AXI Protocol Checker v2.0

LogiCORE IP Product Guide

Vivado Design Suite

PG101 April 4, 2018

XILINX
ALL PROGRAMMABLE™

Table of Contents

Appendix B: Migrating and Upgrading

Appendix C: Additional Resources and Legal Notices

Introduction

The AXI Protocol Checker core monitors AXI interfaces. When attached to an interface, it actively checks for protocol violations and provides an indication of which violation occurred.

The checks are synthesizable versions of the System Verilog protocol assertions provided by ARM in the *AMBA®4, AXI4™, AXI4-Lite™, and AXI4-Stream™ Protocol Assertions User Guide* [Ref 2].

Features

- Supports checking for AXI3, AXI4 and AXI4-Lite protocols
- Interface data widths:
  - AXI4 and AXI3: 32, 64, 128, 256, 512 or 1024 bits
  - AXI4-Lite: 32 or 64 bits
- Address width: Up to 64 bits
- USER width: Up to 1024 bits (per channel)
- ID width: Up to 32 bits
- Programmable messaging levels for simulation operation
- Supports monitoring of multiple outstanding READ and WRITE transactions
- Instrumented to support Vivado® Design Suite Debug Nets and connections to Vivado Logic Analyzer monitoring
- AXI4-Lite control register slave interface to read protocol check status

LogiCORE IP Facts Table	
Core Specifics	
Supported Device Family(1)	UltraScale+™, UltraScale™, Xilinx® 7 series
Supported User Interfaces	AXI4, AXI4-Lite, AXI3
Resources	Performance and Resource Utilization web page
Provided with Core	
Design Files	RTL
Example Design	Not Provided
Test Bench	Not Provided
Constraints File	Not Provided
Simulation Model	Not Provided
Supported S/W Driver	N/A
Tested Design Flows(2)	
Design Entry	Vivado Design Suite
Simulation	For support simulators, see the Xilinx Design Tools: Release Notes Guide.
Synthesis	Vivado Synthesis
Support	
Provided by Xilinx at the Xilinx Support web page	

Notes:

1. For a complete list of supported devices, see the Vivado IP catalog.
2. For the supported versions of the tools, see the Xilinx Design Tools: Release Notes Guide.

Overview

The AXI Protocol Checker is used to debug interface signals in systems using AXI4, AXI3, or AXI4-Lite protocols by monitoring traffic between the AXI Master and AXI Slave Cores.

The interface is checked against the rules outlined in the AXI Specification [Ref 2] to determine if violations occur. Violations are reported in a simulation log file message, and as a debug net in the Vivado Logic Analyzer. In addition, the violations appear on the status vector output port from the core.

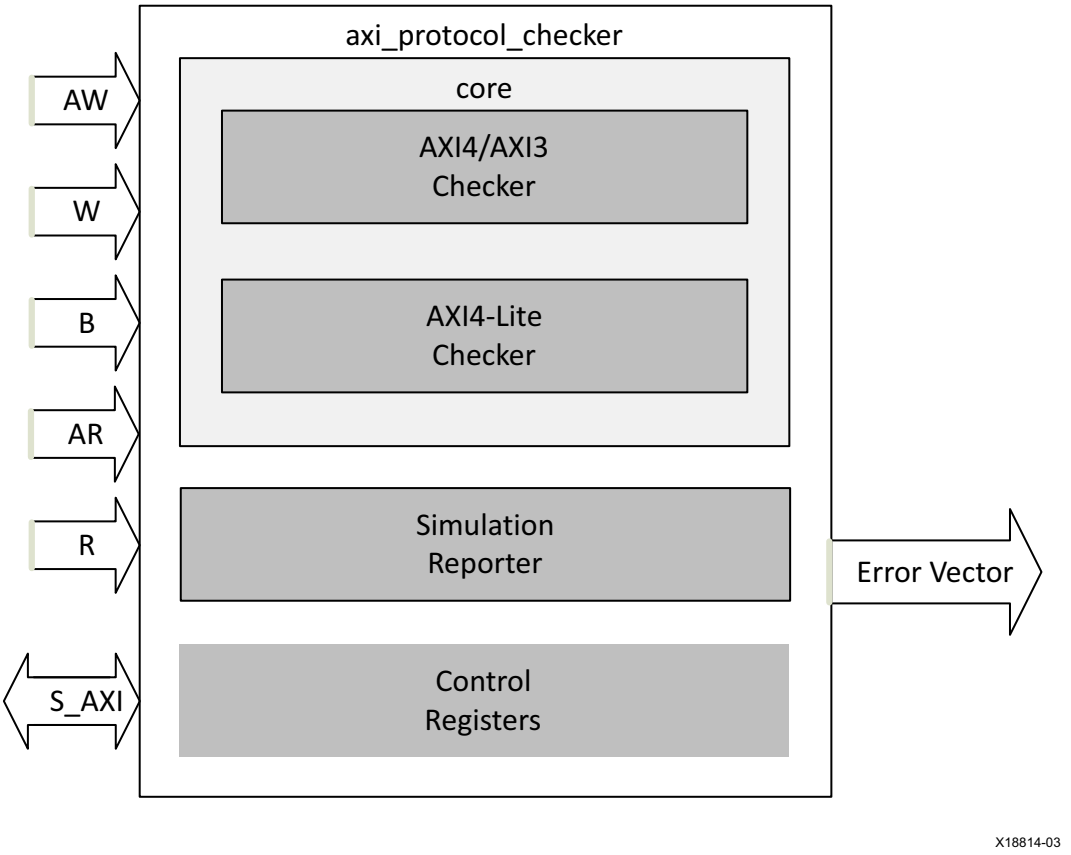

Figure 1-1: **AXI Protocol Checker Block Diagram**

Applications

The AXI Protocol Checker is typically used by system designers during the debug of systems and custom AXI IP to ensure that traffic on a given AXI connection complies with the AXI protocol.

Licensing and Ordering Information

This Xilinx LogiCORE IP module is provided at no additional cost with the Xilinx Vivado Design Suite under the terms of the Xilinx End User License.

Information about this and other Xilinx LogiCORE IP modules is available at the Xilinx Intellectual Property page. For information on pricing and availability of other Xilinx LogiCORE IP modules and tools, contact your local Xilinx sales representative.

Product Specification

The AXI Protocol Checker monitors the connection for AXI4, AXI3, and AXI4-Lite protocol violations. The AXI Protocol Checker is designed around the ARM System Verilog assertions that have been conveRREADY is Lowrted into synthesizable HDL. When a protocol violation occurs, the AXI Protocol Checker asserts the corresponding bit on the `pc_status` output vector. The output vector bit mapping can be found in Table 2-6 and Table 2-7. The value of the status vector can also be read via the optional AXI4-Lite control register slave interface.

Bits of the `pc_status` vector are synchronously set when a protocol violation occurs. Multiple bits can be triggered on the same or different cycles. When the bit within the `pc_status` vector has been set, it remains asserted until the connection has been reset with `aresetn`, or the core has been reset with `system_resetn`. The `pc_asserted` output signal is also asserted while any bit of the `pc_status` vector is asserted.

Standards

The AXI interfaces conform to the Advanced Microcontroller Bus Architecture (AMBA®) AXI version 4 specification from Advanced RISC Machine (ARM®), including the AXI4-Lite control register interface subset. See ARM AMBA® AXI Protocol v2.0 [Ref 1].

Performance

For details about performance, visit Performance and Resource Utilization.

Maximum Frequencies

For details Maximum Frequencies, visit Performance and Resource Utilization.

Resource Utilization

For details about Resource Utilization, visit Performance and Resource Utilization.

Port Descriptions

This section contains details about the AXI Protocol Checker ports.

Protocol Independent Port Descriptions

Table 2-1 lists the ports that apply to all protocols.

Table 2-1: **Protocol Independent Port Descriptions**

Signal Name	Direction	Default	Width	Description
aclk	Input	Required	1	Interface clock input. Used by both the AXI PC Monitor interface, and the optional AXI4-Lite Control register slave interface.
aresetn	Input	Required	1	Interface reset input (active-Low). Resets both the AXI PC Monitor interface and the optional AXI4-Lite Control register slave interface.
system_resetn	Input	Optional	1	System reset (active-Low).
pc_status	Output		160	Active-High vector of protocol violations or warnings.
pc_asserted	Output		1	Active-High signal is asserted when any bit of the pc_status vector is asserted.

Monitor Port Descriptions for AXI4 Protocol

Table 2-2 lists the interface signals for the AXI Protocol Checker monitor interface when it is configured to check an AXI4 Interface.

Table 2-2: **AXI4 Protocol Port Descriptions**

Signal Name	Direction	Default	Width	Description
pc_axi_awid	Input	0	ID_WIDTH	Write Address Channel Transaction ID
pc_axi_awaddr	Input	Required	ADDR_WIDTH	Write Address Channel Transaction Address (12-64)
pc_axi_awlen	Input	0	8	Write Address Channel Transaction Burst Length (0-255)
pc_axi_awsize	Input	Required	3	Write Address Channel Transfer Size Code (0-7)
pc_axi_awburst	Input	Required	2	Write Address Channel Burst Type Code (0-2)
pc_axi_awlock	Input	0b0	1	Write Address Channel Atomic Access Type (0-1)
pc_axi_awcache	Input	0b0000	4	Write Address Channel Cache Characteristics
pc_axi_awprot	Input	0b000	3	Write Address Channel Protection Characteristics
pc_axi_awqos	Input	0b0000	4	Write Address Channel Quality of Service
pc_axi_awregion	Input	0b0000	4	Write Address Channel Region Index
pc_axi_awuser	Input		AWUSER_WIDTH	Write Address Channel User-Defined Signals
pc_axi_awvalid	Input	Required	1	Write Address Channel Valid
pc_axi_awready	Input	Required	1	Write Address Channel Ready
pc_axi_arid	Input	0	ID_WIDTH	Read Address Channel Transaction ID
pc_axi_araddr	Input	Required	ADDR_WIDTH	Read Address Channel Transaction Address (12-64)
pc_axi_arlen	Input	0	8	Read Address Channel Transaction Burst Length (0-255)
pc_axi_arsize	Input	Required	3	Read Address Channel Transfer Size Code (0-7)
pc_axi_arburst	Input	Required	2	Read Address Channel Burst Type Code (0-2)
pc_axi_arlock	Input	0b0	1	Read Address Channel Atomic Access Type (0-1)
pc_axi_arcache	Input	0b0000	4	Read Address Channel Cache Characteristics
pc_axi_arprot	Input	0b000	3	Read Address Channel Protection Characteristics

