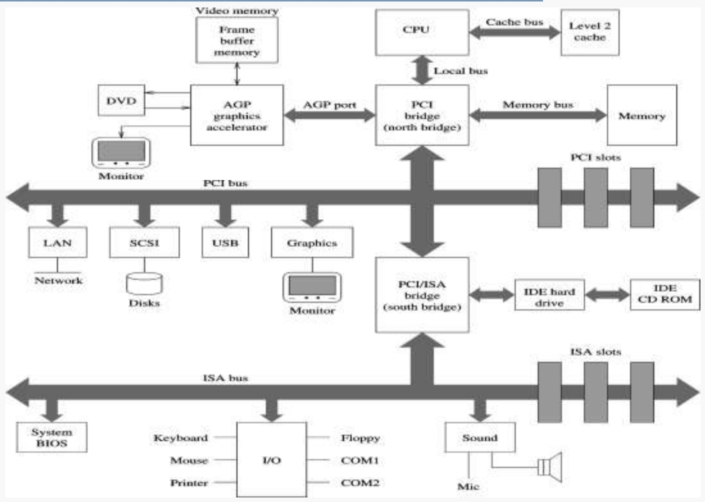
Interfacing

D.N.Rakhmatov

Adopted (with modifications) from:
R. Katz, UC-Berkeley
F. Vahid, UC-Riverside
D. Givone, SUNY-Buffalo
J. Armstrong, Virginia Tech
C. Hamacher et al, *Computer Organization*, 6/E, © 2011 McGraw-Hill
S. Dandamudi, *Fundamentals of Computer Organization and Design*, © 2002 Springer

System Integration



System Bus I

- Bus = address + data + control (command/status)
- Some typical control bus signals:
  - Memory read/write
    - This includes **memory-mapped I/O** read and write
    - For **isolated I/O**, need an extra signal indicating address type
  - Interrupt-Request, Interrupt-Acknowledge
  - Address-Valid, Data-Valid, Master-Ready, Slave-Ready
  - Bus-Request, Bus-Grant, Bus-Busy
  - Clock, Reset, Enable, Wait
- Bus master – the device that is allowed to initiate bus transactions at any given time
  - Only **one** master at a time, everyone else is a slave
  - Bus mastership can be changed through bus arbitration

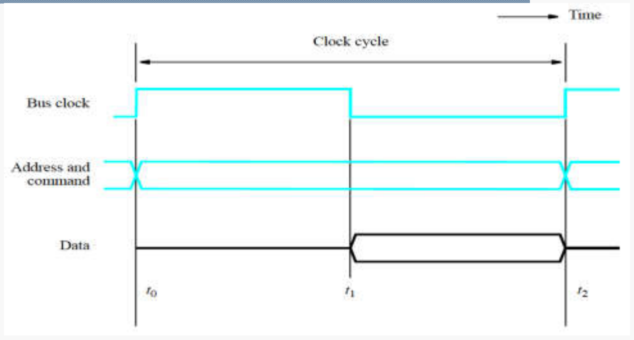
System Bus II

- General bus operation:
  - Master places desired slave's address onto address bus
    - All slaves must examine the address and decide if they are being referenced (address decoding)
  - Master (or selected slave) places data onto data bus for a write (read) operation
    - Selected slave (or master) gates the data into its internal registers to complete the operation
  - Operation is directed by the control bus signals
- Bus is NOT a static connection mechanism
  - Devices must be enabled (given access to write to the bus) only when they are participating in a data transfer
    - Only **one** device may drive the bus at any given time
    - Other devices should be in high-impedance state

Synchronous Bus

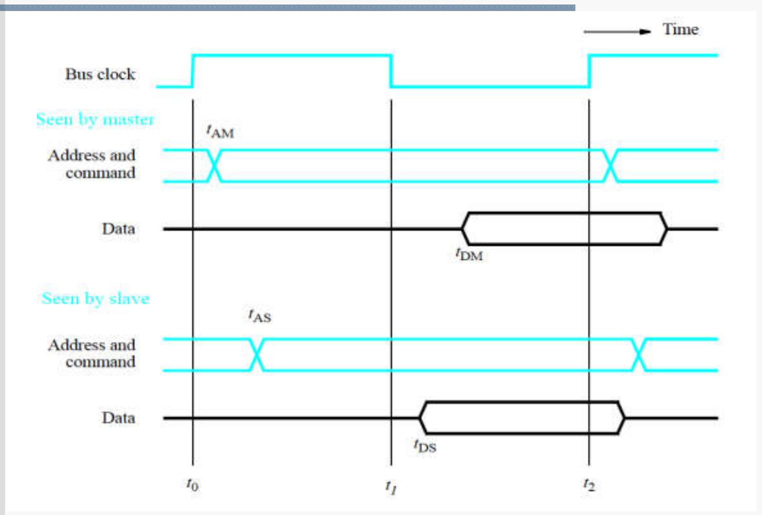
- In a synchronous bus, the timing of all devices is synchronized to the bus clock
  - Delay issues:
    - Certain delays are introduced by the bus drivers that place new information on the address or data lines
    - An additional delay is encountered as information propagates along the bus wires
    - The receiving end requires a certain (minimum) amount of setup time to latch the incoming data correctly
- Example: **Read** operation
  - At the rising clock edge (**t0**) – the master places the device's address on the bus address lines
  - At the falling clock edge (**t1**) – the slave responds by placing its data on the bus data lines
  - At the next rising clock edge (**t2**) – the master latches the data on the bus data lines into its input buffer

