AMBA® 4 AXI4™, AXI4-Lite™, and AXI4-Stream™ Protocol Assertions

Revision: r0p0

User Guide

ARM®

ARM DUI 0534A (ID063010)

AMBA 4 AXI4, AXI4-Lite, and AXI4-Stream Protocol Assertions
User Guide

Release Information

The following changes have been made to this book.

Change history

Date	Issue	Confidentiality	Change
30 June 2010	A	Non-Confidential	First issue for r0p0

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.

Product Status

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

Web Address

`http://www.arm.com`

Contents
AMBA 4 AXI4, AXI4-Lite, and AXI4-Stream Protocol Assertions User Guide

List of Tables
AMBA 4 AXI4, AXI4-Lite, and AXI4-Stream Protocol Assertions User Guide

List of Figures
AMBA 4 AXI4, AXI4-Lite, and AXI4-Stream Protocol Assertions User Guide

Preface

This preface introduces the *AMBA® 4 AXI4™, AXI4-Lite™, and AXI4-Stream™ Protocol Assertions User Guide*. It contains the following sections:

About this book

This is the *User Guide* for the *AMBA 4 AXI4, AXI4-Lite, and AXI4-Stream Protocol Assertions*.

Intended audience

This book is written for system designers, system integrators, and verification engineers who want to confirm that a design complies with the relevant AMBA 4 protocol. This can be AXI4, AXI4-Lite, or AXI4-Stream.

Using this book

This book is organized into the following chapters:

Chapter 1 ***Introduction***

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

Chapter 2 ***Implementation and Integration***

Read this for a description of where to locate the protocol assertions 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 assertions parameters.

Chapter 4 ***Protocol Assertions Descriptions***

Read this for a description of the protocol assertions module.

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 ix
- *Timing diagrams* on page ix
- *Signals* on page ix.

Typographical

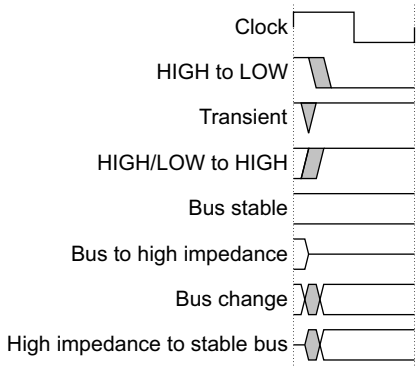
The typographical conventions are:

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* 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 v2.0 Specification* (ARM IHI 0022)
- *AMBA 4 AXI4-Stream™ Protocol v1.0 Specification* (ARM IHI 0051).

Other publications

This section lists relevant documents published by third parties:

- 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 0534A
- 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 assertions. It contains the following sections:

1.1 About the protocol assertions

You can use the protocol assertions with any interface that is designed to implement the AMBA® 4 AXI4™, AXI4-Lite™, or AXI4-Stream™ Protocol v1.0. The behavior of the interface you test is checked against the protocol by a series of assertions.

This guide describes the contents of the SystemVerilog files, and how to integrate them 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 assertions are written in SystemVerilog. SystemVerilog is 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.

Note

The version of System Verilog supported is IEEE 1800-2005.

Chapter 2
Implementation and Integration

This chapter describes the location of the protocol assertions 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 assertions SystemVerilog file with a design.

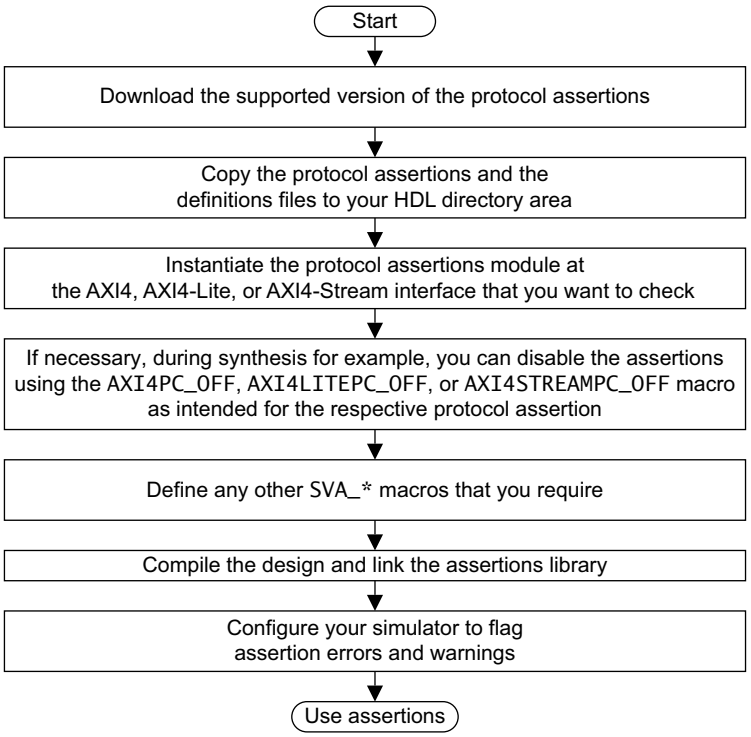


Figure 2-1 Integration flow

2.2 Implementing the protocol assertions in your design directory

You can implement the protocol assertions for:

- AXI4™
- AXI4-Lite™
- AXI4-Stream™.

This section describes:

- *AXI4 protocol assertions files*
- *Location of AXI4 protocol assertions files*.

2.2.1 AXI4 protocol assertions files

Figure 2-2 shows the contents of the directory that contains the protocol assertions. It shows the files that are required for each of the different protocols, AXI4, AXI4-Lite, and AXI4-Stream.

