

# SoC 101:

a.k.a., *“Everything you wanted to know about a computer but were afraid to ask”*

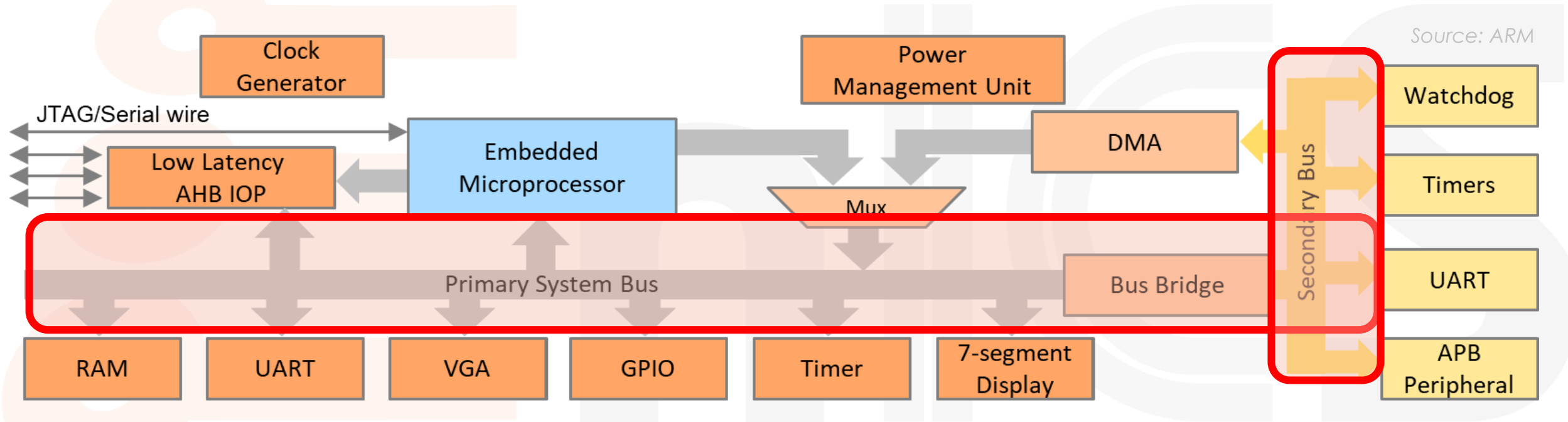
## On-Chip Interconnect

Prof. Adam Teman  
EnICS Labs, Bar-Ilan University

1 May 2023



# This Lecture



# Lecture Overview



On-Chip Communication

**nICS**  
Emerging Nanoscaled  
Integrated Circuits and Systems Labs


The Alexander Kofkin  
Faculty of Engineering  
Bar-Ilan University



Connecting with Peripherals

**nICS**  
Emerging Nanoscaled  
Integrated Circuits and Systems Labs

The Alexander Kofkin  
Faculty of Engineering  
Bar-Ilan University



Simple Bus Operation

**nICS**  
Emerging Nanoscaled  
Integrated Circuits and Systems Labs

The Alexander Kofkin  
Faculty of Engineering  
Bar-Ilan University



Higher Performance Buses

**nICS**  
Emerging Nanoscaled  
Integrated Circuits and Systems Labs

The Alexander Kofkin  
Faculty of Engineering  
Bar-Ilan University

On-Chip  
Communication

Connecting with  
Peripherals

Simple Bus  
Operation

Higher  
Performance  
Buses

# On-Chip Communication

# Typical Computing System

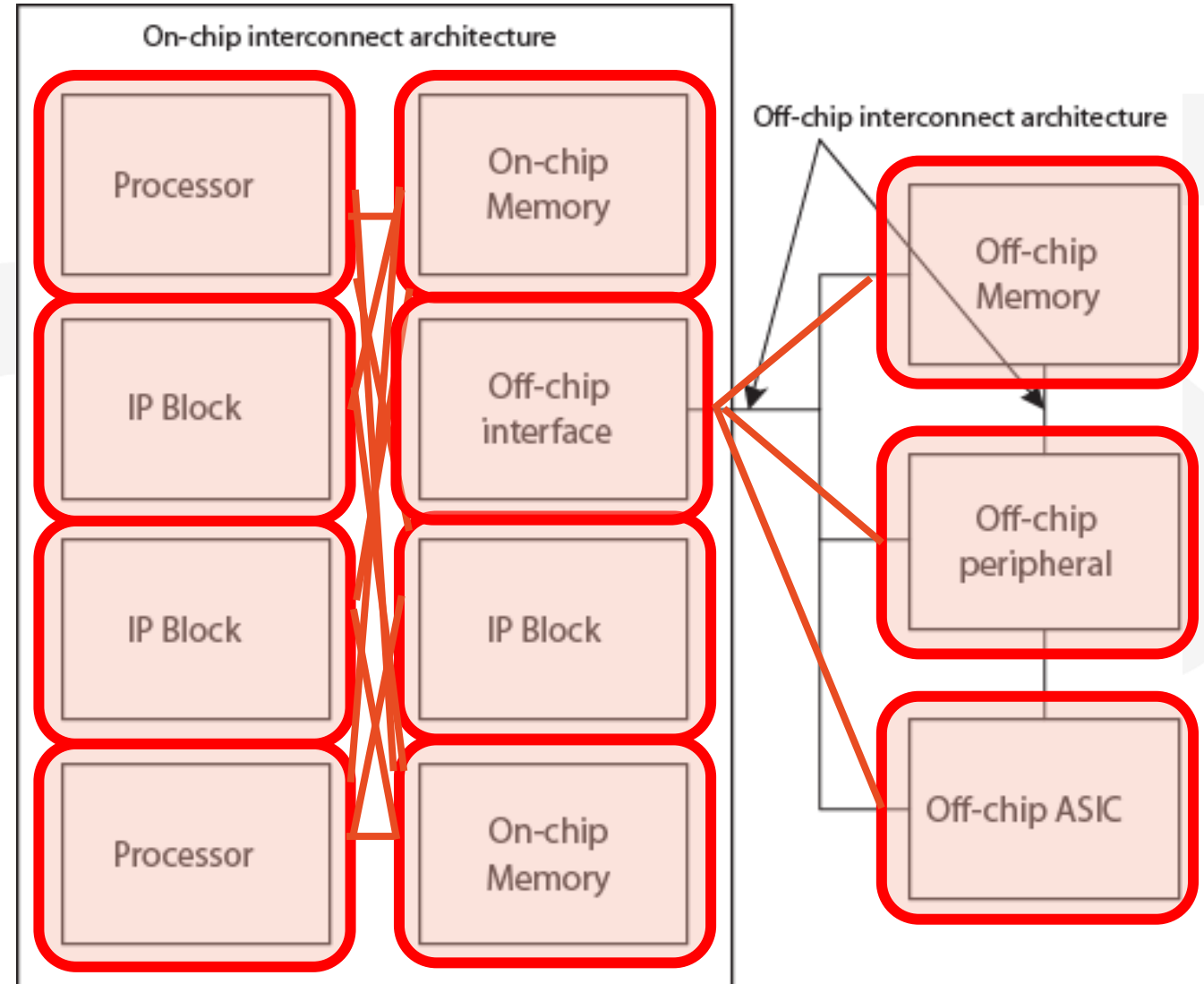
- **On-Chip Interconnect**

- Processors
- IP Blocks
- On Chip Memory

- **Off-Chip Interconnect**

- Off-chip peripherals
- Off-chip memory
- Off-chip ASICs

- In this lecture, we will focus on **On-Chip Interconnect**



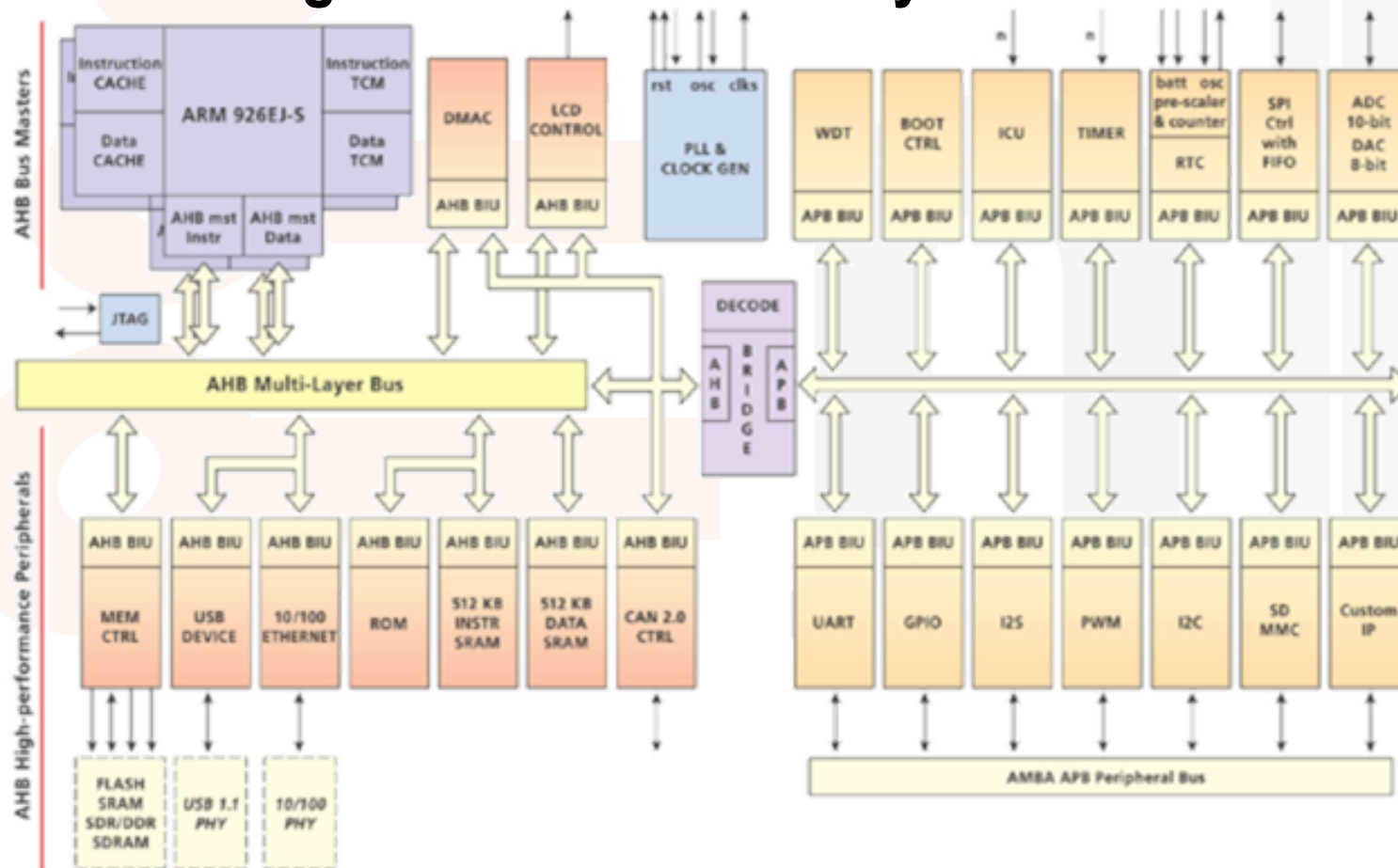
# Communication Considerations

**System-level issues and specifications for choosing communication architecture:**

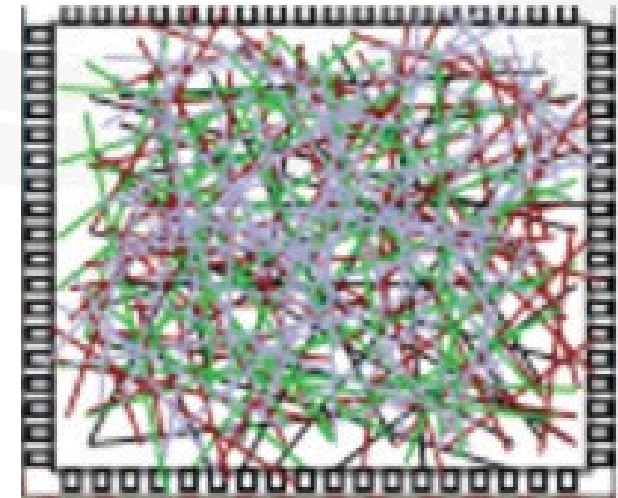
- **Communication Bandwidth**
  - Rate of information transfer (bytes/sec)
- **Communication Latency**
  - Time delay between a request and response
  - Application dependent, e.g., Video Streaming vs. two-way communication
- **Master and Slave**
  - Who can control transactions? What can be controlled?
- **Concurrency Requirement**
  - The number of independent simultaneous channels open in parallel.
- **Multiple Clock Domains**
  - Different IPs may operate at different frequencies.

# System-level Trends

- Heterogeneity among components that need to be interconnected
- Increasing volume and diversity of traffic



- Complexity of communication logic can easily compare to a small microprocessor!

