



고려대학교
KOREA UNIVERSITY

COMP427 Embedded Systems

Lecture 7. AMBA

Prof. Taeweon Suh

Computer Science & Engineering

Korea University

AMBA

- Advanced Microcontroller Bus Architecture
 - On-chip bus protocol from ARM
 - On-chip interconnect specification for the connection and management of functional blocks including processor and peripheral devices
 - Introduced in 1996
 - AMBA is a registered trademark of ARM Limited.
 - AMBA is an open standard

AMBA History

- **AMBA**

- ASB
- APB

- **AMBA 2** (1999)

- AHB
 - widely used on ARM7, ARM9 and ARM Cortex-M based designs
- ASB
- APB2 (or APB)

- ACE: AXI Coherency Extensions
- AXI: Advanced eXtensible Interface
- AHB: Advanced High-performance Bus
- ASB: Advanced System Bus
- APB: Advanced Peripheral Bus
- ATB: Advanced Trace Bus

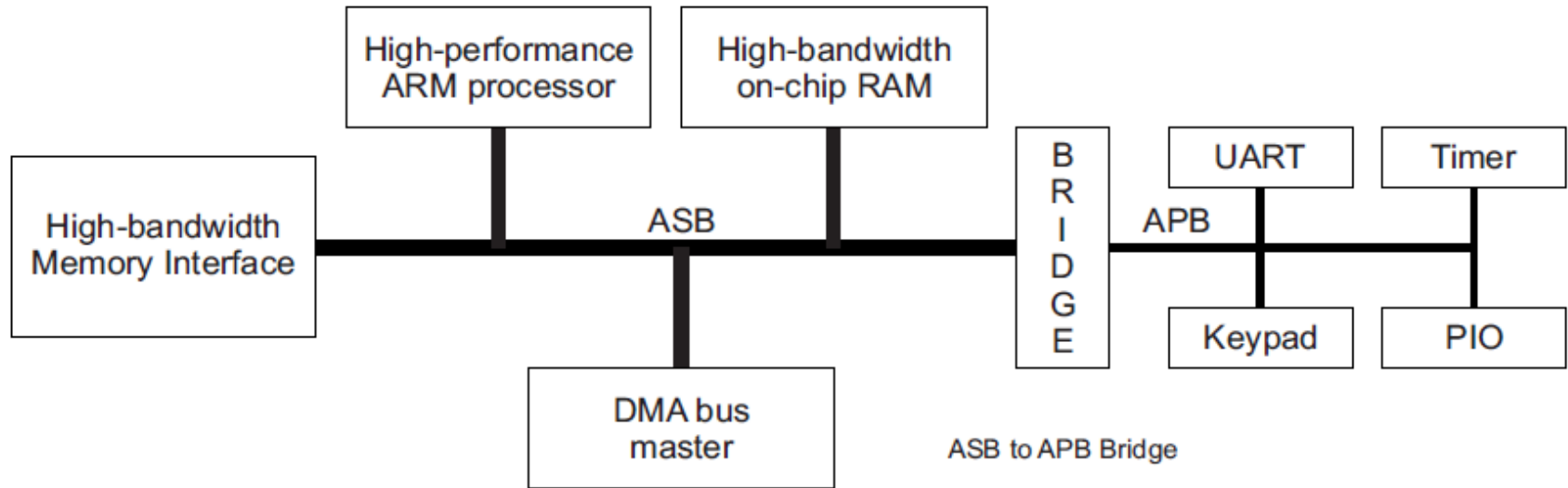
- **AMBA 3** (2003)

- AXI3 (or AXI v1.0)
 - widely used on ARM Cortex-A processors including Cortex-A9
- AHB-Lite v1.0
- APB3 v1.0
- ATB v1.0

- **AMBA 4** (2010)

- ACE
 - widely used on the latest ARM Cortex-A processors including Cortex-A7 and Cortex-A15
- ACE-Lite
- AXI4
- AXI4-Lite
- AXI-Stream v1.0
- ATB v1.1
- APB4 v2.0

ASB



AMBA Advanced System Bus (ASB)

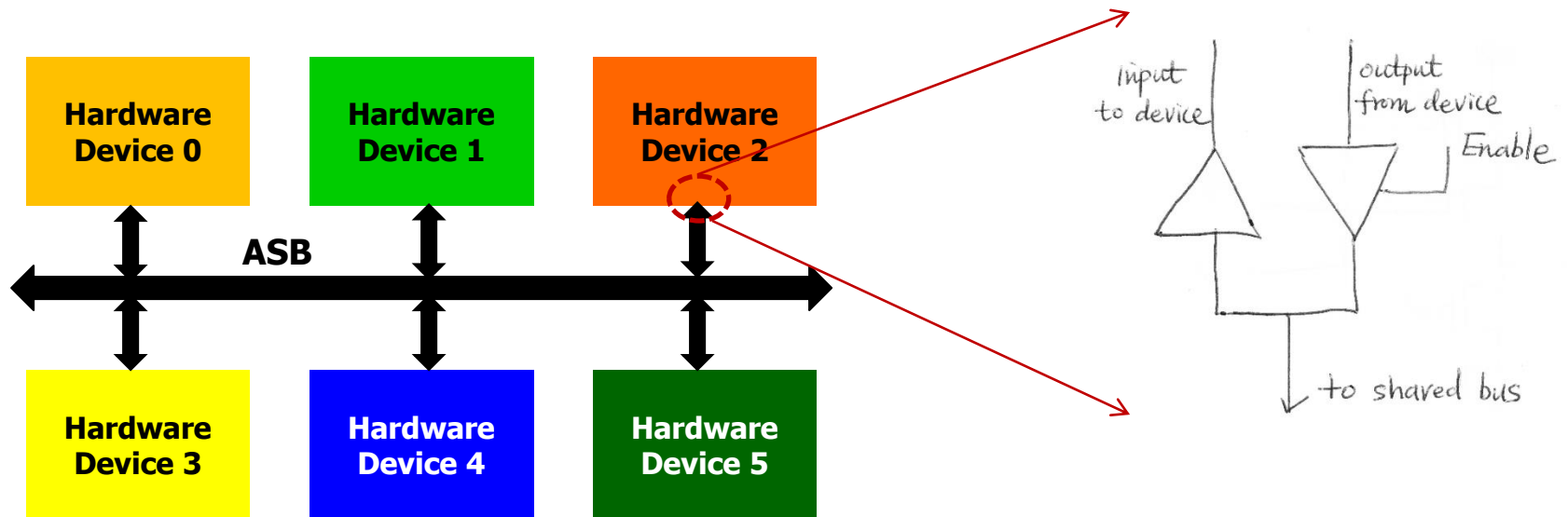
- * High performance
- * Pipelined operation
- * Burst transfers
- * Multiple bus masters

AMBA Advanced Peripheral Bus (APB)

