AMBA® 3 AXI™ Protocol Checker

r0p1

User Guide

ARM®

ARM DUI 0305C (ID070609)

AMBA 3 AXI Protocol Checker
User Guide

Release Information

The *Change History* table shows the amendments that have been made to this guide:

Change History

Date	Issue	Confidentiality	Change
07 June 2005	A	Non-Confidential	First release for v1.0
20 April 2006	B	Non-Confidential	Second release for v1.0
22 June 2009	C	Non-Confidential	First release for r0p1

Proprietary Notice

Confidentiality Status

This document is Non-Confidential. The right to use, copy and disclose this document may be subject to license restrictions in accordance with the terms of the agreement entered into by ARM and the party that ARM delivered this document to.

Unrestricted Access is an ARM internal classification.

Product Status

The information in this document is final, that is for a developed product.

Web Address

`http://www.arm.com`

Contents
AMBA 3 AXI Protocol Checker User Guide

List of Tables
AMBA 3 AXI Protocol Checker User Guide

List of Figures
AMBA 3 AXI Protocol Checker User Guide

Preface

This preface introduces the *AMBA 3 AXI Protocol Checker r0p1 User Guide*. It contains the following sections:

About this book

This is the *User Guide* for the *AMBA 3 AXI Protocol Checker*.

Intended audience

This book is written for system designers, system integrators, and verification engineers who want to confirm that a design complies with the AMBA 3 AXI Protocol.

Using this book

This book is organized into the following chapters:

Chapter 1 ***Introduction***

Read this for a high-level description of the protocol checker.

Chapter 2 ***Implementation and Integration***

Read this for a description of where to locate the protocol checker in your design, the integration flow, information about specific signal connections with an example file listing, and setting up your simulator,.

Chapter 3 ***Parameter Descriptions***

Read this for a description of the protocol checker parameters.

Chapter 4 ***Assertion Descriptions***

Read this for a description of the protocol checker assertions.

Appendix A ***Example Usage***

Read this for an example of a design that does not comply with the protocol.

Appendix B ***Revisions***

Read this for a description of the technical changes between released issues of this book.

Glossary Read this for definitions of terms used in this guide.

Conventions

Conventions that this book can use are described in:

- *Typographical* on page xi
- *Timing diagrams* on page xi
- *Signals* on page xii.

Typographical

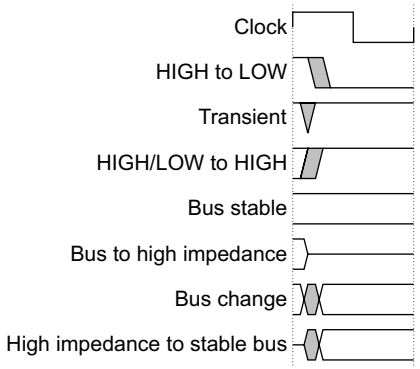
The typographical conventions are:

italic	Highlights important notes, introduces special terminology, denotes internal cross-references, and citations.
bold	Highlights interface elements, such as menu names. Denotes signal names. Also used for terms in descriptive lists, where appropriate.
`monospace`	Denotes text that you can enter at the keyboard, such as commands, file and program names, and source code.
`mono`space	Denotes a permitted abbreviation for a command or option. You can enter the underlined text instead of the full command or option name.
`monospace italic`	Denotes arguments to monospace text where the argument is to be replaced by a specific value.
`monospace bold`	Denotes language keywords when used outside example code.
< and >	Enclose replaceable terms for assembler syntax where they appear in code or code fragments. For example: `MRC p15, 0 <Rd>, <CRn>, <CRm>, <Opcode_2>`

Timing diagrams

The figure named *Key to timing diagram conventions* on page xii 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:

Signal level 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.

Lower-case n Denotes an active-LOW signal.

Prefix A Denotes global *Advanced eXtensible Interface* (AXI) signals:

Prefix AR Denotes AXI read address channel signals.

Prefix AW Denotes AXI write address channel signals.

Prefix B Denotes AXI write response channel signals.

Prefix C Denotes AXI low-power interface signals.

Prefix R Denotes AXI read data channel signals.

Prefix W Denotes AXI write data channel signals.

Additional reading

This section lists publications by ARM and by third parties.

See Infocenter, `http://infocenter.arm.com`, for access to ARM documentation.

ARM publications

This book contains information that is specific to this product. See the following document for other relevant information:

- *AMBA AXI Protocol v1.0 Specification* (ARM IHI 0022).

Other publications

This section lists relevant documents published by third parties:

- Accellera *Open Verification Library* (OVL) Technical Committee documentation and downloads (`http://www.eda.org/ovl`)
- SystemVerilog technical papers, tutorials, and downloads (`http://www.systemverilog.org/`)
- Accellera *SystemVerilog 3.1a Language Reference Manual* (`http://www.eda.org/sl`)
- 1800-2005 *IEEE Standard for SystemVerilog: Unified Hardware Design, Specification and Verification Language* (`http://www.systemverilog.org`).

Feedback

ARM welcomes feedback on this product and its documentation.

Feedback on this product

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

- The product name.
- The product revision or version.
- An explanation with as much information as you can provide. Include symptoms and diagnostic procedures if appropriate.

Feedback on content

If you have comments on content then send an e-mail to `errata@arm.com`. Give:

- the title
- the number, ARM DUI 0305C
- the page numbers to which your comments apply
- a concise explanation of your comments.

ARM also welcomes general suggestions for additions and improvements.

Chapter 1
Introduction

This chapter introduces the protocol checker. It contains the following sections:

1.1 About the protocol checker

You can use the protocol checker with any interface that is designed to implement the AMBA 3 AXI Protocol v1.0. The behavior of the interface you test is checked against the protocol by a series of assertions in the protocol checker.

This guide describes the contents of the Verilog file and how to integrate it into a design. It also describes the correct use of these assertions with simulators to flag errors, warnings, or both during design simulation.

1.2 Tools

The protocol checker is supplied as either:

Verilog OVL A standard for defining, reporting and parameterizing assertions. OVL assertions are predefined assertions standardized by the Accellera committee. They are free to use and download. You can obtain them and more information from `http://www.eda.org/ovl/`.

The protocol checker supports the standard OVL defines from Accellera, v2.3 (09 June 2008).