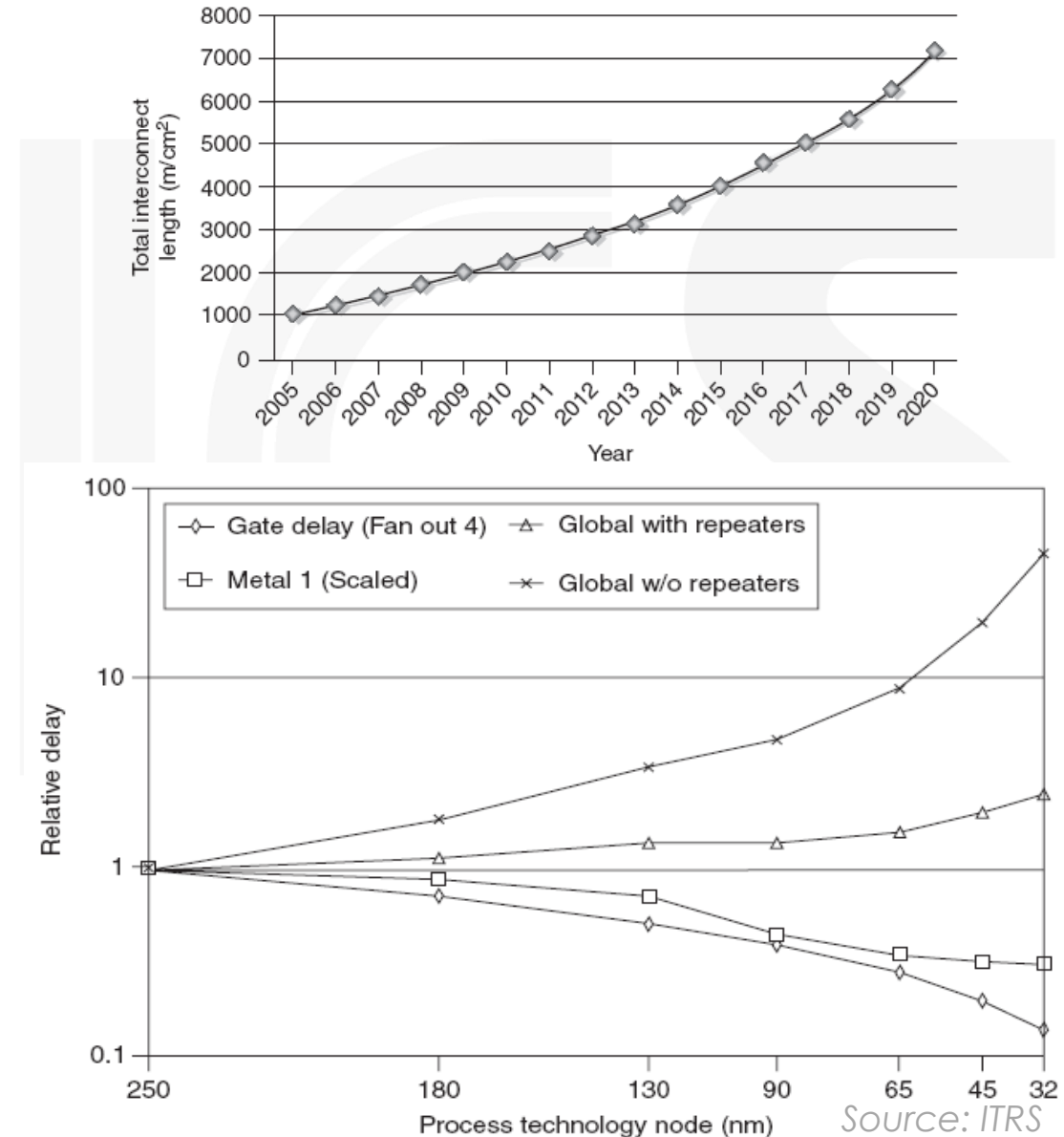


# Interconnect Scaling Trends

- Global wires **scale slower** than transistors/gates
  - Gates, local wires scale with technology, global wires do not
  - Global on-chip comm to operation delay changed from 2:1 to 9:1 over a few technology generations

Operation	Delay	
	(0.13um)	(0.05um)
32b ALU Operation	650ps	250ps
32b Register Read	325ps	125ps
Read 32b from 8KB RAM	780ps	300ps
Transfer 32b across chip (10mm)	1400ps	2300ps
Transfer 32b across chip (20mm)	2800ps	4600ps

Source: Bill Dally, DAC 2009 keynote





# Need for Communication-centric Design

- **Communication is THE most critical aspect affecting system performance**
- Communication architecture consumes up to **50% of total on-chip power**
- Ever increasing number of wires, repeaters, bus components (arbiters, bridges, decoders etc.) **increases system cost**
- Communication architecture design, customization, exploration, verification and implementation takes up the **largest chunk of a design cycle**

**Communication Architectures** in today's complex systems **significantly** affect performance, power, cost and time-to-market!

# On-Chip Communication Architecture Design

Three topics to consider when discussing on-chip communication architecture:

- **Communication Topology**

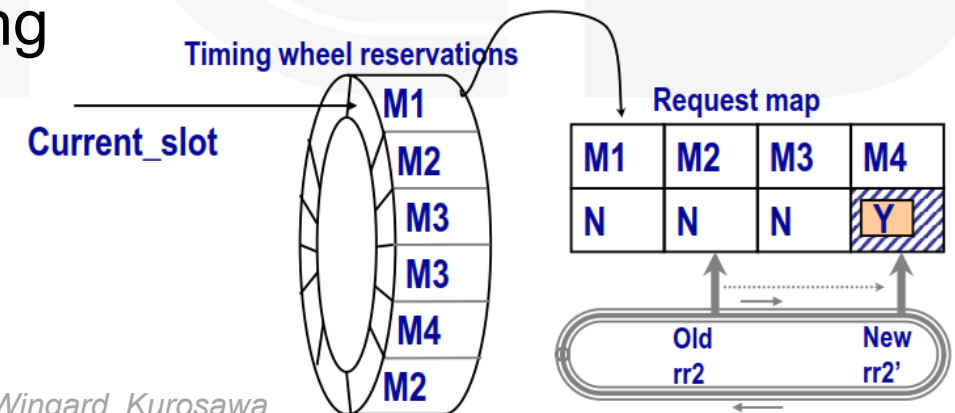
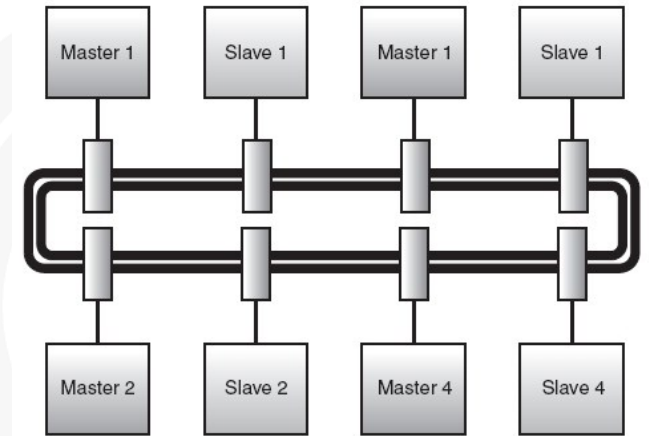
- How the communication resources are connected
- Simple shared bus, hierarchical bus structures, rings, mesh, custom bus networks

- **Protocols**

- How you manage the communication resources
- Static priority, TDMA, round-robin, token passing

- **Mapping of System Communications**

- Which components connect where?
- e.g., exploit locality, by putting close components on same bus



Wingard, Kurosawa,  
IEEE CICC, 1998

© Adam Teman, 2023

On-Chip  
Communication

Connecting with  
Peripherals

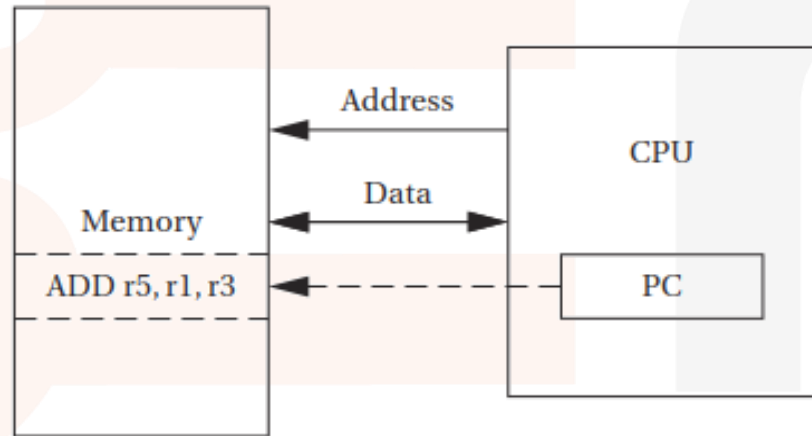
Simple Bus  
Operation

Higher  
Performance  
Buses

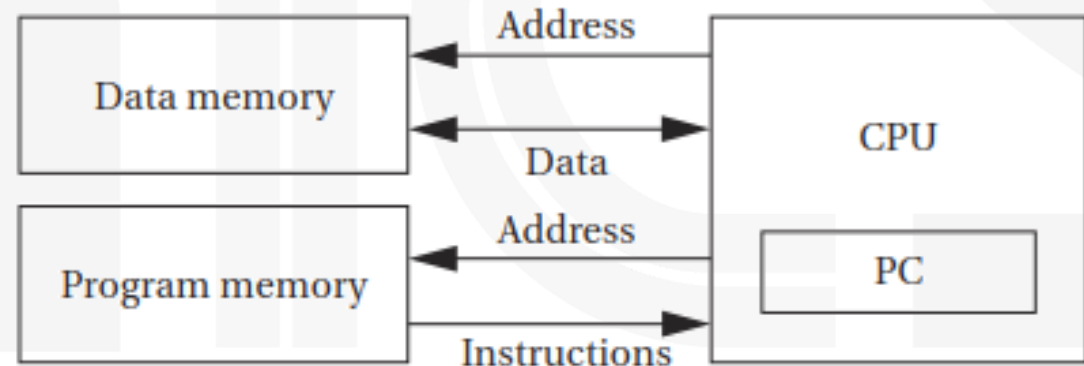
# Connecting with Peripherals

# Connecting with Memory

- In our discussion of Microprocessors, we *assumed* the existence of external memory components:
  - In a [Princeton Architecture](#), one homogenous memory space.
  - In a [Harvard Architecture](#), separate channels for [Instruction](#) and [Data Memory](#)



Princeton Architecture



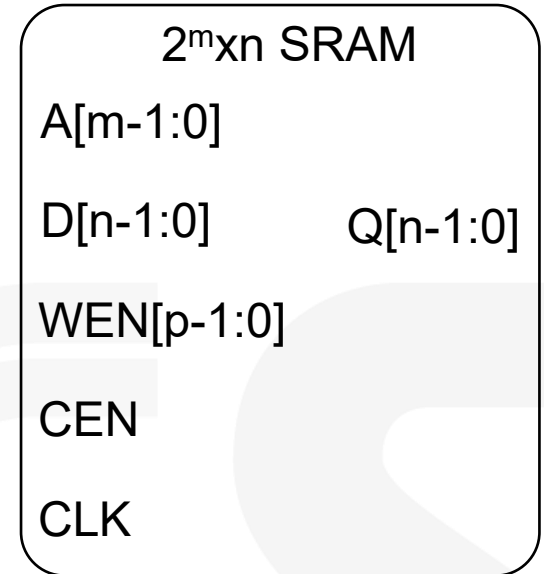
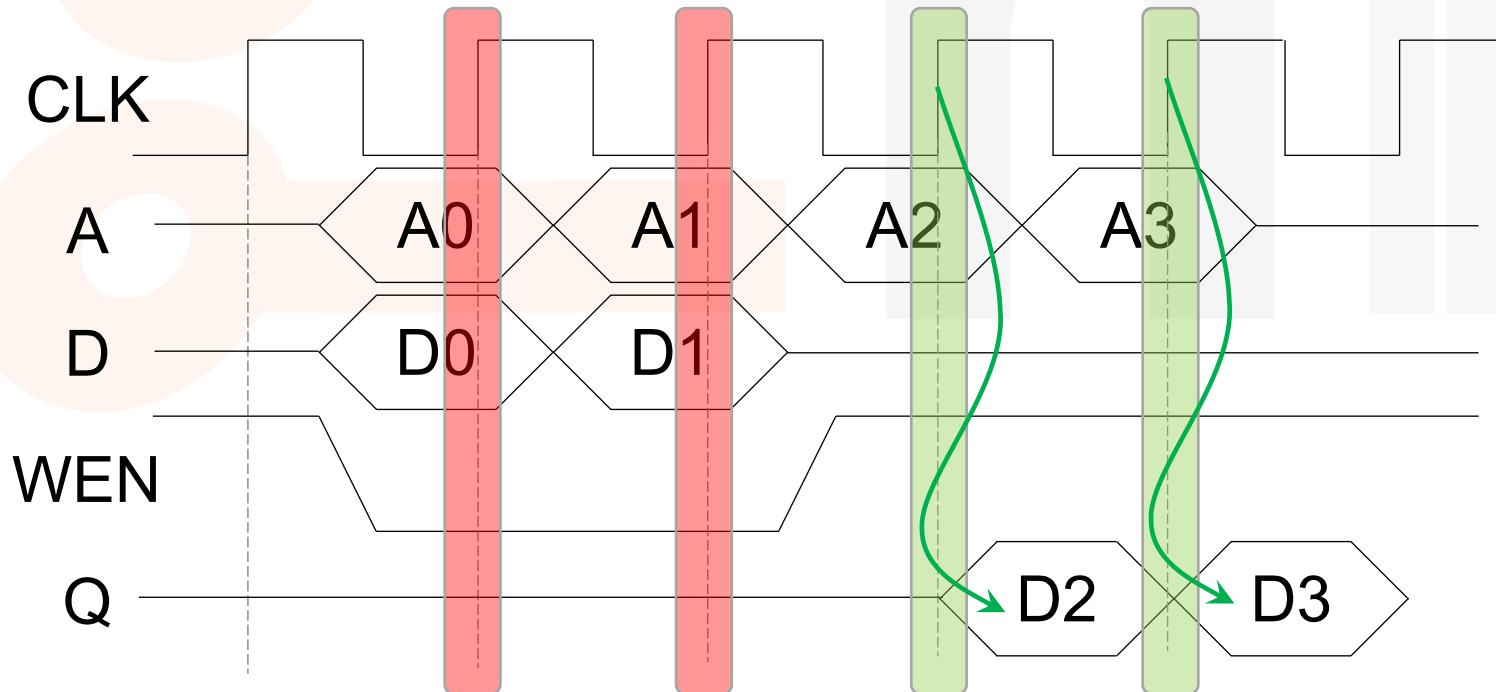
Harvard Architecture

Source: Wolf,  
*Computers as Components*

- Before we go into more complex interconnect options, let's start by looking at how these [tightly-coupled](#) memory blocks are interfaced with the CPU.

