08: AXI4 Lite

Test

→ Jupiter Notebook
↑
→ P3 issues

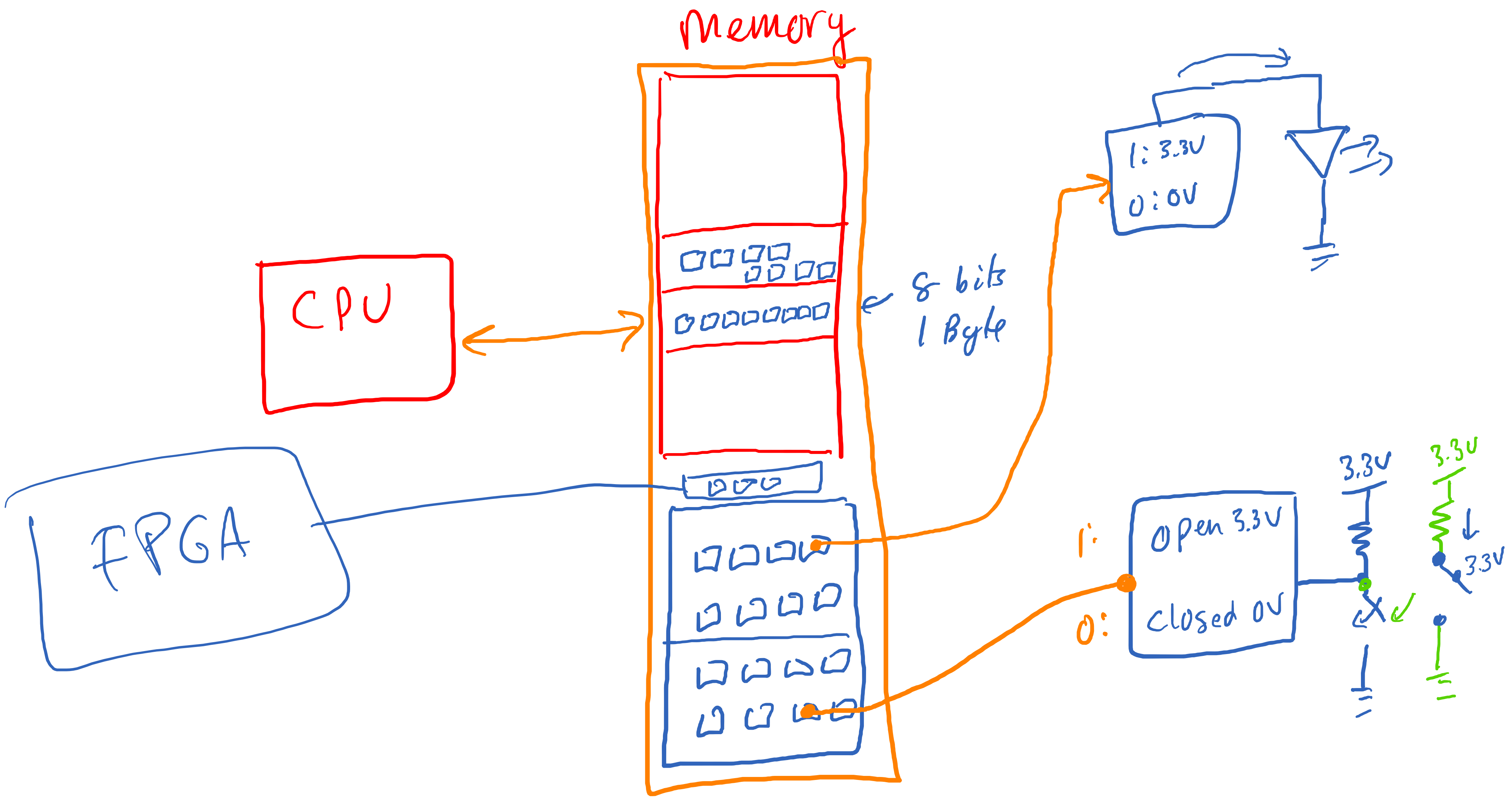
Engr 315: Hardware / Software Codesign

Andrew Lukefahr
Indiana University

Announcements

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

Review: Memory-Mapped I/O

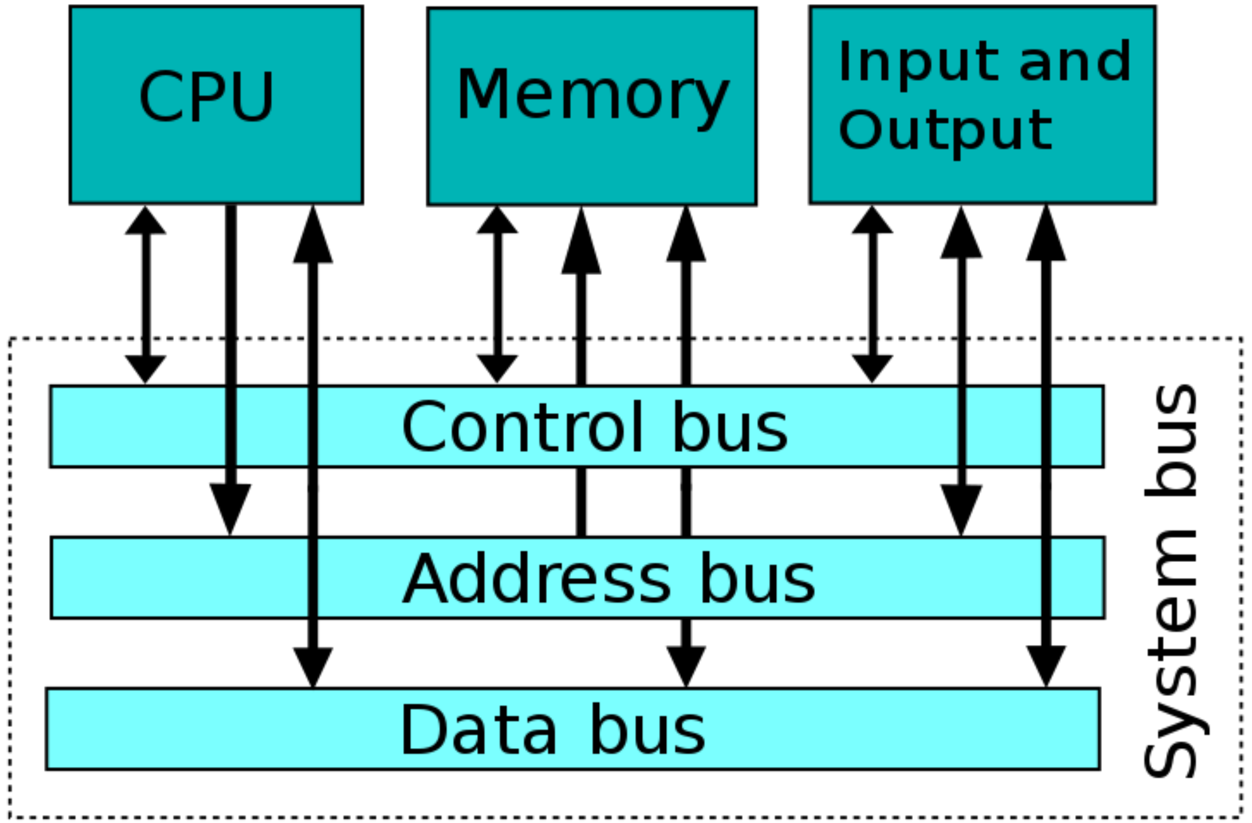


Use `volatile` for MMIO addresses!