The protocol checker is backward compatible with older versions of the Accellera OVL Library. To use the April 2003 version of the library, set the following define:

```
`define AXI_USE_OLD_OVL
```

SystemVerilog A *Hardware Description and Verification Language* (HDVL) standard that extends the established Verilog language. It was developed to improve productivity in the design of large gate count, IP-based, bus-intensive chips. SystemVerilog is targeted at the chip implementation and verification flow, with links to the system level design flow.

Chapter 2
Implementation and Integration

This chapter describes the location of the protocol checker and the integration flow. It contains the following sections:

2.1 Implementation and integration flow

Figure 2-1 shows the design flow for implementing and integrating the protocol checker Verilog file with a design.

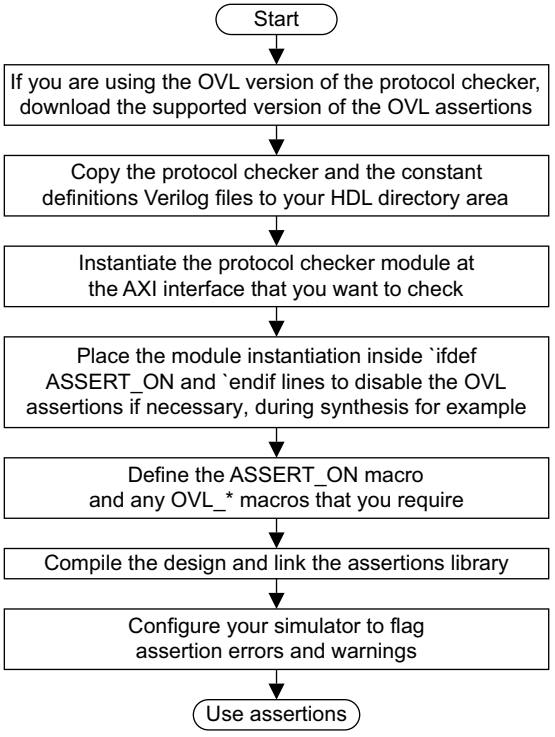


Figure 2-1 Integration flow

Configuring OVL on page 2-6 lists the `OVL_*` macros with example values.

2.2 Implementing the protocol checker in your design directory

Figure 2-2 shows the location of the protocol checker Verilog file in your design RTL.

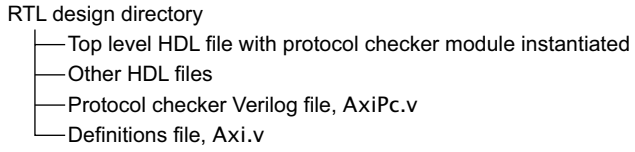


Figure 2-2 Directory structure

2.3 Instantiating the protocol checker module

The protocol checker module contains the Verilog port list. Connect the AXI signals to the corresponding signals in your design.

Place the module instantiation inside `ifdef ASSERT_ON and `endif lines to disable the OVL assertions if necessary, during synthesis for example. *Example Verilog file listing* shows the module instantiated in a top level Verilog file inside these compiler directives.

The AXI signals are signals that the *AMBA AXI Protocol v1.0 Specification* describes.

The low-power interface signals are defined as weak pull-up and you can leave them unconnected if you are not using them. They are named:

- **CSYSREQ** for the low-power request signal
- **CSYSACK** for the low-power request acknowledgement signal
- **CACTIVE** for the clock active signal.

The Verilog file contains checks for user-configurable sideband signals. These signals are defined as weak pull-down and you can leave them unconnected. The *AMBA AXI Protocol v1.0 Specification* does not support these signals.

2.3.1 Example Verilog file listing

Example 2-1 shows part of a design HDL file instantiating the protocol checker module. You can, if necessary, override any of the protocol checker parameters by using `defparam` at this level.

Example 2-1 Example Verilog file listing

```
`ifdef ASSERT_ON

    AxiPC u_axi_ovl
          (
          .ACLK (ACLK),
          .ARESETn (ARESETn),
          .AWID (AWID),
          .AWADDR (AWADDR),
          .AWLEN (AWLEN),
          .AWSIZE (AWSIZE),
          .AWBURST (AWBURST),
          .AWLOCK (AWLOCK),
          .AWCACHE (AWCACHE),
          .AWPROT (AWPROT),
          .AWUSER ({32{1'b0}}),
```

```
            .AWVALID (AWVALID),
            .AWREADY (AWREADY),
            .WID (WID),
            .WLAST (WLAST),
            .WDATA (WDATA),
            .WSTRB (WSTRB),
            .WUSER ({32{1'b0}}),
            .WVALID (WVALID),
            .WREADY (WREADY),
            .BID (BID),
            .BRESP (BRESP),
            .BUSER ({32{1'b0}}),
            .BVALID (BVALID),
            .BREADY (BREADY),
            .ARID (ARID),
            .ARADDR (ARADDR),
            .ARLEN (ARLEN),
            .ARSIZE (ARSIZE),
            .ARBURST (ARBURST),
            .ARLOCK (ARLOCK),
            .ARCACHE (ARCACHE),
            .ARPROT (ARPROT),
            .ARUSER ({32{1'b0}}),
            .ARVALID (ARVALID),
            .ARREADY (ARREADY),
            .RID (RID),
            .RLAST (RLAST),
            .RDATA (RDATA),
            .RRESP (RRESP),
            .RUSER ({32{1'b0}}),
            .RVALID (RVALID),
            .RREADY (RREADY),
            .CACTIVE (CACTIVE),
            .CSYSREQ (CSYSREQ),
            .CSYSACK (CSYSACK)
            );
`endif // `ifdef ASSERT_ON
```

2.4 Configuring your simulator

Most simulators support the use of assertions in RTL and enable you to configure the simulator appropriately using command variables that define the available assertion options. These can include:

- suppress or enable assertion warnings
- select assertion report messages to display
- set minimum severity level for which assertion report messages are output
- set minimum severity level for which an assertion causes the simulator to stop.

The following sections describe how to configure your simulator for different versions of the protocol checker:

- *Configuring OVL*
- *Configuring SVA* on page 2-7.

2.4.1 Configuring OVL

See *Tools* on page 1-3 for the versions of OVL assertions that the OVL version of the protocol checker supports, and how to configure the protocol checker for backward compatibility.

To enable the OVL assertions, you must define `ASSERT_ON`. ARM also recommends that you set additional simulation variables. The following are the OVL macros with example values:

```
+define+OVL_ASSERT_ON                          // Enable OVL assertions
+define+OVL_MAX_REPORT_ERROR=1                 // Message limit per OVL instance
+define+OVL_END_OF_SIMULATION=tb_AxiPC_simulation.simulation_done
+define+OVL_INIT_MSG                          // Report initialization messages
+define+OVL_INIT_COUNT=tb_AxiPC_simulation.ovl_init_count
```

Note

Defining `OVL_MAX_REPORT_ERROR` to 1 or perhaps 2 avoids getting multiple occurrences of the same error because this can hinder you during debugging.

