LogiCORE IP AXI Traffic Generator v1.0

Product Guide for Vivado Design Suite

PG125 March 20, 2013

XILINX®

XILINX

Table of Contents

IP Facts

Chapter 1: Overview

Modes Description 7
Programming Sequence 20
Applications 21
Licensing and Ordering Information 22

Chapter 2: Product Specification

Performance 23
Resource Utilization 26
Port Descriptions 29
Register Space 31

Chapter 3: Designing with the Core

Clocking 39
Resets 39

Chapter 4: Customizing and Generating the Core

Vivado Integrated Design Environment (IDE) 40

Chapter 5: Constraining the Core

Appendix A: Migrating

Appendix B: Debugging

Finding Help on Xilinx.com 46
Debug Tools 48
Hardware Debug 48
Interface Debug 49

Appendix C: Additional Resources

Xilinx Resources 51

References **51**
Revision History **52**
Notice of Disclaimer **52**

Introduction

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

Features

- AXI4 interface for register access and data transfers
- Multi-Mode operation (AXI4 Master, AXI4-Lite Master, and AXI4-Stream Master)
- Flexible data width capability (32/64-bit) on Slave and (32/64/128/256/512-bit) on Master AXI4 interface
- Supports AXI4-Lite Master interface for system initialization in processor-less system
- Interrupt support for indicating completion for traffic generation.
- 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, Virtex®-7, Kintex™-7, Artix™-7
Supported User Interfaces	AXI4, AXI4-Stream, AXI4-Lite
Resources	See Table 2-2 to Table 2-4.
Provided with Core	
Design Files	Verilog
Example Design	Not Provided
Test Bench	Not Provided
Constraints File	XDC
Simulation Model	Not Provided
Supported S/W Driver(2)	Standalone
Tested Design Flows(3)	
Design Entry	Vivado™ Design Suite
Simulation	Mentor Graphics Questa® SIM, Vivado Simulator
Synthesis	Vivado Synthesis
Support	
Provided by Xilinx @ www.xilinx.com/support	

Notes:

1. For a complete listing of supported devices, see the Vivado IP catalog.
2. Standalone driver details can be found in the SDK directory (<install_directory>/doc/usenglish/xilinx_drivers.htm). Linux OS and driver support information is available from //wiki.xilinx.com.
3. For the supported versions of the tools, see the Xilinx Design Tools: Release Notes Guide.

Overview

The AXI Traffic Generator is fully synthesizable AXI4-complaint core with the following features:

- Configurable option to generate and accept data according to different traffic profiles
- Supports dependent/independent transaction between read/write master port with configurable delays
- Programmable repeat count for each transaction with constant/increment/random address

IMPORTANT: *This product guide replaces the PG094 LogiCORE™ IP AXI Exerciser.*

The core generates AXI4, AXI4-Lite, or AXI4-Stream based on the mode selected. There are six different modes the core can be configured and it is detailed in Table 1-1.

Table 1-1: **AXI Traffic Generator Modes**

Mode	Traffic Type	Description
Advanced	AXI4	AXI4 support.
Basic	AXI4	Lightweight mode with basic AXI-MM features support (for example, narrow/ unaligned, out of order transfers not supported).
Static	AXI4	Simple AXI4 traffic generator mode with Fixed address, incremental transactions based on UI configuration. Minimum processor overhead.
System Init	AXI4-Lite	AXI4-Lite master write interface for system initialization. Transactions initiated based on memory initialization file.
Streaming	AXI4-Stream	AXI4-Stream interface with Master, Slave, and Loopback mode option.

The architecture of the core is broadly separated into a Master and Slave block, each contains the Write and Read blocks. Other support functions are provided by the Control registers and Internal RAMs.

Figure 1-1 shows the top-level AXI4 Traffic Generator block diagram.

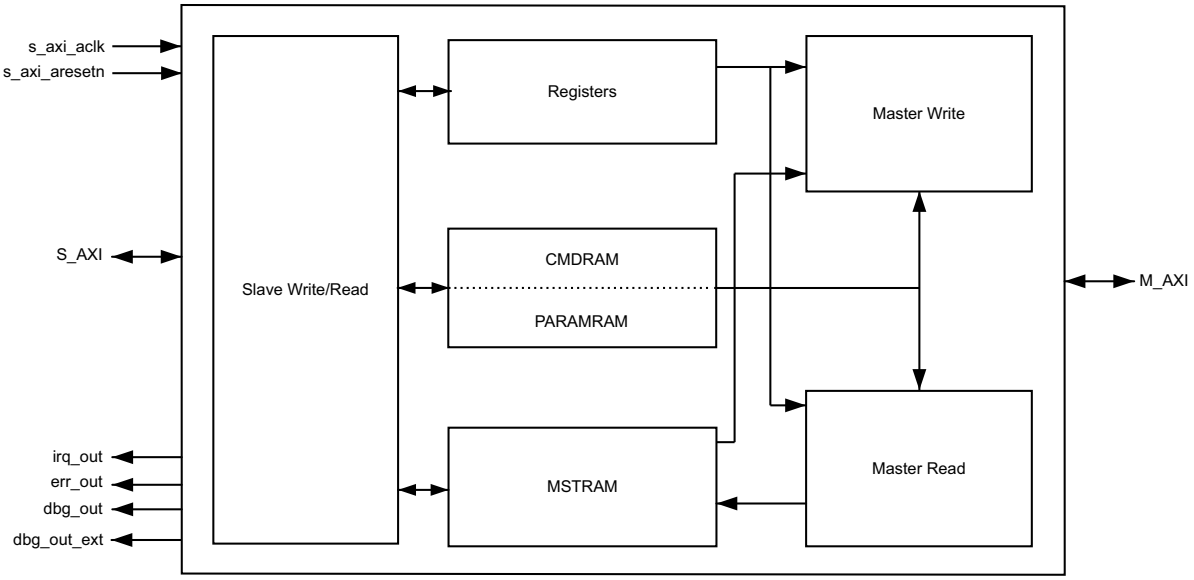

Figure 1-1: **AXI4 Traffic Generator Block Diagram**

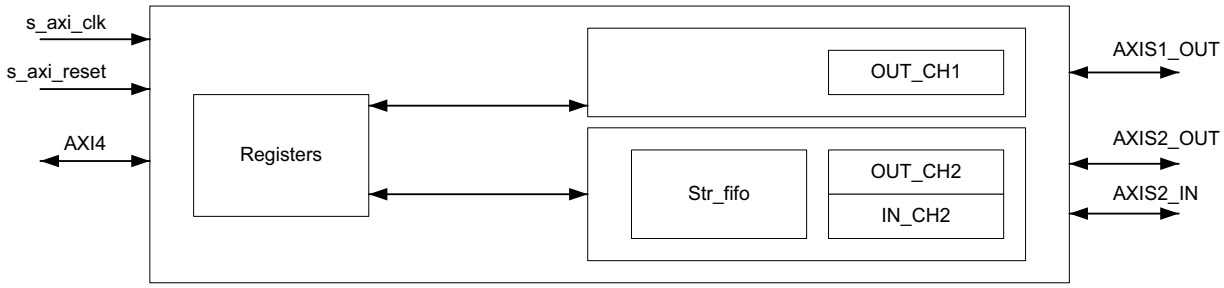

Figure 1-2: **AXI4-Stream Traffic Generator Block Diagram**

Modes Description

The AXI Traffic Generator has five different modes that include:

- Advanced Mode
- Basic Mode
- Static Mode
- System Init Mode
- Streaming Mode

Advanced Mode

Advanced mode allows full control over the traffic generation. Control registers are provided to you 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 1-2 is 128 bits wide.

Table 1-2: **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])
+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. **64-bit mode**: 000 = All bytes valid 001 = Only Bit[0] is valid 010 = Only Bits[1:0] are valid and so on. **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 zero 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 1-2: **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

Notes:

1. Valid_cmd: There should be at least one command with valid_cmd bit set to zero 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 1-3 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.

