ARM® AMBA® 5 CHI Protocol Checker

Revision: r0p0

User Guide

Confidential

DUI0733A (ID090114)

ARM AMBA 5 CHI Protocol Checker
User Guide

Release Information

The following changes have been made to this book.

Change history

Date	Issue	Confidentiality	Change
2 July 2014	A	Confidential	First release for r0p0

Proprietary Notice

Confidentiality Status

This document is Confidential. This document may only be used and distributed 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
ARM AMBA 5 CHI Protocol Checker User Guide

Preface

This preface introduces the *ARM® AMBA® 5 CHI Protocol Checker User Guide*. It contains the following sections:

About this book

This is the User Guide for the ARM AMBA 5 CHI Protocol Checker.

Product revision status

The r*n*p*n* identifier indicates the revision status of the product described in this book, for example, r0p0, where:

r*n* Identifies the major revision of the product.

p*n* Identifies the minor revision or modification status of the product.

Intended audience

This book is for system designers, system integrators, and verification engineers who want to check that a design is not violating any of the rules or recommendations of the AMBA 5 CHI protocol. You must have experience of writing or integrating SystemVerilog blocks.

Using this book

This book is organized into the following chapters:

Chapter 1 *Introduction*

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

Chapter 2 *Implementation and Integration*

Read this chapter for information on where to place the protocol checker in your design, the integration flow, specific signal connections with an example file listing, and setting up your simulator.

Chapter 3 *Configuring and controlling the AMBA 5 CHI protocol checker*

Read this chapter for a description of the protocol checker parameters.

Appendix A *Example Usage*

Read this appendix for an example of the protocol checker detecting a violation of protocol.

Appendix B *Revisions*

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

Glossary

The *ARM® Glossary* is a list of terms used in ARM documentation, together with definitions for those terms. The *ARM® Glossary* does not contain terms that are industry standard unless the ARM meaning differs from the generally accepted meaning.

See *ARM® Glossary* `http://infocenter.arm.com/help/topic/com.arm.doc.aeg0014-/index.html`.

Conventions

This book uses the conventions that are described in:

- *Typographical conventions* on page vi.
- *Timing diagrams* on page vi.
- *Signals* on page vii.

Typographical conventions

The following table describes the typographical conventions:

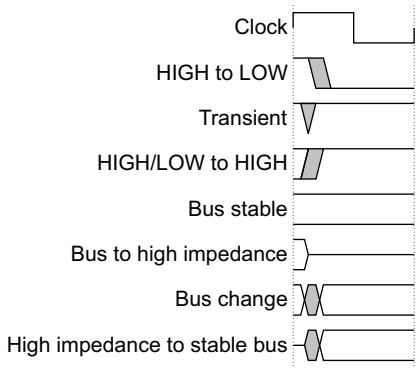
Typographical conventions

Style	Purpose
italic	Introduces special terminology, denotes 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>	Encloses replaceable terms for assembler syntax where they appear in code or code fragments. For example: `MRC p15, 0 <Rd>, <CRn>, <CRm>, <Opcode_2>`
SMALL CAPITALS	Used in body text for a few terms that have specific technical meanings, that are defined in the *ARM® Glossary*. For example, IMPLEMENTATION DEFINED, IMPLEMENTATION SPECIFIC, UNKNOWN, and UNPREDICTABLE.

Timing diagrams

The figure *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.
- LOW for active-LOW signals.

Lower-case n At the start or end of a signal name denotes an active-LOW signal.

Additional reading

This section lists publications by ARM and by third parties.

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

ARM publications

This book contains information that is specific to this product. The following publication is confidential and only available to licensees:
- *ARM® AMBA® 5 CHI Architecture Specification* (ARM IHI 0050).

Other publications

This section lists relevant documents published by third parties:

- SystemVerilog technical papers, tutorials, and downloads http://www.systemverilog.org.
- 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, DUI0733A.
- The page numbers that your comments apply to.
- A concise explanation of your comments.

ARM also welcomes general suggestions for additions and improvements.

