The AMBA AXI4 Bus Architecture

Outline

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

ARM

Outline

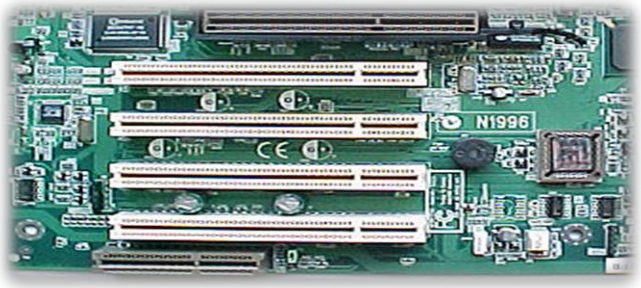
- What is a Bus
- ARM AMBA System Buses
- AXI Components and Topology
- Channel Architecture
- Channel Timing
- Resources

ARM

What is a Bus

In passato i bus consentivano di connettere periferiche con la cpu ed erano intese come linee fisiche alle quali erano conessi diversi dispositivi. Erano connessioni che sfruttavano la logica theeestate ovvero facevano in modo che tutti i componenti connessi a quei fili potessero orchestrare la comunicazione in diversi versi. In un certo momento qualcuno iniziava a parlare e tutti gli altri ascoltavano.

- 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

ARM

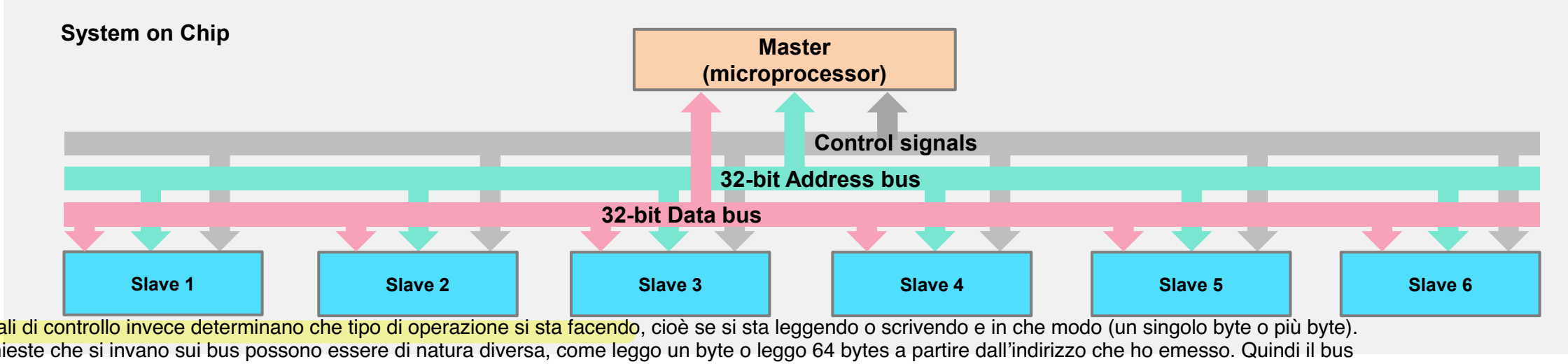
Bus Types

- 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.

Cosa gira sui bus? Dati (in lettura e scrittura) , indirizzi fisici (che ci permettono di identificare il destinatario). Fin ora abbiamo visto indirizzi che venivano emessi dalle istruzioni di load store fossero locazioni di memoria, ma in realtà nello spazio di indirizzamento ci possiamo trovare con alcune regioni di indirizzi che mappano ad esempio registri interni di una specifica periferica. Può succedere che a determinati indirizzi corrispondono zone di memoria ma anche a periferiche.

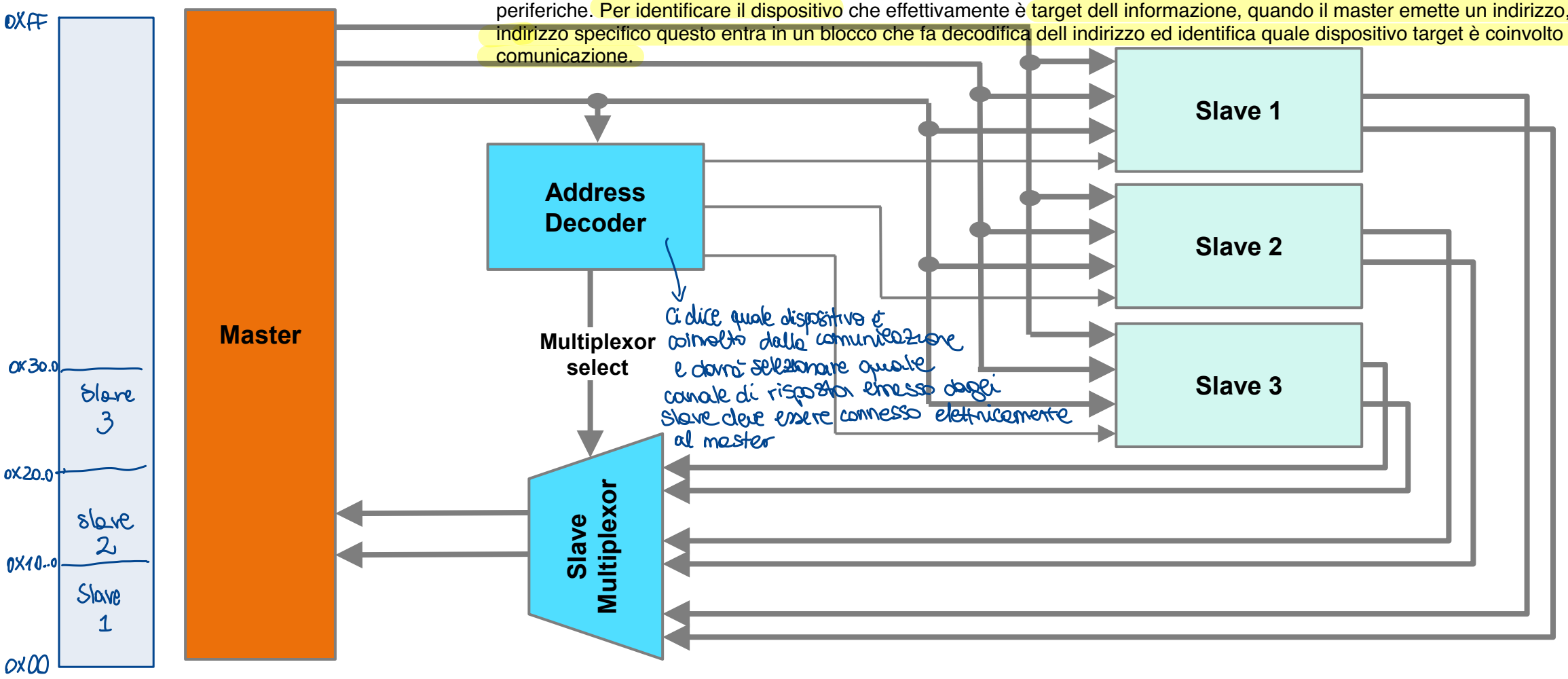


I segnali di controllo invece determinano che tipo di operazione si sta facendo, cioè se si sta leggendo o scrivendo e in che modo (un singolo byte o più byte). Le richieste che si invano sui bus possono essere di natura diversa, come leggo un byte o leggo 64 bytes a partire dall'indirizzo che ho emesso. Quindi il bus deve consentire di convogliare questi 3 tipi di informazioni che sono informazione sul dato letto, sull indirizzo e sul controllo del tipo di trasferimento che si vuol fare. Relativamente al controllo esistono identificativi di chi sta leggendo o scrivendo e a quale lettura\scrittura sta facendo riferimento quella determina comunicazione. Fin ora abbiamo sempre immaginato che il trasferimento di dato sia un operazione atomica, cioè quando decido di leggere una variabile non faccio altro nell attesa che venga letto. In realtà esistono tipi di bus definiti out of order che consentono di avere più transazioni concorrenti e non bloccanti.

ARM

Bus Terminology

Esistono anche bus che sono bloccanti, ovvero che quando si chiede un dato questi restano bloccati finché non ci viene restituito, per cui L operazione è atomica. Esistono anche bus non bloccanti che consentono di avere più richieste simultaneamente Dalla stessa unità o da diverse unità che sono maester. Quindi chi ha iniziato più richieste deve essere in grado di abbinare i riscontri alle relative richieste.
Ci immaginiamo che il bus sia un filo dritto che interconnette vari componenti (iniziatori e target), in realtà i bus non sono più così. Se ho un bus a 32 bit , non condivido le linee elettriche su cui passano i dati tra master e slave ma le replico, nel senso che ho canali mono direzionali.
Abbiamo sempre un master, uno o più slave, un canale delle informazioni (canale degli indirizzi), che è condiviso in linea concettuale con tuti gli slave, poi ci sarà un canale che condivide i dati e un canale che condivide le informazioni di controllo. Questi vengono mandati a tutte le periferiche. Per identificare il dispositivo che effettivamente è target dell informazione, quando il master emette un indirizzo, in funzione dell indirizzo specifico questo entra in un blocco che fa decodifica dell indirizzo ed identifica quale dispositivo target è coinvolto dalla comunicazione.



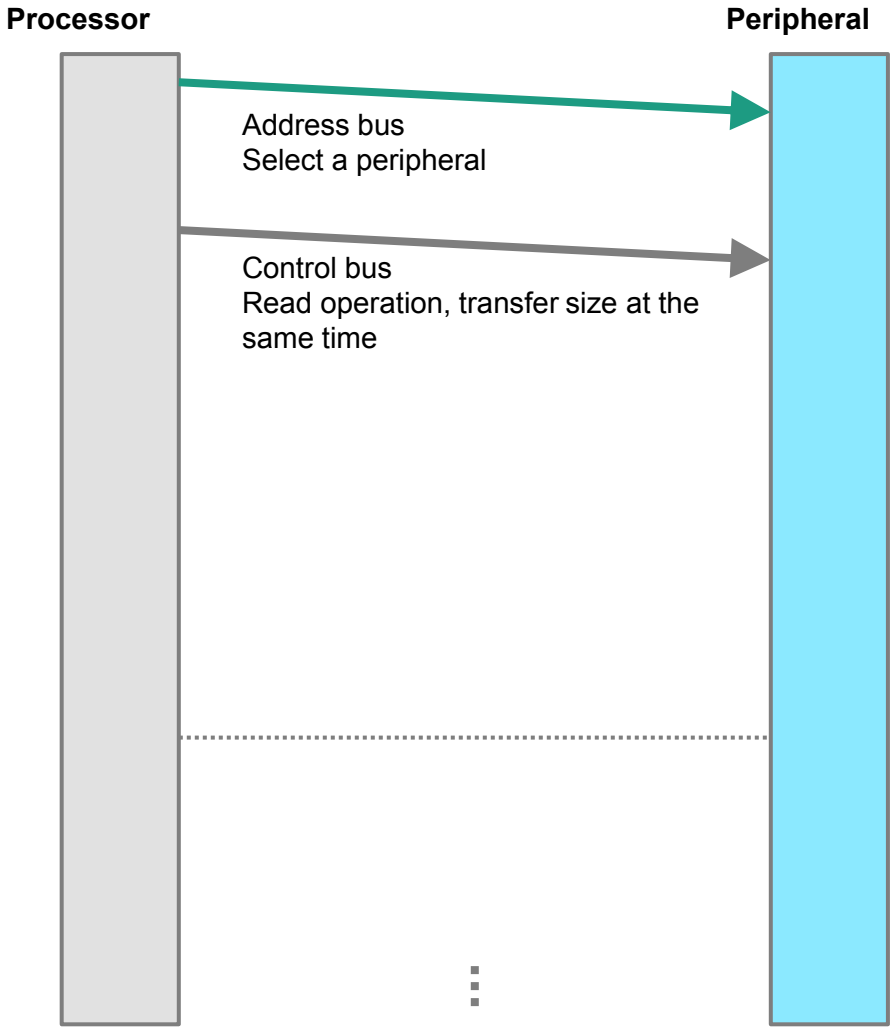
L address decoder identifica il dispositivo che si trova nella determinata regione della mappa degli indirizzi.

