Bridge of Life Education

SOC Design
Interconnect - AXI

Jiin Lai

https://developer.arm.com/-/media/Arm%20Developer%20Community/PDF/Learn%20the%20Architecture/102202_0100_01_Introduction_to_AMBA_AXI.pdf?revision=369ad681-f926-47b0-81be-42813d39e132

A Brief History about AXI

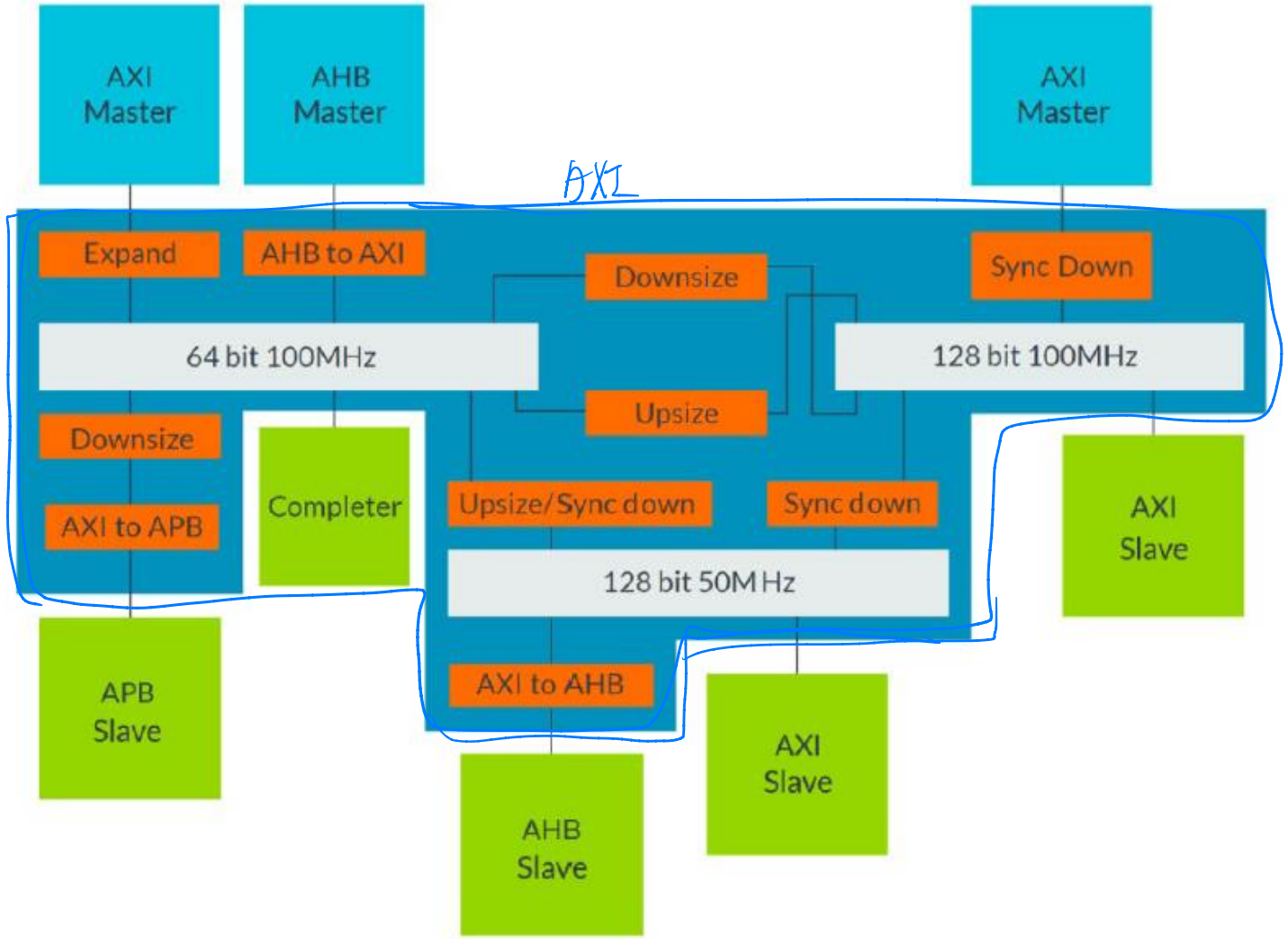
- Arm Advanced Microcontroller Bus Architecture (AMBA)
- Advanced eXtensible Interface (AXI), ACE (Advanced Topic)
- On-chip interconnect

ARM - IP.

Interconnect Converts

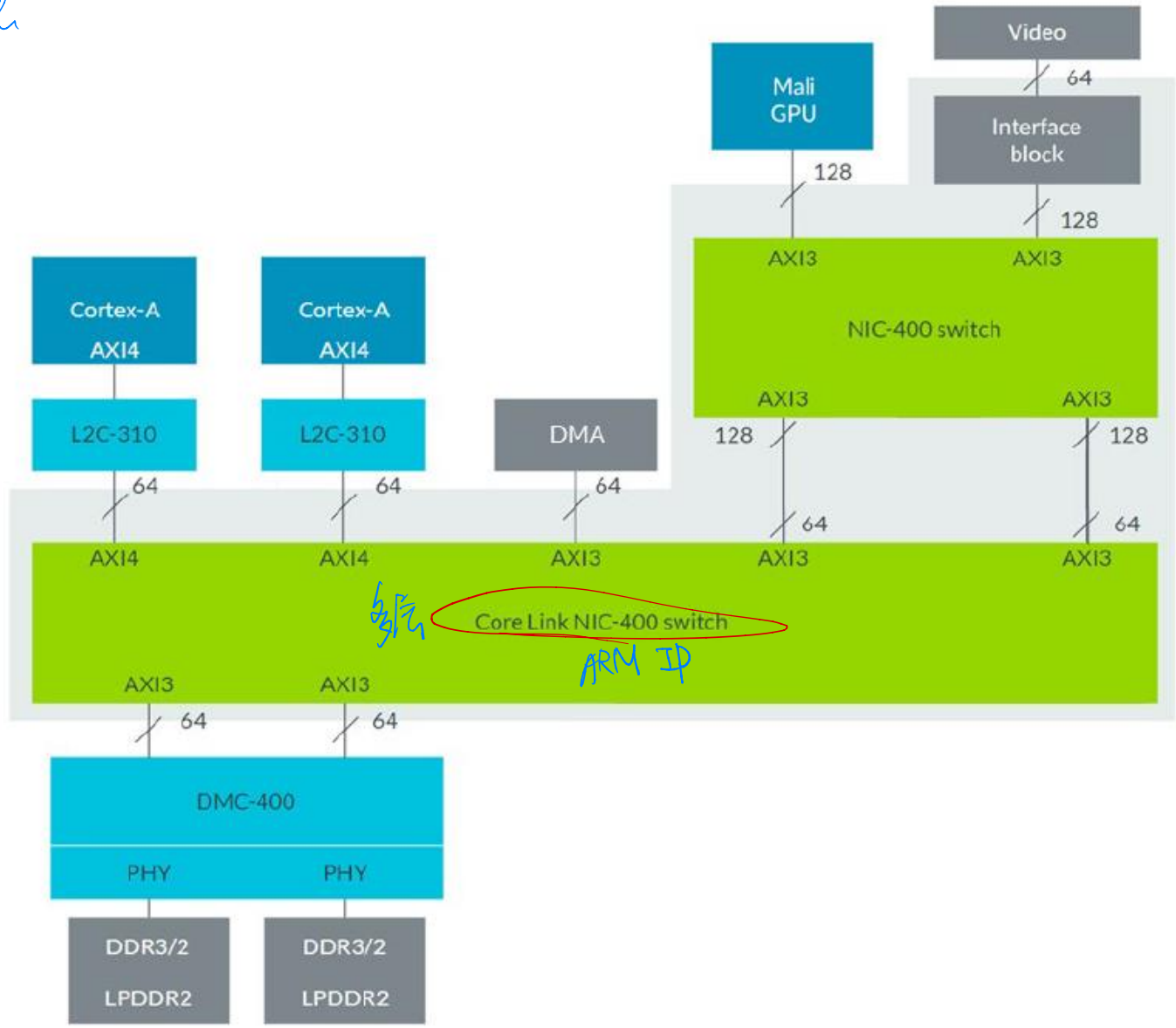
- Protocol
- Data Width
- Frequency

high performance 128b
other 32-b



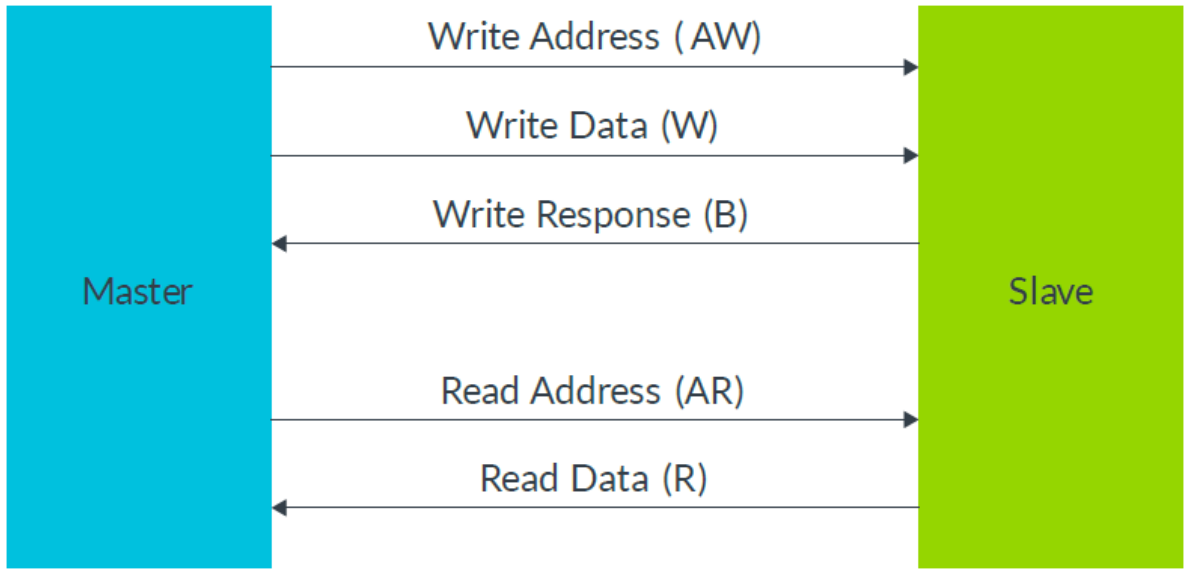
IP
ARM : Design environment & port... Tool

Example of an SoC



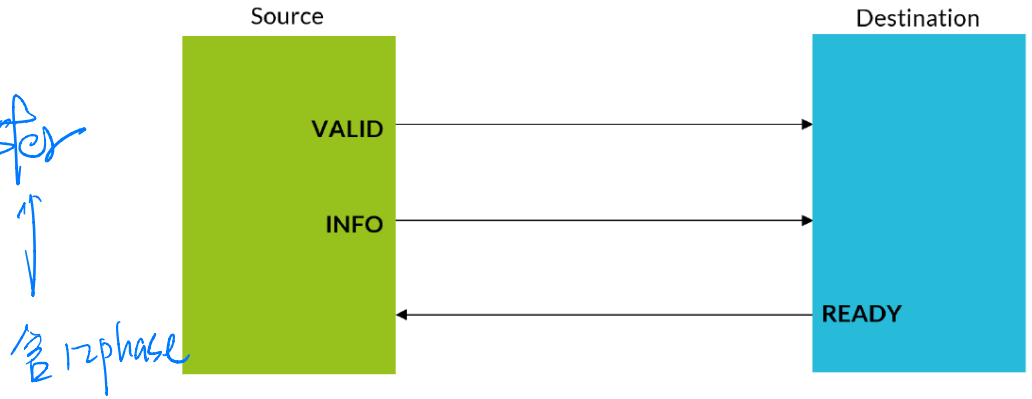
AXI Channels – AW, W, B, AR, R

- Independent read and write channels
- Multiple outstanding addresses
- No strict timing relationship between address and data operations
- Support for unaligned data transfers
- **Out of-order transaction completion**
- Burst transactions based on start address

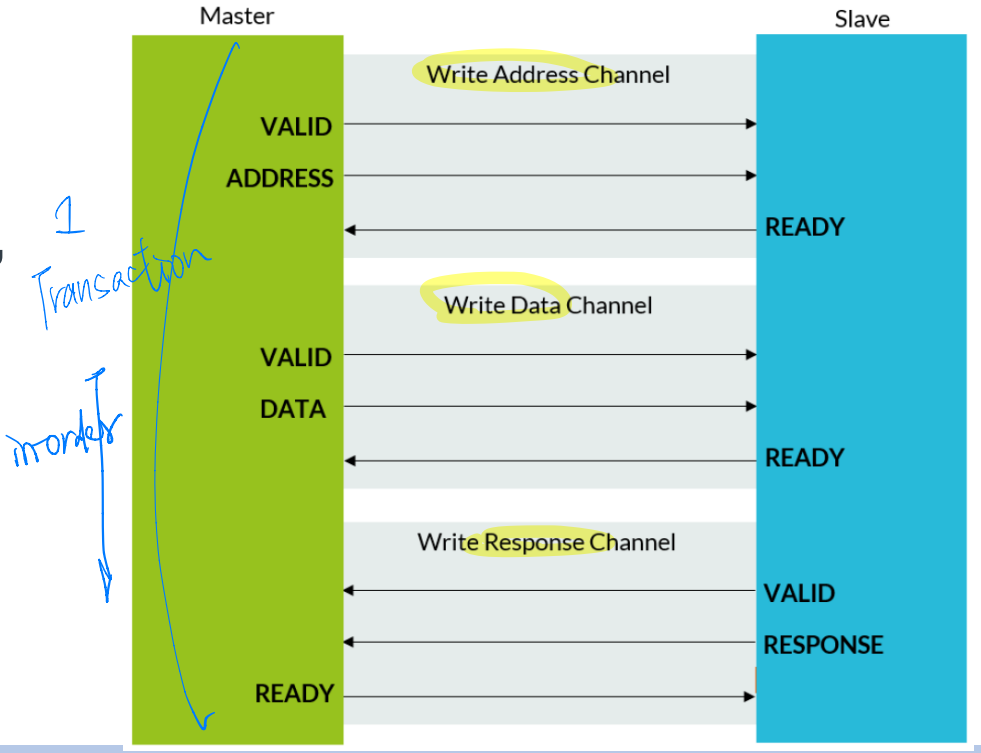


Transfer & Transaction

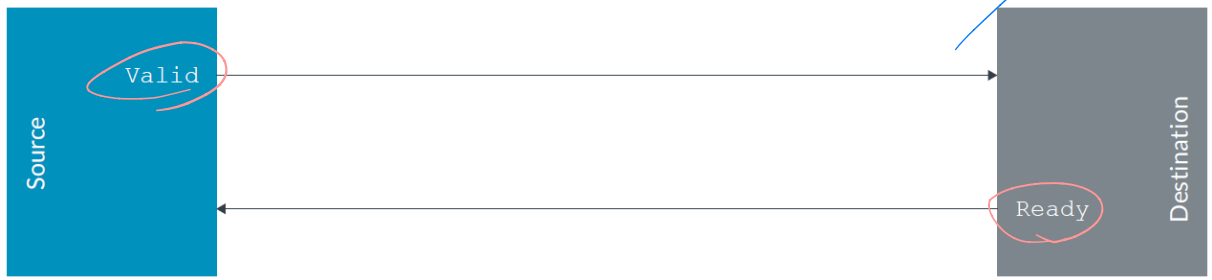
A transfer is a single exchange of information, with one VALID and READY handshake. (sample both active at clock)

