LogiCORE IP ChipScope AXI Monitor (v3.03.a)

DS810 April 24, 2012 **Product Specification**

Introduction

The ChipScope™ AXI Monitor core is designed to monitor and debug AXI interfaces. The core allows the probing of any signals going from a peripheral to the AXI interconnect. For example, the user can instantiate a monitor on a MicroBlaze™ processor instruction or data interface to observe all memory transactions going in and out of the processor.

Each monitor core works independently which allows the changing of trigger outputs to enable taking system-level measurements. By using the auxiliary trigger input and trigger output ports of a monitor core, multi-level triggering situations can be created to simplify complex system level measurements. For example, if a system consists of a master device operating at 100 MHz and a slave device operating at 50 MHz, the transfer of data going from one time domain to the next can be analyzed with the multi-tiered triggering functionality of monitor cores.

Moreover, with this system level measurement, not only can complex multi-time domain system level issues be debugged, but latency restrictions in a system can be analyzed as well.

Features

- Selectable data samples
- Generic Trigger/Data Unit with selectable width
- Auto-generated CDC file
- Multiple monitor support in single system through the use of trigger in and trigger out ports
- Allows multiple match units per trigger group
- Added functionality for more than one match unit per trigger group
- Adjustable counter size for triggers
- Selectable AXI Protocol Check monitoring for the AXI4 Memory Map and AXI4-Lite interfaces
- Supports connection to AXI3 Protocol Cores
- Supports EDK and standard CORE Generator flows

LogiCORE IP Facts Table			
Core Specifics			
Supported Device Family[1]	Zynq ™-7000, Virtex®-7, Kintex™-7[6], Virtex-6[3], Spartan®-6[4]		
Supported User Interfaces	AXI4, AXI4-Lite, AXI4-Stream		
Resources Used			
Configuration[5]	**LUTs**	**Flip Flops**	**BRAMs**
1	729	1141	9
2	1099	1451	12
3	865	1360	14
4	730	1155	5
5	1252	1663	6
6	853	1369	7
Provided with Core			
Documentation	Product Specification; UG029 ChipScope Pro Software and Cores User Guide; AMBA 4 AXI4, AXI4-Lite, and AXI4-Stream Protocol Assertions User Guide		
Design Files	VHDL		
Example Design	Not Provided		
Test Bench	Not Provided		
Constraints File	Not Provided		
Simulation Model	N/A		
Tested Design Tools [2]			
Design Entry Tools	EDK and CORE Generator		
Simulation	N/A		
Synthesis Tools	XST		
Support			
Provided by Xilinx @ www.xilinx.com/support			

Notes:

1. For a list of supported derivative devices, see http://www.xilinx.com/ise/embedded/ddsupport.htm.
2. For the supported versions of the tools, see the ISE Design Suite 14: Release Notes Guide.
3. For more information on the Virtex-6 devices, see the DS150, Virtex-6 Family Overview.
4. For more information on the Spartan-6 devices, see the DS160, Spartan-6 Family Overview.
5. For configuration details, see Table 4.
6. For more information, see DS180, 7 Series FPGAs Overview.

© Copyright 2010-2012 Xilinx, Inc. Xilinx, the Xilinx logo, Artix, ISE, Kintex, Spartan, Virtex, 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.

Functional Description

The AXI Monitor is used to debug top-level signals in a system that uses AXI4 protocol specifications by connecting it to an AXI Core using Xilinx® Platform Studio (XPS). When placed in an AXI system, the connection of the AXI Monitor probes between the AXI Interconnect and the AXI Core. The signal list is shown in Table 1. The user can monitor either AXI Memory Map signals through the MON_AXI bus interface or AXI Streaming signals through the MON_AXI_S bus interface (but not both) with a single AXI Monitor core. Communication with the ILA core is conducted using a connection to the JTAG port through the ICON core. See Figure 1.

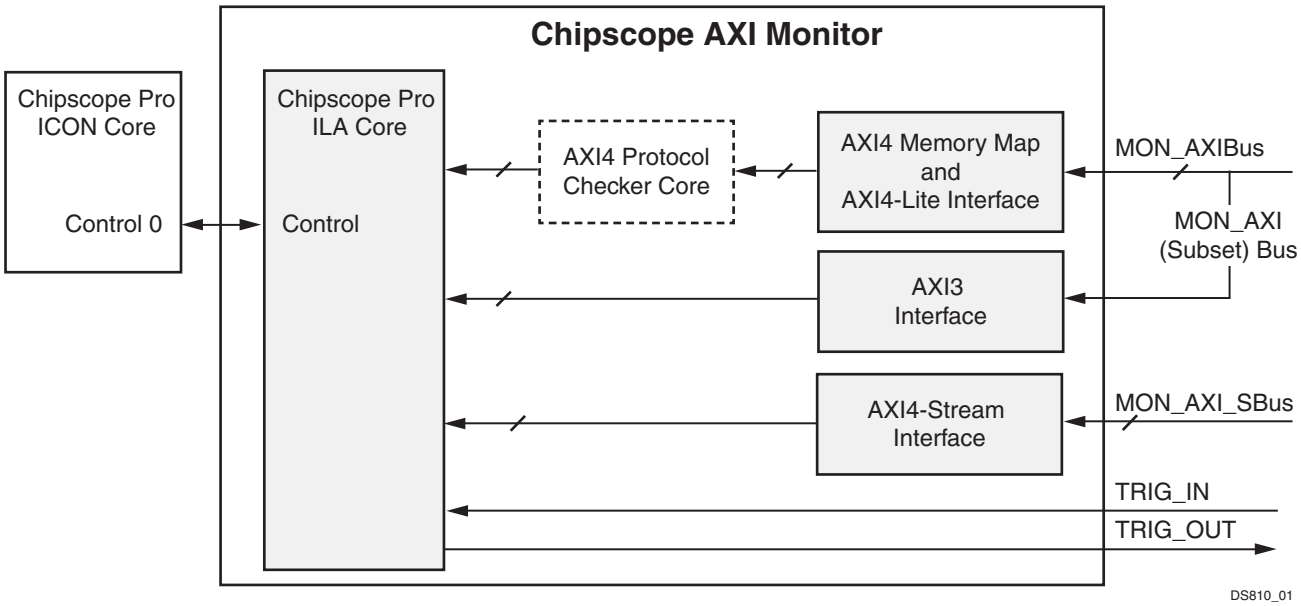

Figure 1: **ChipScope AXI Monitor Block Diagram**

The AXI Monitor is a wrapper for the ChipScope ILA core. It functions the same way as the ChipScope ILA, except that the wrapper creates a specific ILA for monitoring AXI signals by creating trigger groups designed for debugging purposes. When connected to a core in an AXI system, the user will specify which bus interface is to be connected (AXI4 Memory Map, AXI4-Streaming, AXI3), then supply values for parameters such as sample size and enable trigger out. These parameters are listed in Table 2.

