Microprocessor IO Subsystems

Chun-Jen Tsai
NYCU
12/09/2022

Interface btw. μP and Devices

- There are several methods to design communication interface between processors and devices of a system:
  - Dedicated I/O instructions – obselete
  - Memory-mapped I/O interface
  - Interrupt-driven interface – mailbox device
  - Co-processor interface – ARM does it, with FP Unit
- In any cases, the set of wires that connects HW modules together is called a bus

Dedicated I/O Instructions

- In early CPU ISA designs, device I/O and memory R/W are done using different instructions
- For example, in Intel x86 ISA:

device I/O

```
IN  BX, 19h
ADD BX, 1
OUT 19h, BX
```

device I/O

V.S.

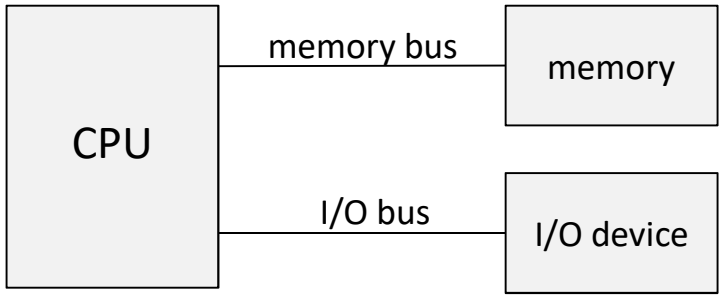
Memory R/W

```
MOV BX, [19h]
ADD BX, 1
MOV [19h], BX
```

memory I/O

load/store in RISC

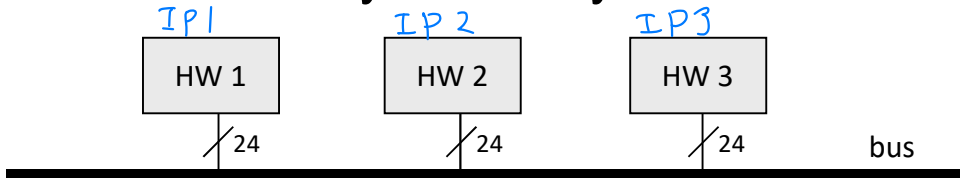
  - In previous example, I/O address 19h and memory address [19h] points to different space.



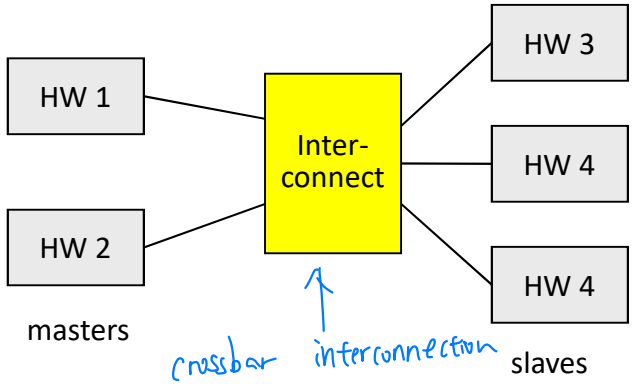
Bus Topology

Ethernet

- Shared bus – a set of wires is shared among modules
  - Shared bus saves both design and production costs
  - Performance of the system may suffer



- Point-to-point bus – dedicated links btw. two modules
  - Complex inter-connect module is required when there are more than two modules

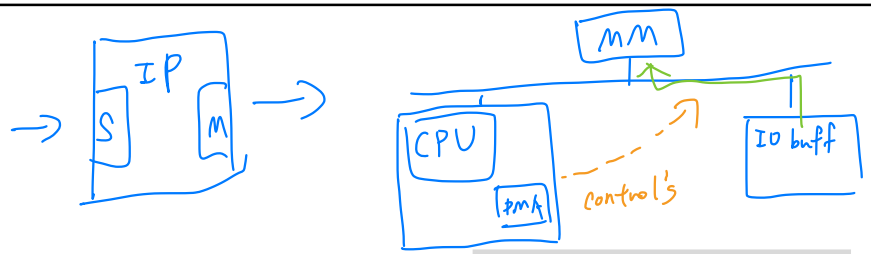


Network
N.o C Chip

HW1 – HW3
&
HW 2 – HW4
can talk simultaneously
(bandwidth ↑ !!

crossbar interconnection
more expensive.

Bus Components

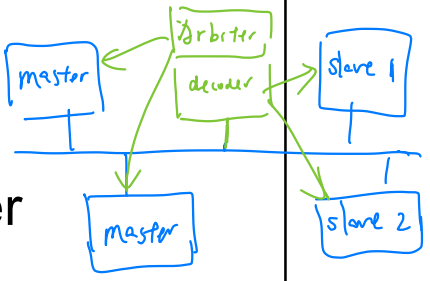


- Bus master:
  - A HW module that is able to initiate read and write requests
  - Typical masters: CPU and DMA

Direct Memory Access (D/S large chunk of memory)

- Bus slave:
  - Responds to read/write requests from the bus masters
  - A slave signals back to the master the results of data transfer
  - Typical slaves: memory controller and I/O device

bus master ↑ bus slave ↑



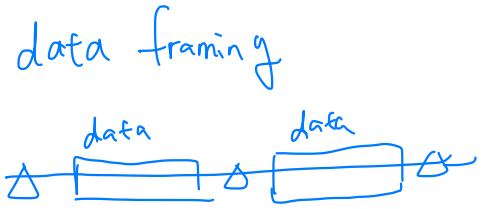
- Bus controller:
  - A bus controller is composed of an arbiter and a decoder
  - Arbiter: determines which master gets to use the bus
  - Decoder: determines which slave should reply to a request

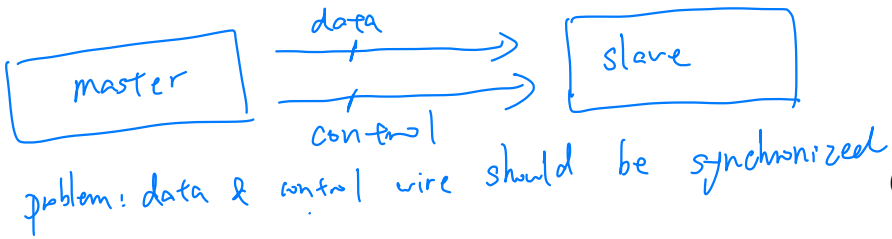
Although without a decoder, the architecture can still work, the slave must listen to the bus, waiting for a request. With a decoder, the slave IP can be turned on only when needed, saving power.

Bus Protocols

- Bus protocol controls communication among modules:
  - Determines who can use the bus at any particular time
  - Governs length, style of communication

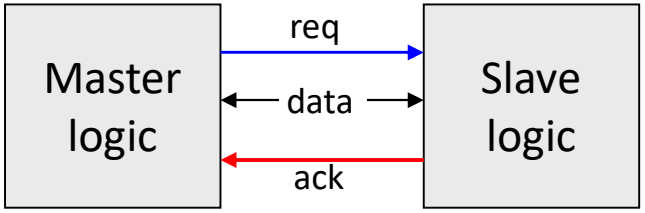
- Data framing vs. control wires
  - A bus architecture with few wires usually uses “data framing” (at data-link layer) to signal transactions
    - Control and data signals shares the same wires
    - Sometimes referred to as “in-band” signaling
  - A bus architecture with many (parallel) wires uses “control wires” (at physical layer) to signal transactions
    - Control and data signals use different wires

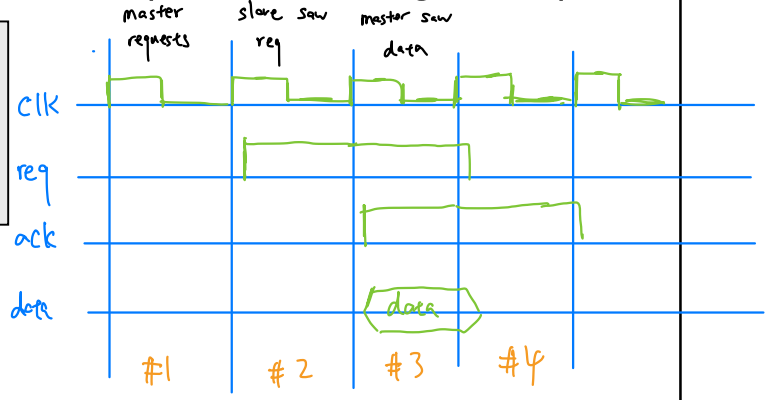




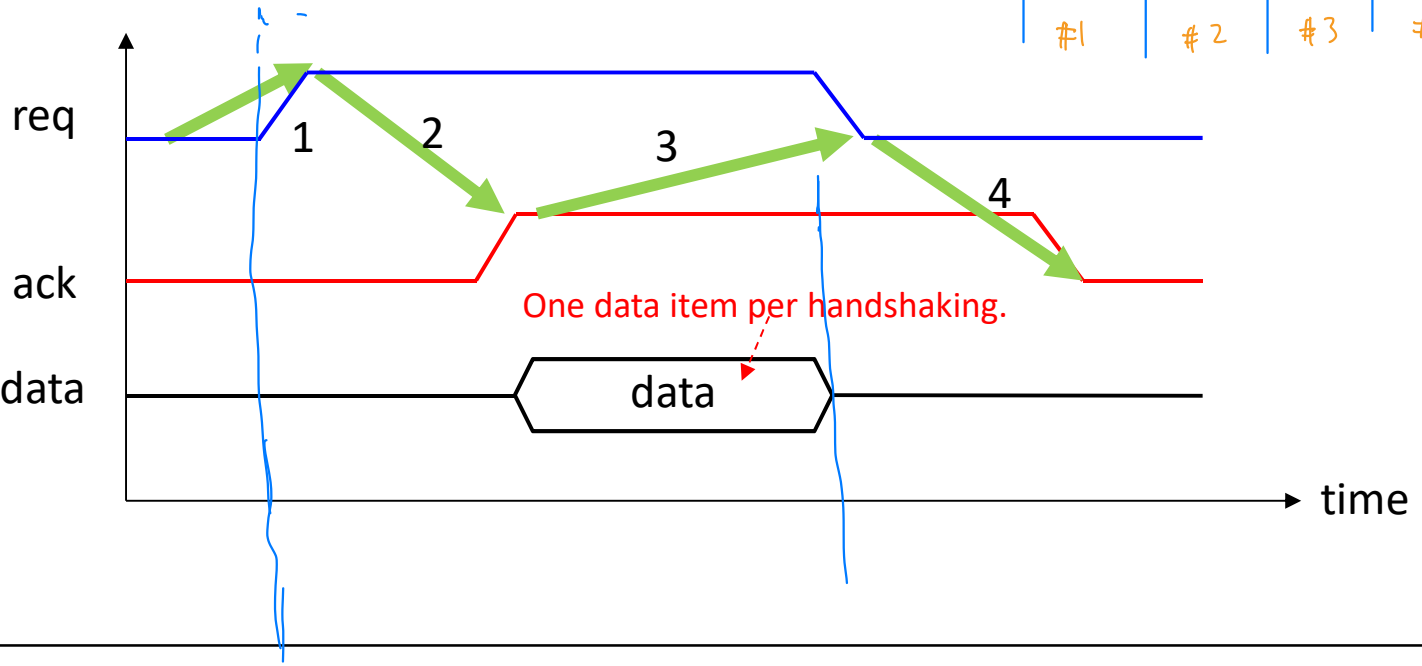
Four-Cycle Bus Protocols

- Four-cycle handshake is the basis of many protocols
  - Use two control wires: req (request) and ack (acknowledgment)