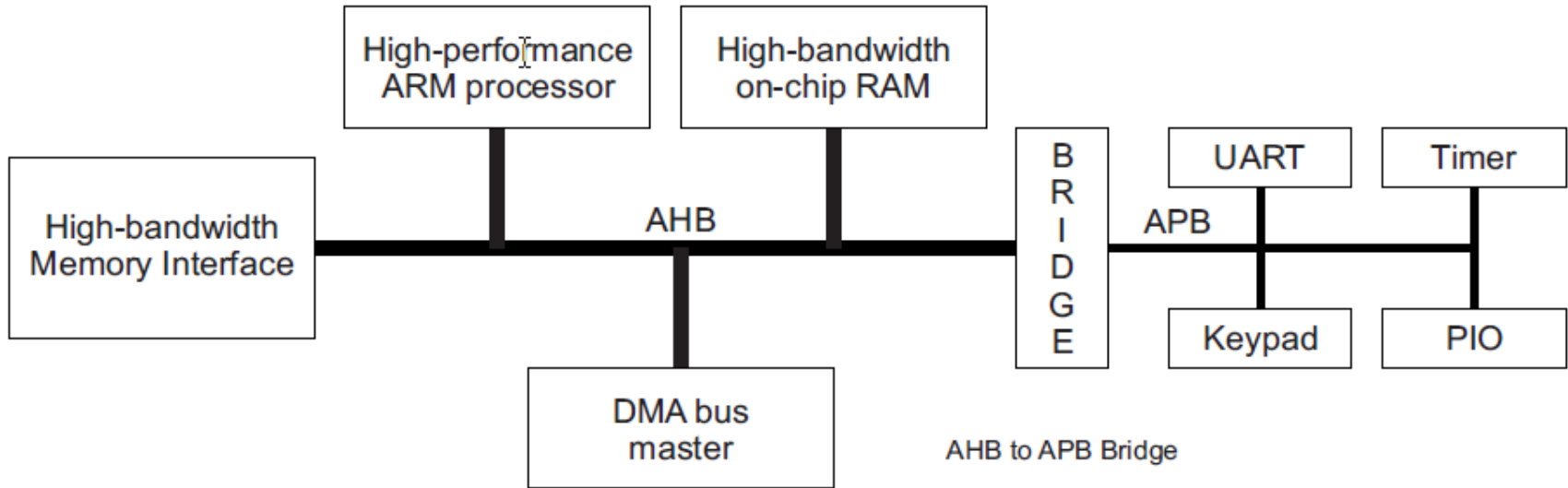
- * Low power
- * Latched address and control
- * Simple interface
- * Suitable for many peripherals

Figure 4-1 A typical AMBA system

ASB



AHB



AMBA Advanced High-performance Bus (AHB)

- * High performance
- * Pipelined operation
- * Burst transfers
- * Multiple bus masters
- * Split transactions

AMBA Advanced Peripheral Bus (APB)

- * Low power
- * Latched address and control
- * Simple interface
- * Suitable for many peripherals

Figure 3-1 A typical AMBA AHB-based system

AHB with 3 Masters and 4 Slaves

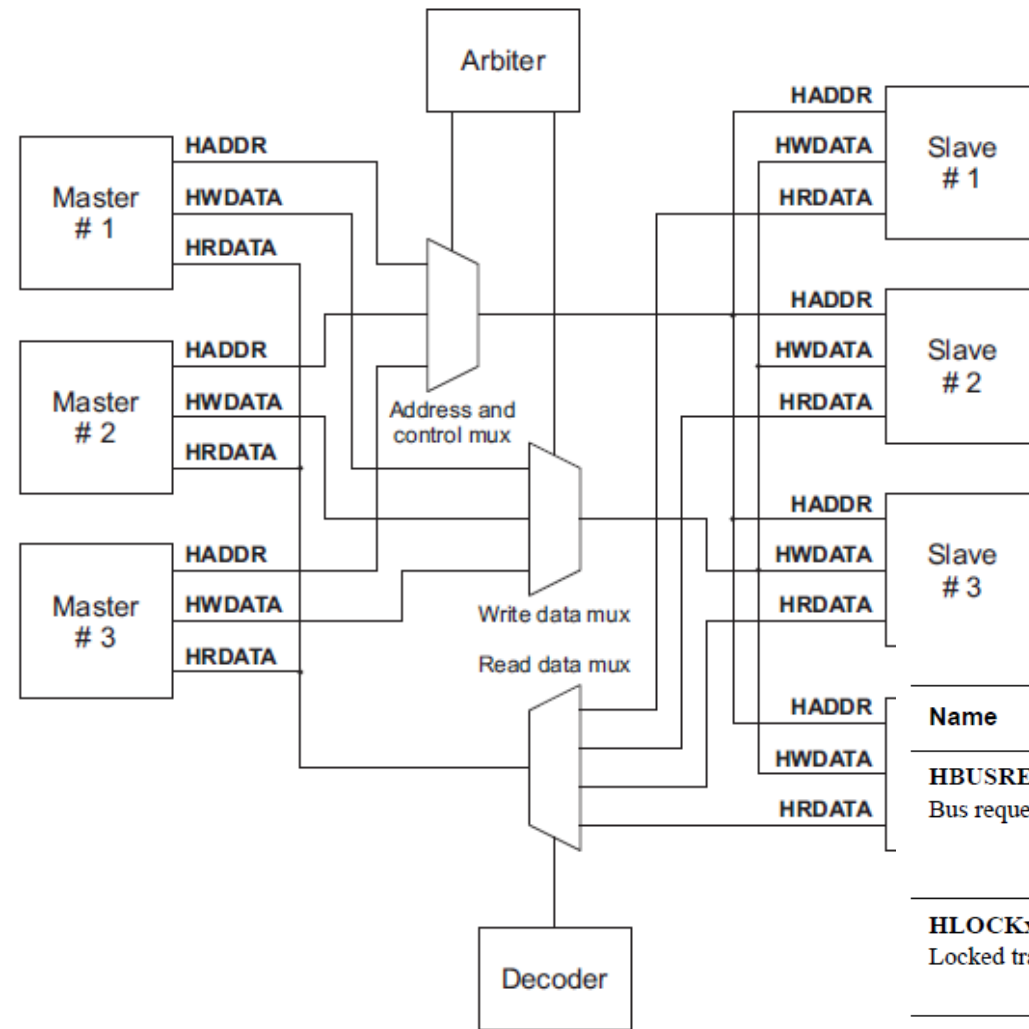


Figure 3-2 Multiplexor int

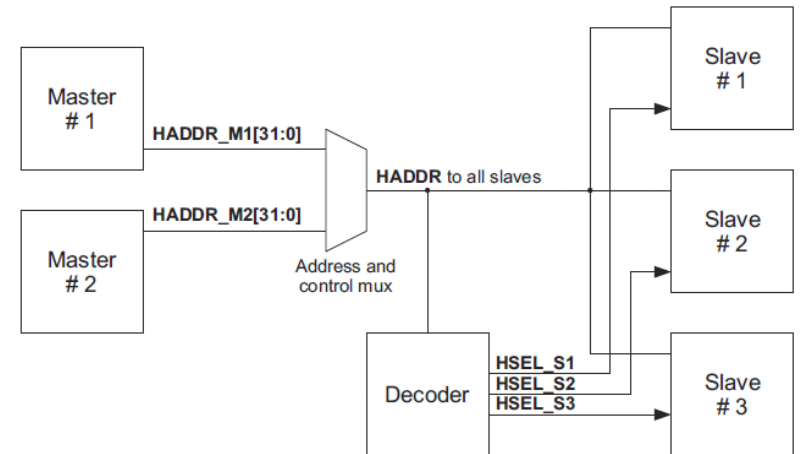


Figure 3-12 Slave select signals

Table 2-2 Arbitration signals

Name	Source	Description
HBUSREQx Bus request	Master	A signal from bus master x to the bus arbiter which indicates that the bus master requires the bus. There is an HBUSREQx signal for each bus master in the system, up to a maximum of 16 bus masters.
HLOCKx Locked transfers	Master	When HIGH this signal indicates that the master requires locked access to the bus and no other master should be granted the bus until this signal is LOW.
HGRANTx Bus grant	Arbiter	This signal indicates that bus master x is currently the highest priority master. Ownership of the address/control signals changes at the end of a transfer when HREADY is HIGH, so a master gets access to the bus when both HREADY and HGRANTx are HIGH.

