LogiCORE IP AXI Exerciser v4.00a

Product Guide

PG094 December 18, 2012

Table of Contents

SECTION I: SUMMARY

Introduction

The Xilinx LogiCORE™ IP Advanced eXtensible Interface (AXI) exerciser is a core that stresses the AXI4 interconnect and other AXI4 peripherals in the system. It generates a wide variety of AXI4 transactions based on the core programming.

Features

- AXI4 interface to program core for different traffic generation options
- Works as a master to issue read/write commands based on the programming sequence to CMDRAM
- Flexible data width capability (32/64-bit) on Slave and (32/64/128/256/512-bit) on Master AXI4 interface
- Interrupt pin indicating core completed generation of traffic
- Error interrupt pin indicating error occurred during core operation. Error registers can be read to understand the error occurred.

LogiCORE IP Facts Table	
Core Specifics	
Supported Device Family[1]	Zynq™-7000[2], Virtex®-7, Kintex™-7, Artix™-7
Supported User Interfaces	AXI4
Resources	See Table 2-1 to Table 2-3.
Provided with Core	
Design Files	ISE®: Verilog 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[3]	
Design Entry	ISE Design Suite 14.4 Vivado Design Suite 2012.4[4]
Simulation	Mentor Graphics ModelSim, Vivado Simulator
Synthesis	Xilinx Synthesis Technology (XST) 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. Supported in ISE Design Suite implementations only.
3. For the supported versions of the tools, see the Xilinx Design Tools: Release Notes Guide.
4. Supports only 7 series devices.

Overview

The AXI exerciser IP is designed to generate AXI4 traffic which can be used to stress different modules/interconnect connected in the system. Different configurable options allow the user to generate a wide variety of traffic based on their requirements.

Architecture of the core is broadly separated into a master and slave block, each of which contains the write block and read block. Other support functions are provided by the Control registers and Internal RAMs.

Figure 1-1 shows the top-level AXI exerciser block diagram.

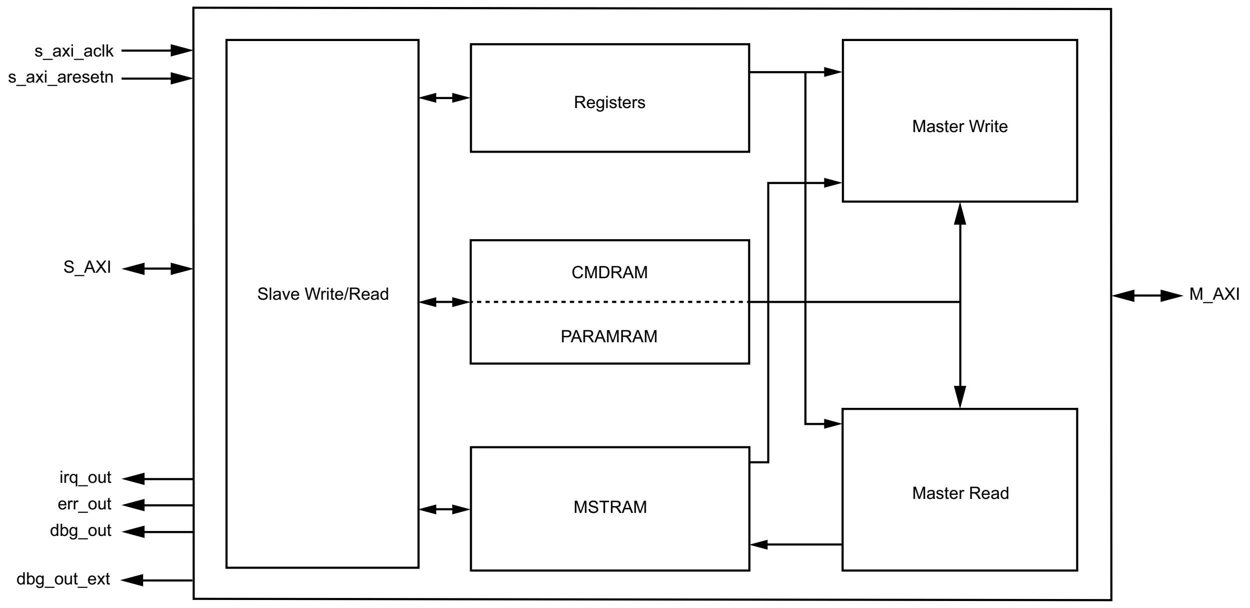


Figure 1-1: **AXI Exerciser Block Diagram**

The commands to be issued by the AXI exerciser are loaded in a 128-bit wide, 512 deep command RAM through AXI Slave interface. After the core is enabled, control logic issues the write/read commands based on the command settings programmed. The core updates the Status registers and asserts interrupts (based on enablement) on the completion of issuing programmed commands.

Control Registers and Internal RAMs

Control registers are provided to the user to program the core to generate different AXI4 transactions. For more information on each register, see the Register Space section.

Three internal RAMs are used as follows:

- Command RAM (CMDRAM)
- Parameter RAM (PARAMRAM)
- Master RAM (MSTRAM)

Command RAM

The CMDRAM is divided into two 4 KB regions, one for reads and one for writes. Each region of CMDRAM can hold 256 commands. Reads and writes operate simultaneously and independently, using the dual-ported CMDRAM block RAM to allow read and write requests to be started simultaneously. This means the slave access to the CMDRAM is prohibited while the master logic is enabled (when Reg0_m_enable is set).

Reads are issued to the master-read block AR channel from CMDRAM (0x000 to 0xFFF) locations (up to 256 commands of 128 bits each). Writes are issued to the master-write block AW channel from CMDRAM (0x1000 to 0x1FFF) locations (up to 256 commands of 128 bits each). Each command does not indicate whether it is a read or a write, because it is implied by its position in the CMDRAM.

CMD Memory Format

Each CMDRAM command in Table 1-1 is 128 bits wide.

Table 1-1: **CMDRAM Memory Format**

Word Offset	Bits	Description
+00	31:0	**AXI_Address[31:0]**: Address to drive on ar_addr or aw_addr (a*_addr[31:0])

Table 1-1: **CMDRAM Memory Format *(Cont'd)***

Word Offset	Bits	Description
+01	31	**Valid_cmd**[(1)]: When set, this is a valid command. When clear, halt the master logic for this request type (read or write).
	30:28	**last_addr[2:0]**: Should be set to 0 for C_M_AXI_DATA_WIDTH > 64. For writes, indicates the valid bytes in the last data cycle. For 64-bit mode: 000 = All bytes valid 001 = Only Bit[0] is valid 010 = Only Bits[1:0] are valid and so on. For 32-bit mode: 000 = All bytes valid 100 = Only Bit[0] is valid 101 = Only Bits[1:0] are valid 110 = Only Bits[2:0] are valid
	27:24	Reserved
	23:21	**Prot[2:0]**: Driven to a*_prot[2:0]
	20:15	**Id[5:0]**: Driven to a*_id[5:0]
	14:12	**Size[2:0]**: Driven to a*_size[2:0]
	11:10	**Burst[1:0]**: Driven to a*_burst[1:0]
	9	Reserved
	8	**Lock**: Driven to a*_lock
	7:0	**Len[7:0]**: Driven to a*_len[7:0].
+02	31	Reserved
	30:22	**My_depend[8:0]**: This command does not begin until this master logic has at least completed up to this command number. A value of 0 in this field means do not wait. This allows a command to wait until previous commands have completed for ordering.
	21:13	**Other_depend[8:0]**: This command does not begin until the other master logic has completed up to this command number. For example, if a write command had 0x04 in this field, the write would not begin until the read logic had at least completed its CMDs 0x00 through 0x03. A value of 0 in this field means do not wait, but commands can only be started in order for each master type. For example, if Write CMD[0x05] waits for Read 0x03, then Write CMD[0x06] cannot start until Read 0x03 completes as well. A read completes when it receives the last cycle of data, and a write completes when it receives BRESP.
	12:0	**Mstram_index[12:0]**[(2)]: Index into MstRAM for this transaction (reads will write to this MstRAM address, writes take data from this address)

Table 1-1: **CMDRAM Memory Format *(Cont'd)***

Word Offset	Bits	Description
+03	31:20	Reserved
	19:16	**qos[3:0]**: Driven to a*_qos[3:0]
	15:8	**user[7:0]**: Driven to a*_user[7:0]
	7:4	**cache[3:0]**: Driven to a*_cache[3:0]
	3	Reserved
	2:0	**Expected_resp**: 0x0 to 0x1 = Only OKAY is allowed 0x2 = Only EX_OK is allowed 0x3 = EX_OK or OK is allowed 0x4 = Only DECERR or SLVERR is allowed 0x7 = Any response is allowed

1. Valid_cmd: There should be at least one command with valid_cmd bit set to 0 for both reads and writes.
2. Mstram_index: MstRAM is shared by both Read/Write master logic. To avoid memory collision issues, ensure no write command data overlaps with read-command data by selecting proper index values.
   MstRAM index should be aligned the same as the master (Read/Write) address. Example: For a 64-bit aligned transaction, the least three bits should be zero. For a 128-bit aligned transaction, the least four bits should be zero.
   Address generation for MstRAM is based on the burst type selected for command.
3. It is recommended to write 0s to the command RAM reserved bits.

PARAMRAM

The PARAMRAM extends the command programmability provided through command RAM, by adding extra 32 bits to the decode of each command. Figure 1-2 shows how the PARAMRAM is addressed in relation to the CMDRAM. Only write access is allowed to the PARAMRAM from the slave interface. Reads to PARAMRAM from the slave interface are routed to the register address space.

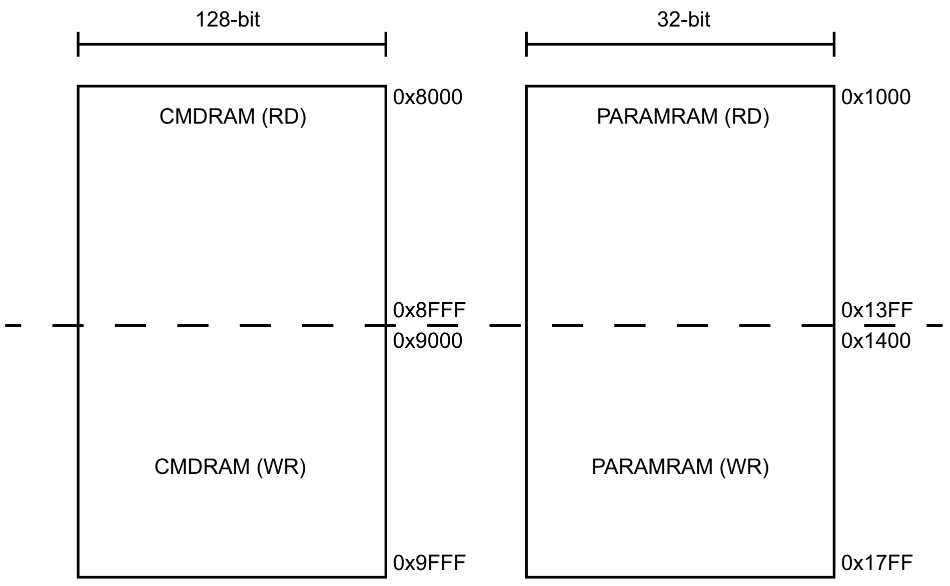


Figure 1-2: **PARAMRAM vs. CMDRAM**

Each entry in the PARAMRAM modifies its corresponding CMDRAM entry. The encoding and opcodes are described in Table 1-2 to Table 1-6.