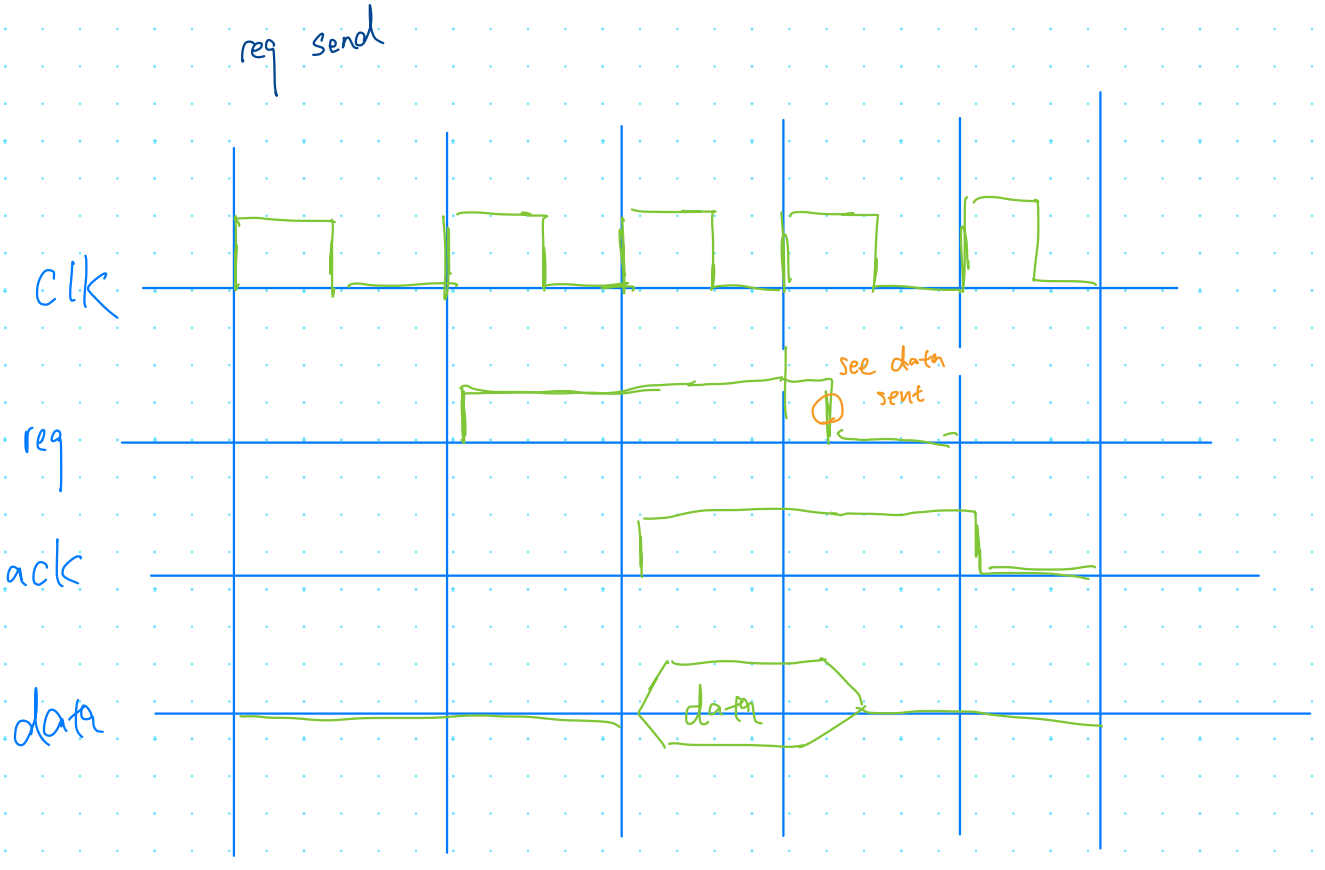




  - Example of transaction:



may be multiple → data lines eg 16 — data


req send
clk
req
see data sent
ack
data
data

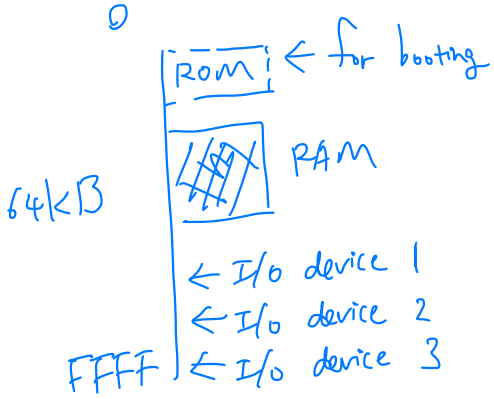
Formal BUS Behavior

- Formal properties of a BUS protocol:
  - Following a reset, everything should be set to idle
  - If a request is stalled (hit wait state), it should be maintained
  - There shall be no responses without prior requests
  - All requests get responses
  - Once a transaction is done, everything should return to idle

A request must maintain because the IP of bus bridges may latch the request.

Interface Using Memory-Mapped I/O

- Processor interface can be more flexible with the memory mapped I/O (MMIO) model
- In MMIO model, some “memory” addresses are mapped to the internal control registers of the device
  - Processor write data into these addresses to control the device
  - Processor read data from these addresses to retrieve results from the device



Typical Bus Signals for MMIO

FF ID for slave

- ADDR: multi-bit ID for slave devices
- RW: signals a read or write request

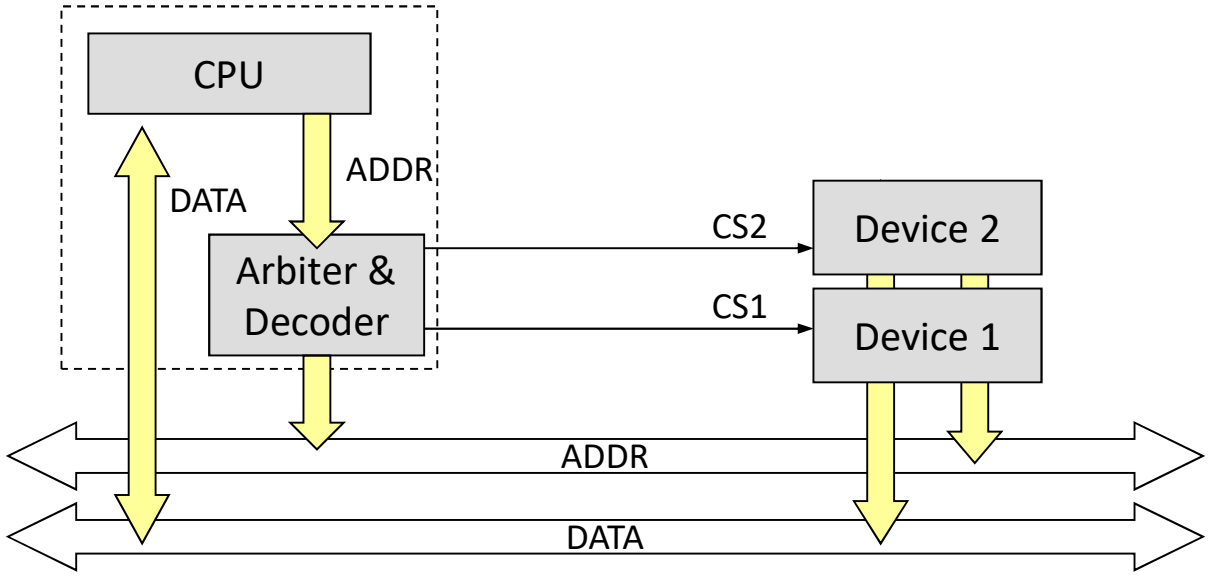
Request from master

- DATA: multi-bit data signal (to or from slave)
- READY: from slave (ack)

Ack from slave

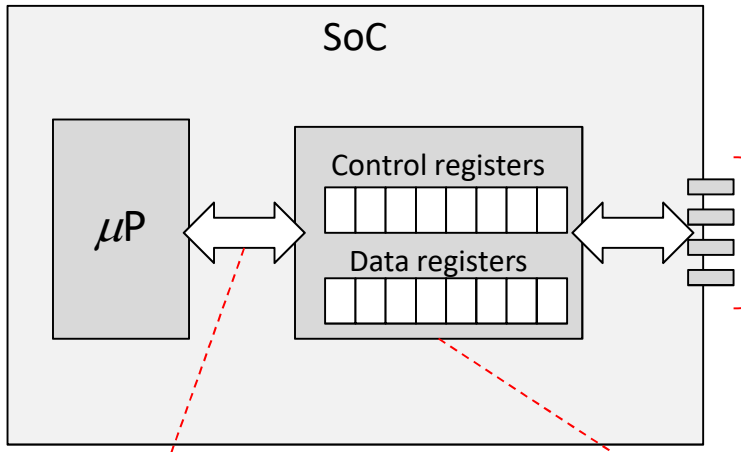
- CS: device selection signal (for power saving)

chip-select.
(en, enable)



MMIO Wrapper for Devices

- Since CPU uses load/store instructions to talk to various devices, we need a wrapper circuit to help converting read/write operations to bus signals
  - These wrappers are typically inside I/O controllers



GPIO : registers ⟷ waveform

Connects to external devices of some bus protocols, such as I^2C or SPI

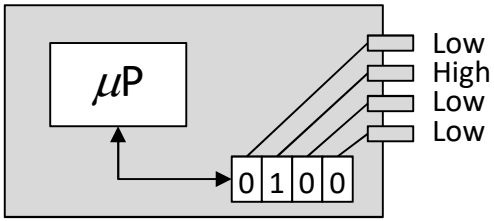
On-chip memory-mapping bus protocols, such as AHB or AXI .

an I/O controller, a wrapper of external device bus protocols.

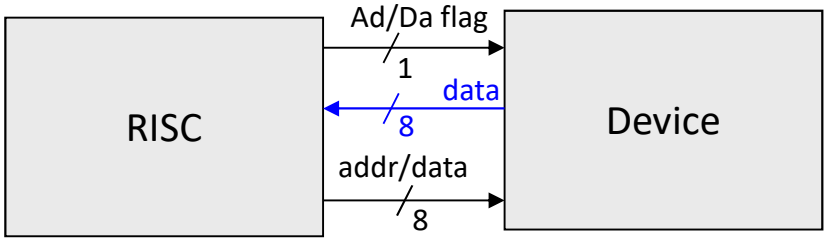
μP Interface Using GPIO Pins

- Most microcontrollers have some GPIO pins that can be used for interfacing with the device logic
  - GPIO is a simple HW device that maps a register bit value to a input/output wire of the CPU pin

GPIO could talk to low speed devices
eg. SD-card, 1602



- The number of GPIO pins of a CPU is usually small
  - Multiplexing technique can be used to issue a request



Talking to ASIC from C via GPIOs

- Typical GPIO device contains a n-bit register
  - Each register is connected to a I/O pin
  - Physical pins are implemented as “inout” ports

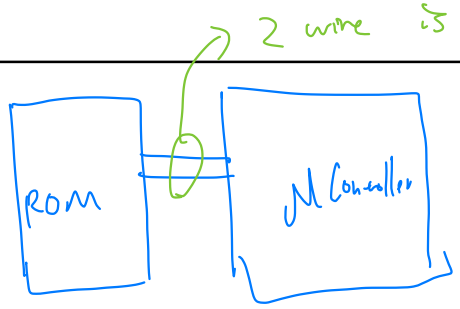
- C code example:

```
unsigned int *PIO = 0x03FF5008; // GPIO register address

/* write */
*PIO = 0xB0;

/* read */
Data = (*PIO >> 8) & 0xFF;
```