# Synchronous SRAM Interface

- A typical **on-chip synchronous SRAM** features:
  - Single-cycle write/read latency
  - Byte write mask
  - Active low Write Enable (i.e.,  $\overline{WEN}=1 \rightarrow$  Read Enable)
- The timing diagram can be viewed, as follows:

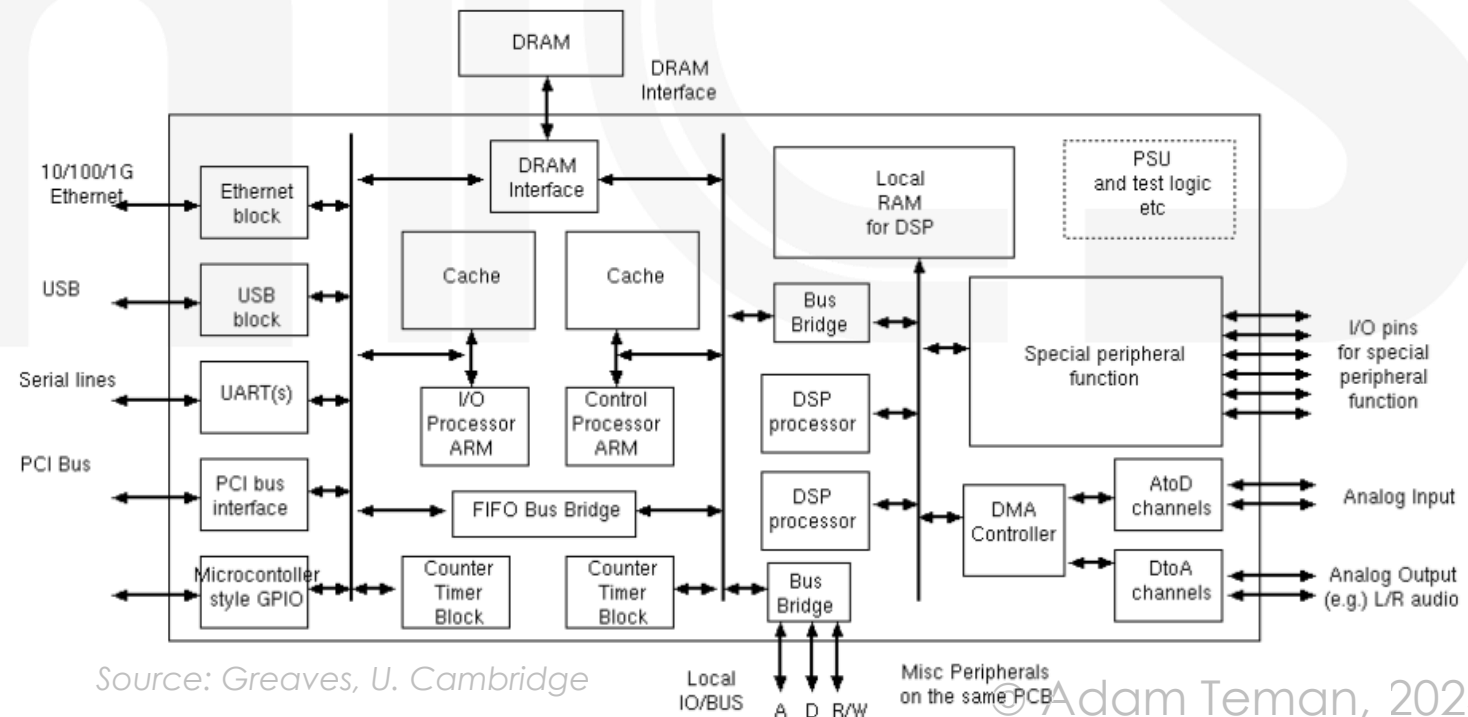


(1) Rising edge of the clock results in **WRITE**, when **WEN** is low.

(2) Rising edge of the clock results in **READ**, when **WEN** is high. Valid data appears on the output after a delay.

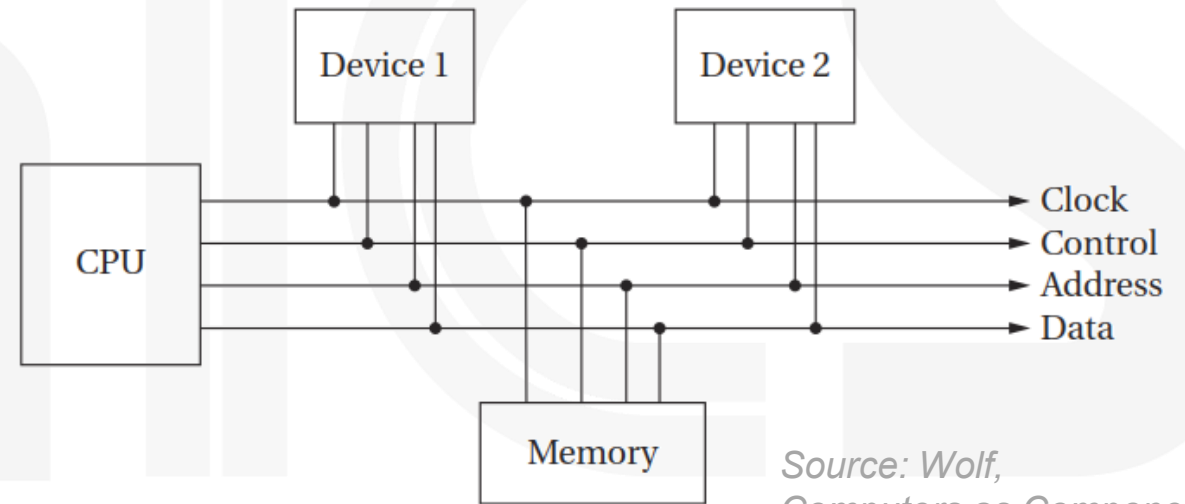
# Scaling to a larger network

- The previous SRAM interface is an example of a *point-to-point (p2p)* link
- **P2P links** are simple and fast, but not scalable
  - Every additional link added requires a *full (private)* set of signals and control
- Such an approach cannot even accommodate a *simple microcontroller*, much less a *complex SoC*.
  - Large amounts of SRAM
  - Slower, higher density memory (DRAM, Flash)
  - Peripherals and accelerators
- Therefore, we need a **System Bus**.



# System Bus

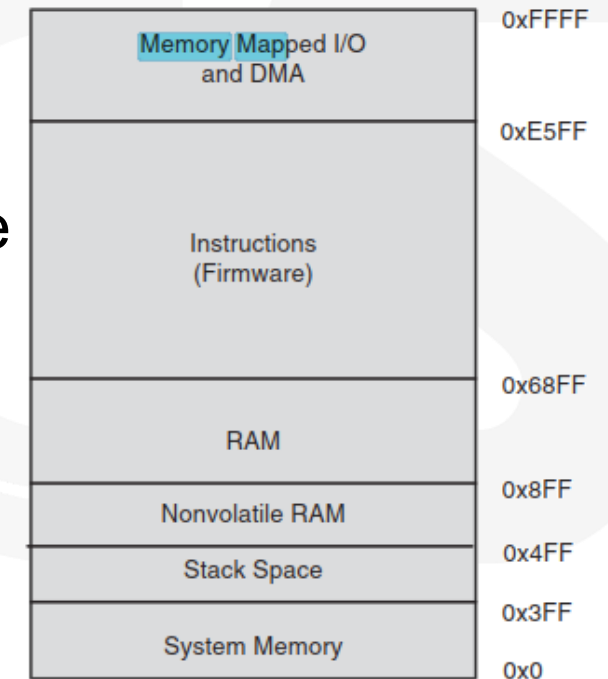
- A collection of **signals** (wires) to which one or more **IP components** (which need to communicate data with each other) are connected.
- In addition to the clock, a **synchronous bus** consists of:
  - An **Address** Bus
  - A **Data** Bus
  - A **Control** Bus
- In a typical system, the CPU serves as the **bus master** (a.k.a. “**manager**”) and initiates all transfers.
- Other devices are typically called **slaves** (a.k.a. “**subordinates**”) and they react to transfers initiated by the **master**.



Source: Wolf,  
Computers as Components

# Memory Mapping

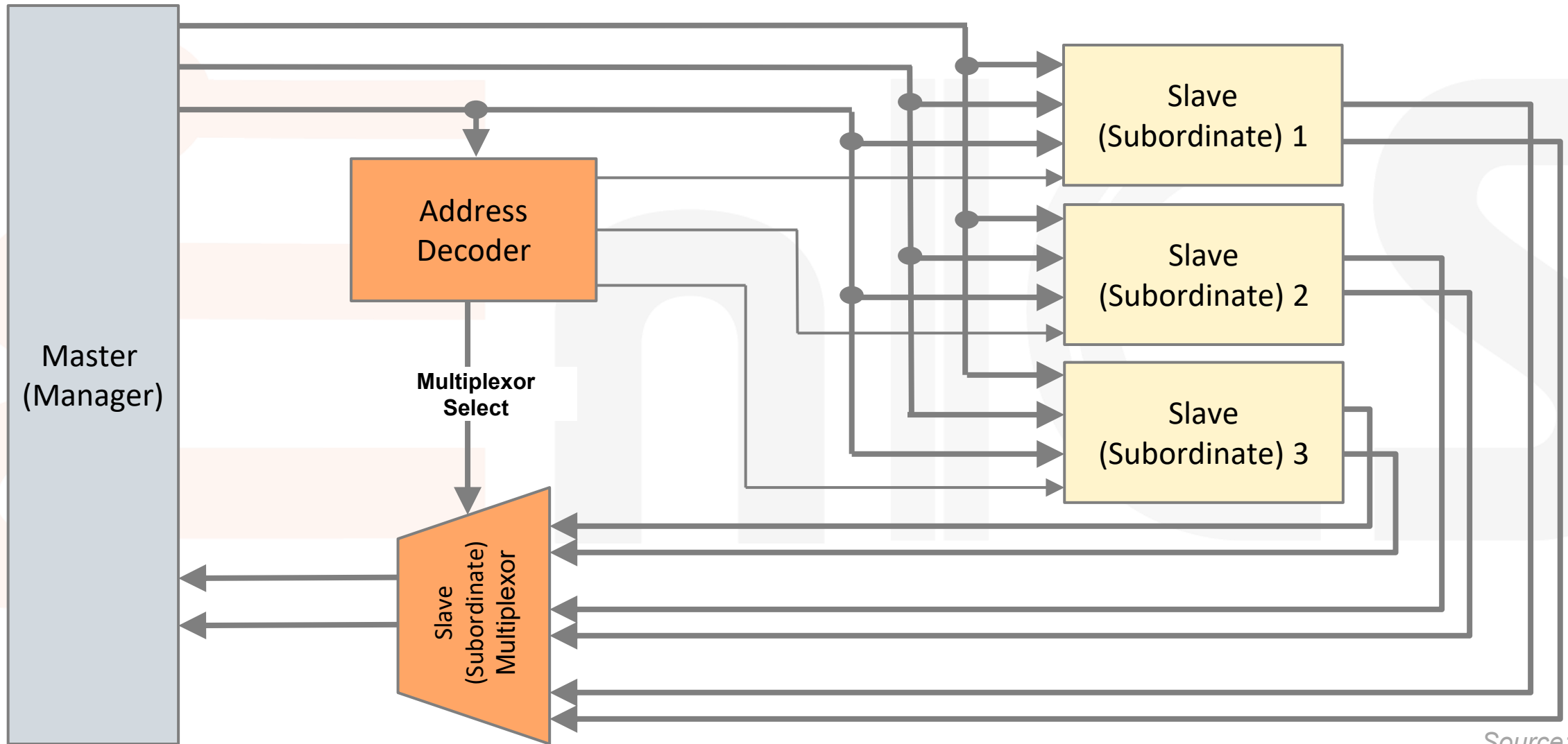
- Amazingly, the *three bus components* described above (**address**, **data** and **control**) can facilitate the majority of required control and data transfer.
- This is thanks to the concept of **Memory Mapping**
  - An *n-bit bus* supplies  $2^n$  unique byte addresses
  - With a wide bus (e.g., *32-bits*) only a small portion of these addresses are required for data storage (i.e., memory)
  - Therefore, every other device connected to the system is just treated as a memory address.
  - For example, *registers* of *peripherals* and *accelerators* are given **addresses** in the system **memory map**.
  - These *registers* are used to control the devices (e.g., “*start operation*” command) as well as to transfer data to and from them.



Source: Peckol, Embedded Systems



# Bus Terminology



Source: ARM

# Bus Terminology

- **Master (or Manager)**

- Component that initiates a read or write data transfer.

- **Slave (or Subordinate)**

- Component that does not initiate transfers and only responds to incoming transfer requests.

- **Decoder**

- Determines which component a transfer is intended for.

- **Bridge**

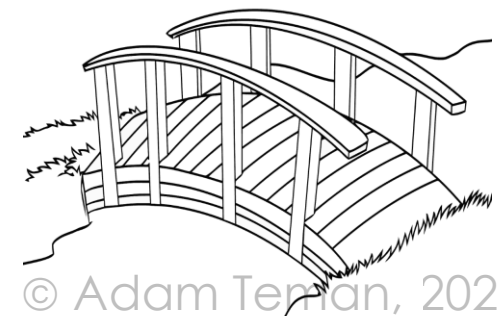
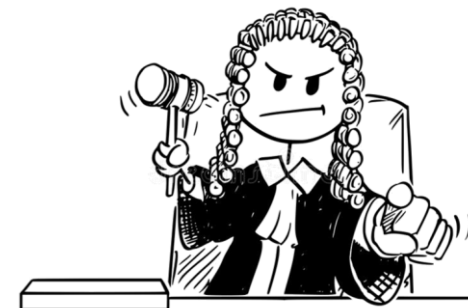
- Connects two busses. Slave on one side and master on the other.

- **Arbiter**

- Controls access to the shared bus.  
Selects master to grant access to bus.



