arm

AMBA AXI4 Bus Architecture

slides from https://github.com/arm-university/Advanced-System-on-Chip-Design-Education-Kit

© 2021 Arm

Learning Outcomes

At the end of this module, you will be able to:

- Describe the purpose and operation of a bus and communication protocol in an SoC.
- Identify the characteristics of various Arm AMBA System Buses including AMBA 3 and AMBA 4, AXI4-Lite, and AXI4-stream.
- Outline the functionality and characteristics of the Arm AMBA AXI4-Lite and AXI4-stream.
- Describe the transaction channels read and write operations for the AMBA AXI protocol.
- Explain the channel timing mechanism for AXI, including the clock, reset, and VALID and READY handshake mechanism.

© 2021 Arm

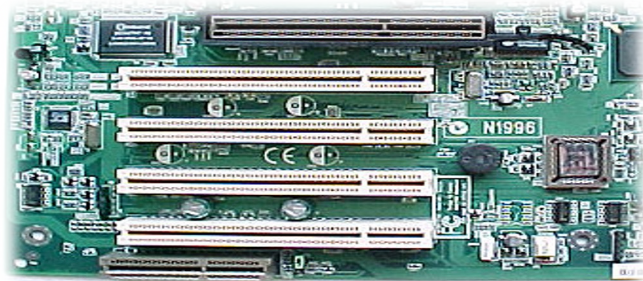
Inclusive Language Warning

- AMBA/AXI docs used to say Master/Slave instead of Manager/Subordinate terms
- You will likely encounter these references in code or older documentation

© 2021 Arm

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 the 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

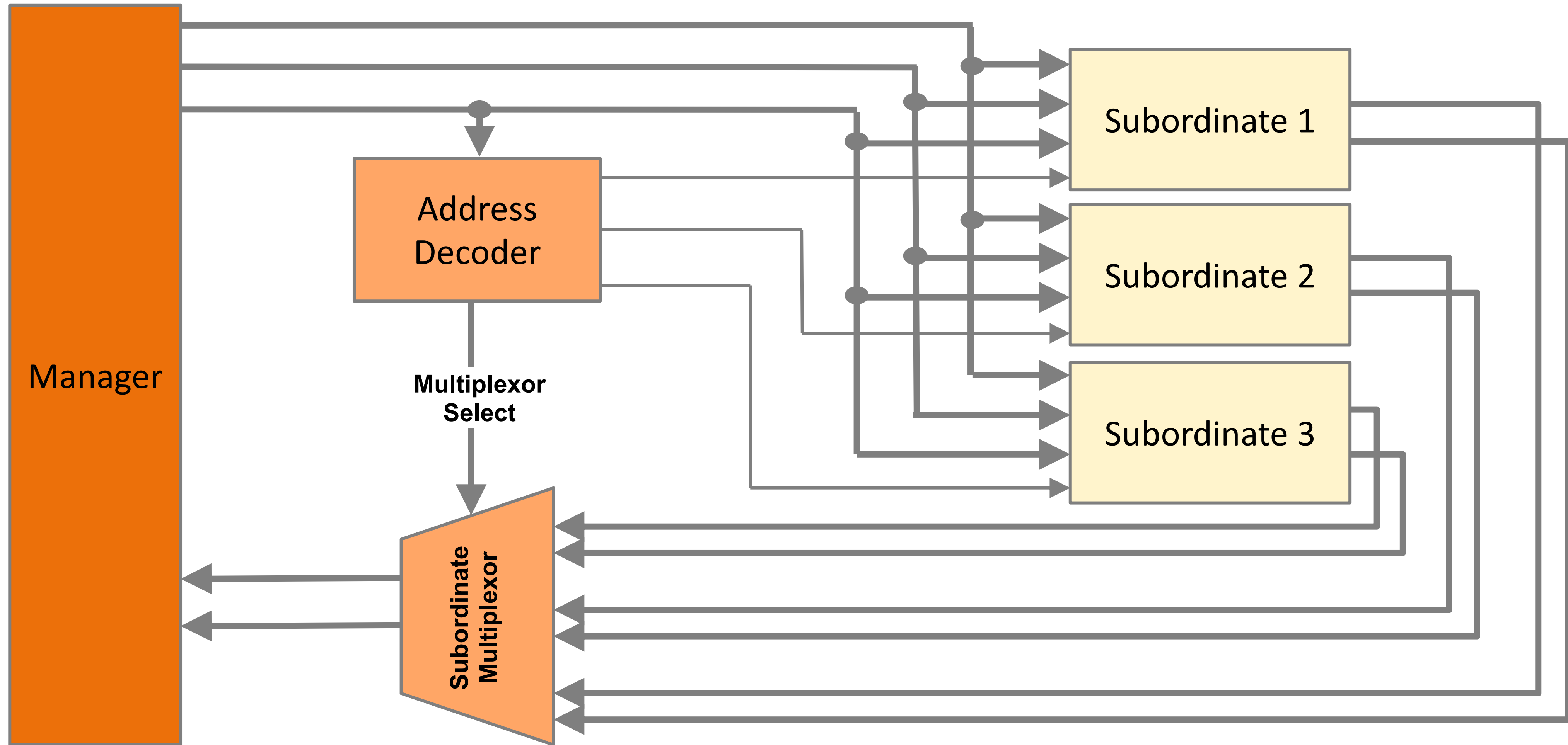
© 2021 Arm

Bus Types

- External bus
  - Used to connect external devices, such as a printer, to a computer
  - E.g., USB stands for Universal Serial Bus
- Internal bus
  - Used to connect internal components inside a computer, such as a CPU to memory
  - Also known as system bus
  - Less overhead, e.g., no need for electrical characteristics handling and configuration detection
  - Thus, typically runs faster than the external bus
  - In an SoC design, the internal bus is integrated onto a single chip; thus, it can also be referred to as an on-chip system bus.