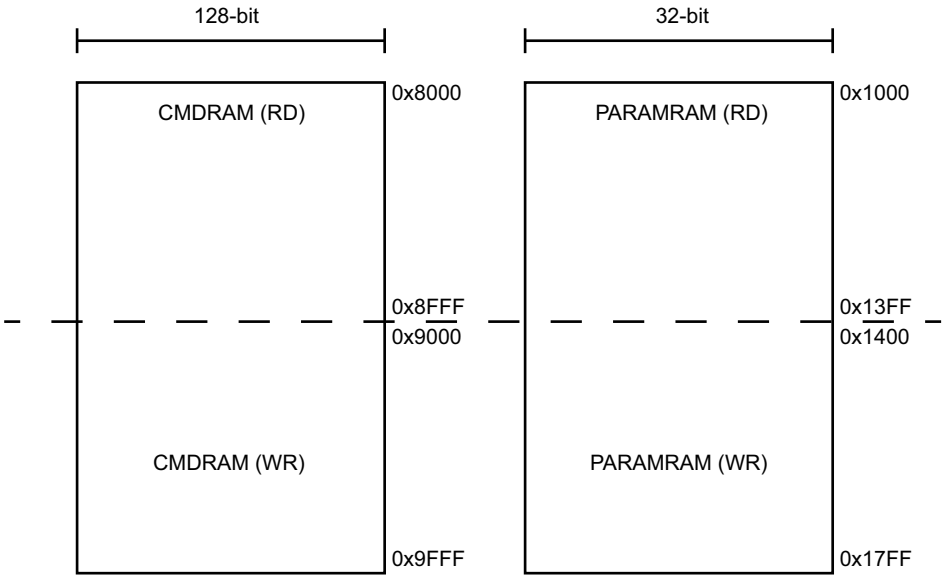

Figure 1-3: **PARAMRAM vs. CMDRAM**

Each entry in the PARAMRAM modifies its corresponding CMDRAM entry. The encoding and opcodes are described in Table 1-3 to Table 1-7.

Table 1-3: **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]

Table 1-3: **PARAMRAM Entry Control Signals *(Cont'd)***

Bits	Name	Description
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 1-7.

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 1-7.

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

Table 1-4: **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 1-5: **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 1-6: **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 1-7: **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 you.
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 zero.

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 1-4.

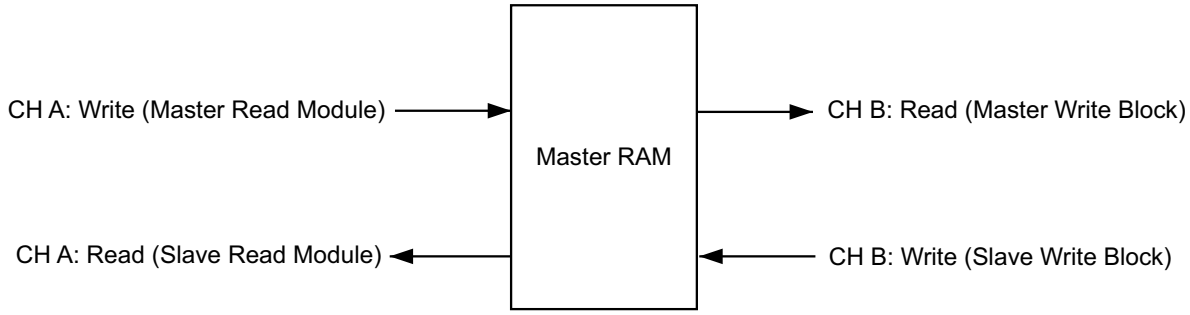

Figure 1-4: **Master RAM Channels**

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

Figure 1-5 shows the slave logic.

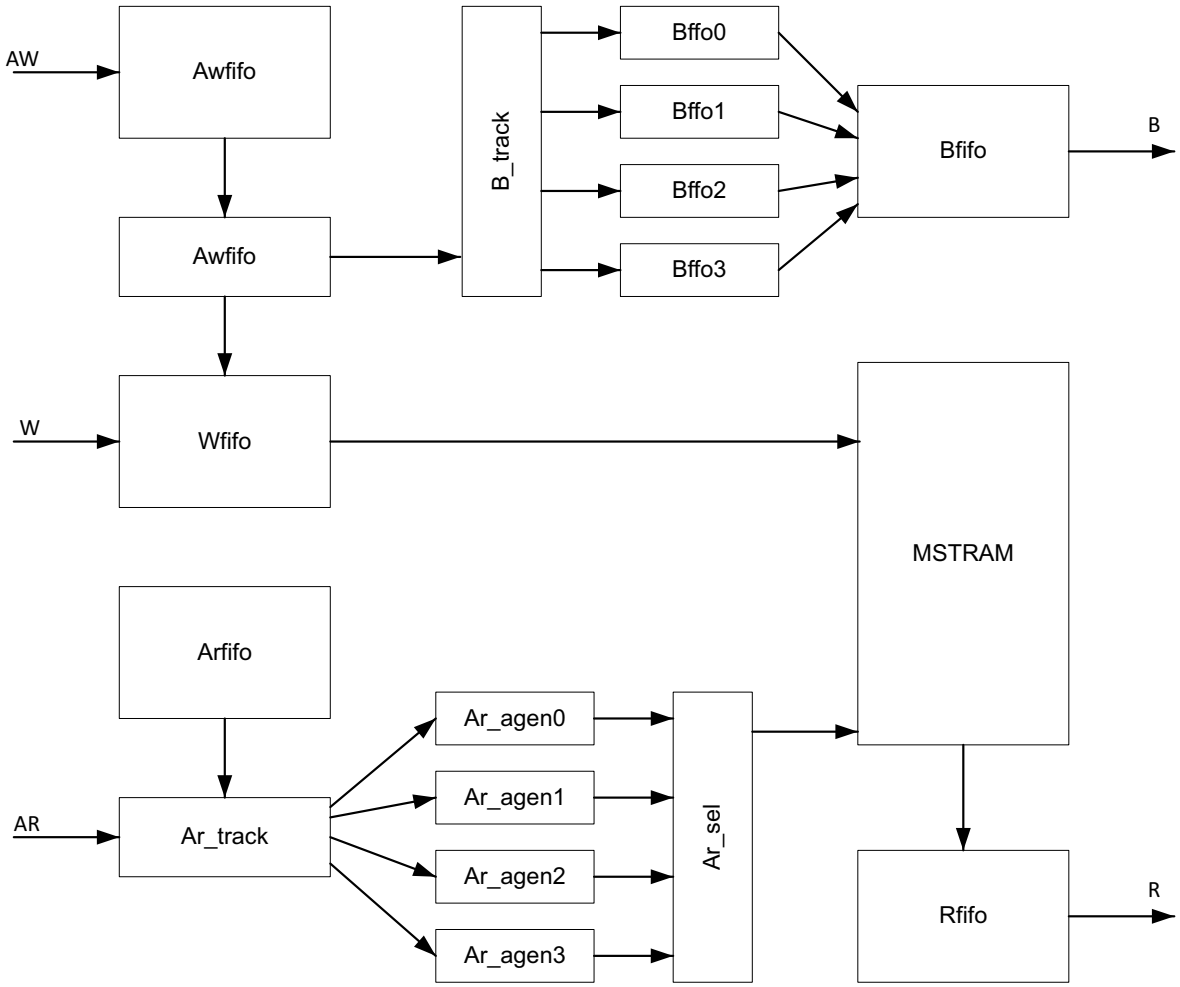

Figure 1-5: **AXI_Traffic_Generator Slave**

- **Slave-write** – The slave AR, AW, and W ports each have a FIFO to collect data from the switch. The output buses B and R also use a small FIFO to buffer their outgoing data. The write addresses from the AW bus then goes to an Aw_agen block which generates the proper MSTRAM addresses and writes the corresponding data word to the MSTRAM. After a transaction is complete, the ID information is passed to the B_track tracker which writes the completion ID to one of Bfifo0 to BFIFO3. These Bfifos then arbitrate to write the completions into the final Bfifo, allowing the creation of out-of-order write responses.
- **Slave-read** – Read addresses are placed in the Arfifo which then use the Ar_track tracker to distribute the requests across Ar_agen0 to Ar_agen3. These generate the proper addresses to the MSTRAM for each single request. The Ar_agen0 to Ar_agen3 arbitrates for access to the MSTRAM at each cycle in the Ar_sel block. The data is placed in the small Rfifo and then driven to the switch on the R bus.

Table 1-8 shows the address map for different regions accessed by slave-write/slave-read.

Table 1-8: **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_Traffic_Generator ignores address Bits[31:23] for its decode, but decodes Bits[22:0].

For slave logic the write interleaving depth is one.

Master Modules

Figure 1-6 shows the master logic.