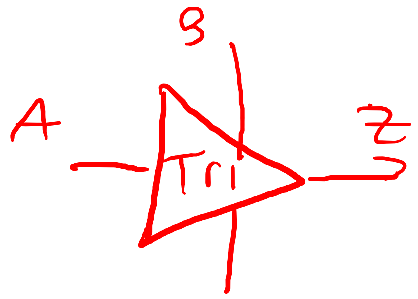
```
#define SW_ADDR 0xfffe
volatile uint32_t * SW_REG = (uint32_t * SW_ADDR);

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

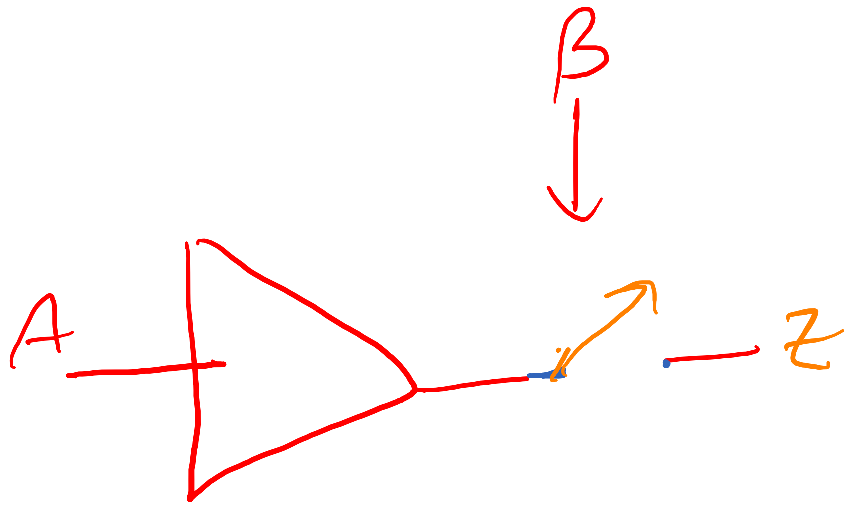
The System Bus



Tri-State Buffer



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

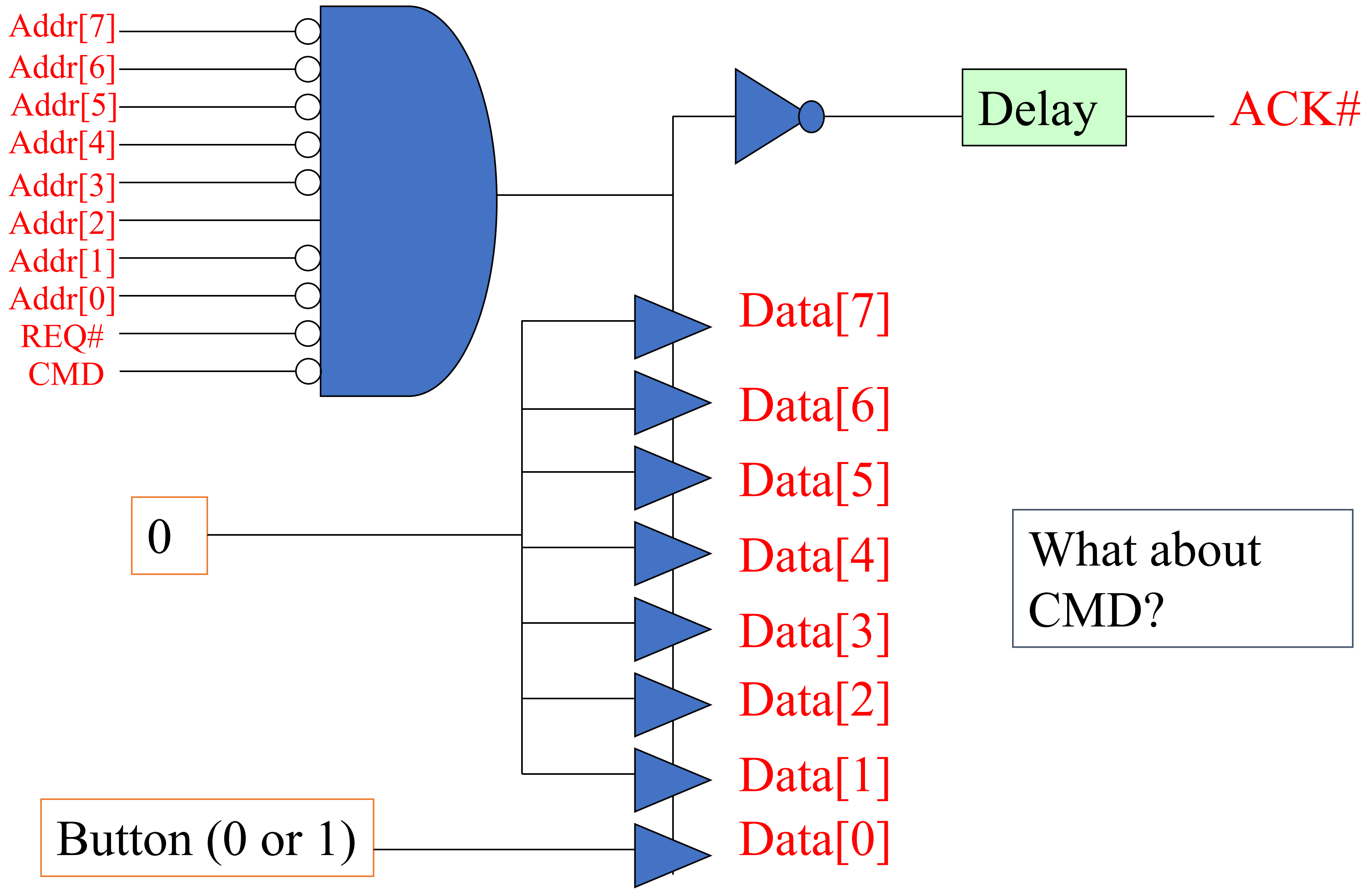


A	B	Z
0	0	High-Impedance (High-Z)
0	1	0
1	0	High Z
1	1	1

Tri-State: can be 0, 1, Z

The push-button

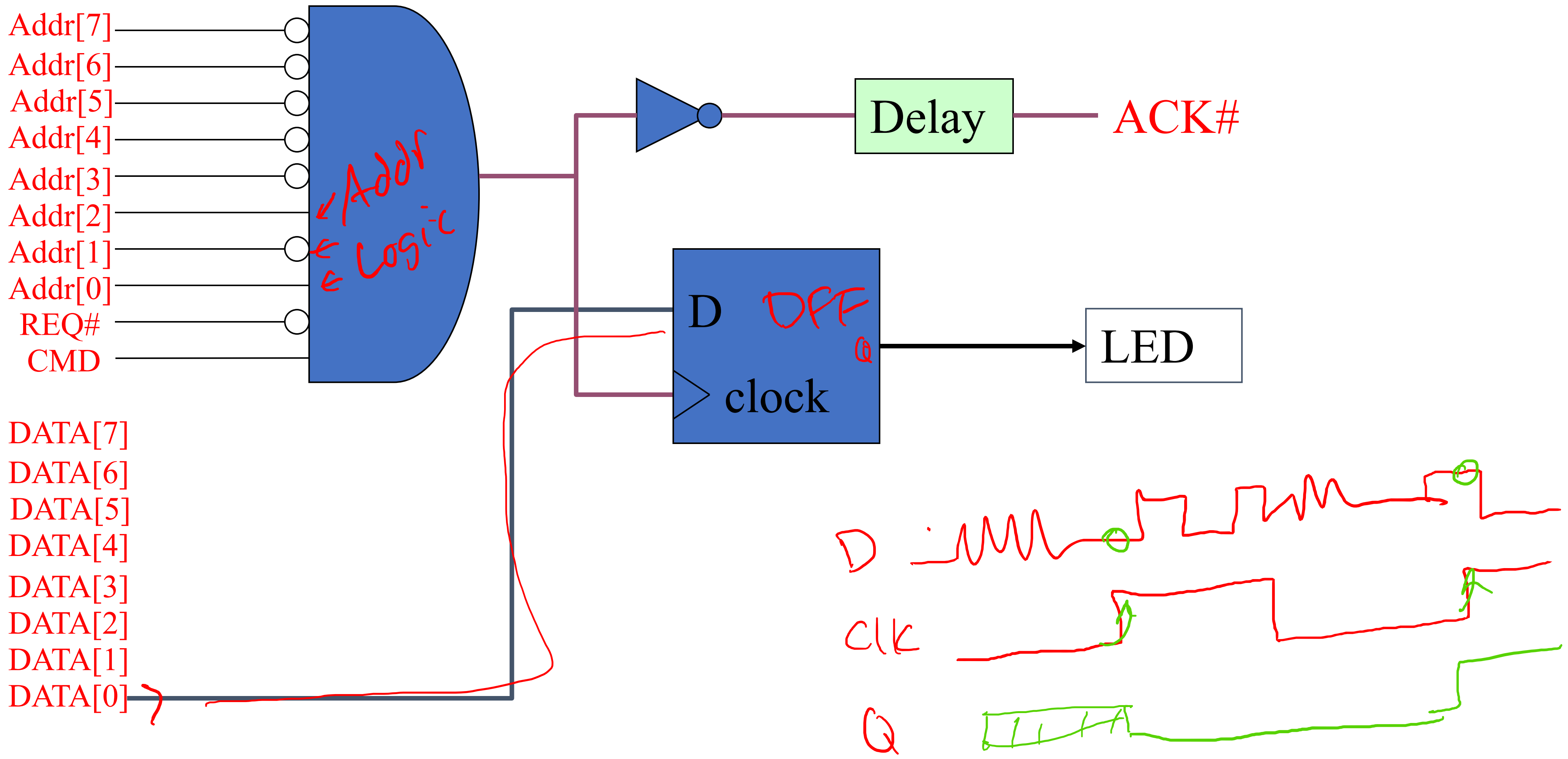
(if Addr=0x04 read 0 or 1 depending on button)



What about CMD?

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

r0 $\boxed{0x5}$

r8 $\boxed{0x4}$

r12 $\boxed{\text{switch}}$

```
        mov   r8, 0x4
        mov   r0, 0x5
loop:   ldr   r12, [r8]
        str   r12, [r0]
        b     loop
```

In ASM:

```
        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

```
#define  PB   0x4
#define  LED  0x5


int  main(){
      int  value;  (r12)
      for (;;){
            value = *((volatile uint32_t *)(PB));
            *(volatile uint32_t *)(LED)) = value;
      }
}
```

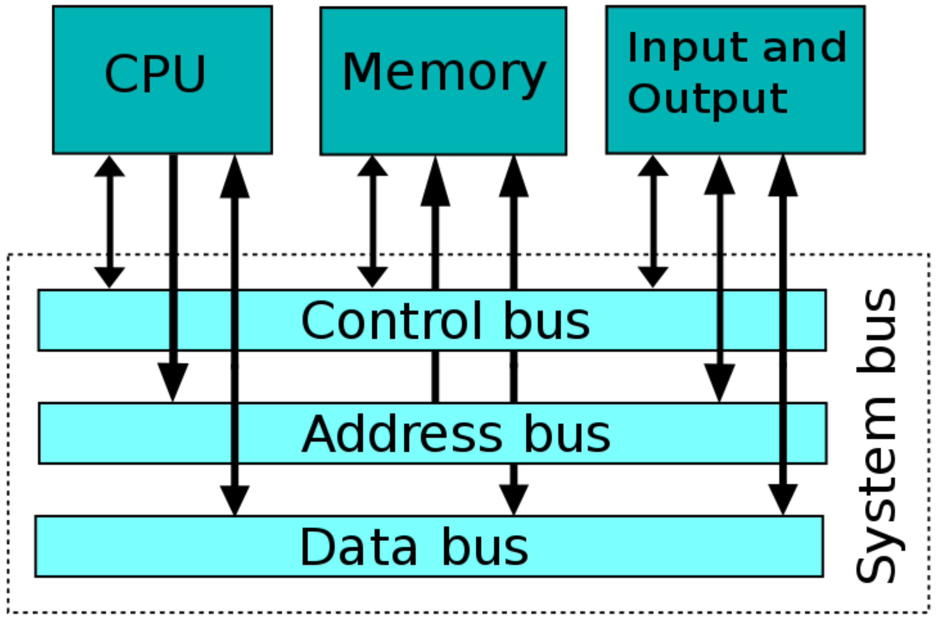
ARM AXI Bus

- “Advanced eXtensible Interface” Bus Version 4, “AXI4”

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

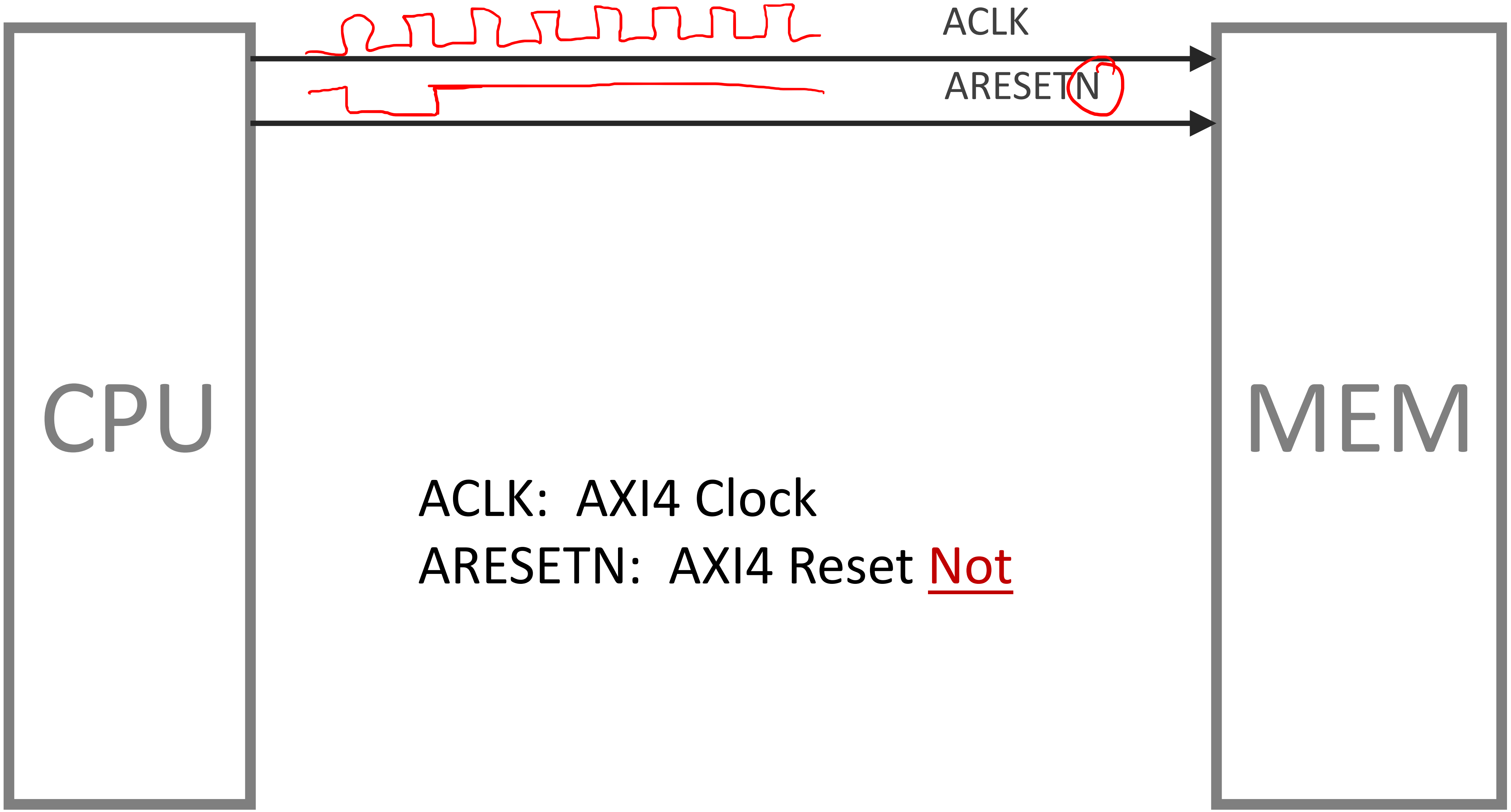
Why AXI4 Lite?