Table 2-2: **AXI4 Protocol Port Descriptions *(Cont'd)***

Signal Name	Direction	Default	Width	Description
pc_axi_arqos	Input	0b0000	4	Read Address Channel Quality of Service
pc_axi_arregion	Input	0b0000	4	Read Address Channel Region Index
pc_axi_aruser	Input		ARUSER_WIDTH	Read Address Channel User-Defined Signals
pc_axi_arvalid	Input	Required	1	Read Address Channel Valid
pc_axi_arready	Input	Required	1	Read Address Channel Ready
pc_axi_wlast	Input	0b1	1	Write Data Channel Last Data Beat
pc_axi_wdata	Input		DATA_WIDTH	Write Data Channel Data
pc_axi_wstrb	Input	All Ones	DATA_WIDTH/8	Write Data Channel Byte Strobes
pc_axi_wuser	Input		WUSER_WIDTH	Write Data Channel User-Defined Signal
pc_axi_wvalid	Input	Required	1	Write Data Channel Valid
pc_axi_wready	Input	Required	1	Write Data Channel Ready
pc_axi_rid	Input		ID_WIDTH	Read Data Channel Transaction ID
pc_axi_rlast	Input	1	1	Read Data Channel Last Data Beat
pc_axi_rdata	Input		DATA_WIDTH	Read Data Channel Data
pc_axi_rresp	Input	0b00	2	Read Data Channel Response code (0-3)
pc_axi_ruser	Input		RUSER_WIDTH	Read Data Channel User-Defined Signal
pc_axi_rvalid	Input	Required	1	Read Data Channel Valid
pc_axi_rready	Input	Required	1	Read Data Channel Ready
pc_axi_bid	Input		ID_WIDTH	Write Response Channel Transaction ID
pc_axi_bresp	Input	0b00	2	Write Response Channel Response Code (0-3)
pc_axi_buser	Input		BUSER_WIDTH	Write Response Channel User-Defined Signal
pc_axi_bvalid	Input	Required	1	Write Response Channel Valid
pc_axi_bready	Input	Required	1	Write Response Channel Ready

Monitor Port Descriptions for AXI3 Protocol

Table 2-3 lists the interface signals for the AXI Protocol Checker monitor interface when it is configured to check an AXI3 Interface.

Table 2-3: **AXI3 Protocol Port Descriptions**

Signal Name	Direction	Default	Width	Description
pc_axi_awid	Input	0	ID_WIDTH	Write Address Channel Transaction ID
pc_axi_awaddr	Input	Required	ADDR_WIDTH	Write Address Channel Transaction Address (12-64)
pc_axi_awlen	Input	0	4	Write Address Channel Transaction Burst Length (0-16)
pc_axi_awsize	Input	Required	3	Write Address Channel Transfer Size code (0-7)
pc_axi_awburst	Input	Required	2	Write Address Channel Burst Type code (0-2)
pc_axi_awlock	Input	0b00	2	Write Address Channel Atomic Access Type (0-3)
pc_axi_awcache	Input	0b0000	4	Write Address Channel Cache Characteristics
pc_axi_awprot	Input	0b000	3	Write Address Channel Protection Characteristics
pc_axi_awqos	Input	0b0000	4	Write Address Channel Quality of Service
pc_axi_awuser	Input		AWUSER_WIDTH	Write Address Channel User-Defined Signals
pc_axi_awvalid	Input	Required	1	Write Address Channel Valid
pc_axi_awready	Input	Required	1	Write Address Channel Ready
pc_axi_arid	Input	0	ID_WIDTH	Read Address Channel Transaction ID
pc_axi_araddr	Input	Required	ADDR_WIDTH	Read Address Channel Transaction Address (12-64)
pc_axi_arlen	Input	0	4	Read Address Channel Transaction Burst Length (0-16)
pc_axi_arsize	Input	Required	3	Read Address Channel Transfer Size Code (0-7)
pc_axi_arburst	Input	Required	2	Read Address Channel Burst Type Code (0-2)
pc_axi_arlock	Input	0b00	2	Read Address Channel Atomic Access Type (0-3)
pc_axi_arcache	Input	0b0000	4	Read Address Channel Cache Characteristics
pc_axi_arprot	Input	0b000	3	Read Address Channel Protection Characteristics
pc_axi_arqos	Input	0b0000	4	Read Address Channel Quality of Service

Table 2-3: **AXI3 Protocol Port Descriptions *(Cont'd)***

Signal Name	Direction	Default	Width	Description
pc_axi_aruser	Input		ARUSER_WIDTH	Read Address Channel User-Defined Signals
pc_axi_arvalid	Input	Required	1	Read Address Channel Valid
pc_axi_arready	Input	Required	1	Read Address Channel Ready
pc_axi_wid	Input		ID_WIDTH	Write Data Channel Transaction ID
pc_axi_wlast	Input	0b1	1	Write Data Channel Last Data Beat
pc_axi_wdata	Input		DATA_WIDTH	Write Data Channel Data
pc_axi_wstrb	Input	All Ones	DATA_WIDTH/8	Write Data Channel Byte Strobes
pc_axi_wuser	Input		WUSER_WIDTH	Write Data Channel User-Defined Signal
pc_axi_wvalid	Input	Required	1	Write Data Channel Valid
pc_axi_wready	Input	Required	1	Write Data Channel Ready
pc_axi_rid	Input		ID_WIDTH	Read Data Channel Transaction ID
pc_axi_rlast	Input	1	1	Read Data Channel Last Data Beat
pc_axi_rdata	Input		DATA_WIDTH	Read Data Channel Data
pc_axi_rresp	Input	0b00	2	Read Data Channel Response code (0-3)
pc_axi_ruser	Input		RUSER_WIDTH	Read Data Channel User-Defined Signal
pc_axi_rvalid	Input	Required	1	Read Data Channel Valid
pc_axi_rready	Input	Required	1	Read Data Channel Ready
pc_axi_bid	Input		ID_WIDTH	Write Response Channel Transaction ID
pc_axi_bresp	Input	0b00	2	Write Response Channel Response Code (0-3)
pc_axi_buser	Input		BUSER_WIDTH	Write Response Channel User-Defined Signal
pc_axi_bvalid	Input	Required	1	Write Response Channel Valid
pc_axi_bready	Input	Required	1	Write Response Channel Ready

Monitor Port Descriptions for AXI4-Lite Protocol

Table 2-4 lists the interface signals for the AXI Protocol Checker monitor interface when it is configured to check an AXI4-Lite Interface.

Table 2-4: **AXI4-Lite Protocol Port Descriptions**

Signal Name	Direction	Default	Width	Description
pc_axi_awaddr	Input	Required	ADDR_WIDTH	Write Address Channel Transaction Address (12-64)
pc_axi_awprot	Input	0b000	3	Write Address Channel Protection Characteristics
pc_axi_awvalid	Input	Required	1	Write Address Channel Valid
pc_axi_awready	Input	Required	1	Write Address Channel Ready
pc_axi_araddr	Input	Required	ADDR_WIDTH	Read Address Channel Transaction Address (12-64)
pc_axi_arprot	Input	0b000	3	Read Address Channel Protection Characteristics
pc_axi_arvalid	Input	Required	1	Read Address Channel Valid
pc_axi_arready	Input	Required	1	Read Address Channel Ready
pc_axi_wdata	Input		DATA_WIDTH	Write Data Channel Data
pc_axi_wstrb	Input	All Ones	DATA_WIDTH/8	Write Data Channel Byte Strobes
pc_axi_wvalid	Input	Required	1	Write Data Channel Valid
pc_axi_wready	Input	Required	1	Write Data Channel Ready
pc_axi_rdata	Input		DATA_WIDTH	Read Data Channel Data
pc_axi_rresp	Input	0b00	2	Read Data Channel Response code (0-3)
pc_axi_rvalid	Input	Required	1	Read Data Channel Valid
pc_axi_rready	Input	Required	1	Read Data Channel Ready
pc_axi_bresp	Input	0b00	2	Write Response Channel Response Code (0-3)
pc_axi_bvalid	Input	Required	1	Write Response Channel Valid
pc_axi_bready	Input	Required	1	Write Response Channel Ready

Control Register Slave Port Descriptions

Table 2-5 lists the interface signals for the AXI4-Lite control register slave interface, when enabled. The control register slave interface is read-only.

Table 2-5: **Control Register Slave Port Descriptions**

Signal Name	Direction	Default	Width	Description
s_axi_araddr	Input	Required	10	Read Address
s_axi_arvalid	Input	Required	1	Read Address Channel Valid
s_axi_arready	Output	Required	1	Read Address Channel Ready
s_axi_rdata	Output		32	Read Data
s_axi_rresp	Output		2	Read Response code (always 0)
s_axi_rvalid	Output	Required	1	Read Data Channel Valid
s_axi_rready	Input	Required	1	Read Data Channel Ready

Checks and Descriptions

AXI Protocol Checks and Descriptions

The AXI Protocol Checks and descriptions, listed in Table 2-6, correspond to the assertions that are found in the ARM AXI assertions.

Table 2-6: **AXI4/AXI3 and AXI4-Lite Protocol Checks and Descriptions**

Name of Protocol Check	Bit	Notes	Protocol Support	Description
AXI_ERRM_AWADDR_BOUNDARY	0	-	AXI4/ AXI3	A write burst cannot cross a 4KB boundary.
AXI_ERRM_AWADDR_WRAP_ALIGN	1	-	AXI4/ AXI3	A write transaction with burst type WRAP has an aligned address.
AXI_ERRM_AWBURST	2	-	AXI4/ AXI3	A value of 2'b11 on AWBURST is not permitted when AWVALID is High.
AXI_ERRM_AWLEN_LOCK	3	-	AXI4/ AXI3	Exclusive access transactions cannot have a length greater than 16 beats. This check is not implemented.
AXI_ERRM_AWCACHE	4	-	AXI4/ AXI3	If not cacheable (AWCACHE[1] == 1'b0), AWCACHE = 2'b00.
AXI_ERRM_AWLEN_FIXED	5	-	AXI4/ AXI3	Transactions of burst type FIXED cannot have a length greater than 16 beats.
AXI_ERRM_AWLEN_WRAP	6	-	AXI4/ AXI3	A write transaction with burst type WRAP has a length of 2, 4, 8, or 16.
AXI_ERRM_AWSIZE	7	L(1)	AXI4/ AXI3	The size of a write transfer does not exceed the width of the data interface.
AXI_ERRM_AWVALID_RESET	8	-	AXI4/ AXI3/ Lite	AWVALID is Low for the first cycle after ARESETn goes High.
AXI_ERRM_AWADDR_STABLE	9	-	AXI4/ AXI3/ Lite	Handshake Checks: AWADDR must remain stable when AWVALID is asserted and AWREADY Low.
AXI_ERRM_AWBURST_STABLE	10	-	AXI4/ AXI3	Handshake Checks: AWBURST must remain stable when AWVALID is asserted and AWREADY Low.
AXI_ERRM_AWCACHE_STABLE	11	-	AXI4/ AXI3	Handshake Checks: AWCACHE must remain stable when AWVALID is asserted and AWREADY Low.
AXI_ERRM_AWID_STABLE	12	-	AXI4/ AXI3	Handshake Checks: AWID must remain stable when AWVALID is asserted and AWREADY Low.
AXI_ERRM_AWLEN_STABLE	13	-	AXI4/ AXI3	Handshake Checks: AWLEN must remain stable when AWVALID is asserted and AWREADY Low.

