input device after loading the contents to DATAIN, before the processor reads the contents of DATAIN. This is called as double-buffering.

Serial port (contd..)

Serial interfaces require fewer wires, and hence serial transmission is convenient for connecting devices that are physically distant from the computer.

Speed of transmission of the data over a serial interface is known as the "bit rate".

- Bit rate depends on the nature of the devices connected.

In order to accommodate devices with a range of speeds, a serial interface must be able to use a range of clock speeds.

Several standard serial interfaces have been developed:

- Universal Asynchronous Receiver Transmitter (UART) for low-speed serial devices.
- RS-232-C for connection to communication links.

Standard I/O Interfaces

Standard I/O interfaces

I/O device is connected to a computer using an interface circuit.

Do we have to design a different interface for every combination of an I/O device and a computer?

A practical approach is to develop standard interfaces and protocols.

A personal computer has:

- A motherboard which houses the processor chip, main memory and some I/O interfaces.
- A few connectors into which additional interfaces can be plugged.

Processor bus is defined by the signals on the processor chip.

- Devices which require high-speed connection to the processor are connected directly to this bus.

Standard I/O interfaces (contd..)

Because of electrical reasons only a few devices can be connected directly to the processor bus.

Motherboard usually provides another bus that can support more devices.

- Processor bus and the other bus (called as expansion bus) are interconnected by a circuit called "bridge".
- Devices connected to the expansion bus experience a small delay in data transfers.

Design of a processor bus is closely tied to the architecture of the processor.

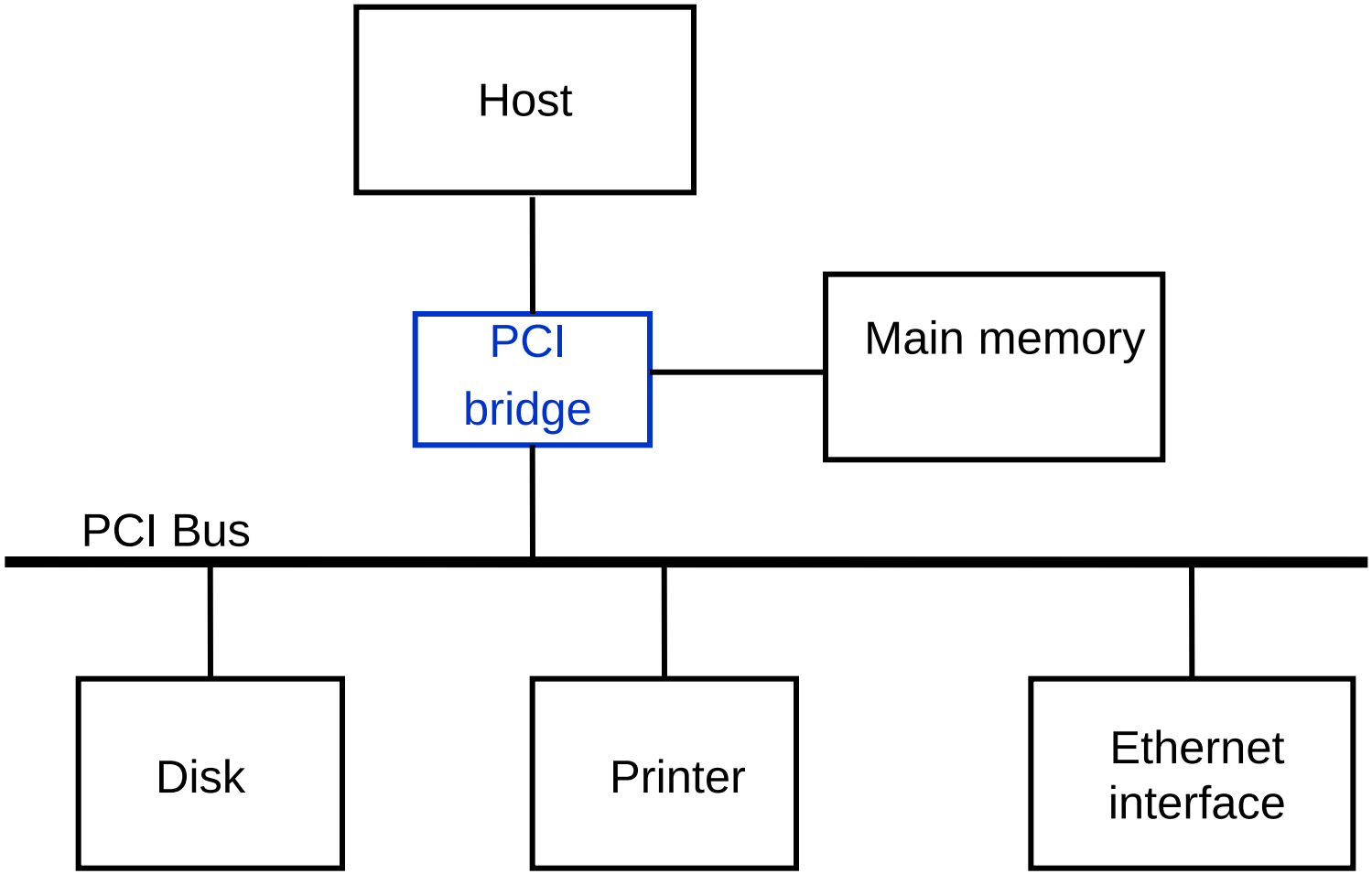
- No uniform standard can be defined.

Expansion bus however can have uniform standard defined.

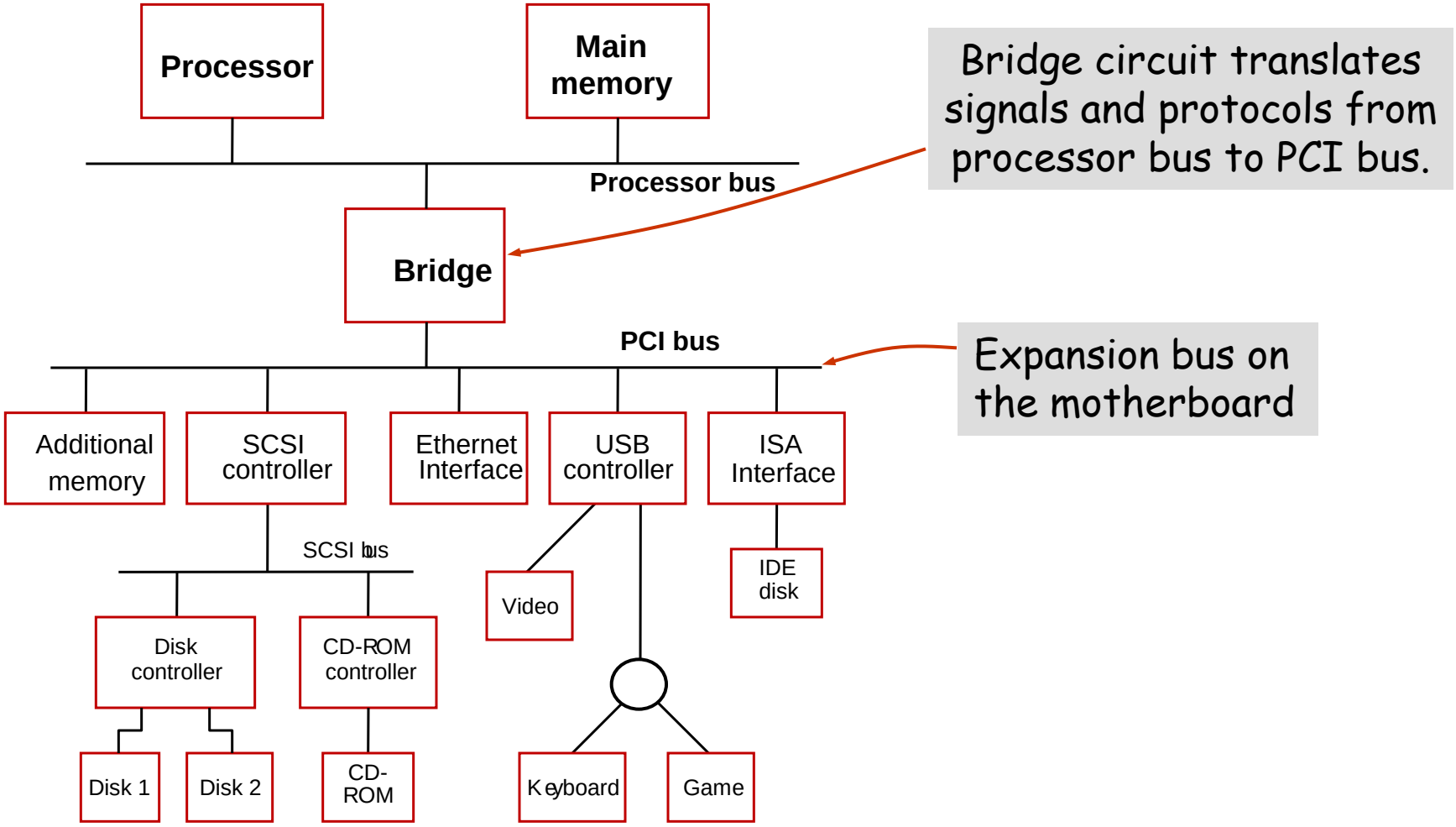
Standard I/O interfaces (contd..)