Xilinx AXI Reference Guide:

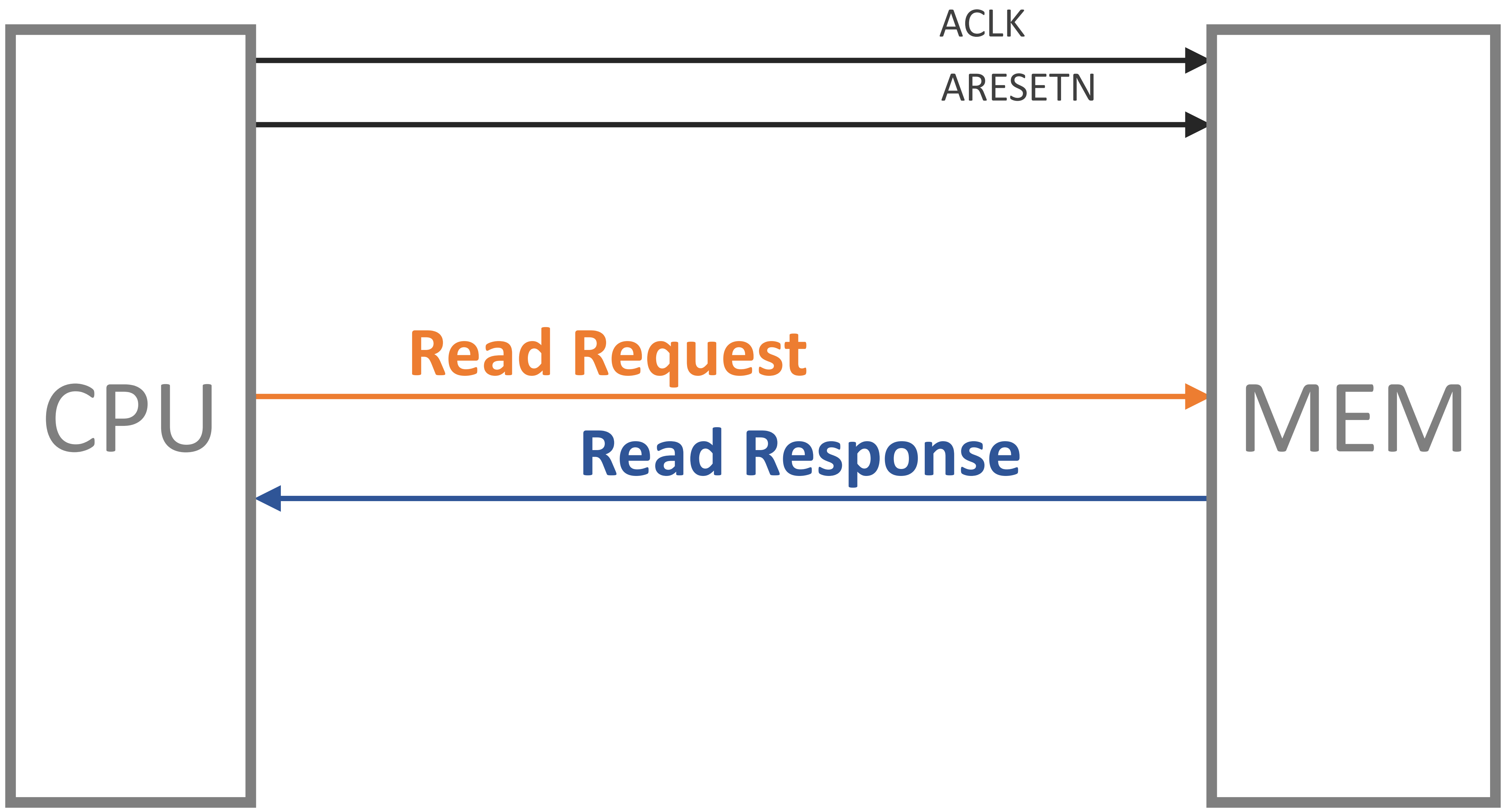
"AXI4-Lite is a light-weight, single transaction memory mapped interface. It has a small logic footprint and is a simple interface to work with both in design and usage. "

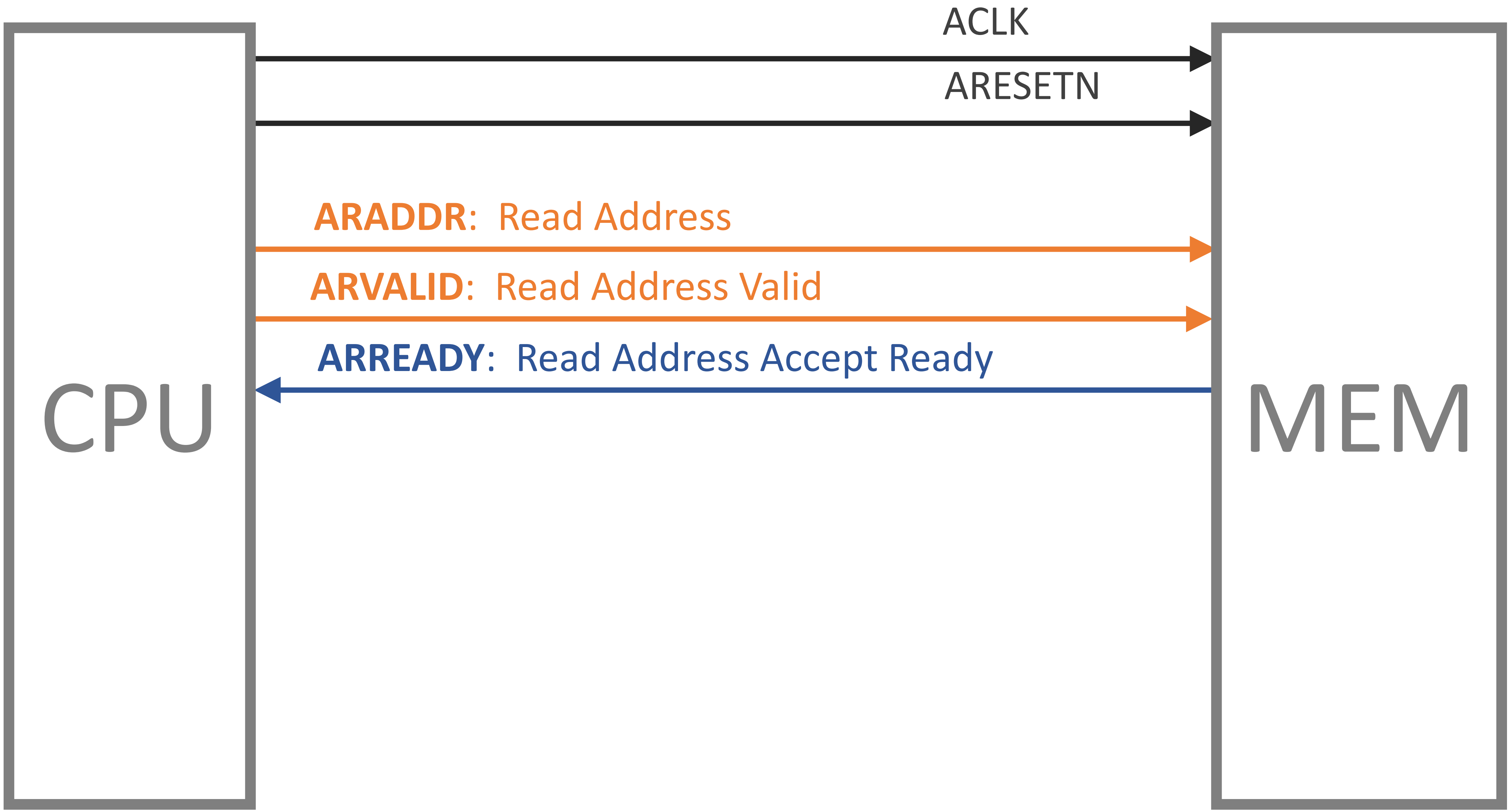
CPU
MEM



ACLK: AXI4 Clock

ARESETN: AXI4 Reset Not


ACLK
ARESETN
Read Request
Read Response
CPU
MEM


ACLK
ARESETN
ARADDR: Read Address
ARVALID: Read Address Valid
ARREADY: Read Address Accept Ready
CPU
MEM

