Interfacing

D.N.Rakhmatov

Adopted (with modifications) from:

R. Katz, UC-Berkeley

F. Vahid, UC-Riverside

D. Givone, SUNY-Buffalo

J. Armstrong, Virginia Tech

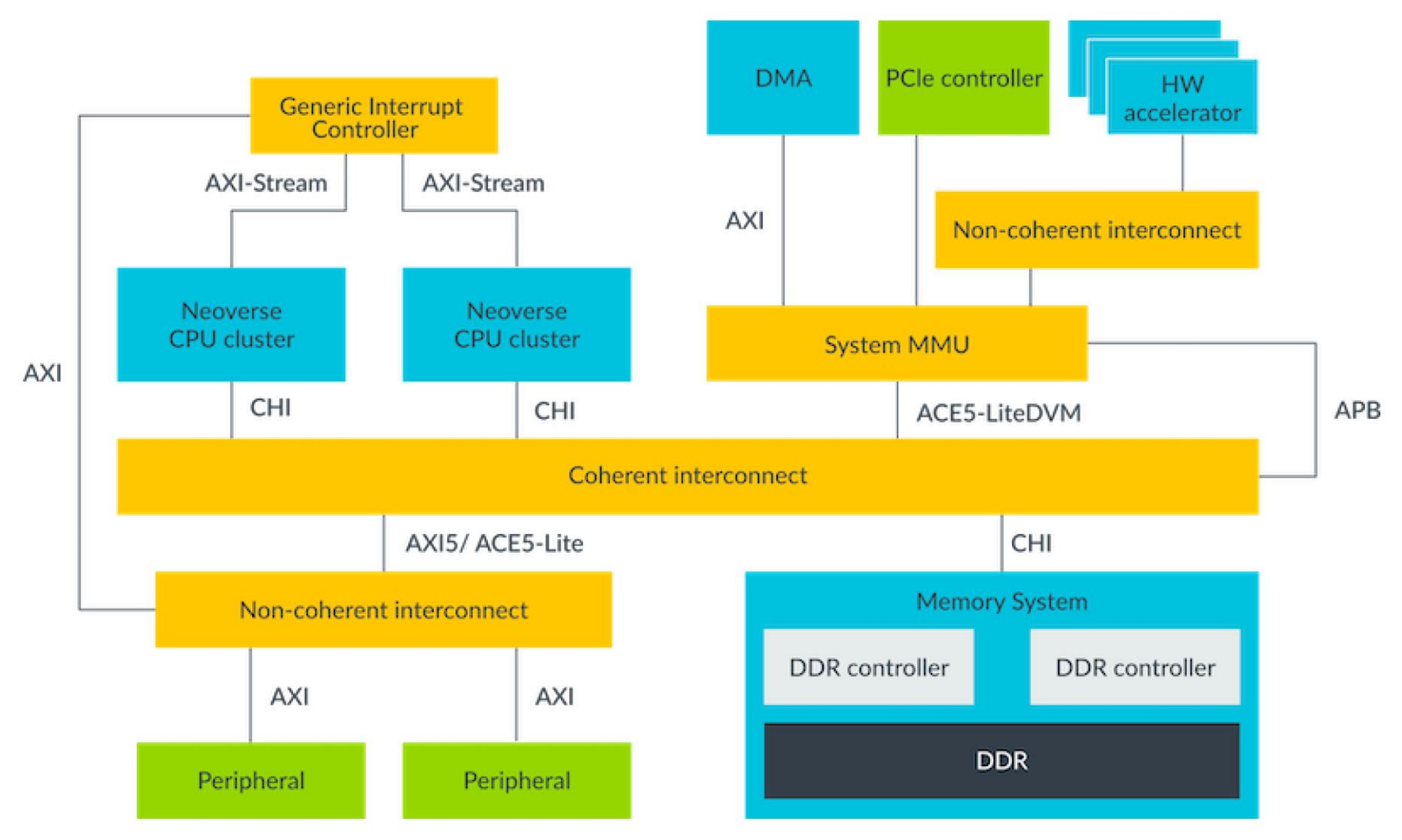
AMBA APB/AHB Protocol Specifications, © 2021 ARM

An Introduction to AMBA AXI Version 3.0, © 2022 ARM

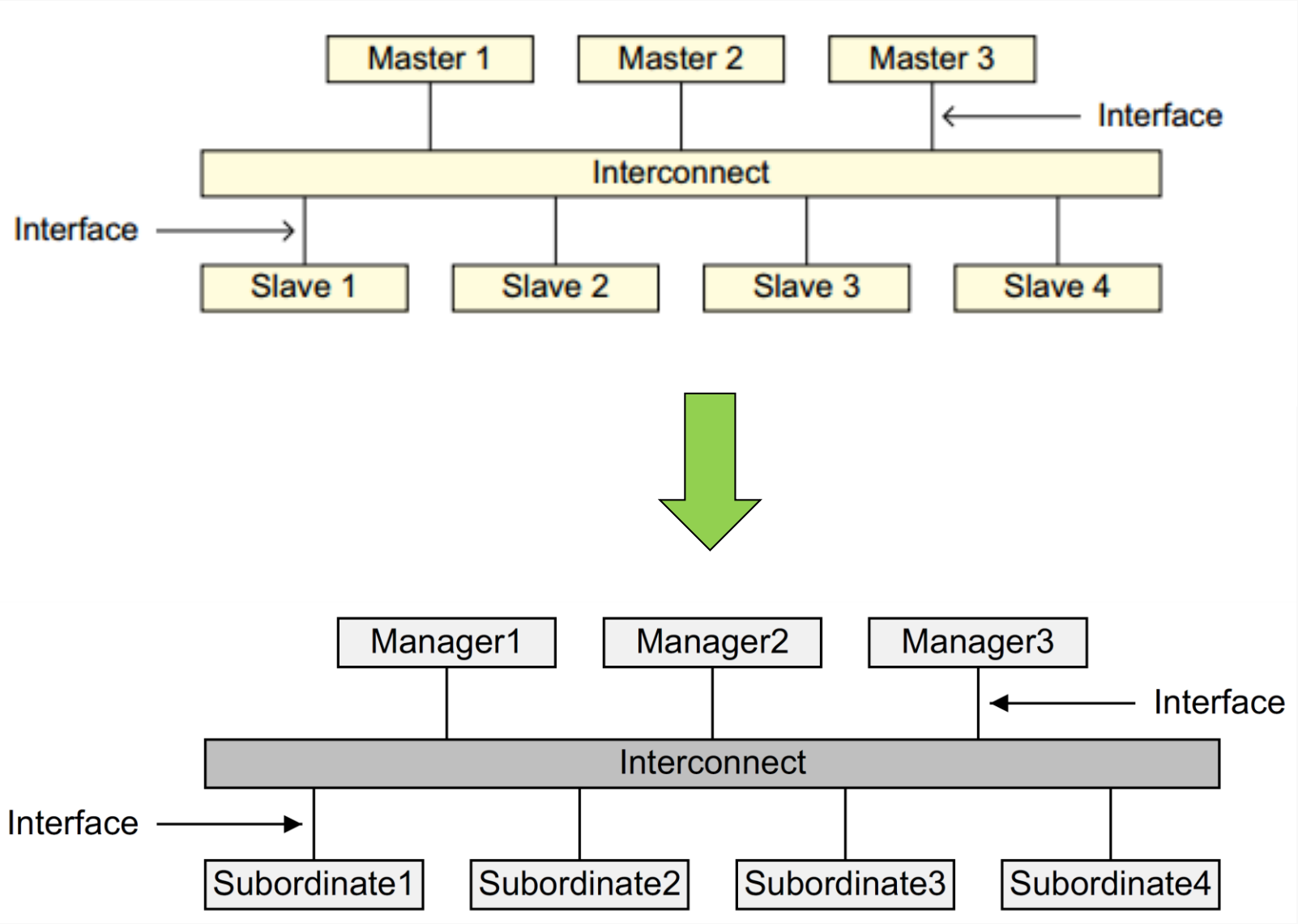
C. Hamacher et al, *Computer Organization*, 6/E, © 2011 McGraw-Hill

S. Dandamudi, *Fundamentals of Computer Organization and Design*, © 2002 Springer

System Integration Example



Changing Terminology



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

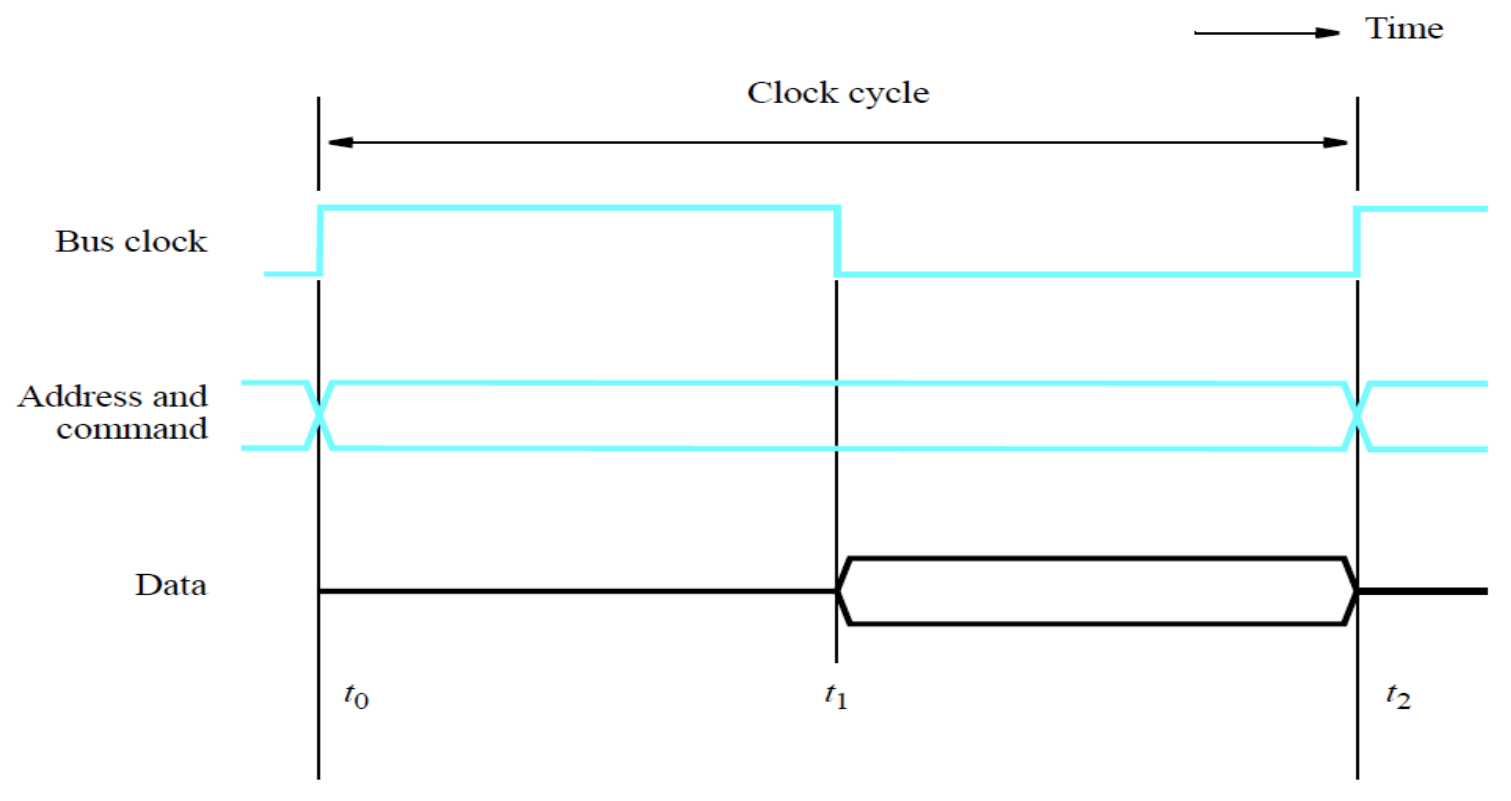
System Bus II

- 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
- 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

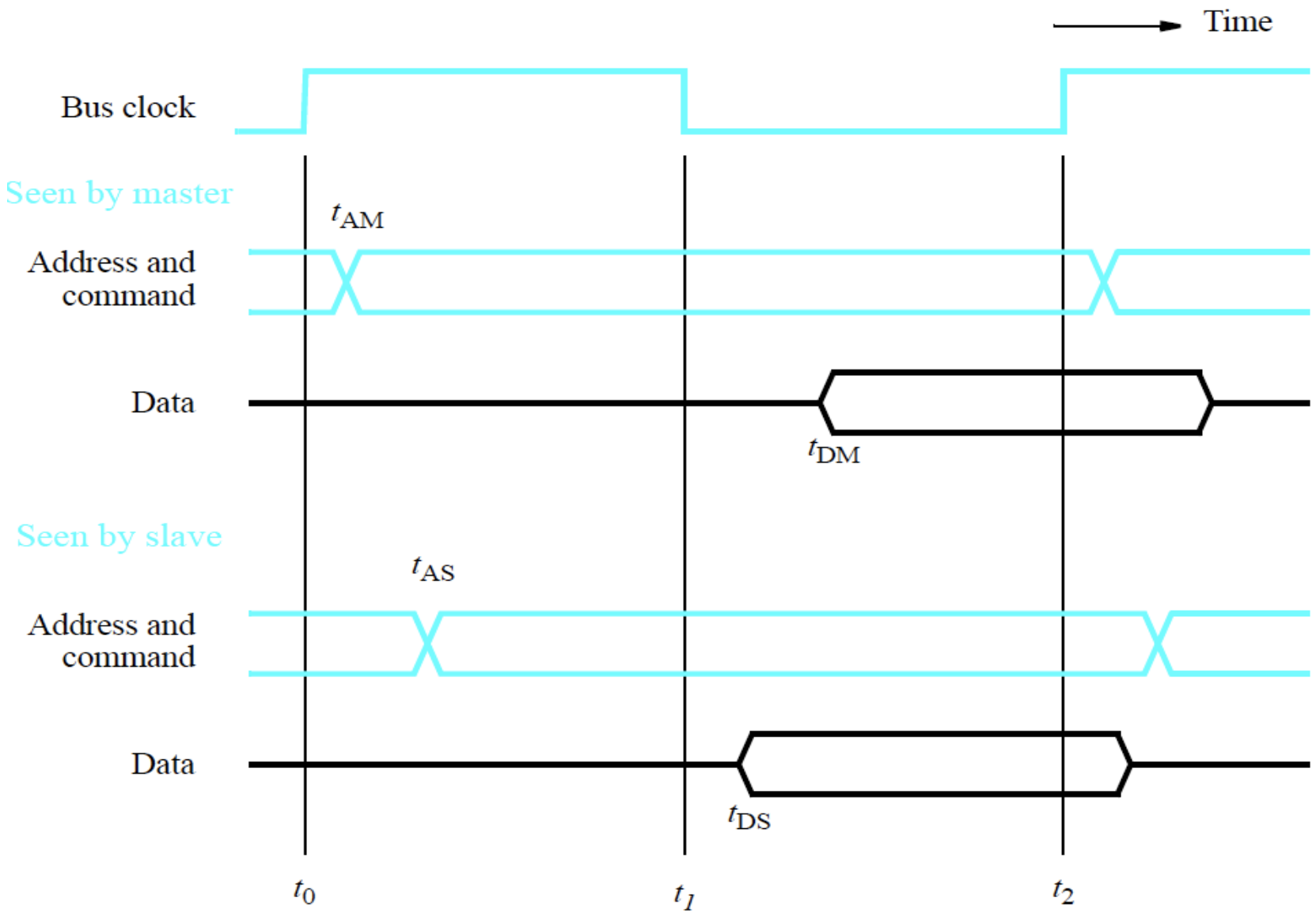
Synchronous Read Example



1. **(t1 – t0)** must be longer than the maximum propagation delay between the master and the slave + the slave's decoding delay to process the address and control signals

2. **(t2 – t1)** must be longer than the maximum propagation delay between the master and the slave + the master's setup time

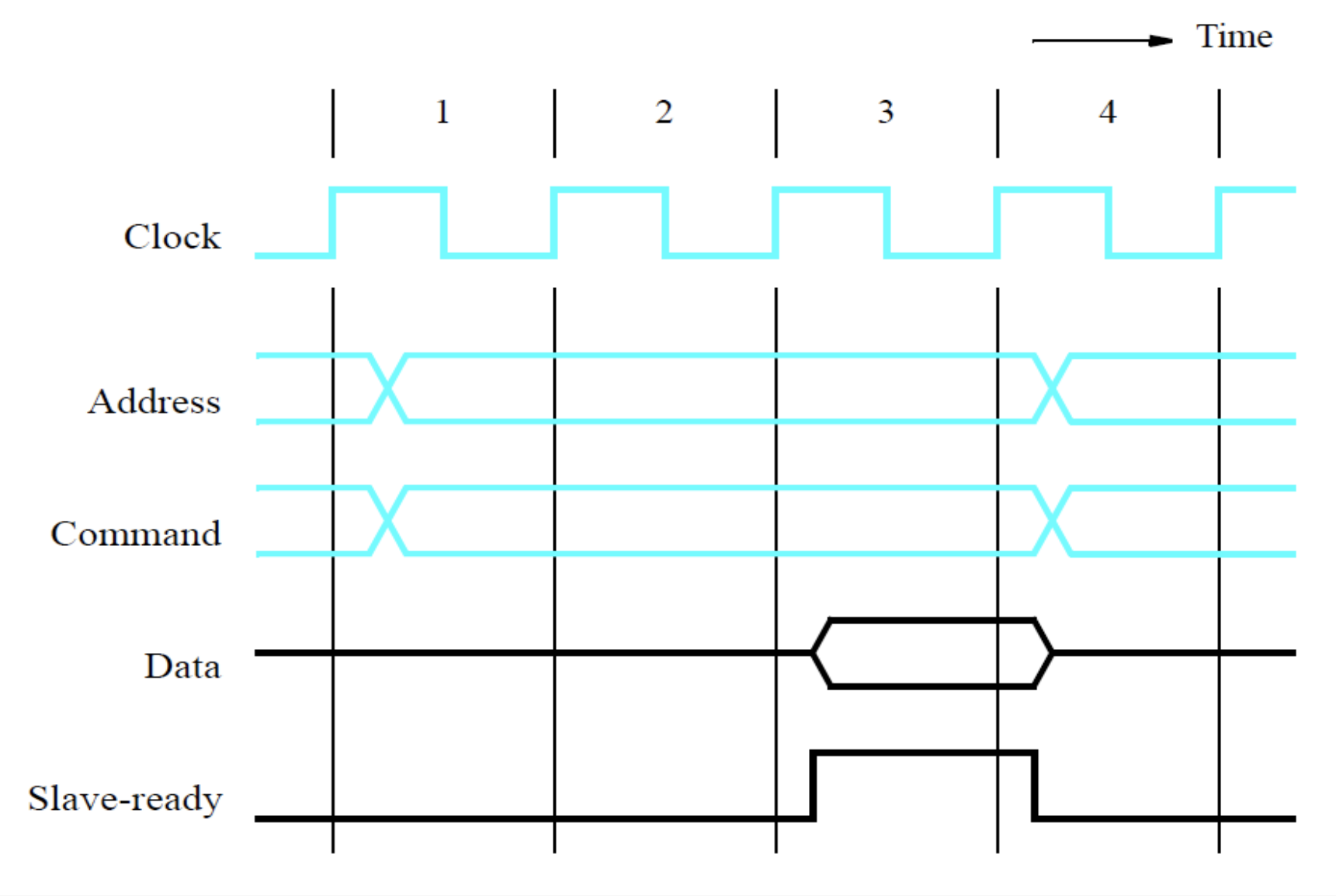
More Detailed Picture (Read)



