AXI Traffic Generator v3.0

LogiCORE IP Product Guide

Vivado Design Suite

PG125 (v3.0) October 18, 2023

Table of Contents

Chapter 1: Introduction **4**
Features 4
IP Facts 5

Chapter 2: Overview **6**
Modes Description 8
Programming Sequence 29
Applications 31
Licensing and Ordering 33

Chapter 3: Product Specification **34**
Performance 34
Resource Use 36
Port Descriptions 36
Register Space 39
LFSR Implementation used for Random Generation 52

Chapter 4: Designing with the Core **53**
Clocking 53
Resets 53

Chapter 5: Design Flow Steps **54**
Customizing and Generating the Core 54
Constraining the Core 68
Simulation 69
Synthesis and Implementation 69

Chapter 6: Example Design **70**
Implementing the Example Design 71
Simulating the Example Design 71

Chapter 7: Test Bench **73**

Example Sequences for Custom Mode....73
Example Sequences for High Level Traffic (Video,PCIe, Ethernet, USB) Modes....75
Example Sequences for High Level Traffic (Data).... 75

Appendix A: Upgrading.... 76
Migrating to the Vivado Design Suite....76
Upgrading in the Vivado Design Suite....76

Appendix B: Debugging....77
Finding Help with AMD Adaptive Computing Solutions....77
Debug Tools.... 78
Hardware Debug.... 79
Interface Debug.... 80

Appendix C: Additional Resources and Legal Notices.... 81
Finding Additional Documentation....81
Support Resources....82
References....82
Revision History....82
Please Read: Important Legal Notices.... 85

Chapter 1

Introduction

The AXI Traffic Generator AMD LogiCORE™ IP generates traffic over 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 output AXI4 Slave, (32/64/128/256/512-bit) on output AXI4 Master interface.
- Flexible address width capability from 32 to 64-bit on AXI4 Master Interface.
- Flexible data width capability from 8-bit to 1,024-bit in multiples of eight output AXI4-Stream Master/Slave 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. Only supported in Advanced mode.
- Initialization support through Memory initialization files to internal RAM (CMDRAM, PARAMRAM, and MSTRAM) allows you to initialize the contents of all RAMs for a desired traffic profile.
- External global start/stop to synchronize multiple AXI Traffic Generators in the system and to enable AXI Traffic Generator without processor intervention.
- Supports high level traffic generation for different traffic profiles.

IP Facts

AMD LogiCORE™ IP Facts Table	
Core Specifics	
Supported Device Family[1]	AMD UltraScale+™Families, AMD UltraScale™ Architecture, AMD Zynq™ 7000 SoC, 7 series FPGAs
Supported User Interfaces	AXI4, AXI4-Lite, AXI4-Stream
Resources	Performance and Resource Utilization web page
Provided with Core	
Design Files	Verilog
Example Design	Verilog
Test Bench	Verilog
Constraints File	Xilinx Design Constraints (XDC)
Simulation Model	Not Provided
Supported S/W Driver[2]	Standalone and Linux
Tested Design Flows[3]	
Design Entry	AMD Vivado™ Design Suite
Simulation	For supported simulators, see the *Vivado Design Suite User Guide: Release Notes, Installation, and Licensing* (UG973).
Synthesis	Vivado Synthesis
Support	
Release Notes and Known Issues	Master Answer Record: 54426
All Vivado IP Change Logs	Master Vivado IP Change Logs: 72775
Support web page	

Notes:

1. For a complete list of supported devices, see the AMD Vivado™ IP catalog.
2. Standalone driver details can be found in <install_directory>/Vitis/<release>/data/embeddedsw/doc/xilinx_drivers_api_toc.htm.
3. For the supported versions of third-party tools, see the *Vivado Design Suite User Guide: Release Notes, Installation, and Licensing* (UG973).

Chapter 2

Overview

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

- Configurable option to generate and accept data according to different traffic profiles
- Configurable address width for Master AXI4 interface
- Supports dependent/independent transaction between read/write master port with configurable delays
- Programmable repeat count for each transaction with constant/increment/random address
- External start/stop to generate traffic without processor intervention
- Generates IP-specific traffic on AXI interface for pre-defined protocols

Note: This product guide replaces the *LogiCORE IP AXI Exerciser Product Guide* (PG094).

The core generates AXI4, AXI4-Lite, or AXI4-Stream traffic based on the mode selected. The AXI Traffic Generator core can be configured in six different modes, as detailed in the following table.

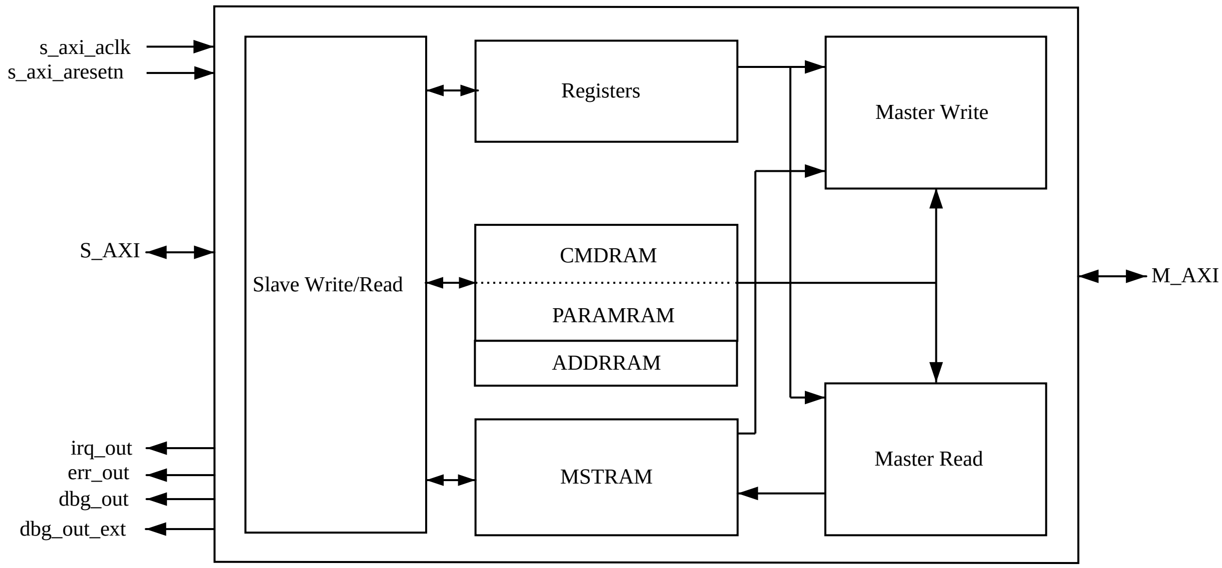
Table 1: **AXI Traffic Generator Modes**

Mode	Traffic Type	Description
Advanced	AXI4	Supports all AXI4 features.
Basic	AXI4	Lightweight mode with basic AXI4 features support (narrow, unaligned, out-of-order transfers are not supported).
Static	AXI4	Simple AXI4 traffic generator mode with fixed address, incremental transactions based on UI configuration. Minimum processor overhead.
High Level Traffic	AXI4	Generates IP specific traffic on AXI4 interface for pre-defined protocols.
System Init/Test	AXI4-Lite	AXI4-Lite interface for system initialization or simple system testing. Transactions initiated based on memory initialization files.
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, and each contains the Write and Read blocks. Other support functions are provided by the Control registers and internal RAMs.

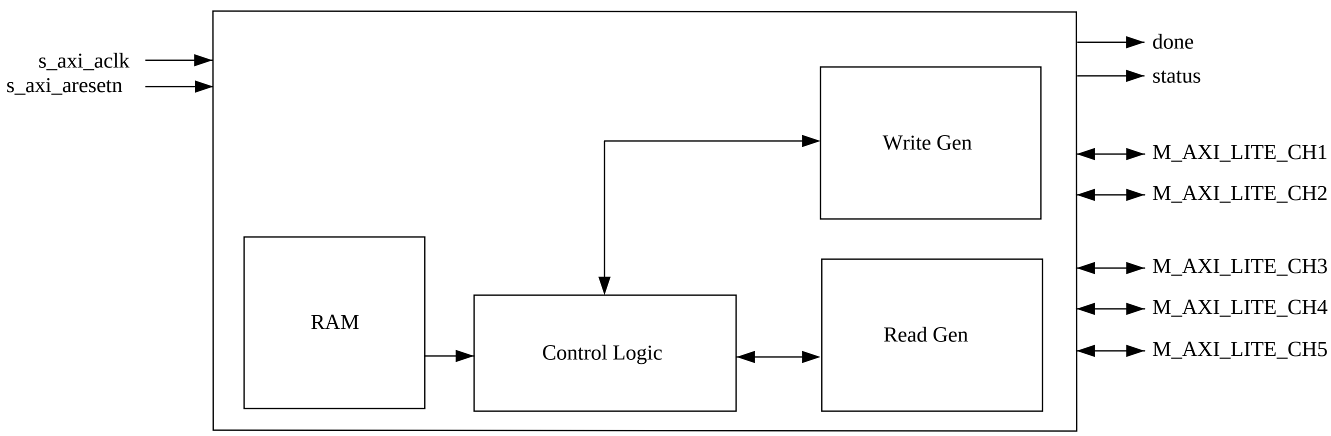
The following figure shows the top-level AXI4 Traffic Generator block diagram.