Note

For AxiPC, you MUST define `ASSERT_ON` and `OVL_ASSERT_ON`. The other macros are optional but can affect the results, for example, if you do not define `OVL_END_OF_SIMULATION`, the end-of-simulation checks are not performed.

The OVL version of the protocol checker is tested with a number of simulators. Contact your simulator supplier and see your documentation for more guidance on using Verilog OVL assertions.

Note

There is a known timing problem with some simulators that issue some false fails. To avoid these timing issues, you can check the assertions on the opposite edge of the clock by defining the following:

```
+define+AXI_OVL_CLK=~ACLK
```

See the simulator documentation for more information.

2.4.2 Configuring SVA

The SVA version of the protocol checker is written with SystemVerilog version 3.1a.

The SVA version of the protocol checker is tested with a number of simulators. Contact your simulator supplier and see your documentation for more guidance on using SystemVerilog Assertions.

Chapter 3
Parameter Descriptions

This chapter provides descriptions of the protocol checker parameters. It contains the following sections:

Caution

An additional set of defined parameters are derived from the base set of parameters that this chapter describes. Do not modify them.

3.1 Interface

Table 3-1 lists the user-defined parameters for setting the interface characteristics. Change them to match your design specification.

Table 3-1 Interface parameters

Name	Description	Default
DATA_WIDTH	Width of the system data buses.	64
ID_WIDTH	Number of channel ID bits required, address, write, read, and write response.	4
WDEPTH	Write-interleave depth of monitored slave interface.	1
MAXRBURSTS	Size of FIFOs for storing outstanding read bursts. This must be equal to or greater than the number of outstanding read bursts to the slave interface.	16
MAXWBURSTS	Size of FIFOs for storing outstanding write bursts. This must be equal to or greater than the number of outstanding write bursts to the slave interface.	16
ADDR_WIDTH	Width of the address bus.	32
EXMON_WIDTH	The exclusive access monitor width required.	4
AWUSER_WIDTH	Width of the user AW sideband field.	32
WUSER_WIDTH	Width of the user W sideband field.	32
BUSER_WIDTH	Width of the user B sideband field.	32
ARUSER_WIDTH	Width of the user AR sideband field.	32
RUSER_WIDTH	Width of the user R sideband field.	32

Note

The *AMBA AXI Protocol v1.0 Specification* does not define the user signals shown in Table 3-1.

3.2 Performance checking

Table 3-2 lists the user-defined parameter for performance checking.

Table 3-2 Performance checking parameter

Name	Description	Default
MAXWAITS	Maximum number of cycles between **VALID** to **READY** HIGH before a warning is generated	16

3.3 Property type

Table 3-3 lists the available settings for the PropertyType parameter in each OVL instantiation for specific groups of assertions. You can use them to control how these groups of assertions are used during formal verification.

For example, if you are formally verifying compliance of a master interface then you want to prove all of the master rules, ERRM and RECM, but assume that all slave rules, ERRS and RECS, are true. To do this, you leave the master rules set as 0, the default in Table 3-3, but change the slave rules to 1. This means that you test the master in an environment where the slave is compliant. This is only performed as part of formal verification.

Table 3-3 Property type parameters

Name	Description	Default
AXI_ERRL_PropertyType	*Device Under Test* (DUT) low-power interface compliance with the rules.	0
AXI_ERRM_PropertyType	DUT master interface compliance with the rules.	0
AXI_RECM_PropertyType	DUT master interface compliance with the recommendations.	0
AXI_AUXM_PropertyType	DUT master interface compliance with the internal auxiliary logic.	0
AXI_ERRS_PropertyType	DUT slave interface compliance with the rules.	0
AXI_RECS_PropertyType	DUT slave interface compliance with the recommendations.	0
AXI_AUXS_PropertyType	DUT slave interface compliance with the internal auxiliary logic.	0

You can set the OVL PropertyType parameter to the following values:

0 Assert, this is the default.

1 Assume.

2 Ignore.

3 Assert 2-state.

4 Assume 2-state.

3.4 Disabling recommended rules

Table 3-4 lists the user-defined parameter for configuring recommended rules from the protocol checker.

Table 3-4 Display parameters

Name	Description	Default
RecommendOn	Enable or disable reporting of protocol recommendations	1'b1, enabled
RecMaxWaitOn	Enable or disable the recommended MAX_WAIT rules	1'b1, enabled

Note

RecMaxWaitOn is a subset of RecommendOn, and if RecommendOn is 1'b0, disabled, then the MAX_WAIT rules are disabled regardless of the settings of RecMaxWaitOn.

If RecommendOn is disabled, the following warning is issued:

```
AXI_WARN: All recommended AXI rules have been disabled by the RecommendOn
parameter
```

If RecommendOn is enabled, the default, but RecMaxWaitOn is disabled, the following warning is issued:

```
AXI_WARN: Five recommended MAX_WAIT rules have been disabled by the RecMaxWaitOn
parameter
```

Chapter 4
Assertion Descriptions

This chapter describes the protocol checker assertions and indicates the area of the *AMBA AXI Protocol v1.0 Specification* that they apply to. It contains the following sections:

4.1 Write address channel checks

Table 4-1 lists the write address channel checking rules.

