

# AMBA® AXI Protocol

v1.0

## Specification

**ARM®**

# **AMBA AXI Protocol**

## **Specification**

Copyright © 2003, 2004 ARM Limited. All rights reserved.

### **Release Information**

Change history		
Date	Issue	Change
16 June, 2003	A	First release
19 March 2004	B	Define read and write address channels

### **Proprietary Notice**

Words and logos marked with ® or ™ are registered trademarks or trademarks of ARM Limited in the EU and other countries, except as otherwise stated below in this proprietary notice. Other brands and names mentioned herein may be the trademarks of their respective owners.

Neither the whole nor any part of the information contained in, or the product described in, this document may be adapted or reproduced in any material form except with the prior written permission of the copyright holder.

The product described in this document is subject to continuous developments and improvements. All particulars of the product and its use contained in this document are given by ARM Limited in good faith. However, all warranties implied or expressed, including but not limited to implied warranties of merchantability, or fitness for purpose, are excluded.

This document is intended only to assist the reader in the use of the product. ARM Limited shall not be liable for any loss or damage arising from the use of any information in this document, or any error or omission in such information, or any incorrect use of the product.

### **AMBA Specification License**

1. Subject to the provisions of Clauses 2 and 3, ARM hereby grants to LICENSEE a perpetual, non-exclusive, nontransferable, royalty free, worldwide licence to use and copy the AMBA Specification for the purpose of developing, having developed, manufacturing, having manufactured, offering to sell, selling, supplying or otherwise distributing products which comply with the AMBA Specification.

2. THE AMBA SPECIFICATION IS PROVIDED "AS IS" WITH NO WARRANTIES EXPRESS, IMPLIED OR STATUTORY, INCLUDING BUT NOT LIMITED TO ANY WARRANTY OF SATISFACTORY QUALITY, MERCHANTABILITY, NONINFRINGEMENT OR FITNESS FOR A PARTICULAR PURPOSE.

3. No licence, express, implied or otherwise, is granted to LICENSEE, under the provisions of Clause 1, to use the ARM trademark, or AMBA trademark in connection with the AMBA Specification or any products based thereon. Nothing in Clause 1 shall be construed as authority for LICENSEE to make any representations on behalf of ARM in respect of the AMBA Specification.

**Confidentiality Status**

This document is Open Access. This document has no restriction on distribution.

**Product Status**

The information in this document is Final (information on a developed product).

**Web Address**

<http://www.arm.com>



# Contents

# AMBA AXI Protocol Specification

## Preface

About this document .....	xiv
Feedback .....	xviii

## Chapter 1

### Introduction

1.1 About the AXI protocol .....	1-2
1.2 Architecture .....	1-3
1.3 Basic transactions .....	1-7
1.4 Additional features .....	1-11

## Chapter 2

### Signal Descriptions

2.1 Global signals .....	2-2
2.2 Write address channel signals .....	2-3
2.3 Write data channel signals .....	2-4
2.4 Write response channel signals .....	2-5
2.5 Read address channel signals .....	2-6
2.6 Read data channel signals .....	2-7
2.7 Low-power interface signals .....	2-8

<b>Chapter 3</b>	<b>Channel Handshake</b>	
3.1	Handshake process .....	3-2
3.2	Relationships between the channels .....	3-6
3.3	Dependencies between channel handshake signals .....	3-7
<b>Chapter 4</b>	<b>Addressing Options</b>	
4.1	About addressing options .....	4-2
4.2	Burst length .....	4-3
4.3	Burst size .....	4-4
4.4	Burst type .....	4-5
4.5	Burst address .....	4-7
<b>Chapter 5</b>	<b>Additional Control Information</b>	
5.1	Cache support .....	5-2
5.2	Protection unit support .....	5-5
<b>Chapter 6</b>	<b>Atomic Accesses</b>	
6.1	About atomic accesses .....	6-2
6.2	Exclusive access .....	6-3
6.3	Locked access .....	6-7
<b>Chapter 7</b>	<b>Response Signaling</b>	
7.1	About response signaling .....	7-2
7.2	Response types .....	7-4
<b>Chapter 8</b>	<b>Ordering Model</b>	
8.1	About the ordering model .....	8-2
8.2	Transfer ID fields .....	8-3
8.3	Read ordering .....	8-4
8.4	Normal write ordering .....	8-5
8.5	Write data interleaving .....	8-6
8.6	Read and write interaction .....	8-8
8.7	Interconnect use of ID fields .....	8-9
8.8	Recommended width of ID fields .....	8-10
<b>Chapter 9</b>	<b>Data Buses</b>	
9.1	About the data buses .....	9-2
9.2	Write strobes .....	9-3
9.3	Narrow transfers .....	9-4
9.4	Byte invariance .....	9-5
<b>Chapter 10</b>	<b>Unaligned Transfers</b>	
10.1	About unaligned transfers .....	10-2
10.2	Examples .....	10-3

<b>Chapter 11</b>	<b>Clock and Reset</b>	
11.1	Clock and reset requirements .....	11-2
<b>Chapter 12</b>	<b>Low-power Interface</b>	
12.1	About the low-power interface .....	12-2
12.2	Low-power clock control .....	12-3



# List of Tables

## AMBA AXI Protocol Specification

Table 2-1	Change history .....	ii
Table 2-2	Global signals .....	2-2
Table 2-3	Write address channel signals .....	2-3
Table 2-4	Write data channel signals .....	2-4
Table 2-5	Write response channel signals .....	2-5
Table 2-6	Read address channel signals .....	2-6
Table 2-7	Read data channel signals .....	2-7
Table 4-1	Low-power interface signals .....	2-8
Table 4-2	Burst length encoding .....	4-3
Table 4-3	Burst size encoding .....	4-4
Table 5-1	Burst type encoding .....	4-5
Table 5-2	Cache encoding .....	5-3
Table 6-1	Protection encoding .....	5-6
Table 7-1	Atomic access encoding .....	6-2
	RRESP[1:0] and BRESP[1:0] encoding .....	7-2



# List of Figures

## AMBA AXI Protocol Specification

Figure 1-1	Key to timing diagram conventions .....	xvi
Figure 1-2	Channel architecture of reads .....	1-3
Figure 1-3	Channel architecture of writes .....	1-4
Figure 1-4	Interface and interconnect .....	1-5
Figure 1-5	Read burst .....	1-7
Figure 1-6	Overlapping read bursts .....	1-8
Figure 3-1	Write burst .....	1-9
Figure 3-2	VALID before READY handshake .....	3-2
Figure 3-3	READY before VALID handshake .....	3-3
Figure 3-4	VALID with READY handshake .....	3-3
Figure 3-5	READY with VALID handshake .....	3-3
Figure 9-1	Read transaction handshake dependencies .....	3-7
Figure 9-2	Write transaction handshake dependencies .....	3-8
Figure 9-3	Byte lane mapping .....	9-3
Figure 9-4	Narrow transfer example with 8-bit transfers .....	9-4
Figure 9-5	Narrow transfer example with 32-bit transfers .....	9-4
Figure 10-1	Example mixed-endian data structure .....	9-5
Figure 10-2	Aligned and unaligned word transfers on a 32-bit bus .....	10-3
Figure 10-3	Aligned and unaligned word transfers on a 64-bit bus .....	10-4
Figure 11-1	Aligned wrapping word transfers on a 64-bit bus .....	10-4
Figure 12-1	Exit from reset .....	11-2
Figure 12-2	CSYSREQ and CSYSACK handshake .....	12-3
Figure 12-3	Acceptance of a low-power request .....	12-4
Figure 12-4	Denial of a low-power request .....	12-5
	Low-power clock control sequence .....	12-6



# Preface

This preface introduces the *AMBA Advanced eXtensible Interface (AXI) Protocol Specification* and its reference documentation. It contains the following sections:

- *About this document* on page xiv
- *Feedback* on page xviii.

## About this document

This is the AMBA AXI Protocol Specification v1.0. This issue supersedes the previous r0p0 version of the specification.

## Intended audience

This specification is written to help hardware and software engineers who want to become familiar with the *Advanced Microcontroller Bus Architecture* (AMBA) and design systems and modules that are compatible with the AXI protocol.

## Using this specification

This specification is organized into the following chapters:

### **Chapter 1 *Introduction***

Read this chapter to learn about the AXI protocol architecture and the basic transactions that it defines.

### **Chapter 2 *Signal Descriptions***

Refer to this chapter for definitions of the AXI global, write address channel, write data channel, write response channel, read address channel, read data channel, and low-power interface signals.

### **Chapter 3 *Channel Handshake***

Read this chapter to learn about the AXI channel handshake process.

### **Chapter 4 *Addressing Options***

Read this chapter to learn about AXI burst types and how to calculate addresses and byte lanes for transfers within a burst.

### **Chapter 5 *Additional Control Information***

Read this chapter to learn how to use the AXI protocol to support system level caches and protection units.

### **Chapter 6 *Atomic Accesses***

Read this chapter to learn how to perform exclusive accesses and locked accesses.

### **Chapter 7 *Response Signaling***

Read this chapter to learn about the four transaction responses of AXI slaves.

**Chapter 8 Ordering Model**

Read this chapter to learn how the AXI protocol uses transaction ID tags to enable out-of-order transaction processing.

**Chapter 9 Data Buses**

Read this chapter to learn how to do transactions of varying sizes on the AXI read and write data buses and how to use byte-invariant endianness to handle mixed-endian data.

**Chapter 10 Unaligned Transfers**

Read this chapter to learn how the AXI protocol handles unaligned transfers.

**Chapter 11 Clock and Reset**

Read this chapter to learn about the timing of the AXI clock and reset signals.

**Chapter 12 Low-power Interface**

Read this chapter to learn how to use the AXI clock control interface to enter into and exit from a low-power state.

**Conventions**

Conventions that this specification can use are described in:

- *Typographical*
- *Timing diagrams* on page xvi
- *Signals* on page xvii.

**Typographical**

The typographical conventions are:

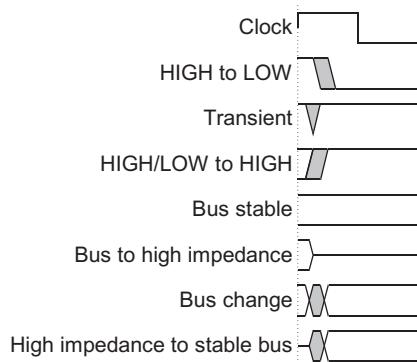
<b>italic</b>	Highlights important notes, introduces special terminology, denotes internal cross-references, and citations.
<b>bold</b>	Highlights interface elements, such as menu names. Denotes ARM signal names. Also used for terms in descriptive lists, where appropriate.
<b>monospace</b>	Denotes text that you can enter at the keyboard, such as commands, file and program names, and source code.

<b><u>monospace</u></b>	Denotes a permitted abbreviation for a command or option. You can enter the underlined text instead of the full command or option name.
<i>monospace italic</i>	Denotes arguments to monospace text where the argument is to be replaced by a specific value.
<b><u>monospace bold</u></b>	Denotes language keywords when used outside example code.
<b>&lt; and &gt;</b>	Angle brackets enclose replaceable terms for assembler syntax where they appear in code or code fragments. They appear in normal font in running text. For example:
	<ul style="list-style-type: none"> <li>• MRC p15, 0 &lt;Rd&gt;, &lt;CRn&gt;, &lt;CRm&gt;, &lt;Opcode_2&gt;</li> <li>• The Opcode_2 value selects which register is accessed.</li> </ul>

## Timing diagrams

The figure named *Key to timing diagram conventions* explains the components used in timing diagrams. Variations, when they occur, have clear labels. You must not assume any timing information that is not explicit in the diagrams.

Shaded bus and signal areas are undefined, so the bus or signal can assume any value within the shaded area at that time. The actual level is unimportant and does not affect normal operation.



**Key to timing diagram conventions**

## Signals

The signal conventions are:

<b>Signal level</b>	The level of an asserted signal depends on whether the signal is active-HIGH or active-LOW. Asserted means HIGH for active-HIGH signals and LOW for active-LOW signals.
<b>Prefix A</b>	Denotes AXI global, write address, and read address channel signals.
<b>Prefix B</b>	Denotes AXI write response channel signals.
<b>Prefix C</b>	Denotes AXI low-power interface signals.
<b>Prefix R</b>	Denotes AXI read data channel signals.
<b>Prefix W</b>	Denotes AXI write data channel signals.
<b>Suffix n</b>	Denotes that the AXI <b>ARESETn</b> global reset signal is active-LOW.

## Further reading

This section lists publications by ARM Limited.

ARM periodically provides updates and corrections to its documentation. See <http://www.arm.com> and <http://www.amba.com> for current errata sheets, addenda, and the ARM and AMBA Frequently Asked Questions list.

## ARM publications

This specification contains information that is specific to the AXI protocol. Refer to the following documents for other relevant information:

- *AMBA Specification (Rev 2.0)* (ARM IHI 0011)
- *ARM Architecture Reference Manual* (ARM DDI 0100).

## Feedback

ARM Limited welcomes feedback on the AMBA AXI protocol and its documentation.

### Feedback on this product

If you have any comments or suggestions about this product, contact your supplier giving:

- the product name
- a concise explanation of your comments.