- A number of standards have been developed for the expansion bus.
  - Some have evolved by default.
  - For example, IBM's Industry Standard Architecture.
- Three widely used bus standards:
  - PCI (Peripheral Component Interconnect)
  - SCSI (Small Computer System Interface)
  - USB (Universal Serial Bus)

Peripheral Component Interconnect Bus



Computer system using different interface Standards



PCI

PCI

Main goals:

- To Improve data-transfers to/from peripheral devices
- To Eliminate (or reduce) platform dependencies
- To Simplify adding/removing peripheral devices
- To Lower total consumption of electrical power

PCI Bus

Peripheral Component Interconnect

Introduced in 1992

Low-cost bus

Processor independent

Plug-and-play capability

Most memory transfers involve a burst of data rather than just one word. The PCI is designed to support this mode of operation.

The bus supports three independent address spaces: memory, I/O, and configuration.

The master maintains the address information on the bus until data transfer is completed. But, the address is needed only long enough for the slave to be selected. Thus, the address is needed on the bus for one clock cycle only, freeing the address lines to be used for sending data in subsequent clock cycles. The result is a significant cost reduction.

A master is called an initiator in PCI terminology. The addressed device that responds to read and write commands is called a target.

Data transfer signals on the PCI bus.

Name	Function
CLK	A 33-MHz or 66-MHz clock.
FRAME#	Sent by the initiator to indicate the duration of a transaction.
AD	32 address/data lines, which may be optionally increased to 64.
C/BE#	4 command/byte-enable lines (8 for a 64-bit bus).
IRD Y#, TRD Y#	Initiator-ready and Target-ready signals.
DEVSEL#	A response from the device indicating that it has recognized its address and is ready for a data transfer transaction.
IDSEL#	Initialization Device Select.