Note

ARM tests the PDF only in Adobe Acrobat and Acrobat Reader, and cannot guarantee the quality of the represented document when used with any other PDF reader.

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 to check that your design is not violating any of the rules or recommendations of the AMBA 5 CHI protocol.

The behavior of the interface that you test is checked against the protocol by a series of assertions. For information on the AMBA 5 CHI protocol rules, see the *ARM® AMBA® 5 CHI Architecture Specification*. See *ARM publications on page vii*.

This guide describes the contents of the SystemVerilog files, and how to integrate them into a design. It also describes the correct use of the protocol checker assertions with simulators to flag errors, warnings, or both, during design simulation.

1.2 Tools

The release note provides information about the tools tested and that support the AMBA 5 CHI protocol checker.

The protocol checker is written in SystemVerilog. SystemVerilog is a *Hardware Description and Verification Language* (HDVL) standard that extends the established Verilog language.

Note

The version of SystemVerilog supported is IEEE 1800-2005.

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 SystemVerilog files with a design. For more information on the protocol checker files, see *AMBA 5 CHI protocol checker files on page 2-3*.

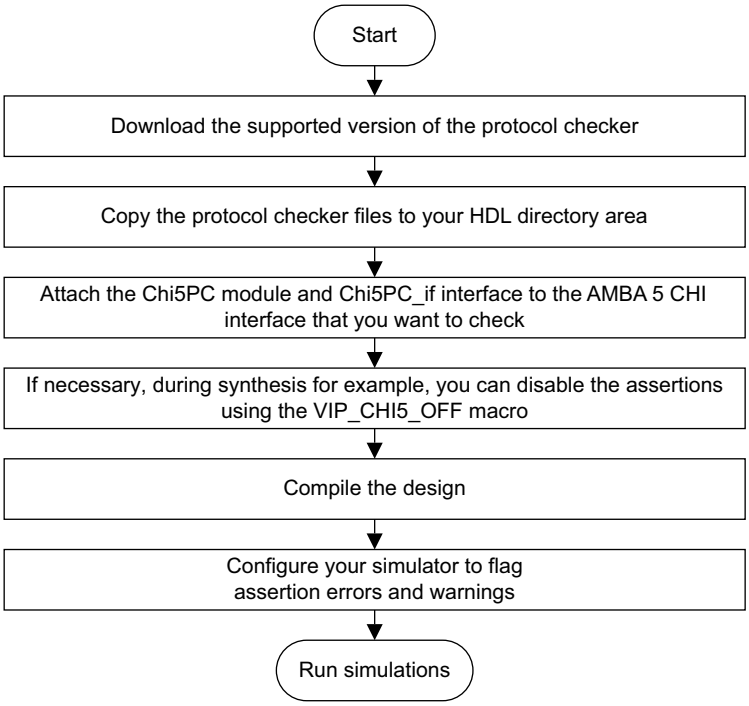


Figure 2-1 Integration flow

2.2 Implementing the protocol checker in your design directory

You can implement the protocol checker for the following node types:

- RN-F.
- RN-D.
- RN-I.
- SN-F.
- SN-I.

Use the NODE_TYPE parameter to set the node type. See *Configuring the AMBA 5 CHI interface on page 3-2* for more information.

This section describes:

- *AMBA 5 CHI protocol checker files*.
- *Location of AMBA 5 CHI protocol checker files*.

2.2.1 AMBA 5 CHI protocol checker files

Figure 2-2 shows the contents of the directory that contains the protocol checker.

```
sva/
├──Chi5PC.sv
├──Chi5PC_Chi5_defines.v
├──Chi5PC_Chi5_flit_defines.svh
├──Chi5PC_EXCL.sv
├──Chi5PC_FlitTrace.sv
├──Chi5PC_Retry_Crdgrnt.sv
├──Chi5PC_SAM_pkg.svh
├──Chi5PC_SnoopTrace.sv
├──Chi5PC_XLA.sv
├──Chi5PC_defines.v
├──Chi5PC_if.sv
├──Chi5PC_pkg.svh
└──files.vc
```

Figure 2-2 Protocol checker directory contents for AMBA 5 CHI

2.2.2 Location of AMBA 5 CHI protocol checker files

You can locate the AMBA 5 CHI protocol checker files in any location that is included in the search path of your simulator. For information on viewing or setting the search path, see the simulator documentation.