AXI4 Handshaking

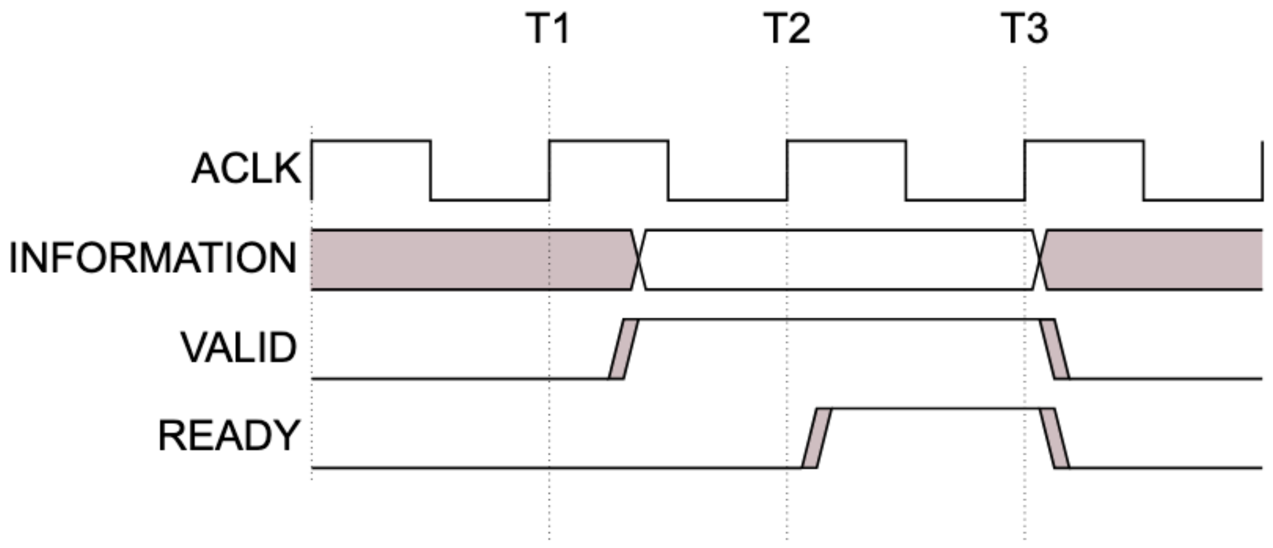
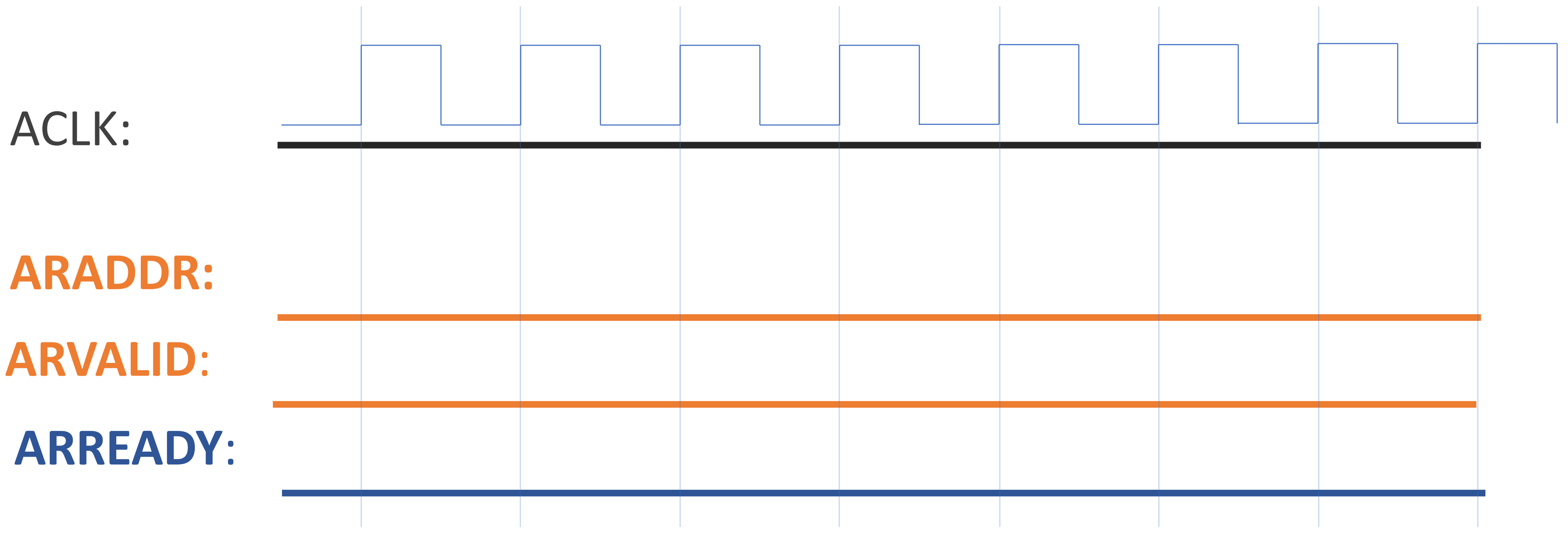
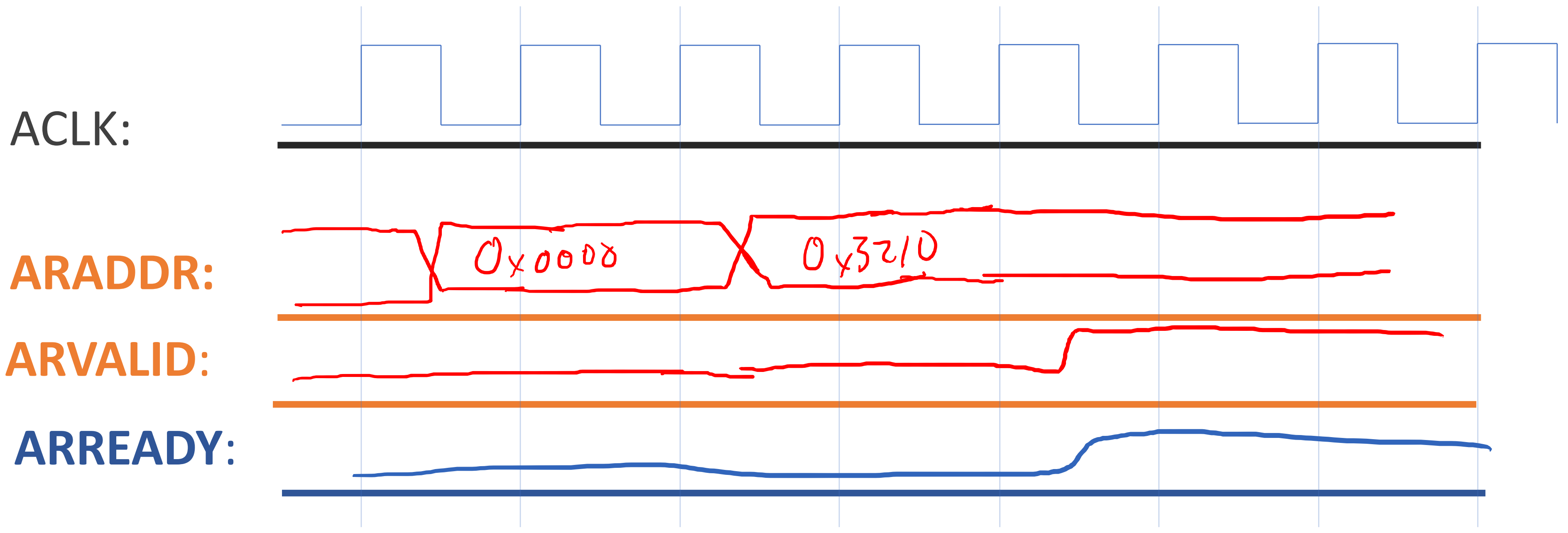


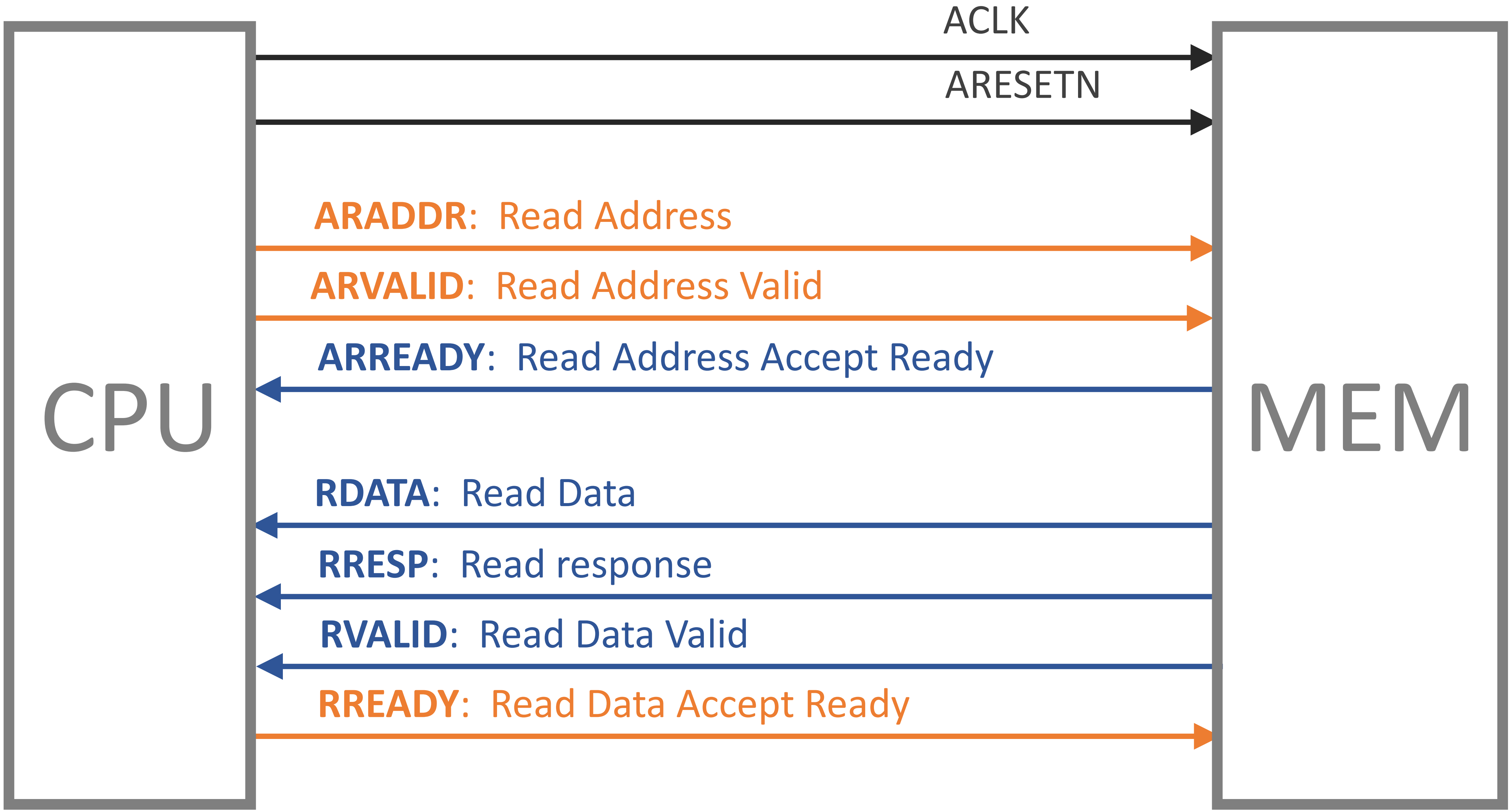
Figure A3-2 VALID before READY handshake

AXI4 Lite Read Transaction



What if?




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

What is RRESP?