Read operation on the PCI bus

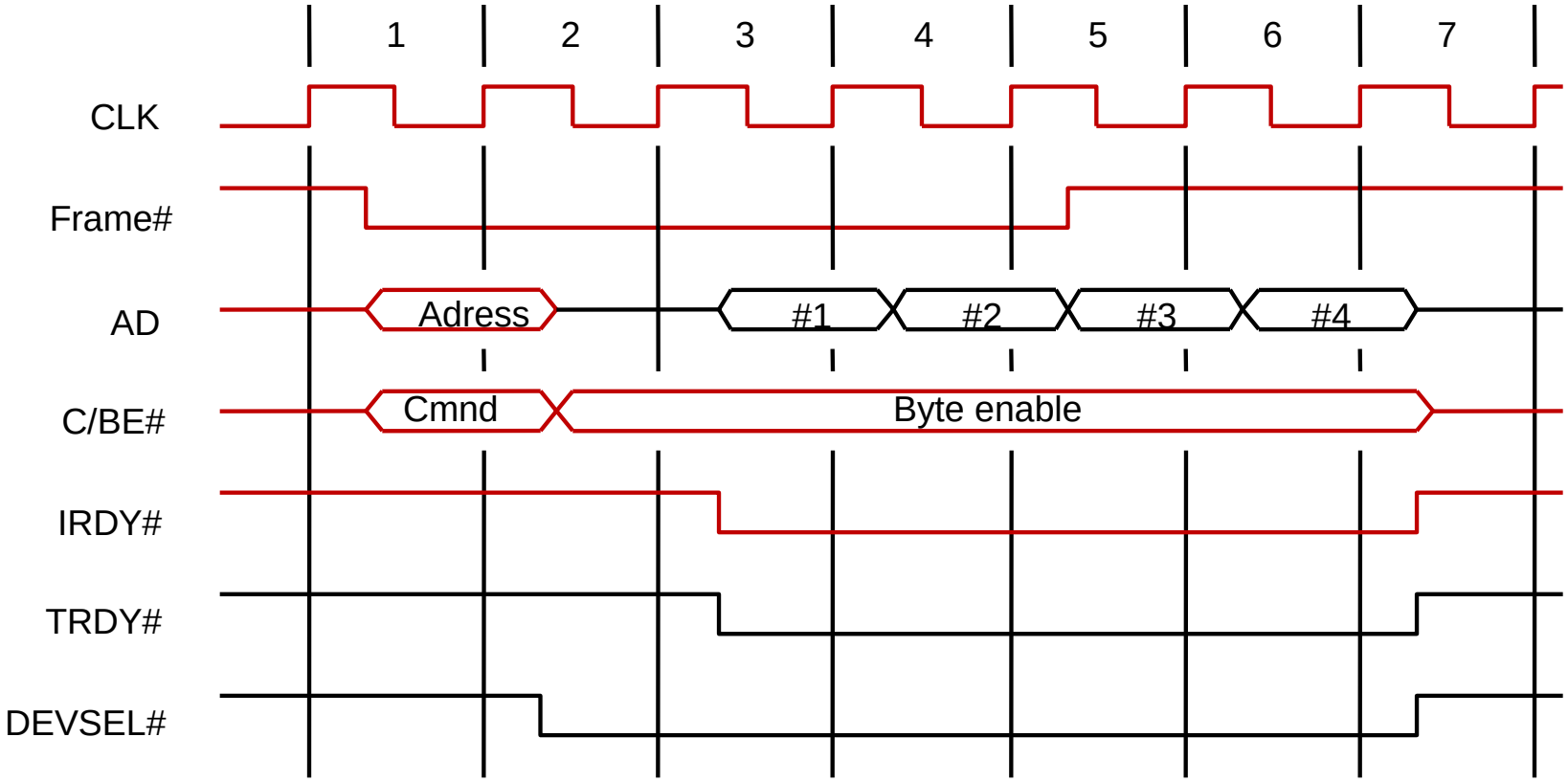
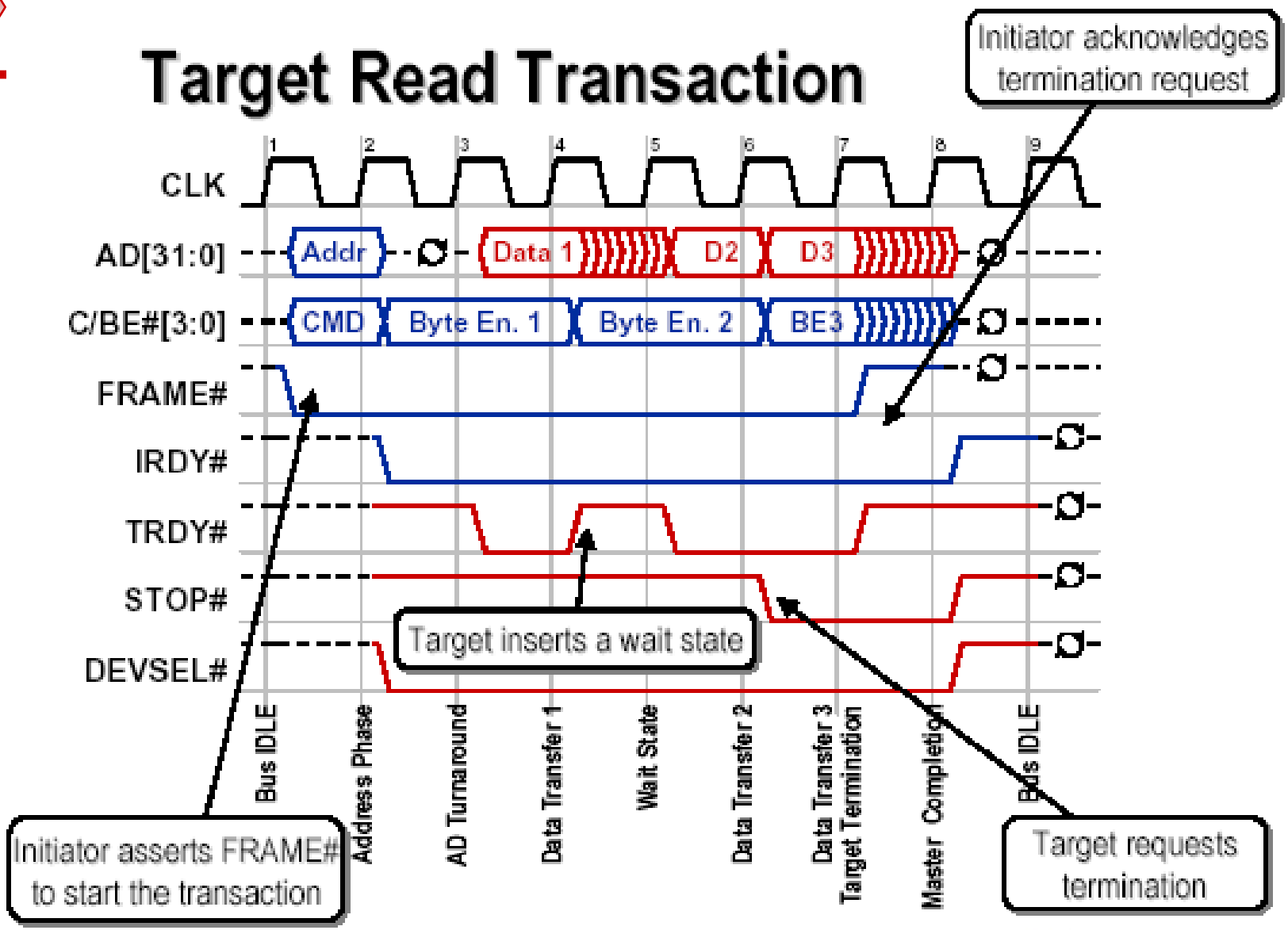
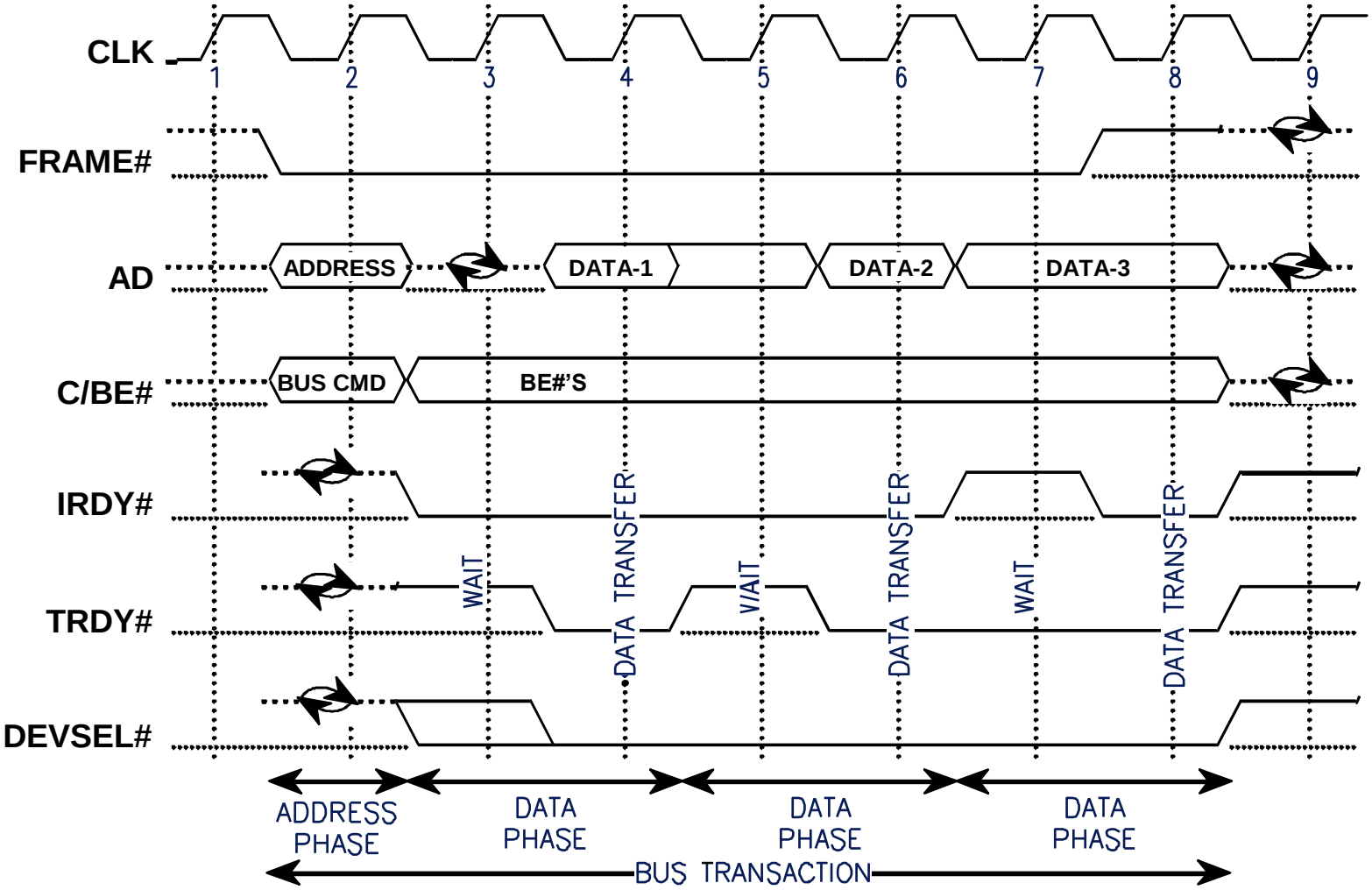


