AXI bus 양영식

NXSOL-OJT-2024

Exported on 02/27/2024

Table of Contents

1 Introduction

AXI is designed to provide a high-performance communication interface between hardware components.

- **Terminology**: A transaction is a sequence of multiple operations performed as a single unit.

2 Key features of AXI

- **High Performance**: Supports high-speed data transfer and efficient communication between different parts of a system.
- **Separate Read and Write Channels**: It has independent channels for read and write operations, enhancing throughput and efficiency.
- **Separate Address(control) and Data channels**: It allows address information to be issued ahead of the actual data transfer.
- **Support for Burst Transfers**: AXI allows burst-based transfers, which is efficient for moving large blocks of data.
- **Out-of-Order Transaction**: It allows for transactions to complete in any order, not necessarily the order they were initiated.
- **Flexible and Scalable**: It's adaptable to various types of implementations and can be scaled according to the system requirements.
- **Support for Multiple Masters and Slaves**: AXI can handle communication between multiple master and slave devices, making it suitable for complex SoC (System on Chip) designs.

3 Architecture

The AXI protocol is designed to be burst-based and supports high-performance data transfers between master and slave components.

3.1 Channels

The AXI protocol defines five independent channels, each employing a `VALID` and `READY` handshake mechanism to ensure proper data transfers.

- **Master Devices**: Initiate transactions.
- **Slave Devices**: These are the responders to the transactions initiated by the masters.

Channels

- **Read Address and Write Address Channels**: Read and write transactions each have their own address channel. These carry the address and control information for their respective transactions.
- **Read Data Channel**: This channel conveys both the read data and read response from the slave to the master.
- **Write Data Channel**: This channel carries the write data from the master to the slave.
- **Write Response Channel**: Provides a way for the slave to respond to write transactions. All write transactions use completion signaling, which occurs once for each burst, not for each individual data transfer within the burst.

3.2 Separate address and data channel

Shows how a read transaction uses the read address and read data channels.

3.3 Figure 1-1

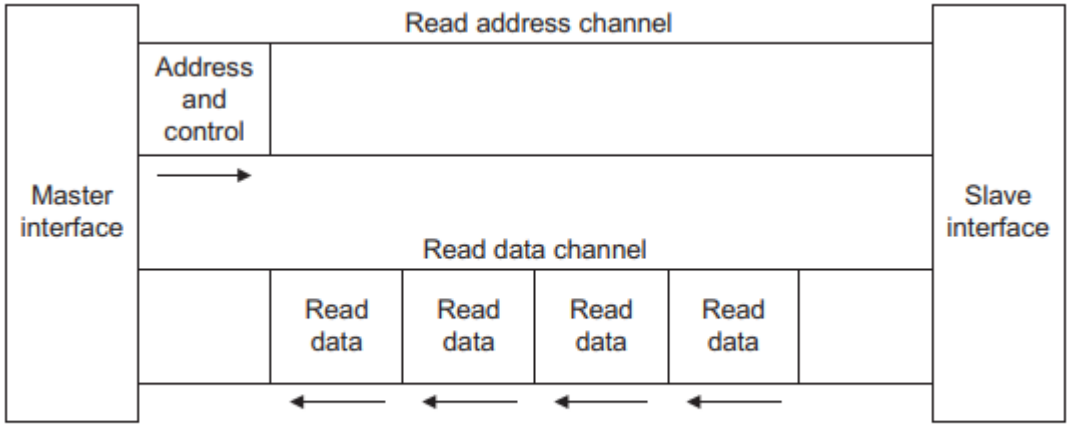


Figure 1-1 Channel architecture of reads

3.4 Figure 1-2

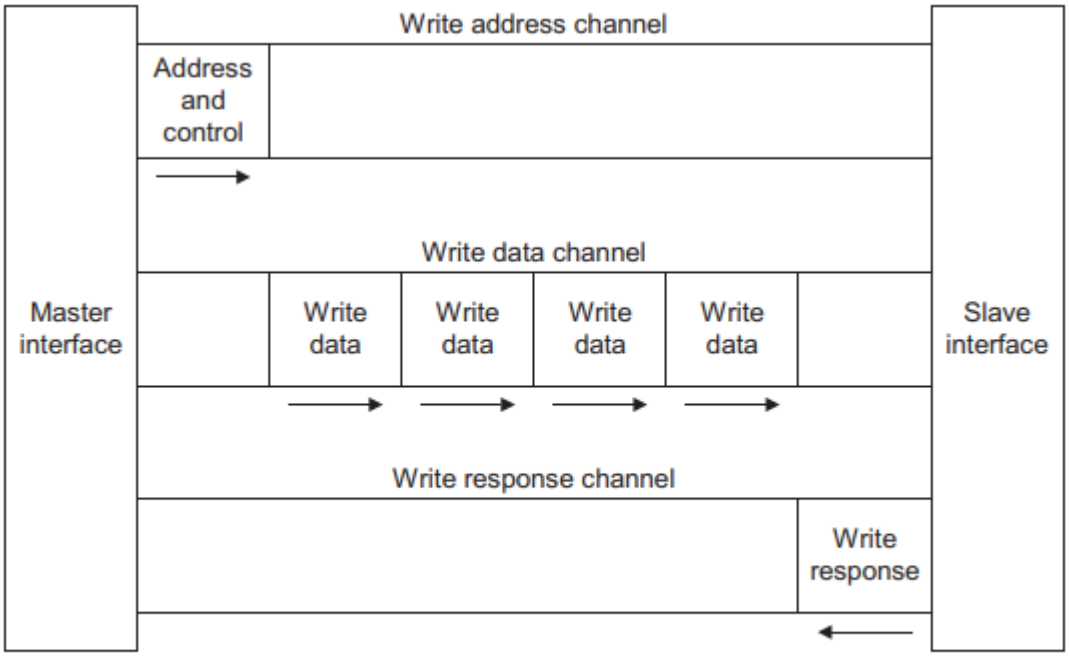


Figure 1-2 Channel architecture of writes

- **Mandatory Relationships**:

a. Read data must always follow the address to which the data relates.
b. Write response must always follow the last write transfer in the write transaction.