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



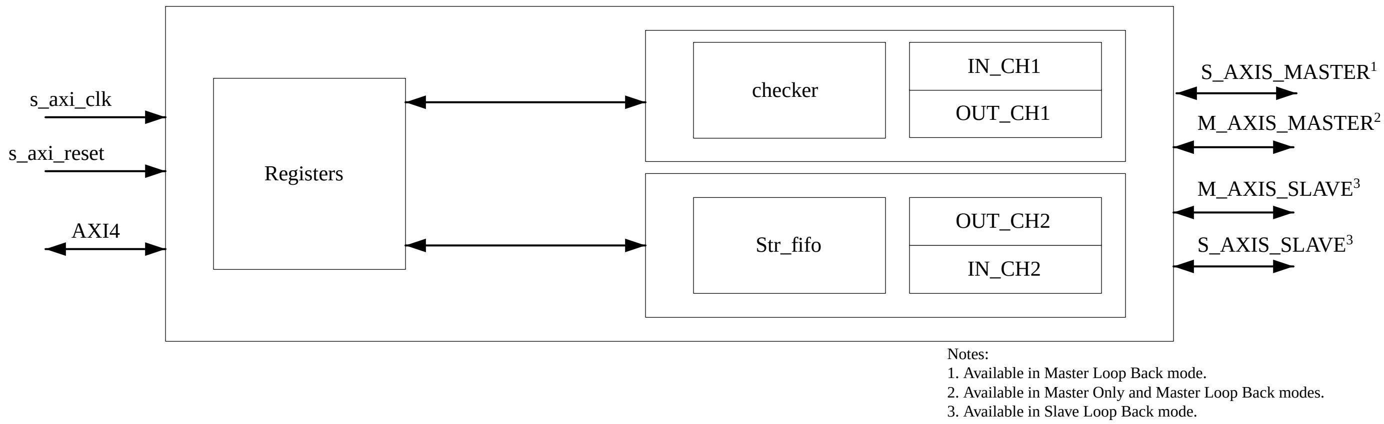
The following figure shows the top-level AXI4-Lite Traffic Generator block diagram.

Figure 2: **AXI4-Lite Traffic Generator Block Diagram**



The following figure shows the top-level AXI4-Stream Traffic Generator block diagram.

Figure 3: **AXI4-Stream Traffic Generator Block Diagram**



Modes Description

The AXI Traffic Generator has two major profile selection modes:

- Custom: This mode allows you to select different AXI4 interface traffic generation. The available options are AXI4, AXI4-Stream, and AXI4-Lite that include these modes:
  - Advanced Mode
  - Basic Mode
  - Static Mode
  - System Init/Test Mode
  - Streaming Mode
- High Level Traffic: This mode allows you to generate IP-specific traffic on the AXI interface for pre-defined protocols. The currently supported traffic profiles include:
  - Video Mode
  - PCIe® Mode
  - Ethernet Mode
  - USB Mode
  - Data Mode

Advanced Mode

Advanced mode allows full control over the traffic generation. Control registers are provided to program the core and generate different AXI4 transactions. For more information on each register, see Register Space.

Three internal RAMs are used as follows:

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

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. Read and write commands are executed simultaneously and independently. CMDRAM is realized using the dual-port block RAM. Access to CMDRAM is prohibited after the master logic of the core is enabled (Bit[20] MSTEN of Master Control).

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 the following table is 128-bits wide.

Table 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])

Table 2: **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. **64-bit mode**: 000 = All bytes valid 001 = Only Byte 0 is valid 010 = Only Bytes 0 and 1 are valid ... **32-bit mode**: 000 = All bytes valid 100 = Only Byte 0 is valid 101 = Only Bytes 0 and 1 are valid 110 = Only Bytes 0, 1, and 2 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 commands (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 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 OKAY is allowed 0x4 = Only DECERR or SLVERR is allowed 0x7 = Any response is allowed

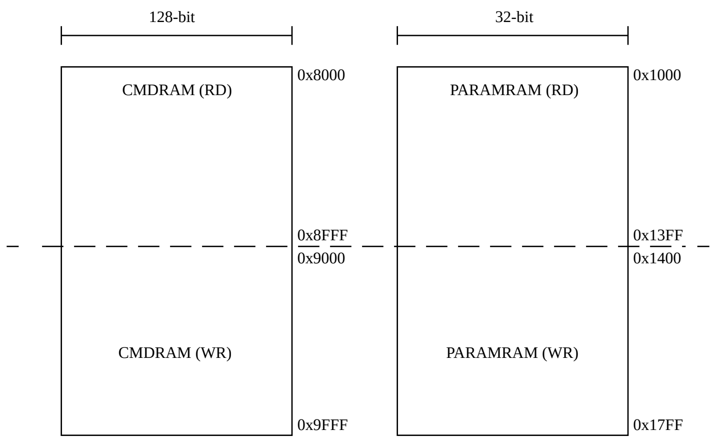
Notes:

1. `Valid_cmd`: There should be at least one command with valid_cmd bit set to zero (that is, one invalid command) 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. The MSTRAM index should be aligned the same as the master (Read/Write) address. For 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 the command.
3. It is recommended to write 0s to the command RAM reserved bits.
4. Address and burst length should be selected such that the transaction does not cross the 4K boundary.
5. Reads to CMDRAM from slave interface should be burst length 0.

PARAMRAM

The PARAMRAM extends the command programmability provided through command RAM by adding 32-bits to the decode of each command. The following figure 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 4: **PARAMRAM vs. CMDRAM**



Each entry in the PARAMRAM modifies its corresponding CMDRAM entry. The encoding and opcodes are described in the following tables.

Table 3: **PARAMRAM Entry Control Signals**

<table>
<tr><th>Bits</th><th>Name</th><th>Description</th></tr>
<tr><td>31:29</td><td>Opcode</td><td>The opcode defines how the op_control bits are used. Currently, four operations are supported:

Table 3: PARAMRAM Entry Control Signals
<table>
<tr><th>Bits</th><th>Name</th><th>Description</th></tr>
<tr><td>000</td><td>NOP</td><td>The command in the CMDRAM executes unaltered.</td></tr>
<tr><td>001</td><td>OP_REPEAT</td><td>The command in the CMDRAM repeats multiple times.</td></tr>
<tr><td>010</td><td>OP_DELAY</td><td>The command in the CMDRAM delays before execution.</td></tr>
<tr><td>010</td><td>OP_FIXEDREPEAT_DELAY</td><td>The command in the CMDRAM repeats multiple times. The repeat count depends on the Vivado IDE option (Repeat Count) provided by you. Delay and address range can be constrained.</td></tr>
</table></td></tr>
<tr><td>28</td><td>Idmode</td><td>Unused</td></tr>
<tr><td>27:26</td><td>Intervalmode</td><td>Control interval delay validated by OP_FIXEDREPEAT_DELAY.
00 = Constant Delay as programmed with Bits[19:8]</td></tr>
</table>

Table 3: **PARAMRAM Entry Control Signals** *(cont'd)*

<table>
<tr><th>Bits</th><th>Name</th><th>Description</th></tr>
<tr><td>25:24</td><td>Addrmode</td><td>Control addressing when a command is repeated.

Table 3: PARAMRAM Entry Control Signals
<table>
<tr><th>Bits</th><th>Name</th><th>Description</th></tr>
<tr><td>00</td><td>Constant</td><td>Address does not change</td></tr>
<tr><td>01</td><td>Increment</td><td>Address increments ((BUSWIDTH / 8) × (AXI_LEN + 1)) between repeated transactions</td></tr>
<tr><td>10</td><td>Random</td><td>Address is randomly generated within a address range. Valid only when OP_FIXEDREPEAT_DELAY is selected.</td></tr>
</table></td></tr>
<tr><td>23:0</td><td>PARAMRAM Opcodes</td><td>The definition for Bits[23:0] depend on the selected PARAMRAM opcodes. The details are described in the following tables.</td></tr>
</table>

Notes:

1. When using PARAMRAM in Address increment or random address generation mode, ensure that the address specified is aligned to burst boundaries. Failing to do so results in gaps being inserted in the address range. For example, in a 32-bit transaction with 16 being the burst length, the last three bits of the address have to be 0.
2. All transactions in Random mode are generated with data width aligned addresses.

PARAMRAM Opcodes

Each of the four commands uses 24 bits of `op_control` space to shape the command. Each of the four op_control fields is described in the following tables.

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

Table 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 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 6: **OP_DELAY**

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

Notes:

1. Delay observed between two transactions ≥ the programmed value.
2. If the programmed value is ≤ 6, the minimum delay observed is six clock cycles.

Table 7: **OP_FIXEDREPEAT_DELAY**

<table>
<tr><th>Bits</th><th>Name</th><th>Description</th></tr>
<tr><td>23:20</td><td>Addr Range Encoded</td><td>Core issues a new random address within the range encoded below starting with base address you programmed.

Table 7: OP_FIXEDREPEAT_DELAY
<table>
<tr><th>Encoded</th><th>Range (KB)</th><th>Encoded</th><th>Range (MB)</th></tr>
<tr><td>0</td><td>4</td><td>8</td><td>1</td></tr>
<tr><td>1</td><td>8</td><td>9</td><td>2</td></tr>
<tr><td>2</td><td>16</td><td>10</td><td>4</td></tr>
<tr><td>3</td><td>32</td><td>11</td><td>8</td></tr>
<tr><td>4</td><td>64</td><td>12</td><td>16</td></tr>
<tr><td>5</td><td>128</td><td>13</td><td>32</td></tr>
<tr><td>6</td><td>256</td><td>14</td><td>64</td></tr>
<tr><td>7</td><td>512</td><td>15</td><td>128</td></tr>
</table>
</td></tr>
<tr><td>19:8</td><td>Delay Count</td><td>Each command execution is delayed for this many cycles.</td></tr>
<tr><td>7:0</td><td>Delay Range Encoded</td><td>Unused</td></tr>
</table>

Notes:

1. Delay observed between two transactions ≥ the programmed value.

PARAMRAM should be filled with valid data for the corresponding entry in the CMDRAM. CMDRAM should be filled with valid data until the first invalid command entry.