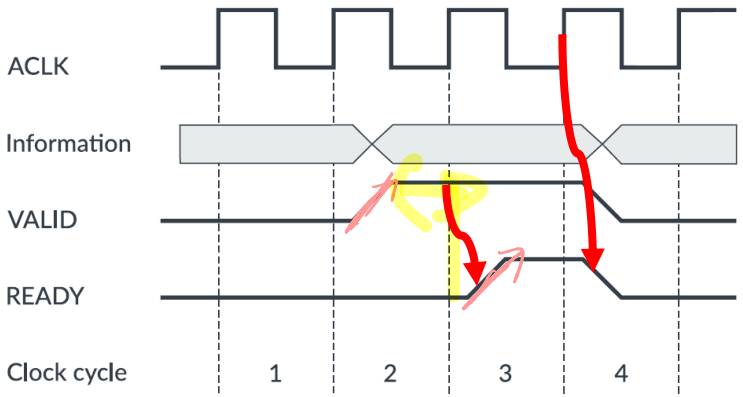


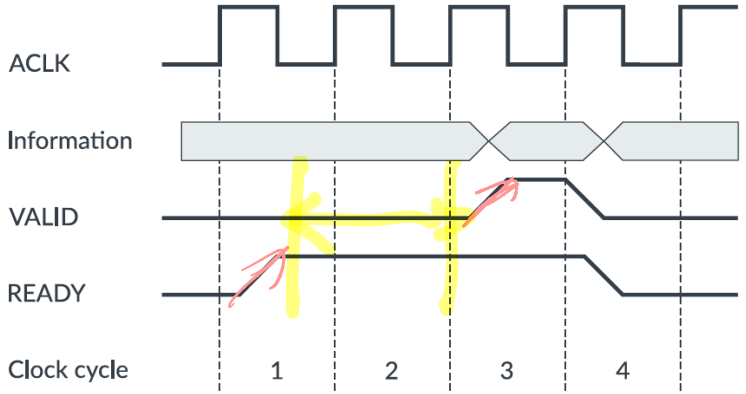
A transaction is an entire burst of transfers, containing an address transfer, one or more data transfers, and/or response transfer (for write)

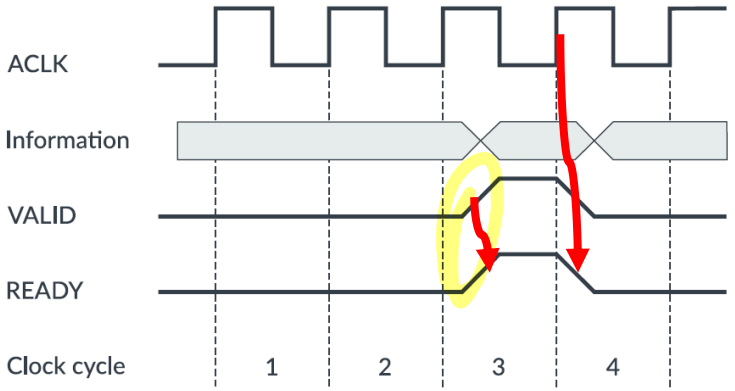


Channel Handshake : Transfer & Transaction



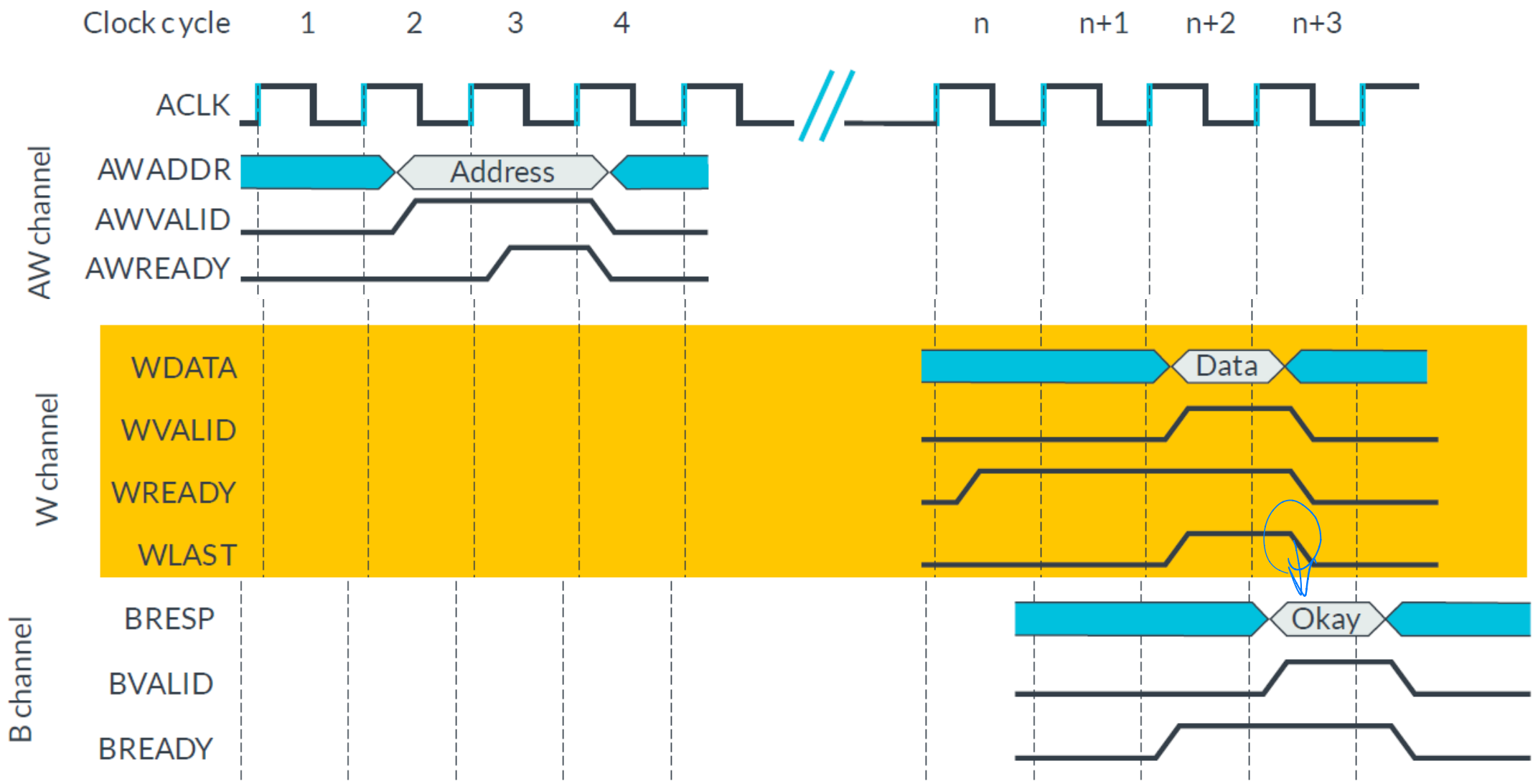




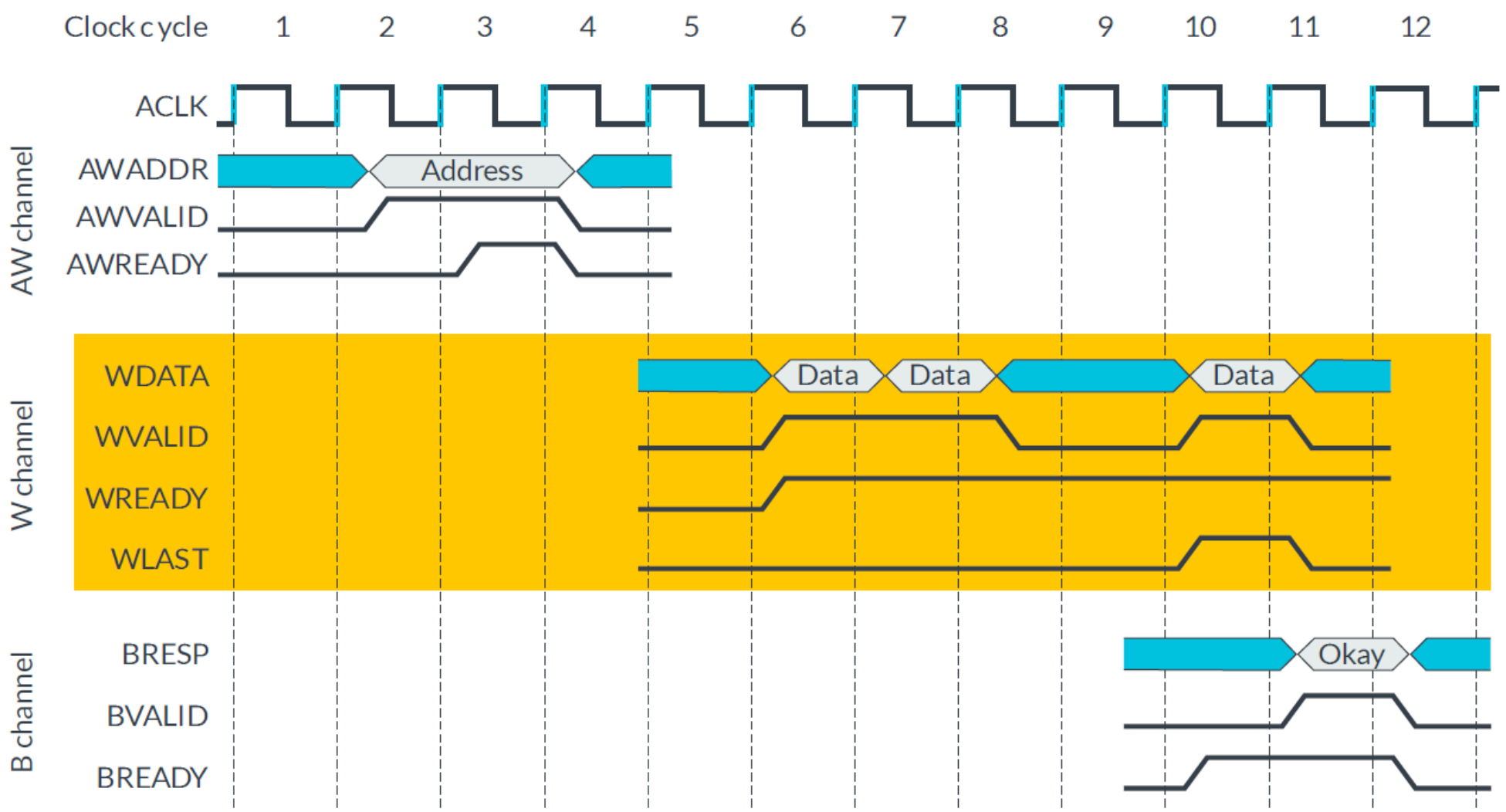


Which is possible for back-to-back zero-wait-state transfer, i.e. 1-1-1-.....?
What is the design scheme ?

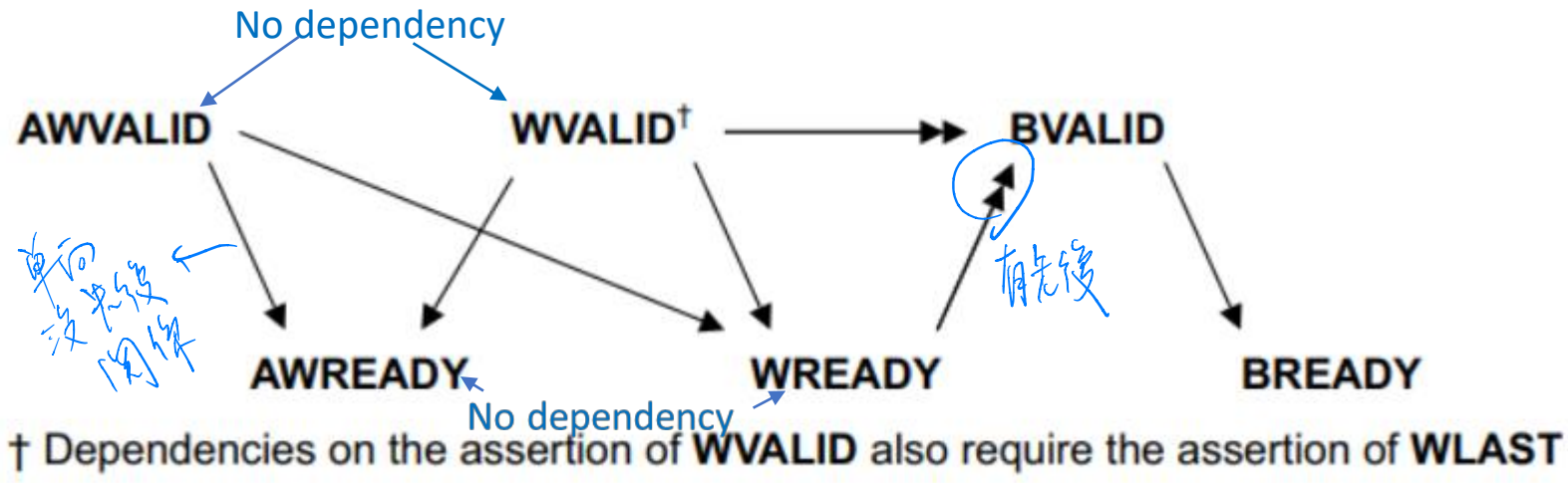
Write Transaction: Single Data



Write Transaction: Multiple Data



Write Transaction Handshake Dependencies



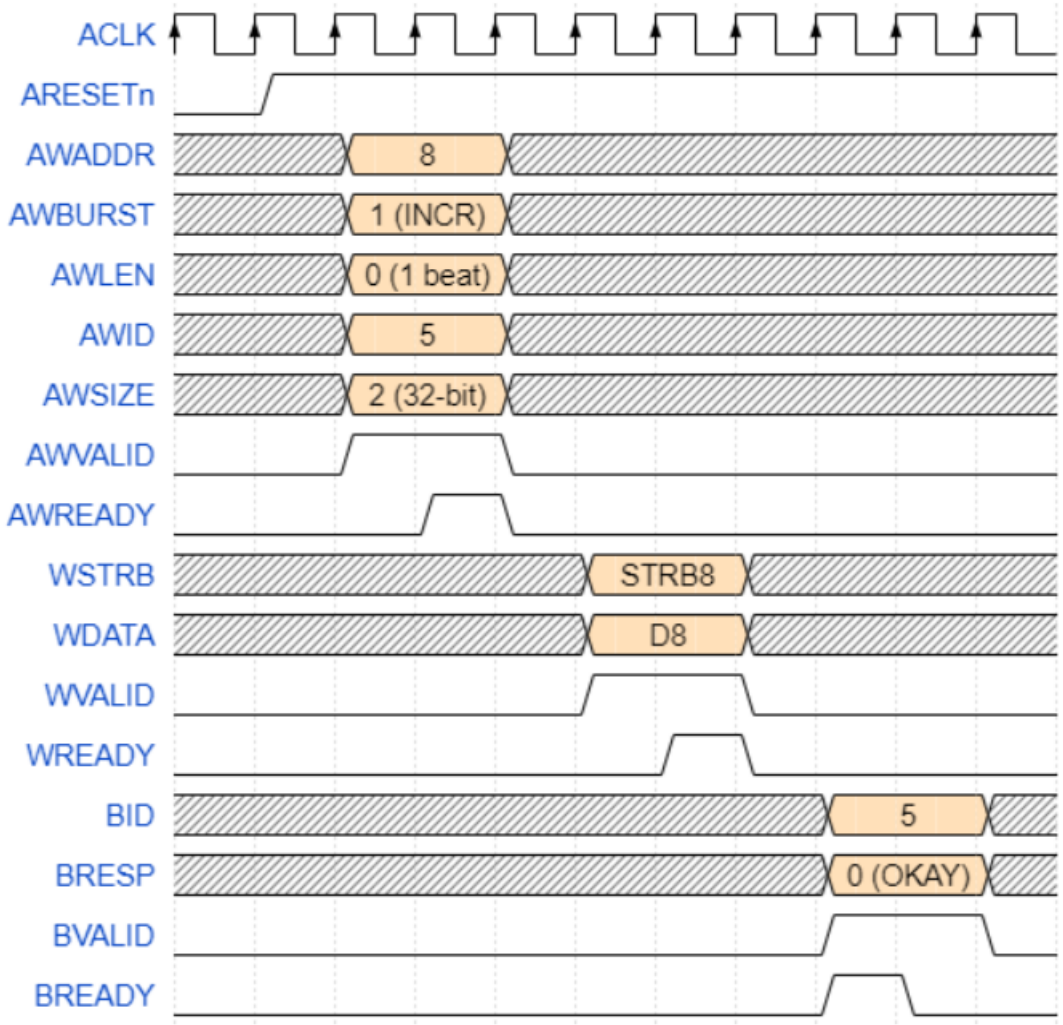
1. One of the common mistakes is to assume that the master is allowed to wait for AW to be accepted before putting W channel data. This assumption is wrong and may create a deadlock. For optimal performance, it is recommended to put AW channel data and W channel data as early as possible.

2. After AW and W channel the last beat is accepted, the slave is allowed to assert the BVALID signaling response on the B channel. B channel contains a BID signal which should match the same value as in the AW channel request. BRESP matches its encoding with RRESP.

