LogiCORE IP AXI Protocol Checker v1.0

Product Guide

PG101 June 19, 2013

XILINX®

XILINX®

Table of Contents

Appendix A: Debugging

Appendix B: Additional Resources

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 Assertion" library [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 (per channel): Up to 1024 bits
- 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

LogiCORE IP Facts Table	
Core Specifics	
Supported Device Family(1)	Virtex®-7, Kintex®-7, Artix®-7
Supported User Interfaces	AXI4, AXI4-Lite, AXI3
Resources	See Table 2-1.
Provided with Core	
Design Files	Vivado 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	Mentor Graphics Questa® SIM Vivado Simulator
Synthesis	Vivado Synthesis.
Support	
Provided by Xilinx @ www.xilinx.com/support	

Notes:

1. For a complete list of supported derivative devices, see Embedded Edition Derivative Device Support.
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 the AXI4, AXI3 or AXI4-Lite protocol. When placed in an AXI system, the connection of the AXI Protocol Checker monitors the traffic between the AXI Master and AXI Slave Core.

The interface is checked against the rules outlined in the AXI Specification to determine if a violation has occurred [Ref 2]. These 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.

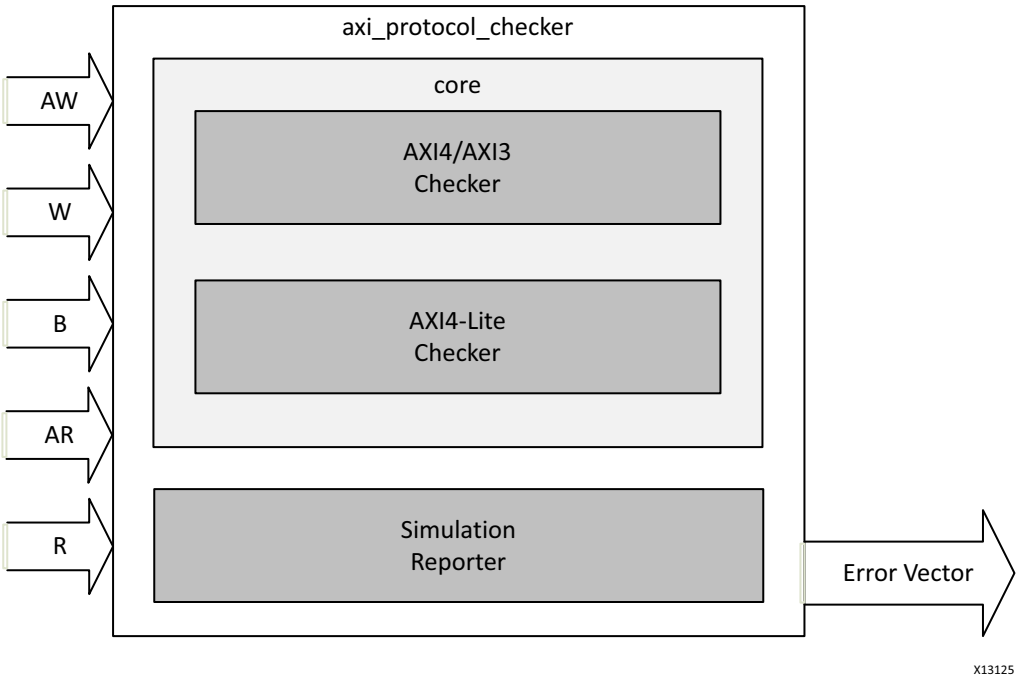

Figure 1-1: **AXI Protocol Checker Block Diagram**

Applications

The AXI Protocol Checker is typically used by system designers 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 converted 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.

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 either the connection has been reset with `aresetn` or the core has been reset with `system_resetn`.

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 2].

Performance

This section includes information about the core performance.

Maximum Frequencies

The maximum frequency with a 64-bit AXI data width is 200 MHz.

Resource Utilization

Resources required for the AXI Protocol Checker have been estimated for the Kintex®-7 XC7K325T FPGA (Table 2-1). These values were generated using Vivado™ IP Catalog v2012.4. They are derived from post-synthesis reports, and might change during MAP and PAR.

