

The AMBA AXI4 Bus Architecture

Outline

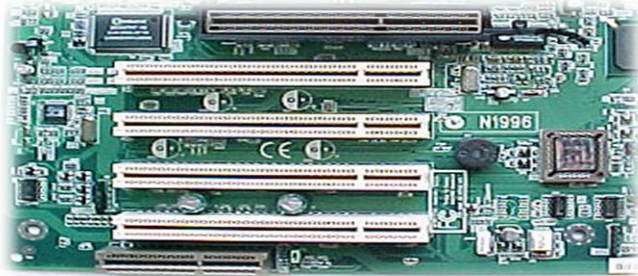
- What is a Bus
- ARM AMBA System Buses
- AXI Components and Topology
- Channel Architecture
- Channel Timing
- Resources

Outline

- What is a Bus
- ARM AMBA System Buses
- AXI Components and Topology
- Channel Architecture
- Channel Timing
- Resources

What is a Bus

- Traditionally, a bus is a communication system that allows data to be transferred between different components in a computer.
- The infrastructures is defined in both hardware and software :
 - Hardware infrastructure includes the physical implementation, such as cables or wires. For example, the PCI uses PCI cable to connect components inside a desktop.
 - Software infrastructure includes the bus protocol, e.g. PCI bus protocol.



PCI socket on a mother board



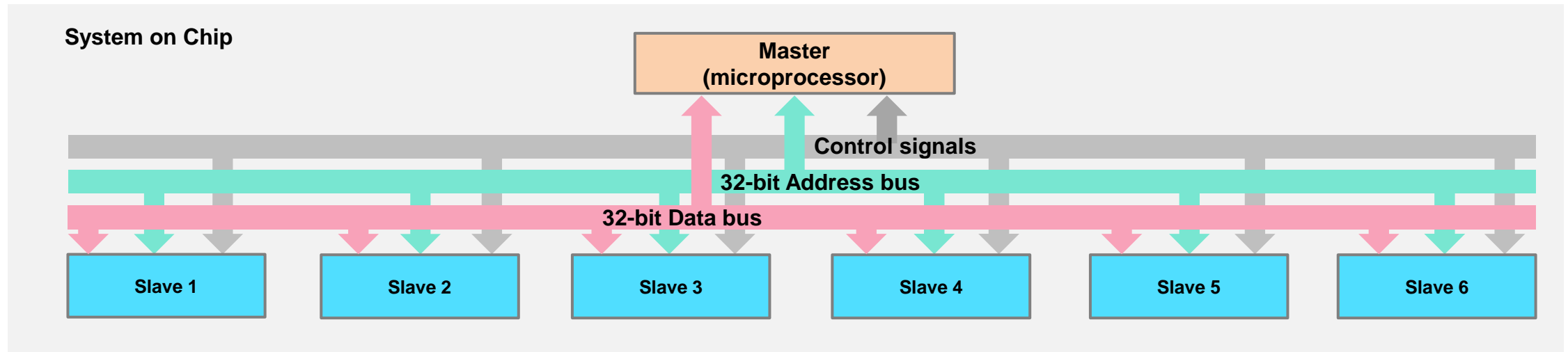
PCI bus cable

Bus Types

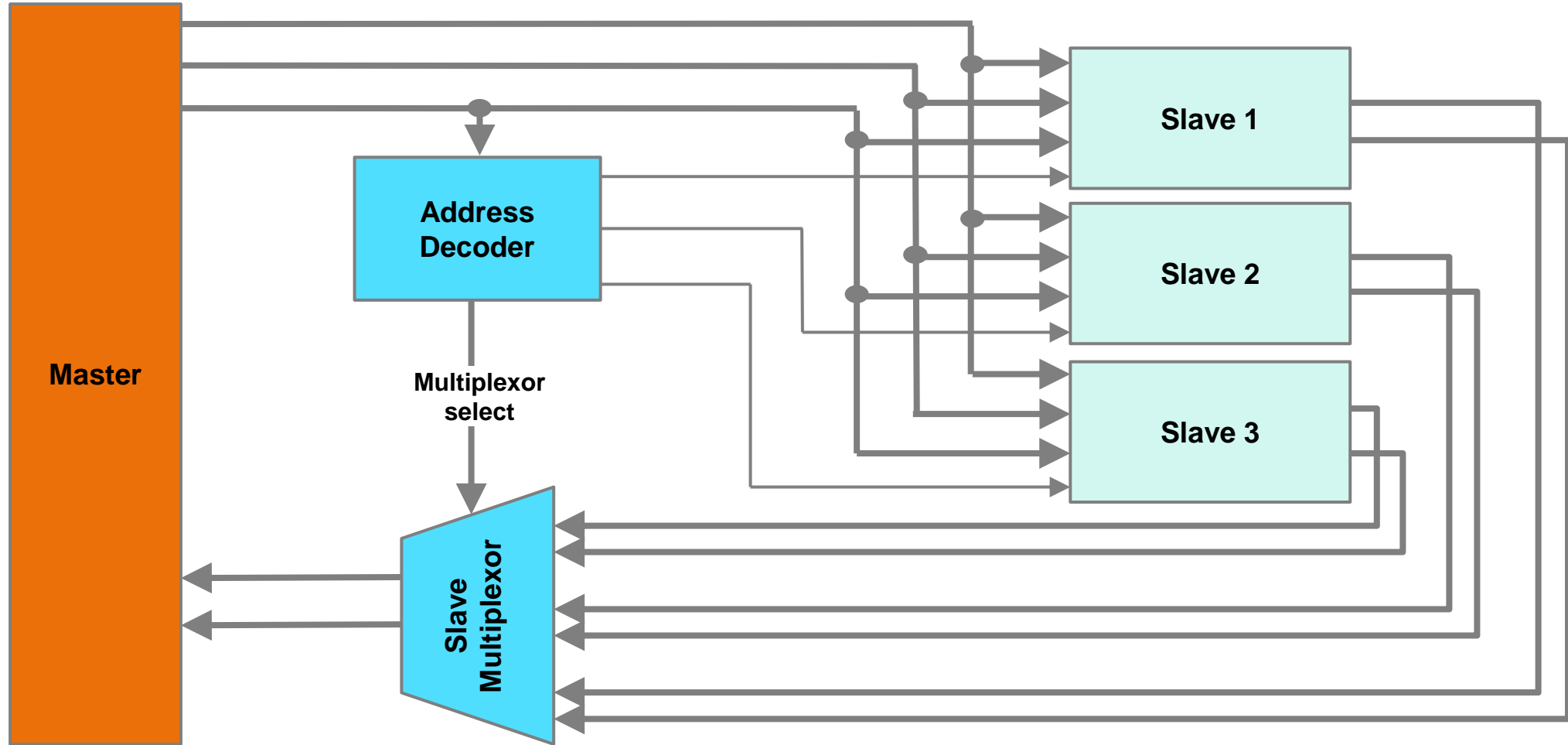
- External bus
 - Used to connect external devices, such as a computer to a printer;
- Internal bus
 - Used to connect internal components inside a computer, such as a CPU to a memory.
 - Also known as system bus.
 - Less overhead, e.g. not need for electrical characteristics handling and configuration detection etc.
 - Thus typically runs faster than the external bus.
 - In a SoC design, the internal bus is integrated onto a single chip, thus can also be referred on-chip system bus.

Bus Operation in General

- A bus typically consists of three types of signal lines:
 1. **Data bus** is used to exchange data information.
 2. **Address bus** is used to select one of the peripherals (or one register of a peripheral).
 3. **Control signals** are used to synchronize and identify transactions, such as ready, write/ read, transfer mode signals.