Table 1-2: **PARAMRAM Entry Control Signals**

Bits	**Name**	**Description**
31:29	Opcode	The opcode defines how the op_control bits are used. Currently four operations are supported: **Bits** | **Name** | **Description** 00 | NOP | The command in the CMDRAM executes unaltered. 01 | OP_REPEAT | The command in the CMDRAM repeats multiple times. 10 | OP_DELAY | The command in the CMDRAM delays before execution. 11 | OP_FIXEDREPEAT_DELAY | The command in the CMDRAM repeats 255 times. Delay and address range can be constrained.
28	Idmode	Unused
27:26	Intervalmode	Control interval delay validated by OP_FIXEDREPEAT_DELAY. 00 = Constant Delay as programmed with Bits[19:8]
25:24	Addrmode	Control addressing when a command is being repeated. **Bits** | **Name** | **Description** 00 | Constant | Address does not change 01 | Increment | Address increments ((BUSWIDTH / 8) × (AXI_LEN + 1)) between repeated transactions 10 | Random | Address is randomly generated within a address range. Valid only when OP_FIXEDREPEAT_DELAY is selected.
23:0	PARAMRAM Opcodes	The definition for Bits[23:0] depend on the selected PARAMRAM opcodes. Details are described in Table 1-3 to Table 1-6.

PARAMRAM Opcodes

Each of the four commands uses 24 bits of op_control space to shape the command. Each of the four op_control field is described in Table 1-3 to Table 1-6.

The OP_NOP command is ignored and the command within the CMDRAM is executed normally.

Table 1-3: **OP_NOP**

Bits	**Name**	**Description**
23:0	Unused	N/A

The entire op_control field of 24 bits is used as a counter for repeating the command in the CMDRAM entry.

Table 1-4: **OP_REPEAT**

Bits	Name	Description
23:0	Repeat Count	Command repeats this many times.

The entire Op_control field of 24 bits is used as a delay counter for issuance of the command in the CMDRAM entry.

Table 1-5: **OP_DELAY**

Bits	Name	Description
23:0	Delay Count	Command execution is delayed for this many cycles.

1. Delay observed between two transactions is greater than or equal to the programmed value.

Table 1-6: **OP_FIXEDREPEAT_DELAY**

Bits	Name	Description
23:20	Addr Range Encoded	Core issues a new random address within the range encoded below starting with base address programmed by the user.
19:8	Delay Count	Each command execution is delayed for this many cycles.
7:0	Delay Range Encoded	Unused

Encoded	Range (KB)	Encoded	Range (MB)
0	4	8	1
1	8	9	2
2	16	10	4
3	32	11	8
4	64	12	16
5	128	13	32
6	256	14	64
7	512	15	128

1. Delay observed between two transactions is greater than or equal to the programmed value.

PARAMRAM should be filled with valid data for corresponding entry in the CMDRAM. CMDRAM should be filled with valid data until the first invalid command entry. PARAMRAM features are valid for C_M_AXI_DATA_WIDTH of 32 and 64. For other data widths, the PARAMRAM entries should be programmed to 0.

Master RAM

The MSTRAM has 8 KB of internal RAM used for the following:

- Take data from this RAM for write transactions

- Store data to this RAM for read transaction

The RAM address to use for a read/write transaction is controlled through command RAM programming.

The Master RAM A and B channels are shown connected in Figure 1-3.

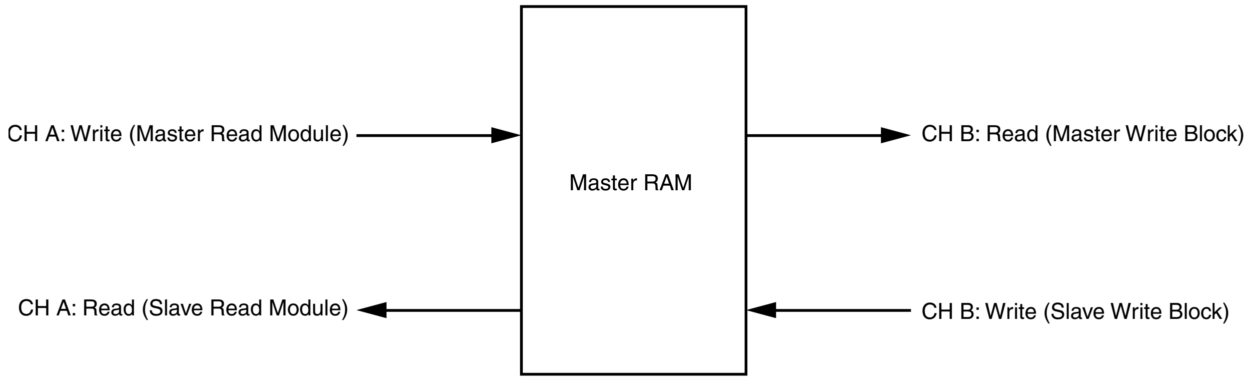

CH A: Write (Master Read Module)
CH B: Read (Master Write Block)
Master RAM
CH A: Read (Slave Read Module)
CH B: Write (Slave Write Block)

Figure 1-3: **Master RAM Channels**

Address Generation

The address generation to the MSTRAM from each of the mentioned block is through the addrgen block. Addrgen blocks receive input for the address information, length, and bus size. Then, it generates output for the an index into the MSTRAM for each beat of the transfer. It also tracks the transfer length and signals to other logic when a transfer is complete.

Addrgen block considers the "mstram_index" and "AXI_Address" in the Command RAM entries to generate the MSTRAM address.

Mstram_index should be selected in such a way that it matches AXI_Address offset. The following examples illustrate the mstram_index selection:

- **32-bit Aligned Transfer** – Lower Bits[1:0] of AXI_Address are zero. Mstram_index should also be selected in such a way that the lower Bits[1:0] are zero.

  Example:

  AXI_Address = 0xC000_0004
  Mstram_index = 0x0001_0004

- **32-bit Unaligned Transfer** – Lower Bits[1:0] of AXI_Address are offset by the byte from which transfer should start. Mstram_index should also be selected in such a way that the lower Bits[1:0] are offset by the same byte offset as indicated by AXI_Address.

Example:

AXI_Address = 0xC000_0005 (offset by 1-byte)
Mstram_index = 0x0001_0005 (offset by 1-byte)

- **64-bit Aligned Transfer** – Lower Bits[2:0] of AXI_Address are zero. Mstram_index should also be selected in such a way that the lower Bits[2:0] are zero.

  Example:

  AXI_Address = 0xC000_0008
  Mstram_index = 0x0001_0008

- **64-bit Unaligned Transfer** – Lower Bits[2:0] of AXI_Address are offset by the byte from which transfer should start. Mstram_index should also be selected in such a way that the lower Bits[2:0] are offset by the same byte offset as indicated by AXI_Address.

  Example:

  AXI_Address = 0xC000_0005 (offset by 5 bytes)
  Mstram_index = 0x0001_0005 (offset by 5 bytes)

Similar rules apply for higher data width (128/256/512) transactions. Only aligned transfers are supported for 128/256/512 bit width selection.

Data Generation

MSTRAM is organized as 64-bit wide, 1024-deep memory. For data widths of 32 and 64, the data from MSTRAM is sent to corresponding modules without any truncation/expansion in data width.

But for data widths greater than 64, to save multiple RAM instances, the same 64-bit data is duplicated/truncated based on the current data width selection of master channels.

Example: Data width of 128:

- During read access from master write block,

  wdata_m[127:0] = 2{read_data_from_mstram[63:0]}

  That is, 64-bit data is duplicated on write-data bus to make it 128 bits wide.

- During write access by master read block,

  write_data_to_mastram[63:0] = rdata_m[63:0]

  That is, lower 64 bits of read data bus are stored in MSTRAM.

Example: Data width of 256:

- During read access from master write block,

  wdata_m[255:0] = 4{read_data_from_mstram[63:0]}

  That is, 64-bit data is duplicated on write-data bus to make it 256 bits wide.

- During write access by master read block,

  write_data_to_mastram[63:0] = rdata_m[63:0]

  That is, lower 64 bits of read data bus are stored in MSTRAM.

Slave Modules

- **Slave-write** – The slave-write logic replies to writes received on the slave AW and W ports. It generates and writes completions (on B).
- **Slave-read** – The slave-read logic replies to reads received on the slave AR port. It generates read returns (on the R port).

Table 1-7 shows the address map for different regions accessed by slave-write/slave-read.

Table 1-7: **Slave-Write/Slave-Read Address Map**

Region	Description
0x0000_0000–0x0000_0FFF	Internal registers
0x0000_1000–0x0000_17FF	PARAMRAM (2 KB)
0x0000_8000–0x0000_FFFF	CMDRAM (8 KB)
0x0001_0000–0x007F_FFFF	MSTRAM (8 KB)

All regions are only partially decoded. Accessing offset +0x00 or +0x80 reads Register 0. The AXI_Exerciser ignores address Bits[31:23] for its decode, but decodes Bits[22:0].

For slave logic the write interleaving depth is 1.

Master Modules

Figure 1-4 shows the master logic.