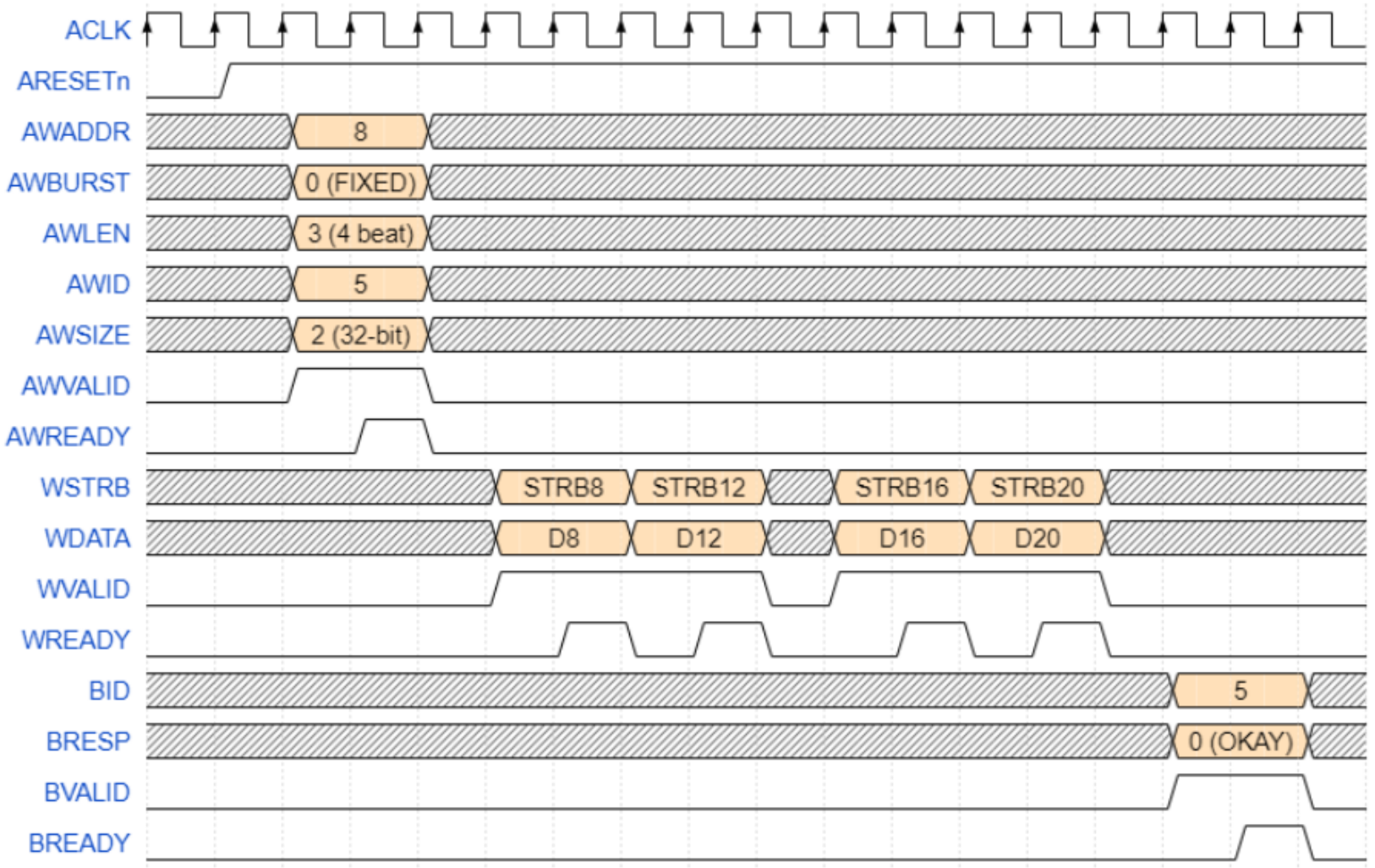
3. **Masters have to put W channel data in the same order as AW channel's data**. Mixing orders between multiple transactions on AW and W channels is not allowed.

1. **Single-headed arrows point to signals that can be asserted before or after the signal at the start of the arrow.**
2. **Double-headed arrows point to signals that must be asserted only after assertion of the signal at the start of the arrow.**

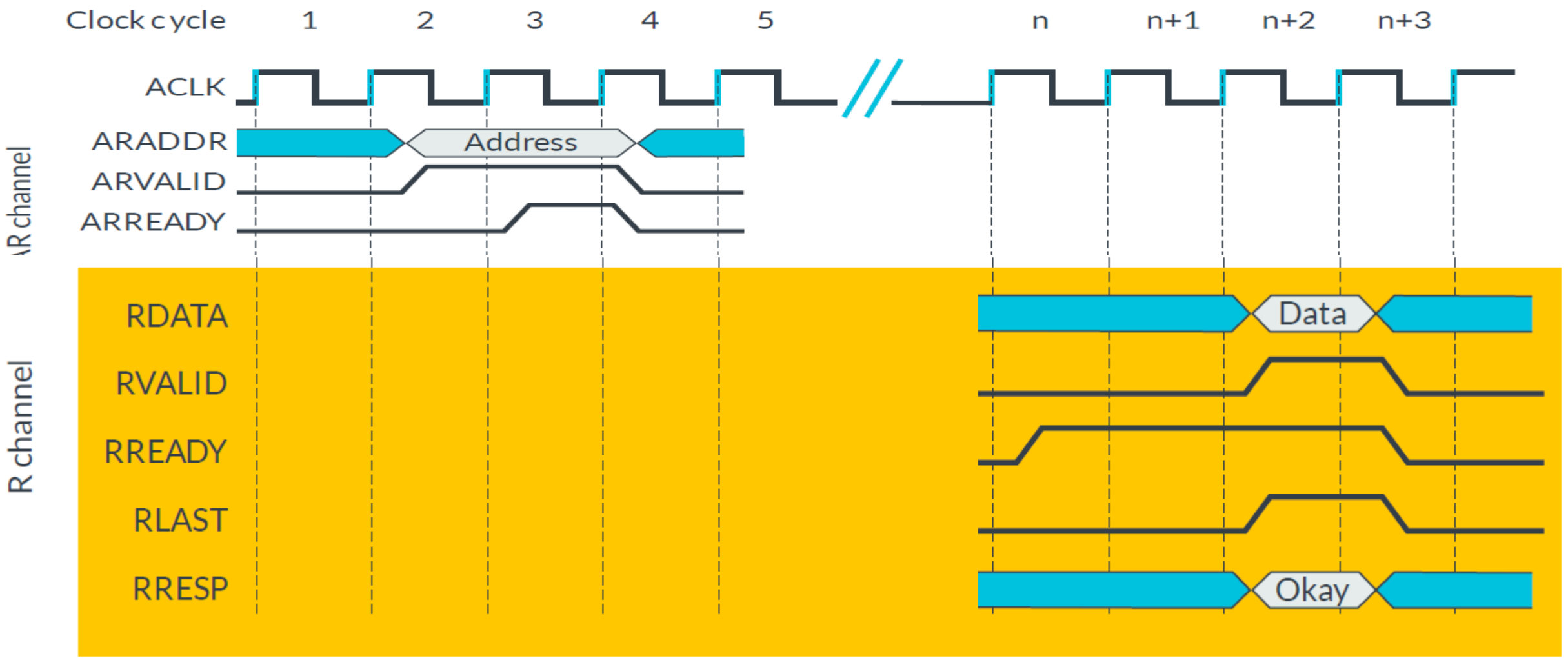
Write Transaction - One beat (INCR)



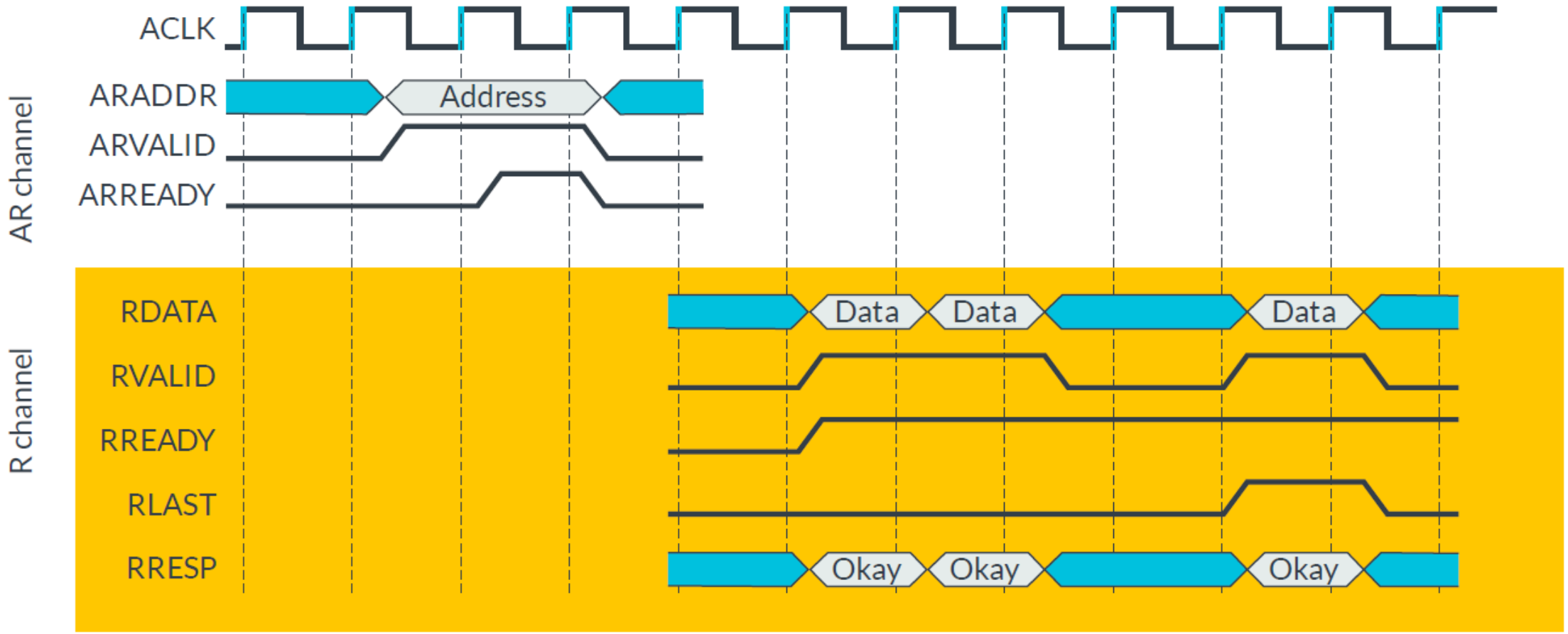
Write Transaction - 4 beats (FIXed)