Alternatively, you can locate the protocol checker files in the same directory as the interface you want to check. Figure 2-3 shows an example location where you can place the protocol checker SystemVerilog files within your design RTL.

```
RTL design directory
├──Top-level HDL file with protocol assertions module and interface instantiated
├──Other HDL files
└──Protocol checker SystemVerilog files
```

Figure 2-3 Location of the AMBA 5 CHI protocol checker SystemVerilog files

Note

If you locate the protocol checker files in another location then you must ensure that they are on the simulator search path. For information on viewing or setting the search path, see the simulator documentation. Alternatively, you can update the `files.vc` file in the `sva` directory to contain the correct relative path to find the installation location of the protocol checker, then include it in the `.vc` file for your own design.

2.3 Instantiating the protocol checker interface and module

You must instantiate the protocol checker so that it can check the correct use of all signals and transitions on the interface.

Do this using the following steps:

1. Instantiate both the Chi5PC module and the Chi5PC_if interface.
2. Attach the Chi5PC to the Chi5PC_if interface by setting the Chi5PC_if interface port of the Chi5PC to the instance name of the Chi5PC_if.
3. Connect the AMBA 5 CHI signals in your design environment to the Chi5PC_if interface.

Note

You must instantiate both the Chi5PC_if interface and the Chi5PC module for the protocol checker to operate correctly. This is because the protocol checker modules access the AMBA 5 CHI interface signals through the Chi5PC_if SystemVerilog interface.

Example SystemVerilog to instantiate the AMBA 5 CHI protocol checker shows an example of how the Chi5PC module and Chi5PC_if interface are instantiated in a top-level Verilog file.

See *ARM publications on page vii* for the specifications that describe the AMBA 5 CHI signals.

2.3.1 Example SystemVerilog to instantiate the AMBA 5 CHI protocol checker

Example 2-1 shows sample HDL that instantiates the protocol checker module and interface for AMBA 5 CHI. You can, if necessary, override any of the protocol checker parameters by using `defparam` at this level.

Example 2-1 Example SystemVerilog to instantiate the AMBA 5 CHI protocol checker

```
Chi5PC_if #( .MAX_OS_REQ(32),
             .MAX_OS_SNP(16)
             .MAX_OS_EXCL(8),
             .numChi5nodes(6),
             .nodeIdQ('{0,1,2,4,8,16}),
             .NODE_ID(2),
             .NODE_TYPE(Chi5PC_pkg::SNI),
             .devQ('{Chi5PC_pkg::HNF, // 0
                     Chi5PC_pkg::SNF, // 1
                     Chi5PC_pkg::SNI, // 2
                     Chi5PC_pkg::SNF, // 4
                     Chi5PC_pkg::SNI, // 8
                     Chi5PC_pkg::HNI}), // 16
             .DAT_FLIT_WIDTH(Chi5PC_pkg::CHI5PC_DAT_128B)  // 128 Bit
           ) CHI5PCInt();

 Chi5PC #(    .ErrorOn_SW(1),
                .RecommendOn(1),
                .RecommendOn_Haz(1),
                .MAX_OS_REQ(32),
                .MAX_OS_EXCL(8),
                .MAX_OS_SNP(16),
                .NODE_ID(2),
                .CRDGRANT_BEFORE_RETRY(1),
                .MAXLLCREDITS(16),
                .DAT_FLIT_WIDTH(128),
 ) u_Chi5PC ( .Chi5_in (CHI5PCInt), .SCLK (SCLK));
```

2.4 Configuring the system address map

The *ARM® AMBA® 5 CHI Architecture Specification* states that each RN and HN in the system must have a *System Address Map* (SAM) to determine the target ID of a request. The protocol checker requires knowledge of this mapping function so that it can translate the request type and address into a target ID that it uses in protocol checking algorithms.

