AXI bus

NXSOL-OJT-2024

Exported on 2024-01-18 16:27:02

Table of Contents

1 Introduction 4

1.1 Key features of AXI 4

1.2 1.2 Architecture 4

1.2.1 Figure 1-1 4

1.2.2 Shows how a read transaction uses the read address and read data channels.(Separate address and data channel) 5

1.2.3 Figure 1-2 5

1.2.4 1.2.1 Channel Definition 5

1.2.5 1.2.2 Interface and Interconnect 6

2 Channel Handshake 7

2.1 3.1 Handshake Process 7

2.1.1 (Details) 7

2.2 3.2 Relationships between the channels 8

2.3 3.3 Dependencies between channel handshake signals 8

2.3.1 Handshake Signal Dependencies: 8

3 Addressing Options 10

3.1 4.1 About Addressing Options 10

3.2 4.2 Burst Length 10

3.3 4.3 Burst Size 10

3.4 4.4 Burst Type 11

4 Additional Control Information 12

4.1 5.1 Cache Support 12

4.2 5.2 Protection unit support 12

5 Atomic Accesses 14

5.1 6.1 About Atomic Accesses 14

5.2 6.2 Exclusive Access 14

5.2.1 6.2.1 Exclusive Access Process 14

5.2.2 6.2.3 Exclusive Access from the Perspective of the Slave 14

5.3 6.3 Locked Access 15

6 Response Signaling 16

6.1 Response channel: 16

6.2 Number of responses 16

7 Ordering Model 17

7.1 8.1 About the Ordering Model (Similar to pipeline out of order) 17

7.2 8.2 Transfer ID Fields 17

7.3 8.3 Read Ordering 17

7.4 8.4 Normal Write Ordering 18

7.5 8.5 Write Data Interleaving 18

7.6 8.6 Read and Write Interaction 18

7.7 8.7 Interconnect Use of ID Fields 19

7.8 8.8 Recommended Width of ID Fields(Details) 19

8 Data buses 20

8.1 9.1 About the data buses 20

8.2 9.2 Write strobes 20

8.3 9.3 Narrow Transfers 20

8.3.1 Example of Byte Lanes Use 20

8.4 9.4 Byte Invariance 21

9 10 Unaligned Transfer 22

9.1 10.1 About unaligned transfers 22

9.1.1 Alignment 22

9.1.2 Example 22

10 Clock and Reset 23

10.1 11.1 Clock and Reset 23

10.1.1 11.1.1 Clock 23

10.1.2 11.1.2 Reset 23

10.1.3 Figure 11-1 Exit from reset 23

11 Low-power Interface 24

11.1 12.1 About the Low-Power Interface 24

11.2 12.2 Low-power clock control 24

11.2.1 Example 24

11.2.2 12.2.1 Acceptance of low-power request 25

11.2.3 12.2.2 Denial of a low-power request 25

# 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.

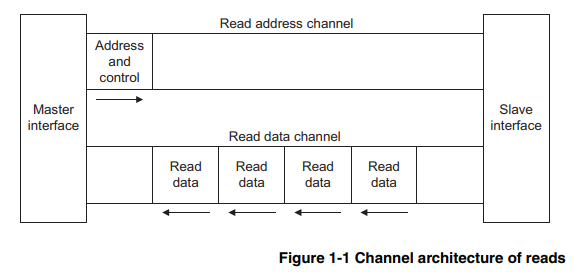
## Key features of AXI

* **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.

## 1.2 Architecture

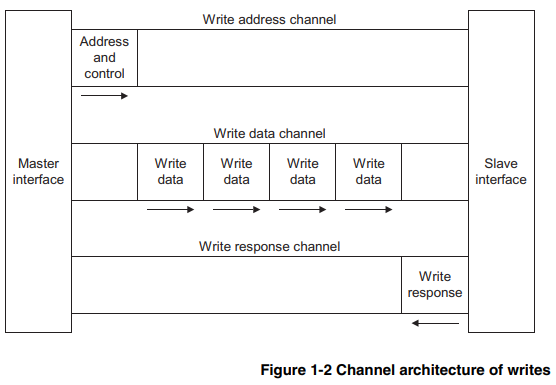
The AXI protocol is designed to be burst-based and supports high-performance data transfers between master and slave components.