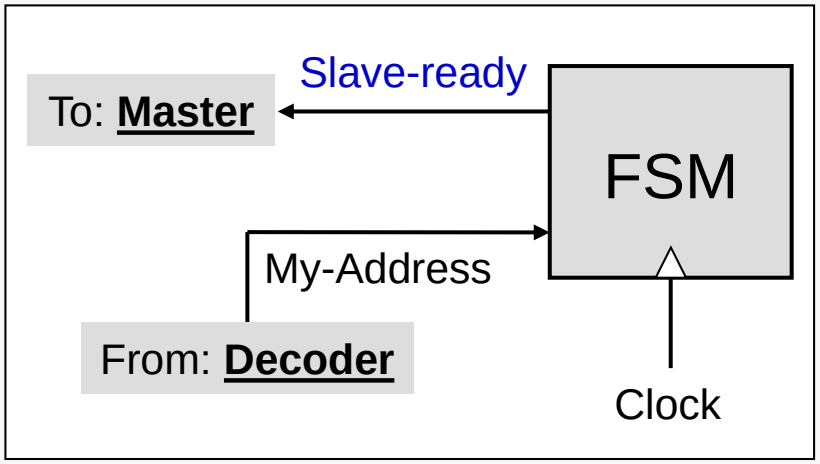
Multi-Cycle Bus Transfers

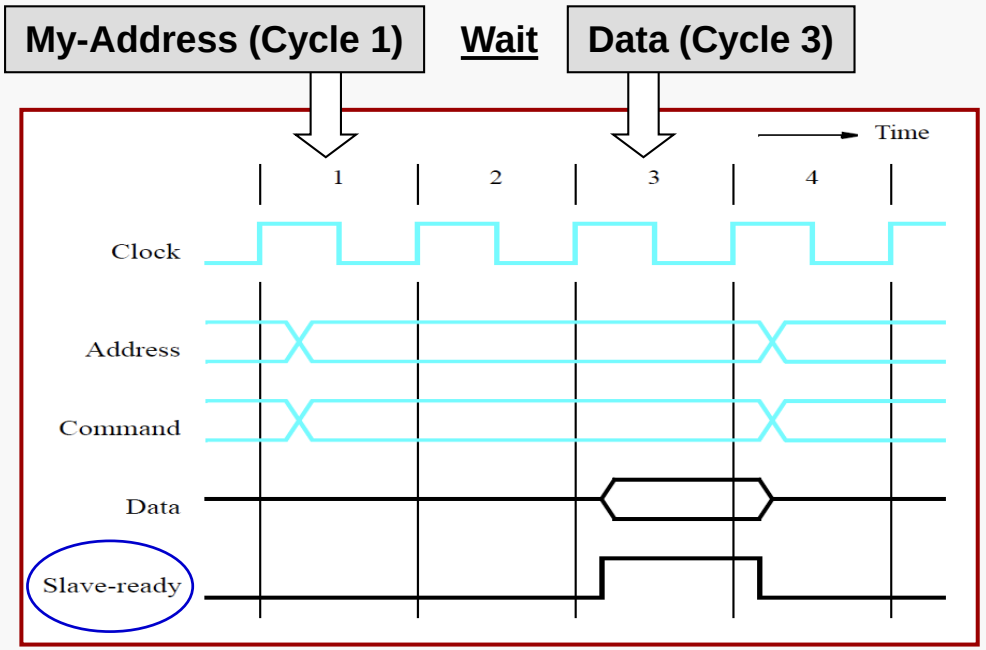
- **Single-cycle** bus transfer mechanism forces all devices to operate at the speed of the slowest one
  - Also, the master cannot determine if the selected slave has responded correctly
- **Multi-cycle** bus transfers increase bus utilization and transfer reliability
  - Example: multi-cycle read operation
    - ↑ **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

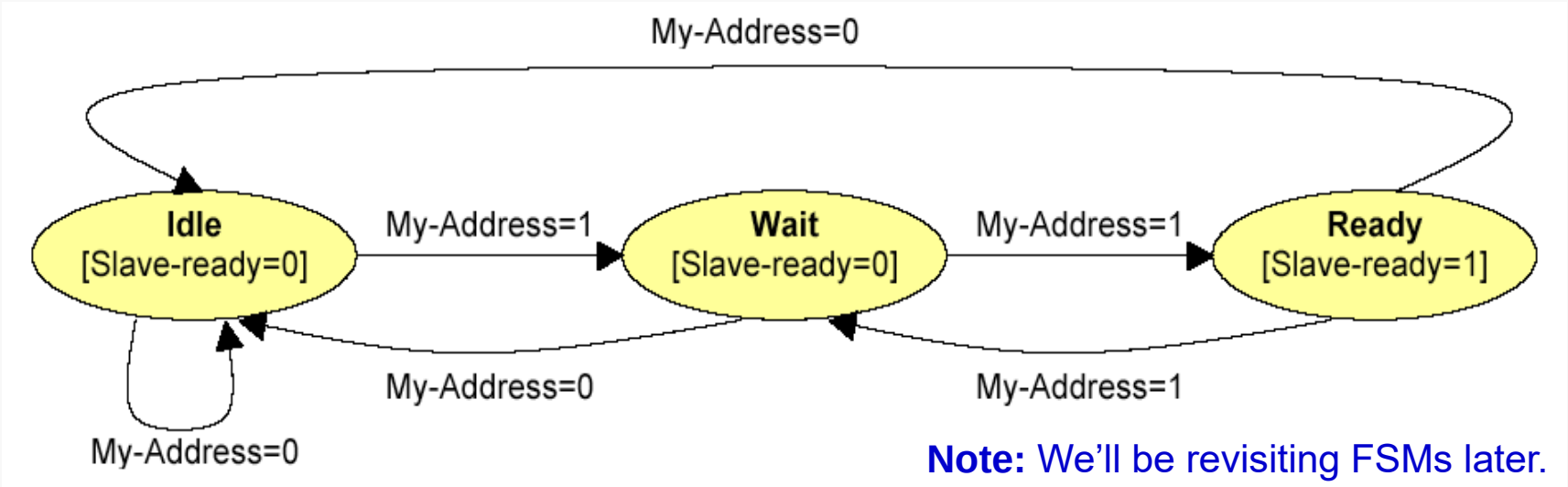
Example: Multi-Cycle Read



Example: Finite State Machine

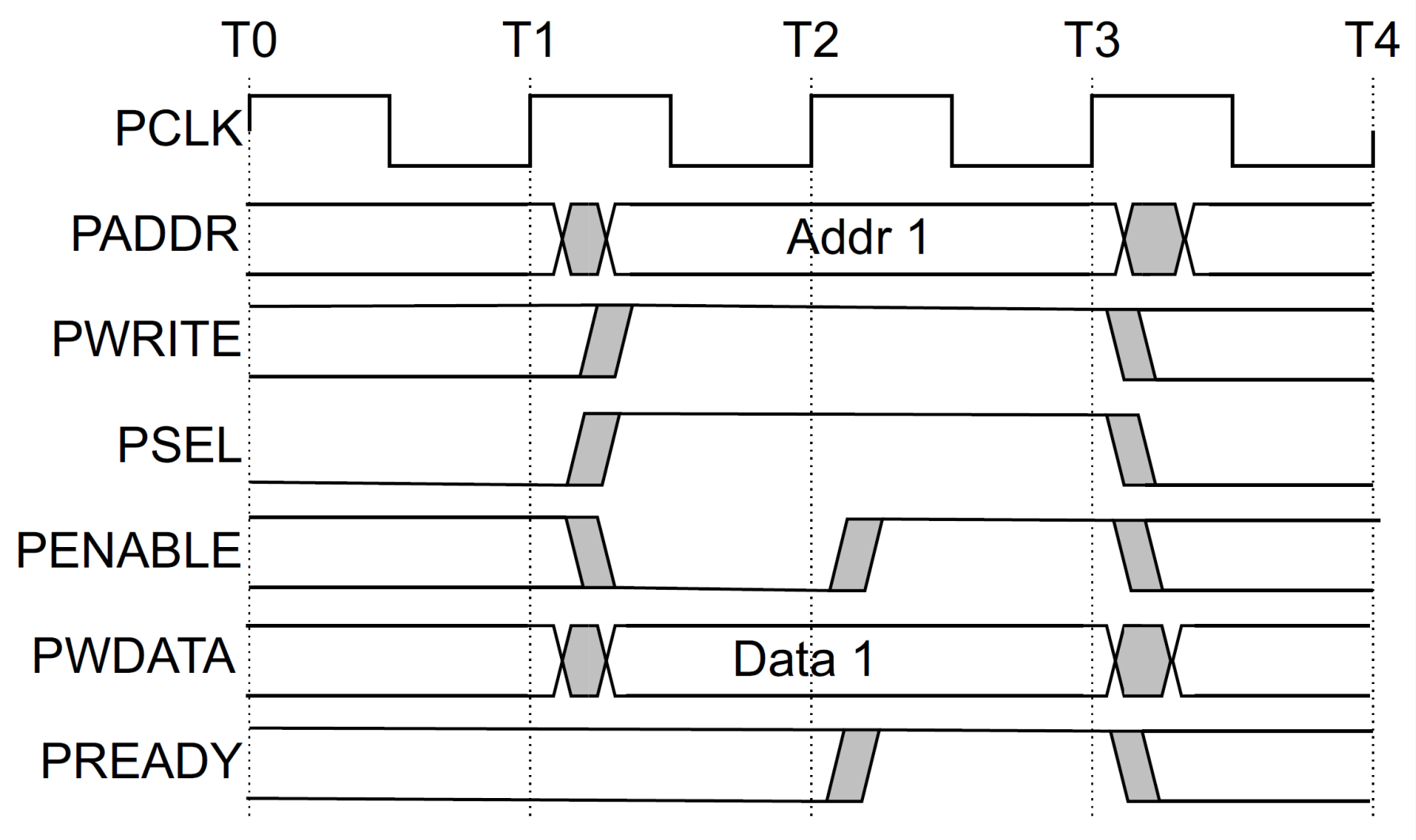






Note: We'll be revisiting FSMs later.