Table 2-1: **Kintex-7 FPGA Resource Estimates**

Protocol	Data Width	ID Width	Maximum Read/ Write Transactions	Maximum READY Wait Cycles	LUTs	FFs
AXI4	1024	3	32	16	4132	5072
	1024	3	32	0	4044	4982
	1024	3	2	16	3565	4780
	1024	3	2	0	3474	4686
	1024	0	32	16	3620	4806
	1024	0	32	0	3524	4716
	1024	0	2	16	3516	4738
	1024	0	2	0	3424	4652
	128	3	32	16	1467	1264
	128	3	32	0	1375	1174
	128	3	2	16	886	972
	128	3	2	0	798	882
	128	0	32	16	930	994
	128	0	32	0	844	908
	128	0	2	16	837	934
	128	0	2	0	745	844
	64	3	32	16	1279	992
	64	3	32	0	1197	902
	64	3	2	16	691	696
	64	3	2	0	603	610
	64	0	32	16	746	722
	64	0	32	0	654	632
	64	0	2	16	642	662
	64	0	2	0	553	572
	32	3	32	16	1210	856
	32	3	32	0	1119	766
	32	3	2	16	623	564
	32	3	2	0	532	474
	32	0	32	16	672	590
	32	0	32	0	583	500
	32	0	2	16	574	526
	32	0	2	0	486	436

Table 2-1: **Kintex-7 FPGA Resource Estimates *(Cont'd)***

Protocol	Data Width	ID Width	Maximum Read/ Write Transactions	Maximum READY Wait Cycles	LUTs	FFs
AXI4-Lite	64	0	32	16	528	591
	64	0	32	0	437	501
	64	0	2	16	429	527
	64	0	2	0	340	437
	32	0	32	16	449	455
	32	0	32	0	357	365
	32	0	2	16	348	391
	32	0	2	0	258	301

Port Descriptions

This section contains details about the AXI Protocol Checker ports.

Protocol Independent Port Descriptions

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

Table 2-2: **Protocol independent port descriptions**

Signal Name	Direction	Default	Width	Description
aclk	Input	Required	1	Interface clock input.
aresetn	Input	Required	1	Interface reset input (active-Low).
system_resetn	Input	Optional	1	System reset (active-Low).
pc_status	Output		90	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.

AXI4 Port Descriptions

Table 2-3 lists the interface signals for the AXI Protocol Checker core when it is configured to check an AXI4 Interface.

Table 2-3: **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
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

Table 2-3: **AXI4 Protocol Port Descriptions *(Cont'd)***

Signal Name	Direction	Default	Width	Description
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

AXI3 Port Descriptions

Table 2-4 lists the interface signals for the AXI Protocol Checker core when it is configured to check an AXI3 Interface.

Table 2-4: **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)

Table 2-4: **AXI3 Protocol Port Descriptions *(Cont'd)***

Signal Name	Direction	Default	Width	Description
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
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

Table 2-4: **AXI3 Protocol Port Descriptions *(Cont'd)***

Signal Name	Direction	Default	Width	Description
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

AXI4-Lite Port Descriptions

Table 2-5 lists the interface signals for the AXI Protocol Checker core when it is configured to check an AXI4-Lite Interface.

Table 2-5: **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

Table 2-5: **AXI4-Lite Protocol Port Descriptions *(Cont'd)***

Signal Name	Direction	Default	Width	Description
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

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	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	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.

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

Name of Protocol Check	Bit	Protocol Support	Description
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.
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	AXI4/AXI3/Lite	Recommended that AWREADY is asserted within MAXWAITS cycles of AWVALID being asserted.

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

Name of Protocol Check	Bit	Protocol Support	Description
AXI_ERRM_WDATA_NUM	21	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.
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	AXI4/AXI3/Lite	Recommended that WREADY is asserted within MAXWAITS cycles of WVALID being asserted.
AXI_ERRS_BRESP_WLAST	29	AXI4/AXI3	A slave must not take BVALID HIGH until after the last write data is handshake is complete.
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	32	AXI4/AXI3/Lite	A slave must not take BVALID HIGH until after the write address is handshake is complete.
AXI_ERRS_BID_STABLE	33	AXI4/AXI3	Handshake Checks: BID must remain stable when BVALID is asserted and BREADY Low .

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

Name of Protocol Check	Bit	Protocol Support	Description
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	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.
AXI_ERRM_ARSIZE	44	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.

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

Name of Protocol Check	Bit	Protocol Support	Description
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	AXI4/AXI3/Lite	Recommended that ARREADY is asserted within MAXWAITS cycles of ARVALID being asserted.
AXI_ERRS_RDATA_NUM	58	AXI4/AXI3	The number of read data items must match the corresponding ARLEN.
AXI_ERRS_RID	59	AXI4/AXI3	The read data must always follow the address that it relates to. Therefore, a slave can only give read data with an ID to match an outstanding read transaction.
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.
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	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.