A bus can accommodate **multiple Masters** and **multiple Slaves**



© Adam Teman, 2023

# Basic Bus Topologies

- **Shared bus**

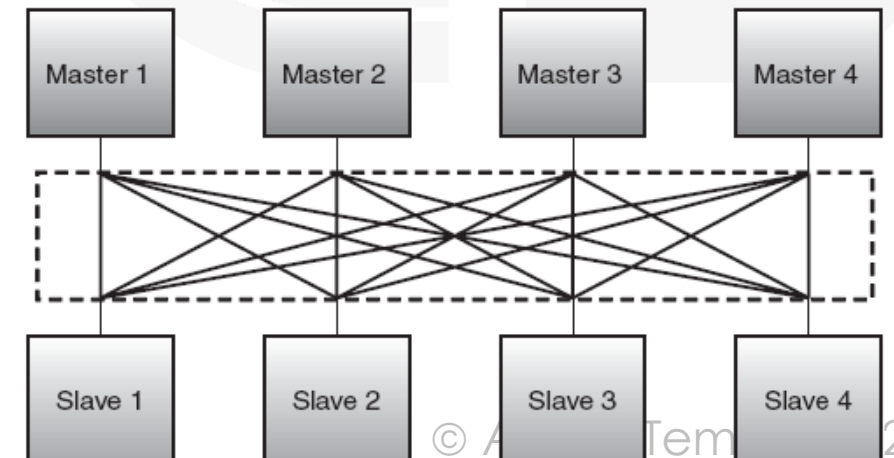
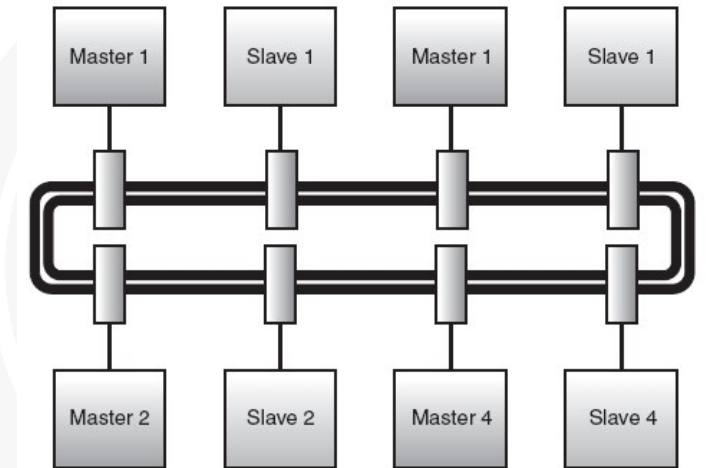
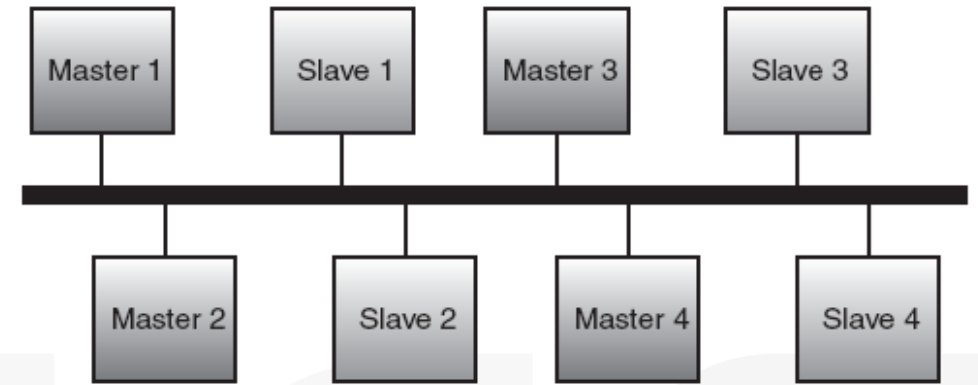
- Components share a single set of signals.
- Only one transaction can exist at a time.
- Hierarchical busses enable multiple transactions.

- **Ring**

- Very low cost connection (only connect to neighbor).
- Potential long propagation delay.
- Multiple concurrent transactions.

- **Crossbar**

- Point-to-point connection between all.
- Very high throughput, but very costly wiring.
- Can be reduced into partial crossbar/matrix.



On-Chip  
Communication

Connecting with  
Peripherals

Simple Bus  
Operation

Higher  
Performance  
Buses

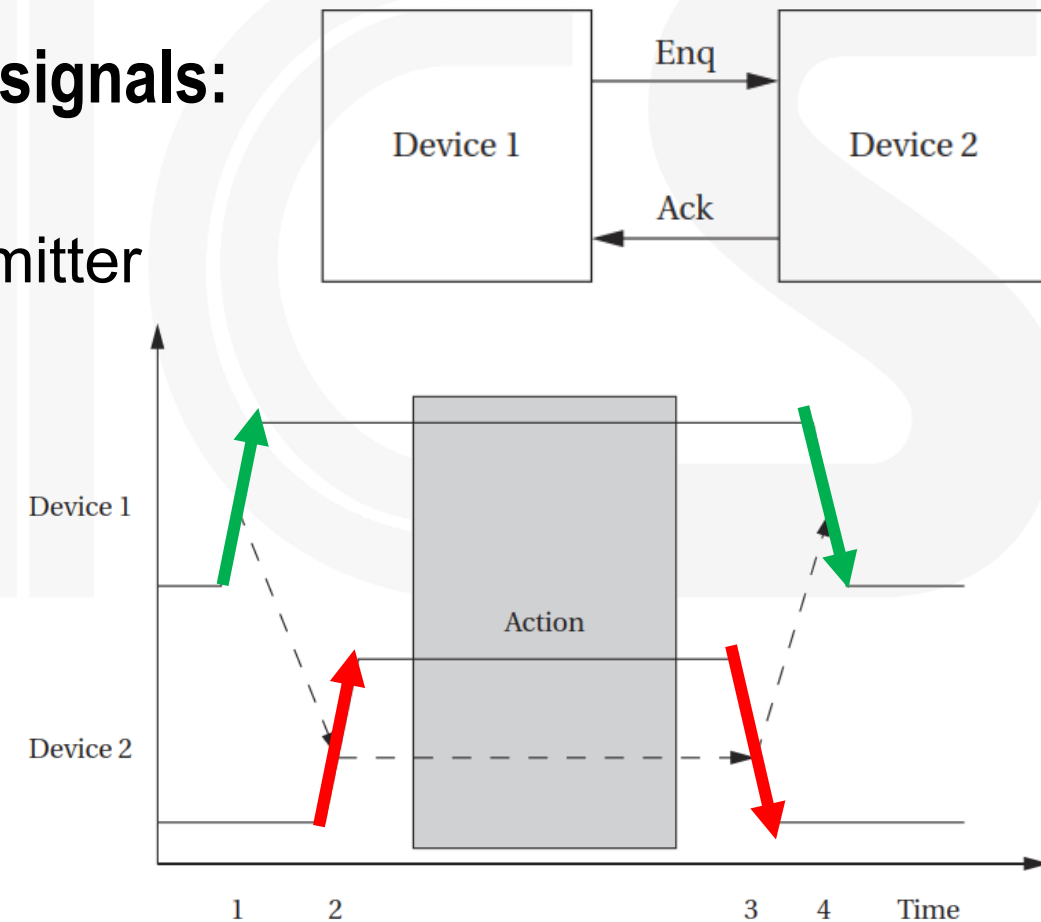
# Simple Bus Operation

# How does communication work?

- **Communication through ports consisting of:**
  - An **Interface** – set of pins/wires that connect the components.
  - A **Protocol** – set of rules for changing the logic levels and meaning of data.
- **Flow-control is implemented through handshaking**
  - Data is transferred when both the sender and receiver are happy to receive.
  - “**ack**”s and “**nack**”s are used for communicating readiness.
- **Communication is convened between:**
  - A **Master** – the port initiating the communication.
  - A **Slave** – the port responding to the communication.
- **Interfaces can be:**
  - **Synchronous** – both sides are clocked by the same clock.
  - **Asynchronous** – data is transferred through a clock domain crossing.

# Handshaking

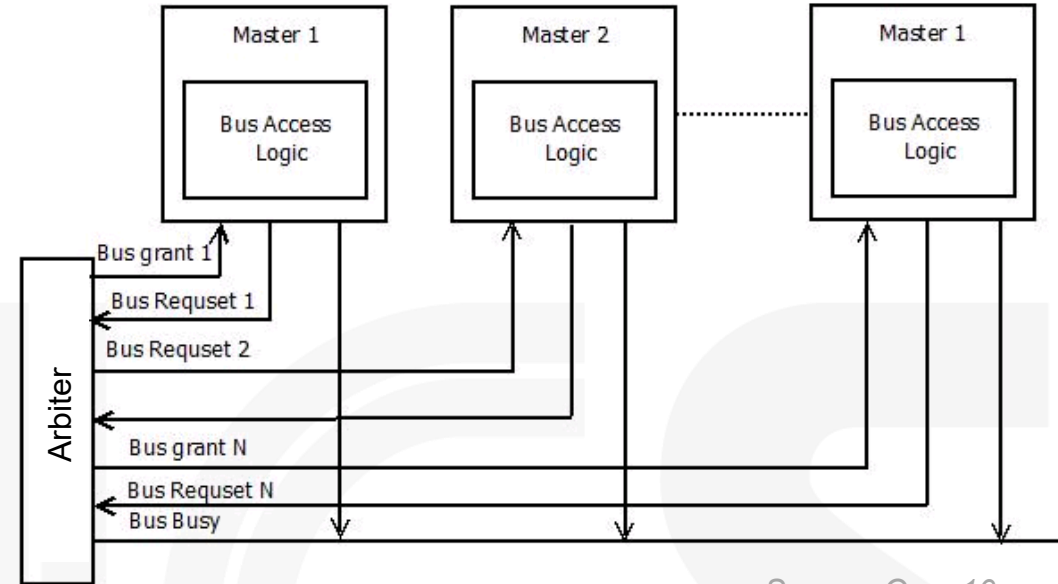
- In order to ensure that both devices are ready to communicate over the bus, a *handshaking protocol* is required.
- A conceptual handshake protocol utilizes two signals:
  - **ENQ** (enquiry) – from transmitter to receiver
  - **ACK** (acknowledge) – from receiver to transmitter
- The **four-cycle handshake** process includes:
  - **Device 1** raises **ENQ** to initiate transfer
  - **Device 2** raises **ACK**, when ready and transmission can start
  - **Device 2** lowers **ACK** to signal that data was received
  - **Device 1** lowers **ENQ** to finish



Source: Wolf,  
*Computers as Components*

# Bus Arbitration

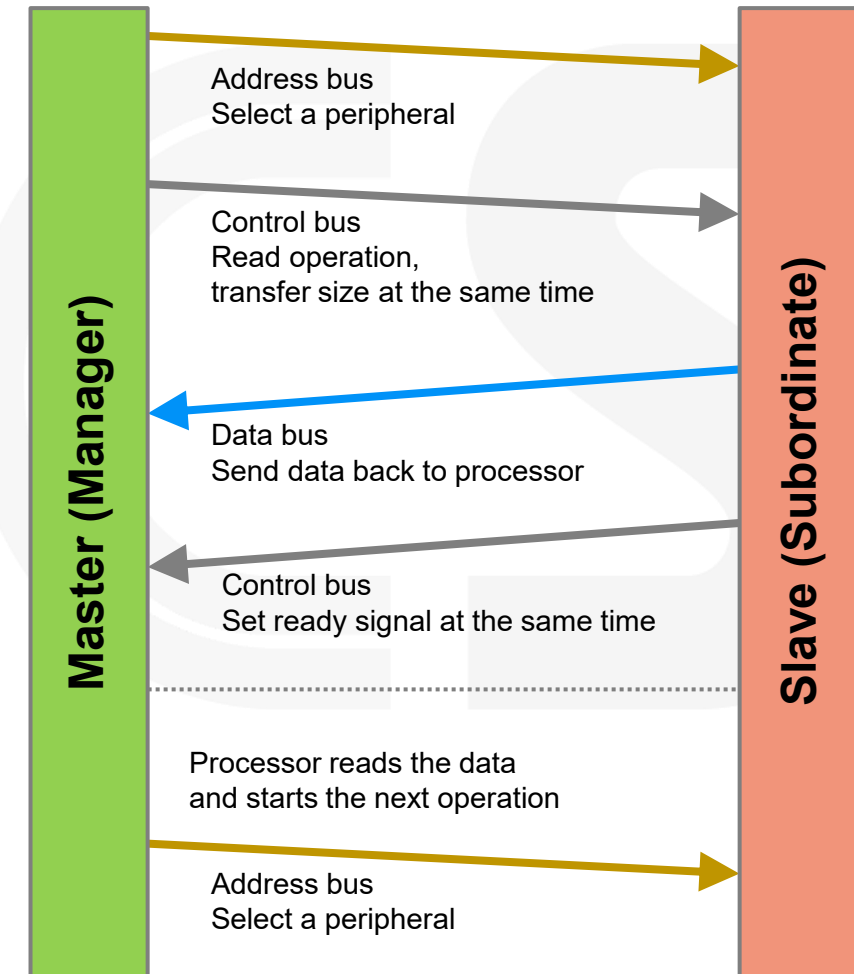
- Only one **master** can control the bus
  - Need some way of deciding who is **master**
  - And to make sure the right **slave** answers
- **Arbitration**
  - Decides which **master** can use the shared bus if several **masters** request bus access simultaneously
  - Common **arbitration schemes** include:  
Random, Priority-based, Round Robin, Time Division Multiplexing (TDMA), etc.
- **Decoding**
  - Determines the target for any transfer initiated by a **master**
  - Tells the right **slave** to put the response on the bus



Source: Ques10.com

# A Typical Bus Operation Example

- The following steps illustrate a typical operation to access a peripheral:
  - The **Master** (e.g., a processor) selects one **Slave** (e.g., peripheral or register) by giving the **address** to the address bus.
  - At the same time, it sets **control signals**, such as *read* or *write* and *transfer size*.
  - The **Master** waits for the **Slave** to respond.
  - Once the **Slave** is ready, it sends back the requested data to the **Master**.
  - At the same time, it sets the *ready* signal on the **control bus**.
  - Finally, the **Master** reads the transmitted data and starts another communication cycle