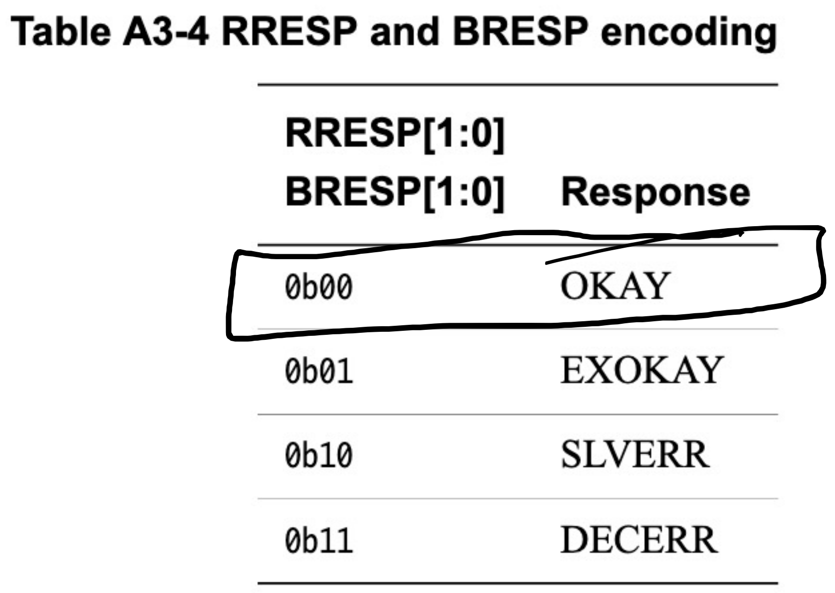


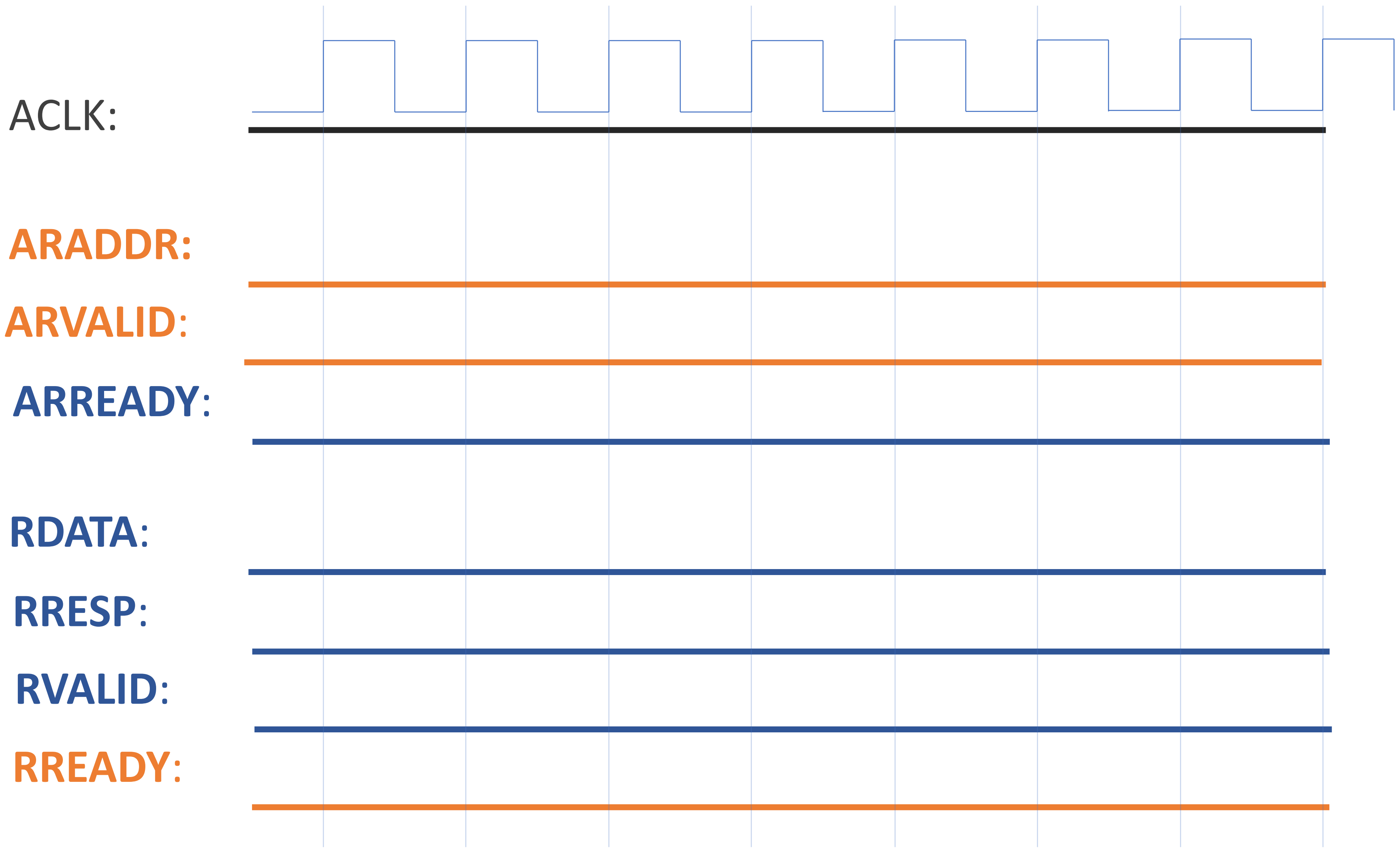
Table A3-4 RRESP and BRESP encoding

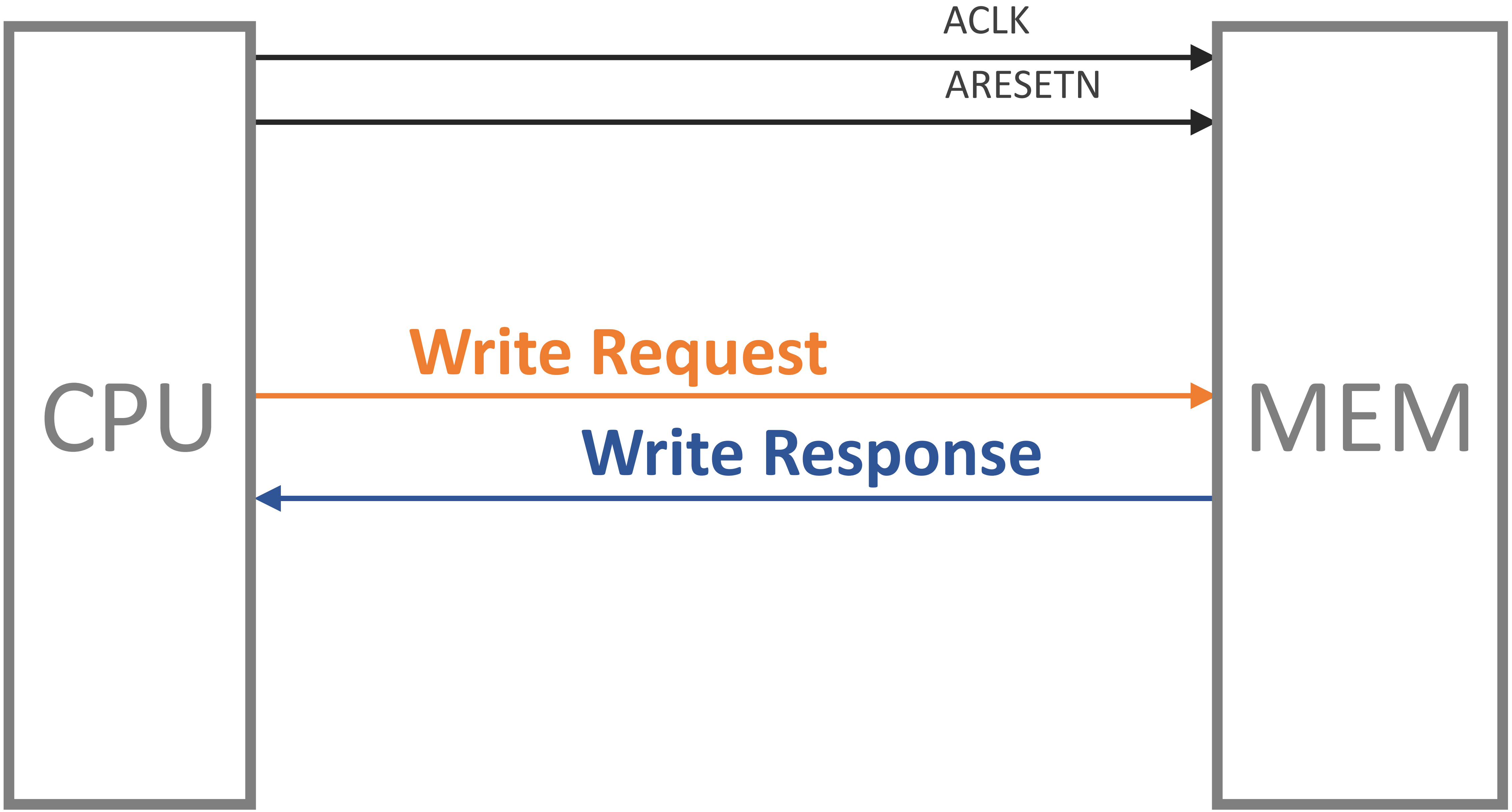
RRESP[1:0] BRESP[1:0]	Response
0b00	OKAY
0b01	EXOKAY
0b10	SLVERR
0b11	DECERR

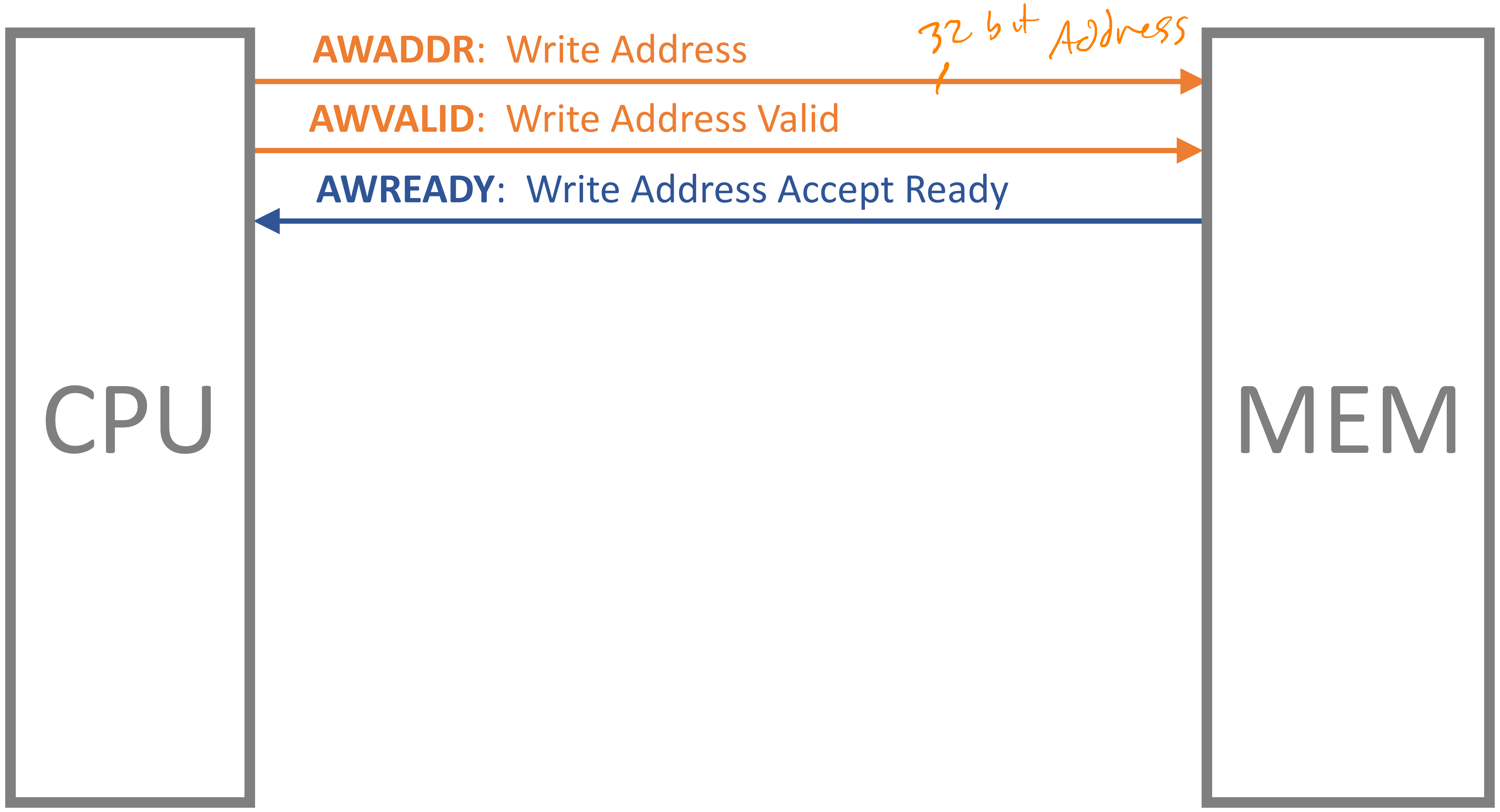
- Mostly used to send error codes back to CPU
- We'll always just use 0b00