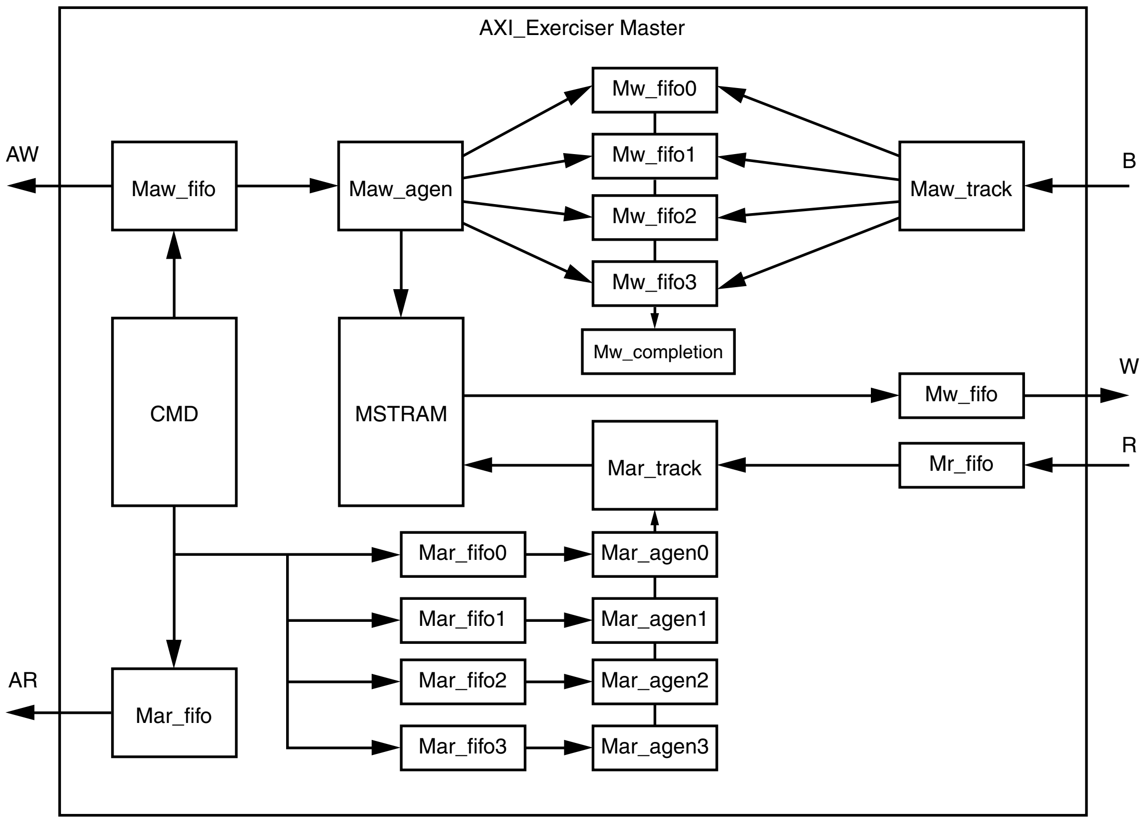


Figure 1-4: **AXI_Exerciser Master**

Issuing Read Transactions

For reads, each CMD is read from the CMDRAM and pushed to the 2-deep Mar_fifo. Mar_track decides which Mar_fifo0–3 it is also pushed into. The first ID goes to Mar_fifo0, the next ID goes to Mar_fifo1, and so on. The Mar_fifo sends the information to the AXI_M AR signals. Mar_fifo0–3 holds the requests before sending them to Mar_agen0–3. If the Mar_track assigns ID = 0x12 to Mar_fifo1, any further ID = 0x12 transactions are pushed onto Mar_fifo1.

After four unique IDs are valid at once, no further Read CMDs can be processed until one of the Mar_fifo0–3 is empty. Read data returned from the switch is placed in Mr_fifo, then popped out. Each ID is searched across each Mar_agen0–3, which selects the proper Mar_agen and drives the address to the MSTRAM to write in the R data.

On the last data cycle, the corresponding Mar_fifo0–3 is popped, and the next entry is prepared. This strategy allows at least four simultaneous reads with any arbitrary ID and often allows more if the same ID is reused in multiple requests.

Issuing Write Transactions

For writes, each CMD is read from the CMDRAM and pushed to the 2-deep Maw_fifo and Maw_fifow. Maw_fifo is connected to the AXI_M AW signals and drives the request to the switch. Maw_fifow holds two requests heading to the Maw_agen block which generates addresses into the MSTRAM. Data read from MSTRAM is pushed into the Mw_fifo, which is connected to the AXI_M W signals.

To return BRESP out of order, Maw_agen feeds into Maw_track which tracks up to four IDs in a similar way to Mar_track. A write ID is assigned to an Mw_fifo0–3. When that ID receives a BRESP, it pops the corresponding Mw_fifo0–3. This allows the master write logic to handle receiving BRESPs out of order.

Programming Sequence

1. Load CMDRAM RAM with the required number of commands.
2. Load PARAM RAM with the desired opcodes.
3. Load MSTRAM memory with data to be issued during write transactions.
4. Enable the desired interrupt/status bits.
5. Enable the core through register control signal.
6. Wait for interrupt (if enabled) or poll Enable register control signal to check for completion of issuing the commands.

Applications

The AXI exerciser(s) can be connected to an AXI-based system to stress the modules connected to the interconnect.

Figure 1-5 shows the AXI exerciser connected to a MicroBlaze™ processor. The MicroBlaze programs the AXI exercisers and the AXI exercisers generate traffic to other cores.

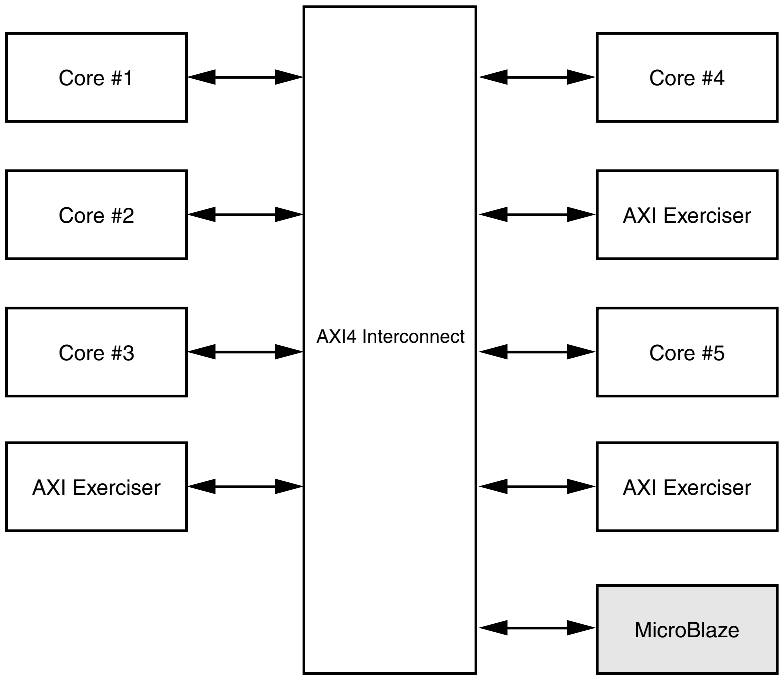


Figure 1-5: **MicroBlaze System**

Figure 1-6 shows the AXI exerciser connected to a Zynq™ platform, The PL part can include DMA or any other master type peripherals which can be programmed by AXI exerciser.

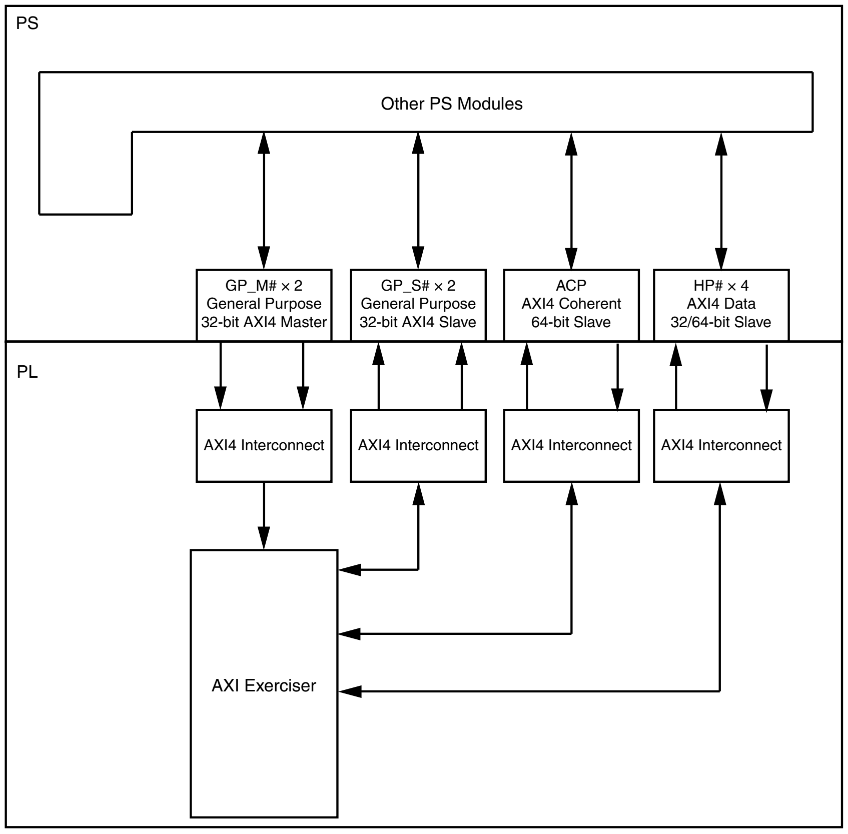


Figure 1-6: **Zynq System**

Licensing and Ordering Information

This Xilinx LogiCORE IP module is provided at no additional cost with the Xilinx Vivado Design Suite and ISE Design Suite Embedded Edition tools 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.

Chapter 2

Product Specification

Performance

This section details the performance information for various core configurations.

Maximum Frequencies

The maximum frequency achieved on standalone runs is 200 MHz for Virtex-7/Kintex-7 families and 150 MHz for Artix-7 family as detailed in the Resource Utilization section.

Latency

The AXI exerciser has a write and read command issuing latency.

Write Command Issuing Latency

Latency is calculated from the point where the core is enabled by writing to Reg0_master_control and the AWVALID assertion on Master Ports. Latency is 9 clock cycles with delay parameters in PARAMRAM set to 0.

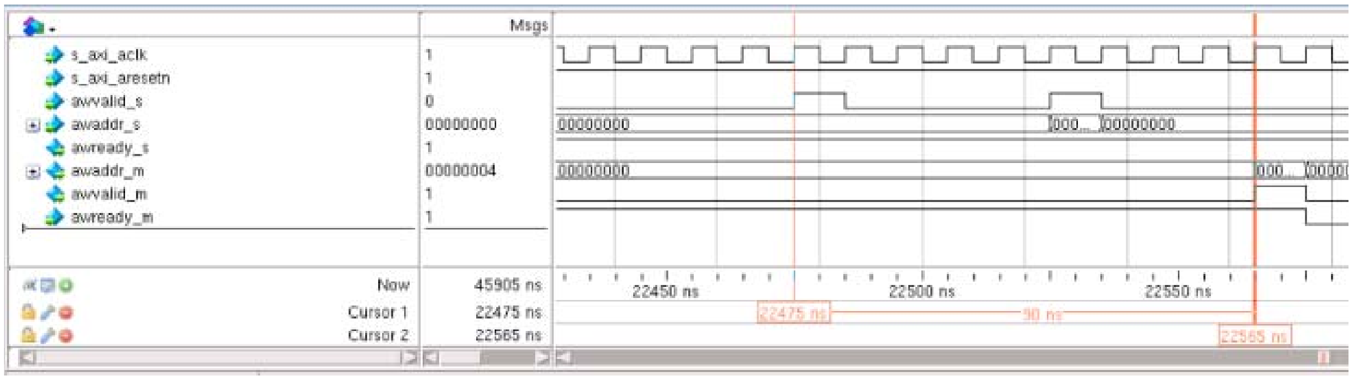


Figure 2-1: **Write Command Issuing Latency**

Read Command Issuing Latency

Latency is calculated from the point where the core is enabled by writing to Reg0_master_control and the ARVALID assertion on Master Ports. Latency is 9 clock cycles with delay parameters in PARAMRAM set to 0.