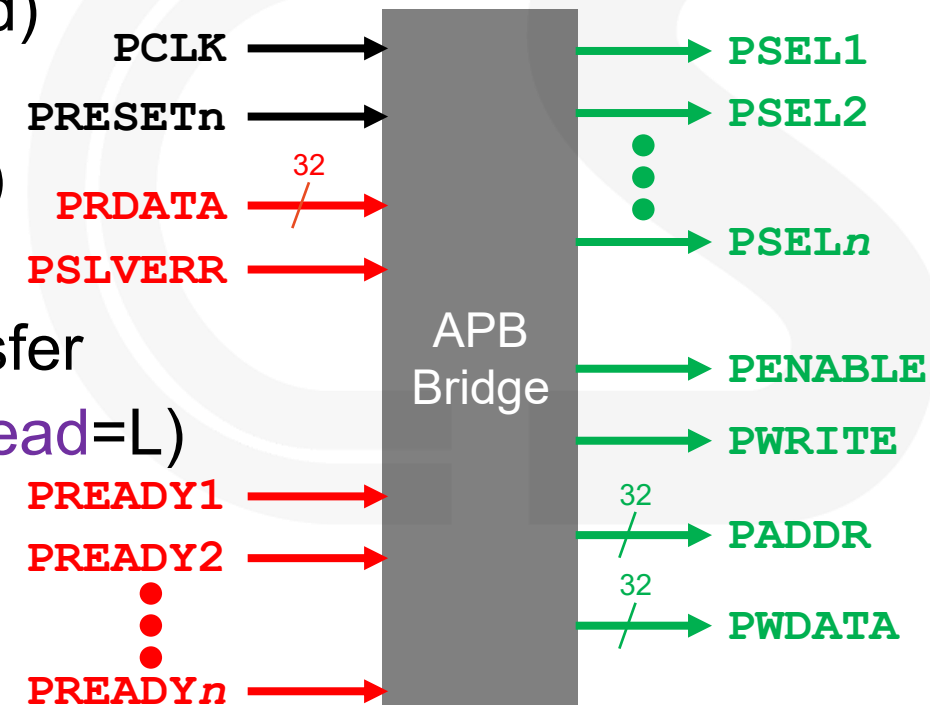
Source: ARM



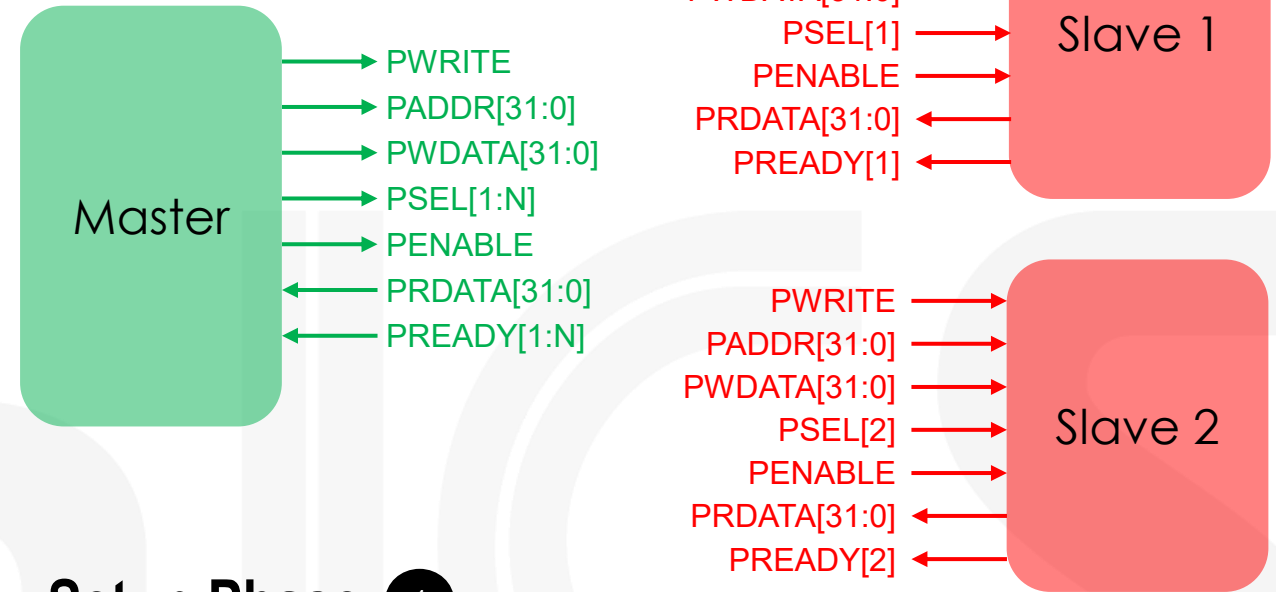
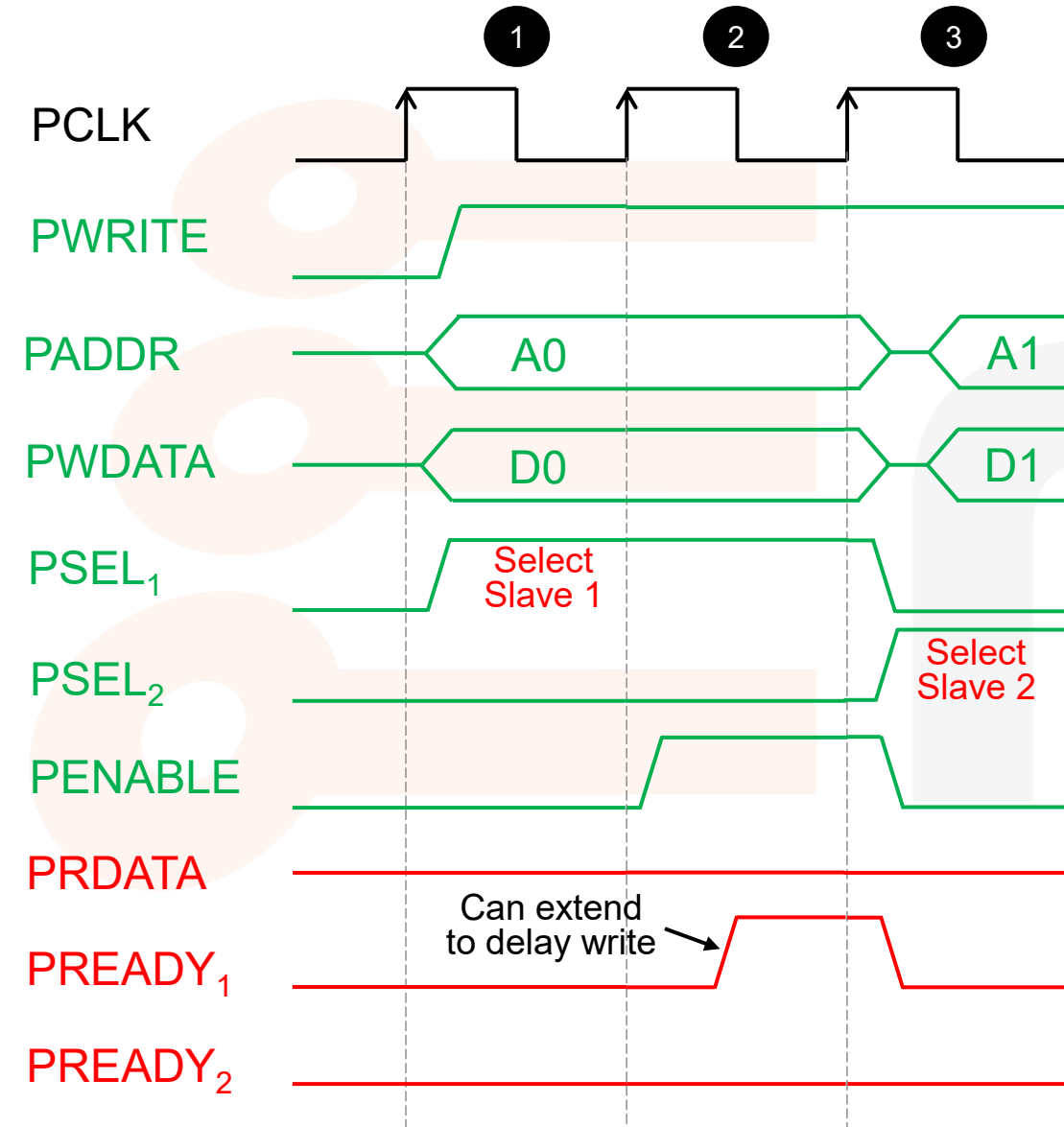
# Example: The Advanced Peripheral Bus (APB)

- **APB** is a simple, low performance bus, part of the AMBA specification by ARM.
- **APB** uses the following signals (**Master/Slave**):

- **PCLK**: the bus **clock** source (rising-edge triggered)
- **PRESETn**: the bus **reset** signal (active low)
- **PADDR**: the APB **address** bus (up to 32-bits wide)
- **PSELx**: the **select** line for each slave device
- **PENABLE**: indicates the 2<sup>nd</sup> cycle of an APB transfer
- **PWRITE**: indicates transfer direction (**Write**=H, **Read**=L)
- **PWDATA**: the **write data** bus (up to 32-bits wide)
- **PREADYx**: used to extend a transfer
- **PRDATA**: the **read data** bus (up to 32-bits wide)
- **PSLVERR**: indicates a transfer error (**OKAY**=L, **ERROR**=H)

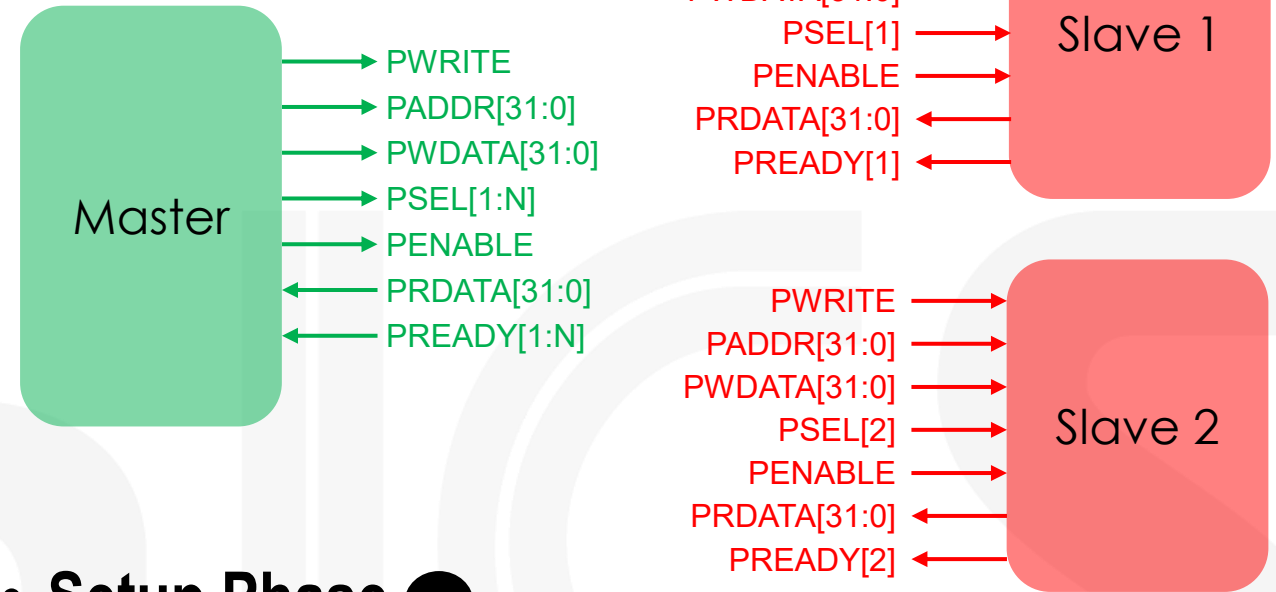
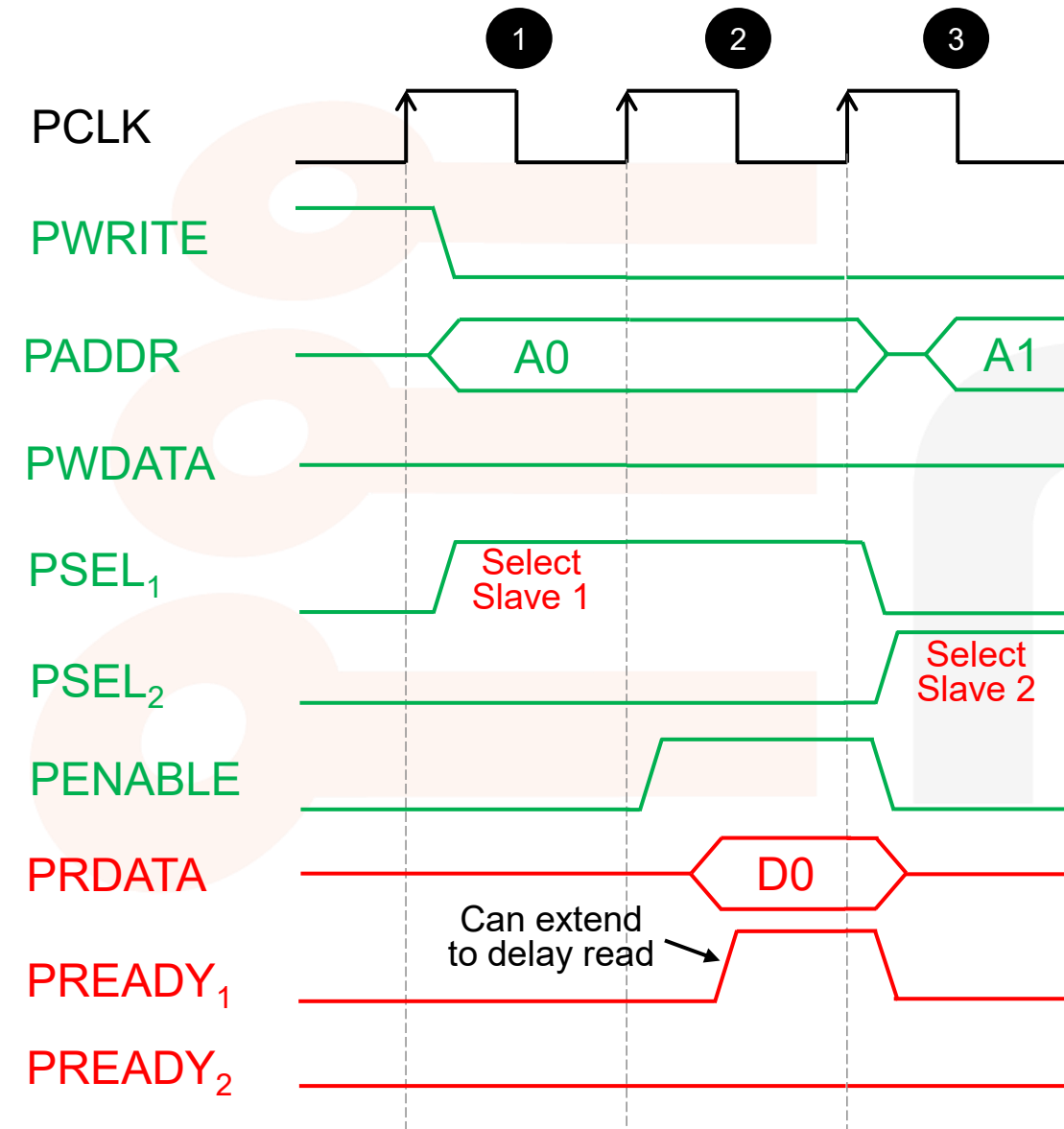


# APB Write Transfer



- **Setup Phase 1**
  - PWRITE, PADDR, PWDATA are set
  - PSEL is raised for selected Slave
- **Access Phase 2**
  - PENABLE is raised, with other signals held
  - When selected Slave acks, PREADY is raised
- **Next Transfer 3**
  - PENABLE is lowered by Master
  - PREADY may be lowered by Slave