Fig. A read operation on the PCI bus

Target Read Transaction

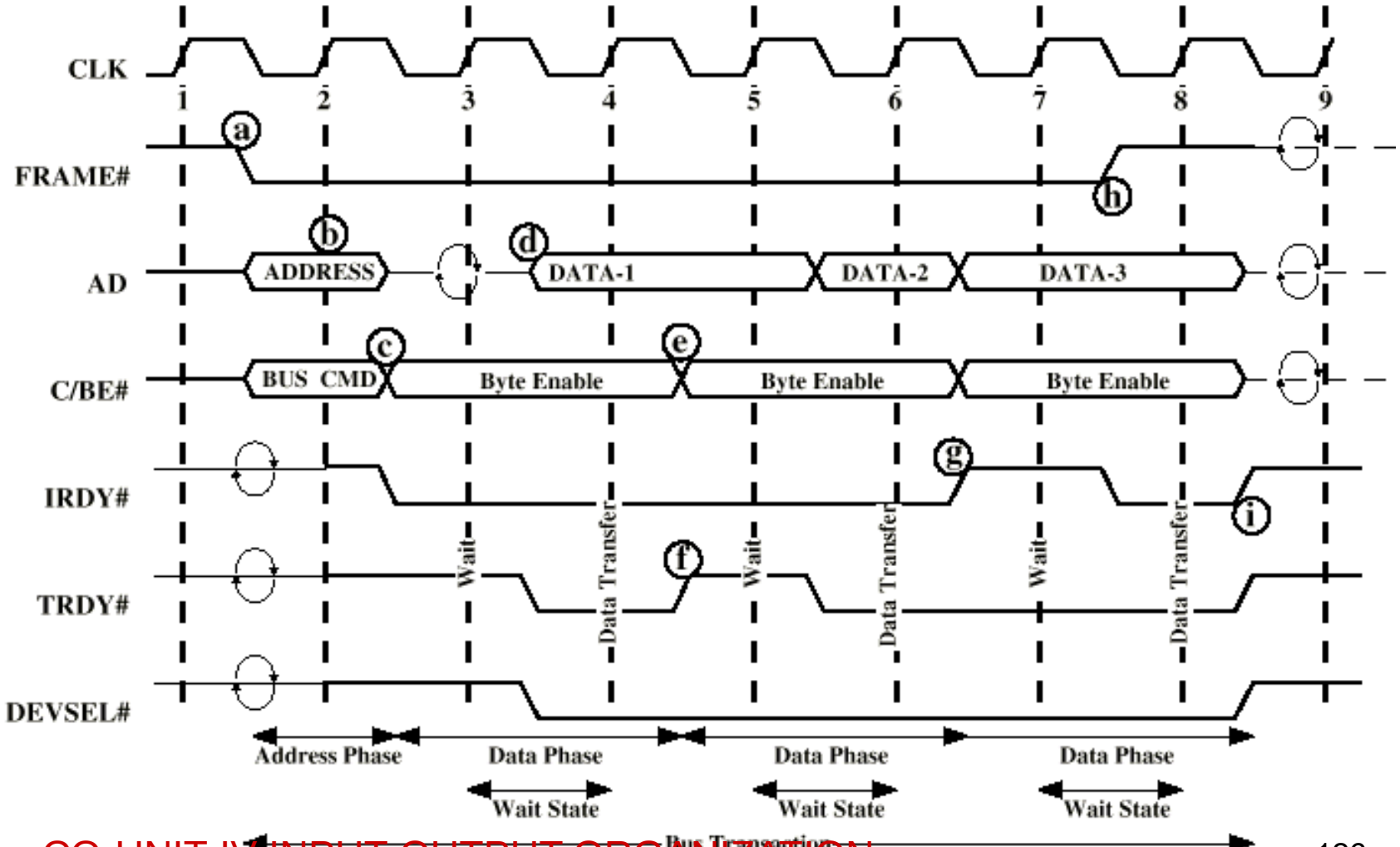


PCI Bus Read

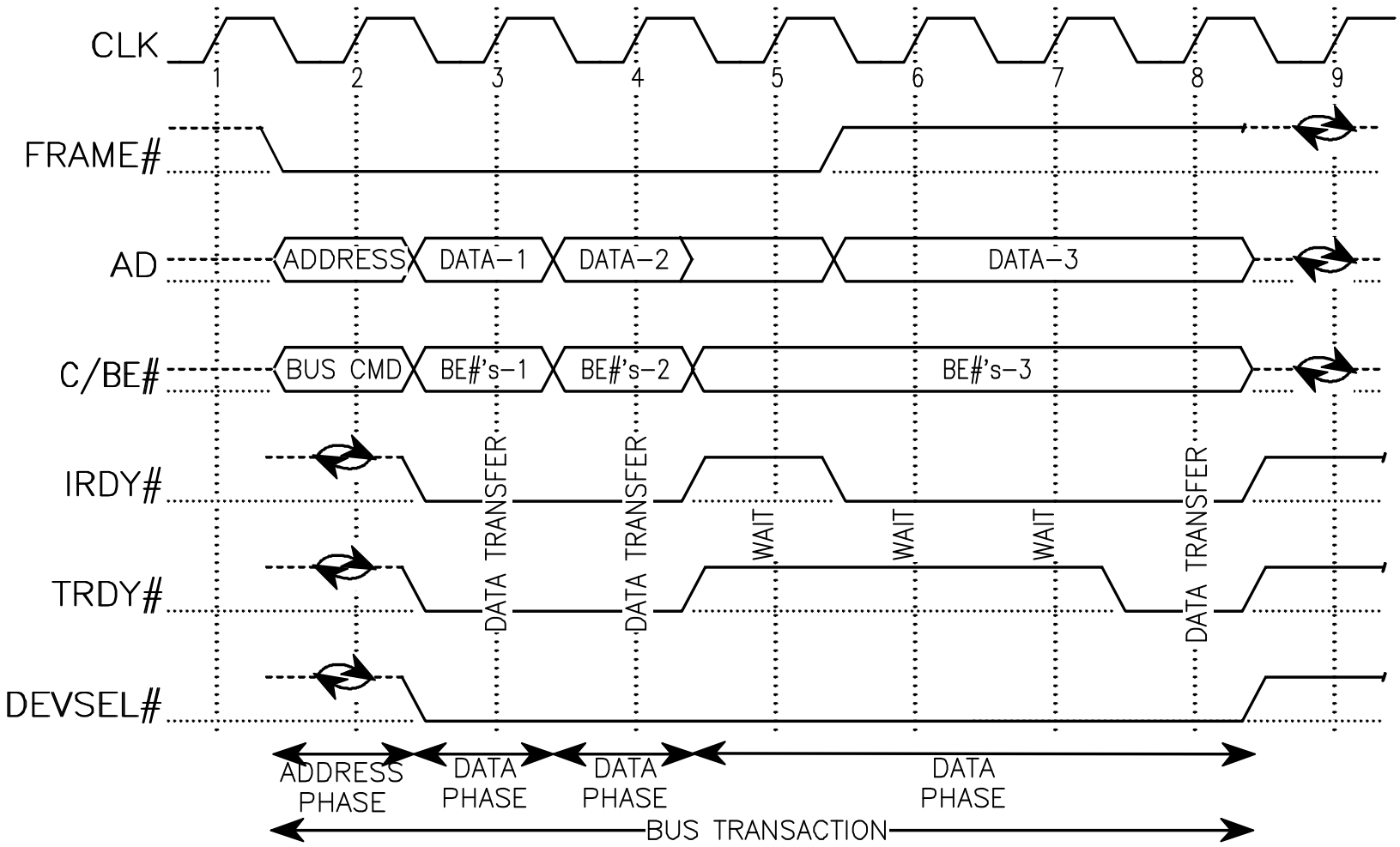

CLK
1
2
3
4
5
6
7
8
9
FRAME#
AD
ADDRESS
DATA-1
DATA-2
DATA-3
C/BE#
BUS CMD
BE#'S
IRDY#
TRDY#
DEVSEL#
WAIT
DATA TRANSFER
WAIT
DATA TRANSFER
WAIT
DATA TRANSFER
ADDRESS PHASE
DATA PHASE
DATA PHASE
DATA PHASE
BUS TRANSACTION