Table 4-1 Write address channel checking rules

Assertion	Description	Specification reference
AXI_ERRM_AWID_STABLE	**AWID** must remain stable when **AWVALID** is asserted and **AWREADY** is LOW	*Handshake process* on Page 3-2
AXI_ERRM_AWID_X	A value of X on **AWID** is not permitted when **AWVALID** is HIGH	*Write address channel* on Page 3-3
AXI_ERRM_AWADDR_BOUNDARY	A write burst cannot cross a 4KB boundary	*About addressing options* on Page 4-2
AXI_ERRM_AWADDR_WRAP_ALIGN	A write transaction with burst type WRAP has an aligned address	*Wrapping burst* on Page 4-6
AXI_ERRM_AWADDR_STABLE	**AWADDR** remains stable when **AWVALID** is asserted and **AWREADY** is LOW	*Handshake process* on Page 3-2
AXI_ERRM_AWADDR_X	A value of X on **AWADDR** is not permitted when **AWVALID** is HIGH	*Write address channel* on Page 3-3
AXI_ERRM_AWLEN_WRAP	A write transaction with burst type WRAP has a length of 2, 4, 8, or 16	*Wrapping burst* on Page 4-6
AXI_ERRM_AWLEN_STABLE	**AWLEN** remains stable when **AWVALID** is asserted and **AWREADY** is LOW	*Handshake process* on Page 3-2
AXI_ERRM_AWLEN_X	A value of X on **AWLEN** is not permitted when **AWVALID** is HIGH	*Write address channel* on Page 3-3
AXI_ERRM_AWSIZE_STABLE	**AWSIZE** remains stable when **AWVALID** is asserted and **AWREADY** is LOW	*Handshake process* on Page 3-2
AXI_ERRM_AWSIZE	The size of a write transfer does not exceed the width of the data interface	*Burst size* on Page 4-4
AXI_ERRM_AWSIZE_X	A value of X on **AWSIZE** is not permitted when **AWVALID** is HIGH	*Write address channel* on Page 3-3
AXI_ERRM_AWBURST	A value of 2'b11 on **AWBURST** is not permitted when **AWVALID** is HIGH	Table 4-3 on Page 4-5
AXI_ERRM_AWBURST_STABLE	**AWBURST** remains stable when **AWVALID** is asserted and **AWREADY** is LOW	*Handshake process* on Page 3-2
AXI_ERRM_AWBURST_X	A value of X on **AWBURST** is not permitted when **AWVALID** is HIGH	*Write address channel* on Page 3-3

Table 4-1 Write address channel checking rules (continued)

Assertion	Description	Specification reference
AXI_ERRM_AWLOCK	A value of 2'b11 on **AWLOCK** is not permitted when **AWVALID** is HIGH	Table 6-1 on Page 6-2
AXI_ERRM_AWLOCK_END	A master must wait for an unlocked transaction at the end of a locked sequence to complete before issuing another write address	*Locked access* on Page 6-7
AXI_ERRM_AWLOCK_ID	A sequence of locked transactions must use a single ID	*Locked access* on Page 6-7
AXI_ERRM_AWLOCK_LAST	A master must wait for all locked transactions to complete before issuing an unlocked write address	*Locked access* on Page 6-7
AXI_ERRM_AWLOCK_STABLE	**AWLOCK** remains stable when **AWVALID** is asserted and **AWREADY** is LOW	*Handshake process* on Page 3-2
AXI_ERRM_AWLOCK_START	A master must wait for all outstanding transactions to complete before issuing a write address that is the first in a locked sequence	*Locked access* on Page 6-7
AXI_ERRM_AWLOCK_X	A value of X on **AWLOCK** is not permitted when **AWVALID** is HIGH	*Write address channel* on Page 3-3
AXI_RECM_AWLOCK_BOUNDARY	Recommended that all locked transaction sequences remain within the same 4KB address region	*Locked access* on Page 6-7
AXI_RECM_AWLOCK_CTRL	Recommended that a master must not change **AWPROT** or **AWCACHE** during a sequence of locked accesses	*Locked access* on Page 6-7
AXI_RECM_AWLOCK_NUM	Recommended that locked transaction sequences are limited to two transactions	*Locked access* on Page 6-7
AXI_ERRM_AWCACHE	When **AWVALID** is HIGH and **AWCACHE[1]** is LOW then **AWCACHE[3:2]** are also LOW	Table 5-1 on Page 5-3
AXI_ERRM_AWCACHE_STABLE	**AWCACHE** remains stable when **AWVALID** is asserted and **AWREADY** is LOW	*Handshake process* on Page 3-2
AXI_ERRM_AWCACHE_X	A value of X on **AWCACHE** is not permitted when **AWVALID** is HIGH	*Write address channel* on Page 3-3
AXI_ERRM_AWPROT_STABLE	**AWPROT** remains stable when **AWVALID** is asserted and **AWREADY** is LOW	*Handshake process* on Page 3-2

Table 4-1 Write address channel checking rules (continued)

Assertion	Description	Specification reference
AXI_ERRM_AWPROT_X	A value of X on **AWPROT** is not permitted when **AWVALID** is HIGH	*Write address channel* on Page 3-3
AXI_ERRM_AWVALID_RESET	**AWVALID** is LOW for the first cycle after **ARESETn** goes HIGH	*Reset* on Page 11-2
AXI_ERRM_AWVALID_STABLE	When **AWVALID** is asserted then it remains asserted until **AWREADY** is HIGH	*Write address channel* on Page 3-3
AXI_ERRM_AWVALID_X	A value of X on **AWVALID** is not permitted when not in reset	-
AXI_ERRS_AWREADY_X	A value of X on **AWREADY** is not permitted when not in reset	-
AXI_RECS_AWREADY_MAX_WAIT	Recommended that **AWREADY** is asserted within MAXWAITS cycles of **AWVALID** being asserted	-

4.2 Write data channel checks

Table 4-2 lists the write data channel checking rules.

Table 4-2 Write data channel checking rules

Assertion	Description	Specification reference
AXI_ERRM_WID_STABLE	**WID** remains stable when **WVALID** is asserted and **WREADY** is LOW.	*Handshake process* on Page 3-2
AXI_ERRM_WID_X	A value of X on **WID** is not permitted when **WVALID** is HIGH.	*Write data channel* on Page 3-4
AXI_ERRM_WDATA_NUM	The number of write data items matches **AWLEN** for the corresponding address. This is triggered when any of the following occurs: • write data arrives and **WLAST** set and the **WDATA** count is not equal to **AWLEN** • write data arrives and **WLAST** not set and the **WDATA** count is equal to **AWLEN** • **ADDR** arrives, **WLAST** already received, and the **WDATA** count is not equal to **AWLEN**.	Table 4-1 on Page 4-3
AXI_ERRM_WDATA_ORDER	The order in which addresses and the first write data item are produced must match.	*Write data interleaving* on Page 8-6
AXI_ERRM_WDATA_STABLE	**WDATA** remains stable when **WVALID** is asserted and **WREADY** is LOW.	*Handshake process* on Page 3-2
AXI_ERRM_WDATA_X	A value of X on **WDATA** is not permitted when **WVALID** is HIGH.	-
AXI_ERRM_WSTRB	Write strobes must only be asserted for the correct byte lanes as determined from the: • start address • transfer size • beat number.	*Write strobes* on Page 9-3
AXI_ERRM_WSTRB_STABLE	**WSTRB** remains stable when **WVALID** is asserted and **WREADY** is LOW.	*Handshake process* on Page 3-2
AXI_ERRM_WSTRB_X	A value of X on **WSTRB** is not permitted when **WVALID** is HIGH.	-
AXI_ERRM_WLAST_STABLE	**WLAST** remains stable when **WVALID** is asserted and **WREADY** is LOW.	*Handshake process* on Page 3-2