An example programming sequence is to generate a write transaction with 64 beats transferred to the slave every 5 µs. When the clock frequency is running at 100 MHz, you could have the `awlen` (other AXI parameters to be set as per requirement) set to `0x3F` which corresponds to decimal value 63 to have 64 beats transferred in one transaction.

Also, you can have the PARAMRAM programmed to value `0x400001F4`, which decodes as IP is in OP_DELAY mode. The minimum delay between two transactions is `1F4` cycles, which is 500 cycles. If you want the transaction to be repeated, the PARAMRAM can be programmed as `0x8001F400` to set IP in FIXEDREPEAT_DELAY mode. The repeat count would be dependent on the AMD Vivado™ IDE value selected.

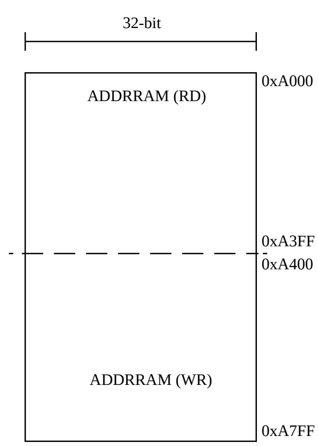
Address RAM

When the address width is configured in Vivado IDE is > 32, the extended address capability for the Master AXI4 interface in the AXI4 mode is available. In cases when address width is configured to 32, the Address RAM is not present and cannot be accessed.

The Address RAM entries correspond to the MSB bits of address and are concatenated to Bits[31:0] in CMDRAM. All 32-bits of RAM are accessible when the address width is > 32, but only the appropriate bits are considered and driven on the `m_axi_*addr` pins.

For example, if the address width is configured to be 36 in the Vivado IDE, enter a 32-bit value in the Address RAM such as 0x12345678. The 8 is concatenated to address bits in the CMDRAM and are driven on the address lines in the Master AXI4 interface.

Figure 5: **Address RAM**



Master RAM

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

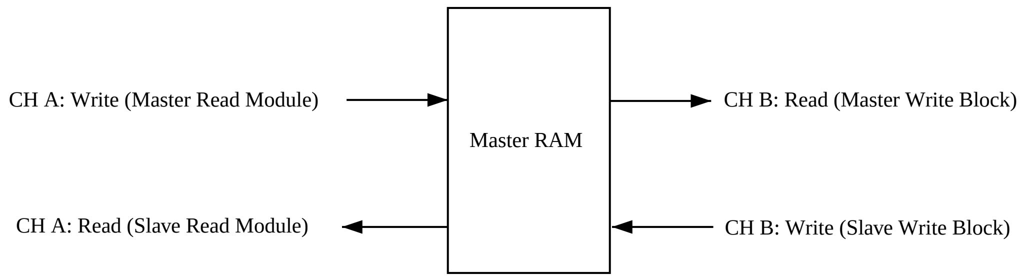
- Take data from this RAM for write transactions
- Store data to this RAM for read transaction

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

***Note*:** For AXI Fixed Burst, the data is written or read from the same MSTRAM location as mentioned in command word.

The Master RAM A and B channels are shown connected in the following figure.

Figure 6: **Master RAM Channels**



MSTRAM Index defines where to take data from MSTRAM (in case of write) and where to store the data (in case of read) in Advanced/Basic mode of operation. The following table shows details for the write data.

Table 8: **Write**

Address	Data	MSTRAM Entry Number (Index Entered in CMDRAM Programming)
0xC000	0x11111111	0
0xC004	0x22222222	
0xC008	0x33333333	1
0xC00C	0x44444444	
0xC010	0xABCD1234	2
0xC014	0xFAAB1234	
...		

In the case of Read, the Index definition changes based on the data width of the AXI4 Master interface. The following table shows details for data widths ≤ 64.

Table 9: **Read**

Address	Data	MSTRAM Entry Number (Index Entered in CMDRAM Programming)
0xC000	0x11111111	0
0xC004	0x22222222	

Table 9: **Read** *(cont'd)*

Address	Data	MSTRAM Entry Number (Index Entered in CMDRAM Programming)
0xC008	0x33333333	1
0xC00C	0x44444444	
0xC010	0xABCD1234	2
0xC014	0xFAAB1234	
...		

For data width > 64, rdata is stored at the 128/256/512 aligned locations (mstram_index should be set in the same manner).

AXI data width = 128-bit (mstram_index valid values = `0x0`, `0x10`, `0x20`, `0x30`, `0x40` ,...)

Example:

mstram_index = `0x10`

First Incoming data beat (lsb2msb) = `0xAABBCCDD_00112233_44556677_88888888`

Second Incoming data beat (lsb2msb) = `0xFFEEDDCC_55555555_44556677_88888888`

Table 10: **Master RAM**

Address	Data
0x0	000000000
0x	000000000
0x	000000000
0x	000000000
0x10	AABBCCDD
0x	00112233
0x	000000000
0x	000000000
0x20	FFEEDDCC
0x	55555555
0x	000000000
0x	000000000
0x30	:
0x	:
0x	000000000

AXI data width = 256-bit (mstram_index valid values = `0x0`, `0x20`, `0x40` ...)

Example:

mstram_index = `0x20`

First Incoming data beat (lsb2msb) =
`0xAABBCCDD_00112233_44556677_88888888_55665566_22113344_AAAAAAAA_FFFFFFFF`

Second Incoming data beat (lsb2msb) =
`0xFFEEDDCC_55555555_44556677_88888888_55665566_22113344_AAAAAAAA_FFFFFFFF`

Table 11: **Master RAM**

Address	Data
0x0	000000000
0x	000000000
0x	000000000
0x	000000000
0x	000000000
0x	000000000
0x	000000000
0x	000000000
0x20	AABBCCDD
0x	00112233
0x	000000000
0x	000000000
0x	000000000
0x	000000000
0x	000000000
0x	000000000
0x40	FFEEDDCC
0x	55555555
0x	000000000
0x	000000000
0x	000000000
0x	000000000
0x	000000000
0x	000000000
0x60	:
0x	:

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 = `0x0000_C004`

- **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 = `0x0000_C005` (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 = `0x0000_C008`

- **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 = `0x0000_C005` (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 32 and 64, the data from MSTRAM is sent to corresponding modules without any truncation/expansion in data width.

To save multiple RAM instances in data widths > 64, the same 64-bit data is duplicated/ truncated based on the current data width selection of master channels.

The following example uses a data width of 128:

- During read access from the master write block,

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

  That is, 64-bit data is duplicated on the 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.

For data width of 256:

- During read access from the master write block,

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

  That is, 64-bit data is duplicated on the write-data bus to make it 256-bits wide.
- During write access by master read block,

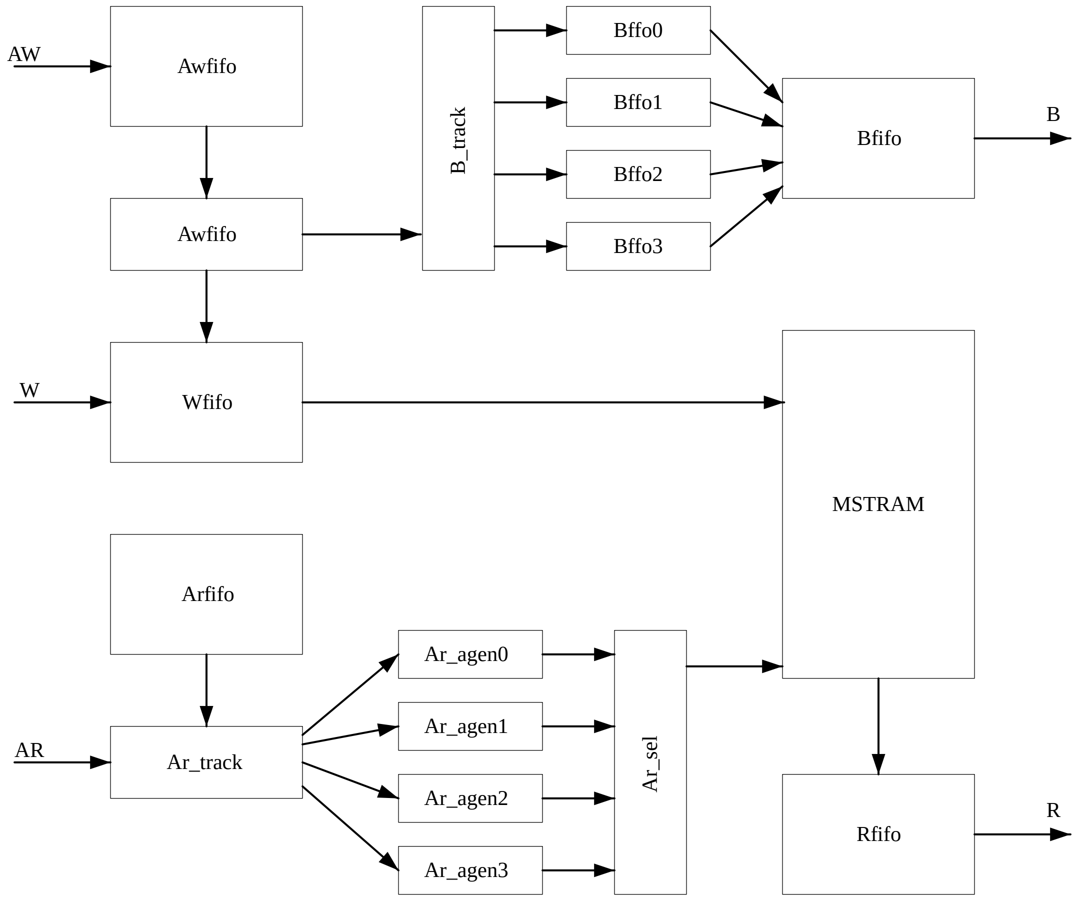
  write_data_to_mastram[63:0] = rdata_m[63:0]

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

Slave Modules

The following figure shows the slave logic.