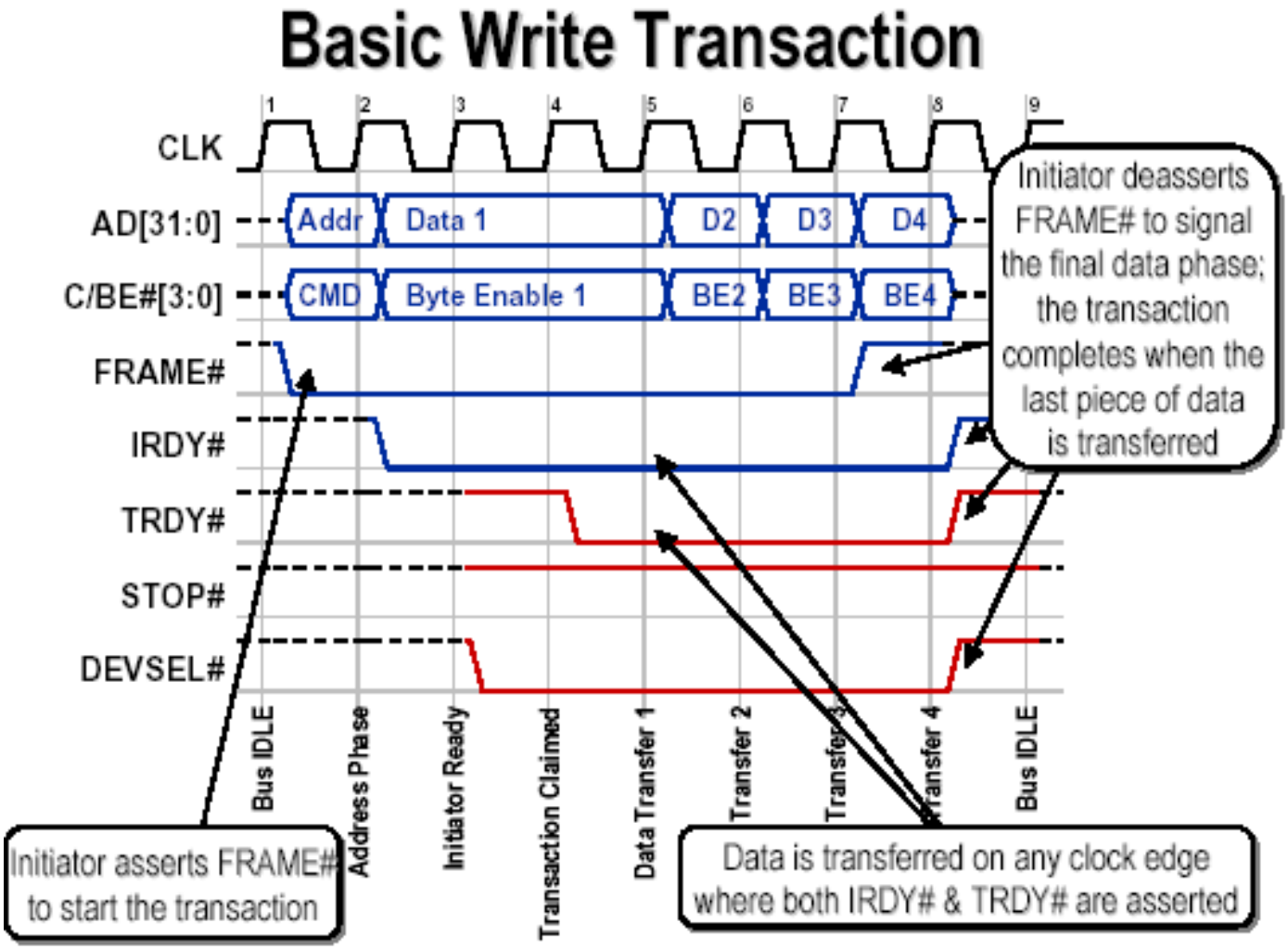
PCI Read Timing Diagram for a block of size 3

1st–occurs ASAP, 2nd – target not ready for 1cycle, 3rd – initiator not ready for 1 cycle



PCI Bus Write




Basic Write Transaction
1
2
3
4
5
6
7
8
9
CLK
AD[31:0]
Addr
Data 1
D2
D3
D4
C/BE#[3:0]
CMD
Byte Enable 1
BE2
BE3
BE4
FRAME#
IRDY#
TRDY#
STOP#
DEVSEL#
Initiator deasserts FRAME# to signal the final data phase; the transaction completes when the last piece of data is transferred
Bus IDLE
Address Phase
Initiator Ready
Transaction Claimed
Data Transfer 1
Transfer 2
Transfer 3
Transfer 4
Bus IDLE
Initiator asserts FRAME# to start the transaction
Data is transferred on any clock edge where both IRDY# & TRDY# are asserted

Device Configuration

When an I/O device is connected to a computer, several actions are needed to configure both the device and the software that communicates with it.

PCI incorporates in each I/O device interface a small configuration ROM memory that stores information about that device.

The configuration ROMs of all devices are accessible in the configuration address space. The PCI initialization software reads these ROMs and determines whether the device is a printer, a keyboard, an Ethernet interface, or a disk controller. It can further learn bout various device options and characteristics.

Devices are assigned addresses during the initialization process.

This means that during the bus configuration operation, devices cannot be accessed based on their address, as they have not yet been assigned one.

Hence, the configuration address space uses a different mechanism. Each device has an input signal called Initialization Device Select, IDSEL#

Electrical characteristics: PCI bus has been defined for operation with either a 5 or 3.3 V power supply

SCSI Bus

SCSI Bus

The acronym SCSI stands for Small Computer System Interface.

It refers to a standard bus defined by the American National Standards Institute (ANSI) under the designation X3.131 .

In the original specifications of the standard, devices such as disks are connected to a computer via a 50-wire cable, which can be up to 25 meters in length and can transfer data at rates up to 5 megabytes/s.

The SCSI bus standard has undergone many revisions, and its data transfer capability has increased very rapidly, almost doubling every two years.

SCSI-2 and SCSI-3 have been defined, and each has several options.

Because of various options SCSI connector may have 50, 68 or 80 pins.

SCSI Bus (Contd.,)

Devices connected to the SCSI bus are not part of the address space of the processor

The SCSI bus is connected to the processor bus through a SCSI controller. This controller uses DMA to transfer data packets from the main memory to the device, or vice versa.

A packet may contain a block of data, commands from the processor to the device, or status information about the device.

A controller connected to a SCSI bus is one of two types – an initiator or a target.

An initiator has the ability to select a particular target and to send commands specifying the operations to be performed. The disk controller operates as a target. It carries out the commands it receives from the initiator.

The initiator establishes a logical connection with the intended target.

Once this connection has been established, it can be suspended and restored as needed to transfer commands and bursts of data.

While a particular connection is suspended, other device can use the bus to transfer information.

This ability to overlap data transfer requests is one of the key features of the SCSI bus that leads to its high performance.

SCSI Bus (Contd.,)

- Data transfers on the SCSI bus are always controlled by the target controller.
- To send a command to a target, an initiator requests control of the bus and, after winning arbitration, selects the controller it wants to communicate with and hands control of the bus over to it.
- Then the controller starts a data transfer operation to receive a command from the initiator.

SCSI Bus (Contd.,)

Assume :

The processor needs to read block of data from a disk drive and that data are stored in disk sectors that are not contiguous.

The processor sends a command to the SCSI controller, which causes the following sequence of events to take place:

- The SCSI controller, acting as an initiator, contends for control of the bus.
- When the initiator wins the arbitration process, it selects the target controller and hands over control of the bus to it.
- The target starts an output operation (from initiator to target); in response to this, the initiator sends a command specifying the required read operation.
- The target, realizing that it first needs to perform a disk seek operation, sends a message to the initiator indicating that it will temporarily suspend the connection between them. Then it releases the bus.
- The target controller sends a command to the disk drive to move the read head to the first sector involved in the requested read operation. Then, it reads the data stored in that sector and stores them in a data buffer. When it is ready to begin transferring data to the initiator, the target requests control of the bus. After it wins arbitration, it reselects the initiator controller, thus restoring the suspended connection.