Table 4-2 Write data channel checking rules (continued)

Assertion	Description	Specification reference
AXI_ERRM_WLAST_X	A value of X on **WLAST** is not permitted when **WVALID** is HIGH.	-
AXI_ERRM_WVALID_RESET	**WVALID** is LOW for the first cycle after **ARESETn** goes HIGH.	*Reset* on Page 11-2
AXI_ERRM_WVALID_STABLE	When **WVALID** is asserted then it must remain asserted until **WREADY** is HIGH.	*Write data channel* on Page 3-4
AXI_ERRM_WVALID_X	A value of X on **WVALID** is not permitted when not in reset.	-
AXI_RECS_WREADY_MAX_WAIT	Recommended that **WREADY** is asserted within MAXWAITS cycles of **WVALID** being asserted.	-
AXI_ERRS_WREADY_X	A value of X on **WREADY** is not permitted when not in reset.	-
AXI_ERRM_WDEPTH	A master can interleave a maximum of WDEPTH write data bursts.	*Write data interleaving* on Page 8-6

4.3 Write response channel checks

Table 4-3 lists the write response channel checking rules.

Table 4-3 Write response channel checking rules

Assertion	Description	Specification reference
AXI_ERRS_BID_STABLE	**BID** remains stable when **BVALID** is asserted and **BREADY** is LOW	*Handshake process* on Page 3-2
AXI_ERRS_BID_X	A value of X on **BID** is not permitted when **BVALID** is HIGH	-
AXI_ERRS_BRESP	A slave must only give a write response after the last write data item is transferred	*Dependencies between channel handshake signals* on Page 3-7
AXI_ERRS_BRESP_ALL_DONE_EOS	All write transaction addresses are matched with a corresponding buffered response	-
AXI_ERRS_BRESP_EXOKAY	An EXOKAY write response can only be given to an exclusive write access	*Exclusive access from the perspective of the slave* on Page 6-4
AXI_ERRS_BRESP_STABLE	**BRESP** remains stable when **BVALID** is asserted and **BREADY** is LOW	*Handshake process* on Page 3-2
AXI_ERRS_BRESP_X	A value of X on **BRESP** is not permitted when **BVALID** is HIGH	*Write response channel* on Page 3-4
AXI_ERRS_BVALID_RESET	**BVALID** is LOW for the first cycle after **ARESETn** goes HIGH	*Reset* on Page 11-2
AXI_ERRS_BVALID_STABLE	When **BVALID** is asserted then it must remain asserted until **BREADY** is HIGH	*Write response channel* on Page 3-4
AXI_ERRS_BVALID_X	A value of X on **BVALID** is not permitted when not in reset	-
AXI_RECM_BREADY_MAX_WAIT	Recommended that **BREADY** is asserted within MAXWAITS cycles of **BVALID** being asserted	-
AXI_ERRM_BREADY_X	A value of X on **BREADY** is not permitted when not in reset	-
AXI_RECS_BRESP	A slave must not give a write response before the write address	http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.faqs/11424.html

4.4 Read address channel checks

Table 4-4 lists the read address channel checking rules.

Table 4-4 Read address channel checking rules

Assertion	Description	Specification reference
AXI_ERRM_ARID_STABLE	**ARID** remains stable when **ARVALID** is asserted and **ARREADY** is LOW	*Handshake process* on Page 3-2
AXI_ERRM_ARID_X	A value of X on **ARID** is not permitted when **ARVALID** is HIGH	*Read address channel* on Page 3-4
AXI_ERRM_ARADDR_BOUNDARY	A read burst cannot cross a 4KB boundary	*About addressing options* on Page 4-2
AXI_ERRM_ARADDR_STABLE	**ARADDR** remains stable when **ARVALID** is asserted and **ARREADY** is LOW	*Handshake process* on Page 3-2
AXI_ERRM_ARADDR_WRAP_ALIGN	A read transaction with a burst type of WRAP must have an aligned address	*Wrapping burst* on Page 4-6
AXI_ERRM_ARADDR_X	A value of X on **ARADDR** is not permitted when **ARVALID** is HIGH	*Read address channel* on Page 3-4
AXI_ERRM_ARLEN_STABLE	**ARLEN** remains stable when **ARVALID** is asserted and **ARREADY** is LOW	*Handshake process* on Page 3-2
AXI_ERRM_ARLEN_WRAP	A read transaction with burst type of WRAP must have a length of 2, 4, 8, or 16	*Wrapping burst* on Page 4-6
AXI_ERRM_ARLEN_X	A value of X on **ARLEN** is not permitted when **ARVALID** is HIGH	*Read address channel* on Page 3-4
AXI_ERRM_ARSIZE	The size of a read transfer must not exceed the width of the data interface	*Burst size* on Page 4-4
AXI_ERRM_ARSIZE_STABLE	**ARSIZE** remains stable when **ARVALID** is asserted and **ARREADY** is LOW	*Handshake process* on Page 3-2
AXI_ERRM_ARSIZE_X	A value of X on **ARSIZE** is not permitted when **ARVALID** is HIGH	*Read address channel* on Page 3-4
AXI_ERRM_ARBURST	A value of 2'b11 on **ARBURST** is not permitted when **ARVALID** is HIGH	Table 4-3 on Page 4-5
AXI_ERRM_ARBURST_STABLE	**ARBURST** remains stable when **ARVALID** is asserted and **ARREADY** is LOW	*Handshake process* on Page 3-2
AXI_ERRM_ARBURST_X	A value of X on **ARBURST** is not permitted when **ARVALID** is HIGH	*Read address channel* on Page 3-4

Table 4-4 Read address channel checking rules (continued)