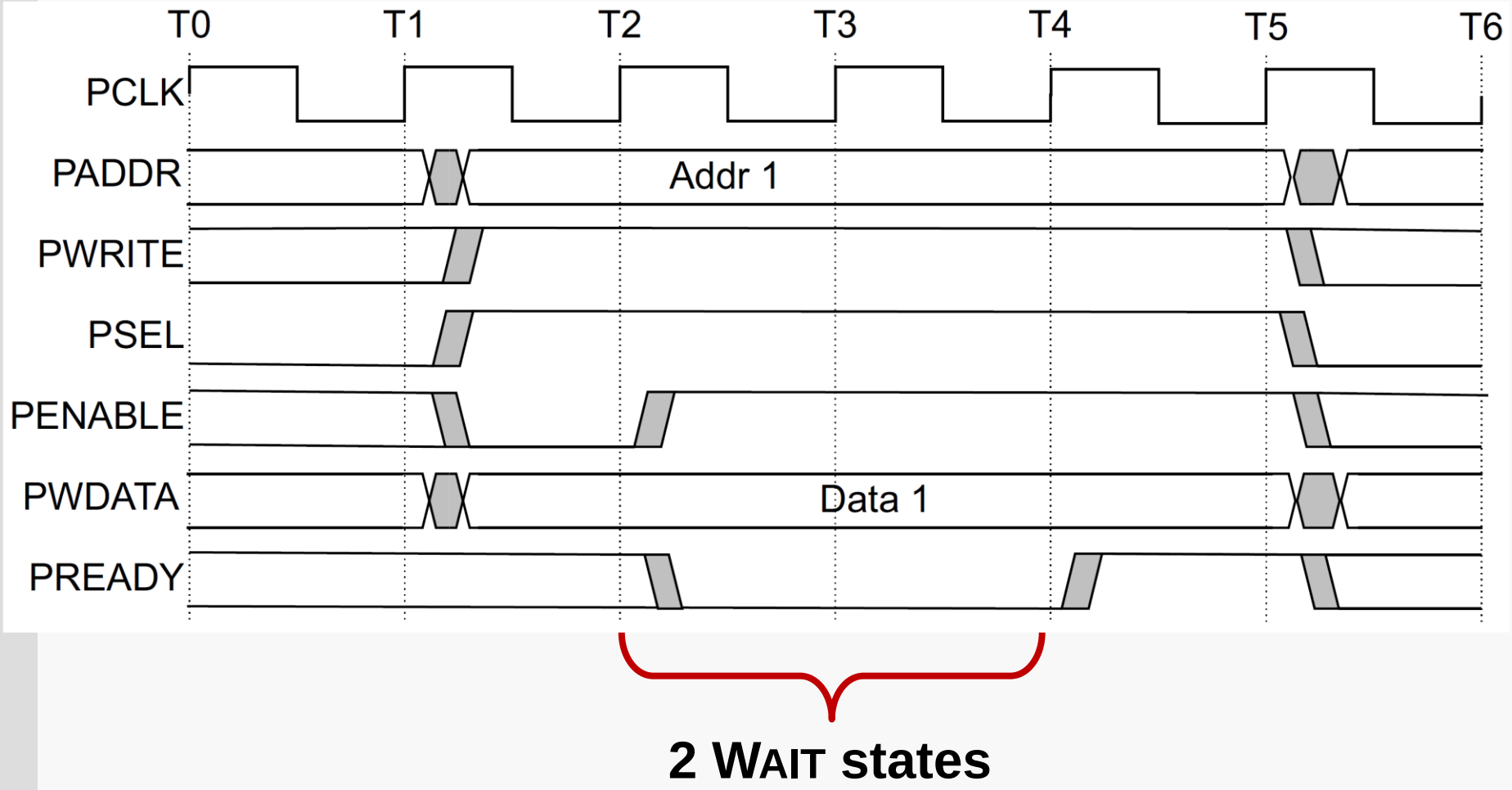
Example: AMBA APB Write



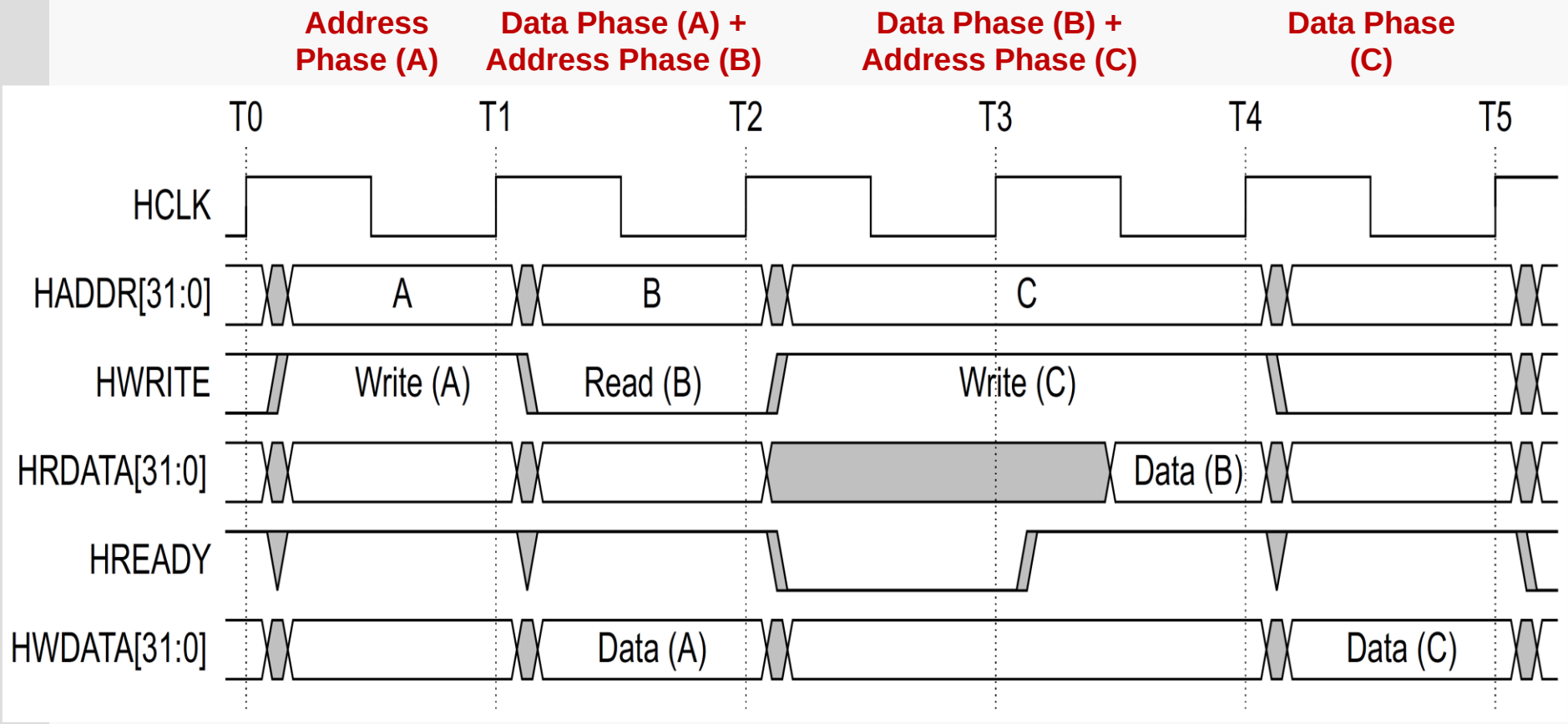
Setup Phase: **PENABLE = 0**

Access Phase: **PENABLE = 1**

Example: APB Write with Wait



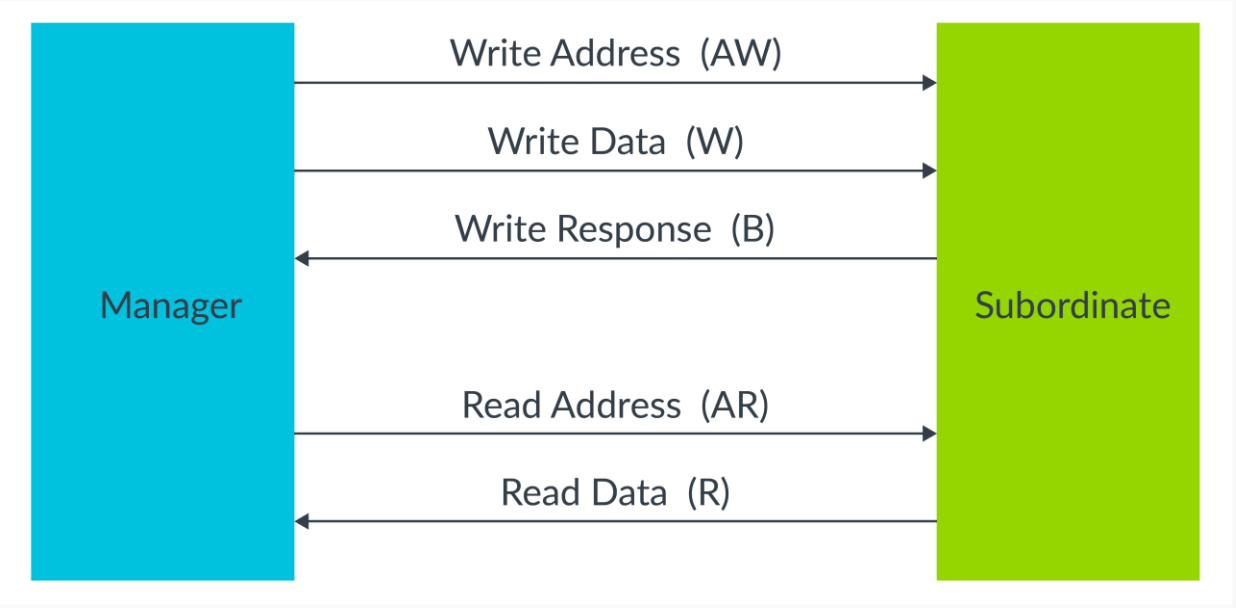
Example: AMBA AHB Transfers

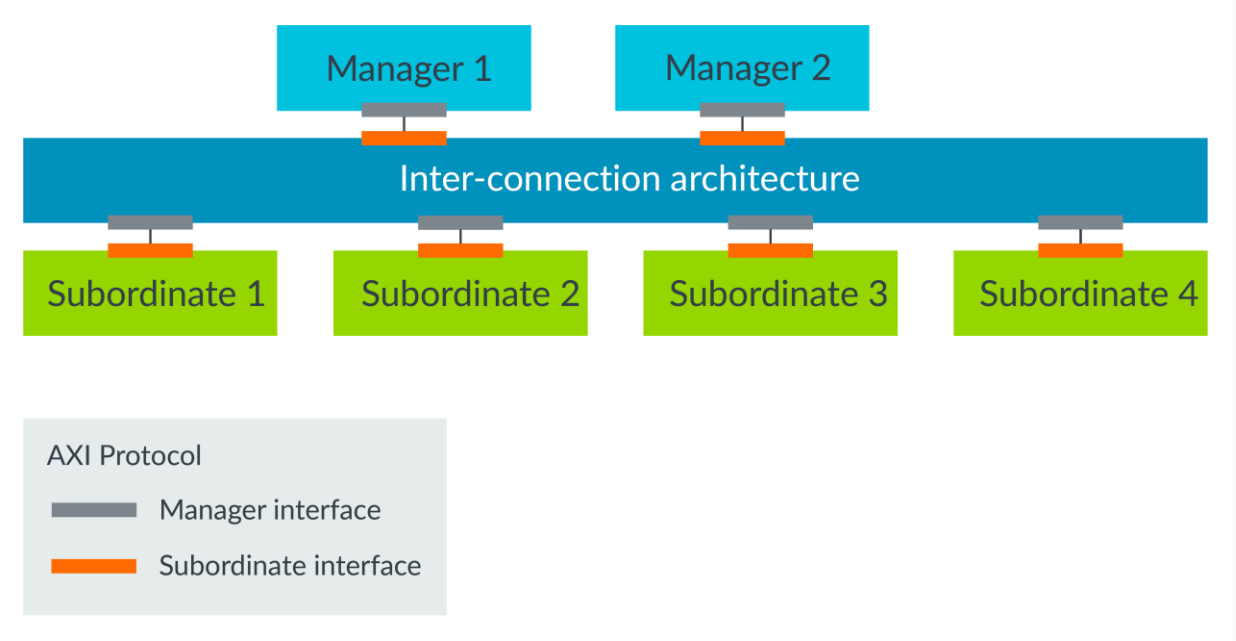


The transfers to addresses A and C are zero wait state

The transfer to address B is one wait state

Example: AMBA AXI Protocol

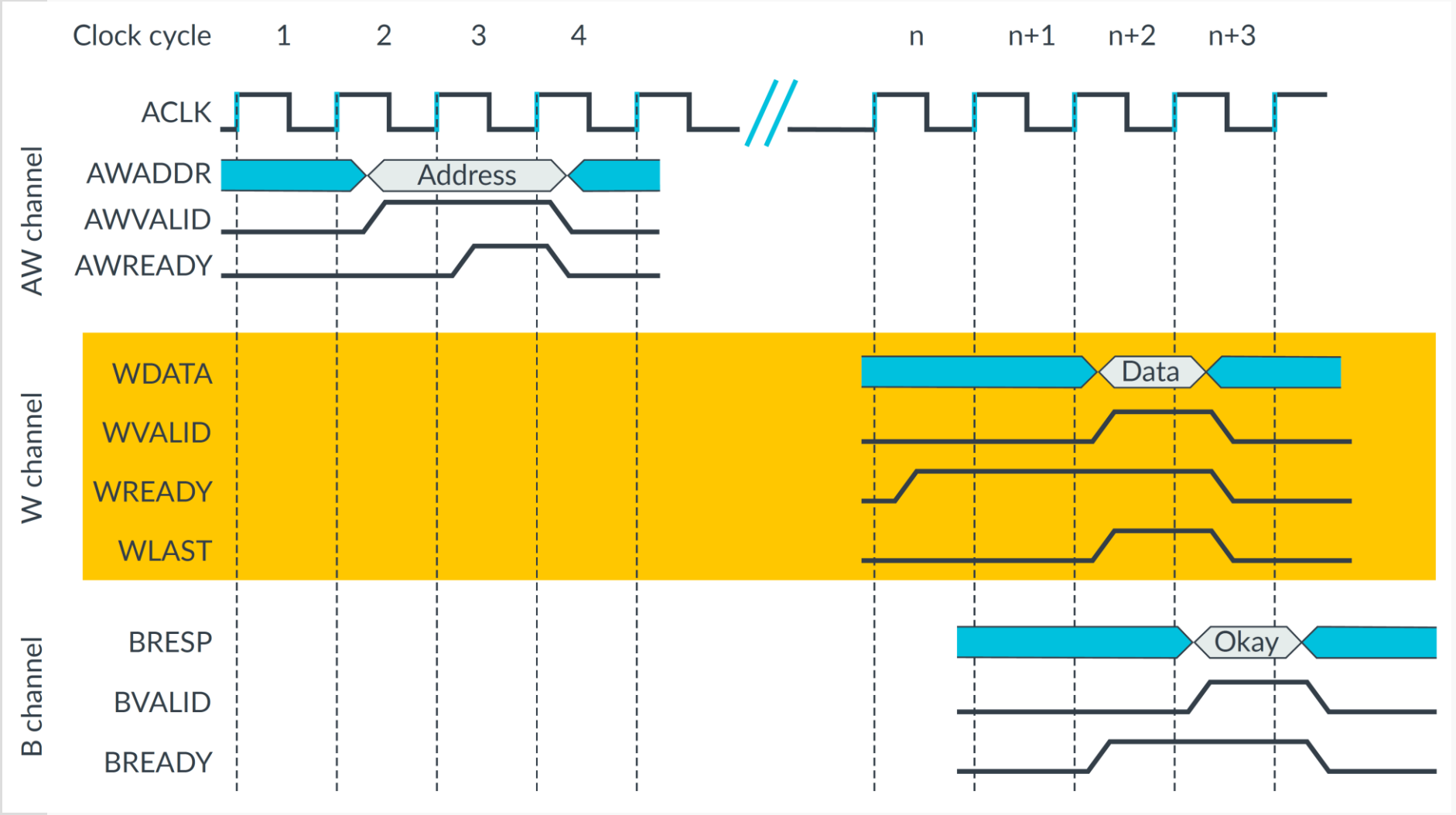




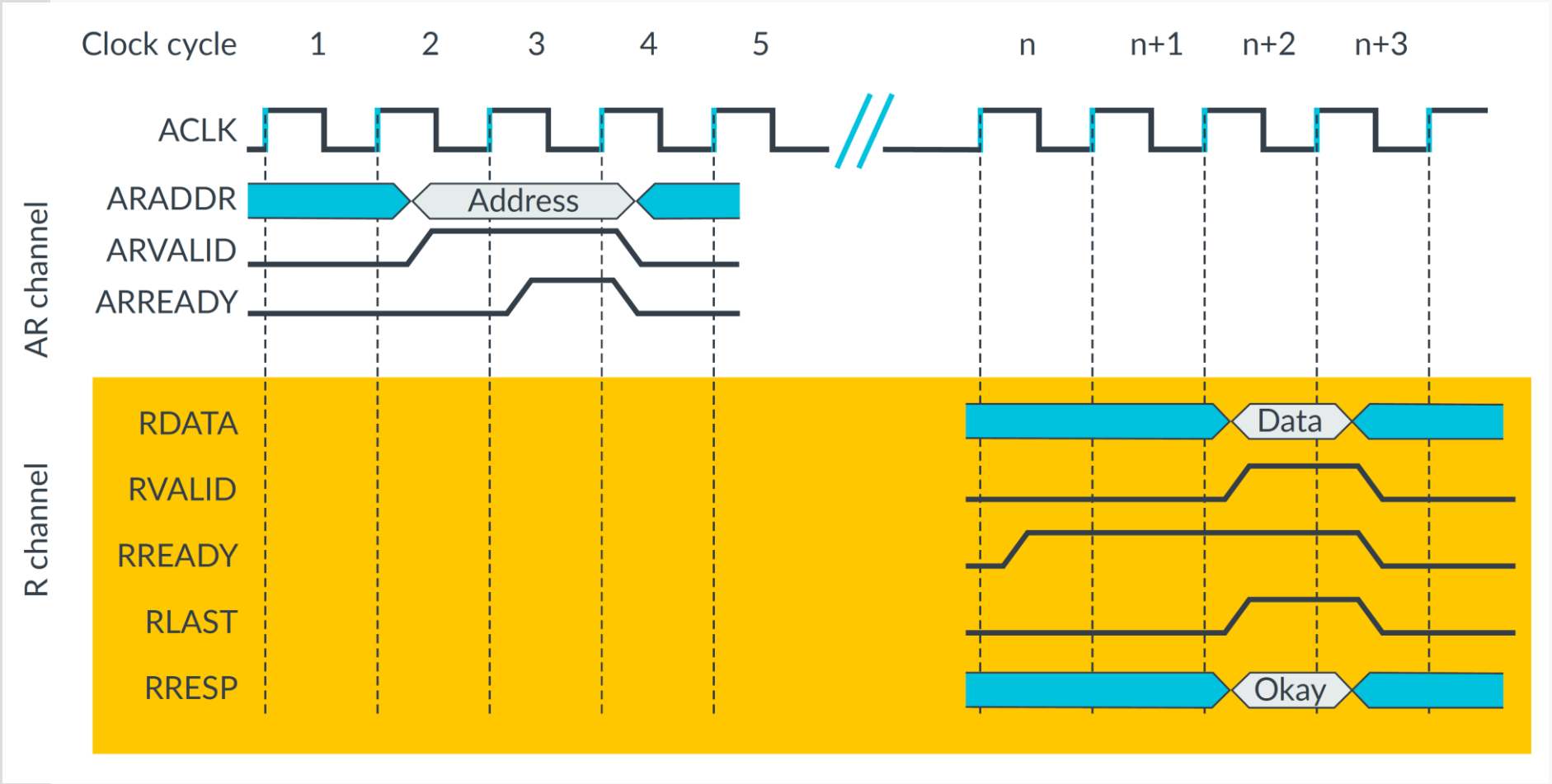
AMBA AXI Write/Read

- **Write** operation:
  1. The manager sends an address on the **Write Address** (AW) channel and transfers data on the **Write Data** (W) channel to the subordinate
  2. The subordinate writes the data to the specified address and then responds with a message to the manager on the **Write Response** (B) channel
- **Read** operation:
  1. The manager sends the address it wants to read on the **Read Address** (AR) channel
  2. The subordinate sends the data from the requested address to the manager on the **Read Data** (R) channel
     - The subordinate can also return an error message on the **Read Data** (R) channel when, for example, the address is not valid, or the data is corrupted, or the access lacks security permissions