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

Name of Protocol Check	Bit	Protocol Support	Description
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.
AXI_ERRS_RUSER_STABLE	77	AXI4/AXI3	RUSER must remain stable when RVALID is asserted and RREADY Low.
AXI_AUXM_RCAM_OVERFLOW	78	AXI4/AXI3/Lite	Read CAM overflow, increase MAXRBURSTS parameter.
AXI_AUXM_RCAM_UNDERFLOW	79	AXI4/AXI3/Lite	Read CAM underflow.
AXI_AUXM_WCAM_OVERFLOW	80	AXI4/AXI3/Lite	Write CAM overflow, 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.

Xilinx Configuration Checks and Descriptions

The Xilinx Configuration Checks are listed and described in Table 2-7.

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

Name of Protocol Check	Bit	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_AWVALID_RESET	92	AXI4/AXI3/Lite	AWREADY is Low for the first cycle after ARESETn goes High.
XILINX_WVALID_RESET	93	AXI4/AXI3/Lite	WREADY is Low for the first cycle after ARESETn goes High.
XILINX_BVALID_RESET	94	AXI4/AXI3/Lite	BREADY is Low for the first cycle after ARESETn goes High.
XILINX_ARVALID_RESET	95	AXI4/AXI3/Lite	ARREADY is Low for the first cycle after ARESETn goes High.
XILINX_RVALID_RESET	96	AXI4/AXI3/Lite	RREADY is Low for the first cycle after ARESETn goes High.

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.

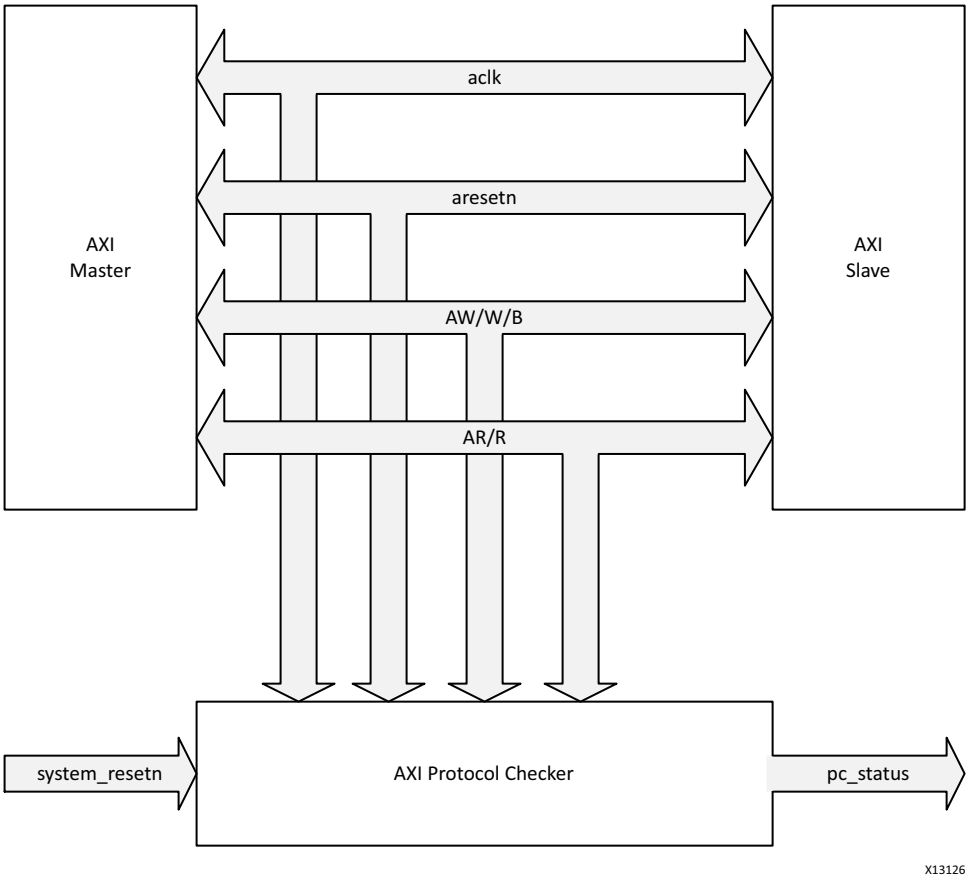

Figure 3-1: **Example Topology**

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	Description
<instance_path>	The simulation hierarchy to the protocol checker instance that is issuing the message.
<bit_number>	Indicates which bit in the pc_status vector that is asserted. 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 GUI.
<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 greater amount of resources are 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 or 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 named in both Table 2-6 and Table 2-7 are available as Unassigned Debug Nets in the synthesized design. See UG936, *Vivado Design Suite Tutorial: Programming and Debugging* on the Vivado Design Suite documentation page.