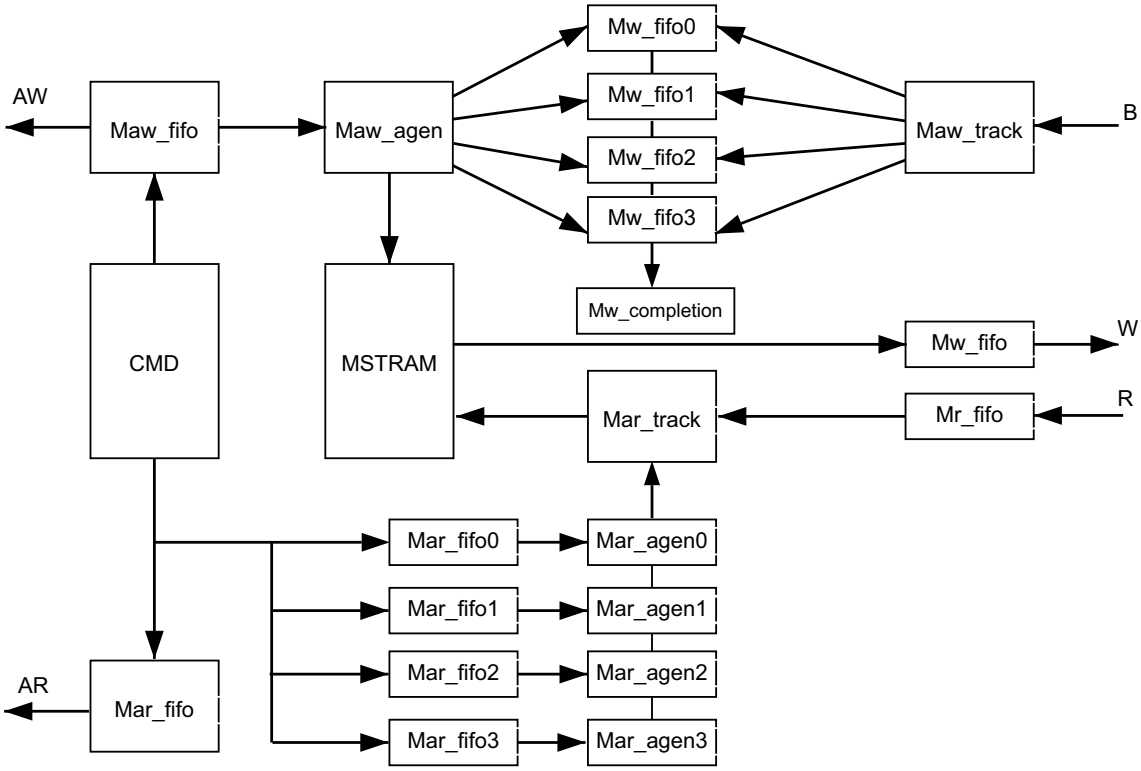

Figure 1-6: **AXI_Traffic_Generator Master**

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 to Mar_fifo3 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. The Mar_fifo0 to Mar_fifo3 hold the requests before sending them to Mar_agen0 to Mar_agen3. 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 to Mar_fifo3 is empty. Read data returned from the switch is placed in Mr_fifo, then popped out. Each ID is searched across each Mar_agen0 to Mar_agen3, 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 to Mar_fifo3 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 to 3. When that ID receives a BRESP, it pops the corresponding Mw_fifo0 to 3. This allows the master write logic to handle receiving BRESPs out of order.

Basic Mode

Basic mode allows basic AXI4 traffic generation with less resource overhead. This mode is a lightweight version of the advanced mode with the following AXI features not supported:

- Out-of-order transactions
- Narrow transfers
- Holes in write strobe

Table 1-9 shows the ports that are tied/assumed to default value.

Table 1-9: **Default Ports**

Port	Description
Lock = 0	No exclusive access.
Burst = 1	Only INCR transfers.
Prot = 0	Only Data access.
Cache = 3	Cache signals driven to zero.
User = 0	User signals driven to zero.
Qos = 0	QoS signals driven to zero.
Size	Full data width support. For example, 2 for 32-bit data width, 3 for 64-bit data width, and others.

PARAMRAM features are not supported in this mode.

RECOMMENDED: *You are recommended to write the default values in Table 1-9 into the command RAM when programming the command RAM entries.*

Static Mode

Static mode allows you to generate a simple AXI4 traffic with very less resource overhead and minimum processor intervention. In static mode, after the core is enabled using the Static Control register as defined in Register Space, page 31, the core continuously generates fixed address and fixed length INCR type read and writes transfers.

You can configure the Read/Write address based on the system configuration and the transfer length from GUI parameters. Transfer length can also be configured through the Static Length register. Read or Write channels can be enabled separately from the GUI parameter. This mode can be used to stress interconnect and other modules in a system. The Burst Length, Data Width, and Start Address should be selected by you such that the transaction do not cross the 4K boundary.

System Init Mode

System Init mode is a special mode where core provides only AXI4-Lite Master write interface. This mode can be used in a system without a processor to initialize the system peripherals with preconfigured values on system reset.

In System Init mode, after the core comes out of reset, it reads the COE files (address and data) from the ROM and generates AXI4-Lite transactions. You have to provide two COE files for this mode.

- **Address COE File** – Provides the sequence of addresses to be issued
- **Data COE File** – Provides the sequence of data corresponding to the address specified in Address COE File

You need to fill the entries in the COE files to match the requirement. First entry address entry in Address COE file corresponds to first data entry in the Data COE file.

Operation

1. After ATG comes out of reset, it reads the ADDR and DATA ROMs.
2. It initiates AXI4-Lite write transactions to a specified address and data in the COE files.
3. The core goes to idle state after AXI4-Lite transactions are issued.

Figure 1-7 shows the example use-case where ATG (System Init mode) is used to initialize peripherals in a system without a processor.

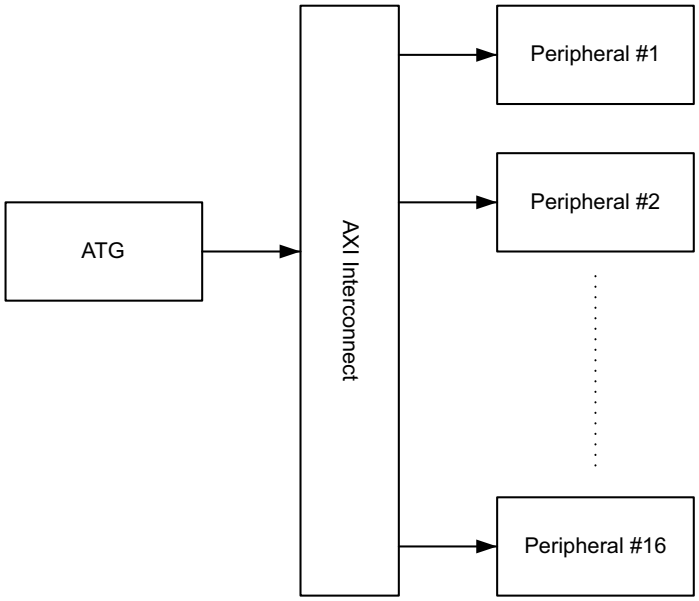

Figure 1-7: **System Init Mode Block Diagram**

The number of entries in the COE file are user programmable. Allowed values are 16, 32, 64, 128, and 256. You can insert NOP (No Operation) defined by address (0xFFFFFFFF) in the middle of a COE address file. The core stops generating further transactions (including the current NOP address of 0xFFFFFFFF) after NOP address is present.

Streaming Mode

The streaming mode provides the following:

- Master interface
- Loopback interfaces

Master Interface

The core can be configured to generate streaming traffic based on the register configuration. See the register description for streaming mode on the possible traffic profiles that can be generated.

Loopback Interfaces

The core uses an internal FIFO of depth 14, which first takes the streaming packet from slave streaming port-2 and loops back the same packet on master streaming port-2.

IMPORTANT: *Ensure that the internal FIFO does not overflow.*

Programming Sequence

Programming Sequence for Advanced/Basic Mode

1. Load CMDRAM RAM with the required number of commands.
2. Load PARAM RAM with the desired opcodes. PARAM RAM applicable only in Advanced mode.
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.

Programming Sequence for Static Mode

1. Select the desired Address/length from the GUI while generating the core.
2. Enable the core by writing to Bit[0] of Static Control register.
3. To change the burst length, disable the core, program new burst length in Static Length register, and re-enable the core.

Programming Sequence for System Init Mode

1. Provide the required COE files during core customization from the GUI.
2. This initiates the AXI4-Lite transactions on the Master interface when the core comes out of reset.

Applications

The AXI Traffic Generator can be connected to an AXI-based system to stress the modules connected to the interconnect.

Figure 1-8 shows the AXI Traffic Generator connected to a MicroBlaze™ processor. The MicroBlaze programs the AXI Traffic Generator and generates traffic to other cores.