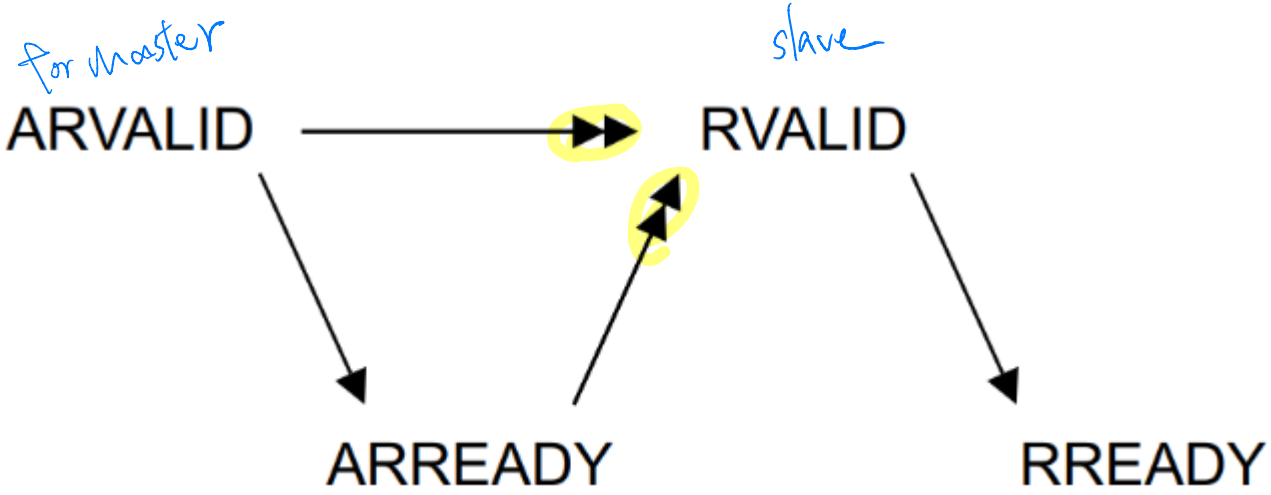
Read Transaction: Single Data



Read Transaction: Multiple Data



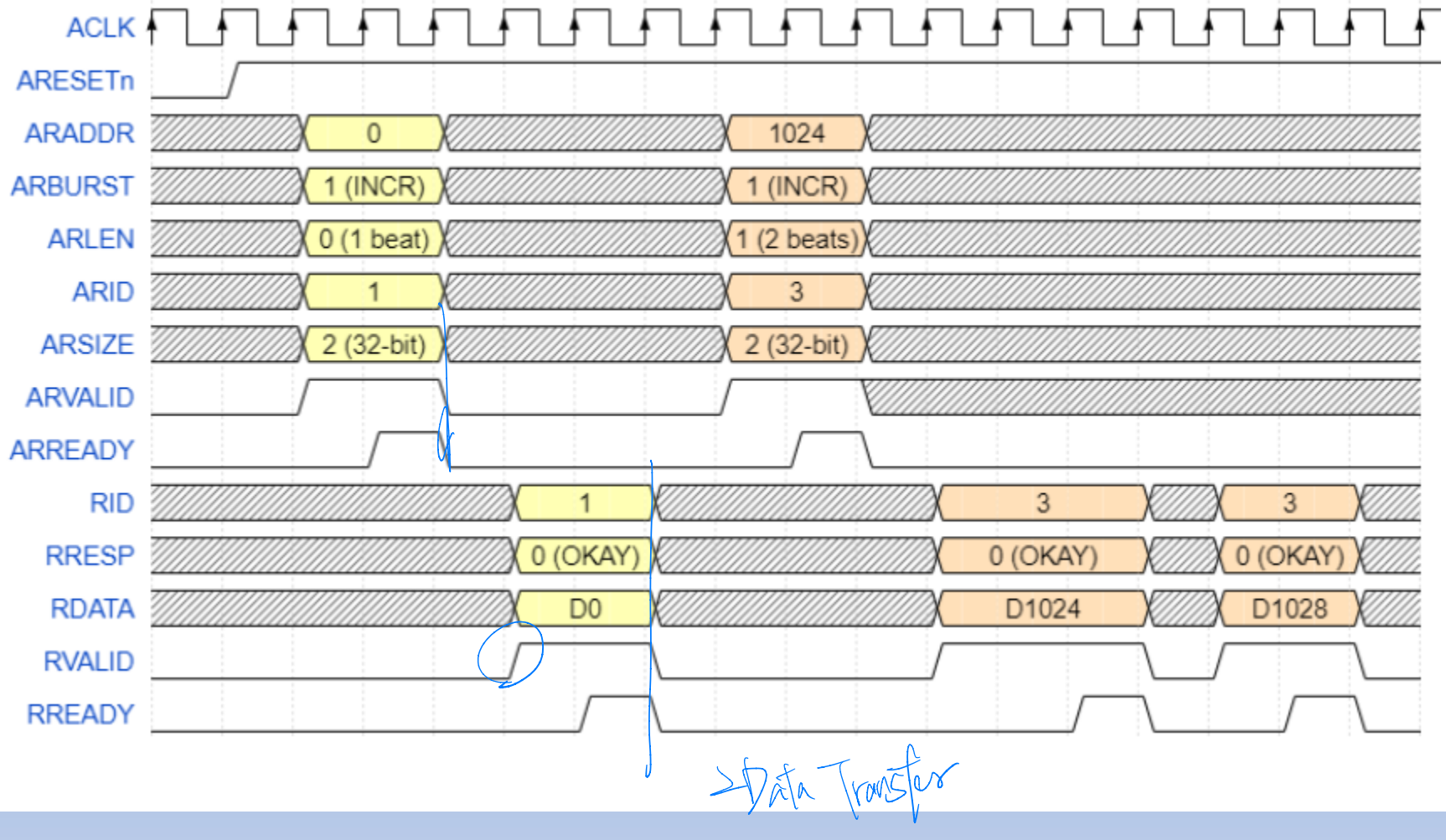
Read Transaction Handshake Dependencies



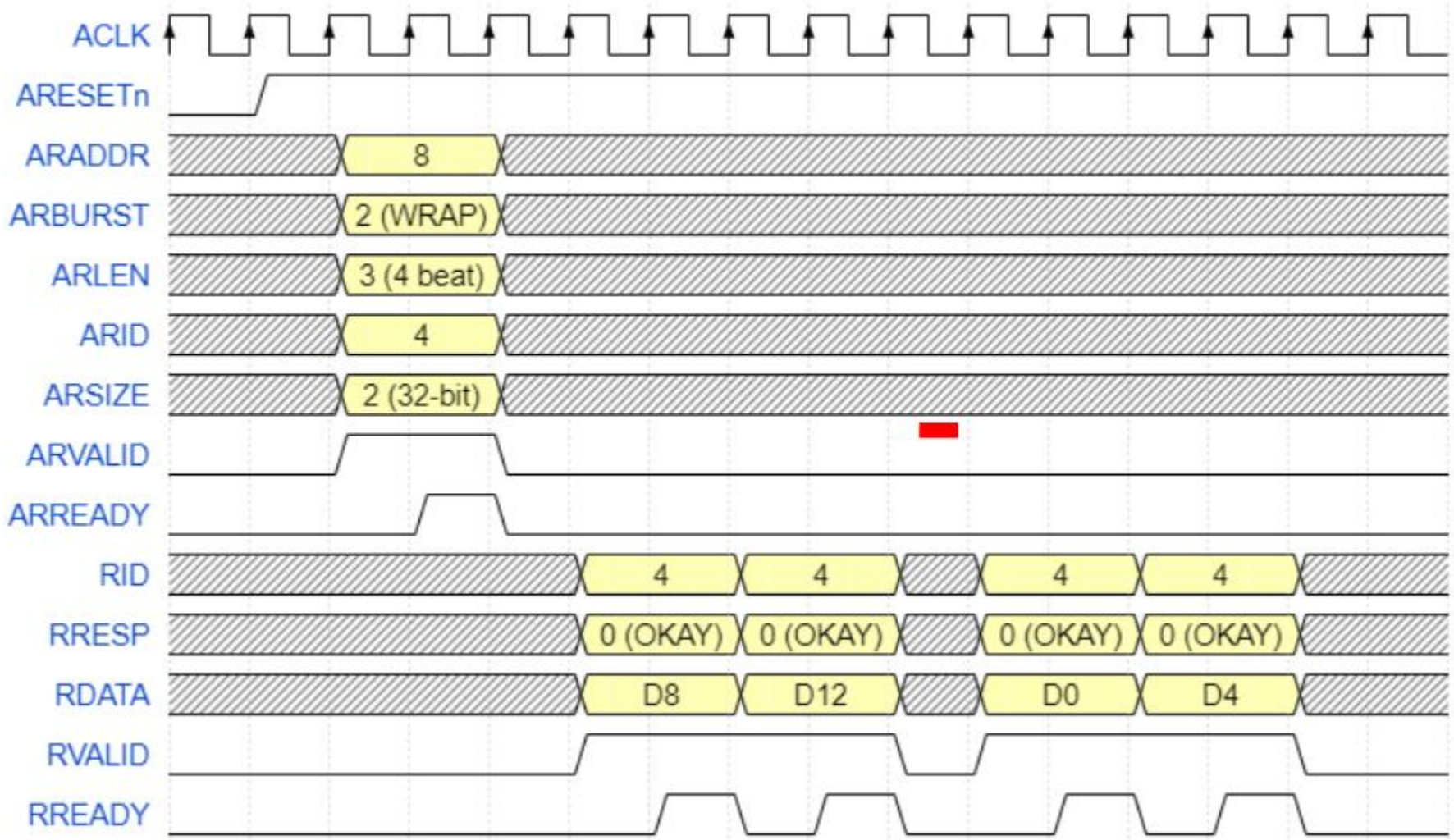
1. Single-headed arrows point to signals that can be asserted before or after the signal at the start of the arrow.
2. Double-headed arrows point to signals that must be asserted only after assertion of the signal at the start of the arrow.

https://habr.com/en/articles/572926/

Read Transaction - One beat (incr) -> two beats (incr)

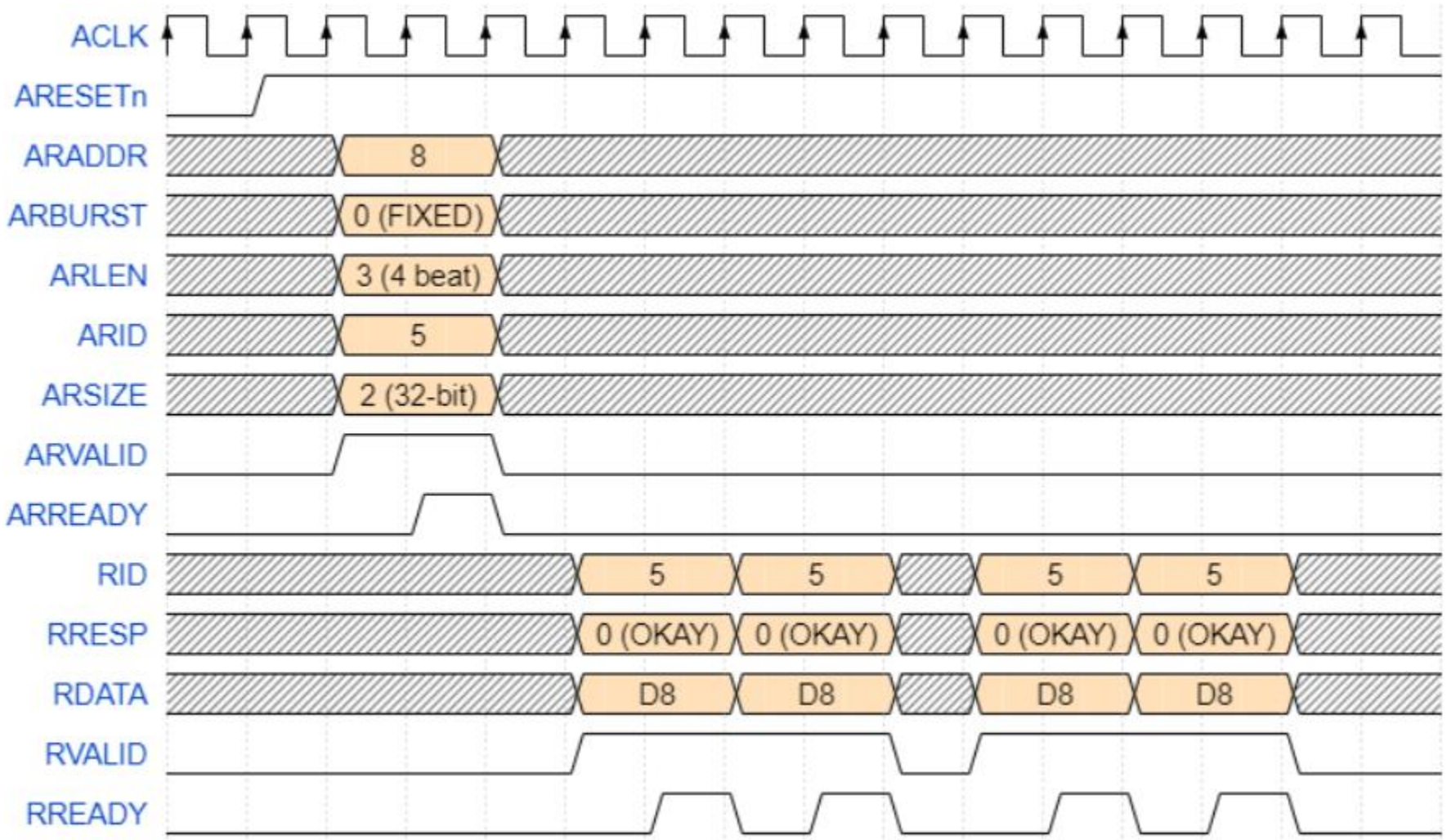


Read Transaction - 4 beats – Burst Type: Wrap



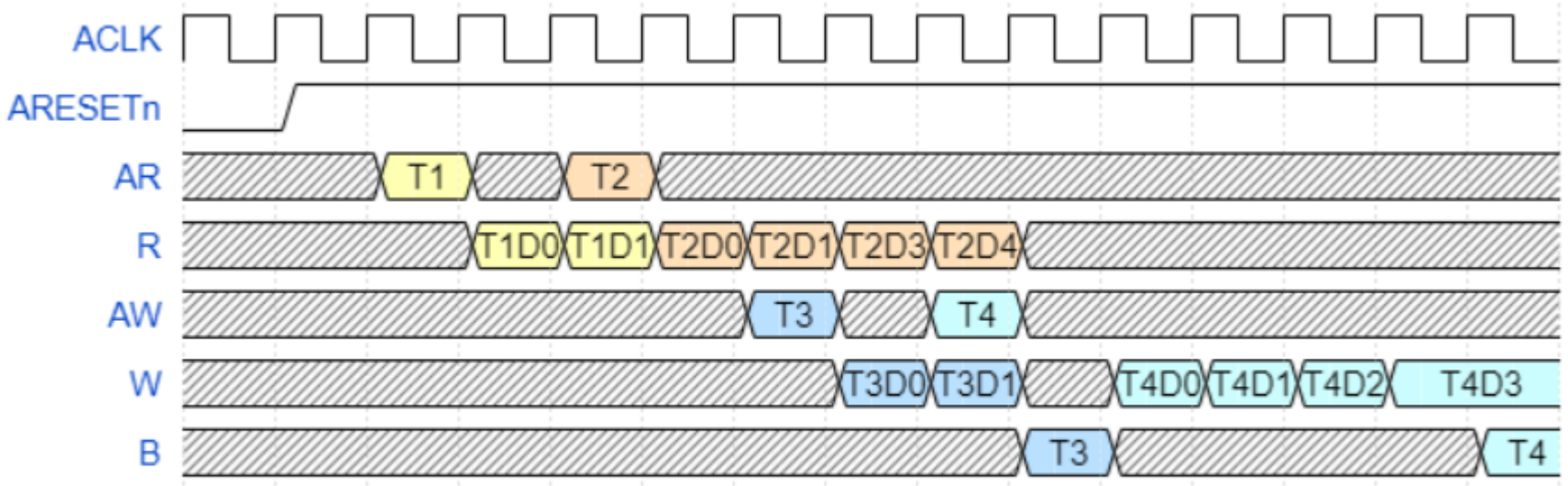
Read Transaction - 4 beats (FIXED)

What is the usage scenario?



Address and Burst Logic

- High latency system (DRAM)
  - Burst
  - Transfer length
  - AxADDR, AxSIZE, AxLEN, AxBURST.



Example of multiple transactions: T1 (arlen=1), T2 (arlen=3) read transactions and T3 (awlen=1), T4 (awlen=3) write transactions.

Write channel signals

Write Address (AW) channel signals	AXI version
AWVALID	AXI3 and AXI4
AWREADY	AXI3 and AXI4
AWADDR[31:0]	AXI3 and AXI4
AWSIZE[2:0]	AXI3 and AXI4
AWBURST[1:0]	AXI3 and AXI4
AWCACHE[3:0]	AXI3 and AXI4
AWPROT[2:0]	AXI3 and AXI4
AWID[x:0]	AXI3 and AXI4
AWLEN[3:0] AWLEN[7:0]	AXI3 only AXI4 only
AWLOCK[1:0] AWLOCK	AXI3 only AXI4 only
AWQOS[3:0]	AXI4 only
AWREGION[3:0]	AXI4 only
AWUSER[x:0]	AXI4 only

Write Data (W) channel signals	AXI version
WVALID	AXI3 and AXI4
WREADY	AXI3 and AXI4
WLAST	AXI3 and AXI4
WDATA[x:0]	AXI3 and AXI4
WSTRB[x:0]	AXI3 and AXI4
WID[x:0]	AXI3 only
WUSER[x:0]	AXI4 only

Write Response (B) channel signals	AXI version
BVALID	AXI3 and AXI4
BREADY	AXI3 and AXI4
BRESP[1:0]	AXI3 and AXI4
BID[x:0]	AXI3 and AXI4
BUSER[x:0]	AXI4 only

Read Channel Signals

Read Address (AR) channel signals	AXI version
ARVALID	AXI3 and AXI4
AREADY	AXI3 and AXI4
ARADDR[31:0]	AXI3 and AXI4
ARSIZE[2:0]	AXI3 and AXI4
ARBURST[1:0]	AXI3 and AXI4
ARCACHE[3:0]	AXI3 and AXI4
ARPROT[2:0}	AXI3 and AXI4
ARID[x:0]	AXI3 and AXI4
ARLEN[3:0] ARLEN[7:0]	AXI3 only AXI4 only
ARLOCK[1:0] ARLOCK	AXI3 only AXI4 only
ARQOS[3:0]	AXI4 only
ARREGION[3:0]	AXI4 only
ARUSER[x:0]	AXI4 only

Read Data (R) channel signals	AXI version
RVALID	AXI3 and AXI4
READY	AXI3 and AXI4
RLAST	AXI3 and AXI4
RDATA[x:0]	AXI3 and AXI4
RRESP[1:0]	AXI3 and AXI4
RID[x:0]	AXI3 and AXI4
RUSER[x:0]	AXI4 only

AxLEN : Address Length
AxSIze : Transfer data size

- AxLEN describes the length of the transaction in the number of transfers.
  - For AXI3, AxLEN[3:0] has 4 bits, which specifies a range of 1-16 transfers in a transaction.
  - For AXI4, AxLEN[7:0] has 8 bits, which specifies a range of 1-256 data transfers in a transaction.

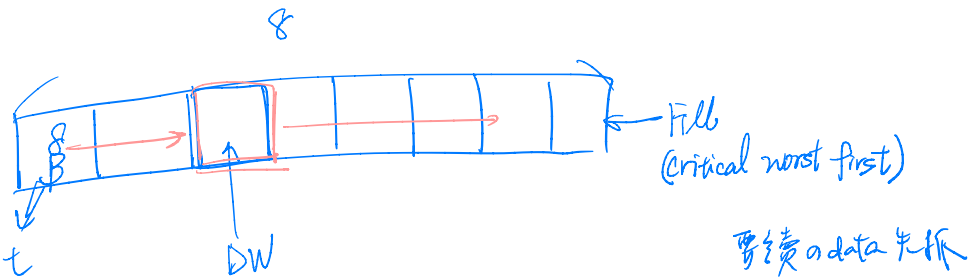
- AxSize[2:0] : Max number of bytes to transfer in each transaction
  - 2^(AxSize[2:0]) : 1, 2, 4, 8, 16, 32, 64, or 128 bytes per transfer

AxBURST[1:0] – Match Cacheline fill order – Critical Word First

Value	Burst type	Usage notes	Length (number of transfers)	Alignment
0x00	FIXED	Reads the same address repeatedly. Useful for FIFOs.	1-16	Fixed byte lanes only defined by start address and size.
0x01	INCR	Incrementing burst. The slave increments the address for each transfer in the burst from the address for the previous transfer. The incremental value depends on the size of the transfer, as defined by the AxSIZE attribute. Useful for block transfers.	AXI3: 1-16 AXI4: 1-256	Unaligned transfers are supported.
0x10	WRAP	Wrapping burst. Similar to an incrementing burst, except that if an upper address limit is reached, the address wraps around to a lower address. Commonly used for cache line accesses.	2, 4, 8, or 16	The start address must be aligned to the transfer size.
0x11	RESERVED	Not for use.	-	-

Cache Fill (Critical Word First)-> Bus Order (WRAP) -> DRAM Order (Interleave)

Chache line
(block)



64 DRAM BUS width

burst length = 8

mode { inc
wrap

DRAM Burst Order Matches Cacheline Fill

Table 3 — Burst Type and Burst Order

Burst Length	READ/ WRITE	Starting Column ADDRESS (A2,A1,A0)	burst type = Sequential (decimal) A3 = 0	burst type = Interleaved (decimal) A3 = 1	Notes
4 Chop	READ	0 0 0	0,1,2,3,T,T,T,T	0,1,2,3,T,T,T,T	1, 2, 3
		0 0 1	1,2,3,0,T,T,T,T	1,0,3,2,T,T,T,T	1, 2, 3
		0 1 0	2,3,0,1,T,T,T,T	2,3,0,1,T,T,T,T	1, 2, 3
		0 1 1	3,0,1,2,T,T,T,T	3,2,1,0,T,T,T,T	1, 2, 3
		1 0 0	4,5,6,7,T,T,T,T	4,5,6,7,T,T,T,T	1, 2, 3
		1 0 1	5,6,7,4,T,T,T,T	5,4,7,6,T,T,T,T	1, 2, 3
		1 1 0	6,7,4,5,T,T,T,T	6,7,4,5,T,T,T,T	1, 2, 3
		1 1 1	7,4,5,6,T,T,T,T	7,6,5,4,T,T,T,T	1, 2, 3
	WRITE	0,V,V	0,1,2,3,X,X,X,X	0,1,2,3,X,X,X,X	1, 2, 4, 5
		1,V,V	4,5,6,7,X,X,X,X	4,5,6,7,X,X,X,X	1, 2, 4, 5
8	READ	0 0 0	0,1,2,3,4,5,6,7	0,1,2,3,4,5,6,7	2
		0 0 1	1,2,3,0,5,6,7,4	1,0,3,2,5,4,7,6	2
		0 1 0	2,3,0,1,6,7,4,5	2,3,0,1,6,7,4,5	2
		0 1 1	3,0,1,2,7,4,5,6	3,2,1,0,7,6,5,4	2
		1 0 0	4,5,6,7,0,1,2,3	4,5,6,7,0,1,2,3	2
		1 0 1	5,6,7,4,1,2,3,0	5,4,7,6,1,0,3,2	2
		1 1 0	6,7,4,5,2,3,0,1	6,7,4,5,2,3,0,1	2
		1 1 1	7,4,5,6,3,0,1,2	7,6,5,4,3,2,1,0	2
	WRITE	V,V,V	0,1,2,3,4,5,6,7	0,1,2,3,4,5,6,7	2, 4

NOTE 1 In case of burst length being fixed to 4 by MR0 setting, the internal write operation starts two clock cycles earlier than for the BL8 mode. This means that the starting point for tWR and tWTR will be pulled in by two clocks. In case of burst length being selected on-the-fly via A12/BC#, the internal write operation starts at the same point in time like a burst of 8 write operation. This means that during on-the-fly control, the starting point for tWR and tWTR will not be pulled in by two clocks.