■ "H" indicates AHB signals

AHB Basic Transfer Example with Wait

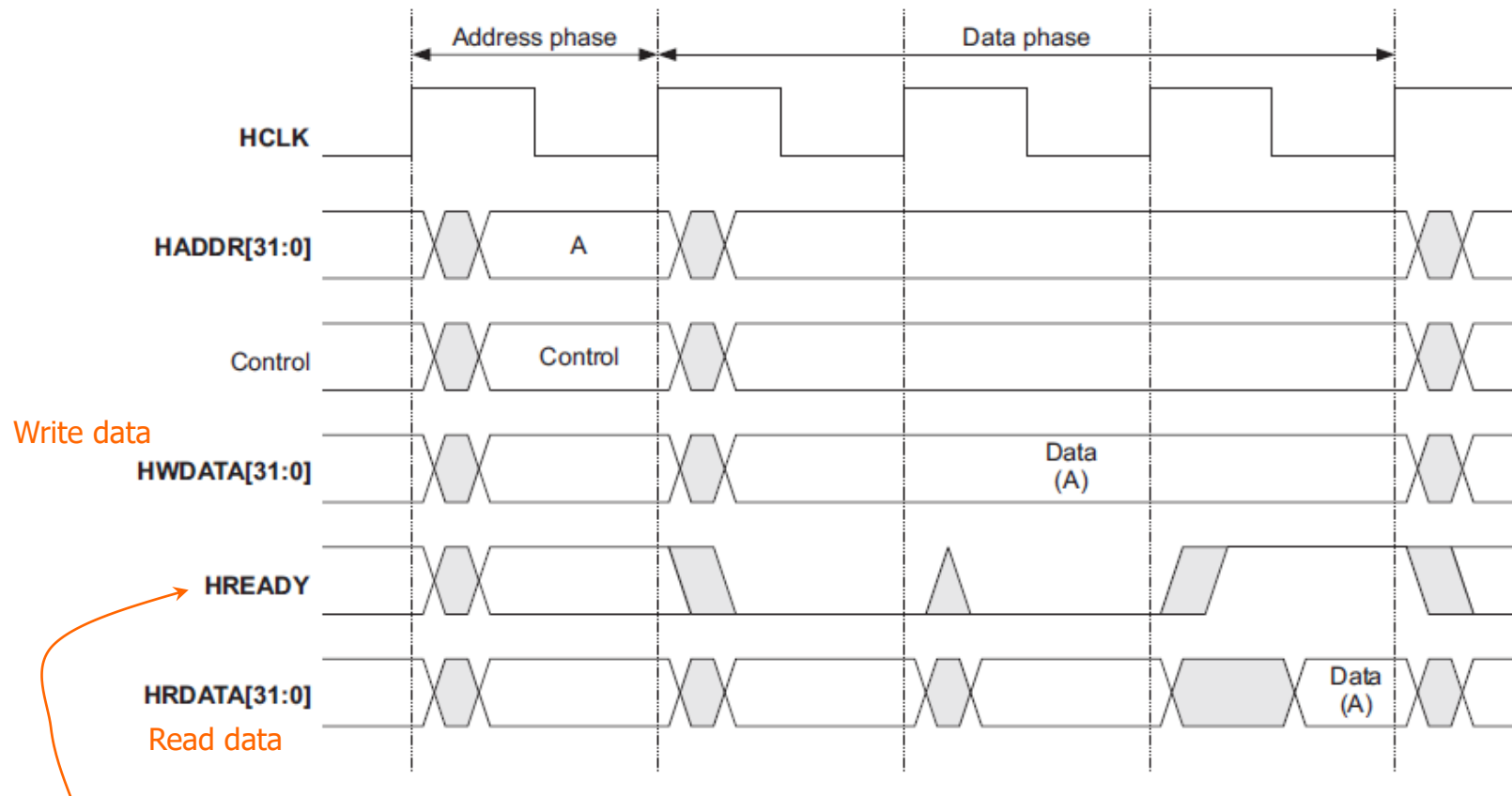


Figure 3-4 Transfer with wait states

AHB Burst Transfer Example

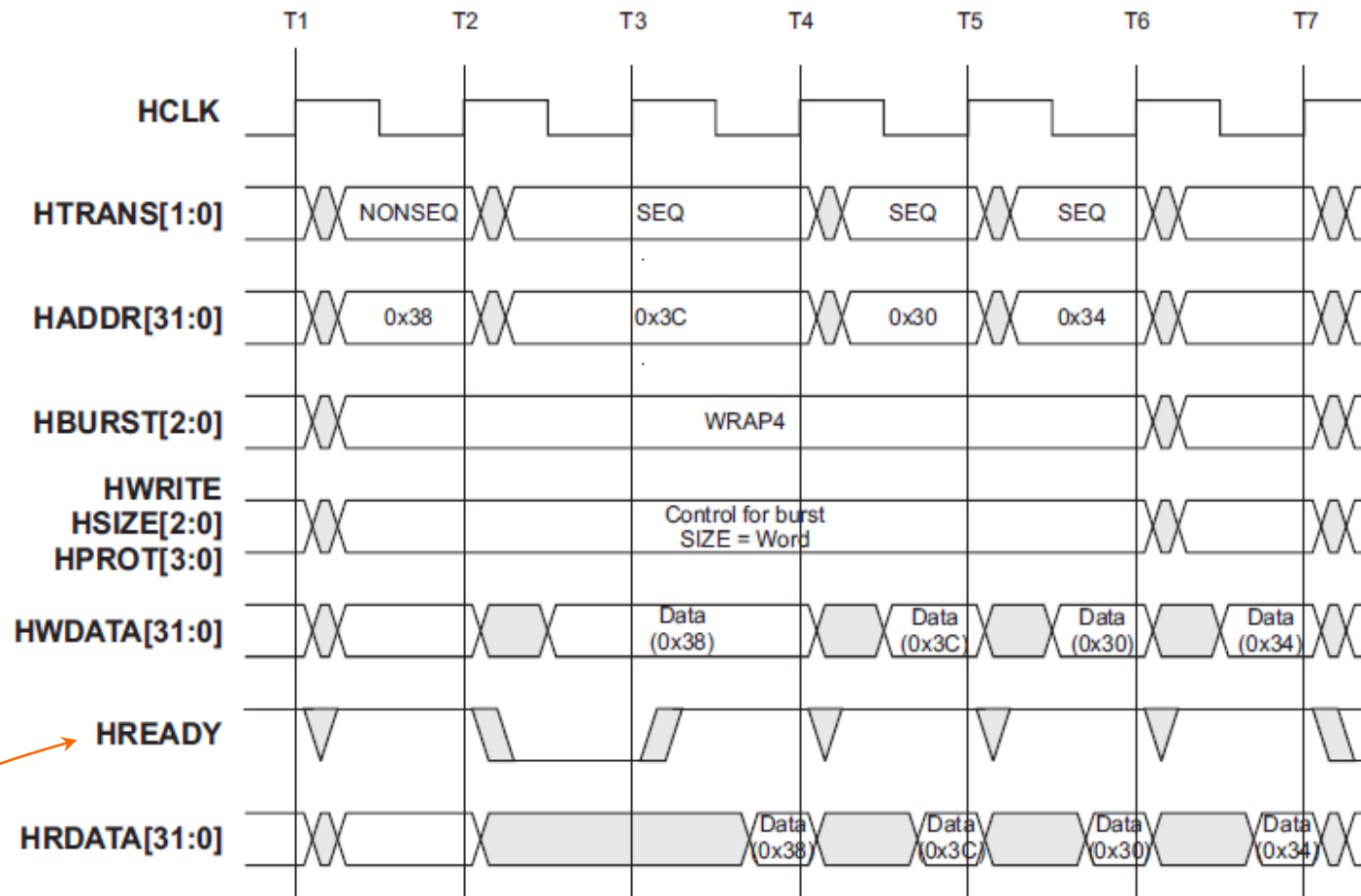


Figure 3-7 Four-beat wrapping burst

HREADY Source: Slave

AHD Split Transaction

- If slave decides that it may take a number of cycles to obtain and provide data, it gives a SPLIT transfer response
- Arbiter grants use of the bus to other masters