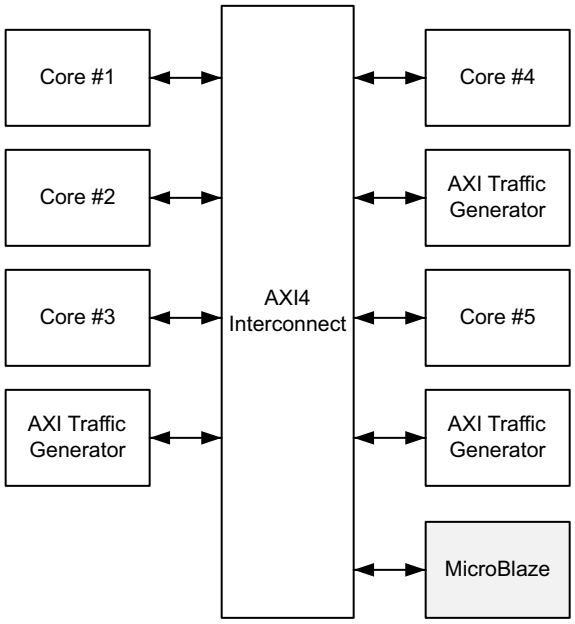

Figure 1-8: **MicroBlaze System**

Figure 1-9 shows the AXI Traffic Generator connected to a Zynq™ platform, The AXI Traffic Generator can be programmed from ARM® to the exerciser AXI peripherals.

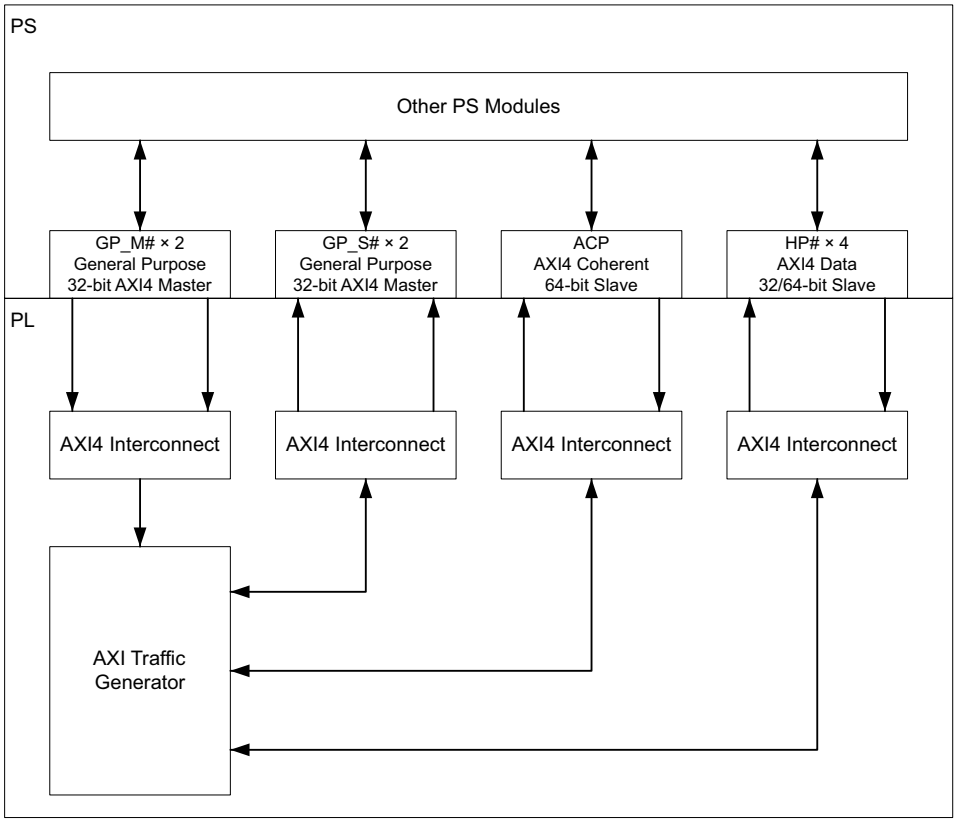

Figure 1-9: **Zynq System**

Licensing and Ordering Information

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

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

Product Specification

Performance

Performance characterization of this core has been done using margin system methodology. The details of the margin system characterization methodology is described in Appendix A, "IP Characterization and F_{Max} Margin System Methodology," of the *Vivado Design Suite User Guide: Designing With IP* (UG896) [Ref 2].

Table 2-1: **Maximum Frequencies**

Family	Speed Grade	F_{Max} (MHz)		
		AXI4	AXI4-Lite	AXI4-Stream
Virtex-7	-1	200	150	200
Kintex-7		180	150	180
Artix-7		150	100	150
Virtex-7	-2	220	150	220
Kintex-7		200	150	200
Artix-7		160	100	160
Virtex-7	-3	250	150	250
Kintex-7		220	150	220
Artix-7		170	100	170

Latency

The AXI Traffic Generator 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 Master Control register and the AWVALID assertion on Master Ports. Latency is nine clock cycles with delay parameters in PARAMRAM set to zero.

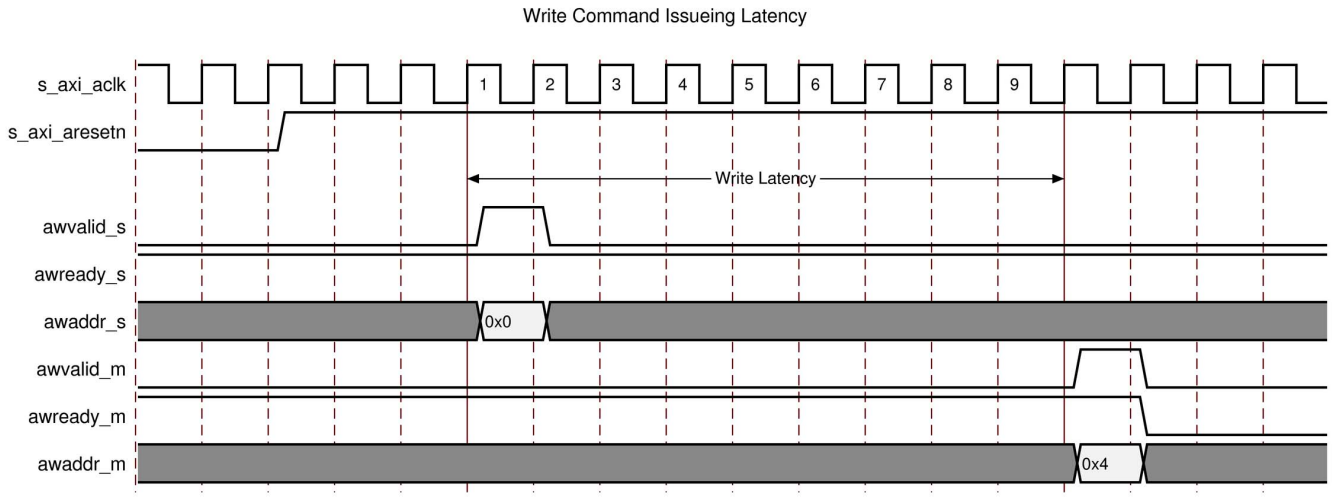

Figure 2-1: **Write Command Issuing Latency**

Read Command Issuing Latency

Latency is calculated from the point where the core is enabled by writing to Master Control register and the ARVALID assertion on Master Ports. Latency is nine clock cycles with delay parameters in PARAMRAM set to zero.

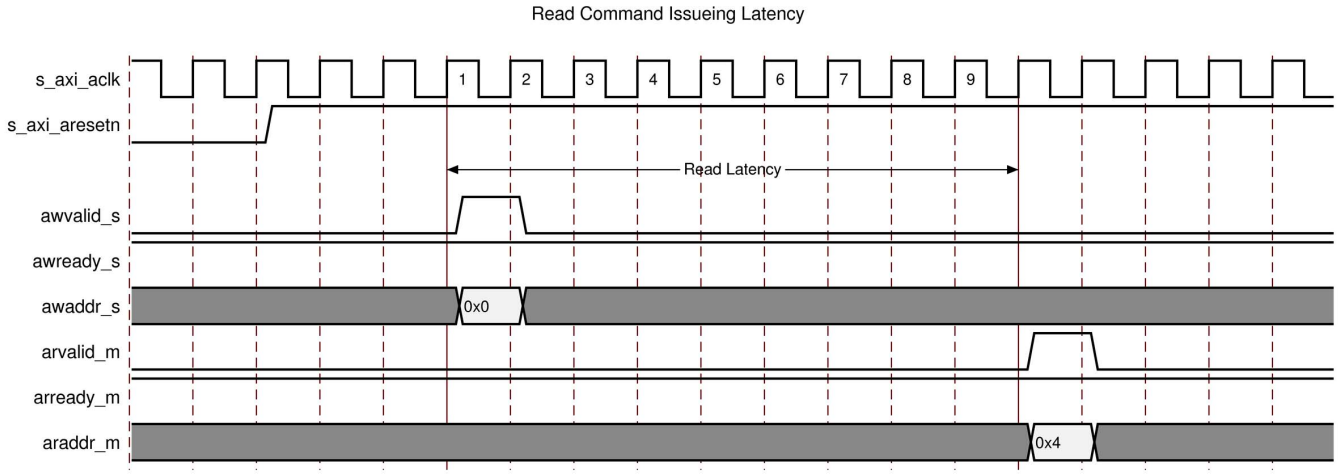

Figure 2-2: **Read Command Issuing Latency**

Throughput

The AXI Traffic Generator 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.