The protocol checker files include an editable package file called Chi5PC_SAM_pkg.svh. Whenever a target ID issued at the *Device Under Test* (DUT) interface is subject to remapping, you must replicate this remapping function in the code in Chi5PC_SAM_pkg.svh. By default, the contents of Chi5PC_SAM_pkg.svh represent the case where the target ID is not remapped.

Chapter 3
Configuring and controlling the AMBA 5 CHI protocol checker

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

Caution

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

3.1 Configuring the AMBA 5 CHI interface

You must configure the AMBA 5 CHI interface and system parameters before you can use the AMBA 5 CHI protocol checker.

Table 3-1 shows the user-defined parameters for setting the interface characteristics for AMBA 5 CHI. Use the Required in column to identify the parts of the protocol checker that require that parameter and change the parameter values to match your design specification. For more information on AMBA 5 CHI, see *ARM publications on page vii*.

Note

The parameters passed to the Chi5PC module and Chi5PC_if interface must match. Therefore, you must set to the same value any parameters that are required in both Chi5PC and Chi5PC_if.

Table 3-1 AMBA 5 CHI interface and protocol checker parameters

Name	Description	Default value	Required in
DAT_FLIT_WIDTH	Width of the data flit. You can calculate this width by adding the widths of all the fields that make up a data flit. Fields RSVDC, BE and Data are variable, and BE is dependent on Data. All other fields have fixed width.	194	Chi5PC and Chi5PC_if
MAX_OS_REQ	Maximum number of outstanding transactions on the CHI interface. This includes outstanding transactions that are in a retry state.	8	Chi5PC and Chi5PC_if
MAX_OS_SNP	Maximum number of outstanding snoops on the CHI interface.	16	Chi5PC and Chi5PC_if
MAX_OS_EXCL	Maximum number of exclusive accesses active on the CHI interface.	8	Chi5PC and Chi5PC_if
NODE_TYPE	Type of node that the protocol checker is attached to. The possible values for this parameter are: • RNF. • RND. • RNI. • SNF. • SNI.	RNF	Chi5PC and Chi5PC_if
numChi5nodes	Total number of CHI nodes in the system. This includes the MN and HN node types. See *Protocol checker numChi5nodes, devQ and nodeIdQ parameters on page 3-5*.	4	Chi5PC and Chi5PC_if

Table 3-1 AMBA 5 CHI interface and protocol checker parameters (continued)

Name	Description	Default value	Required in
devQ	Vector of node types in the system, including the DUT node. The possible node types are: • RNF. • RND. • RNI. • SNF. • SNI. • HNF. • HNI. • MN. • HNI_MN. • HNF_MN. • HNF_HNI. • HNF_HNI_MN. See *Protocol checker numChi5nodes, devQ and nodeIdQ parameters on page 3-5*.	RNF,RNF,RNF,RNF	Chi5PC_if only
nodeIdQ	Vector that provides the node ID values of the nodes identified by devQ. See *Protocol checker numChi5nodes, devQ and nodeIdQ parameters on page 3-5*.	0,0,0,0	Chi5PC_if only
MAXLLCREDITS	Maximum number of link layer credits this interface can issue.	16	Chi5PC and Chi5PC_if
PCMODE	Determines whether a race condition is permitted at a specified location relative to an AMBA 5 CHI node. See *Protocol checker PCMODE parameter on page 3-6* for more information.	LOCAL	Chi5PC only
REQ_RSVDC_WIDTH	Width, in bits, of the Reserved-for-Customer field in the request flits.The possible values for this parameter are: • 0. • 4. • 8. • 12. • 16. • 24. • 32.	4	Chi5PC and Chi5PC_if
DAT_RSVDC_WIDTH	Width, in bits, of the Reserved-for-Customer field in the data flits.The possible values for this parameter are: • 0. • 4. • 8. • 12. • 16. • 24. • 32.	4	Chi5PC and Chi5PC_if
MAXLLCREDITS_IN_RXDEACTIVATE	Maximum number of link layer credits this link can issue while in the deactivate state.	MAXLLCREDITS	Chi5PC and Chi5PC_if