Synchronous Read Example



1. **(t1 – t0)** must be longer than the maximum propagation delay between the master and the slave, plus the slave's delay of decoding the address and control signals
2. **(t2 – t1)** must be longer than the maximum propagation delay between the master and the slave, plus the master's setup time

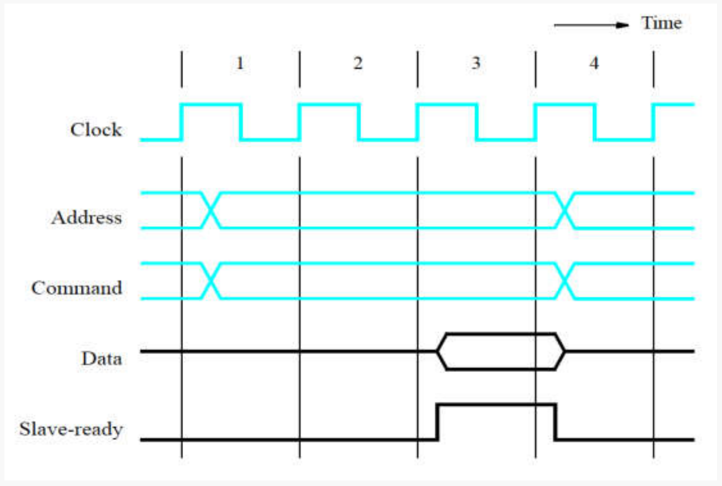
More Detailed Picture (Read)



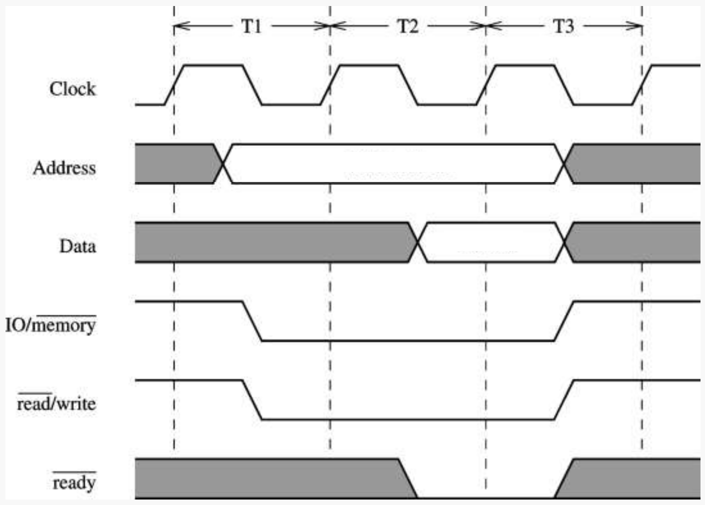
Multi-Cycle Bus Transfers

- **Single-cycle** bus transfer mechanism forces all devices to operate at the speed of the slowest one
  - Also, the master cannot determine if the selected slave has responded correctly
- **Multi-cycle** bus transfers increase bus utilization and transfer reliability
  - Example: multi-cycle read operation
    - ↑ **Cycle 1** – master puts the slave's address and command on the bus, along with the (OPTIONAL) master-ready signal
    - ↑ **Cycle 2** – slave decodes this information and accesses the requested data
      - ✓ This step may take more than one clock cycle, delaying subsequent steps
    - ↑ **Cycle 3** – slave puts the ready data on the bus and asserts the (REQUIRED) slave-ready signal
    - ↑ **Cycle 4** – the master receives this information and latches the data into its input buffer