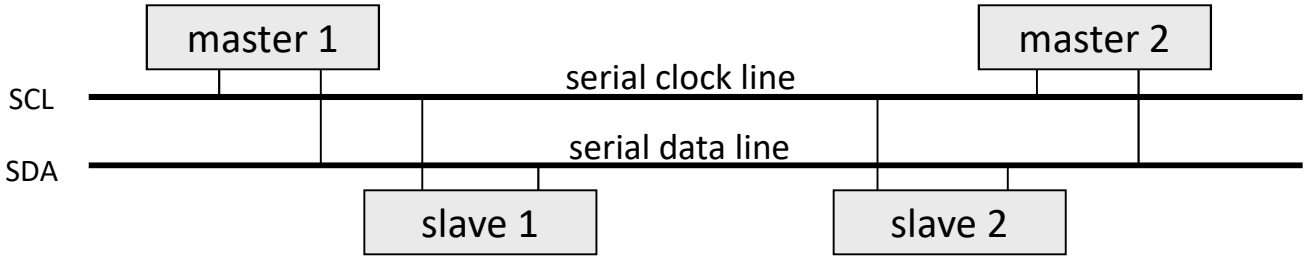
Inter-Integrated Circuit

I²C Bus

Originally designed for TV controllers



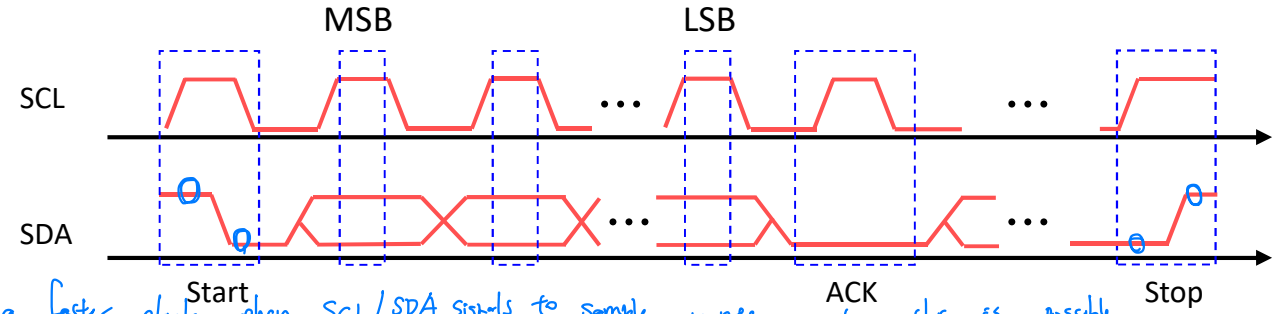
- Designed by Phillips in 1982
  - Two-wire, synchronous serial link from 100kHz ~ 5MHz
  - Shared bus with multi-master support using fixed-arbitration
  - Most SoC's come with built-in I²C controllers
    - Boot code stored in an I²C flash, and read into RAM at power-on
- I²C signaling
  - SDA carries 1 by default, sender pulls it down to send 0
  - If sender sends a 1 and gets a 0, others are sending data



Any master who detects a collision shall not keep sending

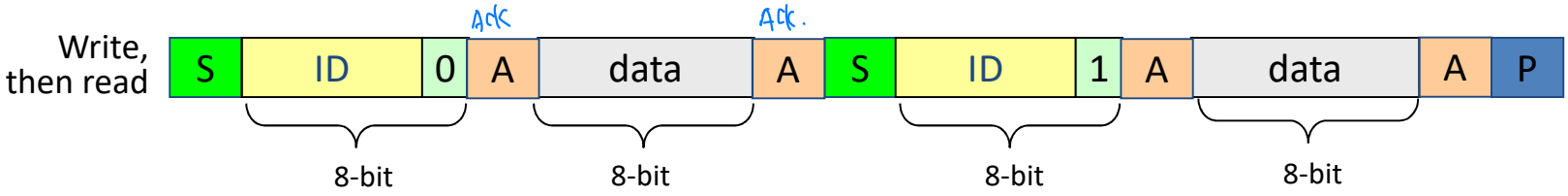
I²C Signaling

- I²C transmissions occur in 8-bit bytes
  - Master raises a START pattern
  - Send 8 bits (slave ID & R/W, or data)
  - Target slave responds with an ACK pattern
  - Master sends a STOP pattern when the transaction is done



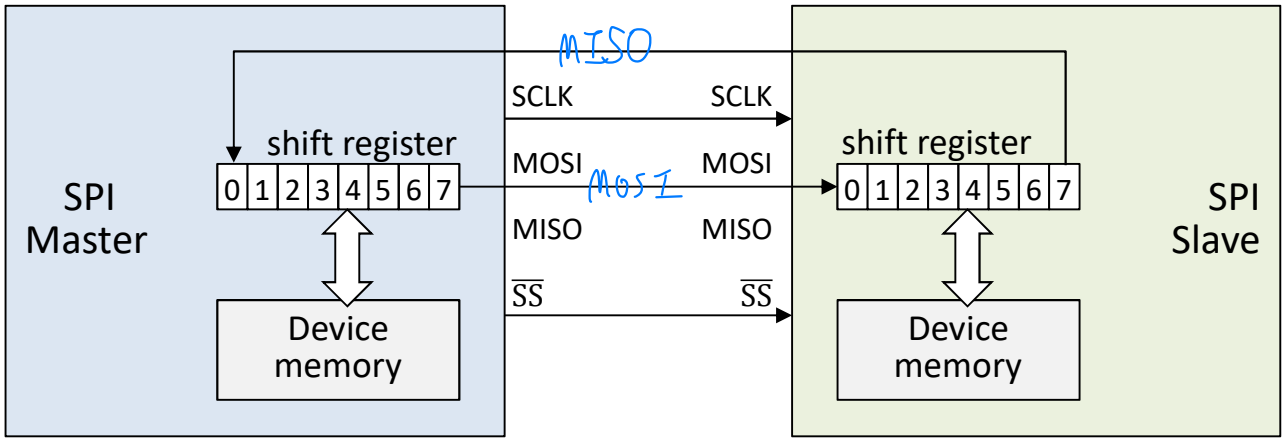
We need a faster clock than SCL/SDA signals to sample points on them, this is possible because I²C is a low speed protocol.

- An example of transmission:



Serial Peripheral Interconnect (SPI)

- Designed by Motorola in 1980's
  - Short-distance, full-duplex, synchronous serial transmission
  - One master, multiple slaves
  - Often used for SD cards, flash ROMs, etc.
  - Master selects the target slave via $\overline{SS}$
  - Data sizes are 8, 12, or 16 bits per transaction

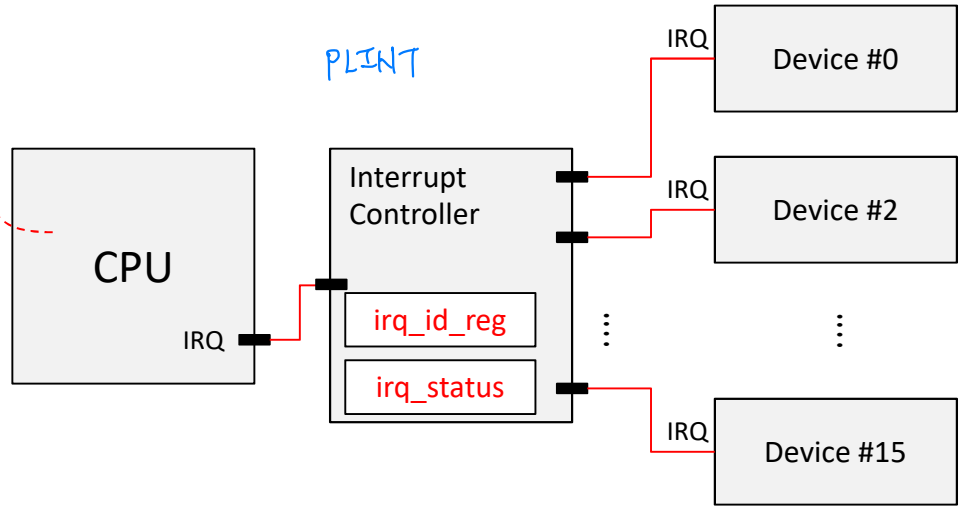


Interrupt Signaling from IP to CPU

- Sometimes, a HW module sends an interrupt to CPU to signal a request or an acknowledgment (ack)
- Typical system diagram with interrupt-capable devices

```
my_isr()
{
  push_registers();

  switch(irq_id_reg)
  {
  case 0: // ISR for device #0
        ...
        break;
     . . .

  case 15: // ISR for device #15
        ...
        break;
  }

  pop_registers();
  enable_irq(irq_status);
}
```

Interrupt Service Routine (ISR)

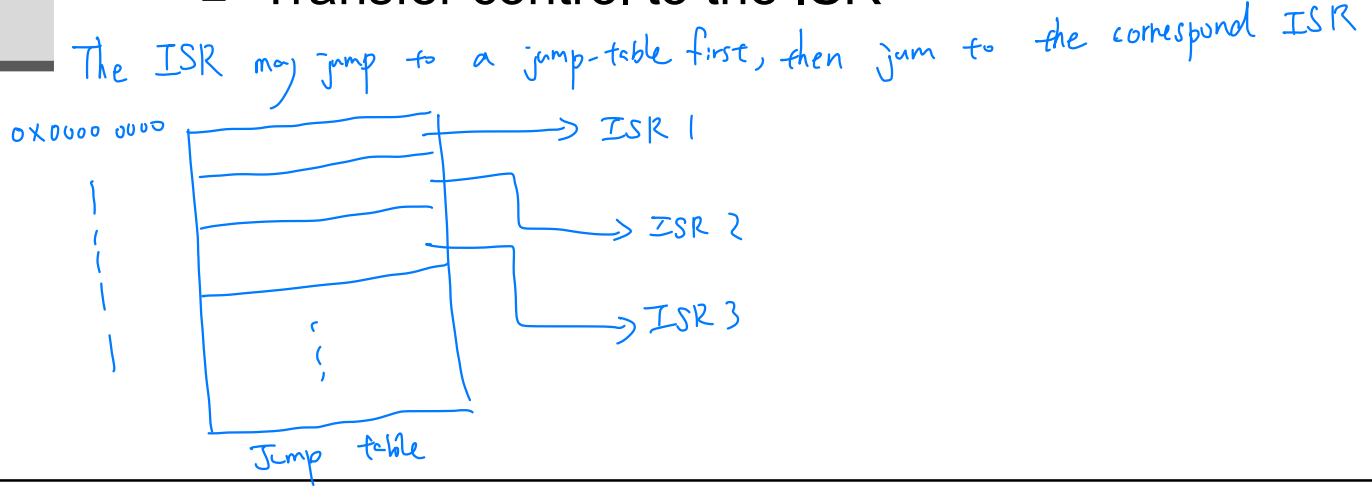


Example: RISC-V Interrupt Interface

- A RISC-V core has 3 execution modes: M, U, and S
  - Only M mode, the machine mode, is mandatory.
  - Current execution mode is stored in a private register
- Interrupt info is in the Control & Status Registers (CSR)
  - Each mode has its own set of registers
- The execution mode can only be changed by a trap, an interrupt, or `mret`/`sret`/`uret` return instructions
  - The core stores the interrupted mode, and resume its value upon return

RISC-V Interrupt Flow

- For example, for M-mode interrupt:
  - Store the interrupted PC in a CSR register `mepc`
  - Set the interrupt cause in a CSR register `mcause`
  - Disable any interrupts by flagging the `mie` bit in `mstatus`
  - Look up the interrupt handler address in `mtvec`:
    - `mtvec` is composed of `{BASE[31:2], MODE[1:0] }`
    - In direct-mode, `BASE` is the ISR address
    - In vector-mode, the `BASE+4×mcause` is the ISR address
  - Transfer control to the ISR