Load 0x1234, response: 0xabcd

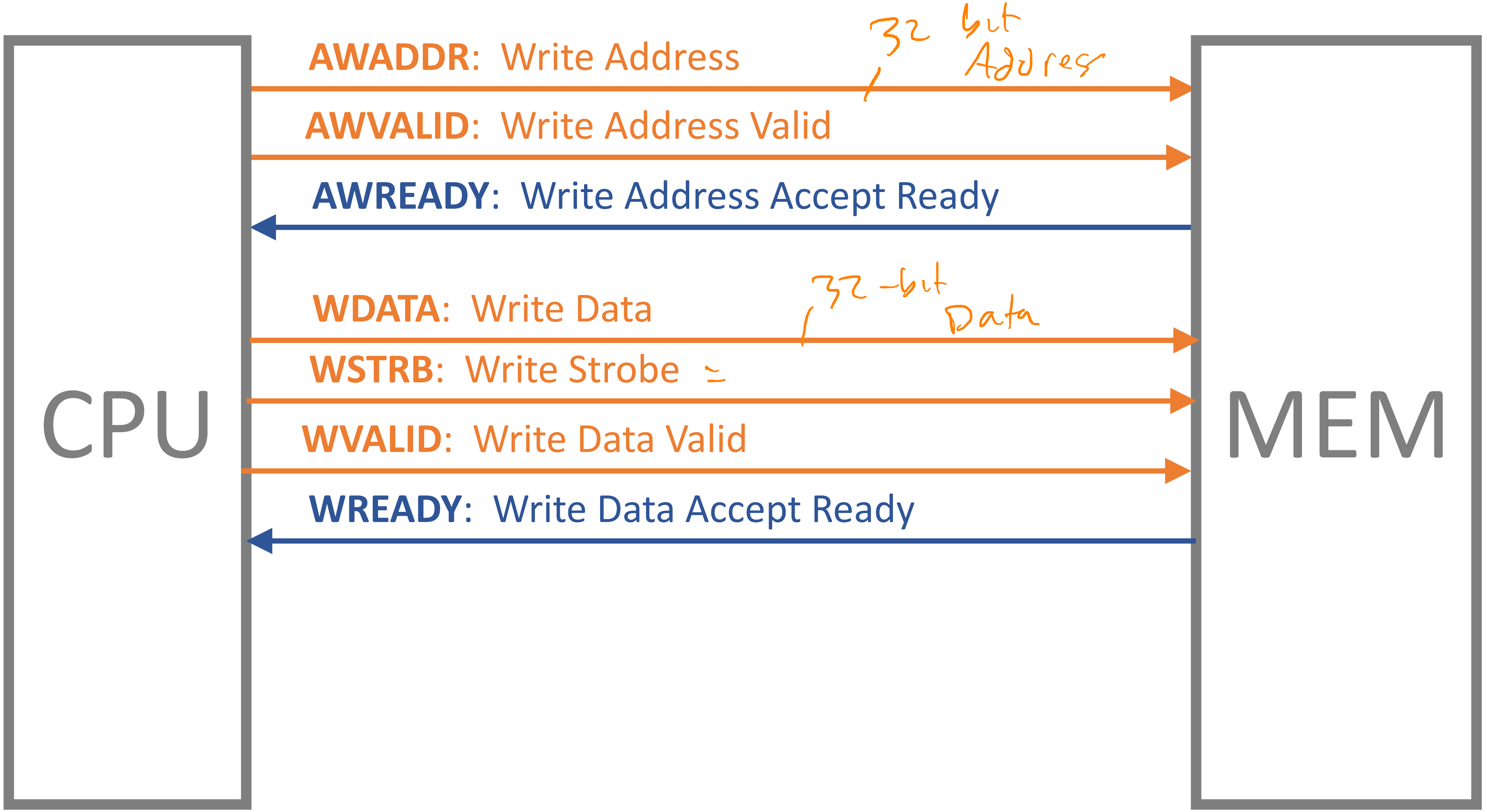
assume
ARESETN=1




ACLK
ARESETN
CPU
Write Request
Write Response
MEM


AWADDR: Write Address
32 bit Address
AWVALID: Write Address Valid
AWREADY: Write Address Accept Ready
CPU
MEM

ACLK and ARESETN not shown



ACLK and ARESETN not shown

Q: How do you send a 1-byte (8-bit) value on a 32-bit bus?

- A: WSTB: Write Strobe

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?

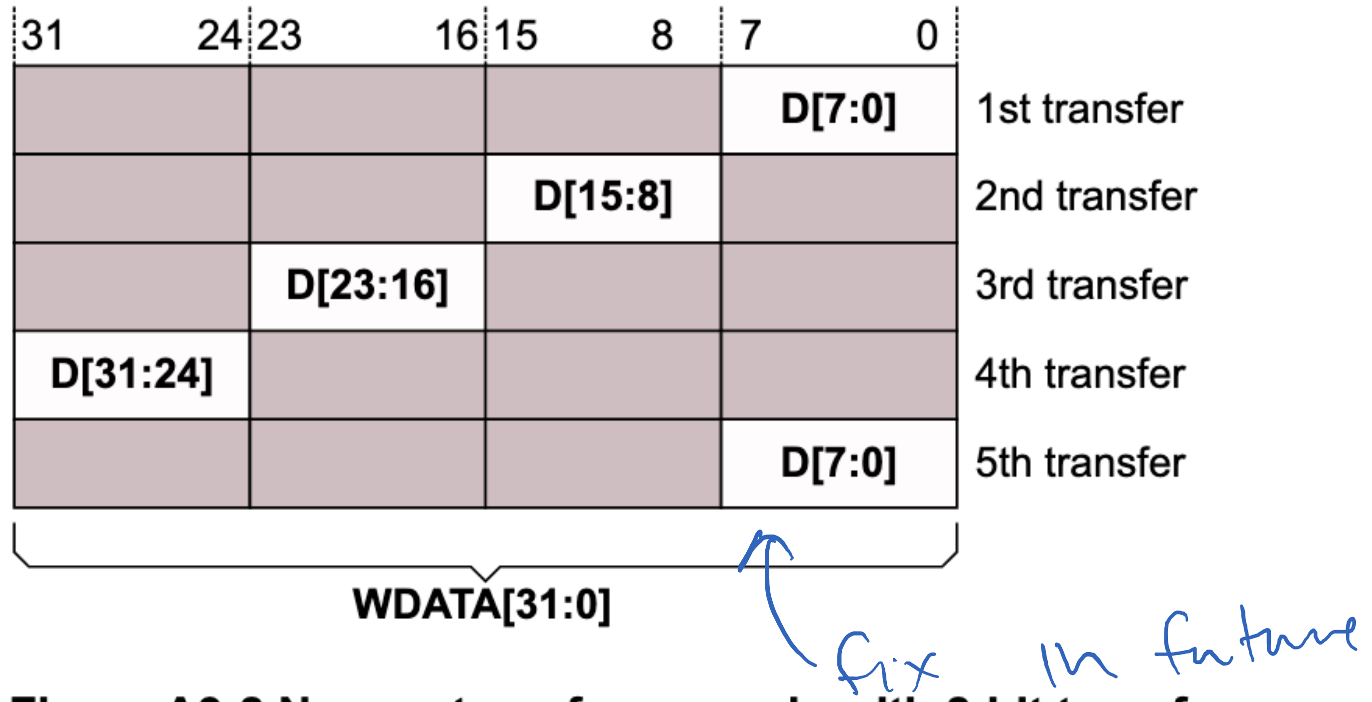
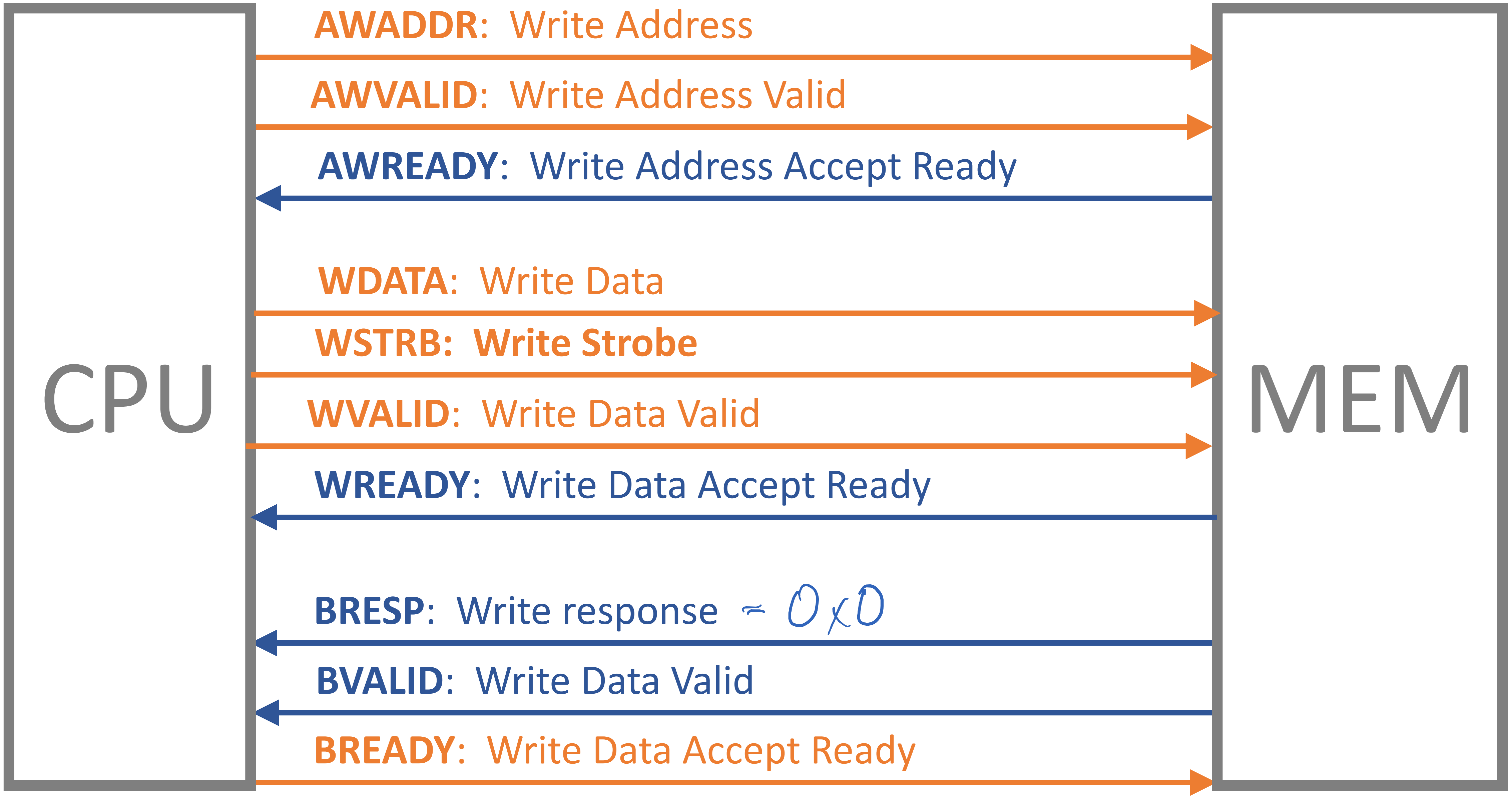