Quando si hanno più dispositivi master e slave c'è bisogno di un interconnessione, quindi una crossbar ad esempio. In questo caso ho un demux che consente di passare da una risposta data da tutti gli slave verso un singolo ingresso del dispositivo master.

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.

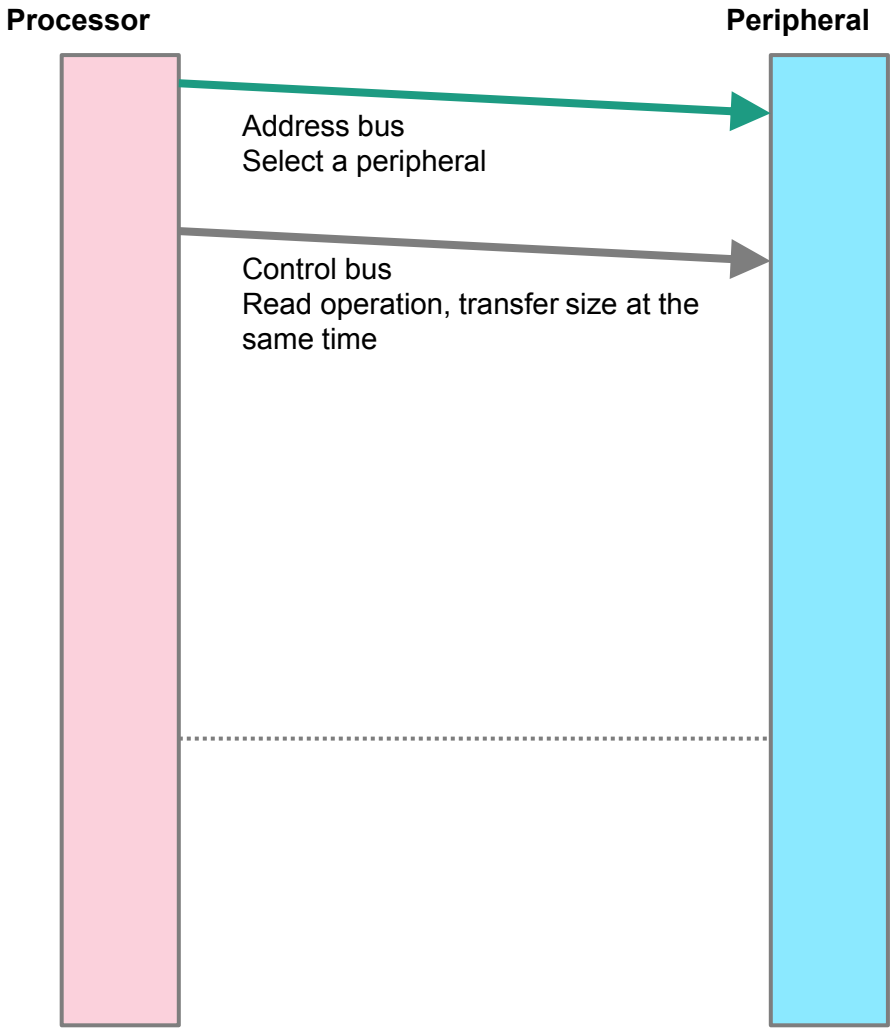
Il livello di parallelismo di un bus (es 32 bit) non vincola la lettura massima a quel valore... è possibile leggere 64 bits, ma metà alla volta. Il parallelismo influenza la velocità con cui viene fatto il trasferimento.
Il livello di // vincola la banda.



ARM

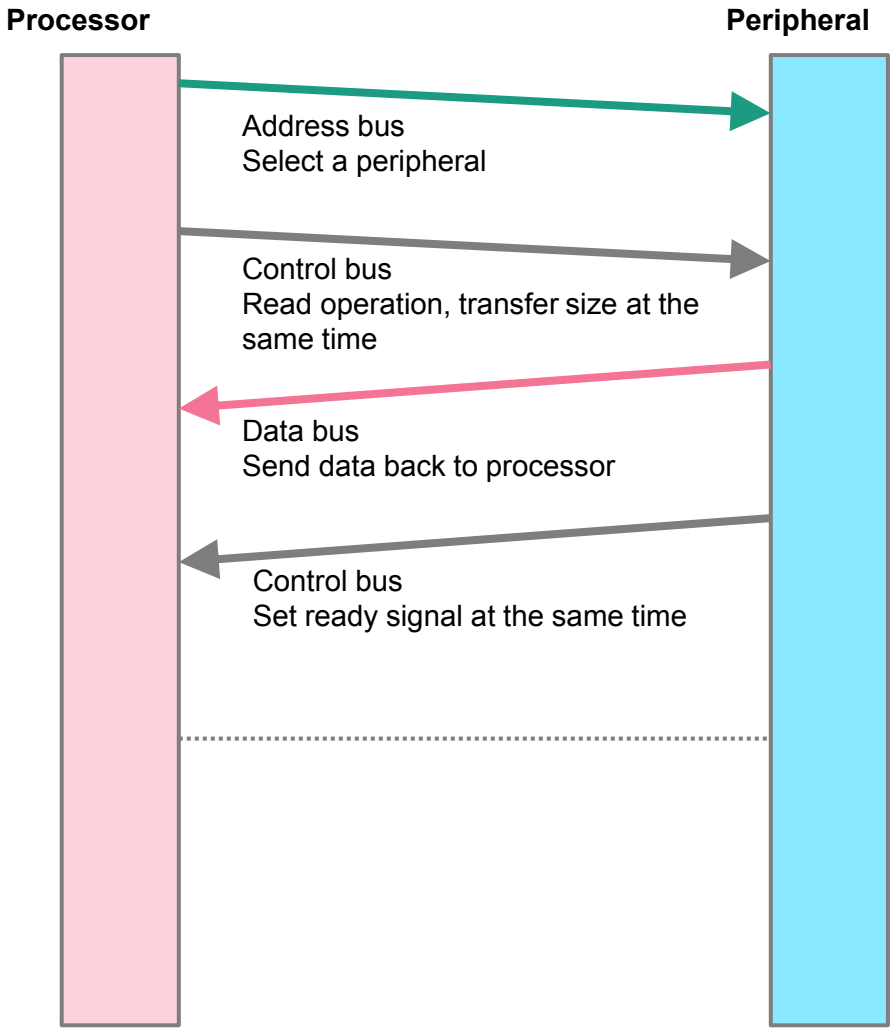
A Typical Bus Operation Example

- 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.

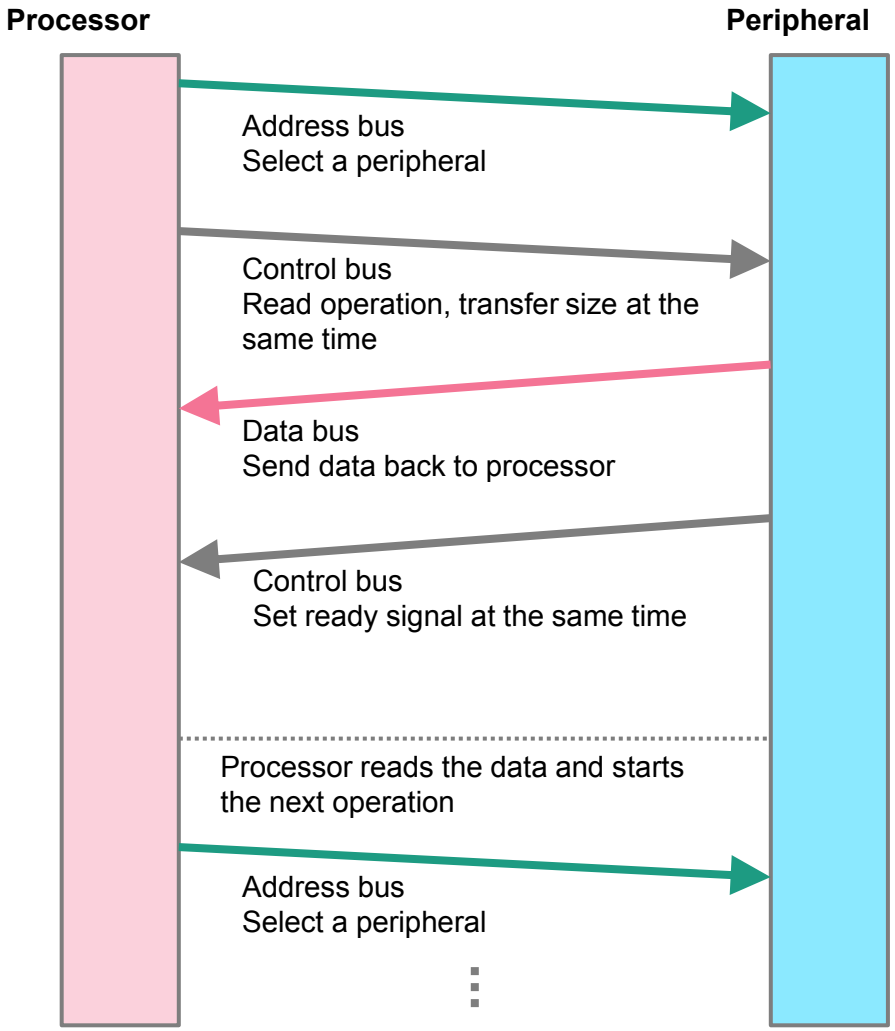


ARM

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.

Quando l'operazione è terminata se ne può iniziare un'altra.