After downloading the bitstream from the design to the FPGA, the ChipScope Analyzer Software tool is used to set up triggering and view waveforms from the system. The core generates a CDC file which is used by the ChipScope Analyzer tool to label the AXI signals with the appropriate header information and to define the trigger groups. More information on the ChipScope ILA or the ChipScope Analyzer can be found in the DS299 ChipScope ILA document and the UG029 ChipScope Analyzer document.

The AXI Monitor core is capable of using multiple AXI Monitors in a single AXI system to monitor multiple cores or can be used together by creating a trigger condition for the TRIG_OUT port from one monitor, then connecting it to the TRIG_IN port of another monitor. In XPS, when instantiating the AXI Monitor, connect the trigger out and trigger in pins on separate monitors, then use the ChipScope Analyzer to create the trigger condition on the specific monitor which will trigger the other to begin capturing data.

AXI Protocol Checker

The AXI Protocol Checker can be optionally included in the ChipScope AXI Monitor to check for AXI4-Memory Map and AXI4-Lite Protocol violations. The AXI Protocol Checker is designed around the ARM system verilog

assertions which have been converted into synthesizable HDL. The AXI Protocol Checker supplies a flag to the ILA which can be triggered on when a violation of the protocol is detected.

Upon enabling the AXI Protocol Checker, the user will be given a choice as to which types of protocol checks are to be monitored (See Note 1). See Table 3 for the complete list of protocol checks or see the AMBA® AXI4, AXI4-Lite, and AXI4-Stream Protocol Assertions User Guide. When the user enables a group of protocol checks, that group of flags will be OR'ed, then the OR'ed signal will be connected to a trigger port of the ChipScope ILA core. If the C_MON_AXI_PC_TRACE parameter is set to "Store," the actual error flags will be connected to the DATA port of the ILA and will be able to be monitored. It is important to note that the user must trace the AXI signals that correspond to the protocol check to determine where a protocol violation occurs. Depending on the protocol check, the latency of the violation flag triggering data capture, can occur from 1 to 3 clock cycles after the violation has occurred. It is necessary to position the violation flag trigger inside the capture window to more than 3 samples in the position entry to ensure that the AXI protocol violation will be displayed.

1. For the ChipScope AXI Monitor core, you can enable the Chks only.

I/O Signals

The core I/O signals are listed and described in the subsequent table.

Table 1: **I/O Signal Description**

Signal Name	Interface	Signal Type	Init Status	Description
AXI4 Memory Map Signals				
CHIPSCOPE_ICON_CONTROL(35:0)	N/A	I/O		Control bus connection to the ICON core. Mandatory. ***Note:*** For XPS designs, the direction of this port is IN.
MON_AXI_TRIG_OUT	N/A	O		Trigger output port. (Optional)
MON_AXI_TRIG_IN	N/A	I		Trigger input port. (Optional)
MON_AXI_ACLK	AXI4	I		Clock
MON_AXI_ARESETN (1)	AXI4	I		Reset (active low)
MON_AXI_AWID(C_MON_AXI_ID_WIDTH-1:0)	AXI4	I		Write address channel transaction ID
MON_AXI_AWADDR(C_MON_AXI_ADDR_WIDTH-1:0)	AXI4	I		Write address channel address
MON_AXI_AWLEN(7:0)	AXI4	I		Write address burst length: Gives the exact number of transfers in a burst
MON_AXI_AWSIZE(2:0)	AXI4	I		Write address burst size: Indicates the size of each transfer in the burst
MON_AXI_AWBURST(1:0)	AXI4	I		Write address burst type
MON_AXI_AWLOCK	AXI4	I		Write address lock type
MON_AXI_AWCACHE(3:0)	AXI4	I		Write address cache type
MON_AXI_AWPROT(2:0)	AXI4	I		Write address protection type
MON_AXI_AWQOS(3:0)	AXI4	I		Write address channel quality of service
MON_AXI_AWREGION(3:0)	AXI4	I		Selects address range within multirange slave
MON_AXI_AWVALID	AXI4	I		Write address valid: Indicates a valid write address and control information is available

Table 1: **I/O Signal Description *(Cont'd)***

Signal Name	Interface	Signal Type	Init Status	Description
MON_AXI_AWREADY	AXI4	I		Write address ready: slave is ready to accept address and control information
MON_AXI_AWUSER	AXI4	I		Write address channel USER signals
MON_AXI_WID(C_MON_AXI_ID_WIDTH)-1:0)	AXI3	I		Write data channel transaction ID
MON_AXI_WDATA(C_MON_AXI_DATA_WIDTH-1:0)	AXI4	I		Write data bus
MON_AXI_WSTRB(C_MON_AXI_DATA_WIDTH/8)-1:0)	AXI4	I		Write strobes: Indicates which byte lanes have valid data. MON_AXI_WSTRB[n] corresponds to MON_AXI_WDATA[(8xn)]+7:(8xn)]
MON_AXI_WLAST	AXI4	I		Indicates last write data word
MON_AXI_WVALID	AXI4	I		Write valid: Indicated valid write data and strobes are available. 1 = write data and strobes available 0 = write data and strobes not available
MON_AXI_WREADY	AXI4	I		Write ready: Indicates the slave can accept the write data 1= slave ready 0 = slave not ready
MON_AXI_WUSER	AXI4	I		Write data channel USER signals
MON_AXI_BID(C_MON_AXI_ID_WIDTH-1:0)	AXI4	I		Write response channel ID
MON_AXI_BRESP(1:0)	AXI4	I		Write response: Indicates the status of the write transaction
MON_AXI_BVALID	AXI4	I		Write response valid: Indicates a valid write response is available 1= write response available 0 = write response not available
MON_AXI_BREADY	AXI4	I		Write response ready: Indicates the master can accept the response information 1 = master ready 0 = master not ready
MON_AXI_BUSER	AXI4	I		Write response channel USER signals
MON_AXI_ARID(C_MON_AXI_ID_WIDTH-1:0)	AXI4	I		Read address ID
MON_AXI_ARADDR(C_MON_AXI_ADDR_WIDTH-1:0)	AXI4	I		Read address bus
MON_AXI_ARLEN(7:0)	AXI4	I		Read address burst length: Gives the exact number of transfers in a burst
MON_AXI_ARSIZE(2:0)	AXI4	I		Read address burst size: Indicates the size of each transfer in the burst
MON_AXI_ARBURST(1:0)	AXI4	I		Read address burst type
MON_AXI_ARLOCK	AXI4	I		Read address lock type
MON_AXI_ARCACHE(3:0)	AXI4	I		Read address cache type
MON_AXI_ARPROT(2:0)	AXI4	I		Read address protection type