Single Write Transaction



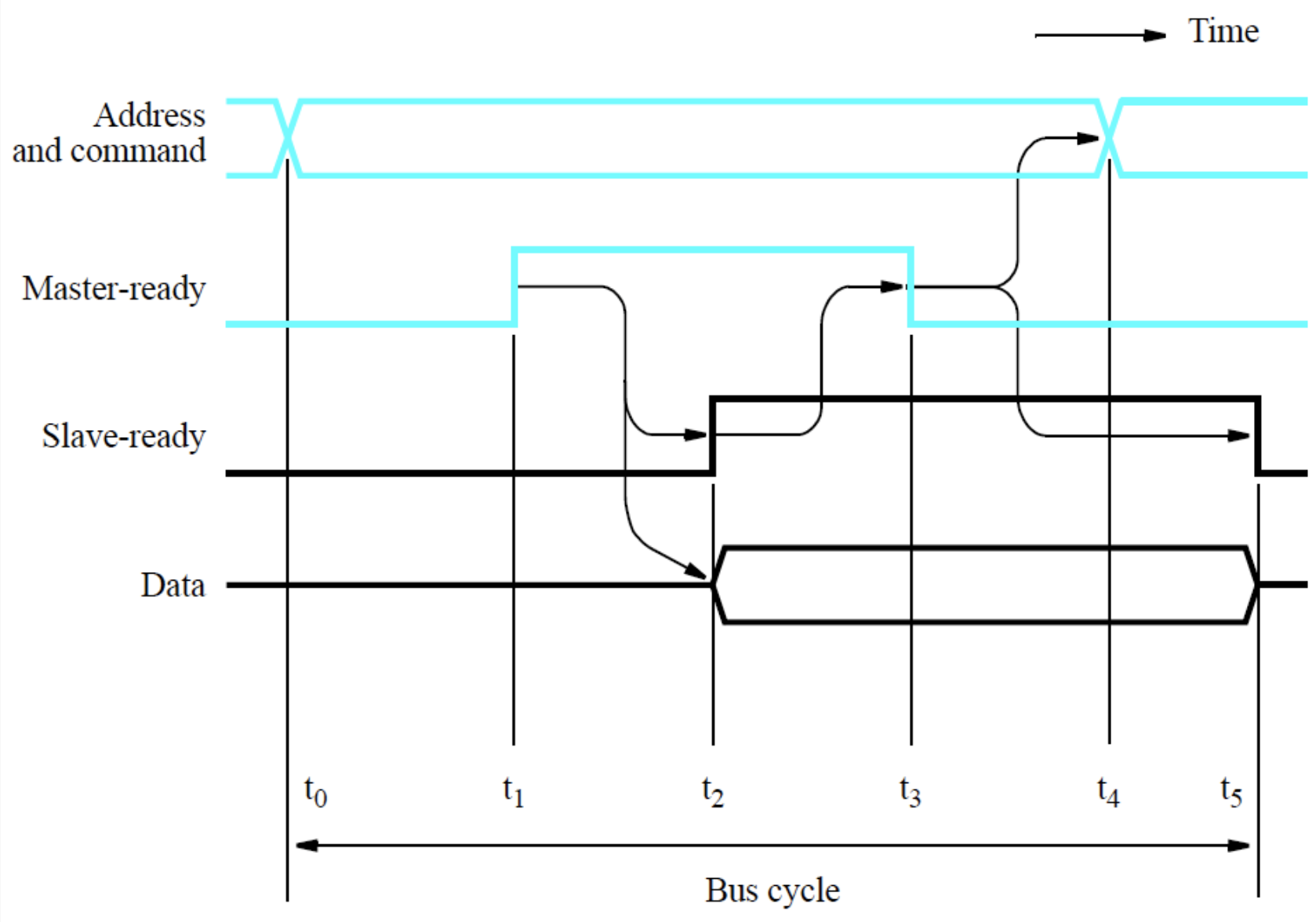
Single Read Transaction



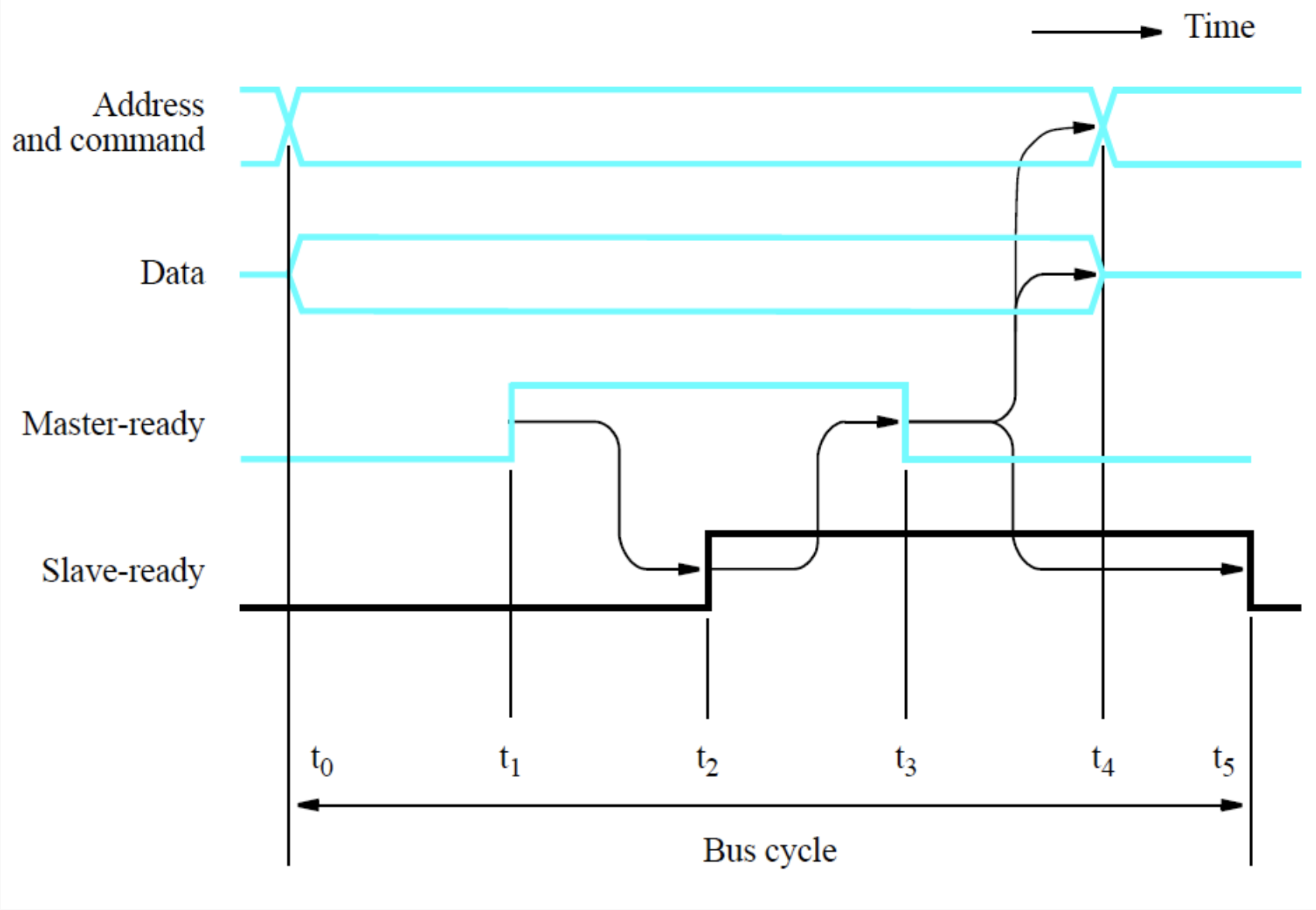
Asynchronous Bus

- 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

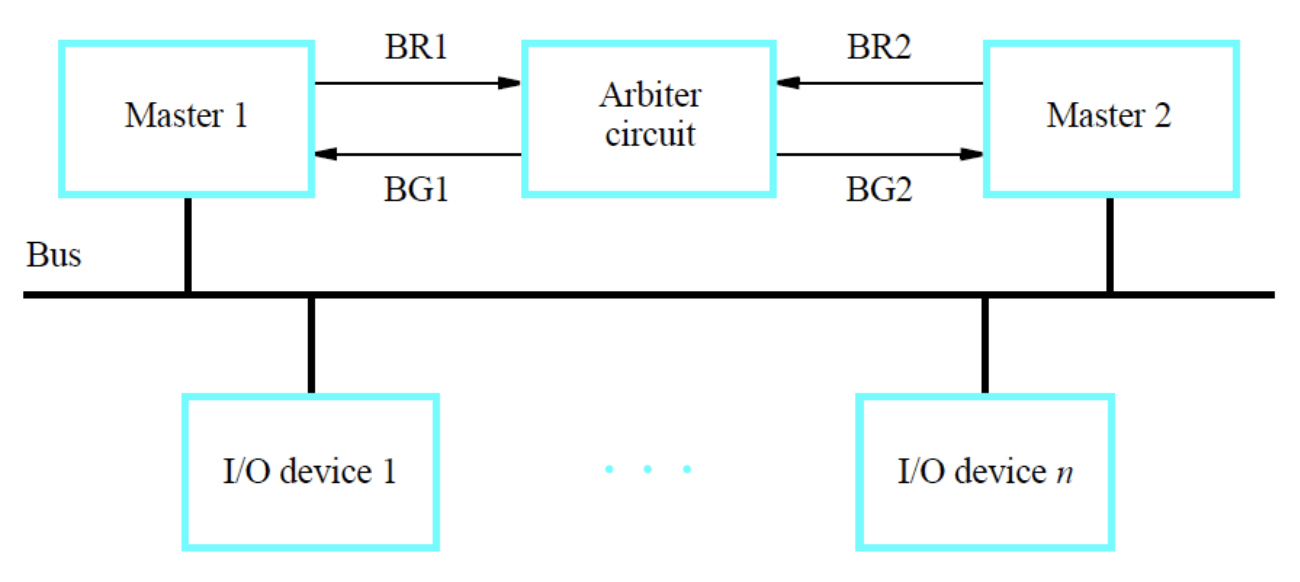


Key Points

- 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

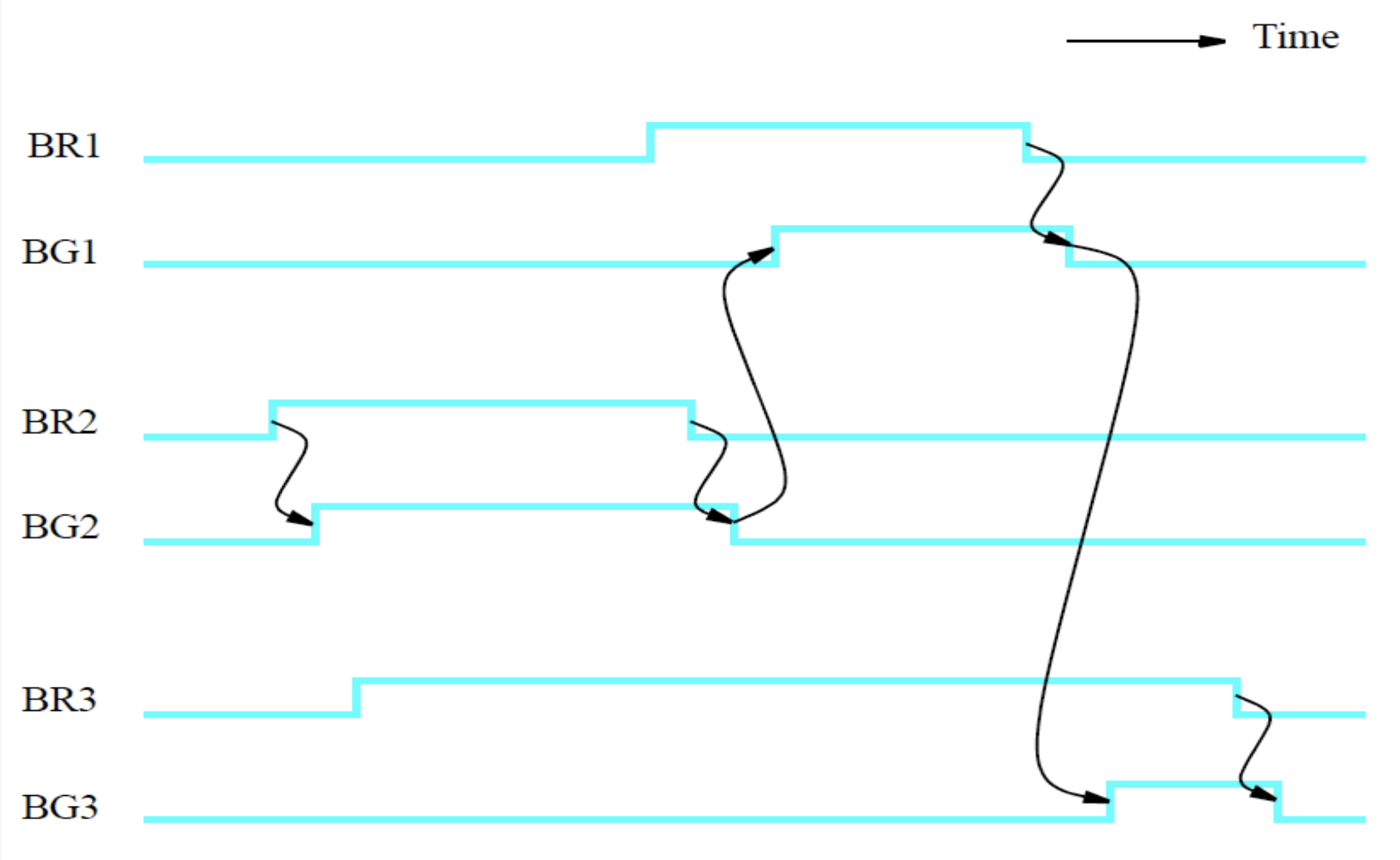
Bus Arbitration

- 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)

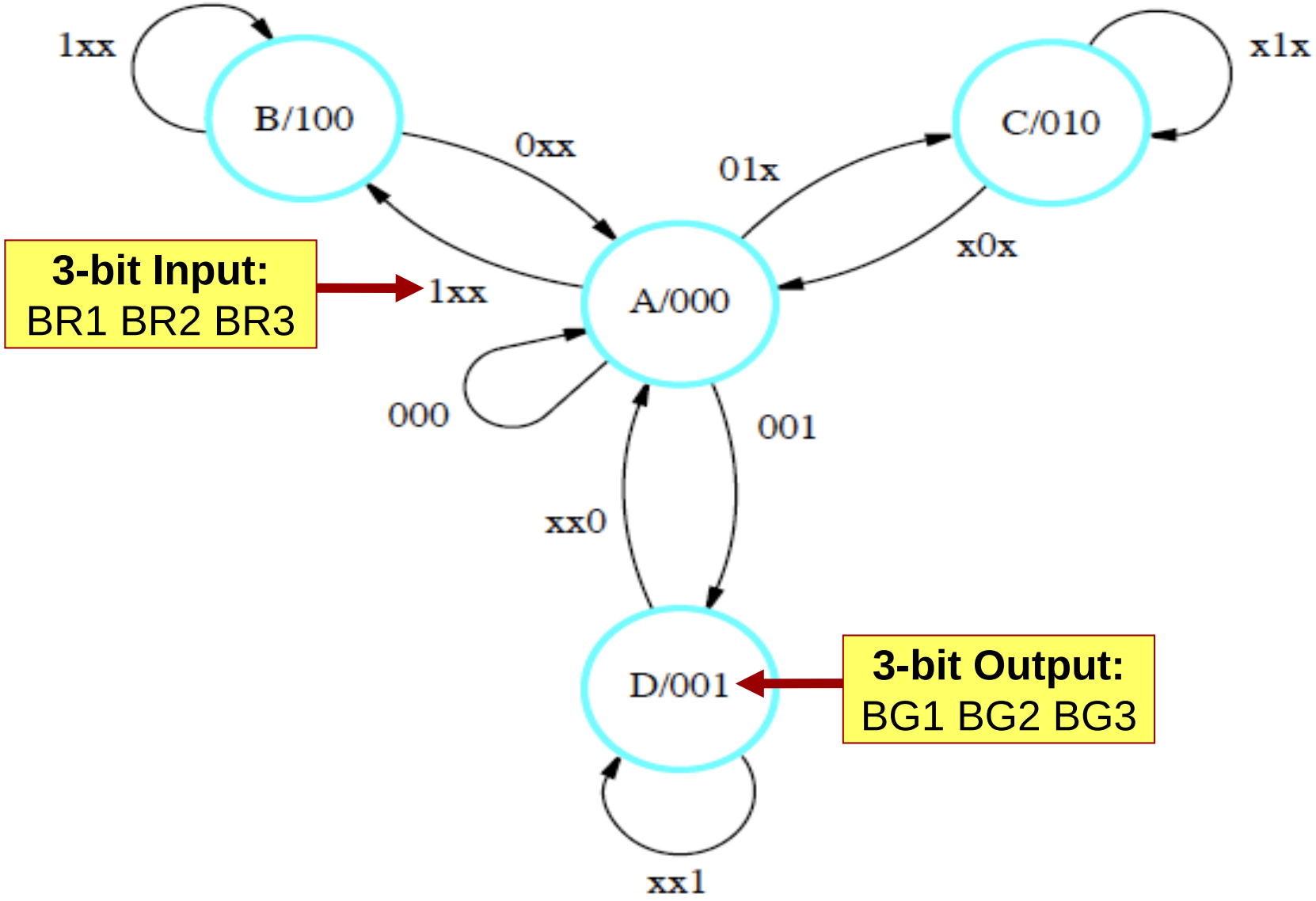


Centralized Arbitration I

Priorities: BR1 > BR2 > BR3

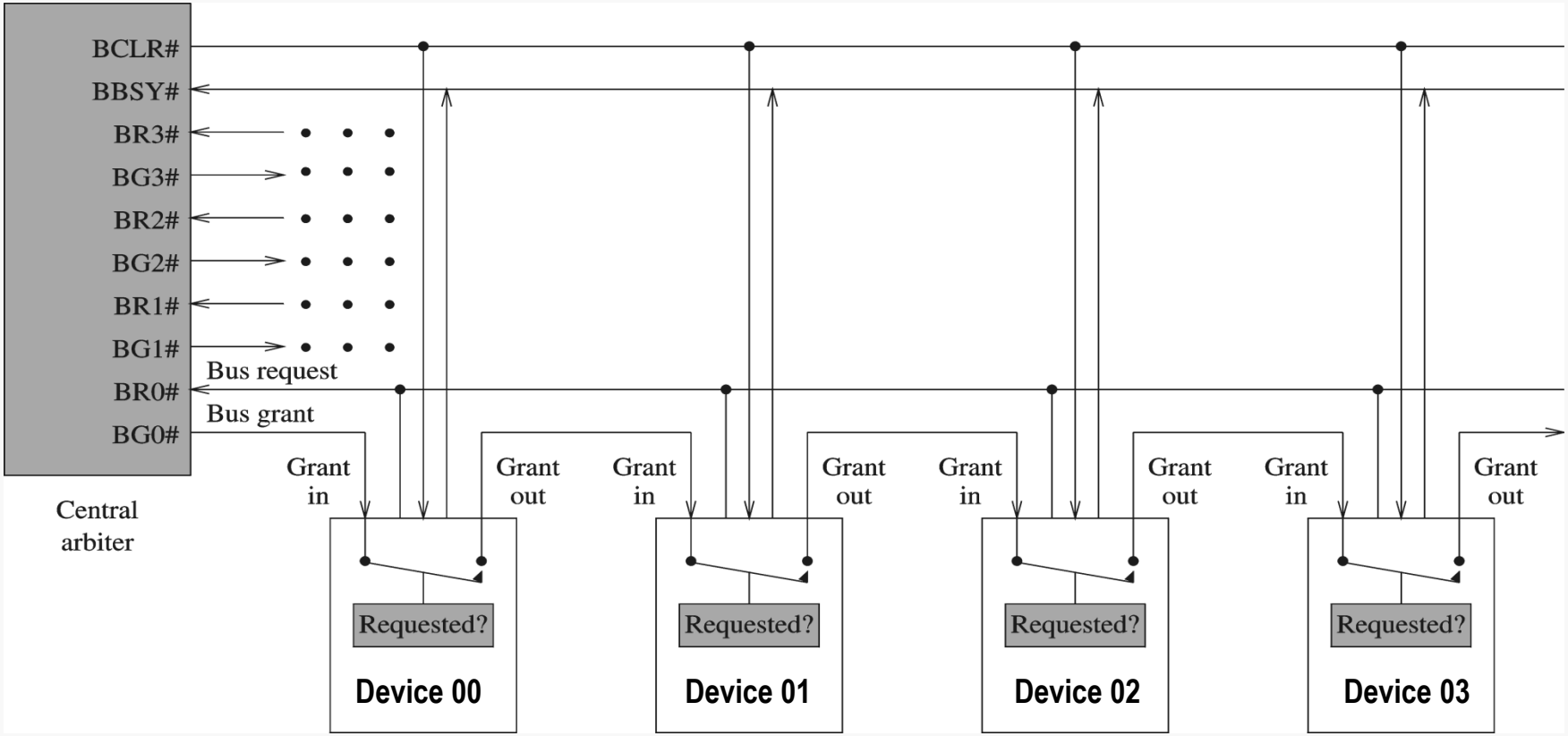


Centralized Arbitration II

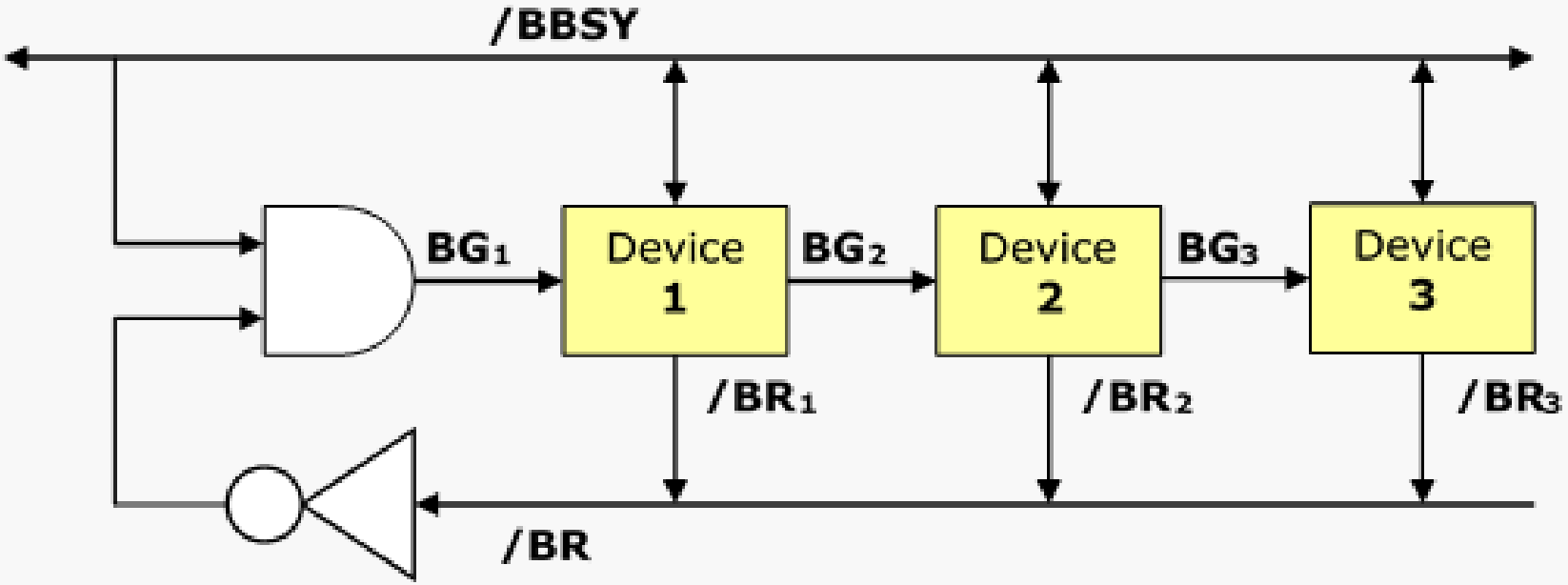


Hybrid Daisy Chain

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

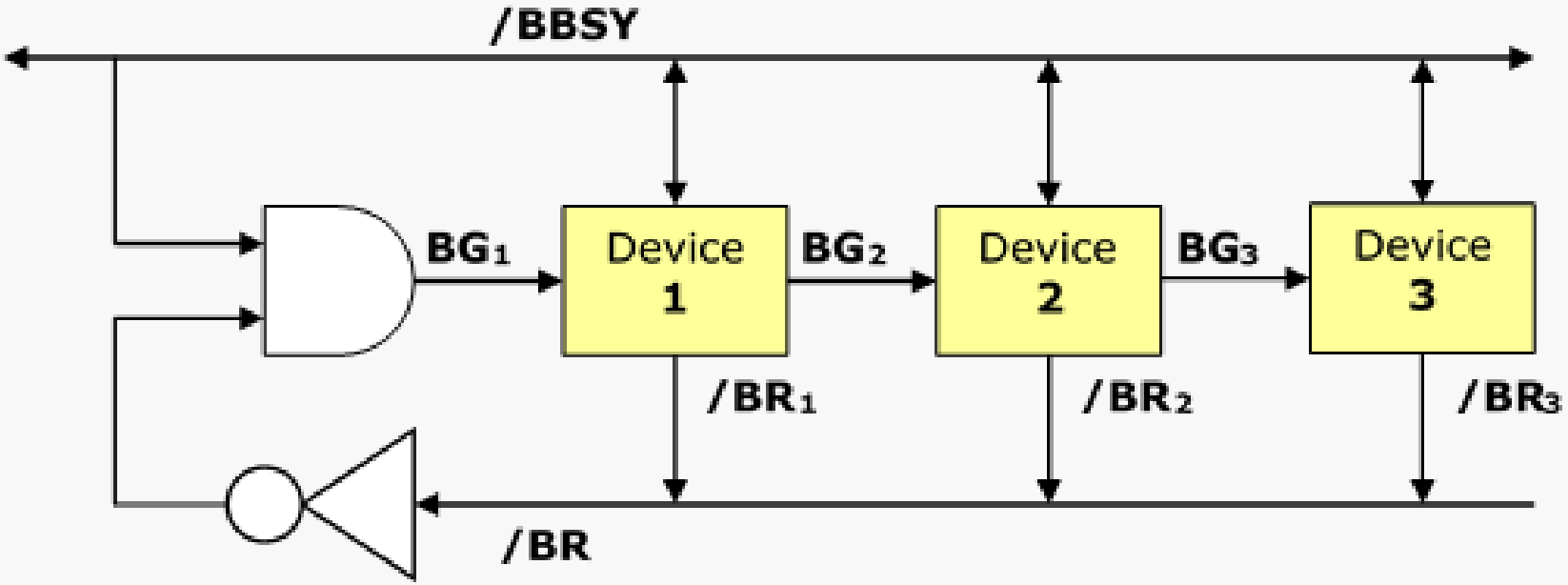


Daisy Chain Example: Setup



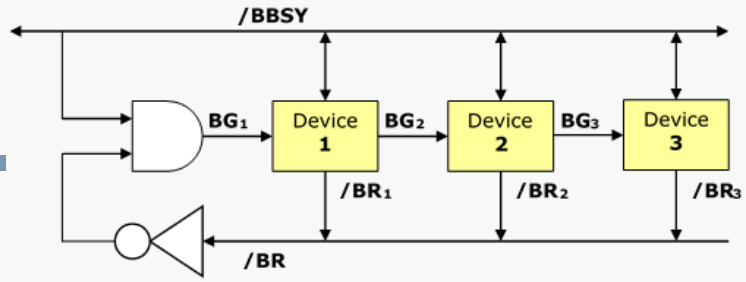
- ✓ The input-to-output signal propagation delays are equal to **d** for all three devices, the inverter, and the **AND** gate
- ✓ Any of the three devices will need to use the granted bus for only **3d** time units