Table 3-1 AMBA 5 CHI interface and protocol checker parameters (continued)

Name	Description	Default value	Required in
NODE_ID	The ID of the node that the protocol checker is monitoring.	0	Chi5PC and Chi5PC_if
CRDGRANT_BEFORE_RETRY	Indicates whether the interface can support the issuing of a PCrdGrant before the corresponding RetryAck for a given transaction. If you enable this parameter by setting it to 1, complex checks are enabled in the protocol checker. You can achieve a significant simulation performance improvement by setting this parameter to 0 to disable these checks. **Note** If you set the parameter to 0 and a PCrdGrant is observed before a RetryAck for a given transaction then the protocol checker issues an error message.	1	Chi5PC only
Barrier_Order	Rules are applied to the ordering of transactions around barriers. By default, the protocol checker expects all transactions to observe barrier ordering. However, it is possible that some transactions are not required to observe barrier ordering, in which case this parameter can be set to 0.	1	Chi5PC only

3.2 Protocol checker numChi5nodes, devQ and nodeIdQ parameters

To perform certain rule checks when attached to a single AMBA 5 CHI interface, the protocol checker requires knowledge of other nodes with which the DUT node communicates. To obtain information about other nodes, the protocol checker uses the following parameters:

`numChi5nodes` Provides the number of nodes that are interacting with the DUT node. This number includes the DUT node.

`devQ` Provides a vector of the node types in the system with which the DUT exchanges transactions. The vector includes the DUT node.

`nodeIdQ` Provides the node ID values of the nodes identified by `devQ`. The values provided include the ID of the DUT node.

Note

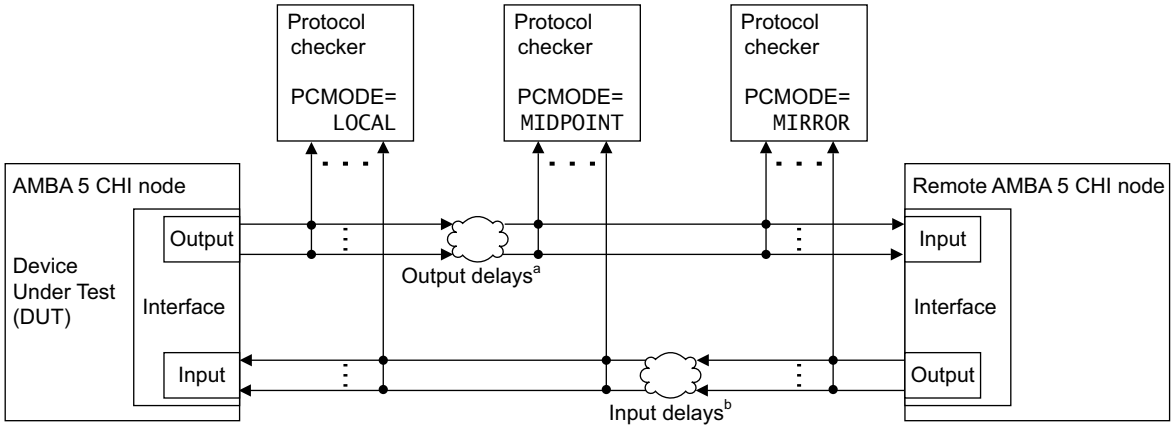
- The vector size of `devQ` and `nodeIdQ` must equal that of `numChi5nodes`.
- There must be a positional one-one mapping between `devQ` and the corresponding entries in `nodeIdQ`.

3.3 Protocol checker PCMODE parameter

Variable delays within a system can result in race conditions occurring on the inputs of an AMBA 5 CHI interface. This means that, even though the protocol might govern the order in which an interface sends output signals, those signals might arrive as inputs on another interface in a different order.