### Figure 1-1



### Shows how a read transaction uses the read address and read data channels.(Separate address and data channel)

### Figure 1-2



Shows how a write transaction uses the write address, write data, and write response channels.

### 1.2.1 Channel Definition

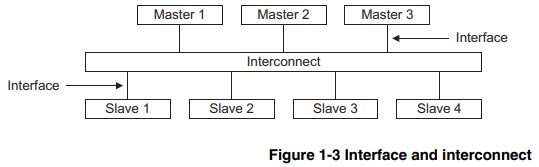
The AXI protocol defines five independent channels, each employing a VALID and READY handshake mechanism to ensure proper data transfers.

* The source(sender) uses the VALID signal to show when valid data or control information is available on the channel.
* The destination(recipient) uses the READY signal to show when it is prepared to accept the data.
* A **LAST** signal is also employed to denote the end of a burst of data transfers.
* **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.

Each channel supports different operations, such as variable-length bursts, transfers of various sizes (from 8 to 1024 bits), and system-level caching and buffering control. These features contribute to the flexibility and efficiency of the AXI protocol.

### 1.2.2 Interface and Interconnect

#### Figure 1-3



Illustrates the structure of a typical system that includes multiple master and slave devices connected through an interconnect.

* **Master Devices**: Initiate transactions.
* **Slave Devices**: These are the responders to the transactions initiated by the masters.
* **Interconnect**: The interconnect allows communication between master and slave devices.

#### 1.2.3 Register Slices

* **Unidirectional Information Transfer**: Each AXI channel is designed to transfer information in only one direction.
* **Register Slices**: These can be inserted into any channel to help with timing and isolation, potentially adding a cycle of latency but allowing for a **trade-off** between latency(register slice) and maximum frequency of operation(direct, fast).

# Channel Handshake

## 3.1 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.
* **Conditions**: Both VALID and READY signals must be high for the data transfer to take place.

**Timing Diagrams**

1. **VALID before READY Handshake (Figure 3-1)**:
   * 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.
2. **READY before VALID Handshake (Figure 3-2)**:
   * 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.
3. **VALID with READY Handshake (Figure 3-3)**:
   * 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.

### (Details)

**3.1.1 Write Address Channel**

* **AWVALID**: Asserted by the master to indicate a valid write address and control information are being driven.
* **AWREADY**: Slave's signal to indicate readiness to accept the write address.

*Defaults:*

* **AWREADY**: Can be HIGH or LOW, but HIGH is recommended unless a specific case dictates otherwise.

**3.1.2 Write Data Channel**

* **WVALID**: Asserted by the master to signal valid write data is on the bus.
* **WREADY**: Slave's signal to indicate readiness to accept write data.
* **WLAST**: Asserted with the final write transfer in a burst to indicate the end of the burst.

*Defaults:*

* **WREADY**: Recommended to be HIGH if the slave can accept write data in a single cycle.

**3.1.3 Write Response Channel**

* **BVALID**: Asserted by the slave when it drives a valid write response.
* **BREADY**: Master's acknowledgment signal, indicating it can accept the write response.

*Defaults:*

* **BREADY**: Should be HIGH if the master can accept a write response in a single cycle.

**3.1.4 Read Address Channel**

* **ARVALID**: Asserted by the master when it drives a valid read address and control information.
* **ARREADY**: Slave's signal to indicate readiness to accept the read address.

*Defaults:*

* **ARREADY**: Recommended to be HIGH to ensure the master can proceed without delay.

## 3.2 Relationships between the channels

* **Flexibility of Channels**: The AXI protocol defines separate channels for address, read, write, and write response that can operate independently for flexibility.
* **Write Data and Address Synchronization**: Write data can appear at the interface before the write address related to it. However, when the interconnect must determine the destination, it realigns the address and write data to ensure the correct slave device receives the data.
* **Mandatory Relationships**:
  1. Read data must always follow the address to which the data relates.
  2. Write response must always follow the last write transfer in the write transaction.