Table 2-6: **AXI4/AXI3 and AXI4-Lite Protocol Checks and Descriptions *(Cont'd)***

Name of Protocol Check	Bit	Notes	Protocol Support	Description
AXI_ERRM_AWLOCK_STABLE	14	-	AXI4/ AXI3	Handshake Checks: AWLOCK must remain stable when AWVALID is asserted and AWREADY Low. This check is not implemented.
AXI_ERRM_AWPROT_STABLE	15	-	AXI4/ AXI3/ Lite	Handshake Checks: AWPROT must remain stable when AWVALID is asserted and AWREADY Low.
AXI_ERRM_AWSIZE_STABLE	16	-	AXI4/ AXI3	Handshake Checks: AWSIZE must remain stable when AWVALID is asserted and AWREADY Low.
AXI_ERRM_AWQOS_STABLE	17	-	AXI4/ AXI3	Handshake Checks: AWQOS must remain stable when AWVALID is asserted and AWREADY Low.
AXI_ERRM_AWREGION_STABLE	18	-	AXI4	Handshake Checks: AWREGION must remain stable when ARVALID is asserted and AWREADY Low.
AXI_ERRM_AWVALID_STABLE	19	-	AXI4/ AXI3/ Lite	Handshake Checks: Once AWVALID is asserted, it must remain asserted until AWREADY is High.
AXI_RECS_AWREADY_MAX_WAIT	20	L[(1)]	AXI4/ AXI3/ Lite	Recommended that AWREADY is asserted within MAXWAITS cycles of AWVALID being asserted.
AXI_ERRM_WDATA_NUM	21	L[(1)]	AXI4/ AXI3	The number of write data items matches AWLEN for the corresponding address. This is triggered when any of the following occurs: • Write data arrives, WLAST is set, and the WDATA count is not equal to AWLEN • Write data arrives, 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
AXI_ERRM_WSTRB	22	-	AXI4/ AXI3/ Lite	Write strobes must only be asserted for the correct byte lanes as determined from the: Start Address, Transfer Size and Beat Number.
AXI_ERRM_WVALID_RESET	23	-	AXI4/ AXI3/ Lite	WVALID is LOW for the first cycle after ARESETn goes High.
AXI_ERRM_WDATA_STABLE	24	-	AXI4/ AXI3/ Lite	Handshake Checks: WDATA must remain stable when WVALID is asserted and WREADY Low.
AXI_ERRM_WLAST_STABLE	25	-	AXI4/ AXI3	Handshake Checks: WLAST must remain stable when WVALID is asserted and WREADY Low.
AXI_ERRM_WSTRB_STABLE	26	-	AXI4/ AXI3/ Lite	Handshake Checks: WSTRB must remain stable when WVALID is asserted and WREADY Low.

Table 2-6: **AXI4/AXI3 and AXI4-Lite Protocol Checks and Descriptions *(Cont'd)***

Name of Protocol Check	Bit	Notes	Protocol Support	Description
AXI_ERRM_WVALID_STABLE	27	-	AXI4/ AXI3/ Lite	Handshake Checks: Once WVALID is asserted, it must remain asserted until WREADY is High.
AXI_RECS_WREADY_MAX_WAIT	28	L(1)	AXI4/ AXI3/ Lite	Recommended that WREADY is asserted within MAXWAITS cycles of WVALID being asserted.
AXI_ERRS_BRESP_EXOKAY	30	-	AXI4/ AXI3	An EXOKAY write response can only be given to an exclusive write access. This check is not implemented.
AXI_ERRS_BVALID_RESET	31	-	AXI4/ AXI3/ Lite	BVALID is Low for the first cycle after ARESETn goes High.
AXI_ERRS_BRESP_AW Or AXI_ERRS_BRESP_WLAST	32	L(1)	AXI4/ AXI3/ Lite	A slave must not take BVALID HIGH until after the write address handshake is complete. Or A slave must not take BVALID HIGH until after the last write data handshake is complete. **Note:** The IP does not distinguish between these two causes.
AXI_ERRS_BID_STABLE	33	-	AXI4/ AXI3	Handshake Checks: BID must remain stable when BVALID is asserted and BREADY Low.
AXI_ERRS_BRESP_STABLE	34	-	AXI4/ AXI3/ Lite	Checks BRESP must remain stable when BVALID is asserted and BREADY Low.
AXI_ERRS_BVALID_STABLE	35	-	AXI4/ AXI3/ Lite	Once BVALID is asserted, it must remain asserted until BREADY is High.
AXI_RECM_BREADY_MAX_WAIT	36	L(1)	AXI4/ AXI3/ Lite	Recommended that BREADY is asserted within MAXWAITS cycles of BVALID being asserted.
AXI_ERRM_ARADDR_BOUNDARY	37	-	AXI4/ AXI3	A read burst cannot cross a 4KB boundary.
AXI_ERRM_ARADDR_WRAP_ALIGN	38	-	AXI4/ AXI3	A read transaction with a burst type of WRAP must have an aligned address.
AXI_ERRM_ARBURST	39	-	AXI4/ AXI3	A value of 2'b11 on ARBURST is not permitted when ARVALID is High.
AXI_ERRM_ARLEN_LOCK	40	-	AXI4/ AXI3	Exclusive access transactions cannot have a length greater than 16 beats. This check is not implemented.
AXI_ERRM_ARCACHE	41	-	AXI4/ AXI3	When ARVALID is HIGH, if ARCACHE[1] is LOW, then ARCACHE[3:2] must also be Low.
AXI_ERRM_ARLEN_FIXED	42	-	AXI4/ AXI3	Transactions of burst type FIXED cannot have a length greater than 16 beats.
AXI_ERRM_ARLEN_WRAP	43	-	AXI4/ AXI3	A read transaction with burst type of WRAP must have a length of 2, 4, 8, or 16.

Table 2-6: **AXI4/AXI3 and AXI4-Lite Protocol Checks and Descriptions *(Cont'd)***

Name of Protocol Check	Bit	Notes	Protocol Support	Description
AXI_ERRM_ARSIZE	44	L[1]	AXI4/ AXI3	The size of a read transfer must not exceed the width of the data interface.
AXI_ERRM_ARVALID_RESET	45	-	AXI4/ AXI3/ Lite	ARVALID is Low for the first cycle after ARESETn goes High.
AXI_ERRM_ARADDR_STABLE	46	-	AXI4/ AXI3/ Lite	ARADDR must remain stable when ARVALID is asserted and ARREADY Low.
AXI_ERRM_ARBURST_STABLE	47	-	AXI4/ AXI3	ARBURST must remain stable when ARVALID is asserted and ARREADY Low.
AXI_ERRM_ARCACHE_STABLE	48	-	AXI4/ AXI3	ARCACHE must remain stable when ARVALID is asserted and ARREADY Low.
AXI_ERRM_ARID_STABLE	49	-	AXI4/ AXI3	ARID must remain stable when ARVALID is asserted and ARREADY Low.
AXI_ERRM_ARLEN_STABLE	50	-	AXI4/ AXI3	ARLEN must remain stable when ARVALID is asserted and ARREADY Low.
AXI_ERRM_ARLOCK_STABLE	51	-	AXI4/ AXI3	ARLOCK must remain stable when ARVALID is asserted and ARREADY Low. This check is not implemented.
AXI_ERRM_ARPROT_STABLE	52	-	AXI4/ AXI3/ Lite	ARPROT must remain stable when ARVALID is asserted and ARREADY Low.
AXI_ERRM_ARSIZE_STABLE	53	-	AXI4/ AXI3	ARSIZE must remain stable when ARVALID is asserted and ARREADY Low.
AXI_ERRM_ARQOS_STABLE	54	-	AXI4/ AXI3	ARQOS must remain stable when ARVALID is asserted and ARREADY Low.
AXI_ERRM_ARREGION_STABLE	55	-	AXI4	ARREGION must remain stable when ARVALID is asserted and ARREADY Low.
AXI_ERRM_ARVALID_STABLE	56	-	AXI4/ AXI3/ Lite	Once ARVALID is asserted, it must remain asserted until ARREADY is High.
AXI_RECS_ARREADY_MAX_WAIT	57	L[1]	AXI4/ AXI3/ Lite	Recommended that ARREADY is asserted within MAXWAITS cycles of ARVALID being asserted.
AXI_ERRS_RDATA_NUM	58	L[1]	AXI4/ AXI3	The number of read data items must match the corresponding ARLEN.
AXI_ERRS_RID	59	L[1]	AXI4/ AXI3/ Lite	The read data must always follow the address that it relates to. If IDs are used, RID must also match ARID of an outstanding address read transaction. This violation can also occur when RVALID is asserted with no preceding AR transfer.
AXI_ERRS_RRESP_EXOKAY	60	-	AXI4/ AXI3	An EXOKAY write response can only be given to an exclusive read access. This check is not implemented.

Table 2-6: **AXI4/AXI3 and AXI4-Lite Protocol Checks and Descriptions *(Cont'd)***

Name of Protocol Check	Bit	Notes	Protocol Support	Description
AXI_ERRS_RVALID_RESET	61	-	AXI4/ AXI3/ Lite	RVALID is Low for the first cycle after ARESETn goes High.
AXI_ERRS_RDATA_STABLE	62	-	AXI4/ AXI3/ Lite	RDATA must remain stable when RVALID is asserted and RREADY Low.
AXI_ERRS_RID_STABLE	63	-	AXI4/ AXI3	RID must remain stable when RVALID is asserted and RREADY Low.
AXI_ERRS_RLAST_STABLE	64	-	AXI4/ AXI3	RLAST must remain stable when RVALID is asserted and RREADY Low.
AXI_ERRS_RRESP_STABLE	65	-	AXI4/ AXI3/ Lite	RRESP must remain stable when RVALID is asserted and RREADY Low.
AXI_ERRS_RVALID_STABLE	66	-	AXI4/ AXI3/ Lite	Once RVALID is asserted, it must remain asserted until RREADY is High.
AXI_RECM_RREADY_MAX_WAIT	67	L(1)	AXI4/ AXI3/ Lite	Recommended that RREADY is asserted within MAXWAITS cycles of RVALID being asserted.
AXI_ERRM_EXCL_ALIGN	68	-	AXI4/ AXI3	The address of an exclusive access is aligned to the total number of bytes in the transaction. This check is not implemented.
AXI_ERRM_EXCL_LEN	69	-	AXI4/ AXI3	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. This check is not implemented.
AXI_RECM_EXCL_MATCH	70	-	AXI4/ AXI3	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. This check is not implemented.
AXI_ERRM_EXCL_MAX	71	-	AXI4/ AXI3	128 is the maximum number of bytes that can be transferred in an exclusive burst. This check is not implemented.
AXI_RECM_EXCL_PAIR	72	-	AXI4/ AXI3	Recommended that every exclusive write has an earlier outstanding exclusive read with the same ID. This check is not implemented.
AXI_ERRM_AWUSER_STABLE	73	-	AXI4/ AXI3	AWUSER must remain stable when AWVALID is asserted and AWREADY Low.
AXI_ERRM_WUSER_STABLE	74	-	AXI4/ AXI3	WUSER must remain stable when WVALID is asserted and WREADY Low.
AXI_ERRS_BUSER_STABLE	75	-	AXI4/ AXI3	BUSER must remain stable when BVALID is asserted and BREADY Low.
AXI_ERRM_ARUSER_STABLE	76	-	AXI4/ AXI3	ARUSER must remain stable when ARVALID is asserted and ARREADY Low.