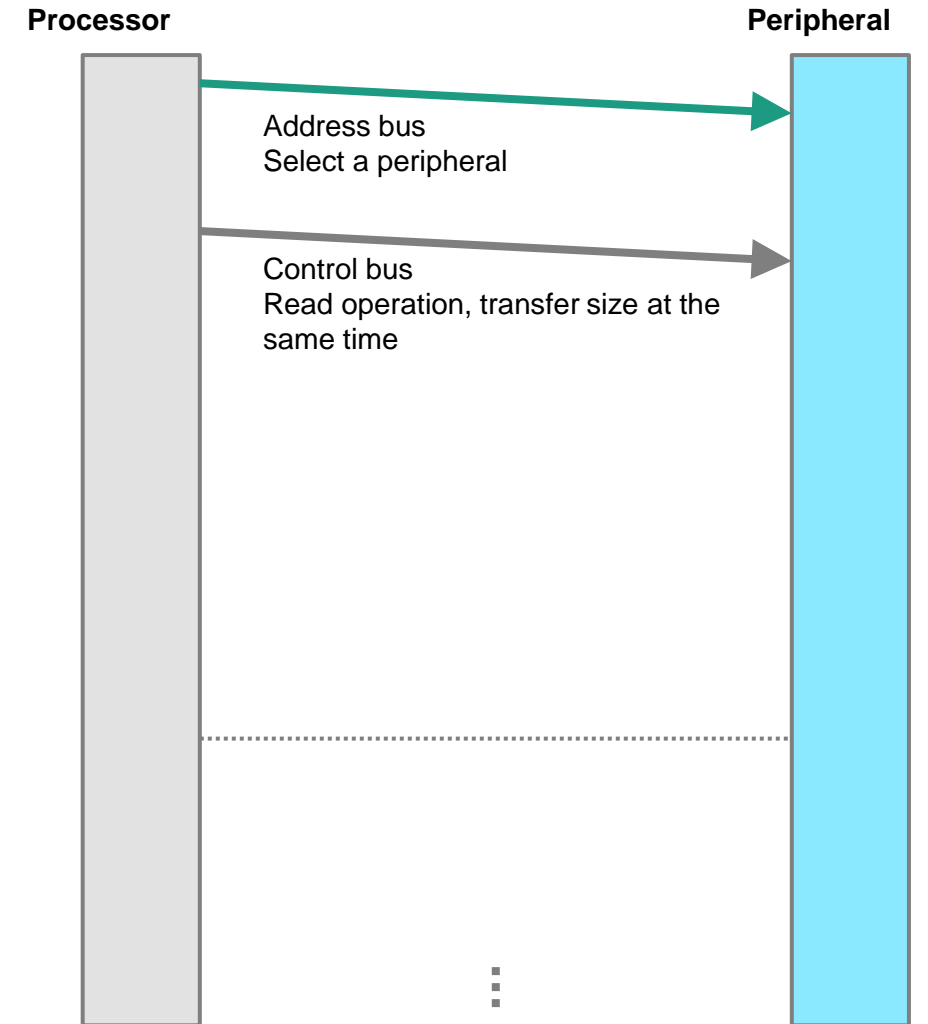


Bus Terminology



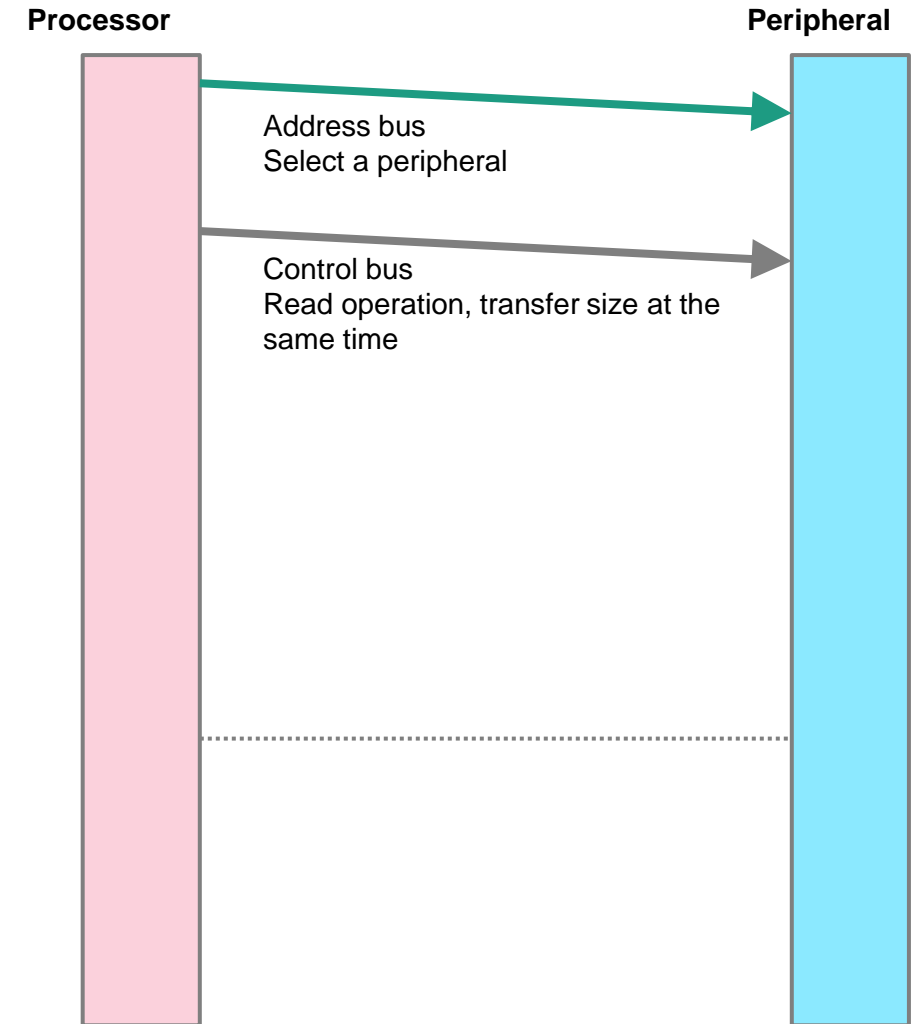
A Typical Bus Operation Example

- A typical operation to access a peripheral mainly consists of:
 - I. The master (e.g. a processor) selects one peripheral (or one register) by giving the address to the address bus. At the same time, it sets control signals, such as read or write, transfer size and so forth.



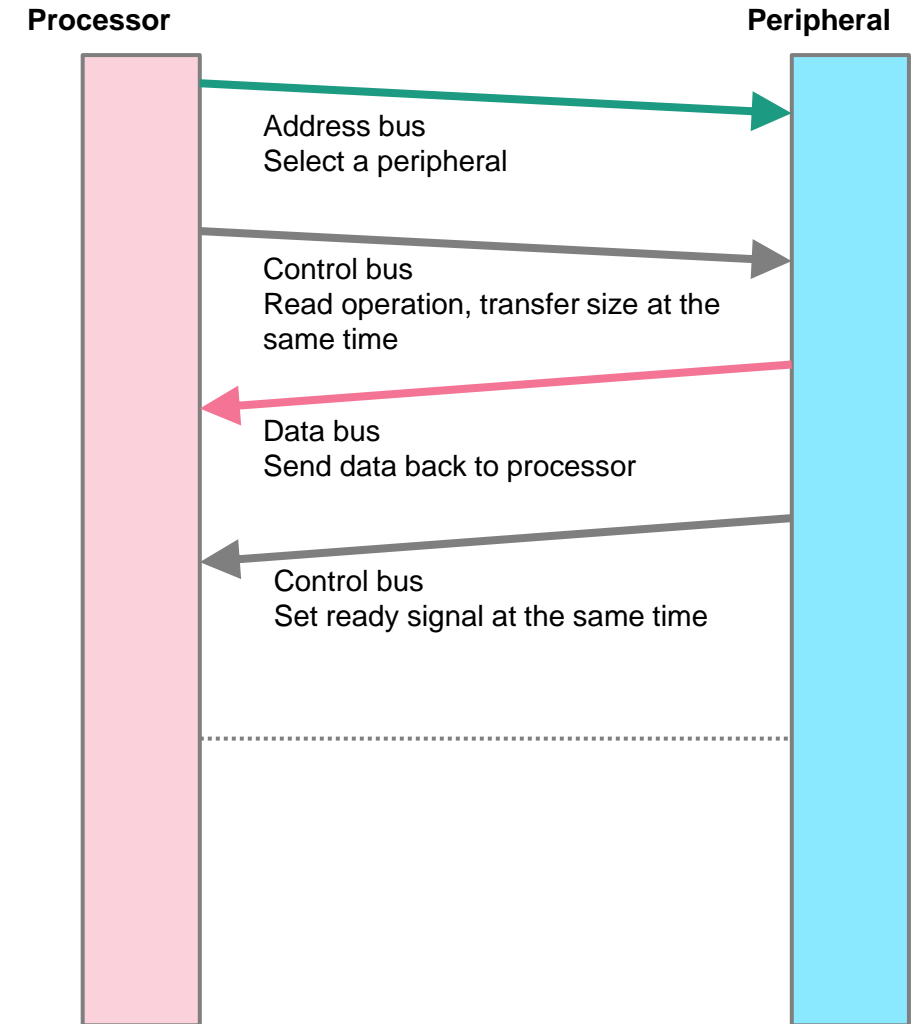
A Typical Bus Operation Example

- A typical operation to access a peripheral mainly consists of:
 1. The master (e.g. a processor) selects one peripheral (or one register) by giving the address to the address bus; At the same time, it sets control signals, such as read or write, transfer size and so forth.
 2. The master waits for the slave (e.g. peripheral) to respond.



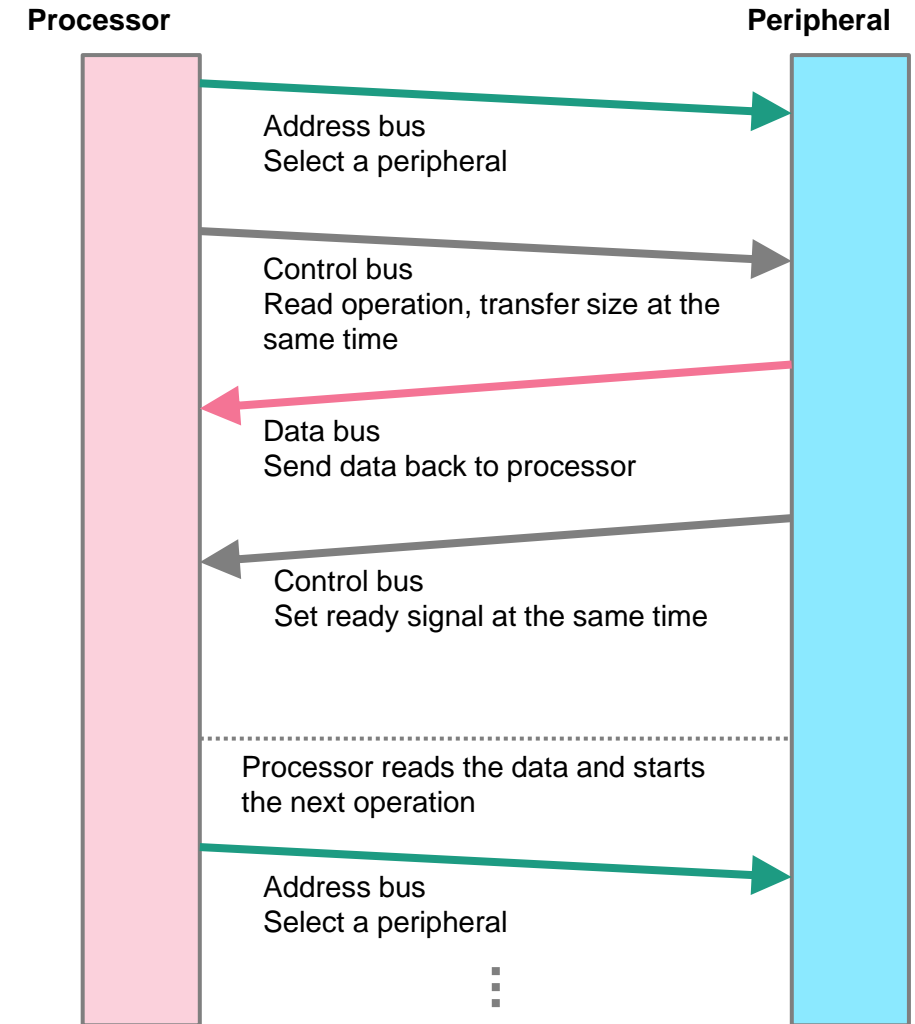
A Typical Bus Operation Example

- A typical operation to access a peripheral mainly consists of:
 1. The master (e.g. a processor) selects one peripheral (or one register) by giving the address to the address bus; At the same time, it sets control signals, such as read or write, transfer size and so forth.
 2. The master waits for the slave (e.g. peripheral) to respond.
 3. Once the slave is ready, it sends back the requested data to the processor, At the same time it sets the ready signal on the control bus.



A Typical Bus Operation Example

- A typical operation to access a peripheral mainly consists of:
 1. The master (e.g. a processor) selects one peripheral (or one register) by giving the address to the address bus. At the same time, it sets control signals, such as read or write, transfer size and so forth.
 2. The master waits for the slave (e.g. peripheral) to respond.
 3. Once the slave is ready, it sends back the requested data to the processor. At the same time it sets the ready signal on the control bus.
 4. Finally the master reads the transmitted data and start another communication cycle.