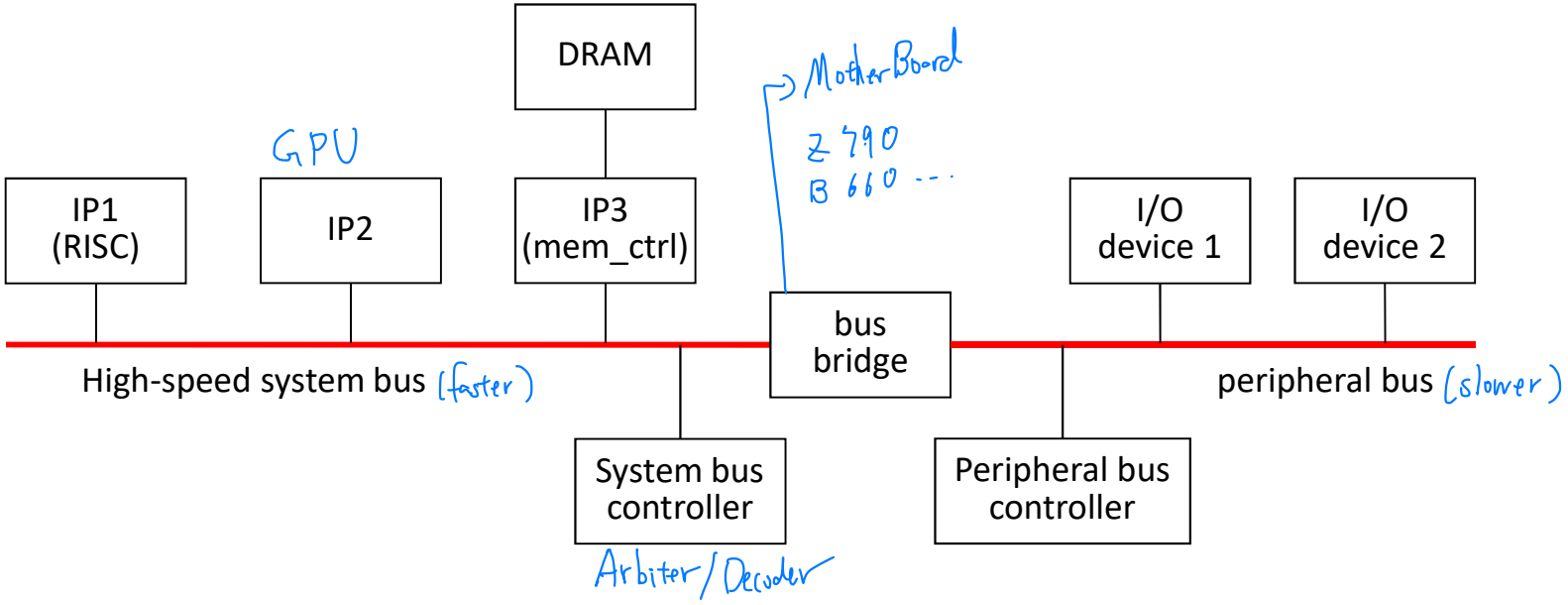


On-Chip Bus Architecture

- On-chip bus is one of the crucial components for SoC
  - Facilitates platform-based design paradigm
  - Enables reusable IP design platform based SoC Design.
- Popular SoC bus architecture:
  - AMBA – ARM's bus architecture
    - AHB/APB: shared bus topology
    - AXI: point-to-point bus topology
  - CoreConnect – IBM's bus architecture (for PowerPC)
  - Wishbone – the OpenCores bus for open source IP design

Typical SoC Bus Architecture

- A traditional bus architecture (e.g., ARM's AHB and APB) are designed for the following system model:



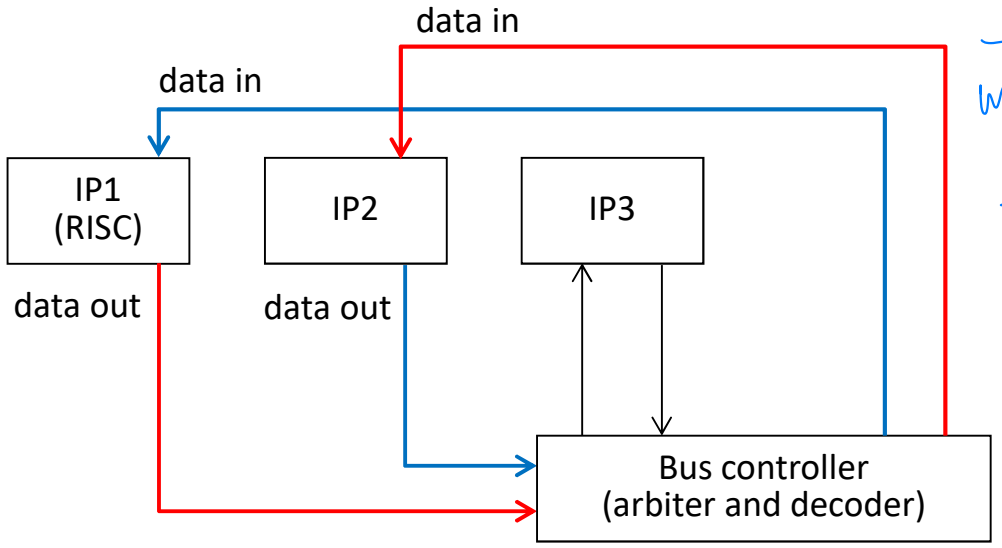
- Today, the buses often become the system bottleneck!

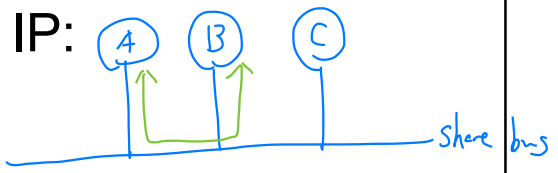
NoC : Put a mini Network on Chip to interconnect IPs.

Use wireless communication inside a chip.

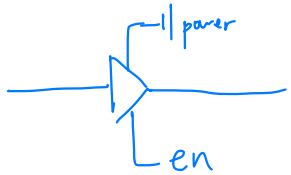
Physical Separation of I/O Busses

- For high-speed buses, the input and output lines are usually separated into two unidirectional buses
  - The system bus controller (contains the bus arbiter and bus decoder logics) will establish a unidirectional high-speed link from the data source IP to the data destination IP:

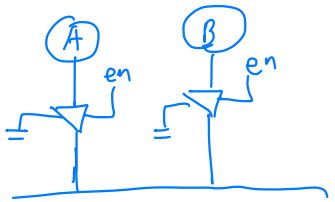




We need a tri-state buffer



if "en" is turned off, it's like disconnect wire



drawback: low clock rate

ARM Bus Evolution

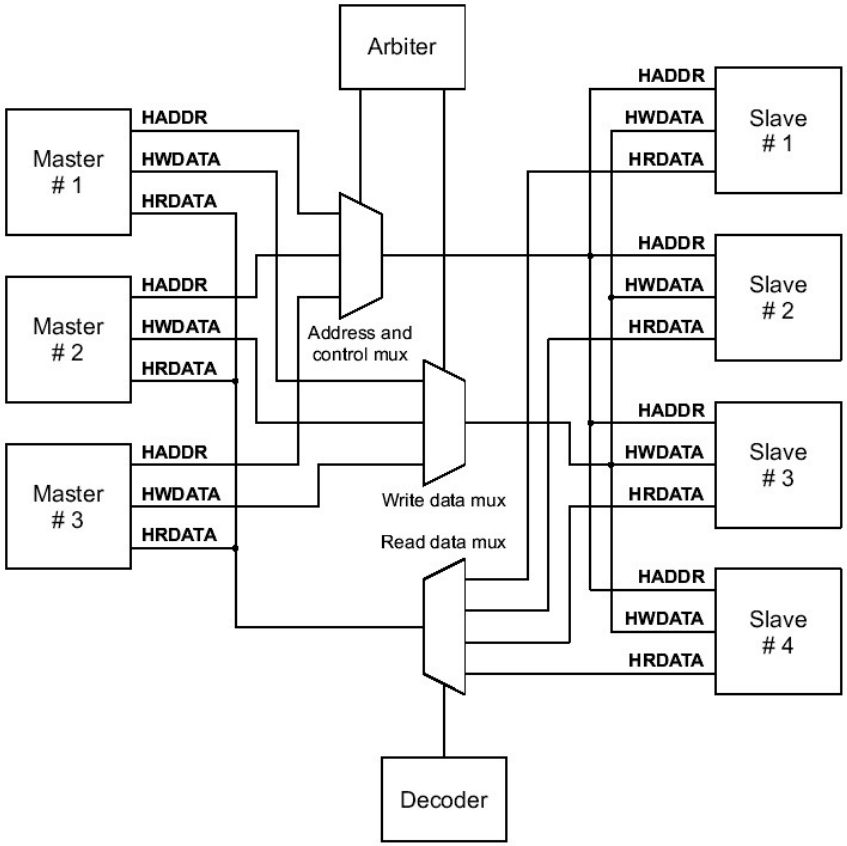
- The Advanced Microcontroller Bus Architecture (AMBA) is defined by ARM, Ltd

- Versions of AMBA:
  - AMBA 1.0 – ASB and APB, 1996
  - AMBA 2.0 – AHB and APB, 1999
  - AMBA 3.0 – AXI 1.0 and ATB (for CoreSight), 2003
  - AMBA 4.0 – AXI4 (aka, AXI 2.0) and ACE, 2010-2011
  - AMBA 5.0 – CHI (Coherent Hub Interface), 2013

- ARM buses are free of licenses

AHB Multiplexer Interconnection

- AHB is a multiplexed bus with a centralized arbiter

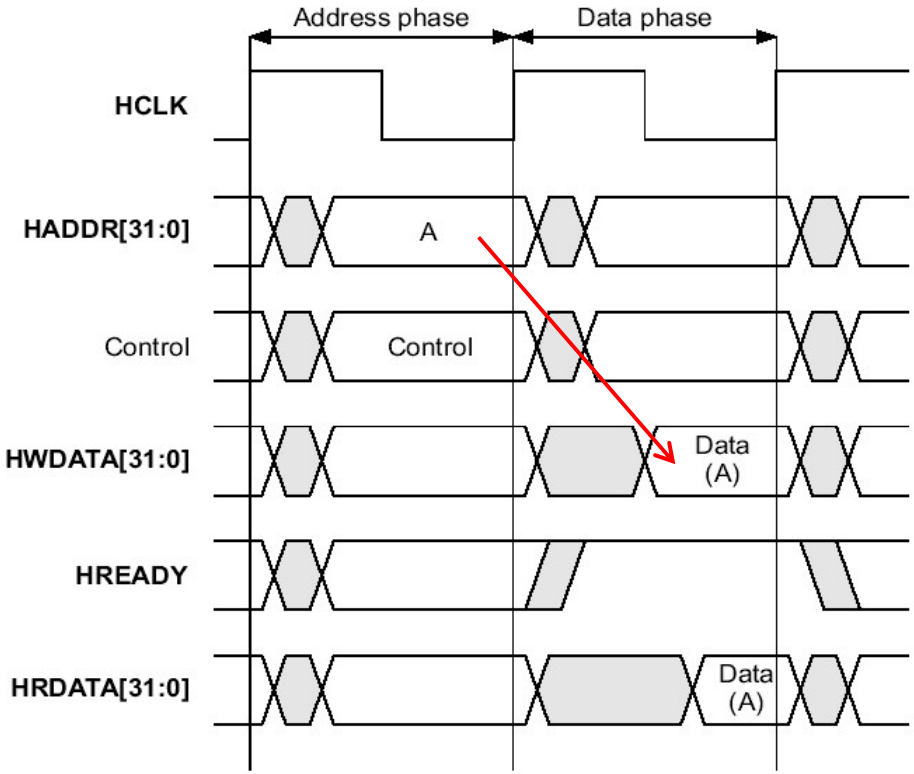


† Figure extracted from: ARM Limited, *AMBA Specification Rev 2.0*, ARM IHI 0011A, May 13, 1999, figure 3-2, page 38.