### Feedback on this specification

If you have any comments on this specification, send email to [errata@arm.com](mailto:errata@arm.com) giving:

- the title
- the number
- the relevant page number(s) to which your comments apply
- a concise explanation of your comments.

ARM Limited also welcomes general suggestions for additions and improvements.

# Chapter 1

## Introduction

This chapter describes the architecture of the AXI protocol and the basic transactions that the protocol defines. It contains the following sections:

- *About the AXI protocol* on page 1-2
- *Architecture* on page 1-3
- *Basic transactions* on page 1-7
- *Additional features* on page 1-11.

## 1.1 About the AXI protocol

The AMBA AXI protocol is targeted at high-performance, high-frequency system designs and includes a number of features that make it suitable for a high-speed submicron interconnect.

The objectives of the latest generation AMBA interface are to:

- be suitable for high-bandwidth and low-latency designs
- enable high-frequency operation without using complex bridges
- meet the interface requirements of a wide range of components
- be suitable for memory controllers with high initial access latency
- provide flexibility in the implementation of interconnect architectures
- be backward-compatible with existing AHB and APB interfaces.

The key features of the AXI protocol are:

- separate address/control and data phases
- support for unaligned data transfers using byte strobes
- burst-based transactions with only start address issued
- separate read and write data channels to enable low-cost *Direct Memory Access* (DMA)
- ability to issue multiple outstanding addresses
- out-of-order transaction completion
- easy addition of register stages to provide timing closure.

As well as the data transfer protocol, the AXI protocol includes optional extensions that cover signaling for low-power operation.

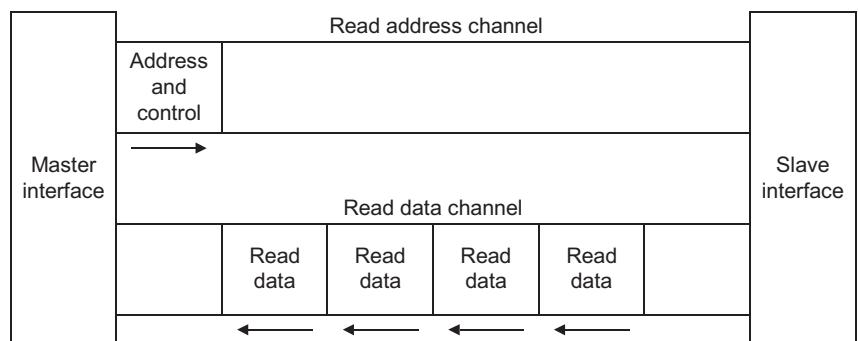
## 1.2 Architecture

The AXI protocol is burst-based. Every transaction has address and control information on the address channel that describes the nature of the data to be transferred. The data is transferred between master and slave using a write data channel to the slave or a read data channel to the master. In write transactions, in which all the data flows from the master to the slave, the AXI protocol has an additional write response channel to allow the slave to signal to the master the completion of the write transaction.

The AXI protocol enables:

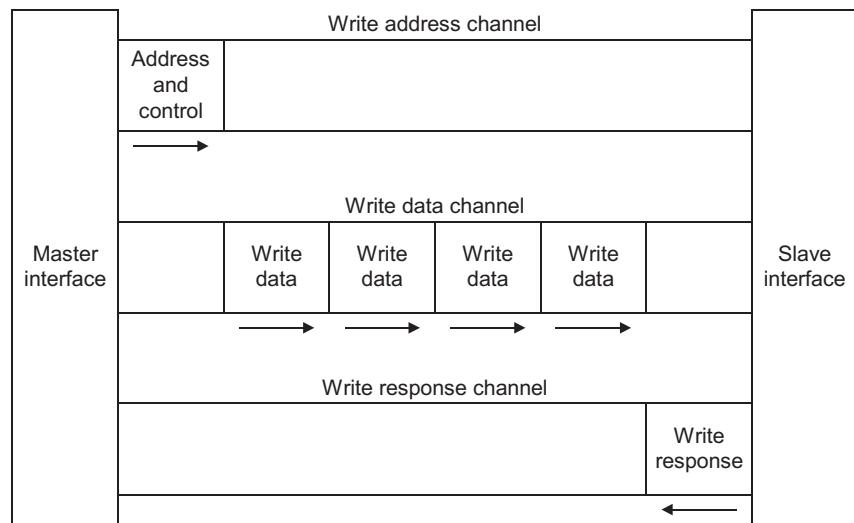
- address information to be issued ahead of the actual data transfer
- support for multiple outstanding transactions
- support for out-of-order completion of transactions.

Figure 1-1 shows how a read transaction uses the read address and read data channels.



**Figure 1-1 Channel architecture of reads**

Figure 1-2 on page 1-4 shows how a write transaction uses the write address, write data, and write response channels.

**Figure 1-2 Channel architecture of writes**

### 1.2.1 Channel definition

Each of the five independent channels consists of a set of information signals and uses a two-way **VALID** and **READY** handshake mechanism.

The information source uses the **VALID** signal to show when valid data or control information is available on the channel. The destination uses the **READY** signal to show when it can accept the data. Both the read data channel and the write data channel also include a **LAST** signal to indicate when the transfer of the final data item within a transaction takes place.

#### Read and write address channels

Read and write transactions each have their own address channel. The appropriate address channel carries all of the required address and control information for a transaction. The AXI protocol supports the following mechanisms:

- variable-length bursts, from 1 to 16 data transfers per burst
- bursts with a transfer size of 8-1024 bits
- wrapping, incrementing, and non-incrementing bursts
- atomic operations, using exclusive or locked accesses
- system-level caching and buffering control

- secure and privileged access.

### Read data channel

The read data channel conveys both the read data and any read response information from the slave back to the master. The read data channel includes:

- the data bus, which can be 8, 16, 32, 64, 128, 256, 512, or 1024 bits wide
- a read response indicating the completion status of the read transaction.

### Write data channel

The write data channel conveys the write data from the master to the slave and includes:

- the data bus, which can be 8, 16, 32, 64, 128, 256, 512, or 1024 bits wide
- one byte lane strobe for every eight data bits, indicating which bytes of the data bus are valid.

Write data channel information is always treated as buffered, so that the master can perform write transactions without slave acknowledgement of previous write transactions.

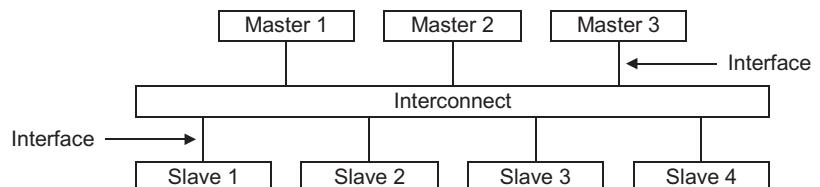
### Write response channel

The write response channel provides a way for the slave to respond to write transactions. All write transactions use completion signaling.

The completion signal occurs once for each burst, not for each individual data transfer within the burst.

## 1.2.2 Interface and interconnect

A typical system consists of a number of master and slave devices connected together through some form of interconnect, as shown in Figure 1-3.



**Figure 1-3 Interface and interconnect**

The AXI protocol provides a single interface definition for describing interfaces:

- between a master and the interconnect
- between a slave and the interconnect
- between a master and a slave.

The interface definition enables a variety of different interconnect implementations. The interconnect between devices is equivalent to another device with symmetrical master and slave ports to which real master and slave devices can be connected.

Most systems use one of three interconnect approaches:

- shared address and data buses
- shared address buses and multiple data buses
- multilayer, with multiple address and data buses.

In most systems, the address channel bandwidth requirement is significantly less than the data channel bandwidth requirement. Such systems can achieve a good balance between system performance and interconnect complexity by using a shared address bus with multiple data buses to enable parallel data transfers.

### 1.2.3 Register slices

Each AXI channel transfers information in only one direction, and there is no requirement for a fixed relationship between the various channels. This is important because it enables the insertion of a register slice in any channel, at the cost of an additional cycle of latency. This makes possible a trade-off between cycles of latency and maximum frequency of operation.

It is also possible to use register slices at almost any point within a given interconnect. It can be advantageous to use a direct, fast connection between a processor and high-performance memory, but to use simple register slices to isolate a longer path to less performance-critical peripherals.

## 1.3 Basic transactions

This section gives examples of basic AXI protocol transactions. Each example shows the **VALID** and **READY** handshake mechanism. Transfer of either address information or data occurs when both the **VALID** and **READY** signals are HIGH. The examples are provided in:

- *Read burst example*
- *Overlapping read burst example* on page 1-8
- *Write burst example* on page 1-9.

This section also describes *Transaction ordering* on page 1-9.

### 1.3.1 Read burst example

Figure 1-4 shows a **read burst** of four transfers. In this example, the master drives the address, and the slave accepts it one cycle later.

— Note —

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

After the address appears on the address bus, the data transfer occurs on the read data channel. The slave keeps the **VALID** signal LOW until the read data is available. For the final data transfer of the burst, the slave asserts the **RLAST** signal to show that the last data item is being transferred.

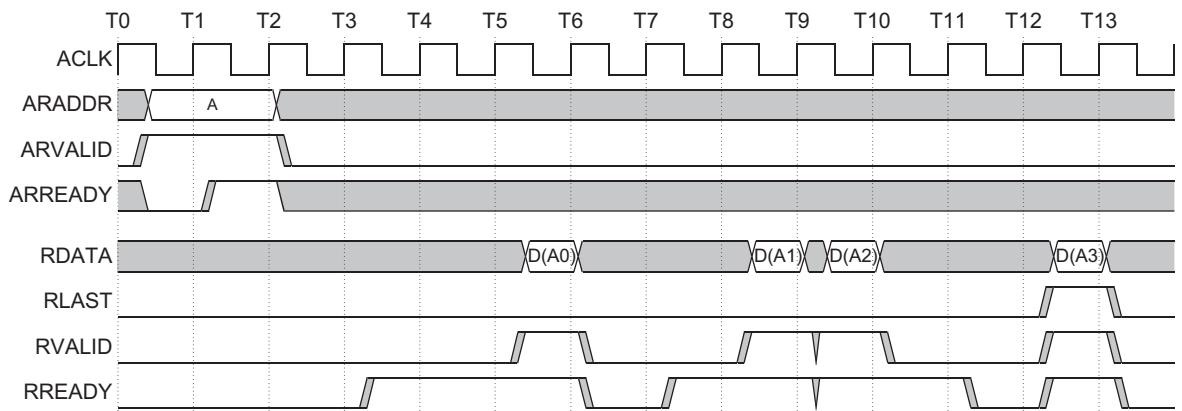


Figure 1-4 Read burst

### 1.3.2 Overlapping read burst example

Figure 1-5 shows how a master can drive another burst address after the slave accepts the first address. This enables a slave to begin processing data for the second burst in parallel with the completion of the first burst.

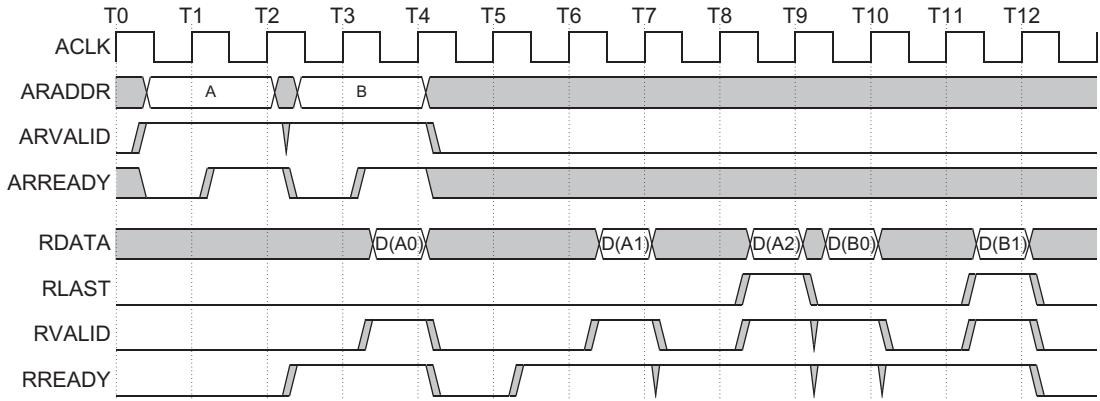


Figure 1-5 Overlapping read bursts

### 1.3.3 Write burst example

Figure 1-6 shows a write transaction. The process starts when the master sends an address and control information on the write address channel. The master then sends each item of write data over the write data channel. When the master sends the last data item, the **WLAST** signal goes HIGH. When the slave has accepted all the data items, it drives a write response back to the master to indicate that the write transaction is complete.