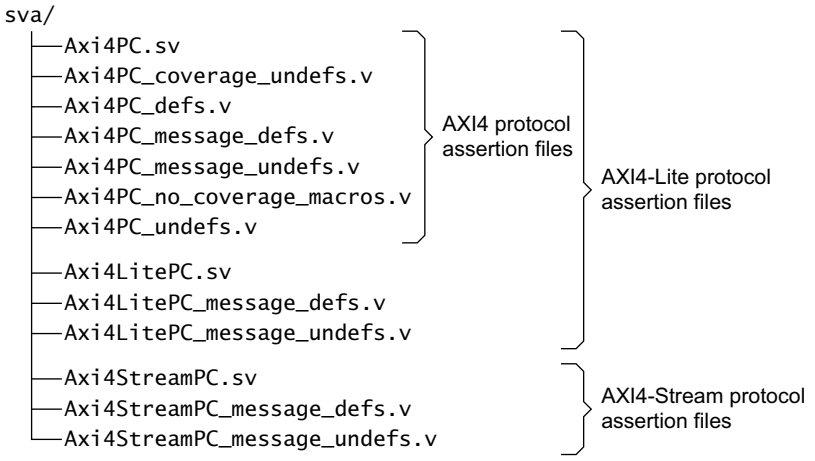


Figure 2-2 Protocol assertions directory structure for AXI4, AXI4-Lite, and AXI4-Stream

2.2.2 Location of AXI4 protocol assertions files

Figure 2-3 shows the location of the protocol assertions SystemVerilog files in your design RTL.

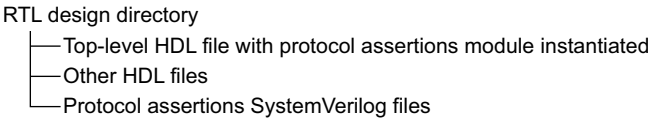


Figure 2-3 Location of the AMBA 4 AXI4 protocol assertions SystemVerilog files

2.3 Instantiating the protocol assertions module

The protocol assertions module contains a port list. Connect the AXI4, AXI4-Lite, or AXI4-Stream module ports to the corresponding signals in your design.

Example Verilog file listing for AXI4 protocol assertions instantiation shows the module instantiated in a top-level Verilog file.

See *ARM publications* on page x for the specifications that describe the AXI4, AXI4-Lite, and AXI4-Stream signals.

The low-power interface signals of the AXI interface 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 AXI4 SystemVerilog files contain checks for user-configurable sideband signals. These signals are defined as weak pull-down and you can leave them unconnected.

2.3.1 Example Verilog file listing for AXI4 protocol assertions instantiation

Example 2-1 shows part of a design HDL file instantiating the protocol assertions module for AXI4. You can, if necessary, override any of the protocol assertions parameters by using `defparam` at this level.

Example 2-1 Example Verilog file listing for AXI

```
Axi4PC u_axi4_sva
      (
      .ACLK (ACLK),
      .ARESETn (ARESETn),
      .AWID (AWID),
      .AWADDR (AWADDR),
      .AWLEN (AWLEN),
      .AWSIZE (AWSIZE),
      .AWBURST (AWBURST),
      .AWLOCK (AWLOCK),
      .AWCACHE (AWCACHE),
      .AWPROT (AWPROT),
      .AWQOS (AWQOS),
      .AWREGION (AWREGION),
      .AWUSER ({32{1'b0}}),
      .AWVALID (AWVALID),
      .AWREADY (AWREADY),
      .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),
          .ARQOS (ARQOS),
          .ARREGION (ARREGION),
          .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)
          );
```

2.3.2 Example Verilog file listing for AXI4-Lite protocol assertions instantiation

Example 2-2 shows part of a design HDL file instantiating the protocol assertions module for AXI4-Lite. You can, if necessary, override any of the protocol assertions parameters by using `defparam` at this level.

Example 2-2 Example Verilog file listing for AXI4-Lite

```
Axi4LitePC u_axi4lite_sva
          (
          .ACLK (ACLK),
          .ARESETn (ARESETn),
          .AWADDR (AWADDR),
          .AWPROT (AWPROT),
          .AWVALID (AWVALID),
          .AWREADY (AWREADY),
          .WDATA (WDATA),
          .WSTRB (WSTRB),
          .WVALID (WVALID),
          .WREADY (WREADY),
          .BRESP (BRESP),
          .BVALID (BVALID),
          .BREADY (BREADY),
          .ARADDR (ARADDR),
          .ARPROT (ARPROT),
          .ARVALID (ARVALID),
          .ARREADY (ARREADY),
          .RDATA (RDATA),
          .RRESP (RRESP),
          .RVALID (RVALID),
          .RREADY (RREADY)
          );
```

2.3.3 Example Verilog file listing for AXI4-Stream protocol assertions instantiation

Example 2-3 shows part of a design HDL file instantiating the protocol assertions module for AXI4-Stream. You can, if necessary, override any of the protocol assertions parameters by using `defparam` at this level.

Example 2-3 Example Verilog file listing for AXI4-Stream

```
Axi4StreamPC u_axi4stream_sva
    (
    .ACLK (ACLK),
    .ARESETn (ARESETn),
    .TDATA (TDATA),
    .TSTRB (TSTRB),
    .TKEEP (TKEEP),
    .TLAST (TLAST),
    .TID (TID),
    .TDEST (TDEST),
    .TUSER ({32{1'b0}}),
    .TVALID (TVALID),
    .TREADY (TREADY)
    );
```

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 a minimum severity level for which assertion report messages are output
- set a minimum severity level for which an assertion causes the simulator to stop.

The protocol assertions are written using SystemVerilog version 3.1a, and are tested with a number of simulators. Contact your simulator supplier and see your documentation for more information on using SystemVerilog Assertions.

Chapter 3
Parameter Descriptions

This chapter provides descriptions of the protocol assertions 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

This section describes:

- *AXI4 and AXI4-Lite interfaces*
- *AXI4-Stream interface*.

3.1.1 AXI4 and AXI4-Lite interfaces

Table 3-1 shows the user-defined parameters for setting the interface characteristics for AXI4™ and AXI4-Lite™. Change them to match your design specification.

Table 3-1 Interface parameters for AXI4 and AXI4-Lite

Name	Description	AXI4 default	AXI4-Lite default
DATA_WIDTH	Width of the system data buses.	64	64
ID_WIDTH	Number of channel ID bits required, address, write, read, and write response.	4	-
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	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	16
ADDR_WIDTH	Width of the address bus.	32	32
EXMON_WIDTH	Width of the exclusive access monitor 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	-

3.1.2 AXI4-Stream interface

Table 3-2 shows the user-defined parameters for setting the interface characteristics for AXI4-Stream™. Change them to match your design specification.