SCSI Bus (Contd.,)

- The target transfers the contents of the data buffer to the initiator and then suspends the connection again
- The target controller sends a command to the disk drive to perform another seek operation. Then, it transfers the contents of the second disk sector to the initiator as before. At the end of this transfers, the logical connection between the two controllers is terminated.
- As the initiator controller receives the data, it stores them into the main memory using the DMA approach.
- The SCSI controller sends as interrupt to the processor to inform it that the requested operation has been complete

Operation of SCSI bus from H/W point of view

Category	Name	Function
Data	– DB(0) to – DB(7)	Data lines: Carry one byte of information during the information transfer phase and identify device during arbitration, selection and reselection phases
	– DB(P)	Parity bit for the data bus
Phase	– BSY	Busy: Asserted when the bus is not free
	– SEL	Selection: Asserted during selection and reselection
Information type	– C/D	Control/Data: Asserted during transfer of control information (command, status or message)
	– MSG	Message: indicates that the information being transferred is a message

Table 4. The SCSI bus signals.

Table 4. The SCSI bus signals.(cont.)

Category	Name	Function
Handshake	– REQ	Request: Asserted by a target to request a data transfer cycle
	– ACK	Acknowledge: Asserted by the initiator when it has completed a data transfer operation
Direction of transfer	– I/O	Input/Output: Asserted to indicate an input operation (relative to the initiator)
Other	– ATN	Attention: Asserted by an initiator when it wishes to send a message to a target
	– RST	Reset: Causes all device controls to disconnect from the bus and assume their start-up state

Main Phases involved

Arbitration

- A controller requests the bus by asserting BSY and by asserting it's associated data line
- When BSY becomes active, all controllers that are requesting bus examine data lines

Selection

- Controller that won arbitration selects target by asserting SEL and data line of target. After that initiator releases BSY line.
- Target responds by asserting BSY line
- Target controller will have control on the bus from then

Information Transfer

- Handshaking signals are used between initiator and target
- At the end target releases BSY line Reselection

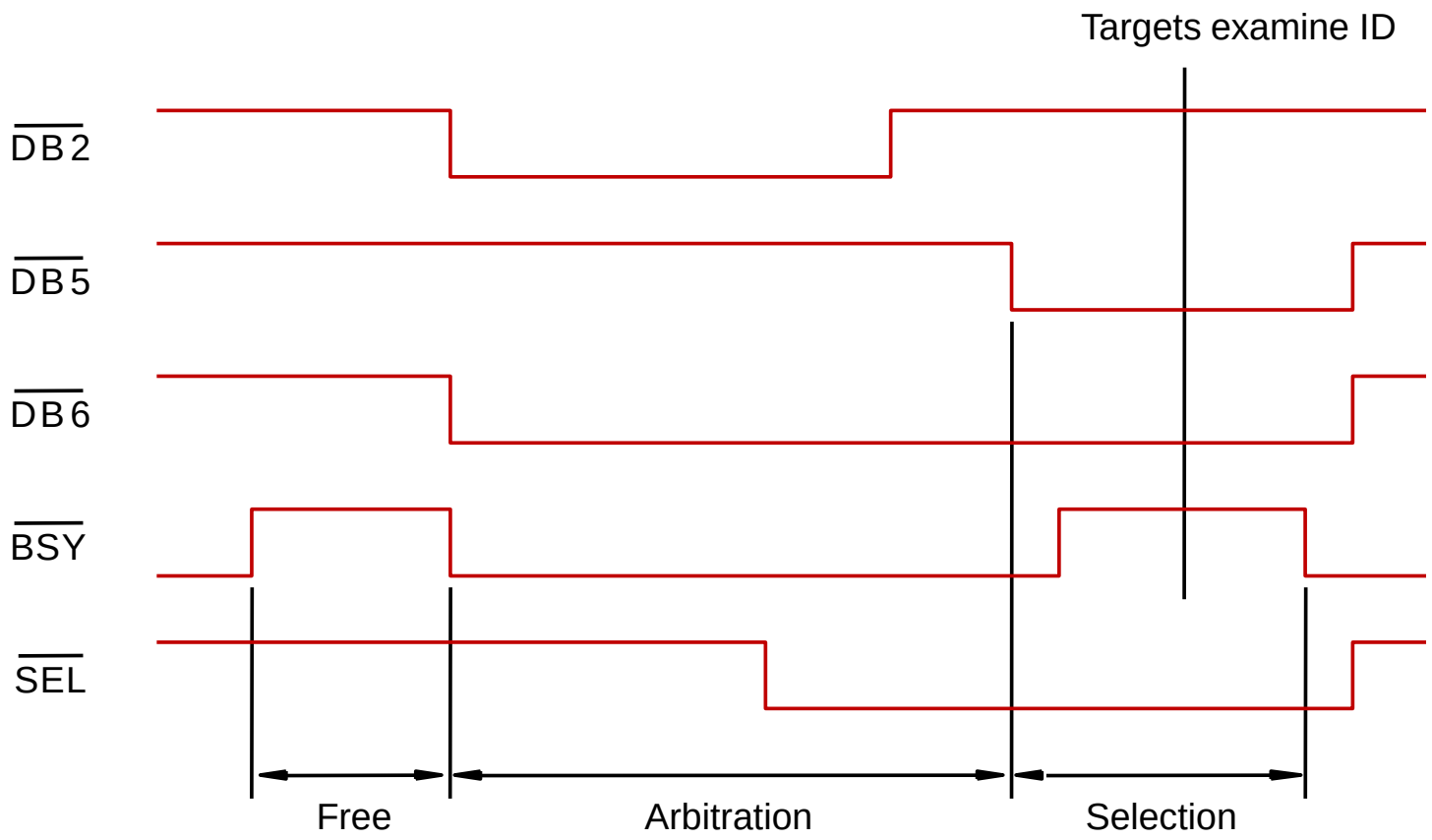


Fig. Arbitration and selection on the SCSI bus.
Device 6 wins arbitration and selects device 2.

Universal Serial Bus (USB)

USB

Universal Serial Bus (USB) is an industry standard developed through a collaborative effort of several computer and communication companies, including Compaq, Hewlett-Packard, Intel, Lucent, Microsoft, Nortel Networks, and Philips.

Speed : The US supports three speeds of operation

- Low-speed (USB 1.0) rate produces upto 1.5 Mb/s
- Full-speed (USB 1.1 rate produces upto - 12 Mb/s
- High-speed (USB 2.0) rate produces upto 480 Mb/s
- SuperSpeed (USB 3.0) rate produces upto 4800 Mbit/s

The USB has been designed to meet several key objectives:

1)Provide a simple, low-cost, and easy to use interconnection system that overcomes the difficulties due to the limited number of I/O ports available on a computer.

2)Accommodate a wide range of data transfer characteristics for I/O devices, including telephone and Internet connections.