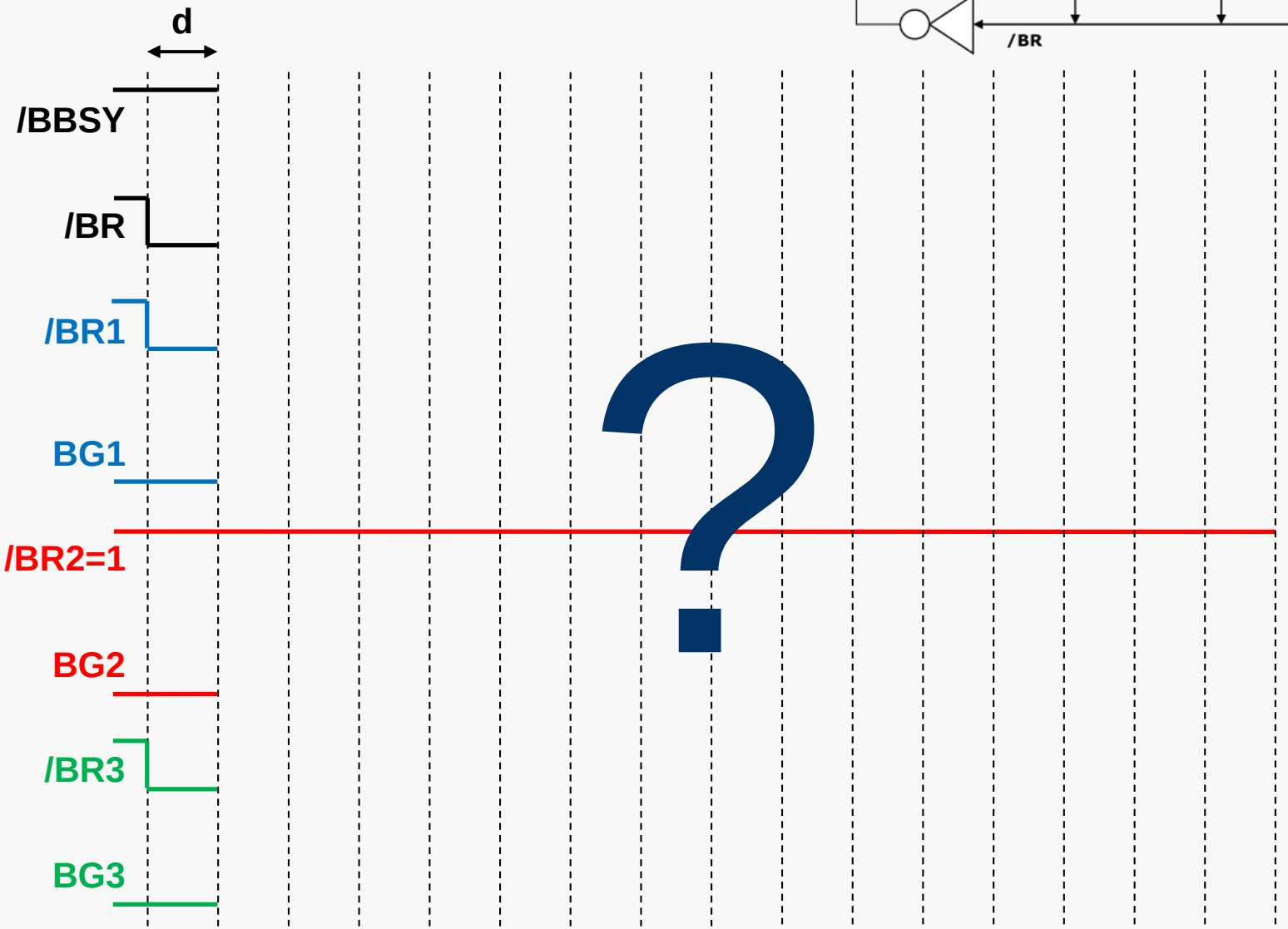
Daisy Chain Example: Setup



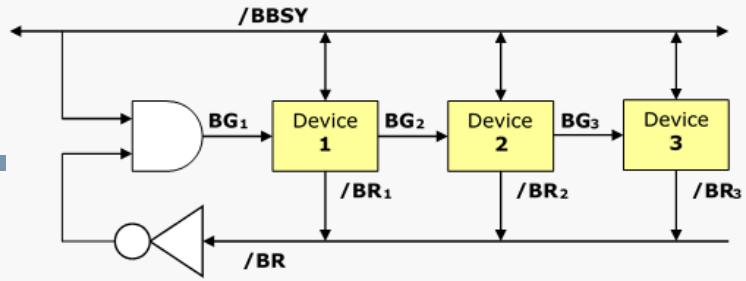
- ✓ Device **x** is able to start using the bus (making **/BRx = 1** and **/BBSY = 0**) only when it receives a **0-1** transition on its bus-grant input **BGx** and detects that the bus is not currently busy (i.e., **/BBSY = 1**)
- ✓ Device **x** lets the bus-grant propagate through only when it is neither requesting nor using the bus

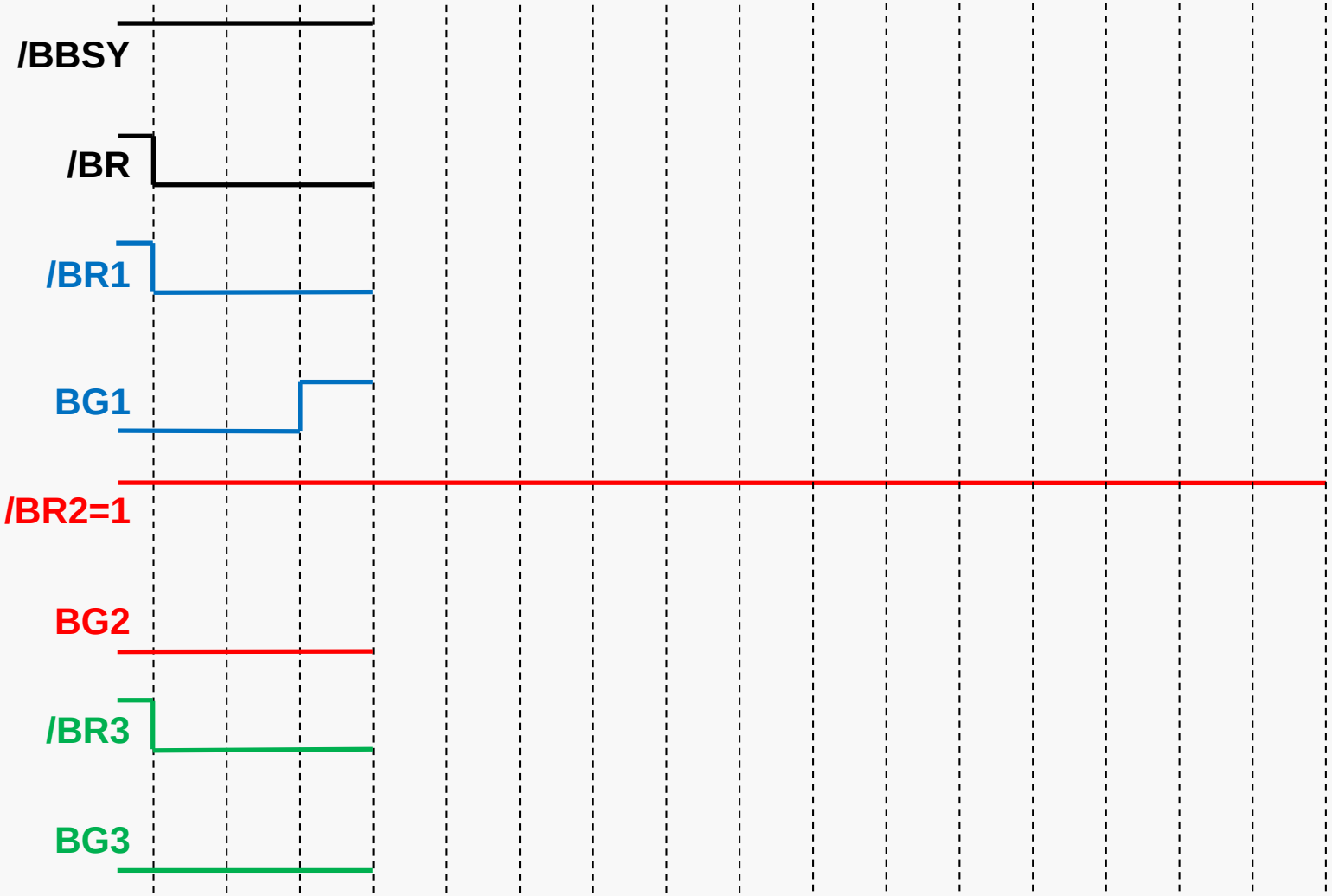
Waveforms


/BBSY
BG1
Device 1
BG2
Device 2
BG3
Device 3
/BR1
/BR2
/BR3
/BR

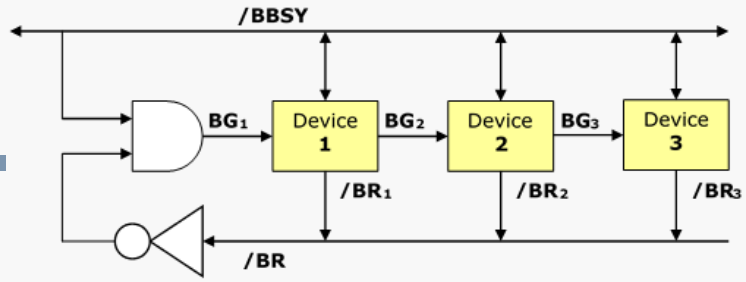

d
/BBSY
/BR
/BR1
BG1
/BR2=1
BG2
/BR3
BG3
?