ARM

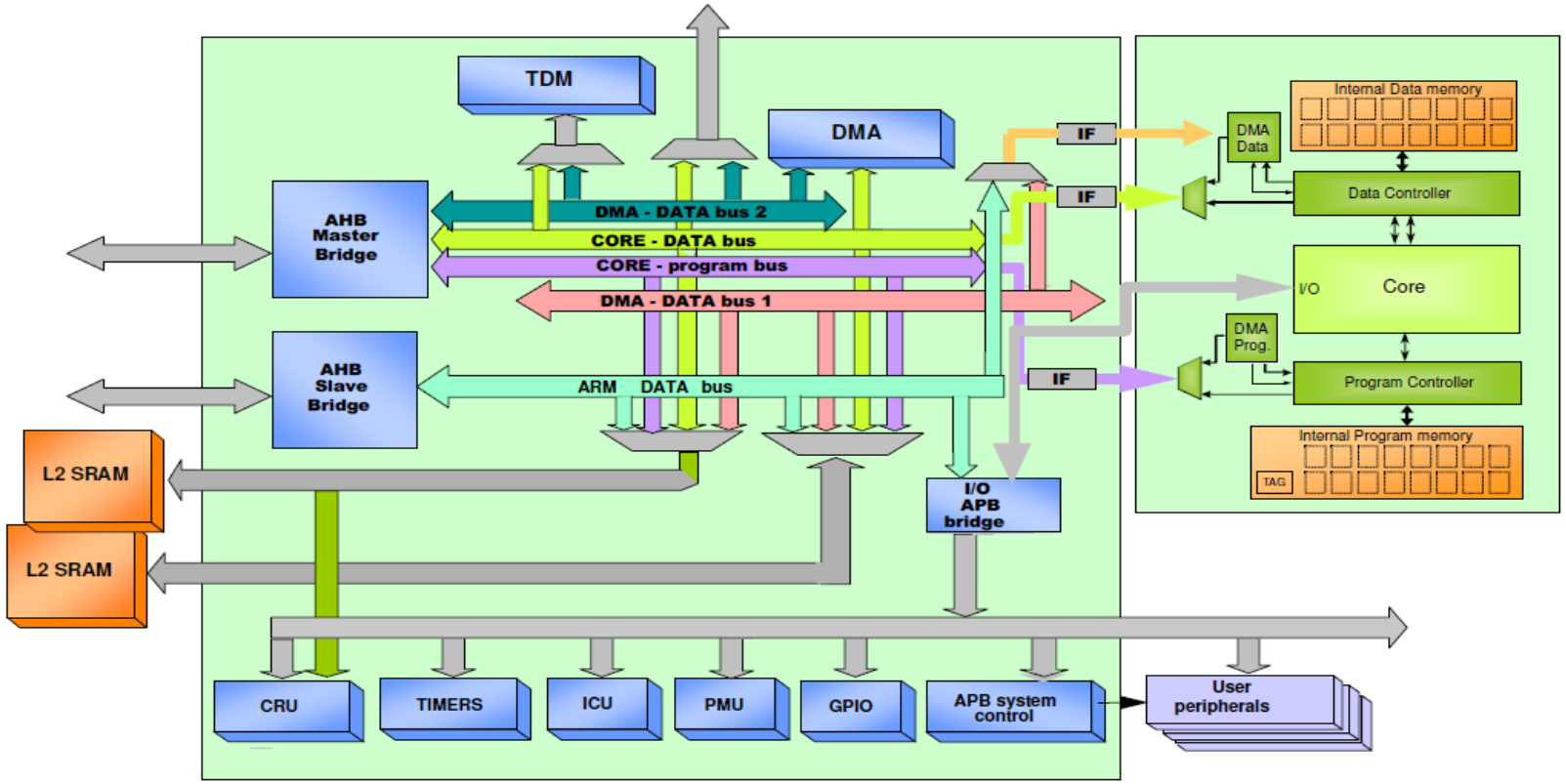
Outline

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

ARM

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

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

ARM AMBA Bus Families

NETWORK on CHIP : si comunica da un master alla periferica emettendo un indirizzo, ???

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.

ARM

AMBA 4 Specifications

riduzione del protocollo AXI in cui non esiste il canale degli indirizzi

- **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.

ARM

Outline

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

ARM

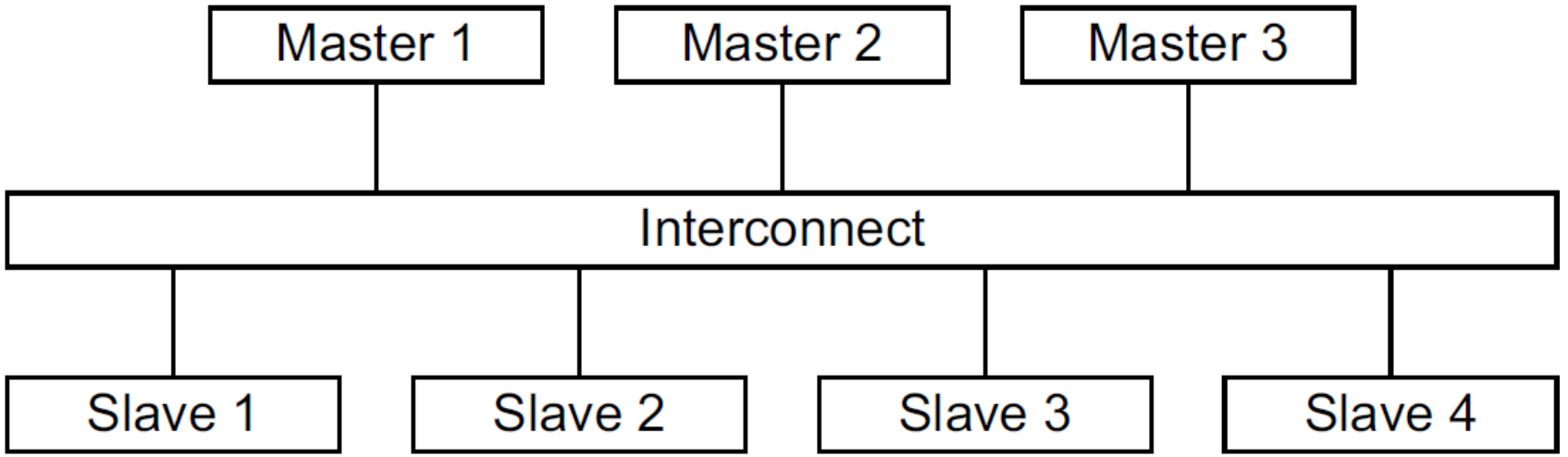
AXI Components and Topology

- **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** consente di connettere + master a + slave
  - 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.

ARM

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.



ARM

Outline

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

ARM

Transaction Channels

Transazione: intero insieme di operazioni necessarie ad effettuare una comunicazione: richiesta di indirizzo, scambio di segnali di controllo e trasferimento dei dati. Il trasferimento sarà composto da un dato che può essere trasferito da un burst,c ne è un trasferimento di dati composto da più bit.

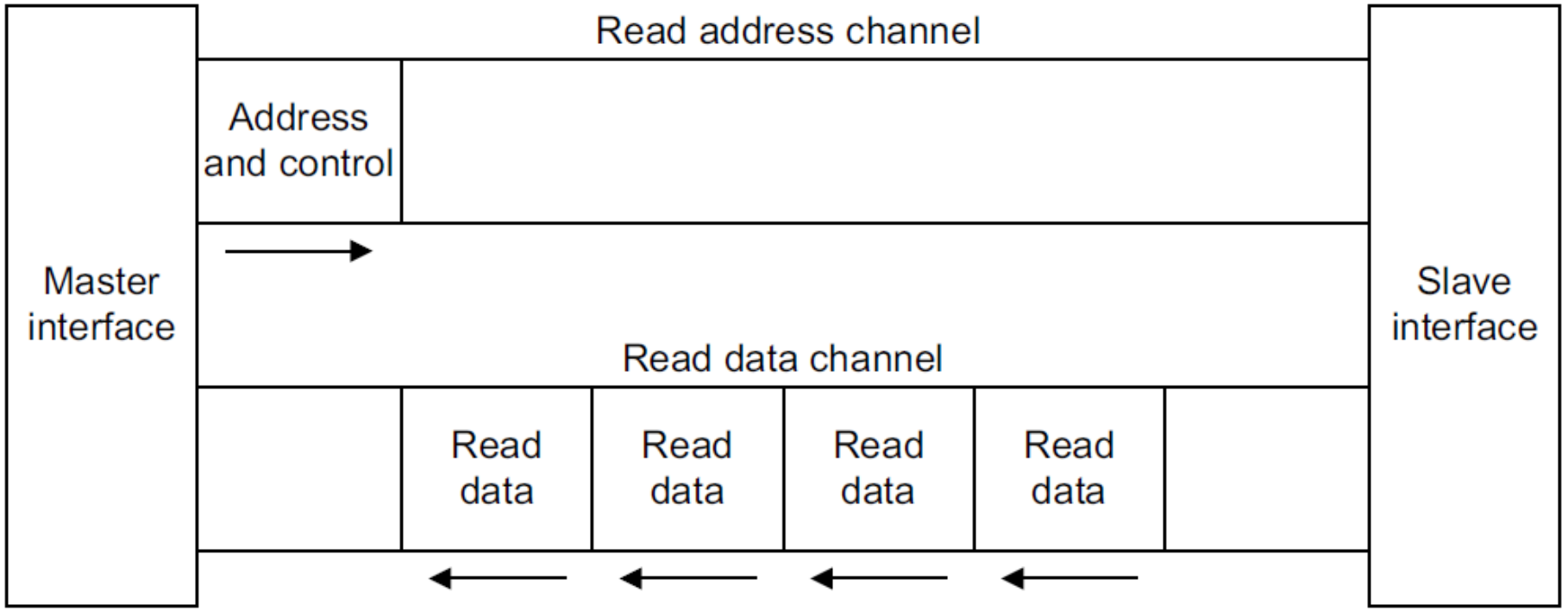
- When an AXI master initiates an AXI operation, targeting an AXI slave
  - the complete set of required operations on the AXI bus form the AXI **Transaction**
  - any required payload data is transferred as an AXI **Burst**
  - a burst can comprise multiple data transfers, or AXI **Beats.**
- The AXI 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).

Il protocollo ai ha 5 canali di comunicazione, c'è separazione tra i canali di lettura e scrittura sia per gli indirizzi che per i dati. Questo vuol dire che idealmente un master puo effettuare contemporaneamente delle lettura e delle scritture perché viaggiano su canali diversi.
Quindi abbiamo un canale di read address e uno di write address, uno contiene gli indirizzi emessi per operazioni di lettura e uno quelli per operazioni di scrittura. I canali address sono sempre comandati dal master, che governa quei segnali e li genera per inizializzare una transazione di lettura o scrittura.
Read data canale: lo slave può comunicare i dati letti verso il master
Write data canale: comandato dal master e serve quando si vuole fare in operazione di scrittura verso lo slave.
Write response: canale comandato dallo slave che serve per comunicare al master che la scrittura è terminata.

