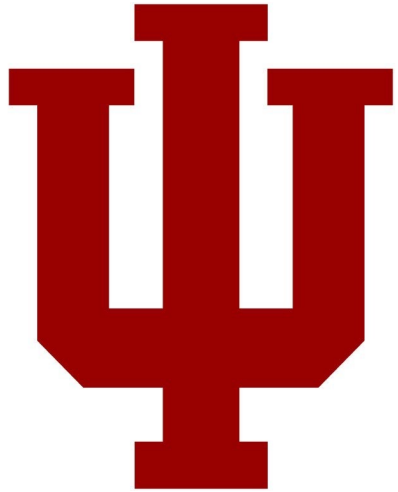
P4 due

09: AXI4 Lite II

Engr 315: Hardware / Software Codesign

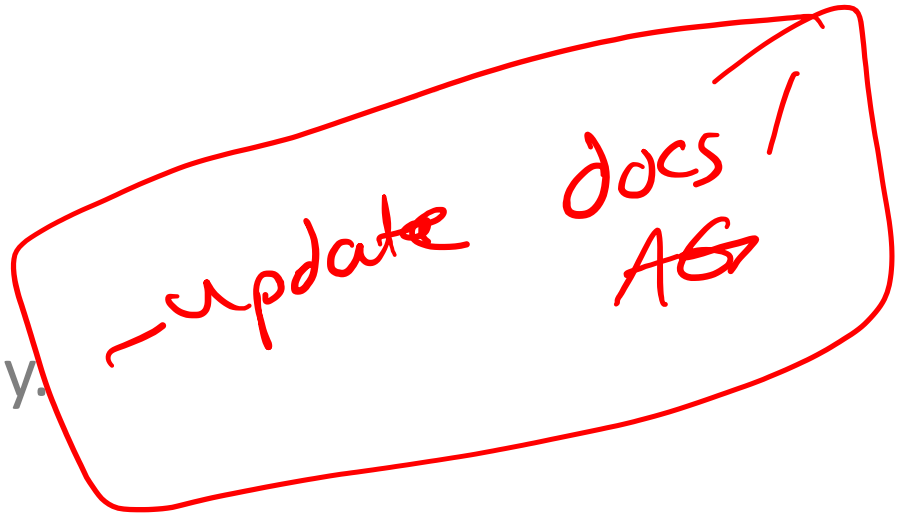
Andrew Lukefahr
Indiana University



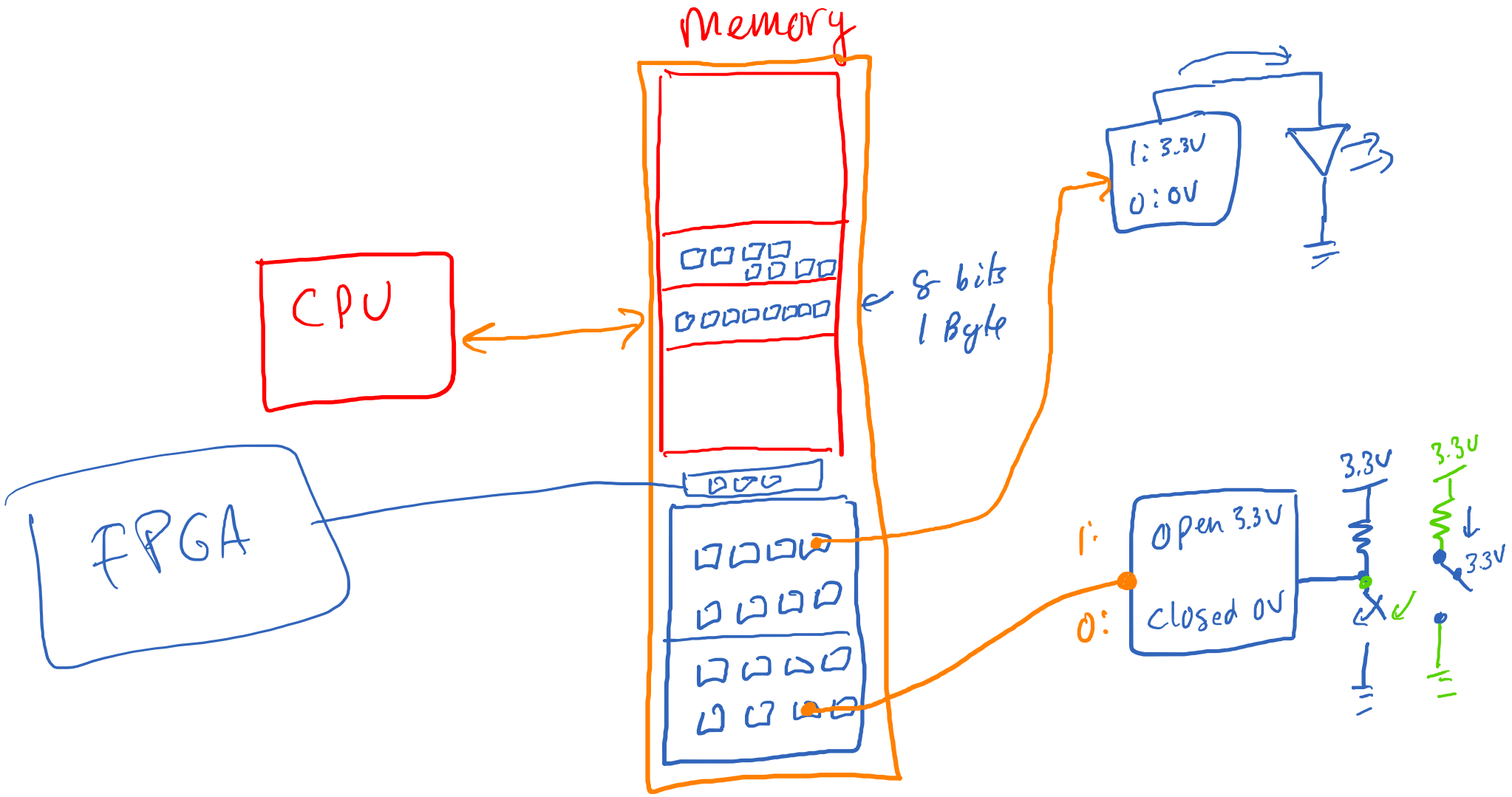
Some material taken from:
EECS 373, University of Michigan

Announcements

- P4: Due Next Wednesday.
- P5: Out soon.



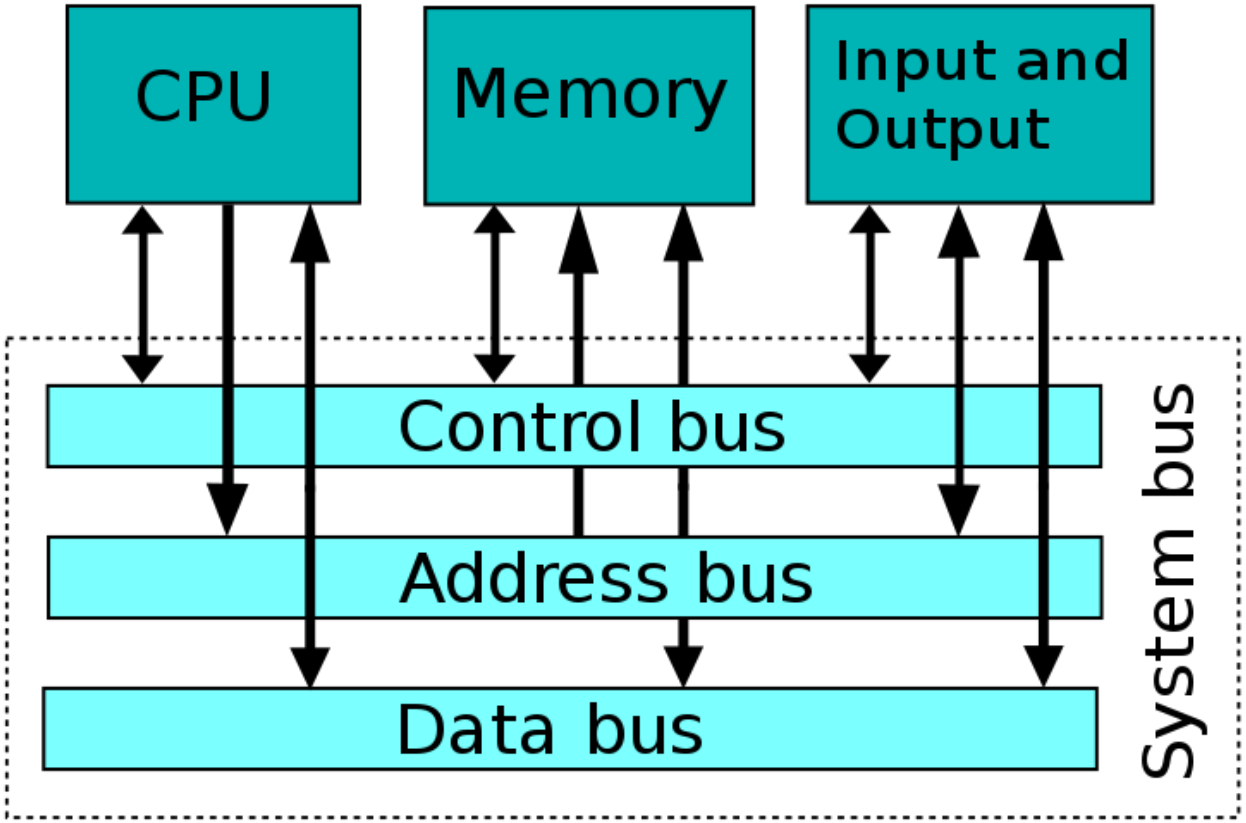
Review: Memory-Mapped I/O



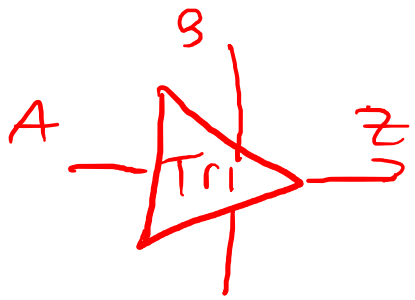
Use `volatile` for MMIO addresses!

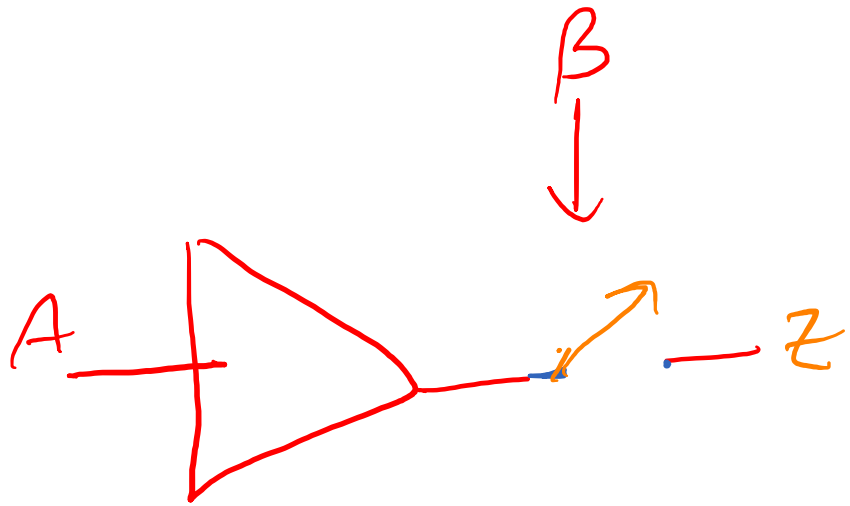
```
#define SW_ADDR 0xfffe
volatile uint32_t * SW_REG = (uint32_t * SW_ADDR);

int quit = (*SW_REG);
while(!quit)
{
    //more code
    quit = (*SW_REG);
}
```