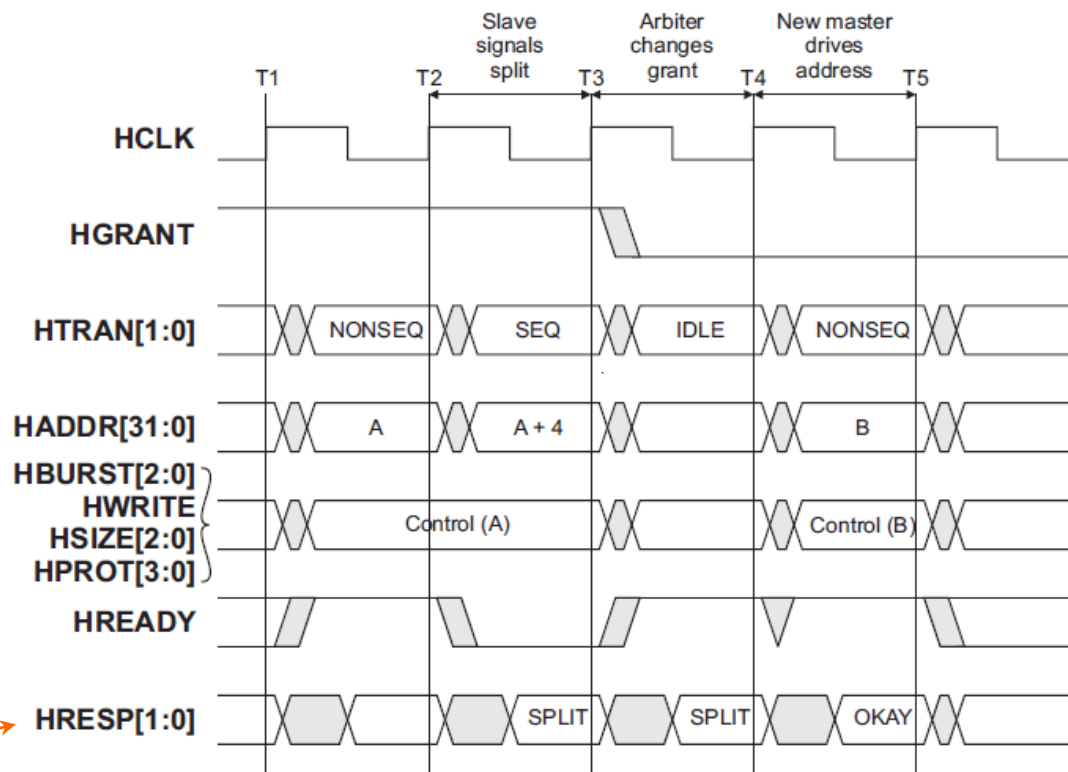


Figure 3-20 Handover after split transfer

HRESP: Transfer response from slave (OKAY, ERROR, RETRY, and SPLIT)