Figure A3-8 Narrow transfer example with 8-bit transfers


CPU
MEM
AWADDR: Write Address
AWVALID: Write Address Valid
AWREADY: Write Address Accept Ready
WDATA: Write Data
WSTRB: Write Strobe
WVALID: Write Data Valid
WREADY: Write Data Accept Ready
BRESP: Write response = 0x0
BVALID: Write Data Valid
BREADY: Write Data Accept Ready

ACLK and ARESETN not shown

BRESP is just like RRESP

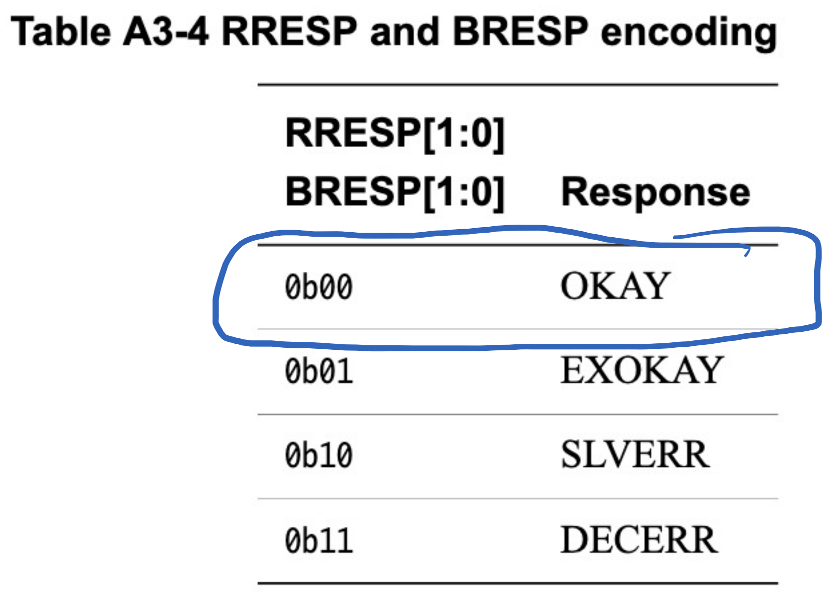
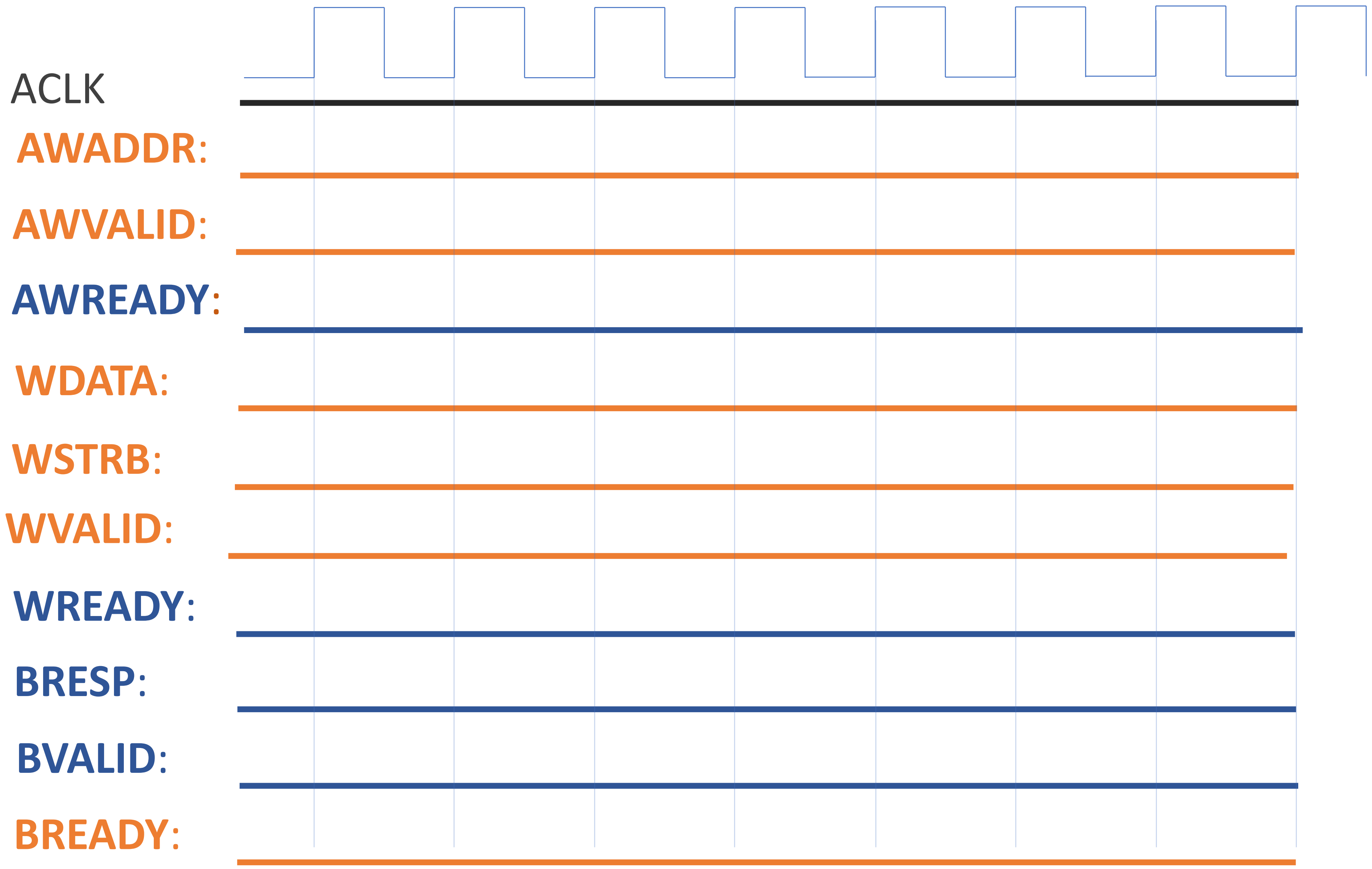


Table A3-4 RRESP and BRESP encoding

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

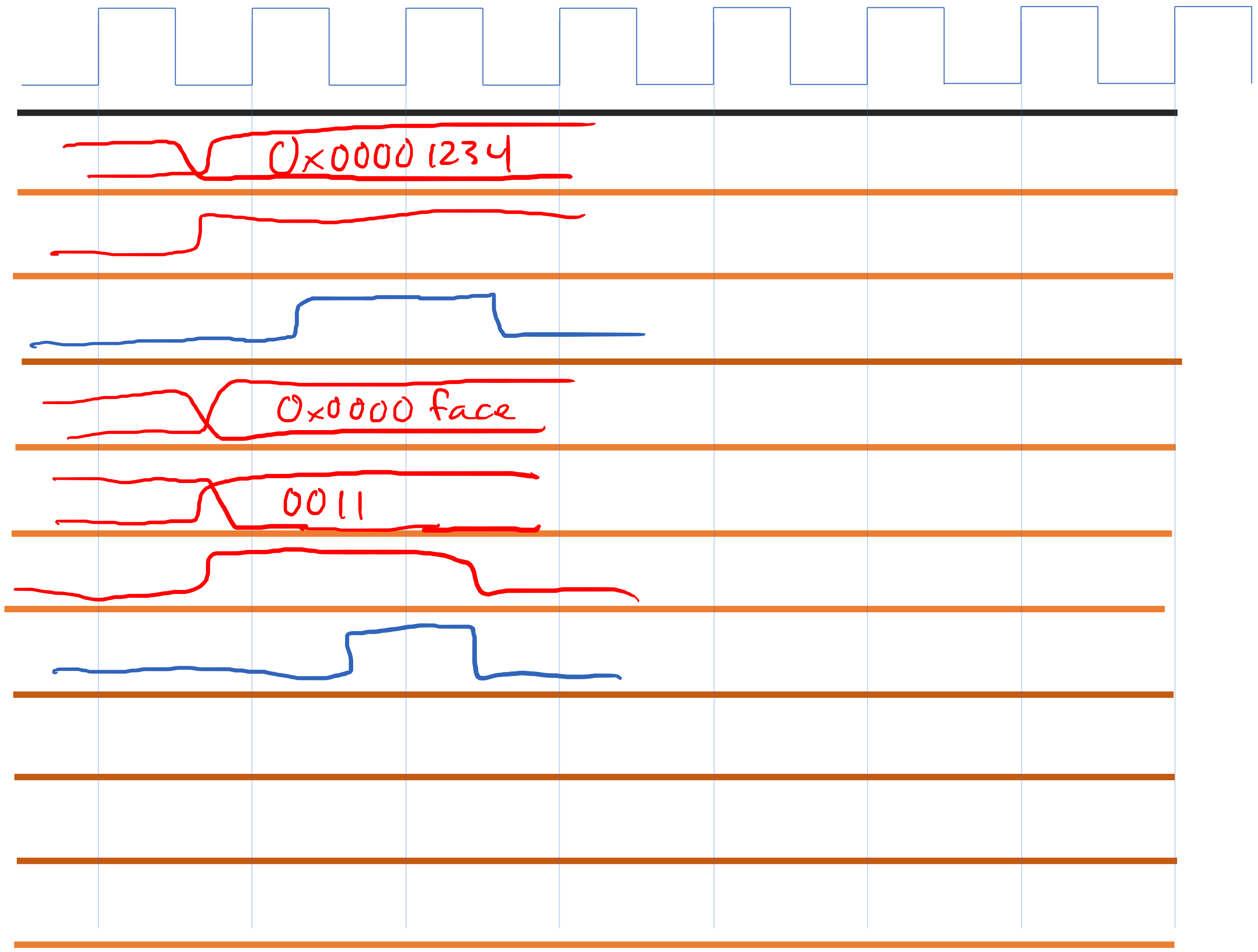
- Mostly used to send error codes back to CPU
- We'll always just use 0b00