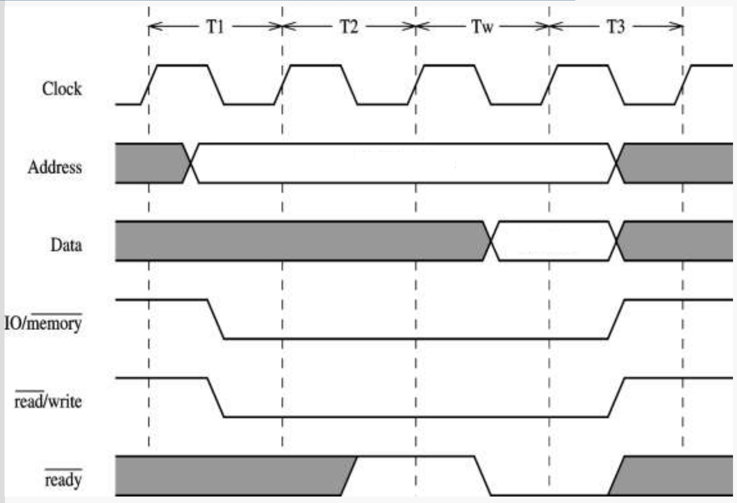
Multi-Cycle Read Example



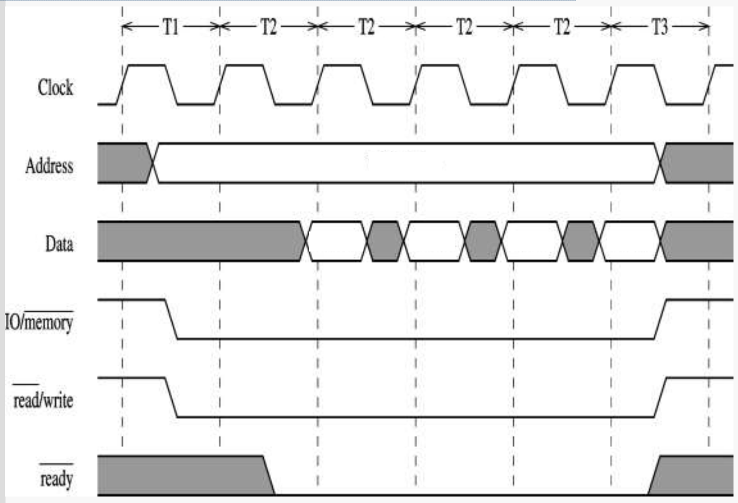
More Examples: Read, no Wait


T1
T2
T3
Clock
Address
Data
IO/$\overline{\text{memory}}$
$\overline{\text{read}}$/write
$\overline{\text{ready}}$

More Examples: Read, with Wait


T1
T2
Tw
T3
Clock
Address
Data
IO/$\overline{\text{memory}}$
$\overline{\text{read}}$/write
$\overline{\text{ready}}$

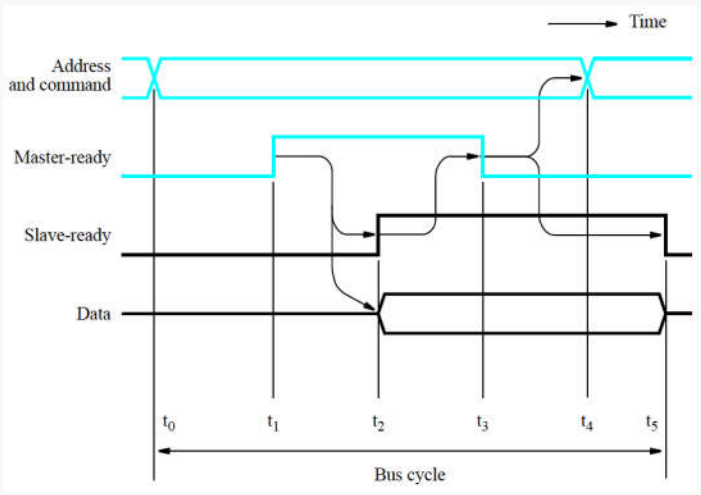
More Examples: Block Read


T1
T2
T2
T2
T2
T3
Clock
Address
Data
IO/$\overline{\text{memory}}$
$\overline{\text{read}}$/write
$\overline{\text{ready}}$

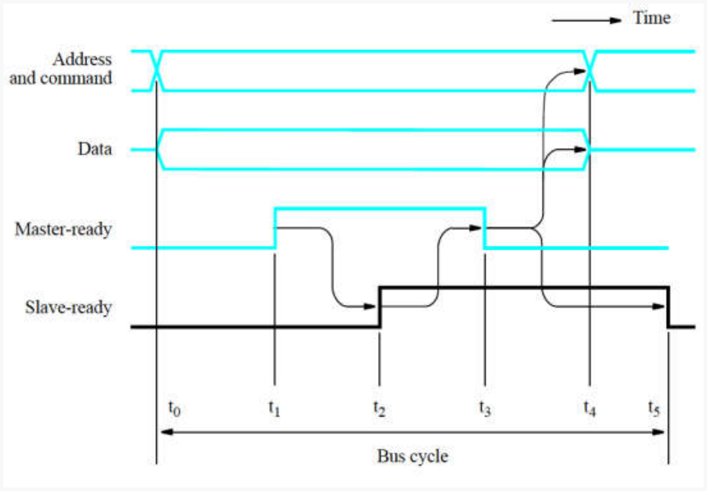
Asynchronous Bus

- In an asynchronous bus, the clock becomes optional and is replaced by two REQUIRED signals, Master-ready and Slave-ready
  - The clock may still be present, but only as a reference
- **Handshake** idea:
  - The master places the address and command on the bus and asserts Master-ready
    - All devices on the bus decode the address
  - The selected slave performs the requested operation and asserts Slave-ready
    - The master waits for Slave-ready to be asserted before removing its signals from the bus
    - Once Master-ready is deasserted, the slave removes its signals from the bus

Asynchronous Read Example



Asynchronous Write Example

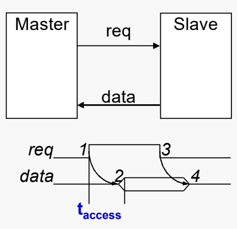


More on Asynchronous Bus

- Bus design tradeoffs:
  - Simplicity of the device interface circuit
  - Total time required for a bus transfer
  - Ability to accommodate devices with different delays
  - Ability to detect errors during a transfer
- Asynchronous bus advantages
  - No need for a synchronizing clock = simpler timing
  - Insensitivity to device delay variations
  - High degree of flexibility and reliability
- Asynchronous bus disadvantages
  - Fully interlocked handshaking involves two round-trip delays per each transfer
    - **Master-ready** must wait for **Slave-ready** and vice versa

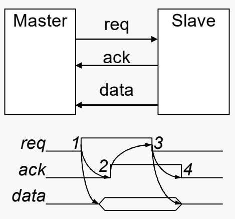
Protocol Concepts (e.g., Read)

Strobe Protocol



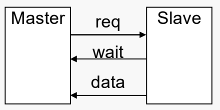
1. Master asserts ***req*** to receive data
2. Slave puts data on bus within time t_{access}
3. Master receives data, deasserts ***req***
4. Slave is ready for next request