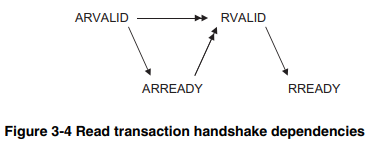
## 3.3 Dependencies between channel handshake signals

* **Avoiding Deadlocks**: It's crucial to observe the handshake signals' dependencies to prevent deadlocks.
* **Transaction Rules**:
  1. VALID signals should not be dependent on READY signals of other components in the transaction.
  2. The READY signal can wait for the VALID signal.
* **Efficient Design Note**: It's acceptable and often more efficient to assert READY by default, before VALID.

### Handshake Signal Dependencies:

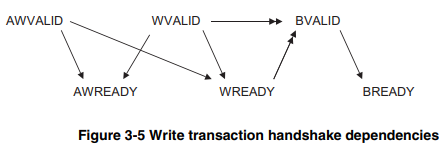
The starting end in the direction indicates the dependency(precondition)

#### Read Transaction:



* Slave waits for ARVALID before asserting ARREADY.
* Slave waits for both ARVALID and ARREADY before returning data with RVALID.

#### Write Transaction:



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

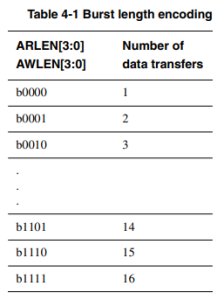
# Addressing Options

## 4.1 About Addressing Options

* **AXI Protocol**: A burst-based communication protocol where the master initiates each burst by providing control information and the first byte's address.
* **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.

## 4.2 Burst Length

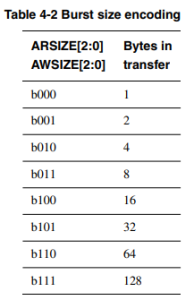
* **Signals**: AWLEN or ARLEN signals indicate the number of data transfers in a burst, ranging from 1 to 16 transfers.
* **Encoding**: The table details how binary codes (b0000 to b1111) correspond to the number of data transfers.



* **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.

## 4.3 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.



* **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.

## 4.4 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.

# Additional Control Information

## 5.1 Cache Support

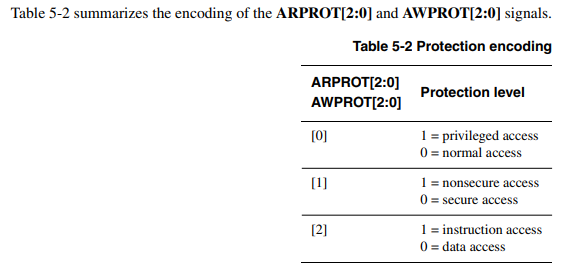
The ARCACHE[3:0] or AWCACHE[3:0] signal supports system-level caches by providing the bufferable, cacheable, and allocate attributes of the transaction:

* **Bufferable (B) bit(ARCACHE[0] and AWCACHE[0])**:
  + When high, allows delay in transaction reaching its final destination for writes, meaning that the data can be held in a buffer before reaching its final destination.
* **Cacheable (C) bit(ARCACHE[1] and AWCACHE[1])**:
  + When high, the transaction at the final destination doesn't have to match original transaction characteristics. (Because when a transaction is cacheable, the system can perform various caching operations (like reordering, combining writes, etc.) which may modify the characteristics.)
* **Read Allocate (RA) bit(ARCACHE[2] and AWCACHE[2])**:
  + When high, allows read transactions to allocate cache lines if they miss in the cache.
  + RA should not be high if C is low.
* **Write Allocate (WA) bit(ARCACHE[3] and AWCACHE[3])**:
  + When high, allows write transactions to allocate cache lines if they miss in the cache.
  + WA should not be high if C is low.