Table 3-2 Interface parameters for AXI4-Stream

Name	Description	Default
DATA_WIDTH_BYTES	Width of the data bus in bytes	4
DEST_WIDTH	Width of **TDEST** in bits	4
ID_WIDTH	Number of channel ID bits required for **TID**	4
USER_WIDTH	Width of the **TUSER** bus in bits	32

3.2 Performance checking

Table 3-3 shows the user-defined parameter for performance checking.

Table 3-3 Performance checking parameter

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

3.3 Disabling recommended rules

This section describes:

- *Disabling recommended rules for AXI4 and AXI4-Lite*
- *Disabling recommended rules for AXI4-Stream*.

3.3.1 Disabling recommended rules for AXI4 and AXI4-Lite

Table 3-4 shows the user-defined parameters for configuring recommended rules from the protocol assertions.

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, that is, disabled, then the `MAX_WAIT` rules are disabled regardless of the settings of `RecMaxWaitOn`.

If `RecommendOn` is disabled, the following warning is issued:

```
AXI4_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:

```
AXI4_WARN: All recommended MAX_WAIT rules have been disabled by the RecMaxWaitOn
parameter
```

3.3.2 Disabling recommended rules for AXI4-Stream

Table 3-5 shows the user-defined parameter for configuring recommended rules from the protocol assertions.

Table 3-5 Display parameters

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

If `RecommendOn` is disabled, the following AXI4-Stream warning is issued:

```
AXI4STREAM_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:

```
AXI4STREAM_WARN: All recommended MAX_WAIT rules have been disabled by the RecMaxWaitOn
parameter
```

3.4 End of simulation rules

This section describes the end of simulation rules for:

- *AXI4 and AXI4-Lite*
- *AXI4-Stream*.

3.4.1 AXI4 and AXI4-Lite

Some of the rules in the assertions report whether there are outstanding transactions at the end of the simulation. To use these assertions, you must ensure that:

- The testbench that you are using has a signal called **EOS_signal**. You must drive **EOS_signal** HIGH at the end of the simulation for at least one clock cycle.
- You use `+define+AXI4_END_OF_SIMULATION=tb.EOS_signal` when compiling.

3.4.2 AXI4-Stream

Some of the rules in the assertions report whether there are active streams at the end of the simulation. To use these assertions you must ensure that:

- The testbench that you are using has a signal called **EOS_signal**. You must drive **EOS_signal** HIGH at the end of the simulation for at least one clock cycle.
- You use `+define+AXI4STREAM_END_OF_SIMULATION=tb.EOS_signal` when compiling.

3.5 X-check rules

If you want to disable the X-propagation assertions on AXI4 or AXI4-Lite interfaces, you must use the following rule when compiling:

```
+define+AXI4_XCHECK_OFF
```

If you want to disable the X-propagation assertions on AXI4-Stream interfaces, you must use the following rule when compiling:

```
+define+AXI4STREAM_XCHECK_OFF
```

3.6 Disabling protocol assertions

In circumstances where the protocol assertion module has been automatically inserted in a testbench, and you want to disable it without editing the testbench, you can compile with the following options:

`+define+AXI4PC_OFF` To disable the `AXI4PC` protocol assertions module.

`+define+AXI4LITEPC_OFF`

To disable the `AXI4LITEPC` protocol assertions module.

`+define+AXI4STREAMPC_OFF`

To disable the `AXI4STREAMPC` protocol assertions module.

Chapter 4
Protocol Assertions Descriptions

This chapter describes the protocol assertions and indicates the area of the *AMBA® AXI Protocol v2.0 Specification* to which they apply. It contains the following sections:

4.1 AXI4™ and AXI4-Lite™ protocol assertion descriptions

This section contains the following subsections:

- *Write address channel checks*
- *Write data channel checks* on page 4-4
- *Write response channel checks* on page 4-5
- *Read address channel checks* on page 4-6
- *Read data channel checks* on page 4-8
- *Low-power interface rules* on page 4-9
- *Exclusive access checks* on page 4-10
- *Internal logic checks* on page 4-10.

4.1.1 Write address channel checks

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

Table 4-1 Write address channel checking rules

Assertion	Description	Specification reference	AXI4-Lite
AXI4_ERRM_AWID_STABLE	**AWID** must remain stable when **AWVALID** is asserted and **AWREADY** is LOW	*Handshake process* on Page 3-2	-
AXI4_ERRM_AWID_X	A value of X on **AWID** is not permitted when **AWVALID** is HIGH	-	-
AXI4_ERRM_AWADDR_BOUNDARY	A write burst cannot cross a 4KB boundary	*About addressing options* on Page 4-2	-
AXI4_ERRM_AWADDR_WRAP_ALIGN	A write transaction with burst type WRAP has an aligned address	*Wrapping burst* on Page 4-6	-
AXI4_ERRM_AWADDR_STABLE	**AWADDR** remains stable when **AWVALID** is asserted and **AWREADY** is LOW	*Handshake process* on Page 3-2	Valid
AXI4_ERRM_AWADDR_X	A value of X on **AWADDR** is not permitted when **AWVALID** is HIGH	-	Valid
AXI4_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	-
AXI4_ERRM_AWLEN_STABLE	**AWLEN** remains stable when **AWVALID** is asserted and **AWREADY** is LOW	*Handshake process* on Page 3-2	-
AXI4_ERRM_AWLEN_X	A value of X on **AWLEN** is not permitted when **AWVALID** is HIGH	-	-
AXI4_ERRM_AWSIZE_STABLE	**AWSIZE** remains stable when **AWVALID** is asserted and **AWREADY** is LOW	*Handshake process* on Page 3-2	-
AXI4_ERRM_AWSIZE	The size of a write transfer does not exceed the width of the data interface	*Burst size* on Page 4-4	-
AXI4_ERRM_AWSIZE_X	A value of X on **AWSIZE** is not permitted when **AWVALID** is HIGH	-	-
AXI4_ERRM_AWBURST	A value of 2'b11 on **AWBURST** is not permitted when **AWVALID** is HIGH	Table 4-3 on Page 4-5	-