© 2021 Arm

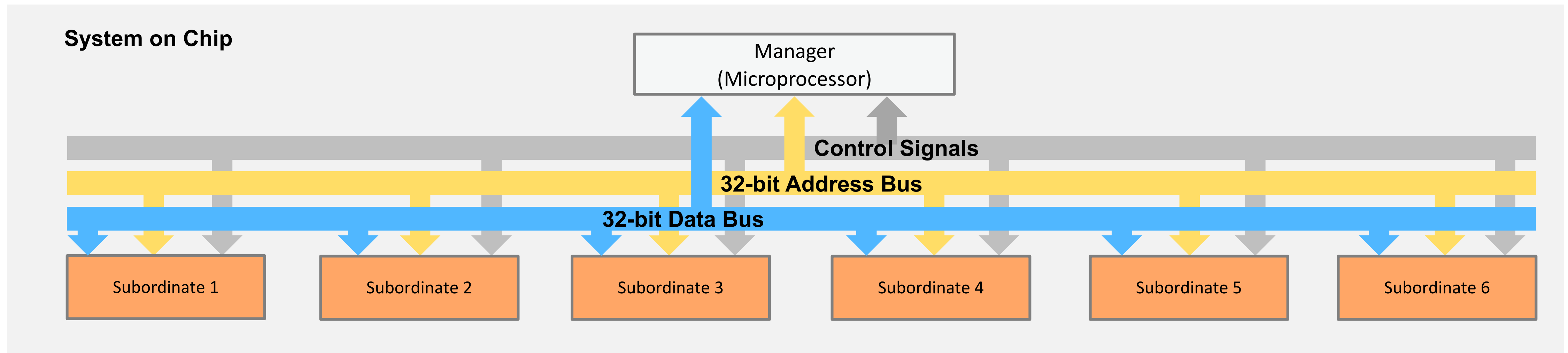
Bus Terminology



© 2021 Arm

Bus Operation in General

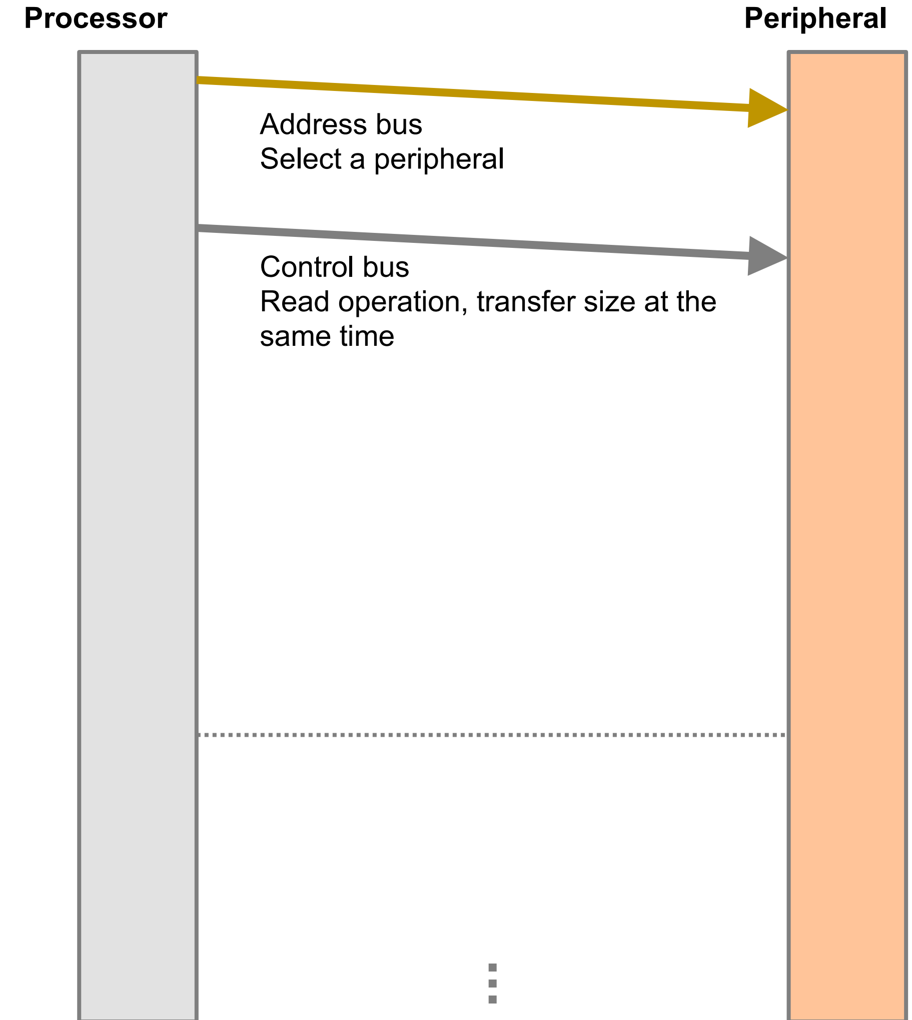
- A bus typically consists of three types of signal lines:
  - **Data bus** is used to exchange data information
  - **Address bus** is used to select one of the peripherals (or one register of a peripheral)
  - **Control signals** are used to synchronize and identify transactions, such as ready, write/read,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



© 2021 Arm

A Typical Bus Operation Example

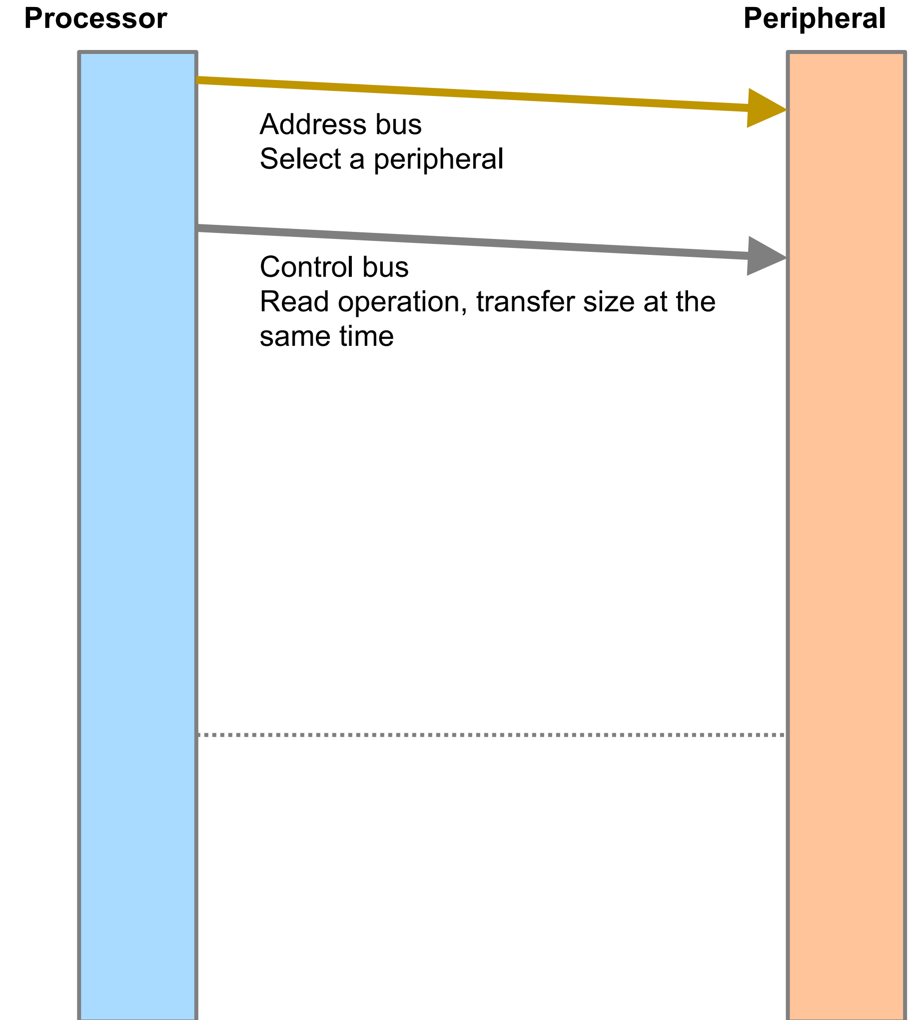
- A typical operation to access a peripheral mainly consists of the following:
  - The Manager (e.g., a processor) selects one peripheral (or one register) by giving the address to the address bus. At the same time, it sets control signals, such as read or write, transfer size, and so forth.