Table 1: **I/O Signal Description *(Cont'd)***

Signal Name	Interface	Signal Type	Init Status	Description
MON_AXI_ARQOS(3:0)	AXI4	I		Read address channel quality of service
MON_AXI_ARREGION(3:0)	AXI4	I		Selects address range within multirange slave
MON_AXI_ARVALID	AXI4	I		Read address valid: When HIGH this signal indicates the read address and control information is valid and will remain valid until MON_AXI_ARREADY is HIGH 1 = Address and control information valid 0= Address and control information not valid
MON_AXI_ARREADY	AXI4	I		Address ready: Indicates the slave is ready to accept an address and associated control signals
MON_AXI_ARUSER	AXI4	I		Read address channel USER signals
MON_AXI_RID(C_MON_AXI_ID_WIDTH-1:0)	AXI4	I		Read data ID
MON_AXI_RDATA(C_MON_AXI_DATA_WIDTH-1:0)	AXI4	I		Read data bus
MON_AXI_RRESP(1:0)	AXI4	I		Read response: Indicates the status of the read transaction.
MON_AXI_RLAST	AXI4	I		Read last: Indicates last read data word
MON_AXI_RVALID	AXI4	I		Read data valid: Indicates the read data is available and the read transfer can complete 1 = read data available 0 = read data not available
MON_AXI_RREADY	AXI4	I		Read ready: Indicates the master can accept the read data and response information 1 = master ready 0 = master not ready
MON_AXI_RUSER	AXI4	I		Read data channel USER signals
AXI4-Stream Signals				
MON_AXI_S_TVALID	AXI4-Stream	I		AXI4-Stream valid
MON_AXI_S_TREADY	AXI4-Stream	I		AXI4-Stream ready
MON_AXI_S_TDATA(C_MON_AXI_S_TDATA_WIDTH-1:0)	AXI4-Stream	I		AXI4-Stream data bus
MON_AXI_S_TKEEP(C_MON_AXI_S_TDATA_WIDTH/8-1:0)	AXI4-Stream	I		AXI4-Stream byte qualifier
MON_AXI_S_TLAST	AXI4-Stream	I		AXI4-Stream last word
MON_AXI_S_TID(C_MON_AXI_S_TID_WIDTH-1:0)	AXI4-Stream	I		AXI4-Stream ID
MON_AXI_S_TDEST(C_MON_AXI_S_TDEST_WIDTH-1:0)	AXI4-Stream	I		AXI-Stream destination
MON_AXI_S_TUSER(C_MON_AXI_S_TUSER_WIDTH-1:0)	AXI4-Stream	I		AXI-Stream user data

Notes:

1. If the ChipScope AXI Monitor is connected to a core that does not have the AXI4 ARESETN signal, the signal is shown as a constant 0 in the Chipscope Analyzer, but otherwise does not assert reset or affect the behavior of the ChipScope AXI Monitor. If an auxiliary reset signal is needed, manually connect this signal to the Trigger In port of the ChipScope AXI Monitor.

Design Parameters

The core design parameters are listed and described in the subsequent table.

Table 2: **Design Parameters**

Feature Description	Parameter Name	Allowable Values	Default Values	Type
User Specified AXI Implemented Parameters				
Active bus interface type	C_USE_INTERFACE	(0: AXI4/AXI4-Lite, 1: AXI4-Stream, 2: AXI3 Memory map)	0	integer
Sets number of data samples	C_NUM_DATA_SAMPLES	(1024, 2048, 4096, 8192, 16384, 32768, 65536, 131072)	1048	integer
Maximum number of sequencer levels	C_MAX_SEQUENCER_LEVELS	2	(1:16)	integer
Enable trigger in	C_USE_TRIG_IN	(0,1)	0	integer
Trigger input width	C_TRIG_IN_WIDTH	(1:255)	1	integer
AWLEN/ARLEN Bus Width	C_MON_AXI_BURST_LENGTH	(4,8)	8	integer
AWLOCK/ARLOCK Bus Width	C_MON_AXI_LOCK_LENGTH	(1,2)	1	integer
ARADDR Number of Match Units	C_MON_AXI_ARADDR_NUM_OF_MATCH	(0:4)	1	integer
ARADDRCONTROL Number of Match Units	C_MON_AXI_ARADDRCONTROL_NUM_OF_MATCH	(0:4)	1	integer
AWADDR Number of Match Units	C_MON_AXI_AWADDR_NUM_OF_MATCH	(0:4)	1	integer
AWADDRCONTROL Number of Match Units	C_MON_AXI_AWADDRCONTROL_NUM_OF_MATCH	(0:4)	1	integer
BRESP Number of Match Units	C_MON_AXI_BRESP_NUM_OF_MATCH	(0:4)	1	integer
GLOBAL Number of Match Units	C_MON_AXI_GLOBAL_NUM_OF_MATCH	(0:4)	1	integer
RDATA Number of Match Units	C_MON_AXI_RDATA_NUM_OF_MATCH	(0:4)	1	integer
RDATACONTROL Number of Match Units	C_MON_AXI_RDATACONTROL_NUM_OF_MATCH	(0:4)	1	integer
WDATA Number of Match Units	C_MON_AXI_WDATA_NUM_OF_MATCH	(0:4)	1	integer
WDATACONTROL Number of Match Units	C_MON_AXI_WDATACONTROL_NUM_OF_MATCH	(0:4)	1	integer
AWUSER Number of Match Units	C_MON_AXI_AWUSER_NUM_OF_MATCH	(0:4)	0	integer
WUSER Number of Match Units	C_MON_AXI_WUSER_NUM_OF_MATCH	(0:4)	0	integer
BUSER Number of Match Units	C_MON_AXI_BUSER_NUM_OF_MATCH	(0:4)	0	integer
ARUSER Number of Match Units	C_MON_AXI_ARUSER_NUM_OF_MATCH	(0:4)	0	integer

Table 2: **Design Parameters *(Cont'd)***

Feature Description	Parameter Name	Allowable Values	Default Values	Type
RUSER Number of Match Units	C_MON_AXI_RUSER_NUM_OF_MATCH	(0:4)	0	integer
TCONTROL Number of Match Units	C_MON_AXI_S_TCONTROL_NUM_OF_MATCH	(0:4)	0	integer
TDATA Number of Match Units	C_MON_AXI_S_TDATA_NUM_OF_MATCH	(0:4)	0	integer
TUSER Number of Match Units	C_MON_AXI_S_TUSER_NUM_OF_MATCH	(0:4)	0	integer
ARADDR Match Type	C_MON_AXI_ARADDR_MATCH_TYPE	(basic=basic, basic with edges=basic with edges, extended=extended, extended with edges=extended with edges, range=range, range with edges=range with edges)	range with edges	string
ARADDRCONTROL Match Type	C_MON_AXI_ARADDRCONTROL_MATCH_TYPE	(basic=basic, basic with edges=basic with edges, extended=extended, extended with edges=extended with edges, range=range, range with edges=range with edges)	basic with edges	string
AWADDR Match Type	C_MON_AXI_AWADDR_MATCH_TYPE	(basic=basic, basic with edges=basic with edges, extended=extended, extended with edges=extended with edges, range=range, range with edges=range with edges)	range with edges	string
AWADDRCONTROL Match Type	C_MON_AXI_AWADDRCONTROL_MATCH_TYPE	(basic=basic, basic with edges=basic with edges, extended=extended, extended with edges=extended with edges, range=range, range with edges=range with edges)	basic with edges	string
BRESP Match Type	C_MON_AXI_BRESP_MATCH_TYPE	(basic=basic, basic with edges=basic with edges, extended=extended, extended with edges=extended with edges, range=range, range with edges=range with edges)	basic with edges	string