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

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

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

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

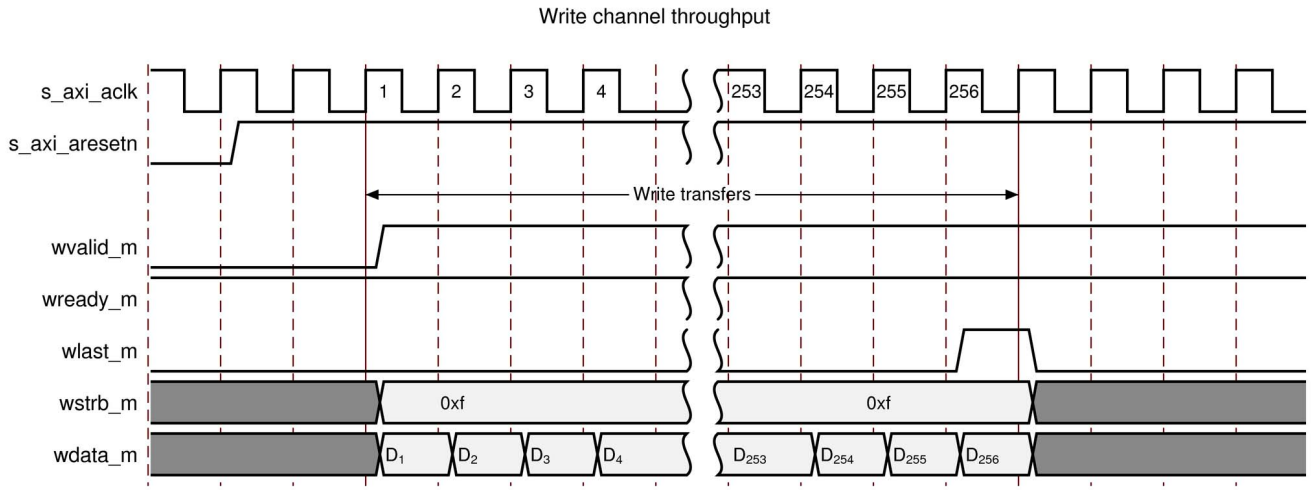

Figure 2-3: **Master Write Channel Throughput**

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.

$$\text{Throughput} = (A - B) \times 100 / (\text{Total beats in the transaction}) \quad \textit{Equation 2-3}$$

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

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

$$\text{Throughput} = (256 - 0) \times 100 / 256 = 100\% \quad \textit{Equation 2-4}$$

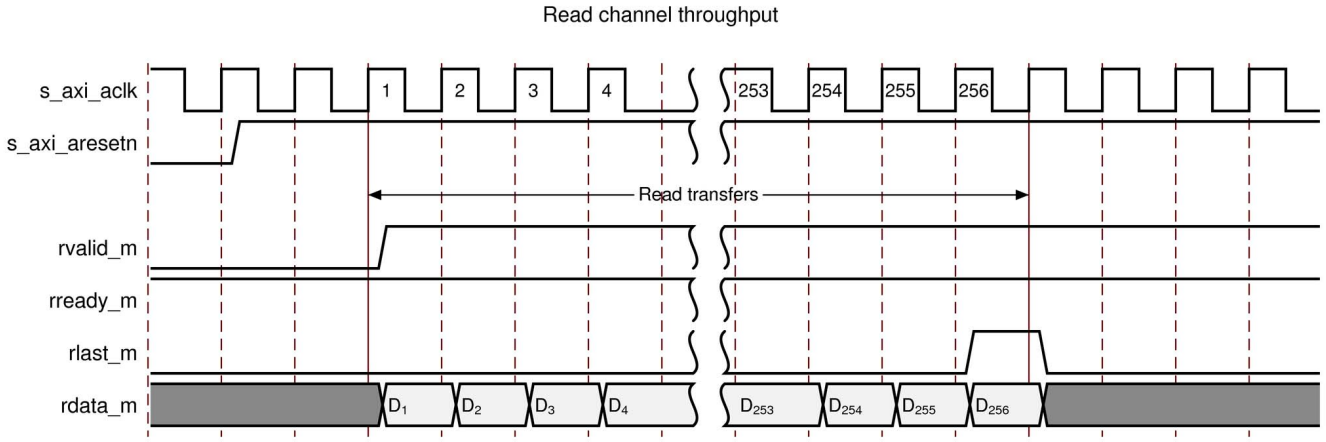

Figure 2-4: **Master Read Channel Throughput**

Resource Utilization

Resources required for the AXI Traffic Generator core have been estimated for the 7 series FPGAs (Table 2-2 to Table 2-4). These values were generated using Vivado™ IP Catalog. They are derived from post-synthesis reports, and might change during MAP and PAR.

Note: Resources numbers for Zynq™-7000 devices are expected to be similar to 7 series device numbers.

The AXI Traffic Generator resource utilization for various parameter combinations measured with a Virtex®-7 FPGA as the target device are detailed in Table 2-2.

Table 2-2: **Resource Estimation for Virtex-7 (XC7VX690TFFG1927-1) FPGAs**

Parameter Values		Device Resources	
ATG Mode	**Data Width**	**Slice LUTs**	**Slice Registers**
Advanced	32	4,224	3,457
	64	4,413	3,598
	128	5,083	3,724
	256	5,805	3,980
	512	7,749	4,492
Basic	32	1,951	1,973
	64	2,070	2,069
	128	2,150	2,134
	256	2,298	2,200
	512	2,247	2,201

Table 2-2: **Resource Estimation for Virtex-7 (XC7VX690TFFG1927-1) FPGAs *(Cont'd)***

Parameter Values		Device Resources	
ATG Mode	**Data Width**	**Slice LUTs**	**Slice Registers**
Static	32	112	205
	64	112	205
	128	116	205
	256	116	205
	512	116	205
Streaming	32	261	351
	64	310	391
	128	400	471
	256	600	631
	512	969	951
System Init	32	26	37

The AXI Traffic Generator resource utilization for various parameter combinations measured with a Kintex™-7 FPGA as the target device are detailed in Table 2-3.

Table 2-3: **Resource Estimates for Kintex-7 (XC7K480TFFG1156-1) FPGAs**

Parameter Values		Device Resource	
ATG Mode	**Data Width**	**Slice LUTs**	**Slice Registers**
Advanced	32	4,219	3,457
	64	4,416	3,598
	128	5,085	3,724
	256	5,804	3,980
	512	7,723	4,492
Basic	32	1,953	1,973
	64	2,067	2,069
	128	2,154	2,134
	256	2,297	2,200
	512	2,241	2,201
Static	32	114	205
	64	114	205
	128	116	205
	256	116	205
	512	116	205

Table 2-3: **Resource Estimates for Kintex-7 (XC7K480TFFG1156-1) FPGAs *(Cont'd)***

Parameter Values		Device Resource	
ATG Mode	**Data Width**	**Slice LUTs**	**Slice Registers**
Streaming	32	261	351
	64	310	391
	128	400	471
	256	600	631
	512	969	951
System Init	32	26	37

The AXI Traffic generator resource utilization for various parameter combinations measured with a Artix™-7 FPGA as the target device are detailed in Table 2-4.

Table 2-4: **Resource Estimates for Artix-7 (XC7A200TFFG1156-1) FPGAs**

Parameter Values		Device Resource	
ATG Mode	**Data Width**	**Slice LUTs**	**Slice Registers**
Advanced	32	3,989	3,457
	64	4,155	3,598
	128	4,765	3,724
	256	5,625	3,980
	512	7,708	4,492
Basic	32	1,810	1,973
	64	1,904	2,069
	128	1,985	2,134
	256	2,124	2,200
	512	2,095	2,201
Static	32	116	205
	64	116	205
	128	118	205
	256	118	205
	512	118	205
Streaming	32	253	351
	64	302	391
	128	398	471
	256	600	631
	512	963	951
System Init	32	26	37

Port Descriptions

The AXI Traffic Generator signals are listed and described in Table 2-5.