3)Enhance user convenience through a ''plug-and-play'' mode of operation

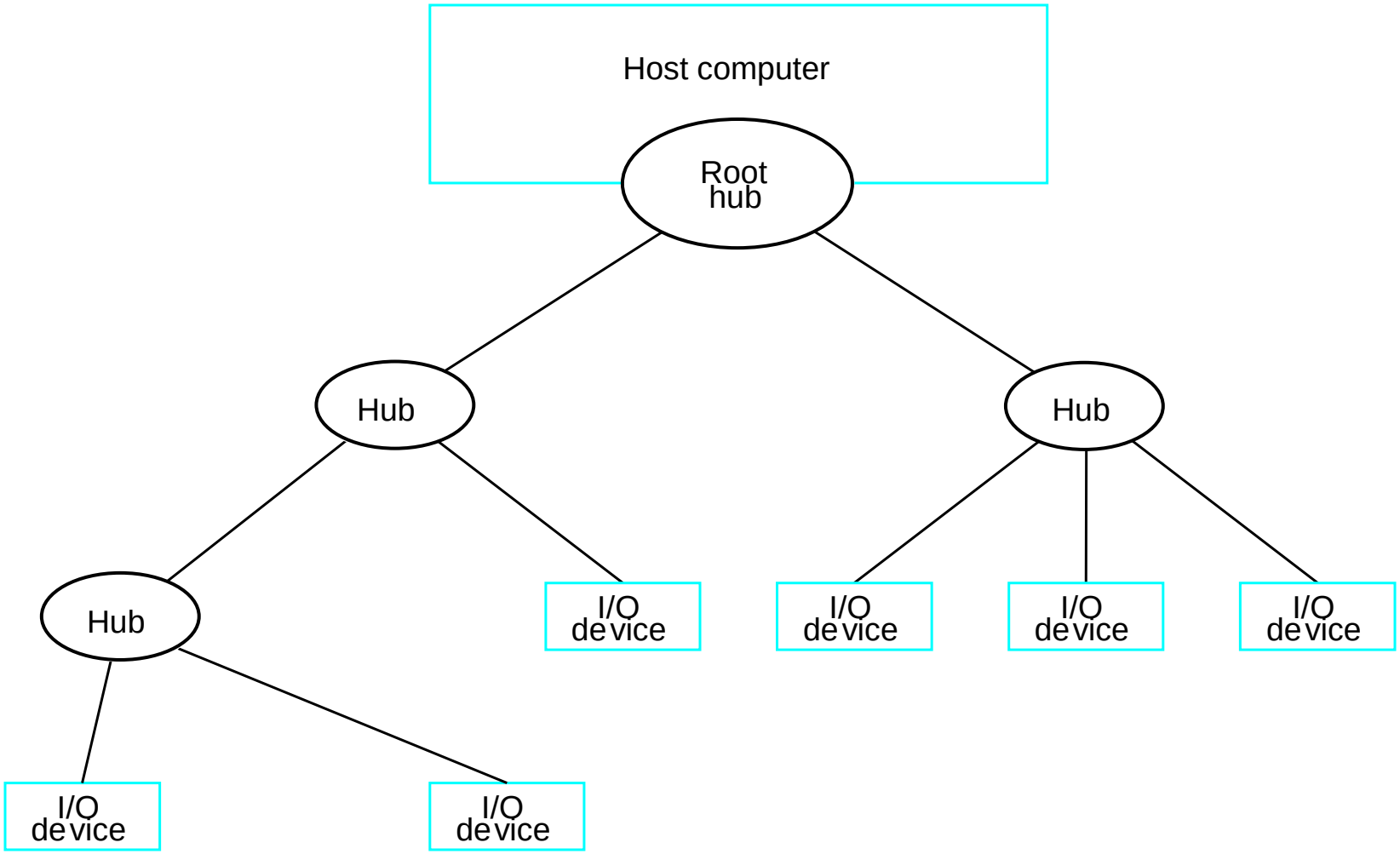
USB (cont'd)

- Motivation for USB
  - Avoid device-specific interfaces
    - Eliminates multitude of interfaces
      - PS/2, serial, parallel, monitor, microphone, keyboard,…
  - Avoid non-shareable interfaces
    - Standard interfaces support only one device
  - Avoid I/O address space and IRQ problems
    - USB does not require memory or address space
  - Avoid installation and configuration problems
    - Don't have to open the box to install and configure jumpers
  - Allow hot attachment of devices

USB (cont'd)

- Additional advantages of USB
  - Power distribution
    - Simple devices can be bus-powered
      - Examples: mouse, keyboards, floppy disk drives, wireless LANs, coffee warmers, reading lights, MP3 players, …
  - Control peripherals
    - Possible because USB allows data to flow in both directions
  - Expandable through hubs
  - Power conservation
    - Enters suspend state if there is no activity for 3 ms
  - Error detection and recovery
    - Uses CRC

Universal Serial Bus tree structure



Universal Serial Bus tree structure

USB has the tree structure

It accommodate a large number of devices .

Easy either to add or remove any device at any time.

Each node of the tree has a device called a hub, which acts as an intermediate control point between the host and the I/O devices. At the root of the tree, a root hub connects the entire tree to the host computer.

The leaves of the tree are the I/O devices being served (for example, keyboard, Internet connection, speaker, or digital TV)

In normal operation, a hub copies a message that it receives from its upstream connection to all its downstream ports. As a result, a message sent by the host computer is broadcast to all I/O devices, but only the addressed device will respond to that message.

However, a message from an I/O device is sent only upstream towards the root of the tree and is not seen by other devices. Hence, the USB enables the host to communicate with the I/O devices, but it does not enable these devices to communicate with each other.

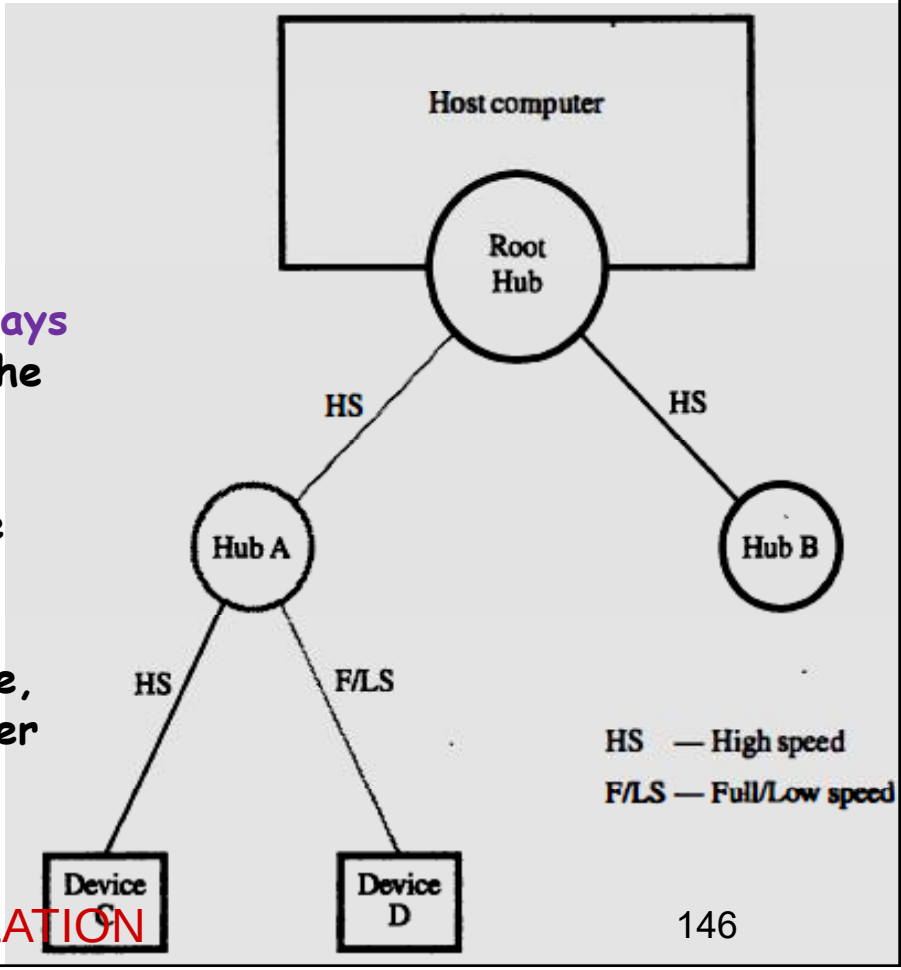
The USB operates strictly on the basis of polling. A device may send a message only in response to a poll message from the host.