Table 2-6: **AXI4/AXI3 and AXI4-Lite Protocol Checks and Descriptions *(Cont'd)***

Name of Protocol Check	Bit	Notes	Protocol Support	Description
AXI_ERRS_RUSER_STABLE	77	-	AXI4/ AXI3	RUSER must remain stable when RVALID is asserted and RREADY Low.
AXI_AUXM_RCAM_OVERFLOW	78	L(1)	AXI4/ AXI3/ Lite	Read CAM overflow. The number of outstanding read transactions exceeds the capacity of the storage implemented in the Protocol Checker core. This does not indicate any failure of the system being monitored, but it may render any subsequent monitoring results invalid. To resolve this, increase MAXRBURSTS parameter.
AXI_AUXM_RCAM_UNDERFLOW	79	-	AXI4/ AXI3/ Lite	Read CAM underflow.
AXI_AUXM_WCAM_OVERFLOW	80	L(1)	AXI4/ AXI3/ Lite	Write CAM overflow. The number of outstanding write transactions exceeds the capacity of the storage implemented in the Protocol Checker core. This does not indicate any failure of the system being monitored, but it may render any subsequent monitoring results invalid. To resolve this, increase MAXWBURSTS parameter.
AXI_AUXM_WCAM_UNDERFLOW	81	-	AXI4/ AXI3/ Lite	Write CAM underflow.
AXI_AUXM_EXCL_OVERFLOW	82	-	AXI4/ AXI3	Exclusive access monitor overflow.
AXI4LITE_ERRS_BRESP_EXOKAY	83	-	Lite	A slave must not give an EXOKAY response on an AXI4-Lite interface.
AXI4LITE_ERRS_RRESP_EXOKAY	84	-	Lite	A slave must not give an EXOKAY response on an AXI4-Lite interface.
AXI4LITE_AUXM_DATA_WIDTH	85	-	Lite	DATA_WIDTH parameter is 32 or 64.

Notes:

1. L: This check remains enabled in Lightweight mode.

Xilinx-Specific Configuration Checks and Descriptions

Table 2-7 lists Xilinx-specific checks for configuration-dependent signaling requirements and recommendations.

Table 2-7: **Xilinx Configuration Checks and Descriptions**

Name of Protocol Check	Bit	Notes	Protocol Support	Description
XILINX_AW_SUPPORTS_NARROW_BURST	86	-	AXI4/ AXI3	When the connection does not support narrow transfers, the AW Master cannot issue a transfer with AWLEN > 0 and AWSIZE is less than the defined interface DATA_WIDTH.
XILINX_AR_SUPPORTS_NARROW_BURST	87	-	AXI4/ AXI3	When the connection does not support narrow transfers, the AR Master cannot issue a transfer with ARLEN > 0 and ARSIZE is less than the defined interface DATA_WIDTH.
XILINX_AW_SUPPORTS_NARROW_CACHE	88	-	AXI4/ AXI3	When the connection does not support narrow transfers, the AW Master cannot issue a transfer with AWLEN > 0 and AWCACHE modifiable bit is not asserted.
XILINX_AR_SUPPORTS_NARROW_CACHE	89	-	AXI4/ AXI3	When the connection does not support narrow transfers, the AR Master cannot issue a transfer with ARLEN > 0 and ARCACHE modifiable bit is not asserted.
XILINX_AW_MAX_BURST	90	-	AXI4/ AXI3	AW Master cannot issue AWLEN greater than the configured maximum burst length.
XILINX_AR_MAX_BURST	91	-	AXI4/ AXI3	AR Master cannot issue ARLEN greater than the configured maximum burst length.
XILINX_AWREADY_RESET	92	-	AXI4/ AXI3/ Lite	AWREADY is Low for the first cycle after ARESETn goes High. Xilinx recommends that slaves drive all READY outputs low during reset to avoid dropping a transfer in case the connected master recovers from reset during an earlier cycle.
XILINX_WREADY_RESET	93	-	AXI4/ AXI3/ Lite	WREADY is Low for the first cycle after ARESETn goes High. Xilinx recommends that slaves drive all READY outputs low during reset to avoid dropping a transfer in case the connected master recovers from reset during an earlier cycle.

Table 2-7: **Xilinx Configuration Checks and Descriptions *(Cont'd)***

Name of Protocol Check	Bit	Notes	Protocol Support	Description
XILINX_BREADY_RESET	94	-	AXI4/ AXI3/ Lite	BREADY is Low for the first cycle after ARESETn goes High. Xilinx recommends that masters drive all READY outputs low during reset.
XILINX_ARREADY_RESET	95	-	AXI4/ AXI3/ Lite	ARREADY is Low for the first cycle after ARESETn goes High. Xilinx recommends that slaves drive all READY outputs low during reset to avoid dropping a transfer in case the connected master recovers from reset during an earlier cycle.
XILINX_RREADY_RESET	96	-	AXI4/ AXI3/ Lite	RREADY is Low for the first cycle after ARESETn goes High. Xilinx recommends that masters drive all READY outputs low during reset.
XILINX_RECS_CONTINUOUS_RTRANSFERS_MAX_WAIT	97	L (1)	AXI4/ AXI3/ Lite	RVALID should be asserted within MAX_CONTINUOUS_RTRANSFERS_WAITS cycles of either the AR command transfer or the previous R transfer while there are outstanding AR commands.
XILINX_RECM_CONTINUOUS_WTRANSFERS_MAX_WAIT	98	L (1)	AXI4/ AXI3/ Lite	WVALID should be asserted within MAX_CONTINUOUS_WTRANSFERS_WAITS cycles of either the AW command transfer or previous W transfer while there are outstanding AW commands.
XILINX_RECM_WLAST_TO_AWVALID_MAX_WAIT	99	L (1)	AXI4/ AXI3/ Lite	AWVALID should be asserted within MAX_WLAST_TO_AWVALID_WAITS cycles of a WLAST transfer (or AXI4-Lite W transfer) or previous AW transfer if there are yet more WLAST transfers outstanding.
XILINX_RECS_WRITE_TO_BVALID_MAX_WAIT	100	L (1)	AXI4/ AXI3/ Lite	BVALID should be asserted within MAX_WRITE_TO_BVALID_WAITS cycles of an AW command transfer or WLAST transfer (or AXI4-Lite W transfer), whichever is later, or previous B transfer if there are yet more completed AW and WLAST transfers outstanding.
XILINX_ARESETN_PULSE_WIDTH	101		AXI4/ AXI3/ Lite	ARESETn must be low for at least 16 ACLKn cycles.

Table 2-7: **Xilinx Configuration Checks and Descriptions *(Cont'd)***

Name of Protocol Check	Bit	Notes	Protocol Support	Description
XILINX_AXI4_ERRM_NO_STRB_ADDRESS	102	L (1)	AXI4/ AXI3/ Lite	When there is no WSTRB, address must be aligned with the data width
XILINX_AXI4_ERRM_SUPPORTS_NARROW_BURST_SIZE	103	L (1)	AXI4/ AXI3/ Lite	When there is support narrow burst along with WSTRB, then AWSIZE has to be greater than or equal to data width
XILINX_AXI4_RD_RESP_SLVERR	104	-	AXI4/ AXI3/ Lite	When CHK_ERR_RESP is enabled, this check detects the SLVERR on Read Response
XILINX_AXI4_RD_RESP_DECERR	105	-	AXI4/ AXI3/ Lite	When CHK_ERR_RESP is enabled, this check detects the DECERR on Read Response
XILINX_AXI4_WR_RESP_SLVERR	106	-	AXI4/ AXI3/ Lite	When CHK_ERR_RESP is enabled, this check detects the SLVERR on Write Response channel
XILINX_AXI4_WR_RESP_DECERR	107	-	AXI4/ AXI3/ Lite	When CHK_ERR_RESP is enabled, this check detects the DECERR on Write Response channel

Notes:

1. L: This check remains enabled in Lightweight mode.

Simulation-Only Assertions

Table 2-8 lists signal checks required by AXI protocol that are checked only during simulation. These checks appear only as assertion messages and do not appear in the `pc_status` output signal.

Table 2-8: **Simulation-Only Assertions**

Name of Protocol Check	Protocol Support	Description
AXI_ERRM_ARADDR_X	AXI4/AXI3/Lite	When ARVALID is high, a value of X on ARADDR is not permitted.
AXI_ERRM_ARBURST_X	AXI4/AXI3	When ARVALID is high, a value of X on ARBURST is not permitted.
AXI_ERRM_ARCACHE_X	AXI4/AXI3	When ARVALID is high, a value of X on ARCACHE is not permitted.
AXI_ERRM_ARID_X	AXI4/AXI3	When ARVALID is high, a value of X on ARID is not permitted.
AXI_ERRM_ARLEN_X	AXI4/AXI3	When ARVALID is high, a value of X on ARLEN is not permitted.
AXI_ERRM_ARLOCK_X	AXI4/AXI3	When ARVALID is high, a value of X on ARLOCK is not permitted.