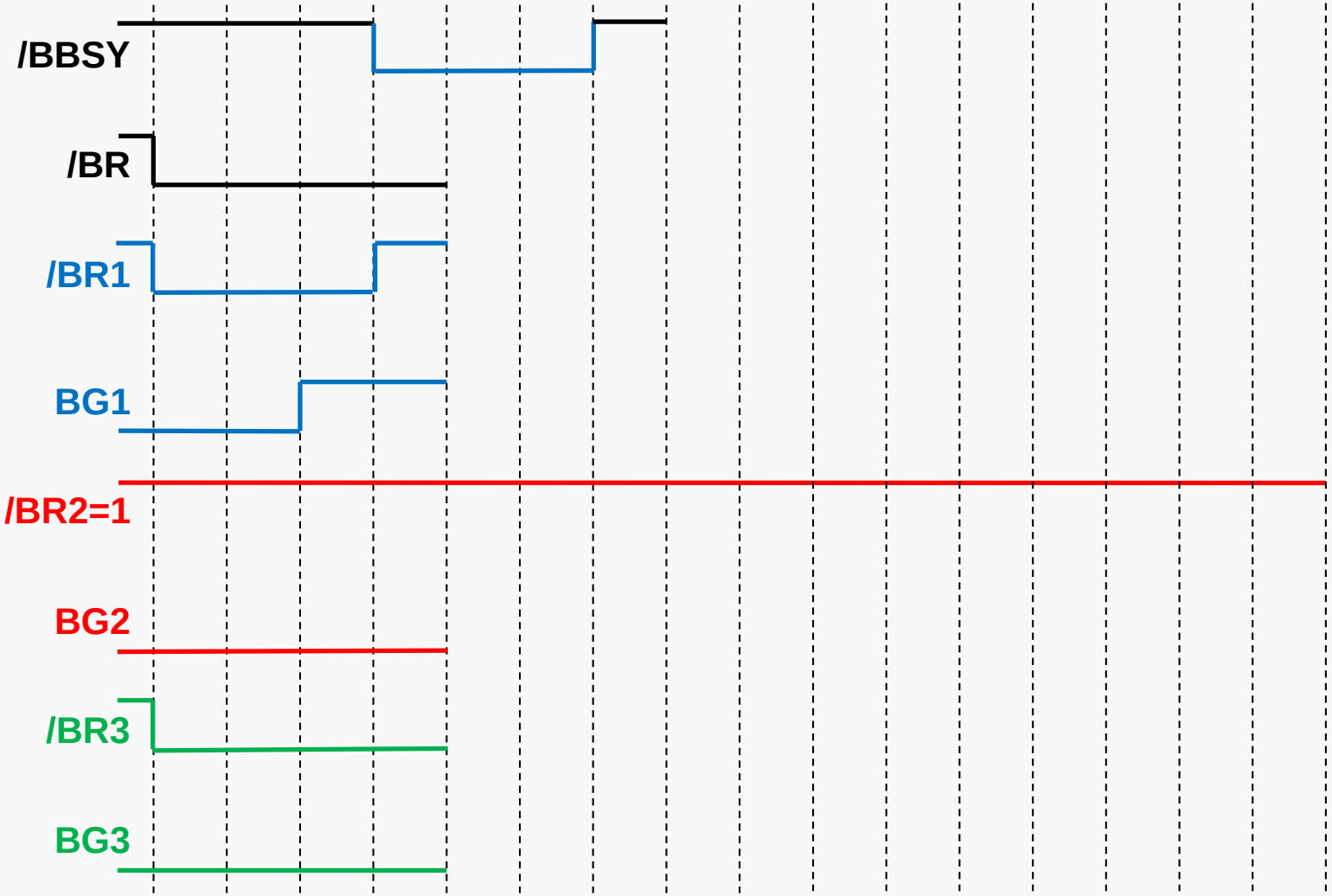
Waveforms


/BBSY
BG1
Device 1
BG2
Device 2
BG3
Device 3
/BR1
/BR2
/BR3
/BR

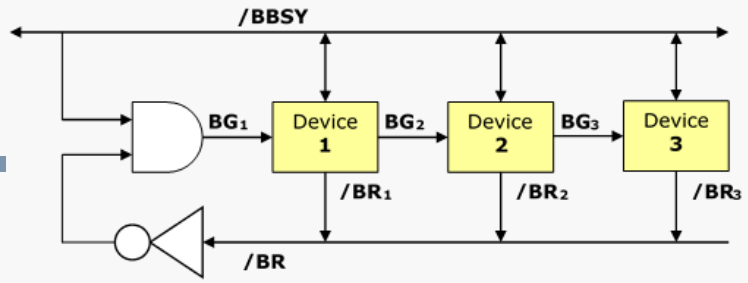

/BBSY
/BR
/BR1
BG1
/BR2=1
BG2
/BR3
BG3

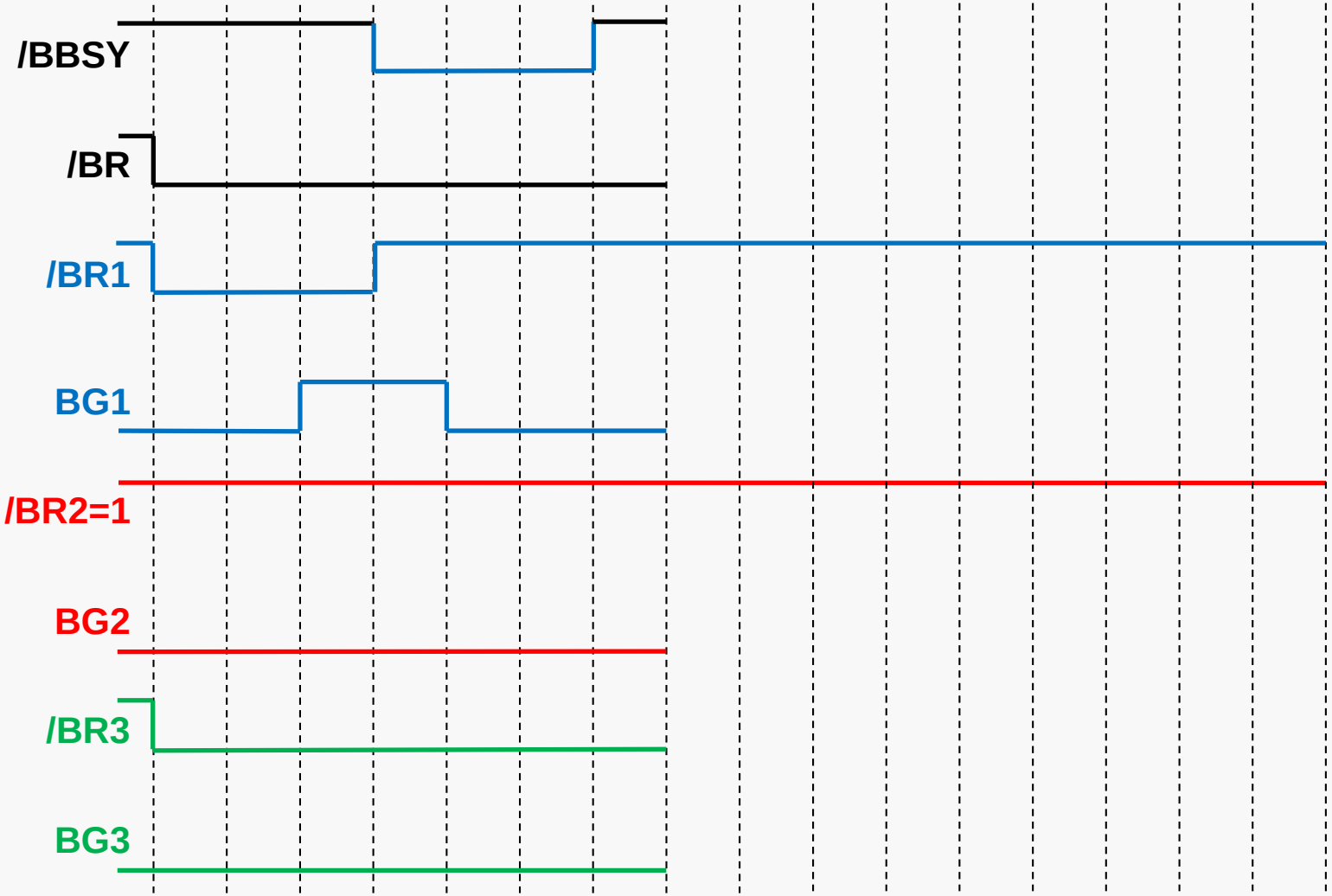
Waveforms


/BBSY
BG1
Device 1
BG2
Device 2
BG3
Device 3
/BR1
/BR2
/BR3
/BR

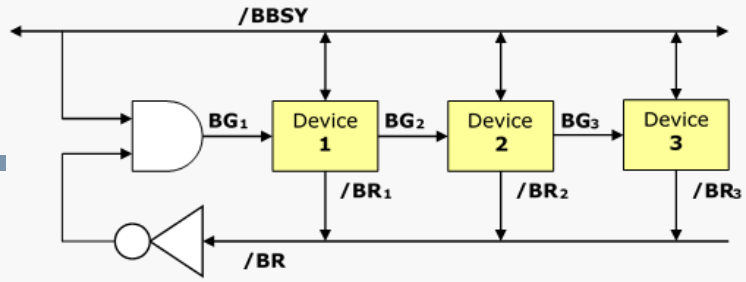

/BBSY
/BR
/BR1
BG1
/BR2=1
BG2
/BR3
BG3

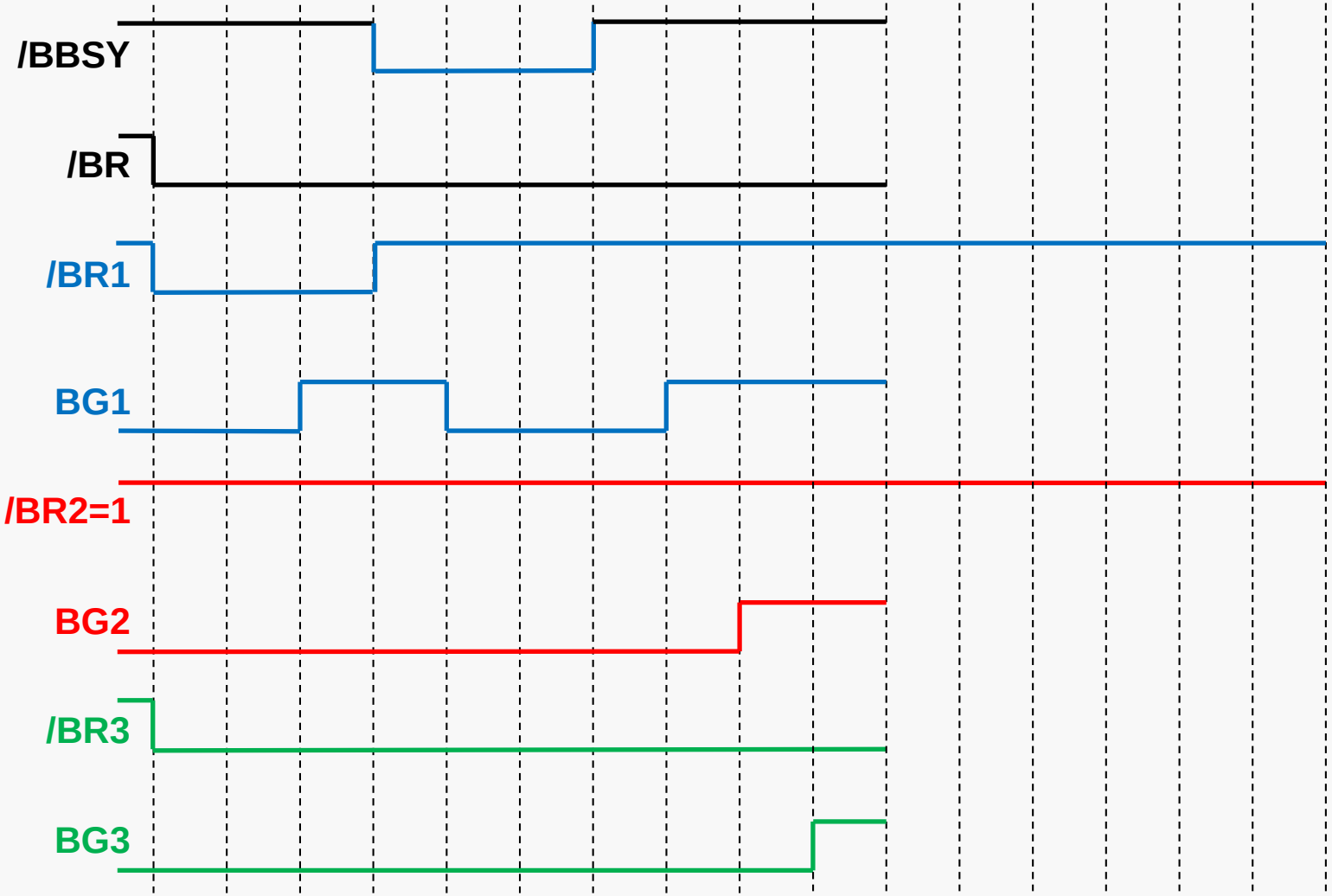
Waveforms


/BBSY
BG1
Device 1
BG2
Device 2
BG3
Device 3
/BR1
/BR2
/BR3
/BR

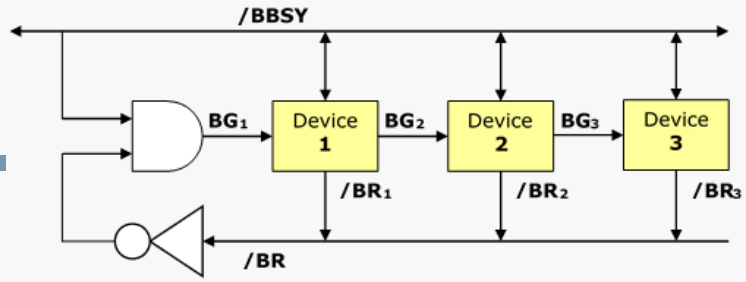

/BBSY
/BR
/BR1
BG1
/BR2=1
BG2
/BR3
BG3

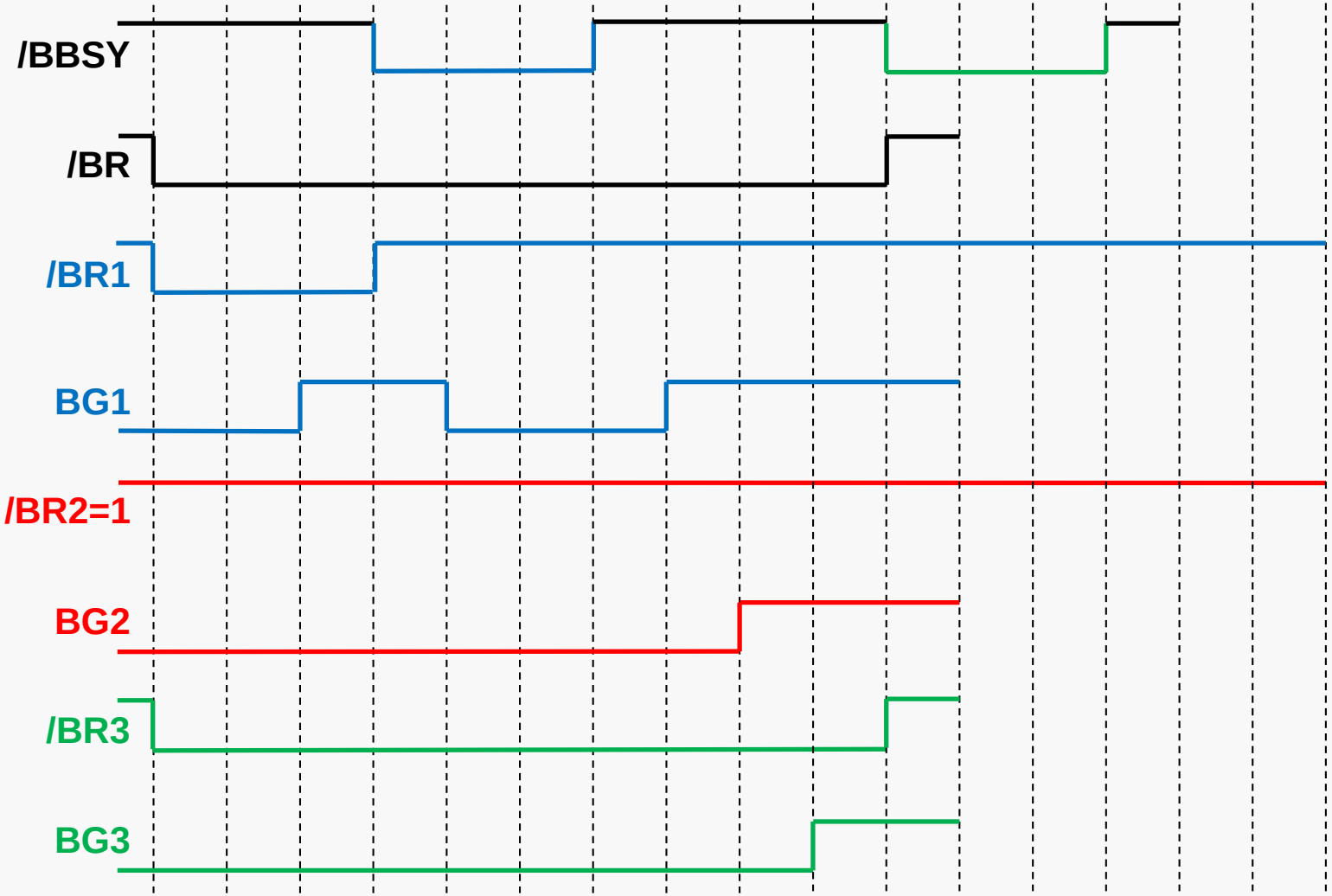
Waveforms


/BBSY
BG1
Device 1
BG2
Device 2
BG3
Device 3
/BR1
/BR2
/BR3
/BR

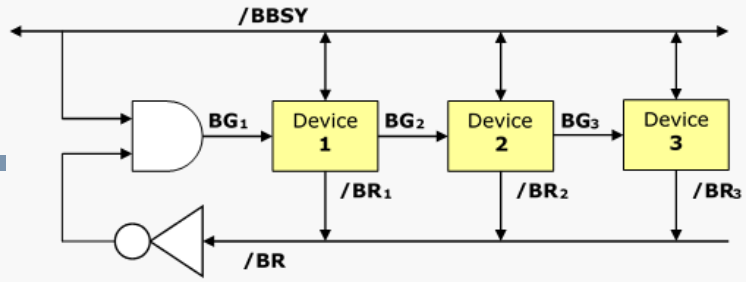

/BBSY
/BR
/BR1
BG1
/BR2=1
BG2
/BR3
BG3

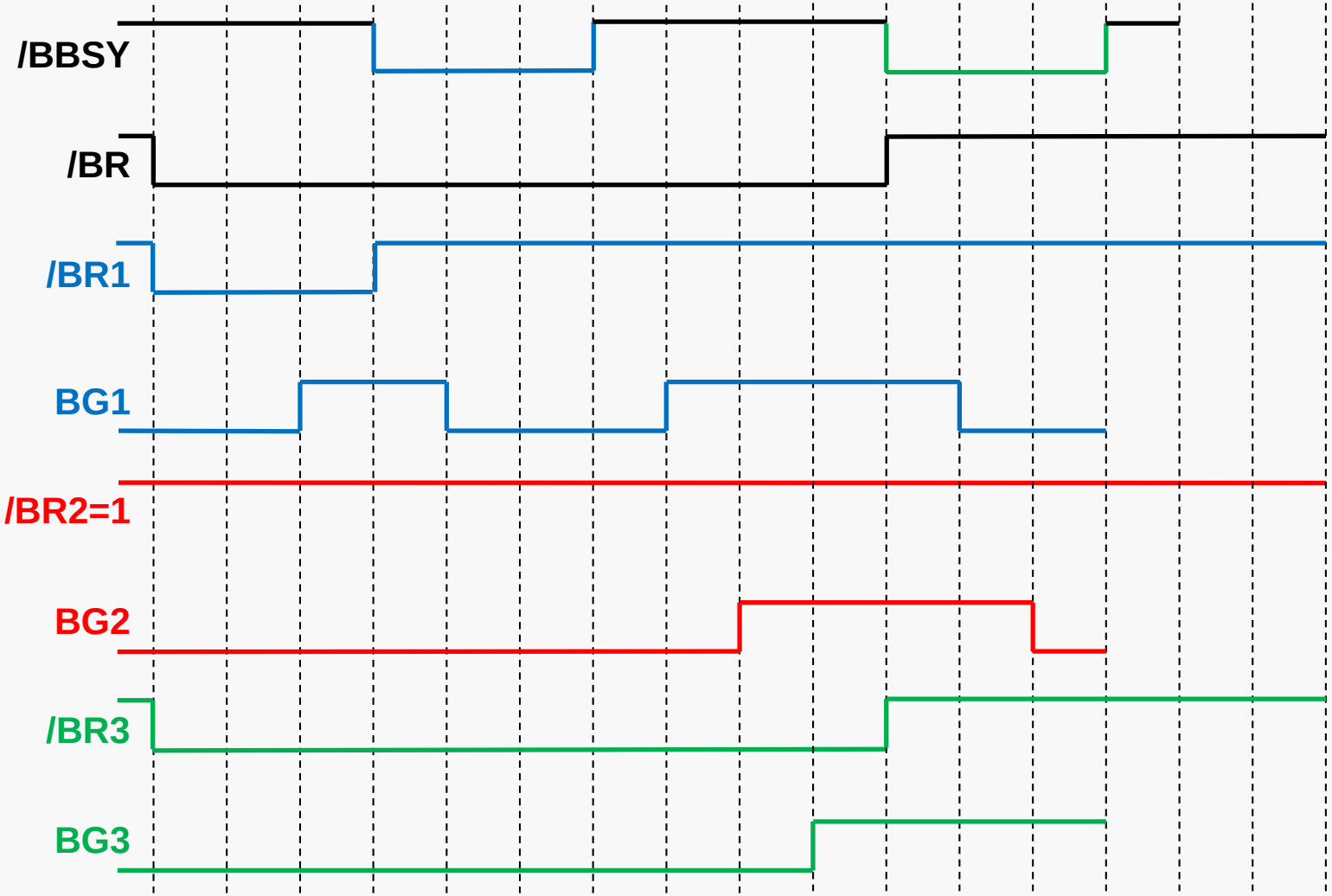
Waveforms


/BBSY
BG1
Device 1
BG2
Device 2
BG3
Device 3
/BR1
/BR2
/BR3
/BR

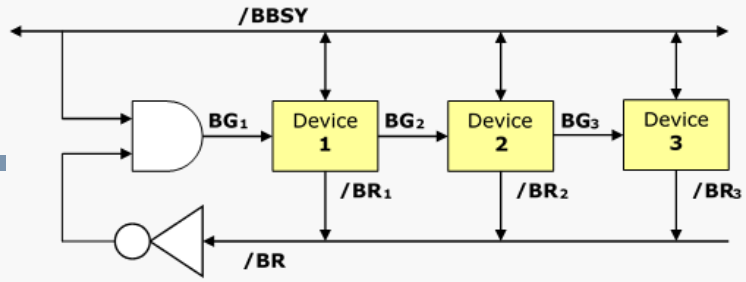

/BBSY
/BR
/BR1
BG1
/BR2=1
BG2
/BR3
BG3

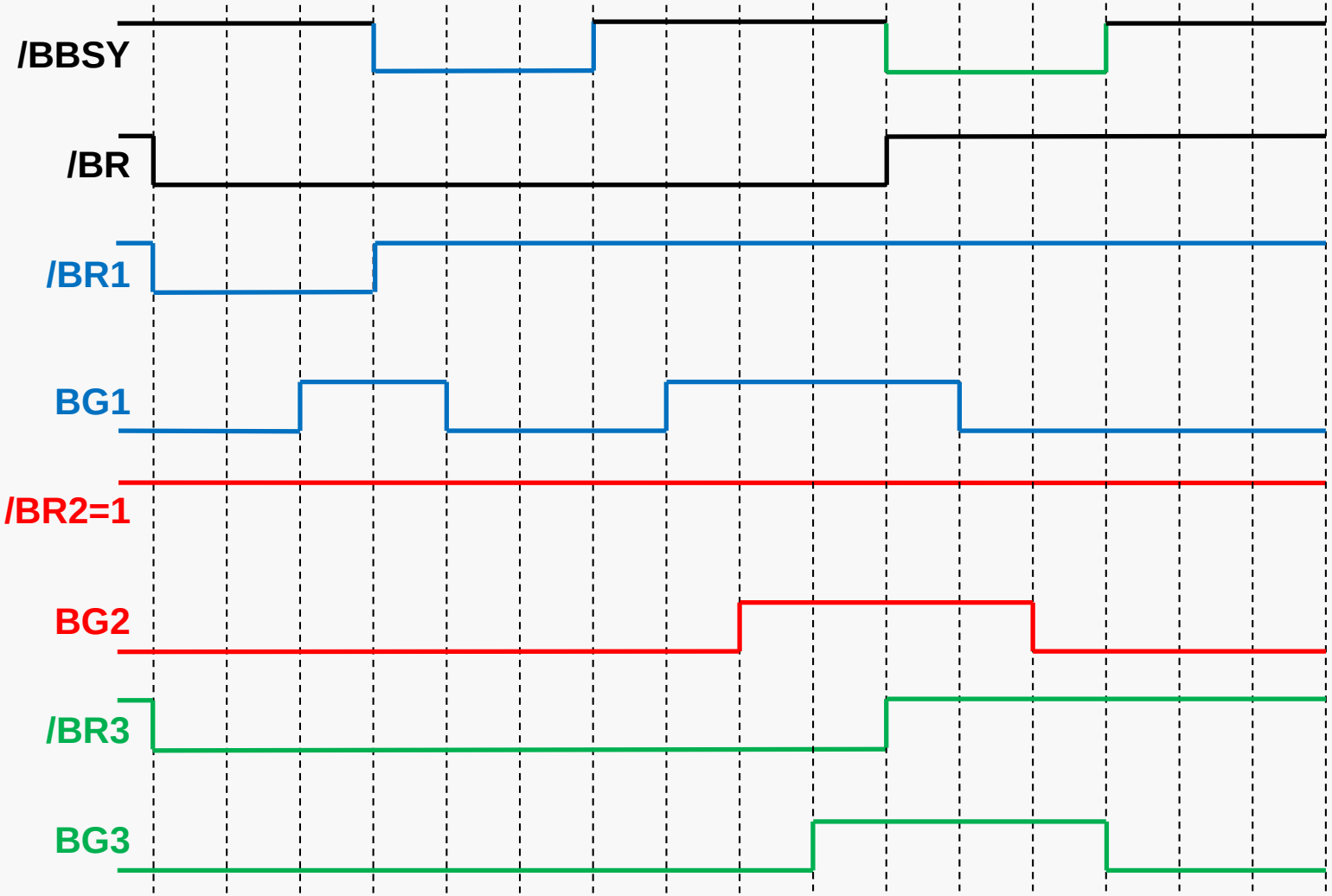