Figure 7: **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.

The following table shows the address map for different regions accessed by slave-write/slave-read.

Table 12: **Slave-Write/Slave-Read Address Map**

Region	Description
0x0000_0000–0x0000_0FFF	Internal registers
0x0000_1000–0x0000_17FF	PARAMRAM (2 KB)
0x0000_8000–0x0000_9FFF	CMDRAM (8 KB)
0x0000_A000–0x0000_AFFF	ADDRRAM (2 KB)
0x0000_C000–0x0000_DFFF	MSTRAM (8 KB)

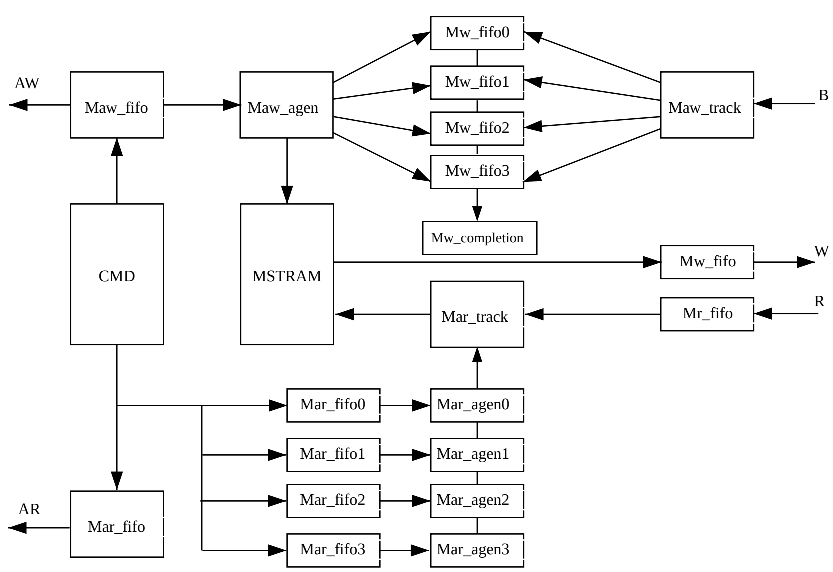
Unused memory locations in the memory space (considering `0x00000000` to `0x0000FFFF`) with respect to the preceding table are reserved; accessing these does not give any error response. Address aliasing applied for memory space is defined in the preceding table except PARAMRAM.

For slave logic, the write interleaving depth is one.

Master Modules

The following figure shows the master logic.

Figure 8: **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, etc. 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

The following table shows the ports that are tied/assumed to default value.

Table 13: **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	Quality of Service (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: *Write the default values in the preceding table into the command RAM when programming the command RAM entries.*

Static Mode

Static mode allows you to generate a simple AXI4 traffic with much less resource overhead and minimum processor intervention. After the core is enabled in Static mode using the Static Control register (Register Space), it continuously generates fixed data and fixed length INCR type read and writes transfers with optional address sweep capability. When the address sweep option is enabled, IP generates a transaction with an incrementing address between the Base and High address programmed.

You can configure the Read/Write address based on the system configuration and the transfer length from Vivado Integrated Design Environment (IDE) parameters. Transfer length can also be configured through the Static Length register. Read or Write channels can be enabled separately from the Vivado IDE 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 such that the transaction does not cross the 4K boundary. Failing to do so might result in gaps in the generated addresses,

If the address sweep option is enabled, the burst lengths allowed are only (2^n – 1). Care has to be taken when the length is programmed using the Control register.

When the address width is configured to be > 32, the Vivado IDE entries for Write Base Address (MSB), Write High Address (MSB), Read Base Address (MSB), and Read High Address (MSB) determine the address driven on the `m_axi_*addr` lines.

System Init/Test Mode

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

After the core comes out of reset in System Init mode, it reads the coefficient (COE) files (address and data) from the ROM and generates AXI4-Lite transactions. You must provide two COE files for this mode. Entries in all of the COE files are 32-bit.

- **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

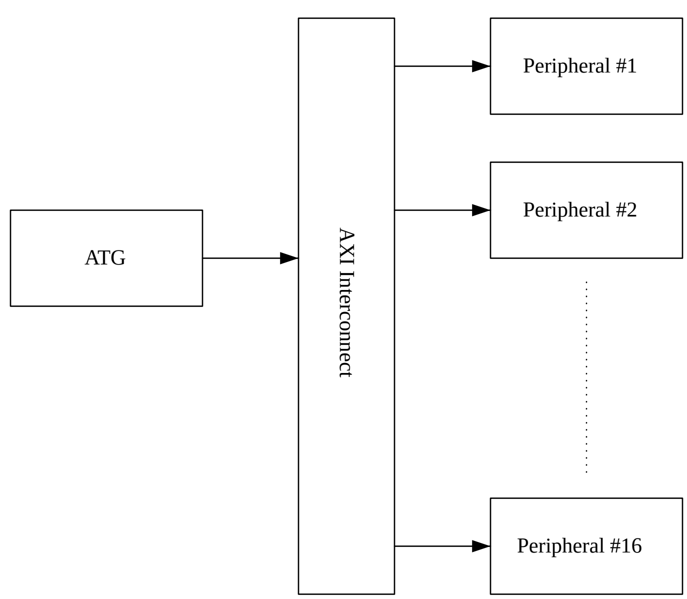
IMPORTANT! *You need to fill the entries in the COE files to match the requirement. The first address entry in the Address COE file corresponds to the first data entry in the Data COE file.*

Operation

1. After AXI Traffic Generator (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 an idle state after AXI4-Lite transactions are issued.

The following figure shows the example use case where ATG (System Init mode) is used to initialize peripherals in a system without a processor.

Figure 9: **System Init Mode Block Diagram**



The number of entries in the COE file is 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 the NOP address is present. You need to ensure at least one NOP address is present in the address COE file.

System Test mode is an enhancement to the System Init mode with support to generate read transactions. This mode also allows you to write test applications using Traffic Generator supported micro-commands with the help of the additional COE files Control and Mask. Completion and status of the core operation are reported through `done` and `status` ports.

Table 14: **Control COE File – 32-bit Control information**

Bits	Description
31:18	Reserved. Must be filled to zeros.
17	Count as Error Checks the status of the transaction. For Write: BRESP is monitored to be OKAY. For Read: RDATA compared against the entry in Data COE File. 0 = check the BRESP/RDATA and do not increment error counter 1 = check the BRESP/RDATA and increment error counter
16	0 = read transaction 1 = write transaction
15:8	Next COE entry to be fetched upon successful completion of the current transaction.
7:0	Next COE entry to be fetched if the current transaction failed.

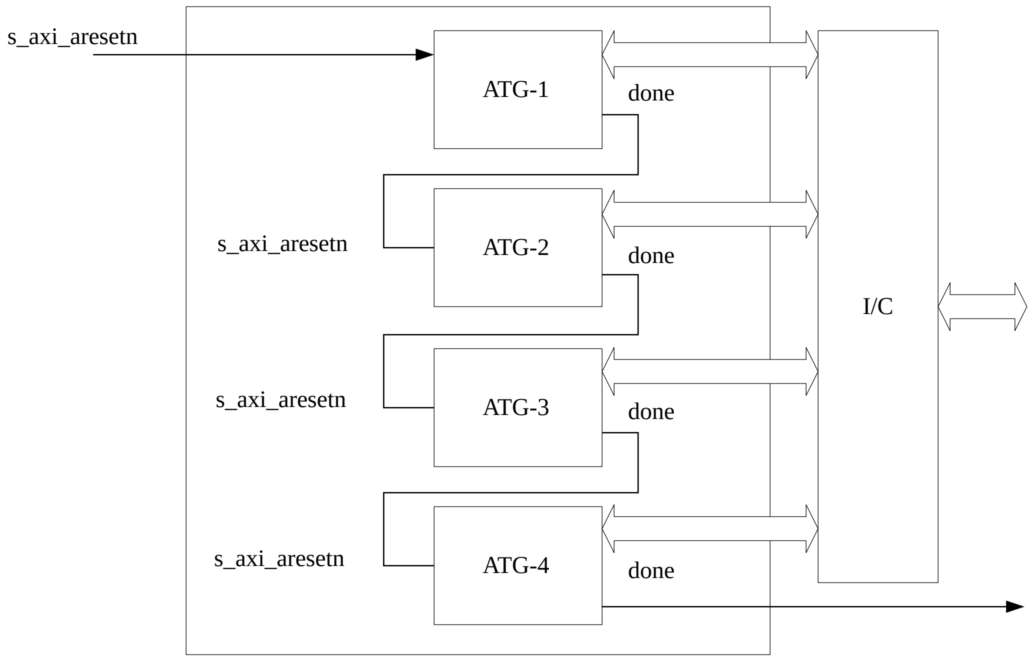
- Mask COE file represents the bits to mask before comparing the read data versus expected data. For write transactions, these bits are ignored by the IP.
- Mask bit value of 1 indicates the corresponding bit is used for comparing incoming read data with expected data.
- Mask bit value of 0 indicates the corresponding bit is not used for comparing incoming read data with expected data.