Handshake Protocol

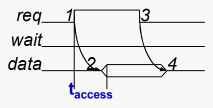


1. Master asserts ***req*** to receive data
2. Slave puts data on bus, asserts ***ack***
3. Master receives data, deasserts ***req***
4. Slave stops driving bus, deasserts ***ack***
5. Slave is ready for next request

Strobe/Handshake Compromise

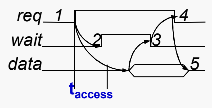


Fast-response case



1. Master asserts ***req*** to receive data
2. Slave puts data on bus within t_{access} (***wait*** is unused)
3. Master receives data, deasserts ***req***
4. Slave is ready for next request

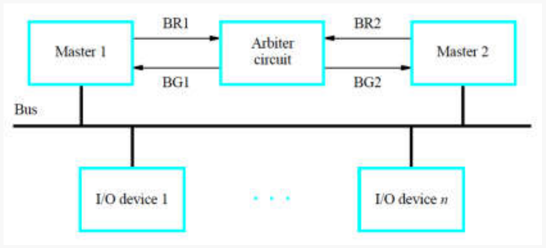
Slow-response case



1. Master asserts ***req*** to receive data
2. Slave cannot put data within t_{access}, it asserts ***wait***
3. Slave puts data on bus, deasserts ***wait***
4. Master receives data, deasserts ***req***
5. Slave stops driving bus
6. Slave is ready for next request

Bus Arbitration

- Bus arbitration – the process of determining which device is to become the next bus master and then transferring the mastership to it
  - Need a priority scheme for gaining access to the bus
- Two approaches: **distributed** (all devices are involved) and **centralized** (similar to interrupts)

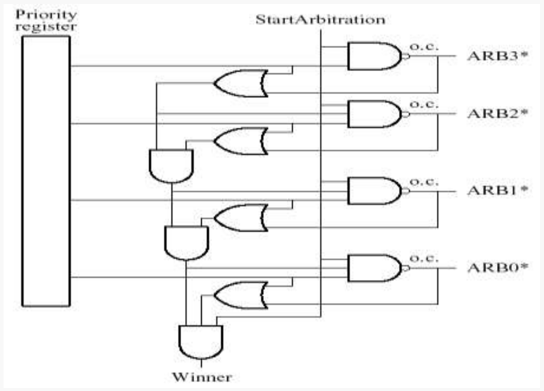


Distributed Arbitration I

- All devices are responsible for carrying out the arbitration process, without using a central arbiter
- Example:
  - Each device has its 4-bit **ID**, which is put on the bus control lines with asserted signal Start-Arbitration (when requesting a bus access)
  - A winner is selected as a result of the interaction among the signals transmitted by all current contenders
    - Assume devices **A** (**ID=5**) and **B** (**ID=6**) are requesting the use of the bus
    - **A** puts **0101** and **B** puts **0110** – the open-collector (o.c.) driven lines will be at state **0111** (i.e., logical *OR* of **0101** and **0110**)
    - **A** and **B** compare **0111** to their **ID**s starting from the MSB
    - **A** detects a mismatch earlier and disables its **ID** drivers, thus making the lines change their state to **0110** – **B** wins!

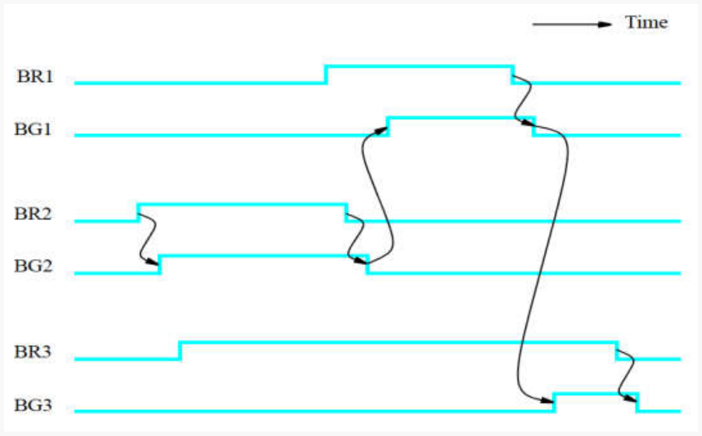
Distributed Arbitration II

- Let device's **ID** be stored in its Priority Register
  - 1111 means the highest priority, 0000 – the lowest
  - If a device wins, it asserts its output **Winner**