3.5 Interconnect

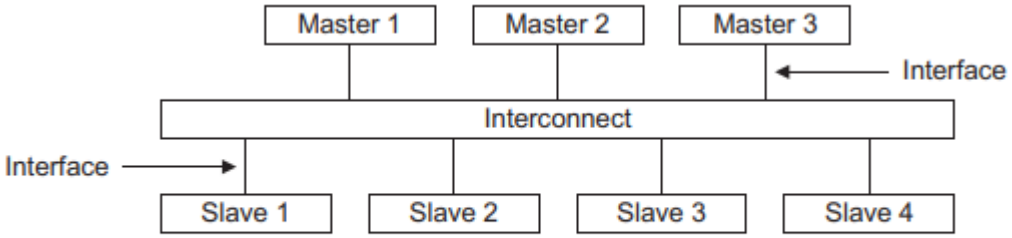


Figure 1-3 Interface and interconnect

- **Interconnect**: The interconnect allows communication between master and slave devices.

4 Channel Handshake

4.1 Handshake Process

It occurs on each burst

- **Purpose**: The handshake process in AXI is used to synchronize data and control information between the master and slave devices.(like TCP handshake)
- **Mechanism**: It uses a two-way VALID/READY flow control system to ensure data transfer occurs at an appropriate rate without loss or error.
- **Signals**:
    - **VALID**: Indicates data or control information is available from the source.
    - **READY**: Indicates the destination can accept the data or control information.
    - A **LAST** signal is also employed to denote the end of a burst of data transfers.
- **Conditions**: Both VALID and READY signals must be high for the data transfer to take place.

4.1.1 Timing Diagrams

VALID before READY Handshake:

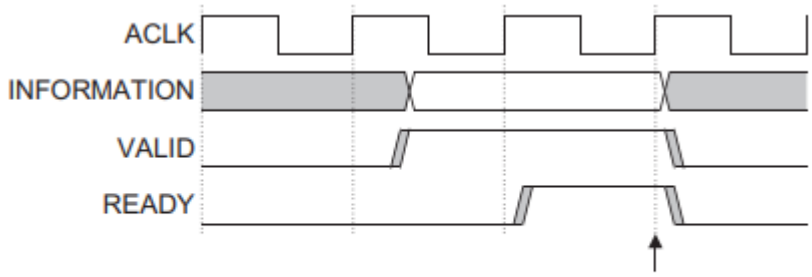


Figure 3-1 VALID before READY handshake

- The source sets the VALID signal high when data or control information is ready to be sent.
- The destination sets the READY signal high to accept the transfer.
- The transfer takes place when both signals are high.

READY before VALID Handshake:

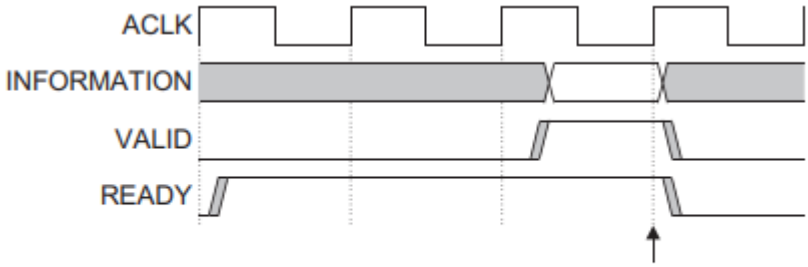


Figure 3-2 READY before VALID handshake

- The destination first indicates readiness to accept data by setting READY high.
- The source then presents data or control information, setting the VALID signal high.
- The transfer occurs when both the VALID and READY signals are high.

VALID with READY Handshake:

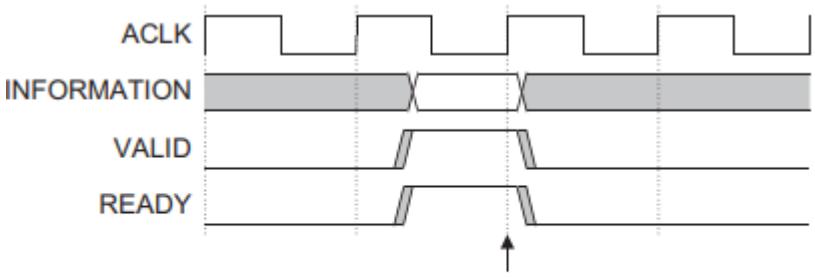


Figure 3-3 VALID with READY handshake

- The source and destination simultaneously indicate data availability and readiness to accept, respectively.
- Both VALID and READY signals go high in the same cycle, allowing immediate data transfer.