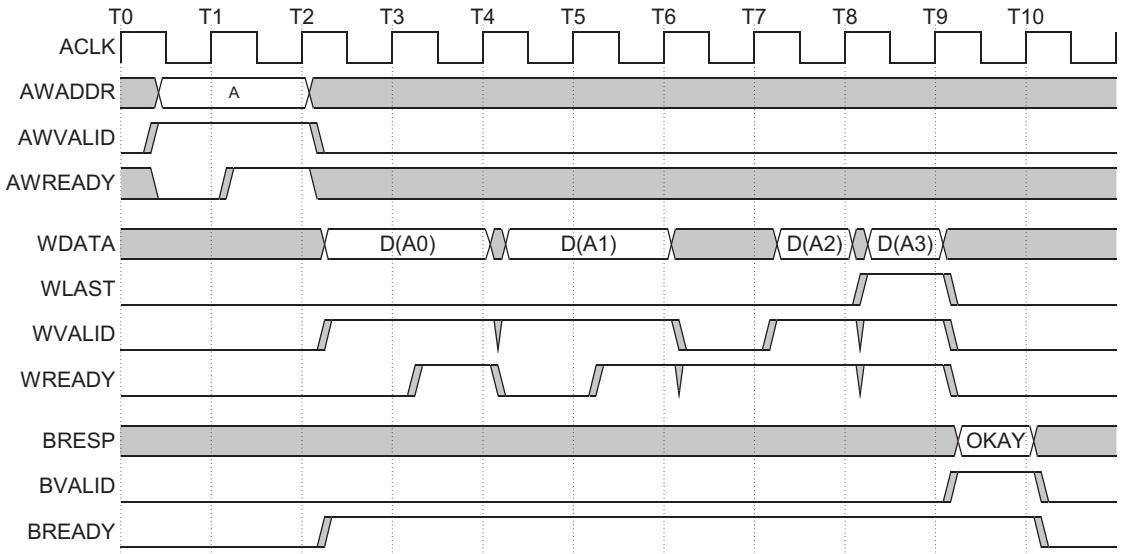


Figure 1-6 Write burst

### 1.3.4 Transaction ordering

The AXI protocol enables out-of-order transaction completion. It gives an ID tag to every transaction across the interface. The protocol requires that transactions with the same ID tag are completed in order, but transactions with different ID tags can be completed out of order.

Out-of-order transactions can improve system performance in two ways:

- The interconnect can enable transactions with fast-responding slaves to complete in advance of earlier transactions with slower slaves.

- Complex slaves can return read data out of order. For example, a data item for a later access might be available from an internal buffer before the data for an earlier access is available.

If a master requires that transactions are completed in the same order that they are issued, then they must all have the same ID tag. If, however, a master does not require in-order transaction completion, it can supply the transactions with different ID tags, enabling them to be completed in any order.

In a multimaster system, the interconnect is responsible for appending additional information to the ID tag to ensure that ID tags from all masters are unique. The ID tag is similar to a master number, but with the extension that each master can implement multiple virtual masters within the same port by supplying an ID tag to indicate the virtual master number.

Although complex devices can make use of the out-of-order facility, simple devices are not required to use it. Simple masters can issue every transaction with the same ID tag, and simple slaves can respond to every transaction in order, irrespective of the ID tag.

## 1.4 Additional features

The AXI protocol also supports the following additional features:

### Burst types

The AXI protocol supports three different burst types that are suitable for:

- **normal memory accesses**
- **wrapping cache line bursts**
- **streaming data to peripheral FIFO locations.**

See Chapter 4 *Addressing Options*.

### System cache support

The cache-support signal of the AXI protocol enables a master to provide to a system-level cache the bufferable, cacheable, and allocate attributes of a transaction.

See *Cache support* on page 5-2.

### Protection unit support

To enable both privileged and secure accesses, the AXI protocol provides three levels of protection unit support.

See *Protection unit support* on page 5-5.

### Atomic operations

The AXI protocol defines mechanisms for both exclusive and locked accesses.

See Chapter 6 *Atomic Accesses*.

### Error support

The AXI protocol provides error support for both address decode errors and slave-generated errors.

See Chapter 7 *Response Signaling*.

### Unaligned address

To enhance the performance of the initial accesses within a burst, the AXI protocol supports unaligned burst start addresses.

See Chapter 10 *Unaligned Transfers*.



# Chapter 2

## Signal Descriptions

This chapter defines the AXI signals. Although bus width and transaction ID width are implementation-specific, the tables in this chapter show a 32-bit data bus, a four-bit write data strobe, and four-bit ID fields. This chapter contains the following sections:

- *Global signals* on page 2-2
- *Write address channel signals* on page 2-3
- *Write data channel signals* on page 2-4
- *Write response channel signals* on page 2-5
- *Read address channel signals* on page 2-6
- *Read data channel signals* on page 2-7
- *Low-power interface signals* on page 2-8.

## 2.1 Global signals

Table 2-1 lists the global AXI signals.

**Table 2-1 Global signals**

Signal	Source	Description
<b>ACLK</b>	Clock source	Global clock signal. All signals are sampled on the <b>rising edge</b> of the global clock.
<b>ARESETn</b>	Reset source	Global reset signal. This signal is <b>active LOW</b> .

## 2.2 Write address channel signals

Table 2-2 lists the AXI write address channel signals.

**Table 2-2 Write address channel signals**

Signal	Source	Description
<b>AWID[3:0]</b>	Master	Write address ID. This signal is the identification tag for the write address group of signals.
<b>AWADDR[31:0]</b>	Master	Write address. The write address bus gives the address of the first transfer in a write burst transaction. The associated control signals are used to determine the addresses of the remaining transfers in the burst.
<b>AWLEN[3:0]</b>	Master	Burst length. The burst length gives the exact number of transfers in a burst. This information determines the number of data transfers associated with the address. See Table 4-1 on page 4-3.
<b>AWSIZE[2:0]</b>	Master	Burst size. This signal indicates the size of each transfer in the burst. Byte lane strobes indicate exactly which byte lanes to update. See Table 4-2 on page 4-4.
<b>AWBURST[1:0]</b>	Master	Burst type. The burst type, coupled with the size information, details how the address for each transfer within the burst is calculated. See Table 4-3 on page 4-5.
<b>AWLOCK[1:0]</b>	Master	Lock type. This signal provides additional information about the atomic characteristics of the transfer. See Table 6-1 on page 6-2.
<b>AWCACHE[3:0]</b>	Master	Cache type. This signal indicates the bufferable, cacheable, write-through, write-back, and allocate attributes of the transaction. See Table 5-1 on page 5-3.
<b>AWPROT[2:0]</b>	Master	Protection type. This signal indicates the normal, privileged, or secure protection level of the transaction and whether the transaction is a data access or an instruction access. See <i>Protection unit support</i> on page 5-5.
<b>AWVALID</b>	Master	Write address valid. This signal indicates that valid write address and control information are available: 1 = address and control information available 0 = address and control information not available. The address and control information remain stable until the address acknowledge signal, <b>AWREADY</b> , goes HIGH.
<b>AWREADY</b>	Slave	Write address ready. This signal indicates that the slave is ready to accept an address and associated control signals: 1 = slave ready 0 = slave not ready.

## 2.3 Write data channel signals

Table 2-3 lists the AXI write data channel signals.

**Table 2-3 Write data channel signals**

Signal	Source	Description
<b>WID[3:0]</b>	Master	Write ID tag. This signal is the ID tag of the write data transfer. The <b>WID</b> value must match the <b>AWID</b> value of the write transaction.
<b>WDATA[31:0]</b>	Master	Write data. The write data bus can be 8, 16, 32, 64, 128, 256, 512, or 1024 bits wide.
<b>WSTRB[3:0]</b>	Master	Write strobes. This signal indicates which byte lanes to update in memory. There is one write strobe for each eight bits of the write data bus. Therefore, <b>WSTRB[n]</b> corresponds to <b>WDATA[(8 × n) + 7:(8 × n)]</b> .
<b>WLAST</b>	Master	Write last. This signal indicates the last transfer in a write burst.
<b>WVALID</b>	Master	Write valid. This signal indicates that valid write data and strobes are available: 1 = write data and strobes available 0 = write data and strobes not available.
<b>WREADY</b>	Slave	Write ready. This signal indicates that the slave can accept the write data: 1 = slave ready 0 = slave not ready.

## 2.4 Write response channel signals

Table 2-4 lists the AXI write response channel signals.

**Table 2-4 Write response channel signals**

Signal	Source	Description
<b>BID[3:0]</b>	Slave	Response ID. The identification tag of the write response. The <b>BID</b> value must match the <b>AWID</b> value of the write transaction to which the slave is responding.
<b>BRESP[1:0]</b>	Slave	<b>Write response.</b> This signal indicates the <b>status of the write transaction</b> . The allowable responses are <b>OKAY, EXOKAY, SLVERR, and DECERR</b> .
<b>BVALID</b>	Slave	<b>Write response valid.</b> This signal <b>indicates that a valid write response is available:</b> <b>1 = write response available</b> <b>0 = write response not available.</b>
<b>BREADY</b>	Master	<b>Response ready.</b> This signal <b>indicates that the master can accept the response information.</b> <b>1 = master ready</b> <b>0 = master not ready.</b>

## 2.5 Read address channel signals

Table 2-2 on page 2-3 lists the AXI read address channel signals.

**Table 2-5 Read address channel signals**

Signal	Source	Description
<b>ARID[3:0]</b>	Master	Read address ID. This signal is the identification tag for the read address group of signals.
<b>ARADDR[31:0]</b>	Master	Read address. The read address bus gives the initial address of a read burst transaction. Only the start address of the burst is provided and the control signals that are issued alongside the address detail how the address is calculated for the remaining transfers in the burst.
<b>ARLEN[3:0]</b>	Master	Burst length. The burst length gives the exact number of transfers in a burst. This information determines the number of data transfers associated with the address. See Table 4-1 on page 4-3.
<b>ARSIZE[2:0]</b>	Master	Burst size. This signal indicates the size of each transfer in the burst. See Table 4-2 on page 4-4.
<b>ARBURST[1:0]</b>	Master	Burst type. The burst type, coupled with the size information, details how the address for each transfer within the burst is calculated. See Table 4-3 on page 4-5.
<b>ARLOCK[1:0]</b>	Master	Lock type. This signal provides additional information about the atomic characteristics of the transfer. See Table 6-1 on page 6-2.
<b>ARCACHE[3:0]</b>	Master	Cache type. This signal provides additional information about the cacheable characteristics of the transfer. See Table 5-1 on page 5-3.
<b>ARPROT[2:0]</b>	Master	Protection type. This signal provides protection unit information for the transaction. See <i>Protection unit support</i> on page 5-5.
<b>ARVALID</b>	Master	Read address valid. This signal indicates, when HIGH, that the read address and control information is valid and will remain stable until the address acknowledge signal, <b>ARREADY</b> , is high. 1 = address and control information valid 0 = address and control information not valid.
<b>ARREADY</b>	Slave	Read address ready. This signal indicates that the slave is ready to accept an address and associated control signals: 1 = slave ready 0 = slave not ready.

## 2.6 Read data channel signals

Table 2-6 lists the AXI read data channel signals.

**Table 2-6 Read data channel signals**

Signal	Source	Description
<b>RID[3:0]</b>	Slave	Read ID tag. This signal is the ID tag of the read data group of signals. The <b>RID</b> value is generated by the slave and must match the <b>ARID</b> value of the read transaction to which it is responding.
<b>RDATA[31:0]</b>	Slave	Read data. The read data bus can be 8, 16, 32, 64, 128, 256, 512, or 1024 bits wide.
<b>RRESP[1:0]</b>	Slave	Read response. This signal indicates the status of the read transfer. The allowable responses are OKAY, EXOKAY, SLVERR, and DECERR.
<b>RLAST</b>	Slave	Read last. This signal indicates the last transfer in a read burst.
<b>RVALID</b>	Slave	Read valid. This signal indicates that the required read data is available and the read transfer can complete: 1 = read data available 0 = read data not available.
<b>RREADY</b>	Master	Read ready. This signal indicates that the master can accept the read data and response information: 1 = master ready 0 = master not ready.

## 2.7 Low-power interface signals

Table 2-7 lists the signals of the optional low-power interface.

**Table 2-7 Low-power interface signals**

<b>Signal</b>	<b>Source</b>	<b>Description</b>
<b>CSYSREQ</b>	Clock controller	System low-power request. This signal is a request from the system clock controller for the peripheral to enter a low-power state.
<b>CSYSACK</b>	Peripheral device	Low-power request acknowledgement. This signal is the acknowledgement from a peripheral of a system low-power request.
<b>CACTIVE</b>	Peripheral device	Clock active. This signal indicates that the peripheral requires its clock signal: 1 = peripheral clock required 0 = peripheral clock not required.

# Chapter 3

## Channel Handshake

This chapter describes the master/slave handshake process and outlines the relationships and default values of the **READY** and **VALID** handshake signals. It contains the following sections:

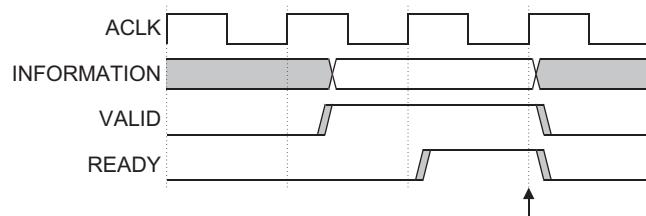
- *Handshake process* on page 3-2
- *Relationships between the channels* on page 3-6
- *Dependencies between channel handshake signals* on page 3-7.