Table 2: **Design Parameters *(Cont'd)***

Feature Description	Parameter Name	Allowable Values	Default Values	Type
GLOBAL Match Type	C_MON_AXI_GLOBAL_MATCH_TYPE	(basic=basic, basic with edges=basic with edges, extended=extended, extended with edges=extended with edges, range=range, range with edges=range with edges)	basic with edges	string
RDATA Match Type	C_MON_AXI_RDATA_MATCH_TYPE	(basic=basic, basic with edges=basic with edges, extended=extended, extended with edges=extended with edges, range=range, range with edges=range with edges)	extended with edges	string
RDATACONTROL Match Type	C_MON_AXI_RDATACONTROL_MATCH_TYPE	(basic=basic, basic with edges=basic with edges, extended=extended, extended with edges=extended with edges, range=range, range with edges=range with edges)	basic with edges	string
WDATA Match Type	C_MON_AXI_WDATA_MATCH_TYPE	(basic=basic, basic with edges=basic with edges, extended=extended, extended with edges=extended with edges, range=range, range with edges=range with edges)	extended with edges	string
WDATACONTROL Match Type	C_MON_AXI_WDATACONTROL_MATCH_TYPE	(basic=basic, basic with edges=basic with edges, extended=extended, extended with edges=extended with edges, range=range, range with edges=range with edges)	basic with edges	string
AWUSER Match Type	C_MON_AXI_AWUSER_MATCH_TYPE	(basic=basic, basic with edges=basic with edges, extended=extended, extended with edges=extended with edges, range=range, range with edges=range with edges)	basic with edges	string
WUSER Match Type	C_MON_AXI_WUSER_MATCH_TYPE	(basic=basic, basic with edges=basic with edges, extended=extended, extended with edges=extended with edges, range=range, range with edges=range with edges)	basic with edges	string

Table 2: **Design Parameters *(Cont'd)***

Feature Description	Parameter Name	Allowable Values	Default Values	Type
BUSER Match Type	C_MON_AXI_BUSER_MATCH_TYPE	(basic=basic, basic with edges=basic with edges, extended=extended, extended with edges=extended with edges, range=range, range with edges=range with edges)	basic with edges	string
ARUSER Match Type	C_MON_AXI_ARUSER_MATCH_TYPE	(basic=basic, basic with edges=basic with edges, extended=extended, extended with edges=extended with edges, range=range, range with edges=range with edges)	basic with edges	string
RUSER Match Type	C_MON_AXI_RUSER_MATCH_TYPE	(basic=basic, basic with edges=basic with edges, extended=extended, extended with edges=extended with edges, range=range, range with edges=range with edges)	basic with edges	string
TCONTROL Match Type	C_MON_AXI_S_TCONTROL_MATCH_TYPE	(basic=basic, basic with edges=basic with edges, extended=extended, extended with edges=extended with edges, range=range, range with edges=range with edges)	basic with edges	string
TDATA Match Type	C_MON_AXI_S_TDATA_MATCH_TYPE	(basic=basic, basic with edges=basic with edges, extended=extended, extended with edges=extended with edges, range=range, range with edges=range with edges)	basic with edges	string
TUSER Match Type	C_MON_AXI_S_TUSER_MATCH_TYPE	(basic=basic, basic with edges=basic with edges, extended=extended, extended with edges=extended with edges, range=range, range with edges=range with edges)	basic with edges	string
ARADDR Trigger Counter Width	C_MON_AXI_ARADDR_TRIG_COUNT_WIDTH	(1:32)	1	integer
ARADDRCONTROL Trigger Counter Width	C_MON_AXI_ARADDRCONTROL_TRIG_COUNT_WIDTH	(1:32)	1	integer
AWADDR Trigger Counter Width	C_MON_AXI_AWADDR_TRIG_COUNT_WIDTH	(1:32)	1	integer

Table 2: **Design Parameters *(Cont'd)***

Feature Description	Parameter Name	Allowable Values	Default Values	Type
AWADDRCONTROL Trigger Counter Width	C_MON_AXI_AWADDRCONTROL _TRIG_COUNT_WIDTH	(1:32)	1	integer
BRESP Trigger Counter Width	C_MON_AXI_BRESP_TRIG_ COUNT_WIDTH	(1:32)	1	integer
GLOBAL Trigger Counter Width	C_MON_AXI_GLOBAL_TRIG_ COUNT_WIDTH	(1:32)	1	integer
RDATA Trigger Counter Width	C_MON_AXI_RDATA_TRIG_ COUNT_WIDTH	(1:32)	1	integer
RDATACONTROL Trigger Counter Width	C_MON_AXI_RDATACONTROL_ TRIG_COUNT_WIDTH	(1:32)	1	integer
WDATA Trigger Counter Width	C_MON_AXI_WDATA_TRIG_ COUNT_WIDTH	(1:32)	1	integer
WDATACONTROL Trigger Counter Width	C_MON_AXI_WDATACONTROL_ TRIG_COUNT_WIDTH	(1:32)	1	integer
AWUSER Trigger Counter Width	C_MON_AXI_AWUSER_TRIG_ COUNT_WIDTH	(1:32)	1	integer
WUSER Trigger Counter Width	C_MON_AXI_WUSER_TRIG_ COUNT_WIDTH	(1:32)	1	integer
BUSER Trigger Counter Width	C_MON_AXI_BUSER_TRIG_ COUNT_WIDTH	(1:32)	1	integer
ARUSER Trigger Counter Width	C_MON_AXI_ARUSER_TRIG_ COUNT_WIDTH	(1:32)	1	integer
RUSER Trigger Counter Width	C_MON_AXI_RUSER_TRIG_ COUNT_WIDTH	(1:32)	1	integer
TCONTROL Trigger Counter Width	C_MON_AXI_S_TCONTROL_ TRIG_COUNT_WIDTH	(1:32)	1	integer
TDATA Trigger Counter Width	C_MON_AXI_S_TDATA_TRIG_ COUNT_WIDTH	(1:32)	1	integer
TUSER Trigger Counter Width	C_MON_AXI_S_TUSER_TRIG_C OUNT_WIDTH	(1:32)	1	integer
GLOBAL Trigger Store/Trace to ILA Data Port	C_MON_AXI_GLOBAL_TRACE	(0: Do Not Store, 1: Store)	1	string
AWADDRCONTROL Trigger Store/Trace to ILA Data Port	C_MON_AXI_AWADDRCONTROL _TRACE	(0: Do Not Store, 1: Store)	1	string
AWADDR Trigger Store/Trace to ILA Data Port	C_MON_AXI_AWADDR_TRACE	(0: Do Not Store, 1: Store)	1	string
WDATACONTROL Trigger Store/Trace to ILA Data Port	C_MON_AXI_WDATACONTROL_ TRACE	(0: Do Not Store, 1: Store)	1	string
WDATA Trigger Store/Trace to ILA Data Port	C_MON_AXI_WDATA_TRACE	(0: Do Not Store, 1: Store)	1	string
BRESP Trigger Store/Trace to ILA Data Port	C_MON_AXI_BRESP_TRACE	(0: Do Not Store, 1: Store)	1	string
ARADDRCONTROL Trigger Store/Trace to ILA Data Port	C_MON_AXI_ARADDRCONTROL _TRACE	(0: Do Not Store, 1: Store)	1	string
ARADDR Trigger Store/Trace to ILA Data Port	C_MON_AXI_ARADDR_TRACE	(0: Do Not Store, 1: Store)	1	string