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

Assertion	Description	Specification reference	AXI4-Lite
AXI4_ERRM_AWBURST_STABLE	**AWBURST** remains stable when **AWVALID** is asserted and **AWREADY** is LOW	*Handshake process* on Page 3-2	-
AXI4_ERRM_AWBURST_X	A value of X on **AWBURST** is not permitted when **AWVALID** is HIGH	-	-
AXI4_ERRM_AWLOCK_STABLE	**AWLOCK** remains stable when **AWVALID** is asserted and **AWREADY** is LOW	*Handshake process* on Page 3-2	-
AXI4_ERRM_AWLOCK_X	A value of X on **AWLOCK** is not permitted when **AWVALID** is HIGH	-	-
AXI4_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	-
AXI4_ERRM_AWCACHE_STABLE	**AWCACHE** remains stable when **AWVALID** is asserted and **AWREADY** is LOW	*Handshake process* on Page 3-2	-
AXI4_ERRM_AWCACHE_X	A value of X on **AWCACHE** is not permitted when **AWVALID** is HIGH	*Write address channel* on Page 3-3	-
AXI4_ERRM_AWPROT_STABLE	**AWPROT** remains stable when **AWVALID** is asserted and **AWREADY** is LOW	*Handshake process* on Page 3-2	Valid
AXI4_ERRM_AWPROT_X	A value of X on **AWPROT** is not permitted when **AWVALID** is HIGH	-	Valid
AXI4_ERRM_AWVALID_RESET	**AWVALID** is LOW for the first cycle after **ARESETn** goes HIGH	*Reset* on Page 11-2	Valid
AXI4_ERRM_AWVALID_STABLE	When **AWVALID** is asserted, then it remains asserted until **AWREADY** is HIGH	*Write address channel* on Page 3-3	Valid
AXI4_ERRM_AWVALID_X	A value of X on **AWVALID** is not permitted when not in reset	-	Valid
AXI4_ERRS_AWREADY_X	A value of X on **AWREADY** is not permitted when not in reset	-	Valid
AXI4_RECS_AWREADY_MAX_WAIT	Recommended that **AWREADY** is asserted within MAXWAITS cycles of **AWVALID** being asserted	-	Valid
AXI4_ERRM_AWUSER_STABLE	**AWUSER** remains stable when **AWVALID** is asserted and **AWREADY** is LOW	*Handshake process* on Page 3-2	-
AXI4_ERRM_AWUSER_X	A value of X on **AWUSER** is not permitted when **AWVALID** is HIGH	-	-
AXI4_ERRM_AWQOS_STABLE	**AWQOS** remains stable when **AWVALID** is asserted and **AWREADY** is LOW	*Handshake process* on Page 3-2	-
AXI4_ERRM_AWQOS_X	A value of X on **AWQOS** is not permitted when **AWVALID** is HIGH	-	-
AXI4_ERRM_AWREGION_STABLE	**AWREGION** remains stable when **AWVALID** is asserted and **AWREADY** is LOW	*Handshake process* on Page 3-2	-
AXI4_ERRM_AWREGION_X	A value of X on **AWREGION** is not permitted when **AWVALID** is HIGH	-	-

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

Assertion	Description	Specification reference	AXI4-Lite
AXI4_ERRM_AWLEN_FIXED	Transactions of burst type FIXED cannot have a length greater than 16 beats	*Limitations of use* on Page 13-2	-
AXI4_ERRM_AWLEN_LOCK	Exclusive access transactions cannot have a length greater than 16 beats	*Limitations of use* on Page 13-2	-
AXI4_ERRM_AWUSER_TIEOFF	**AWUSER** must be stable when **AWUSER_WIDTH** has been set to zero	-	-
AXI4_ERRM_AWID_TIEOFF	**AWID** must be stable when **ID_WIDTH** has been set to zero	-	-

4.1.2 Write data channel checks

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

Table 4-2 Write data channel checking rules

Assertion	Description	Specification reference	AXI4-Lite
AXI4_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** is set, and the **WDATA** count is not equal to **AWLEN** • write data arrives and **WLAST** is not set, and the **WDATA** count is equal to **AWLEN** • **ADDR** arrives, **WLAST** is already received, and the **WDATA** count is not equal to **AWLEN**.	Table 4-1 on Page 4-3	-
AXI4_ERRM_WDATA_STABLE	**WDATA** remains stable when **WVALID** is asserted and **WREADY** is LOW.	*Handshake process* on Page 3-2	Valid
AXI4_ERRM_WDATA_X	A value of X on **WDATA** valid byte lanes is not permitted when **WVALID** is HIGH.	-	Valid
AXI4_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	Valid
AXI4_ERRM_WSTRB_STABLE	**WSTRB** remains stable when **WVALID** is asserted and **WREADY** is LOW.	*Handshake process* on Page 3-2	Valid
AXI4_ERRM_WSTRB_X	A value of X on **WSTRB** is not permitted when **WVALID** is HIGH.	-	Valid
AXI4_ERRM_WLAST_STABLE	**WLAST** remains stable when **WVALID** is asserted and **WREADY** is LOW.	*Handshake process* on Page 3-2	-
AXI4_ERRM_WLAST_X	A value of X on **WLAST** is not permitted when **WVALID** is HIGH.	-	-

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

Assertion	Description	Specification reference	AXI4-Lite
AXI4_ERRM_WVALID_RESET	**WVALID** is LOW for the first cycle after **ARESETn** goes HIGH.	*Reset* on Page 11-2	Valid
AXI4_ERRM_WVALID_STABLE	When **WVALID** is asserted, then it must remain asserted until **WREADY** is HIGH.	*Write data channel* on Page 3-3	Valid
AXI4_ERRM_WVALID_X	A value of X on **WVALID** is not permitted when not in reset.	-	Valid
AXI4_RECS_WREADY_MAX_WAIT	Recommended that **WREADY** is asserted within MAXWAITS cycles of **WVALID** being asserted.	-	Valid
AXI4_ERRS_WREADY_X	A value of X on **WREADY** is not permitted when not in reset.	-	Valid
AXI4_ERRM_WUSER_STABLE	**WUSER** must remain constant whilst **WVALID** is asserted and **WREADY** is de-asserted.	*Handshake process* on Page 3-2	-
AXI4_ERRM_WUSER_X	A value of X on **WUSER** is not permitted when **WVALID** is HIGH.	-	-
AXI4_ERRM_WUSER_TIEOFF	**WUSER** must be stable when **WUSER_WIDTH** has been set to zero.	-	-