© 2021 Arm
arm

A Typical Bus Operation Example

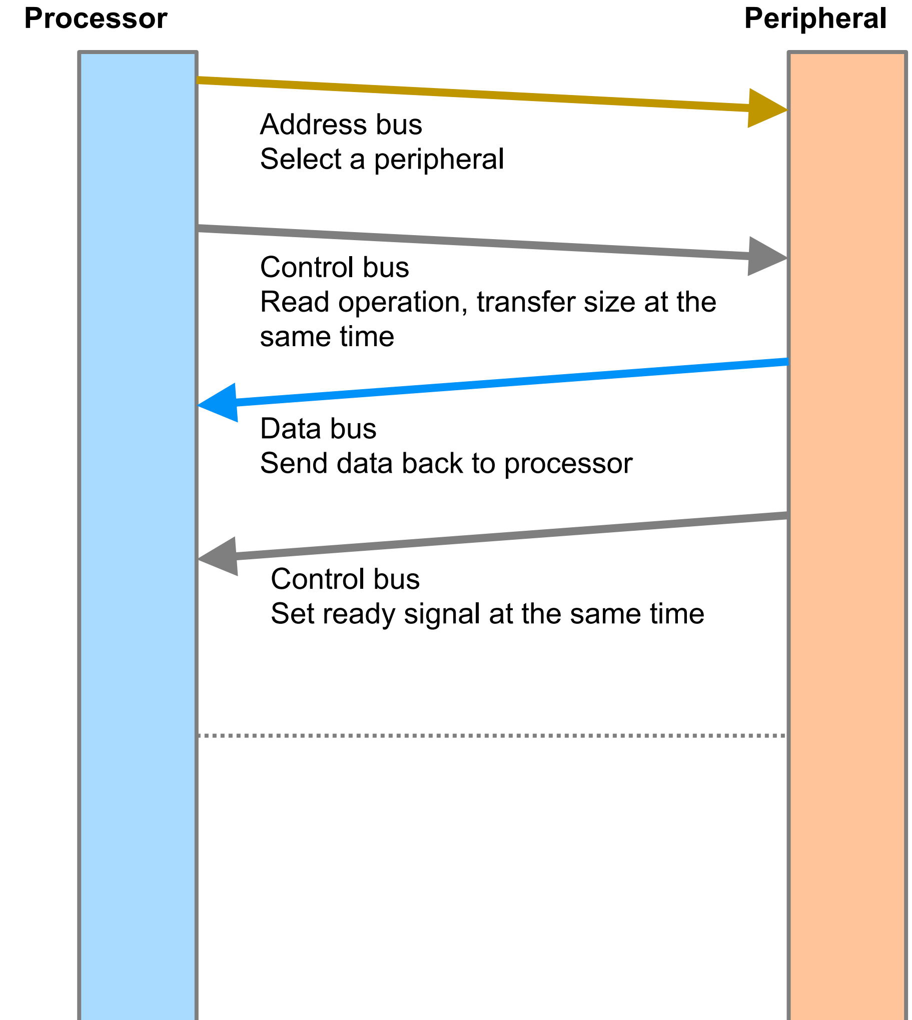
- A typical operation to access a peripheral mainly consists of the following:
  - The Manager (e.g., a processor) selects one peripheral (or one register) by giving the address to the address bus. At the same time, it sets control signals, such as read or write, transfer size, and so forth.
  - The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



© 2021 Arm
arm

A Typical Bus Operation Example

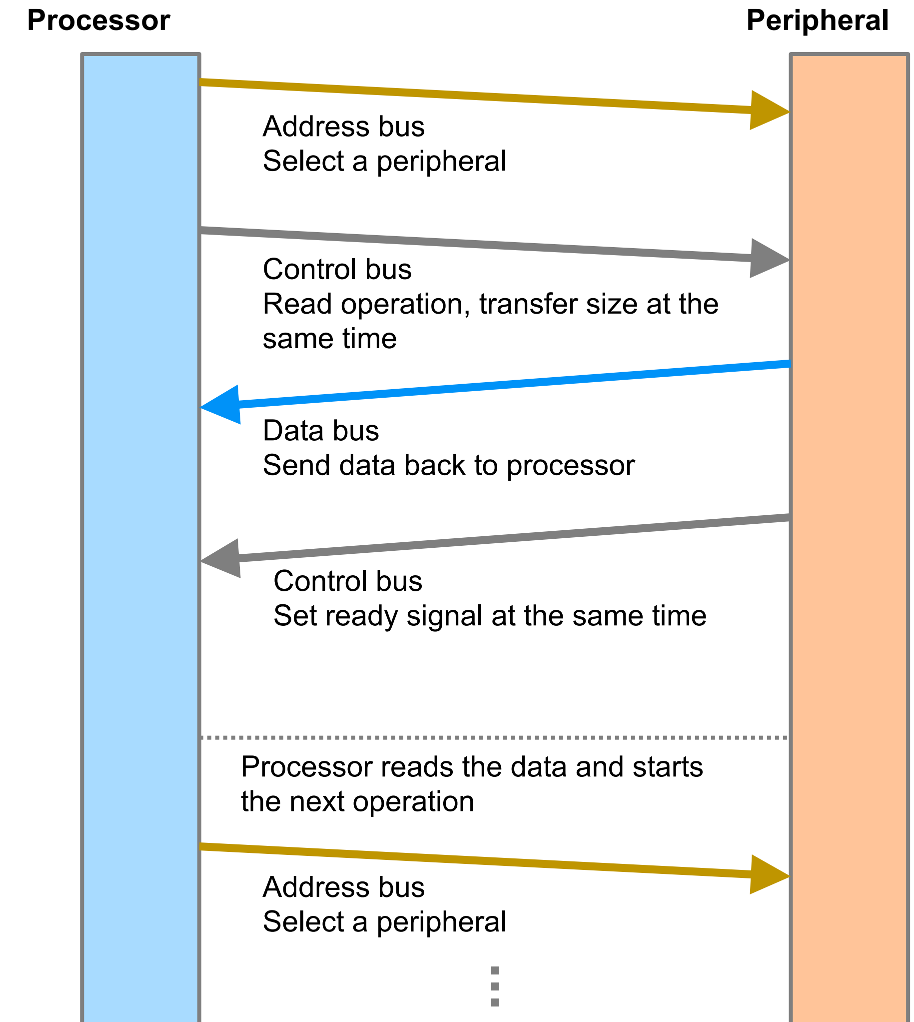
- A typical operation to access a peripheral mainly consists of the following:
  - The Manager (e.g., a processor) selects one peripheral (or one register) by giving the address to the address bus. At the same time, it sets control signals, such as read or write, transfer size, and so forth.
  - The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



© 2021 Arm
arm

A Typical Bus Operation Example

- A typical operation to access a peripheral mainly consists of the following:
  - The Manager (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.
  - The Manager waits for the Subordinate (e.g., peripheral) to respond.
  - Once the Subordinate is ready, it sends back the requested data to the processor. At the same time, it sets the ready signal on the control bus.
  - Finally, the Manager reads the transmitted data and starts another communication cycle.