The System Bus



Tri-State Buffer





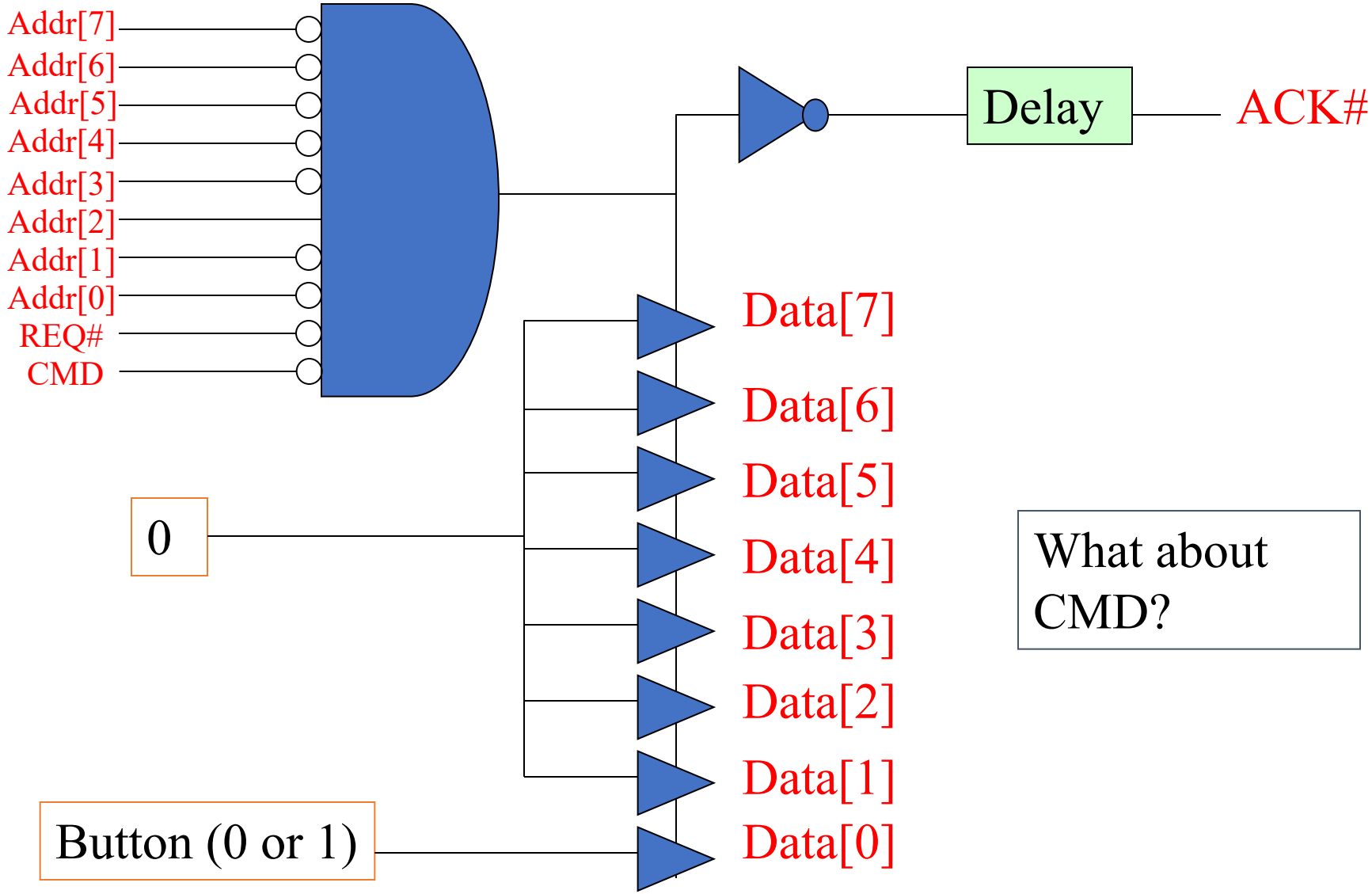
A	B	Z
0	0	High Z
0	1	0
1	0	High Z
1	1	1

A	B	Z
0	0	High-Impedance (High-Z)
0	1	0
1	0	High Z
1	1	1

Tri-State: can be 0, 1, Z

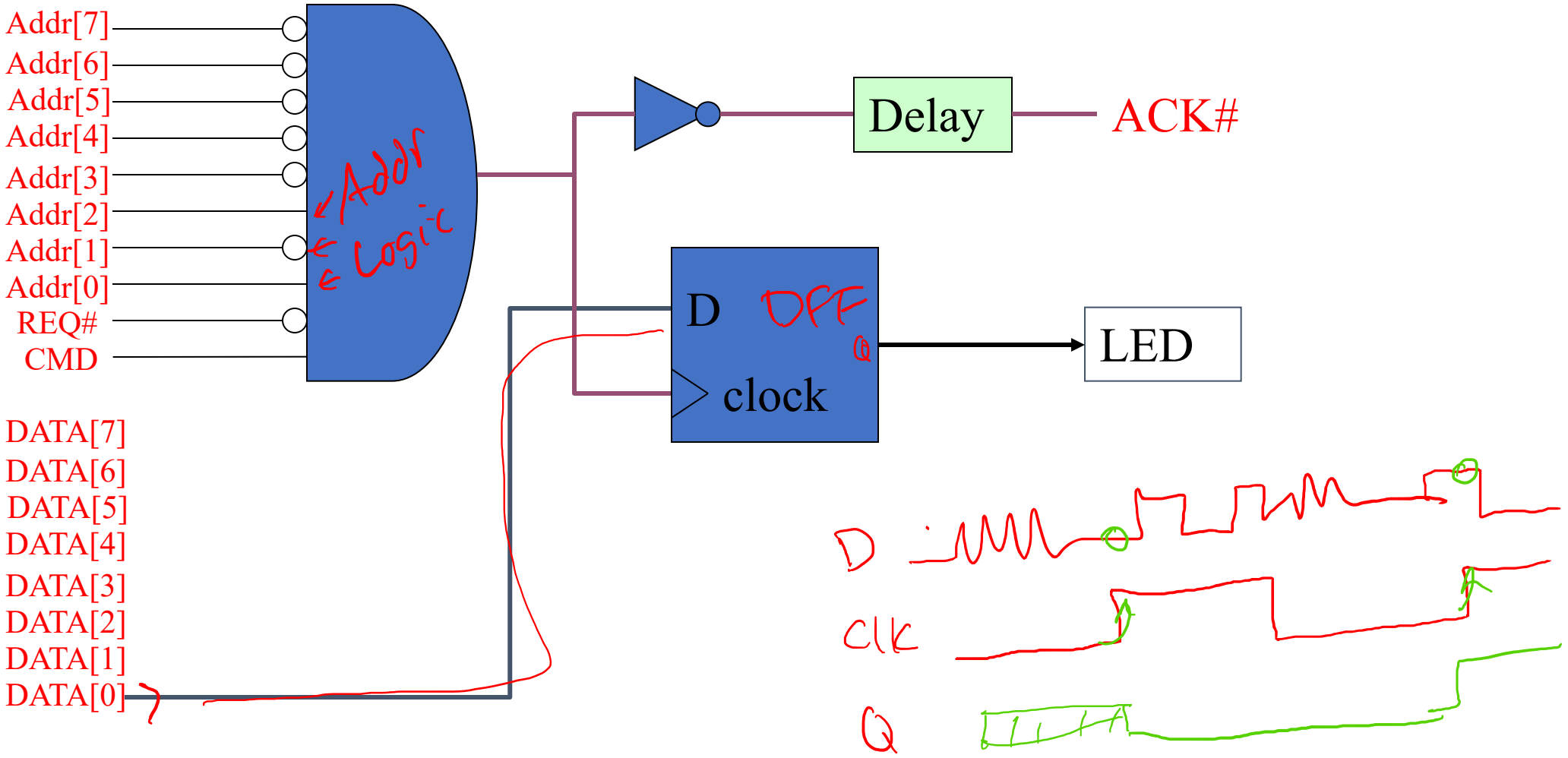
The push-button

(if Addr=0x04 read 0 or 1 depending on button)



The LED

(1 bit reg written by LSB of address 0x05)



Let’s write a simple C program to turn the LED on if button is pressed.

Peripheral Details

- 0x04: Push Button - Read-Only
  - Pushed -> 1
  - Not Pushed -> 0
- 0x05: LED Driver - Write-Only
  - On -> 1
  - Off -> 0

```
        mov r0, #0x4   % PB
        mov r1, #0x5   % LED
loop:   ldr r2, [r0, #0]
        str r2 [r1, #0]
        b loop
```

Let’s write a simple C program to turn the LED on if button is pressed.