Table 2: **Design Parameters *(Cont'd)***

Feature Description	Parameter Name	Allowable Values	Default Values	Type
RDATACONTROL Trigger Store/Trace to ILA Data Port	C_MON_AXI_RDATACONTROL_ TRACE	(0: Do Not Store, 1: Store)	1	string
RDATA Trigger Store/Trace to ILA Data Port	C_MON_AXI_RDATA_TRACE	(0: Do Not Store, 1: Store)	1	string
AWUSER Trigger Store/Trace to ILA Data Port	C_MON_AXI_AWUSER_TRACE	(0: Do Not Store, 1: Store)	0	string
WUSER Trigger Store/Trace to ILA Data Port	C_MON_AXI_WUSER_TRACE	(0: Do Not Store, 1: Store)	0	string
BUSER Trigger Store/Trace to ILA Data Port	C_MON_AXI_BUSER_TRACE	(0: Do Not Store, 1: Store)	0	string
ARUSER Trigger Store/Trace to ILA Data Port	C_MON_AXI_ARUSER_TRACE	(0: Do Not Store, 1: Store)	0	string
RUSER Trigger Store/Trace to ILA Data Port	C_MON_AXI_RUSER_TRACE	(0: Do Not Store, 1: Store)	0	string
TCONTROL Trigger Store/Trace to ILA Data Port	C_MON_AXI_S_TCONTROL_ TRACE	(0: Do Not Store, 1: Store)	1	string
TDATA Trigger Store/Trace to ILA Data Port	C_MON_AXI_S_TDATA_TRACE	(0: Do Not Store, 1: Store)	1	string
TUSER Trigger Store/Trace to ILA Data Port	C_MON_AXI_S_TUSER_TRACE	(0: Do Not Store, 1: Store)	0	string
System Specified AXI Implemented Parameters				
Device family	C_FAMILY	virtex6, spartan6	spartan6	string
Device/part	C_DEVICE		xc6slx45t	string
Device package	C_PACKAGE		fgg484	string
Device speed grade	C_SPEEDGRADE	(-1, -2, -3)	-2	string
AXI memory map BUSIF name	C_MON_AXI_PROTOCOL	axi4, axi4-lite, axi3	axi4	string
AXI stream BUSIF name	C_MON_AXI_S_PROTOCOL		generic	string
System supports threads	C_MON_AXI_SUPPORTS_ THREADS	(0,1)	0	integer
AXI memory map ID width	C_MON_AXI_ID_WIDTH		1	integer
AXI stream ID width	C_MON_AXI_S_TID_WIDTH		1	integer
AXI memory map address width	C_MON_AXI_ADDR_WIDTH	32	32	integer
AXI stream TDEST width	C_MON_AXI_S_TDEST_WIDTH	(0:32)	32	integer
AXI4 memory map data width	C_MON_AXI_DATA_WIDTH	(32, 64, 128, 256)	32	integer
AXI stream data width	C_MON_AXI_S_TDATA_WIDTH	(32, 64, 128, 256)	32	integer
Write address USER port bus width	C_MON_AXI_AWUSER_WIDTH	(1:256)[(1)]	1	integer
Write data USER port bus width	C_MON_AXI_WUSER_WIDTH	(1:256)[(1)]	1	integer
Write response USER port bus width	C_MON_AXI_BUSER_WIDTH	(1:256)[(1)]	1	integer

Table 2: **Design Parameters *(Cont'd)***

Feature Description	Parameter Name	Allowable Values	Default Values	Type
Read address USER port bus width	C_MON_AXI_ARUSER_WIDTH	(1:256)[(1)]	1	integer
Read data USER port bus width	C_MON_AXI_RUSER_WIDTH	(1:256)[(1)]	1	integer
Streaming USER port bus width	C_MON_AXI_S_TUSER_WIDTH	(1:256)[(1)]	1	integer
System supports read operations	C_MON_AXI_SUPPORTS_READ	(0,1)	1	integer
System supports write operations	C_MON_AXI_SUPPORTS_WRITE	(0,1)	1	integer
AXI Protocol Checker Parameters				
Protocol Checker Number of Match Units	C_MON_AXI_PC_NUM_OF_ MATCH	(0:1)	0	integer
Protocol Checker Match Type	C_MON_AXI_PC_MATCH_TYPE	(basic=basic, basic with edges=basic with edges, extended=extended, extended with edges=extended with edges, range=range, range with edges=range with edges)	basic with edges	string
Protocol Checker Trigger Counter Width	C_MON_AXI_PC_TRIG_COUNT_ WIDTH	(1:32)	1	integer
Protocol Checker Trigger Store/Trace to ILA Data Port	C_MON_AXI_PC_TRACE	(0: Do Not Store, 1: Store)	0	string
Handshake Protocol Checks Enable	C_MON_AXI_EN_HANDSHAKE_ CHECKS	(0,1)	0	integer
Complex Protocol Checks Enable	C_MON_AXI_EN_COMPLEX_ CHECKS	(0,1)	0	integer
Exclusive Access Protocol Checks Enable	C_MON_AXI_EN_EXCLUSIVE_ CHECKS	(0,1)	0	integer
Ignore Protocol Checks Enable	C_MON_AXI_EN_IGNORE_ CHECKS	(0,1)	0	integer
Illegal Value Protocol Checks Enable	C_MON_AXI_EN_ILLEGAL_ VALUE_CHECKS	(0,1)	0	integer
Reset Protocol Checks Enable	C_MON_AXI_EN_RESET_CHECK S	(0,1)	0	integer
Auxiliary Protocol Checks Enable	C_MON_AXI_EN_AUX_CHECKS	(0,1)	0	integer
ARM Recommended Max Wait Protocol Checks Enable	C_MON_AXI_EN_ARM_REC_ WAIT_CHECKS	(0,1)	0	integer
ARM Recommended Protocol Checks Enable	C_MON_AXI_EN_ARM_REC_ ONLY_CHECKS	(0,1)	0	integer
Protocol Checker Error Bus Width	C_MON_AXI_PC_ERROR_ COUNT	95	95	integer
Size of FIFO for Storing Outstanding Read Bursts	C_MON_AXI_PC_MAXRBURST	(2:32)	16	integer