4.1.3 Write response channel checks

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

Table 4-3 Write response channel checking rules

Assertion	Description	Specification reference	AXI4-Lite
AXI4_ERRS_BID_STABLE	**BID** remains stable when **BVALID** is asserted and **BREADY** is LOW	*Handshake process* on Page 3-2	-
AXI4_ERRS_BID_X	A value of X on **BID** is not permitted when **BVALID** is HIGH	-	-
AXI4_ERRS_BRESP_ALL_DONE_EOS	All write transaction addresses are matched with a corresponding buffered response	-	Valid
AXI4_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	Valid
AXI4_ERRS_BRESP_STABLE	**BRESP** remains stable when **BVALID** is asserted and **BREADY** is LOW	*Handshake process* on Page 3-2	Valid
AXI4_ERRS_BRESP_X	A value of X on **BRESP** is not permitted when **BVALID** is HIGH	-	Valid
AXI4_ERRS_BVALID_RESET	**BVALID** is LOW for the first cycle after **ARESETn** goes HIGH	*Reset* on Page 11-2	Valid
AXI4_ERRS_BVALID_STABLE	When **BVALID** is asserted, then it must remain asserted until **BREADY** is HIGH	*Write response channel* on Page 3-3	Valid
AXI4_ERRS_BVALID_X	A value of X on **BVALID** is not permitted when not in reset	-	Valid

Table 4-3 Write response channel checking rules (continued)

Assertion	Description	Specification reference	AXI4-Lite
AXI4_RECM_BREADY_MAX_WAIT	Recommended that **BREADY** is asserted within MAXWAITS cycles of **BVALID** being asserted	-	Valid
AXI4_ERRM_BREADY_X	A value of X on **BREADY** is not permitted when not in reset	-	Valid
AXI4_ERRS_BRESP_AW	A slave must not take **BVALID** HIGH until after the write address is handshaken	*Write response dependencies* on Page 13-6	Valid
AXI4_ERRS_BUSER_STABLE	**BUSER** remains stable when **BVALID** is asserted and **BREADY** is LOW	*Handshake process* on Page 3-2	-
AXI4_ERRS_BUSER_X	A value of X on **BUSER** is not permitted when **BVALID** is HIGH	-	-
AXI4_ERRS_BRESP_WLAST	A slave must not take **BVALID** HIGH until after the last write data is handshaken	*Dependencies between channel handshake signals* on Page 3-6	Valid
AXI4_ERRS_BUSER_TIEOFF	**BUSER** must be stable when **BUSER_WIDTH** has been set to zero	-	-
AXI4_ERRS_BID_TIEOFF	**BID** must be stable when **ID_WIDTH** has been set to zero	-	-

4.1.4 Read address channel checks

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

Table 4-4 Read address channel checking rules

Assertion	Description	Specification reference	AXI4-Lite
AXI4_ERRM_ARID_STABLE	**ARID** remains stable when **ARVALID** is asserted, and **ARREADY** is LOW	*Handshake process* on Page 3-2	-
AXI4_ERRM_ARID_X	A value of X on **ARID** is not permitted when **ARVALID** is HIGH	-	-
AXI4_ERRM_ARADDR_BOUNDARY	A read burst cannot cross a 4KB boundary	*About addressing options* on Page 4-2	-
AXI4_ERRM_ARADDR_STABLE	**ARADDR** remains stable when **ARVALID** is asserted and **ARREADY** is LOW	*Handshake process* on Page 3-2	Valid
AXI4_ERRM_ARADDR_WRAP_ALIGN	A read transaction with a burst type of WRAP must have an aligned address	*Wrapping burst* on Page 4-6	-
AXI4_ERRM_ARADDR_X	A value of X on **ARADDR** is not permitted when **ARVALID** is HIGH	-	Valid
AXI4_ERRM_ARLEN_STABLE	**ARLEN** remains stable when **ARVALID** is asserted and **ARREADY** is LOW	*Handshake process* on Page 3-2	-
AXI4_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	-
AXI4_ERRM_ARLEN_X	A value of X on **ARLEN** is not permitted when **ARVALID** is HIGH	-	-