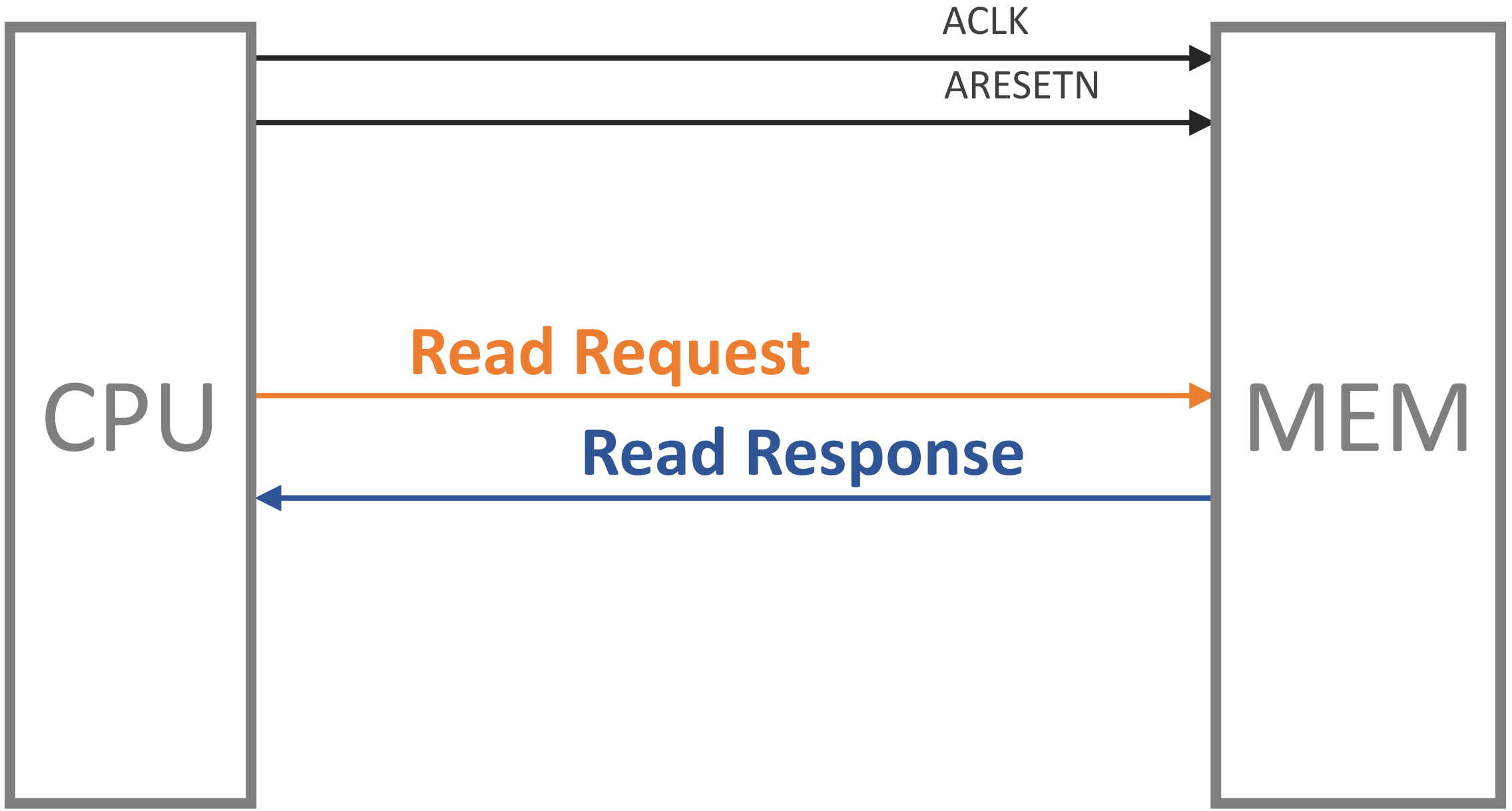
Peripheral Details

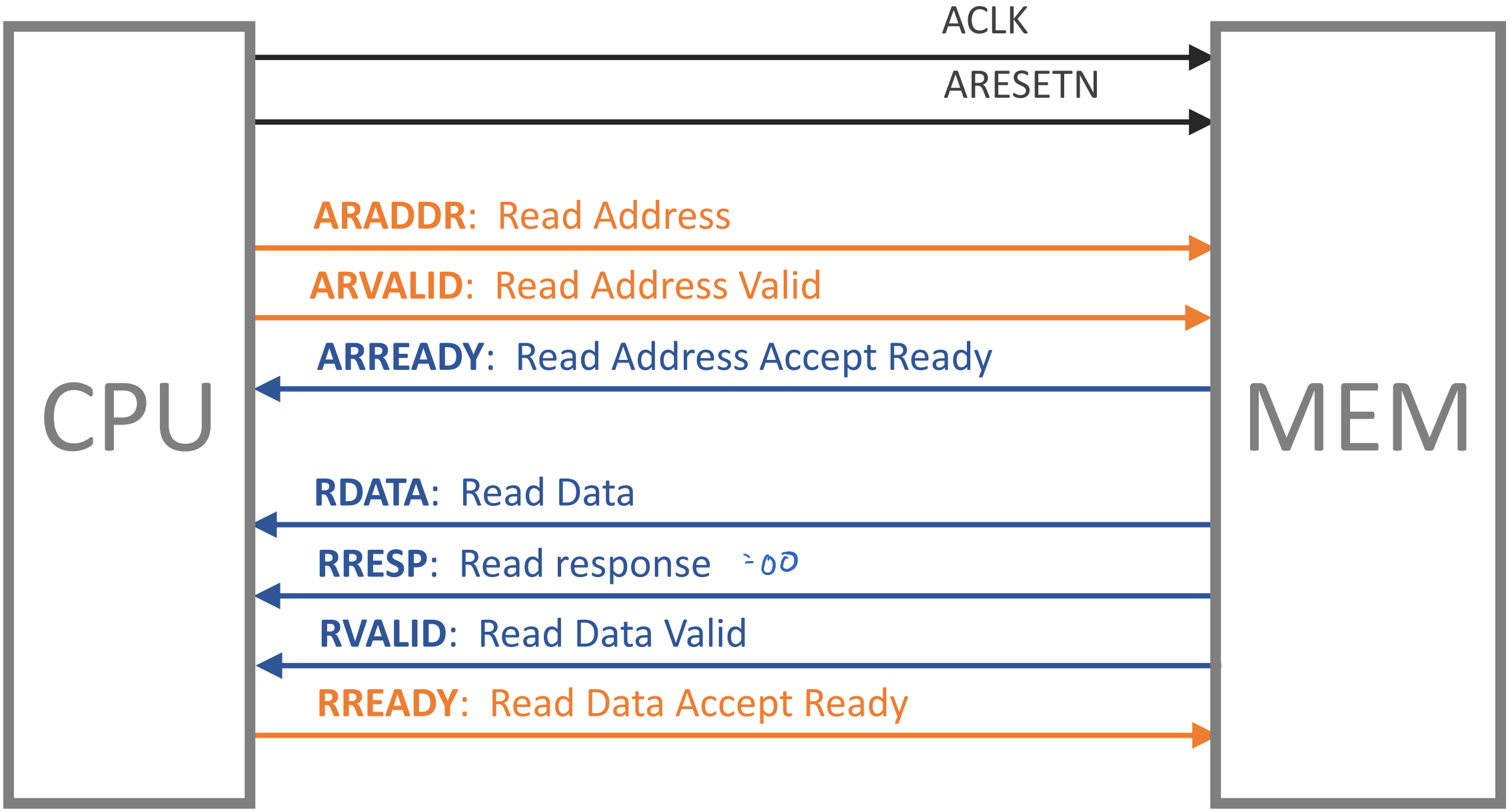
- 0x04: Push Button - Read-Only
  - Pushed -> 1
  - Not Pushed -> 0
- 0x05: LED Driver - Write-Only
  - On -> 1
  - Off -> 0

```
#include <stdio.h>
#include <inttypes.h>

#define PB_ADDR 0x4
#define LED_ADDR 0x5

int main () {
  volatile uint8_t *PB = (uint8_t*)(PB_ADDR);
  volatile uint8_t *LED = (uint8_t*)(LED_ADDR);

  while (1)
    *LED = *PB;
}
```

ARM AXI Bus

- "Advanced eXtensible Interface" Bus Version 4, "AXI4"
- Three Variants
  - AXI4: Fast but complicated; Memory-mapped
  - AXI4 Lite: Slow but simple; Memory-mapped
  - AXI4 Stream: Fast and simple; Not memory-mapped
    - P3 secretly uses this


ACLK
ARESETN
CPU
Read Request
Read Response
MEM


ACLK
ARESETN
ARADDR: Read Address
ARVALID: Read Address Valid
ARREADY: Read Address Accept Ready
CPU
MEM
RDATA: Read Data
RRESP: Read response
RVALID: Read Data Valid
RREADY: Read Data Accept Ready

What is RRESP?

Table A3-4 RRESP and BRESP encoding

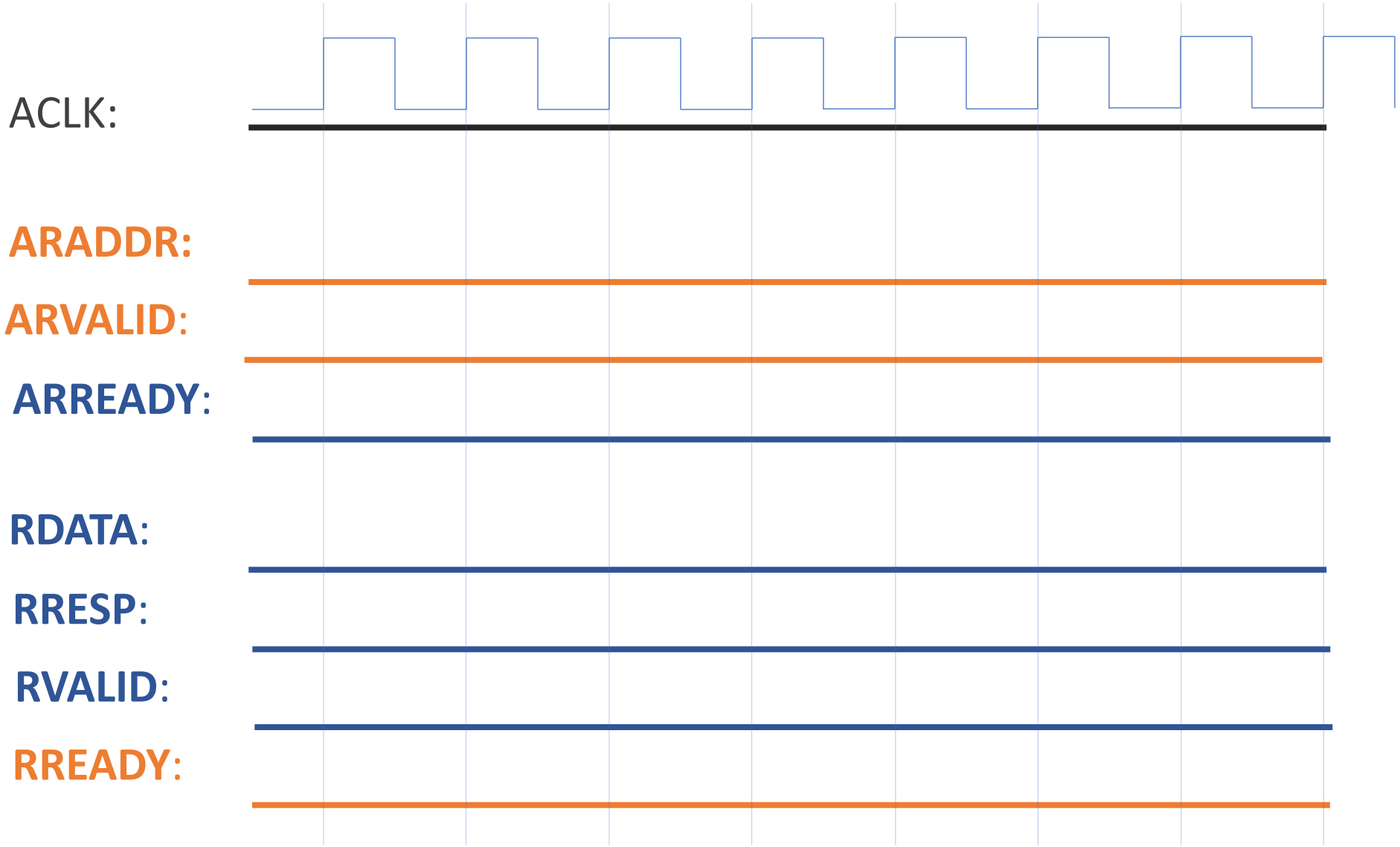
RRESP[1:0] BRESP[1:0]	Response
0b00	OKAY
0b01	EXOKAY
0b10	SLVERR
0b11	DECERR

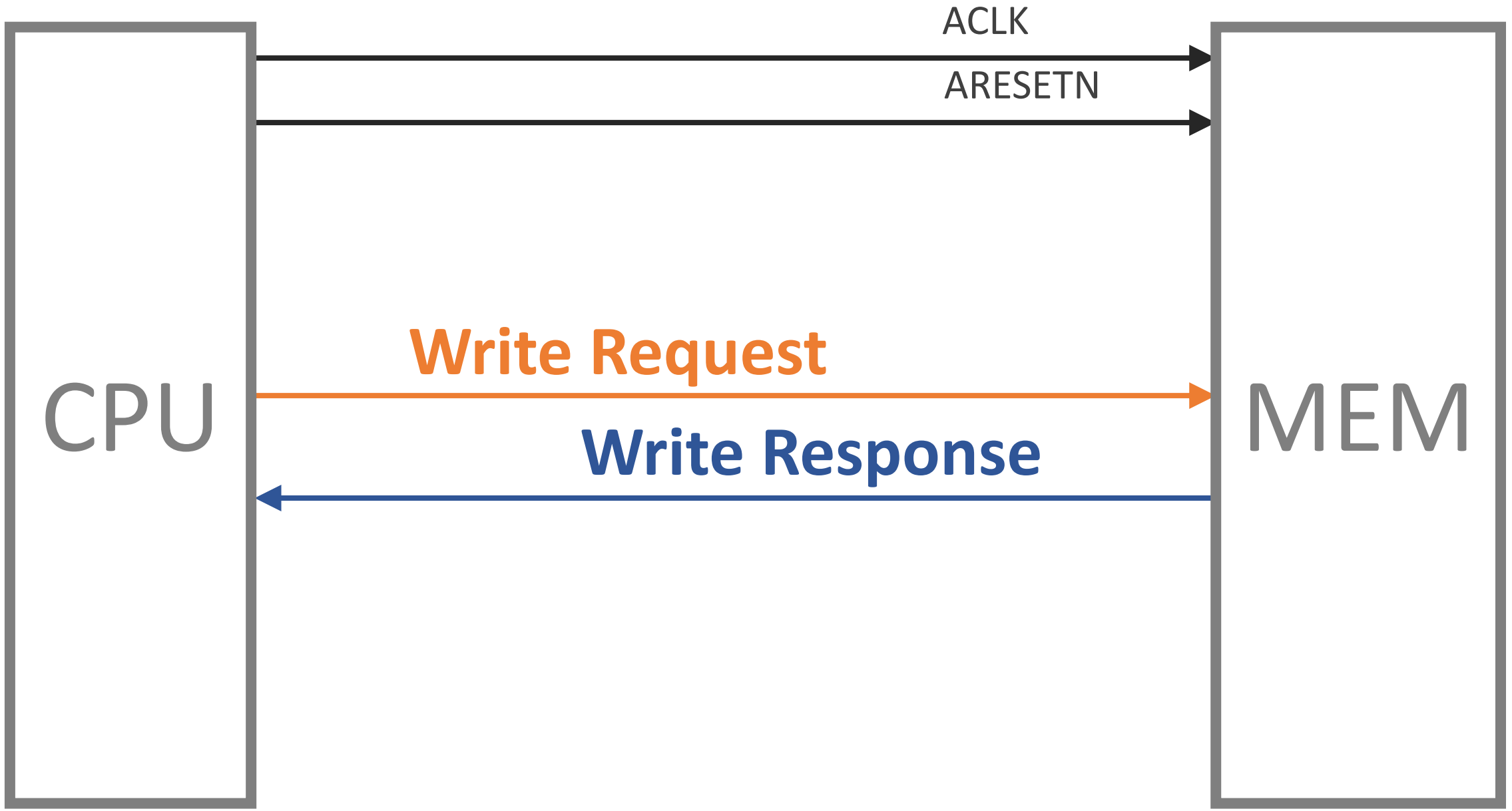
- Mostly used to send error codes back to CPU