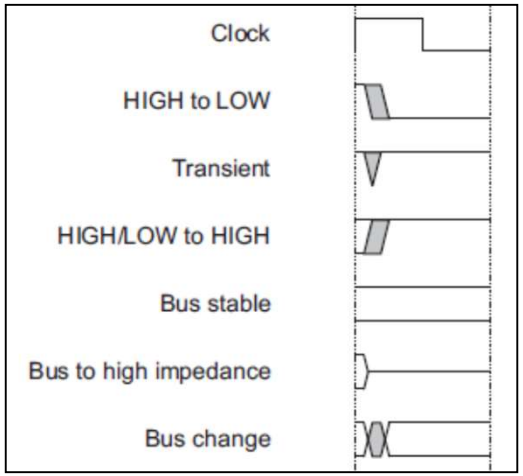
AHB Simple Transfer

(一筆資料)
single-beat transfer

- Each transfer consists of an address/control cycle and one or more data cycles:

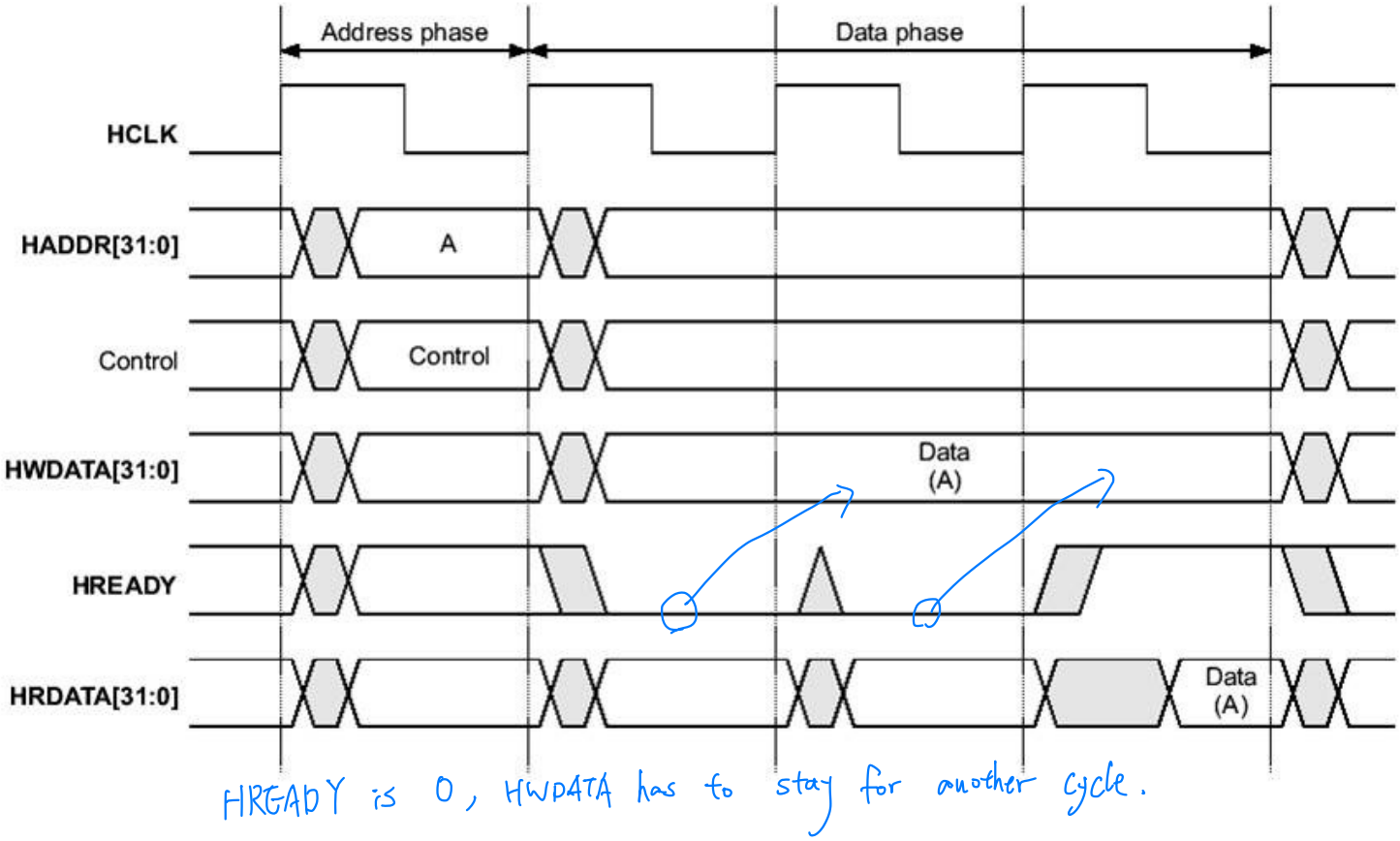


Bus diagram convention:



† Figure extracted from: ARM Limited, *AMBA Specification Rev 2.0*, ARM IHI 0011A, May 13, 1999, figure 3-3, page 40.

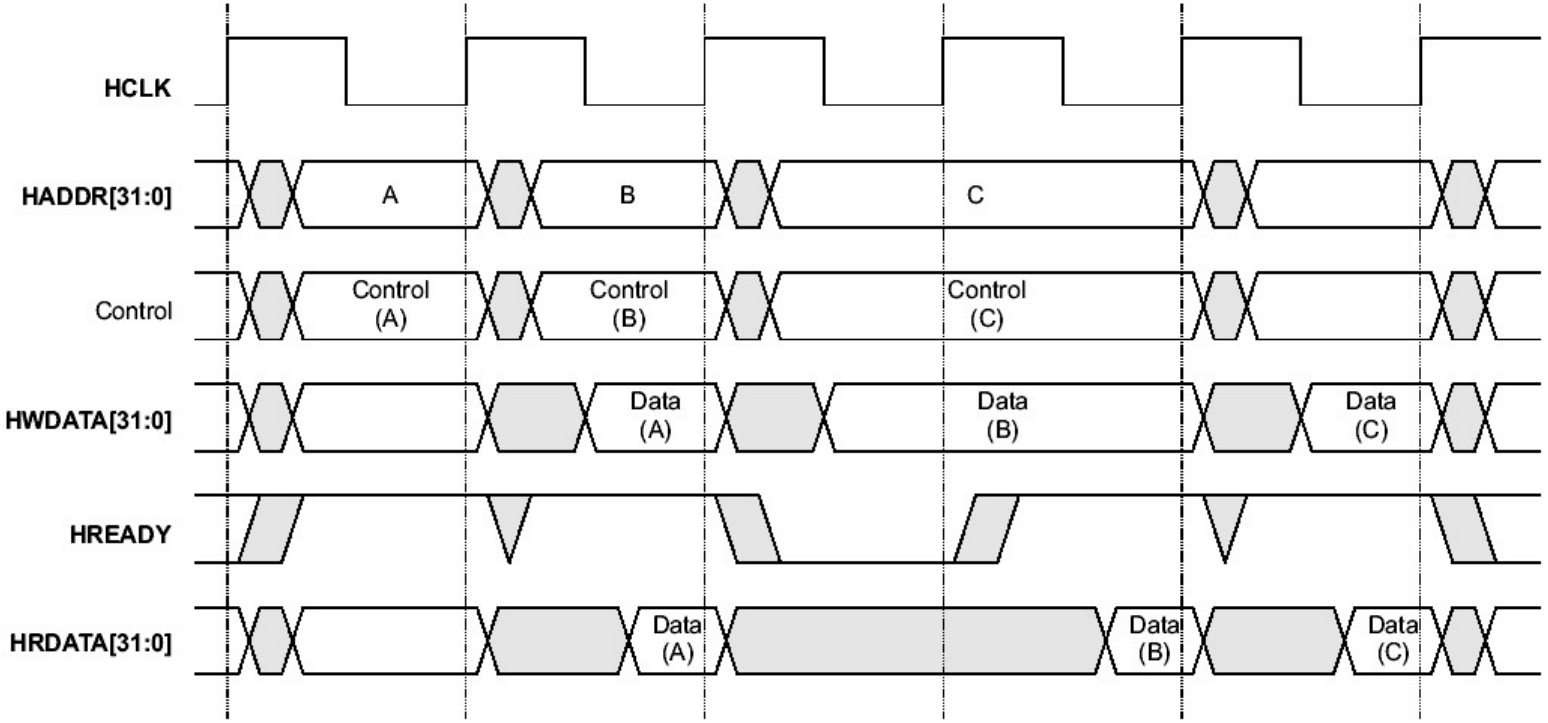
AHB Transfer with Wait State



† Figure extracted from: ARM Limited, *AMBA Specification Rev 2.0*, ARM IHI 0011A, May 13, 1999, figure 3-4, page 41.

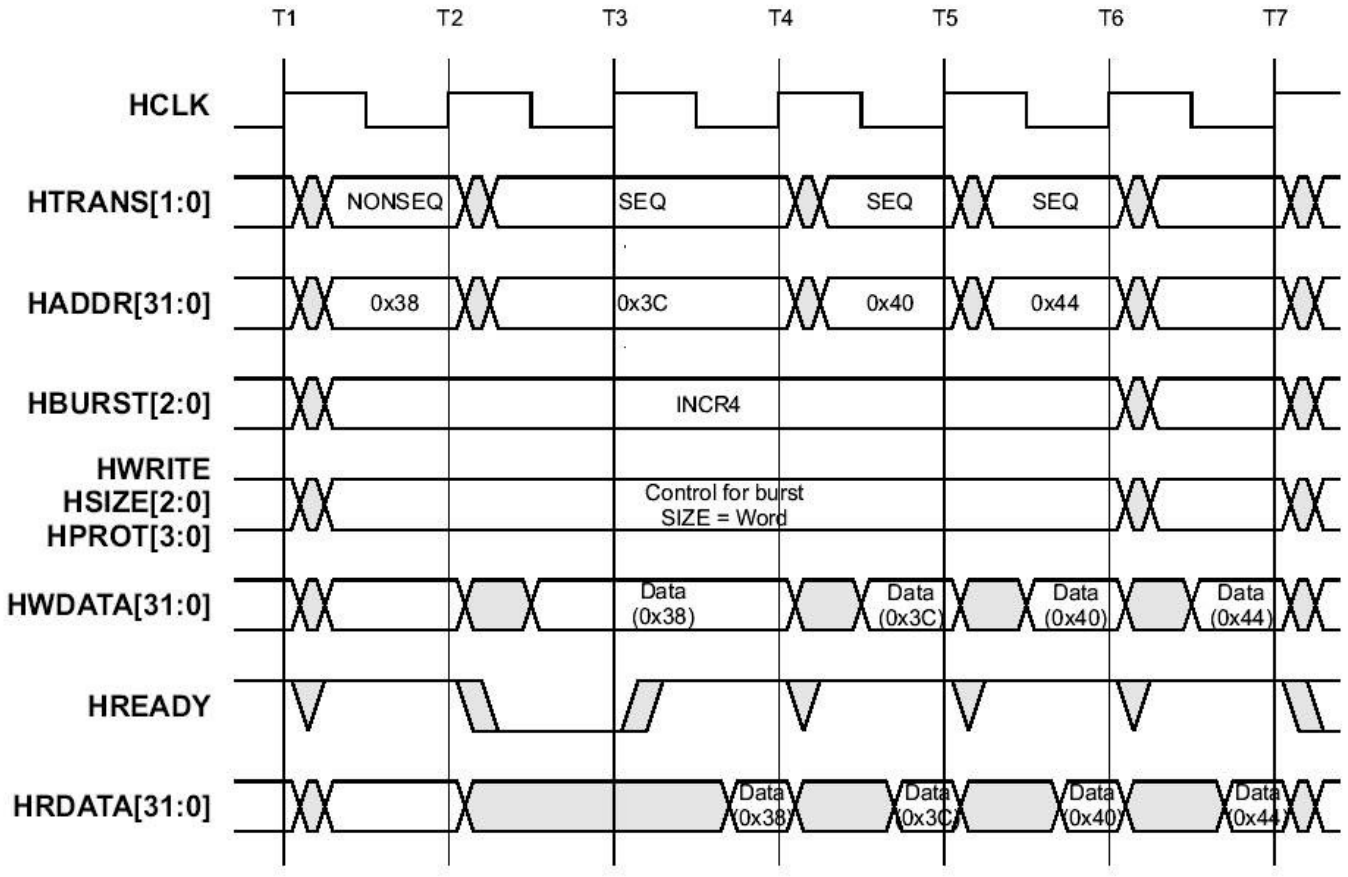
AHB Multiple Transfer

2-stage pipelining bus



† Figure extracted from: ARM Limited, *AMBA Specification Rev 2.0*, ARM IHI 0011A, May 13, 1999, figure 3-5, page 42.