APB Write/Read

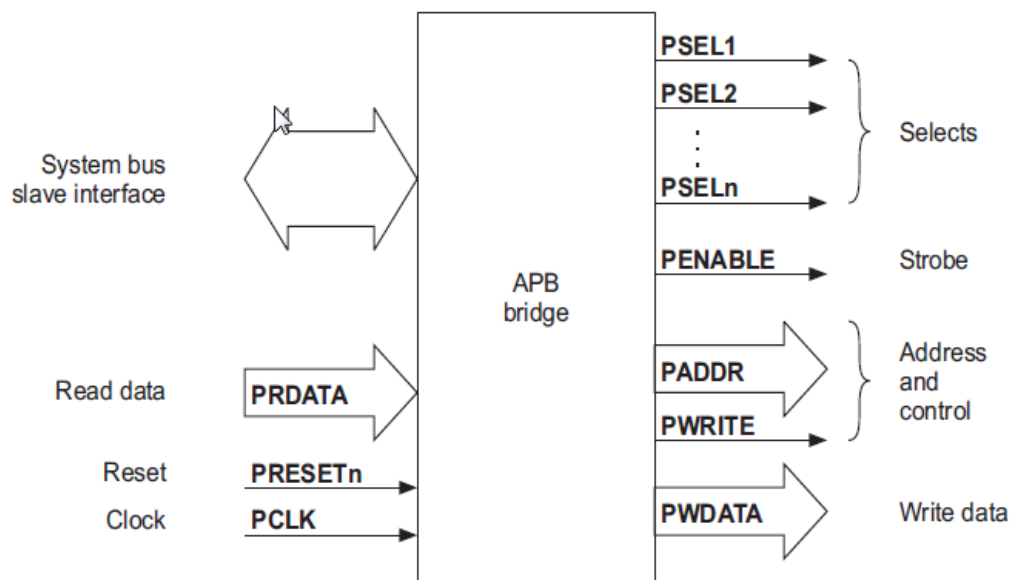


Figure 5-5 APB bridge interface diagram

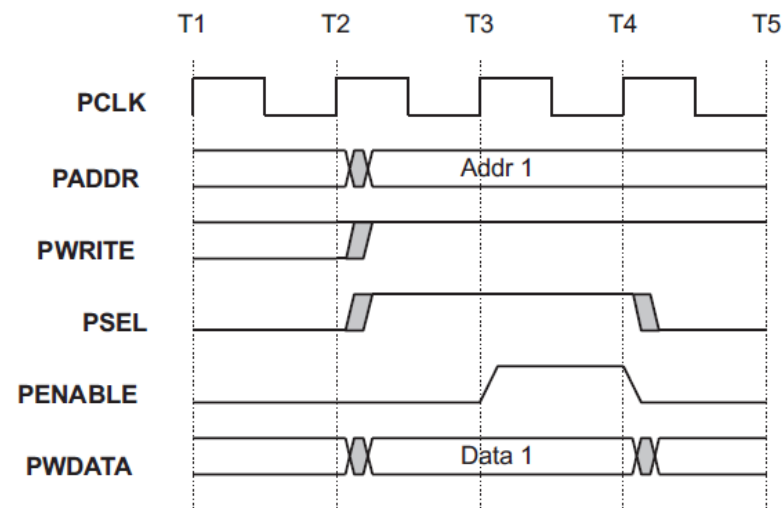


Figure 5-3 Write transfer

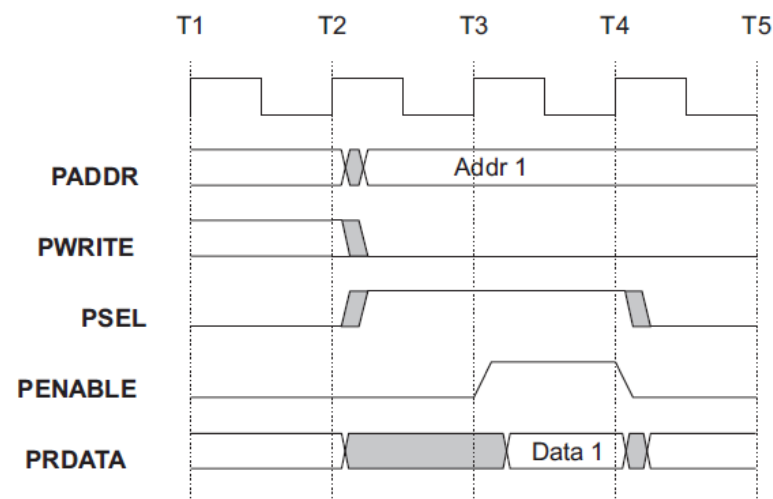


Figure 5-4 Read transfer

AXI v1.0

- AMBA AXI protocol is targeted at high-performance, high-frequency system designs
- AXI key features
 - Separate address/control and data phases
 - Support for unaligned data transfers using byte strobes
 - Separate read and write data channels to enable low-cost Direct Memory Access (DMA)
 - Ability to issue multiple outstanding addresses
 - Out-of-order transaction completion
 - Easy addition of register stages to provide timing closure

5 Independent Channels

- **Read address channel** and **Write address channel**
 - Variable length burst: 1 ~ 16 data transfers
 - Burst with a transfer size of 8 ~ 1024 bits (1B ~ 128B)
- **Read data channel**
 - Convey data and any read response info.
 - Data bus can be 8, 16, 32, 64, 128, 256, 512, or 1024 bits
- **Write data channel**
 - Data bus can be 8, 16, 32, 64, 128, 256, 512, or 1024 bits
- **Write response channel**
 - Write response info.

AXI Read Operation

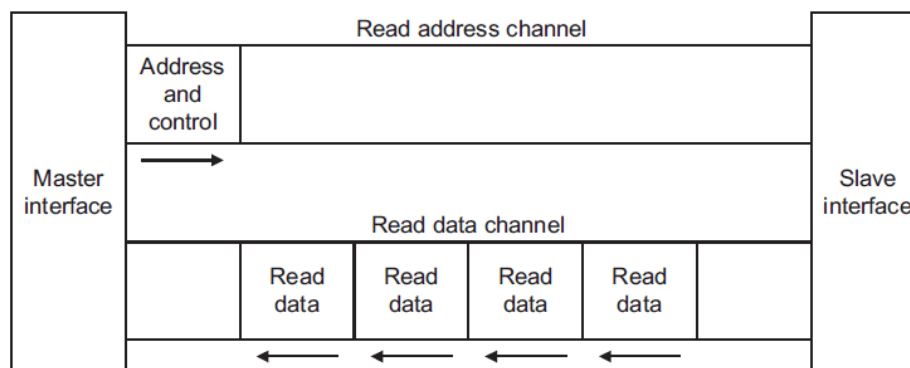
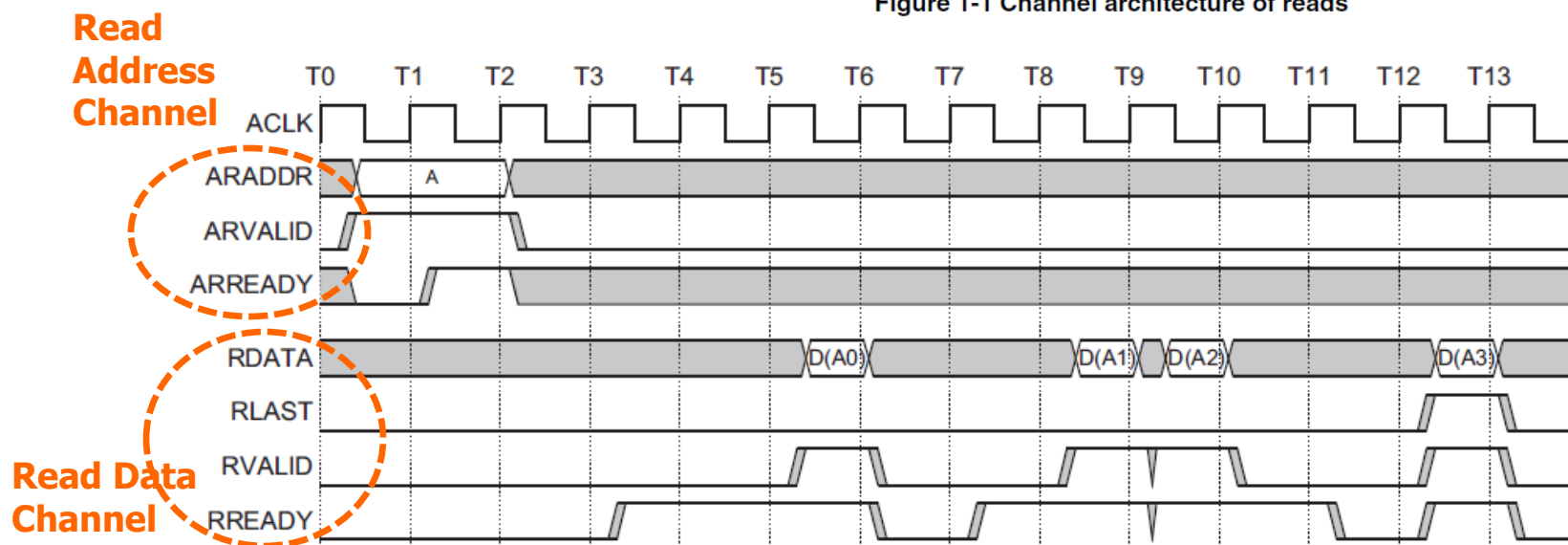