NOTE 2 0...7 bit number is value of CA[2:0] that causes this bit to be the first read during a burst.

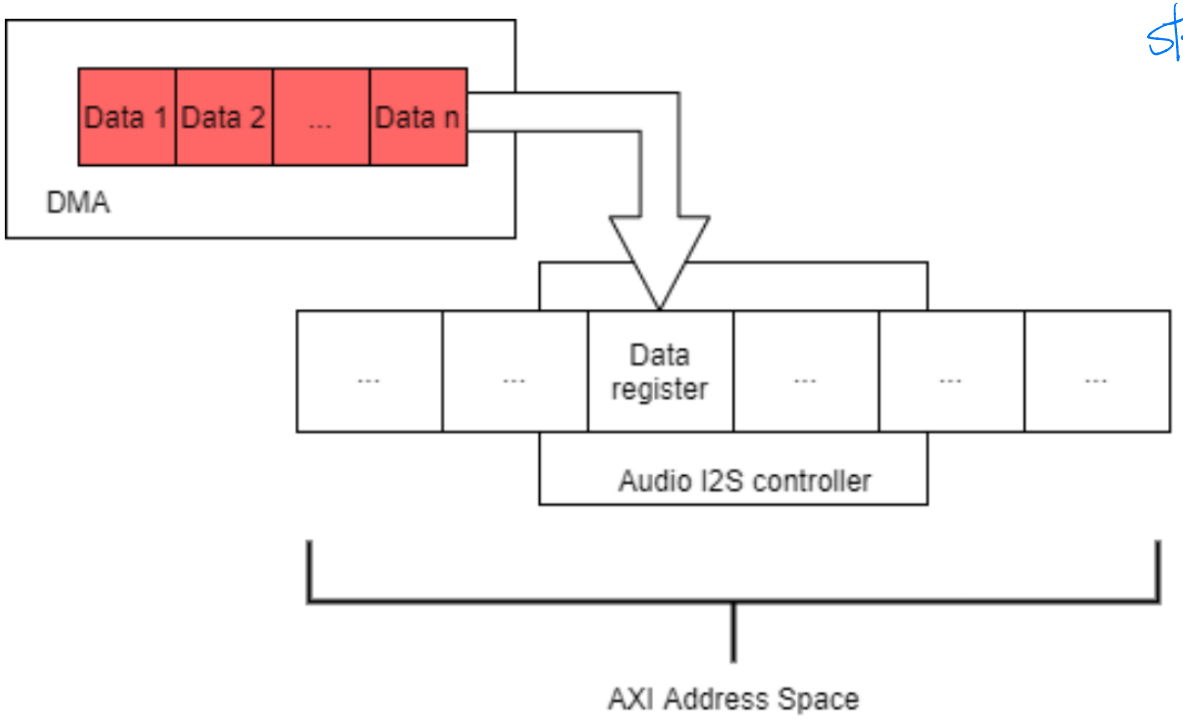
NOTE 3 T: Output driver for data and strobes are in high impedance.

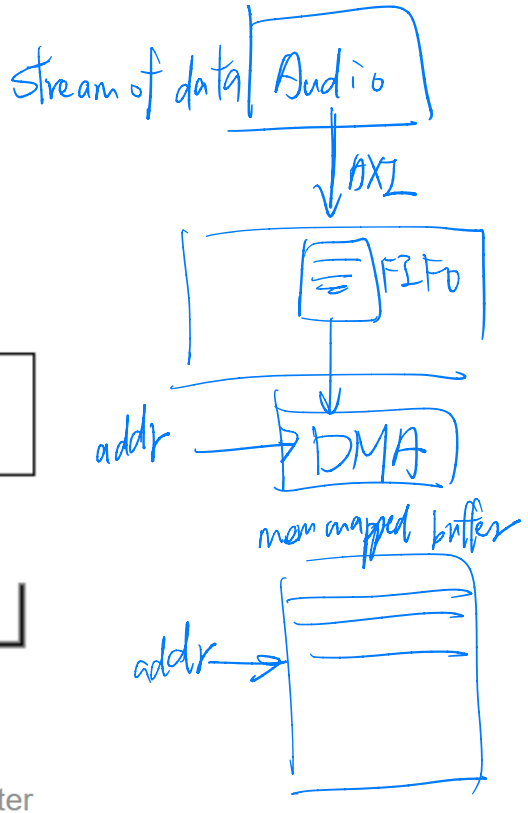
NOTE 4 V: a valid logic level (0 or 1), but respective buffer input ignores level on input pins.

NOTE 5 X: Don't Care.

Example to use FIXED BURST

you have audio samples that are being read from memory and are streamed to the audio controller, which accepts data in a single address location. Lab-FIR is another example.





Fixed burst done by DMA to one fixed location: Audio controller's data input register

AxPROT

AxPROT[0] (P) identifies an access as unprivileged or privileged:

- 1 indicates privileged access.
- 0 indicates unprivileged access.

[Note]: Although some processors support multiple levels of privilege, the only distinction that AXI can provide is between privileged and unprivileged access.

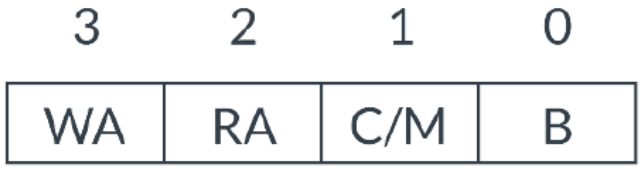
- AxPROT[1] (NS) identifies an access as Secure or Non-secure:
  - 1 indicates a Non-secure transaction.
  - 0 indicates a Secure transaction.

- AxPROT[2] (I) indicates whether the transaction is an instruction access or a data access:
  - 1 indicates an instruction access.
  - 0 indicates a data access.

  [Note]
  - The AXI protocol defines this indication as a hint.
  - It is not accurate in all cases, for example, where a transaction contains a mix of instruction and data items.
  - The Arm AXI specification for both AXI 3 and AXI 4 recommends that a master sets bit 2 to zero to indicate a data access, unless the access is specifically known to be an instruction access.

- Example Usage
  - Boot memory with a bootloader can be made invisible to supervisor-level software (kernel)

Cache support – AWCACHE, ARCACHE



MMU or MMIO

- AxCACHE [0] (B) is the bufferable bit.
  - 1 : Response can come from an intermediate point
  - 0 : Response must from destination slave

- AxCACHE [1] (C/M) - Cacheable in AXI3, Modifiable bit in AXI4.
  - Write - Several different writes can be merged, or a single write can be broken into multiple transactions.
  - Read - Prefetched or a single fetch can be used for multiple read transactions.

- AxCACHE [2] (RA – Read Allocation)
  - Read allocation is recommended

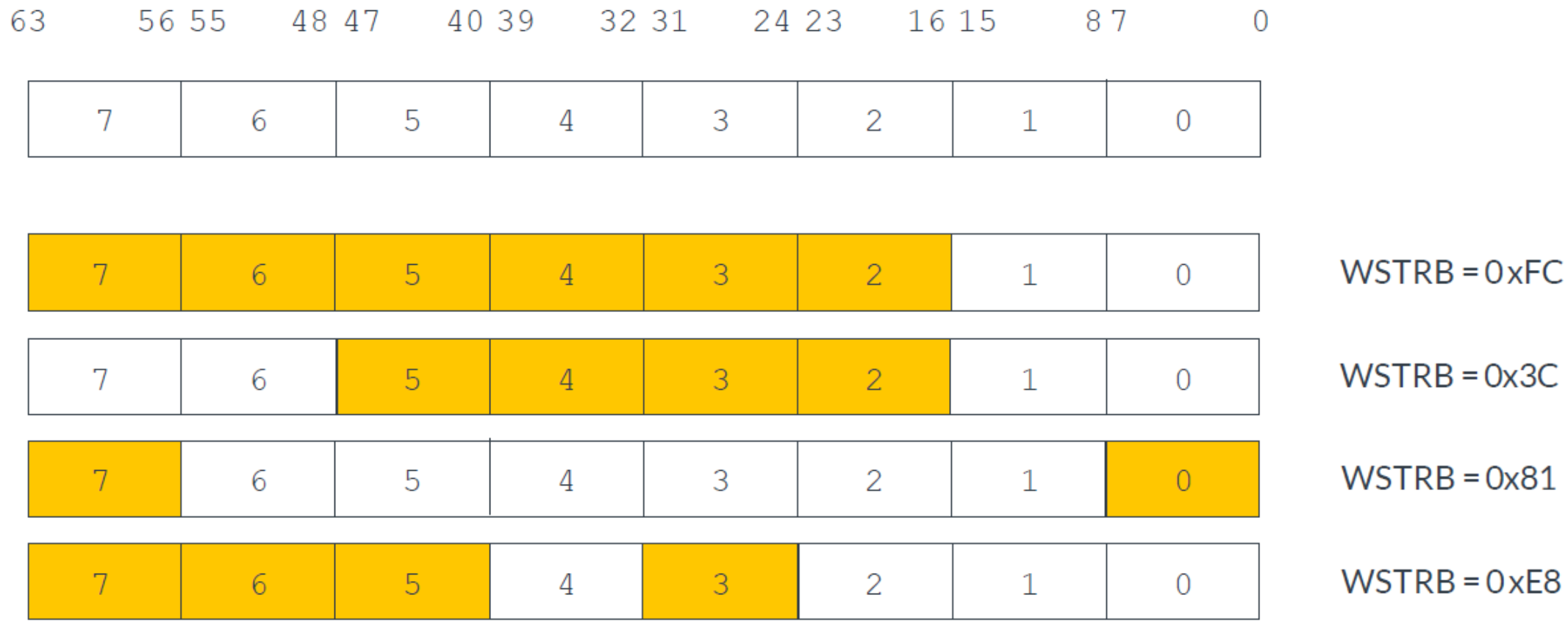
- AxCACHE [3] (WA – Write Allocation)
  - Write allocation is recommended

Note: AxCACHE [2] or AxCACHE [3] is asserted, then the transaction must be looked up in a cache as it could have been allocated in this cache by another master.

Write Data Strobe

Byte enable

For 64 bit WDATA bus



Response signaling

Response code	Description
00 - OKAY	Normal access success or exclusive access failure. OKAY is the response that is used for most transactions. OKAY indicates that a normal access has been successful. This response can also indicate that an exclusive access has failed. An exclusive access is when more than one master can access a slave at once, but these masters cannot access the same memory range.
01 - EXOKAY	Exclusive access okay. EXOKAY indicates that either the read or write portion of an exclusive access has been successful.
10 – SLVERR	Slave error. SLVERR is used when the access has reached the slave successfully, but the slave wants to return an error condition to the originating master. This indicates an unsuccessful transaction. For example, when there is an unsupported transfer size attempted, or a write access attempted to read-only location.
11 - DECERR	Decode error. DECERR is often generated by an interconnect component to indicate that there is no slave at the transaction address.

Atomic Accesses

- Read, Modify, Write

- **Locked** – only for AXI3
  - master locks a slave, and prevent other master access

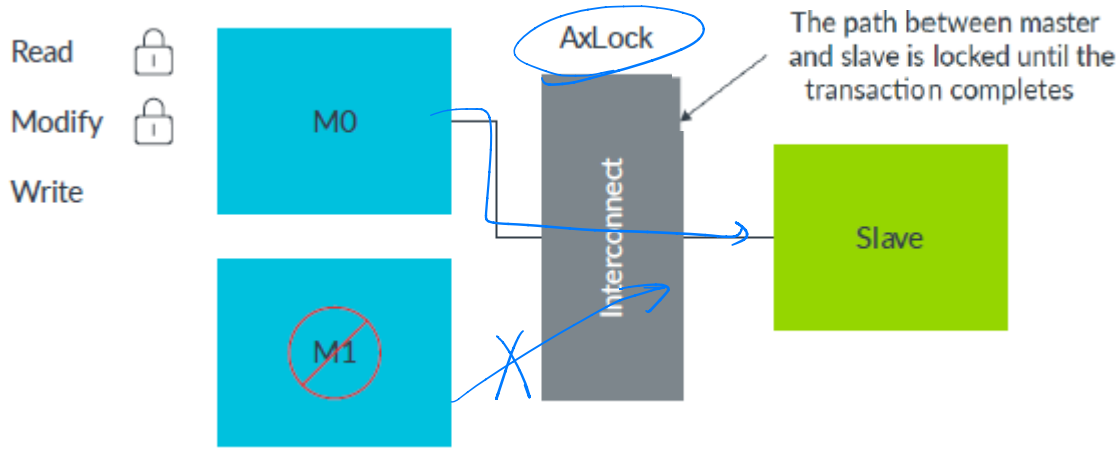
- **Exclusive** – for AXI4
  - Master locks a memory region in the slave, other master is allowed to access the same slave, but not the locked memory region

Locked Access

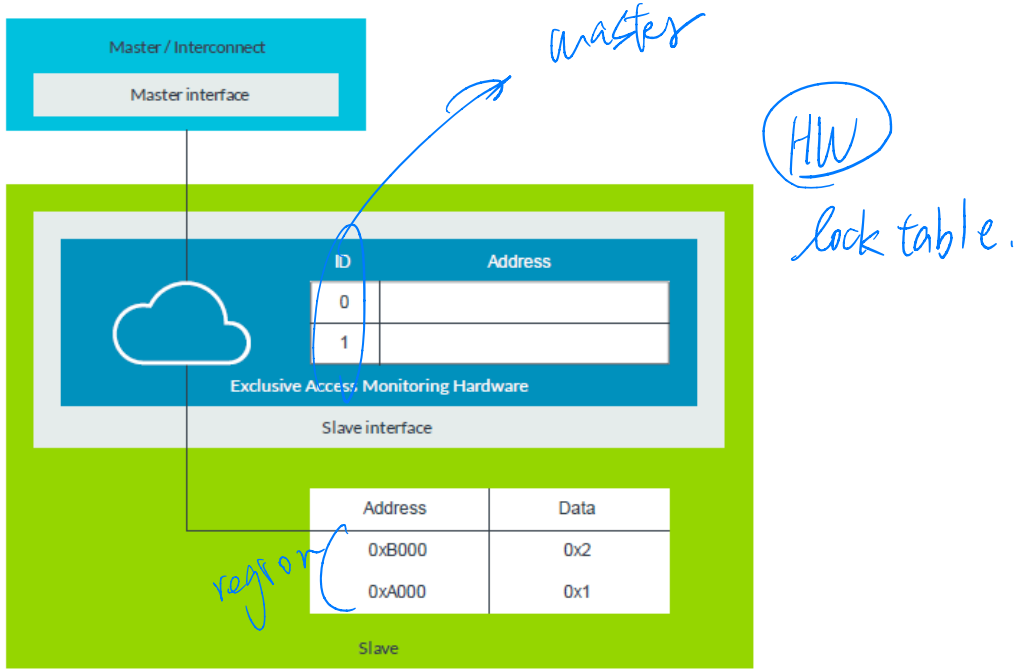
1. Master M0 initiates a sequence of READ, MODIFY, and WRITE. The first transaction, READ, has the LOCK signal asserted, indicating that it starts a locked transaction.