Assertion	Description	Specification reference
AXI_ERRM_ARLOCK	A value of 2'b11 on **ARLOCK** is not permitted when **ARVALID** is HIGH	Table 6-1 on Page 6-2
AXI_ERRM_ARLOCK_END	A master must wait for an unlocked transaction at the end of a locked sequence to complete before issuing another read address	*Locked access* on Page 6-7
AXI_ERRM_ARLOCK_ID	A sequence of locked transactions must use a single ID	*Locked access* on Page 6-7
AXI_ERRM_ARLOCK_LAST	A master must wait for all locked transactions to complete before issuing an unlocked read address	*Locked access* on Page 6-7
AXI_ERRM_ARLOCK_STABLE	**ARLOCK** remains stable when **ARVALID** is asserted and **ARREADY** is LOW	*Handshake process* on Page 3-2
AXI_ERRM_ARLOCK_START	A master must wait for all outstanding transactions to complete before issuing a read address that is the first in a locked sequence	*Locked access* on Page 6-7
AXI_ERRM_ARLOCK_X	A value of X on **ARLOCK** is not permitted when **ARVALID** is HIGH	*Read address channel* on Page 3-4
AXI_RECM_ARLOCK_BOUNDARY	Recommended that all locked transaction sequences are kept within the same 4KB address region	*Locked access* on Page 6-7
AXI_RECM_ARLOCK_CTRL	Recommended that a master must not change **ARPROT** or **ARCACHE** during a sequence of locked accesses	*Locked access* on Page 6-7
AXI_RECM_ARLOCK_NUM	Recommended that locked transaction sequences are limited to two transactions	*Locked access* on Page 6-7
AXI_ERRM_ARCACHE	When **ARVALID** is HIGH, if **ARCACHE[1]** is LOW then **ARCACHE[3:2]** must also be LOW	Table 5-1 on Page 5-3
AXI_ERRM_ARCACHE_STABLE	**ARCACHE** remains stable when **ARVALID** is asserted and **ARREADY** is LOW	*Handshake process* on Page 3-2
AXI_ERRM_ARCACHE_X	A value of X on **ARCACHE** is not permitted when **ARVALID** is HIGH	*Read address channel* on Page 3-4
AXI_ERRM_ARPROT_STABLE	**ARPROT** remains stable when **ARVALID** is asserted and **ARREADY** is LOW	*Handshake process* on Page 3-2

Table 4-4 Read address channel checking rules (continued)

Assertion	Description	Specification reference
AXI_ERRM_ARPROT_X	A value of X on **ARPROT** is not permitted when **ARVALID** is HIGH	*Read address channel* on Page 3-4
AXI_ERRM_ARVALID_RESET	**ARVALID** is LOW for the first cycle after **ARESETn** goes HIGH	*Reset* on Page 11-2
AXI_ERRM_ARVALID_STABLE	When **ARVALID** is asserted then it remains asserted until **ARREADY** is HIGH	*Read address channel* on Page 3-4
AXI_ERRM_ARVALID_X	A value of X on **ARVALID** is not permitted when not in reset	-
AXI_ERRS_ARREADY_X	A value of X on **ARREADY** is not permitted when not in reset	-
AXI_RECS_ARREADY_MAX_WAIT	Recommended that **ARREADY** is asserted within MAXWAITS cycles of **ARVALID** being asserted	-

4.5 Read data channel checks

Table 4-5 lists the read data channel checking rules.

Table 4-5 Read data channel checking rules

Assertion	Description	Specification reference
AXI_ERRS_RID	The read data must always follow the address that it relates to. Therefore, a slave can only give read data with an ID to match an outstanding read transaction.	*Read ordering* on Page 8-4
AXI_ERRS_RID_STABLE	**RID** remains stable when **RVALID** is asserted and **RREADY** is LOW.	*Handshake process* on Page 3-2
AXI_ERRS_RID_X.	A value of X on **RID** is not permitted when **RVALID** is HIGH.	-
AXI_ERRS_RDATA_NUM	The number of read data items must match the corresponding **ARLEN**.	Table 4-1 on Page 4-3
AXI_ERRS_RDATA_STABLE	**RDATA** remains stable when **RVALID** is asserted and **RREADY** is LOW.	*Handshake process* on Page 3-2
AXI_ERRS_RDATA_X	A value of X on RDATA valid byte lanes is not permitted when RVALID is HIGH.	-
AXI_ERRS_RRESP_EXOKAY	An EXOKAY read response can only be given to an exclusive read access.	*Exclusive access from the perspective of the slave* on Page 6-4
AXI_ERRS_RRESP_STABLE	**RRESP** remains stable when **RVALID** is asserted and **RREADY** is LOW.	*Handshake process* on Page 3-2
AXI_ERRS_RRESP_X	A value of X on **RRESP** is not permitted when **RVALID** is HIGH.	-
AXI_ERRS_RLAST_ALL_DONE_EOS	All outstanding read bursts must have completed.	-
AXI_ERRS_RLAST_STABLE	**RLAST** remains stable when **RVALID** is asserted and **RREADY** is LOW.	*Handshake process* on Page 3-2
AXI_ERRS_RLAST_X	A value of X on **RLAST** is not permitted when **RVALID** is HIGH.	-
AXI_ERRS_RVALID_RESET	**RVALID** is LOW for the first cycle after **ARESETn** goes HIGH.	*Reset* on Page 11-2

Table 4-5 Read data channel checking rules (continued)

Assertion	Description	Specification reference
AXI_ERRS_RVALID_STABLE	When **RVALID** is asserted then it must remain asserted until **RREADY** is HIGH.	*Read data channel* on Page 3-5
AXI_ERRS_RVALID_X	A value of X on **RVALID** is not permitted when not in reset.	-
AXI_ERRM_RREADY_X	A value of X on **RREADY** is not permitted when not in reset.	-
AXI_RECM_RREADY_MAX_WAIT	Recommended that **RREADY** is asserted within MAXWAITS cycles of **RVALID** being asserted.	-

4.6 Low-power interface rules

Table 4-6 lists the low-power interface checking rules.

Table 4-6 Low-power interface checking rules

Assertion	Description	Specification reference
AXI_ERRL_CSYSREQ_FALL	**CSYSREQ** is only permitted to change from HIGH to LOW when **CSYSACK** is HIGH	*Low-power clock control* on Page 12-3
AXI_ERRL_CSYSREQ_RISE	**CSYSREQ** is only permitted to change from LOW to HIGH when **CSYSACK** is LOW	*Low-power clock control* on Page 12-3
AXI_ERRL_CSYSREQ_X	A value of X on **CSYSREQ** is not permitted when not in reset	-
AXI_ERRL_CSYSACK_FALL	**CSYSACK** is only permitted to change from HIGH to LOW when **CSYSREQ** is LOW	*Low-power clock control* on Page 12-3
AXI_ERRL_CSYSACK_RISE	**CSYSACK** is only permitted to change from LOW to HIGH when **CSYSREQ** is HIGH	*Low-power clock control* on Page 12-3
AXI_ERRL_CSYSACK_X	A value of X on **CSYSACK** is not permitted when not in reset	-
AXI_ERRL_CACTIVE_X	A value of X on **CACTIVE** is not permitted when not in reset	-

4.7 Exclusive access checks

Table 4-7 lists the address channel exclusive access checking rules.