### 3.1 Handshake process

All five channels use the same **VALID/READY** handshake to transfer data and control information. This two-way flow control mechanism enables both the master and slave to control the rate at which the data and control information moves. The source generates the **VALID** signal to indicate when the data or control information is available. The destination generates the **READY** signal to indicate that it accepts the data or control information. Transfer occurs only when both the **VALID** and **READY** signals are HIGH.

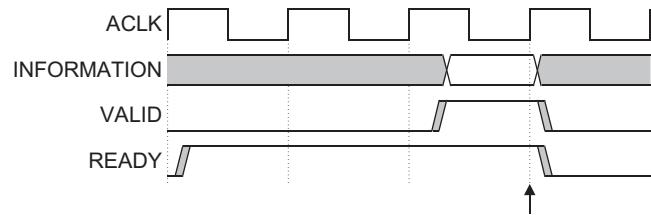
There must be no combinatorial paths between input and output signals on both master and slave interfaces.

Figure 3-1 to Figure 3-3 on page 3-3 show examples of the handshake sequence. In Figure 3-1, the source presents the data or control information and drives the **VALID** signal HIGH. The data or control information from the source remains stable until the destination drives the **READY** signal HIGH, indicating that it accepts the data or control information. The arrow shows when the transfer occurs.

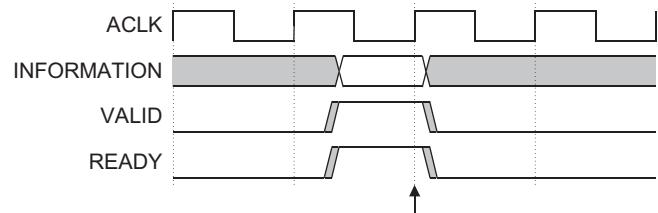


**Figure 3-1 VALID before READY handshake**

In Figure 3-2 on page 3-3, the destination drives **READY** HIGH before the data or control information is valid. This indicates that the destination can accept the data or control information in a single cycle as soon as it becomes valid. The arrow shows when the transfer occurs.

**Figure 3-2 READY before VALID handshake**

In Figure 3-3, both the source and destination happen to indicate in the same cycle that they can transfer the data or control information. In this case the transfer occurs immediately. The arrow shows when the transfer occurs.

**Figure 3-3 VALID with READY handshake**

The individual AXI protocol channel handshake mechanisms are described in:

- *Write address channel*
- *Write data channel* on page 3-4
- *Write response channel* on page 3-4
- *Read address channel* on page 3-4
- *Read data channel* on page 3-5.

### 3.1.1 Write address channel

The master can assert the AWVALID signal only when it drives valid address and control information. It must remain asserted until the slave accepts the address and control information and asserts the associated AWREADY signal.

The default value of AWREADY can be either HIGH or LOW. The recommended default value is HIGH, although if AWREADY is HIGH then the slave must be able to accept any valid address that is presented to it.

A default **AWREADY** value of LOW is possible but not recommended, because it implies that the transfer takes at least two cycles, one to assert **AWVALID** and another to assert **AWREADY**.

### 3.1.2 Write data channel

During a write burst, the master can assert the **WVALID** signal only when it drives valid write data. **WVALID** must remain asserted until the slave accepts the write data and asserts the **WREADY** signal.

The default value of **WREADY** can be HIGH, but only if the slave can always accept write data in a single cycle.

The master must assert the **WLAST** signal when it drives the final write transfer in the burst.

When **WVALID** is LOW, the **WSTRB[3:0]** signals can take any value, although it is recommended that they are either driven LOW or held at their previous value.

### 3.1.3 Write response channel

The slave can assert the **BVALID** signal only when it drives a valid write response. **BVALID** must remain asserted until the master accepts the write response and asserts **BREADY**.

The default value of **BREADY** can be HIGH, but only if the master can always accept a write response in a single cycle.

### 3.1.4 Read address channel

The master can assert the **ARVALID** signal only when it drives valid address and control information. It must remain asserted until the slave accepts the address and control information and asserts the associated **ARREADY** signal.

The default value of **ARREADY** can be either HIGH or LOW. The recommended default value is HIGH, although if **ARREADY** is HIGH then the slave must be able to accept any valid address that is presented to it.

A default **ARREADY** value of LOW is possible but not recommended, because it implies that the transfer takes at least two cycles, one to assert **ARVALID** and another to assert **ARREADY**.

### 3.1.5 Read data channel

The slave can assert the **RVALID** signal only when it drives valid read data. **RVALID** must remain asserted until the master accepts the data and asserts the **RREADY** signal. Even if a slave has only one source of read data, it must assert the **RVALID** signal only in response to a request for the data.

The master interface uses the **RREADY** signal to indicate that it accepts the data. The default value of **RREADY** can be HIGH, but only if the master is able to accept read data immediately, whenever it performs a read transaction.

The slave must assert the **RLAST** signal when it drives the final read transfer in the burst.

## 3.2 Relationships between the channels

The relationship between the address, read, write, and write response channels is flexible.

For example, the write data can appear at an interface before the write address that relates to it. This can occur when the write address channel contains more register stages than the write data channel. It is also possible for the write data to appear in the same cycle as the address.

When the interconnect must determine the destination address space or slave space, it must realign the address and write data. This is required to assure that the write data is signaled as valid only to the slave for which it is destined.

Two relationships that must be maintained are:

- read data must always follow the address to which the data relates
- a write response must always follow the last write transfer in the write transaction to which the write response relates.

### 3.3 Dependencies between channel handshake signals

To prevent a deadlock situation, you must observe the dependencies that exist between the handshake signals.

In any transaction:

- the **VALID** signal of one AXI component must not be dependent on the **READY** signal of the other component in the transaction
- the **READY** signal can wait for assertion of the **VALID** signal.

**Note**

While it is acceptable to wait for **VALID** to be asserted before asserting **READY**, it is also acceptable to assert **READY** by default prior to the assertion of **VALID** and this can result in a more efficient design.

Figure 3-4 and Figure 3-5 on page 3-8 show the handshake signal dependencies. The single-headed arrows point to signals that can be asserted before or after the previous signal is asserted. Double-headed arrows point to signals that must be asserted only after assertion of the previous signal.

Figure 3-4 shows that, in a read transaction:

- the slave can wait for **ARVALID** to be asserted before it asserts **ARREADY**
- the slave must wait for both **ARVALID** and **ARREADY** to be asserted before it starts to return read data by asserting **RVALID**.

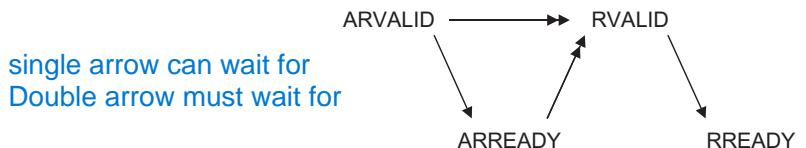


Figure 3-4 Read transaction handshake dependencies

Figure 3-5 on page 3-8 shows that, in a write transaction:

- the master must not wait for the slave to assert **AWREADY** or **WREADY** before asserting **AWVALID** or **WVALID**
- the slave can wait for **AWVALID** or **WVALID**, or both, before asserting **AWREADY**

- the slave can wait for **AWVALID** or **WVALID**, or both, before asserting **WREADY**
- the slave must wait for both **WVALID** and **WREADY** to be asserted before asserting **BVALID**.

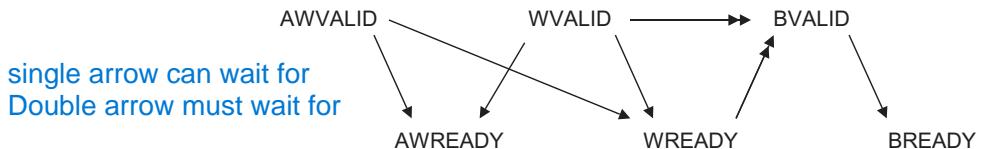


Figure 3-5 Write transaction handshake dependencies

———— Note ————

It is important that during a write transaction, a master must not wait for **AWREADY** to be asserted before driving **WVALID**. This could cause a deadlock condition if the slave is conversely waiting for **WVALID** before asserting **AWREADY**.

---

# Chapter 4

## Addressing Options

This chapter describes AXI burst types and how to calculate addresses and byte lanes for transfers within a burst. It contains the following sections:

- *About addressing options* on page 4-2
- *Burst length* on page 4-3
- *Burst size* on page 4-4
- *Burst type* on page 4-5
- *Burst address* on page 4-7.

## 4.1 About addressing options

The AXI protocol is burst-based, and the master begins each burst by driving transfer control information and the address of the first byte in the transfer. As the burst transaction progresses, it is the responsibility of the slave to calculate the addresses of subsequent transfers in the burst.

Bursts must not cross 4KB boundaries to prevent them from crossing boundaries between slaves and to limit the size of the address incrementer required within slaves.

## 4.2 Burst length

The **AWLEN** or **ARLEN** signal specifies the number of data transfers that occur within each burst. As Table 4-1 shows, each burst can be 1-16 transfers long.

**Table 4-1 Burst length encoding**

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

For wrapping bursts, the length of the burst must be 2, 4, 8, or 16 transfers.

Every transaction must have the number of transfers specified by **ARLEN** or **AWLEN**. No component can terminate a burst early to reduce the number of data transfers. During a write burst, the master can disable further writing by deasserting all the write strobes, but it must complete the remaining transfers in the burst. During a read burst, the master can discard further read data, but it must complete the remaining transfers in the burst.

### Caution

Discarding read data that is not required can result in lost data when accessing a read-sensitive device such as a FIFO. A master must never access such a device using a burst length longer than required.

## 4.3 Burst size

Table 4-2 shows how the **ARSIZE** or **AWSIZE** signal specifies the maximum number of data bytes to transfer in each beat, or data transfer, within a burst.

**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

The AXI determines from the transfer address which byte lanes of the data bus to use for each transfer.

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.

The size of any transfer must not exceed the data bus width of the components in the transaction.

## 4.4 Burst type

The AXI protocol defines three burst types described in:

- *Fixed burst*
- *Incrementing burst*
- *Wrapping burst* on page 4-6.

Table 4-3 shows how the **ARBURST** or **AWBURST** signal selects the burst type.

**Table 4-3 Burst type encoding**

<b>ARBURST[1:0] AWBURST[1:0]</b>	<b>Burst type</b>	<b>Description</b>	<b>Access</b>
b00	<b>FIXED</b>	Fixed-address burst	FIFO-type
b01	<b>INCR</b>	Incrementing-address burst	Normal sequential memory
b10	<b>WRAP</b>	Incrementing-address burst that wraps to a lower address at the wrap boundary	Cache line
b11	Reserved	-	-

### 4.4.1 Fixed burst

In a **fixed burst**, the address remains the same for every transfer in the burst. This burst type is for repeated accesses to the same location such as when loading or emptying a peripheral FIFO.

### 4.4.2 Incrementing burst

In an **incrementing burst**, the address for each transfer in the burst is an increment of the previous transfer address. The increment value depends on the size of the transfer. For example, the address for each transfer in a burst with a size of four bytes is the previous address plus four.

#### **4.4.3 Wrapping burst**

A wrapping burst is similar to an incrementing burst, in that the address for each transfer in the burst is an increment of the previous transfer address. However, in a wrapping burst the address wraps around to a lower address when a wrap boundary is reached. The wrap boundary is the size of each transfer in the burst multiplied by the total number of transfers in the burst.

Two restrictions apply to wrapping bursts:

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

## 4.5 Burst address

This section provides some simple formulas for determining the address and byte lanes of transfers within a burst. The formulas use the following variables:

<b>Start_Address</b>	The start address issued by the master.
<b>Number_Bytes</b>	The maximum number of bytes in each data transfer.
<b>Data_Bus_Bytes</b>	The number of byte lanes in the data bus.
<b>Aligned_Address</b>	The aligned version of the start address.
<b>Burst_Length</b>	The total number of data transfers within a burst.
<b>Address_N</b>	The address of transfer N within a burst. N is an integer from 2-16.
<b>Wrap_Boundary</b>	The lowest address within a wrapping burst.
<b>Lower[Byte_Lane]</b>	The byte lane of the lowest addressed byte of a transfer.
<b>Upper[Byte_Lane]</b>	The byte lane of the highest addressed byte of a transfer.
<b>INT(x)</b>	The rounded-down integer value of x.