4.2 Dependencies between channel handshake signals

- **Avoiding Deadlocks**: It's crucial to observe the handshake signals' dependencies to prevent deadlocks.
- **Transaction Rules**:
  a. VALID signals should not be dependent on READY signals of other components in the transaction.
  b. The READY signal can wait for the VALID signal.

4.2.1 Read Transaction:

The starting end in the direction indicates the dependency(precondition)

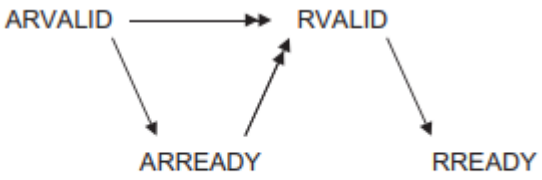


Figure 3-4 Read transaction handshake dependencies

- Slave waits for ARVALID before asserting ARREADY.
- Slave waits for both ARVALID and ARREADY before RVALID.

4.2.2 Write Transaction:

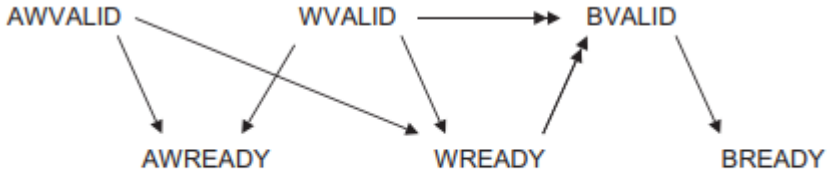


Figure 3-5 Write transaction handshake dependencies

- Master should not wait for AWREADY or WREADY before asserting AWVALID or WVALID.
- Slave waits for AWVALID or WVALID before asserting AWREADY.

5 Addressing Options

- **Responsibility**: The slave is responsible for calculating the addresses of subsequent transfers in the burst.
- **Boundary Limits**: Bursts should not cross 4KB boundaries to avoid crossing between slaves and to minimize the address incrementer within slaves.

5.1 Burst Length

- **Signals**: AWLEN or ARLEN signals indicate the number of data transfers in a burst, ranging from 1 to 16 transfers.
- **Encoding**: The table details how binary codes (b0000 to b1111) correspond to the number of data transfers.

Table 4-1 Burst length encoding

ARLEN[3:0] AWLEN[3:0]	Number of data transfers
b0000	1
b0001	2
b0010	3
. . .	
b1101	14
b1110	15
b1111	16

- **Transaction Rules**:
  - Each transaction must use the number of transfers specified by ARLEN or AWLEN without early termination.
  - The master can discard further writing and read data but it must complete the remaining transfers in the burst
- **Caution**: Avoid using burst lengths that exceed the requirements of FIFO-type devices to prevent data loss.

5.2 Burst Size

- **Signals**: ARSIZE and AWSIZE signals define the maximum number of bytes in each transfer within a burst.
- **Encoding**: The table outlines the binary codes (b0000 to b111) that correspond to bytes per transfer.

Table 4-2 Burst size encoding

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

- **Addressing**: The AXI determines which byte lanes of the data bus to use for each transfer based on the transfer address.
- **Data Transfer Consistency**: For incrementing or wrapping bursts with transfer sizes narrower than the data bus, data transfers are on different byte lanes for each beat of the burst. The address of a fixed burst remains constant, and every transfer uses the same byte lanes.
- **Transfer Size Limitations**: The size of any transfer must not exceed the data bus width of the components in the transaction.

5.3 Burst Type

Three types of bursts for data transfer according to how the address for each transfer is handled:

1. **Fixed Burst**: The address remains constant for every transfer in the burst, suitable for repeated access to the same location, like FIFO-type access.
2. **Incrementing Burst**: The address increments from the previous one. (Works like sequential memory access)
3. **Wrapping Burst**: The address increments from the previous one until it hits a wrap boundary(similar to incrementing burst), after which it wraps around to a lower address. (Like the circular queue)

Two restrictions apply to wrapping bursts:

- The start address must be aligned to the size of the transfer.
- The burst length must be a power of two, specifically 2, 4, 8, or 16.

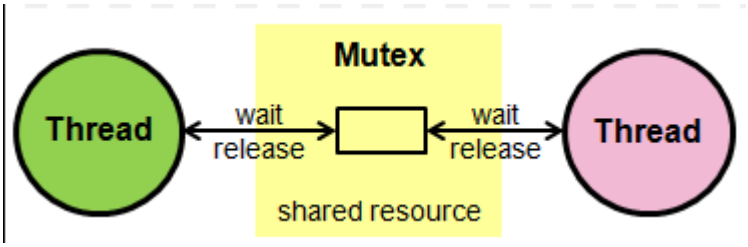
6 Atomic Accesses

Table 6-1 Atomic access encoding

ARLOCK[1:0] AWLOCK[1:0]	Access type
b00	Normal access
b01	Exclusive access
b10	Locked access
b11	Reserved

The ARLOCK[1:0] or AWLOCK[1:0] signal provides exclusive access and locked access.

6.1 Exclusive Access



Exclusive access notifies if a specific location has been altered by another master between a read and write operation by the initiating master. This semaphore type operation does not critically impact either access latency or the bandwidth

- Responses `BRESP[1:0]` or `RRESP[1:0]` indicate the success or failure of exclusive operations.
- A fail-safe mechanism is provided to indicate when a master attempts an exclusive access to a slave that does not support it.