## 5.2 Protection unit support

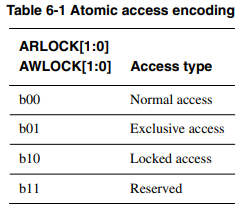
The AXI bus system includes a protection unit to manage access rights during transactions. This is done through the use of the ARPROT and AWPROT signals, which provide three levels of access protection:

1. **Normal or Privileged Access (ARPROT[0] and AWPROT[0])**
   * LOW indicates a normal access.
   * HIGH indicates a privileged access, which allows for a greater level of access within a system.
2. **Secure or Non-Secure Access (ARPROT[1] and AWPROT[1])**
   * LOW indicates a secure access.
   * HIGH indicates a non-secure access, allowing for differentiation between processing modes.
3. **Instruction or Data Access (ARPROT[2] and AWPROT[2])**
   * LOW indicates a data access.
   * This bit gives an indication if the transaction is an instruction or a data access.



# Atomic Accesses

## 6.1 About Atomic Accesses



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

## 6.2 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.

### 6.2.1 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.

### 6.2.3 Exclusive Access from the Perspective of the Slave

* Slaves not supporting exclusive access must always respond with OKAY to both normal and exclusive accesses.
* Slaves that do support exclusive access must have monitoring hardware to track exclusive read and write operations.

## 6.3 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 begin.
4. **Consistency in Locked Sequences:**
   * All transactions within a locked sequence must share the same ARID or AWID values to maintain consistency.

**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

# 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.

## Response channel:

* **When data is read**, the response from the slave (the peripheral or memory being accessed) is sent along with the data itself.
* **For write operations**, the response is sent on a separate write response channel.

## 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.

# 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.

**Transaction IDs (IDs):**

All transactions with a given ID must be ordered, but there is no restriction on the ordering of transactions with different IDs.

* **AWID**: Write address ID tag.
* **WID**: Write transaction ID tag.
* **BID**: Write response ID tag.
* **ARID**: Read address ID tag.
* **RID**: Read transaction ID tag.

## 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 Read Ordering

The AXI protocol ensures that read operations are ordered according to the ARID (Address Read ID):

* Read transactions with the same ARID must have their data returned in the same order as the addresses were issued.
* If data is received from different ARID values, it can be interleaved or returned in any order.
* The RID (Read ID) of returned data must match the ARID of the address it responds to.
* Sequences of read transactions with the same ARID from different slaves must complete in the order issued.
* The read data reordering depth, a static value determined by the slave designer, is the number of addresses that can be reordered within the slave.

## 8.4 Normal Write Ordering

For write operations, the following principles apply according to the AXI protocol:

* Most slave designs do not support write data interleaving and consequently these types of slave design must receive write data in the same order that they receive the addresses.
* If an interconnect combines write transactions from different masters to one slave, it must ensure the data is combined in address order.

These ordering rules apply even if the write transactions have different AWID values.

## 8.5 Write 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.

Key points about write data interleaving:

* **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.
* **Order of Data(=address)**: The order in which data items are received should match the order of address reception.
* **AWID as a single unit**: Interleaving is not allowed between transactions with the same AWID. However, data with different AWID values can be interleaved.
* **Different AWIDs accessing overlapping addresses**: When transactions with different AWID values access the same or overlapping address locations, there isn't a defined processing order. Higher-level protocols must ensure correct processing.
* **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.

## 8.6 Read and Write Interaction

* No ordering restrictions exist between read and write transactions; they can complete in any order unless a specific relationship is required by the master.
* For peripherals, transactions typically need to wait for earlier transactions to complete when switching between reads and writes.
* Masters can implement checks against outstanding transactions to determine if a new transaction could be to the overlapping, address region

## 8.7 Interconnect Use of ID Fields

* Interconnects append additional bits to the ARID, AWID, and WID that are unique to the master port, which:
  + Allows masters to be unaware of other masters' ID values.
  + Makes ID width at the slave interface wider than at the master interface.
* For read data, the interconnect manipulates the RID field to route the data to the correct master port.