ARM

Channel Architecture of Reads

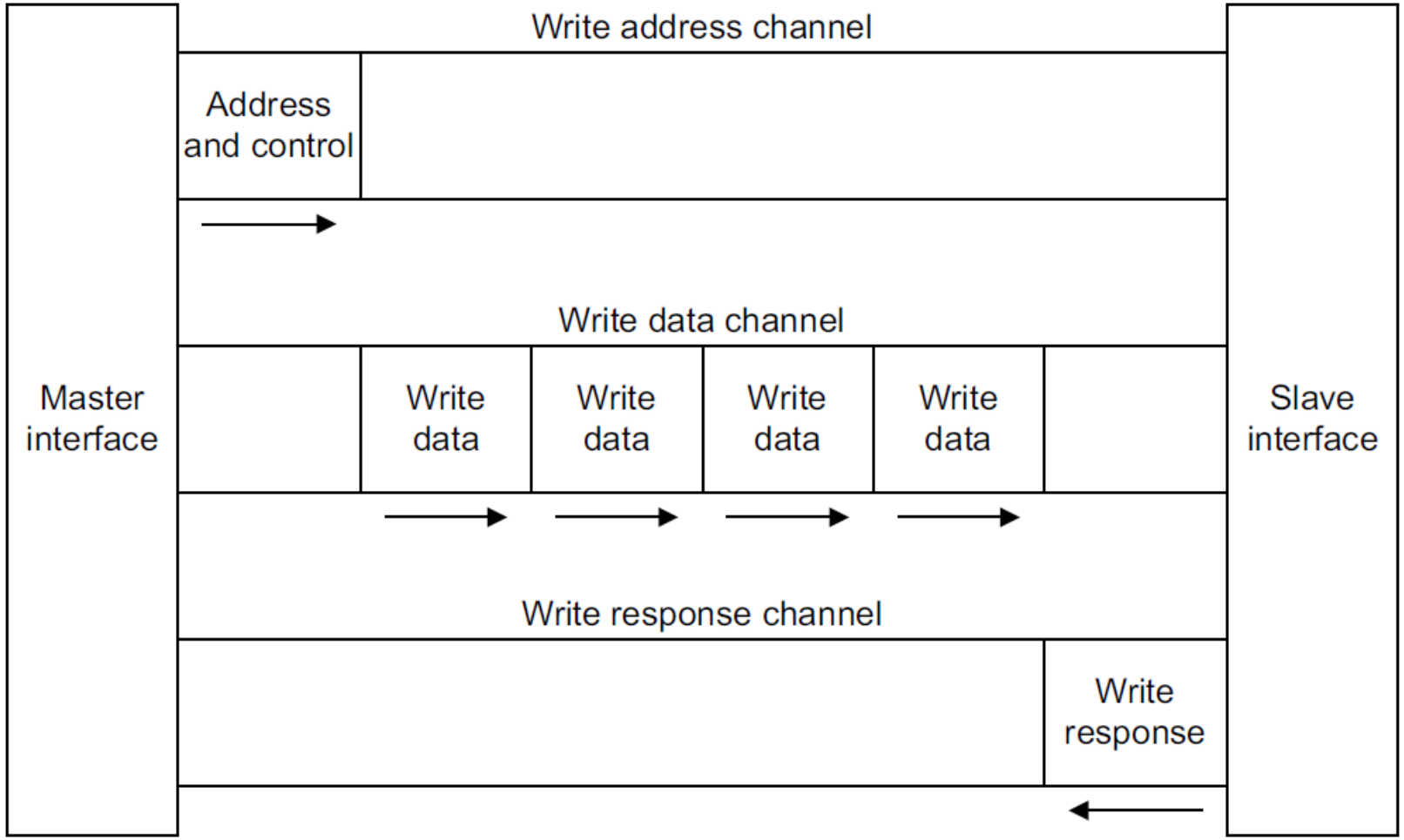
Procedura di lettura: il master emette un indirizzo dei segnali di controllo che sono parte del canale di read address, e questi vengono emessi allo slave. Lo slave risponde trasferendo treni di dati verso il master. Non c'è bisogno poi di un handshake a posteriori, perché il master sa qual e la richiesta di dati letti e quindi può contare i pacchetti che riceve quindi sa che è terminata la transazione di lettura.



ARM

Channel Architecture of Writes

Nel caso di scrittura il master emette un indirizzo dei segnali di controllo, successivamente emette pacchetti dati in scrittura verso lo salve. Quando lo slave ha terminato di processare i dati comunica che la scrittura è terminata e lui ha immagazzinato quei dati che sono stati trasferiti. I protocolli amba sono fatti per essere tolleranti alla latenza, la combinazione può operare diversi cicli di ci. Inoltre il master può operare a un segnale di ci diverso da quello dello slave, quindi la periferica può funzionare a una frequenza diversa da quella del master.



Basic Signals

comunica che è stato effettuato la scrittura.

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, ARESETn				

- 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).

Pee tutti I canali esistono dei segnali di handske che governano il trasferimento dei dati. Per tutti i canali esistono segnali di handshake. Lo scambio dei dati avviene quando entrambi. Isengali valid ready sono alti e arriva un colpo di ck.
Il segnale di valid è governato da chi controlla il canale, e il ready viene alzato da chi risponde al canale.

ARM

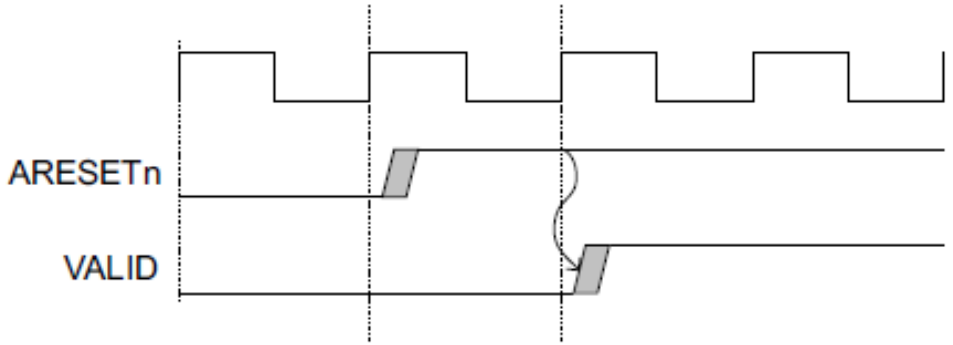
AXI Coherency Extensions (ACE) protocol

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.



ARM

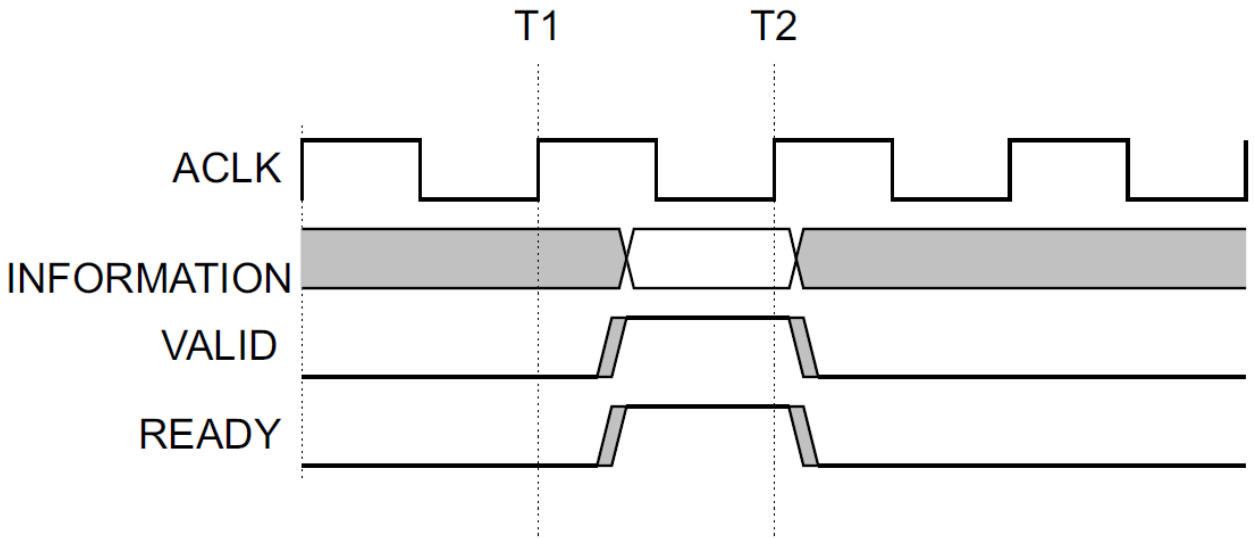
Outline

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

ARM

Channel Timing Example: VALID with READY handshake

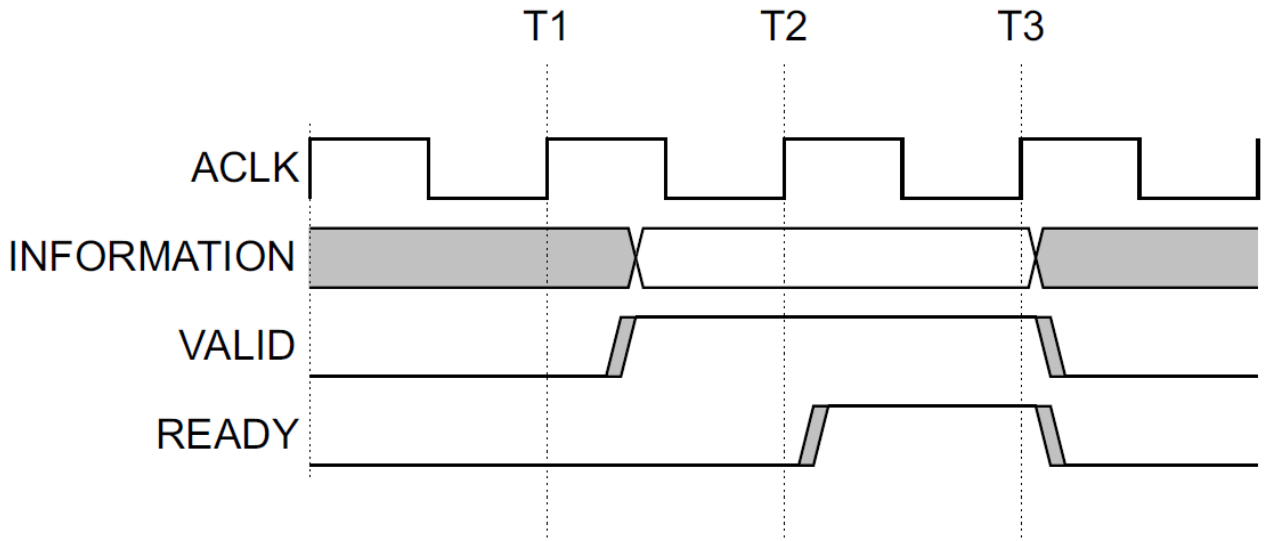
- After T1, both the source and destination indicate a data transferring.
- The transfer occurs at the rising clock edge (after both VALID and READY signals are asserted)
- The transfer occurs at T2.