Table 2-5: **AXI Traffic Generator I/O Signals**

Signal Name	Interface	I/O	Description
System Signals			
s_axi_aclk	System	I	Clock
s_axi_aresetn	System	I	Active-Low reset
irq_out	System	O	Interrupt on traffic generation completion
err_out	System	O	Error interrupt
AXI4 Master Interface Signals			
m_axi_*	M_AXI		AXI4-Lite Slave Interface signals. See Appendix A of the *AXI Reference Guide* (UG761) [Ref 5] for AXI4, AXI4-Lite and AXI Stream Signals.
AXI4 Slave Interface Signals			
S_axi_*	S_AXI		AXI4-Lite Slave Interface signals. See Appendix A of the *AXI Reference Guide* (UG761) [Ref 5] for AXI4, AXI4-Lite and AXI Stream Signals.
AXI4-Stream Interface Signals			
m_axis_1_tready	M_AXIS_1	I	See AXIS Bus definition
m_axis_1_tvalid	M_AXIS_1	O	See AXIS Bus definition
m_axis_1_tlast	M_AXIS_1	O	See AXIS Bus definition
m_axis_1_tdata	M_AXIS_1	O	See AXIS Bus definition
m_axis_1_tstrb	M_AXIS_1	O	See AXIS Bus definition
m_axis_1_tkeep	M_AXIS_1	O	See AXIS Bus definition
m_axis_1_tuser	M_AXIS_1	O	See AXIS Bus definition
s_axis_2_tready	S_AXIS_2	O	See AXIS Bus definition
s_axis_2_tvalid	S_AXIS_2	I	See AXIS Bus definition
s_axis_2_tlast	S_AXIS_2	I	See AXIS Bus definition
s_axis_2_tdata	S_AXIS_2	I	See AXIS Bus definition
s_axis_2_tstrb	S_AXIS_2	I	See AXIS Bus definition
s_axis_2_tkeep	S_AXIS_2	I	See AXIS Bus definition
s_axis_2_tuser	S_AXIS_2	I	See AXIS Bus definition
m_axis_2_tready	M_AXIS_2	I	See AXIS Bus definition
m_axis_2_tvalid	M_AXIS_2	O	See AXIS Bus definition

Table 2-5: **AXI Traffic Generator I/O Signals *(Cont'd)***

Signal Name	Interface	I/O	Description
m_axis_2_tlast	M_AXIS_2	O	See AXIS Bus definition
m_axis_2_tdata	M_AXIS_2	O	See AXIS Bus definition
m_axis_2_tstrb	M_AXIS_2	O	See AXIS Bus definition
m_axis_2_tkeep	M_AXIS_2	O	See AXIS Bus definition
m_axis_2_tuser	M_AXIS_2	O	See AXIS Bus definition
AXI4-Lite Master Write Interface			
m_axi_lite_awaddr	M_AXI_LITE	O	See AXI4-Lite Bus definition
m_axi_lite_awprot	M_AXI_LITE	O	See AXI4-Lite Bus definition
m_axi_lite_awvalid	M_AXI_LITE	O	See AXI4-Lite Bus definition
m_axi_lite_awready	M_AXI_LITE	I	See AXI4-Lite Bus definition
m_axi_lite_wdata	M_AXI_LITE	O	See AXI4-Lite Bus definition
m_axi_lite_wstrb	M_AXI_LITE	O	See AXI4-Lite Bus definition
m_axi_lite_wvalid	M_AXI_LITE	O	See AXI4-Lite Bus definition
m_axi_lite_wready	M_AXI_LITE	I	See AXI4-Lite Bus definition
m_axi_lite_bresp	M_AXI_LITE	I	See AXI4-Lite Bus definition
m_axi_lite_bvalid	M_AXI_LITE	I	See AXI4-Lite Bus definition
m_axi_lite_bready	M_AXI_LITE	O	See AXI4-Lite Bus definition

Notes:

1. AXIS refers to streaming interface.
2. AXI4-Lite refers to AXI4-Lite Master interface.

Register Space

The AXI Traffic Generator provides registers to control its behavior, provide status or debug information, and to control external signals. The register space is only partially decoded.

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.

Advanced/Basic Mode Register Map

Table 2-6 is available only in AXI4 Advanced and Basic mode. For any other mode, these registers are not accessible.

Table 2-6: **Advanced/Basic Mode Register Map**

Offset	Register Name	Description
0x00	Master Control	To control master logic.
0x04	Slave Control	To control slave logic.
0x08	Error Status	Different errors reported during core operation.
0x0C	Error Enable	Enable register to report intended error.
0x10	Master Error Interrupt Enable	To generate/mask external error interrupt.
0x14	Config Status	Stores the current configuration of the core.
0x18 to 0x2C	Reserved	Reserved
0xB4	Slave Error	Access to this register returns the SLVERR response.

Master Control

Master Control register allows you to configure the master interface ID width and control to enable the AXI traffic.

Table 2-7: **Master Control (0x00)**

Bits	Name	Reset Value	Access Type	Description
31:24	REV	0x47	R	Revision of the core.
23:21	MSTID	0x0	R	M_ID_WIDTH, where: 0x0 = Indicates 1-bit width 0x1 = Indicates 2-bit width ... 0x7 = Indicates 8-bit width
20	MSTEN	0x0	R/W	Master Enable 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 Traffic Generator is done.
19:0	Reserved	N/A	N/A	Reserved

Back to Top

Slave Control

Slave Control register allows you to configure the slave interface of AXI Traffic Generator to control /enable slave capabilities.

Table 2-8: **Slave Control (0x04)**

Bits	Name	Reset Value	Access Type	Description
31:20	Reserved	N/A	N/A	Reserved
19	BLKRD	0x0	R/W	Enable Block Read 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	DISEXCL	0x0	R/W	Disable Exclusive Access When set, disables exclusive access support and error response ability for reads on Slave Error register.
17	WORDR	0x0	R/W	Enable in Order Write Response When set, forces all BRESPs to be issued in the order the requests were received.
16	RORDR	0x0	R/W	Enable in Order Read Response When set, forces all slave reads to be done in the order received.
15	ERREN	0x0	R/W	Enable Error Generation When set, if any bit in Error Status register Bits[15:0] is set, then err_out is asserted.

Table 2-8: **Slave Control (0x04) *(Cont'd)***

Bits	Name	Reset Value	Access Type	Description
14:0	Reserved	N/A	N/A	Reserved

Back to Top

Error Status

Error Status register reports the errors occurred during the operation of AXI Traffic Generator core.

Table 2-9: **Error Status (0x08)**

Bits	Name	Reset Value	Access Type	Description
31	MSTDONE	0x0	R/W1C	Master Completion Set when both master write and master read CMD logic completes and Error Enable register Bit[31] is 1. When set, irq_out is driven to 1.
30:21	Reserved	N/A	N/A	Reserved
20	RIDER	0x0	R/W1C	Master Read ID Error On master interface Received an RVALID with a RID that did not match any pending reads.
19	WIDER	0x0	R/W1C	Master Write ID Error Received a BVALID with a BID that did not match any pending writes.
18	WRSPER	N/A	R/W1C	Master Write Response Error On a master write completion, the response returned was not allowed by expected_resp[2:0].
17	RERRSP	0x0	R/W1C	Master Read Response Error On a master read completion, the response returned was not allowed by expected_resp[2:0].
16	RLENER	0x0	R/W1C	Master Read Length Error On the master interface Rlast either when it was not expected or was not signaled when it was expected.
15:2	Reserved	N/A	N/A	Reserved
1	SWSTRB	0x0	R/W1C	Slave Write Strobe Error On the slave interface, a WSTRB assertion was detected on an illegal byte lane.
0	SWLENER	0x0	R/W1C	Slave Write Length Error On the slave interface W, Last was signaled either when it was not expected or was not signaled when it was expected.

1. W1C – Write 1 to Clear (to clear register bit, user has to write 1 to corresponding bits).

Back to Top

Error Enable

Error Enable register allows you to enable the particular error condition in the AXI Traffic Generator. If an error occurs but the corresponding bit in Error Enable register is not set, then the bit in Error Status register is not set and no error signaling occurs. To enable all errors, set Error Enable register to 0xFFFF_FFFF.

This enables/disables error reporting on Error Status register.