6.1.1 Exclusive Access Process

1. A master performs an exclusive read from an address location.
2. At some later time, the master attempts to complete the exclusive operation by performing an exclusive write to the same address location.
3. The exclusive write access of the master is signalled as:

- **Successful** if no other master has written to that location between the read and write accesses.
- **Failed** if another master has written to that location between the read and write accesses. In this case the address location is not updated.

6.2 Locked Access

Locked access ensures a group of transactions occurs without any other access by other masters in between, where the bus interconnect locks the bus to the master that initiated the locked transaction.

Key Points:

1. **Signal for Locked Transfer:**
   - When `ARLOCK[1:0]` or `AWLOCK[1:0]` signals indicate a locked transfer, it is the interconnect's responsibility to ensure exclusive access to the initiating master until an unlocked transfer completes.
2. **Starting a Locked Sequence:**
   - A master initiating a locked sequence must not have other outstanding transactions pending.
3. **Completing a Locked Sequence:**
   - All previous locked transactions must be completed before issuing the final unlocking transaction. The final unlocking transaction is necessary before any further transactions begi

Notes:

- Locked accesses can affect interconnect performance as they prevent other transactions during the sequence.
- It is advised to use locked accesses only for legacy devices due to their impact

7 Response Signaling

The AXI protocol supports response signaling for both read and write operations.

Responses Defined:

- **OKAY(b00)**: Indicates a normal access has been successful. It can also signify the absence of an exclusive access failure.
- **EXOKAY(b01)**: Marks that an exclusive access read or write operation was successful.
- **SLVERR(b10)**: Signifies a successful access to the slave, but the slave cannot perform the requested action and is returning an error.
- **DECERR(b11)**: Indicates a decode error, which occurs typically when no slave is present at the specified transaction address.

7.1 Response channel:

- **When data is read**, the response from the slave (the peripheral or memory being accessed) is sent along with the data itself.

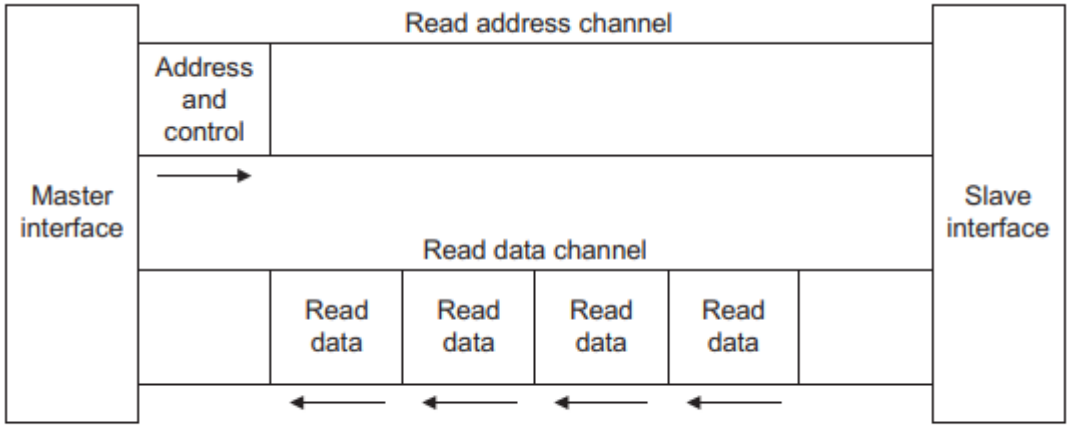


Figure 1-1 Channel architecture of reads

- **For write operations**, the response is sent on a separate write response channel.

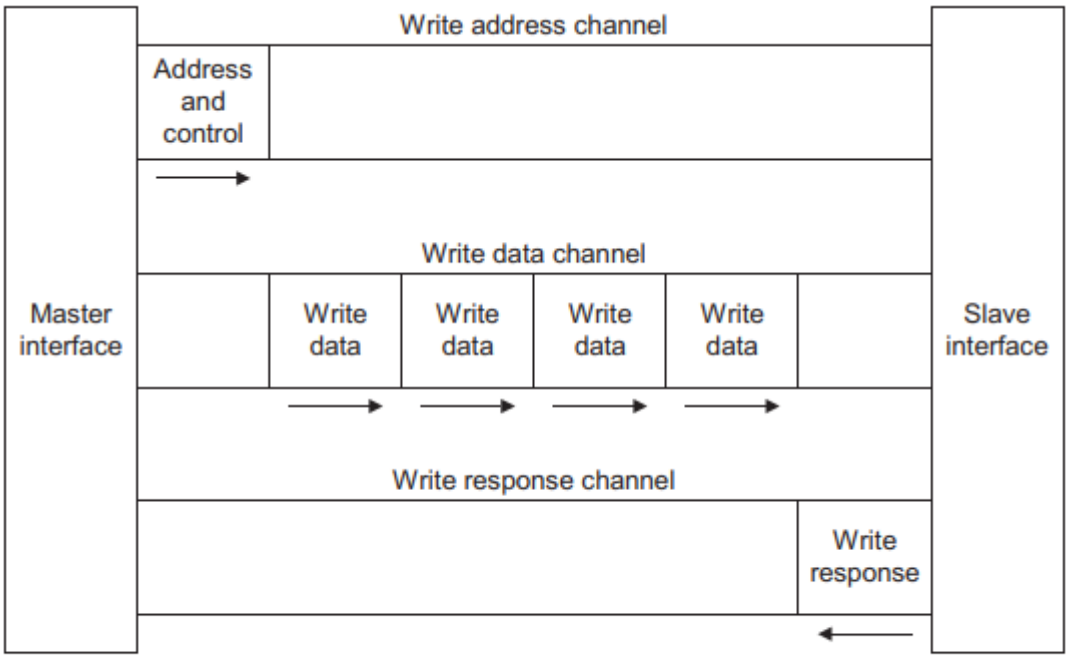


Figure 1-2 Channel architecture of writes

7.1.1 Number of responses

- **For write operations**, there's only one response for the entire burst of data transfers, not for each transfer within the burst.
- **For read transactions**, the slave can signal different responses for each data transfer within a burst.