Table 2: **Design Parameters *(Cont'd)***

Feature Description	Parameter Name	Allowable Values	Default Values	Type
Size of FIFO for Storing Outstanding Read Bursts	C_MON_AXI_PC_MAXWBURST	(2:32)	16	integer
Width of Exclusive Access Monitor	C_MON_AXI_PC_EXMON_WIDTH	(1:32)	1	integer

Notes:

1. These parameters are set by the XPS tool automatically after the ChipScope AXI Monitor is connected to another Xilinx AXI IP core. If any port(s) for which these parameters set the width are not used, retain the default widths and the XPS tool will ground the port(s). Do not enter a value of 0 in the MHS file to disable the default value.

AXI Protocol Checks and Descriptions

The AXI Protocol Checks are listed and described in the subsequent table.

Table 3: **AXI Protocol Checks and Descriptions**

Name of Protocol Check	Type	Description
AXI4_ERRM_AWADDR_ STABLE	Ready/Valid Handshake Checks	AWADDR must remain stable when AWVALID is asserted and AWREADY low
AXI4_ERRM_AWBURST_ STABLE	Ready/Valid Handshake Checks	AWBURST must remain stable when AWVALID is asserted and AWREADY low
AXI4_ERRM_AWCACHE _STABLE	Ready/Valid Handshake Checks	AWCACHE must remain stable when AWVALID is asserted and AWREADY low
AXI4_ERRM_AWID_ STABLE	Ready/Valid Handshake Checks	AWID must remain stable when AWVALID is asserted and AWREADY low
AXI4_ERRM_AWLEN_ STABLE	Ready/Valid Handshake Checks	AWLEN must remain stable when AWVALID is asserted and AWREADY low
AXI4_ERRM_AWLOCK_ STABLE	Ready/Valid Handshake Checks	AWLOCK must remain stable when AWVALID is asserted and AWREADY low
AXI4_ERRM_AWPROT_S TABLE	Ready/Valid Handshake Checks	AWPROT must remain stable when AWVALID is asserted and AWREADY low
AXI4_ERRM_AWSIZE_ STABLE	Ready/Valid Handshake Checks	AWSIZE must remain stable when AWVALID is asserted and AWREADY low
AXI4_ERRM_AWQOS_ STABLE	Ready/Valid Handshake Checks	AWQOS must remain stable when AWVALID is asserted and AWREADY low
AXI4_ERRM_AWREGION _STABLE	Ready/Valid Handshake Checks	AWREGION must remain stable when ARVALID is asserted and AWREADY low
AXI4_ERRM_AWVALID_ STABLE	Ready/Valid Handshake Checks	Once AWVALID is asserted, it must remain asserted until AWREADY is high
AXI4_ERRM_WDATA_ STABLE	Ready/Valid Handshake Checks	WDATA must remain stable when WVALID is asserted and WREADY low
AXI4_ERRM_WLAST_ STABLE	Ready/Valid Handshake Checks	WLAST must remain stable when WVALID is asserted and WREADY low
AXI4_ERRM_WSTRB_ STABLE	Ready/Valid Handshake Checks	WSTRB must remain stable when WVALID is asserted and WREADY low
AXI4_ERRM_WVALID_ STABLE	Ready/Valid Handshake Checks	Once WVALID is asserted, it must remain asserted until WREADY is high
AXI4_ERRS_BID_ STABLE	Ready/Valid Handshake Checks	BID must remain stable when BVALID is asserted and BREADY low

Table 3: **AXI Protocol Checks and Descriptions *(Cont'd)***

Name of Protocol Check	Type	Description
AXI4_ERRS_BRESP_ STABLE	Ready/Valid Handshake Checks	BRESP must remain stable when BVALID is asserted and BREADY low
AXI4_ERRS_BVALID_ STABLE	Ready/Valid Handshake Checks	Once BVALID is asserted, it must remain asserted until BREADY is high
AXI4_ERRM_ARADDR_ STABLE	Ready/Valid Handshake Checks	ARADDR must remain stable when ARVALID is asserted and ARREADY low
AXI4_ERRM_ARBURST_ STABLE	Ready/Valid Handshake Checks	ARBURST must remain stable when ARVALID is asserted and ARREADY low
AXI4_ERRM_ARCACHE_ STABLE	Ready/Valid Handshake Checks	ARCACHE must remain stable when ARVALID is asserted and ARREADY low
AXI4_ERRM_ARID_ STABLE	Ready/Valid Handshake Checks	ARID must remain stable when ARVALID is asserted and ARREADY low
AXI4_ERRM_ARLEN_ STABLE	Ready/Valid Handshake Checks	ARLEN must remain stable when ARVALID is asserted and ARREADY low.
AXI4_ERRM_ARLOCK_ STABLE	Ready/Valid Handshake Checks	ARLOCK must remain stable when ARVALID is asserted and ARREADY low
AXI4_ERRM_ARPROT_ STABLE	Ready/Valid Handshake Checks	ARPROT must remain stable when ARVALID is asserted and ARREADY low
AXI4_ERRM_ARSIZE_ STABLE	Ready/Valid Handshake Checks	ARSIZE must remain stable when ARVALID is asserted and ARREADY low
AXI4_ERRM_ARQOS_ STABLE	Ready/Valid Handshake Checks	ARQOS must remain stable when ARVALID is asserted and ARREADY low
AXI4_ERRM_ARREGION _STABLE	Ready/Valid Handshake Checks	ARREGION must remain stable when ARVALID is asserted and ARREADY low
AXI4_ERRM_ARVALID_ STABLE	Ready/Valid Handshake Checks	Once ARVALID is asserted, it must remain asserted until ARREADY is high
AXI4_ERRS_RDATA_ STABLE	Ready/Valid Handshake Checks	RDATA must remain stable when RVALID is asserted and RREADY low
AXI4_ERRS_RID_ STABLE	Ready/Valid Handshake Checks	RID must remain stable when RVALID is asserted and RREADY low
AXI4_ERRS_RLAST_ STABLE	Ready/Valid Handshake Checks	RLAST must remain stable when RVALID is asserted and RREADY low
AXI4_ERRS_RRESP_ STABLE	Ready/Valid Handshake Checks	RRESP must remain stable when RVALID is asserted and RREADY low
AXI4_ERRS_RVALID_ STABLE	Ready/Valid Handshake Checks	Once RVALID is asserted, it must remain asserted until RREADY is high
AXI4_ERRM_AWUSER_ STABLE	Ready/Valid Handshake Checks and ARM Recommended Checks	AWUSER must remain stable when AWVALID is asserted and AWREADY low
AXI4_ERRM_WUSER_ STABLE	Ready/Valid Handshake Checks	WUSER must remain stable when WVALID is asserted and WREADY low
AXI4_ERRS_BUSER_ STABLE	Ready/Valid Handshake Checks	BUSER must remain stable when BVALID is asserted and BREADY low
AXI4_ERRM_ARUSER_ STABLE	Ready/Valid Handshake Checks	ARUSER must remain stable when ARVALID is asserted and ARREADY low
AXI4_ERRS_RUSER_ STABLE	Ready/Valid Handshake Checks	RUSER must remain stable when RVALID is asserted and RREADY low