ARM

Channel Timing Example: VALID before READY handshake

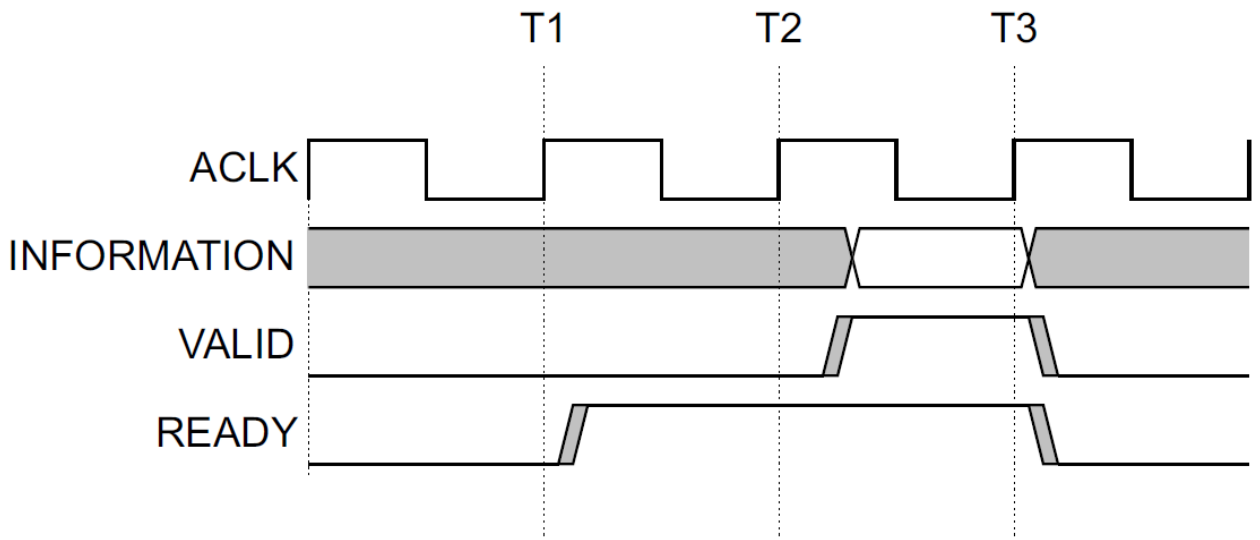
- After T1, the source presents the address, data or control information and asserts the VALID signal.
- The destination asserts the READY signal after T2.
- The source has to keep its information stable until the transfer occurs at T3.



ARM

Channel Timing Example: READY before VALID handshake

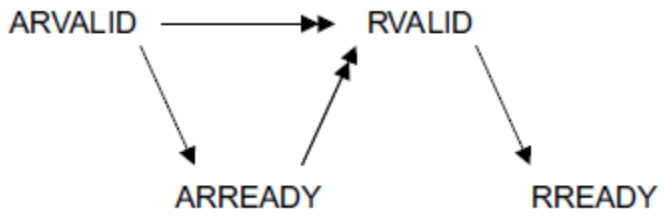
- After T1, the destination asserts the READY signal (before the address, data or control information is valid) to indicate that it can accept the information.
- After T2, the source presents the information, and asserts VALID
- The transfer occurs at T3 (when this assertion is recognized



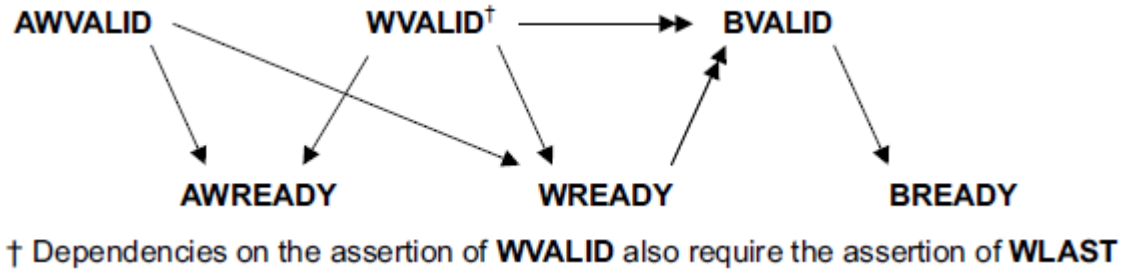
ARM

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



Write

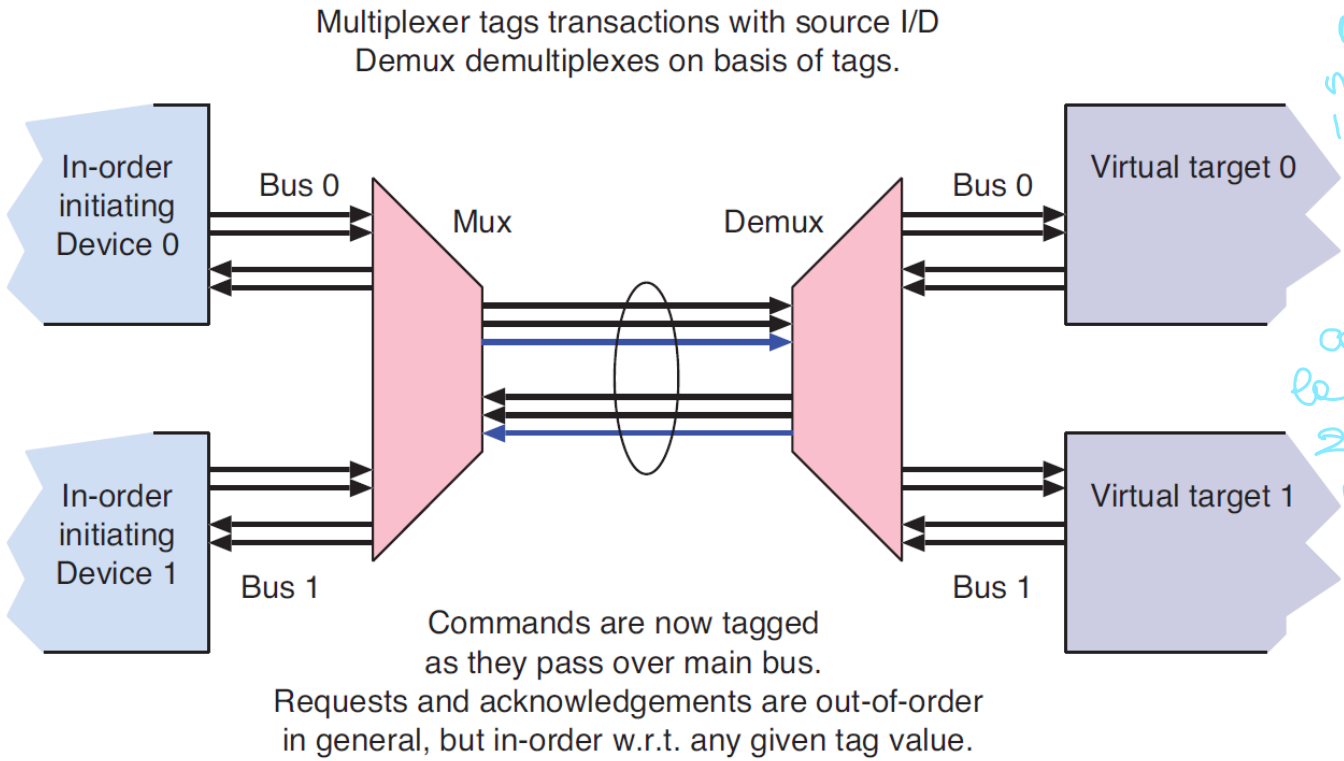
ARM

AXI Read & Write channel

Bisogna stallare operazioni di lettura che arrivano dopo una scrittura verso lo stesso indirizzo perché se non lo facesse non ci sarebbe garanzia che la lettura avvenga a valle della scrittura.

- 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



Affinché un target possa rispondere alla richiesta fatta da uno specifico master, il campo del transaction ID viene completato con le informazioni relative a qual è la sorgente della comunicazione. Quindi i comandi vengono composti x identificare qual è la sorgente di una data comunicazione, così che lo slave possa rispondere esattamente a quel transaction ID.

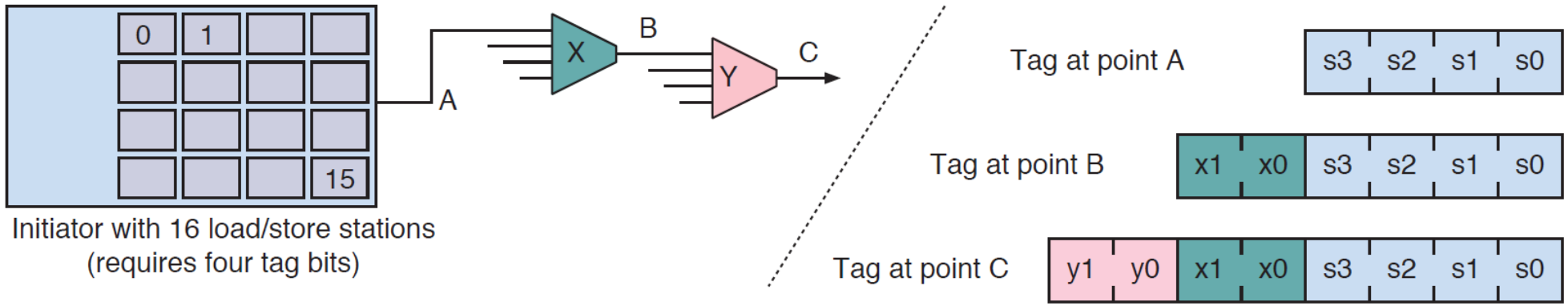
Se voglio garantire una sequenzialita degli accessi, posso asciare un id e poi usarlo in sede di risposta per ordinare le varie risposte.

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

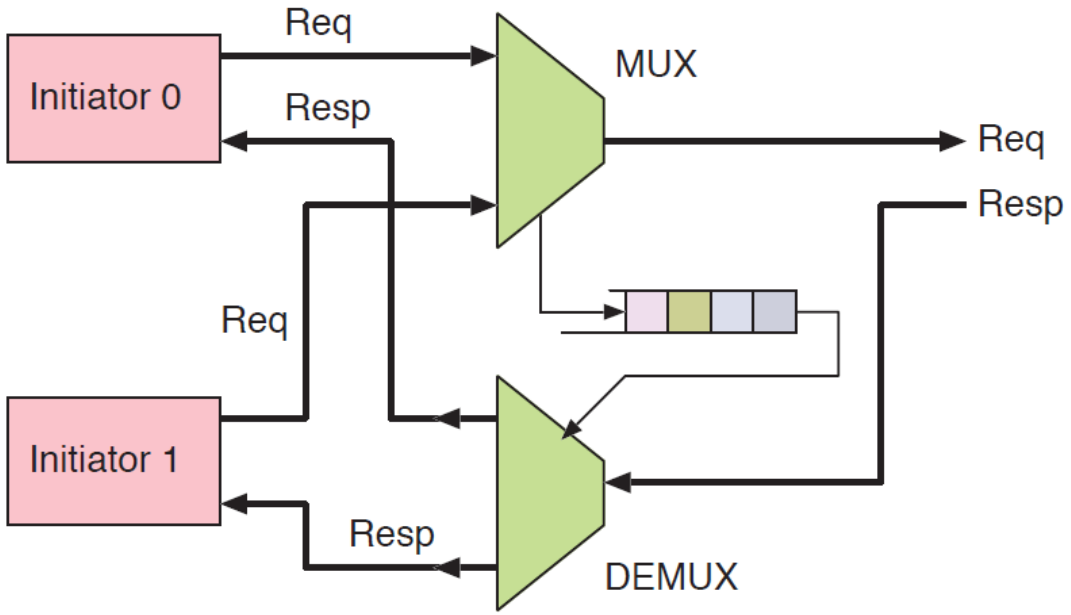
- TAG Size must be large enough to:
  - distinguish between different initiators that multiplex together
  - support the maximum number of differently numbered outstanding transactions generated by an initiator
- Simplest management technique
  - for each individual source to generate tags with a width 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.