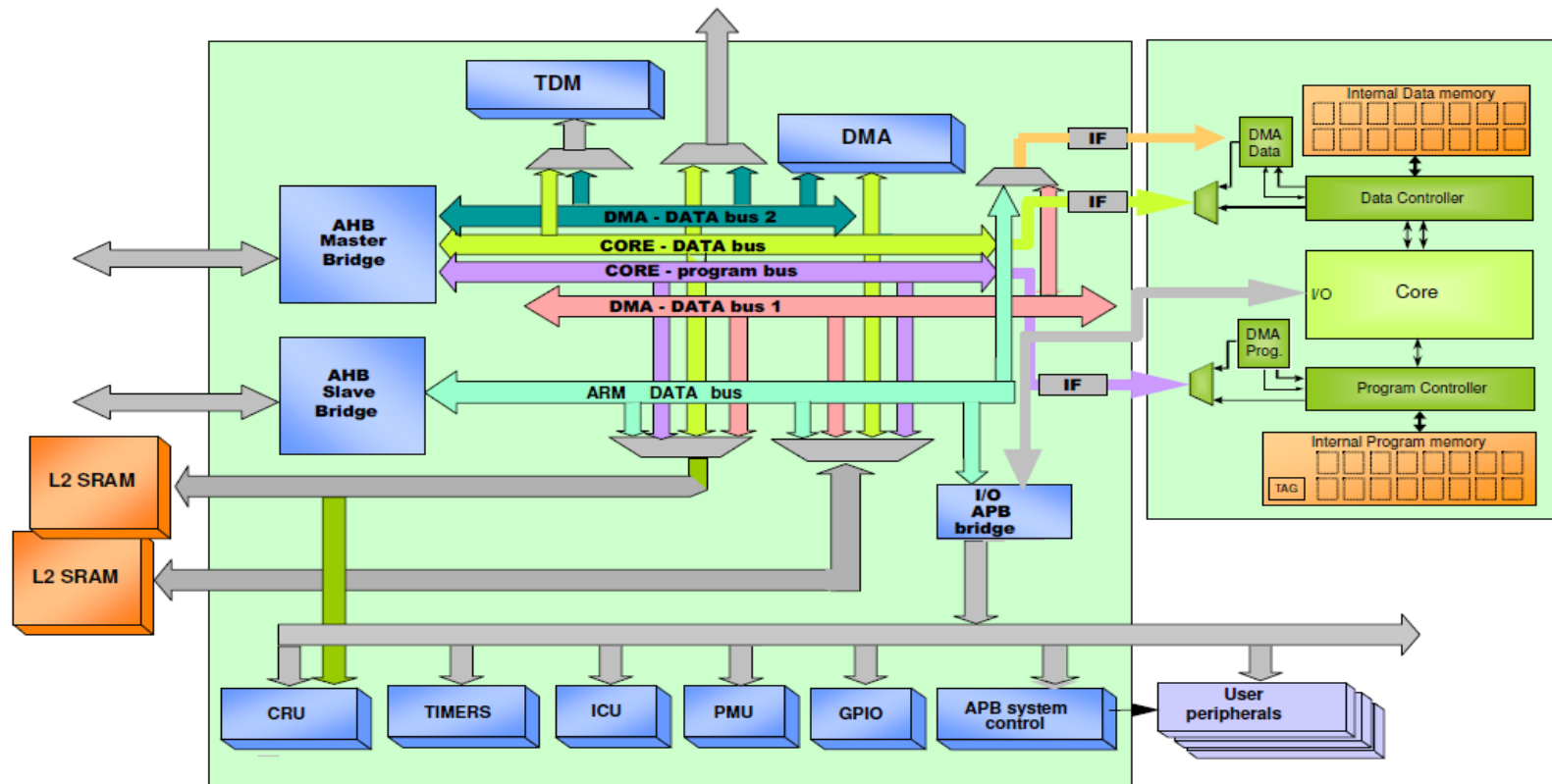


Outline

- What is a Bus
- ARM AMBA System Buses
- AXI Components and Topology
- Channel Architecture
- Channel Timing
- Resources

Communication Architecture Standards

- **Why** do we need Communication Standards?
 - Modular design approach.
 - Allows design reuse.
 - Facilitates IPs integration into a system on a chip design.



Picture source: <http://www.ecs.soton.ac.uk/> (SoC Advance design Technique)

ARM AMBA System Bus

- **AMBA: Advanced Microcontroller Bus Architecture**

- AMBA protocol is an open standard (except AMBA-5), on-chip interconnect specification.
- Used as the on-chip bus in ARM-based SoC designs.
- Provides the interface standard that enables IP re-use.
- Facilitates right-first-time development of multi-processor designs with large numbers of controllers and peripherals.
- Widely used in modern portable mobile devices, such as tablets and smartphones.



ARM AMBA Bus Families

AMBA family	Bus protocol	Processor
AMBA 5	CHI	Cortex-A57,A53
AMBA 4	ACE, ACE-Lite	Cortex-A7,A15
	AXI4, AXI4-Lite, AXI4-Stream	
AMBA 3	AXI	Cortex-A9,A8, R4, R5
	AHB (AHB-Lite)	Cortex-M0, M3, M4
	APB	Cortex-M0, M3, M4
	ATB	
AMBA 2	AHB, APB	ARM7,ARM9
AMBA 1	ASB, APB	

AMBA 3 Specifications

- **AXI - Advanced eXtensible Interface**
 - The most widespread AMBA interface.
 - Connectivity up to hundreds of Masters and Slaves in complex SoCs.
- AMBA 3 defines a set of four interface protocols:
 - AMBA 3 AXI Interface.
 - AMBA 3 AHB Interface.
 - AMBA 3 APB Interface.
 - AMBA 3 ATB Interface.
- Between them, they cover the on-chip data traffic requirements from data intensive processing components requiring:
 - High data throughput.
 - Low bandwidth communication requiring low gate count and power,
 - On-chip test and debug access.

AMBA 3 AXI Interface

- The AMBA 3 AXI interface specification provides the characteristics to support highly effective data traffic throughput.
- The five unidirectional channels with flexible relative timing between them, and multiple outstanding transactions with out-of-order data capability enable:
 - Pipelined interconnect for high speed operation.
 - Efficient bridging between frequencies for power management.
 - Simultaneous read and write transactions.
 - Efficient support of high initial latency peripherals.

AMBA 4 Specifications

- The AMBA 4 specification adds another five interface protocols to the AMBA 3 specifications:
 - ACE.
 - ACE-Lite.
 - AXI4.
 - AXI4-Lite.
 - AXI4-Stream.
- The AXI and ACE protocol specification Issue E, released February 2013, adds new optional properties for AXI ordering, ACE cache behaviour, and ARMv8 DVM messaging.

AMBA 4 Specifications

- **AXI4**

- Update to AXI3 to enhance the performance and utilization of the interconnect when used by multiple masters.
- Support for burst lengths up to 256 beats.
- Quality of Service signalling.
- Support for multiple region interfaces.

- **AXI4-Lite**

- Subset of the AXI4 protocol intended for communication with simpler, smaller control register-style interfaces in components.
- All transactions are burst length of one.
- All data accesses are the same size as the width of the data bus.
- Exclusive accesses are not supported.
- Does not support AXI IDs.

AMBA 4 Specifications

- **AXI4-Stream**

- Designed for unidirectional data transfers from master to slave with greatly reduced signal routing.
- Supports single and multiple data streams using the same set of shared wires.
- Support for multiple data widths within the same interconnect.
- Ideal for implementation in FPGA.

Outline

- What is a Bus
- ARM AMBA System Buses
- AXI Components and Topology
- Channel Architecture
- Channel Timing
- Resources

AXI Components and Topology

- **Master component**

- A component that initiates transactions.

- **Slave component**

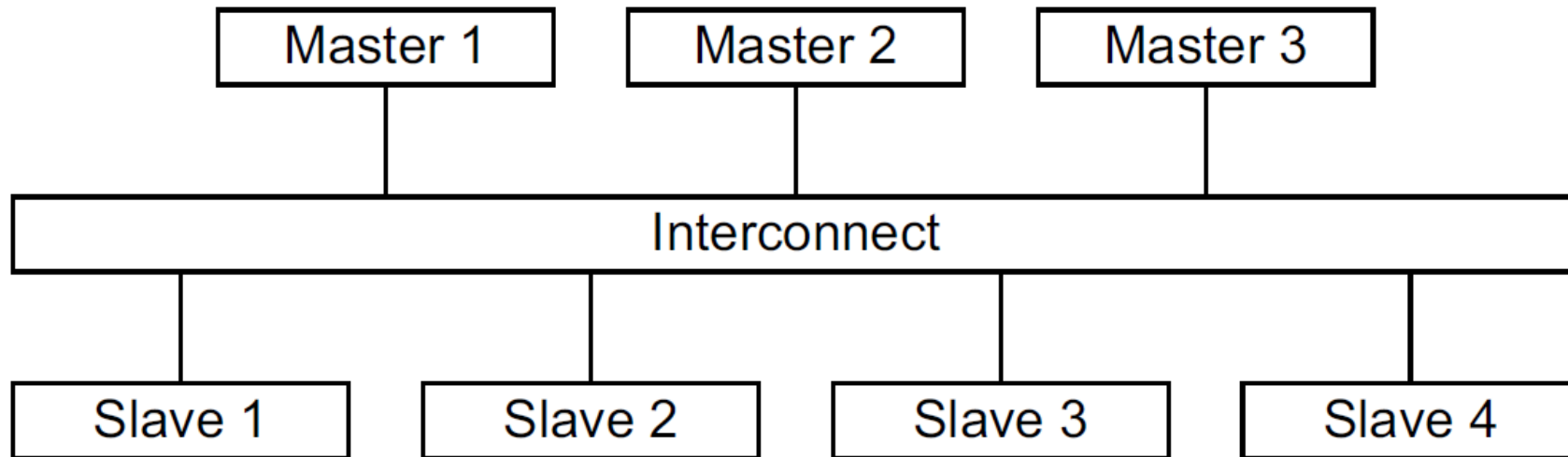
- A component that receives transactions and responds to them.
- Slave components include **Memory slave components** and **Peripheral slave components**.

- **Interconnect component**

- A component with more than one AMBA interface that connects one or more master components to one or more slave components.
- An interconnect component can be used to group together either:
 - a set of masters so that they appear as a single master interface,
 - a set of slaves so that they appear as a single slave interface.

AXI Components and Topology

- Most systems use one of three topologies:
 - shared address and data buses,
 - shared address buses and multiple data buses,
 - multilayer, with multiple address and data buses.



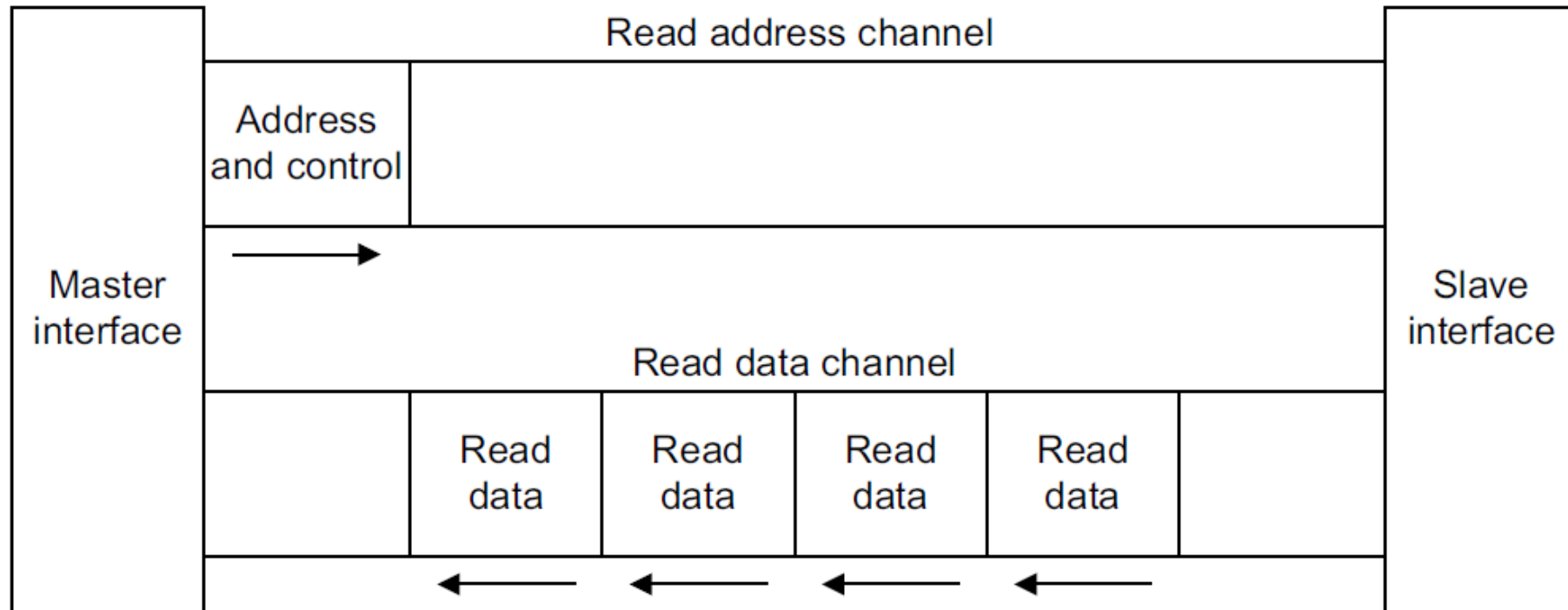
Outline

- What is a Bus
- ARM AMBA System Buses
- AXI Components and Topology
- Channel Architecture
- Channel Timing
- Resources