Cascading ATGs to Achieve Higher Number of Transactions Support

The allowed maximum number of entries in COE files is restricted to 256. Higher COE file depths (greater number of transactions) can be achieved by cascading multiple ATGs as shown in the following figure.

For example, to achieve a COE depth of 1,024, four ATGs each with 256 can be connected in a daisy chain fashion. The `done` output of one ATG is used to drive the reset of the next ATG. This ensures that only one ATG is active at a given time. Interconnect can be used to route all of the channels to a single channel.

Figure 10: **Cascading ATGs Diagram**



Streaming Mode

The Streaming mode provides one of the following selectable features:

- Master Only
- Master Loopback
- Slave Loopback

Master Only Mode

Master only generates streaming traffic on the `M_AXIS_MASTER` interface based on the register programming. This mode allows you to generate raw streaming data with programmable length, size, and delays on the `M_AXIS_MASTER` interface.

Master Loopback Mode

Master loopback mode has a built-in streaming data generator and checker logic. In this mode, the core generates streaming traffic on `M_AXIS_MASTER`. The same traffic (after processing by target IP) when fed to `S_AXIS_MASTER` interface compares the transaction generated at `M_AXIS_MASTER` versus a transaction received at `S_AXIS_MASTER`. Fields TDATA, TSTRB, and TKEEP are considered for comparison, and error count is reported through `axis_err_count` port.

Slave Loopback Mode

Slave loopback mode provides a simple stream loopback function. In this mode, the core receives streaming traffic over the `S_AXIS_SLAVE` interface and loops back the same traffic over `M_AXIS_SLAVE`. The core uses an internal FIFO of depth 14 to allow throttling on `M_AXIS_SLAVE` while still accepting data `S_AXIS_SLAVE` interface.

High Level Traffic

This mode allows you to generate IP-specific traffic on the AXI interface for pre-defined protocols like PCIe, Ethernet, and others. Based on the selected profile, AXI Traffic Generator generates AXI traffic to meet the desired throughput as if a real IP delivering the AXI transactions. This helps to characterize the system without the real IP in hand.

Different pre-defined traffic profiles available include:

- **Video:** This selection can be used to mimic video IP which processes video information and generates AXI Traffic. Different available options include HSize, VSize, Pixel bits, etc.
- **PCIe:** This selection mimics PCIe IP which processes PCIe packets and generates AXI Traffic. Different available options include PCIe Lanes, lane rate, etc. PCIe load option can be used to generate a partial load on the bus combined with the PCIe related options.
- **Ethernet:** This selection mimics Ethernet IP which processes Ethernet packets and generates AXI Traffic. Different available options include Ethernet speed and Ethernet load. The load option can be used to generate a partial load on the bus combined with the Ethernet-related options.
- **USB:** This selection mimics USB IP which processes USB packets and generates AXI Traffic. Options available include USB Mode which can be ISOC or BULK. Load option can be used to generate a partial load on the bus combined with the USB-related options.
- **Data:** This is a generic mode that can be used to generate user-intended traffic when one of the above options does not meet the requirements. Options available include read channel/ write channel share, minimum, maximum, average constraint on the transaction length generated, etc.

When the address width is configured to be > 32, the Vivado IDE entries for AXI Base Address (MSB) and AXI High Address (MSB) determines the address driven on the `m_axi_*addr` lines.

Note: All of the High Level Traffic Protocol options (Ethernet, Video, PCIe, USB) do not actually generate the corresponding protocol packets. They mimic the AXI side throughput only. For example, USB outputs a throughput worth 48 MB/s on AXI. So the ATG controls the length and gap between the transactions to achieve this throughput. At the system level on the AXI side, this appears as if a real USB is pumping the AXI traffic which helps to model/tune the system further.

IMPORTANT! *A low burst length value might prevent achieving the required throughput. A PCIe mode with a data width of 512, burst length of two with four PCIe lanes, eight (GT/s) line rate, and 100% channel load, the maximum channel capacity is 6,400 MBps. The expected throughput would be 3,940 MB/s but this throughput is not achieved by the core as a single transaction. It would at least require four cycles to finish (data transfer + response) to achieve the required data rate with this burst length. The transaction has to finish in two cycles which is incorrect. In this case, the burst length has to be at least four to get the required throughput.*

Programming Sequence

Advanced/Basic Mode with Processor

1. Load CMDRAM RAM with the required number of commands.
2. Load PARAM RAM with the desired opcodes. PARAM RAM is 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 the register control signal.
6. Wait for interrupt (if enabled) or poll Enable register control signal to check for completion of issuing the commands.

Advanced/Basic Mode without Processor Intervention

1. Edit the default generated `mif` files (`default_<componentname>_cmdram.mif`, `default_<componentname>_prmram.mif`, and `default_<componentname>_mstram.mif`) to match desired profile.
2. Generate an input start pulse to the core_ext_start port.
3. Wait for `irq_out` to check for completion of issuing the commands.

Static Mode with Processor

1. Select the desired Address/length from the Vivado IDE while generating the core.
2. Enable the core through the register control signal.
3. To change the burst length, disable the core, program a new burst length in the Static Length register, and re-enable the core.

Static Mode without Processor Intervention

1. Select the desired address/length from the Vivado IDE while generating the core.
2. Generate an input start pulse at the `core_ext_start` port.
3. Generate an input stop pulse at the `core_ext_stop` port when you want to stop generating traffic.

Streaming Mode with Processor

1. Program Streaming Config and Transfer Length registers as desired.
2. Enable the core through register control.
3. Wait for command completion by polling the done status in the Control register.

   Note: Slave loopback mode does not need any input. The core receives transactions on the slave interface and generates them on the master interface.

Note: When you need to set up the Streaming Config register (0x34), you must also set up the Streaming Error* registers (0x7*) if needed. Then, configure the Streaming Config register to ensure the correct functionality of IP in streaming mode when using a custom Transfer Length.

Streaming Mode without Processor Intervention

1. Generate an input start pulse at the `core_ext_start` port.
2. Generate an input stop pulse at the `core_ext_stop` port when you want to stop generating traffic.

   Note: Slave loopback mode does not need any input. The core receives transactions on the slave interface and generates them on the master interface.

High Level Traffic

1. Generate an input start pulse at the `core_ext_start` port.
2. Generate an input stop pulse at the `core_ext_stop` port when you want to stop generating traffic.

Note: For Data mode with Traffic Gen selected to One-shot, it is not necessary to provide the `core_ext_stop`, core automatically stops after one-shot of traffic is generated.

System Init/Test Mode

1. Provide the required COE files during core customization from the Vivado IDE.
2. This initiates the AXI4-Lite transactions on the Master interface when the core comes out of reset.
3. Status of the core operation is available on `status` port (see Port Descriptions for details) when `done` is asserted.

Example

Assume a test core operates in the following method when the core is enabled through a register bit:

- Checks for the Configuration register values
- Generates output based on the configuration
- Updates the Status register after completed

This is tested in hardware using the System Test mode of the AXI Traffic Generator by developing the COE files.

After following the example, this can be tested with this test sequence.

1. Program Configuration register (`0x4`) to a value of `0x12A0_1100`.
2. Enable completion status bit in Status Enable register (`0x8`) with a value `0x0000_000F`.
3. Enable the core by writing to the Control register (`0x0`) with a value of `0x0000_0001`.
4. Read the Status register (`0xC`) until the status value is `0x0000_000F`.

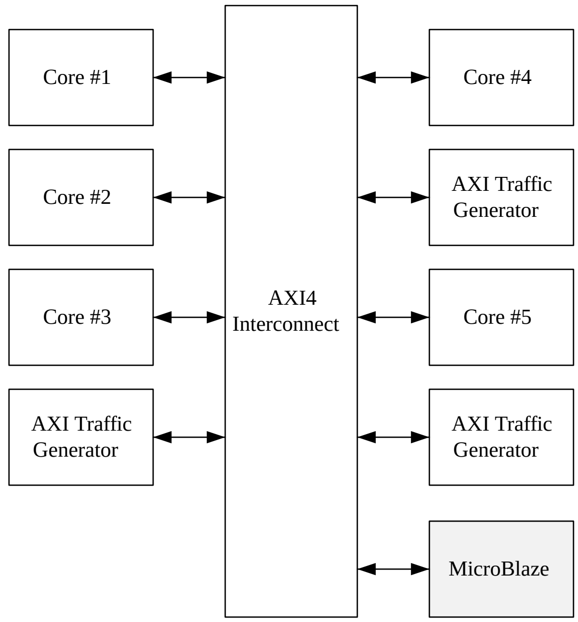
Based on the test sequence, four COE files need to be developed. Using these COE files the AXI Traffic Generator generates the test sequence and asserts `done` with the appropriate `status`.