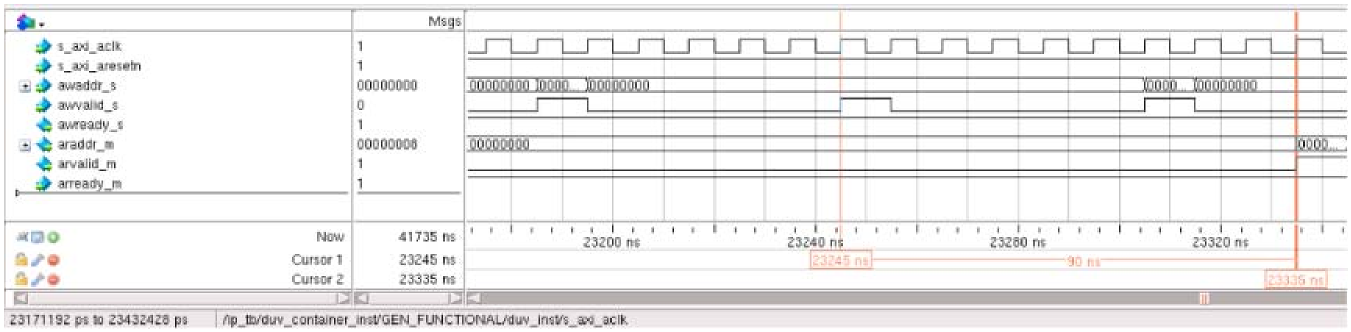


Figure 2-2: **Read Command Issuing Latency**

Throughput

The AXI exerciser has a master write and read channel throughput.

Master Write Channel

Throughput is measured on master write channel for transaction with Length = 255 (Maximum burst length) for a 32-bit data bus width.

$$\text{Throughput} = (A - B) \times 100 / (\text{Total beats in the transaction}) \qquad \text{Equation 2-1}$$

A = Number of clock cycles WVALID and WREADY are asserted = 256

B = Number of clock cycles WVALID is deasserted, WREADY is asserted = 0

$$\text{Throughput} = (256 - 0) \times 100 / 256 = 100\% \qquad \text{Equation 2-2}$$

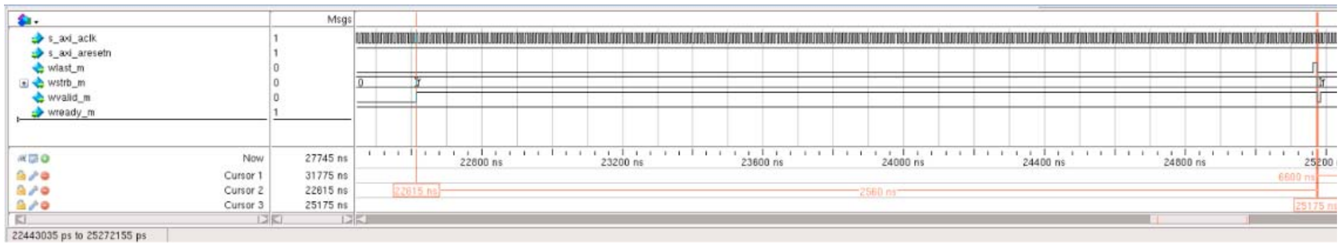


Figure 2-3: **Master Write Channel Throughput**

Master Read Channel

Throughput is measured on master read channel for transaction with Length = 255 (Maximum burst length) for a 32-bit data bus width.

Throughput = (A - B) × 100 / (Total beats in the transaction) *Equation 2-3*

A = Number of clock cycles RREADY and RVALID are asserted = 256

B = Number of clock cycles RREADY is deasserted, RVALID is asserted = 0

Throughput = (256 - 0) × 100 / 256 = 100% *Equation 2-4*

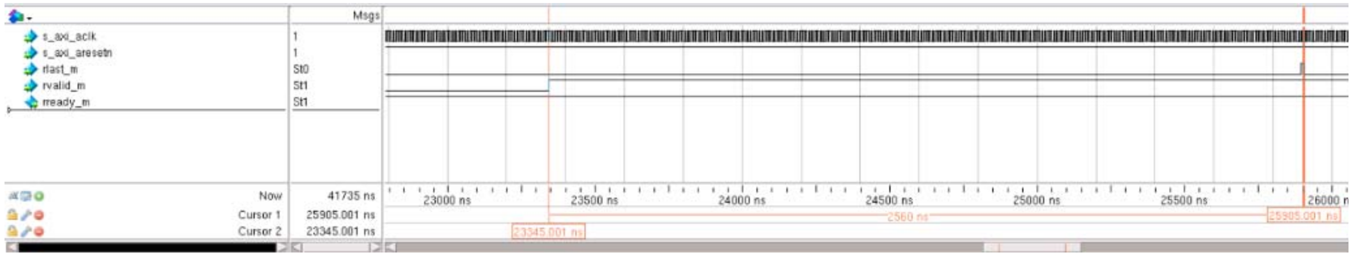


Figure 2-4: **Master Read Channel Throughput**

Resource Utilization

Resources required for the AXI exerciser core have been estimated for the Virtex®-7 Field Programmable Gate Array (FPGA) in Table 2-1, Kintex™-7 FPGA and Zynq™-7000 in Table 2-2, and Artix™-7 FPGA and Zynq-7000 device in Table 2-3. They are derived from post-synthesis reports, and might change during MAP and PAR.

Table 2-1: **Virtex-7 FPGA (XC7VX1140TFLG1930-1) Resource Estimates**

Parameter Values (Other Parameters at Default Value)				Device Resources			Performance
C_ZERO_INVALID	**C_NO_EXCL**	**C_S_AXI_DATA_WIDTH**	**C_M_AXI_DATA_WIDTH**	**Slices**	**Slice Flip-Flops**	**LUTs**	**F_{MAX} (MHz)**
0	0	32	32	1,894	3,493	5,066	200
0	0	32	64	1,867	3,592	5,120	202
0	0	64	32	2,055	3,625	5,325	200
0	0	64	64	1,972	3,724	5,405	200
0	1	32	32	1,797	3,237	4,714	200
0	1	32	64	1,710	3,336	4,813	202
0	1	64	32	1,908	3,369	4,984	200

Table 2-1: **Virtex-7 FPGA (XC7VX1140TFLG1930-1) Resource Estimates *(Cont'd)***

Parameter Values (Other Parameters at Default Value)				Device Resources			Performance
C_ZERO_INVALID	C_NO_EXCL	C_S_AXI_DATA_WIDTH	C_M_AXI_DATA_WIDTH	Slices	Slice Flip-Flops	LUTs	F_{MAX} (MHz)
0	1	64	64	1,927	3,468	5,075	200
1	0	32	32	1,998	3,493	5,374	200
1	0	32	64	2,122	3,591	5,486	200
1	0	64	32	2,176	3,625	5,619	200
1	0	64	64	2,078	3,723	5,794	200
0	1	32	32	1,831	3,237	4,804	202
0	1	32	64	1,827	3,336	4,950	200
0	1	64	32	2,009	3,368	5,120	200
0	1	64	64	1,687	3,468	5,216	202

Table 2-2: **Kintex-7 FPGA (XC7K480TFFG1156-1) and Zynq-7000 Device Resource Estimates**

Parameter Values (Other Parameters at Default Value)				Device Resources			Performance
C_ZERO_INVALID	C_NO_EXCL	C_S_AXI_DATA_WIDTH	C_M_AXI_DATA_WIDTH	Slices	Slice Flip-Flops	LUTs	F_{MAX} (MHz)
0	0	32	32	1,996	3,493	4,991	200
0	0	32	64	1,905	3,592	5,116	200
0	0	64	32	2,192	3,625	5,309	204
0	0	64	64	2,062	3,724	5,389	207
0	1	32	32	1,804	3,237	4,699	203
0	1	32	64	1,903	3,336	4,785	204
0	1	64	32	1,941	3,369	4,963	201
0	1	64	64	1,862	3,468	5,087	200
1	0	32	32	2,043	3,493	5,321	200
1	0	32	64	2,079	3,591	5,452	204
1	0	64	32	2,149	3,625	5,625	203
1	0	64	64	2,248	3,723	5,742	200
0	1	32	32	1,814	3,237	4,803	210
0	1	32	64	1,917	3,336	4,896	202
0	1	64	32	2,117	3,368	5,052	206
0	1	64	64	2,008	3,468	5,180	200

Table 2-3: **Artix-7 FPGA (XC7A200TFFG1156-1) and Zynq-7000 Device Resource Estimates**

Parameter Values (Other Parameters at Default Value)				Device Resources			Performance
C_ZERO_INVALID	C_NO_EXCL	C_S_AXI_DATA_WIDTH	C_M_AXI_DATA_WIDTH	Slices	Slice Flip-Flops	LUTs	F_{MAX} (MHz)
0	0	32	32	1,942	3,495	4,996	150
0	0	32	64	2,059	3,593	5,109	150
0	0	64	32	2,014	3,623	5,202	150
0	0	64	64	1,911	3,725	5,276	151
0	1	32	32	1,740	3,238	4,691	150
0	1	32	64	1,542	3,335	4,818	150
0	1	64	32	1,695	3,373	4,875	150
0	1	64	64	1,680	3,467	4,967	150
1	0	32	32	2,227	3,496	5,321	150
1	0	32	64	1,844	3,593	5,490	150
1	0	64	32	2,102	3,627	5,508	150
1	0	64	64	1,999	3,726	5,651	150
0	1	32	32	1,997	3,238	4,778	150
0	1	32	64	1,960	3,335	4,885	150
0	1	64	32	2,020	3,374	4,955	150
0	1	64	64	1,663	3,467	5,104	151

Port Descriptions

The AXI exerciser signals are described in Table 2-4. The I/O signals include clock, reset, AXI4 slave port signals, AXI4 master ports signals, and interrupt signals.