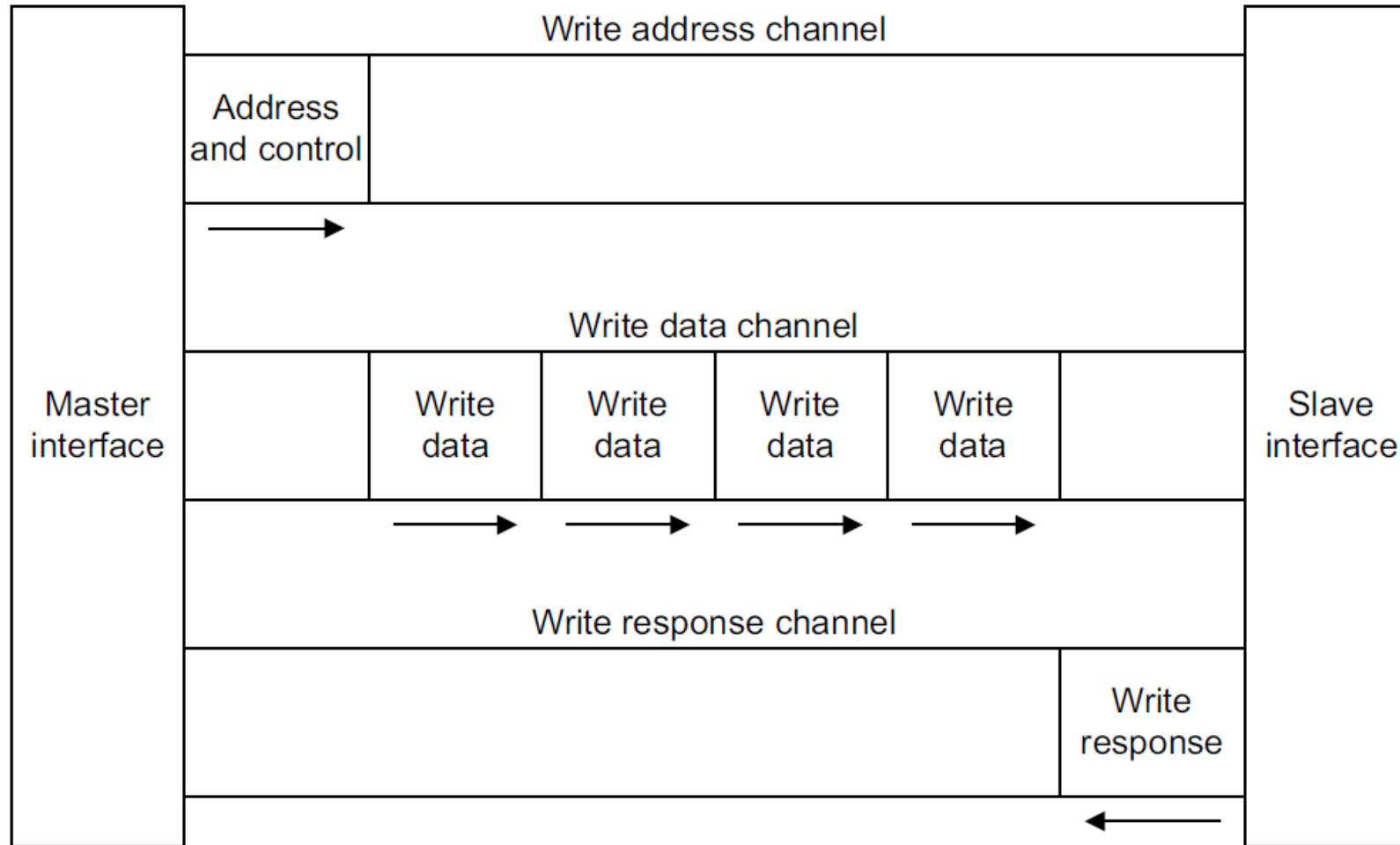
Transaction Channels

- When an AXI master initiates an AXI operation, targeting an AXI slave
 - the complete set of required operations on the AXI bus form the **AXI Transaction**
 - any required payload data is transferred as an **AXI Burst**
 - a burst can comprise multiple data transfers, or **AXI Beats**.
- The AXI protocol is burst-based and defines the following independent transaction channels:
 - read address (AR),
 - read data (R),
 - write address (AW),
 - write data (W),
 - write response (B).

Channel Architecture of Reads



Channel Architecture of Writes



Basic Signals

Signals	Read address	Read data	Write address	Write data	Write response
HANDSHAKE	ARVALID ARREADY	RVALID RREADY	AWVALID AWREADY	WVALID WREADY	BVALID BREADY
INFORMATION	ARADDR	RDATA RLAST	AWADDR	WDATA WLAST	BRESP
GLOBAL	ACLK, ARESET _n				

- A VALID signal is asserted when valid information is driven by the information transmitter.
- A READY signal is asserted when the information receiver is ready to receive.
- A LAST signal to indicate the transfer of the final data item in a transaction (data channels).

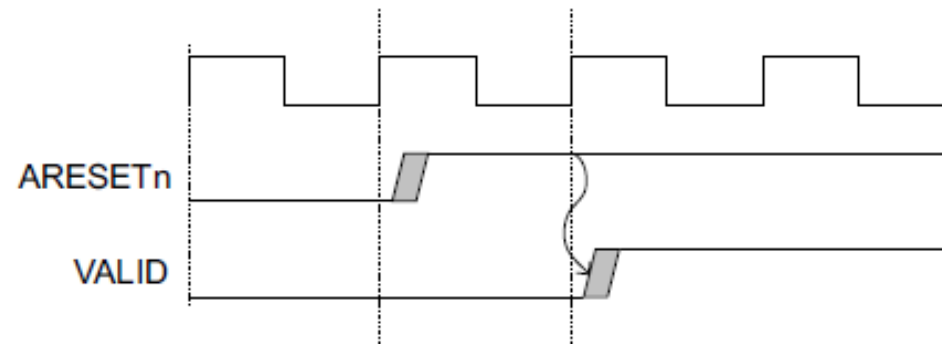
AXI Coherency Extensions (ACE) protocol

Profile	Channels	Other nets	Description
AXI3	AR+R, AW+W+B	Tag ID, WLanes	Bursts 1–16 beats
AXI4	AR+R, AW+W+B	Tag ID, WLanes, QoS	Bursts 1–256 beats
AXI4-Lite	AR+R, AW+W+B		No burst transfers. No byte lanes
AXI4-Stream	W		Simplex. No addressing. Unrestricted length
AXI ACE	All of AXI4	AC+CR+CD	Cache coherency extensions
ACE5-Lite	All of AXI4	AC+CR+CD	Single beat. Out-of-order responses

- Additional three channels to support cache consistency. Based on MESI-like protocols
- AC - Snoop address channel: input to a cached master provides the address and associated control information for snoop transactions.
 - Supports operations such as reading, cleaning or invalidating lines.
 - If the snoop hits a line in the cache, the line may have to change state.
- CR: Snoop response channel: output channel from a cached master that provides a
 - Response to a snoop transaction. Every snoop transaction has a single response associated with it.
 - The snoop response indicates whether an associated data transfer is expected on the CD channel.
- CD: Snoop data channel - optional output channel that passes snoop data out from a master.
 - Needed for a read/clean snoop transaction when the master being snooped has a copy of the data available

Clock and Reset

- Clock
 - Each AXI component uses a single clock signal, ACLK.
 - All input signals are sampled on the rising edge of ACLK.
 - All output signal changes must occur after the rising edge of ACLK.
- Reset
 - A single active LOW reset signal, ARESETn.
 - Can be asserted asynchronously, but deassertion must be synchronous with a rising edge of ACLK.

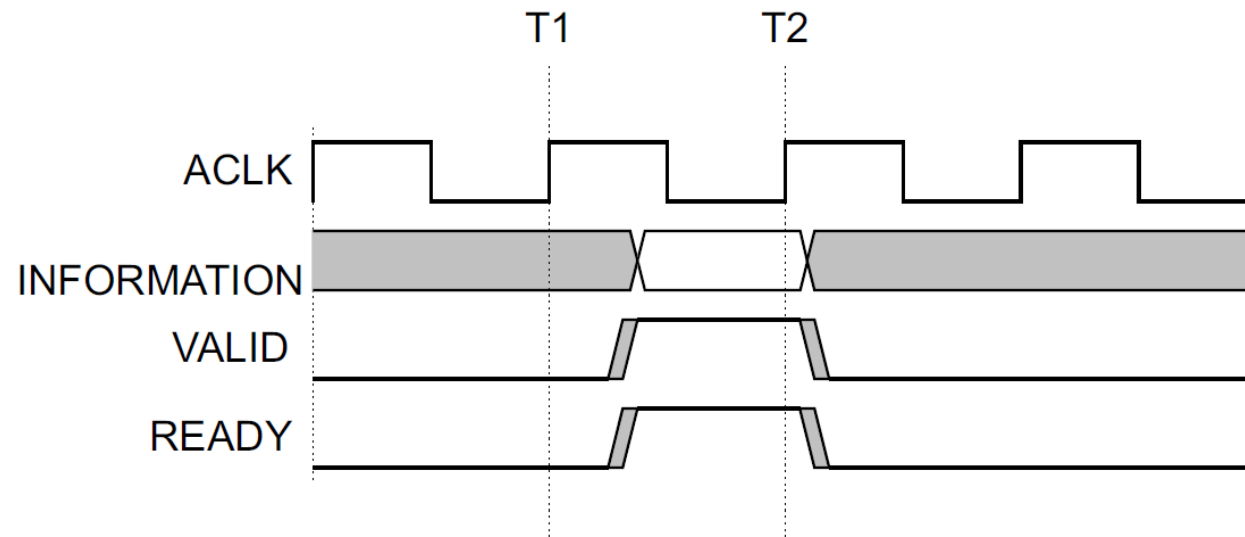


Outline

- What is a Bus
- ARM AMBA System Buses
- AXI Components and Topology
- Channel Architecture
- Channel Timing
- Resources

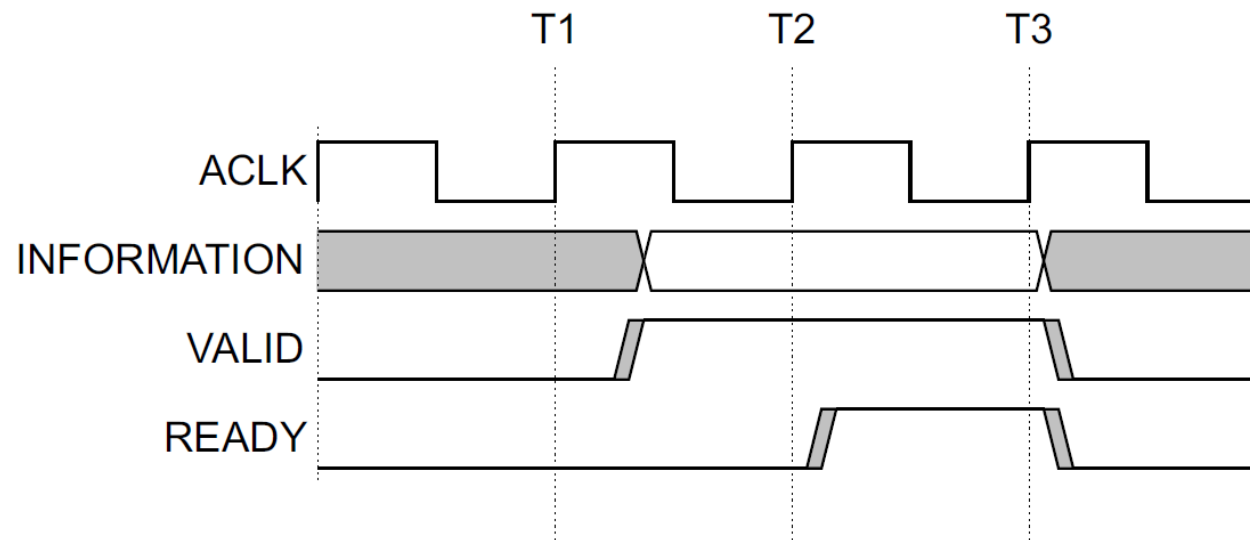
Channel Timing Example: VALID with READY handshake

- After T1, both the source and destination indicate a data transferring.
- The transfer occurs at the rising clock edge (after both VALID and READY signals are asserted)
- The transfer occurs at T2.