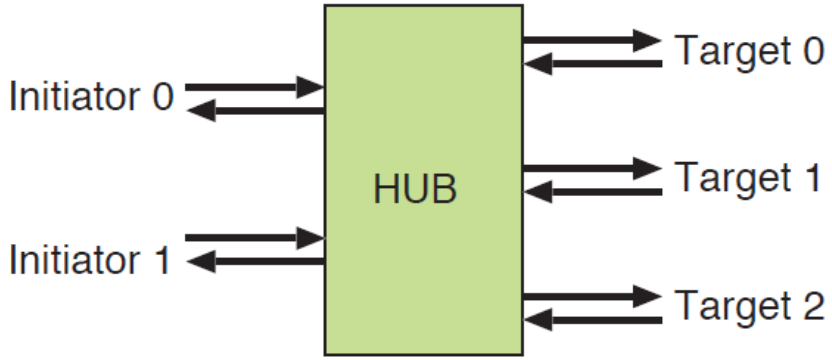


ARM

Ordered and Unordered Interconnects

- In-order interconnect FIFO to enumerate transactions:





ARM

Bibliography

- AMBA4 and ACE especifications, 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

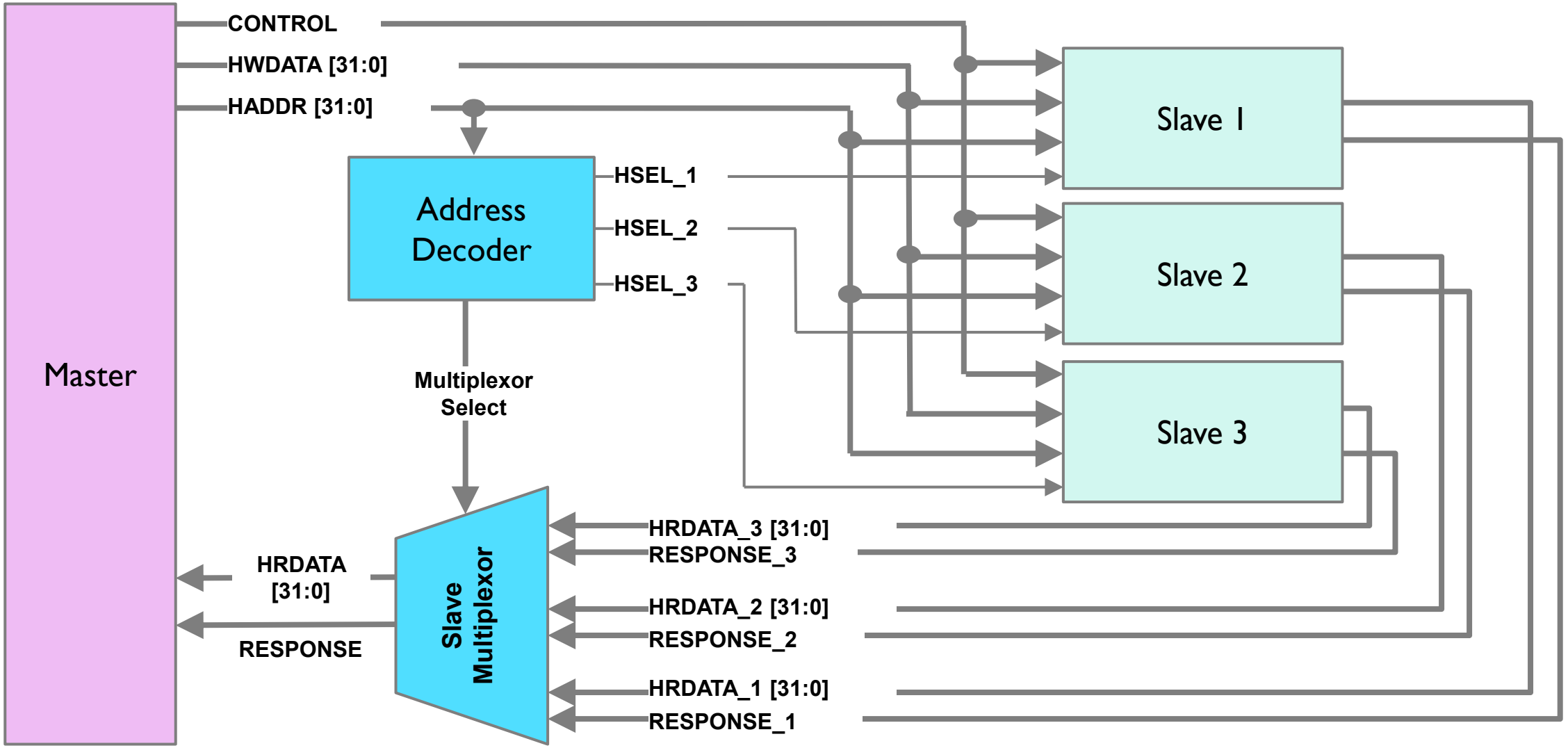
concetto di coerenza?

ARM

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

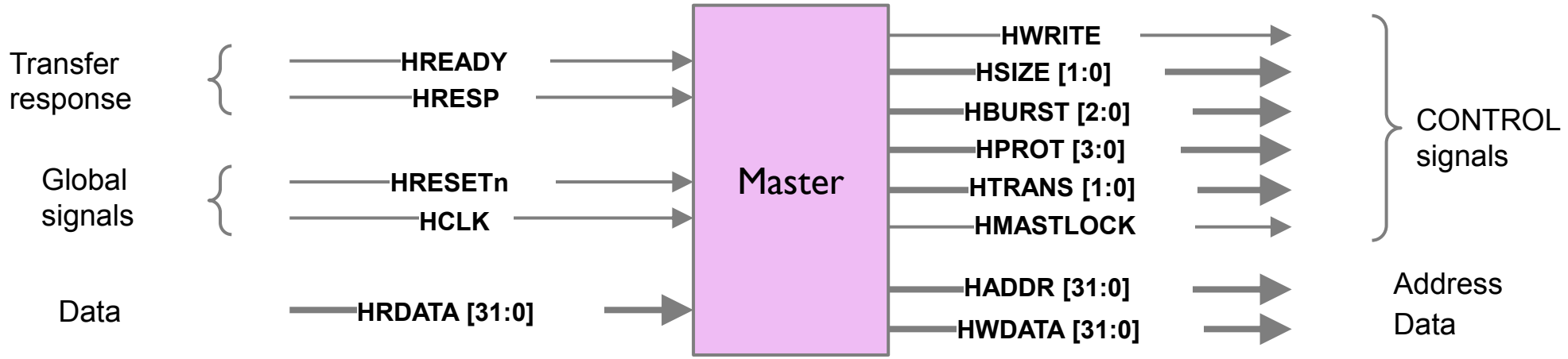
AHB-Lite Bus Block Diagram



ARM

AHB-Lite Master Interface

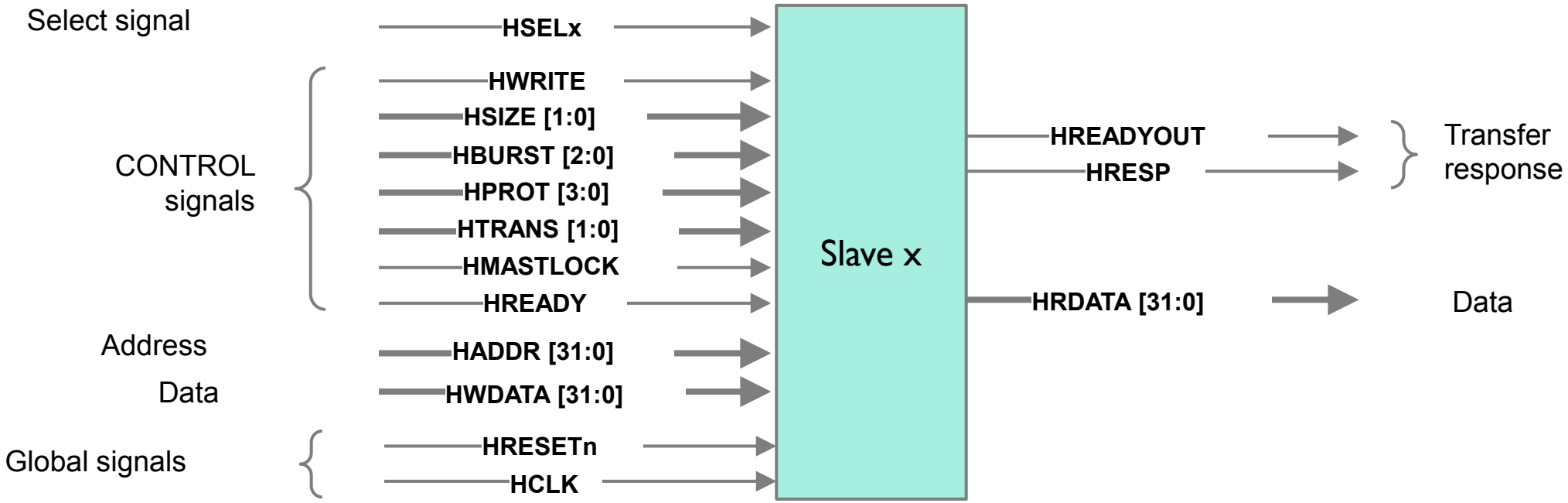
- The AHB-Lite master provides address and control information to initiate read and write operations.
- The master also receives the response from the slave, including data and ready and response signal.



ARM

AHB-Lite Slave Interface

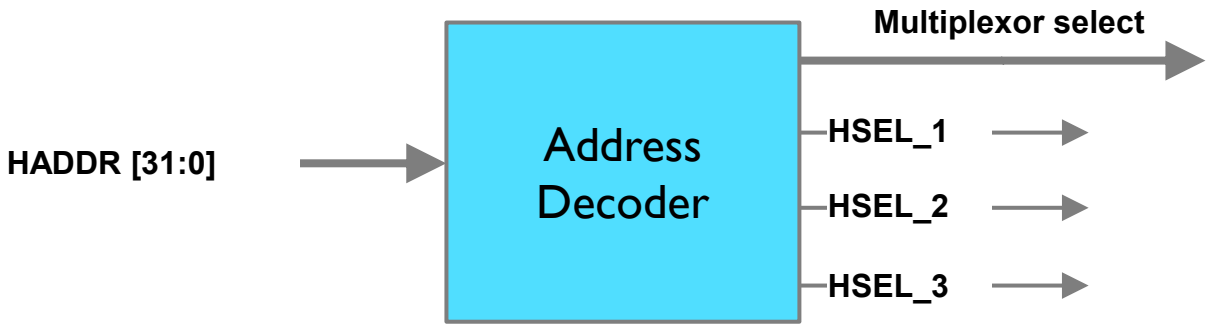
- An AHB-Lite slave responds to transfer initiated by the master in the system.
- The signal HSELx is the output from the address decoder, which is used to select one slave at a time.