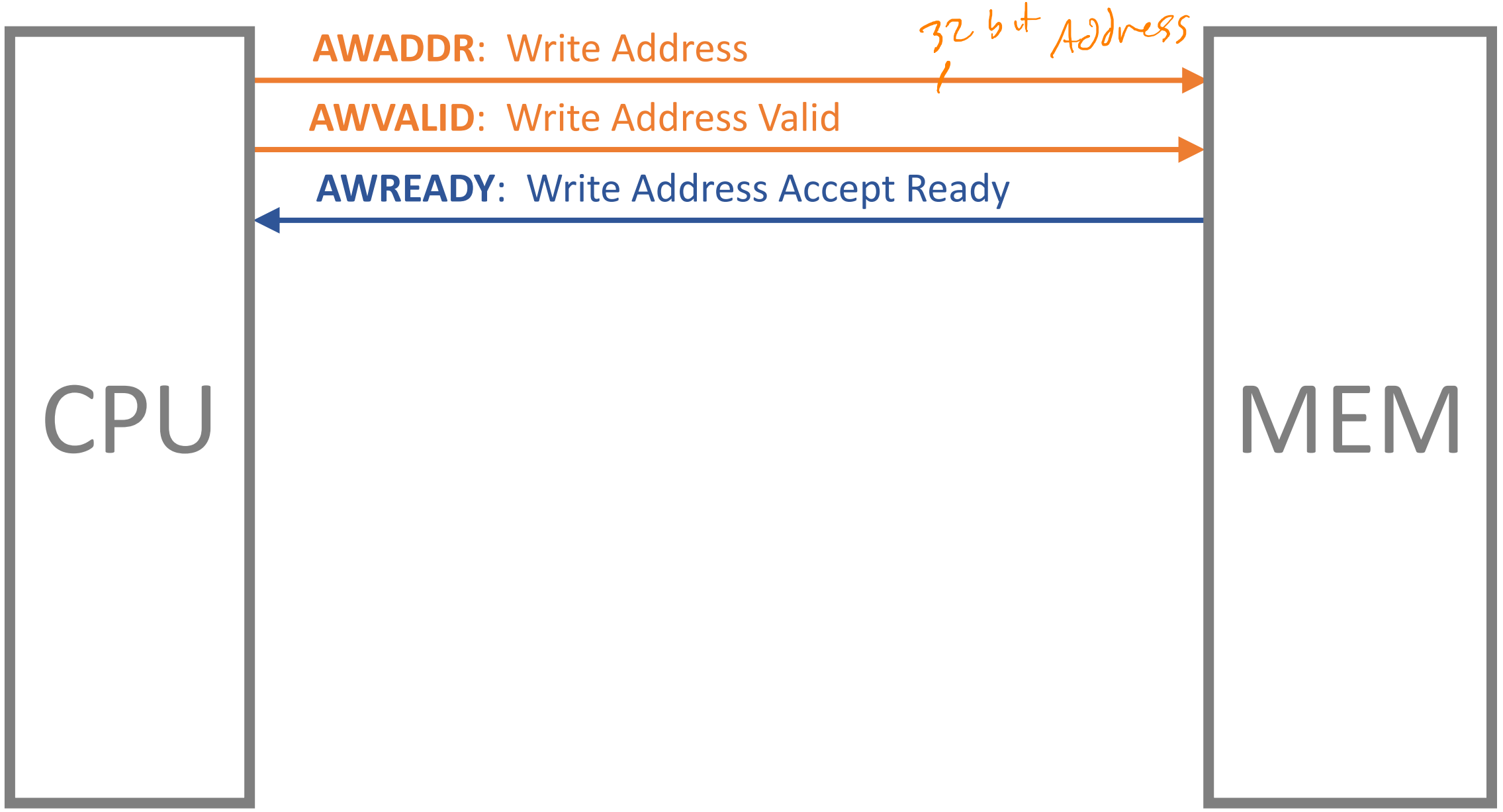
- We'll always just use 0b00

Load 0x1234, response: 0xabcd

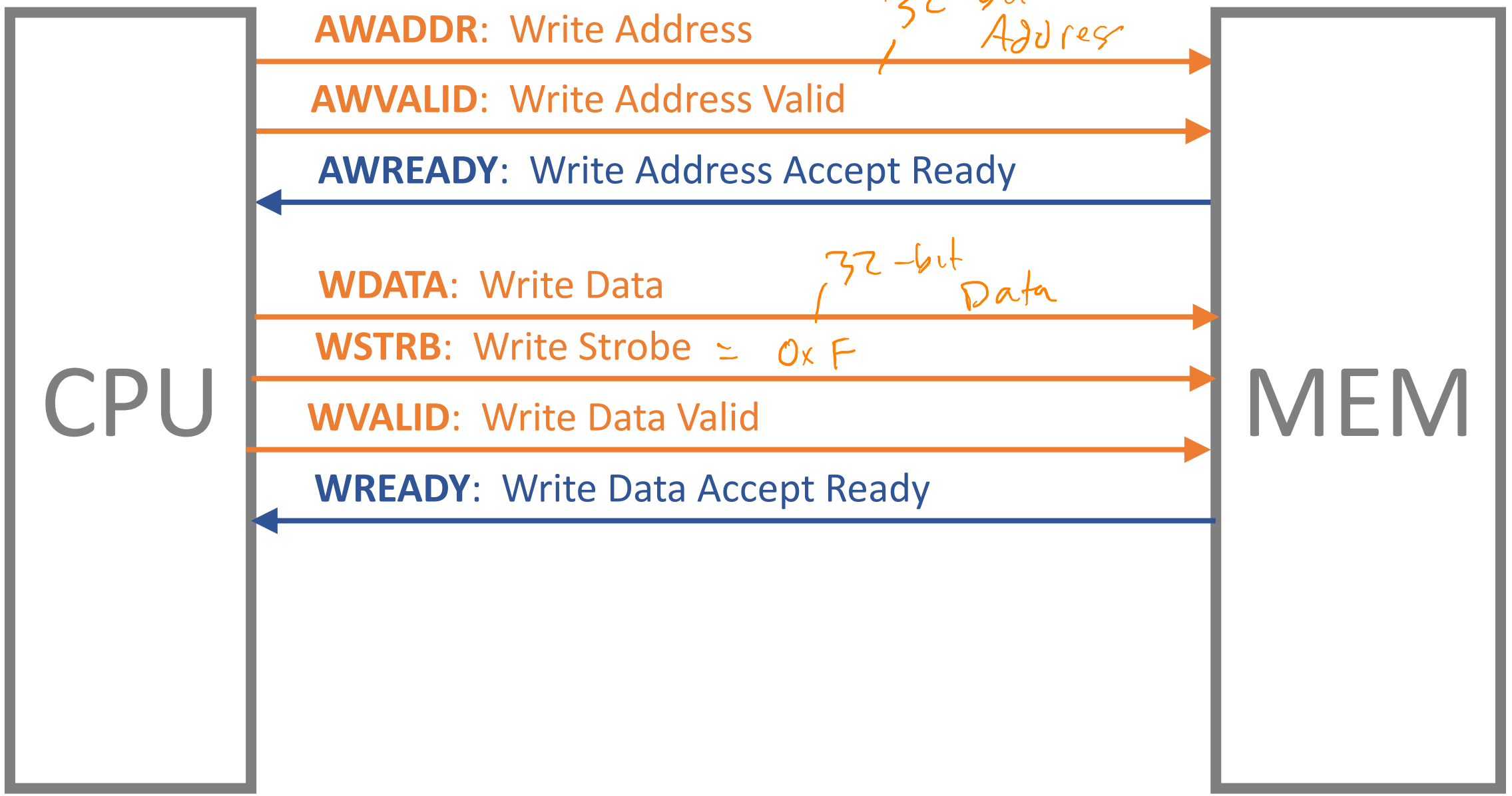
assume ARESETN = 1




ACLK
ARESETN
CPU
Write Request
Write Response
MEM



ACLK and ARESETN not shown



ACLK and ARESETN not shown

What is WSTRB?

The **WSTRB[n:0]** signals when HIGH, specify the byte lanes of the data bus that contain valid information. There is one write strobe for each eight bits of the write data bus, therefore **WSTRB[n]** corresponds to **WDATA[(8n)+7: (8n)]**

Just like TKEEP of AXI-Stream

What is WSTRB here?