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

Assertion	Description	Specification reference	AXI4-Lite
AXI4_ERRM_ARSIZE	The size of a read transfer must not exceed the width of the data interface	*Burst size* on Page 4-4	-
AXI4_ERRM_ARSIZE_STABLE	**ARSIZE** remains stable when **ARVALID** is asserted, and **ARREADY** is LOW	*Handshake process* on Page 3-2	-
AXI4_ERRM_ARSIZE_X	A value of X on **ARSIZE** is not permitted when **ARVALID** is HIGH	-	-
AXI4_ERRM_ARBURST	A value of 2'b11 on **ARBURST** is not permitted when **ARVALID** is HIGH	Table 4-3 on Page 4-5	-
AXI4_ERRM_ARBURST_STABLE	**ARBURST** remains stable when **ARVALID** is asserted, and **ARREADY** is LOW	*Handshake process* on Page 3-2	-
AXI4_ERRM_ARBURST_X	A value of X on **ARBURST** is not permitted when **ARVALID** is HIGH	-	-
AXI4_ERRM_ARLOCK_STABLE	**ARLOCK** remains stable when **ARVALID** is asserted, and **ARREADY** is LOW	*Handshake process* on Page 3-2	-
AXI4_ERRM_ARLOCK_X	A value of X on **ARLOCK** is not permitted when **ARVALID** is HIGH	-	-
AXI4_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	-
AXI4_ERRM_ARCACHE_STABLE	**ARCACHE** remains stable when **ARVALID** is asserted, and **ARREADY** is LOW	*Handshake process* on Page 3-2	-
AXI4_ERRM_ARCACHE_X	A value of X on **ARCACHE** is not permitted when **ARVALID** is HIGH	-	-
AXI4_ERRM_ARPROT_STABLE	**ARPROT** remains stable when **ARVALID** is asserted, and **ARREADY** is LOW	*Handshake process* on Page 3-2	Valid
AXI4_ERRM_ARPROT_X	A value of X on **ARPROT** is not permitted when **ARVALID** is HIGH	*Read address channel* on Page 3-4	Valid
AXI4_ERRM_ARVALID_RESET	**ARVALID** is LOW for the first cycle after **ARESETn** goes HIGH	*Reset* on Page 11-2	Valid
AXI4_ERRM_ARVALID_STABLE	When **ARVALID** is asserted, then it remains asserted until **ARREADY** is HIGH	*Read address channel* on Page 3-4	Valid
AXI4_ERRM_ARVALID_X	A value of X on **ARVALID** is not permitted when not in reset	-	Valid
AXI4_ERRS_ARREADY_X	A value of X on **ARREADY** is not permitted when not in reset	-	Valid
AXI4_RECS_ARREADY_MAX_WAIT	Recommended that **ARREADY** is asserted within MAXWAITS cycles of **ARVALID** being asserted	-	Valid
AXI4_ERRM_ARUSER_STABLE	**ARUSER** remains stable when **ARVALID** is asserted, and **ARREADY** is LOW	*Handshake process* on Page 3-2	-
AXI4_ERRM_ARUSER_X	A value of X on **ARUSER** is not permitted when **ARVALID** is HIGH	-	-
AXI4_ERRM_ARQOS_STABLE	**ARQOS** 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	AXI4-Lite
AXI4_ERRM_ARQOS_X	A value of X on **ARQOS** is not permitted when **ARVALID** is HIGH	-	-
AXI4_ERRM_ARREGION_STABLE	**ARREGION** remains stable when **ARVALID** is asserted, and **ARREADY** is LOW	*Handshake process* on Page 3-2	-
AXI4_ERRM_ARREGION_X	A value of X on **ARREGION** is not permitted when **ARVALID** is HIGH	-	-
AXI4_ERRM_ARLEN_FIXED	Transactions of burst type FIXED cannot have a length greater than 16 beats	*Limitations of use* on Page 13-2	-
AXI4_ERRM_ARLEN_LOCK	Exclusive access transactions cannot have a length greater than 16 beats	*Limitations of use* on Page 13-2	-
AXI4_ERRM_ARUSER_TIEOFF	**ARUSER** must be stable when **ARUSER_WIDTH** has been set to zero	-	-
AXI4_ERRM_ARID_TIEOFF	**ARID** must be stable when **ID_WIDTH** has been set to zero	-	-

4.1.5 Read data channel checks

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

Table 4-5 Read data channel checking rules

Assertion	Description	Specification reference	AXI4-Lite
AXI4_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	-
AXI4_ERRS_RID_STABLE	**RID** remains stable when **RVALID** is asserted, and **RREADY** is LOW.	*Handshake process* on Page 3-2	-
AXI4_ERRS_RID_X.	A value of X on **RID** is not permitted when **RVALID** is HIGH.	-	-
AXI4_ERRS_RDATA_NUM	The number of read data items must match the corresponding **ARLEN**.	Table 4-1 on Page 4-3	Valid
AXI4_ERRS_RDATA_STABLE	**RDATA** remains stable when **RVALID** is asserted, and **RREADY** is LOW.	*Handshake process* on Page 3-2	Valid
AXI4_ERRS_RDATA_X	A value of X on **RDATA** valid byte lanes is not permitted when **RVALID** is HIGH.	-	Valid
AXI4_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	Valid
AXI4_ERRS_RRESP_STABLE	**RRESP** remains stable when **RVALID** is asserted, and **RREADY** is LOW.	*Handshake process* on Page 3-2	Valid
AXI4_ERRS_RRESP_X	A value of X on **RRESP** is not permitted when **RVALID** is HIGH.	-	Valid

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

Assertion	Description	Specification reference	AXI4-Lite
AXI4_ERRS_RLAST_ALL_DONE_EOS	All outstanding read bursts must have completed.	-	-
AXI4_ERRS_RLAST_STABLE	**RLAST** remains stable when **RVALID** is asserted, and **RREADY** is LOW.	*Handshake process* on Page 3-2	-
AXI4_ERRS_RLAST_X	A value of X on **RLAST** is not permitted when **RVALID** is HIGH.	-	-
AXI4_ERRS_RVALID_RESET	**RVALID** is LOW for the first cycle after **ARESETn** goes HIGH.	*Reset* on Page 11-2	Valid
AXI4_ERRS_RVALID_STABLE	When **RVALID** is asserted, then it must remain asserted until **RREADY** is HIGH.	*Read data channel* on Page 3-5	Valid
AXI4_ERRS_RVALID_X	A value of X on **RVALID** is not permitted when not in reset.	-	Valid
AXI4_ERRM_RREADY_X	A value of X on **RREADY** is not permitted when not in reset.	-	Valid
AXI4_RECM_RREADY_MAX_WAIT	Recommended that **RREADY** is asserted within MAXWAITS cycles of **RVALID** being asserted.	-	Valid
AXI4_ERRS_RUSER_X	A value of X on **RUSER** is not permitted when **RVALID** is HIGH.	-	-
AXI4_ERRS_RUSER_STABLE	**RLAST** remains stable when **RVALID** is asserted, and **RREADY** is LOW.	*Handshake process* on Page 3-2	-
AXI4_ERRS_RUSER_TIEOFF	**RUSER** must be stable when **RUSER_WIDTH** has been set to zero.	-	-
AXI4_ERRS_RID_TIEOFF	**RID** must be stable when **ID_WIDTH** has been set to zero.	-	-