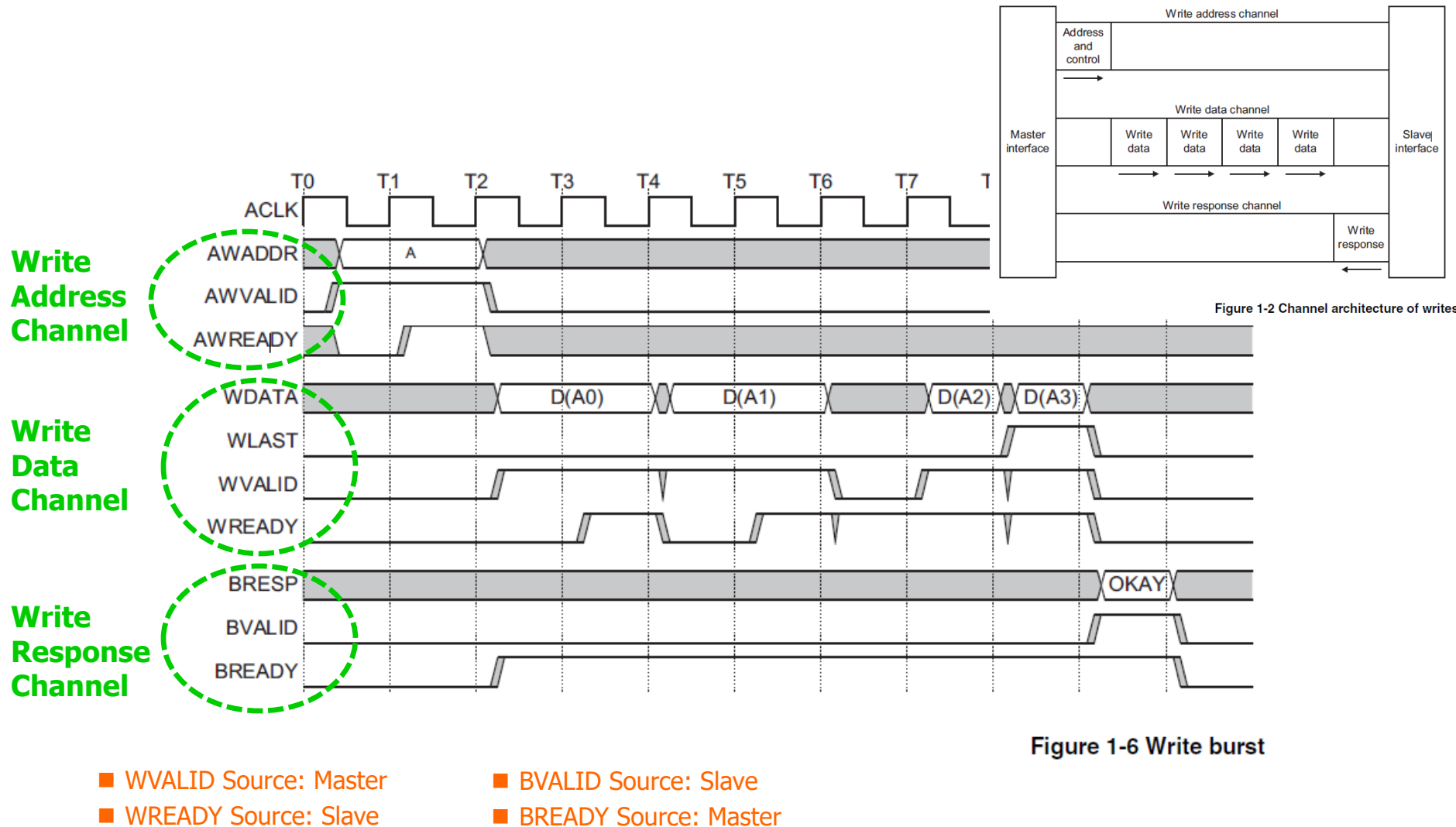
Figure 1-1 Channel architecture of reads



RREADY: From master, indicate that master can accept the read data and response info.

Figure 1-4 Read burst

AXI Write Operation



Out-of-order Completion

- AXI gives an ID tag to every transaction
 - Transactions with the same ID are completed in order
 - Transactions with different IDs can be completed out of order

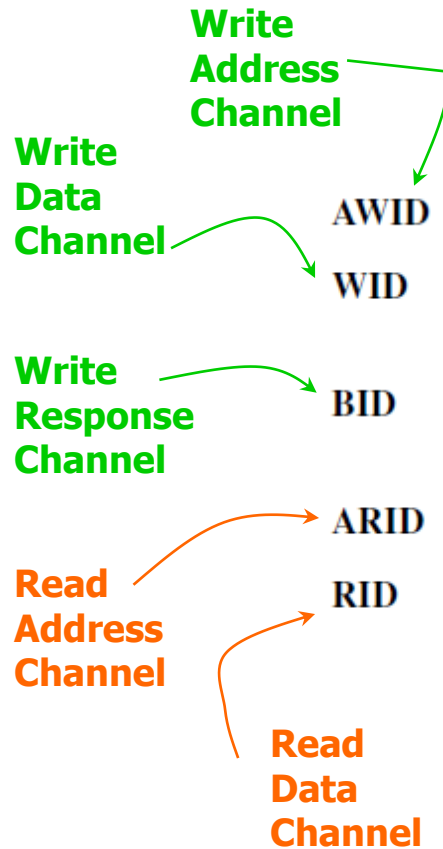
Table 2-2 Write address channel signals

Signal	Source	Description
AWID[3:0]	Master	Write address ID. This signal is the identification tag for the write address group of signals.
AWADDR[31:0]	Master	Write address. The write address bus gives the address of the first transfer in a write burst transaction. The associated control signals are used to determine the addresses of the remaining transfers in the burst.

Table 2-5 Read address channel signals

Signal	Source	Description
ARID[3:0]	Master	Read address ID. This signal is the identification tag for the read address group of signals.
ARADDR[31:0]	Master	Read address. The read address bus gives the initial address of a read burst transaction. Only the start address of the burst is provided and the control signals that are issued alongside the address detail how the address is calculated for the remaining transfers in the burst.

ID Signals



The ID tag for the write address group of signals.

The write ID tag for a write transaction. Along with the write data, the master transfers a **WID** to match the **AWID** of the corresponding address.

The ID tag for the write response. The slave transfers a **BID** to match the **AWID** and **WID** of the transaction to which it is responding.

The ID tag for the read address group of signals.

The read ID tag for a read transaction. The slave transfers an **RID** to match the **ARID** of the transaction to which it is responding.

Out-of-order Completion

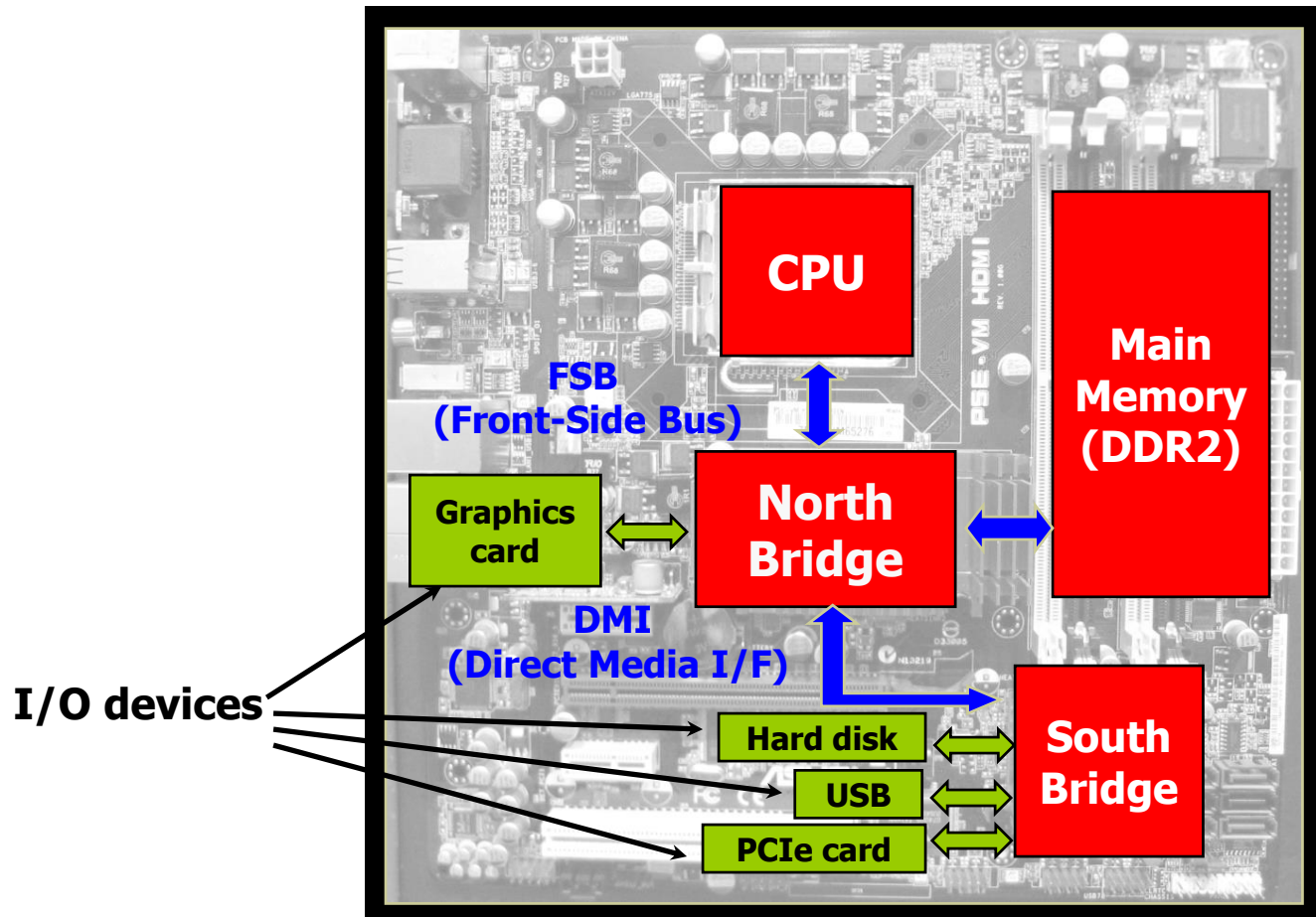
- Out-of-order transactions can improve system performance in 2 ways
 - Fast-responding slaves respond in advance of earlier transactions with slower slaves
 - Complex slaves can return data out of order
 - A data item for a later access might be available before the data for an earlier access is available
- If a master requires that transactions are completed in the same order that they are issued, they must all have the same ID tag
- It is not a required feature
 - Simple masters and slaves can process one transaction at a time in the order they are issued