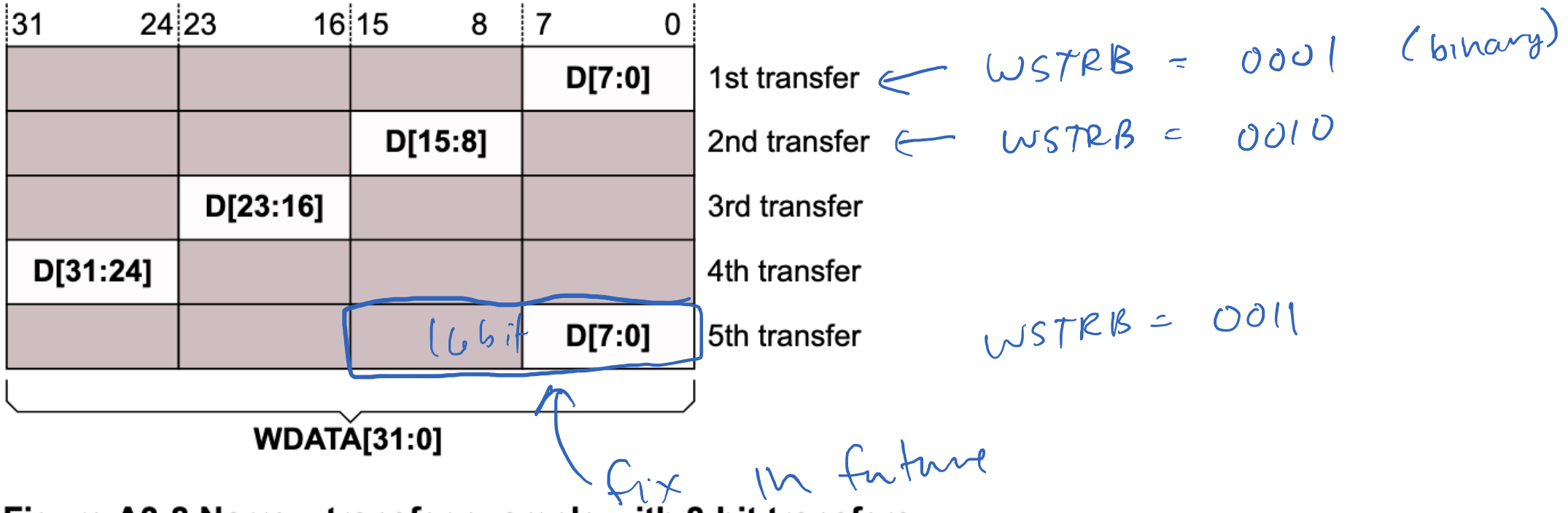
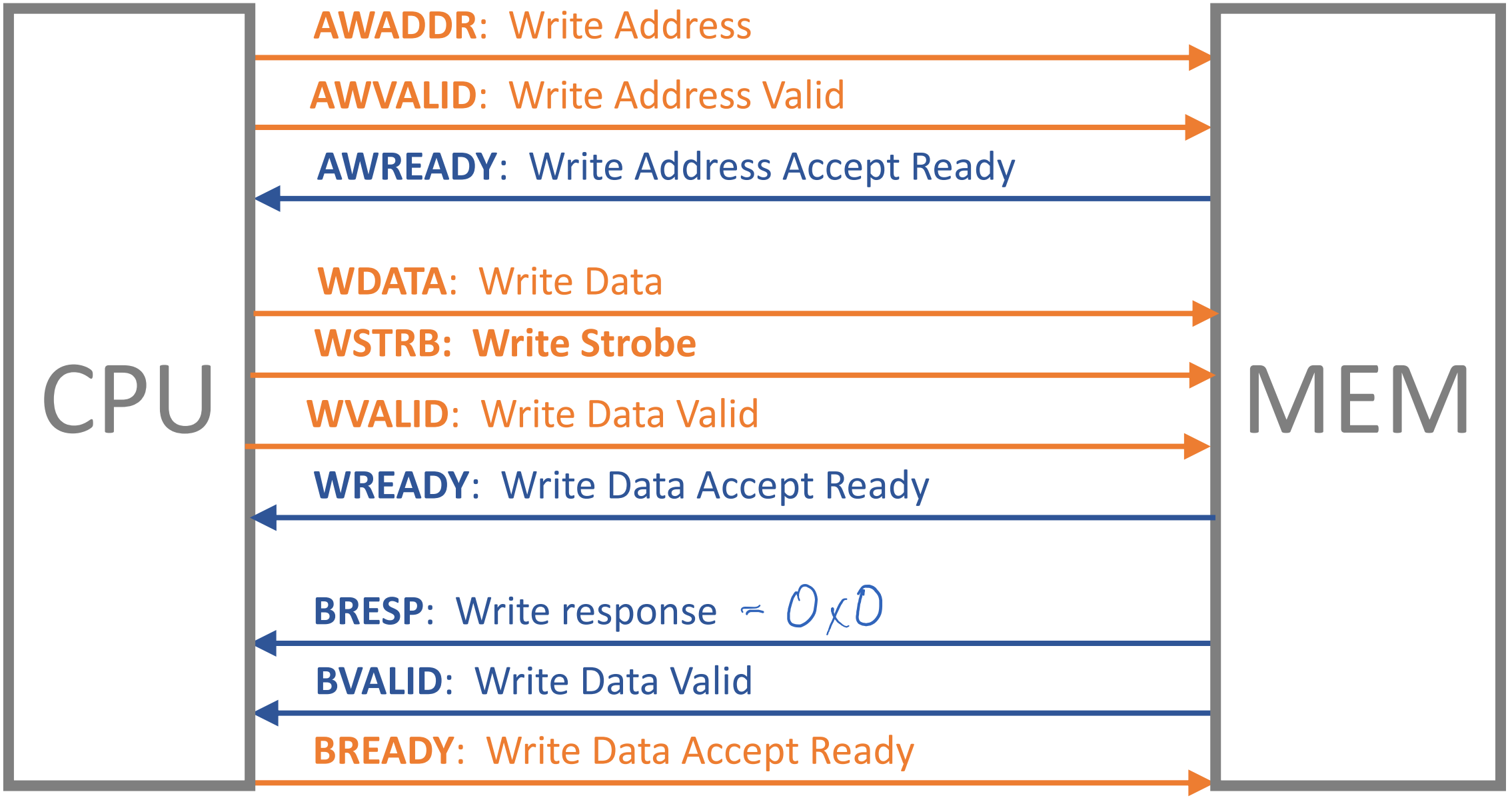


Figure A3-8 Narrow transfer example with 8-bit transfers



ACLK and ARESETN not shown

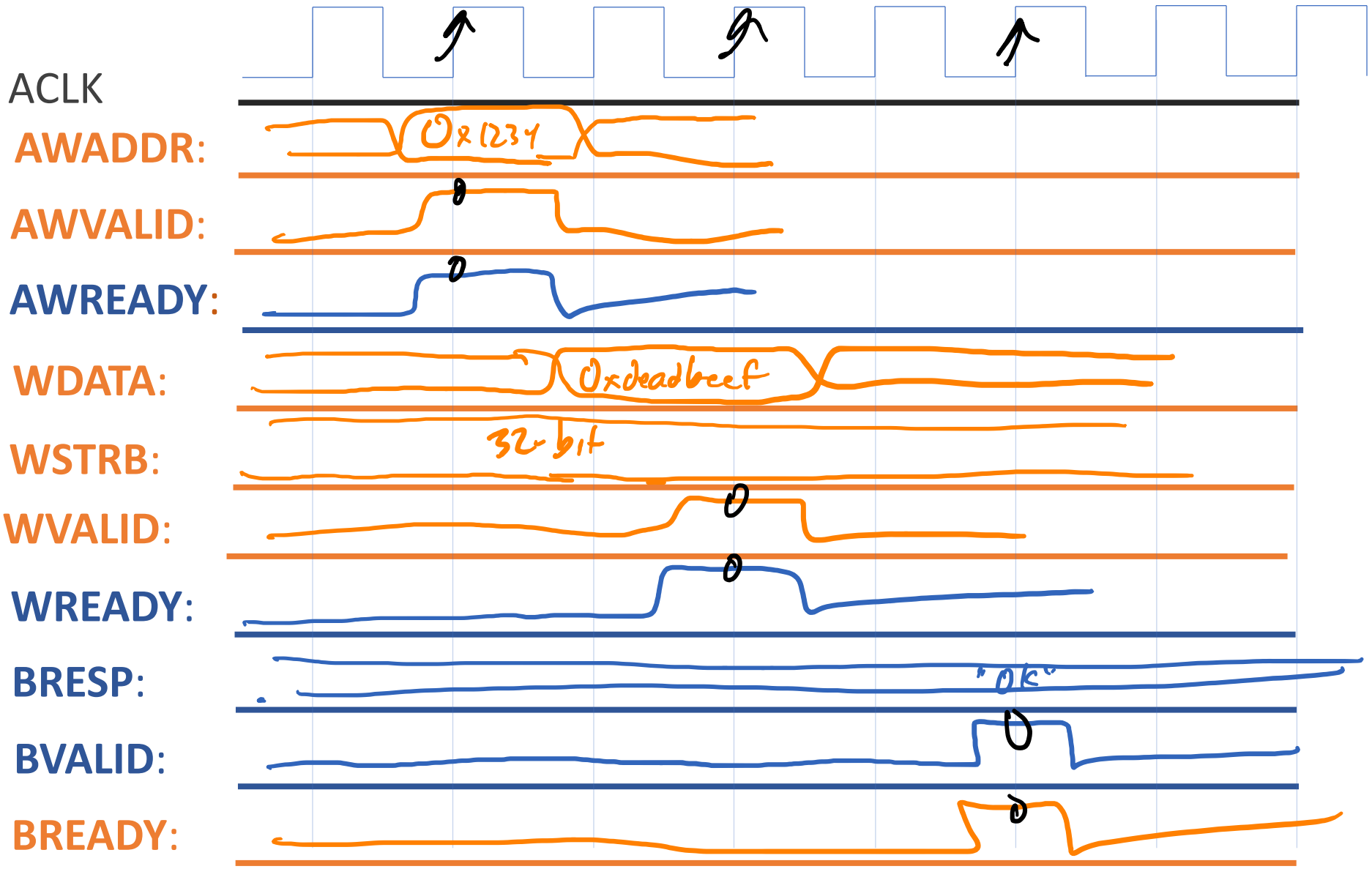
BRESP is just like RRESP

Table A3-4 RRESP and BRESP encoding

RRESP[1:0] BRESP[1:0]	Response
0b00	OKAY
0b01	EXOKAY
0b10	SLVERR
0b11	DECERR

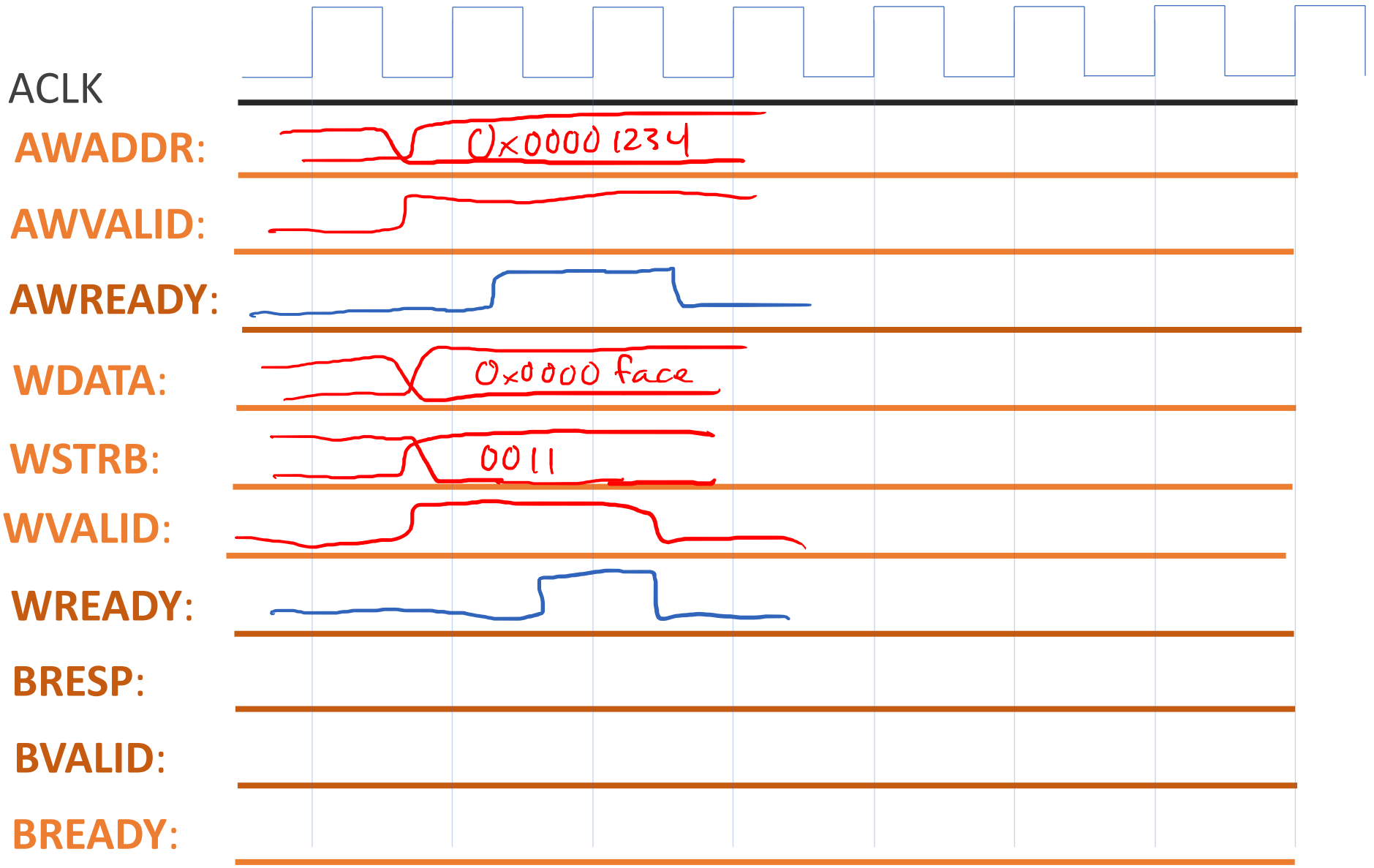
- Mostly used to send error codes back to CPU
- We'll always just use 0b00