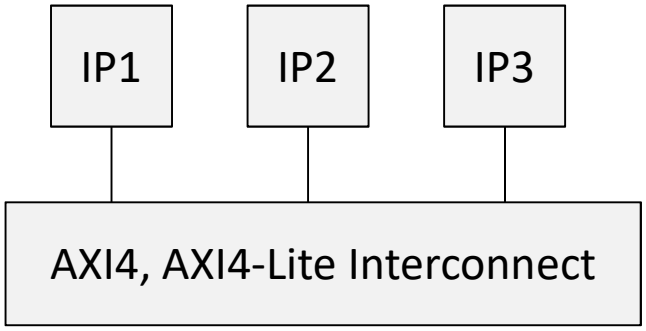
Burst Transfer Example

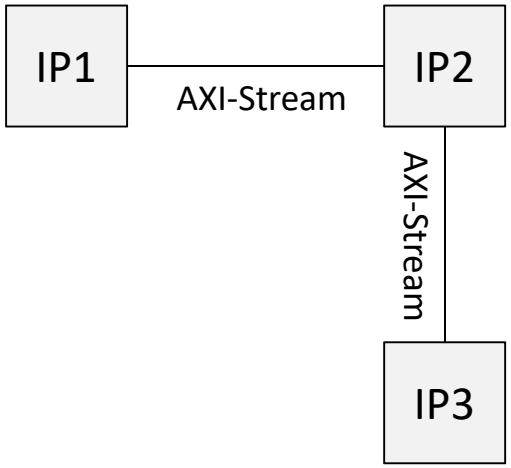


† Figure extracted from: ARM Limited, *AMBA Specification Rev 2.0*, ARM IHI 0011A, May 13, 1999, figure 3-8, page 48.

AXI4 Bus

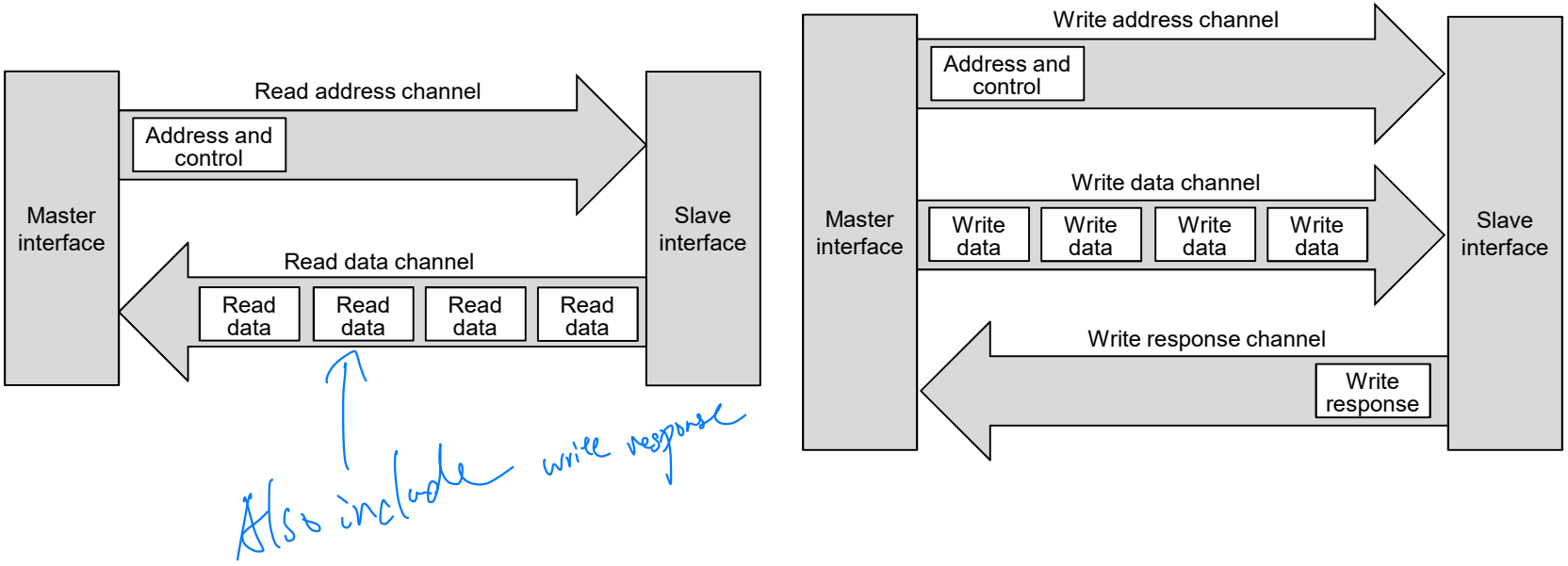
- There are three types of AXI4 interfaces
  - AXI4 – for high-performance memory-mapped requirements
  - AXI4-Lite – for simple, low-throughput memory-mapped communication (e.g. for control registers)
  - AXI4-Stream – for high-speed streaming data





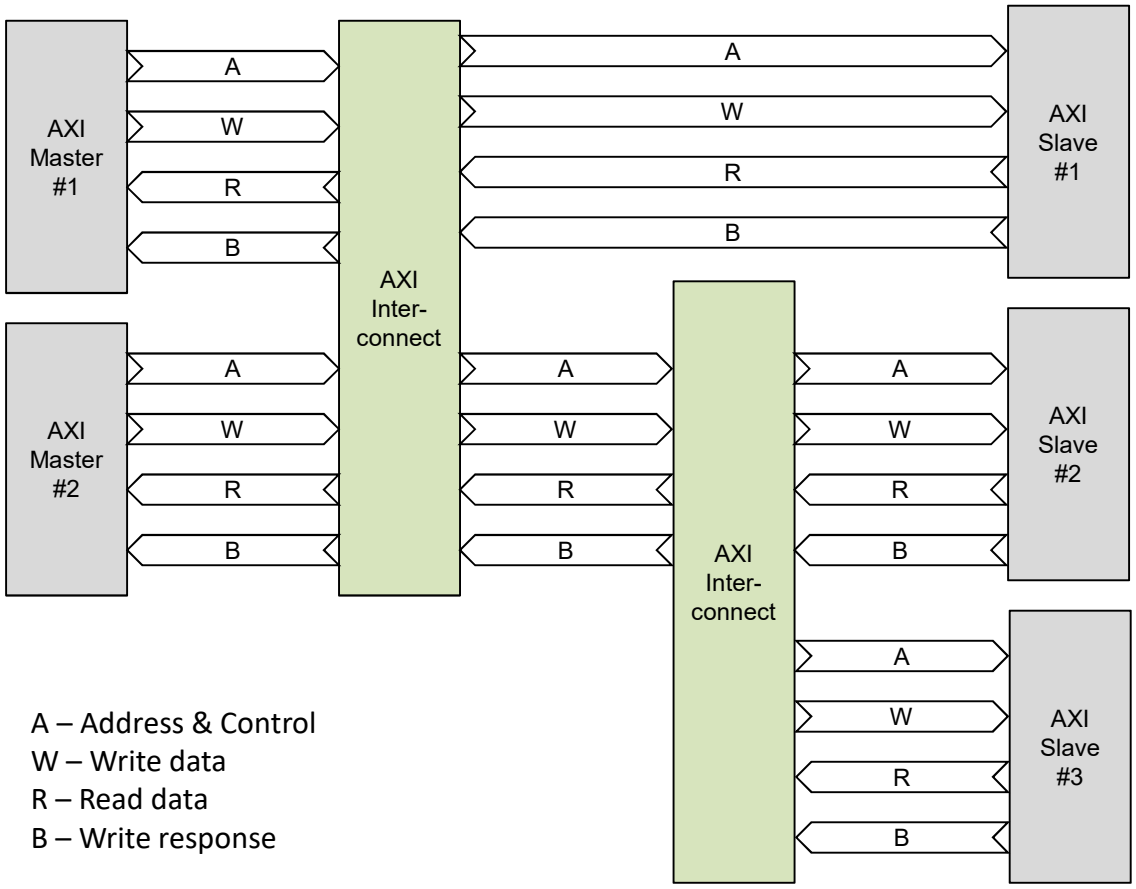
AXI4 and AXI4-Lite Interfaces

- Both AXI4 and AXI4-Lite interfaces consist of five uni-directional channels, read and write channels can operate simultaneously
  - Read Address Channel, Write Address Channel
  - Read Data Channel, Write Data Channel
  - Write Response Channel



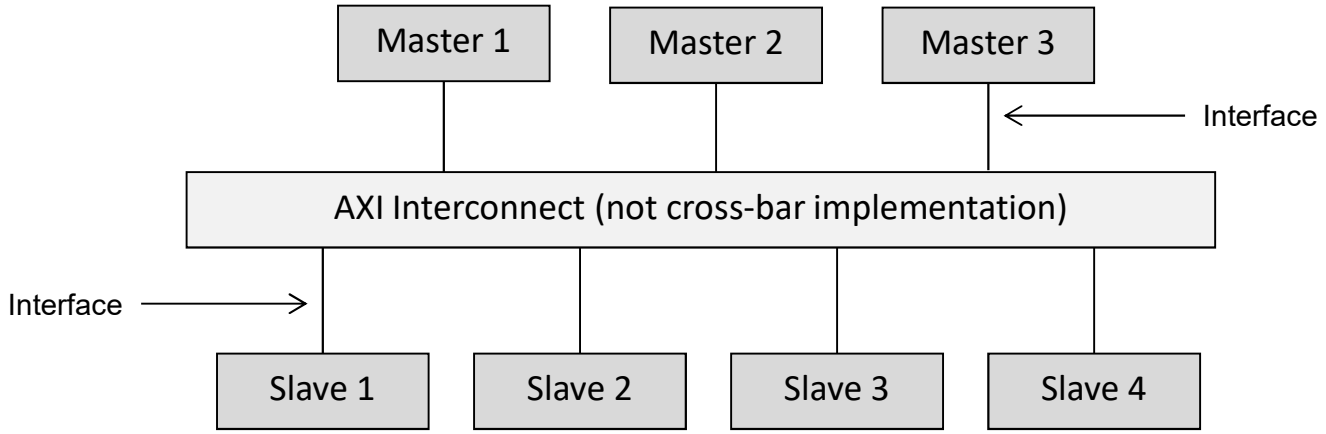
AXI Bus Topology Example (1/2)

- AXI devices do not share a common bus!