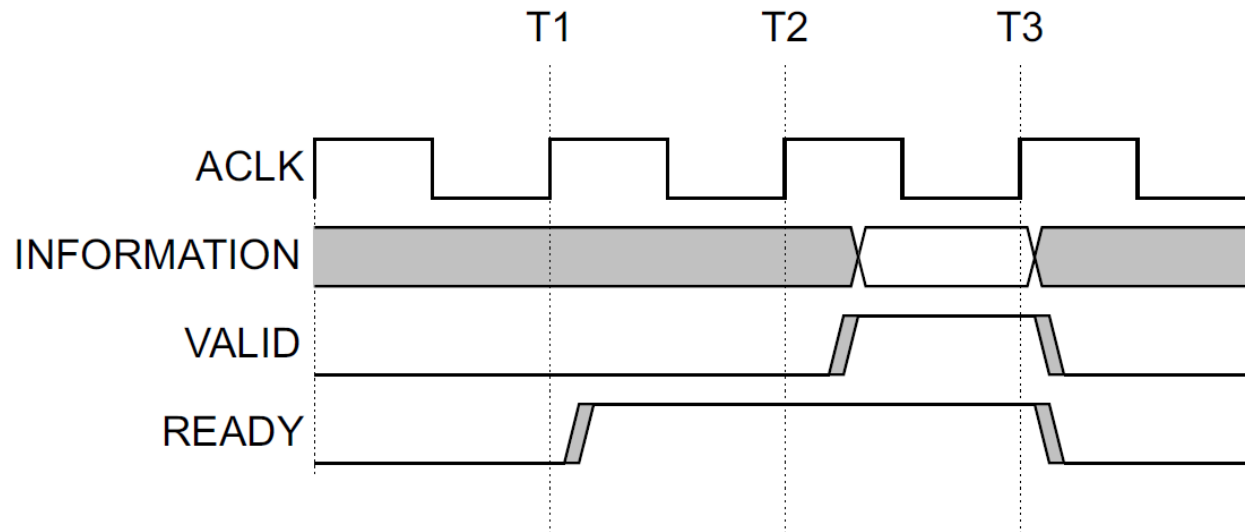
Channel Timing Example: VALID before READY handshake

- After T1, the source presents the address, data or control information and asserts the VALID signal.
- The destination asserts the READY signal after T2.
- The source has to keep its information stable until the transfer occurs at T3.



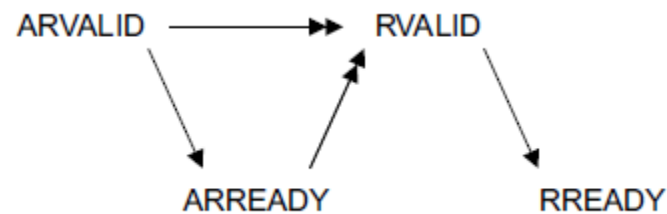
Channel Timing Example: READY before VALID handshake

- After T1, the destination asserts the READY signal (before the address, data or control information is valid) to indicate that it can accept the information.
- After T2, the source presents the information, and asserts VALID
- The transfer occurs at T3 (when this assertion is recognized)

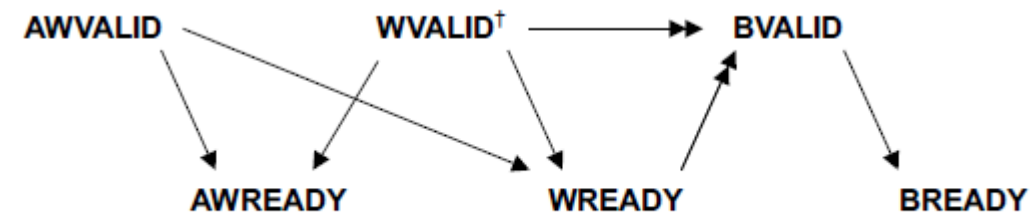


Relationships Between the Channels

- The AXI protocol requires the following relationships to be maintained:
 - A write response must always follow the last write transfer in the write transaction of which it is a part.
 - Read data must always follow the address to which the data relates.
 - Channel handshakes must conform to the dependencies defined for the handshake signals.
- Dependency rules between the handshake signals that must be observed:
 - The VALID signal of the AXI interface sending information must not be dependent on the READY signal of the AXI interface receiving that information.
 - An AXI interface that is receiving information can wait until it detects a VALID signal before it asserts its corresponding READY signal.



Read



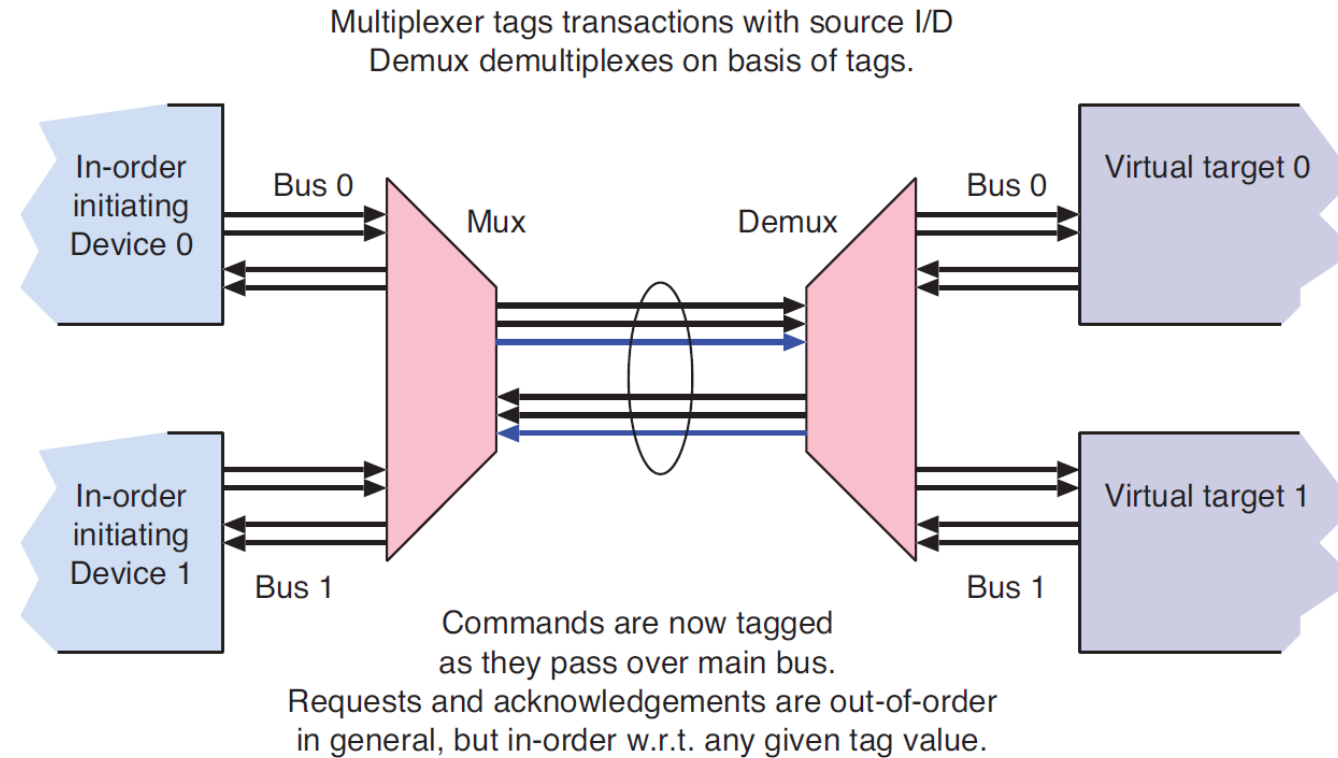
[†] Dependencies on the assertion of **WVALID** also require the assertion of **WLAST**

Write

AXI Read & Write channel

- AXI protocol has no ordering requirements between read and write transactions over the separate channels. => RaW/WaW hazards are possible.
- Same-address RaW/WaW hazards are generally handled in hardware
 - by detecting and stalling a request that is to the same address as an outstanding write
 - by serving it from the write queue.
- Sequential consistency has to be handled by the initiator with fences.
 - An initiator must wait for all outstanding responses to come back before issuing a transaction on any of its load/store ports, which needs to be after a fence.

Ordered and Unordered Interconnects

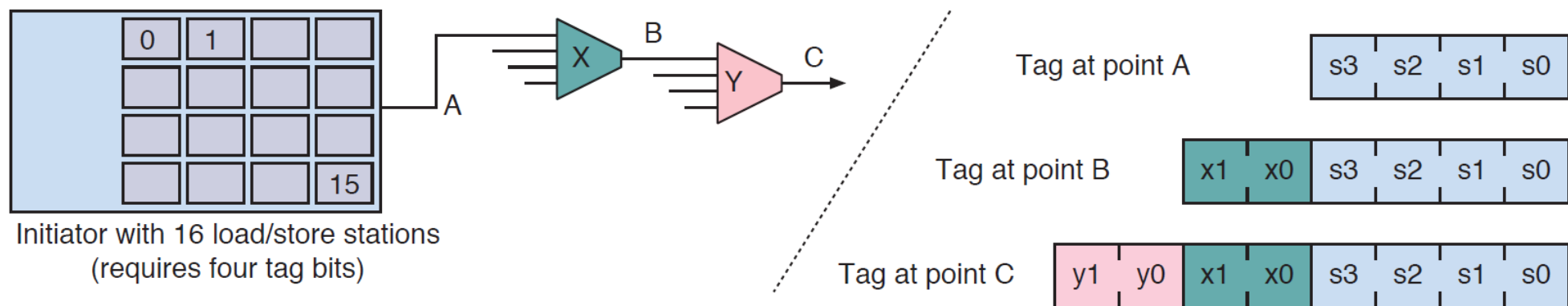


Ordered and Unordered Interconnects

- Out-of-order CPU cores and massively parallel accelerators
 - benefit in issuing multiple outstanding reads
 - can do useful work as soon as any of these are serviced
- Ordering must be controlled so that sequential consistency is preserved
- Transaction tag: positive integer that associates:
 - a command with a response
 - a group of consecutive commands with a group of consecutive responses in the same order.
 - For any given tag, the requests and replies must be kept in order
- Note, if we multiplex a pair of in-order busses onto a common bus:
 - Merge tag all of the traffic from each bus on the common bus according to its in-order initiator => we have a tagged out-of-order bus.