Use these equations to determine addresses of transfers within a burst:

- **Start\_Address = ADDR**
- **Number\_Bytes = 2SIZE**
- **Burst\_Length = LEN + 1**
- **Aligned\_Address = (INT(Start\_Address / Number\_Bytes) ) x Number\_Bytes.**

Use this equation to determine the address of the first transfer in a burst:

- **Address\_1 = Start\_Address.**

Use this equation to determine the address of any transfer after the first transfer in a burst:

- **Address\_N = Aligned\_Address + (N - 1) x Number\_Bytes.**

For wrapping bursts, the **Wrap\_Boundary** variable is extended to account for the wrapping boundary:

- **Wrap\_Boundary = (INT(Start\_Address / (Number\_Bytes x Burst\_Length))) x (Number\_Bytes x Burst\_Length).**

If **Address\_N = Wrap\_Boundary + (Number\_Bytes x Burst\_Length)**, use this equation:

- **Address\_N = Wrap\_Boundary.**

Use these equations to determine which byte lanes to use for the first transfer in a burst:

- $\text{Lower\_Byte\_Lane} = \text{Start\_Address} - (\text{INT}(\text{Start\_Address} / \text{Data\_Bus\_Bytes})) \times \text{Data\_Bus\_Bytes}$
- $\text{Upper\_Byte\_Lane} = \text{Aligned\_Address} + (\text{Number\_Bytes} - 1) - (\text{INT}(\text{Start\_Address} / \text{Data\_Bus\_Bytes})) \times \text{Data\_Bus\_Bytes}.$

Use these equations to determine which byte lanes to use for all transfers after the first transfer in a burst:

- $\text{Lower\_Byte\_Lane} = \text{Address\_N} - (\text{INT}(\text{Address\_N} / \text{Data\_Bus\_Bytes})) \times \text{Data\_Bus\_Bytes}$
- $\text{Upper\_Byte\_Lane} = \text{Lower\_Byte\_Lane} + \text{Number\_Bytes} - 1.$

Data is transferred on:

- $\text{DATA}[(8 \times \text{Upper\_Byte\_Lane}) + 7 : (8 \times \text{Lower\_Byte\_Lane})].$

# Chapter 5

## Additional Control Information

This chapter describes AXI protocol support for system-level caches and protection units. It contains the following sections:

- *Cache support* on page 5-2
- *Protection unit support* on page 5-5.

## 5.1 Cache support

Support for system level caches and other performance enhancing components is provided by the use of the cache information signals, **ARCACHE** and **AWCACHE**. These signals provide additional information about how the transaction can be processed.

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 this bit is HIGH, it means that the interconnect or any component can delay the transaction reaching its final destination for an arbitrary number of cycles. This is usually only relevant to writes.

### **Cacheable (C) bit, ARCACHE[1] and AWCACHE[1]**

When this bit is HIGH, it means that the transaction at the final destination does not have to match the characteristics of the original transaction.

For writes this means that a number of different writes can be merged together.

For reads this means that a location can be pre-fetched or can be fetched just once for multiple read transactions.

To determine if a transaction should be cached this bit should be used in conjunction with the *Read Allocate* (RA) and *Write Allocate* (WA) bits.

### **Read Allocate (RA) bit, ARCACHE[2] and AWCACHE[2]**

When the RA bit is HIGH, it means that if the transfer is a read and it misses in the cache then it should be allocated.

The RA bit must not be HIGH if the C bit is low.

### **Write Allocate (WA) bit, ARCACHE[3] and AWCACHE[3]**

When the WA bit is HIGH, it means that if the transfer is a write and it misses in the cache then it should be allocated.

The WA bit must not be HIGH if the C bit is low.

Table 5-1 shows the encoding of the **ARCACHE[3:0]** and **AWCACHE[3:0]** signals.

**Table 5-1 Cache encoding**

<b>ARCACHE[3:0] AWCACHE[3:0]</b>				<b>Transaction attributes</b>
<b>WA</b>	<b>RA</b>	<b>C</b>	<b>B</b>	
0	0	0	0	Noncacheable and nonbufferable
0	0	0	1	Bufferable only
0	0	1	0	Cacheable, but do not allocate
0	0	1	1	Cacheable and bufferable, but do not allocate
0	1	0	0	Reserved
0	1	0	1	Reserved
0	1	1	0	Cacheable write-through, allocate on reads only
0	1	1	1	Cacheable write-back, allocate on reads only
1	0	0	0	Reserved
1	0	0	1	Reserved
1	0	1	0	Cacheable write-through, allocate on writes only
1	0	1	1	Cacheable write-back, allocate on writes only
1	1	0	0	Reserved
1	1	0	1	Reserved
1	1	1	0	Cacheable write-through, allocate on both reads and writes
1	1	1	1	Cacheable write-back, allocate on both reads and writes

In the case of write transactions, the **AWCACHE** signal can be used to determine which component provides the write response. If a write transaction is indicated as bufferable then it is acceptable for a bridge or system level cache to provide the write response. If, however, the transaction is indicated as being non-bufferable then the write response must be provided from the final destination of the transaction.

The AXI protocol does not determine the mechanism by which buffered or cached data reaches its destination. For example, a system-level cache might have a controller to manage cleaning, flushing, and invalidating cache entries. Another example is a bridge containing a write buffer, which might have control logic to drain the buffer if it receives a nonbufferable write with a matching transaction ID.

## 5.2 Protection unit support

To support complex system designs, it is often necessary for both the interconnect and other devices in the system to provide protection against illegal transactions. The **AWPROT** or **ARPROT** signal gives three levels of access protection:

### Normal or privileged, ARPROT[0] and AWPROT[0]

- LOW indicates a normal access
- HIGH indicates a privileged access.

This is used by some masters to indicate their processing mode. A privileged processing mode typically has a greater level of access within a system.

### Secure or non-secure, ARPROT[1] and AWPROT[1]

- LOW indicates a secure access
- HIGH indicates a non-secure access.

This is used in systems where a greater degree of differentiation between processing modes is required.

————— Note —————

This bit is configured so that when it is HIGH then the transaction is considered non-secure and when LOW, the transaction is considered as secure.

### Instruction or data, ARPROT[2] and AWPROT[2]

- LOW indicates a data access
- HIGH indicates an instruction access.

This bit gives an indication if the transaction is an instruction or a data access.

————— Note —————

This indication is provided as a hint and is not accurate in all cases. For example, where a transaction contains a mix of instruction and data items. It is recommended that, by default, an access is marked as a data access unless it is specifically known to be an instruction access.

Table 5-2 summarizes the encoding of the **ARPROT[2:0]** and **AWPROT[2:0]** signals.

**Table 5-2 Protection encoding**

<b>ARPROT[2:0] AWPROT[2:0]</b>	<b>Protection level</b>
[0]	1 = privileged access 0 = normal access
[1]	1 = nonsecure access 0 = secure access
[2]	1 = instruction access 0 = data access

# Chapter 6

## Atomic Accesses

This chapter describes how the AXI protocol implements exclusive access and locked access mechanisms. It contains the following sections:

- *About atomic accesses* on page 6-2
- *Exclusive access* on page 6-3
- *Locked access* on page 6-7.

## 6.1 About atomic accesses

To enable the implementation of atomic access primitives, the **ARLOCK[1:0]** or **AWLOCK[1:0]** signal provides exclusive access and locked access. Table 6-1 shows the encoding of the **ARLOCK[1:0]** and **AWLOCK[1:0]** signals.

**Table 6-1 Atomic access encoding**

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

## 6.2 Exclusive access

The exclusive access mechanism enables the implementation of semaphore type operations without requiring the bus to remain locked to a particular master for the duration of the operation. The advantage of exclusive access is that semaphore type operations do not impact either the critical bus access latency or the maximum achievable bandwidth.

The **ARLOCK[1:0]** or **AWLOCK[1:0]** signal selects exclusive access, and the **RRESP[1:0]** or **BRESP[1:0]** signal (see Table 7-1 on page 7-2) indicates the success or failure of the exclusive access.

The slave must have additional logic to support exclusive access. The AXI protocol provides a fail-safe mechanism to indicate when a master attempts an exclusive access to a slave that does not support it.

### 6.2.1 Exclusive access process

The basic process for an exclusive access is:

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.

**Note**

A master might not complete the write portion of an exclusive operation. The exclusive access monitoring hardware must monitor only one address per transaction ID. Therefore, if a master does not complete the write portion of an exclusive operation, a subsequent exclusive read changes the address that is being monitored for exclusivity.

### 6.2.2 Exclusive access from the perspective of the master

A master starts an exclusive operation by performing an exclusive read. This usually returns the EXOKAY response from the slave, indicating that the slave recorded the address to be monitored.

---

**Note**

If the master attempts an exclusive read from a slave that does not support exclusive accesses, the slave returns the OKAY response instead of the EXOKAY response. The master can treat this as an error condition indicating that the exclusive access is not supported. It is recommended that the master not perform the write portion of this exclusive operation.

---

At some time after the exclusive read, the master tries an exclusive write to the same location. If the location has not changed since the exclusive read, the exclusive write operation succeeds. The slave returns the EXOKAY response, and the exclusive write updates the memory location.

If the address location has changed since the exclusive read, the exclusive write attempt fails, and the slave returns the OKAY response instead of the EXOKAY response. The exclusive write attempt does not update the memory location.

A master might not complete the write portion of an exclusive operation. If this happens, the slave continues to monitor the address for exclusivity until another exclusive read initiates a new exclusive access.

A master must not commence the write portion of an exclusive access until the read portion is complete.

### 6.2.3 Exclusive access from the perspective of the slave

A slave that is not capable of supporting exclusive accesses can ignore the **ARLOCK[1:0]** and **AWLOCK[1:0]** signals. It must provide an OKAY response for both normal and exclusive accesses.

A slave that supports exclusive access must have monitor hardware. It is recommended that such a slave has a monitor unit for each exclusive-capable master ID that can access it. A single-ported slave can have a standard exclusive access monitor external to the slave, but multiported slaves might require internal monitoring.

The exclusive access monitor records the address and **ARID** value of any exclusive read operation. Then it monitors that location until either a write occurs to that location or until another exclusive read with the same **ARID** value resets the monitor to a different address.

When an exclusive write occurs with a given **AWID** value then the monitor checks to see if that address is being monitored for exclusivity. If it is, then this implies that no write has occurred to that location, and the exclusive write proceeds, completing the exclusive access. The slave returns the EXOKAY response to the master.

If the address is no longer being monitored at the time of an exclusive write, this implies one of the following:

- the location has been updated since the exclusive read
- the monitor has been reset to another location.

In both cases the exclusive write must not update the address location, and the slave must return the OKAY response instead of the EXOKAY response.

#### 6.2.4 Exclusive access restrictions

The following restrictions apply to exclusive accesses:

- The size and length of an exclusive write with a given ID must be the same as the size and length of the preceding exclusive read with the same ID.
- The address of an exclusive access must be aligned to the total number of bytes in the transaction.
- The address for the exclusive read and the exclusive write must be identical.
- The **ARID** field of the read portion of the exclusive access must match the **AWID** of the write portion.
- The control signals for the read and write portions of the exclusive access must be identical.
- The number of bytes to be transferred in an exclusive access burst must be a power of 2, that is, 1, 2, 4, 8, 16, 32, 64, or 128 bytes.
- The maximum number of bytes that can be transferred in an exclusive burst is 128.
- The value of the **ARCACHE[3:0]** or **AWCACHE[3:0]** signals must guarantee that the slave that is monitoring the exclusive access sees the transaction. For example, an exclusive access being monitored by a slave must not have an **ARCACHE[3:0]** or **AWCACHE[3:0]** value that indicates that the transaction is cacheable.

Failure to observe these restrictions causes Unpredictable behavior.

The minimum number of bytes to be monitored during an exclusive operation is defined by the length and size of the transaction. It is acceptable to monitor a larger number of bytes, up to 128 which is the maximum of an exclusive access. However, this might result in occasions when the exclusive access is actually successful but is indicated as failing because a neighboring byte was updated.

## 6.2.5 Slaves that do not support exclusive access

The response signals, **BRESP[1:0]** and **RRESP[1:0]**, include an OKAY response for successful normal accesses and an EXOKAY response for successful exclusive accesses. This means that a slave that does not support exclusive accesses can provide an OKAY response to indicate the failure of an exclusive access.

———— Note ————

An exclusive write to a slave that does not support exclusive access always updates the memory location.

An exclusive write to a slave that supports exclusive access updates the memory location only if the exclusive write is successful.

---

## 6.3 Locked access

When the **ARLOCK[1:0]** or **AWLOCK[1:0]** signals for a transaction show that it is a locked transfer then the interconnect must ensure that only that master is allowed access to the slave region until an unlocked transfer from the same master completes. The arbiter within the interconnect is used to enforce this restriction.

When a master starts a locked sequence of either read or write transactions it must ensure that it has no other outstanding transactions waiting to complete.

Any transaction with **ARLOCK[1:0]** or **AWLOCK[1:0]** set to indicate a locked sequence forces the interconnect to lock the following transaction. Therefore, a locked sequence must always complete with a final transaction that does not have **ARLOCK[1:0]** or **AWLOCK[1:0]** set to indicate a locked access. This final transaction is included in the locked sequence and effectively removes the lock.

When completing a locked sequence a master must ensure that all previous locked transactions are complete before issuing the final unlocking transaction. It must then ensure that the final unlocking transaction has fully completed before any further transactions are commenced.

The master must ensure that all transactions within a locked sequence have the same **ARID** or **AWID** value.

---

— Note —

---

Locked accesses require that the interconnect prevents any other transactions occurring while the locked sequence is in progress and can therefore have an impact on the interconnect performance. It is recommended that locked accesses are only used to support legacy devices.

---

The following restrictions are recommended but not mandatory:

- keep all locked transaction sequences within the same 4KB address region
- limit locked transaction sequences to two transactions.



# Chapter 7

## Response Signaling

This chapter describes the four slave responses in AXI read and write transactions. It contains the following sections:

- *About response signaling* on page 7-2
- *Response types* on page 7-4.