ARM

Address Decoder

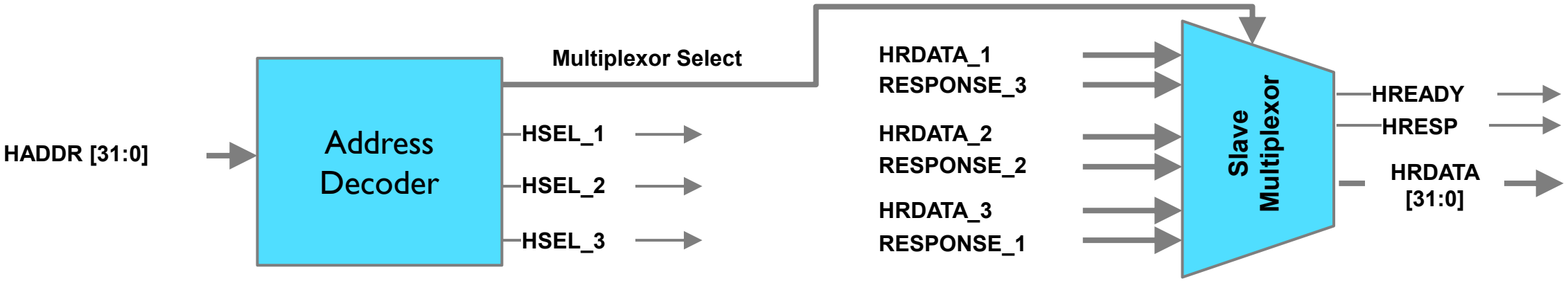
- Address decoder
  - Selects one of the slaves depending on the current address bus
  - Also informs the slave multiplexor



ARM

Slave Multiplexor

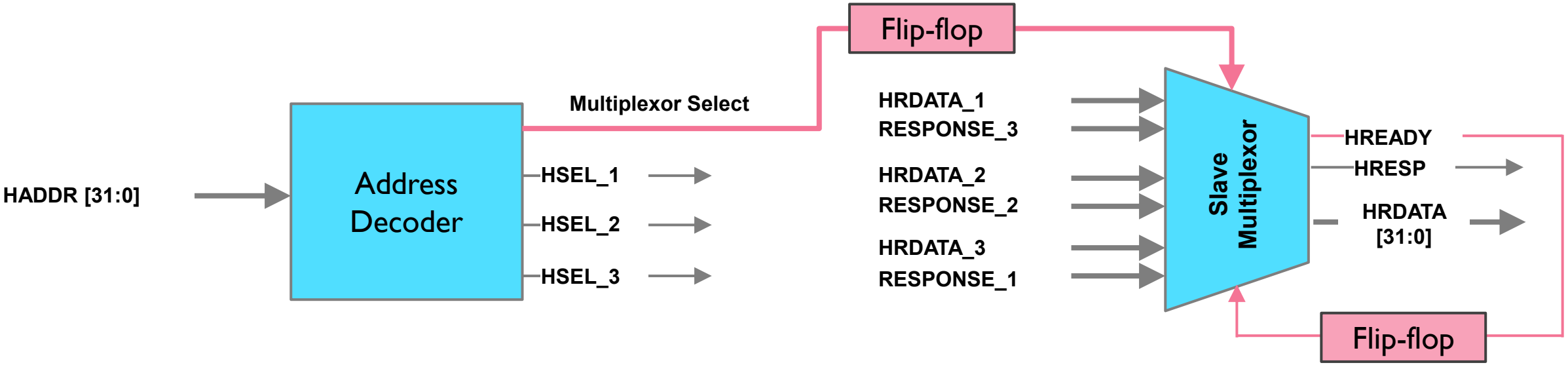
- Slave multiplexor
  - Inputs the response signals (HRDATA, HREADY, and HRESP) from all the slaves, and outputs one of them depending on the selecting signal from the address decoder.



ARM

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.



ARM

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

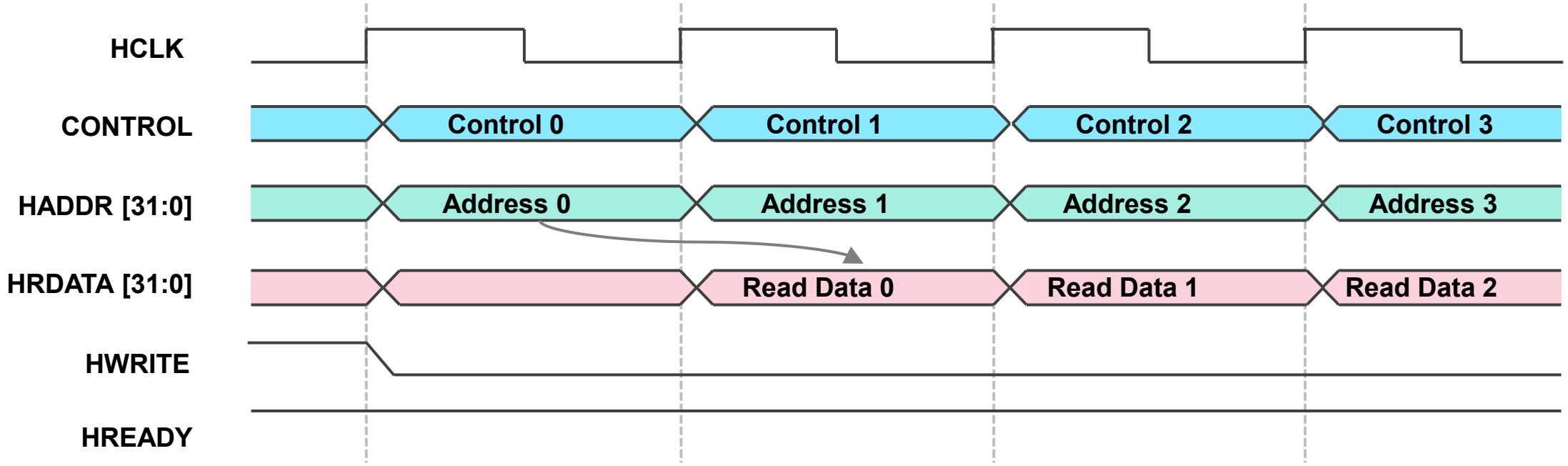
- An AHB-Lite transfer consists of two phases:
  - The address phase, which lasts for a single HCLK cycle unless it is extended by the previous bus transfer.
  - The data phase might require several HCLK cycles. The HREADY signal is used to control the number of clock cycles required to complete the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

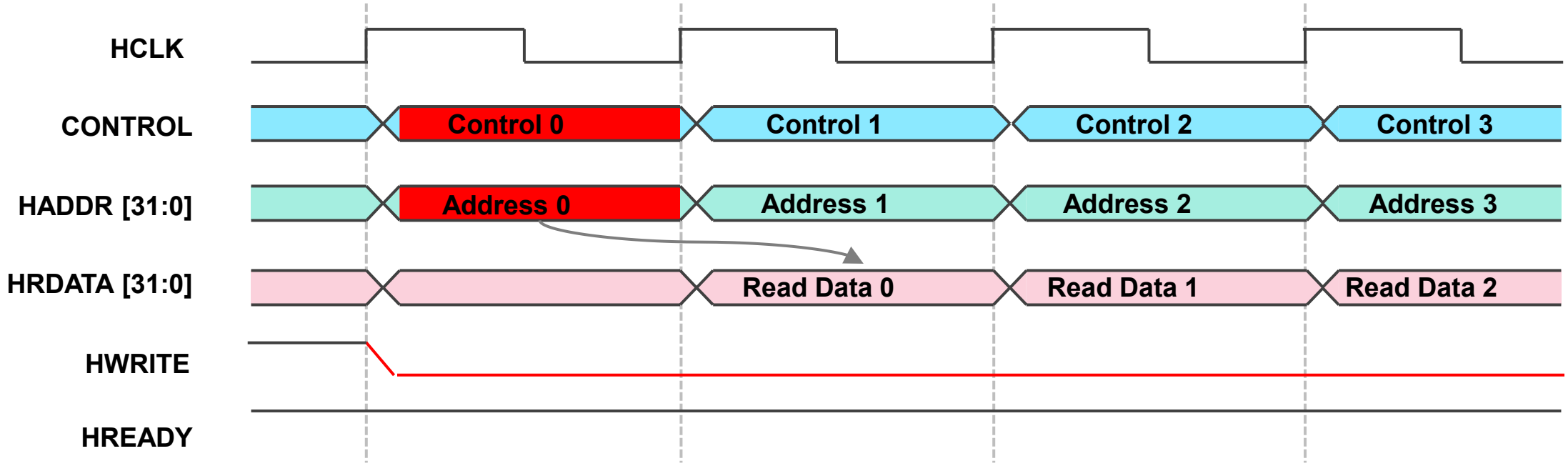
- Consider a simple read transfer with no wait states:



ARM

Basic Read Transfer

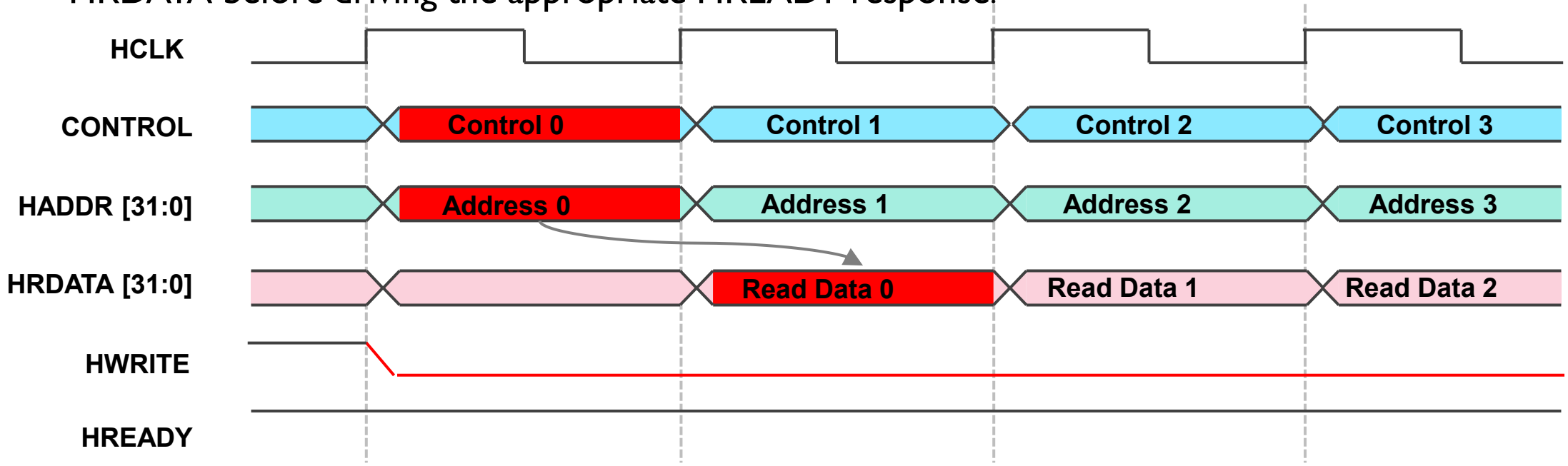
- Consider a simple read transfer with no wait states:
  - The address phase: The master drives the address and control signals onto the bus after the rising edge of HCLK.