## 8.8 Recommended Width of ID Fields(Details)

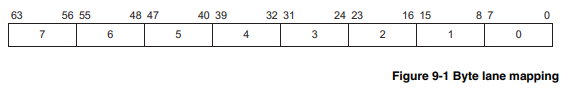
* To leverage AXI's out-of-order transaction capabilities, the recommendation is:
  + Implement up to 4 bits of transaction ID in master components.
  + Allow up to 4 additional bits for master port numbers in the interconnect.
  + Support 8 bits of ID in slave components.
* If a master only supports a single ordered interface, it's acceptable to set the ID to a constant value, like 0.

# Data buses

## 9.1 About the data buses

* The AXI (Advanced eXtensible Interface) protocol features **two** distinct data buses: one dedicated to read operations and the other to write operations.
* 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



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

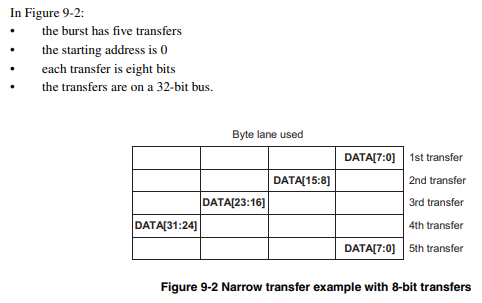
When asserted, a write strobe indicates that the corresponding byte lane of the data bus contains valid information to be updated in memory, which facilitates selective data transfer.

## 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.

### Example of Byte Lanes Use



## 9.4 Byte Invariance

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

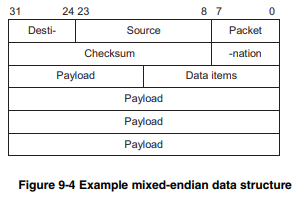
Byte-invariant endianness means that a byte transfer to a given address passes the 8 bits of data on the same data bus wires to the same address location.

**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.



**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

### 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.

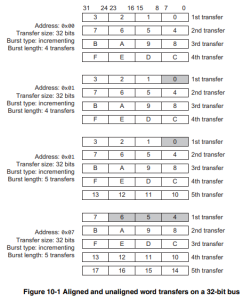
**Note on Slave Actions**:

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

### 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.



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

# 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.

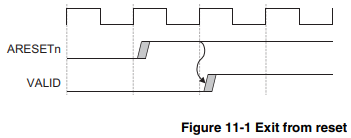
### 11.1.2 Reset

* 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*.

### Figure 11-1 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.



# 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

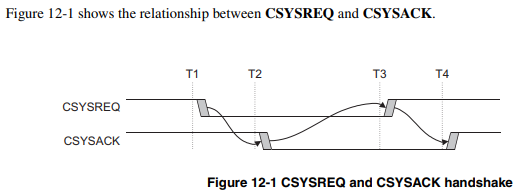
**Overview:**

* 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.

### Example



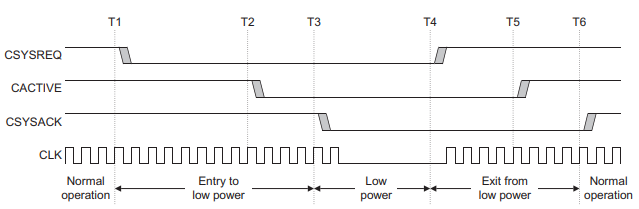
* **T1:** Request to put the peripheral in a low-power state
* **T2**: Request acknowledged
* **T3**: Exit from the low-power state
* **T4**: Exit acknowledged

### 12.2.1 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:**

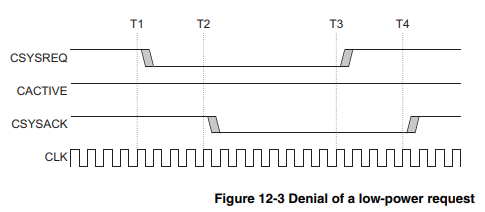


* **T1:** Request to put the peripheral in a low-power state
* **T2-T3:** The peripheral deasserts CACTIVE as it performs its power-down function and acknowledges by deasserting CSYSACK
* **T4:** Exit from the low-power state
* **T5-T6:** The peripheral asserts CACTIVE and completes the exit sequence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

### 12.2.2 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:**



* **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.