© 2021 Arm
arm

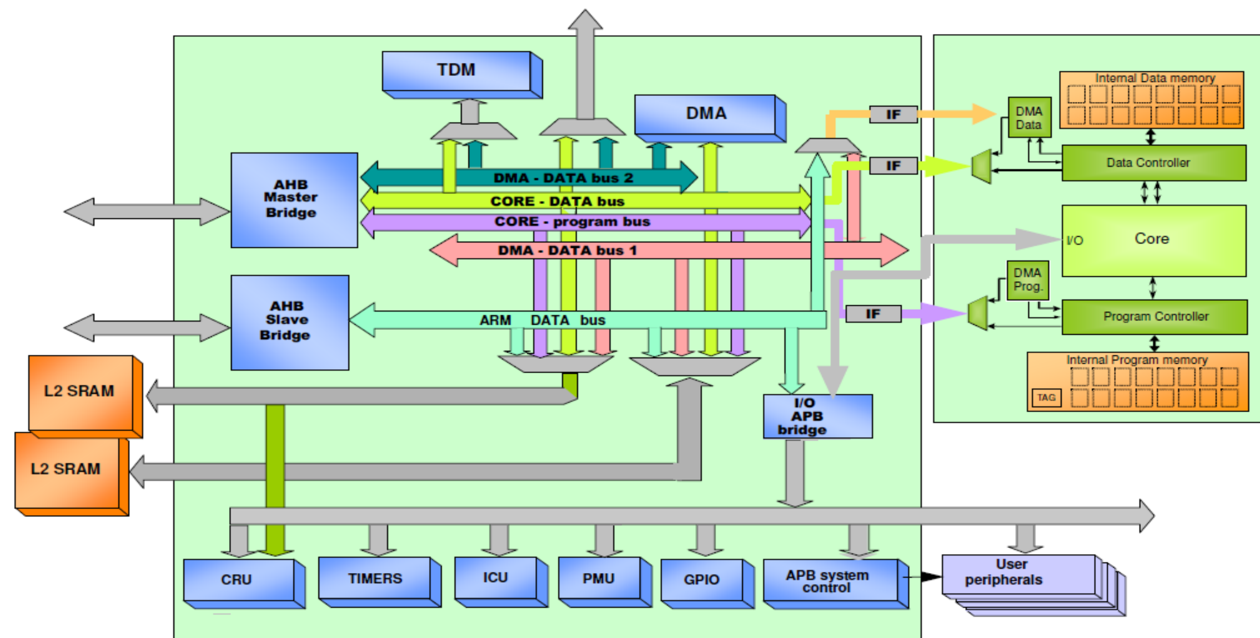
Latency-Insensitive Interfaces

- Subordinate may take variable amounts of time to process a request
- Manager must wait until subordinate's response arrives
  - Instead of pre-computing latency, or always waiting for the worst-case latency, etc.
- Common solution: VALID and READY signals
  - 1-bit signals for sender and receiver to coordinate
  - sender sets VALID signal when it is sending a message to the recipient
    - if VALID is not set, sender is not sending a message
  - READY signal indicates that recipient is ready to receive a message
    - if READY is not set, recipient is busy doing something else
  - actual message is not sent via VALID/READY, but on separate wires
  - message is sent only when VALID and READY are both high in same cycle
  - V/R used for manager to send request to subordinate
    - and for subordinate to send responses back to manager

© 2021 Arm

Communication Architecture Standards

- Why do we need communication standards?
  - Modular design approach
  - Allows design reuse
  - Facilitates IPs integration into an SoC design



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

© 2021 Arm
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 reuse
  - 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

© 2021 Arm

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	

© 2021 Arm

AMBA 3 Specifications

- AXI: Advanced eXtensible Interface
  - The most widely used AMBA interface
  - Connectivity with up to hundreds of Managers and Subordinates 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 these, 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

© 2021 Arm

AMBA 3 AXI Interface

- The AMBA 3 AXI interface specification has 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 operations
  - Efficient bridging between frequencies for power management
  - Simultaneous read and write transactions
  - Efficient support of high initial latency peripherals

© 2021 Arm
arm

AMBA 4 Specifications

- The AMBA 4 specifications add another five interface protocols to the AMBA 3 specifications:
  - ACE
  - ACE-Lite
  - AXI4
  - AXI4-Lite
  - AXI4-Stream
- The AXI and ACE protocol specifications Issue E, released February 2013, adds new optional properties for AXI ordering, ACE cache behavior, and Armv8 DVM messaging.

© 2021 Arm

AMBA 4 Specifications

- AXI4
  - Update for AXI3 to enhance the performance and utilization of the interconnect when used by multiple Managers
  - 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 a 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

© 2021 Arm

AMBA 4 Specifications

- AXI4-stream
  - Designed for unidirectional data transfers from Manager to Subordinate 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

© 2021 Arm

AXI Components and Topology

- Manager component
  - A component that initiates transactions
- Subordinate component
  - A component that receives transactions and responds to them
  - Subordinate components include memory Subordinate components and peripheral Subordinate components
- Interconnect component
  - A component with more than one AMBA interface that connects one or more Manager components to one or more Subordinate components
  - An interconnect component can be used to group together either:
    - a set of Managers so that they appear as a single Manager interface
    - a set of Subordinates so that they appear as a single Subordinate interface

© 2021 Arm

What kind of components will our datapath and memory be?

datapath and memory are both managers

0%

datapath and memory are both subordinates

0%

datapath is a manager, memory is a subordinate

0%

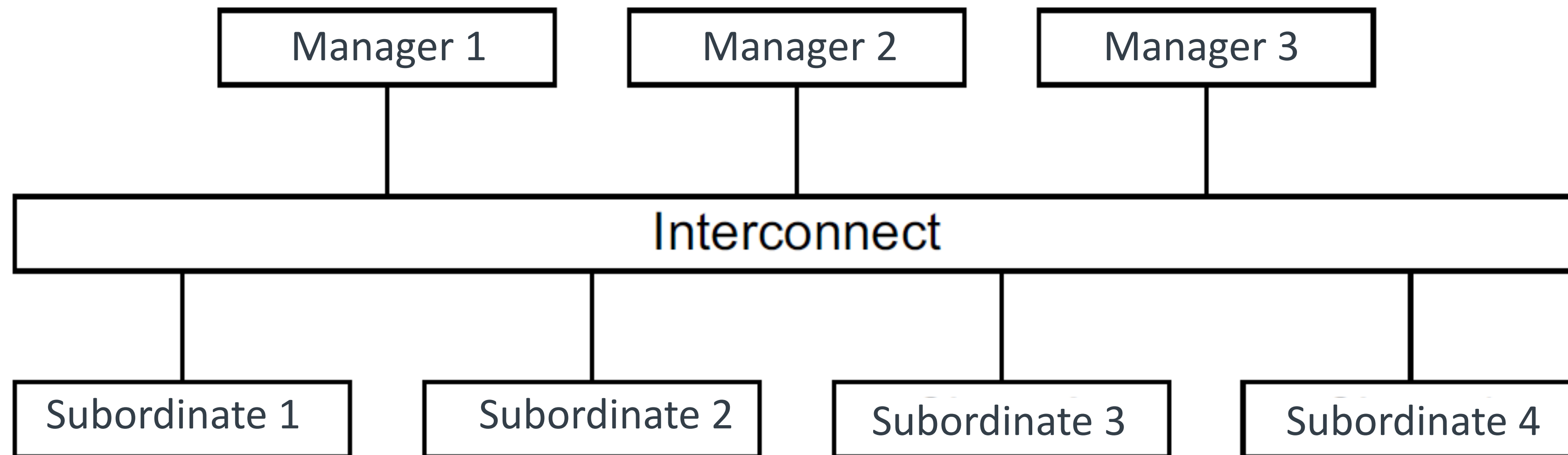
datapath is a subordinate, memory is a manager