2. The interconnect locks out any other transactions. From this point, master M1 has no access to the slave.

3. The final transaction in the sequence, WRITE, does not have the LOCK signal asserted. This transaction indicates the end of the locked sequence. The interconnect removes the lock, and other masters can now access the slave.

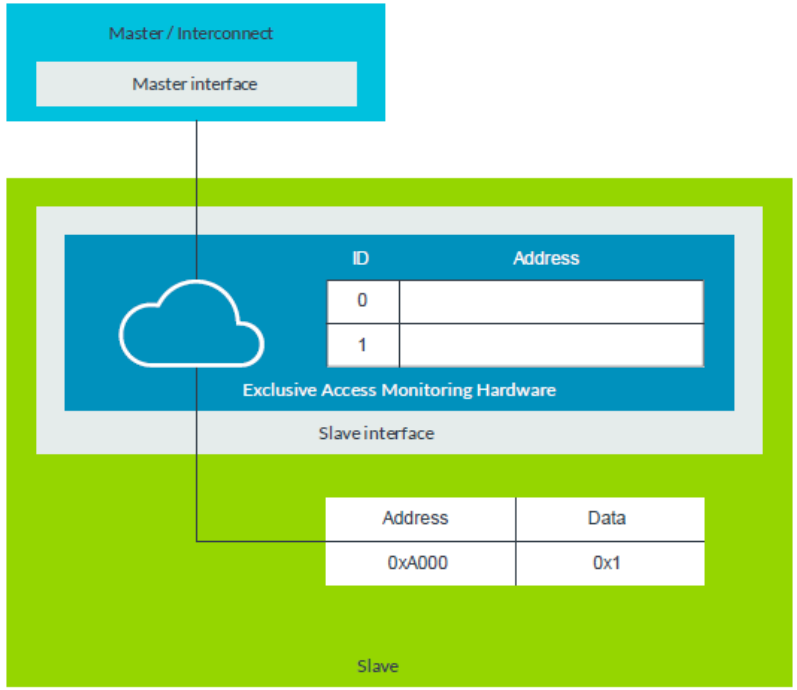


Exclusive Access pair – both succeed



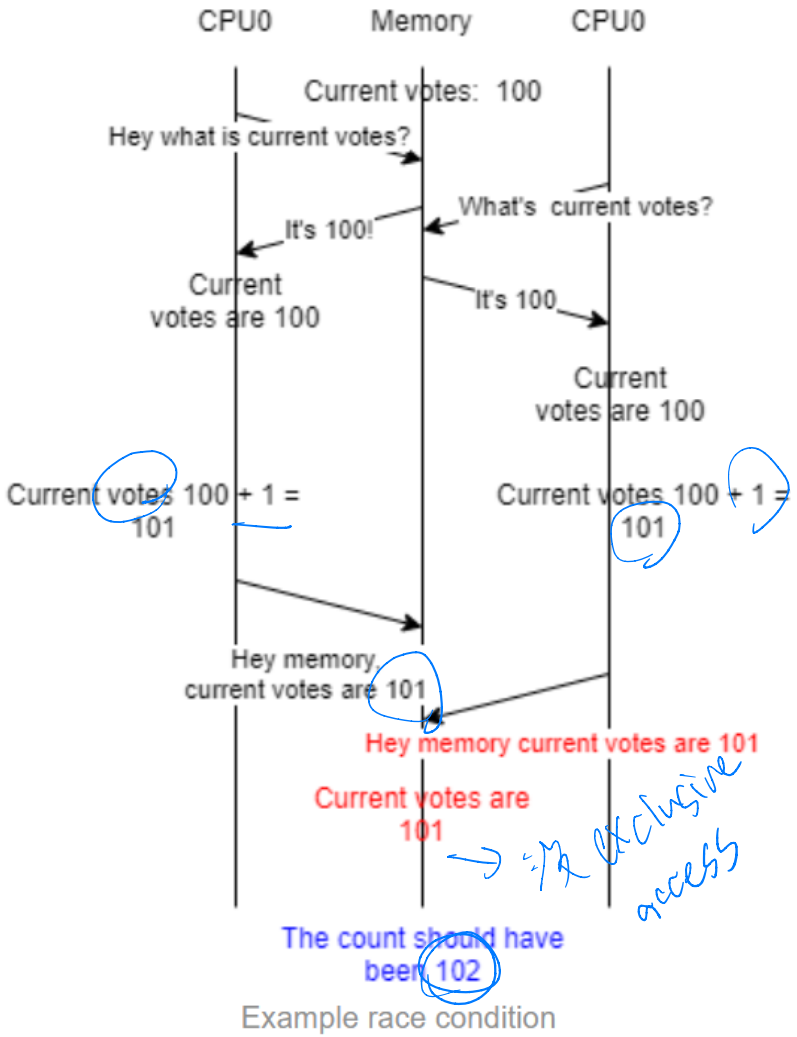
Transaction number	Read or write	Transaction ID	Address	Data	xRESP
1	R	0	0xA000	0x1	EXOKAY
2	R	1	0xB000	0x2	EXOKAY
3	W	0	0xA000	0x3	EXOKAY
4	W	1	0xB000	0x4	EXOKAY

Exclusive Access: One passes, One fails

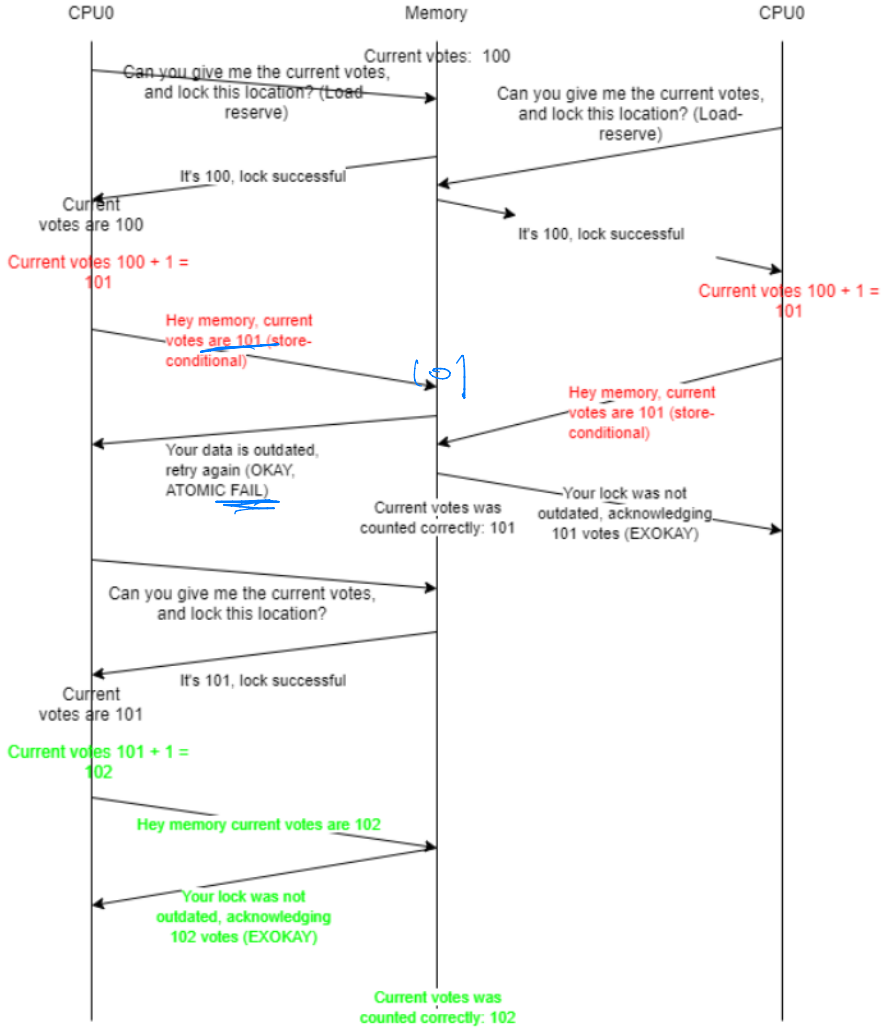


Transaction number	Read or write	Transaction ID	Address	Data	xRESP
1	R	0	0xA000	0x1	EXOKAY
2	R	1	0xA000	0x1	EXOKAY
3	W	0	0xA000	0x3	EXOKAY
4	W	1	0xA000	0x4	OKAY

Example – Compare-and-Swap (CAS)



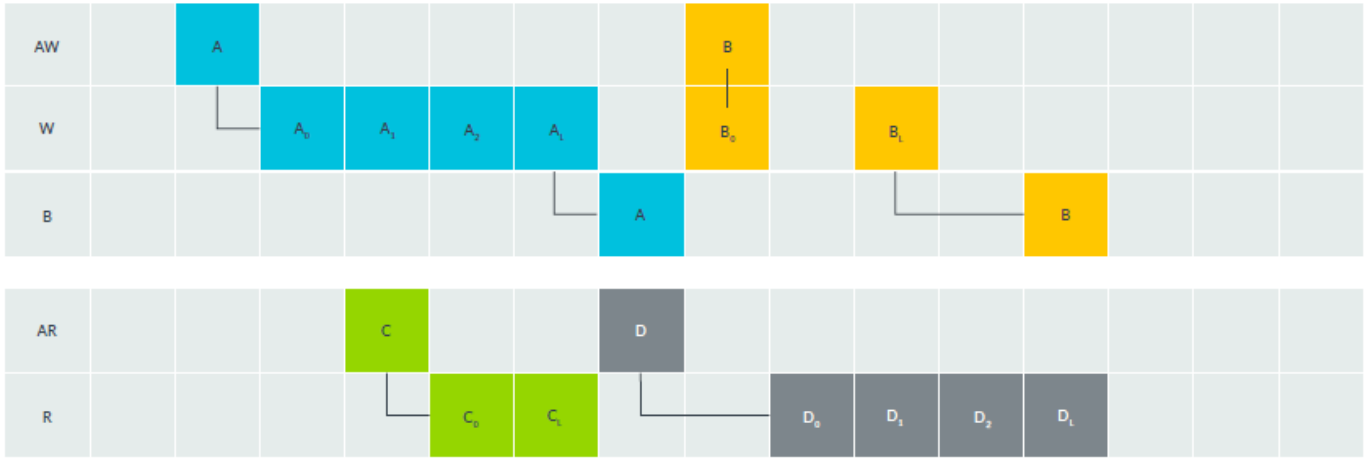
Example race condition



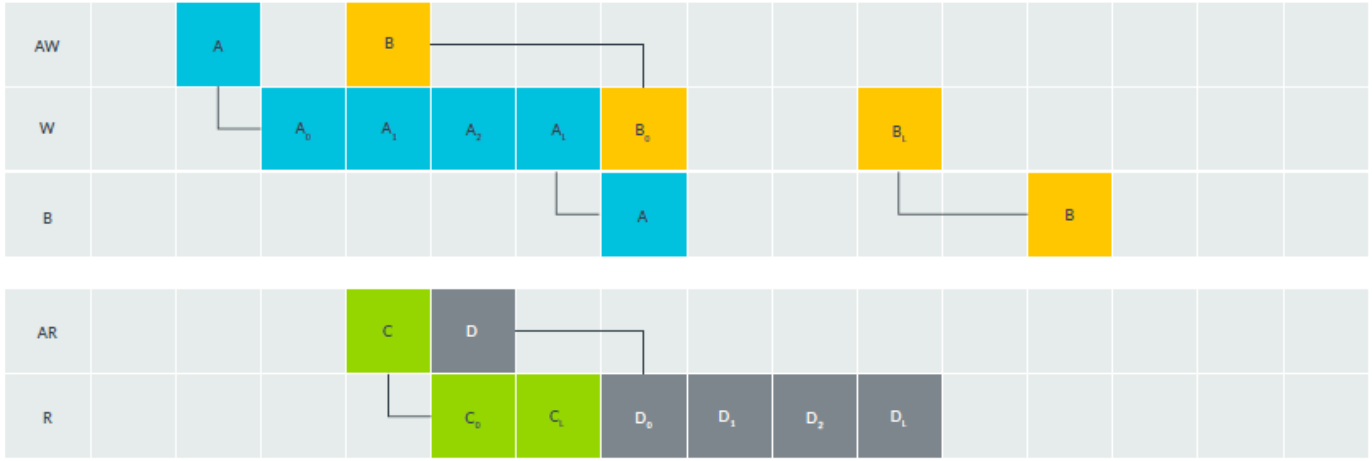
Race condition solved using load-reserve and store-conditional

Non-Pipeline/Pipeline Transaction

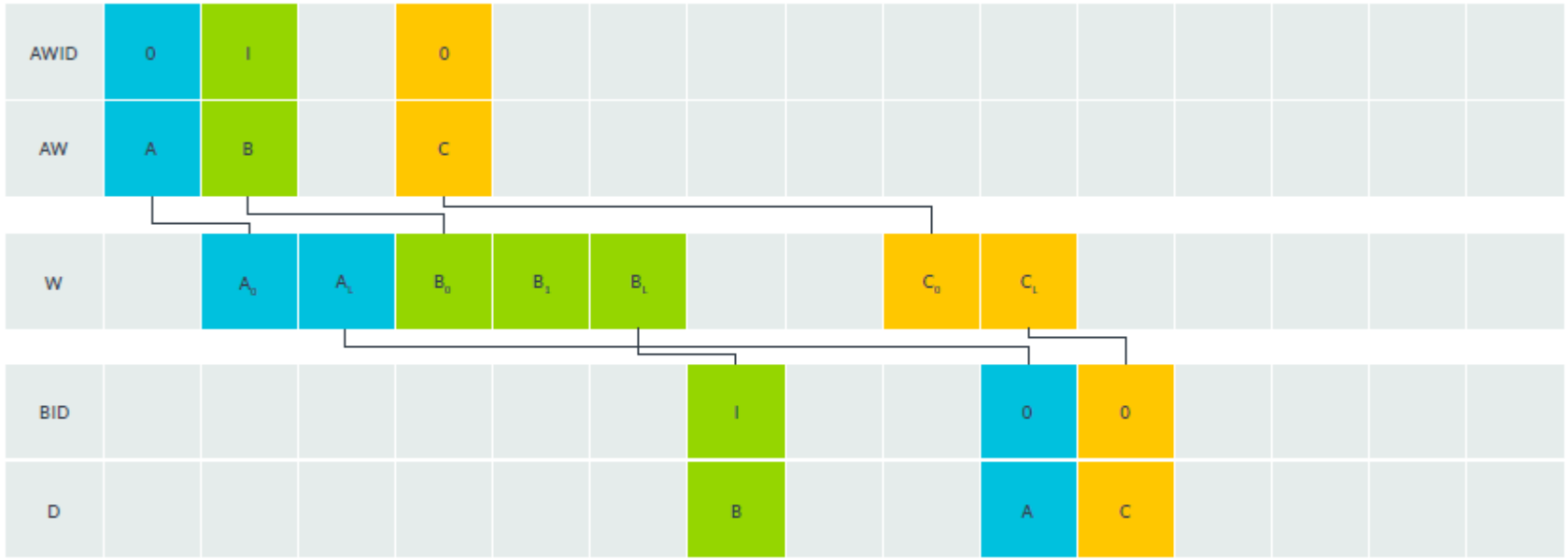
Non-Pipeline



Pipeline



Write Transaction Ordering Rules



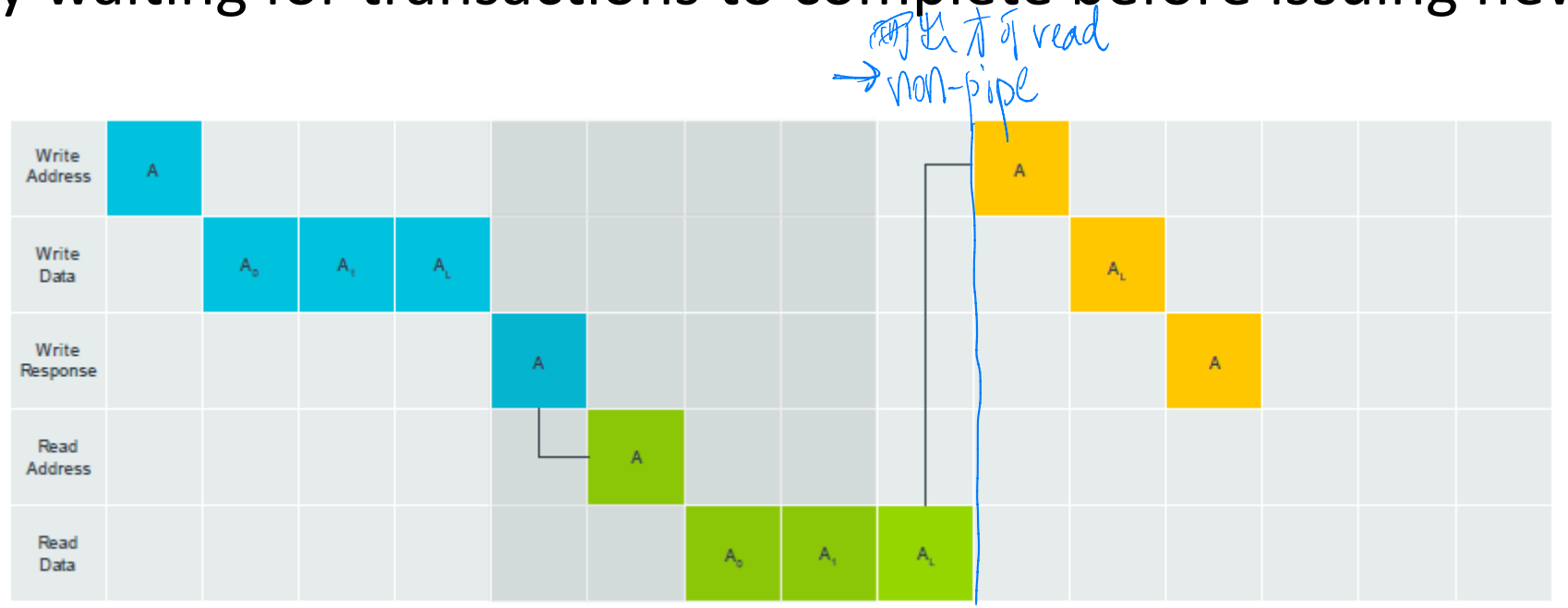
- Write data on the W channel must follow the same order as the address transfers on the AW channel.
- Transactions with different IDs can complete in any order.
- A master can have multiple outstanding transactions with the same ID, but they must be performed in order and complete in order.