Writing 0xdeadbeef to 0x1234



Writing 0xface to 0x1234

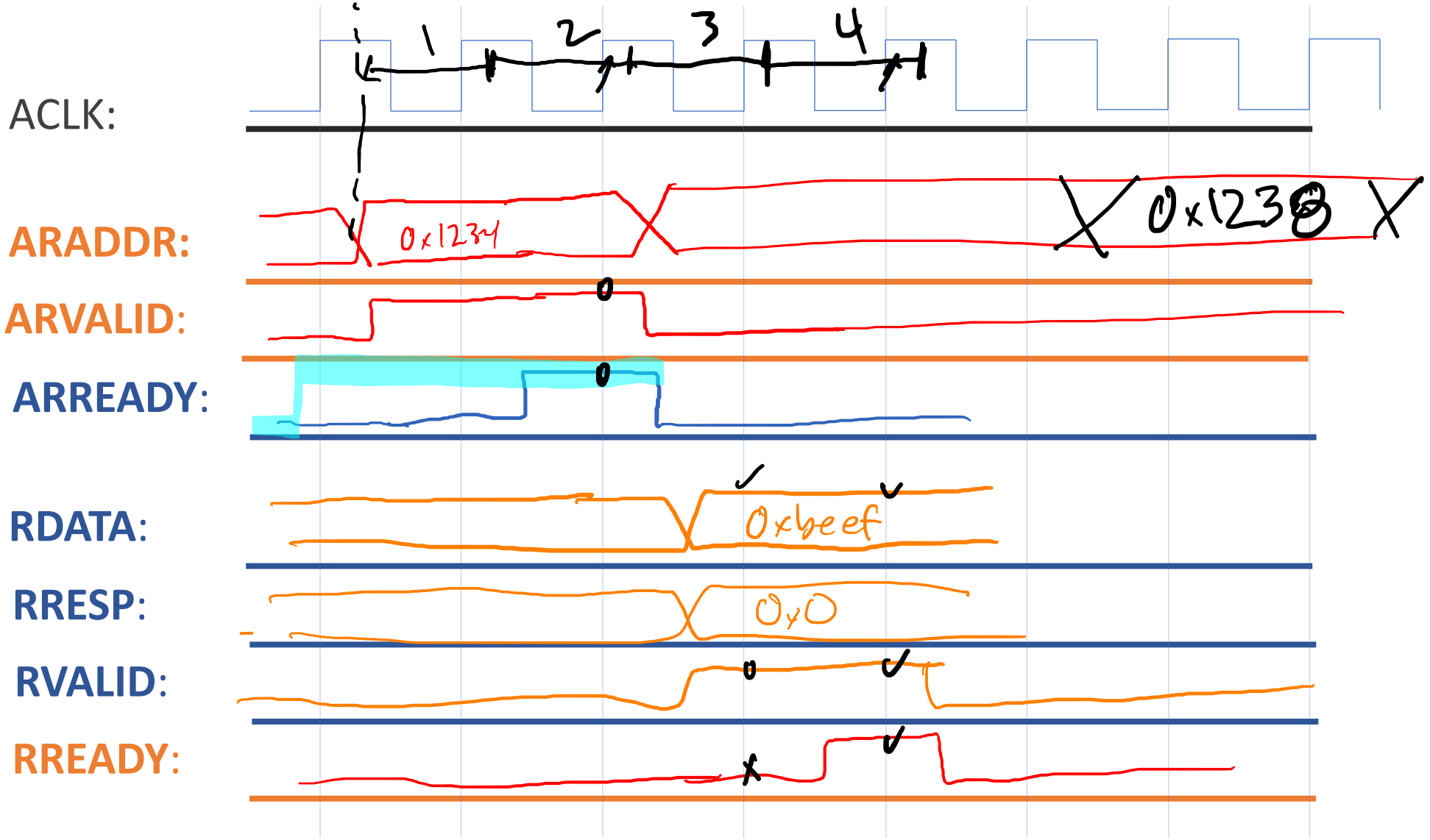
16-bit



ARM AXI Bus

- "Advanced eXtensible Interface" Bus Version 4, "AXI4"
- Three Variants
  - AXI4: Fast but complicated; Memory-mapped
  - AXI4 Lite: Slow but simple; Memory-mapped
  - AXI4 Stream: Fast and simple; Not memory-mapped

How long does a read(load) take?



High-Performance Bus Ideas

- Make single transaction faster

AXI Handshake Speedup

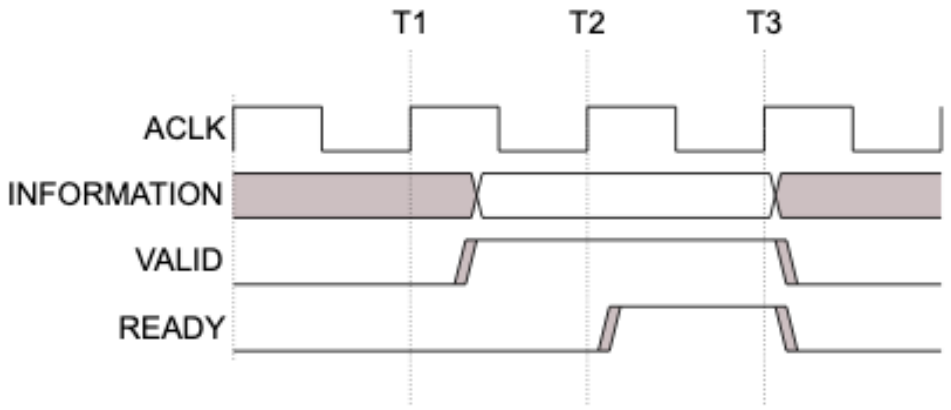


Figure A3-2 VALID before READY handshake

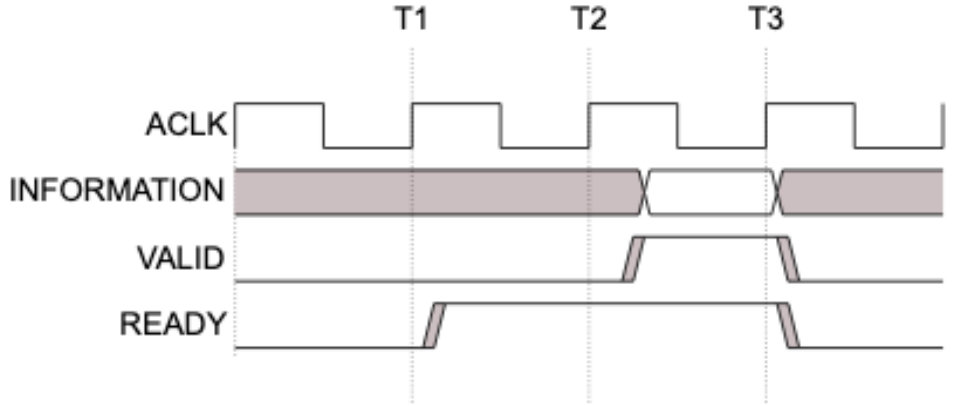
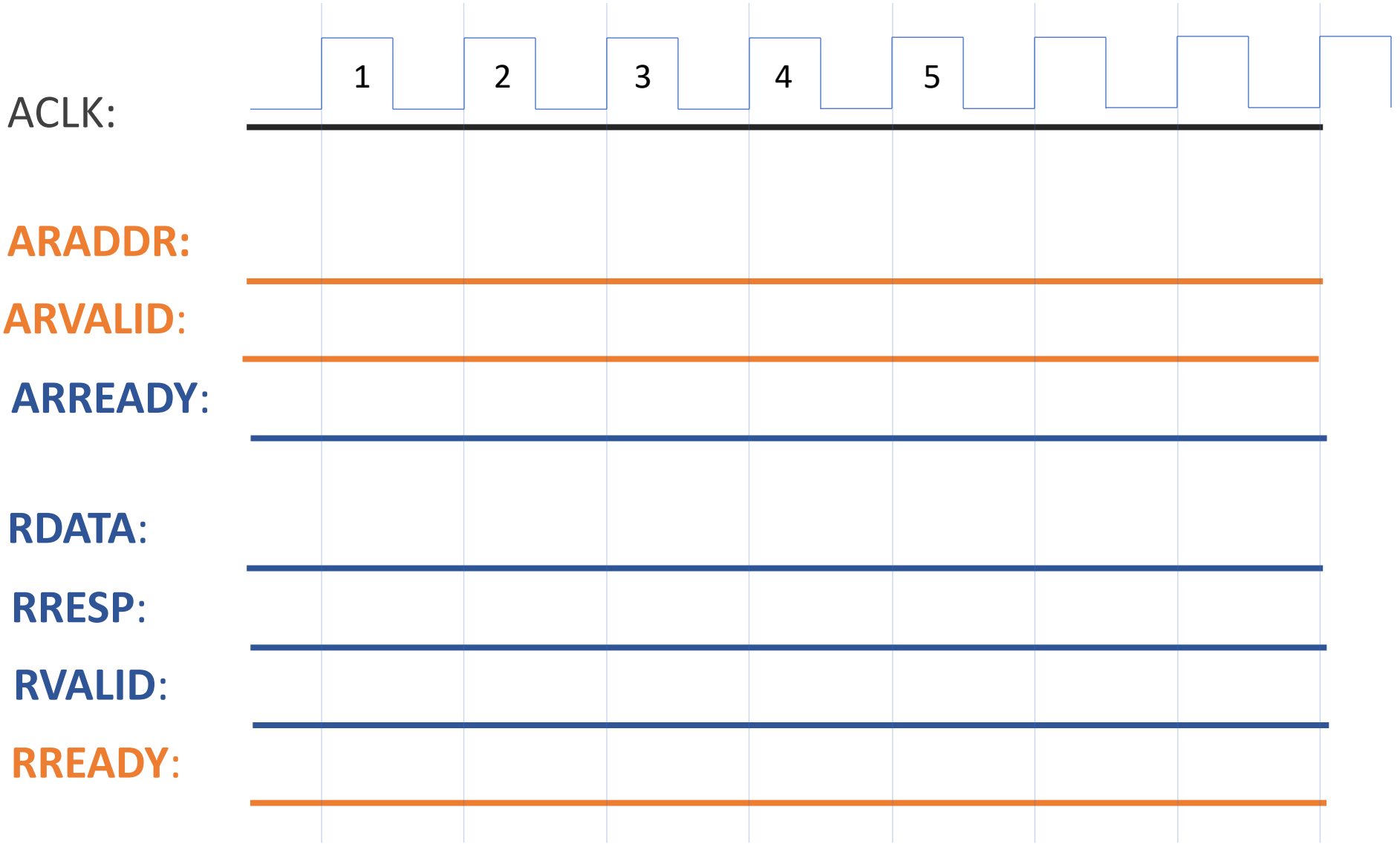


Figure A3-3 READY before VALID handshake

- Both are valid
- Right is faster

What can we do to make this faster?