AXI Bus Topology Example (2/2)

- However, AXI can be used to construct a shared bus as well
  - The Wishbone bus protocol has similar concept
  - The interconnect implementation could be simplified to provide only one single transaction at any given time

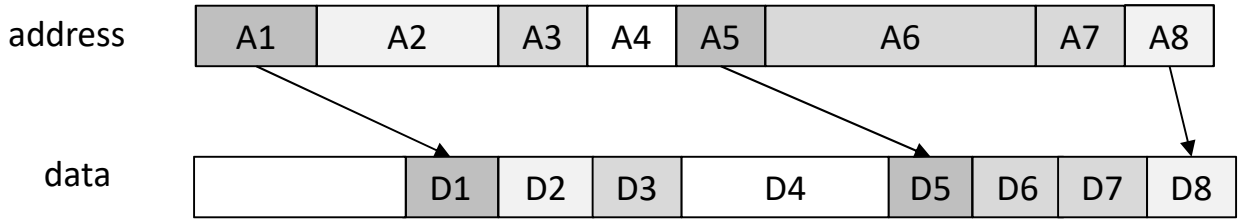


AXI4 Advanced Features

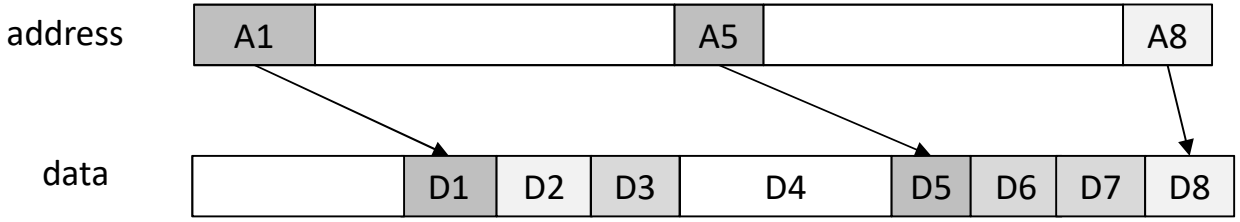
- AXI4 allows a burst transaction of up to 256 data transfers; a burst transfer only needs to transfer a single starting address
- AXI4-Lite allows only one data transfer per transaction
- Features of AXI4
  - Data upsizing and downsizing
  - Multiple outstanding addresses
  - Out-of-order transaction processing
  - Local synchronization clock per master-slave pair
  - Insertion of register slices for improved timing closure

AHB vs. AXI Burst

- AHB Burst
  - Addresses and data are locked together
  - HREADY controls intervals of address and data

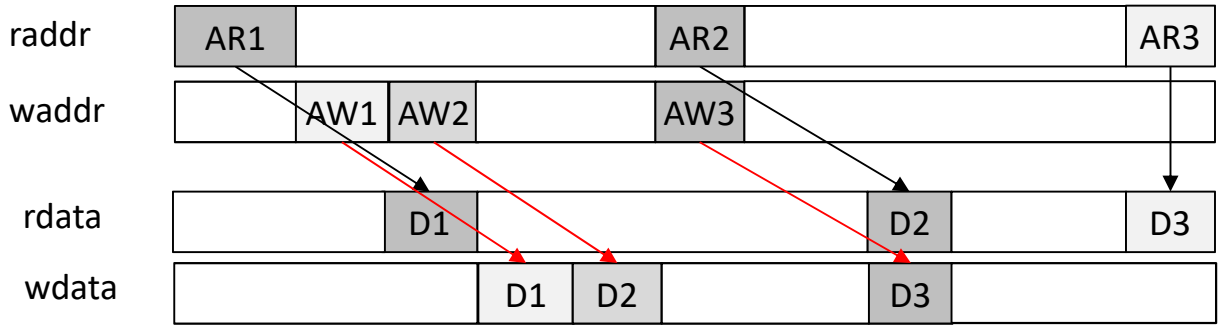


- AXI Burst
  - One address per burst



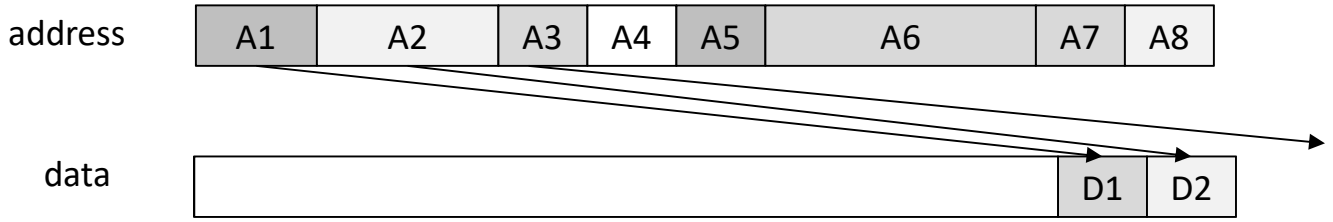
Interleaving Read/Write

- AXI burst allows simultaneous read/write transactions:

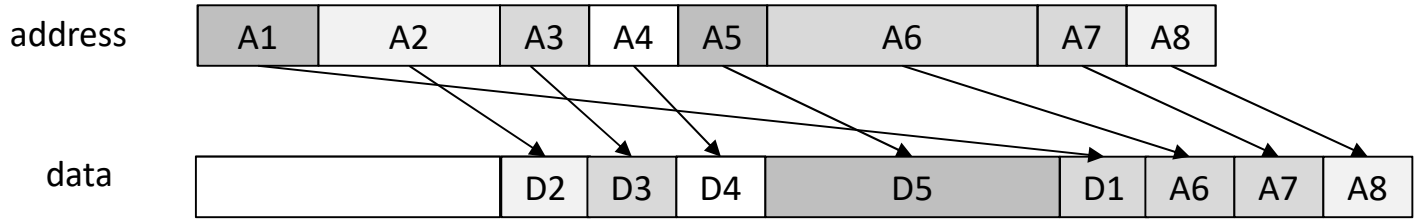


Out-of-Order Transmissions

- With AHB
  - If one slave is very slow, all following transactions have to wait



- With AXI
  - Fast slaves may return data ahead of slow slaves



Transaction ID's

- Out-of-order and interleaving transactions are possible because the AXI bus ties an ID signal to each address, data, and response signals.
- However, neither slaves and masters are required to honor the ID signals
  - Unless both the master and slave know about out-of-order transaction, you should not use this feature
  - A slave is mandatory to mirror the transaction ID from the master in BID (response ID) and RID (read data ID) of the return channels of the transaction

AXI4-Stream Features

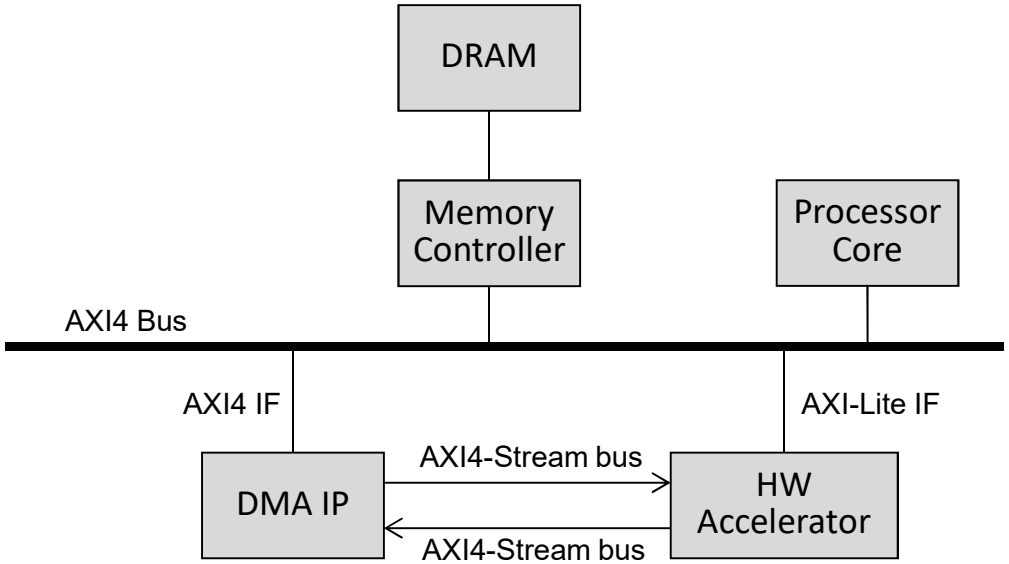
- The AXI4-Stream bus is a single channel bus for transmission of unidirectional streaming data
  - Similar to the write data channel of the AXI4 bus, but there is no limit on the amount of data transfer per transaction
  - Used for point-to-point data transfer without a unified context of “address” between IP’s
  - More efficient than AXI4, but less flexible (e.g., random accesses are not allowed)

AXI Infrastructure IP's

- An infrastructure IP is used as a glue logic to construct a system, for example:
  - AXI interconnect IP
  - AXI FIFO's (for buffering/clock conversion)
  - AXI Direct Memory Access (DMA) engines
  - Register slices (for pipelining)
- These IPs are useful as "middlemen" in moving data around in a system, but not used as endpoints for data

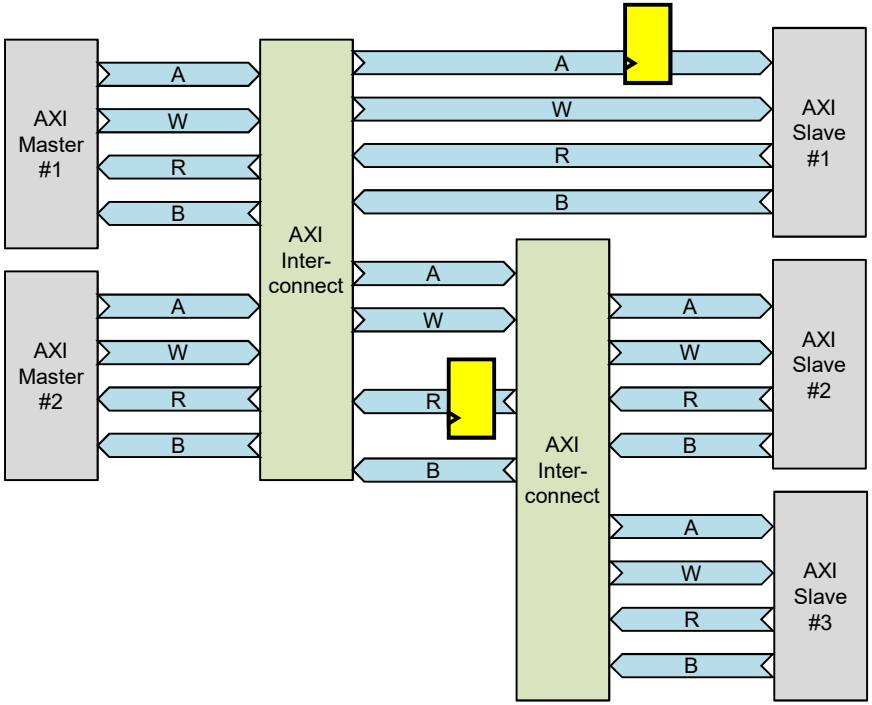
Example: AXI4 DMA Engine IP

- A DMA Engine IP is used to connect a AXI4-Stream interface (IF) to a AXI4 memory-mapped interface, and vise versa.