Table 4-7 Address channel exclusive access checking rules

Assertion	Description	Specification reference
AXI_ERRM_EXCL_ALIGN	The address of an exclusive access is aligned to the total number of bytes in the transaction	*Exclusive access restrictions* on Page 6-5
AXI_ERRM_EXCL_LEN	The number of bytes to be transferred in an exclusive access burst is a power of 2, that is, 1, 2, 4, 8, 16, 32, 64, or 128 bytes	*Exclusive access restrictions* on Page 6-5
AXI_RECM_EXCL_MATCH	Recommended that the address, size, and length of an exclusive write with a given ID is the same as the address, size, and length of the preceding exclusive read with the same ID	*Exclusive access restrictions* on Page 6-5
AXI_ERRM_EXCL_MAX	128 is the maximum number of bytes that can be transferred in an exclusive burst	*Exclusive access restrictions* on Page 6-5
AXI_RECM_EXCL_PAIR	Recommended that every exclusive write has an earlier outstanding exclusive read with the same ID	*Exclusive access from the perspective of the master* on Page 6-3

4.8 Internal logic checks

Table 4-8 lists the internal logic checks.

Table 4-8 Internal logic checks

Assertion	Description	Specification Reference
AXI_AUXM_DATA_WIDTH	DATA_WIDTH parameter is 32, 64, 128, 256, 512, or 1024.	-
AXI_AUXM_RCAM_OVERFLOW	Read CAM overflow, increase MAXRBURSTS parameter.	-
AXI_AUXM_RCAM_UNDERFLOW	Read CAM underflow.	-
AXI_AUXM_WCAM_OVERFLOW	Write CAM overflow, increase MAXWBURSTS parameter.	-
AXI_AUXM_WCAM_UNDERFLOW	Write CAM underflow.	-
AXI_AUXM_ADDR_WIDTH	Parameter ADDR_WIDTH must be between 32 bits and 64 bits inclusive.	-
AXI_AUXM_AWUSER_WIDTH	Parameter AWUSER_WIDTH must be greater than or equal to 1.	-
AXI_AUXM_WUSER_WIDTH	Parameter WUSER_WIDTH must be greater than or equal to 1.	-
AXI_AUXM_BUSER_WIDTH	Parameter BUSER_WIDTH must be greater than or equal to 1.	-
AXI_AUXM_ARUSER_WIDTH	Parameter ARUSER_WIDTH must be greater than or equal to 1.	-
AXI_AUXM_RUSER_WIDTH	Parameter RUSER_WIDTH must be greater than or equal to 1.	-
AXI_AUXM_EXMON_WIDTH	Parameter EXMON_WIDTH must be greater than or equal to 1.	-
AXI_AUXM_ID_WIDTH	Parameter ID_WIDTH must be greater than or equal to 1.	-
AXI_AUXM_WDEPTH	Parameter WDEPTH must be greater than or equal to 1.	-
AXI_AUXM_MAXRBURSTS	Parameter MAXRBURSTS must be greater than or equal to 1.	-
AXI_AUXM_MAXWBURSTS	Parameter MAXWBURSTS must be greater than or equal to 1.	-
AXI_AUXM_EXCL_OVERFLOW	Exclusive access monitor overflow, increase EXMON_WIDTH parameter.	-

Appendix A
Example Usage

This appendix provides an example transcript from the protocol checker. It contains the following section:

- *RDATA stable failure* on page A-2.

A.1 RDATA stable failure

Figure A-1 shows the timing diagram for a failure of the AXI_ERRS_RDATA_STABLE check.

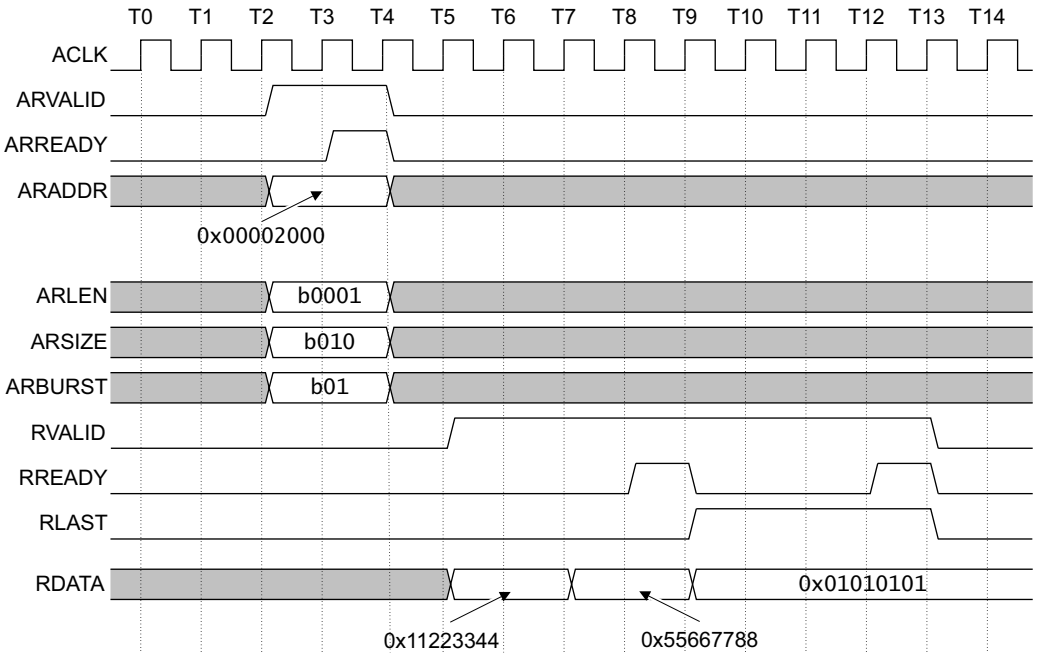


Figure A-1 RDATA stable failure

RDATA changes at T7 when **RVALID** is HIGH and **RREADY** is LOW. The protocol checker samples the change at T8.