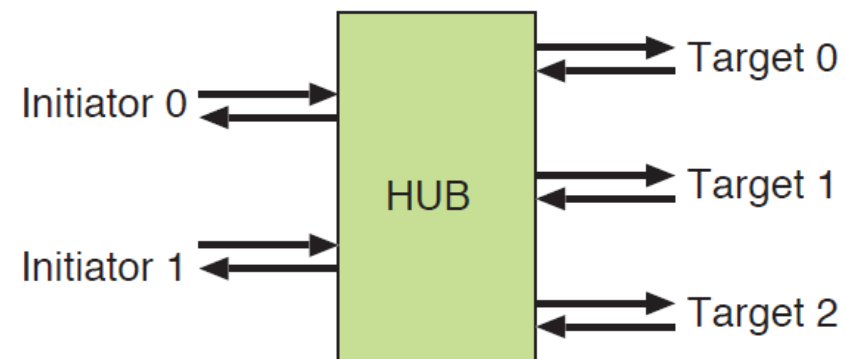
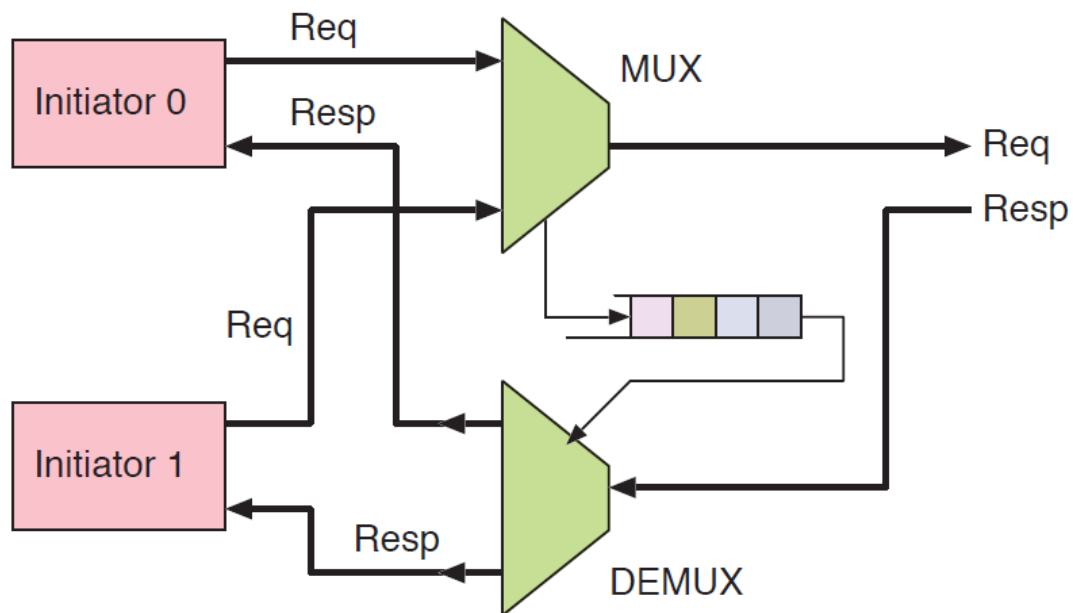
Ordered and Unordered Interconnects

- TAG Size must be large enough to:
 - distinguish between different initiators that multiplex together
 - support the maximum number of differently numbered outstanding transactions generated by an initiator
- Simplest management technique
 - for each individual source to generate tags with a width sufficient to enumerate its number of load/store stations
 - for the command tag width to be extended at each multiplexing point by concatenating the source port number with the source's tag.



Ordered and Unordered Interconnects

- In-order interconnect FIFO to enumerate transactions:



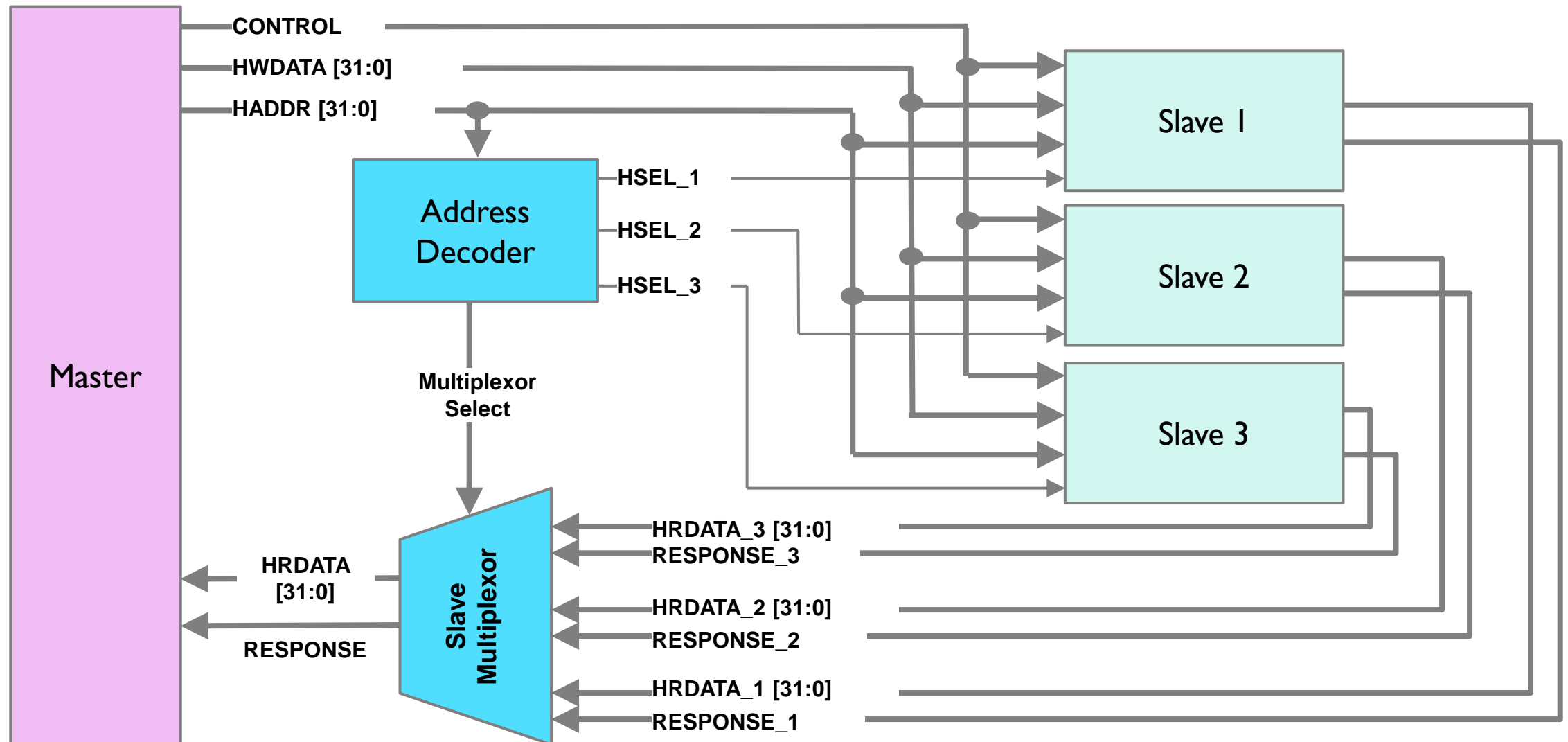
Bibliography

- AMBA4 and ACE specifications, ARM, 2014
 - <http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.set.amba/index.html>
 - (Only for registered customers).
- AMBA® AXI™ and ACE™ Protocol Specification, ARM, 2011
 - https://capocaccia.ethz.ch/capo/raw-attachment/wiki/2014/microblaze14/AXI4_specification.pdf

AMBA 3 AHB-Lite Bus

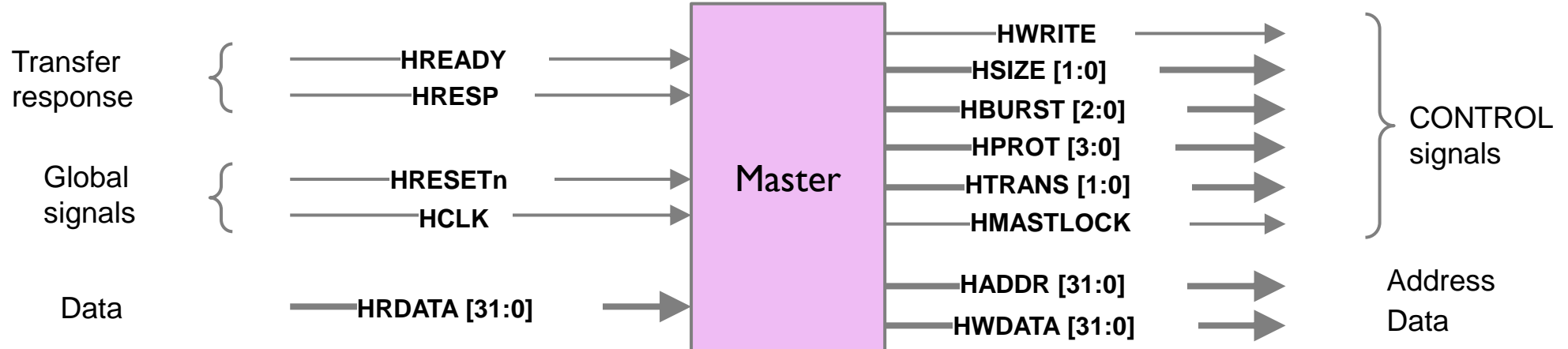
- AHB
 - High-performance synthesizable designs
 - Supports multiple bus masters
 - Provides high-bandwidth operation
- AHB-Lite:
 - A subset of AHB
 - Simplifies the design of AHB bus, e.g., typically with a single master

AHB-Lite Bus Block Diagram



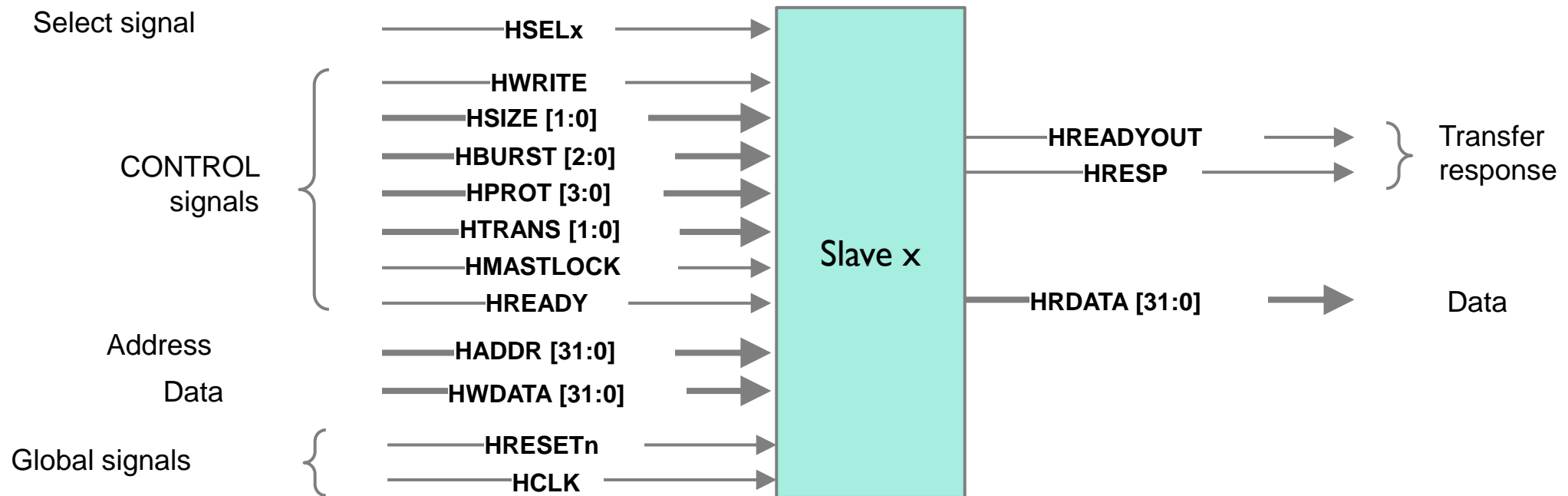
AHB-Lite Master Interface

- The AHB-Lite master provides address and control information to initiate read and write operations.
- The master also receives the response from the slave, including data and ready and response signal.