Table 2-8: **Simulation-Only Assertions *(Cont'd)***

Name of Protocol Check	Protocol Support	Description
AXI_ERRM_ARPROT_X	AXI4/AXI3/Lite	When ARVALID is high, a value of X on ARPROT is not permitted.
AXI_ERRM_ARQOS_X	AXI4/AXI3	When ARVALID is high, a value of X on ARQOS is not permitted.
AXI_ERRM_ARREGION_X	AXI4	When ARVALID is high, a value of X on ARREGION is not permitted.
AXI_ERRM_ARSIZE_X	AXI4/AXI3	When ARVALID is high, a value of X on ARSIZE is not permitted.
AXI_ERRM_ARUSER_X	AXI4/AXI3	When ARVALID is high, a value of X on ARUSER is not permitted.
AXI_ERRM_ARVALID_X	AXI4/AXI3/Lite	When not in reset, a value of X on ARVALID is not permitted.
AXI_ERRM_AWADDR_X	AXI4/AXI3/Lite	When AWVALID is high, a value of X on AWADDR is not permitted.
AXI_ERRM_AWBURST_X	AXI4/AXI3	When AWVALID is high, a value of X on AWBURST is not permitted.
AXI_ERRM_AWCACHE_X	AXI4/AXI3	When AWVALID is high, a value of X on AWCACHE is not permitted.
AXI_ERRM_AWID_X	AXI4/AXI3	When AWVALID is high, a value of X on AWID is not permitted.
AXI_ERRM_AWLEN_X	AXI4/AXI3	When AWVALID is high, a value of X on AWLEN is not permitted.
AXI_ERRM_AWLOCK_X	AXI4/AXI3	When AWVALID is high, a value of X on AWLOCK is not permitted.
AXI_ERRM_AWPROT_X	AXI4/AXI3/Lite	When AWVALID is high, a value of X on AWPROT is not permitted.
AXI_ERRM_AWQOS_X	AXI4/AXI3	When AWVALID is high, a value of X on AWQOS is not permitted.
AXI_ERRM_AWREGION_X	AXI4	When AWVALID is high, a value of X on AWREGION is not permitted.
AXI_ERRM_AWSIZE_X	AXI4/AXI3	When AWVALID is high, a value of X on AWSIZE is not permitted.
AXI_ERRM_AWUSER_X	AXI4/AXI3	When AWVALID is high, a value of X on AWUSER is not permitted.
AXI_ERRM_AWVALID_X	AXI4/AXI3/Lite	When not in reset, a value of X on AWVALID is not permitted.
AXI_ERRM_BREADY_X	AXI4/AXI3/Lite	When not in reset, a value of X on BREADY is not permitted.
AXI_ERRM_RREADY_X	AXI4/AXI3/Lite	When not in reset, a value of X on RREADY is not permitted.
AXI_ERRM_WDATA_X	AXI4/AXI3/Lite	When WVALID is high, a value of X on active byte lanes of WDATA is not permitted.
AXI_ERRM_WLAST_X	AXI4/AXI3	When WVALID is high, a value of X on WLAST is not permitted.

Table 2-8: **Simulation-Only Assertions *(Cont'd)***

Name of Protocol Check	Protocol Support	Description
AXI_ERRM_WSTRB_X	AXI4/AXI3/Lite	When WVALID is high, a value of X on WSTRB is not permitted.
AXI_ERRM_WUSER_X	AXI4/AXI3	When WVALID is high, a value of X on WUSER is not permitted.
AXI_ERRM_WVALID_X	AXI4/AXI3/Lite	When not in reset, a value of X on WVALID is not permitted.
AXI_ERRS_ARREADY_X	AXI4/AXI3/Lite	When not in reset, a value of X on ARREADY is not permitted.
AXI_ERRS_AWREADY_X	AXI4/AXI3/Lite	When not in reset, a value of X on AWREADY is not permitted.
AXI_ERRS_BID_X	AXI4/AXI3	When BVALID is high, a value of X on BID is not permitted.
AXI_ERRS_BRESP_X	AXI4/AXI3/Lite	When BVALID is high, a value of X on BRESP is not permitted.
AXI_ERRS_BUSER_X	AXI4/AXI3	When BVALID is high, a value of X on BUSER is not permitted.
AXI_ERRS_BVALID_X	AXI4/AXI3/Lite	When not in reset, a value of X on BVALID is not permitted.
AXI_ERRS_RDATA_X	AXI4/AXI3/Lite	When RVALID is high, a value of X on RDATA valid byte lanes is not permitted.
AXI_ERRS_RID_X	AXI4/AXI3	When RVALID is high, a value of X on RID is not permitted.
AXI_ERRS_RLAST_X	AXI4/AXI3	When RVALID is high, a value of X on RLAST is not permitted.
AXI_ERRS_RRESP_X	AXI4/AXI3/Lite	When RVALID is high, a value of X on RRESP is not permitted.
AXI_ERRS_RUSER_X	AXI4/AXI3	When RVALID is high, a value of X on RUSER is not permitted.
AXI_ERRS_RVALID_X	AXI4/AXI3/Lite	When not in reset, a value of X on RVALID is not permitted.
AXI_ERRS_WREADY_X	AXI4/AXI3/Lite	When not in reset, a value of X on WREADY is not permitted.

Register Space

The registers listed in Table 2-9 can be read from the AXI4-Lite control register slave interface.

The *Current PC* registers indicate the current state of the `pc_status` vector, which may accumulate more checks over time.

The *Snapshot PC* registers indicate the state of the `pc_status` vector at the time of the first check assertion (when `pc_asserted` first becomes asserted). The Snapshot registers are reset by either `aresetn` or `system_resetn`.

Table 2-9: **Control Register Map**

Address Space Offset	Register Name	Access Type	Default Value	Description
0x000	PC asserted	R	0	Bit 0 = Current value of pc_asserted
0x100	Current PC[31:0]	R	0	Current value of pc_status[31:0]
0x104	Current PC[63:32]	R	0	Current value of pc_status[63:32]
0x108	Current PC[95:64]	R	0	Current value of pc_status[95:64]
0x10C	Current PC[127:96]	R	0	Current value of pc_status[127:96]
0x200	Snapshot PC[31:0]	R	0	First event value of pc_status[31:0]
0x204	Snapshot PC[63:32]	R	0	First event value of pc_status[63:32]
0x208	Snapshot PC[95:64]	R	0	First event value of pc_status[95:64]
0x20C	Snapshot PC[127:96]	R	0	First event value of pc_status[127:96]

Designing with the Core

This chapter includes guidelines and additional information to facilitate designing with the core.

General Design Guidelines

The AXI Protocol Checker should be inserted into a system as shown in Figure 3-1.

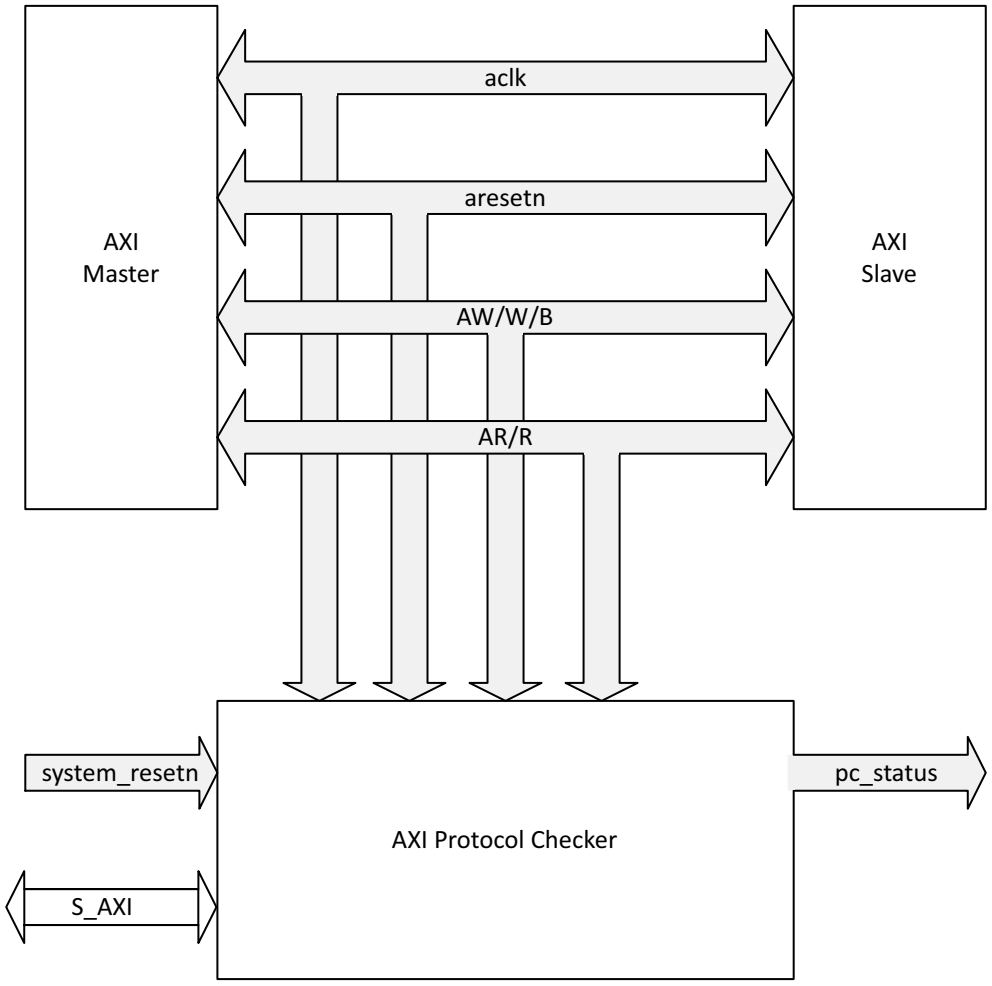

Figure 3-1: **Example Topology**

Finding protocol violations in simulation is generally easier. Therefore, it is highly recommended to debug a system in simulation rather than in hardware. In simulation, it is possible to instantiate protocol checkers on all AXI interfaces.

Simulation

When simulating, the AXI Protocol Checker can be configured to print display messages in the form of:

```
<time>ns : <instance_path> : BIT( <bit_number>) : <log_level> : <violation_message>
```

Table 3-1: **Message Field Descriptions**

Message Field	Description
<instance_path>	The simulation hierarchy to the protocol checker instance that is issuing the message.
<bit_number>	Indicates which bit is asserted in the pc_status vector. This bit can be used to look up the violation in Table 2-6 and Table 2-7.
<log_level>	Messaging level text that can be one of the following values: INFO, WARNING, or ERROR with the ability to stop or finish the simulation. This value is set in the configuration Vivado IDE.
<violation_message>	Descriptive message indicating the name of the violation (for example, AXI_ERRS_RID) and violation. In most cases, this includes the location in the AXI specification where the protocol is described. The violation message name can be used to look up the violation in Table 2-6 and Table 2-7.

For example:

```
73717.00ns : tb.top.AXI4_0.REP : BIT(59) : ERROR : AXI_ERRS_RID. A slave can only
give read data with an ID to match an outstanding read transaction. Spec: section 8.3
on page 8-4.
```

In this example, the core provides a debugging capability for simulation where the AXI Protocol Checker can either finish or stop the simulation at the point of detecting the violation.

Hardware In-system Monitoring

The AXI Protocol Checker tracks Read and Write transactions by their ARID and AWID respectively. Therefore, the wider the ID width, the more resources consumed by the Protocol Checker. The transaction information per ID is stored until the transactions are completed, and the depth of the memory is directly related to the number of outstanding transactions that can occur for a given transaction ID.

IMPORTANT: *Configure the core to ensure the core is not over-monitoring and wasting resources. An ID width greater than 6 bits can lead to very large resource utilization by the core.*

Integration into Vivado Logic Analyzer and Vivado Debug Nets

The AXI Protocol Checker core supports probing using the Vivado ILA 2.0 core and the Vivado Logic Analyzer. All of the protocol checks listed in Table 2-6 and Table 2-7 are available as Unassigned Debug Nets in the synthesized design. See *Vivado Design Suite Tutorial: Programming and Debugging* (UG936) [Ref 5].