## 7.1 About response signaling

The AXI protocol allows response signalling for both read and write transactions. For read transactions the response information from the slave is passed alongside the read data itself, however for writes the response information is conveyed along the write response channel.

The AXI protocol responses are:

- OKAY
- EXOKAY
- SLVERR
- DECERR.

Table 7-1 shows the encoding of the **RRESP[1:0]** and **BRESP[1:0]** signals.

**Table 7-1 RRESP[1:0] and BRESP[1:0] encoding**

RRESP[1:0] BRESP[1:0]	Response	Meaning
b00	OKAY	Normal access okay indicates if a normal access has been successful. Can also indicate an exclusive access failure.
b01	EXOKAY	Exclusive access okay indicates that either the read or write portion of an exclusive access has been successful.
b10	SLVERR	Slave error is used when the access has reached the slave successfully, but the slave wishes to return an error condition to the originating master.
b11	DECERR	Decode error is generated typically by an interconnect component to indicate that there is no slave at the transaction address.

For a write transaction, there is just one response given for the entire burst and not for each data transfer within the burst.

In a read transaction, the slave can give different responses for different transfers within a burst. In a burst of 16 read transfers, for example, the slave might return an OKAY response for 15 of the transfers and a SLVERR response for one of the transfers.

The protocol defines that the required number of data transfers must be performed, even if an error is reported. For example, if a read of 8 transfers is requested from a slave but the slave has an error condition then the slave must perform 8 data transfers, each with an error response. The remainder of the burst is not cancelled if the slave gives a single error response.

This protocol places restrictions on masters that can issue multiple outstanding addresses and that must also support precise error signaling. Such masters must be able to handle an error response for an earlier transfer while later transfers are already underway.

## 7.2 Response types

This section describes the four AXI protocol response types:

- *Normal access success*
- *Exclusive access*
- *Slave error*
- *Decode error.*

### 7.2.1 Normal access success

The OKAY response indicates:

- the success of a normal access
- the failure of an exclusive access
- an exclusive access to a slave that does not support exclusive access.

OKAY is the response for most transactions.

### 7.2.2 Exclusive access

The EXOKAY response indicates the success of an exclusive access. Chapter 6 *Atomic Accesses* describes this response.

### 7.2.3 Slave error

The SLVERR response indicates an unsuccessful transaction. Examples of slave error conditions are:

- FIFO/buffer overrun or underrun condition
- unsupported transfer size attempted
- write access attempted to read-only location
- timeout condition in the slave
- access attempted to an address where no registers are present
- access attempted to a disabled or powered-down function.

To simplify system monitoring and debugging, it is recommended that error responses are used only for error conditions and not for signaling normal, expected events.

### 7.2.4 Decode error

In a system without a fully-decoded address map, there can be addresses at which there are no slaves to respond to a transaction. In such a system, the interconnect must provide a suitable error response to flag the access as illegal and also to prevent the system from locking up by trying to access a nonexistent slave.

When the interconnect cannot successfully decode a slave access, it effectively routes the access to a default slave, and the default slave returns the DECERR response.

An implementation option is to have the default slave also record the details of decode errors for later determination of how the errors occurred. In this way, the default slave can significantly simplify the debugging process.

The AXI protocol requires that all data transfers for a transaction are completed, even if an error condition occurs. Therefore any component giving a DECERR response must meet this requirement.



# Chapter 8

## Ordering Model

This chapter describes how the AXI protocol uses transaction ID tags to enable the issuing of multiple outstanding addresses and out-of-order transaction processing. It contains the following sections:

- *About the ordering model* on page 8-2
- *Transfer ID fields* on page 8-3
- *Read ordering* on page 8-4
- *Normal write ordering* on page 8-5
- *Write data interleaving* on page 8-6
- *Read and write interaction* on page 8-8
- *Interconnect use of ID fields* on page 8-9
- *Recommended width of ID fields* on page 8-10.

## 8.1 About the ordering model

The AXI protocol enables out-of-order transaction completion and the issuing of multiple outstanding addresses. These features enable the implementation of a high-performance interconnect, maximizing data throughput and system efficiency.

The ID signals support out-of-order transactions by enabling each port to act as multiple ordered ports. All transactions with a given ID must be ordered, but there is no restriction on the ordering of transactions with different IDs. The five transaction IDs are:

- AWID** The ID tag for the write address group of signals.
- WID** The write ID tag for a write transaction. Along with the write data, the master transfers a **WID** to match the **AWID** of the corresponding address.
- BID** The ID tag for the write response. The slave transfers a **BID** to match the **AWID** and **WID** of the transaction to which it is responding.
- ARID** The ID tag for the read address group of signals.
- RID** The read ID tag for a read transaction. The slave transfers an **RID** to match the **ARID** of the transaction to which it is responding.

### — Note —

There is no requirement for slaves and masters to use these advanced features. Simple masters and slaves can process one transaction at a time in the order they are issued.

The ability to issue multiple outstanding addresses means that masters can issue transaction addresses without waiting for earlier transactions to complete. This feature can improve system performance because it enables parallel processing of transactions.

The ability to complete transactions out of order means that transactions to faster memory regions can complete without waiting for earlier transactions to slower memory regions. This feature can also improve system performance because it reduces the effect of transaction latency.

### — Note —

The reordering of transactions is always with respect to other transactions. There is no facility for the reordering of data transfers within a burst. The address and control signals that define the burst control the order of transfers within the burst.

## 8.2 Transfer ID fields

The AXI protocol provides an ID field to enable a master to issue a number of separate transactions, each of which must be returned in order.

A master can use the **ARID** or **AWID** field of a transaction to provide additional information about the ordering requirements of the master. The rules governing the ordering of transactions are as follows:

- Transactions from different masters have no ordering restrictions. They can complete in any order.
- Transactions from the same master, but with different ID values, have no ordering restrictions. They can complete in any order.
- The data for a sequence of write transactions with the same **AWID** value must complete in the same order that the master issued the addresses in.
- The data for a sequence of read transactions with the same **ARID** value must be returned in order that:
  - when reads with the same **ARID** are from the same slave then the slave must ensure that the read data returns in the same order that the addresses are received.
  - when reads with the same **ARID** are from different slaves, the interconnect must ensure that the read data returns in the same order that the master issued the addresses in.
- There are no ordering restrictions between read and write transactions with the same **AWID** and **ARID**. If a master requires an ordering restriction then it must ensure that the first transaction is fully completed before the second transaction is issued.

## 8.3 Read ordering

At a master interface, read data from read transactions with the same **ARID** value must arrive in the same order in which the master issued the addresses. Data from read transactions with different **ARID** values can return in any order and it is also acceptable to interleave the read data of transactions with different **ARID** fields.

A slave must return read data from a sequence of read transactions with the same **ARID** value in the same order in which it received the addresses. In a sequence of read transactions with different **ARID** values, the slave can return the read data in a different order than that in which the transactions arrived.

The slave must ensure that the **RID** value of any returned read data matches the **ARID** value of the address to which it is responding.

The interconnect must ensure that a sequence of read transactions with the same **ARID** value from different slaves complete in order.

The read data reordering depth is the number of addresses pending in the slave that can be reordered. A slave that processes all transactions in order has a read data reordering depth of one. The read data reordering depth is a static value that must be specified by the designer of the slave.

## 8.4 Normal write ordering

If a slave does not support write data interleaving (see *Write data interleaving* on page 8-6), the master must issue the data of write transactions in the same order in which it issues the transaction addresses.

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 the interconnect combines write transactions from different masters to one slave, it must ensure that it combines the write data in address order.

These restrictions apply even if the write transactions have different **AWID** values.

## 8.5 Write data interleaving

Write data interleaving enables a slave interface to accept interleaved write data with different **AWID** values. The slave declares a write data interleaving depth that indicates if the interface can accept interleaved write data from sources with different **AWID** values. The write data interleaving depth is statically configured. By default, the write data interleaving depth of any interface is one.

— Note —

It is not permitted to interleave the write data of different transactions that have the same **AWID**.

The write data interleaving depth is the number of different addresses that are currently pending in the slave interface for which write data can be supplied. For example, a slave with a write data interleaving depth of two that has four different addresses, all with different **AWID** values, pending can accept data for either of the first two pending addresses.

The order in which a slave receives the first data item of each transaction must be the same as the order in which it receives the addresses for the transactions.

Write data interleaving can prevent stalling when the interconnect combines multiple streams of write data destined for the same slave. The interconnect might combine one write data stream from a slow source and another write data stream from a fast source. By interleaving the two write data streams, the interconnect can improve system performance.

— Note —

If two write transactions with different **AWID** values access the same or overlapping address locations then the processing order is not defined. A higher-level protocol must ensure the correct order of transaction processing.

A master interface that is capable of generating write data with only one **AWID** value generates all write data in the same order in which it issues the addresses. However, a master interface can interleave write data with different **WID** values if the slave interface has a write data interleaving depth greater than one.

For most masters that can internally control the generation of the write data, write data interleaving is not necessary. Such a master can generate the write data in the same order in which it generates the addresses. However, a master interface that is passing write data from multiple sources with different speeds can interleave the sources to make maximum use of the interconnect.

To avoid a deadlock situation, a slave interface must have a write interleaving depth greater than one only if it can continuously accept interleaved write data. The slave interface must never stall the acceptance of write data in an attempt to change the order of the write data.

## 8.6 Read and write interaction

There are no ordering restrictions between read and write transactions and they are allowed to complete in any order.

If a master requires a given relationship between read and write transaction then it must ensure that the earlier transaction is complete before issuing the later transaction. In the case of reads the earlier transaction can be considered complete when the last read data is returned to the master. In the case of writes the transaction can only be considered complete when the write response is received by the master, it is not acceptable to consider the write transaction complete when all the write data is sent.

For address regions occupied by peripherals this typically means waiting for earlier transactions to complete when switching between read and write transactions that require an ordering restriction.

For memory regions, it is possible for a master to implement an address check against outstanding transactions, to determine if a new transaction could be to the same, or overlapping, address region. If the transactions do not overlap then the new transaction can commence without waiting for earlier transactions to complete.

## 8.7 Interconnect use of ID fields

When a master interface is connected to an interconnect, the interconnect appends additional bits to the **ARID**, **AWID** and **WID** fields that are unique to that master port. This has two effects:

- masters do not have to know what ID values are used by other masters, because the interconnect makes the ID values unique when it appends the master number to the field
- the width of the ID field at a slave interface is wider than the ID field at a master interface.

For read data, the interconnect uses the additional bits of the **RID** field to determine which master port the read data is destined for. The interconnect removes these bits of the **RID** field before passing the **RID** value to the correct master port.

## 8.8 Recommended width of ID fields

To take advantage of the AXI out-of-order transaction capability, use the following recommendations:

- implement a transaction ID up to four bits in master components
- implement up to four additional bits of transaction ID for master port numbers in the interconnect
- implement eight bits of ID support in slave components.

———— **Note** ————

For masters that support only a single ordered interface, it is acceptable to tie the ID outputs to a constant value, such as 0.

---

For slaves which do not make use of the ordering information and simply process all transactions in order, it is possible to use a standard off-the-shelf module to add the ID functionality to the slave, therefore making it possible to design the base functionality of the slave without the ID signaling present.

# Chapter 9

## Data Buses

This chapter describes transfers of varying sizes on the AXI read and write data buses and how the interface uses byte-invariant endianness to handle mixed-endian transfers. It contains the following sections:

- *About the data buses* on page 9-2
- *Write strobes* on page 9-3
- *Narrow transfers* on page 9-4
- *Byte invariance* on page 9-5.

## 9.1 About the data buses

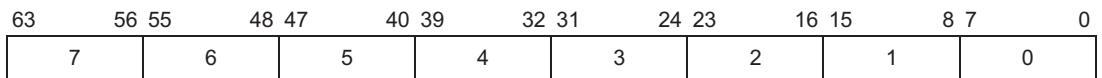
The AXI protocol has two independent data buses, one for read data and one for write data. Because these data buses have their own individual handshake signals, it is possible for data transfers to occur on both buses at the same time.

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

The write strobe signals, **WSTRB**, enable sparse data transfer on the write data bus. Each write strobe signal 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.

There is one write strobe for each eight bits of the write data bus, so **WSTRB[n]** corresponds to **WDATA[(8 × n) + 7: (8 × n)]**. Figure 9-1 shows this relationship on a 64-bit data bus.



**Figure 9-1 Byte lane mapping**

A master must ensure that the write strobes are asserted only for byte lanes that can contain valid data as determined by the control information for the transaction.

## 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 bursts, different byte lanes transfer the data on each beat of the burst. In a fixed burst, the address remains constant, and the byte lanes that can be used also remain constant.

Figure 9-2 and Figure 9-3 give two examples 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.

Byte lane used			
			DATA[7:0]
		DATA[15:8]	
	DATA[23:16]		
DATA[31:24]			
			DATA[7:0]

1st transfer  
2nd transfer  
3rd transfer  
4th transfer  
5th transfer

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

In Figure 9-3:

- the burst has three transfers
- the starting address is 4
- each transfer is 32 bits
- the transfers are on a 64-bit bus.