Table 2-10: **Error Enable (0x0C)**

Bits	Name	Reset Value	Access Type	Description
31	MSTIRQEN	0x0	R/W	Enables interrupt generation for Master transfer completion.
30:21	Reserved	N/A	N/A	Reserved
20	RIDEREN	0x0	R/W	Enables Read ID Error for Error Status register Bit[20].
19	WIDEREN	0x0	R/W	Enables Write ID error for Error Status register Bit[19].
18	WRSPER	N/A	R/W	Enables write response error for Error Status register Bit[18].
17	RERRSP	0x0	R/W	Enables read response error for Error Status register Bit[17].
16	RLENER	0x0	R/W	Enables read length error for Error Status register Bit[16].
15:2	Reserved	N/A	N/A	Reserved
1	SWSTRBEN	0x0	R/W	Enables slave write strobe error for Error Status register Bit[1].
0	SWLENEREN	0x0	R/W	Enables slave write length error for Error Status register Bit[0].

Back to Top

Master Error Interrupt Enable

Master Error Interrupt Enable register enables interrupt generation for AXI4 Master interface based on the Error Status register.

Table 2-11: **Master Error Interrupt Enable (0x10)**

Bits	Name	Reset Value	Access Type	Description
31:16	Reserved	N/A	N/A	Reserved
15	MINTREN	0x0	R/W	Enables Master Interrupt When set, if any bit in Error Status register Bits[30:16] is set, then err_out is asserted.
14:0	Reserved	N/A	N/A	Reserved

Back to Top

Config Status

Config Status register is a read only register and provides you information on the core configuration.

Table 2-12: **Config Status (0x14)**

Bits	Name	Reset Value	Access Type	Description
31	Reserved	N/A	N/A	Reserved
30:28	MWIDTH	0x0	R	Master Width 0x0 = 32-bit 0x1 = 64-bit 0x2 = 128-bit 0x3 = 256-bit 0x4 = 512-bit
27:25	SWIDTH	0x0	N/A	Slave Width 000 = 32-bit 001 = 64-bit
24	MADV	0x0	R	ATG Mode is Advanced
23	MBASIC	0x0	R	ATG Mode is Basic
22:0	Reserved	N/A	N/A	Reserved

Back to Top

Streaming Mode Register Map

Table 2-13 is available only in AXI4-Stream mode. For any other mode, these registers are not accessible.

Table 2-13: **Streaming Mode Register Map**

Offset	Register Name	Description
0x30	Version	Provides the current version of the AXI4-Stream interface.
0x34	Streaming Config	Allows you to configure the Streaming master interface for programmable delays or random delay in transfer length.
0x38	Transfer Length	Allows you to configure the length of packets and transaction count.
0x3C	Reserved	Reserved for future purpose.

Version

Version register provides the current version of the AXI4-Stream interface. This register is available to you only in Streaming mode of operation.

Table 2-14: **Version (0x30)**

Bits	Name	Reset Value	Access Type	Description
31:24	STRMVER	0xD1	R	Streaming Version Register
23:0	Reserved	N/A	N/A	Reserved

Back to Top

Streaming Config

Streaming config register allows you to configure the Streaming master interface for programmable delays or random delay in transfer length. This register is available to you only in Streaming mode of operation.

Table 2-15: **Streaming Config (0x34)**

Bits	Name	Reset Value	Access Type	Description
31:16	PDLY	0x0	R/W	Programmable delay between two streaming packets.
15:2	Reserved	N/A	N/A	Reserved
1	RANDLY	0x0	R/W	Random Delay When set, generates random delay b/n streaming transactions. For example, from TLAST to next TVALID.
0	RANLEN	0x0	R/W	Random Length When set, generates streaming transactions with random length.

Back to Top

Transfer Length

Transfer Length register allows you to configure the length of packets and transaction count. This register is available to you only in Streaming mode of operation.

Table 2-16: **Transfer Length (0x38)**

Bits	Name	Reset Value	Access Type	Description
31:16	TCNT	0x0	R/W	Transfer Count Core generates this many transaction on AXI4-Stream channel and stops. If set to 0, core infinitely generates transactions.
15:0	TLEN	0x0	R/W	Length of Transaction When Random Length in Streaming Config register is not set, Length programmed in this register is used. Actual number of beats are one more than the register setting. For example, setting to 0 gives 1 beat, setting to 1 gives 2 beats and so on.

Back to Top

Static Mode Register Map

Table 2-17 is available only in AXI4 Static mode. For any other mode, this register is not accessible.

Table 2-17: **Static Mode Register Map**

Offset	Register Name	Description
0x60	Static Control	Allows you to start and stop the AXI Traffic Generator in static mode.
0x64	Static Length	Allows you to configure the burst length generated by AXI Traffic Generator in static mode.

Static Control

Static control register allows you to start and stop the AXI Traffic Generator in static mode. This register is available to you only in Static mode of operation.

Table 2-18: **Static Mode Control (0x60)**

Bits	Name	Reset Value	Access Type	Description
31:2	Reserved	N/A	N/A	Reserved
1	DONE	0x0	R/W1C	Transfer Done 0 = indicated core is generating traffic when STEN is 1, else core is in idle mode 1 = indicates traffic generation completed This bit is set to 1 when the core is disabled by setting STTEN to 0 and the current transfer is completed. This bit resets to 0 either writing 1 to this bit or enabling the core with STEN.
0	STEN	0x0	R/W	Static Enable 0 = disable traffic generation 1 = enable traffic generation

Back to Top

Static Length

Static length register allows you to configure the burst length generated by AXI Traffic Generator in static mode. This register is available to you only in Static mode of operation.

Table 2-19: **Static Length (0x64)**

Bits	Name	Reset Value	Access Type	Description
31:8	Reserved	N/A	N/A	Reserved
7:0	BLEN	Burst Length	R/W	Burst Length Configures burst length for AXI4 master interface. Reset value is the value configured for "Burst Length" in the GUI.

Back to Top

Designing with the Core

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

Clocking

The AXI Traffic Generator has a single input clock for AXI4-Slave, AXI4-Master (Advanced, Basic, Static, Master Init), and AXI4-Stream Mode. You should connect the appropriate clock to this clock input.

Resets

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

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.

Vivado Integrated Design Environment (IDE)

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

To access the AXI Traffic Generator, 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 **AXI Traffic Generator** to bring up the AXI Traffic Generator Customize IP dialog box.

Figure 4-1 shows the AXI Traffic Generator Customize IP dialog box with information about customizing ports.