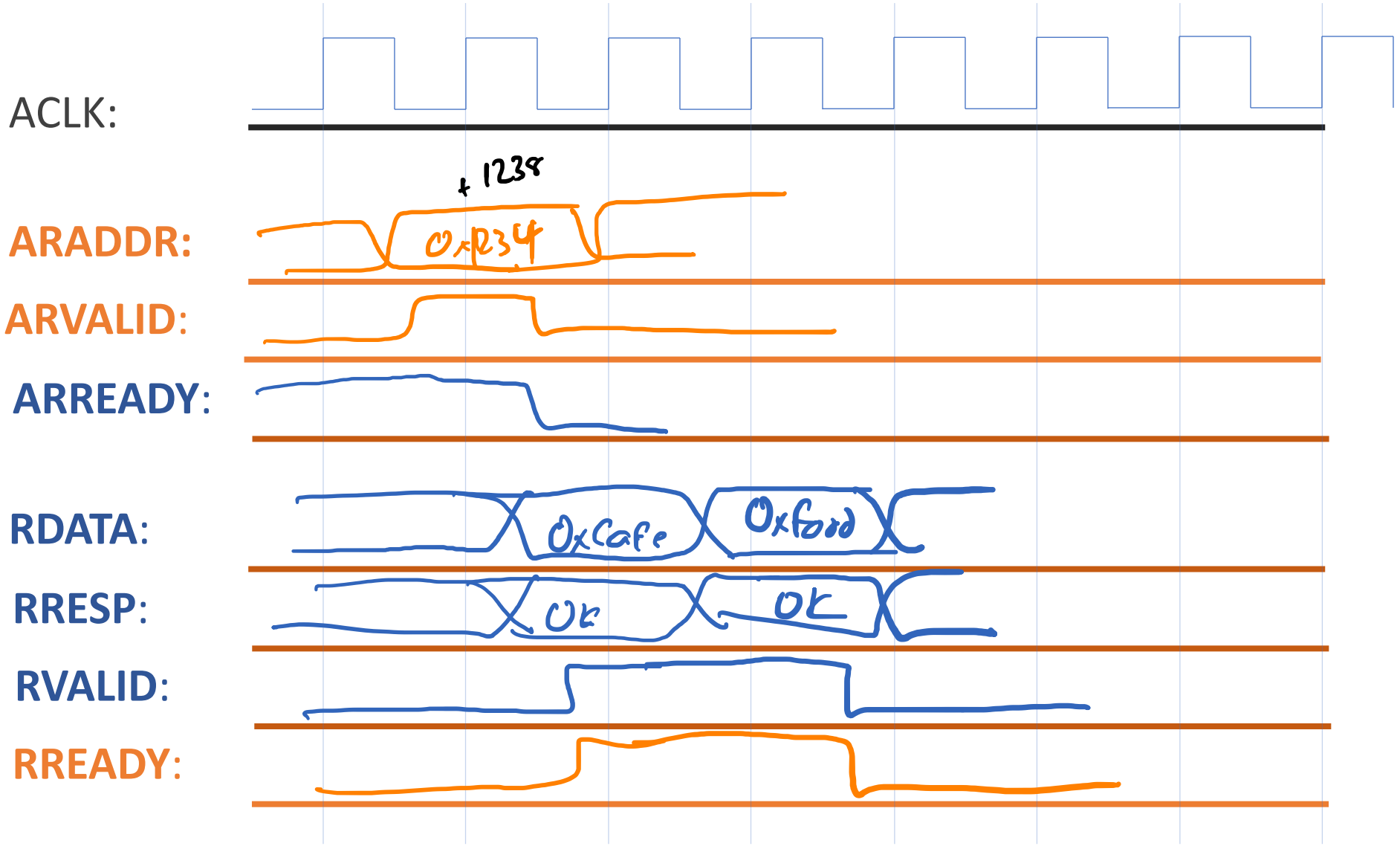


High-Performance Bus Ideas

- Make single transaction faster
- Overlap multiple transactions

Can we load 0x1234 and 0x1238?

assume ARESETN = 1



Burst Transactions

- When a device is transmitting data repeatedly without going through all the steps required to transmit each piece of data in a separate transaction

Burst Transaction

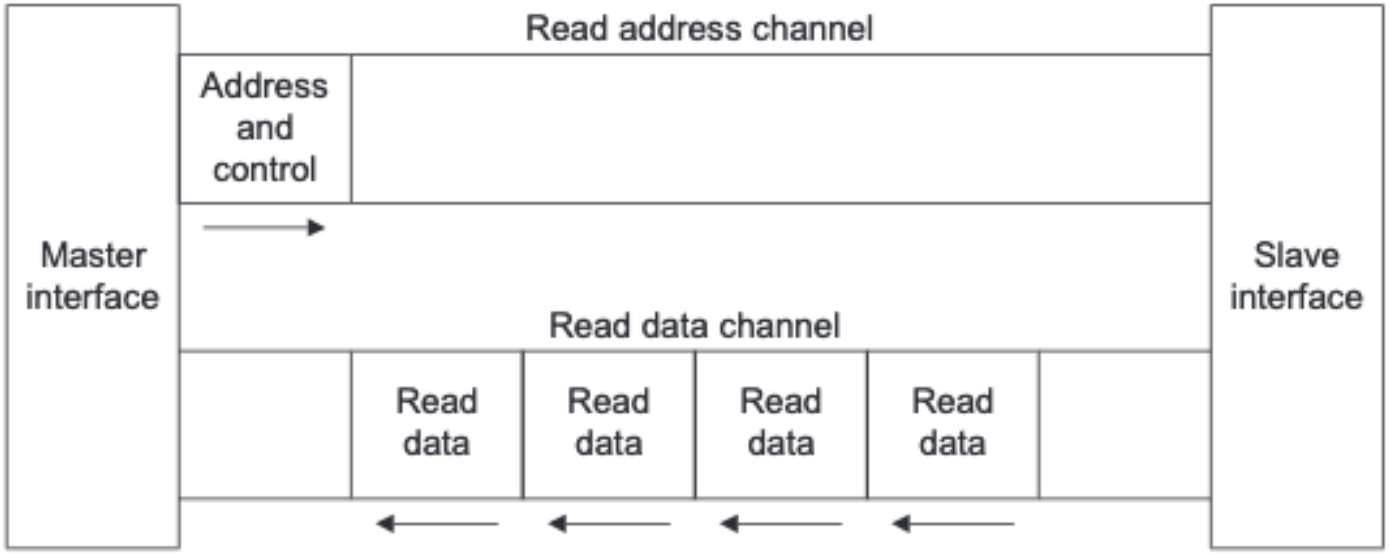
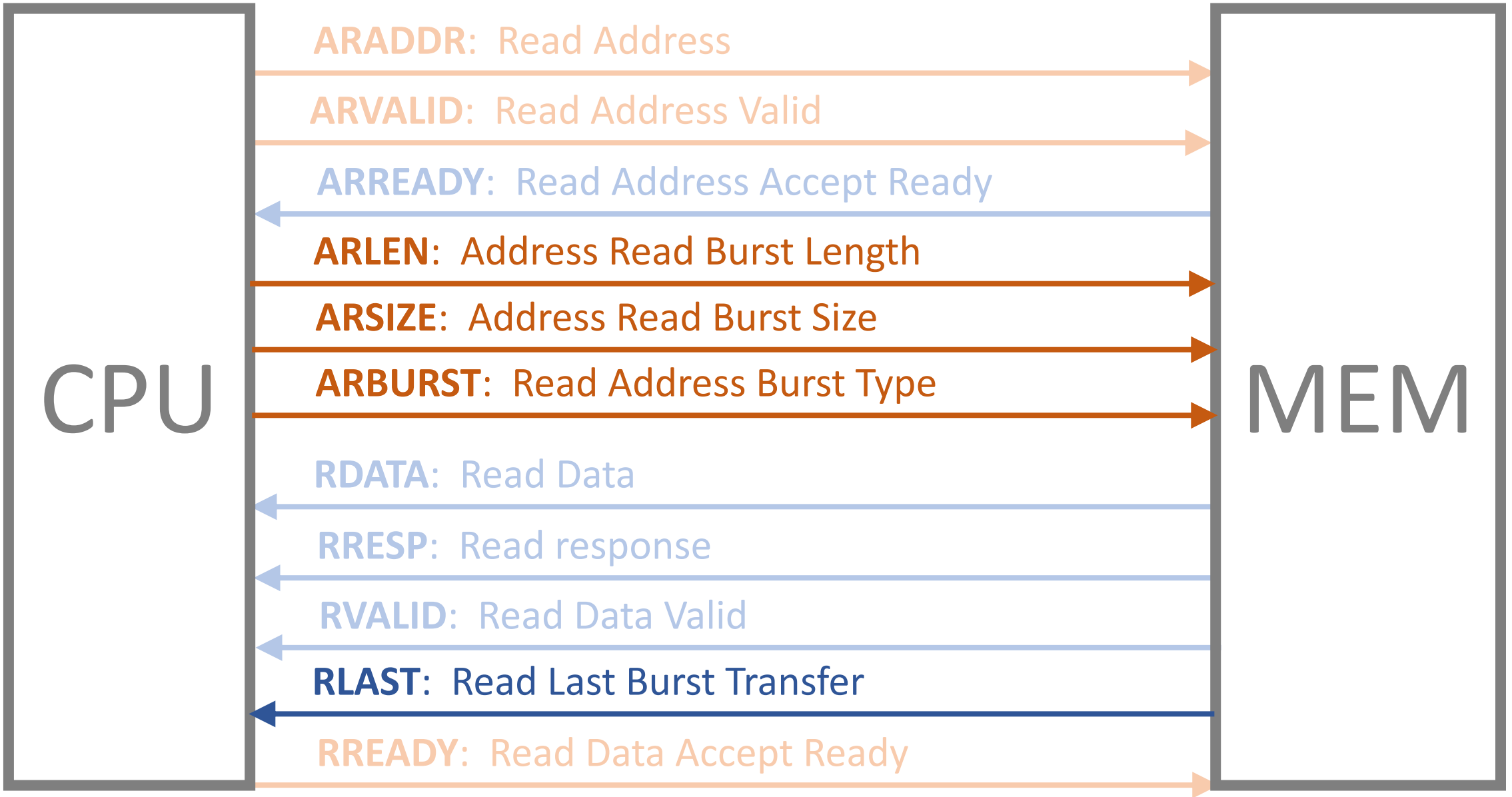


Figure 1-1 Channel architecture of reads


CPU
MEM
ARADDR: Read Address
ARVALID: Read Address Valid
ARREADY: Read Address Accept Ready
ARLEN: Address Read Burst Length
ARSIZE: Address Read Burst Size
ARBURST: Read Address Burst Type
RDATA: Read Data
RRESP: Read response
RVALID: Read Data Valid
RLAST: Read Last Burst Transfer
RREADY: Read Data Accept Ready

What do the new signals do?

ARLEN: Address Read Burst Length

How many bursts should occur? (+1)

Burst_Length = ARLEN[7:0] + 1

What do the new signals do?

ARLEN: Address Read Burst Length

How many bursts should occur? (+1)

ARSIZE: Address Read Burst Size

How many bytes should be in each burst?

2^n Bytes / Transfer

Table A3-2 Burst size encoding

AxSIZE[2:0]	Bytes in transfer
0b000	1
0b001	2
0b010	4
0b011	8
0b100	16
0b101	32
0b110	64
0b111	128

INCR: 0x100, 4 → [0x100], [0x104], [0x108], [0x10C]

Fixed: 0x100, 4 → [0x100], [0x100], [0x100], [0x100]

What do the new signals do?

ARLEN: Address Read Burst Length

How many bursts should occur? (+1)

ARSIZE: Address Read Burst Size

How many bytes should be in each burst?

ARBURST: Read Address Burst Type

Are the addresses incrementing, or repeating?

Table A3-3 Burst type encoding

AxBURST[1:0]	Burst type
0b00	FIXED
0b01	INCR
0b10	WRAP
0b11	Reserved

FIXED: The address is the same for every transfer (Next Address = Address)

INCR: The address for each transfer is an increment of previous transfer

(Next Address = Address + 0x4)

What do the new signals do?

ARLEN: Address Read Burst Length

How many bursts should occur? (+1)

ARSIZE: Address Read Burst Size

How many bytes should be in each burst?