4.1.6 Low-power interface rules

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

Table 4-6 Low-power interface checking rules

Assertion	Description	Specification reference	AXI4-Lite
AXI4_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	-
AXI4_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	-
AXI4_ERRL_CSYSREQ_X	A value of X on **CSYSREQ** is not permitted when not in reset	-	-
AXI4_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	-

Table 4-6 Low-power interface checking rules (continued)

Assertion	Description	Specification reference	AXI4-Lite
AXI4_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	-
AXI4_ERRL_CSYSACK_X	A value of X on **CSYSACK** is not permitted when not in reset	-	-
AXI4_ERRL_CACTIVE_X	A value of X on **CACTIVE** is not permitted when not in reset	-	-

4.1.7 Exclusive access checks

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

Table 4-7 Address channel exclusive access checking rules

Assertion	Description	Specification reference	AXI4-Lite
AXI4_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 6-4	-
AXI4_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 6-4	-
AXI4_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 6-4	-
AXI4_ERRM_EXCL_MAX	128 is the maximum number of bytes that can be transferred in an exclusive burst	*Exclusive access restrictions* on Page 6-4	-
AXI4_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	-

4.1.8 Internal logic checks

Table 4-8 shows the internal logic checks.

Table 4-8 Internal logic checks

Assertion	Description	Specification Reference	AXI4-Lite
AXI4_AUXM_DATA_WIDTH	DATA_WIDTH parameter is 32, 64, 128, 256, 512, or 1024	-	-
AXI4_AUXM_RCAM_OVERFLOW	Read CAM overflow, increase MAXRBURSTS parameter	-	Valid
AXI4_AUXM_RCAM_UNDERFLOW	Read CAM underflow	-	Valid
AXI4_AUXM_WCAM_OVERFLOW	Write CAM overflow, increase MAXWBURSTS parameter	-	Valid
AXI4_AUXM_WCAM_UNDERFLOW	Write CAM underflow	-	Valid
AXI4_AUXM_ADDR_WIDTH	Parameter ADDR_WIDTH must be between 32 bits and 64 bits inclusive	-	Valid
AXI4_AUXM_EXMON_WIDTH	Parameter EXMON_WIDTH must be greater than or equal to 1	-	-

Table 4-8 Internal logic checks (continued)

Assertion	Description	Specification Reference	AXI4-Lite
AXI4_AUXM_MAXRBURSTS	Parameter MAXRBURSTS must be greater than or equal to 1	-	Valid
AXI4_AUXM_MAXWBURSTS	Parameter MAXWBURSTS must be greater than or equal to 1	-	Valid
AXI4_AUXM_EXCL_OVERFLOW	Exclusive access monitor overflow, increase EXMON_WIDTH parameter	-	-

4.1.9 Additional checks for AXI4-Lite

Table 4-9 shows the additional rules for AXI4-Lite.

Table 4-9 Additional AXI4-Lite checks

Assertion	Description	Specification Reference
AXI4LITE_ERRS_RRESP_EXOKAY	A slave must not give an EXOKAY response on an AXI4-Lite interface	*Unsupported signals* on Page 14-3
AXI4LITE_ERRS_BRESP_EXOKAY	A slave must not give an EXOKAY response on an AXI4-Lite interface	*Unsupported signals* on Page 14-3
AXI4LITE_AUXM_DATA_WIDTH	DATA_WIDTH parameter is 32 or 64	-

4.2 AXI4-Stream™ protocol assertion descriptions

This section describes the protocol assertions, and indicates the area of the AMBA 4 AXI4-Stream Protocol v1.0 specification to which they apply. Table 4-10 shows the streaming interface checking rules.

Table 4-10 Streaming channel assertion rules

Assertion	Description	Specification reference
AXI4STREAM_ERRM_TVALID_RESET	**TVALID** is LOW for the first cycle after **ARESETn** goes HIGH	Reset on Page 2-11
AXI4STREAM_ERRM_TID_STABLE	**TID** remains stable when **TVALID** is asserted, and **TREADY** is LOW	*Handshake process* on Page 2-3
AXI4STREAM_ERRM_TDEST_STABLE	**TDEST** remains stable when **TVALID** is asserted, and **TREADY** is LOW	*Handshake process* on Page 2-3
AXI4STREAM_ERRM_TDATA_STABLE	**TDATA** remains stable when **TVALID** is asserted, and **TREADY** is LOW	*Handshake process* on Page 2-3
AXI4STREAM_ERRM_TSTRB_STABLE	**TSTRB** remains stable when **TVALID** is asserted, and **TREADY** is LOW	*Handshake process* on Page 2-3
AXI4STREAM_ERRM_TLAST_STABLE	**TLAST** remains stable when **TVALID** is asserted, and **TREADY** is LOW	*Handshake process* on Page 2-3
AXI4STREAM_ERRM_TKEEP_STABLE	**TKEEP** remains stable when **TVALID** is asserted, and **TREADY** is LOW	*Handshake process* on Page 2-3
AXI4STREAM_ERRM_TVALID_STABLE	When **TVALID** is asserted, then it must remain asserted until **TREADY** is HIGH	*Handshake process* on Page 2-3
AXI4STREAM_RECS_TREADY_MAX_WAIT	Recommended that **TREADY** is asserted within **MAXWAITS** cycles of **TVALID** being asserted	-
AXI4STREAM_ERRM_TID_X	A value of X on **TID** is not permitted when **TVALID** is HIGH	-
AXI4STREAM_ERRM_TDEST_X	A value of X on **TDEST** is not permitted when **TVALID** is HIGH	-
AXI4STREAM_ERRM_TDATA_X	A value of X on **TDATA** is not permitted when **TVALID** is HIGH	
AXI4STREAM_ERRM_TSTRB_X	A value of X on **TSTRB** is not permitted when **TVALID** is HIGH	-
AXI4STREAM_ERRM_TLAST_X	A value of X on **TLAST** is not permitted when **TVALID** is HIGH	-
AXI4STREAM_ERRM_TKEEP_X	A value of X on **TKEEP** is not permitted when **TVALID** is HIGH	-
AXI4STREAM_ERRM_TVALID_X	A value of X on **TVALID** is not permitted when not in reset	-
AXI4STREAM_ERRS_TREADY_X	A value of X on **TREADY** is not permitted when not in reset	-
AXI4STREAM_ERRM_TUSER_X	A value of X on **TUSER** is not permitted when not in reset	-