# APB Read Transfer

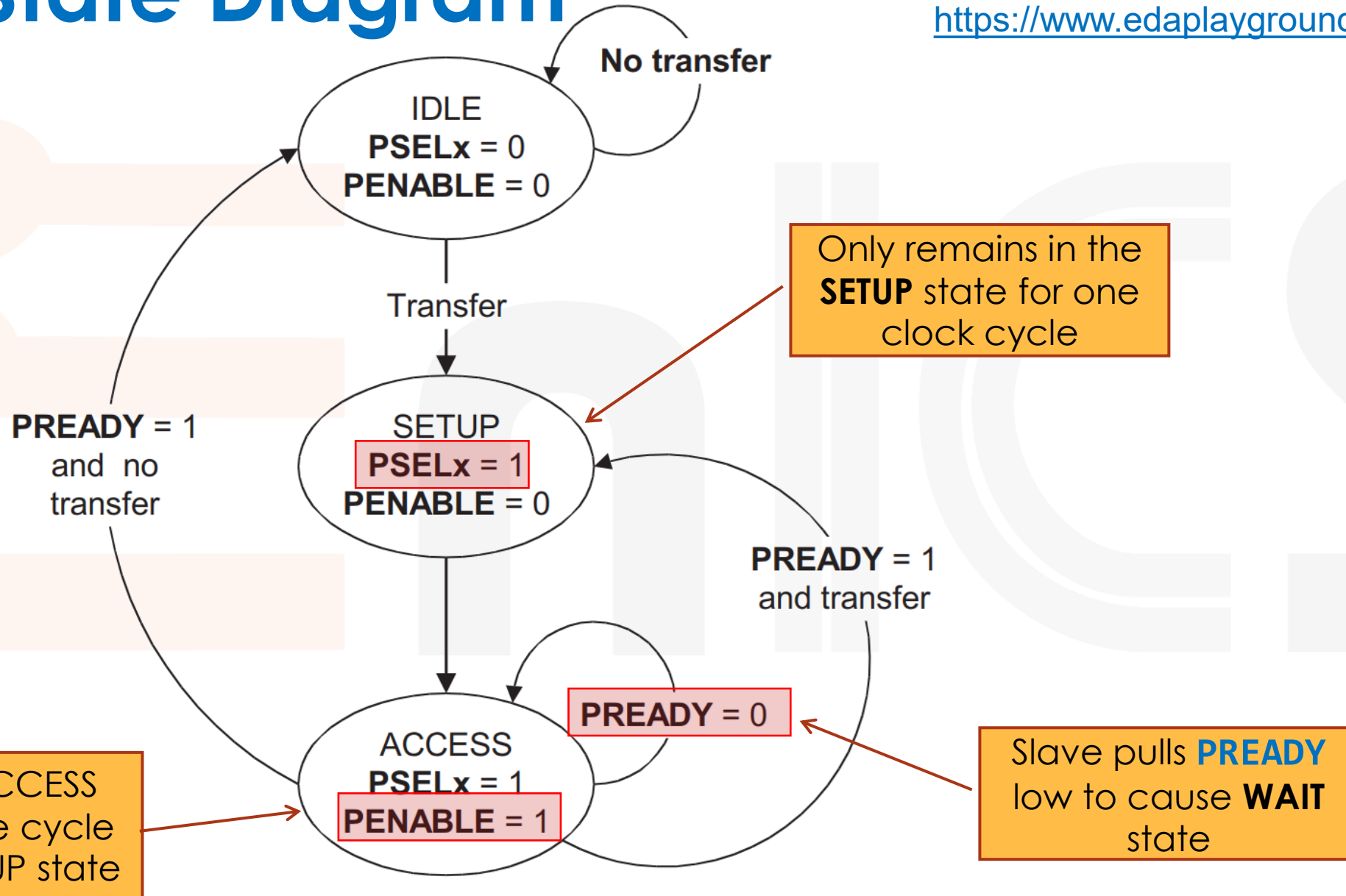


- **Setup Phase ①**
  - PWRITE is lowered, PADDR is set
  - PSEL is raised for selected Slave
- **Access Phase ②**
  - PENABLE is raised, with other signals held
  - Slave raises PREADY and drives PRDATA
- **Next Transfer ③**
  - PENABLE is lowered by Master
  - PREADY may be lowered by Slave

# APB State Diagram

See straightforward RTL implementation  
by Quick Silicon at:

<https://www.edaplayground.com/x/AXrK>



On-Chip  
Communication

Connecting with  
Peripherals

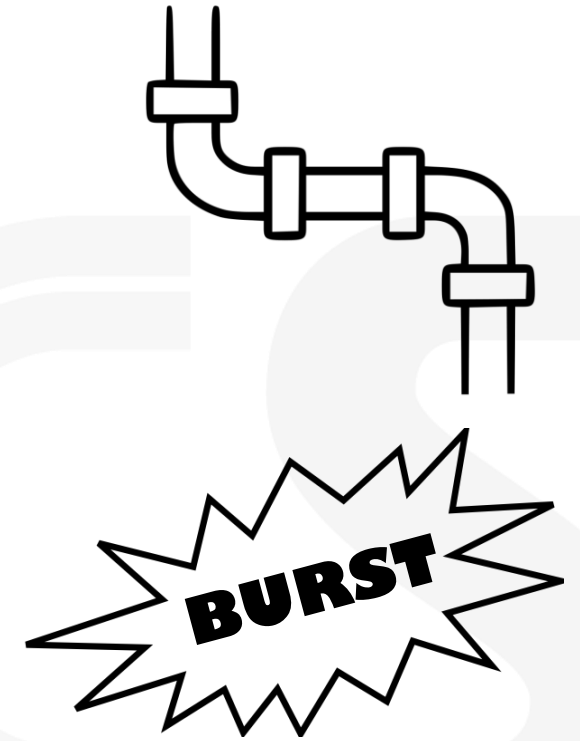
Simple Bus  
Operation

Higher  
Performance  
Buses

# Higher Performance Buses

# Increasing Bus Performance

- The **APB** example was a simple **low-performance bus**
  - *Two-cycles* to carry out a single transfer
  - Each transfer requires the definition of **address** and **data**
- However, this can be easily improved by:
  - Pipelining transfers:  
Overlap **Setup** and **Access** phases to achieve *single-clock edge* transfers.
  - Burst operations:  
Provide a **single address** with **multiple (sequential) data**.
  - Wide bus widths:  
Support **64-bit**, **128-bit**, or even wider buses.

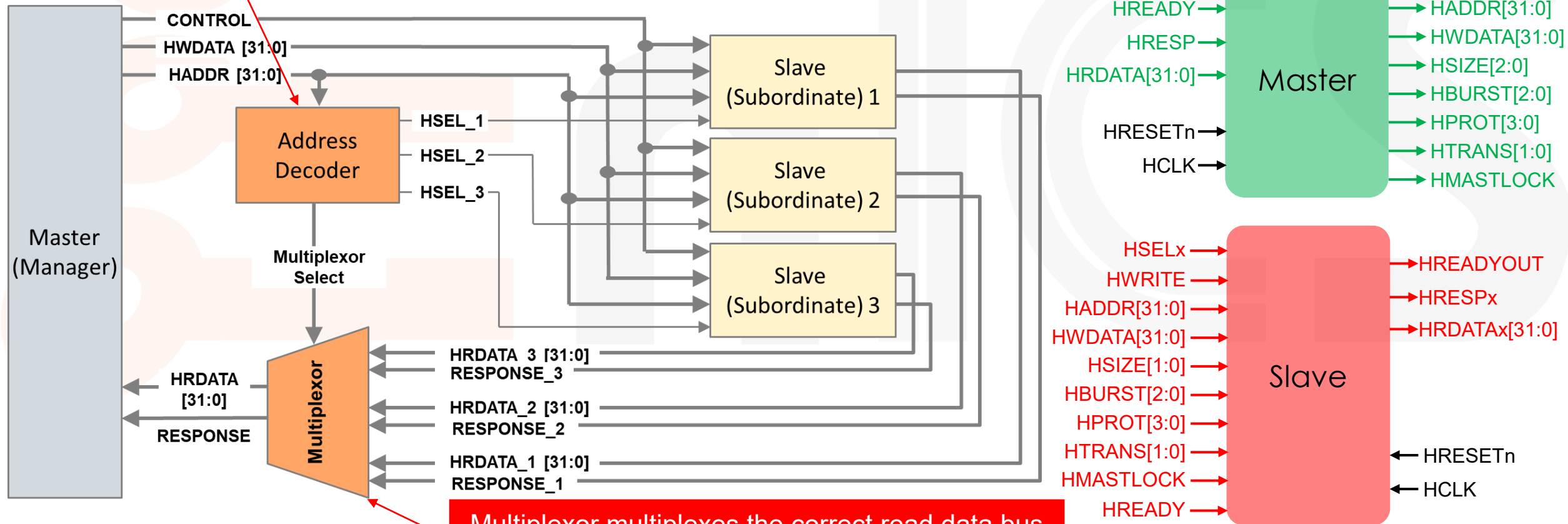


source: [gruzovikpress.ru/](http://gruzovikpress.ru/) © Adam Teman, 2023

# Example: Advanced High-Performance Bus

- A more advanced bus defined within the **AMBA** specification is the **Advanced High-Performance (AHB)** bus.

Decoder selects the correct slave according to the address.

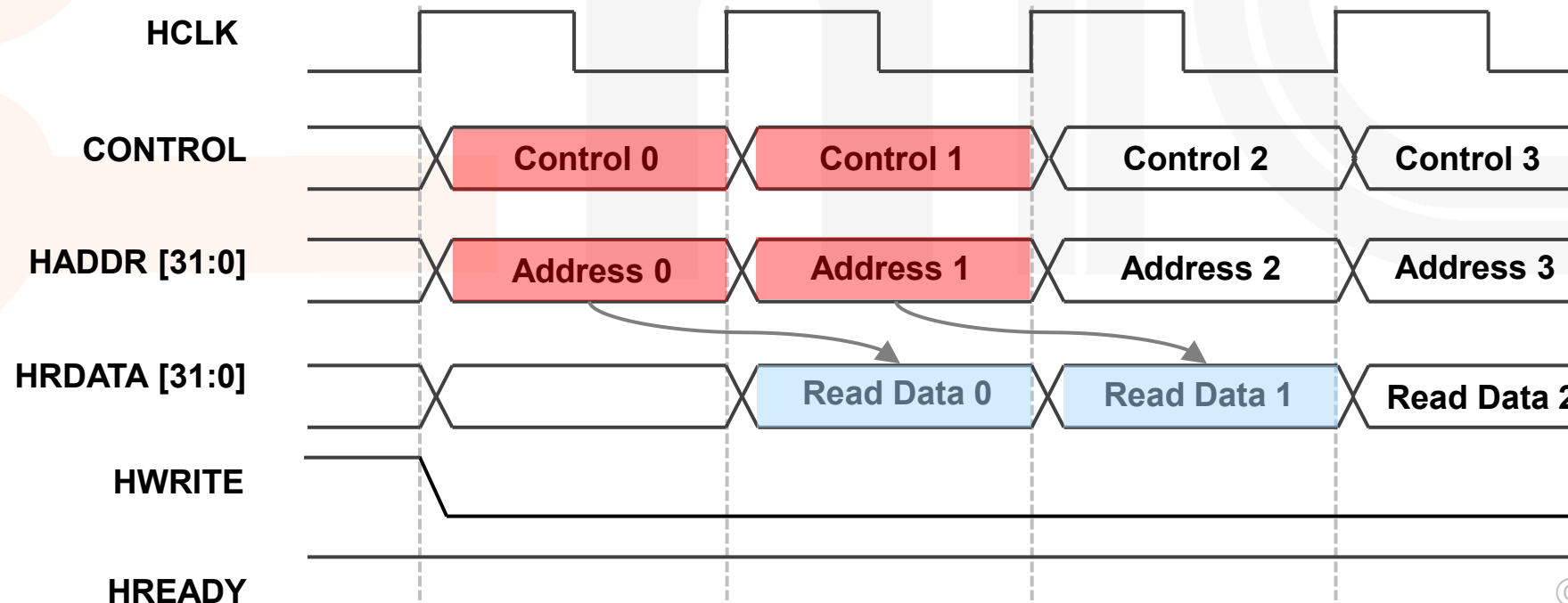


Multiplexor multiplexes the correct read data bus and response from the selected slave.



# AHB Basic Read Operation

- An AHB transfer consists of two (overlapped) phases:
  - The address phase: Master drives address and control signals onto the bus.
  - The data phase: Selected Slave responds with HRDATA.  
HREADY is lowered to extend the data phase
- The *next* address phase is applied during the *current* data phase.





# AHB Burst Transfer

- AHB Supports bursts of different lengths

- Master provides one address and the burst length
- Several operations (W/R) are applied to incrementing addresses
- Allows reducing the overhead of the address phase