Applications

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

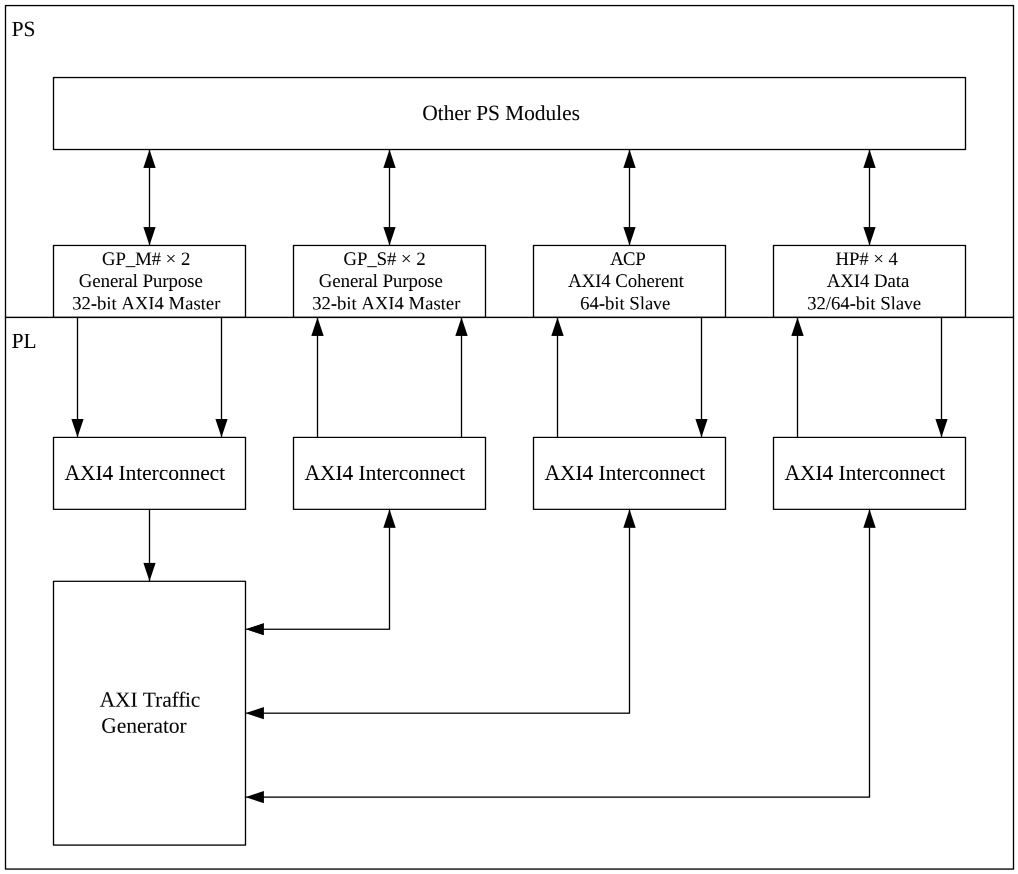
The following figure shows the AXI Traffic Generator connected to a MicroBlaze™ processor. The MicroBlaze programs the AXI Traffic Generator and the AXI Traffic Generator creates traffic to other cores.

Figure 11: **MicroBlaze System**



The following figure shows the AXI Traffic Generator connected to an AMD Zynq™ 7000 platform, The AXI Traffic Generator can be programmed from Arm® to the AXI peripherals.

Figure 12: **Zynq-7000 SoC System**



Licensing and Ordering

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

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

Chapter 3

Product Specification

Performance

For full details about performance and resource use, visit the Performance and Resource Utilization web page.

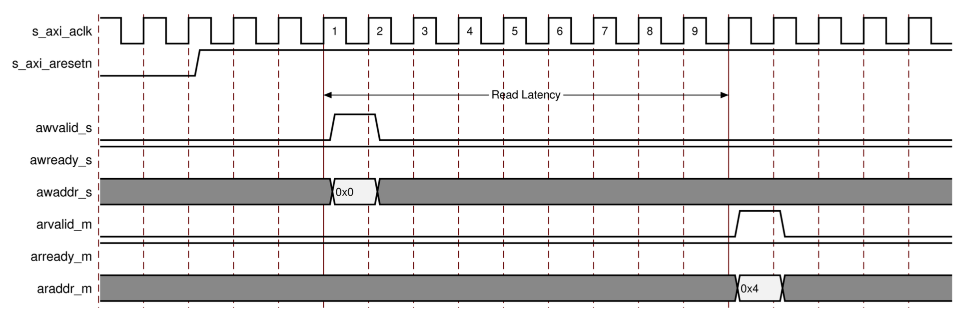
Latency

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 the Master Control register and the `awvalid` assertion on Master Ports. Latency is nine clock cycles with delay parameters in PARAMRAM set to zero.

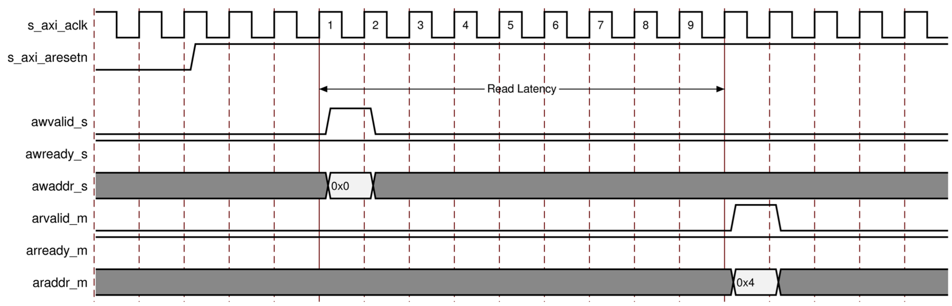
Figure 14: **Write Command Issuing Latency**



Read Command Issuing Latency

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

Figure 15: **Read Command Issuing Latency**



Throughput

The AXI Traffic Generator has a master write and read channel throughput.

Master Write Channel

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

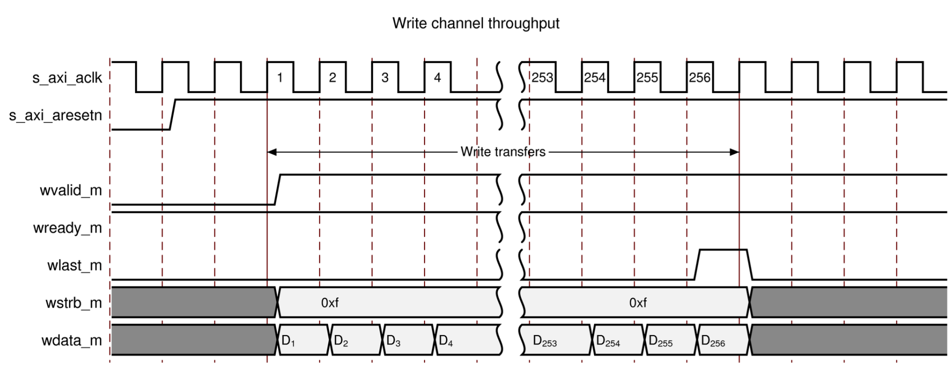
$$\text{Throughput} = (\text{A} - \text{B}) \times 100 \;/\; (\text{Total beats in the transaction})$$

A = Number of clock cycles `wvalid` and `wready` are asserted = 256

B = Number of clock cycles `wvalid` is deasserted, `wready` is asserted = 0

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

Figure 16: **Master Write Channel Throughput**



Master Read Channel

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

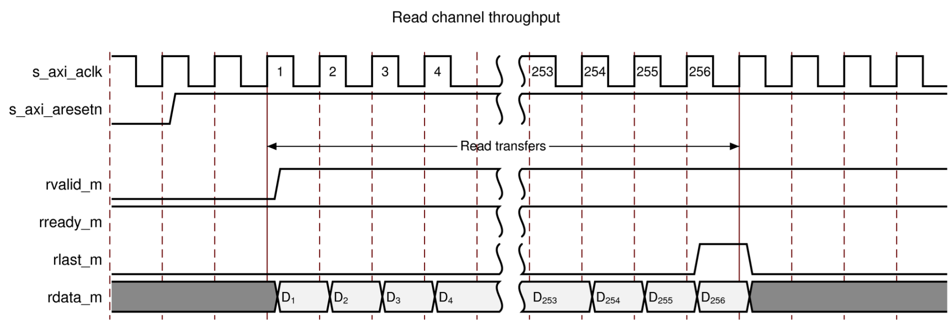
$$\text{Throughput} = (A - B) \times 100 \,/\, (\text{Total beats in the transaction})$$

A = Number of clock cycles `rready` and `rvalid` are asserted = 256

B = Number of clock cycles `rready` is deasserted, `rvalid` is asserted = 0

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

Figure 17: **Master Read Channel Throughput**



Resource Use

For full details about performance and resource use, visit the Performance and Resource Utilization web page.

Port Descriptions

The AXI Traffic Generator signals are listed and described in the following table.

Table 15: **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
done[2]	System	O	Indicates the completion of the sequence for AXI4-Lite mode selection.