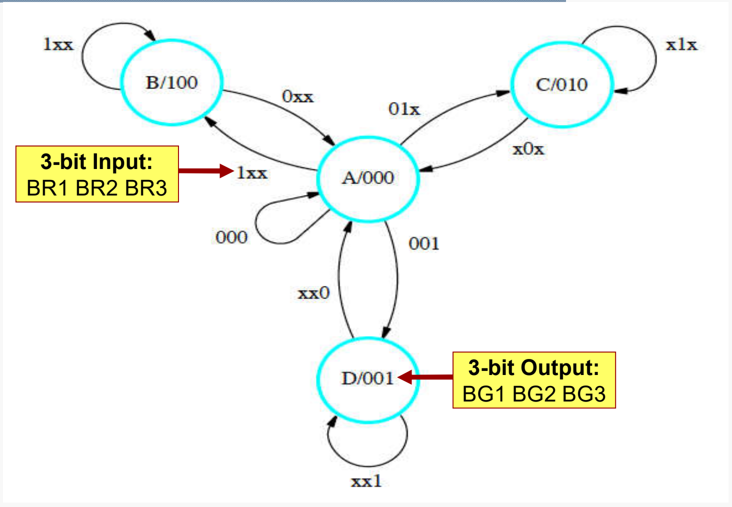


Centralized Arbitration I

Priorities: BR1 > BR2 > BR3

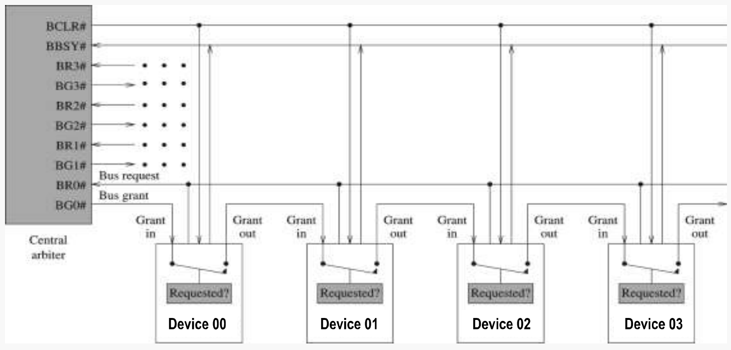


Centralized Arbitration II

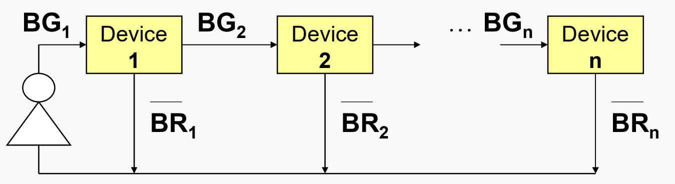


Hybrid Daisy Chain

- Devices are divided into **N** classes
- Each class has *independent* bus-request with bus-grant
- Devices within the same class form a *daisy chain*

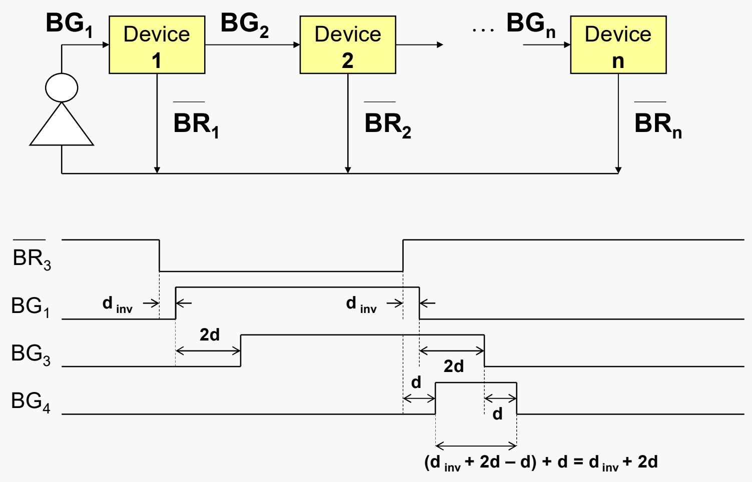


Daisy Chain Example I

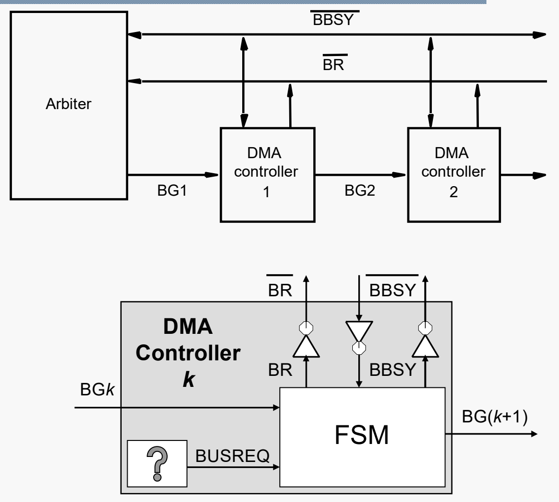


Ignoring propagation delay on wires, assume that the inverter delay is $\mathbf{d_{inv}}$, and the delay from $\mathbf{BG_k}$ to $\mathbf{BG_{k+1}}$ for any device is $\mathbf{d > d_{inv}}$. Assume that device **3** requests the bus and begins using it; when finished, device **3** deactivates $\mathbf{/BR_3}$. As a result, a spurious (unwanted) bus grant pulse will travel down the daisy chain.