Addition of Register Slices

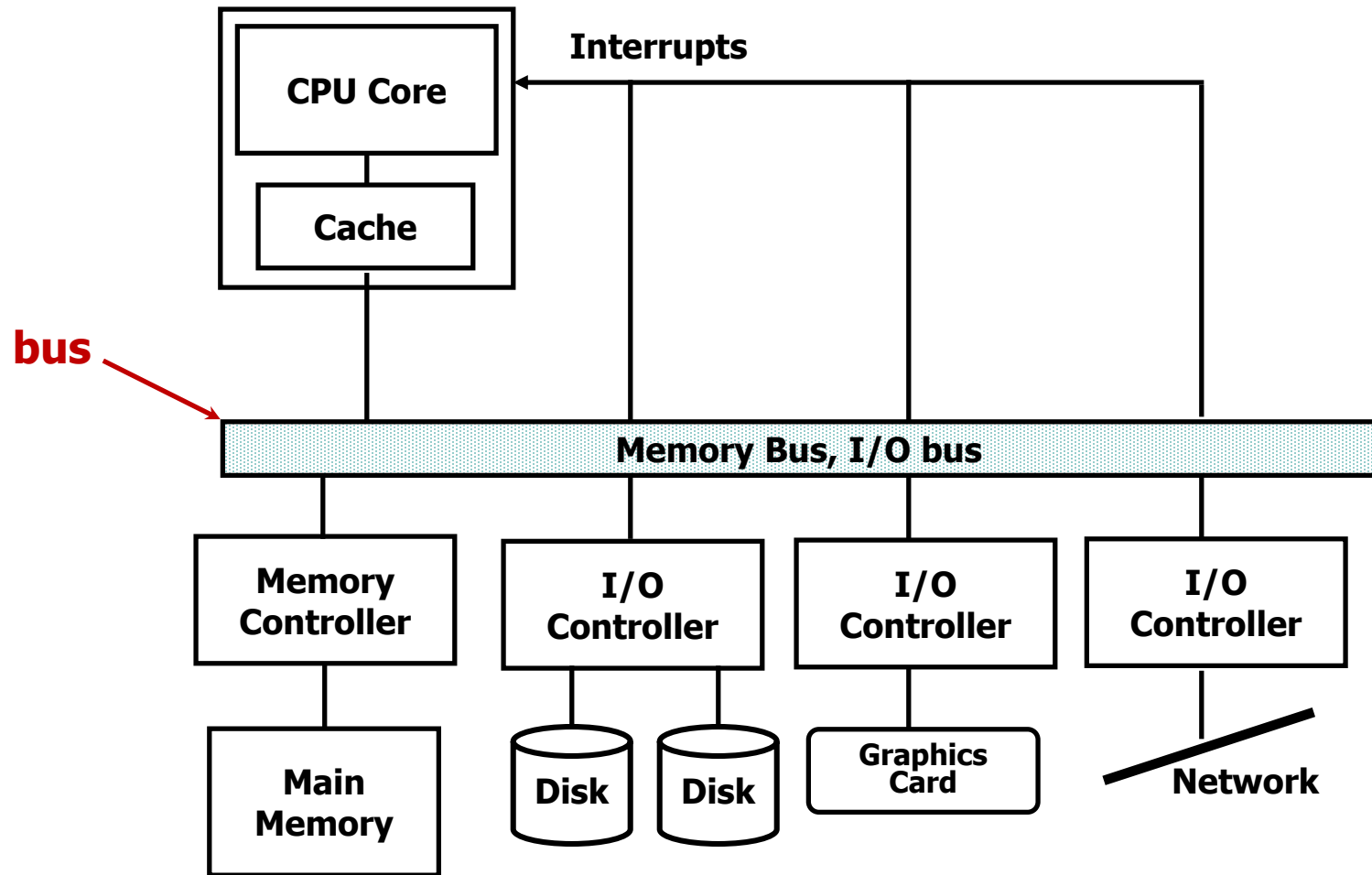
- AXI enables the insertion of a register slice in any channel at the cost of an additional cycle latency
 - Trade-off between latency and maximum frequency
- It can be advantageous to use
 - Direct and fast connection between a processor and high-performance memory
 - Simple register slices to isolate a longer path to less performance-critical peripherals

Backup Slides

A Computer System



A Typical I/O System Schematic (Simplified)



I/O Interconnection

- A **bus** is a shared communication link
 - A single set of wires used to connect multiple components
 - Composed of address bus, data bus, and control bus (read/write)
 - Advantages
 - Versatile – new devices can be added easily and can be moved between computer systems that use the same bus standard
 - Low cost – a single set of wires is shared in multiple ways
 - Disadvantages
 - Communication bottleneck – bus **bandwidth** limits the maximum I/O **throughput**
- The maximum bus speed is largely limited by
 - The **length** of the bus
 - The **number** of devices on the bus

I/O Interconnection (Cont)

- I/O devices and interconnection largely contribute to the performance of computer system
- Traditionally, parallel shared wires had (have) been used to connect I/O devices
- As the clock frequency increases for communicating with I/O devices, parallel shared wires suffer from clock skew and interference among wires
- Industry transitioned from parallel shared buses to high-speed serial point-to-point interconnections