Table 15: **AXI Traffic Generator I/O Signals** *(cont'd)*

<table>
<tr><th>Signal Name</th><th>Interface</th><th>I/O</th><th>Description</th></tr>
<tr><td>status</td><td>System</td><td>O</td><td>Status of the core operation in System Init/System Test mode. 32-bit Status Port Definition

Table 15: AXI Traffic Generator I/O Signals
<table>
<tr><th></th><th></th></tr>
<tr><td>31:16</td><td>Test Errors. Accumulates the number of errors seen during the genration of commands.</td></tr>
<tr><td>15:10</td><td>Reserved</td></tr>
<tr><td>9:2</td><td>Represents the COE index of the core it is currently processing. In case where the core repeatedly tries to issue the same command and exits after the maximum command retry count, this index is useful to debug.</td></tr>
<tr><td>1:0</td><td>Status of the Generation
00 = Reserved
01 = Pass
10 = Fail
11 = Hang</td></tr>
</table>
</td></tr>
<tr><td>core_ext_start[4]</td><td>System</td><td>I</td><td>Active-High pulse. Indicating the system to start generating or accepting the traffic.</td></tr>
<tr><td>core_ext_stop[4]</td><td>System</td><td>I</td><td>Active-High pulse. Indicating the system to stop generating or accepting the traffic.</td></tr>
<tr><td colspan="4">AXI4 Master Interface Signals</td></tr>
<tr><td>m_axi_*</td><td>M_AXI</td><td></td><td>AXI4 Master Interface signals. See Appendix A of the Vivado Design Suite: AXI Reference Guide (UG1037) for AXI4, AXI4-Lite, and AXI4-Stream Signals.</td></tr>
</table>

Table 15: **AXI Traffic Generator I/O Signals** *(cont'd)*

Signal Name	Interface	I/O	Description
AXI4 Slave Interface Signals			
s_axi_*	S_AXI		AXI4 Slave Interface signals. See Appendix A of the *Vivado Design Suite: AXI Reference Guide* (UG1037) for AXI4, AXI4-Lite, and AXI4-Stream Signals.
AXI4-Stream Interface Signals			
m_axis_1_tready	M_AXIS_MASTER	I	See AXIS Bus definition
m_axis_1_tvalid	M_AXIS_MASTER	O	See AXIS Bus definition
m_axis_1_tlast	M_AXIS_MASTER	O	See AXIS Bus definition
m_axis_1_tdata	M_AXIS_MASTER	O	See AXIS Bus definition
m_axis_1_tstrb	M_AXIS_MASTER	O	See AXIS Bus definition
m_axis_1_tkeep	M_AXIS_MASTER	O	See AXIS Bus definition
m_axis_1_tuser	M_AXIS_MASTER	O	See AXIS Bus definition
s_axis_1_tready	S_AXIS_MASTER	O	See AXIS Bus definition
s_axis_1_tvalid	S_AXIS_MASTER	I	See AXIS Bus definition
s_axis_1_tlast	S_AXIS_MASTER	I	See AXIS Bus definition
s_axis_1_tdata	S_AXIS_MASTER	I	See AXIS Bus definition
s_axis_1_tstrb	S_AXIS_MASTER	I	See AXIS Bus definition
s_axis_1_tkeep	S_AXIS_MASTER	I	See AXIS Bus definition
s_axis_1_tuser	S_AXIS_MASTER	I	See AXIS Bus definition
s_axis_2_tready	S_AXIS_SLAVE	O	See AXIS Bus definition
s_axis_2_tvalid	S_AXIS_SLAVE	I	See AXIS Bus definition
s_axis_2_tlast	S_AXIS_SLAVE	I	See AXIS Bus definition
s_axis_2_tdata	S_AXIS_SLAVE	I	See AXIS Bus definition
s_axis_2_tstrb	S_AXIS_SLAVE	I	See AXIS Bus definition
s_axis_2_tkeep	S_AXIS_SLAVE	I	See AXIS Bus definition
s_axis_2_tuser	S_AXIS_SLAVE	I	See AXIS Bus definition
m_axis_2_tready	M_AXIS_SLAVE	I	See AXIS Bus definition
m_axis_2_tvalid	M_AXIS_SLAVE	O	See AXIS Bus definition
m_axis_2_tlast	M_AXIS_SLAVE	O	See AXIS Bus definition
m_axis_2_tdata	M_AXIS_SLAVE	O	See AXIS Bus definition
m_axis_2_tstrb	M_AXIS_SLAVE	O	See AXIS Bus definition
m_axis_2_tkeep	M_AXIS_SLAVE	O	See AXIS Bus definition
m_axis_2_tuser	M_AXIS_SLAVE	O	See AXIS Bus definition
AXI4-Lite Master Write Interface			
m_axi_lite_ch*_awaddr	M_AXI_LITE	O	See AXI4-Lite Bus definition
m_axi_lite_ch*_awprot	M_AXI_LITE	O	See AXI4-Lite Bus definition
m_axi_lite_ch*_awvalid	M_AXI_LITE	O	See AXI4-Lite Bus definition
m_axi_lite_ch*_awready	M_AXI_LITE	I	See AXI4-Lite Bus definition

Table 15: **AXI Traffic Generator I/O Signals** *(cont'd)*

Signal Name	Interface	I/O	Description
m_axi_lite_ch*_wdata	M_AXI_LITE	O	See AXI4-Lite Bus definition
m_axi_lite_ch*_wstrb	M_AXI_LITE	O	See AXI4-Lite Bus definition
m_axi_lite_ch*_wvalid	M_AXI_LITE	O	See AXI4-Lite Bus definition
m_axi_lite_ch*_wready	M_AXI_LITE	I	See AXI4-Lite Bus definition
m_axi_lite_ch*_bresp	M_AXI_LITE	I	See AXI4-Lite Bus definition
m_axi_lite_ch*_bvalid	M_AXI_LITE	I	See AXI4-Lite Bus definition
m_axi_lite_ch*_bready	M_AXI_LITE	O	See AXI4-Lite Bus definition

Notes:

1. AXIS refers to streaming interface.
2. The done port now qualifies the sequence completion in AXI4-Lite mode. `irq_out` is used in the earlier version for this purpose.
3. In System INIT mode, M_AXI_LITE_CH* Read channel is not available as this mode is only intended to initialize the registers in the connected slave.
4. The core_ext_start and core_ext_stop ports can be used to control the traffic generation or reception by the core, without any processor intervention.

Register Space

The AXI Traffic Generator provides registers to control its behavior, provide status, or debug information, and 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 default values (except for the read-only bits). Access to read-only registers issues an OKAY response.

Advanced/Basic Mode Register Map

The following table is available only in AXI4 Advanced and Basic mode. For any other mode, these registers are not accessible.

Table 16: **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.

Table 16: **Advanced/Basic Mode Register Map** *(cont'd)*

Offset	Register Name	Description
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 17: **Master Control (0x00)**

Bits	Name	Reset Value	Access Type	Description
31:24	REV	0x20	R	Revision of the core.
23:21	MSTID	0x0	R	M_ID_WIDTH, where: 0x0 = Indicates 0 or 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 the software that the AXI Traffic Generator is done.

Table 17: **Master Control (0x00)** *(cont'd)*

Bits	Name	Reset Value	Access Type	Description
19	Loop Enable[1]	0x0	R/W	Loop Enable • Loops through the command set created using CMDRAM and PARAMRAM (as applicable) indefinitely when set to 1. • When this bit is reset to 0, the core stops looping after the current command set of transactions is completed. • Dependency (if any, both mydepend and otherdepend) is ignored when loop enable is set. Dependency gets honored after the loop enable is reset to 0. • Both channels loop back to their first command independently without waiting for the outstanding transactions to get completed. • If the interrupt is enabled, core generates `irq_out` after completing the command set following the reset of loop enable to 0. ***Note***: Dependency for the last command set run is based on the point at which the loop enable is reset to 0. For example, a command set with 12 writes and 16 reads is present with the 13th read is dependent on the sixth write. Now if the loop enable is reset to 0 before the sixth write and 13th read of command run, you see the dependency in the last run else the dependency is not seen even after loop enable is reset. For bullet point 4, consider a case of a command set with 50 write commands and two read commands. In such a case, the read command should be repeated more than once before one set of write commands is completed.
18:0	Reserved	N/A	N/A	Reserved

Notes:

1. One Invalid command has to be written in the CMDRAM at the end with/without setting Loop Enable.

Slave Control

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

Table 18: **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.
14:0	Reserved	N/A	N/A	Reserved

Error Status

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

Table 19: **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.

Table 19: **Error Status (0x08)** *(cont'd)*

Bits	Name	Reset Value	Access Type	Description
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.

Notes:

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

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 the Error Enable register is not set, then the bit in the Error Status register is not set and no error signaling occurs. To enable all errors, set the Error Enable register to `0xFFFF_FFFF`.

This enables/disables error reporting on the Error Status register.

Table 20: **Error Enable (0x0C)**

Bits	Name	Reset Value	Access Type	Description
31	MSTIRQEN	0x1	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].

Table 20: **Error Enable (0x0C)** *(cont'd)*

Bits	Name	Reset Value	Access Type	Description
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].

Master Error Interrupt Enable

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

Table 21: **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

Config Status

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

Table 22: **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

Table 22: **Config Status (0x14)** *(cont'd)*

Bits	Name	Reset Value	Access Type	Description
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

Streaming Mode Register Map

The following table is available only in AXI4-Stream mode. For any other mode, these registers are not accessible.

Table 23: **Streaming Mode Register Map**

Offset	Register Name	Description
0x30	Streaming Control	Provides the current version of the AXI4-Stream interface and to enable/disable the core operation.
0x34	Streaming Config	Allows you to configure the streaming master interface (M_AXIS_MASTER) for different traffic parameters.
0x38	Transfer Length	Allows you to configure the length of packets and transaction count.
0x3C	Transfer Count	Reports the number of transactions (tlast count) generated/monitored.
0x40	User STRB/TKEEP Set 1 to 4	Allows you to configure TSTRB/TKEEP value for the last beat of the transfer.
0x44		Allows you to configure TSTRB/TKEEP value for the last beat of the transfer.
0x48		Allows you to configure TSTRB/TKEEP value for the last beat of the transfer.
0x4C		Allows you to configure TSTRB/TKEEP value for the last beat of the transfer.
0x50	Extended Transfer Length	Extended support to Packet Length
0x70	Streaming Error Status Register	Allows you to monitor the error occurrence.
0x74	Streaming Error Enable Register	Allows you to control the error status.
0x78	Streaming Error Interrupt Enable Register	Allows you to control the Error Out (ERR_OUT) interrupt.
0x7C	Streaming Error Count Register	Allows you to monitor the number of errors occurred.

Streaming Control

Streaming Control register provides the current version of the AXI4-Stream interface and allows you to enable the core to generate traffic using the programmed configuration. This register is only available in the Streaming mode of operation.