The relative position of the protocol checker to a particular interface determines the races that are permitted, and therefore, the signal delays that might occur. To determine whether a particular race condition is permitted, the protocol checker uses the PCMODE parameter to model the different combinations of permitted input and output signal races.

Figure 3-1 shows the delays that might be introduced into a system and how you can configure the protocol checker to enable it to interpret these race conditions correctly.



a. Variable delays across the link might cause the DUT outputs to race at the remote interface inputs.
b. Variable delays across the link might cause the remote interface outputs to race at the DUT inputs.

Figure 3-1 Using PCMODE to check for race conditions

Note

Figure 3-1 shows a generic case that applies to both RN and SN node types.

in Figure 3-1, the DUT is the AMBA 5 CHI node. The protocol checker uses the following settings to describe how it handles race conditions:

LOCAL — This setting models the situation where the protocol checker is located at the DUT interface. Races on the DUT inputs are supported, and races on the DUT outputs are not permitted.

MIDPOINT — This setting models the situation where the protocol checker is located somewhere on the link between the DUT and remote interfaces. Races on both the DUT inputs and outputs are supported.

MIRROR — This setting models the situation where the protocol checker is located at the remote interface. Races on the DUT outputs are supported, and races on the DUT inputs are not permitted.

NORACE — This setting models the ideal situation where there are no delays across the link. No races on the DUT inputs or outputs are permitted.

Table 3-2 shows the input and output race conditions that the protocol checker permits for each PCMODE setting.

Table 3-2 PCMODE settings

PCMODE setting	Race condition permitted	
	Output port	Input port
LOCAL	No	Yes
MIRROR	Yes	No
MIDPOINT	Yes	Yes
NORACE	No	No

3.4 Controlling the protocol checker

You can control the protocol checker and the messages it reports using mechanisms provided by both the simulator and the protocol checker.

The following sections describe the controls that the protocol checker provides:

- *Disabling recommended rules*.
- *Disabling the protocol checker on page 3-9*.

In addition to the controls that the protocol checker provides the simulator can provide additional command variables to control the available assertion options. These command variables might provide the ability to:

- Suppress or enable assertion warnings.
- Select assertion report messages to display.
- Set a minimum severity level for displaying assertion report messages.
- Set a minimum severity level for which an assertion causes the simulator to stop.

3.4.1 Disabling recommended rules

The AMBA 5 CHI protocol checker includes several recommended rule sets that you can disable. Table 3-3 shows the user-defined parameters for disabling recommended rules in the protocol checker.

Table 3-3 Display parameters

Name	Description	Default
`RecommendOn`	Enable or disable reporting of protocol recommendations.	`0b1`, enabled
`RecommendOn_Haz`	Enable or disable the hazard checking rules. **Note** Disabling these rules can provide a simulation performance improvement.	`0b1`, enabled
`ErrorOn_SW`	Enable or disable the software rules.	`0b1`, enabled
`ErrorOn_Data_X`	Enable or disable X-checking on data flits.	`0b1`, enabled

Note

`RecommendOn_Haz` is a subset of `RecommendOn`. If `RecommendOn` is `0b0`, that is, disabled, then the hazard checking rules are disabled regardless of the settings of `RecommendOn_Haz`.

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

```
CHI5_WARN: All recommended CHI5 rules have been disabled by the RecommendOn parameter
```

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

```
CHI5_WARN: All CHI5 software rules have been disabled by the ErrorOn_SW parameter
```

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

```
CHI5_WARN: All recommended VIP_CHI5_REC_REQ_HAZ rules have been disabled by the RecommendOn_Haz parameter
```

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

```
CHI5_WARN: All Data X-checks have been disabled by the ErrorOn_Data_X parameter
```

3.4.2 Disabling the protocol checker

You can disable the protocol checker at compile time without removing the code that instantiates the protocol checker. You might want to do this if you are using a structural design file in both simulation and non-simulation tools.

To disable the protocol checker, compile it with the following option:

```
+define+CHI5PC_OFF
```

Appendix A
Example Usage

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

- *Illegal ReadNoSnp request failure on page A-2*.