Read Transaction Ordering Rules

- Read data for different IDs on the R channel has no ordering restrictions. This means that the slave can send it in any order.
- The read data for the different IDs on the R channel can be interleaved, with the RID value differentiating which transaction the data relates to.
- For transactions with the same ID, read data on the R channel must be returned in the order that they were requested.

Read and write channel ordering – NO ORDERING REQUIREMENT

if a master requires ordering for a specific sequence of reads and writes, the master must ensure that the transaction order is respected by explicitly waiting for transactions to complete before issuing new ones.



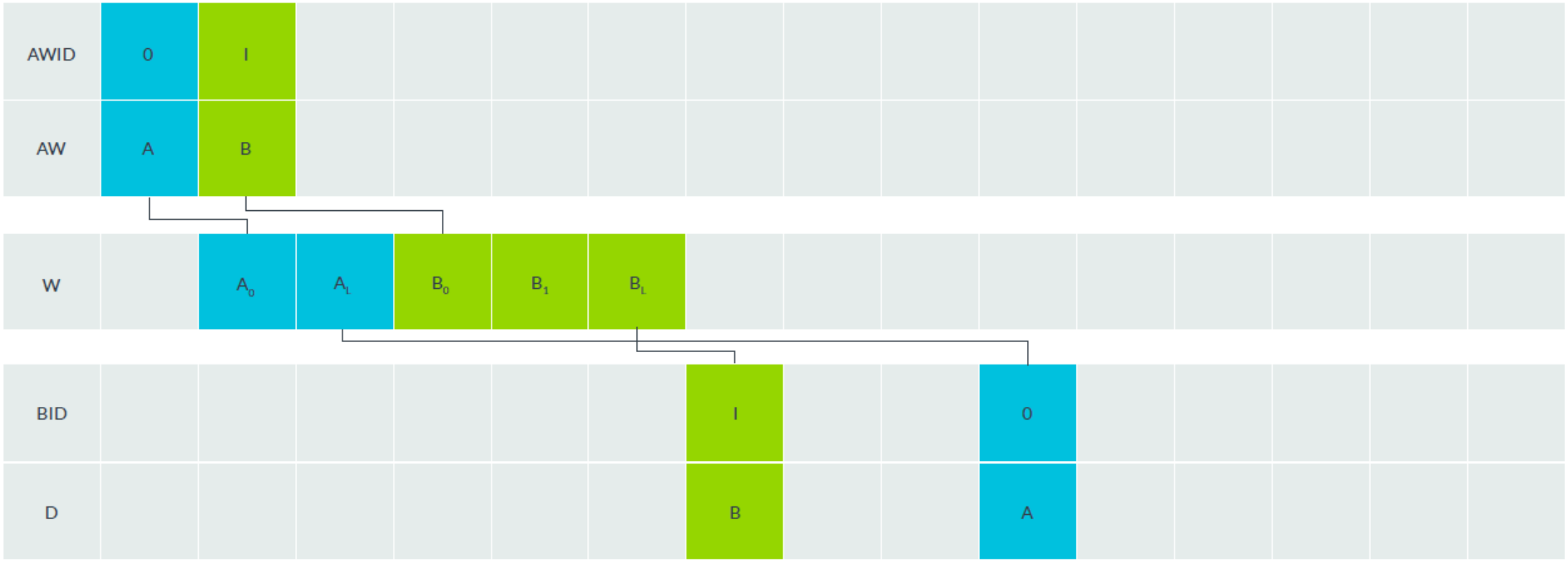
AXI Channel Dependencies

- WLAST transfer must complete before BVALID is asserted.
  - The master must send all the write data before a write response can be seen by the master. This dependency does not exist in AXI3 but is introduced for AXI4:
  - In AXI3, the address does not have to be seen before a write response is sent.
  - In AXI4, all of the data and the address must have been transferred before the master can see a write response.
- RVALID cannot be asserted until ARADDR has been transferred.
  - The slave cannot transfer any read data without it seeing the address first. • WVALID can assert before AWVALID.
- A master could use the Write Data channel to send data to the slave, before communicating the address where the slave should write these data. => What is the implication ?

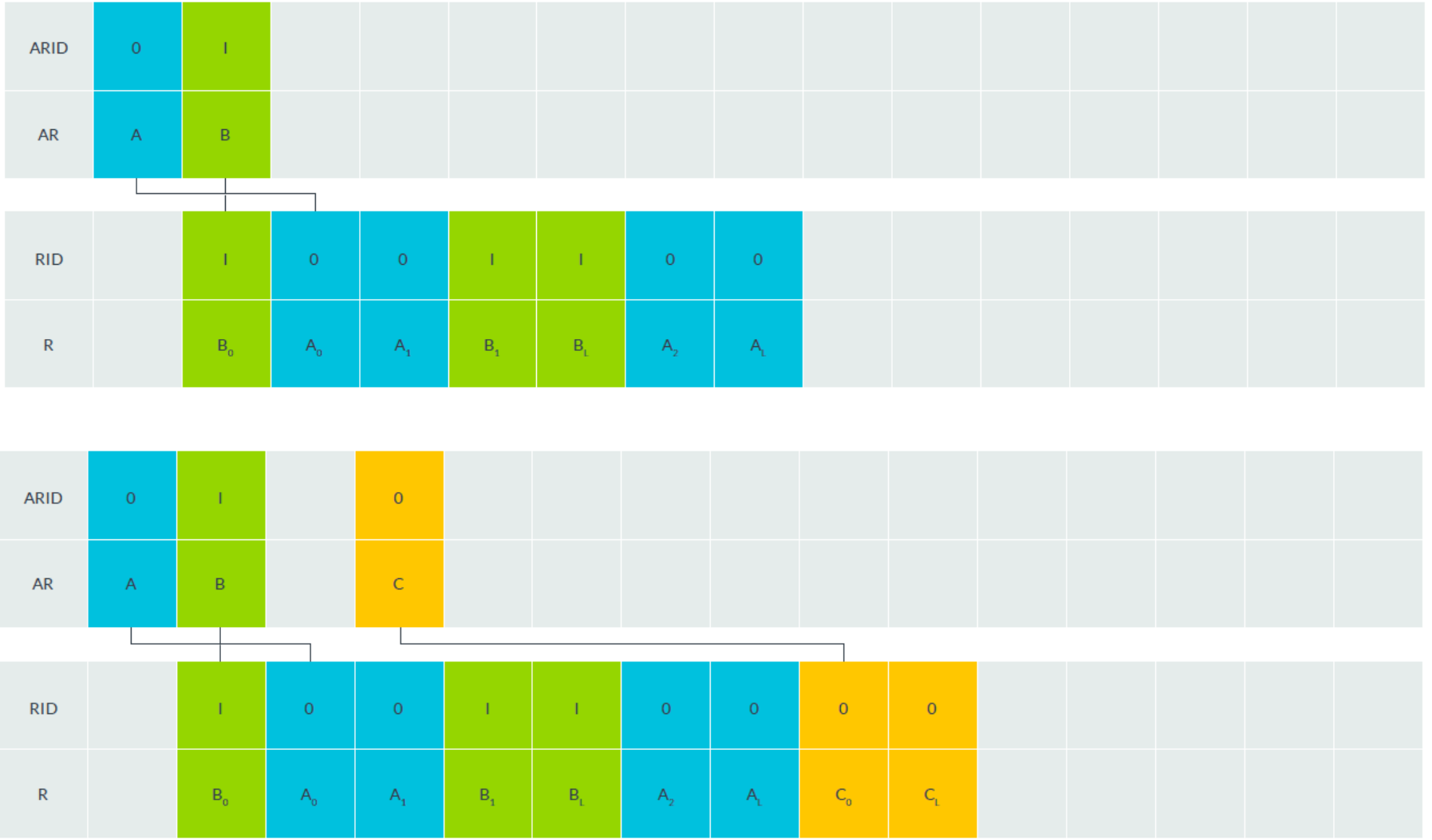
Transfer IDs – AWID, WID, BID, ARID, RID

- All transactions with a given ID must be ordered.
- There is no restriction on the ordering of transactions with different IDs.
- When working with transfer IDs, follow these rules:
  - All transfers must have an ID.
  - All transfers in a transaction must have the same ID.
  - Masters can support multiple IDs for multiple threads.
  - Slaves generally need a configurable ID width.
- ID width
  - write ID width - Number of bits for AWID, WID and BID
  - read ID width - Number of bits for ARID and RID

Write Transaction Out-of-Order

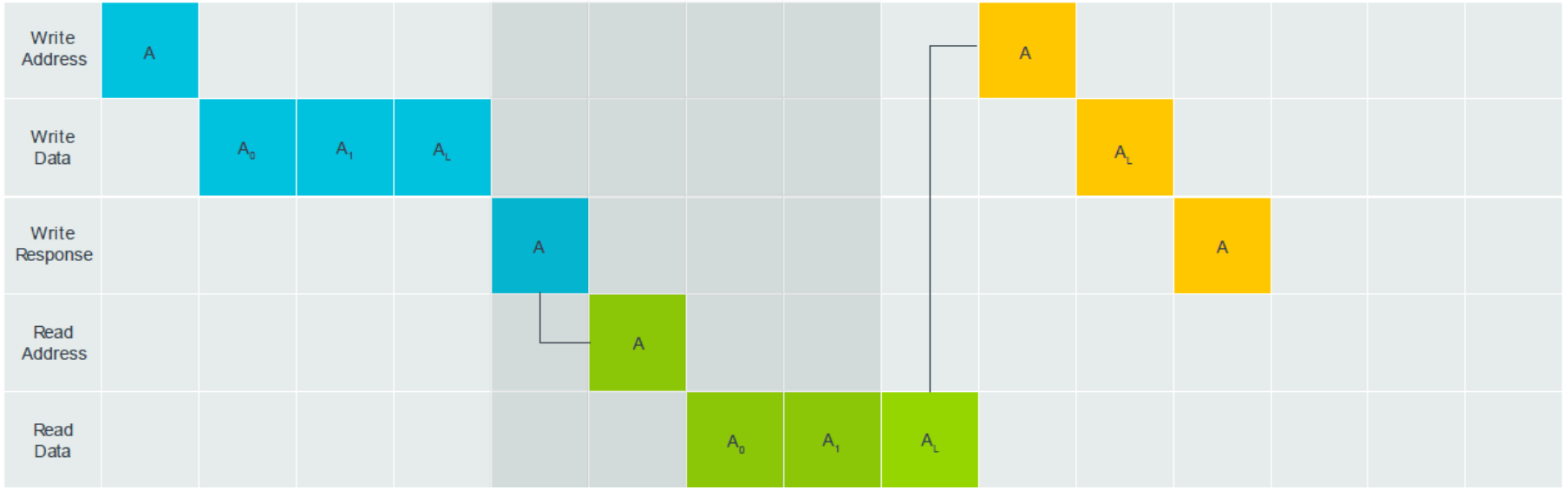


Read out-of-Order



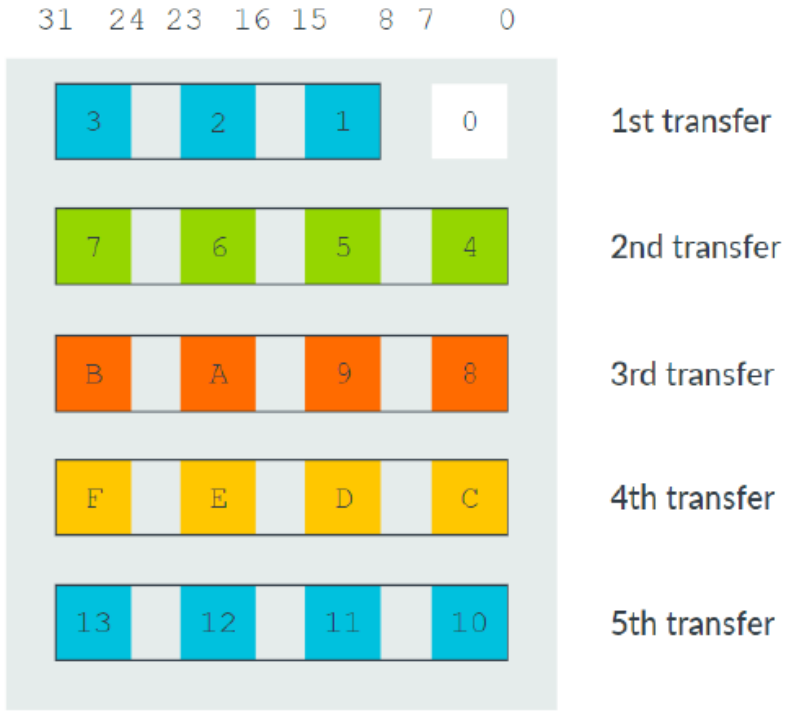
Read / Write Ordering

- Read and write channels have no ordering rules.
- If a master requires a specific sequence of reads and write, explicitly waiting for transaction completion before issuing next transaction:
  1. The master starts the first write transaction.
  2. The master waits for the signal on the Write Response channel.
  3. The master starts the read transaction.
  4. The master waits for the final response on the Read Data channel

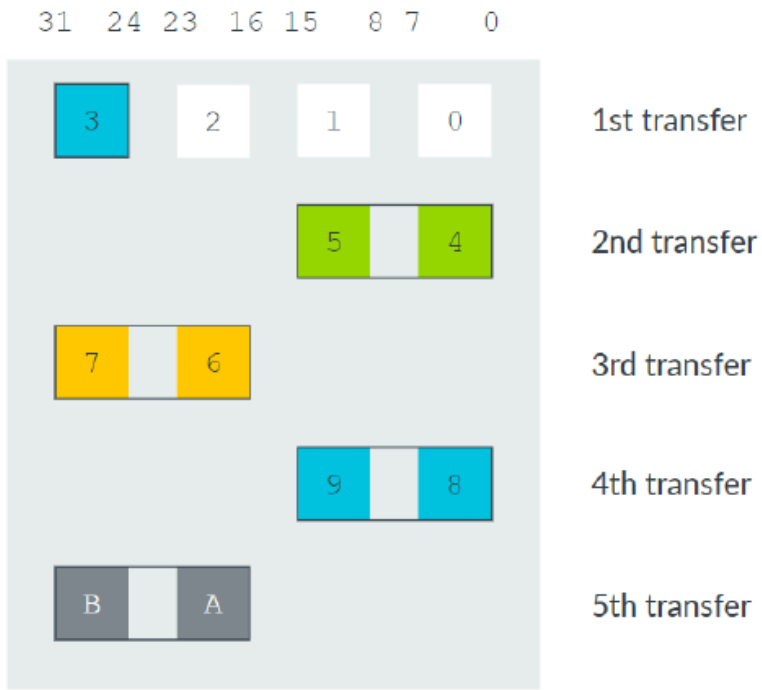


Unaligned Transfer – use WSTRB

AxADDR: 0x01
AxSIZE: 32 bits
AxBURST: INCR
AxLEN: 5 transfers



AxADDR: 0x03
AxSIZE: 16 bits
AxBURST: INCR
AxLEN: 5 transfers



Questions:

1. What is the purpose of transfer IDs?
2. What burst type must a master issue if it wants to write to a FIFO: fixed, wrapping, or incrementing?
3. Which signals can be used to protect against illegal transactions downstream in the system?