0%

Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**

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



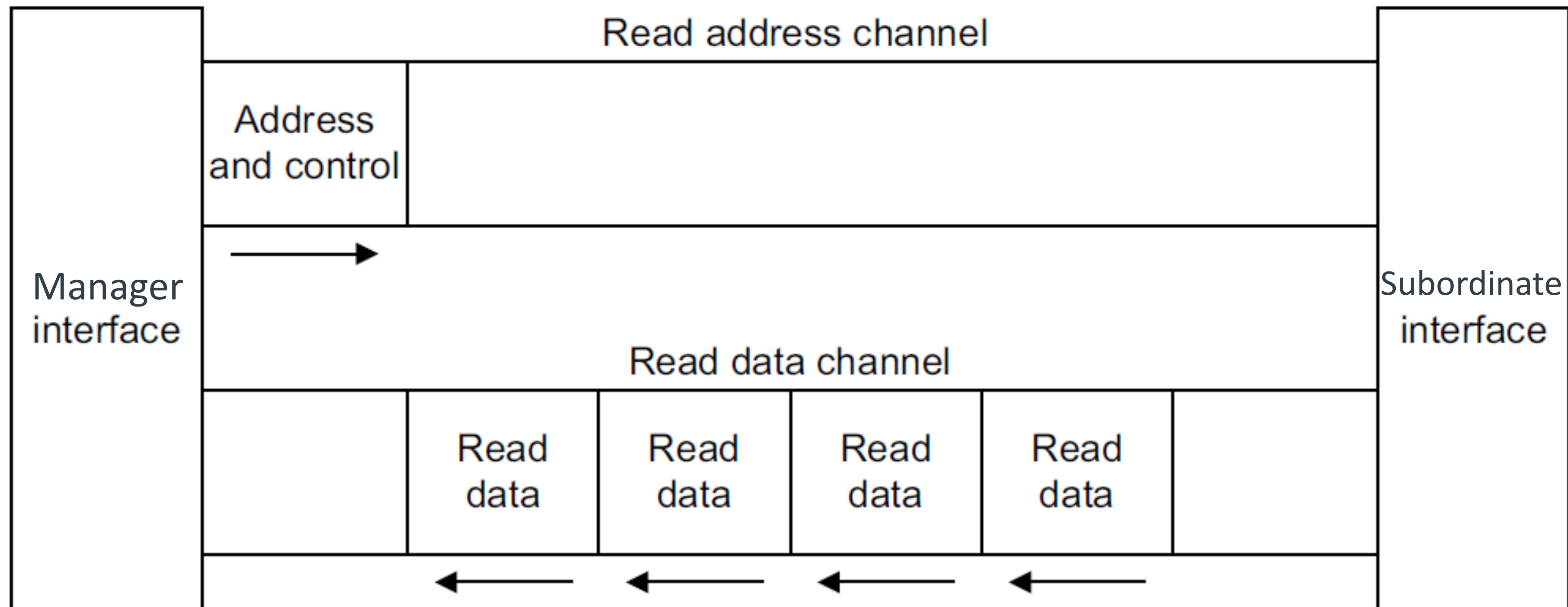
© 2021 Arm

Transaction Channels

- When an AXI Manager initiates an AXI operation targeting an AXI Subordinate,
  - 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)

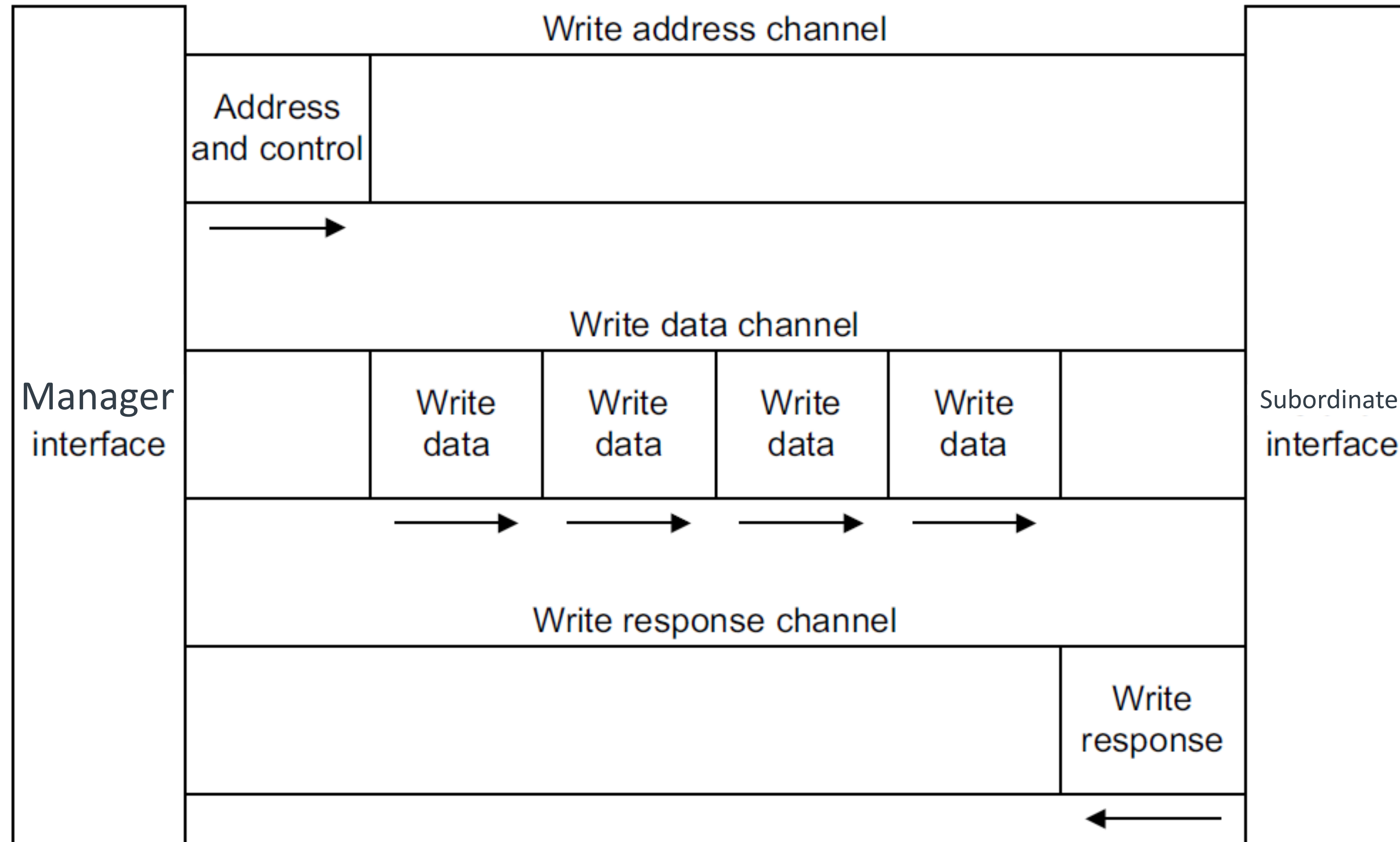
© 2021 Arm

Channel Architecture of Reads



© 2021 Arm
arm

Channel Architecture of Writes



© 2021 Arm
arm

Basic AXI4 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, 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 is to indicate the transfer of the final data item in a transaction (data channels).
  - No LAST signal in AXI-Lite

© 2021 Arm

AMBA AXI4-Lite

- AXI4-Lite:
  - Suitable for simpler control interfaces, register-style
  - Light version of full AXI4
- Key features:
  - All transactions are of burst length one
    - Hence, no need for LAST signals
  - All data accesses are based on full-width data bus (AXI4-Lite supports a data bus of 32-bit width or 64-bit width)
  - All accesses are non-modifiable, Non-bufferable
  - No support of exclusive accesses

© 2021 Arm

In AXI4, which signal must be raised first?

VALID

0%

READY

0%

either one can be raised first

0%

Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**

AXI4-Lite Signals

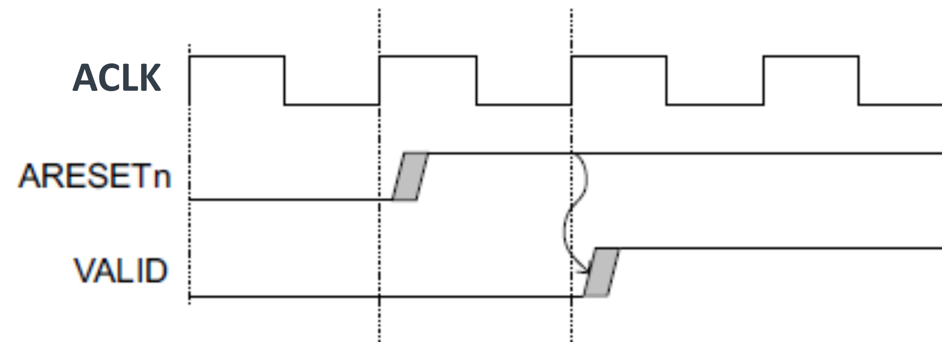
Global	Read Address	Read Data	Write Address	Write Data	Write Response
ACLK	ARVALID	RVALID	AWVALID	WVALID	BVALID
ARESETn	ARREADY	RREADY	AWREADY	WREADY	BREADY
	ARADDR	RDATA	AWADDR	WDATA	BRESP
	ARPROT	RRESP	AWPROT	WSTRB	