Clocking

The same `aclk` that is connected to the AXI Master and AXI Slave should also be connected to the AXI Protocol Checker.

The `aclk` input is also used by the optional AXI4-Lite control register slave interface. By connecting to the control interface through an AXI Interconnect IP, the Interconnect IP will automatically convert from the clock domain of the AXI4-Lite network, if different.

Resets

System Reset and AXI Reset

At a minimum, the AXI Protocol Checker requires the `aresetn` signal. `system_resetn` can be configured to clear the `pc_status` vector without resetting the AXI4 interface. Both reset inputs are synchronous to `aclk`. The assertion of either reset clears the `pc_status` vector and the `pc_asserted` output.

TIP: *Xilinx recommends asserting `aresetn` for a minimum of 16 clock cycles.*

If `system_resetn` is enabled, the Protocol Checker can check the AXI4 specification which defines the state of the interface following the de-assertion of `aresetn`. Protocol violation notifications related to the required behavior of interfaces with respect to `aresetn` are cleared using `system_resetn`.

When a system reset is not available, `system_resetn` should be disabled. When this is done, the following checks are not possible:

- AXI_ERRM_AWVALID_RESET
- AXI_ERRM_ARVALID_RESET
- AXI_ERRM_WVALID_RESET
- AXI_ERRM_RVALID_RESET
- AXI_ERRM_BVALID_RESET

- XILINX_AWREADY_RESET
- XILINX_WREADY_RESET
- XILINX_BREADY_RESET
- XILINX_ARREADY_RESET
- XILINX_RREADY_RESET

Design Flow Steps

This chapter describes customizing and generating the core, constraining the core, and the simulation, synthesis and implementation steps that are specific to this IP core. More detailed information about the standard Vivado® design flows and the IP integrator can be found in the following Vivado Design Suite user guides:

- *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994) [Ref 3]
- *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 6]
- *Vivado Design Suite User Guide: Getting Started* (UG910) [Ref 8]
- *Vivado Design Suite User Guide: Logic Simulation* (UG900) [Ref 7]

Customizing and Generating the Core

This section includes information about using Xilinx tools to customize and generate the core in the Vivado Design Suite environment. You can customize the IP for use in your design by specifying values for the various parameters associated with the IP core using the following steps:

1. Select the IP from the IP catalog.
2. Double-click the selected IP or select the Customize IP command from the toolbar or right-click menu.

For details, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 6] and the *Vivado Design Suite User Guide: Getting Started* (UG910) [Ref 8].

Note: Figures in this chapter are illustrations of the Vivado Integrated Design Environment (IDE). This layout might vary from the current version.

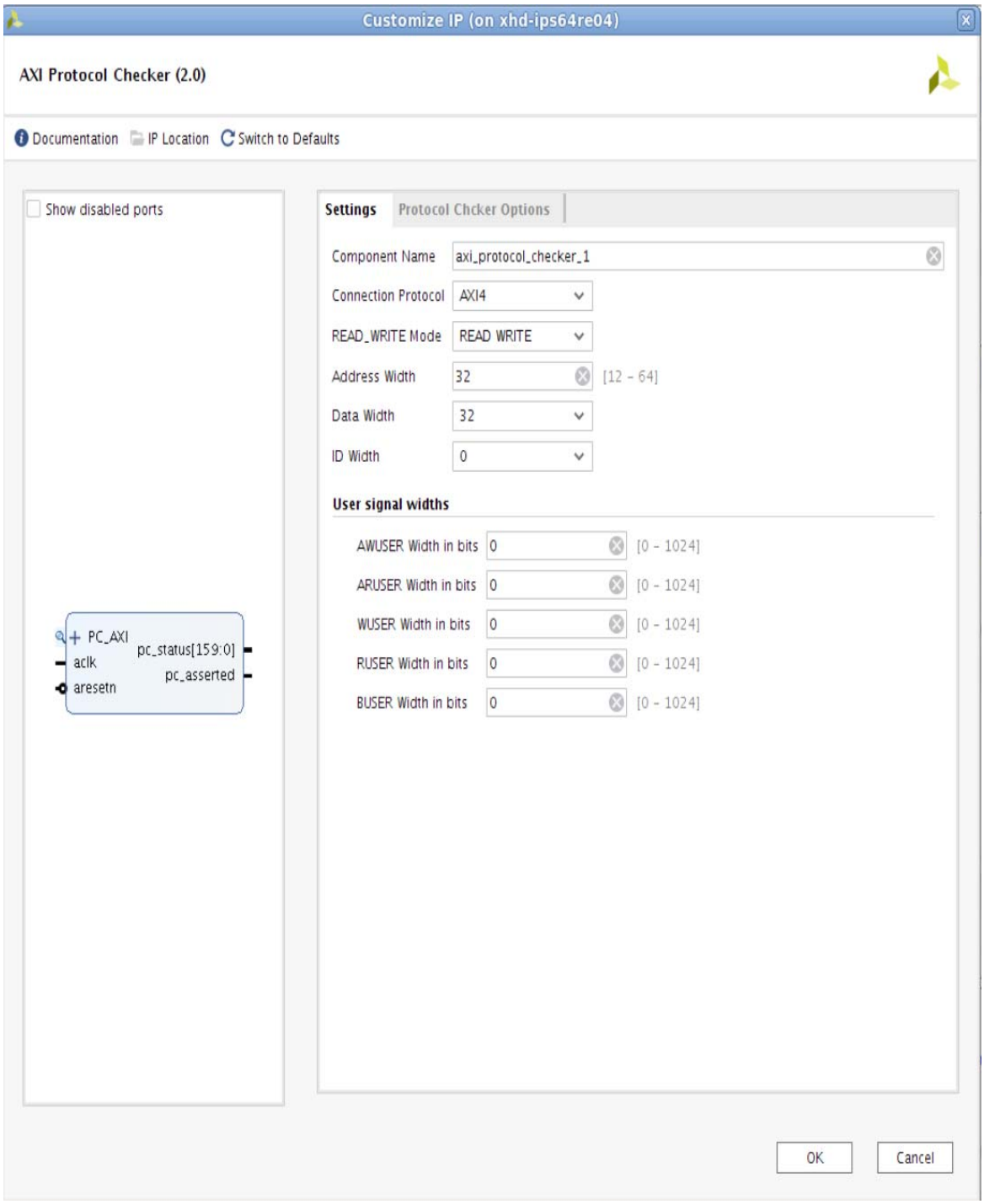

Figure 4-1: **AXI Protocol Checker Settings**

Modify the parameters to meet the requirements of the larger project into which the core is integrated.

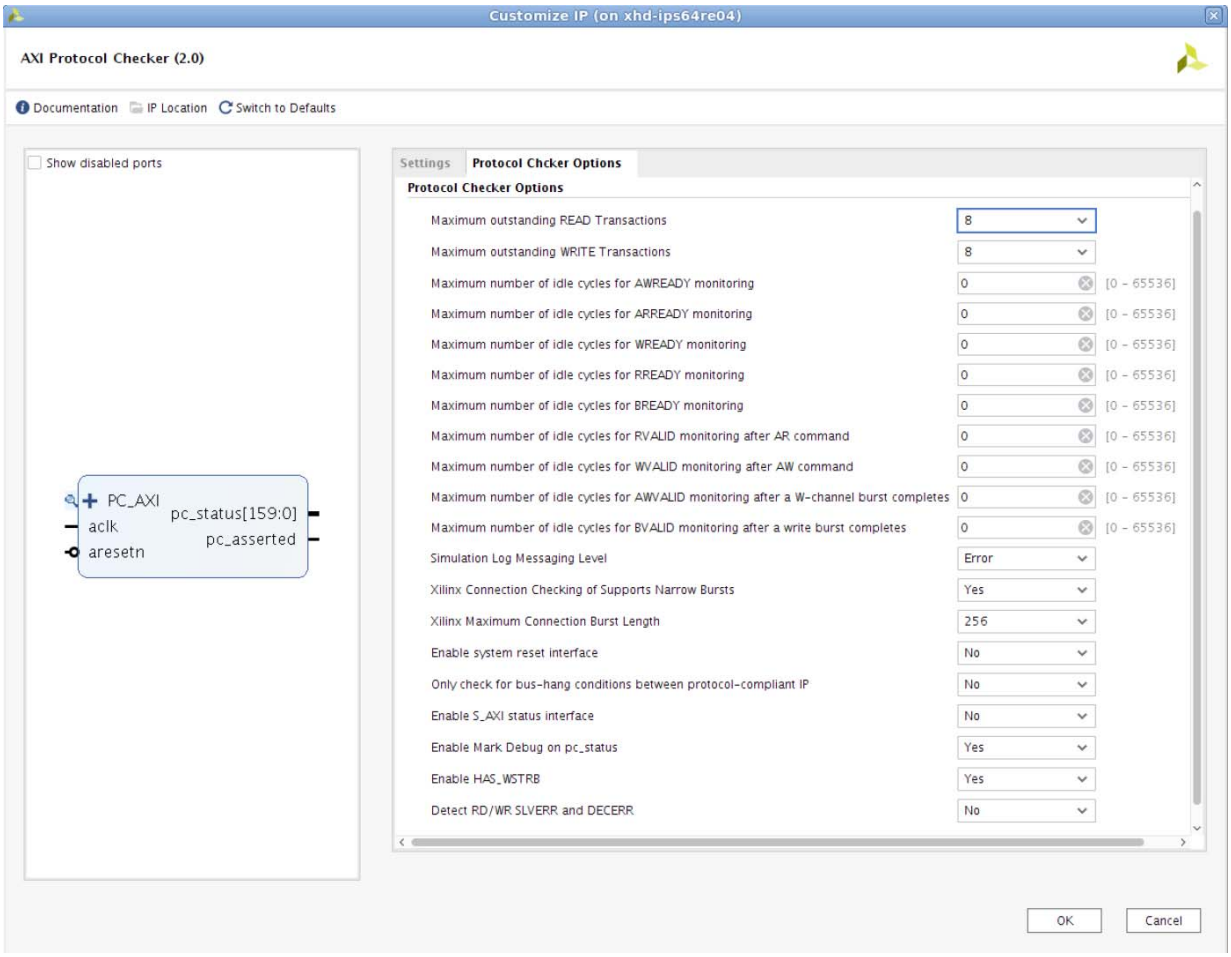

Figure 4-2: **AXI Protocol Checker Options IDE**

The following subsections discuss the options in detail to serve as a guide.

Global Parameters

- **Component Name**: The base name of the output files generated for the core. Names must begin with a letter and can be composed of any of the following characters: a to z, 0 to 9, and "_".
- **Connection Protocol**: Specifies the type of interface for the AXI Protocol Checker to monitor: AXI4, AXI3 or AXI4-Lite. Certain protocol checks are not performed based on this parameter.