Hence, upstream messages do not encounter conflicts or interfere with each other, as no two devices can send messages at the same time. This restriction allows hubs to be simple, low-cost devices.

When a USB is connected to a host computer,its root hub is attached to the processor bus, where it appears as a single device.

The host software communicates with individual devices attached to the USB by sending packets of information, which the root hub forwards to the appropriate device in the USB tree.

USB protocol requires that a message transmitted on a high- speed link is always transmitted at high speed, even when the ultimate receiver is a low-speed device. Hence, a message intended for device D is sent at high speed from the root hub to hub A, then forwarded at low speed to device D. The latter transfer will take a long time, during which high- speed traffic to other nodes is allowed to continue.



Addressing

Each device on the USB, whether it is a hub or an I/O device, is assigned a 7-bit address. This address is local to the USB tree and is not related in any way to the addresses used on the processor bus.

A hub may have any number of devices or other hubs connected to it, and addresses are assigned arbitrarily.

When a device is first connected to a hub, or when it is powered on, it has the address 0. The hardware of the hub to which this device is connected is capable of detecting that the device has been connected, and it records this fact as part of its own status information. Periodically, the host polls each hub to collect status information and learn about new devices that may have been added or disconnected.

When the host is informed that a new device has been connected, it uses a sequence of commands to send a reset signal on the corresponding hub port, read information from the device about its capabilities, send configuration information to the device, and assign the device a unique USB address. Once this sequence is completed the device begins normal operation and responds only to the new address.

USB Protocols

All information transferred over the USB is organized in packets, where a packet consists of one or more bytes of information. There are many types of packets that perform a variety of control functions.

The information transferred on the USB can be divided into two broad categories: control and data.

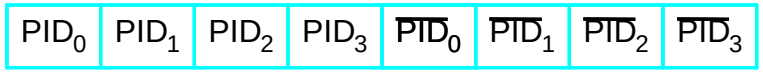
- Control packets perform such tasks as addressing a device to initiate data transfer, acknowledging that data have been received correctly, or indicating an error.
- Data packets carry information that is delivered to a device.

A packet consists of one or more fields containing different kinds of information.

The first field of any packet is called the packet identifier, PID, which identifies the type of that packet.

They are transmitted twice. The first time they are sent with their true values, and the second time with each bit complemented.

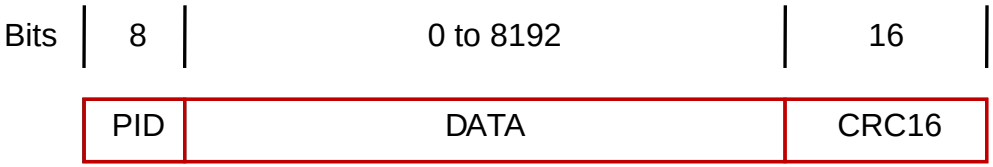
The four PID bits identify one of 16 different packet types. Some control packets, such as ACK (Acknowledge), consist only of the PID byte.



(a) Packet identifier field

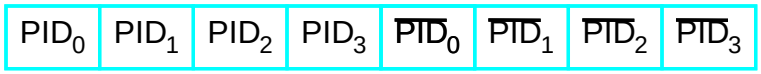
(b) Token packet, IN or OUT

Control packets controls data transfer. Operations are called token packets.



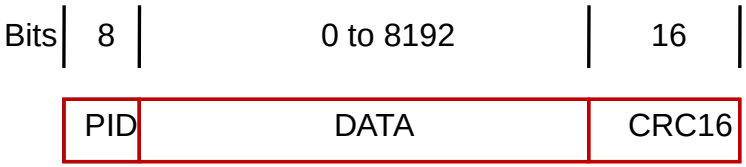
(c)Data packet

Fig.USB packet format



(a) Packet identifier field

(b) Token packet, IN or OUT



(c) Data packet

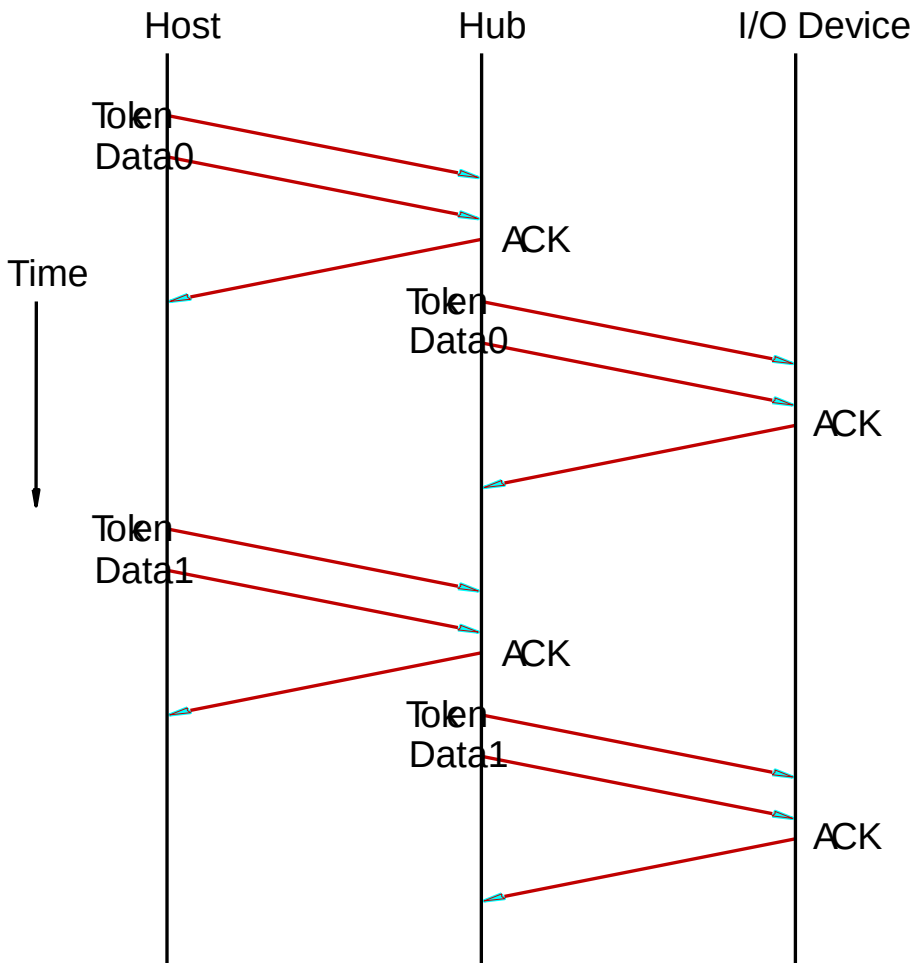


Fig. An output transfer

Isochronous Traffic on USB

One of the key objectives of the USB is to support the transfer of isochronous data.

Devices that generates or receives isochronous data require a time reference to control the sampling process.

To provide this reference. Transmission over the USB is divided into frames of equal length.

A frame is 1ms long for low-and full-speed data.

The root hub generates a Start of Frame control packet (SOF) precisely once every 1 ms to mark the beginning of a new frame.

The arrival of an SOF packet at any device constitutes a regular clock signal that the device can use for its own purposes.

To assist devices that may need longer periods of time, the SOF packet carries an 11-bit frame number.

Following each SOF packet, the host carries out input and output transfers for isochronous devices.

This means that each device will have an opportunity for an input or output transfer once every 1 ms.

Electrical Characteristics

The cables used for USB connections consist of four wires.

Two are used to carry power, +5V and Ground.

- Thus, a hub or an I/O device may be powered directly from the bus, or it may have its own external power connection.

The other two wires are used to carry data.

Different signaling schemes are used for different speeds of transmission.

- At low speed, 1s and 0s are transmitted by sending a high voltage state (5V) on one or the other o the two signal wires. For high-speed links, differential transmission is used.