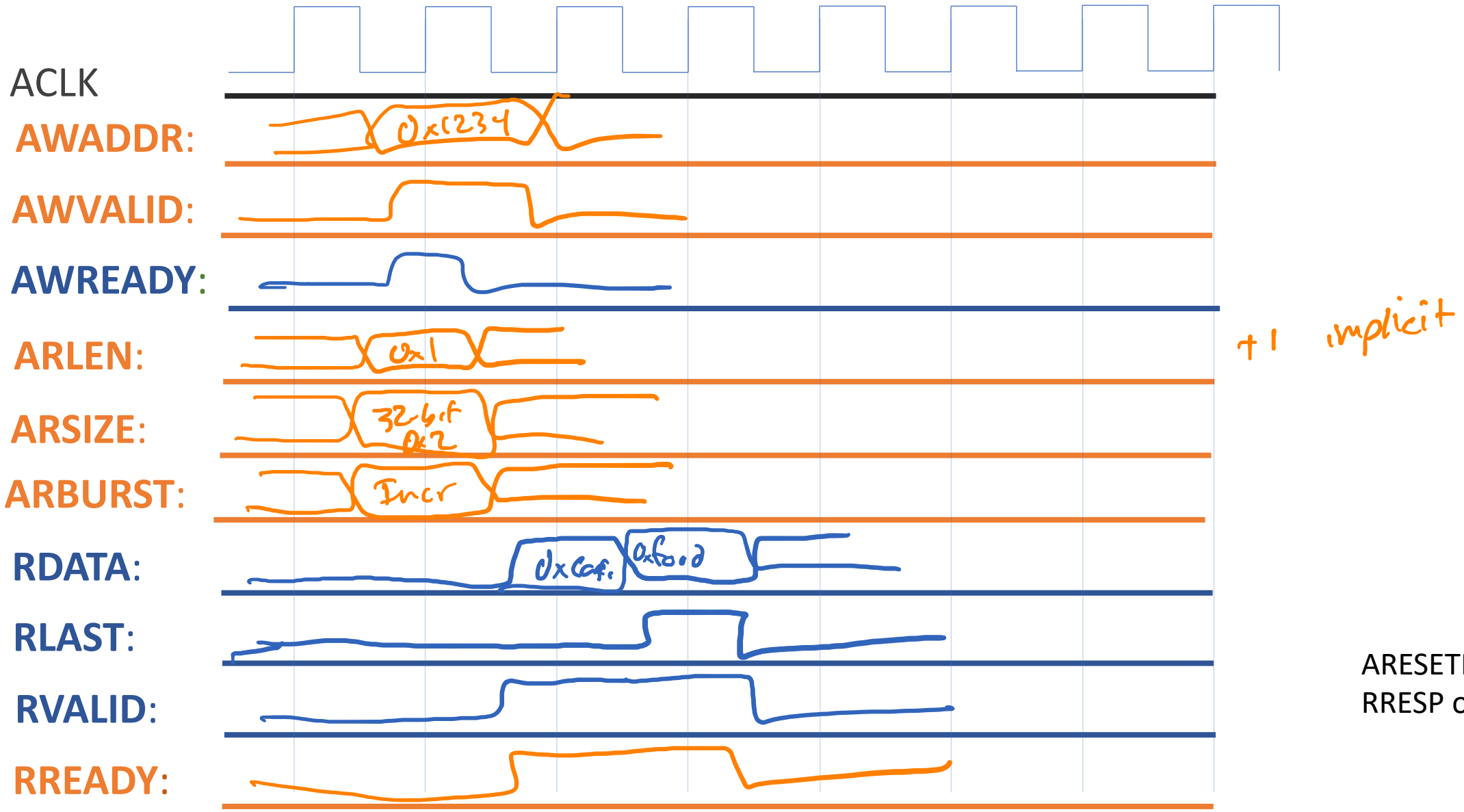
ARBURST: Read Address Burst Type

Are the addresses incrementing, or repeating?

RLAST: Read Last Burst Transfer

Are we done yet?

Reading 0x1234 and 0x1238



ARESETN and
RRESP omitted

Read Burst Example

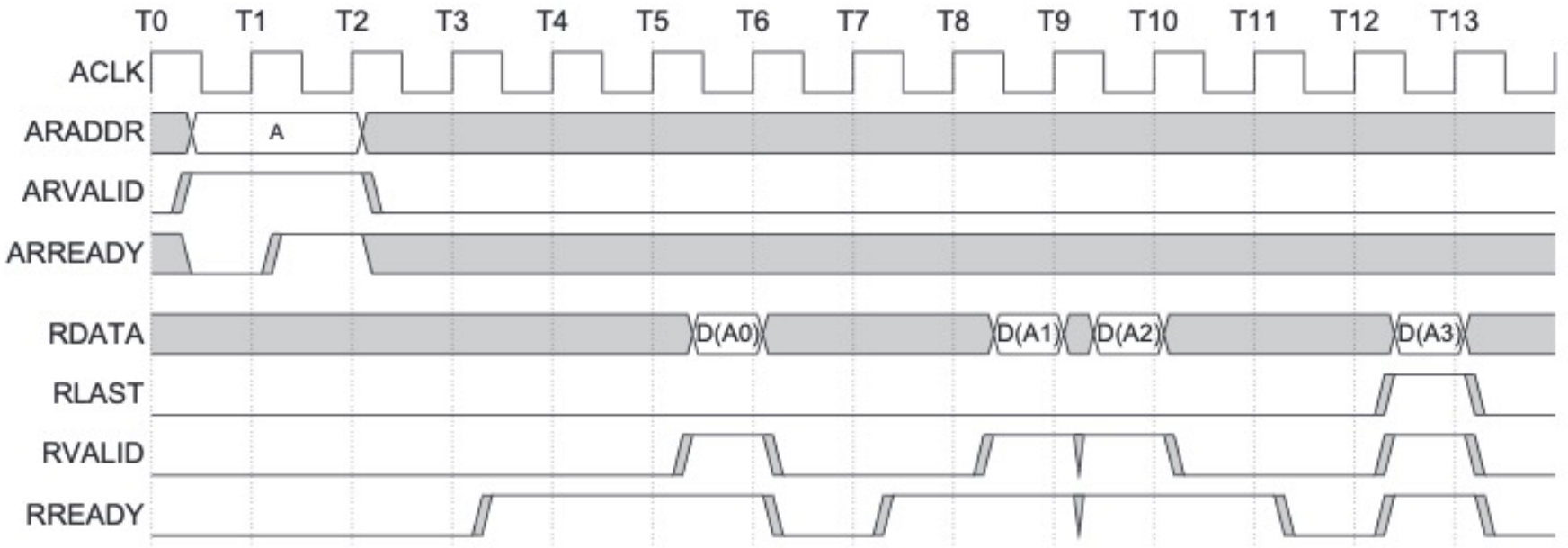


Figure 1-4 Read burst

Note

The master also drives a set of control signals showing the length and type of the burst, but these signals are omitted from the figure for clarity.

Overlapping Read Burst Transactions

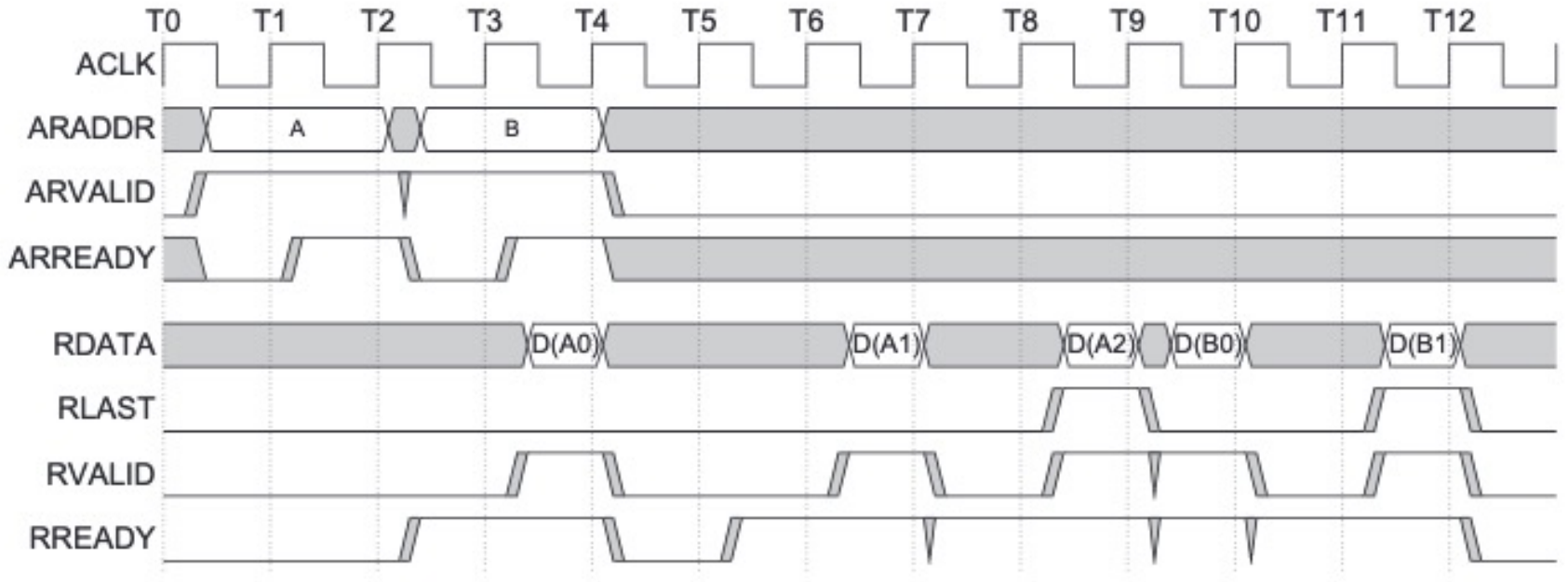
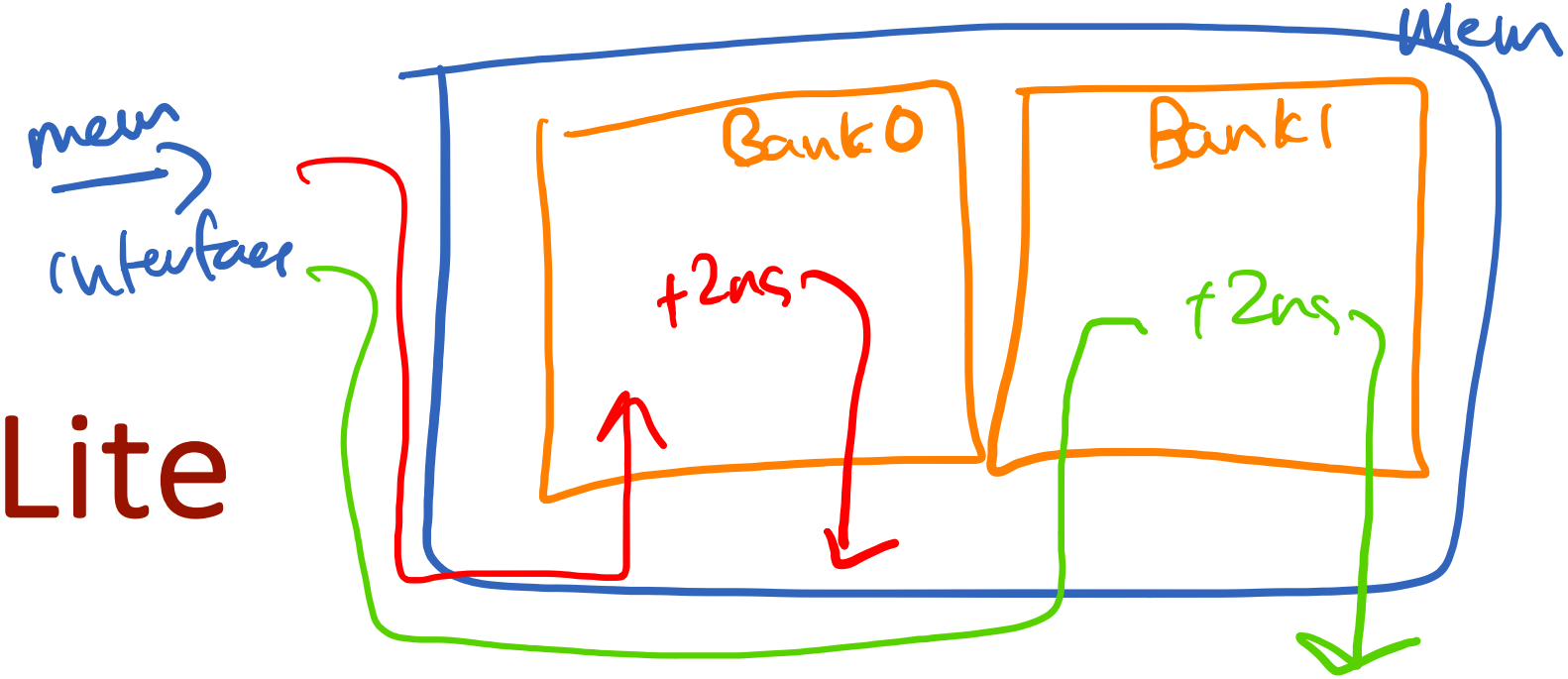


Figure 1-5 Overlapping read bursts

References

- https://www.youtube.com/watch?v=okiTzvihHRA
- https://web.eecs.umich.edu/~prabal/teaching/eecs373/
- https://en.wikipedia.org/wiki/File:Computer_system_bus.svg
- https://www.realdigital.org/doc/a9fee931f7a172423e1ba73f66ca4081

- AMBA® AXI™ and ACE™ ProtocolSpecification



08: AXI4 Lite

Engr 315: Hardware / Software Codesign

Andrew Lukefahr
Indiana University