1 Provide address and burst length

Lower HWRITE

First HTRANS=NONSEQ

2 Slave reads out first address

Master is not ready so sets BUSY

3 Master sets HTRANS=SEQ

Read data (delayed) is ignored

4 Master increments address

Slave reads out second address

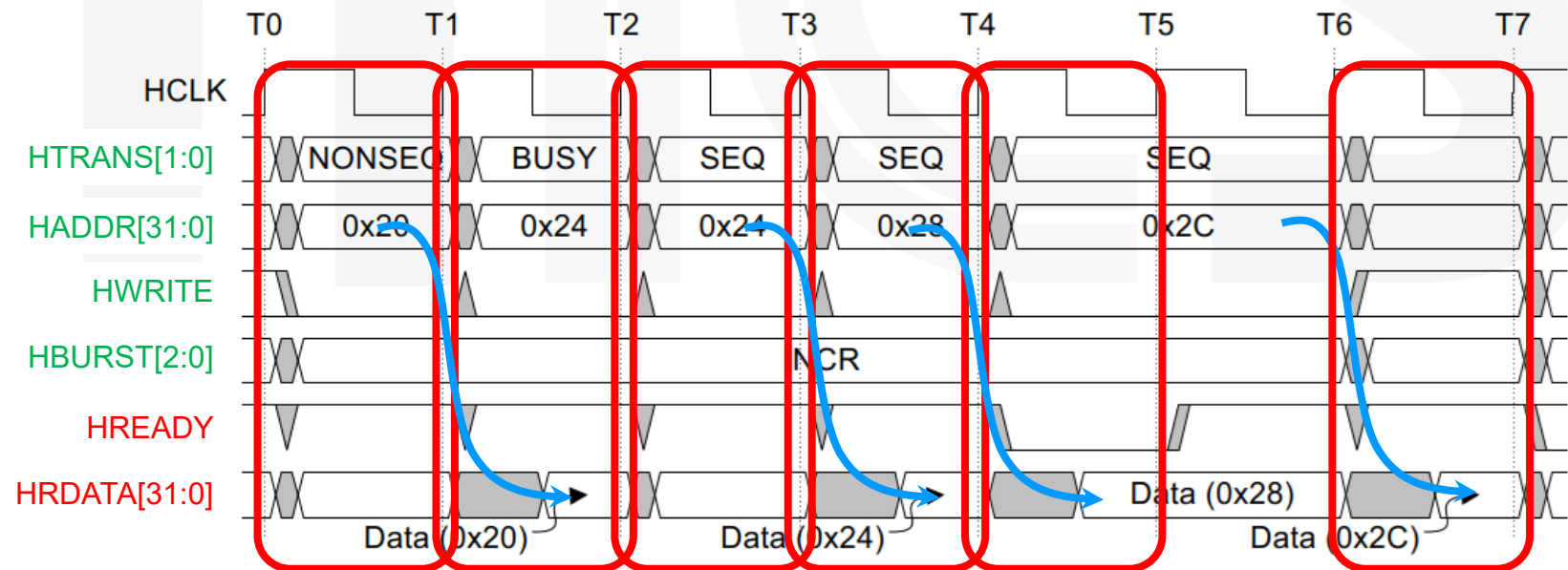
5 Slave reads out third address

But lowers HREADY to delay

6 Slave reads out last address

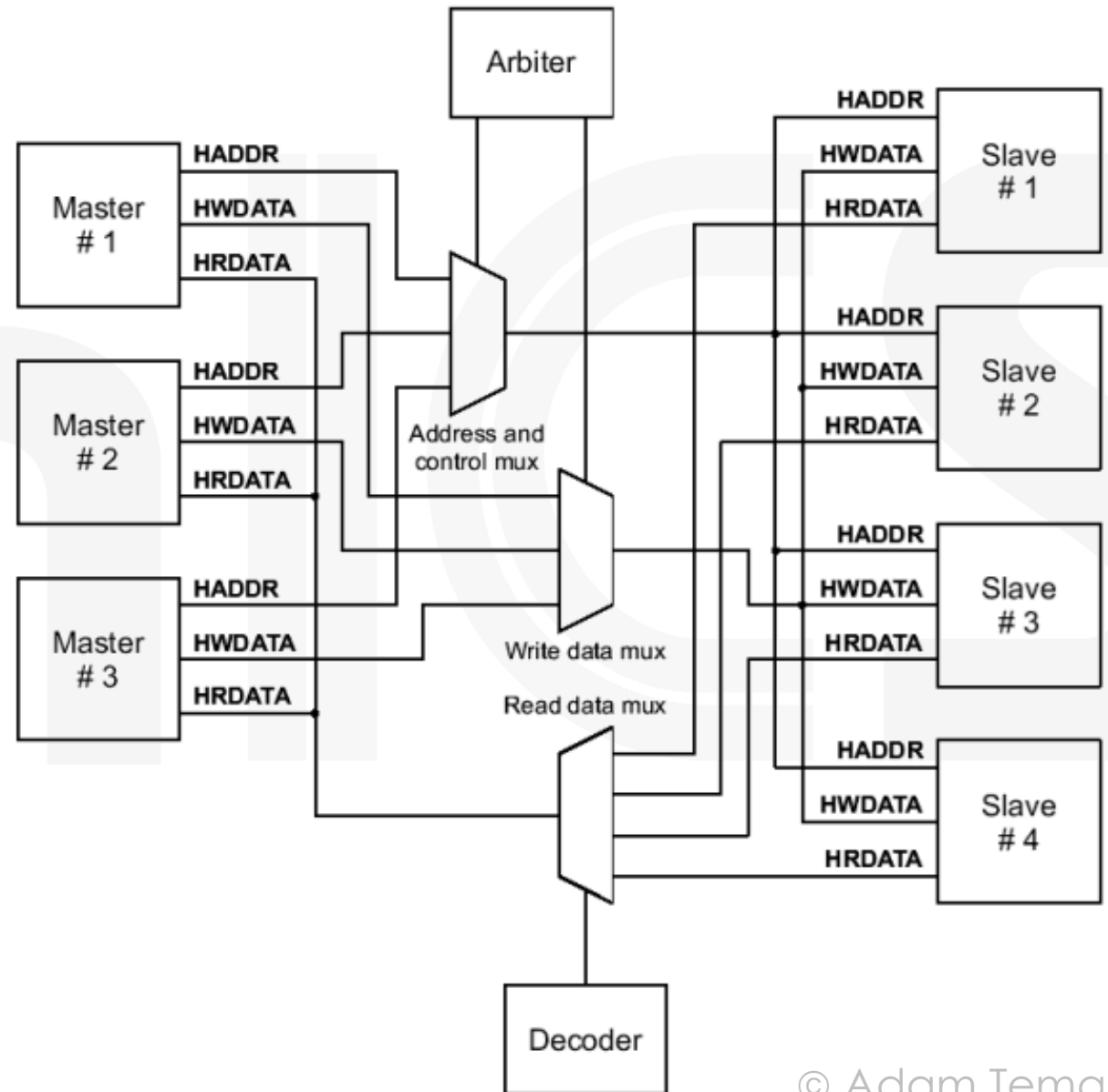
one cycle after raising HREADY

Example: 4-beat Read Burst with “wait” state.



# AHB with Multiple Masters

- Arbiter controls which master initiates current transaction.

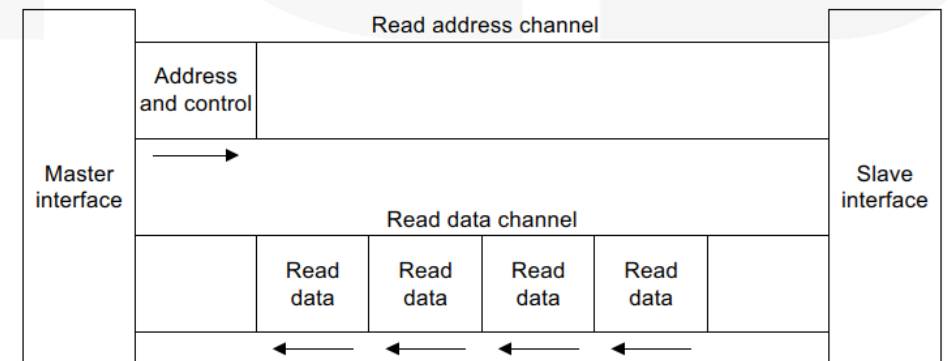
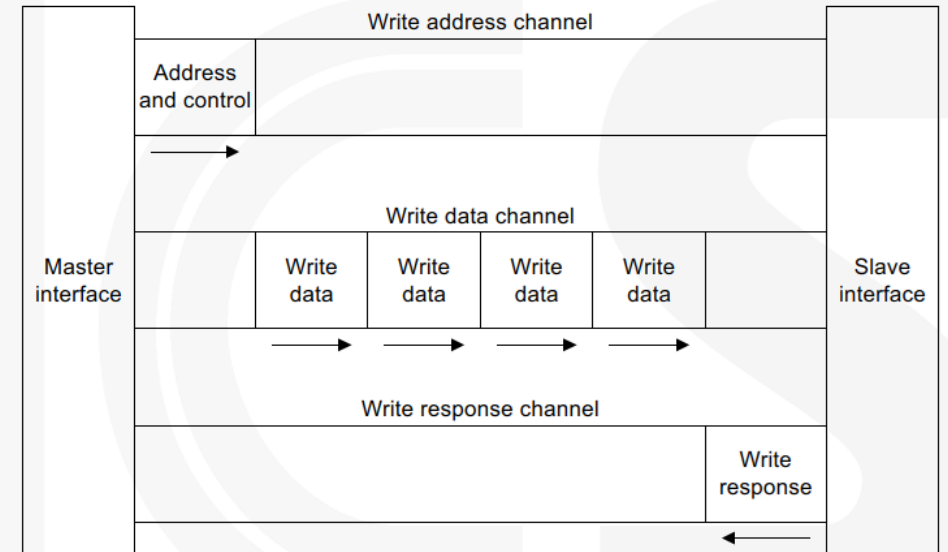
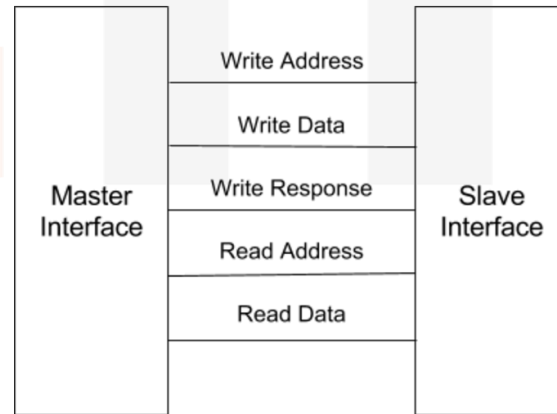


# Even More Performance

- **To get even more performance, a bus can have the following capabilities:**
  - Independent read and write channels:  
Simultaneous reads and writes → Improved bandwidth
  - Multiple outstanding addresses:  
**Master** can issue new transactions without waiting for previous to complete
  - Out-of-order transaction completion:  
A later transaction can complete before a previously launched one.
  - Independent address and data operations:  
If there is no strict timing relationship between address and data operations, they can be arbitrarily separated
- **These and other features are supported by the **Advanced eXtensible Interface (AXI)** bus of the **AMBA** specification.**

# The Advanced eXtensible Interface (AXI)

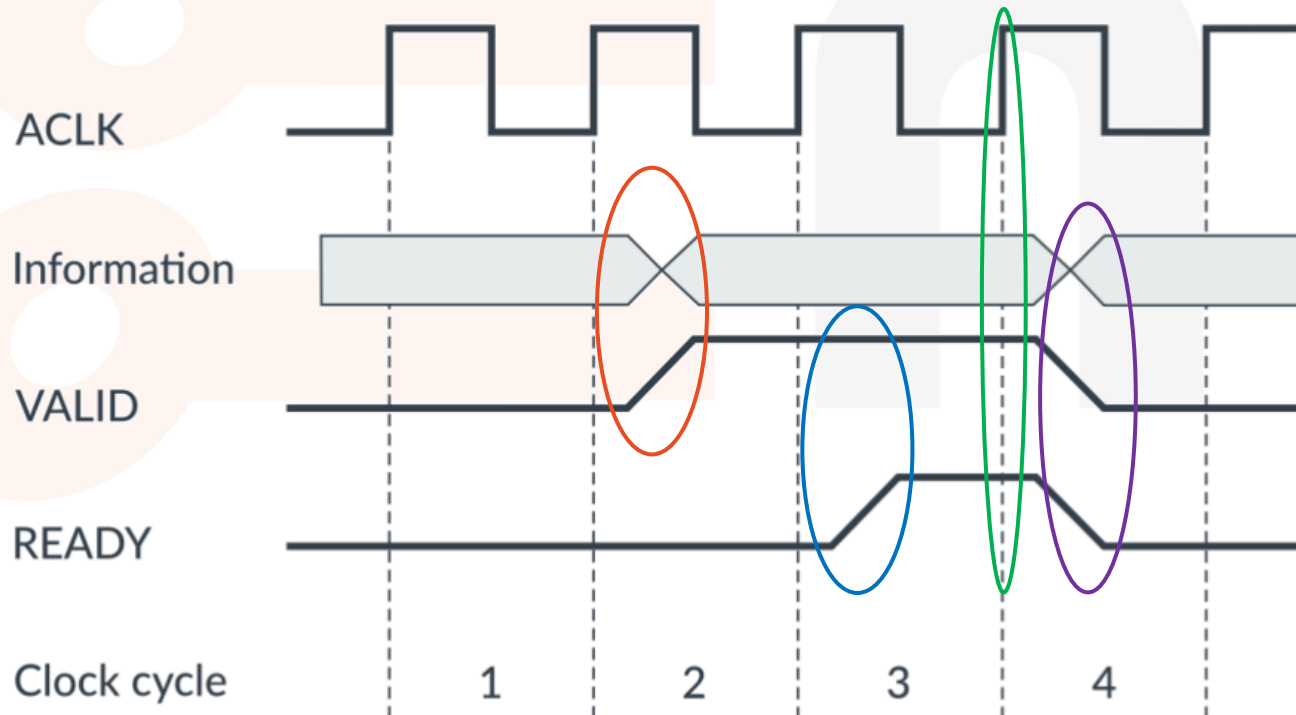
- **AXI** is an interface specification that defines the interface of IP blocks, rather than the interconnect itself.
- **AXI** supports multiple **masters (Managers)** and multiple **slaves (Subordinates)**
- **AXI** uses five main **channels** (i.e., groups of signals) for communication:
  - **Write Address (AW)**
  - **Write Data (W)**
  - **Write Response (B)**
  - **Read Address (AR)**
  - **Read Data (R)**
    - Read response is passed as part of Read Data



# Channel handshake

- All channels have **VALID** (from *source*) and **READY** (from *destination*) signals

- **VALID** remains high until **READY** signal rises.



(1) Source Information is ready.  
VALID goes high.