8 Ordering Model

8.1 About the Ordering Model (Similar to pipeline out of order)

The AXI protocol allows for out-of-order completion and issuing of multiple outstanding addresses. This flexibility optimizes data throughput and system efficiency by allowing:

- **Out-of-Order Transaction Completion**: Enables transactions to faster memory regions to complete without having to wait for transactions to slower memory regions, thus reducing the effect of transaction latency.
- **Multiple Outstanding Addresses**: Masters can issue transaction addresses without waiting for earlier transactions to complete, allowing for parallel processing and transactions.

8.2 Transfer ID Fields

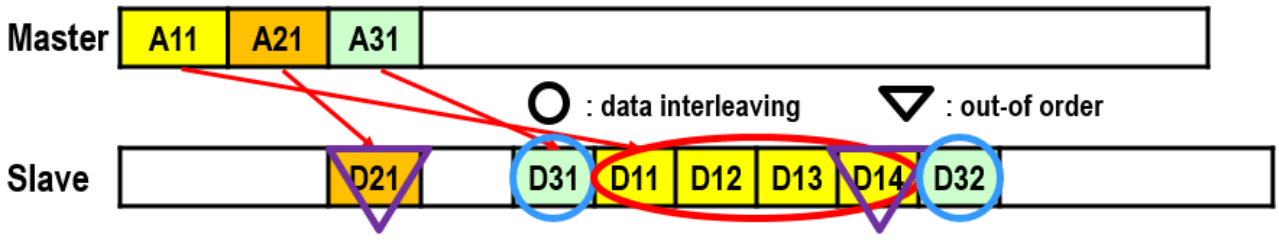
The AXI protocol uses ID fields to manage multiple transactions and maintain order:

- **ARID/AWID Field**: Specifies additional ordering requirements from the master.

Ordering Rules:

- Transactions from different masters have no ordering restrictions.
- Transactions with different IDs from the same master can complete in any order.
- Write transactions with the same AWID must complete in the same order as the master issued the addresses.
- Like the write transactions, read transactions with the same ARID must return data in the same order as addresses were issued.

8.3 Data Interleaving



Write data interleaving allows a slave to handle interleaved write data with different AWID values. This facilitates the simultaneous processing of write data from multiple sources, which can enhance system performance by allowing data from slower sources to be interspersed with data from faster ones.

- **Write Data Interleaving Depth**: This is a static value set by the designer that indicates how many different addresses can have their write data interleaved in the slave interface. A depth of 1 means no interleaving is possible, while a higher value allows for multiple interleaved transactions.
- **AWID as a single unit**: Interleaving is not allowed between transactions with the same AWID. However, data with different AWID values can be interleaved.
- **Avoiding Deadlocks**: To prevent deadlocks, a slave interface with an interleaving depth greater than one must be able to continuously accept interleaved data without stalling.

9 Data buses

9.1 About the data buses

- Each data bus has its own set of handshake signals, which allows for simultaneous data transfers on both buses.
- **Data width not over bus width**: Every transfer generated by a master must be the same width as or narrower than the data bus for the transfer.

9.2 Write strobes

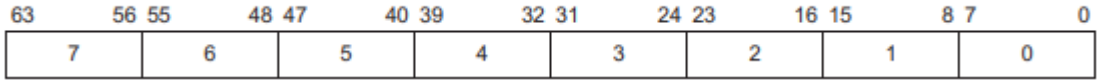


Figure 9-1 Byte lane mapping

Each write strobe signal, `WSTRB` , corresponds to one byte of the write data bus.

When asserted, a write strobe indicates that the corresponding byte lane of the data bus contains valid information to be updated in memory, which facilitates selective data transfer.

9.3 Narrow Transfers

When a master generates **a transfer that is narrower than its data bus**, the address and control information determine which byte lanes the transfer uses.

- **In incrementing or wrapping burst modes**, different byte lanes may be used for each beat of the burst.
- **In a fixed burst**, the address remains constant, and the byte lanes that can be used also remain constant.

9.3.1 Example of Byte Lanes Use

In Figure 9-2:

- the burst has five transfers
- the starting address is 0
- each transfer is eight bits
- the transfers are on a 32-bit bus.

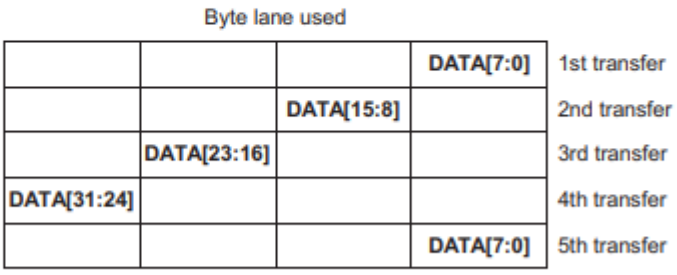


Figure 9-2 Narrow transfer example with 8-bit transfers

9.4 Byte Invariance

Byte invariance refers to a scheme that allows for access to mixed-endian data structures within the same memory space.