  Note: This value is automatically set when using IP integrator.
- **READ_WRITE Mode**: Indicates which channels of the interface are active. When READ_WRITE mode is selected, all five channels are active. In WRITE_ONLY mode, only the AW, W, and B channels are active. In READ_ONLY mode, only AR and R channels are active.

  Note: This value is automatically set when using IP integrator.

- **Address Width**: Specifies the width of the `awaddr` and `araddr` signals to be monitored. Protocol checks use these signals for validation of the transactions. For AXI4 and AXI3 protocols, the minimum width is 12 bits, and for AXI4-Lite, the minimum width is 1 bit.
- **Data Width**: Specifies the width of the Write and Read data path interfaces and its companion signals. Protocol checks use these signals for validation of the transactions. For AXI4 and AXI3 protocols, the value can be any of: 32, 64, 128, 256, 512, or 1024 bits. For AXI4-Lite, the value can be either 32 or 64 bits.

  Note: This value is automatically set when using IP integrator.
- **ID Width**: This parameter specifies the width of the `awid`, `arid`, `bid`, `wid` (for AXI3) and `rid` signals. This parameter is not available when the protocol is AXI4-Lite. The ID width is used for transaction completion tracking and checking and will generate one FIFO for each ID value.

  Note: This value is automatically set when using IP integrator.

User Signal Widths

- **AWUSER Width**: Specifies the width of the `awuser` signals (if any) to be monitored by the protocol checker. This value is only available when the protocol is AXI4 or AXI3. If set to 0, the signal is omitted.

  Note: This value is automatically set when using IP integrator.
- **ARUSER Width**: Specifies the width of the `aruser` signals (if any) to be monitored by the protocol checker. This value is only available when the protocol is AXI4 or AXI3. If set to 0, the signal is omitted.

  Note: This value is automatically set when using IP integrator.
- **WUSER Width**: Specifies the width of the `wuser` signals (if any) to be monitored by the protocol checker. This value is only available when the protocol is AXI4 or AXI3. If set to 0, the signal is omitted.

  Note: This value is automatically set when using IP integrator.
- **RUSER Width**: Specifies the width of the `ruser` signals (if any) to be monitored by the protocol checker. This value is only available when the protocol is AXI4 or AXI3. If set to 0, the signal is omitted.

  Note: This value is automatically set when using IP integrator.
- **BUSER Width**: Specifies the width of the `buser` signals (if any) to be monitored by the protocol checker. This value is only available when the protocol is AXI4 or AXI3. If set to 0, the signal is omitted.

  Note: This value is automatically set when using IP integrator.

Parameter Checker Options

- **Maximum outstanding READ transactions**: Specifies the depth of the FIFOs for storing the outstanding Read transactions. The value should be equal to or greater than the number of outstanding Read transactions expected for that connection.
- **Maximum outstanding WRITE transactions**: Specifies the depth of the FIFOs for storing the outstanding Write transactions. The value should be equal to or greater than the number of outstanding Write transactions expected for that connection.
- **Maximum number of idle cycles for AWREADY monitoring**: This parameter specifies the maximum number of cycles between the assertion of `awvalid` to the assertion of `awready` before an ERROR is generated. When the value is set to 0, this check is disabled.
- **Maximum number of idle cycles for ARREADY monitoring**: This parameter specifies the maximum number of cycles between the assertion of `arvalid` to the assertion of `arready` before an ERROR is generated. When the value is set to 0, this check is disabled.
- **Maximum number of idle cycles for WREADY monitoring**: This parameter specifies the maximum number of cycles between the assertion of `wvalid` to the assertion of `wready` before an ERROR is generated. When the value is set to 0, this check is disabled.
- **Maximum number of idle cycles for RREADY monitoring**: This parameter specifies the maximum number of cycles between the assertion of `rvalid` to the assertion of `rready` before an ERROR is generated. When the value is set to 0, this check is disabled.
- **Maximum number of idle cycles for BREADY monitoring**: This parameter specifies the maximum number of cycles between the assertion of `bvalid` to the assertion of `bready` before an ERROR is generated. When the value is set to 0, this check is disabled.
- **Maximum number of idle cycles for RVALID monitoring after AR command**: This parameter specifies the maximum number of idle cycles for RVALID monitoring after AR command transfer. When the value is set to 0, this check is disabled.
- **Maximum number of idle cycles for WVALID monitoring after AW command**: This parameter specifies the maximum number of idle cycles for WVALID monitoring after AW command transfer. When the value is set to 0, this check is disabled.
- **Maximum number of idle cycles for AWVALID monitoring after a W-channel burst completes**: This parameter specifies the maximum number of idle cycles for AWVALID monitoring after a WLAST transfer. When the value is set to 0, this check is disabled.
- **Maximum number of idle cycles for BVALID monitoring after a write burst completes**: This parameter specifies the maximum number of idle cycles for BVALID monitoring after a write burst completes. When the value is set to 0, this check is disabled.

- **Simulation Log Messaging Level**: When a violation is triggered, this parameter allows the core to indicate to the simulation log file different error levels or to disable all messaging entirely. It is also possible to use this parameter to stop or finish the simulation upon a protocol violation occurrence.
- **Xilinx Connection Checking of Supports Narrow Bursts**: If set to no, this parameter specifies that the protocol checker should trap narrow burst violations on the connection. This attribute is not available for AXI4-Lite interfaces.

  Note: This value is automatically set when using IP integrator.
- **Xilinx Maximum Connection Burst Length**: The protocol checker should trap transactions lengths which exceed the value indicated by this parameter. The default value varies by protocol and is not available for AXI4-Lite interfaces.

  Note: This value is automatically set when using IP integrator.
- **Enable system reset interface**: Enables the `system_resetn` port. When disabled, the port is tied High.
- **Only check for bus-hang conditions between protocol-compliant IP**: Configures core in *Lightweight* mode. Only useful when implementing in hardware. Reduces implementation resources by disabling checks that are not likely to cause the connected AXI network to hang.
- **Enable S_AXI status interface**: Enable the AXI4-Lite control register slave interface.
- **Enable Mark_Debug**: When enabled, the set mark_debug attribute is set to True on the pc_status output bus upon synthesis. Disabling this attribute sets the mark_debug attribute to false.
- **Detect RD/WR SLVERR and DECERR**: When set to Yes, this checker detects SLVERR, DECERR response on Read, Write channels. Detection is off when the parameter checker is set to No.

User Parameters

Table 4-1 shows the relationship between the fields in the Vivado IDE and the User Parameters (which can be viewed in the Tcl console).

Table 4-1: **Vivado IDE Parameter to User Parameter Relationship**

Vivado IDE Parameter/Value	User Parameter/Value	Default Value
Connection Protocol	PROTOCOL	AXI4
READ_WRITE Mode	READ_WRITE_MODE	READ_WRITE
Address Width	ADDR_WIDTH	32
Data Width	DATA_WIDTH	32
ID Width	ID_WIDTH	0
AWUSER Width	AWUSER_WIDTH	0
ARUSER Width	ARUSER_WIDTH	0
WUSER Width	WUSER_WIDTH	0
RUSER Width	RUSER_WIDTH	0
BUSER Width	BUSER_WIDTH	0
Maximum Outstanding Read Transactions	MAX_RD_BURSTS	8
Maximum Outstanding Write Transactions	MAX_WR_BURSTS	8
Maximum Number of Idle Cycles for AWREADY Monitoring(1)	MAX_AW_WAITS	0 (disabled)
Maximum Number of Idle Cycles for ARREADY Monitoring(1)	MAX_AR_WAITS	0 (disabled)
Maximum Number of Idle Cycles for WREADY Monitoring(1)	MAX_W_WAITS	0 (disabled)
Maximum Number of Idle Cycles for RREADY Monitoring(1)	MAX_R_WAITS	0 (disabled)
Maximum Number of Idle Cycles for BREADY Monitoring(1)	MAX_B_WAITS	0 (disabled)
Maximum number of idle cycles for RVALID monitoring after AR command(1)	MAX_CONTINUOUS_RTRANSFERS_WAITS	0 (disabled)
Maximum number of idle cycles for WVALID monitoring after AW command(1)	MAX_CONTINUOUS_WTRANSFERS_WAITS	0 (disabled)
Maximum number of idle cycles for AWVALID monitoring after a W-channel burst completes(1)	MAX_WLAST_TO_AWVALID_WAITS	0 (disabled)
Maximum number of idle cycles for BVALID monitoring after a write burst completes(1)	MAX_WRITE_TO_BVALID_WAITS	0 (disabled)

Table 4-1: **Vivado IDE Parameter to User Parameter Relationship *(Cont'd)***

Vivado IDE Parameter/Value	User Parameter/Value	Default Value
Simulation Log Messaging Level • Quiet • Info • Error • Stop on error • Finish on error	MESSAGE_LEVEL • 0 • 1 • 2 • 3 • 4	2 (Error)
Xilinx Connection Checking of Supports Narrow Burst • No • Yes	SUPPORTS_NARROW_BURST • 0 • 1	1 (Yes)
Xilinx Maximum Connection Burst Length	MAX_BURST_LENGTH	256
Enable System Reset Interface • No • Yes	HAS_SYSTEM_RESET • 0 • 1	0 (No)
Only check for bus-hang conditions between protocol-compliant IP • No • Yes	LIGHT_WEIGHT • 0 • 1	0 (No)
Enable S_AXI status interface • No • Yes	ENABLE_CONTROL • 0 • 1	0 (No)
Enable Mark_Debug on pc_status • No • Yes	ENABLE_Mark_Debug • 0 • 1	1 (Yes)
Detect RD/WR SLVERR and DECERR • No • Yes	CHK_ERR_RESP • 0 • 1	0 (No)

Notes:
1. Timeout check is disabled when the parameter is set to 0.

Output Generation

The AXI Protocol Checker deliverables are organized in the directory `<project_name>/<project_name>.srcs/sources_1/ip/<component_name>` and is designed as the `<ip_source_dir>`.

For details, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 6].

Constraining the Core

This section contains information about constraining the core in the Vivado® Design Suite.

Required Constraints

There are no required constraints for this core.

Device, Package, and Speed Grade Selections

See IP Facts for details on device support.

Clock Frequencies

There are no clock frequency constraints for this core.

Clock Management

There are no clock management constraints for this core.

Clock Placement

There are no clock placement constraints for this core.

Banking

There are no banking constraints for this core.

Transceiver Placement

There are no transceiver placement constraints for this core.

I/O Standard and Placement

There are no I/O constraints for this core.

Simulation

For details, see *Vivado Design Suite User Guide: Logic Simulation* (UG900) [Ref 7].

Synthesis and Implementation

For details about synthesis and implementation, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 6].

Debugging

This appendix includes details about resources available on the Xilinx Support website and debugging tools.

Finding Help on Xilinx.com

To help in the design and debug process when using the AXI Protocol Checker, the Xilinx Support web page contains key resources such as product documentation, release notes, answer records, information about known issues, and links for obtaining further product support.