Clocking

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

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.

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 that defines the state of the interface following the de-assertion of `aresetn`. Protocol violation notifications related to the required behavior of interface with respect to `aresetn` are cleared using `system_resetn`.

When a system reset is not available, the `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

Customizing and Generating the Core

This chapter includes information about using Xilinx tools to customize and generate the core in the Vivado™ Design Suite environment.

GUI

Locate the IP core in the Vivado IP Catalog and click it once to select it. Details regarding the solution are displayed in the Details window. To configure the IP, double-click the IP name in the IP catalog.

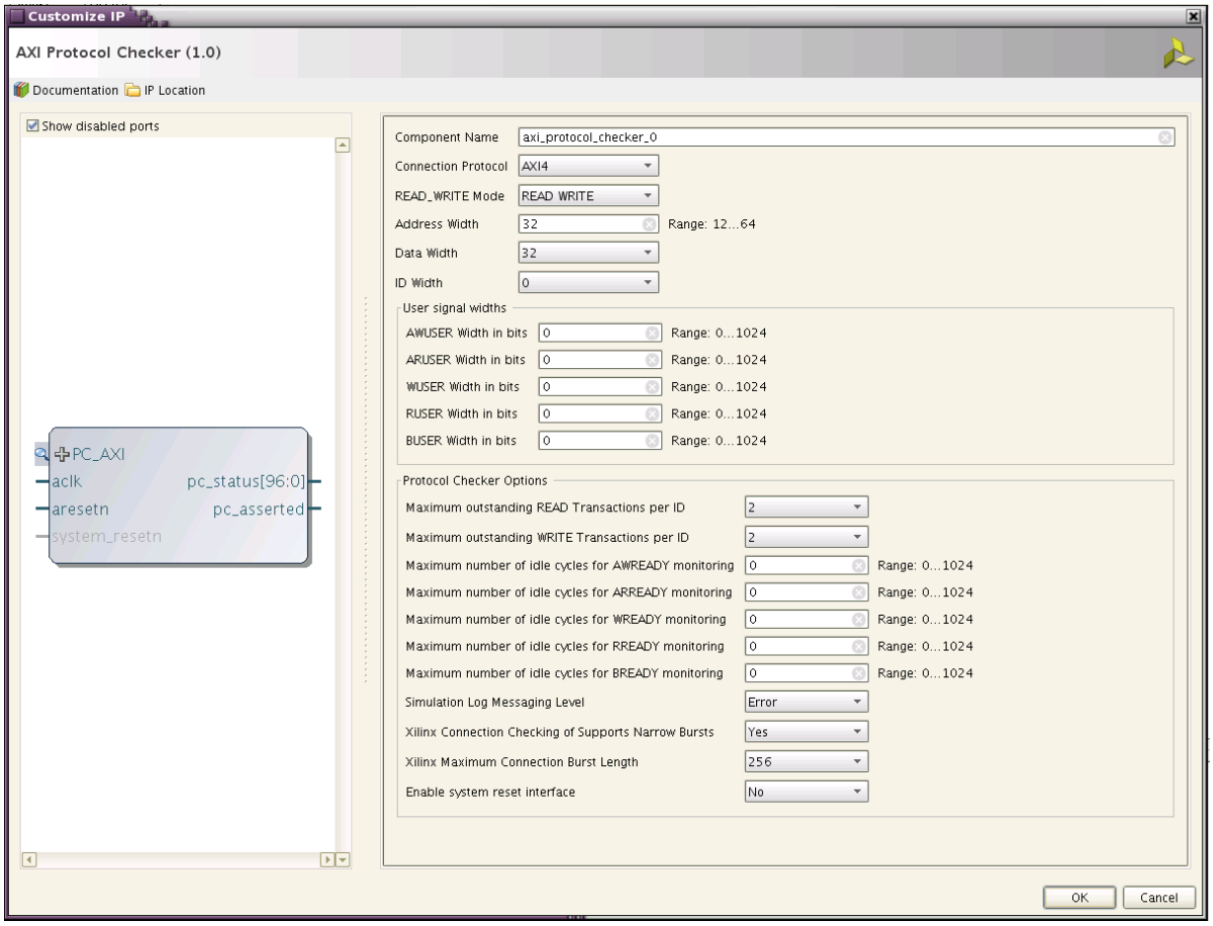

Figure 4-1: **AXI Protocol Checker Customization GUI**

Modify the parameters to meet the requirements of the larger project into which the core is integrated. 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.
- **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.
- **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.
- **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 AXI4Lite. The ID width is used for transaction completion tracking and checking and will generate one FIFO for each ID value.