Byte-invariant endianness means that a byte transfer to a given address passes the 8 bits of data on the same data bus wires to the same address location.

Endian Compatibility:

- Little-endian components can usually connect directly to a byte-invariant interface.
- Big-endian components require a conversion function for byte-invariant operations.

Example:

Figure 9-4 illustrates a data structure requiring byte-invariant access. Header information like source and destination identifiers may be in little-endian, whereas the payload is in big-endian format.

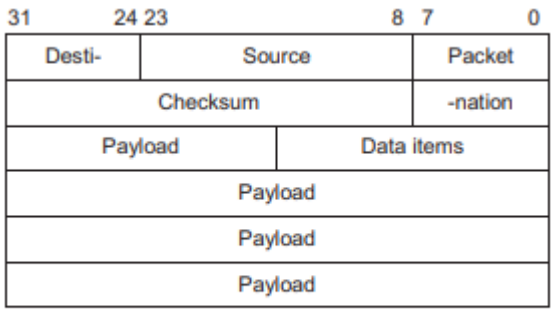


Figure 9-4 Example mixed-endian data structure

Importance of Byte Invariance:

Ensures that little-endian access to parts of the header does not corrupt the big-endian data within the structure.

10 Unaligned Transfer

10.1 About unaligned transfers

10.1.1 Alignment

- **Typically aligned**: Typically, each data transfer is aligned to the size of the transfer. For example, a 32-bit wide transfer is usually aligned to four-byte boundaries. However, there are times when it is desirable to begin a burst at an unaligned address.
- **Sometimes unaligned**: For any burst that is made up of data transfers wider than one byte, it is possible that the first bytes accessed do not align with the natural data width boundary. For example, a 32-bit (four-byte) data packet that starts at a byte address of 0x1002 is not aligned to a 32-bit boundary.

Note on Slave Actions:

- The protocol does not require the slave to take special actions based on alignment information provided by the master.

10.1.2 Example

These are examples of aligned and unaligned transfers on buses with different widths.

- Each row in the figures represents a transfer.
- The shaded cells indicate bytes that are not transferred, based on the address and control information.

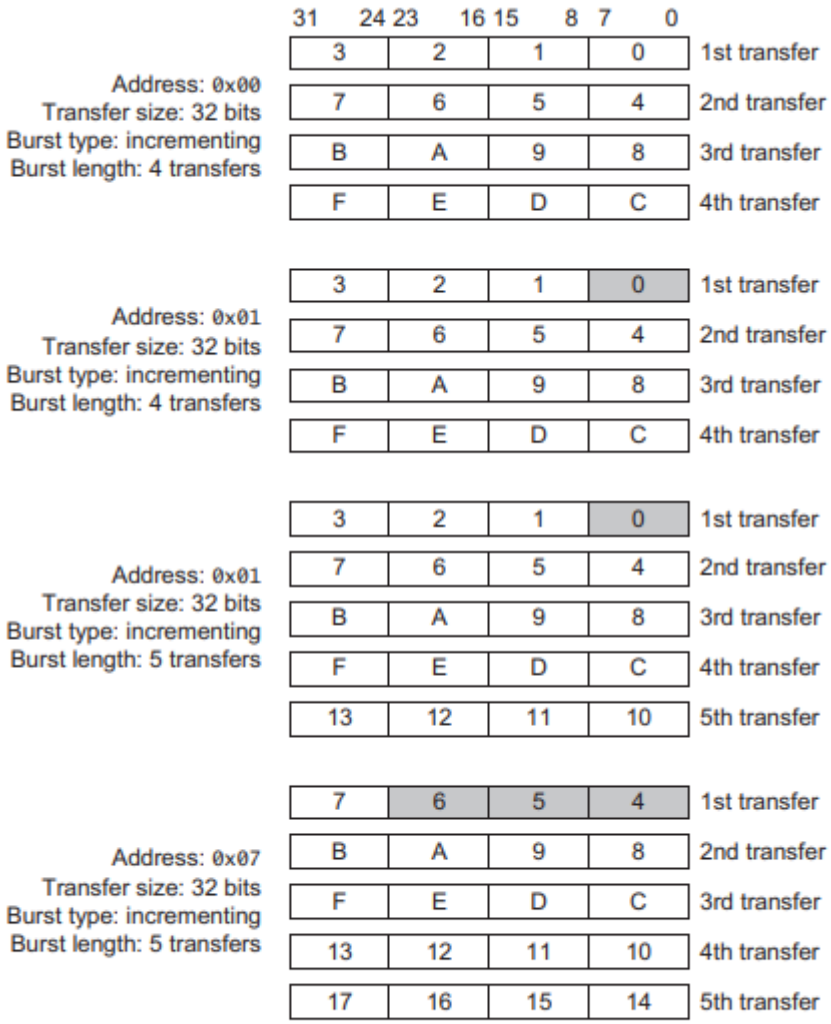


Figure 10-1 Aligned and unaligned word transfers on a 32-bit bus

- First burst: **aligned** (Transfer size: 4 bytes, Address: 0x00)
- Second burst: **unaligned** (Transfer size: 4 bytes, Address: 0x01)

11 Clock and Reset

11.1 Clock and Reset

- Requirements for implementing the ACLKn and ARESETn signals.