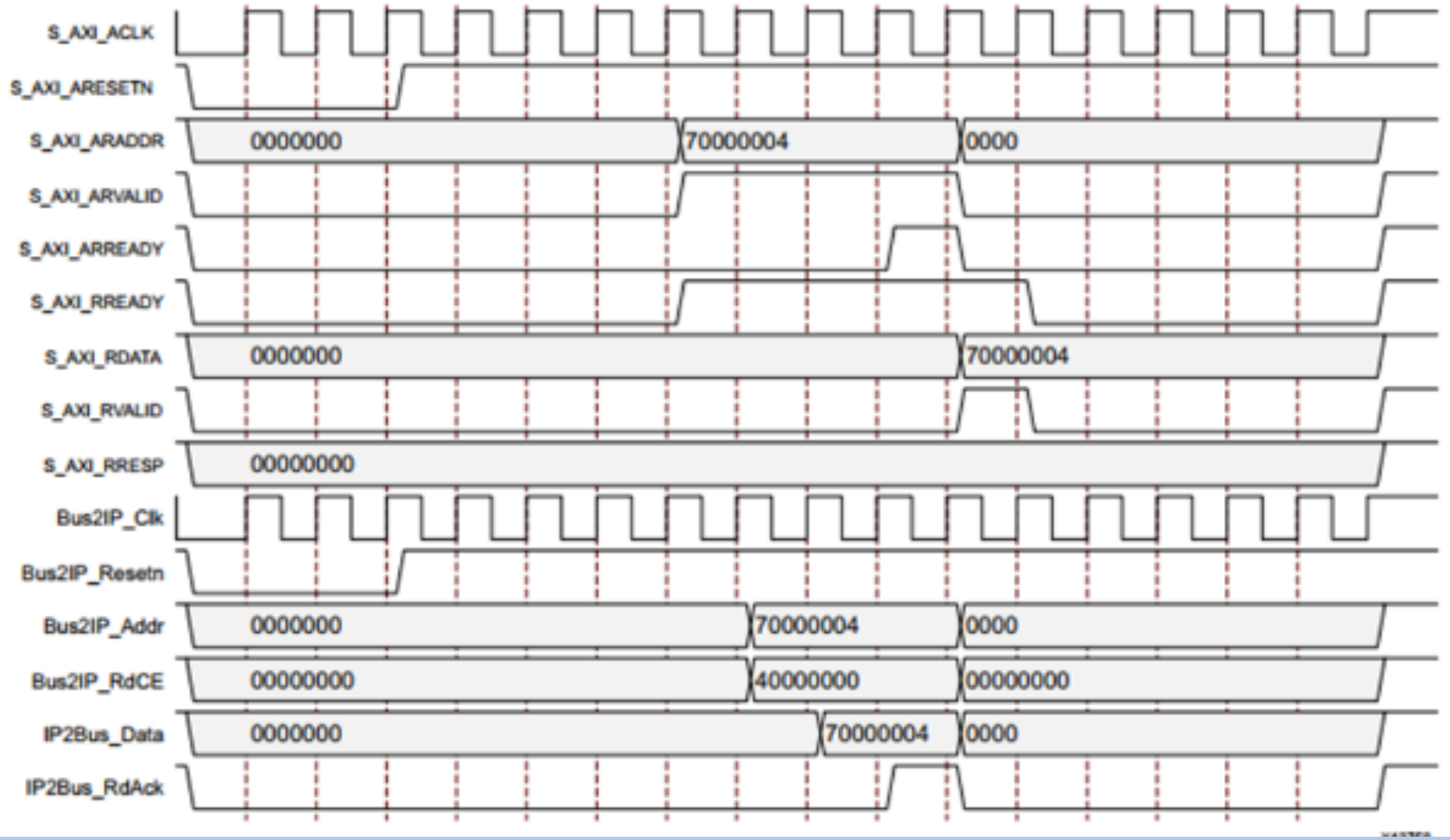
AXI4-Lite Signals

- do not support burst access and transaction ID
- 32-bit Data Bus

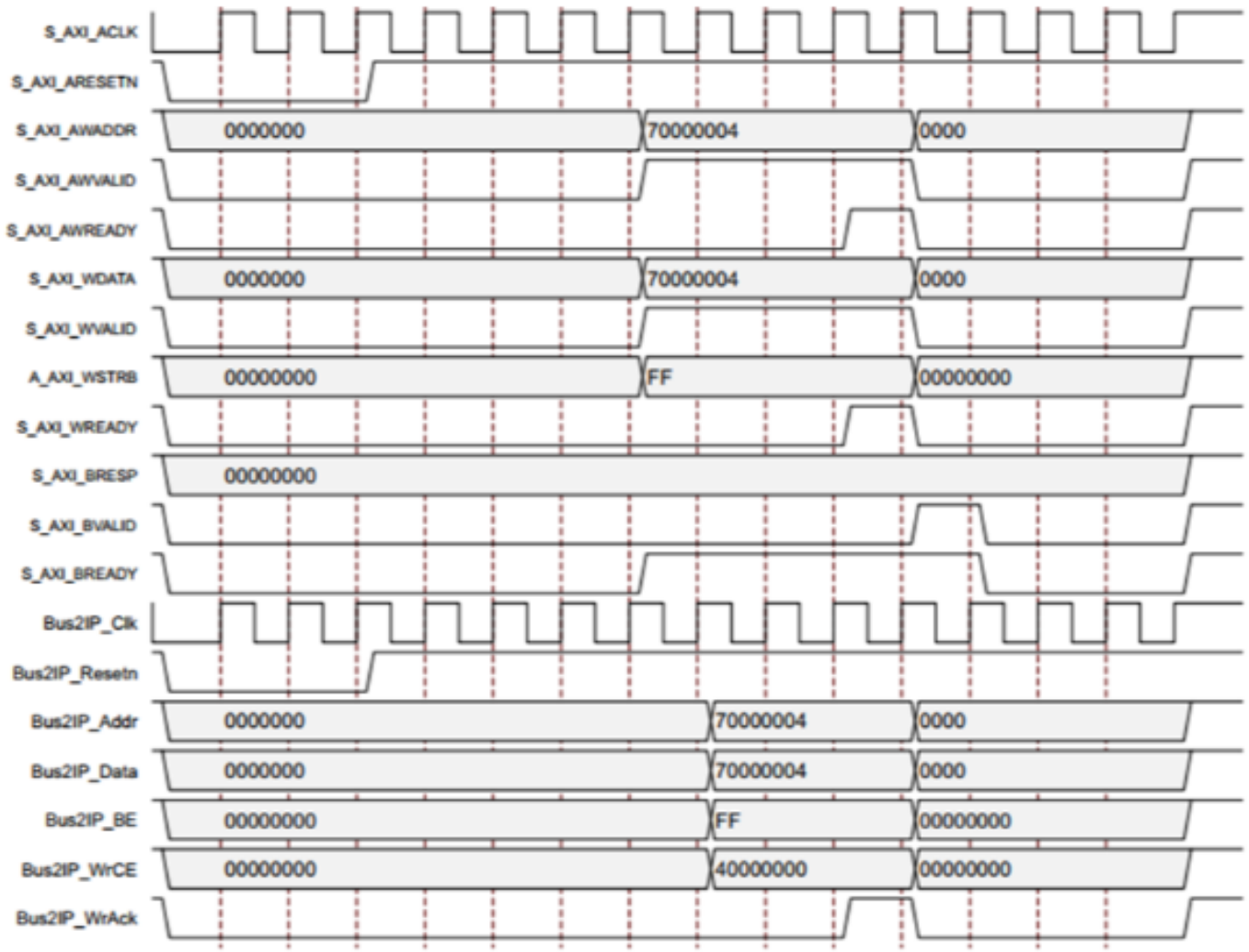
Global	Write address channel	Write data channel	Write response channel	Read address channel	Read data channel
ACLK	AWVALID	WVALID	BVALID	ARVALID	RVALID
ARESETn	AWREADY	WREADY	BREADY	ARREADY	RREADY
–	AWADDR	WDATA	BRESP	ARADDR	RDATA
–	AWPROT	WSTRB	–	ARPROT	RRESP

https://www.realdigital.org/doc/a9fee931f7a172423e1ba73f66ca4081

AXI4-Lite Read Transaction



AXI4-Lite Write Transaction



AXI4 Stream

https://developer.arm.com/documentation/ihi0051/latest/

Interface Signals

Table B-1 Interface signals

Signal	Width	Default	Property	AXI5-Stream	AXI4-Stream
ACLK	1	-	-	Y	Y
ARESETn	1	-	-	Y	Y
TVALID	1	-	-	Y	Y
TREADY	1	1	-	O	O
TDATA	TDATA_WIDTH	LOW	TDATA_WIDTH	C	O
TSTRB	TDATA_WIDTH/8	**TKEEP** or HIGH	-	O	O
TKEEP	TDATA_WIDTH/8	HIGH	-	O	O
TLAST	1	1	-	O	O
TID	TID_WIDTH	LOW	TID_WIDTH	C	O
TDEST	TDEST_WIDTH	LOW	TDEST_WIDTH	C	O
TUSER	TUSER_WIDTH	LOW	TUSER_WIDTH	C	O
TWAKEUP	1	-	Wakeup_Signal	C	N

Signal	Width	Property	AXI5-Stream	AXI4-Stream
TVALIDCHK	1	Check_Type	C	N
TREADYCHK	1	Check_Type	C	N
TDATACHK	TDATA_WIDTH/8	Check_Type & TDATA_WIDTH	C	N
TSTRBCHK	ceil(TDATA_WIDTH/64)	Check_Type	OC	N
TKEEPCHK	ceil(TDATA_WIDTH/64)	Check_Type	OC	N
TLASTCHK	1	Check_Type	OC	N
TIDCHK	ceil(TID_WIDTH/8)	Check_Type & TID_WIDTH	C	N

Signal	Width	Property	AXI5-Stream	AXI4-Stream
TDESTCHK	ceil(TDEST_WIDTH/8)	Check_Type & TDEST_WIDTH	C	N
TUSERCHK	ceil(TUSER_WIDTH/8)	Check_Type & TUSER_WIDTH	C	N
TWAKEUPCHK	1	Check_Type & Wakeup_Signal	C	N

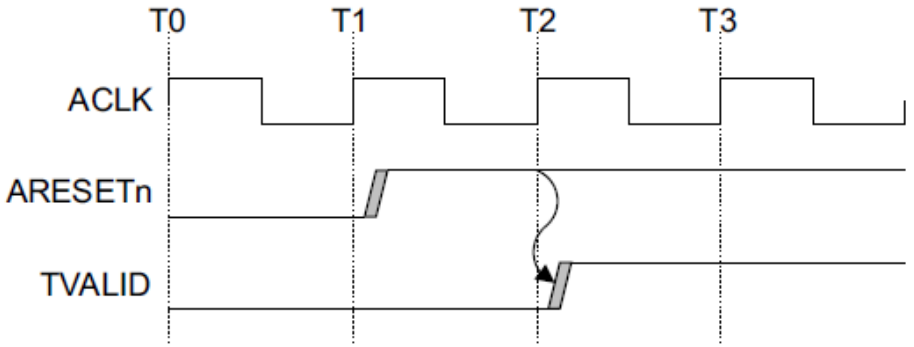
AXI5 Stream with additional features:

- Wake-up signaling
- Interface protection using parity

https://developer.arm.com/documentation/ihi0051/latest/

Clock and Reset

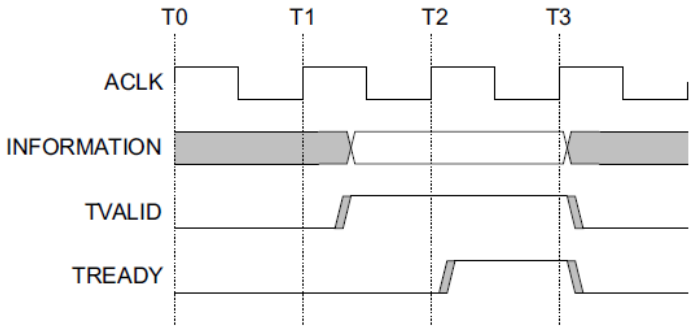
- **Clock - ACLK**
  - All input signals are sampled on the rising edge of ACLK
  - All output signal changes must occur after the rising edge of **ACLK**.
- **Reset - ARESETn**
  - **ARESETn** is a single, active-LOW reset signal. The reset signal can be asserted asynchronously, but deassertion must be synchronous after the rising edge of **ACLK**.
  - During reset, **TVALID** must be driven LOW. All other signals can be driven to any value.
  - A Transmitter interface must only begin driving **TVALID** at a rising **ACLK** edge following a rising edge at which **ARESETn** is asserted HIGH



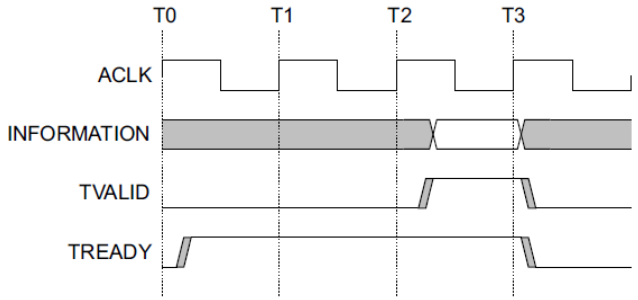
Data Transfer Handshake : TVALID, TREADY

- For a transfer to occur, both **TVALID** and **TREADY** must be asserted
- A Transmitter is not permitted to wait until **TREADY** is asserted before asserting **TVALID**
- Once **TVALID** isasserted, it must remain asserted until the handshake occurs
- A Receiver is permitted to wait for **TVALID** to be asserted before asserting **TREADY**

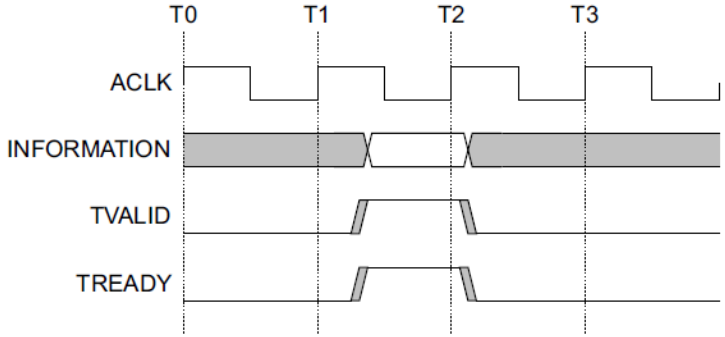
TVALID asserted before TREADY



TREADY asserted before TVALID



TVALID and TREADY asserted simultaneously



TKEEP & TSTRBE

TKEEP	TSTRB	Data Type	Description
HIGH	HIGH	Data byte	The associated byte contains valid information that must be transmitted between source and destination.
HIGH	LOW	Position byte	The associated byte indicates the relative position of the data bytes in a stream, but does not contain any relevant data values.
LOW	LOW	Null byte	The associated byte does not contain information and can be removed from the stream.
LOW	HIGH	Reserved	Must not be used.

Packet - TLAST

- An AXI-Stream packet is similar to an AXI burst.
- When deasserted, **TLAST** indicates that another transfer can follow. This means it is acceptable to delay the
- current transfer for the purpose of upsizing, downsizing, or merging.
- When asserted, **TLAST** can be used by a destination to indicate a packet boundary.
- When asserted, **TLAST** indicates an efficient point to make an arbitration change on a shared link.

TID & TDEST

- **TID** Provides a stream identifier that can be used to differentiate between multiple streams of data that are being transferred across the same interface.
- **TDEST** Provides coarse routing information for the data stream.
- Merging of transfers belonging to different streams (TID & TDEST diffes) is not permitted.
- Interleaving of transfers in different steams is permitted on a per transfer basis and is not limited to **TLAST** boundaries.

Protection by Parity