Types of Buses

- **Processor-memory bus**

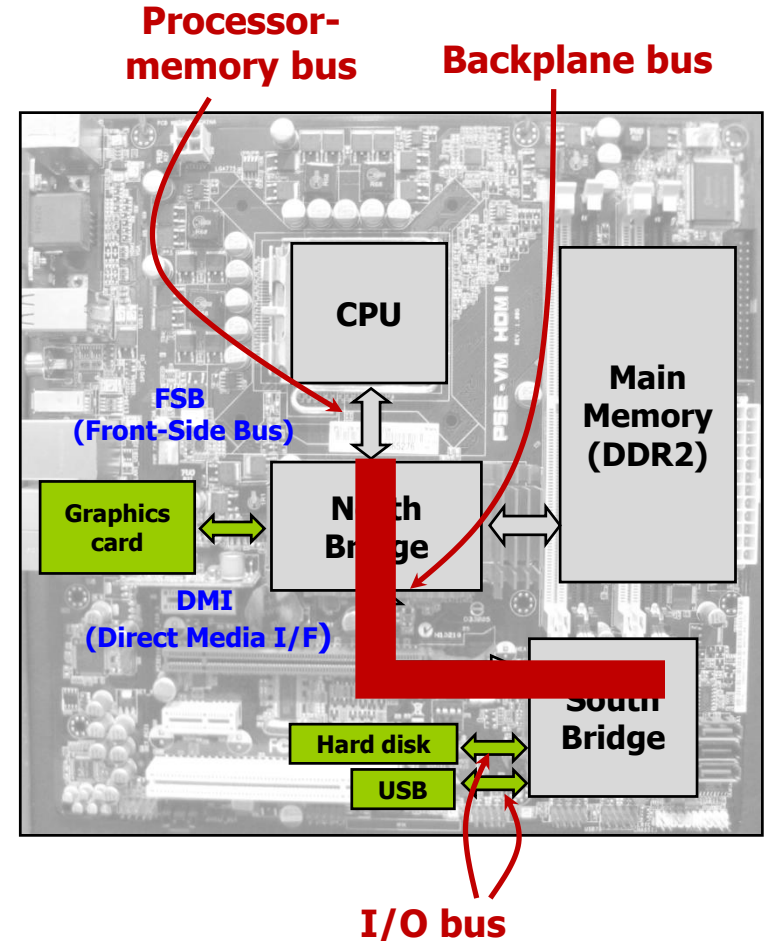
- Front Side Bus (FSB), proprietary bus
 - Replaced by QPI (QuickPath Interconnect) in Intel
 - Replaced by Hypertransport in AMD
- Short and high speed
- Matched to the memory system to maximize the memory-processor bandwidth
- Optimized for cache block transfers

- **Backplane (backbone) bus**

- Industry standard
 - e.g., PCIexpress
- Allow processor, memory and I/O devices to coexist on a single bus
- Used as an intermediary bus connecting I/O busses to the processor-memory bus

- **I/O bus**

- Industry standard
 - e.g., SATA, USB, Firewire
- Usually is lengthy and slower
- Needs to accommodate a wide range of I/O devices

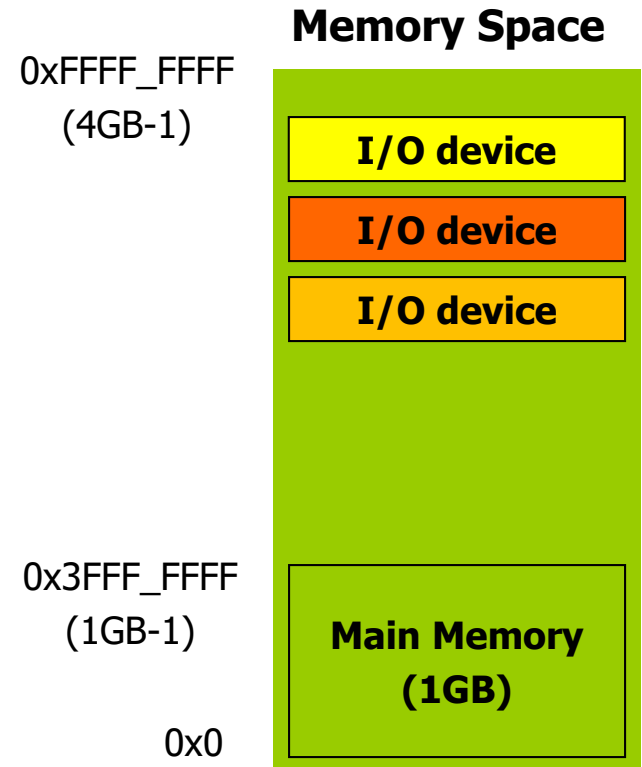


How Does CPU Access I/O Devices?

- All the I/O devices have registers implemented, so software programmers can use them to control the devices
 - Then, for programming, where and how to write to or read from?
 - There are 2 ways to access I/O devices
 - Memory-mapped I/O
 - I/O-mapped I/O

- **Memory-mapped I/O**

- I/O device is mapped to a memory space
- CPU generates a **memory transaction** to access I/O device
- To access I/O device
 - In MIPS, use `lw` or `sw` instructions
 - In x86, use `mov` instruction



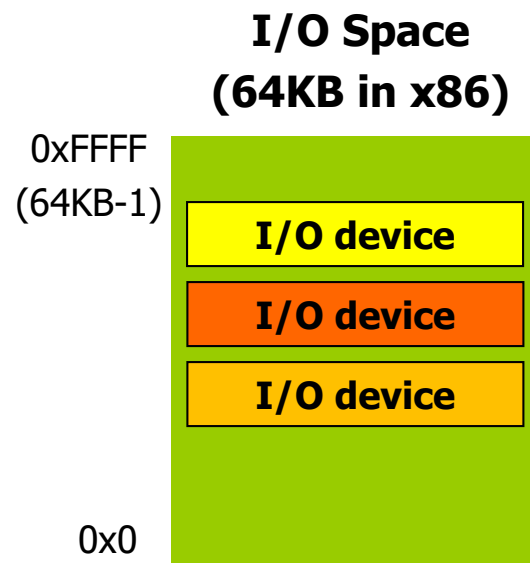
How CPU Accesses I/O Devices?

- **I/O-mapped I/O**

- I/O devices are mapped to I/O space
- CPU generates **I/O transaction** to access I/O device
- To access I/O device
 - In x86, there are `in` and `out` instructions.
 - In x86, I/O space is 64KB

- To differentiate memory space and I/O space, there should be hardware support

- ISA support
 - In x86, `mov` instruction for memory transaction and `in, out` instruction for I/O transaction
- Physical pin from processor indicating the transaction type (memory or I/O)
 - For example, the pin is driven to "1" for memory transaction or "0" for I/O transaction



How I/O Communicates with CPU?

- **Polling**

- CPU periodically checks the status of I/O devices to determine its need for service
 - CPU is totally in control
 - Can waste a lot of CPU time due to speed differences

- **Interrupt**

- I/O device issues an interrupt to indicate that it needs attention
- An I/O interrupt is **asynchronous** wrt (with respect to) instruction execution
 - It is not associated with any instruction, so doesn't prevent any instruction from completing
 - You can pick your own convenient point in the pipeline to handle the interrupt

DMA (Direct Memory Access)

- Typically, moving data from one place to another involve CPU instructions
 - Load (l_w) from a location (e.g. memory in an I/O device)
 - Store (s_w) to another location (e.g. main memory)
 - Moving a large chunk of data with CPU instructions could take a large fraction of CPU time
- DMA has the ability to transfer large blocks of data **directly** to/from the memory **without involving the processor**
 1. The processor initiates the DMA transfer by supplying source and destination addresses, the number of bytes to transfer
 2. The DMA controller manages the entire transfer (possibly thousand of bytes in length), arbitrating for the bus
 3. When the DMA transfer is complete, the DMA controller interrupts the processor to inform that the transfer is complete
- There may be multiple DMA devices in one system
 - Processor and DMA controllers contend for bus cycles and for memory