11.1.1 Clock

- Each AXI component uses a single clock signal, ACLKn.
- All input signals are sampled on the rising edge of ACLKn.
- All output signal changes must occur after the rising edge of ACLKn.

11.1.2 Reset

- AXI protocol includes a single **active LOW** reset signal, ARESETn.
- ARESETn can be asserted asynchronously, but deassertion must be synchronous after the rising edge of ACLKn.
- Interface requirements during reset:
    - A master interface must drive ARVALID, AWVALID, and WVALID *LOW*.
    - A slave interface must drive RVALID and BVALID *LOW*.

11.1.3 Exit from reset

A master interface must begin driving ARVALID, AWVALID, or WVALID HIGH only at a rising ACLK edge after ARESETn is HIGH

Figure 11-1 shows the first point after reset that VALID can be driven HIGH.

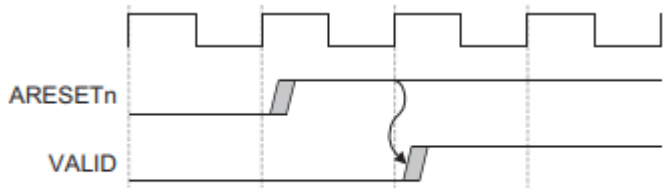


Figure 11-1 Exit from reset

12 Low-power Interface

12.1 About the Low-Power Interface

The low-power interface is an optional extension to the data transfer protocol that targets two different classes of peripherals:

- **Peripherals Requiring Power-Down Sequence**:
  - These peripherals need to enter a low-power state before their clocks can be turned off.
  - They require a signal from the system clock controller to start the power-down sequence.
- **Peripherals Without Power-Down Sequence**:
  - These can independently indicate when their clocks can be turned off without a sequence.

12.2 Low-power clock control

Overview:

- The low-power clock control interface consists of signals from the peripheral and the system clock controller for managing the clock in low-power states.

Signals:

- **CACTIVE:** Signal from the peripheral indicating that it requires its clock to be either enabled(HIGH) or disabled(LOW).
- **CSYSREQ(request):** Handshake signal for the system clock controller to request entry(LOW) into or exit(HIGH) from a low-power state.
- **CSYSACK(acknowledge):** Handshake signal(LOW) for the peripheral to acknowledge the low-power state request or the exit from it.

12.2.1 Example

Figure 12-1 shows the relationship between **CSYSREQ** and **CSYSACK**.

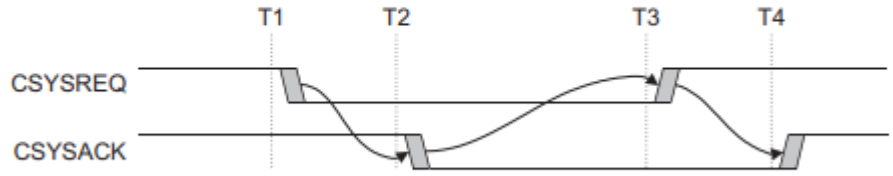


Figure 12-1 CSYSREQ and CSYSACK handshake

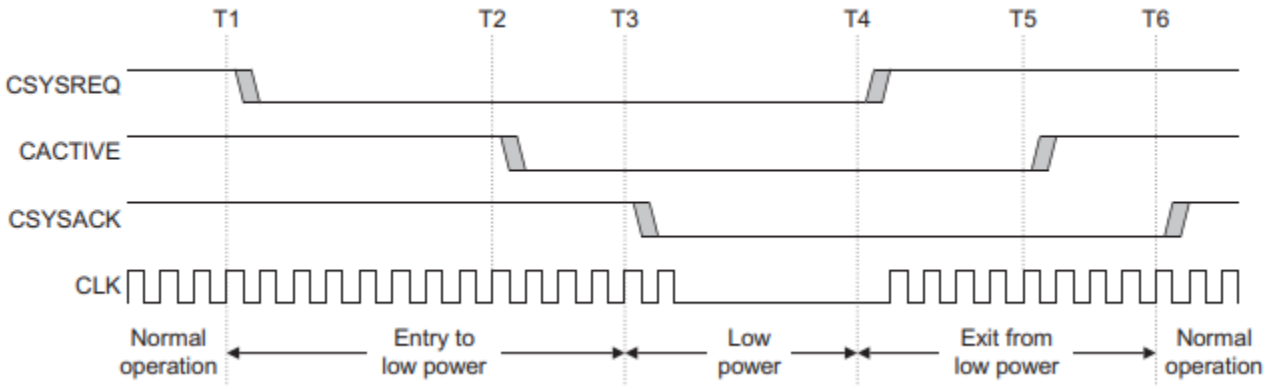
- **T1:** Request to put the peripheral in a low-power state
- **T2**: Request acknowledged
- **T3**: Exit from the low-power state
- **T4**: Exit acknowledged

12.2.2 Acceptance of low-power request

Handshake Mechanism:

- The sequence of events when a peripheral accepts a system low-power request is outlined in Figure 12-2.

Example:



- **T1:** Request to put the peripheral in a low-power state
- **T2-T3:** The peripheral deasserts CACTIVE as it performs its power-down function and acknowledges by deasserting CSYSACK
- **T4:** Exit from the low-power state
- **T5-T6:** The peripheral asserts CACTIVE and completes the exit sequence by asserting CSYSACK.

12.2.3 Denial of a low-power request

- A low-power request is denied by the peripheral by keeping CACTIVE HIGH when acknowledging the request with CSYSACK.