Waveforms


/BBSY
BG1
Device 1
BG2
Device 2
BG3
Device 3
/BR1
/BR2
/BR3
/BR

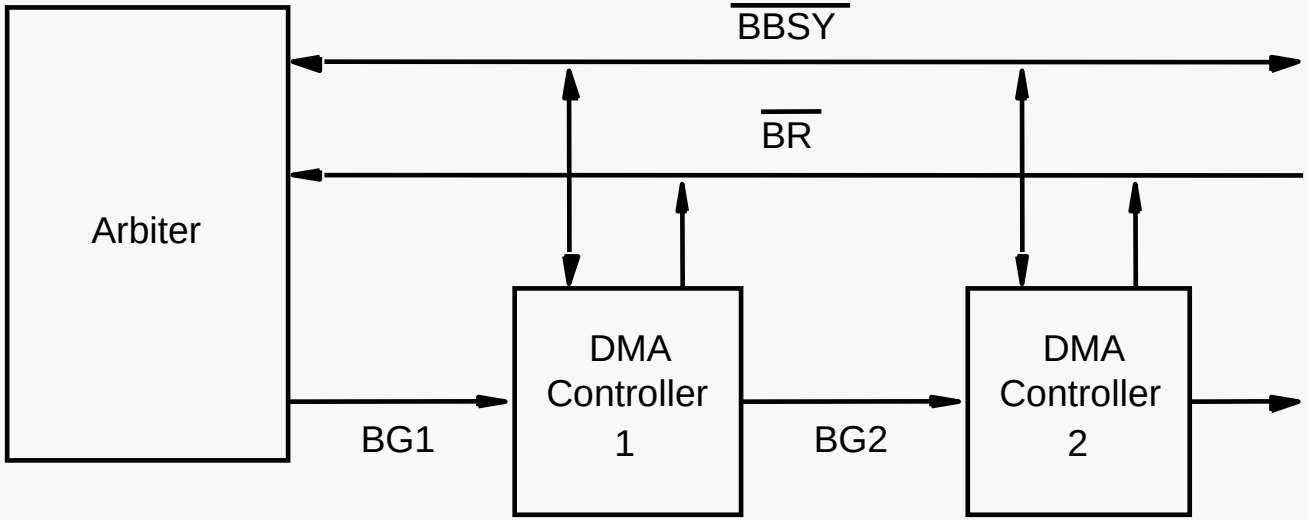

/BBSY
/BR
/BR1
BG1
/BR2=1
BG2
/BR3
BG3

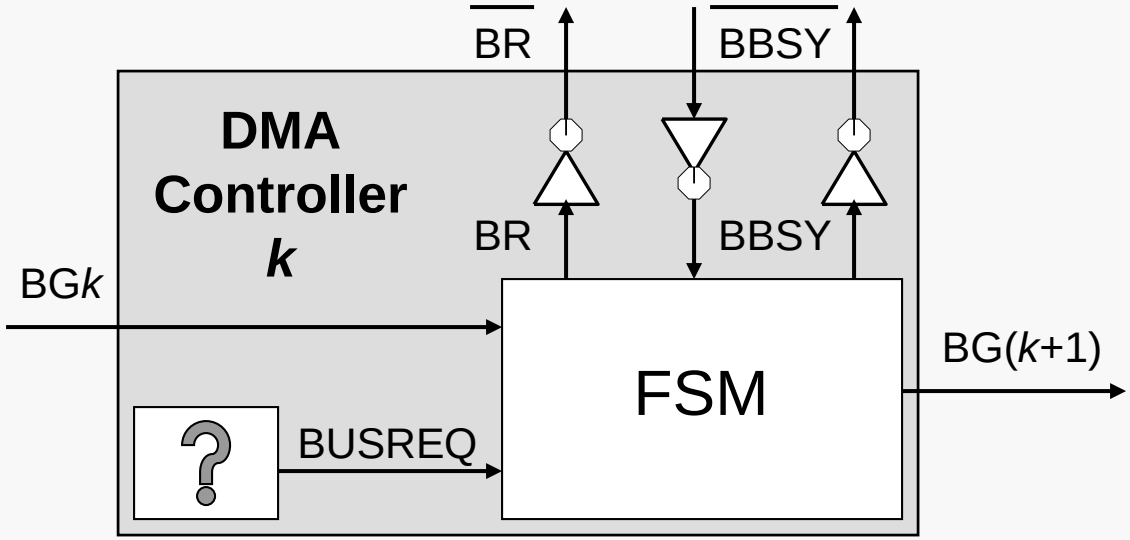
Waveforms


/BBSY
BG1
Device 1
BG2
Device 2
BG3
Device 3
/BR1
/BR2
/BR3
/BR


/BBSY
/BR
/BR1
BG1
/BR2=1
BG2
/BR3
BG3

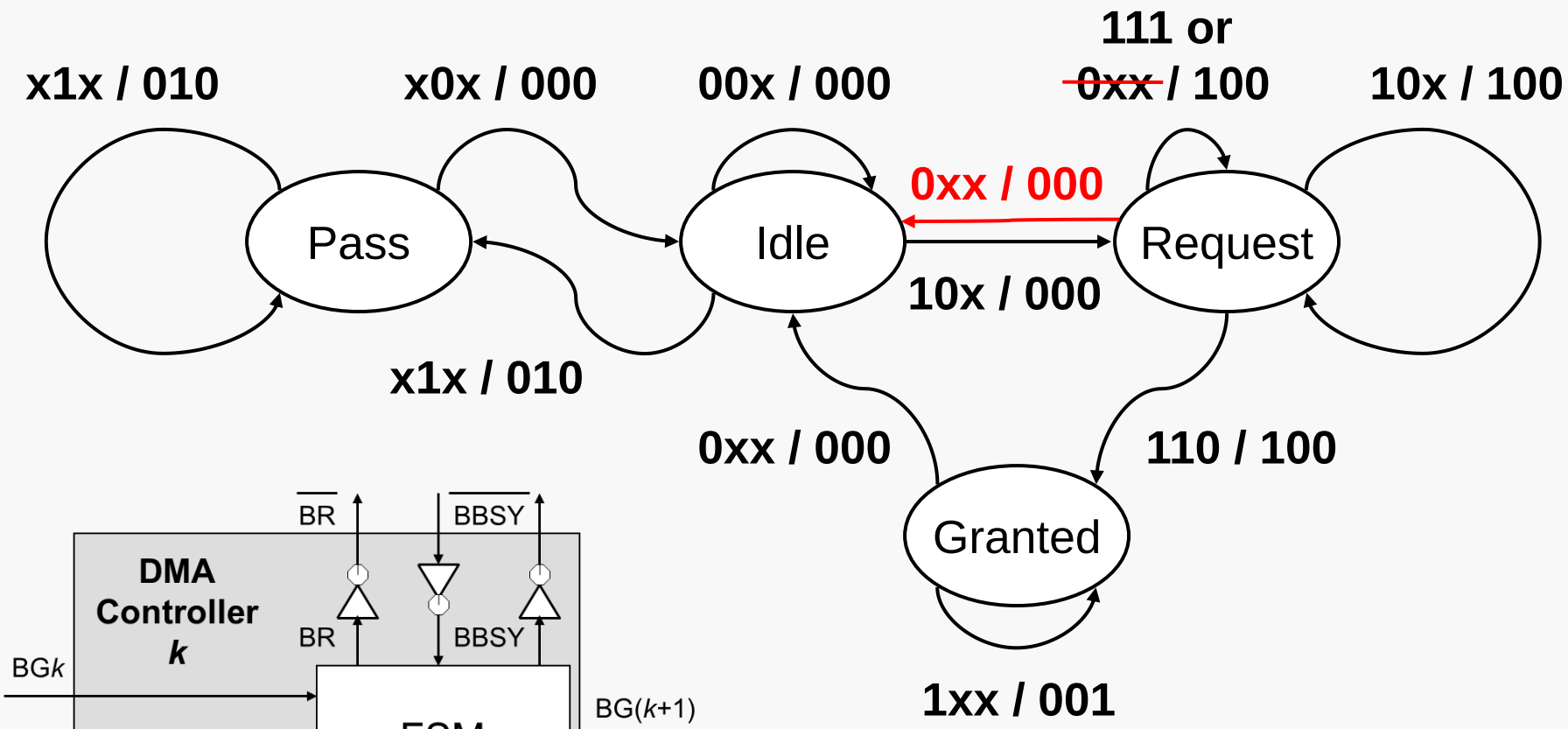
Daisy Chain of DMA Controllers



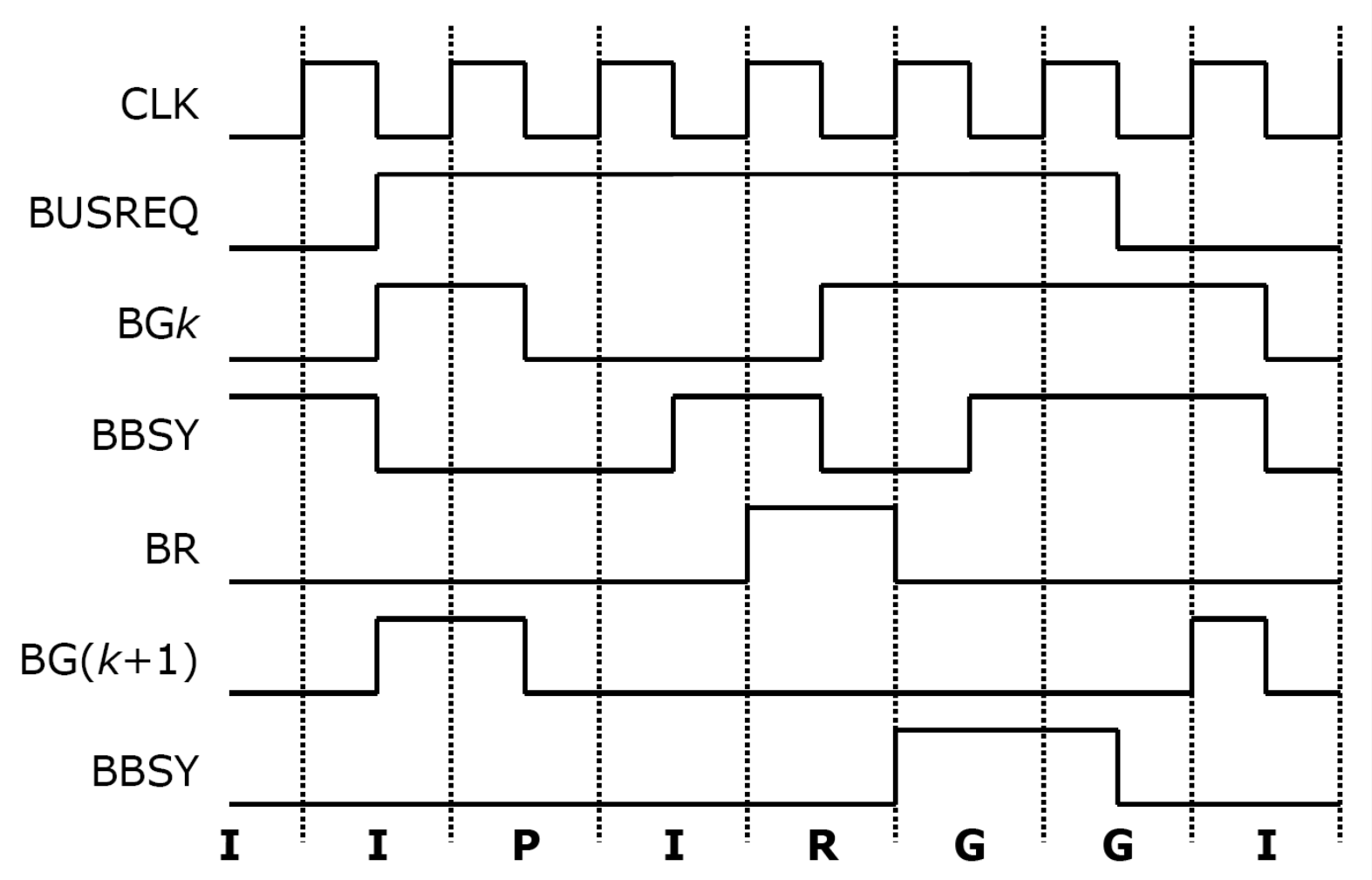


FSM State Diagram Example

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



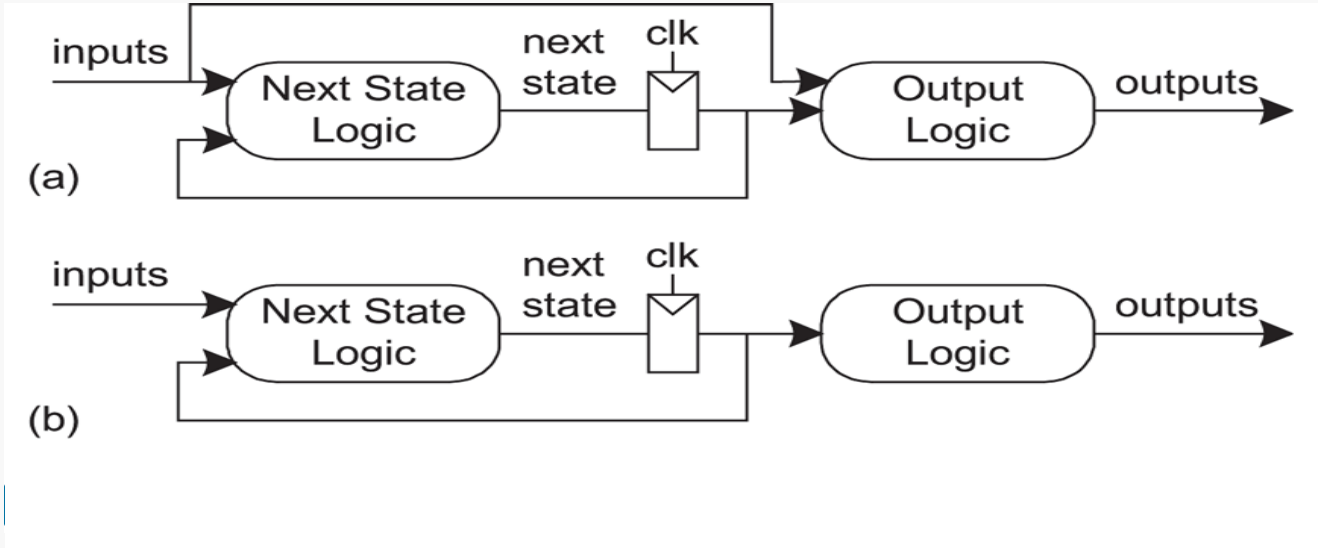
FSM Waveform Example



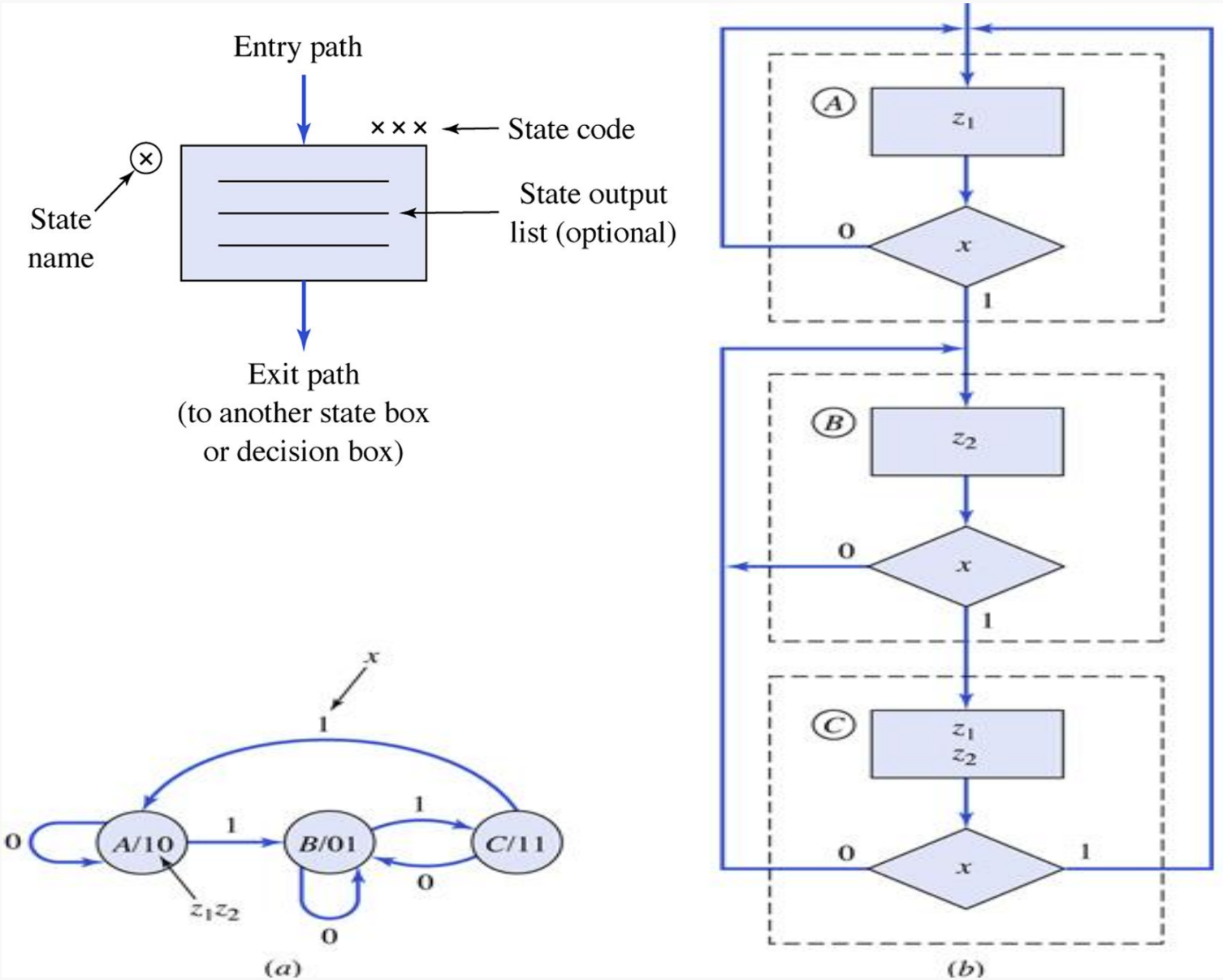
I = Idle **P = Pass** **R = Request** **G = Granted**

FSM Design

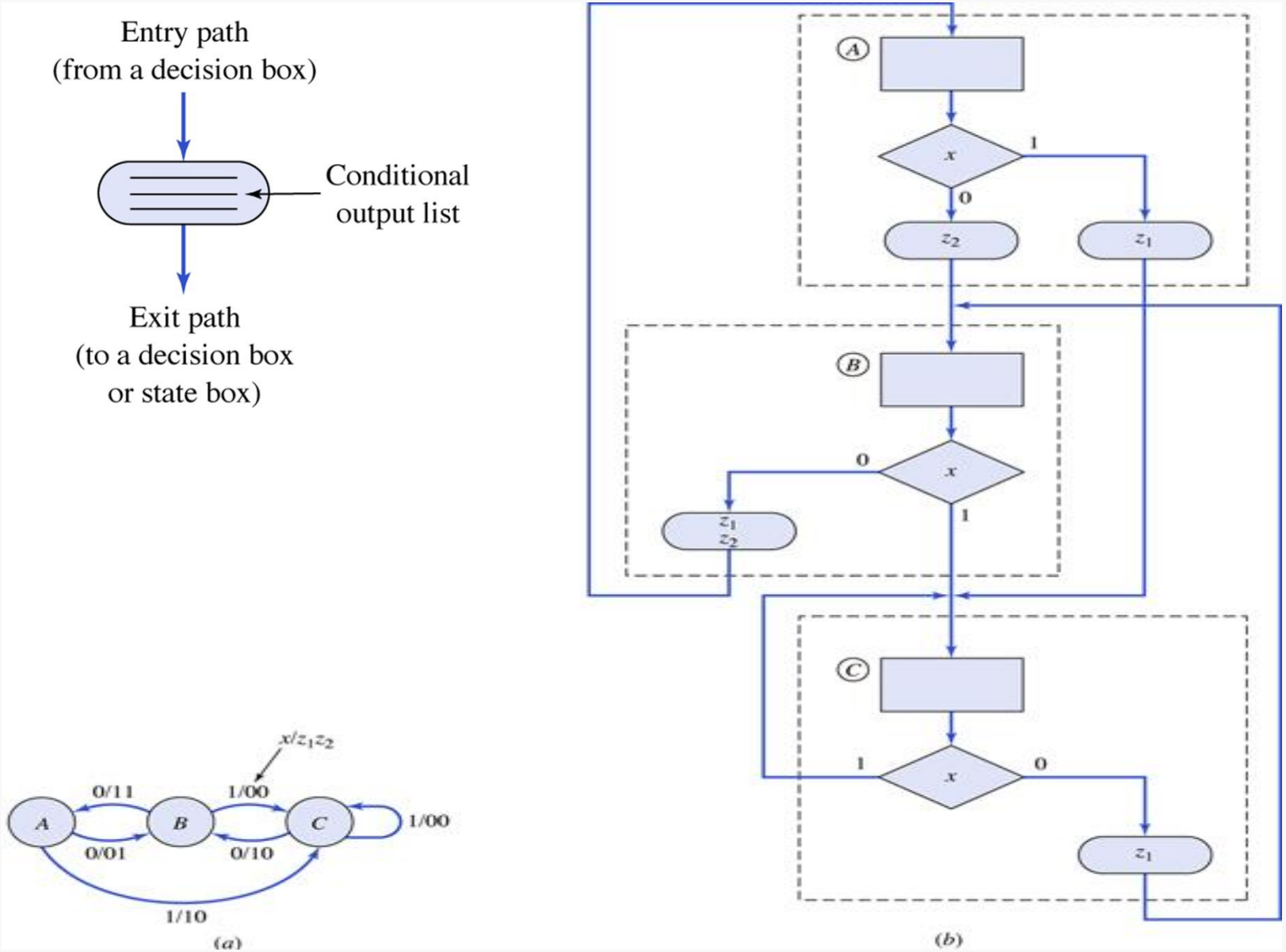
- FSM design procedure
  - Determine state diagram (your **design**)
  - Encode states
  - Implement next-state logic and output logic
- Mealy (a) and Moore (b) FSMs
  - **Mealy**: output depends on input, one cycle faster
  - **Moore**: output does not depend on input, more reliable