Documentation

This product guide is the main document associated with the AXI Protocol Checker. This guide, along with documentation related to all products that aid in the design process, can be found on the Xilinx Support web page or by using the Xilinx Documentation Navigator.

Download the Xilinx Documentation Navigator from the Downloads page. For more information about this tool and the features available, open the online help after installation.

Answer Records

Answer Records include information about commonly encountered problems, helpful information on how to resolve these problems, and any known issues with a Xilinx product. Answer Records are created and maintained daily ensuring that users have access to the most accurate information available.

Answer Records for this core can be located by using the Search Support box on the main Xilinx support web page. To maximize your search results, use proper keywords such as

- Product name
- Tool message(s)
- Summary of the issue encountered

A filter search is available after results are returned to further target the results.

Master Answer Record for the AXI Protocol Checker

The Master Answer Record is Xilinx Answer 57790.

Technical Support

Xilinx provides technical support at the Xilinx Support web page for this LogiCORE™ IP product when used as described in the product documentation. Xilinx cannot guarantee timing, functionality, or support if you do any of the following:

- Implement the solution in devices that are not defined in the documentation.
- Customize the solution beyond that allowed in the product documentation.
- Change any section of the design labeled DO NOT MODIFY.

To contact Xilinx Technical Support, navigate to the Xilinx Support web page.

General Checks

The AXI Protocol Checker design limits the types of problems one may encounter when using the core. In the case where the interface is not fully specified, the system designer must ensure that any unused inputs to the AXI Protocol Checker have been correctly tied off based on the AXI Protocol that is to be monitored using Table 2-2, Table 2-3, or Table 2-4.

Debug Tools

Vivado Design Suite Debug Feature

The Vivado® Design Suite debug feature inserts logic analyzer and virtual I/O cores directly into your design. The debug feature also allows you to set trigger conditions to capture application and integrated block port signals in hardware. Captured signals can then be analyzed. This feature represents the functionality in the Vivado IDE that is used for logic debugging and validation of a design running in Xilinx devices in hardware.

The AXI4 Protocol Checker core supports probing using the Vivado ILA 2.0 core and the Vivado Logic Analyzer. All of the protocol checks named in Table 2-2, Table 2-3, and Table 2-4 are available as Unassigned Debug Nets in the synthesized design. For more details, see *Vivado Design Suite Tutorial: Programming and Debugging* (UG936) [Ref 5].

It is also possible to monitor all or one bit of the `pc_status` vector through any other desired method.

Clocks and Resets

To resolve clocking and reset issues, verify these items:

- Check that `aclk` is connected to the same clock that is driving both the Master and Slave interfaces.
- Check that `aresetn` is connected to the same reset that is driving both the Master and Slave interfaces.
- Ensure that both `aresetn` and `system_resetn` (if enabled) are connected to active-Low polarity.
- Ensure that `aresetn` is both synchronously asserted and released on `aclk`.

Core Size and Optimization

In some cases the size of the core can become very large. The following tips can reduce the size of the core:

- When trying to isolate the cause of a bus hang in hardware, configure the IP in Lightweight mode (select **Only check for bus-hang conditions between protocol-compliant IP**). This may also allow you to deploy Protocol Checkers on more AXI connections in the system.
- Set the **Maximum number of idle cycles for (AW|AR|W|R|B) READY monitoring** to 0. This disables a recommended *VALID to *READY wait checks.
- If possible, reduce the ID Width. This directly reduces the number of ID tracker and block RAMs.
- Reduce either or both of **Maximum outstanding READ transactions PER ID** or **Maximum outstanding WRITE transactions PER ID**.
- Each bit of the `pc_status` vector consumes some amount of resources; therefore, fewer bits observed reduces the overall foot print of the design.

Flags

No Flags Asserted

One simple test to check to see if the AXI Protocol Checker is correctly connected to the interface is to not connect the `pc_axi_aready` or the `pc_axi_awready` input into the protocol checker and to tie those ports to 0. This causes the `pc_asserted` output and

multiple bits in the `pc_status` vector to be asserted when AXI traffic begins because this will violate all the AXI_ERRS_A*_STABLE protocol checks (bits 46 - 56). See Table 2-6 for the descriptions of these checks.

Flags Asserted

If there are bits asserted in the `pc_status` vector and the source/reason of the violation using Table 2-6 is not clear, move the AXI Protocol Checker "upstream" toward the AXI Master generating the transactions.

Migrating and Upgrading

This appendix contains information about migrating a design from ISE® to the Vivado® Design Suite, and for upgrading to a more recent version of the IP core. For customers upgrading in the Vivado Design Suite, important details (where applicable) about any port changes and other impact to user logic are included.

Migrating to the Vivado Design Suite

For information about migrating to the Vivado Design Suite, see the *Vivado Design Suite Migration Methodology Guide* (UG911) [Ref 9].

Upgrading in the Vivado Design Suite

This section provides information about any changes to the user logic or port designations that take place when you upgrade to a more current version of this IP core in the Vivado Design Suite.

Parameter Changes

There are no parameter changes.

Port Changes

There are no port changes.

Additional Resources and Legal Notices

Xilinx Resources

For support resources such as Answers, Documentation, Downloads, and Forums, see Xilinx Support.

References

These documents provide supplemental material useful with this product guide:

1. ARM® AMBA® AXI Protocol v2.0, ARM IHI 0022C
2. AMBA® 4 AXI4™, AXI4-Lite™, and AXI4-Stream™ Protocol Assertions User Guide
3. *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994)
4. *Xilinx Vivado AXI Reference Guide* (UG1037)
5. *Vivado Design Suite Tutorial: Programming and Debugging* (UG936)
6. *Vivado Design Suite User Guide: Designing with IP* (UG896)
7. *Vivado Design Suite User Guide: Logic Simulation* (UG900)
8. *Vivado Design Suite User Guide: Getting Started* (UG910)
9. *Vivado Design Suite Migration Methodology Guide* (UG911)

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
04/04/2018	2.0	• Added four new Protocol Checks ◦ XILINX_AXI4_RD_RESP_SLVERR ◦ XILINX_AXI4_RD_RESP_DECERR ◦ XILINX_AXI4_WR_RESP_SLVERR ◦ XILINX_AXI4_WR_RESP_DECERR • Added parameter checker option description for Detect RD/WR SLVERR and DECERR • Updated GUI screen captures • Removed table note markers from the notes column in the AXI4/AXI3 and AXI4-Lite Protocol Checks and Descriptions table in the Product Specification chapter. • Added a table note about Timeout check when parameter is set to zero under the User Parameters table in Design Flow Steps chapter.
12/20/2017	2.0	• Updated Vivado IDE Parameter to User Parameter Relationship table with details of Enable_Mark_Debug attribute. • Updated Parameter Checker Option with a description of Enable Mark_Debug. • Updated the AXI Protocol Checker Customization IDE screen capture
10/04/2017	2.0	• The pc_status width is now fixed to 160 bits. • Removed user parameter ENABLE_EXT_CHECKS. • Added Assertion check for bit 101 (XILINX_ARESETN_PULSE_WIDTH).
06/07/2017	1.1	Added simulation-only assertions.
04/05/2017	1.1	• Added new timeout checks against the next expected assertion of each VALID handshake output. • Added LIGHT_WEIGHT mode to reduce synthesis utilization. • Added AXI4-Lite control register slave interface.
10/05/2016	1.1	Updated “AXI_ERRS_RID” protocol support and description.
11/18/2015	1.1	Added support for UltraScale+ families.
04/01/2015	1.1	• Added “User Parameters” in the Design Flow Steps chapter.
12/18/2013	1.1	• Added support for UltraScale™ architecture.
10/02/2013	1.1	• Added compatibility with AXI Interconnect v2.1. • Added Chapter 6, Simulation and Chapter 7, Synthesis and Implementation.
06/19/2013	1.0	• Updated Parameter Checker Options in Chapter 4. • Clarified bit options for `pc_status` vector.
12/18/2012	1.0	Initial Xilinx release.

Please Read: Important Legal Notices

The information disclosed to you hereunder (the "Materials") is provided solely for the selection and use of Xilinx products. To the maximum extent permitted by applicable law: (1) Materials are made available "AS IS" and with all faults, Xilinx hereby DISCLAIMS ALL WARRANTIES AND CONDITIONS, EXPRESS, IMPLIED, OR STATUTORY, INCLUDING BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, OR FITNESS FOR ANY PARTICULAR PURPOSE; and (2) Xilinx shall not be liable (whether in contract or tort, including negligence, or under any other theory of liability) for any loss or damage of any kind or nature related to, arising under, or in connection with, the Materials (including your use of the Materials), including for any direct, indirect, special, incidental, or consequential loss or damage (including loss of data, profits, goodwill, or any type of loss or damage suffered as a result of any action brought by a third party) even if such damage or loss was reasonably foreseeable or Xilinx had been advised of the possibility of the same. Xilinx assumes no obligation to correct any errors contained in the Materials or to notify you of updates to the Materials or to product specifications. You may not reproduce, modify, distribute, or publicly display the Materials without prior written consent. Certain products are subject to the terms and conditions of Xilinx's limited warranty, please refer to Xilinx's Terms of Sale which can be viewed at https://www.xilinx.com/legal.htm#tos; IP cores may be subject to warranty and support terms contained in a license issued to you by Xilinx. Xilinx products are not designed or intended to be fail-safe or for use in any application requiring fail-safe performance; you assume sole risk and liability for use of Xilinx products in such critical applications, please refer to Xilinx's Terms of Sale which can be viewed at https://www.xilinx.com/legal.htm#tos.

AUTOMOTIVE APPLICATIONS DISCLAIMER

AUTOMOTIVE PRODUCTS (IDENTIFIED AS "XA" IN THE PART NUMBER) ARE NOT WARRANTED FOR USE IN THE DEPLOYMENT OF AIRBAGS OR FOR USE IN APPLICATIONS THAT AFFECT CONTROL OF A VEHICLE ("SAFETY APPLICATION") UNLESS THERE IS A SAFETY CONCEPT OR REDUNDANCY FEATURE CONSISTENT WITH THE ISO 26262 AUTOMOTIVE SAFETY STANDARD ("SAFETY DESIGN"). CUSTOMER SHALL, PRIOR TO USING OR DISTRIBUTING ANY SYSTEMS THAT INCORPORATE PRODUCTS, THOROUGHLY TEST SUCH SYSTEMS FOR SAFETY PURPOSES. USE OF PRODUCTS IN A SAFETY APPLICATION WITHOUT A SAFETY DESIGN IS FULLY AT THE RISK OF CUSTOMER, SUBJECT ONLY TO APPLICABLE LAWS AND REGULATIONS GOVERNING LIMITATIONS ON PRODUCT LIABILITY.

© Copyright 2012-2018 Xilinx, Inc. Xilinx, the Xilinx logo, Artix, ISE, Kintex, Spartan, Virtex, Vivado, Zynq, and other designated brands included herein are trademarks of Xilinx in the United States and other countries. All other trademarks are the property of their respective owners.