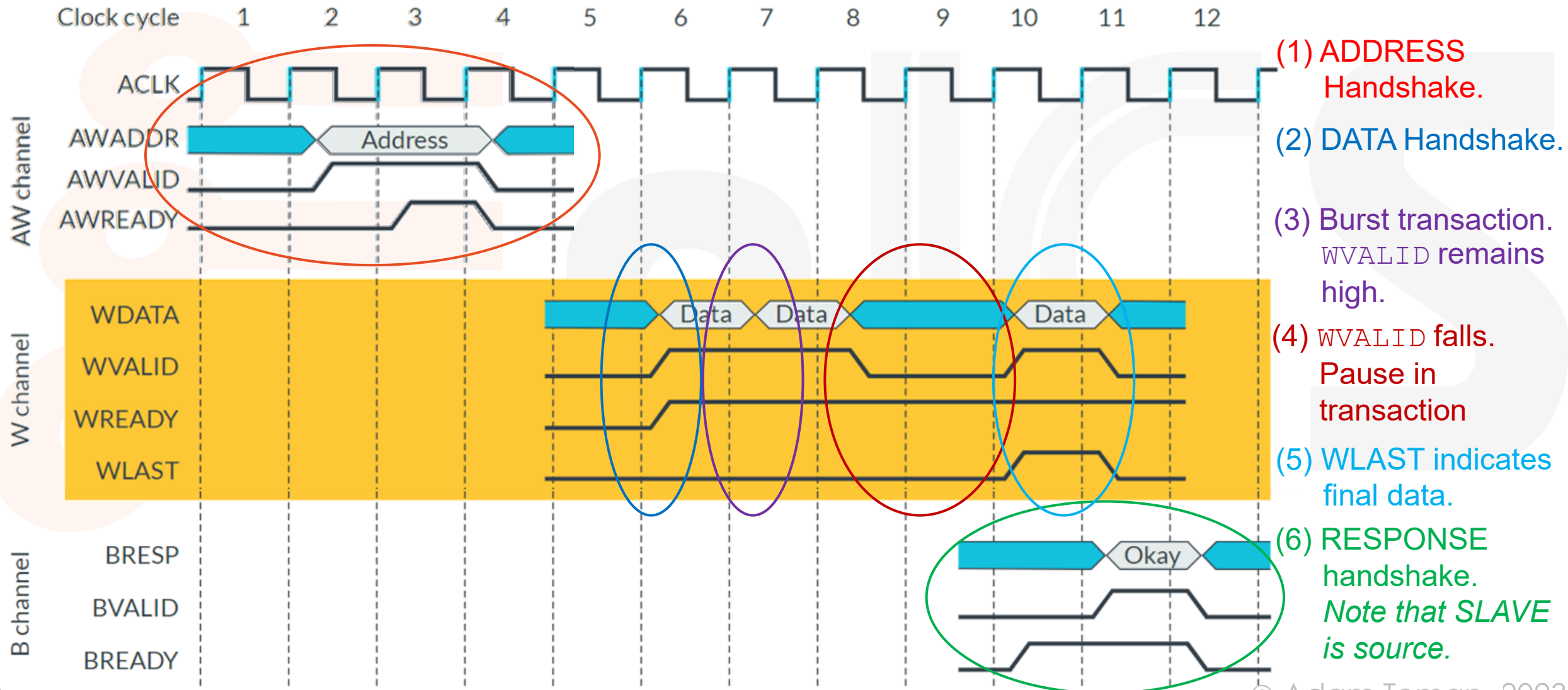
(2) Destination acknowledges  
it is ready to receive information.  
READY goes high.

(3) Information is passed from source to  
destination at rising edge of clock.

(4) Transaction is complete.  
VALID goes low. Information changed.  
READY goes low.

\* Note that **READY** can be asserted before **VALID**

# Example: Write Transaction

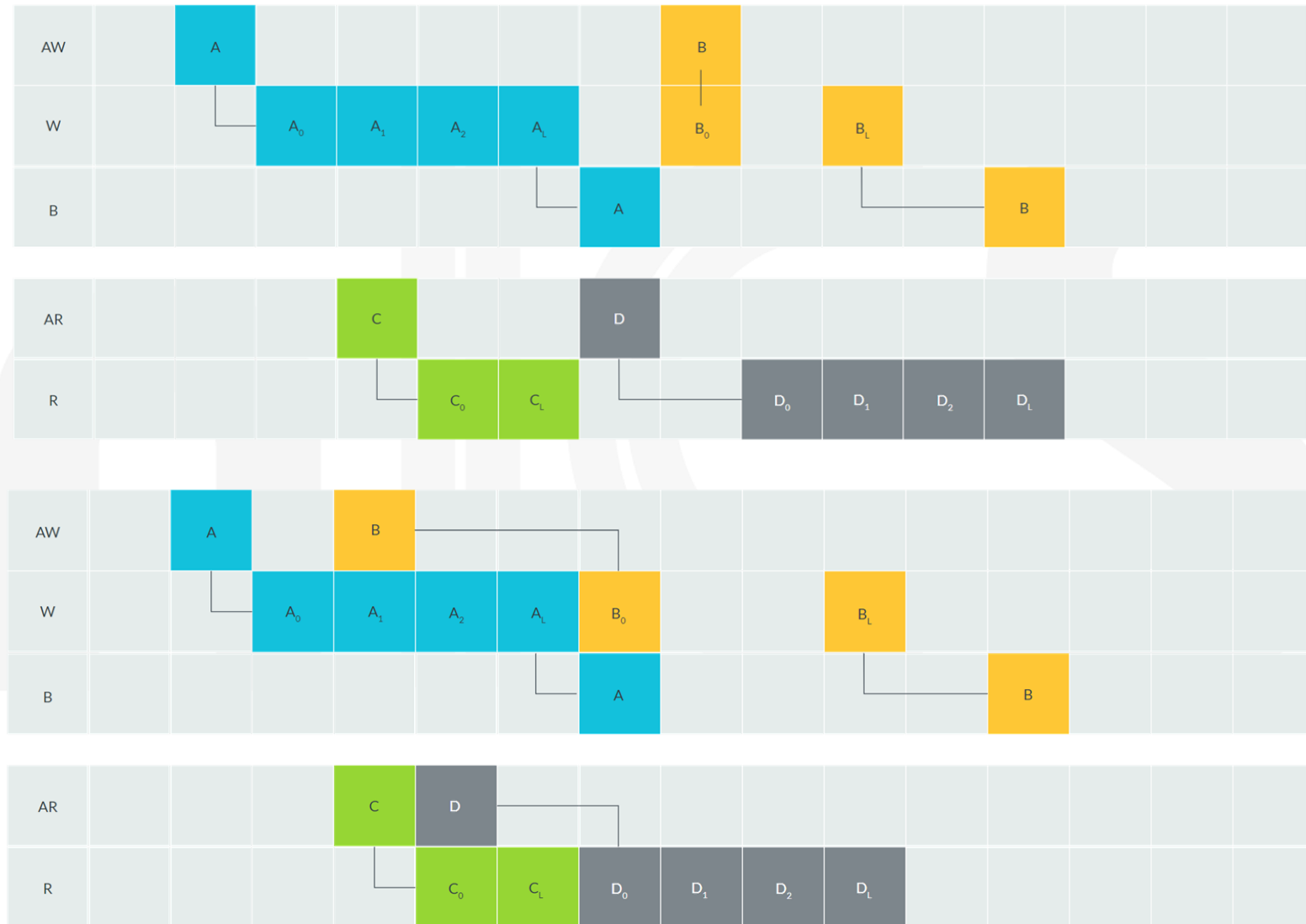


# Transaction Ordering

- **AXI Supports Interleaved/  
Out-of-Order Transactions**

- Example of a simple transaction

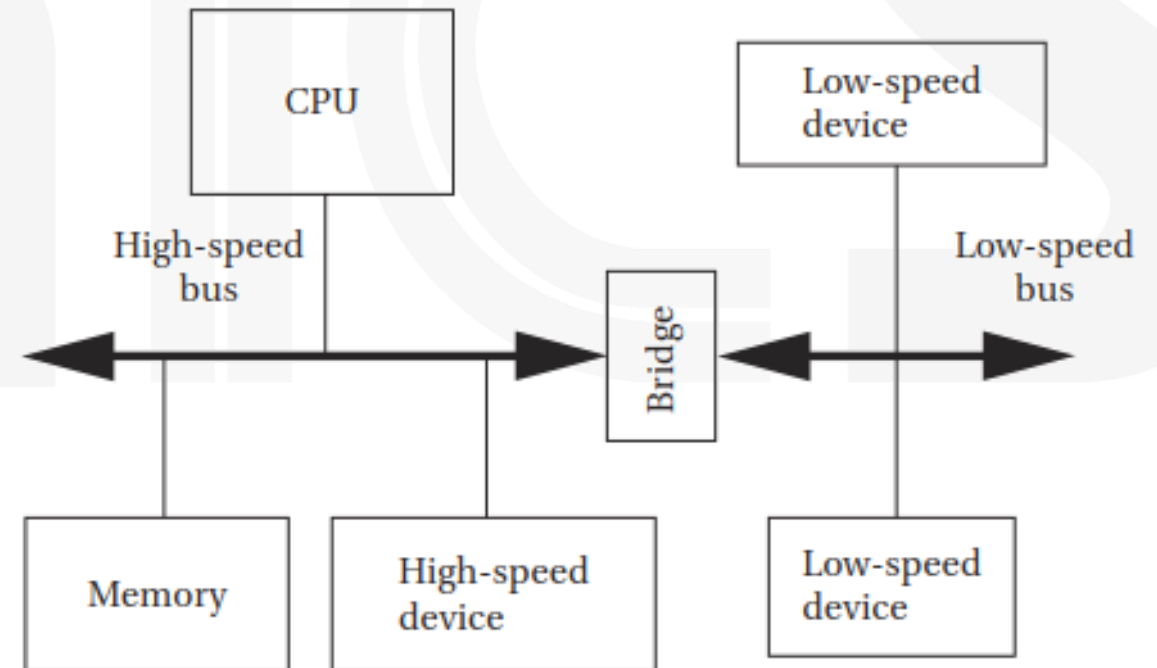
- Example of a more complex transaction



Source: ARM

# Multi-Level Buses

- A microprocessor system often has more than one bus.
  - Complexity: High speed buses are more complex (wider and implement sophisticated protocols), often not required for simple, slower devices.
  - Parallelism: Breaking up the bus can provide less contention between devices that operate independently.
- A **bridge** connects two buses:
  - Acts as a **slave** on one bus (e.g., the fast bus)
  - Acts as a **master** on the second bus (e.g., the slow bus)
  - Provides protocol translation and speed synchronization.

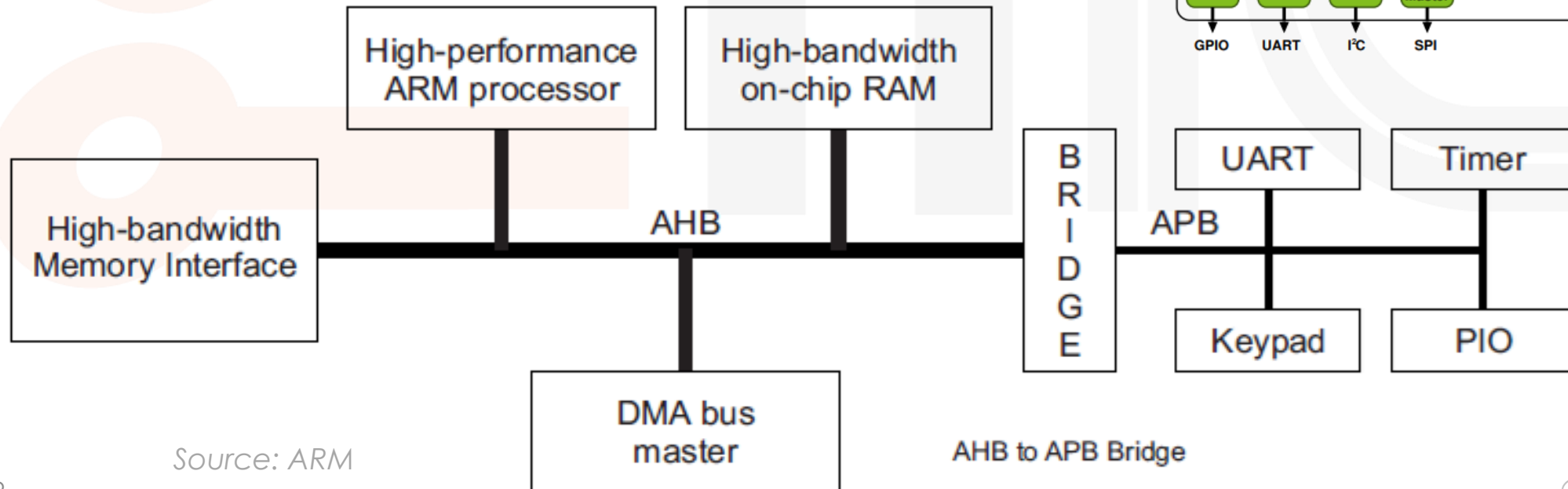


Source: Wolf,  
*Computers as Components*



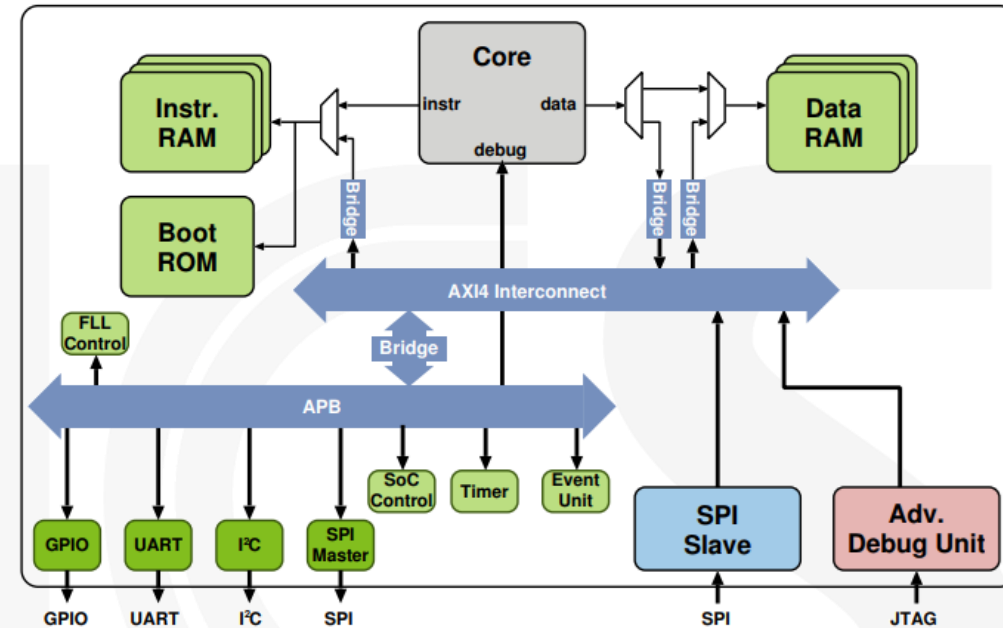
# AMBA Multi-Level Approach

- **AMBA** is designed for multi-level buses
  - Commonly use a **bridge** from a high-speed bus (e.g., **AXI**) to a low-speed bus (e.g., **APB**) to accommodate low-speed peripherals.



Source: ARM

PULPino architecture



<https://pulp-platform.org/>

# References

- **Anand Raghunathan, ECE 695R: System-on-Chip Design**
  - <https://nanohub.org/courses/ECE695R/o1a>
  - Lectures 1.7, 4.1, 4.2
- **Pasricha, Dutt, “On-Chip Communication Architectures”, 2008**
- **Flynn, Luk “Computer System Design: System-on-Chip”, 2011**
- **University of Texas, EE319K Introduction to Embedded Systems**
- **Circuits Basics “BASICS OF UART COMMUNICATION**
- **ARM AMBA Bus specifications**
- **ARM Education Kits**
- **AXI Protocol Overview,  
<https://developer.arm.com/documentation/102202/0200/AXI-protocol-overview>**