Table 3: **AXI Protocol Checks and Descriptions *(Cont'd)***

Name of Protocol Check	Type	Description
AXI4_ERRM_AWADDR_BOUNDARY	Illegal Value Checks	A write burst cannot cross a 4KB boundary
AXI4_ERRM_AWADDR_WRAP_ALIGN	Illegal Value Checks	A write transaction with burst type WRAP has an aligned address
AXI4_ERRM_AWBURST	Illegal Value Checks	A value of 2'b11 on AWBURST is not permitted when AWVALID is HIGH
AXI4_ERRM_AWCACHE	Illegal Value Checks	If not cacheable (AWCACHE[1] == 1'b0), AWCACHE = 2'b00
AXI4_ERRM_AWLEN_FIXED	Illegal Value Checks	Transactions of burst type FIXED cannot have a length greater than 16 beats
AXI4_ERRM_AWLEN_WRAP	Illegal Value Checks	A write transaction with burst type WRAP has a length of 2, 4, 8, or 16
AXI4_ERRM_AWSIZE	Illegal Value Checks	The size of a write transfer does not exceed the width of the data interface
AXI4_ERRM_WSTRB	Illegal Value Checks	Write strobes must only be asserted for the correct byte lanes as determined from the: Start Address, Transfer Size and Beat Number
AXI4_ERRM_ARADDR_BOUNDARY	Illegal Value Checks	A read burst cannot cross a 4KB boundary
AXI4_ERRM_ARADDR_WRAP_ALIGN	Illegal Value Checks	A read transaction with a burst type of WRAP must have an aligned address
AXI4_ERRM_ARBURST	Illegal Value Checks	A value of 2'b11 on ARBURST is not permitted when ARVALID is HIGH
AXI4_ERRM_ARCACHE	Illegal Value Checks	When ARVALID is HIGH, if ARCACHE[1] is LOW, then ARCACHE[3:2] must also be LOW
AXI4_ERRM_ARLEN_FIXED	Illegal Value Checks	Transactions of burst type FIXED cannot have a length greater than 16 beats
AXI4_ERRM_ARLEN_WRAP	Illegal Value Checks	A read transaction with burst type of WRAP must have a length of 2, 4, 8, or 16
AXI4_ERRM_ARSIZE	Illegal Value Checks	The size of a read transfer must not exceed the width of the data interface

Getting Started Tutorial

The following is a tutorial guide to design a hardware system in Xilinx Platform Studio (XPS) containing the ChipScope AXI Monitor for capturing the signals of an AXI interface in the system. This tutorial includes basic steps to create a software application to run on the hardware system to generate AXI traffic for the Chipscope AXI Monitor to capture. It also explains how to use the ChipScope Pro Analyzer to configure the device, set up triggers, and view waveform results of the captured AXI signals.

Creating a Project

1. Launch XPS and create a new project by clicking **Create New Project Using Base System Builder**.
2. Name a project and select **ISE** for **Logic Tool** with **AXI System** as **Interconnect** type.
3. Select an evaluation board (i.e. Virtex-6 ML605 Evaluation Platform) from the drop-down menu.

   For this tutorial, select **Single MicroBlaze Processor System**.
4. For **Processor**, **Cache** and **Peripheral Configuration**, change **Local Memory Size**, **Instruction Cache Size**, and **Data Cache Size** to **16kB**.

5. Create a design with DDR3_SDRAM and RS232_Uart_1 (or any other peripherals of your need) and select **Finish**.

Instantiating the Chipscope AXI Monitor in XPS

There are two ways to instantiate the Chipscope AXI Monitor in XPS:

Option 1: Debug Wizard

1. Click **Debug > Debug Configuration**.
2. Click **Add ChipScope Peripheral** button on bottom-left corner and select **To** monitor **AXI Interconnect** signals (adding AXI Monitor) and click **OK**.
3. In **Monitor Bus Signals**, select **AXI Interface** to monitor from the drop-down menu. For this tutorial, select **DDR3_SDRAM.S_AXI**.
4. Select the number of signal samples you want to collect (i.e. **1024**) and click **OK**.

Option 2: System Assembly View:

1. From the **IP Catalog** window, add **Chipscope AXI Monitor** (under **Debug**) and make the appropriate selections for **AXI Settings** (i.e. AXI4 Memory Map), **ILA Settings**, **Protocol Checker Settings**, and so on.
2. From the **IP Catalog** window, add **Chipscope ICON** (under **Debug**) and make the appropriate settings.

   In **System Assembly View** and under **Bus Interfaces** tab, notice the two IPs specified (**Chipscope AXI Monitor** and **Chipscope ICON**) are listed now.
3. Expand **chipscope_axi_monitor_0** instance and notice that it now shows the **MON_AXI** interface for it. Connect MON_AXI interface to DDR3_SDRAM.S_AXI interface (in the drop-down menu under **Bus Name** column).
4. Clickthe **Ports** tab, expand **chipscope_axi_monitor_0 instance**, connect CHIPSCOPE_ICON_CONTROL to chipscope_icon_0 using the drop-down menu under **Connected Port** column.
   - Bitstream Generation: **Click Hardware > Generate Bitstream**.
   - Download bitstream: Once bitstream generation is finished, click **Device Configuration > Download bitstream** to create a download.bit file in the `<proj_dir>/implementation/` directory.