Example A-1 shows the protocol checker transcript for this failure.

Example A-1 RDATA stable failure

```
# Loading work.TB
# Loading work.AxiPC
# Loading ../ovl/work.assert_implication
# do startup.do
# OVL_ERROR : ASSERT_WIN_UNCHANGE : AXI_ERRS_RDATA_STABLE. RDATA must remain stable when RVALID is
asserted and RREADY low. Spec: section 3.1, and figure 3-1, on page 3-2. : : severity 1 : time 250
ns : TB.uAxiPC.errs_rdata_stable.ovl_error
```

```
# OVL_ERROR  : AXI_ERRS_RDATA_STABLE. RDATA must remain stable when RVALID is asserted and RREADY low.
Spec: section 3.1, and figure 3-1, on page 3-2. :  : severity 1 : time 270 ns :
TB.uAxiPC.errs_rdata_stable.ovl_error
# Break at rdata_stable.v line 69
# Simulation Breakpoint: Break at rdata_stable.v line 69
```

Appendix B
Revisions

This appendix describes the technical changes between released issues of this book.

Table B-1 Differences between issue B and issue C

Change	Location	Affects
Version of OVL defines from Accellera updated	*Tools* on page 1-3.	r0p1
User-defined parameters for setting the interface characteristics added	Table 3-1 on page 3-2.	r0p1
Settings for the `PropertyType` parameter in each OVL instantiation updated	Table 3-3 on page 3-4.	r0p1
OVL `PropertyType` parameters added	*Property type* on page 3-4.	r0p1
Read data channel checking rule added	Table 4-5 on page 4-11.	r0p1
Internal logic checks added	Table 4-8 on page 4-15.	r0p1

Glossary

This glossary describes some of the terms used in technical documents from ARM.

Advanced eXtensible Interface (AXI)
A bus protocol that supports separate address/control and data phases, unaligned data transfers using byte strobes, burst-based transactions with only start address issued, separate read and write data channels to enable low-cost DMA, ability to issue multiple outstanding addresses, out-of-order transaction completion, and easy addition of register stages to provide timing closure.

The AXI protocol also includes optional extensions to cover signaling for low-power operation.

AXI is targeted at high performance, high clock frequency system designs and includes a number of features that make it very suitable for high speed sub-micron interconnect.

Advanced Microcontroller Bus Architecture (AMBA)
A family of protocol specifications that describe a strategy for the interconnect. AMBA is the ARM open standard for on-chip buses. It is an on-chip bus specification that describes a strategy for the interconnection and management of functional blocks that make up a *System-on-Chip* (SoC). It aids in the development of embedded processors with one or more CPUs or signal processors and multiple peripherals. AMBA complements a reusable design methodology by defining a common backbone for SoC modules.

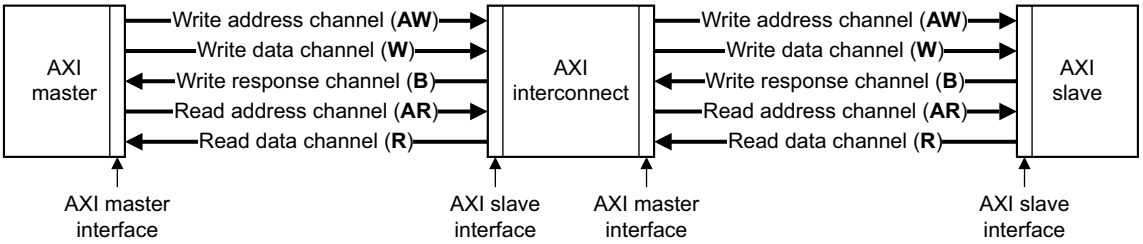
AMBA *See* Advanced Microcontroller Bus Architecture.

AXI *See* Advanced eXtensible Interface.

AXI channel order and interfaces

The block diagram shows:

- the order in which AXI channel signals are described
- the master and slave interface conventions for AXI components.



AXI terminology The following AXI terms are general. They apply to both masters and slaves:

Active read transaction

A transaction for which the read address has transferred, but the last read data has not yet transferred.

Active transfer

A transfer for which the **xVALID**[1] handshake has asserted, but for which **xREADY** has not yet asserted.

Active write transaction

A transaction for which the write address or leading write data has transferred, but the write response has not yet transferred.

Completed transfer

A transfer for which the **xVALID**/**xREADY** handshake is complete.

Payload The non-handshake signals in a transfer.

---

1. The letter **x** in the signal name denotes an AXI channel as follows:

AW	Write address channel.
W	Write data channel.
B	Write response channel.
AR	Read address channel.
R	Read data channel.

Transaction An entire burst of transfers, comprising an address, one or more data transfers and a response transfer (writes only).

Transmit An initiator driving the payload and asserting the relevant **xVALID** signal.

Transfer A single exchange of information. That is, with one **xVALID**/**xREADY** handshake.

The following AXI terms are master interface attributes. To obtain optimum performance, they must be specified for all components with an AXI master interface:

Combined issuing capability

The maximum number of active transactions that a master interface can generate. It is specified for master interfaces that use combined storage for active write and read transactions. If not specified then it is assumed to be equal to the sum of the write and read issuing capabilities.

Read ID capability

The maximum number of different **ARID** values that a master interface can generate for all active read transactions at any one time.

Read ID width

The number of bits in the **ARID** bus.

Read issuing capability

The maximum number of active read transactions that a master interface can generate.

Write ID capability

The maximum number of different **AWID** values that a master interface can generate for all active write transactions at any one time.

Write ID width

The number of bits in the **AWID** and **WID** buses.

Write interleave capability

The number of active write transactions for which the master interface is capable of transmitting data. This is counted from the earliest transaction.

Write issuing capability

The maximum number of active write transactions that a master interface can generate.

The following AXI terms are slave interface attributes. To obtain optimum performance, they must be specified for all components with an AXI slave interface:

Combined acceptance capability

The maximum number of active transactions that a slave interface can accept. It is specified for slave interfaces that use combined storage for active write and read transactions. If not specified then it is assumed to be equal to the sum of the write and read acceptance capabilities.

Read acceptance capability

The maximum number of active read transactions that a slave interface can accept.

Read data reordering depth

The number of active read transactions for which a slave interface can transmit data. This is counted from the earliest transaction.

Write acceptance capability

The maximum number of active write transactions that a slave interface can accept.

Write interleave depth

The number of active write transactions for which the slave interface can receive data. This is counted from the earliest transaction.