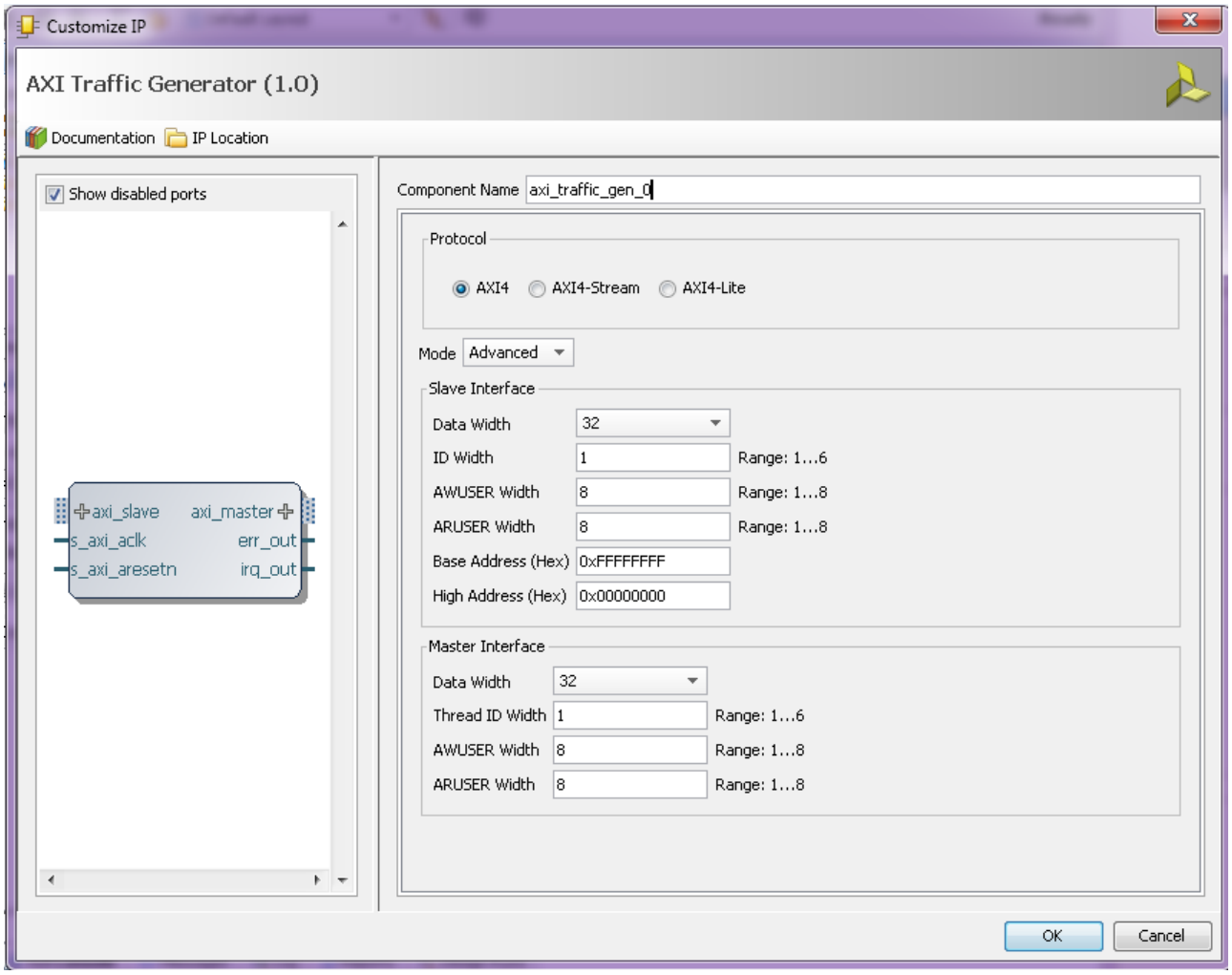

Figure 4-1: **Vivado Customize IP Dialog Box**

- **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 "_".

Protocol Options

- **Protocol** – Select the desired protocol traffic to be generated on the master interface. This core supports AXI4, AXI4-Stream, and AXI4-Lite traffic generation.

AXI4 Protocol

This protocol supports different mode configuration. The available mode of operations are Advanced, Basic, and Static.

Advanced/Basic Mode

For the Advanced/Basic Mode, Advanced mode generates customized traffic on master interface. Basic mode allows basic AXI4 traffic generation with less resource overhead. Available options are given in the following sections.

Slave Interface Options

- **Data Width** – Select the desired slave data width (32 and 64).
- **ID Width** – ID width of the slave.
- **AWUSER** – Write channel user signals width.
- **ARUSER** – Read channel user signals width.
- **Base Address** – Base address of the core (used by the Vivado tool when creating a system using IPI).
- **High Address** – High Address of the core. (used by the Vivado tool when creating a system using IPI).

Master Interface Options

- **Data Width** – Select the desired master data width (32, 64, 128, 256, and 512).
- **ID Width** – ID width of the master.
- **AWUSER** – Write channel user signals width.
- **ARUSER** – Read channel user signals width.

Static Mode

This mode allows you to generate simple AXI4 traffic with fewer resource overhead compared to the Advanced/Basic Mode. Available options are given in the following sections.

Master Interface Options

- **Data Width** – Select the desired master data width (32, 64, 128, 256, and 512).

Static Mode Options

- **Write Address** – Address for write transactions. This has to be configured based on the available memory slaves in the system.
- **Read Address** – Address for read transactions. This has to be configured based on the available memory slaves in the system.
- **Burst Length** – Burst length for read/write transactions.

- **Channel Select** – Select desired channel on which traffic to be generated.

AXI4-Stream Protocol

Streaming mode allows you to generate AXI4-Stream traffic on master interface. It also provides streaming loopback channel.

- **Data Width** – Select the desired streaming data width (32, 64, 128, 256, and 512).
- **User Width** – Width of streaming user signals.

AXI4-Lite Protocol

System Init mode allows you to generate the AXI4 write transaction on the master interface. Transactions are generated based on the configuration file provided by you once core generates all `irq_out`.

- **Transaction Depth** – Maximum number of address and data entries supported in COE file. Available transaction depth are 16, 32, 64, 128, and 256.
- **Data COE File** – Load/create the data COE file. Contains data entries for the corresponding address in the Address COE file.
- **Address COE File** – Load/create the address COE file. Contains address entries for the transactions to be generated on the master interface. End of the transaction is defined by NOP (0xFFFFFFFF). The core stops generating any further transaction after processing NOP.

Constraining the Core

There are no IP specific constraints other than the AXI clock constraint. This core generates the out-of-context (OOC) XDCs.

Appendix A

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 *Vivado Design Suite Migration Methodology Guide* (UG911) [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 to guide you through debugging the AXI Traffic Generator 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 Traffic Generator, 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 Traffic Generator. 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.

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.

Master Answer Record for the AXI Traffic Generator

AR 54426

Contacting 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.

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 Traffic Generator design issues. It is important to know which tools are useful for debugging various situations.

Vivado Lab Tools

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

The Vivado logic analyzer is used to interact with the logic debug LogiCORE IP cores, including:

- ILA 2.0 (and later versions)
- VIO 2.0 (and later versions)

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 Vivado Analyzer tool is a valuable resource to use in hardware debug. The signal names mentioned in the following individual sections can be probed using the Vivado Analyzer 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_axis_aresetn` 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 Vivado Analyzer tool capture that the waveform is correct for accessing the AXI4 interface.

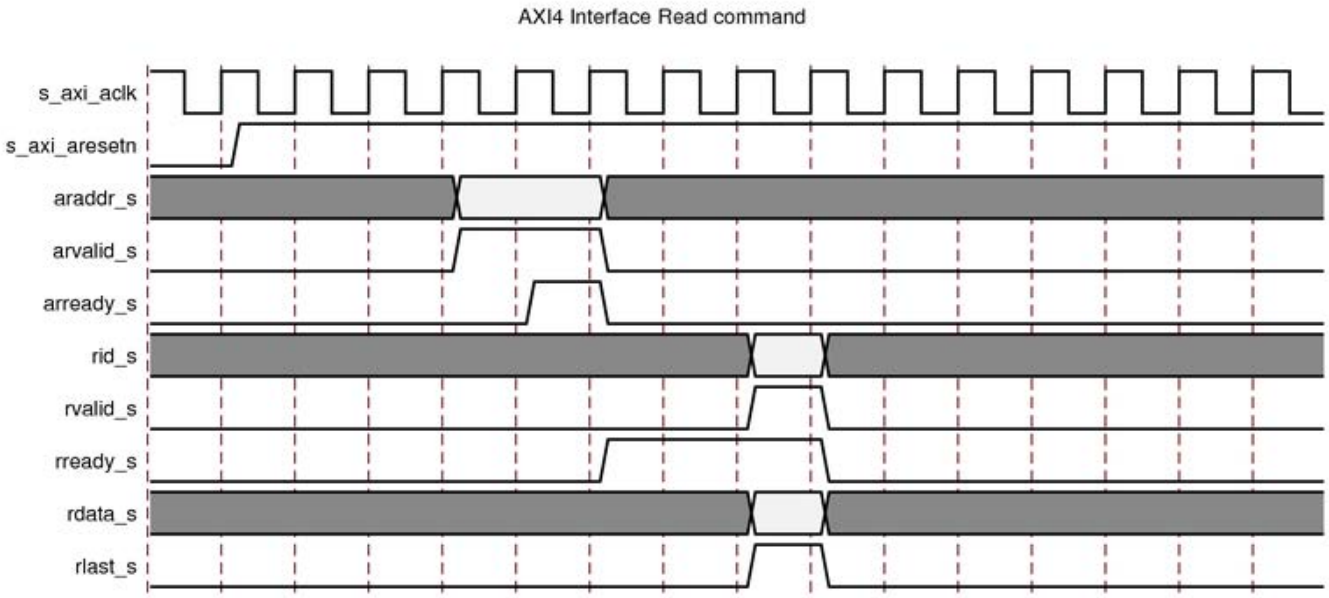

Figure B-1: **AXI4 Read Timing Diagram**

Appendix C

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

These documents provide supplemental material useful with this product guide:

1. Vivado™ Design Suite user documentation
2. *Vivado Design Suite User Guide, Designing with IP* (UG896)
3. *Vivado Design Suite User Guide, Getting Started* (UG910)
4. *Vivado Design Suite, Getting Started Guide* (UG814)
5. *AXI Reference Guide* (UG761)
6. *ARM® AXI4 Memory Mapped Specification*
7. *ARM AMBA AXI Protocol*, version 2.0 (ARM IHI 0022C) (http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ihi0022c/index.html)

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
03/20/2013	1.0	Initial Xilinx release. This Product Guide replaces PG094 AXI Exerciser.

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.

© Copyright 2013 Xilinx, Inc. Xilinx, the Xilinx logo, Artix, ISE, Kintex, Spartan, Virtex, Vivado, Zynq, and other designated brands included herein are trademarks of Xilinx in the United States and other countries. All other trademarks are the property of their respective owners.