Table 2-4: **AXI Exerciser I/O Signals**

Port	Name	Interface	Direction	Description
P0	s_axi_aclk	System	Input	Clock
P1	s_axi_aresetn	System	Input	Active-Low reset
P2	irq_out	System	Output	Interrupt on traffic generation completion

Table 2-4: **AXI Exerciser I/O Signals *(Cont'd)***

Port	Name	Interface	Direction	Description
P3	err_out	System	Output	Error interrupt
P4	dbg_out[24:0]	System	Output	Debug data set 1 for ILA purpose
P5	dbg_out[2:0]	System	Output	Debug data set2 for ILA purpose
P6	global_test_enable_l	System	Input	Not used in this version of the core (Reserved for future use)
P7	awid_s[C_S_AXI_ID_WIDTH - 1:0]	AXI4_S	Input	See AXI4 Bus definition
P8	awaddr_s[31:0]	AXI4_S	Input	See AXI4 Bus definition
P9	awlen_s[7:0]	AXI4_S	Input	See AXI4 Bus definition
P10	awlen3_s[3:0]	AXI4_S	Input	See AXI4 Bus definition
P11	awsize_s[2:0]	AXI4_S	Input	See AXI4 Bus definition
P12	awburst_s[1:0]	AXI4_S	Input	See AXI4 Bus definition
P13	awlock_s	AXI4_S	Input	See AXI4 Bus definition
P14	awcache_s[3:0]	AXI4_S	Input	See AXI4 Bus definition
P15	awprot_s[2:0]	AXI4_S	Input	See AXI4 Bus definition
P16	awqos_s[3:0]	AXI4_S	Input	See AXI4 Bus definition
P17	awuser_s[C_S_AXI_AWUSER_WIDTH - 1:0]	AXI4_S	Input	See AXI4 Bus definition
P18	awvalid_s	AXI4_S	Input	See AXI4 Bus definition
P19	awready_s	AXI4_S	Output	See AXI4 Bus definition
P20	wlast_s	AXI4_S	Input	See AXI4 Bus definition
P21	wdata_s[C_S_AXI_DATA_WIDTH - 1:0]	AXI4_S	Input	See AXI4 Bus definition
P22	wstrb_s[C_S_AXI_DATA_WIDTH / 8 - 1:0]	AXI4_S	Input	See AXI4 Bus definition
P23	wvalid_s	AXI4_S	Input	See AXI4 Bus definition
P24	wready_s	AXI4_S	Output	See AXI4 Bus definition
P25	bid_s[C_S_AXI_ID_WIDTH - 1:0]	AXI4_S	Output	See AXI4 Bus definition
P26	bresp_s[1:0]	AXI4_S	Output	See AXI4 Bus definition
P27	bvalid_s	AXI4_S	Output	See AXI4 Bus definition
P28	bready_s	AXI4_S	Input	See AXI4 Bus definition
P29	arid_s[C_S_AXI_ID_WIDTH - 1:0]	AXI4_S	Input	See AXI4 Bus definition
P30	araddr_s[31:0]	AXI4_S	Input	See AXI4 Bus definition
P31	arlen_s[7:0]	AXI4_S	Input	See AXI4 Bus definition
P32	arlen3_s[3:0]	AXI4_S	Input	See AXI4 Bus definition
P33	arsize_s[2:0]	AXI4_S	Input	See AXI4 Bus definition
P34	arburst_s[1:0]	AXI4_S	Input	See AXI4 Bus definition
P35	arlock_s	AXI4_S	Input	See AXI4 Bus definition

Table 2-4: **AXI Exerciser I/O Signals *(Cont'd)***

Port	Name	Interface	Direction	Description
P36	arcache_s[3:0]	AXI4_S	Input	See AXI4 Bus definition
P37	arprot_s[2:0]	AXI4_S	Input	See AXI4 Bus definition
P38	arqos_s[3:0]	AXI4_S	Input	See AXI4 Bus definition
P39	aruser_s[C_S_AXI_ARUSER_WIDTH - 1:0]	AXI4_S	Input	See AXI4 Bus definition
P40	arvalid_s	AXI4_S	Input	See AXI4 Bus definition
P41	arready_s	AXI4_S	Output	See AXI4 Bus definition
P42	rid_s[C_S_AXI_ID_WIDTH - 1:0]	AXI4_S	Output	See AXI4 Bus definition
P43	rlast_s	AXI4_S	Output	See AXI4 Bus definition
P44	rdata_s[C_S_AXI_DATA_WIDTH - 1:0]	AXI4_S	Output	See AXI4 Bus definition
P45	rresp_s[1:0]	AXI4_S	Output	See AXI4 Bus definition
P46	rvalid_s	AXI4_S	Output	See AXI4 Bus definition
P47	rready_s	AXI4_S	Input	See AXI4 Bus definition
P48	awid_m[C_M_AXI_THREAD_ID_WIDTH - 1:0]	AXI4_M	Output	See AXI4 Bus definition
P49	awaddr_m[31:0]	AXI4_M	Output	See AXI4 Bus definition
P50	awlen_m[7:0]	AXI4_M	Output	See AXI4 Bus definition
P51	awlen3_m[3:0]	AXI4_M	Output	See AXI4 Bus definition
P52	awsize_m[2:0]	AXI4_M	Output	See AXI4 Bus definition
P53	awburst_m[1:0]	AXI4_M	Output	See AXI4 Bus definition
P54	awlock_m	AXI4_M	Output	See AXI4 Bus definition
P55	awcache_m[3:0]	AXI4_M	Output	See AXI4 Bus definition
P56	awprot_m[2:0]	AXI4_M	Output	See AXI4 Bus definition
P57	awqos_m[3:0]	AXI4_M	Output	See AXI4 Bus definition
P58	awuser_m[C_M_AXI_AWUSER_WIDTH - 1:0]	AXI4_M	Output	See AXI4 Bus definition
P59	awvalid_m	AXI4_M	Output	See AXI4 Bus definition
P60	awready_m	AXI4_M	Input	See AXI4 Bus definition
P61	wid_m[C_M_AXI_THREAD_ID_WIDTH - 1:0]	AXI4_M	Output	See AXI4 Bus definition
P62	wlast_m	AXI4_M	Output	See AXI4 Bus definition
P63	wdata_m[C_M_AXI_DATA_WIDTH - 1:0]	AXI4_M	Output	See AXI4 Bus definition
P64	wstrb_m[C_M_AXI_DATA_WIDTH / 8 - 1:0]	AXI4_M	Output	See AXI4 Bus definition
P65	wvalid_m	AXI4_M	Output	See AXI4 Bus definition
P66	wready_m	AXI4_M	Input	See AXI4 Bus definition
P67	bid_m[C_M_AXI_THREAD_ID_WIDTH - 1:0]	AXI4_M	Input	See AXI4 Bus definition
P68	bresp_m[1:0]	AXI4_M	Input	See AXI4 Bus definition
P69	bvalid_m	AXI4_M	Input	See AXI4 Bus definition
P70	bready_m	AXI4_M	Output	See AXI4 Bus definition

Table 2-4: **AXI Exerciser I/O Signals *(Cont'd)***

Port	Name	Interface	Direction	Description
P71	arid_m[C_M_AXI_THREAD_ID_WIDTH - 1:0]	AXI4_M	Output	See AXI4 Bus definition
P72	araddr_m[31:0]	AXI4_M	Output	See AXI4 Bus definition
P73	arlen_m[7:0]	AXI4_M	Output	See AXI4 Bus definition
P74	arlen3_m[3:0]	AXI4_M	Output	See AXI4 Bus definition
P75	arsize_m[2:0]	AXI4_M	Output	See AXI4 Bus definition
P76	arburst_m[1:0]	AXI4_M	Output	See AXI4 Bus definition
P77	arlock_m	AXI4_M	Output	See AXI4 Bus definition
P78	arcache_m[3:0]	AXI4_M	Output	See AXI4 Bus definition
P79	arprot_m[2:0]	AXI4_M	Output	See AXI4 Bus definition
P80	arqos_m[3:0]	AXI4_M	Output	See AXI4 Bus definition
P81	aruser_m[C_M_AXI_ARUSER_WIDTH - 1:0]	AXI4_M	Output	See AXI4 Bus definition
P82	arvalid_m	AXI4_M	Output	See AXI4 Bus definition
P83	arready_m	AXI4_M	Input	See AXI4 Bus definition
P84	rid_m[C_M_AXI_THREAD_ID_WIDTH - 1:0]	AXI4_M	Input	See AXI4 Bus definition
P85	rlast_m	AXI4_M	Input	See AXI4 Bus definition
P86	rdata_m[C_M_AXI_DATA_WIDTH - 1:0]	AXI4_M	Input	See AXI4 Bus definition
P87	rresp_m[1:0]	AXI4_M	Input	See AXI4 Bus definition
P88	rvalid_m	AXI4_M	Input	See AXI4 Bus definition
P89	rready_m	AXI4_M	Output	See AXI4 Bus definition

1. AXI_S refers to AXI4 slave connection.
2. AXI_M refers to AXI4 master connection.

Register Space

The AXI exerciser provides 16 registers to control its behavior, provide status or debug information, and to control external signals. The register space is only partially decoded, so the register offsets repeat every 0x40 bytes, except for the 0xB4 Special Error register.

IMPORTANT: *All registers must be written with full-word writes.*

Byte or halfword writes are interpreted as full-word writes (which can have unpredictable results). All bit descriptions use a Little Endian bit numbering, where 31 is the left-most or MSB, and Bit[0] is the right-most or LSB. All registers reset to zero (except for the read-only bits). Access to read-only registers issue an OKAY response.

Table 2-5: **AXI Exerciser Register Map**

Offset	Register Name	Description
0x00	Reg0_master_control	Master Control
0x04	Reg1_slave_control	Slave Control
0x08	Reg2_errors	Errors
0x0C	Reg3_error_enable	Error Enable
0x10	Reg4_master_control	Master Control
0x14	Reg5_status	Status
0x18	Reg6_errors_alias	Errors Alias
0x1C	Reg7_ex_status1 and Reg8_ex_status2	Exclusive Status 1
0x20	Reg7_ex_status1 and Reg8_ex_status2	Exclusive Status 2
0x24	Reg9_dbgcnt, Reg10_dbgcnt, and Reg11_dbgcnt	Debug Count
0x28	Reg9_dbgcnt, Reg10_dbgcnt, and Reg11_dbgcnt	Debug Count
0x2C	Reg9_dbgcnt, Reg10_dbgcnt, and Reg11_dbgcnt	Debug Count
0x30–0x3F	Reserved	Reserved
0xB4	Reg_error	Error. Any access to this register returns the SLVERR response.

Reg0_master_control

Note: Bits[19:0] should be all zeroes for the current version of the core.

Table 2-6: **Reg0_master_control (0x00)**

Bits	Default Value	Access	Description
31:24	0x47	RO	Revision of the core.
23:21	0	RO	M_ID_WIDTH, where: 0x0 = Indicates 1-bit width 0x1 = Indicates 2-bit width ... 0x7 = Indicates 8-bit width
20	0	RW	When set, the master logic begins. When both the Read and Write state machines complete, this bit is automatically cleared to indicate to software that the AXI_Exerciser is done.
19:10	0	RW	Initial command index for the Master Write logic to begin at.
9:0	0	RW	Initial command index for the Master Read logic to begin at
Back to Top			

Reg1_slave_control

Table 2-7: **Reg1_slave_control (0x04)**

Bits	Default Value	Access	Description
31:20	0	RO	Reserved
19	0	RW	When set, slave reads are not processed if there are any pending writes. On completing each write, at least one read data is returned to prevent starvation.
18	0	RW	When set, disables exclusive access support and error response ability for reads on reg16.
17	0	RW	When set, forces all BRESPs to be issued in the order the requests were received.
16	0	RW	When set, forces all slave reads to be done in the order received.
15	0	RW	When set, if any bit in reg2_errs[15:0] is set, then err_out is asserted.
14:0	–	RW	Reserved
Back to Top			

Reg2_errors

Table 2-8: **Reg2_errors (0x08)**

Bits	Default Value	Access	Description
31	0	RW1C	Set when both master write and master read CMD logic completes and reg3_error_enable[31] is 1. When set, irq_out is driven to 1.
30:21	0	RW1C	Reserved
20	0	RW1C	On master interface Received an RVALID with a RID that did not match any pending reads.
19	0	RW1C	Received a BVALID with a BID that did not match any pending writes.
18	0	RW1C	On a master write completion, the response returned was not allowed by expected_resp[2:0].
17	0	RW1C	On a master read completion, the response returned was not allowed by expected_resp[2:0].
16	0	RW1C	On the master interface R, Last as signaled either when it was not expected or was not signaled when it was expected. This indicates a read length error.
15:2	0	RW1C	Reserved
1	0	RW1C	On the slave interface W, a WSTRB assertion was detected on an illegal byte lane.
0	0	RW1C	On the slave interface W, Last was signaled either when it was not expected or was not signaled when it was expected. This indicates a slave write length error
Back to Top			

Reg3_error_enable

If an error occurs but the corresponding bit in Reg3_error_enable is not set, then the bit in Reg2_errors is not set and no error signaling occurs. To enable all errors, set Reg3_error_enable to 0xFFFF_FFFF.