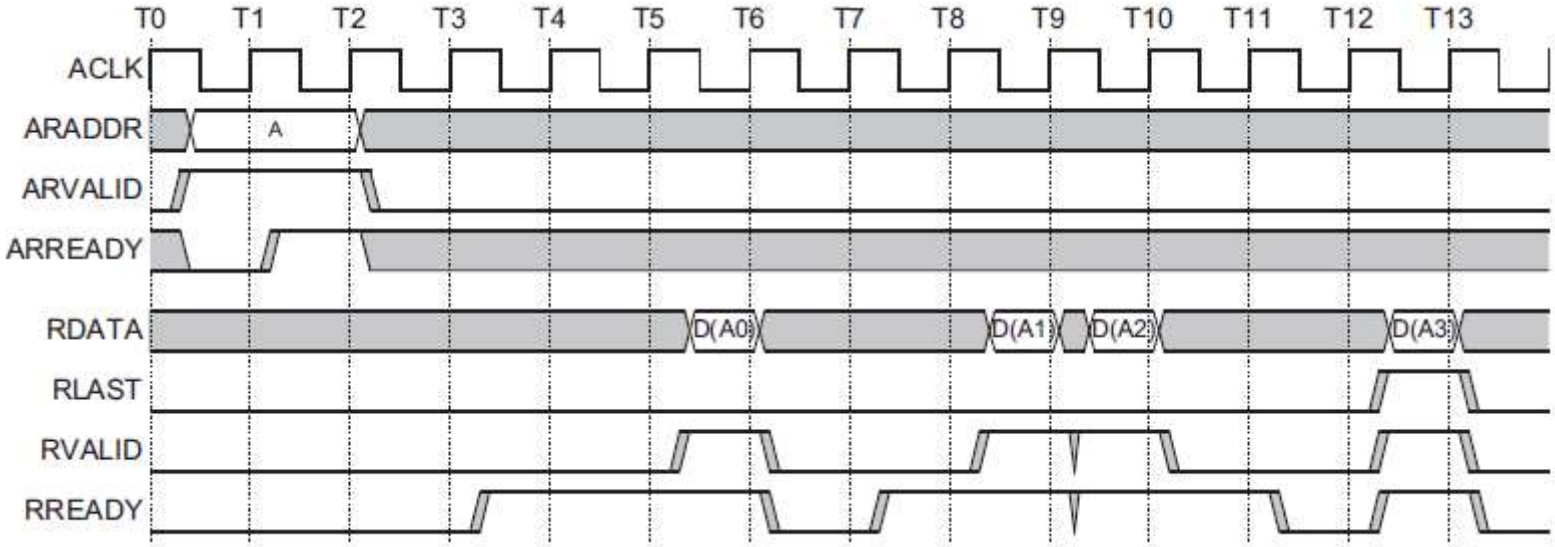
Example: Register Slices

- Each AXI channel transfers information in only one direction, and the architecture does not require any fixed synchronization
  - A register slice can be inserted at any point in any channel to increase working frequency, at the cost of additional latency



AXI Read Burst Example

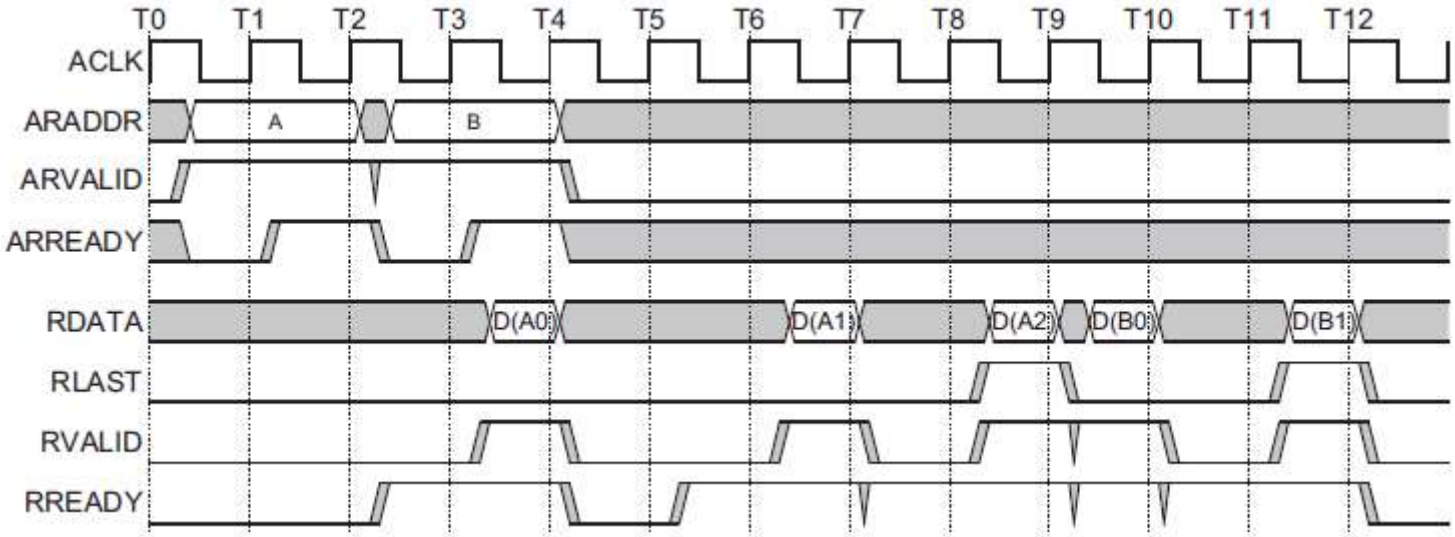
- A master performs a read burst of four data transfers:



† Figure extracted from: ARM Limited, *AMBA AXI Protocol v 1.0*, ARM IHI 0022B, March 19, 2004, figure 1-4, page 25.

Overlapping Burst Example

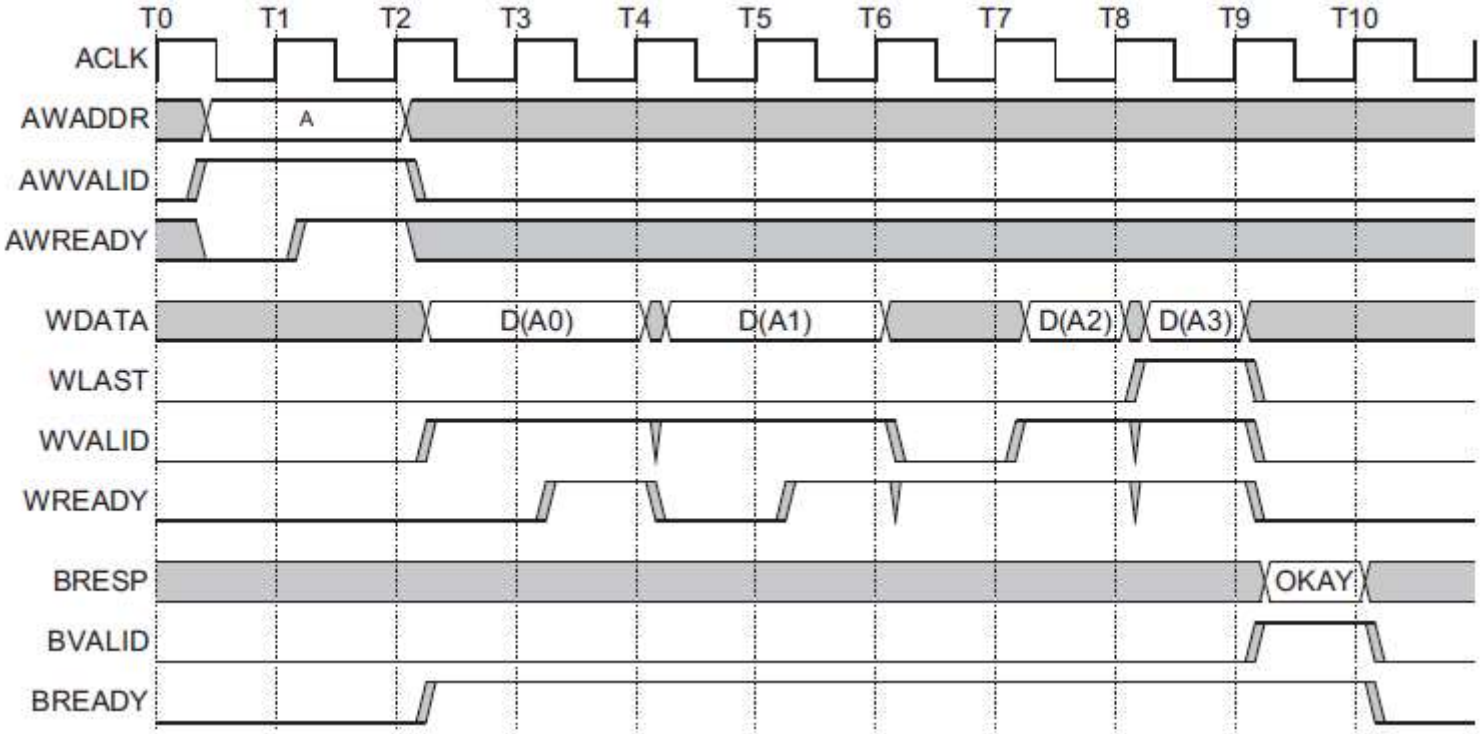
- A burst can start before the previous one is done:



† Figure extracted from: ARM Limited, *AMBA AXI Protocol v 1.0*, ARM IHI 0022B, March 19, 2004, figure 1-5, page 26.

AXI Write Burst Example

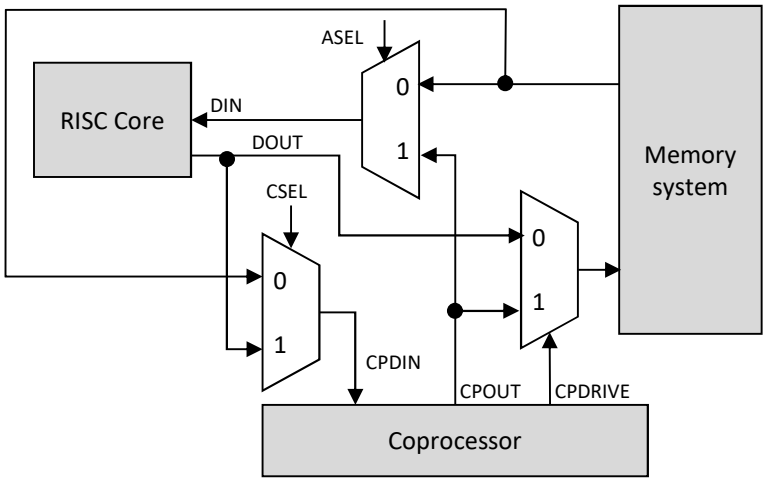
- A master perform a write burst of four data transfers:



† Figure extracted from: ARM Limited, *AMBA AXI Protocol v 1.0*, ARM IHI 0022B, March 19, 2004, figure 1-6, page 27.

Coprocessor Interface

- Traditional coprocessor interface:
  - ISA has coprocessor instructions (identify coprocessor ID)
  - Each coprocessor has its own set of registers
  - Coprocessors are not for high-bandwidth data processing
    - MMIO accelerators are more suitable for such tasks



Note:
Typical coprocessor does not control the external address bus – RISC CPU generates the required addresses.

Discussions

- Plug-and-play bus protocols enables reusable IP business
  - Small IP designers can sell their accelerator IPs to IC design houses as long as their IPs follows standard bus protocols such as the AXI bus
- On the other hand, integrating small circuit modules using AXI bus may be an overkill
  - Lightweight, proprietary bus protocols similar to AHB can be used internally within an IP