Daisy Chain Example II

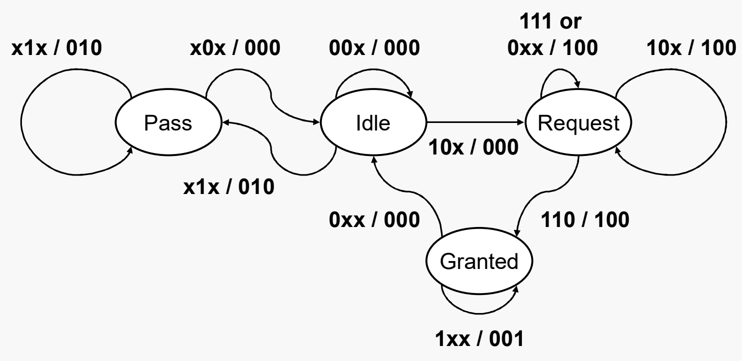


Daisy Chain of DMA Controllers

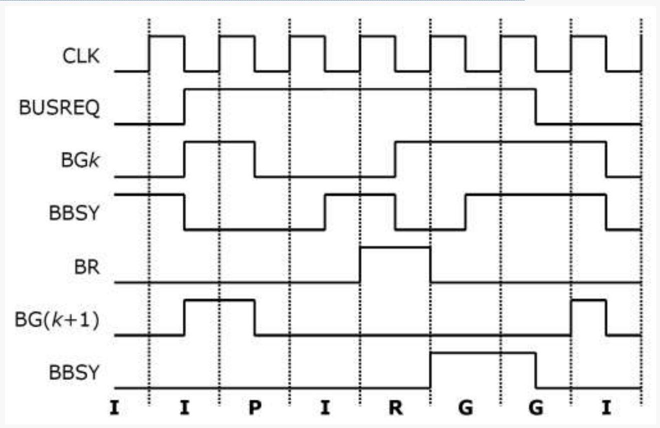


FSM State Diagram

Inputs / Outputs = BUSREQ, BG*k*, BBSY / BR, BG(*k*+1), BBSY



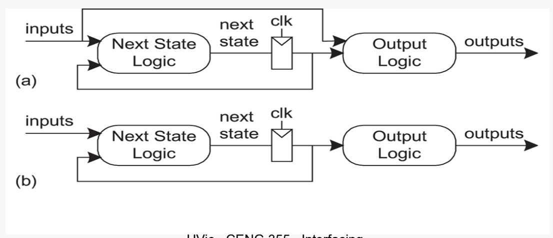
FSM Waveform Example



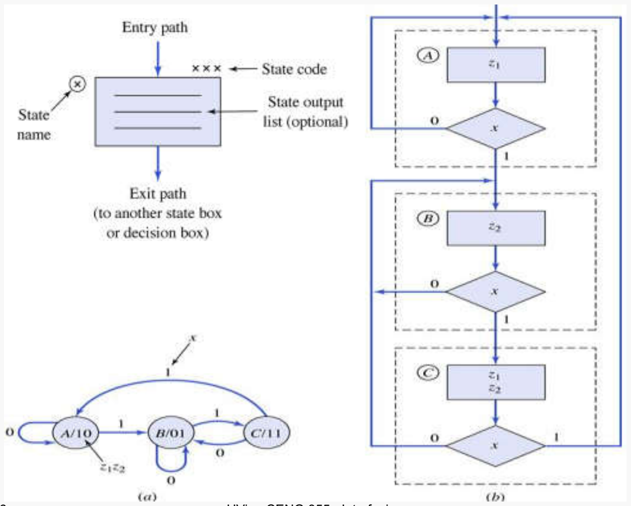
I = Idle P = Pass R = Request G = Granted

FSM Design

- FSM design procedure
  - Determine state diagram (your **design**)
  - Encode states
  - Implement next-state logic and output logic
- Mealy (a) and Moore (b) FSMs
  - **Mealy**: output depends on input, one cycle faster
  - **Moore**: output does not depend on input, more reliable

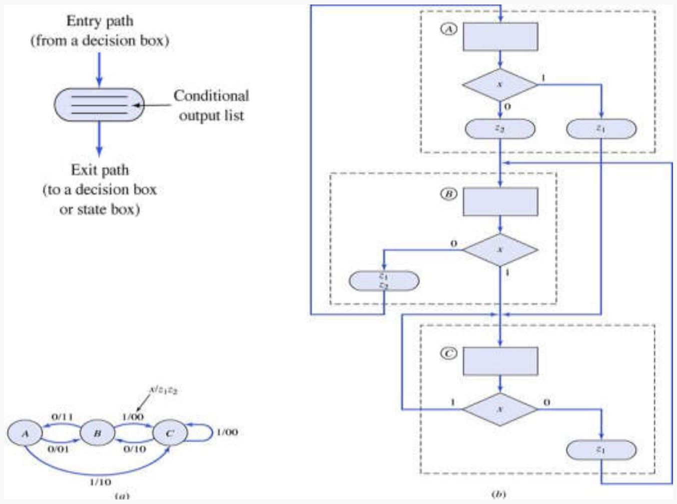


Moore ASM Chart



Entry path
State code
State output list (optional)
State name
Exit path (to another state box or decision box)
A/10
B/01
C/11
z_1
z_2
x
(a)
(b)

Mealy ASM Chart

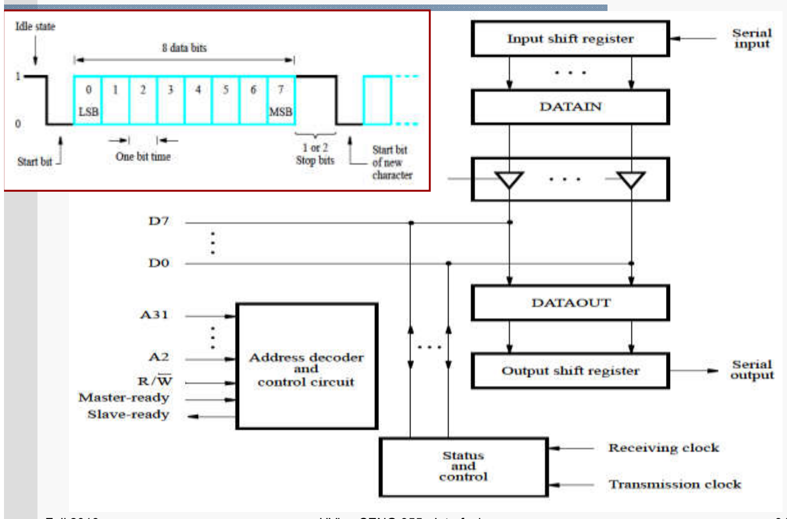


Entry path (from a decision box)
Conditional output list
Exit path (to a decision box or state box)
x/z_1z_2
0/11
1/00
0/01
0/10
1/10
1/00
(a)
(b)