Table 2-9: **Reg3_error_enable (0x0C)**

Bits	Default Value	Access	Description
31	0	RW	When set to 1, when both the MR and MW CMD logic completes, reg2_errors[31] is set to 1 and irq_out driven to 1.
30:0	0	RW	Each bit controls whether the corresponding Reg2 error is enabled.
Back to Top			

Reg4_master_control

Table 2-10: **Reg4_master_control (0x10)**

Bits	Default Value	Access	Description
31:16	0	RO	Reserved
15	0	RW	When set, if any bit in reg2_errs[30:16] is set, then err_out is asserted.
14:0	0	RW	Reserved
Back to Top			

Reg5_status

Table 2-11: **Reg5_status (0x14)**

Bits	Default Value	Access	Description
31	0	RO	Reserved
30	1	RO	AXI4
29	0	RO	Reserved
28	0	RO	C_m_axi_data_width set to 64.
27	0	RO	Reserved
26	0	RO	C_s_axi_data_width set to 64.
25	0	RO	C_coherent set to 1.

Table 2-11: **Reg5_status (0x14) *(Cont'd)***

Bits	Default Value	Access	Description
24	0	RO	C_afi set to 1.
23:0	–	RO	Debug information (data in these bits is depended on internal core state).
Back to Top			

Reg6_errors_alias

This read-only register provides a copy of Reg2_errors, but reading Reg6_errors_alias clears Bit[31] of Reg2_errors.

Table 2-12: **Reg6_errors_alias (0x18)**

Bits	Default Value	Access	Description
31:0	0	RO/Clear on read	Copy of Reg2_errors
Back to Top			

Reg7_ex_status1 and Reg8_ex_status2

These read-only registers provide internal exclusive access debug information, not generally useful for software

Table 2-13: **Reg7_ex_status1 and Reg8_ex_status2 (0x1C and 0x20)**

Bits	Default Value	Access	Description
31:0	–	RO	Debug information (data in these bits is depended on internal core state).
Back to Top			

Reg9_dbgcnt, Reg10_dbgcnt, and Reg11_dbgcnt

These registers are reserved on purpose and is not used in the current version of the core.

Table 2-14: **Reg9_dbgcnt, Reg10_dbgcnt, and Reg11_dbgcnt (0x24, 0x28, and 0x2C)**

Bits	Default Value	Access	Description
31:0	0	RW	Reserved
Back to Top			

Designing with the Core

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

Clocking

The `s_axi_aclk` input is the system clock.

Resets

The `s_axi_aresetn` is an active-Low reset to the core.

Protocol Description

The AXI exerciser uses the standard AXI 2.0 specification (ARM IHI 0022C).

AXI Exerciser Design Parameters

The AXI exerciser design parameters are listed and described in Table 3-1.

Table 3-1: **AXI Exerciser Design Parameters**

Generic	Parameter Name	Type	Allowable Values	Default Value	Feature/Description
			System Parameters		
G1	C_FAMILY	String	Virtex7, Kintex7, Artix7, Zynq	Virtex7	Target FPGA family

Table 3-1: **AXI Exerciser Design Parameters *(Cont'd)***

Generic	Parameter Name	Type	Allowable Values	Default Value	Feature/Description
G2	C_M_AXI_THREAD_ID_WIDTH	Integer	1 to 6	0	Controls the width of the *_ID ports of AXI_M
G3	C_S_AXI_ID_WIDTH	Integer	1 to 6	0	Controls the width of the *_ID ports of AXI_S
G4	C_BASEADDR	Integer	–	0xFFFF_FFFF	Base address of AXI_Exerciser
G5	C_HIGHADDR	Integer	–	0x0000_0000	End address of AXI_Exerciser
G6	C_ZERO_INVALID	Integer	–	1	When set, AXI_Exerciser attempts to drive zeroes on channels when *VALID is zero. This adds an additional LUT on the output paths, so at high-speed, the user might want to set this to zero.
G7	C_IS_AXI4	Integer	–	1	When set, indicates AXI_Exerciser is on an AXI4 bus. This is not directly used by AXI_Exerciser, but is made available to software in Reg5.
G8	C_IS_COHERENT	Integer	–	0	When set, indicates AXI_Exerciser is connected to a coherent slave (ACP) on Zynq-7000. This is not directly used by AXI_Exerciser, but this setting is made available to software in Reg5.
G9	C_IS_AFI	Integer	–	0	When set, indicates AXI_Exerciser is on an AFI bus. This is not directly used by AXI_Exerciser, but is made available to software in Reg5.
G10	C_NO_EXCL	Integer	–	0	When set, removes all support from AXI_S for exclusive access
G11	C_M_AXI_DATA_WIDTH	Integer	–	32	Sets M_AXI data width to 64 bits wide
G12	C_S_AXI_DATA_WIDTH	Integer	–	32	Sets S_AXI data width to 64 bits wide
G13	C_S_AXI_AWUSER_WIDTH	Integer	1 to 8	8	Set by AXI4 tools to the AWUSER width
G14	C_S_AXI_ARUSER_WIDTH	Integer	1 to 8	8	Set by AXI4 tools to the ARUSER width
G15	C_M_AXI_AWUSER_WIDTH	Integer	1 to 8	8	Set to the desired master AWUSER width
G16	C_M_AXI_ARUSER_WIDTH	Integer	1 to 8	8	Set to the desired master ARUSER width

SECTION II: VIVADO DESIGN SUITE

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.

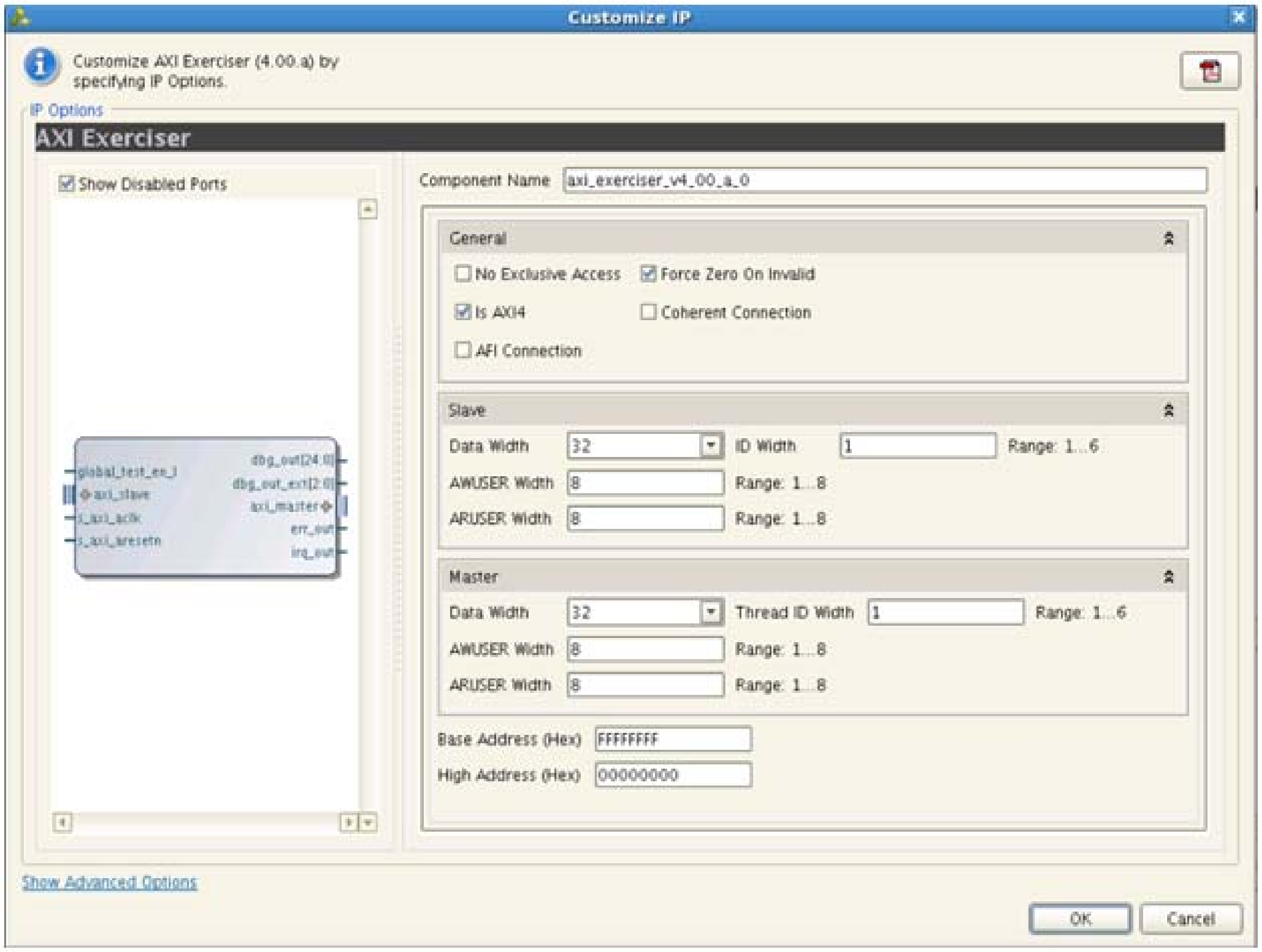
GUI

The AXI exerciser can be found in `/Embedded Processing/Debug & Verification/Debug` in the Vivado IP Catalog.

To access the AXI exerciser, perform the following:

1. Open a project by selecting **File** then **Open Project** or create a new project by selecting **File** then **New Project** in Vivado.
2. Open the IP catalog and navigate to any of the taxonomies.
3. Double-click on **AXI exerciser** to bring up the AXI exerciser GUI.

Figure 4-1 shows the AXI exerciser Customize IP window GUI with information about customizing ports.



Customize AXI Exerciser (4.00.a) by specifying IP Options.

☑ Show Disabled Ports

Component Name: axi_exerciser_v4_00_a_0

General

☐ No Exclusive Access ☑ Force Zero On Invalid

☑ Is AXI4 ☐ Coherent Connection

☐ AFI Connection

Slave

Data Width	32	ID Width	1	Range: 1..6
AWUSER Width	8	Range: 1..8		
ARUSER Width	8	Range: 1..8		

Master

Data Width	32	Thread ID Width	1	Range: 1..6
AWUSER Width	8	Range: 1..8		
ARUSER Width	8	Range: 1..8		

Base Address (Hex): FFFFFFFF

High Address (Hex): 00000000

Figure 4-1: **Vivado Customize IP GUI**

- **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 "_".