Byte lane used	
DATA[63:32]	
	DATA[31:0]
DATA[63:32]	

1st transfer  
2nd transfer  
3rd transfer

**Figure 9-3 Narrow transfer example with 32-bit transfers**

## 9.4 Byte invariance

To access mixed-endian data structures that reside in the same memory space, the AXI protocol uses a byte-invariant endian scheme.

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

Components that have only one transfer width must have their byte lanes connected to the appropriate byte lanes of the data bus. Components that support multiple transfer widths might require a more complex interface to convert an interface that is not naturally byte-invariant.

Most little-endian components can connect directly to a byte-invariant interface. Components that support only big-endian transfers require a conversion function for byte-invariant operation.

Figure 9-4 is an example of a data structure requiring byte-invariant access. It is possible that the header information, such as the source and destination identifiers, is in little-endian format, but the payload is a big-endian byte stream.

31	24 23	8 7	0
Desti-	Source	Packet	
Checksum			-nation
Payload		Data items	
Payload			Payload
Payload			Payload
Payload			Payload

**Figure 9-4 Example mixed-endian data structure**

Byte invariance ensures that little-endian access to parts of the header information does not corrupt other big-endian data within the structure.



# Chapter 10

## Unaligned Transfers

This chapter describes how the AXI protocol handles unaligned transfers. It contains the following sections:

- *About unaligned transfers* on page 10-2
- *Examples* on page 10-3.

## 10.1 About unaligned transfers

The AXI protocol uses burst-based addressing, which means that each transaction consists of a number of data transfers. 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.

For any burst that is made up of data transfers wider than one byte, it is possible that the first bytes that have to be 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.

The AXI protocol enables a master to use the low-order address lines to signal an unaligned start address for a burst. The information on the low-order address lines must be consistent with the information contained on the byte lane strobes.

### — Note —

The AXI protocol does not require the slave to take special action based on any alignment information from the master.

The master can also simply provide an aligned address and, in a write transaction, rely on the byte lane strobes to provide the information about which byte lanes the data is using.

## 10.2 Examples

Figure 10-1, Figure 10-2 on page 10-4, and Figure 10-3 on page 10-4 show 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.

	31	24	23	16	15	8	7	0	
Address:	0x00	3	2	1	0				1st transfer
Transfer size:	32 bits	7	6	5	4				2nd transfer
Burst type:	incrementing	B	A	9	8				3rd transfer
Burst length:	4 transfers	F	E	D	C				4th transfer
Address:	0x01	3	2	1	0				1st transfer
Transfer size:	32 bits	7	6	5	4				2nd transfer
Burst type:	incrementing	B	A	9	8				3rd transfer
Burst length:	4 transfers	F	E	D	C				4th transfer
Address:	0x01	3	2	1	0				1st transfer
Transfer size:	32 bits	7	6	5	4				2nd transfer
Burst type:	incrementing	B	A	9	8				3rd transfer
Burst length:	5 transfers	F	E	D	C				4th transfer
	13	12	11	10					5th transfer
Address:	0x07	7	6	5	4				1st transfer
Transfer size:	32 bits	B	A	9	8				2nd transfer
Burst type:	incrementing	F	E	D	C				3rd transfer
Burst length:	5 transfers	13	12	11	10				4th transfer
	17	16	15	14					5th transfer

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

Figure 10-2 on page 10-4 shows three bursts of 32-bit transfers on a 64-bit bus.

## Unaligned Transfers

	63	56	55	48	47	40	39	32	31	24	23	16	15	8	7	0	
Address: 0x00	7	6	5	4	3	2	1	0									1st transfer
Transfer size: 32 bits	7	6	5	4	3	2	1	0									2nd transfer
Burst type: incrementing	F	E	D	C	B	A	9	8									3rd transfer
Burst length: 4 transfers	F	E	D	C	B	A	9	8									4th transfer
Address: 0x07	7	6	5	4	3	2	1	0									1st transfer
Transfer size: 32 bits	F	E	D	C	B	A	9	8									2nd transfer
Burst type: incrementing	F	E	D	C	B	A	9	8									3rd transfer
Burst length: 4 transfers	17	16	15	14	13	12	11	10									4th transfer
Address: 0x07	7	6	5	4	3	2	1	0									1st transfer
Transfer size: 32 bits	F	E	D	C	B	A	9	8									2nd transfer
Burst type: incrementing	F	E	D	C	B	A	9	8									3rd transfer
Burst length: 5 transfers	17	16	15	14	13	12	11	10									4th transfer
	17	16	15	14	13	12	11	10									5th transfer

**Figure 10-2 Aligned and unaligned word transfers on a 64-bit bus**

Figure 10-3 shows a wrapping burst of 32-bit transfers on a 64-bit bus.

	63	56	55	48	47	40	39	32	31	24	23	16	15	8	7	0	
Address: 0x04	7	6	5	4	3	2	1	0									1st transfer
Transfer size: 32 bits	F	E	D	C	B	A	9	8									2nd transfer
Burst type: wrapping	F	E	D	C	B	A	9	8									3rd transfer
Burst length: 4 transfers	7	6	5	4	3	2	1	0									4th transfer

**Figure 10-3 Aligned wrapping word transfers on a 64-bit bus**

# Chapter 11

## Clock and Reset

This chapter describes the timing of the AXI clock and reset signals. It contains the following section:

- *Clock and reset requirements* on page 11-2.

## 11.1 Clock and reset requirements

This section gives the requirements for implementing the **ACLK** and **ARESETn** signals.

### 11.1.1 Clock

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

There must be no combinatorial paths between input and output signals on both master and slave interfaces.

### 11.1.2 Reset

The AXI protocol includes a single active LOW reset signal, **ARESETn**. The reset signal can be asserted asynchronously, but deassertion must be synchronous after the rising edge of **ACLK**.

During reset the following interface requirements apply:

- a master interface must drive **ARVALID**, **AWVALID**, and **WVALID** LOW
- a slave interface must drive **RVALID** and **BVALID** LOW.

All other signals can be driven to any value.

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 **ARVALID**, **AWVALID**, or **WVALID**, can be driven HIGH.

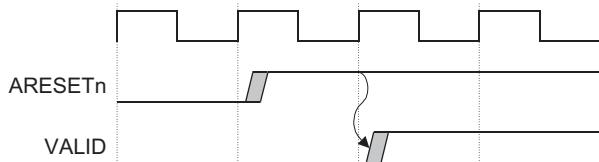


Figure 11-1 Exit from reset

# Chapter 12

## Low-power Interface

This chapter describes the AXI protocol clock control interface during entry into and exit from a low-power state. It contains the following sections:

- *About the low-power interface* on page 12-2
- *Low-power clock control* on page 12-3.

## 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 that require a power-down sequence, and that can have their clocks turned off only after they enter a low-power state. These peripherals require an indication from a system clock controller to determine when to initiate the power-down sequence.
- Peripherals that have no power-down sequence, and that can independently indicate when it is acceptable to turn off their clocks.

## 12.2 Low-power clock control

The low-power clock control interface consists of the following signals:

- a signal from the peripheral indicating when its clocks can be enabled or disabled
- two handshake signals for the system clock controller to request exit or entry into a low-power state.

The primary signal in the clock control interface is **CACTIVE**. The peripheral uses this signal to indicate when it requires its clock to be enabled. The peripheral asserts **CACTIVE** to indicate that it requires the clock, and the system clock controller must enable the clock immediately. The peripheral deasserts **CACTIVE** to indicate that it does not require the clock. The system clock controller can then determine whether to enable or disable the peripheral clock.

A peripheral that can have its clock enabled or disabled at any time can drive **CACTIVE** LOW permanently. A peripheral that must have its clock always enabled must drive **CACTIVE** HIGH permanently.

This simple interface to the system clock controller is sufficient for some peripherals with no power-down or power-up sequence.

For a more complex peripheral with a power-down or power-up sequence, entry into a low-power state occurs only after a request from the system clock controller. The AXI protocol provides a two-wire request/acknowledge handshake to support this request:

**CSYSREQ** To request that the peripheral enter a low-power state, the system clock controller drives the **CSYSREQ** signal LOW. During normal operation, **CSYSREQ** is HIGH.

**CSYSACK** The peripheral uses the **CSYSACK** signal to acknowledge both the low-power state request and the exit from the low-power state.

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

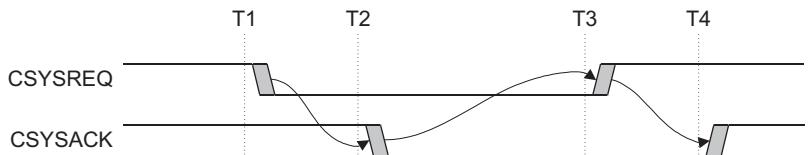


Figure 12-1 CSYSREQ and CSYSACK handshake

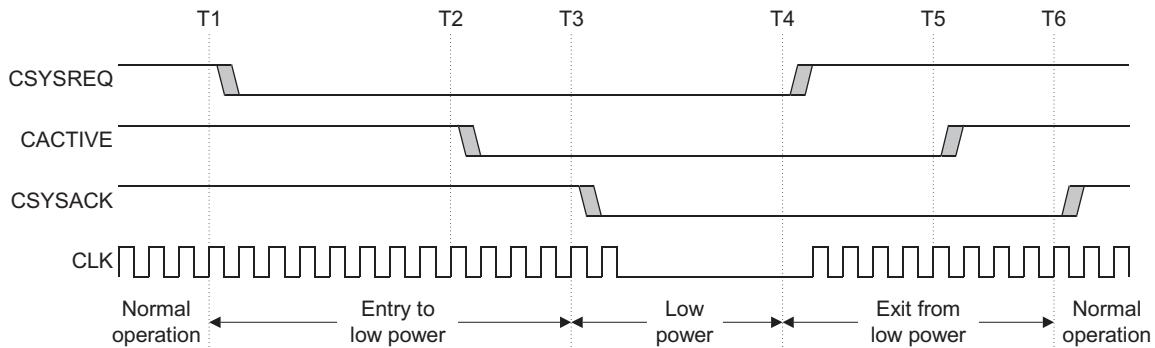
At the start of the sequence in Figure 12-1 on page 12-3, both **CSYSREQ** and **CSYSACK** are HIGH for normal clocked operation. At time T1, the system clock controller deasserts **CSYSREQ**, indicating a request to put the peripheral in a low-power state. The peripheral acknowledges the request at time T2 by deasserting **CSYSACK**. At T3, the system clock controller asserts **CSYSREQ** to indicate the exit from the low-power state, and the peripheral asserts **CSYSACK** at T4 to acknowledge the exit.

This relationship between **CSYSREQ** and **CSYSACK** is a requirement of the AXI protocol.

The peripheral can accept or deny the request for a low-power state from the system clock controller. The level of the **CACTIVE** signal when the peripheral acknowledges the request by deasserting **CSYSACK** indicates the acceptance or denial of the request.

### 12.2.1 Acceptance of low-power request

Figure 12-2 shows the sequence of events when a peripheral accepts a system low-power request.



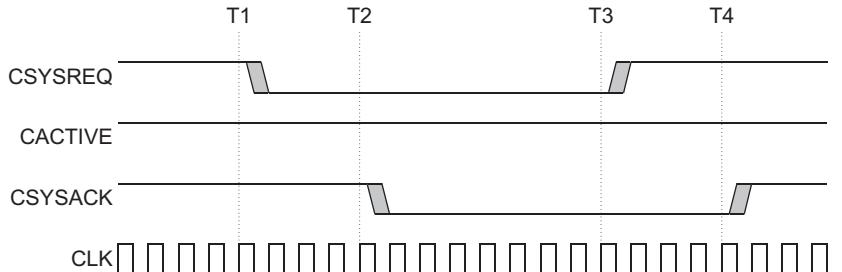
**Figure 12-2 Acceptance of a low-power request**

In Figure 12-2, the sequence begins at T1 when the system clock controller deasserts **CSYSREQ** to request that the peripheral enter a low power state. After the peripheral recognizes the request, it can then perform its power-down function and deassert **CACTIVE**. The peripheral then deasserts **CSYSACK** at T3 to complete the entry into the low-power state.

At T4, the system clock controller begins the low-power state exit sequence by asserting **CSYSREQ**. The peripheral then asserts **CACTIVE** at T5 and completes the exit sequence at T6 by asserting **CSYSACK**.

### 12.2.2 Denial of a low-power request

Figure 12-3 shows the sequence of events when a peripheral denies a system low-power request.



**Figure 12-3 Denial of a low-power request**

In Figure 12-3, the peripheral denies a low-power request by holding **CACTIVE** HIGH when it acknowledges the low-power request. After that point, the system clock controller must complete the low-power request handshake by asserting **CSYSREQ** before it can initiate another request.

### 12.2.3 Exiting a low-power state

Either the system clock controller or the peripheral can request to exit the low-power state and restore the clock. By definition, both **CACTIVE** and **CSYSREQ** are LOW during the low power state, and driving either of these signals HIGH initiates the exit sequence.

The system clock controller can initiate the exit from the low-power state by enabling the clock and driving **CSYSREQ** HIGH. The peripheral can then perform a power-up sequence in which it drives **CACTIVE** HIGH. Then it completes the exit by driving **CSYSACK** HIGH.