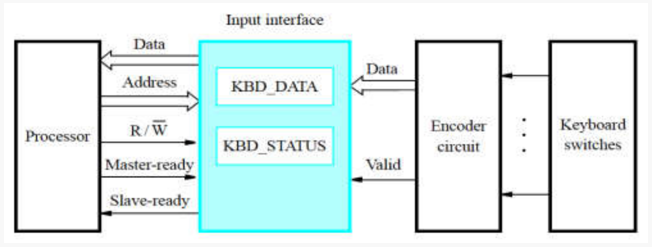
I/O Interface Circuits

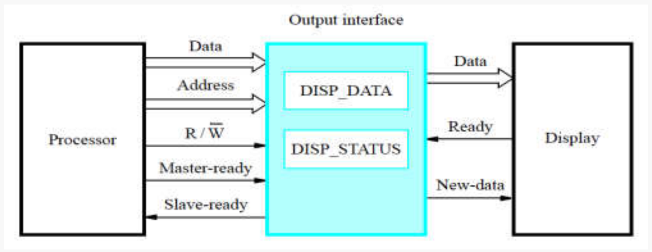
- Basic I/O interface functions:
  - Contains address decoding circuitry that detects when the I/O device is being addressed
  - Generates the appropriate signals required for bus control
  - Provides a buffer for temporary data storage
  - Contains status flags that can be accessed to determine whether the buffer is full (input) or empty (output)
  - Performs necessary data format conversions between the bus and the I/O device (e.g., serial-to-parallel conversion)
- The interface circuit classifies the I/O device as either serial or parallel port