ARM

Basic Read Transfer

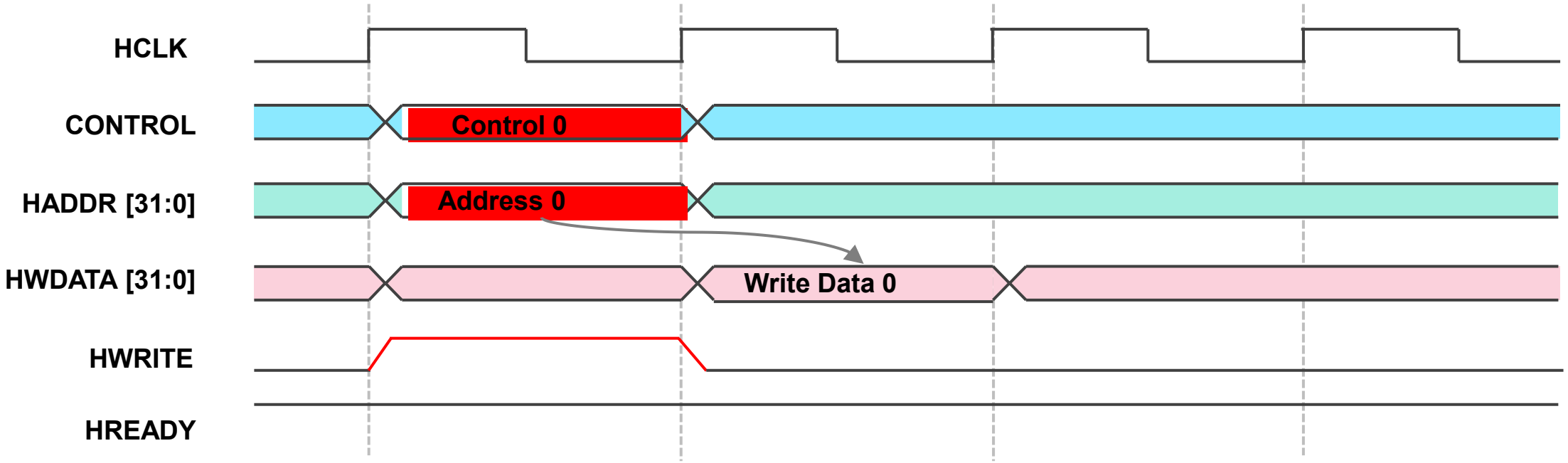
- Consider a simple read transfer with no wait states:
  - The address phase: The master drives the address and control signals onto the bus after the rising edge of HCLK.
  - The data phase: The slave samples the address and control information and make data available at HRDATA before driving the appropriate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



ARM

Basic Write Transfer

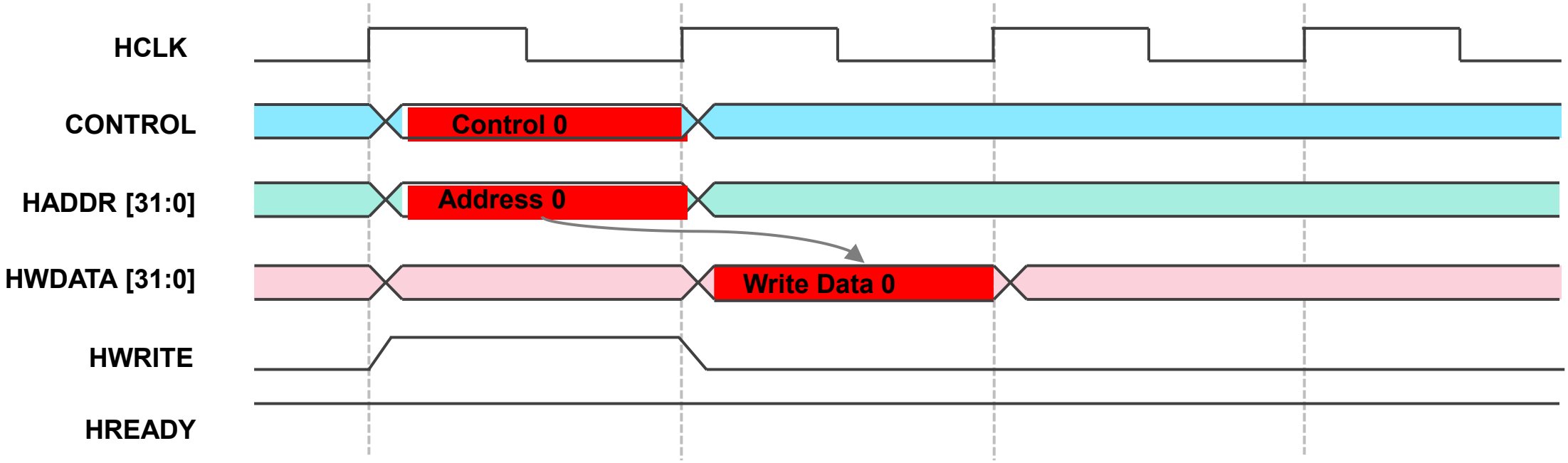
- Consider a simple write transfer with no wait states:
  - The address phase: The master drives the address and control signals onto the bus after the rising edge of HCLK and sets HWRITE to one.
  - The data phase: The slave samples the address and control information and make data available at HRDATA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



ARM

Basic Write Transfer

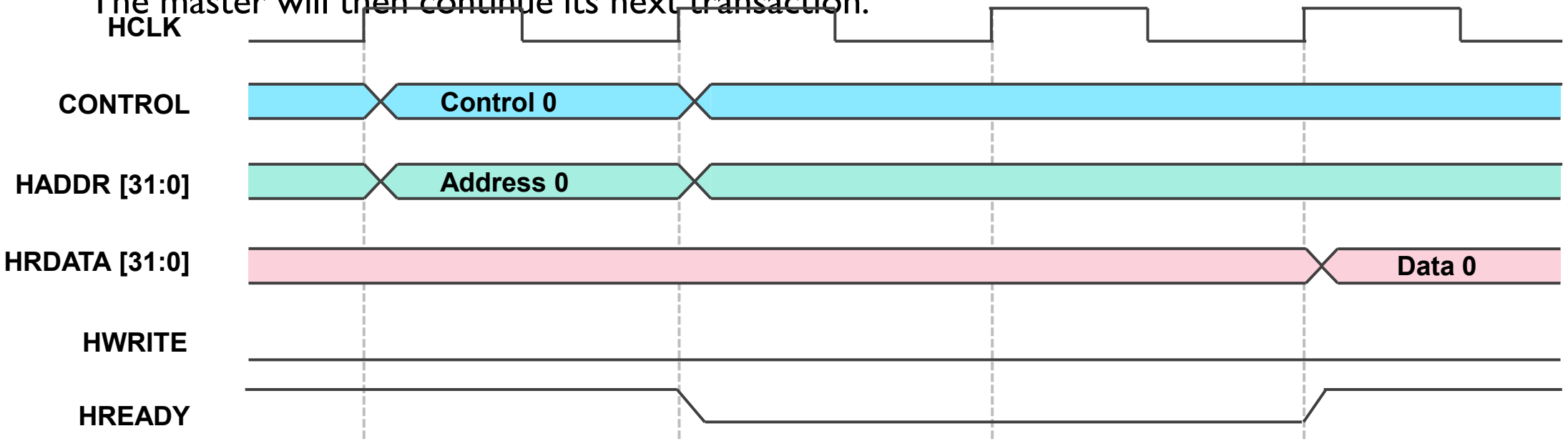
- Consider a simple write transfer with no wait states:
  - The address phase: The master drives the address and control signals onto the bus after the rising edge of HCLK and sets HWRITE to one.
  - The data phase: The slave samples the address and control information and make data available at HRDATA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



ARM

Read Transfer with Wait State

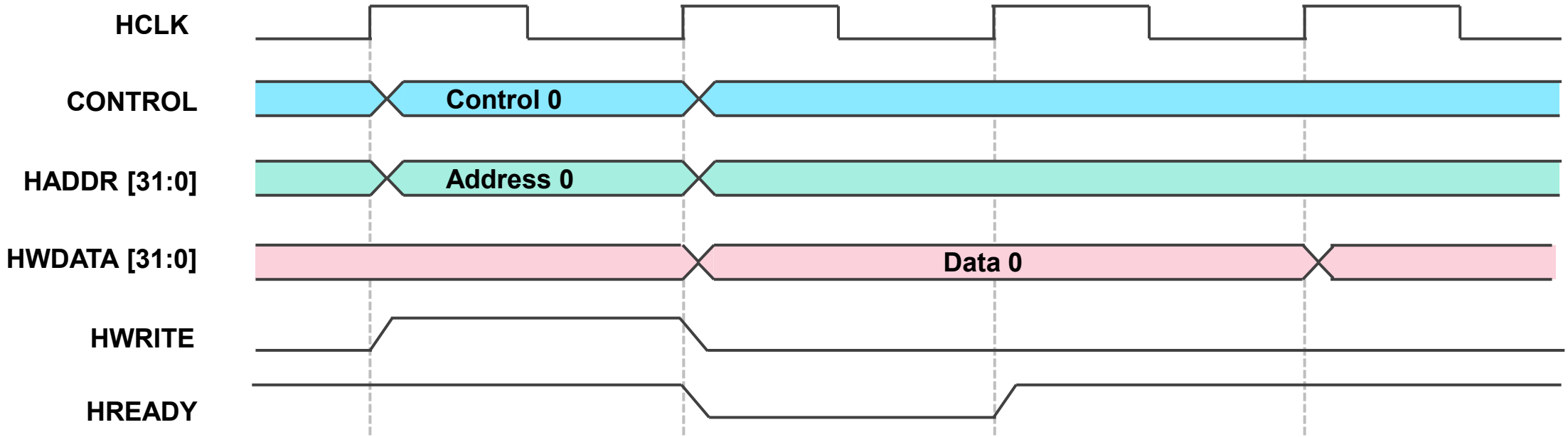
- Address phase (first clock cycle)
  - Give address and control signals; set HWRITE to one.
- Data phase (multiple clock cycles)
  - The slave holds HREADY to zero if it is not ready to provide its data; the master delays its next transaction.
  - When the slave is ready, the data will be given at HRDATA;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



ARM

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.



ARM