Example:

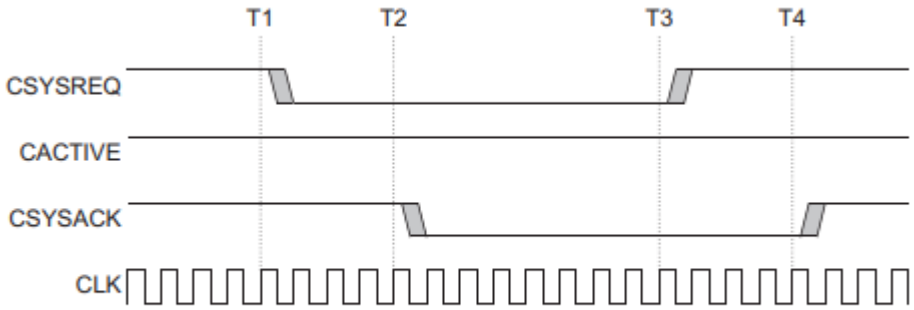


Figure 12-3 Denial of a low-power request

- **T1:** Request to enter a low-power state.
- **T2:** Peripheral denies the request by keeping CACTIVE HIGH when it acknowledges the request.
- **T3-T4:** CSYSREQ is asserted to complete the sequence. The handshake must be completed by asserting CSYSREQ before another request can be initiated.

12.2.4 Clock control sequence summary

Figure 12-4 shows the typical flow for entering and exiting a low-power state.

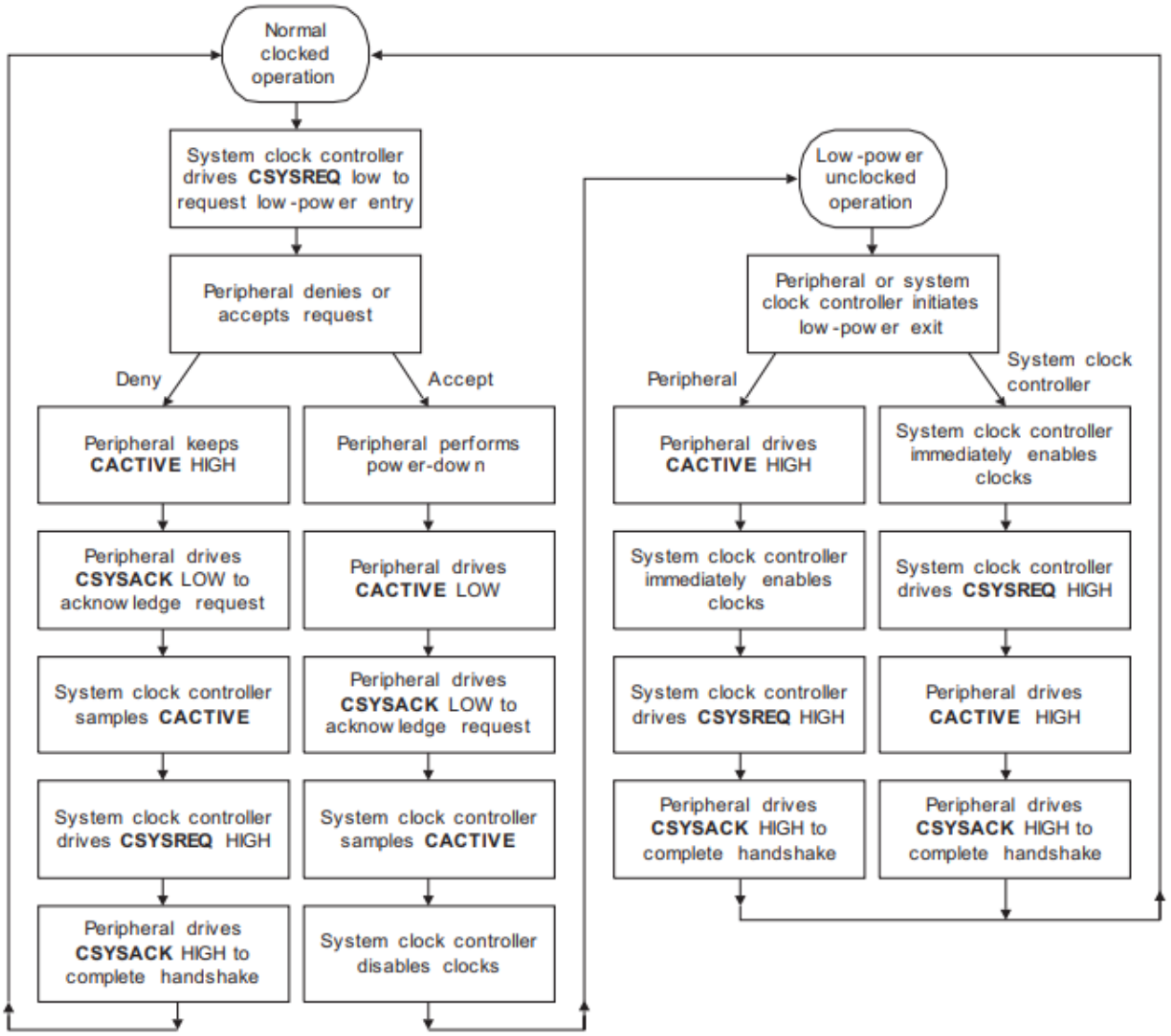


Figure 12-4 Low-power clock control sequence