General

- **No Exclusive Access** – When set, removes all support from AXI_S for exclusive access.
- **Is AXI4** – When set, indicates AXI_Exerciser is on an AXI4 bus. This is not directly used by AXI_Exerciser, but is made available to software in Reg5.
- **AFI Connection** – When set, indicates AXI_Exerciser is on an AFI bus. This is not directly used by AXI_Exerciser, but is made available to software in Reg5.
- **Force Zero On Invalid** – When set, AXI_Exerciser attempts to drive zeroes on channels when *VALID is zero. This adds an additional LUT on the output paths, so at high-speed, the user might want to set this to zero.

Slave

- **Data Width** – S_AXI data bus width selection to 32 or 64 bits wide.
- **AWUSER Width** – Set by AXI4 tools to the AWUSER width with a range of 1 to 8.
- **ARUSER Width** – Set by AXI4 tools to the ARUSER width with a range of 1 to 8.
- **ID Width** – Controls the width of the *_ID ports of AXI_S with a range of 1 to 6.

Master

- **Data Width** – M_AXI data bus width selection to 32 or 64 bits wide.
- **AWUSER Width** – Sets to the desired master AWUSER width with a range of 1 to 8.
- **ARUSER Width** – Sets to the desired master ARUSER width with a range of 1 to 8.
- **Thread ID Width** – Controls the width of the *_ID ports of AXI_M with a range of 1 to 6.

Output Generation

This section provides detailed information about the files and the directory structure generated by the Xilinx Vivado tool.

The output files generated from the Xilinx Vivado IP Catalog are placed in the project directory. The file output list might include some or all of the following files.

The component name of the IP generated is axi_exerciser_v4_00_a_0.

<project name>/<project name.srcs>/ip
Top-level project directory; name is user-defined

- axi_exerciser_v4_00_a_0
  AXI Exerciser folders and files
- sim
  Simulation wrapper
- synth
  Synthesis wrapper
- axi_exerciser.veo/.vho
  Instantiation template files
- <axi_exerciser_v4_00_a_0>/axi_exerciser_v4_00_a/hdl/src/verilog
  AXI Exerciser RTL files

<project name>/<project name.srcs>/ip

The `project` directory contains the `axi_exerciser` core RTL files.

axi_exerciser_v4_00_a_0

The `axi_exerciser_v4_00_a_0` name directory contains the following folders and files.

Table 4-1: **axi_exerciser_v4_00_a_0 Directory**

Name	Description
<project name>/<project name.srcs/ip/axi_exerciser_v4_00_a_0	
sim	Simulation wrapper folder
synth	Synthesis wrapper folder
axi_exerciser.veo/.vho	Instantiation template files

Back to Top

<axi_exerciser_v4_00_a_0>/axi_exerciser_v4_00_a/hdl/src/verilog

The `axi_exerciser` RTL files are delivered under `/ip/axi_exerciser_v4_00_a/hdl/src/verilog` directory.

Chapter 5

Constraining the Core

There are no applicable constraints for this core.

XILINX

SECTION III: ISE DESIGN SUITE

Customizing and Generating the Core

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

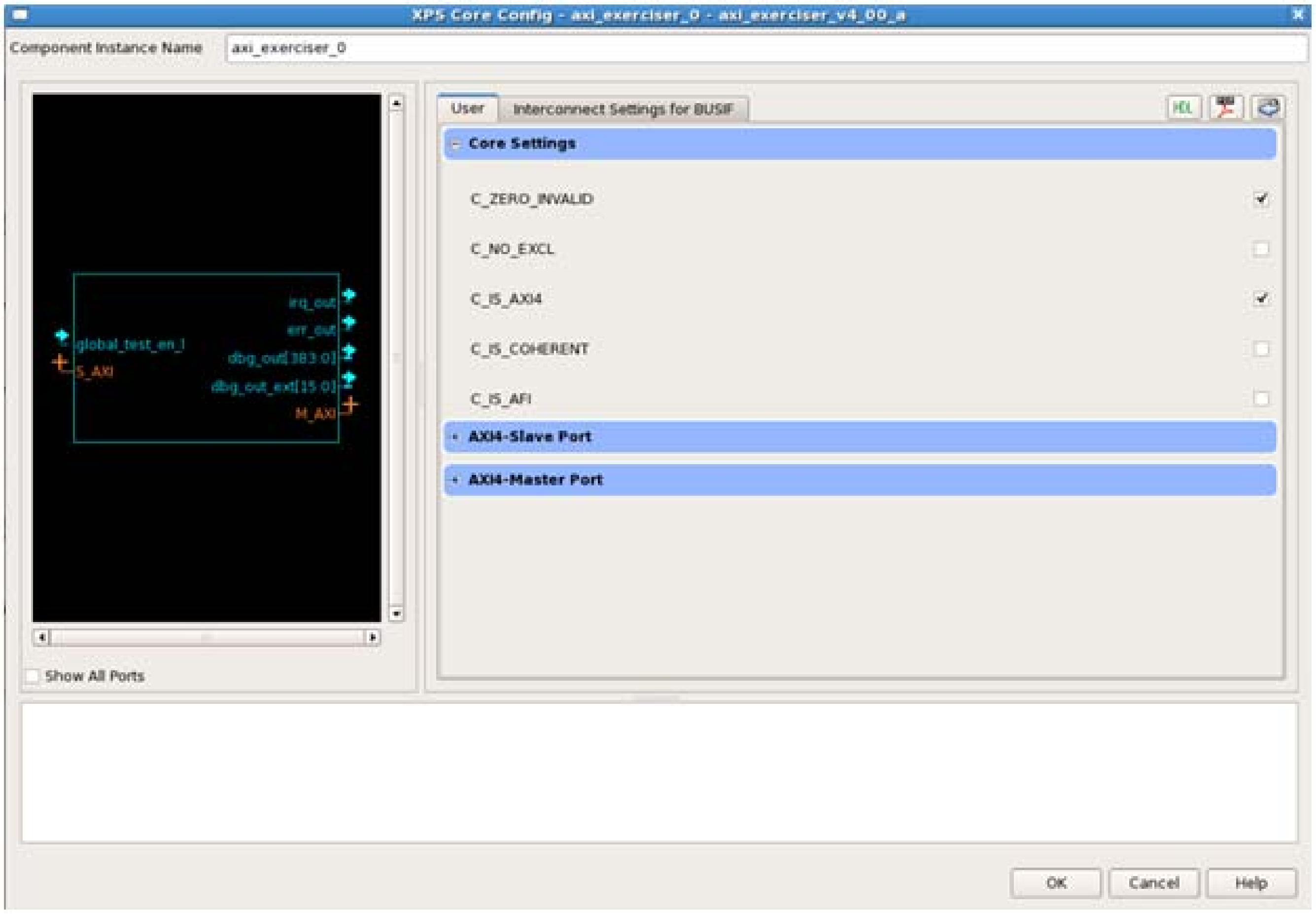
GUI

The AXI exerciser can be found in **Debug** in the Xilinx Platform Studio (XPS) tool GUI View by Function pane.

To access the AXI exerciser, do the following:

1. Open a project by selecting **File** then **Open Project** or create a new project by selecting **File** then **New BSB project**.
2. With an open project, choose **Debug** in the View by Function pane.
3. Double-click on **AXI exerciser**; this brings up the AXI exerciser GUI.

Figure 6-1 and Figure 6-2 show the AXI exerciser Core Configuration window GUIs with information about customizing core settings, AXI4-Slave and AXI4-Master ports.


XPS Core Config - axi_exerciser_0 - axi_exerciser_v4_00_a

Component Instance Name: axi_exerciser_0

Core Settings

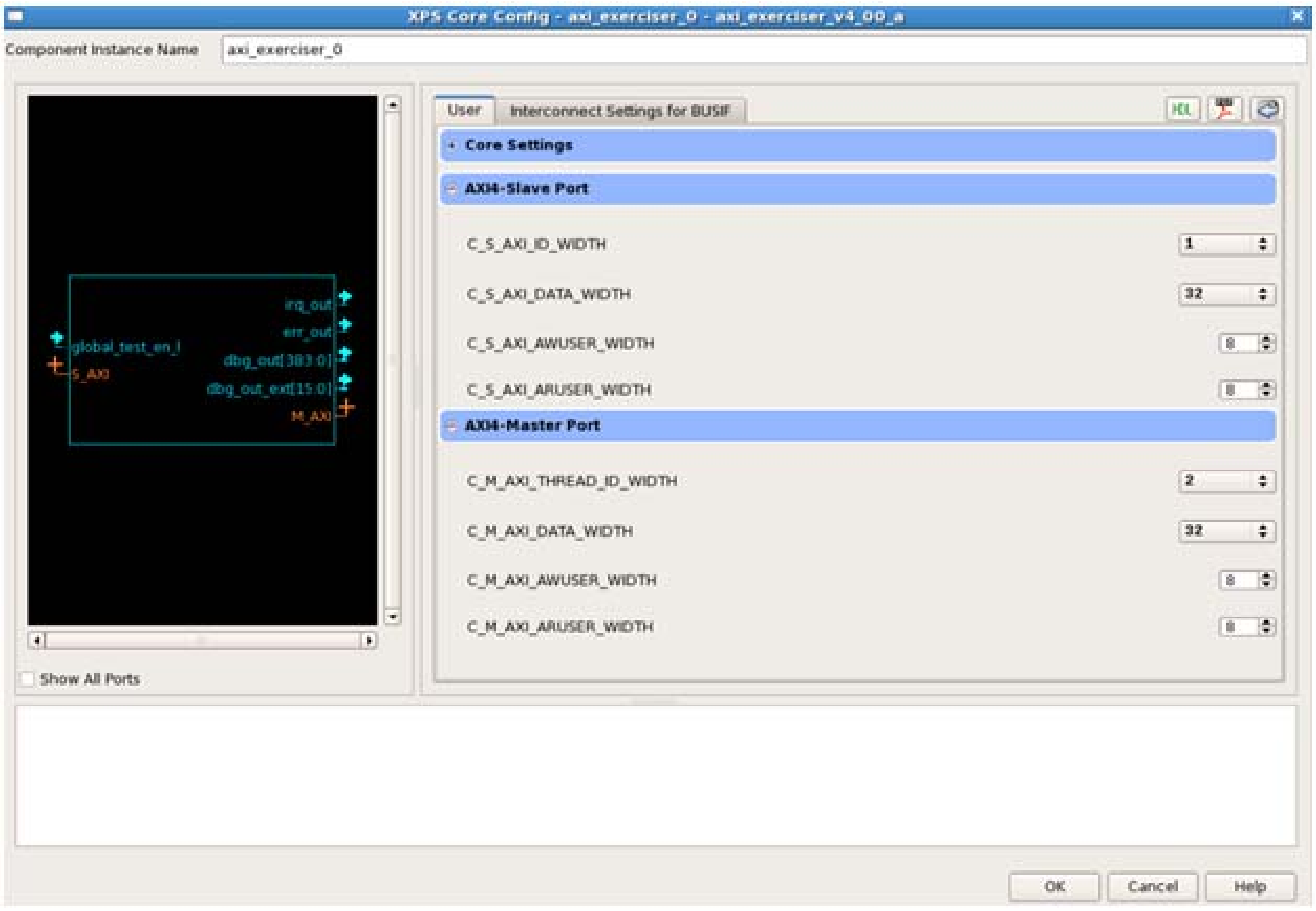
C_ZERO_INVALID	✓
C_NO_EXCL	☐
C_IS_AXI4	✓
C_IS_COHERENT	☐
C_IS_AFI	☐

Figure 6-1: **XPS/ISE Core Settings GUI**

- **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 "_".