The peripheral can initiate the exit from a low-power state by driving **CACTIVE** HIGH. The system clock controller must then immediately restore the clock. It must also drive **CSYSREQ** HIGH to continue the handshake sequence. The peripheral then completes the sequence by driving **CSYSACK** HIGH while exiting the low-power state. The peripheral can keep **CSYSACK** LOW for as many cycles as it requires to complete the exit sequence.

### 12.2.4 Clock control sequence summary

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

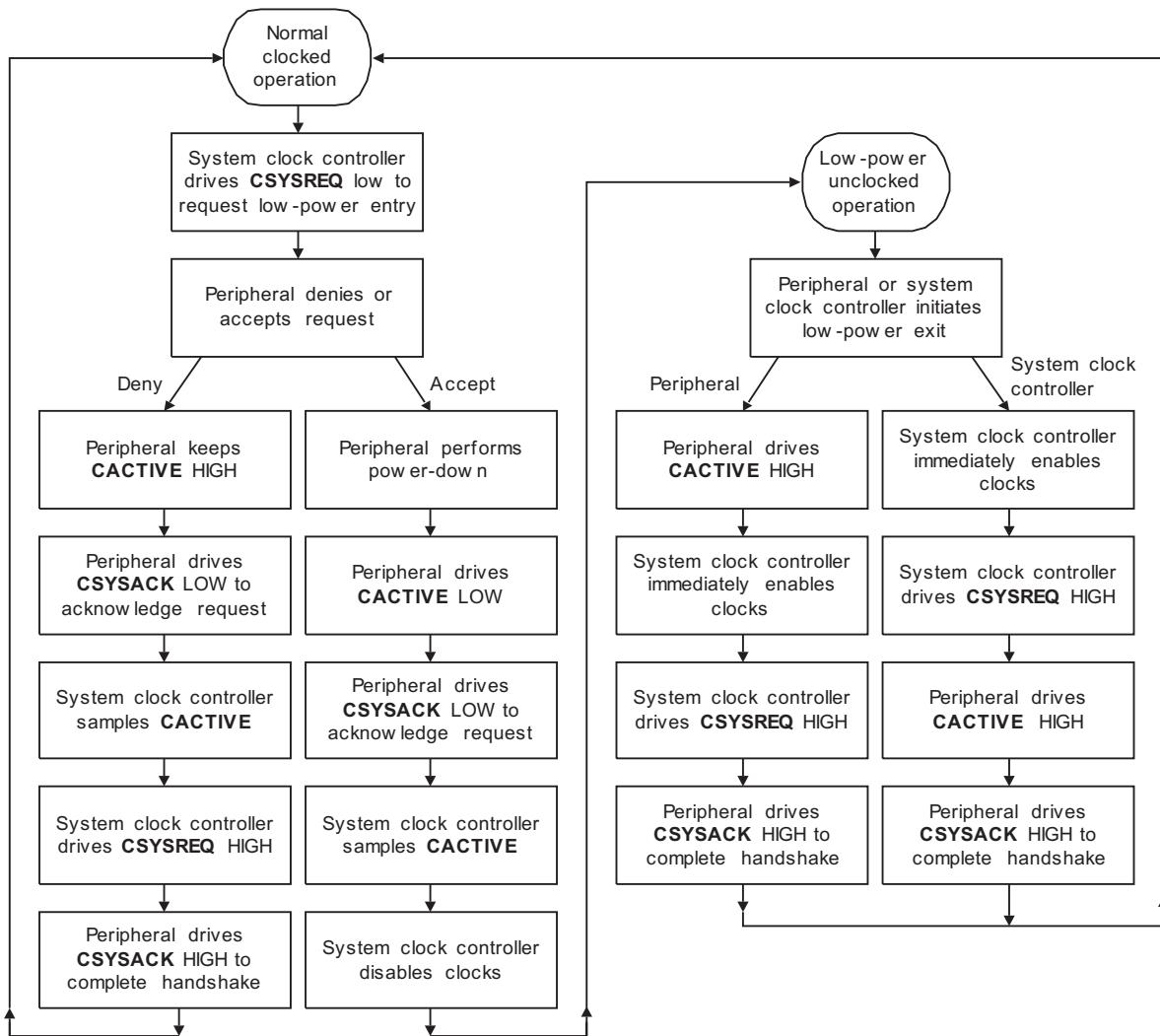


Figure 12-4 Low-power clock control sequence

### 12.2.5 Combining peripherals in a low-power domain

The system clock controller can combine a number of different peripherals within the same low-power clock domain. Then the clock domain can be treated in the same way as a single peripheral if the following rules are observed:

- The clock domain **CACTIVE** signal is the logical OR of all the **CACTIVE** signals within the clock domain. This means that the system clock controller can disable the clocks only when all peripherals indicate that they can be disabled.
- The system clock controller can use a single **CSYSREQ** signal that is routed to all peripherals within the clock domain.
- The clock domain **CSYSACK** signal is generated as follows:
  - the falling edge of **CSYSACK** occurs when the last falling edge from all of the peripherals occurs
  - the rising edge of **CSYSACK** occurs when the last rising edge from all of the peripherals occurs.



# Index

The items in this index are listed in alphabetical order with references to page numbers.

## A

ACLK  
    description 2-2  
Address channel 1-3  
    definition 1-4  
    handshake 1-4, 3-2  
Address ID tag  
    *see* ARID and AWID  
Addressing options 4-2  
Allocate attribute 1-11  
AMBA  
    architecture xiv  
    interface 1-2  
    Specification xvii  
ARADDR 2-6  
    timing example 1-7, 1-8  
ARBURST 2-6  
    encoding 4-5  
ARCACHE 2-6  
    encoding 5-3  
    in exclusive accesses 6-5

ARESETn  
    description 2-2  
    timing 11-2  
ARID 2-6  
    in exclusive accesses 6-4, 6-5  
    out-of-order transactions 8-2  
    uniqueness 8-9  
ARLEN 2-6  
    encoding 4-3  
ARLOCK 2-6  
    encoding 6-2  
ARPROT 2-6  
    encoding 5-6  
ARREADY 2-6  
    timing example 1-7, 1-8  
ARSIZE 2-6  
    encoding 4-4  
ARVALID 2-6  
    reset 11-2  
    timing example 1-7, 1-8  
Atomic access encoding 6-2  
AWADDR 2-3  
    timing example 1-9  
AWBURST 2-3  
    encoding 4-5  
AWCACHE 2-3  
    encoding 5-3  
    in exclusive accesses 6-5  
AWID 2-3  
    in exclusive accesses 6-5  
    out-of-order transactions 8-2  
    uniqueness 8-9  
AWLEN 2-3  
    encoding 4-3  
AWLOCK 2-3  
    encoding 6-2  
AWPROT 2-3  
    encoding 5-6  
AWREADY 2-3  
    timing example 1-9  
AWSIZE 2-3  
    encoding 4-4  
AWVALID 2-3  
    reset 11-2  
    timing example 1-9

AXI protocol  
  features 1-2

**B**

BID 2-5  
  out-of-order transactions 8-2  
Big-endian data structures 9-5  
BREADY 2-5  
  default value 3-4  
  timing example 1-9  
BRESP 2-5  
  encoding 7-2  
  in exclusive accesses 6-6  
  timing example 1-9  
Bufferable attribute 1-11  
  selecting 5-2

Burst  
  address 4-7  
  length 4-7  
Burst length 4-3  
  encoding 4-3  
Burst size 4-4  
  encoding 4-4  
Burst type 4-5  
  encoding 4-5  
  fixed 4-5  
  incrementing 4-5  
  wrapping 4-6  
BVALID 2-5  
  reset requirements 11-2  
  timing example 1-9  
Byte lane strobes 1-5  
  *see also* WSTRB  
Byte lanes  
  eight-bit transfer example 9-4  
  32-bit transfer example 9-4  
Byte-invariant endianness 9-5

**C**

Cache  
  support 5-2  
Cache encoding 5-3  
Cacheable attribute 1-11  
  selecting 5-2

CACTIVE 2-8  
  timing example 12-4, 12-5  
Channel register insertion 1-6  
Clock 11-2  
Completion signaling 1-5, 7-2, 7-4, 7-5  
  *see also* BRESP  
  *see also* RRESP  
Conventions  
  signal naming xvii  
  timing diagram xvi  
  typographical xv  
CSYSACK 2-8  
  timing example 12-4, 12-5  
CSYSREQ 2-8  
  timing example 12-4, 12-5

**D**

Data bus  
  narrow transfers 9-4  
  width 1-5  
DECERR response 7-2, 7-5  
Decode error  
  *see* DECERR response  
Direct memory access  
  *see* DMA  
DMA, support 1-2

**E**

Exclusive access  
  selecting 6-2  
  slave support logic 6-3  
Exclusive access response  
  *see* EXOKAY response  
EXOKAY response 6-3, 6-5, 7-2, 7-4

**F**

Fixed bursts 4-5  
  byte lanes 4-4, 4-8  
  start address 4-7

**G**

Global signals 2-2

**H**

Handshake  
  address channel 3-2  
  read address channel 3-4  
  read data channel 3-2, 3-5  
  signal dependencies 3-7  
  timing example 3-3  
  write address channel 3-3  
  write data channel 3-2, 3-4  
  write response channel 3-2, 3-4

**I**

Incrementing bursts  
  byte lanes 4-4, 4-8  
  increment value 4-5  
  narrow 4-4  
  start address 4-7  
Interconnect  
  combining data streams 8-6  
  implementations 1-6  
  locked accesses 6-7  
  out-of-order transactions 1-9, 8-6  
  realigning address and data 3-6  
Interleaved transactions  
  *see* Out-of-order transactions

**L**

Little-endian data structures 9-5  
Locked access  
  interconnect 6-7  
  selecting 6-2  
Low-power interface  
  signals 2-8

**M**

Master slave handshake 1-4, 1-7, 3-2  
  timing example 3-2, 3-3

**N**

Normal access success  
*see* OKAY response

**O**

OKAY response 6-4, 6-5, 7-2, 7-4  
 Out-of-order transactions 1-3, 1-9  
     from one master 8-3  
     interconnect 1-9, 8-2  
     write reorder depth 8-6

**P**

Parallel transaction processing 1-6,  
     1-8, 8-2  
 Peripheral clock control 12-3  
 Protection  
     encoding 5-6  
 Protection level  
     selecting 5-5

**R**

RDATA 2-7  
     timing example 1-7, 1-8  
 Read address channel  
     handshake 3-4  
     signals 2-6  
 Read allocate attribute  
     selecting 5-2  
 Read data channel 1-3  
     definition 1-5  
     handshake 1-4, 3-2, 3-5  
     signals 2-7  
 Read ID tag  
     *see* RID  
 Register insertion 1-6  
 Reset 11-2  
 Response signaling 7-2  
 RID 2-7  
     out-of-order transactions 8-2  
     uniqueness 8-9  
 RLAST 2-7  
     timing example 1-7, 1-8

RREADY 2-7  
     default value 3-5  
     timing example 1-7, 1-8  
 RRESP 2-7  
     encoding 7-2  
     in exclusive accesses 6-6  
 RVALID 2-7  
     reset requirements 11-2  
     timing example 1-7, 1-8

**S**

Signal naming conventions xvii  
 Signals  
     global 2-2  
     low-power interface 2-8  
     read address channel 2-6  
     read data channel 2-7  
     write address channel 2-3  
     write data channel 2-4  
     write response channel 2-5  
 Slave error response  
     *see* SLVERR response  
 SLVERR response 7-2, 7-4

**T**

Timing diagram conventions xvi  
 Transaction attributes 1-11  
 Transaction ID tag 1-9, 1-10  
     interconnect 8-9  
     *see also* ARID and AWID  
     *see also* RID  
     *see also* WID  
 Transaction order  
     read transactions 8-4  
     rules 8-3  
     *see also* Out-of-order transactions  
     write data interleaving 8-6  
     write reorder depth 8-6  
     write transactions 8-5  
 Typographical conventions xv

**U**

Unaligned transfers 10-1  
     examples 10-3, 10-4  
     signaling unaligned start address 10-2

**V**

Virtual masters 1-10

**W**

WDATA 2-4  
     timing example 1-9  
 WID 2-4  
     out-of-order transactions 8-2  
     uniqueness 8-9  
 WLAST 2-4  
     timing example 1-9  
 Wrapping bursts  
     byte lanes 4-4, 4-8  
     length 4-3, 4-6  
     narrow 4-4  
     start address 4-6, 4-7  
     wrap boundary 4-6, 4-7  
 WREADY 2-4  
     timing example 1-9  
 Write address channel  
     handshake 3-3  
     signals 2-3  
 Write allocate attribute  
     selecting 5-2  
 Write data channel 1-3  
     byte lane strobes 1-5  
     definition 1-5  
     handshake 1-4, 3-2, 3-4  
     signals 2-4  
 Write data interleaving depth 8-6  
 Write ID tag  
     *see* WID  
 Write response channel 1-3  
     definition 1-5  
     handshake 1-4, 3-2, 3-4  
     signals 2-5  
 Write strobe signals  
     *see* WSTRB

Write transactions  
completion signaling 1-5  
**WSTRB** 2-4  
byte lane mapping 9-3  
**WVALID** 2-4  
reset requirements 11-2  
timing example 1-9