© 2021 Arm

arm

Clock and Reset

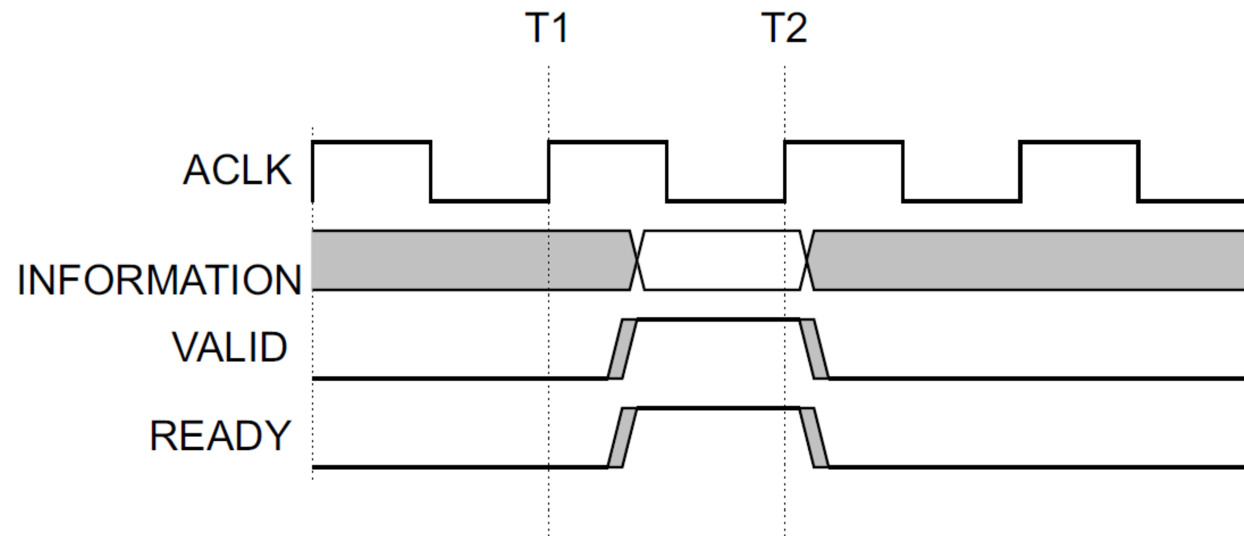
- Clock
  - Each AXI component uses a single clock signal, ACLK.
  - All input signals are sampled on the rising edge of ACLK.
  - All output signal changes must occur after the rising edge of ACLK.
- Reset
  - A single active-LOW reset signal, ARESETn
  - Can be asserted asynchronously, but de-assertion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



© 2021 Arm
arm

Channel Timing Example: VALID with READY Handshake

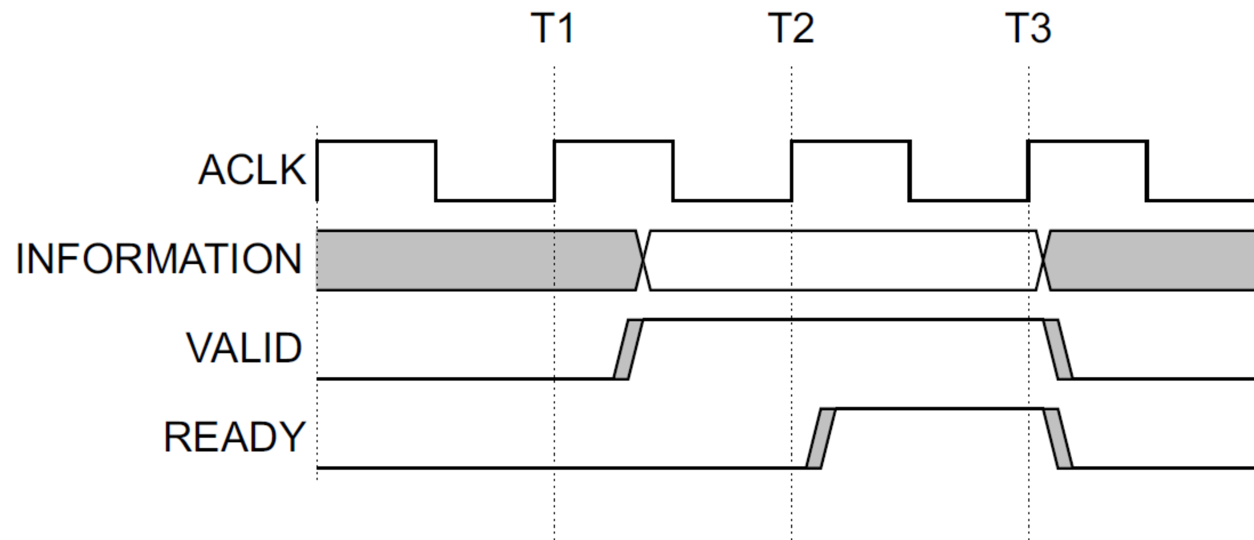
- After T1, both the source and destination indicate a data transfer.
- The transfer occurs at the rising clock edge (after both VALID and READY signals are asserted).
- The transfer occurs at T2.



© 2021 Arm
arm

Channel Timing Example: VALID before READY Handshake

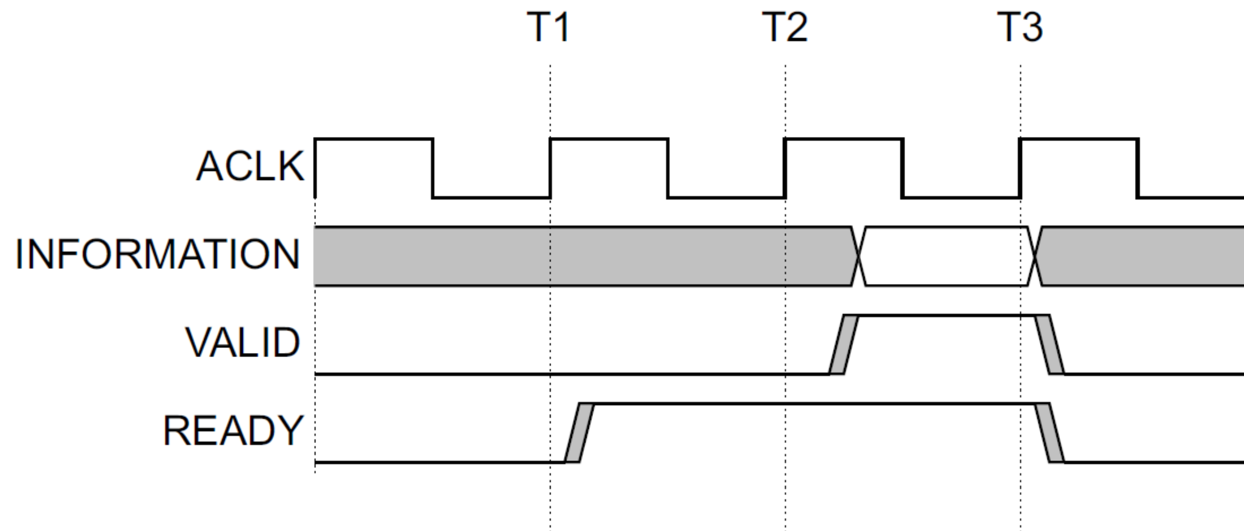
- After T1, the source presents the address, data, or control information and asserts the VALID signal.
- The destination asserts the READY signal after T2.
- The source has to keep its information stable until the transfer occurs at T3.