Core Settings

- **C_ZERO_INVALID** – When set, AXI_Exerciser attempts to drive zeroes on channels when *VALID is zero. This adds an additional LUT on the output paths, so at high-speed, the user might want to set this to zero.
- **C_NO_EXCL** – When set, removes all support from AXI_S for exclusive access.
- **C_IS_AXI4** – When set, indicates AXI_Exerciser is on an AXI4 bus. This is not directly used by AXI_Exerciser, but is made available to software in Reg5.
- **C_IS_COHERENT** – When set, indicates AXI_Exerciser is connected to a coherent slave (ACP) on Zynq-7000. This is not directly used by AXI_Exerciser, but this setting is made available to software in Reg5.
- **C_IS_AFI** – When set, indicates AXI_Exerciser is on an AFI bus. This is not directly used by AXI_Exerciser, but is made available to software in Reg5.



Component Instance Name: axi_exerciser_0

C_S_AXI_ID_WIDTH: 1
C_S_AXI_DATA_WIDTH: 32
C_S_AXI_AWUSER_WIDTH: 8
C_S_AXI_ARUSER_WIDTH: 8
C_M_AXI_THREAD_ID_WIDTH: 2
C_M_AXI_DATA_WIDTH: 32
C_M_AXI_AWUSER_WIDTH: 8
C_M_AXI_ARUSER_WIDTH: 8

Figure 6-2: **XPS/ISE AXI4-Slave and AXI4-Master Ports GUI**

AXI4-Slave Port

- **C_S_AXI_ID_WIDTH** – Controls the width of the *_ID ports of AXI_S with a range of 1 to 6.
- **C_S_AXI_DATA_WIDTH** – S_AXI data bus width selection to 32 or 64 bits wide.
- **C_S_AXI_AWUSER_WIDTH** – Set by AXI4 tools to the AWUSER width with a range of 1 to 8.
- **C_S_AXI_ARUSER_WIDTH** – Set by AXI4 tools to the ARUSER width with a range of 1 to 8.

AXI4-Master Port

- **C_M_AXI_THREAD_ID_WIDTH** – Controls the width of the *_ID ports of AXI_M with a range of 1 to 6.
- **C_M_AXI_DATA_WIDTH** – M_AXI data bus width selection to 32 or 64 bits wide.

- **C_M_AXI_AWUSER_WIDTH** – Sets to the desired master AWUSER width with a range of 1 to 8.
- **C_M_AXI_ARUSER_WIDTH** – Sets to the desired master ARUSER width with a range of 1 to 8.

Output Generation

The output files generated from the XPS tool are placed in the project directory. The file output list can include some or all of the following files.

The output generation from the XPS tool is standard and is available at http://www.xilinx.com/itp/xilinx10/help/platform_studio/ps_r_gst_xps_project_directories.htm.

Chapter 7

Constraining the Core

There are no applicable constraints for this core.

SECTION IV: APPENDICES

Migrating

This appendix describes migrating from older versions of the IP to the current IP release.

For information on migrating to the Vivado™ Design Suite, see UG911, *Vivado Design Suite Migration Methodology Guide* [Ref 1].

Debugging

This appendix includes details about resources available on the Xilinx Support website and debugging tools. In addition, this appendix provides a step-by-step process for debugging process and a flow diagram to guide you through debugging the AXI exerciser core.

The following topics are included in this appendix:

- Finding Help on Xilinx.com
- Debug Tools
- Hardware Debug
- Interface Debug

Finding Help on Xilinx.com

To help in the design and debug process when using the AXI exerciser, the Xilinx Support web page 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 exerciser. 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.

Release Notes

Known issues for all cores, including the AXI exerciser are described in the IP Release Notes Guide (XTP025).

Known Issues

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

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

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

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

Answer Records for the AXI Exerciser

There are no known issues or Answer Records for the AXI exerciser 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.

Debug Tools

There are many tools available to address AXI exerciser design issues. It is important to know which tools are useful for debugging various situations.

ChipScope Pro Tool

The ChipScope™ Pro tool inserts logic analyzer, bus analyzer, and virtual I/O cores directly into your design. The ChipScope Pro tool allows you to set trigger conditions to capture application and integrated block port signals in hardware. Captured signals can then be analyzed through the ChipScope Pro Logic Analyzer tool. For detailed information for using the ChipScope Pro tool, see www.xilinx.com/tools/cspro.htm.

Reference Boards

Various Xilinx development boards support AXI exerciser. These boards can be used to prototype designs and establish that the core can communicate with the system.

- 7 series evaluation boards
  - KC705
  - VC707

License Checkers

If the IP requires a license key, the key must be verified. The ISE and Vivado tool flows have a number of license check points for gating licensed IP through the flow. If the license check succeeds the IP may continue generation, otherwise generation halts with error. License checkpoints are enforced by the following tools:

- ISE flow: XST, NgdBuild, Bitgen
- Vivado flow: Vivado Synthesis, Vivado Implementation, write_bitstream (Tcl command)

IMPORTANT: *IP license level is ignored at checkpoints. The test confirms a valid license exists. It does not check IP license level.*

Hardware Debug

Hardware issues can range from link bring-up to problems seen after hours of testing. This section provides debug steps for common issues. The ChipScope tool is a valuable resource to use in hardware debug. The signal names mentioned in the following individual sections can be probed using the ChipScope tool for debugging the specific problems.

Many of these common issues can also be applied to debugging design simulations. Details are provided in the General Checks section.

General Checks

Ensure that all the timing constraints for the core were properly incorporated from the example design and that all constraints were met during implementation.

- Does it work in post-place and route timing simulation? If problems are seen in hardware but not in timing simulation, this could indicate a PCB issue. Ensure that all clock sources are active and clean.
- If using MMCMs in the design, ensure that all MMCMs have obtained lock by monitoring the LOCKED port.
- If your outputs go to 0, check your licensing.
- If the core is not generating any transactions on Write/Read master interfaces:
  a. Ensure valid_cmd bits are set properly while loading commands to command RAM.
  b. Check My_depend, Other depend fields are set correctly to not to cause dead-lock situation.
  c. Check delay values programmed to PARAMRAM and wait for sufficient time for the core to insert these delays while generating the transactions.
- If the register control bit (reg0_m_enable) is not getting deasserted:
  a. Ensure valid_cmd bits are set properly while loading commands to command RAM.
  b. Check Reg0_master_control [19:0] are set to 0.
  c. Check delay values programmed to PARAMRAM and wait for sufficient time for the core to insert these delays while generating the transactions.

Interface Debug

AXI4 Interface

Read from a register that does not have all 0s (for example, Reg0_master_control) as a default to verify that the interface is functional. See Figure B-1 for a read timing diagram. Output `s_axi_arready` asserts when the read address is valid, and output `s_axi_rvalid` asserts when the read data/response is valid. If the interface is unresponsive, ensure that the following conditions are met:

- The `S_AXI_ACLK` input is connected and toggling.
- The interface is not being held in reset, and `S_AXI_ARESET` is an active-Low reset.
- The main core clocks are toggling and that the enables are also asserted.

- If the simulation has been run, verify in simulation and/or a ChipScope tool capture that the waveform is correct for accessing the AXI4 interface.

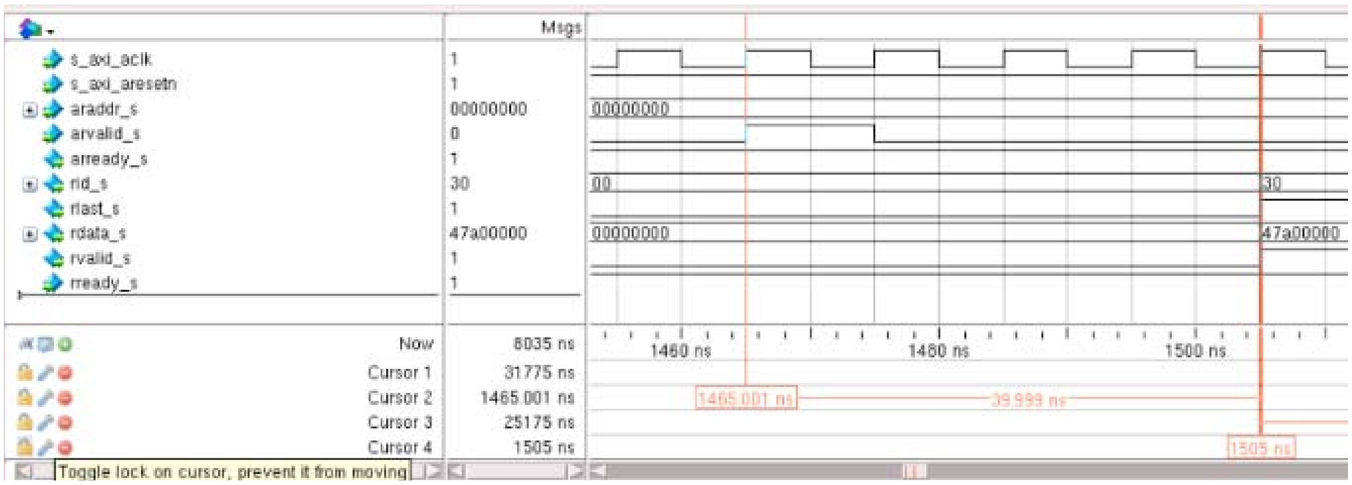


Figure B-1: **AXI4 Read Timing Diagram**

Additional Resources

Xilinx Resources

For support resources such as Answers, Documentation, Downloads, and Forums, see Appendix B, Debugging and 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

Unless otherwise noted, IP references are for the product documentation page. These documents provide supplemental material useful with this product guide:

1. Vivado™ Design Suite user documentation (www.xilinx.com/cgi-bin/docs/rdoc?v=2012.4;t=vivado+docs)
2. *ARM® AXI4 Memory Mapped Specification*
3. *ARM AMBA AXI Protocol*, version 2.0 (ARM IHI 0022C) (http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ihi0022c/index.html)

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.

See the IP Release Notes Guide (XTP025) for more information on this core. For each core, there is a master Answer Record that contains the Release Notes and Known Issues list for the core being used. The following information is listed for each version of the core:

- New Features
- Resolved Issues
- Known Issues

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
12/18/12	1.0	Initial Xilinx release.

Notice of Disclaimer