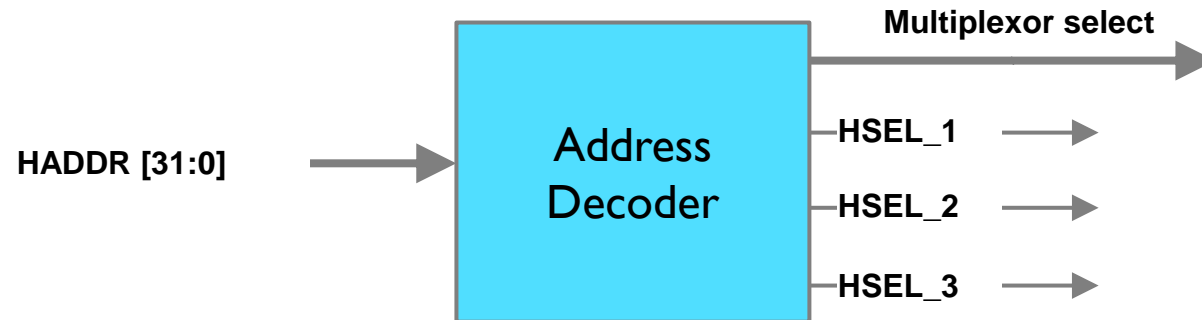
AHB-Lite Slave Interface

- An AHB-Lite slave responds to transfer initiated by the master in the system.
- The signal HSELx is the output from the address decoder, which is used to select one slave at a time.



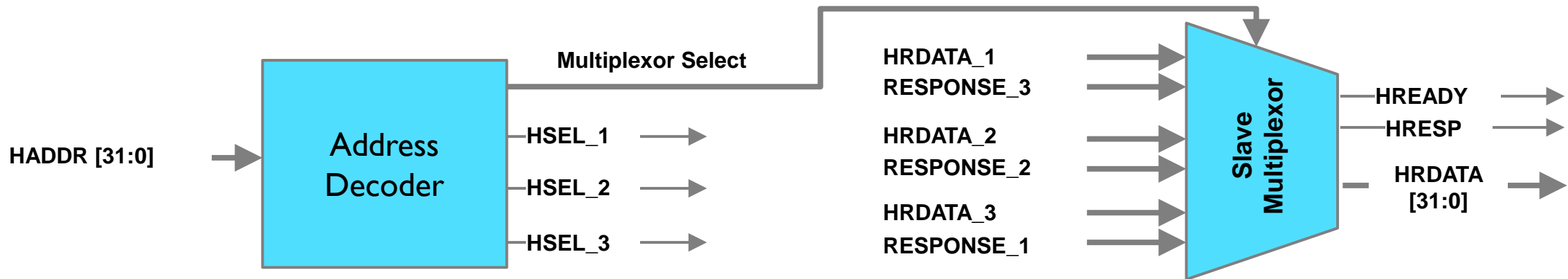
Address Decoder

- Address decoder
 - Selects one of the slaves depending on the current address bus
 - Also informs the slave multiplexor



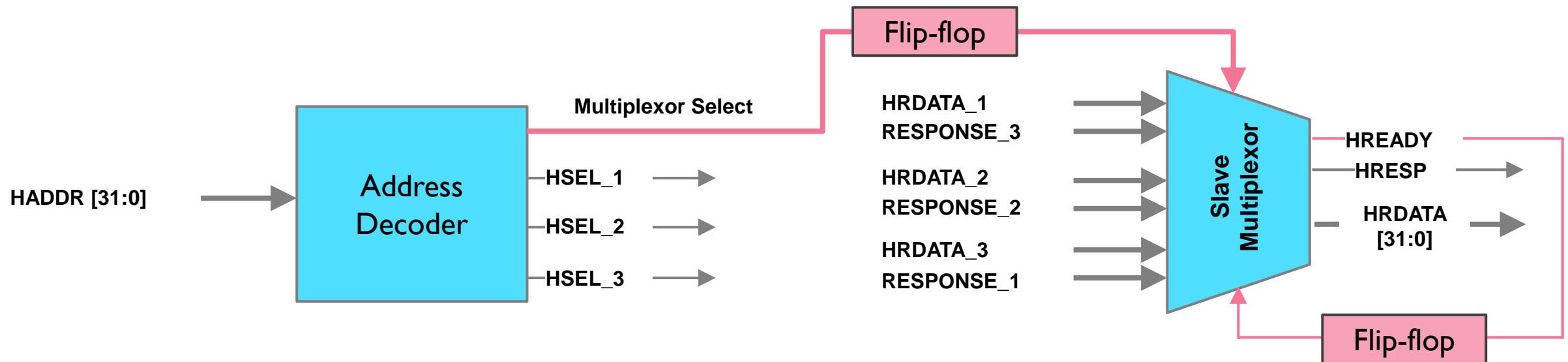
Slave Multiplexor

- Slave multiplexor
 - Inputs the response signals (HRDATA, HREADY, and HRESP) from all the slaves, and outputs one of them depending on the selecting signal from the address decoder.



Hardware Implementation

- Due to the pipelined operation, some signals have to be deliberately delayed:
 - The selecting signals from the decoder to the multiplexor are delayed for one clock cycle.
 - The HREADY signal is delayed for one clock cycle before it is fed back to the multiplexor.
- The detailed implementation can be referred from the code that is provided in the EDK.



AHB-Lite Operation Principles

- AHB-Lite supports three types of transfers:
 - Single
 - Incrementing bursts that do not wrap at address boundaries
 - Wrapping bursts that wrap at particular address boundaries

AHB-Lite Operation Principles

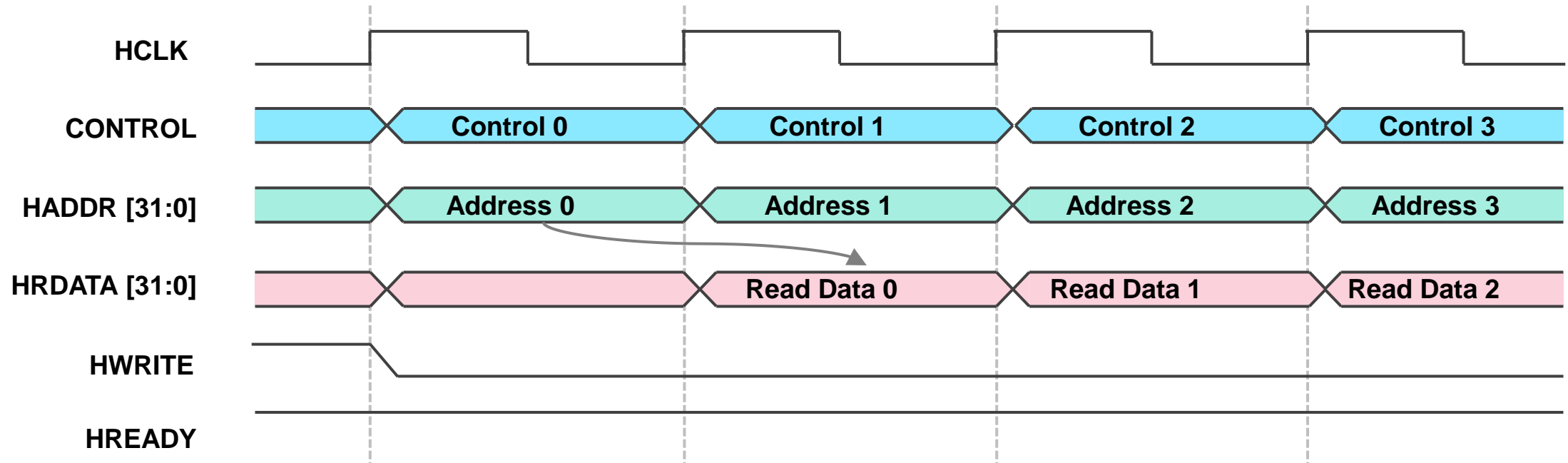
- An AHB-Lite transfer consists of two phases:
 - The address phase, which lasts for a single HCLK cycle unless it is extended by the previous bus transfer.
 - The data phase might require several HCLK cycles. The HREADY signal is used to control the number of clock cycles required to complete the transfer.

AHB-Lite Bus Timing

- This module focuses on the basic bus operation, so we assume the following:
 - No BURST transaction: HBURST[2:0] is always 3'b000.
 - Never generates locked transactions: HMASTLOCK is always 1'b0.
 - All transactions issued are non-sequential transfer: HTRANS[1:0] is either 2'b00 (IDLE) or 2'b10 (non-sequential).

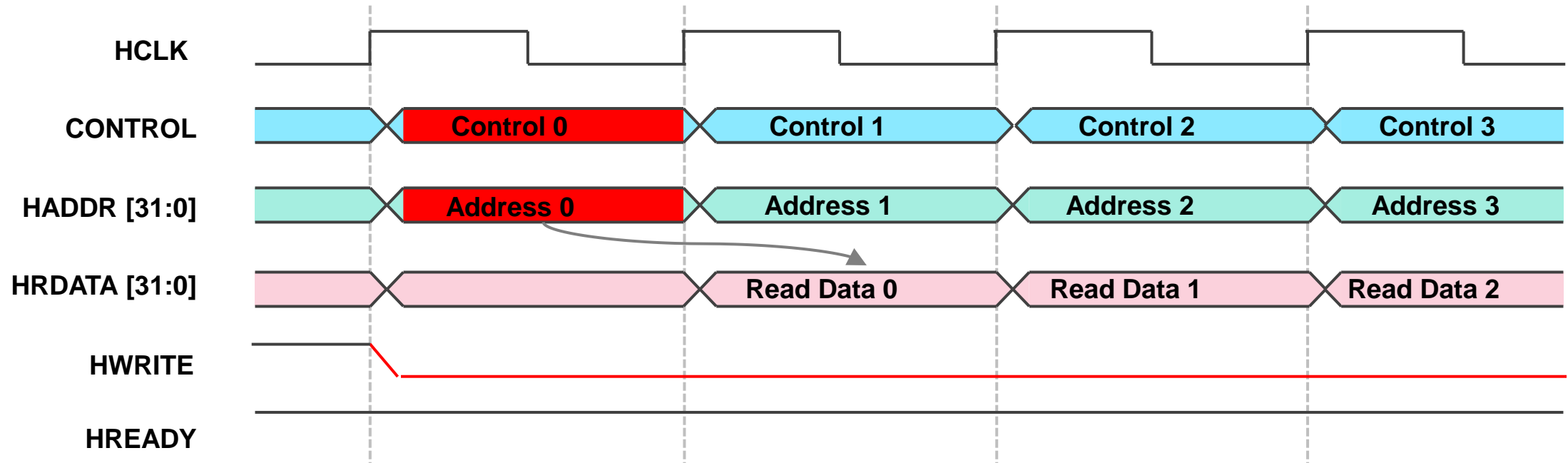
Basic Read Transfer

- Consider a simple read transfer with no wait states:



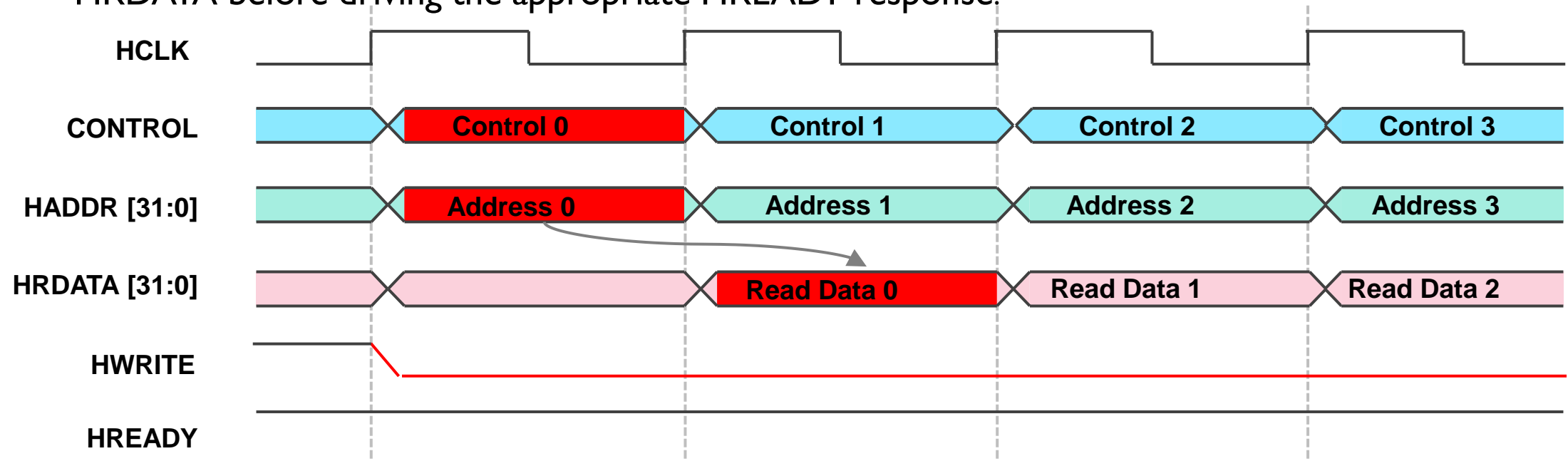
Basic Read Transfer

- Consider a simple read transfer with no wait states:
 - The address phase: The master drives the address and control signals onto the bus after the rising edge of HCLK.



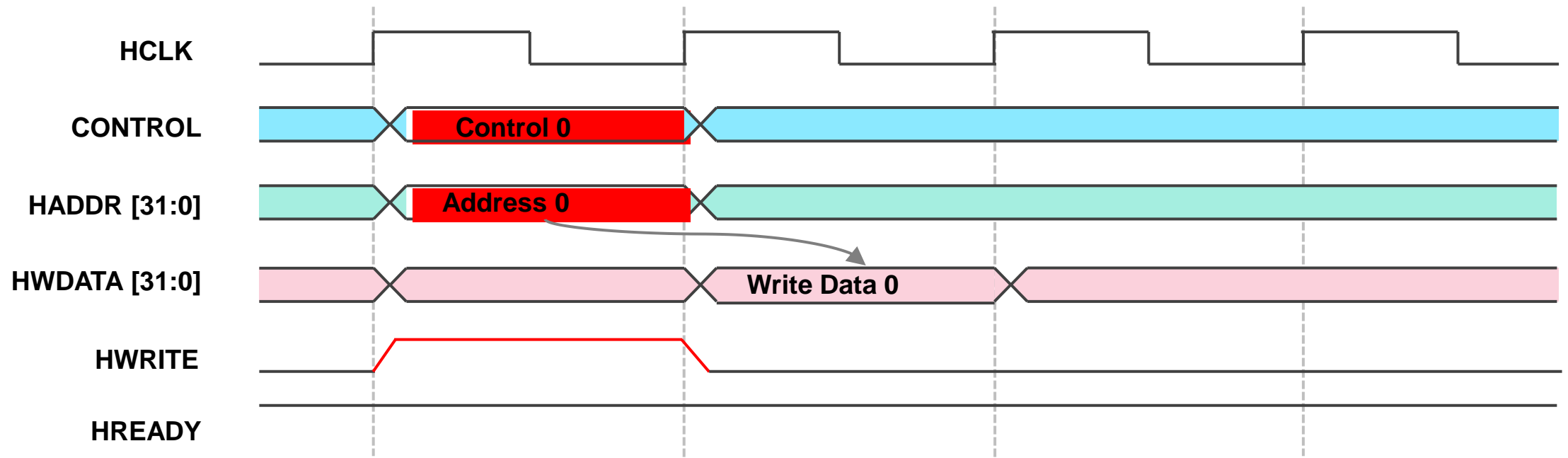
Basic Read Transfer

- Consider a simple read transfer with no wait states:
 - The address phase: The master drives the address and control signals onto the bus after the rising edge of HCLK.
 - The data phase: The slave samples the address and control information and make data available at HRDATA before driving the appropriate HREADY response.



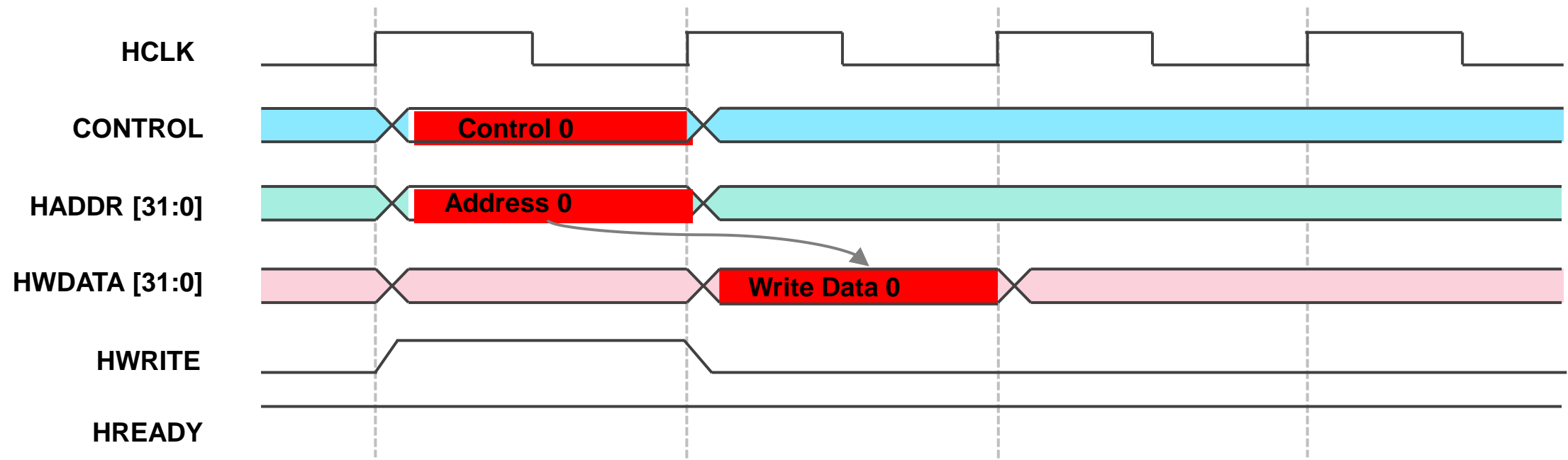
Basic Write Transfer

- Consider a simple write transfer with no wait states:
 - The address phase: The master drives the address and control signals onto the bus after the rising edge of HCLK and sets HWRITE to one.
 - The data phase: The slave samples the address and control information and make data available at HRDATA before driving the appropriate HREADY response.



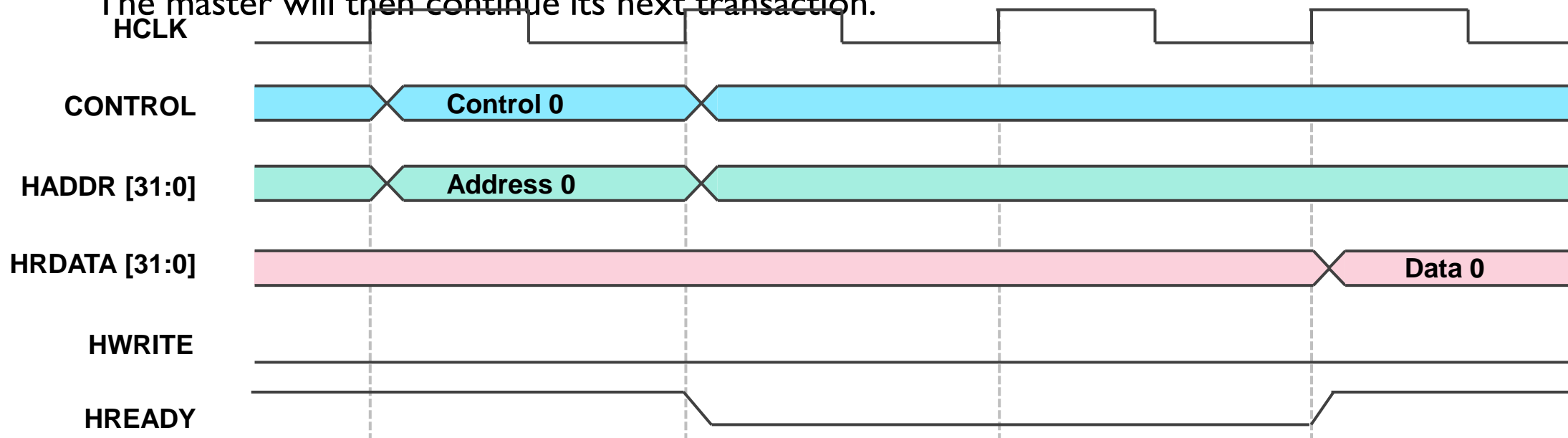
Basic Write Transfer

- Consider a simple write transfer with no wait states:
 - The address phase: The master drives the address and control signals onto the bus after the rising edge of HCLK and sets HWRITE to one.
 - The data phase: The slave samples the address and control information and make data available at HRDATA before driving the appropriate HREADY response.



Read Transfer with Wait State

- Address phase (first clock cycle)
 - Give address and control signals; set HWRITE to one.
- Data phase (multiple clock cycles)
 - The slave holds HREADY to zero if it is not ready to provide its data; the master delays its next transaction.
 - When the slave is ready, the data will be given at HRDATA; at the same time, HREADY is set to one. The master will then continue its next transaction.



Write Transfer with Wait State

- Address phase (first clock cycle)
 - Give address and control signals; clear HWRITE to zero.
- Data phase (multiple clock cycles)
 - The master gives its data at HWDATA. The slave holds HREADY to zero if it is not ready to receive the data; the master delays its next transaction.
 - When the slave is ready, it will receive the data and set HREADY to one. The master will then continue its next transaction.