Moore Chart Example



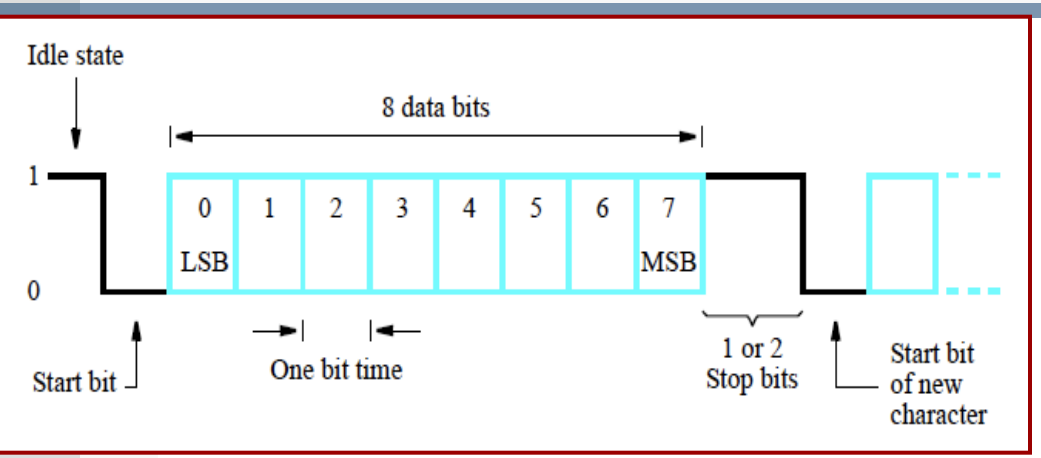
Mealy Chart Example



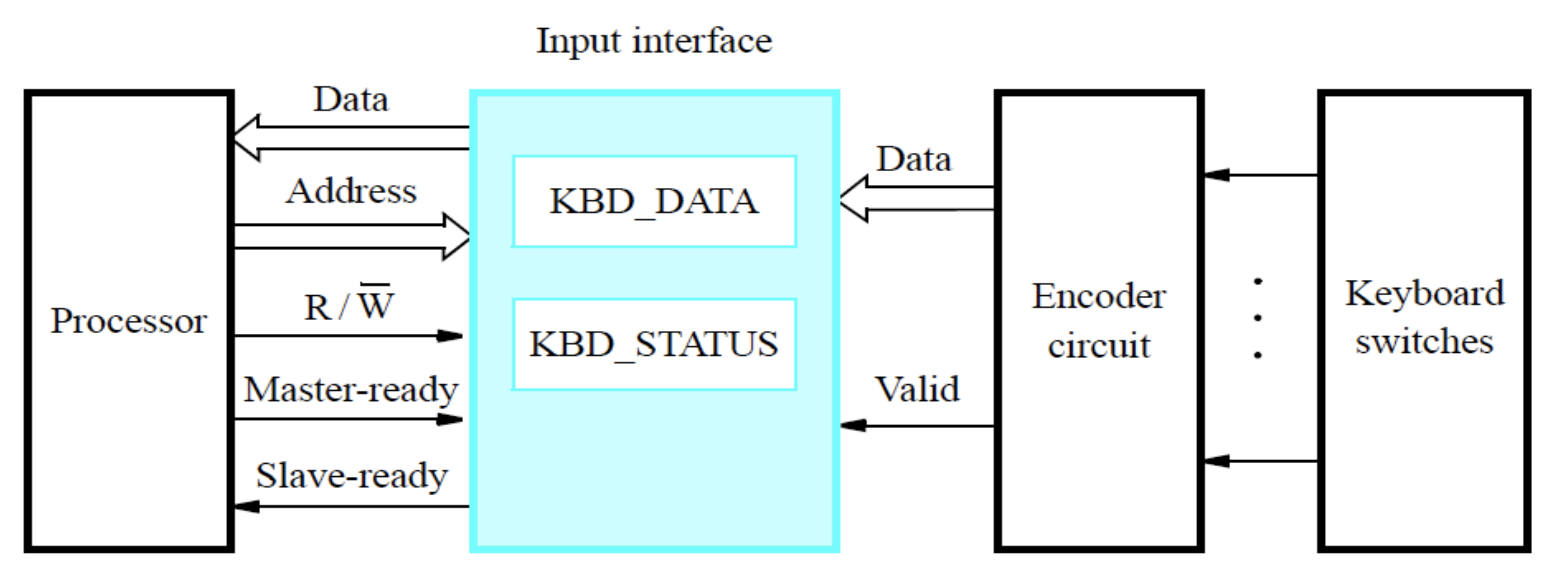
I/O Interface Circuits

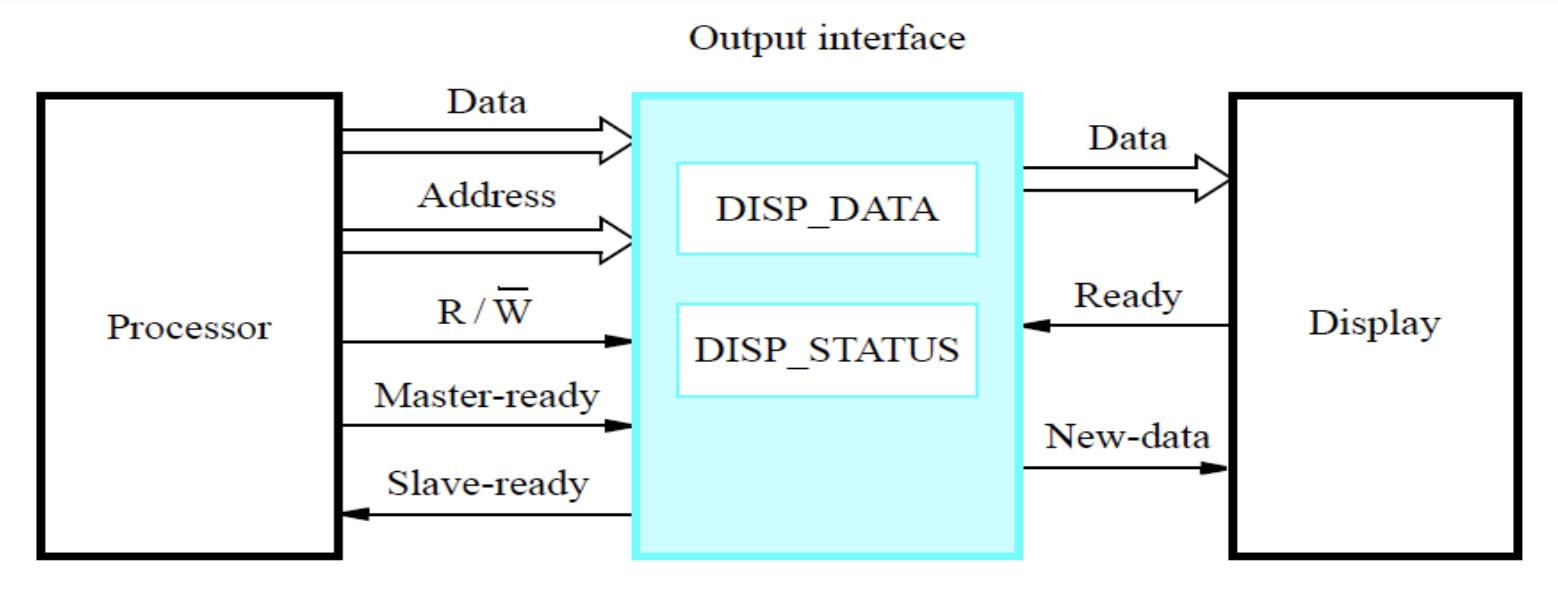
- Basic I/O interface functions:
  - Contains address decoding circuitry that detects when the I/O device is being addressed
  - Generates the appropriate signals required for bus control
  - Provides a buffer for temporary data storage
  - Contains status flags that can be accessed to determine whether 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

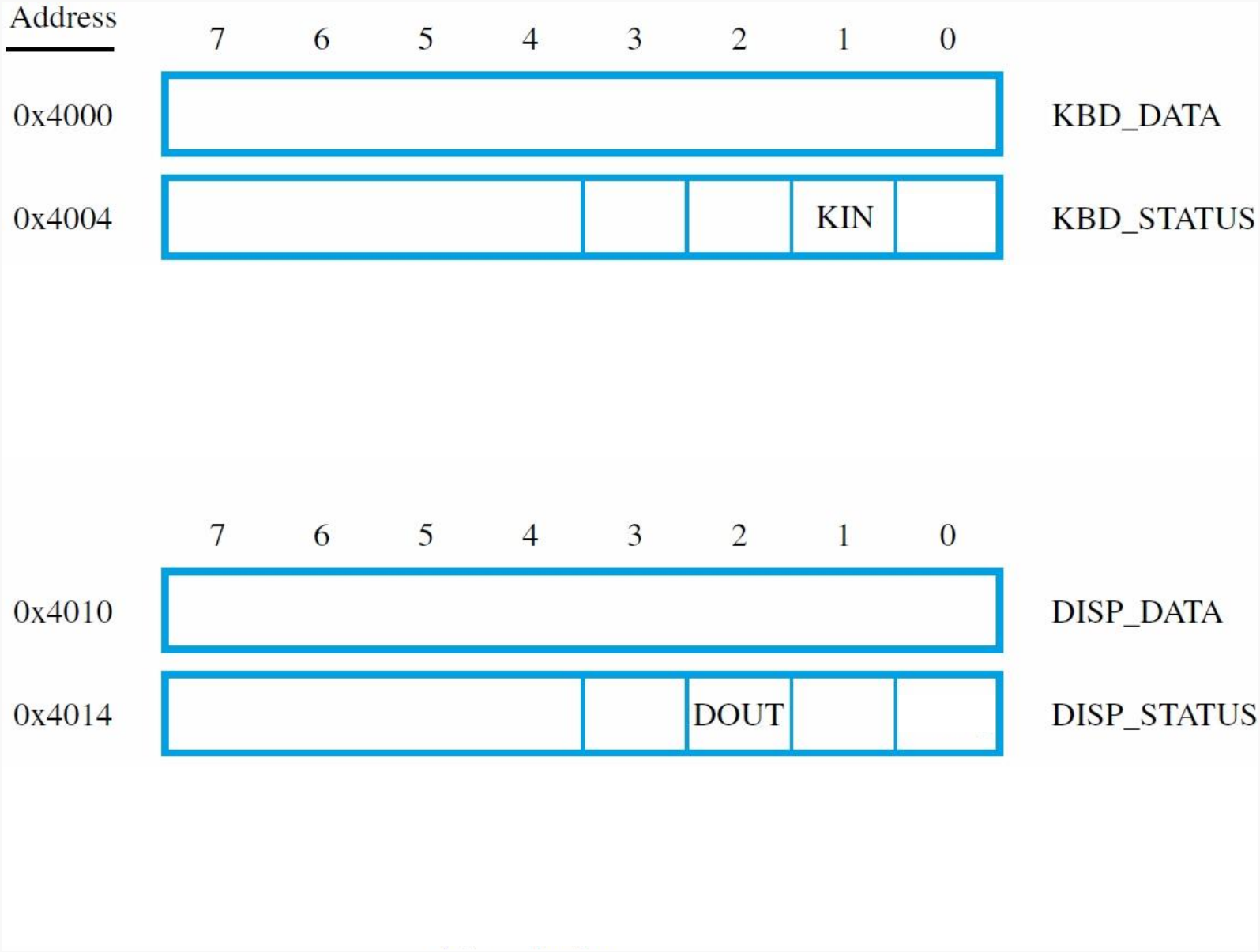


Example: Parallel Ports

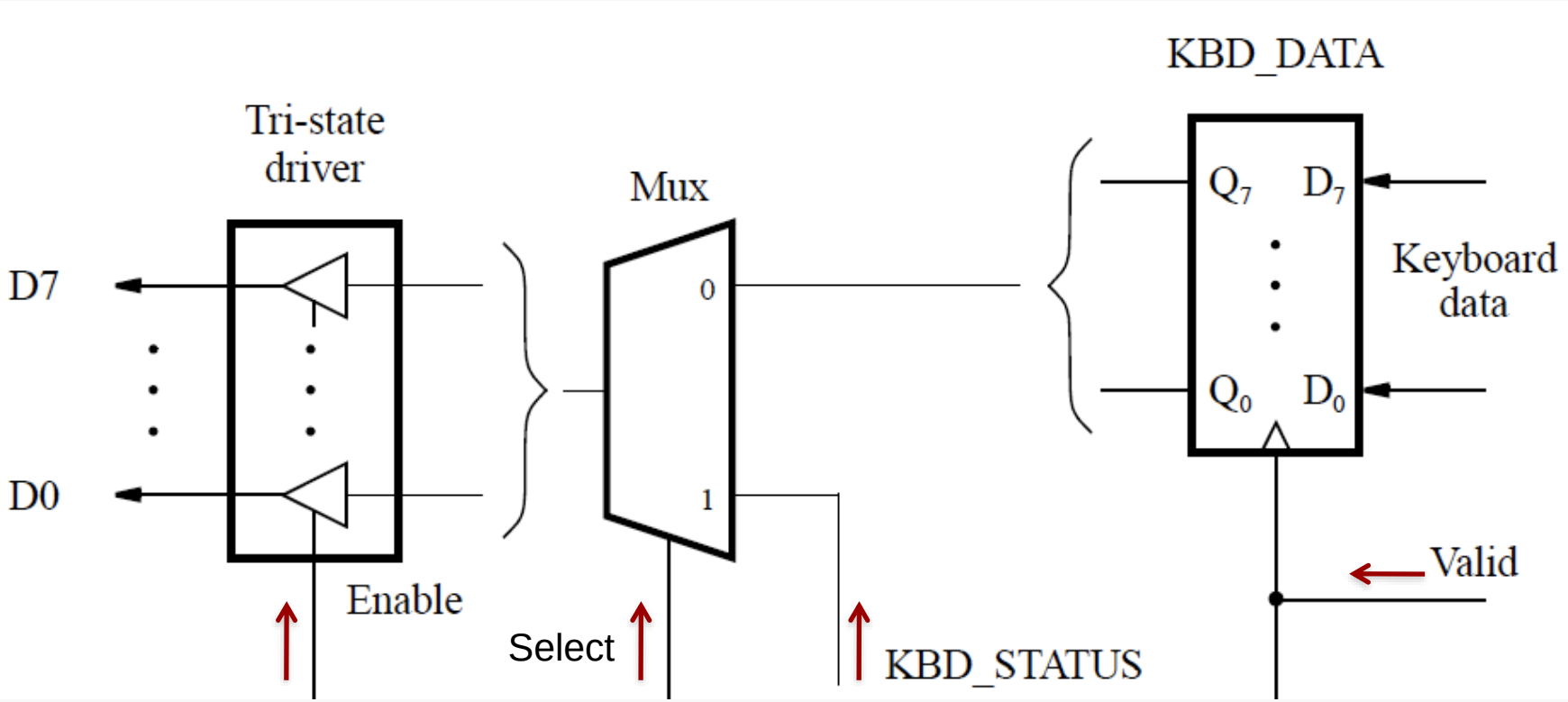




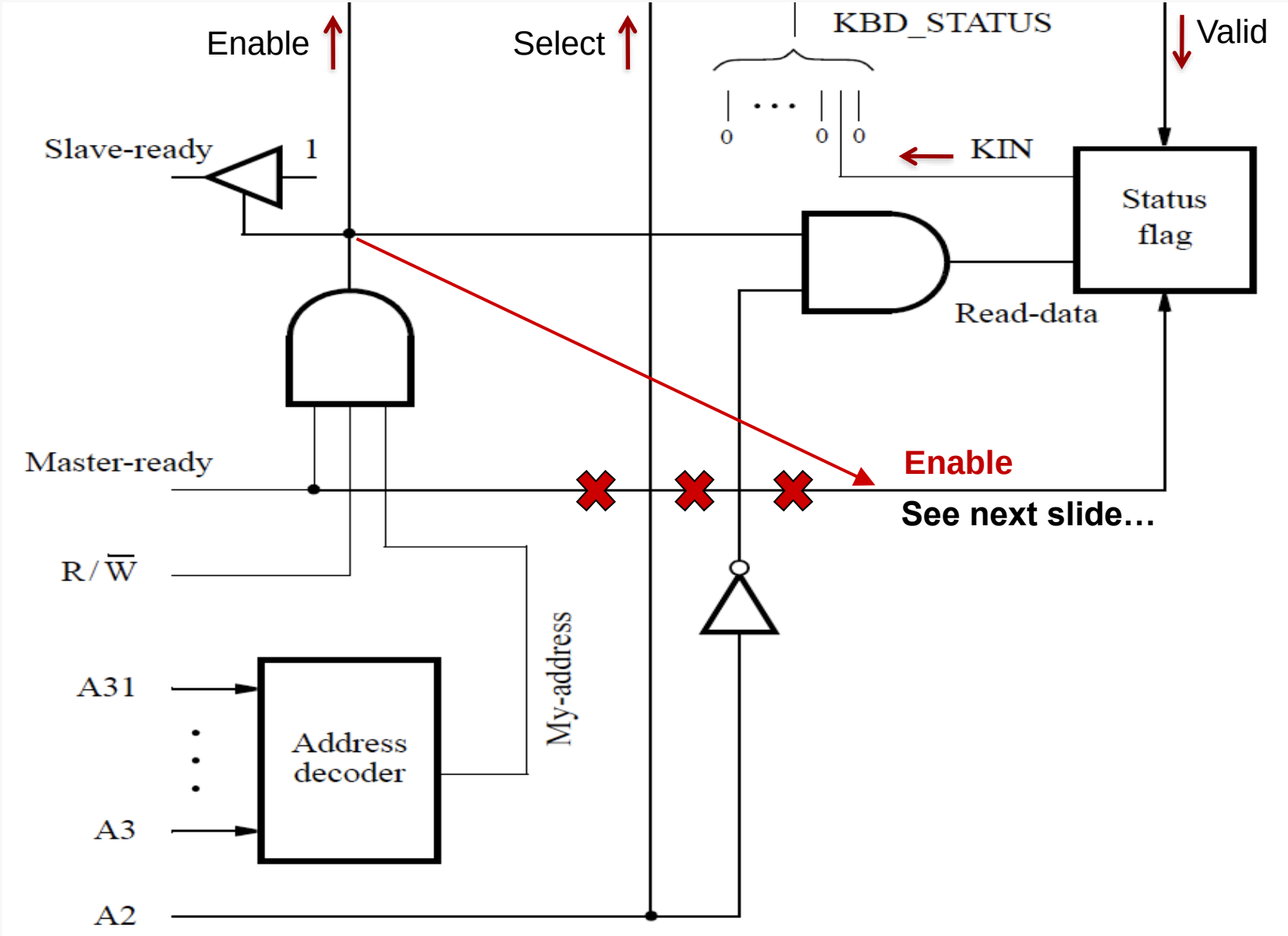
Example: Interface Registers



Parallel Input Interface I

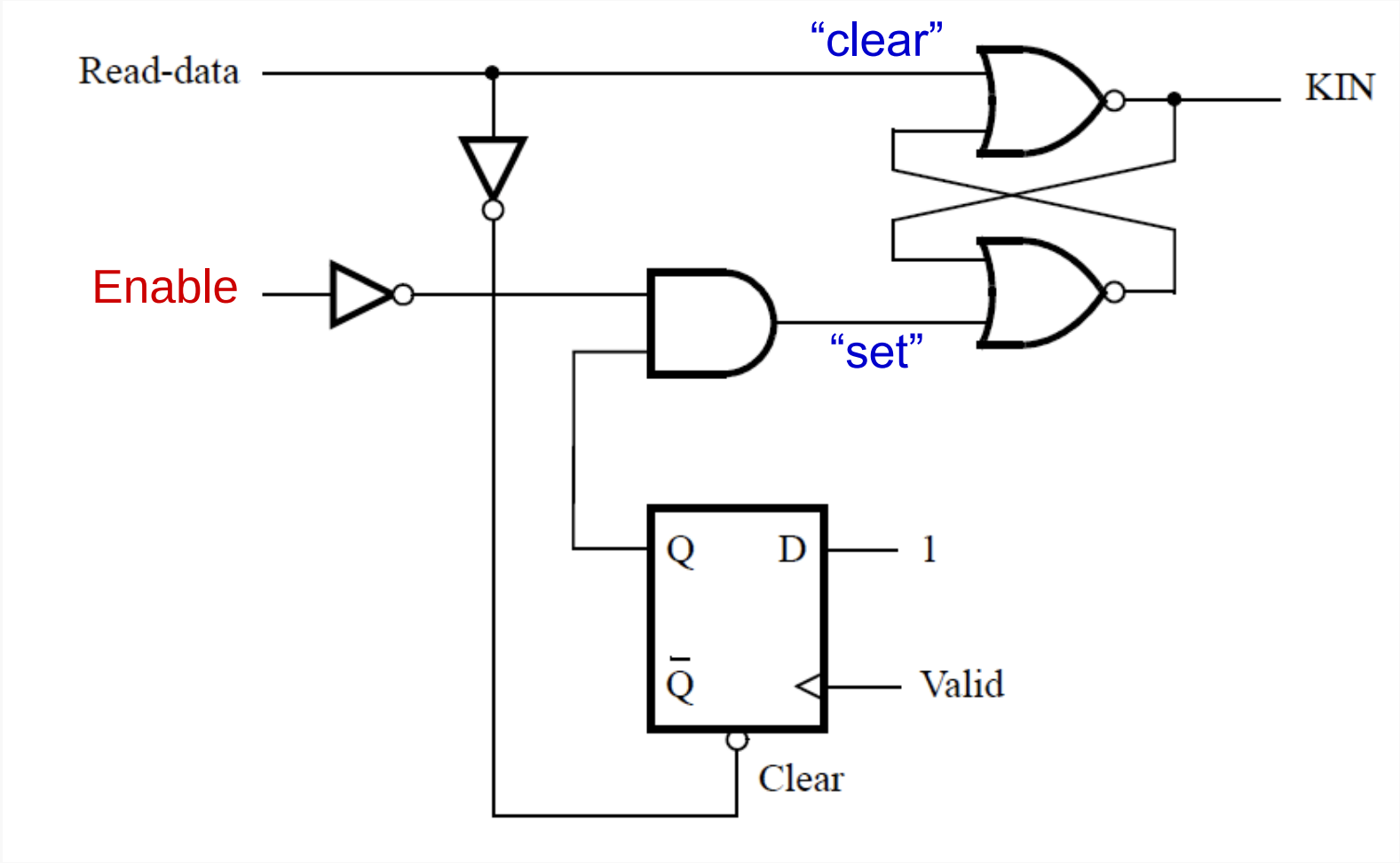


Parallel Input Interface II



Status Flag Control

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



Parallel Output Interface