© 2021 Arm

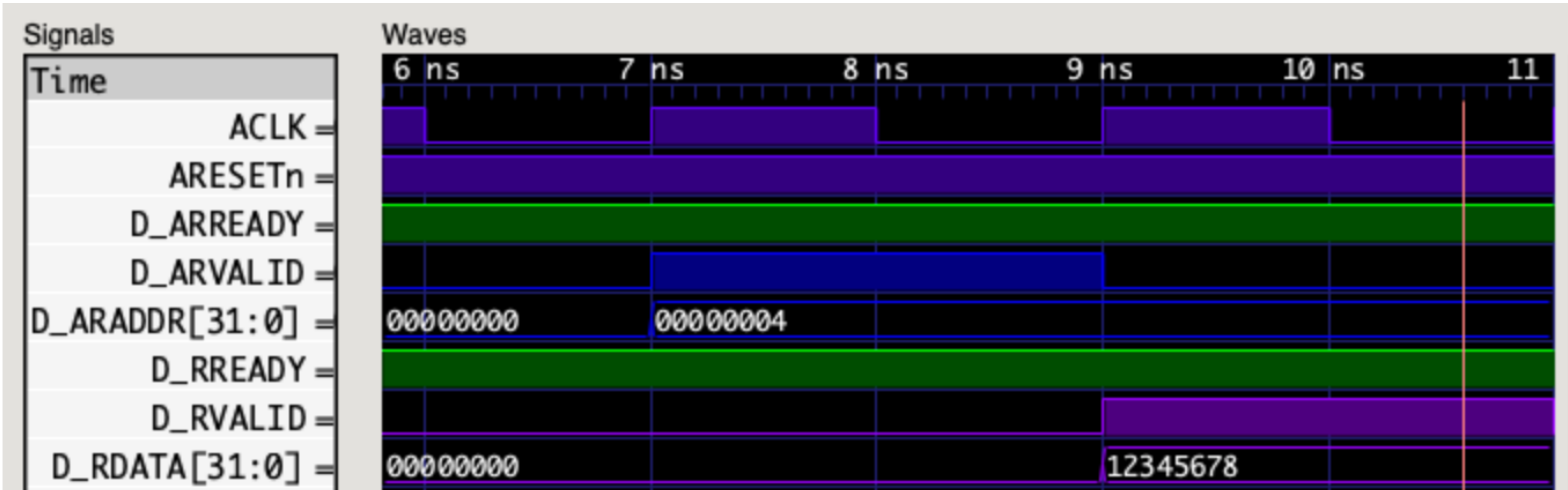
Channel Timing Example: READY before VALID Handshake

- 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).