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.
- **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.
- **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.
- **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.

- **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.

Parameter Checker Options

- **Maximum outstanding Read transactions Per ID**: Specifies the depth of the FIFOs for storing the outstanding Read transactions per ID. The value should be equal to or greater than the number of outstanding Read transactions for that connection.
- **Maximum outstanding Write transactions PER ID**: 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 for that connection.
- **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 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 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.
- **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 Burst**: 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.

- **Xilinx Maximum Connection Burst Length**: The protocol checker should trap transactions lengths exceed the value indicated by this parameter. The default value varies by protocol and is not available for AXI4-Lite interfaces.
- **Enable system reset interface**: Enables the system_resetn port. When disabled, the port is tied High.

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>. The relevant contents or the directories are described in the following sections.

<ip_source_dir>

The Vivado tools project files are located in the root of the <ip_source_dir>.

Table 4-1: **<ip_source_dir>**

Name	Description
<component_name>.xci	Vivado tools IP configuration options file. This file can be imported into any Vivado tools design and be used to generate all other IP source files.
<component_name>.{veo|vho}	AXI Protocol Checker instantiation template.

The files shown in Table 4-2 are located in the subdirectories of <ip_source_dir>.

Table 4-2: **<ip_source_dir> Subdirectories**

Name	Description
hdl/verilog/*.v	AXI Protocol Checker source files.
synth/<component_name>.v	AXI Protocol Checker generated top-level file for synthesis. This file is optional and only generated if synthesis target is selected.
sim/<component_name>.v	AXI Protocol Checker generated top-level file for simulation. This file is optional and only generated if simulation target is selected.

Constraining the Core

This chapter contains information about constraining the core in the Vivado™ Design Suite environment.

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.

Appendix A

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 (www.xilinx.com/support) contains key resources such as product documentation, release notes, answer records, information about known issues, and links for opening a Technical Support WebCase.

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 (www.xilinx.com/support) or by using the Xilinx Documentation Navigator.

Download the Xilinx Documentation Navigator from the Design Tools tab on the Downloads page (www.xilinx.com/download). For more information about this tool and the features available, open the online help after installation.

Technical Support

Xilinx provides technical support at www.xilinx.com/support for this LogiCORE™ IP product when used as described in the product documentation. Xilinx cannot guarantee timing, functionality, or support of product if implemented in devices that are not defined in the documentation, if customized beyond that allowed in the product documentation, or if changes are made to any section of the design labeled DO NOT MODIFY.

See the Embedded Edition Derivative Device Support web page (www.xilinx.com/ise/embedded/ddsupport.htm) for a complete list of supported derivative devices for this core.

Contacting Technical Support

Xilinx provides premier technical support for customers encountering issues that require additional assistance.

To contact Xilinx Technical Support:

1. Navigate to www.xilinx.com/support.
2. Open a WebCase by selecting the WebCase link located under Support Quick Links.

When opening a WebCase, include:

- Target FPGA including package and speed grade.
- All applicable Xilinx Design Tools and simulator software versions.
- Additional files based on the specific issue might also be required. See the relevant sections in this debug guide for guidelines about which file(s) to include with the WebCase.

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 2-3, Table 2-4, or Table 2-5.

Debug Tools

Vivado Lab Tools

Vivado inserts logic analyzer and virtual I/O cores directly into your design. Vivado Lab Tools 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 FPGA 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 2-3, Table 2-4, and Table 2-5 are available as Unassigned Debug Nets in the synthesized design. More details can be found in the *Vivado Design Suite Tutorial: Programming and Debugging on the Vivado Design Suite* [Ref 4].

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:

- 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_arready or the pc_axi_awready input into the protocol checker and to tie those ports to 0. This causes 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.

Appendix B

Additional Resources

Xilinx Resources

For support resources such as Answers, Documentation, Downloads, and Forums, see the Xilinx Support website at:

www.xilinx.com/support.

For a glossary of technical terms used in Xilinx documentation, see:

www.xilinx.com/company/terms.htm.

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. *Xilinx AXI Reference Guide* (UG761)
4. *Vivado Design Suite Tutorial: Programming and Debugging* (UG936)
5. *Vivado Design Suite Migration Methodology Guide* (UG911)
6. Vivado™ Design Suite user documentation (www.xilinx.com/cgi-bin/docs/rdoc?v=2012.4;t=vivado+docs)

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
12/18/12	1.0	Initial Xilinx release.
06/19/13	1.0	• Updated Parameter Checker Options in Chapter 4. • Clarified bit options for `pc_status` vector.

Notice of Disclaimer