Exporting the Hardware Design to SDK

1. Export the hardware design to SDK by:
   - Clicking **Export Design** in the **Navigator** menu

     or
   - Clicking **Export Hardware Design to SDK** under the **Project** tab and then clicking **Export & Launch SDK**.
2. Select a workspace: Create a workspace **`<proj_dir>/sw`** to save the SDK files.
3. Xilinx Board Support Package: Create Standalone Xilinx Board Support Package by clicking **File > New > Xilinx Board Support Package** and then **Finish**.
4. Xilinx C Project for memory test: Create Xilinx C Project by clicking **File > New > Xilinx C Project**.
5. Select **Memory Tests** under **Select Project Template** and click **Next**.
6. Select **Target an existing Board Support Package** which is the same standalone board you just created and click **Finish** to create a `memory_tests_0.elf` file in `<proj_dir>/sw/memory_tests_0/Debug/`.

Using Chipscope Pro Analyzer and Xilinx Multiprocessor Debugger (XMD):

1. Launch Chipscope Pro Analyzer.
2. Connect to the board by clicking the icon (below **File** tab) on top-left corner.

   Once you have connected to the board, the **Poll** button (right to the connect board) turns green.

3. Configure device: Right-click on **MyDevice1** and click **Configure**. Click **Select New File** for **JTAG Configuration** and import the **`system.bit`** file that you generated in XPS. Do not import the **`.cdc`** file without programming the board. Click **OK** to program the board.
4. CDC file import: **Click File > Import > Select New File**, `<proj_dir>/implementation/chipscope_axi_monitor_0_wrapper/chipscope_axi_monitor_0.cdc`.
5. Trigger Setup: Double-click (or right-click) **Trigger Setup** in upper-right corner of the project window.
   - Select and expand M1:MON_AXI_ARADDRCONTROL in "Match Unit" column.
   - Select the field in the very first row, **DDR_SDRAM.S_AXI/MON_AXI_ARVALID**.
   - Set **Function** to **==** and **Value** to **1**.
   - You can set **Output Enable** to either **Pulse** or **Level** signal (active high or low) in **Trig** in the **Trigger Setup**" window to create a trigger for any read transaction (ARVALID == 1′b1) on axi_v6_ddrx.
6. Monitor Trigger: Click the **Play** symbol button located above the **Trigger Setup** box to apply settings and arm the trigger.
7. Disconnect the cable by clicking green **Poll** button. It now turns red.

Xilinx Multiprocessor Debugger

1. Launch **Xilinx ISE Design Suite 13.x > Accessories > ISE Design Suite Command Promt**.
2. Run following commands to load your .elf (which you generated in SDK) and run on Microblaze system:
   - %xmd –nx
   - XMD%mbc
   - XMD%dow C:/Users/ChipScope_AXI_Monitor_Proj/memory_tests_0.elf
   - XMD%run
   - XMD%exit
3. Return to the **ChipScope Pro Analyzer** window and connect the cable by clicking the red **Poll** button. The button then will then turn green.
4. Open Waveform by right-clicking **Waveform** in **New Project** window to view your triggers and other signals.

Device Utilization and Performance Benchmarks

The device utilization will vary based on the parameter combinations set by the system and user.

Configuration

The core configuration details are listed and described in the subsequent table.

Table 4: **Configuration Details**

Configuration	Device	System Setup
1	xc6slx45t-fgg484-3	ChipScope AXI Monitor enabled; AXI Protocol Checker disabled; AXI4 Memory Map; 10 trigger groups enabled; Data Width of 32; Sample depth of 1024
2	xc6slx45t-fgg484-3	ChipScope AXI Monitor enabled; AXI Protocol Checker with 50% of checks enabled; AXI4 Memory Map; 10 trigger groups enabled; Data Width of 32; Sample depth of 1024
3	xc6slx45t-fgg484-3	ChipScope AXI Monitor enabled; AXI Protocol Checker with 100% of checks enabled; AXI4 Memory Map; 10 trigger groups enabled; Data Width of 32; Sample depth of 1024

Table 4: **Configuration Details**

Configuration	Device	System Setup
4	xc6vlx240t-ff1156-1	ChipScope AXI Monitor enabled; AXI Protocol Checker disabled; AXI4 Memory Map; 10 trigger groups enabled; Data Width of 32; Sample depth of 1024
5	xc6vlx240t-ff1156-1	ChipScope AXI Monitor enabled; AXI Protocol Checker with 50% of checks enabled; AXI4 Memory Map; 10 trigger groups enabled; Data Width of 32; Sample depth of 1024
6	xc6vlx240t-ff1156-1	ChipScope AXI Monitor enabled; AXI Protocol Checker with 100% of checks enabled; AXI4 Memory Map; 10 trigger groups enabled; Data Width of 32; Sample depth of 1024

References

- More information on the ChipScope Pro software and cores is available in the *Software and Cores User Guide*, located at http://www.xilinx.com/support/documentation.
- Information about hardware debugging using ChipScope Pro in EDK is available in the Platform Studio 12 online help, located at http://www.xilinx.com/support/documentation.
- Information about hardware debugging using ChipScope Pro in System Generator for DSP is available in the *Xilinx System Generator for DSP User Guide*, located at http://www.xilinx.com/support/documentation.

Support

Xilinx provides technical 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*.

CORE Generator Support

A stand alone core to use in non-XPS systems, Chipscope AXI monitor IP can be generated from the CORE Generator tool. CORE Generator supports all the parameters as supported by XPS. While generating Chipscope AXI monitor from coregen following files are generated:

- Fully synthesized netlist
- UCF file for ILA level timing constraints
- CDC file for specific configuration
- A dummy top level synthesis model file
- Datasheet and readme documents

Ordering Information

This Xilinx LogiCORE IP module is provided at no additional cost with the Xilinx ISE® Design Suite Embedded Edition software under the terms of the Xilinx End User License. The core is generated using the Xilinx ISE Embedded Edition software (EDK).

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 modules and software, please contact your local Xilinx sales representative.

Revision History

The following table shows the revision history for this document:

Date	Version	Description of Revisions
9/21/10	1.0	Initial Xilinx release
12/14/10	1.1	Added functionality to support AXI4-Lite and AXI4-Stream signals
3/1/11	1.2	Updated to v2.00a for 13.1 release; added functionality for increased user customization
6/22/11	1.3	Updated to v3.00a for 13.2 release; added AXI Protocol Checker and AXI3 support
10/19/11	1.4	Updated to v3.01.a for 13.3 release; added Virtex-7 FPGA support
1/18/2012	1.5	Updated to v3.02.a for 13.4 release; added Getting Started Tutorial section.
4/24/2012	1.6	Updated to core version to v3.03.a and add CORE Generator Support for 14.1 release.

Notice of Disclaimer

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 the Limited Warranties which can be viewed at http://www.xilinx.com/warranty.htm; 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 Critical Applications: http://www.xilinx.com/warranty.htm#critapps.