Serial I/O Interface

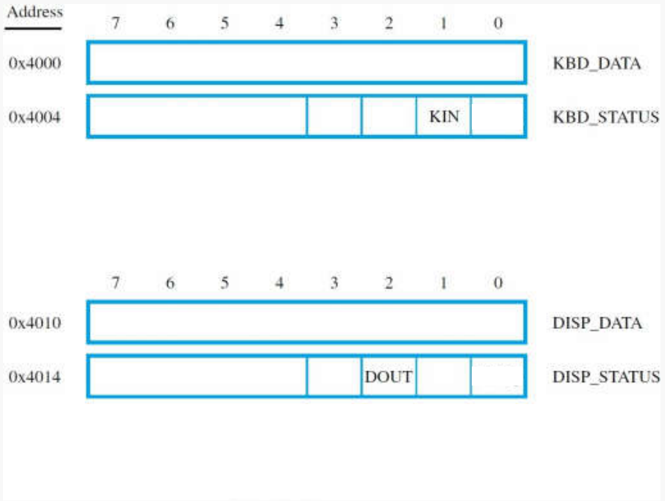


Example: Parallel Ports

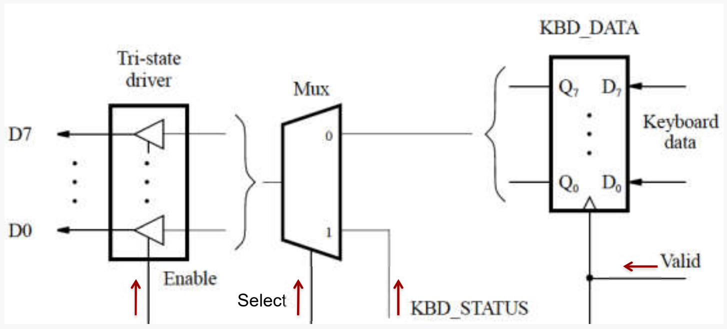




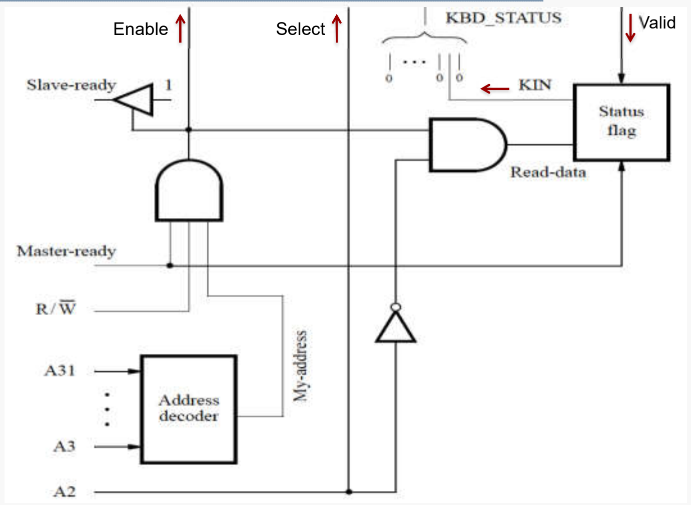
Example: Interface Registers



Parallel Input Interface I

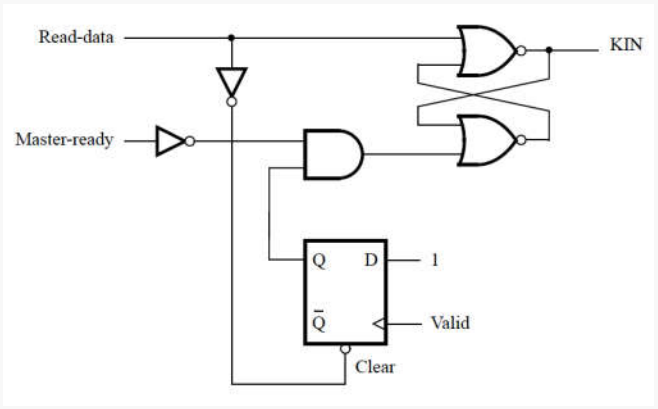


Parallel Input Interface II

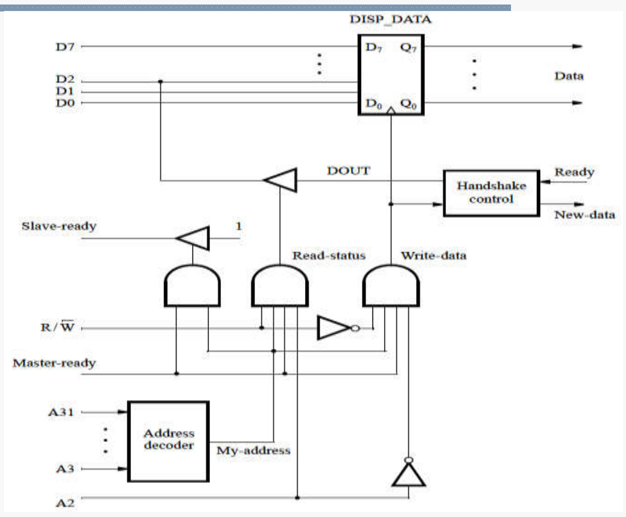


Status Flag Control

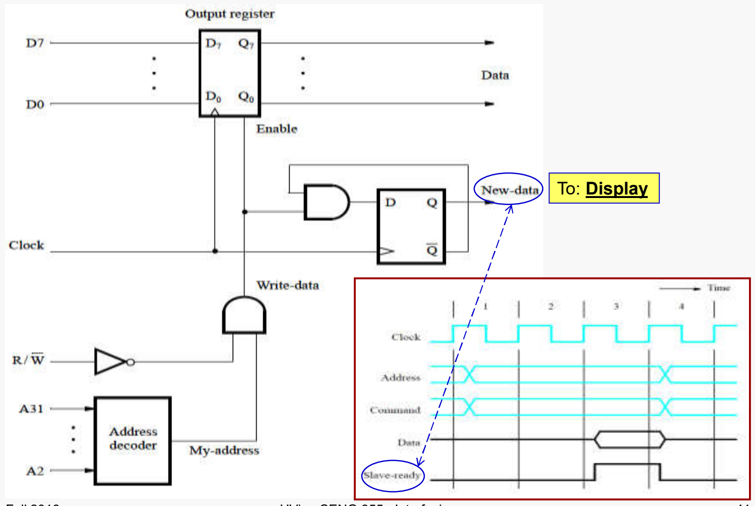
KIN is set by **Valid** (when **Master-ready** is not asserted) and cleared by **Read-data** (when **Master-ready** is asserted)…



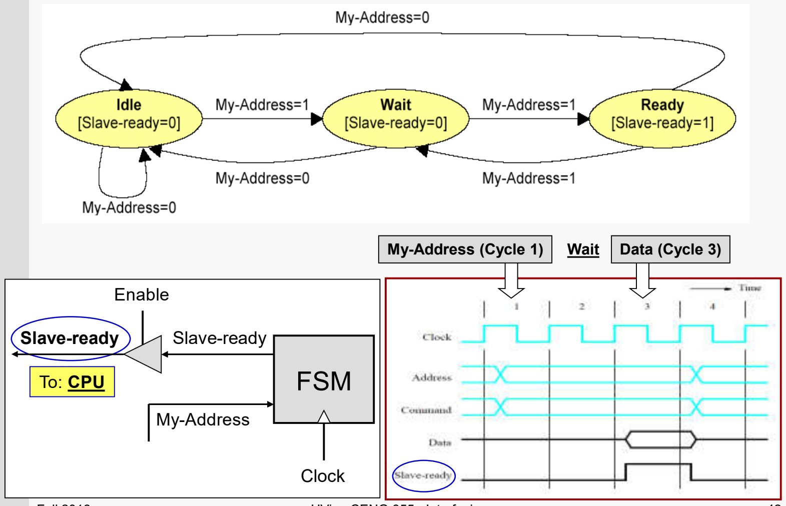
Parallel Output Interface



Synchronous Interfacing I



Synchronous Interfacing II


My-Address=0
Idle [Slave-ready=0]
My-Address=1
Wait [Slave-ready=0]
My-Address=1
Ready [Slave-ready=1]
My-Address=0
My-Address=1
My-Address=0
My-Address (Cycle 1)
Wait
Data (Cycle 3)
Enable
Slave-ready
Slave-ready
To: CPU
FSM
My-Address
Clock
Time
1
2
3
4
Clock
Address
Command
Data
Slave-ready