A.1 Illegal ReadNoSnp request failure

Figure A-1 shows the timing diagram for a failure of the VIP_CHI5_ERR_REQ_CTL_READNOSNP check.

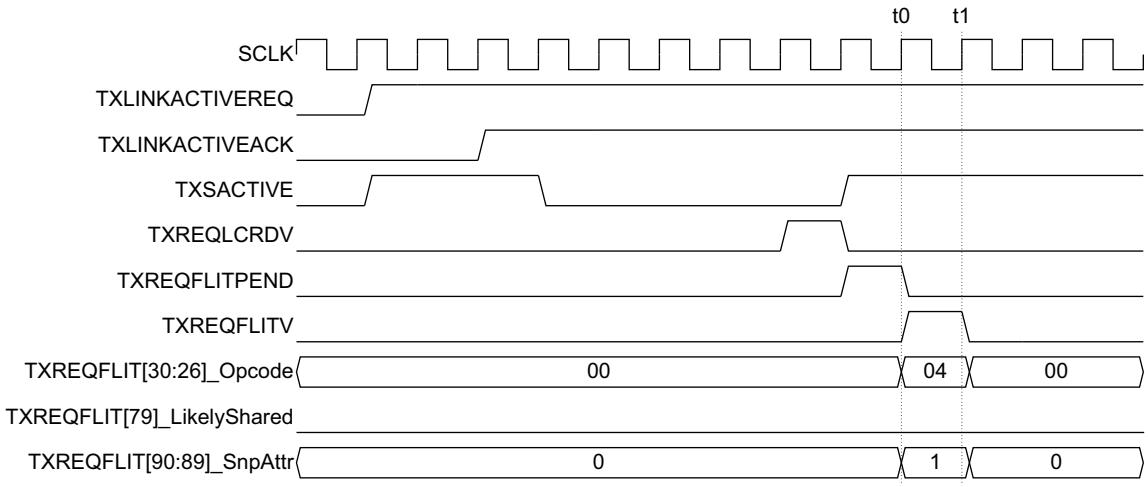


Figure A-1 Illegal ReadNoSnp request failure

TXREQFLIT[30:26]_Opcode changes at t0 to a transaction Opcode that represents the ReadNoSnp operation. **TXREQFLIT[79]_LikelyShared** must remain LOW for the ReadNoSnp operation and therefore is valid in this example. However, the value 0x1 for **TXREQFLIT[90:89]_SnpAttr** is not valid. This is because bit 89, the snoopable bit, is HIGH, and a transaction marked as snoopable is illegal for a ReadNoSnp opcode. The protocol checker samples this failure and generates an error message at t1, that is, at the next rising clock edge.

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

Example A-1 Illegal ReadNoSnp request failure

```
# Loading sv_std.std
# Loading work.VIP_CHI5_PACK
# Loading work.avip_testbench_sv_unit
# Loading work.avip_testbench
# Loading work.Chi5PC_if
# Loading work.Chi5PC
# Loading work.BaseClk
# Loading work.FlitTrace
# Loading work.SnoopTrace
# Loading work.XLA_sm
# Loading work.EXCL_mon
# do startup.do
## Node ID ==           0
# System Node Type vector  = '{RNF, RNI, RND, HNF, HNI, MN, SNF, SNI}, System Node ID vector == '{0, 1, 2, 3, 4,
5, 6, 7}
#
# ** Error: VIP_CHI5_ERR_REQ_CTL_READNOSNP:: TXREQ: Request flits with OpCode ReadNoSnp must have LikelyShared =
'b0, SnoopAttr[Snoopable] = 'b0 and SnoopAttr[SnoopDomain] = 'b0
#    Time: 1750 ns Started: 1750 ns  Scope: avip_testbench.uChi5PC.rnf.u_m_FlitTrace.chi_err_req_ctl_readnosnp
File: ./chi5pc/FlitTrace.sv Line: 1614 Expr: REQFLIT_[89]==0
# ** Note: $finish    : stim.svh(103)
#    Time: 3760 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
First release	-	-