Writing 0xdeadbeef to 0x1234



16-bit

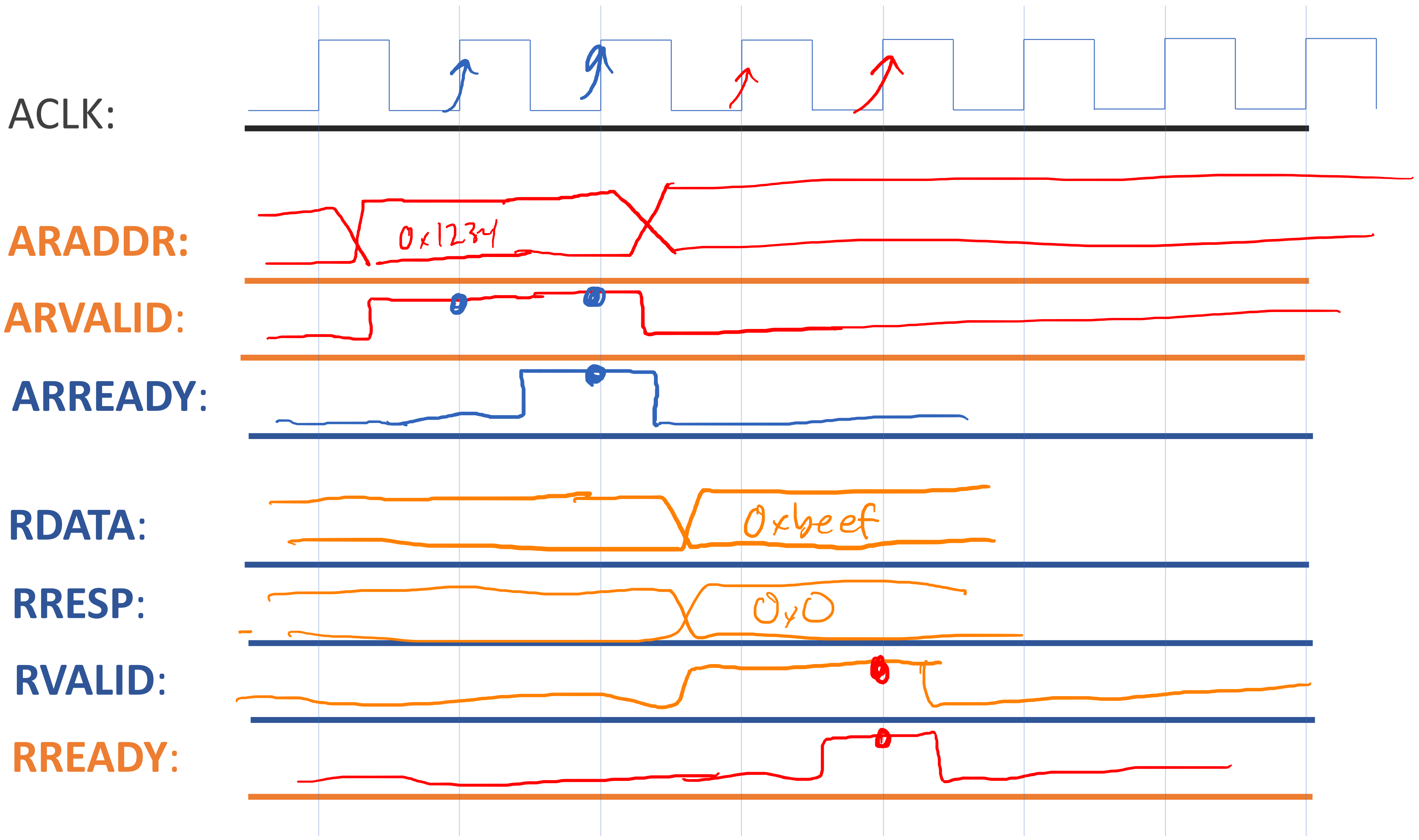
Writing 0xface to 0x1234



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

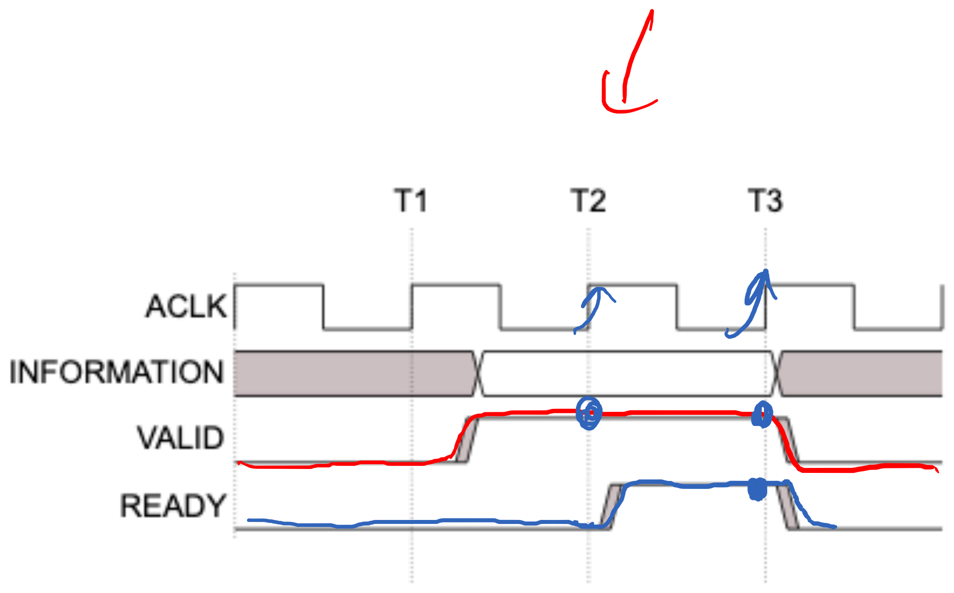


Figure A3-2 VALID before READY handshake

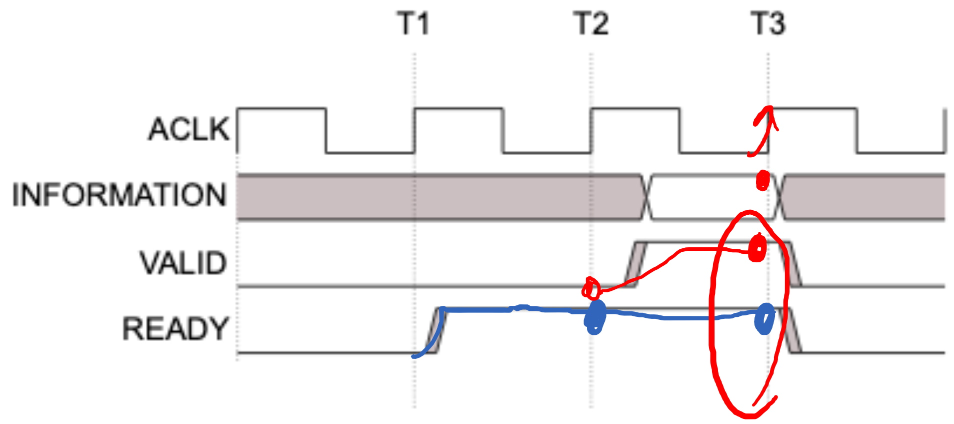
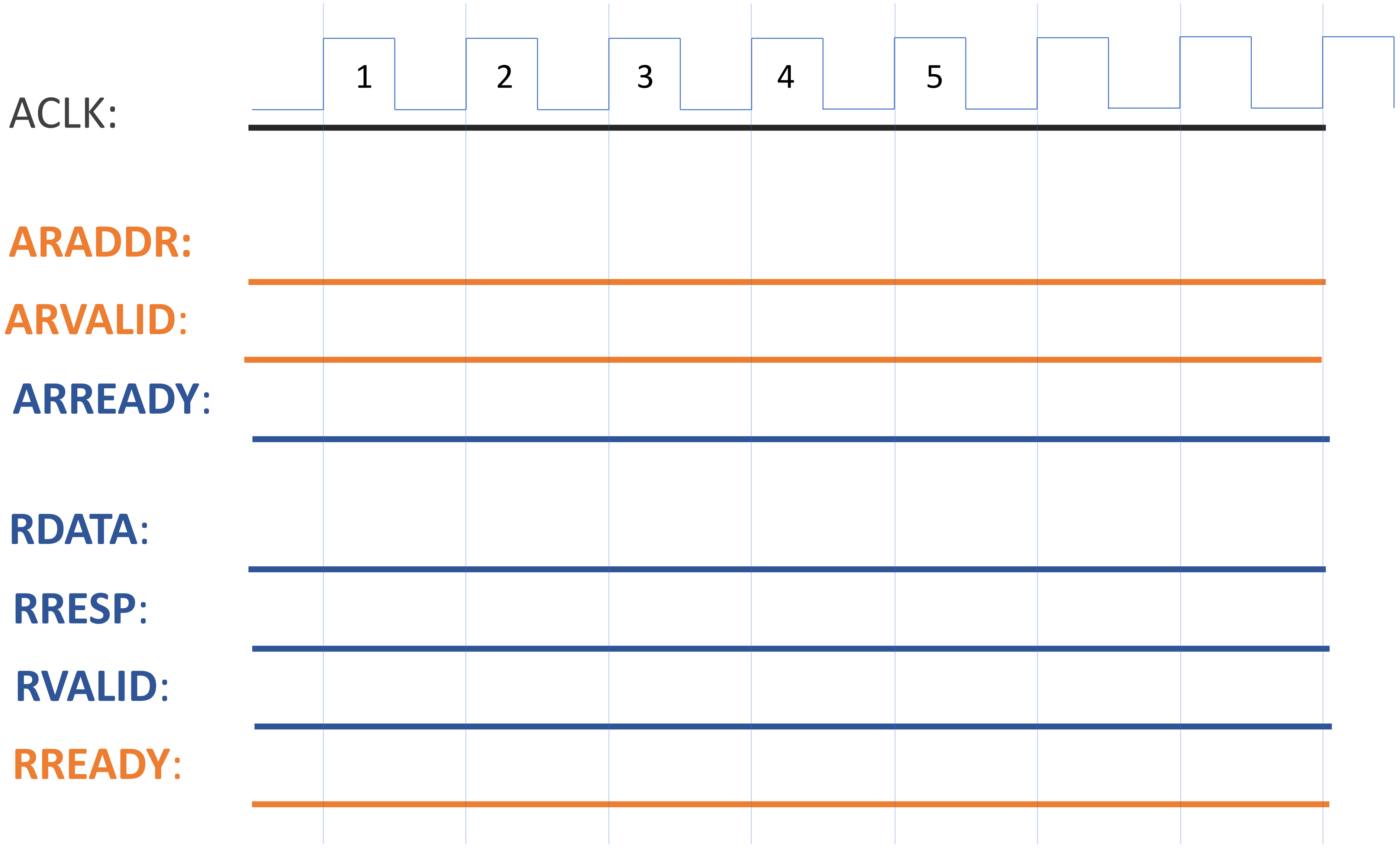


Figure A3-3 READY before VALID handshake

- 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

Next Time

- High-Performance Busses

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