Table 4-10 Streaming channel assertion rules (continued)

Assertion	Description	Specification reference
AXI4STREAM_ERRM_TUSER_STABLE	**TUSER** payload signals must remain constant whilst **TVALID** is asserted, and **TREADY** is de-asserted	*Handshake process* on Page 2-3
AXI4STREAM_ERRM_STREAM_ALL_DONE_EOS	At the end of simulation, all streams have had their corresponding **TLAST** transfer	-
AXI4STREAM_ERRM_TKEEP_TSTRB	If **TKEEP** is de-asserted, then **TSTRB** must also be de-asserted	Table 2-2 on Page 2-9
AXI4STREAM_ERRM_TDATA_TIEOFF	**TDATA** must be stable while **DATA_WIDTH_BYTES** has been set to zero	-
AXI4STREAM_ERRM_TKEEP_TIEOFF	**TKEEP** must be stable while **DATA_WIDTH_BYTES** has been set to zero	-
AXI4STREAM_ERRM_TSTRB_TIEOFF	**TSTRB** must be stable while **DATA_WIDTH_BYTES** has been set to zero	-
AXI4STREAM_ERRM_TID_TIEOFF	**TID** must be stable while **ID_WIDTH** has been set to zero	-
AXI4STREAM_ERRM_TDEST_TIEOFF	**TDEST** must be stable while **DEST_WIDTH** has been set to zero	-
AXI4STREAM_ERRM_TUSER_TIEOFF	**TUSER** must be stable while **USER_WIDTH** has been set to zero	-
AXI4STREAM_AUXM_TID_TDTEST_WIDTH	The value of **ID_WIDTH** + **DEST_WIDTH** must not exceed 24	-

Appendix A
Example Usage

This appendix provides an example transcript from the protocol assertions. 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.

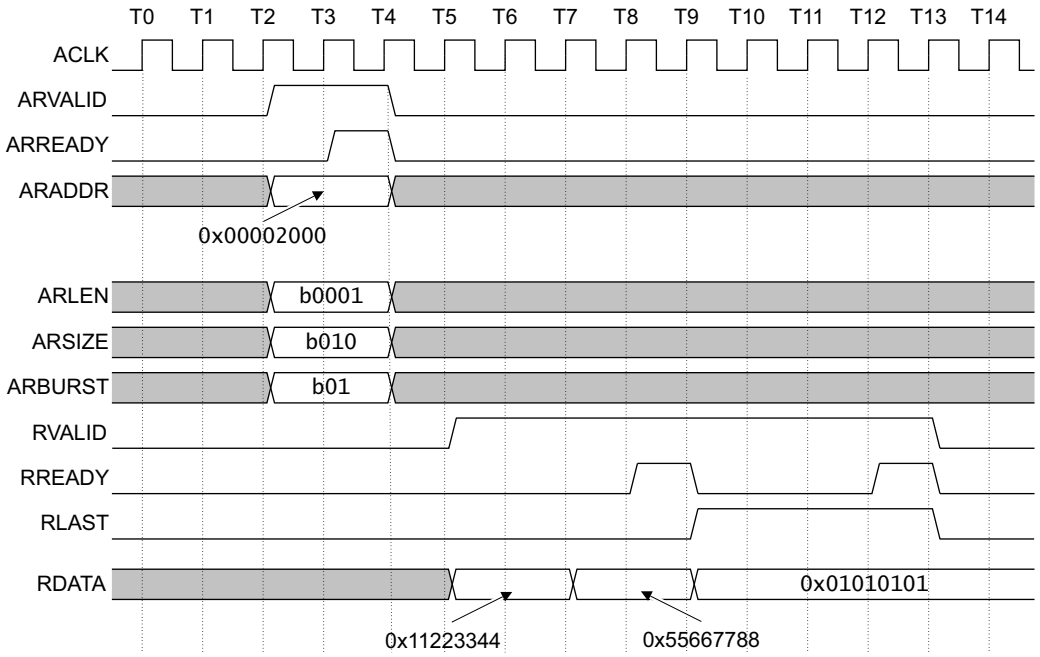


Figure A-1 RDATA stable failure

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

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

Example A-1 RDATA stable failure

```
# Loading sv_std.std
# Loading work.avip_testbench
# Loading work.Axi4PC
# Loading work.BaseClk
# do startup.do
# AXI4_INFO: Running Axi4PC $State
# ** Error: AXI4_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.
# Time: 1050 ns Started: 950 ns Scope: avip_testbench.uAxi4PC.axi4_errs_rdata_stable File: ../Axi4PC.sv
     Line: 2595 Expr: $stable(RDATA|~RdataMask)
# ** Note: $finish    : stim.svh(84)
# Time: 3960 ns  Iteration: 1  Instance: /avip_testbench
```

Appendix B
Revisions

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

Table B-1 Issue A

Change	Location	Affects
No changes, first release	-	-

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.

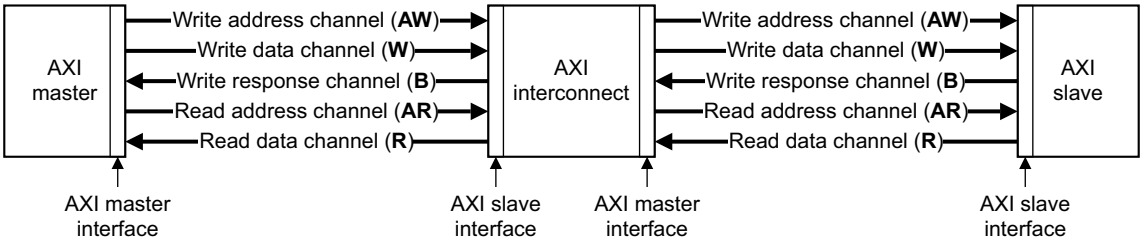
AMBA *See* Advanced Microcontroller Bus Architecture.

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] signal 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.

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.

---

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.

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** bus.

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.