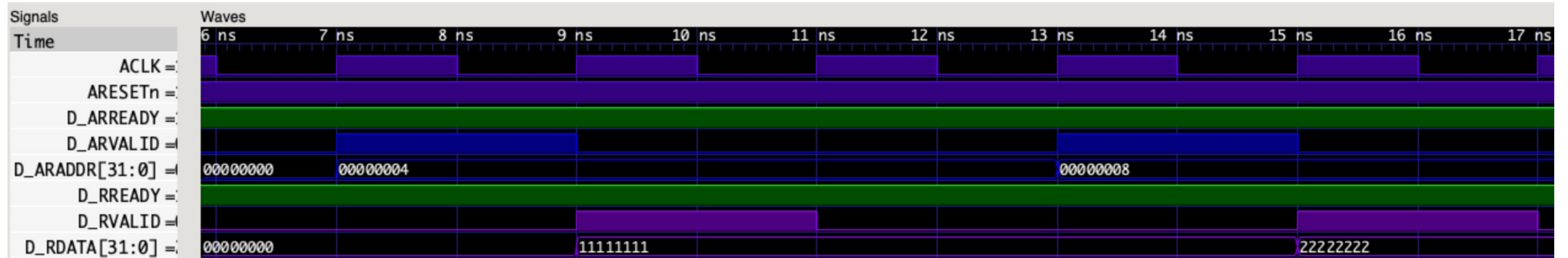
© 2021 Arm
arm

AXI Read Transaction



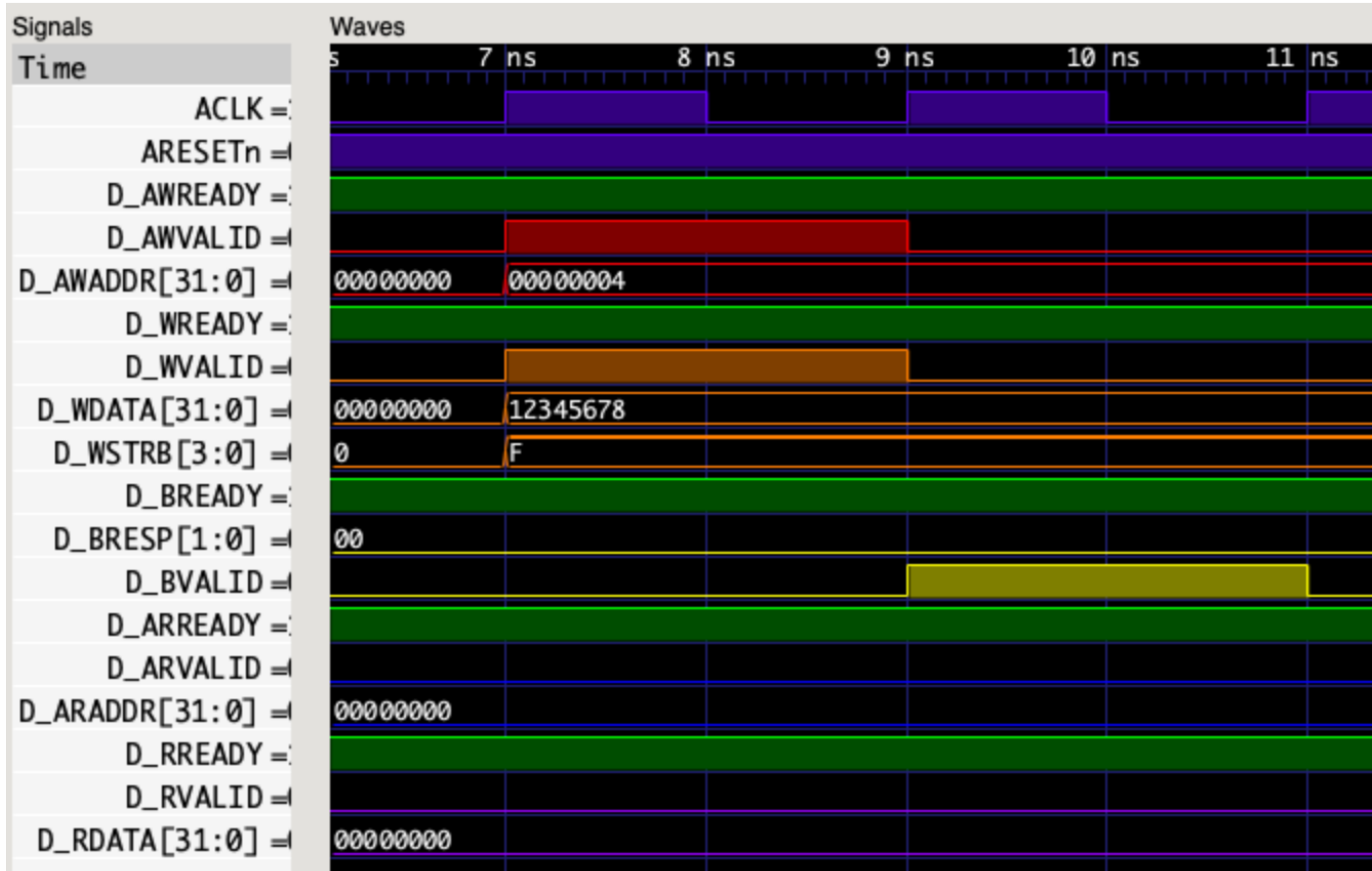
© 2021 Arm

Multiple AXI Read transactions



© 2021 Arm
arm

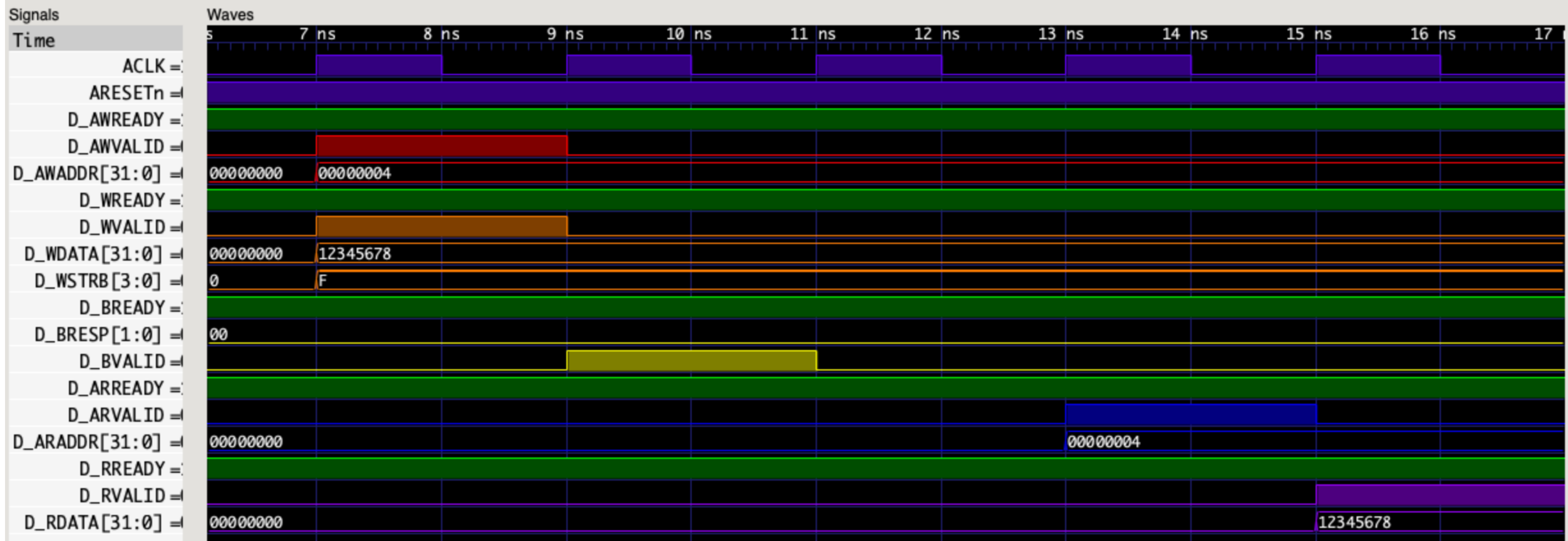
AXI Write Transaction



© 2021 Arm

arm

AXI Write + Read Transactions



© 2021 Arm

arm

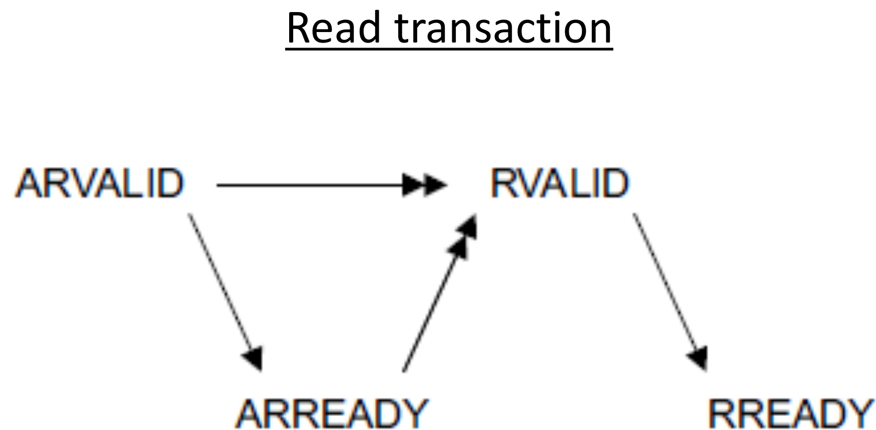
Relationships Between the Channels

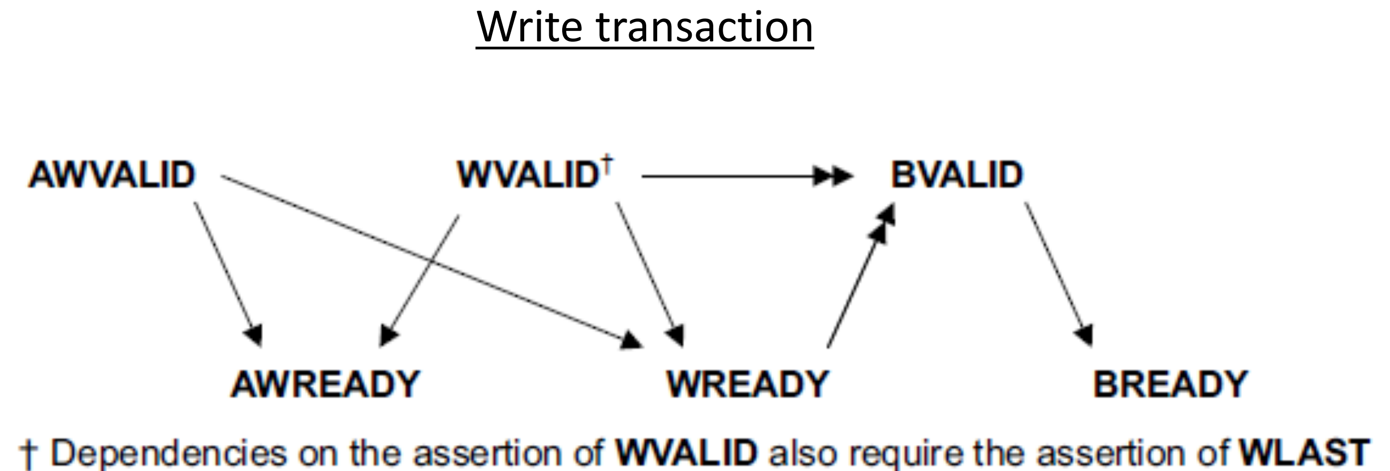
- The AXI protocol requires the following relationships to be maintained:
  - A write response must always follow the last write transfer in the write transaction of which it is a part.
  - Read data must always follow the address to which the data relates.
  - Channel handshakes must conform to the dependencies defined for the handshake signals.
- Dependencies are shown in the next slide.

© 2021 Arm
arm

Relationships Between the Channels

- Dependency rules between the handshake signals that must be observed:
  - The VALID signal of the AXI interface sending information must not depend 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.





© 2021 Arm

HW6 notes

- AXI4-Lite spec says very little about specific timing constraints
  - that’s the whole point of latency-insensitive interfaces!
- HW6 manager/subordinate must handle back-to-back transactions
  - to avoid pipeline stalls
  - makes memory a little more complex
    - keeps pipeline simpler
  - start by building a memory that can speak AXI4-lite
    - then refine it to handle back-to-back reads and writes

© 2021 Arm