Table 24: **Streaming Control (0x30)**

Bits	Name	Reset Value	Access Type	Description
31:24	Version	0x20	R	Version Value
23:2	Reserved	N/A	N/A	Reserved
1	Done	0x0	R/W1C	Transfer Done 0 = Indicates core is generating traffic when STREN 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 STREN to 0 and the current transfer is completed. This bit resets to 0 either writing 1 to this bit or enabling the core with STREN.
0	STREN	0x0	R/W	Streaming Enable 0 = Disable traffic generation 1 = Enable traffic generation

Notes:

1. W1C – Write 1 to Clear (to clear register bit, you must write 1 to corresponding bits).
2. During traffic generation if the core is forced to stop traffic (either by writing STREN to 0 or using `core_ext_stop` pin), the core completes the current transfer gracefully before stopping.

Streaming Config

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

Table 25: **Streaming Config (0x34)**

Bits	Name	Reset Value	Access Type	Description
31:16	PDLY	0x0	R/W	Programmable delay (in clocks) between two streaming packets.
15:8	TDEST	0x0	R/W	Value to drive on TDEST port.
7:3	Reserved	N/A	N/A	Reserved

Table 25: **Streaming Config (0x34)** *(cont'd)*

Bits	Name	Reset Value	Access Type	Description
2	ETKTS	0x0	R/W	Enable User TSTRB/TKEEP Setting When set, core places your specified STRB/KEEP value on the last beat of the transfer. When this bit is 0, core places internally generated STRB/KEEP value on the last beat of the transfer. You need to set Support Sparse Strobe Keep along with this bit to generate sparse STRB/KEEP values.
1	RANDLY	0x0	R/W	Enable Random Delay When set, generates random delay between streaming transactions. For example, from TLAST to next TVALID.
0	RANLEN	0x1	R/W	Enable Random Length When set, generates streaming transactions with random length. When this bit is 0, core generates the streaming transaction with the length specified in Transfer Length register

Transfer Length

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

Table 26: **Transfer Length (0x38)**

Bits	Name	Reset Value	Access Type	Description
31:16	TCNT	0x0	R/W	Transaction Count Core generates this many transaction on AXI4-Stream master 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 further.

Transfer Count

Transfer Count register allows you to monitor the number of transactions generated/received based on the mode in which the core is operating. This register is only available in the Streaming mode of operation.

Table 27: **Transfer Count (0x3C)**

Bits	Name	Reset Value	Access Type	Description
31:0	TLSTCNT	0x0	R	**Master Only**: Reports the number of streaming transactions (count of tlast) on master interface **Master Loopback**: Reports number of streaming transaction (count of tlast) on slave interface **Slave Loopback**:Reports number of streaming transaction (count of tlast) on master interface

User STRB/TKEEP Set 1 to 4

These four registers allow you to set the TSTRB/TKEEP value for the last beat of the transfer. This register along with TLEN allows you to generate transfers with byte level granularity.

Table 28: **User STRB/TKEEP Set 1 to 4 (0x40 to 0x4C)**

Bits	Name	Reset Value	Access Type	Description
31:0	TKTS	0x0	R/W	TSTRB/TKEEP value to be appeared on the last beat of the transfer.

Notes:

1. For a 32-bit wide TDATA bus to generate a TKEEP/TSTRB value of 0x3, register 0x40 needs to be written with 0x3. 0x44 to 0x4C can be ignored in 32-bit TDATA width case.Because maximum TDATA width supported is 1,024 bits, 128 bits are needed to specify TSTRB/TKEEP values. In such a case, your value of TSTRB/TKEEP needs to be written to 0x40 to 0x4C with least significant bits set in 0x40.

Extended Transfer Length

Extended Transfer Length register allows you to extend the length of packets. The value in Bits[7:0] of this register is concatenated to Bits[15:0] of the Transfer Length to determine the Length in a packet. This register is only available in the Streaming mode of operation

Table 29: **Extended Transfer Length (0x50)**

Bits	Name	Reset Value	Access Type	Description
31:8	Reserved	N/A	N/A	Reserved
7:0	Ext-TLEN	0x0	R/W	When Random Length in Streaming Config register is not set, Length programmed in this register is used. TLEN in a packet. This value is concatenated to a value in the Transfer Length Bits[15:0] to give a maximum value of 2^{24} – 1 beats in the Streaming packet.

Streaming Error Status Register

Streaming Error Status Register allows you to monitor the error occurrence. This register is only available in the Streaming mode of operation.

Table 30: **Streaming Error Status Register (0x70)**

Bits	Name	Reset Value	Access Type	Description
31:1	Reserved	N/A	N/A	Reserved
0	AXIS_ESR	0x0	R	Error Status This bit is set to 1 only when an error occurs and the Error Enable bit is enabled.

Notes:

1. Error Status bit is 1 after it is set, so to reset this bit to 0 the register must be read.

Streaming Error Enable Register

Streaming Error Enable Register allows you to control the Error Status. This register is only available in the Streaming mode of operation.

Table 31: **Streaming Error Enable Register (0x74)**

Bits	Name	Reset Value	Access Type	Description
31:1	Reserved	N/A	N/A	Reserved
0	AXIS_EER	0x0	R/W	Error Status 1= Enables the Error Status bit 0= Disables the Error Status bit

Streaming Error Interrupt Enable Register

Streaming Error Interrupt Enable Register allows you to control the Error Out (ERR_OUT) interrupt. This register is only available in the Streaming mode of operation.

Table 32: **Streaming Error Enable Register (0x78)**

Bits	Name	Reset Value	Access Type	Description
31:1	Reserved	N/A	N/A	Reserved
0	AXIS_EIER	0x0	R/W	Error Interrupt Enable 1= Enables the Error Out bit 0= Disables the Error Out bit

Notes:

1. Error Out interrupt is set when an error occurred and Error Interrupt Enable bit is enabled.

Streaming Error Count Register

Streaming Error Count Register allows you to monitor the number of errors occurred. This register is only available in the Streaming mode of operation..

Table 33: **Streaming Error Count Register (0x7C)**

Bits	Name	Reset Value	Access Type	Description
31:16	Reserved	N/A	N/A	Reserved
15:0	AXIS_ECR	0x0	R	Error Count Reports number of errors occurred.

Static Mode Register Map

The following table is available only in AXI4 Static mode. For any other mode, this register is not accessible.

Table 34: **Static Mode Register Map**

Offset	Register Name	Description
0x60	Static Control	Provides the current version of the Static Mode and to enable/disable the core operation.
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.

Table 35: **Static Mode Control (0x60)**

Bits	Name	Reset Value	Access Type	Description
31:24	Version	0x20	R	Version Value
23:2	Reserved	N/A	N/A	Reserved
1	DONE	0x0	R/W1C	Transfer Done 0 = Indicates 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

Notes:

1. W1C – Write 1 to Clear (to clear register bit, you must write 1 to corresponding bits).
2. During traffic generation if the core is forced to stop traffic (either by writing STREN to 0 or using `core_ext_stop` pin) core completes the current transfer gracefully before stopping.

Static Length

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

Table 36: **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. The reset value is the value configured for Burst Length in the Vivado IDE.

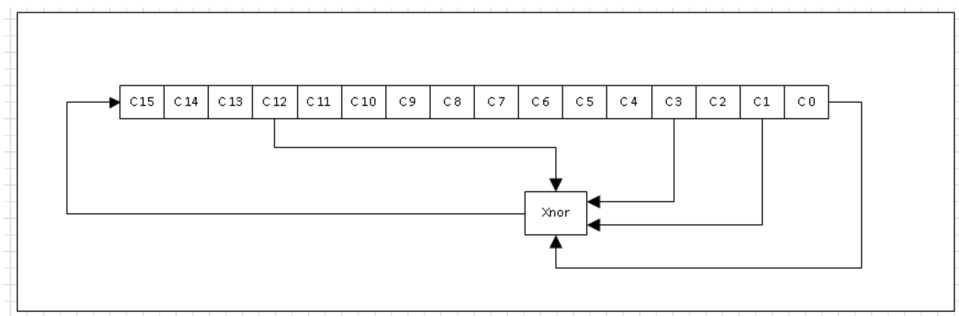
Notes:

1. Value programmed in this register directly appears in awlen/arlen on the M_AXI interface.

LFSR Implementation used for Random Generation

The following linear feed-back shift register (LFSR) is used for random address or data generation.

Figure 18: **LFSR**



Initially, the C0 to C15 flops are loaded with the input seed value. Later, it behaves as a shift register as per the architecture.

Chapter 4

Designing with the Core

This section 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, High Level Traffic, System Init/Test), and AXI4-Stream mode. You should connect the appropriate clock to this clock input.

Resets

The `s_axi_aresetn` is an active-Low synchronous reset to the core. All registers are reset to power-on conditions and all internal logic is returned to power-on conditions.

Chapter 5

Design Flow Steps

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

- *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994)
- *Vivado Design Suite User Guide: Designing with IP* (UG896)
- *Vivado Design Suite User Guide: Getting Started* (UG910)
- *Vivado Design Suite User Guide: Logic Simulation* (UG900)

Customizing and Generating the Core

This section includes information about using AMD tools to customize and generate the core in the AMD Vivado™ Design Suite.

If you are customizing and generating the core in the Vivado IP integrator, see the *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994) for detailed information. IP integrator might auto-compute certain configuration values when validating or generating the design. To check whether the values do change, see the description of the parameter in this chapter. To view the parameter value, run the `validate_bd_design` command in the Tcl console.

You can customize the IP for use in your design by specifying values for the various parameters associated with the IP core using the following steps:

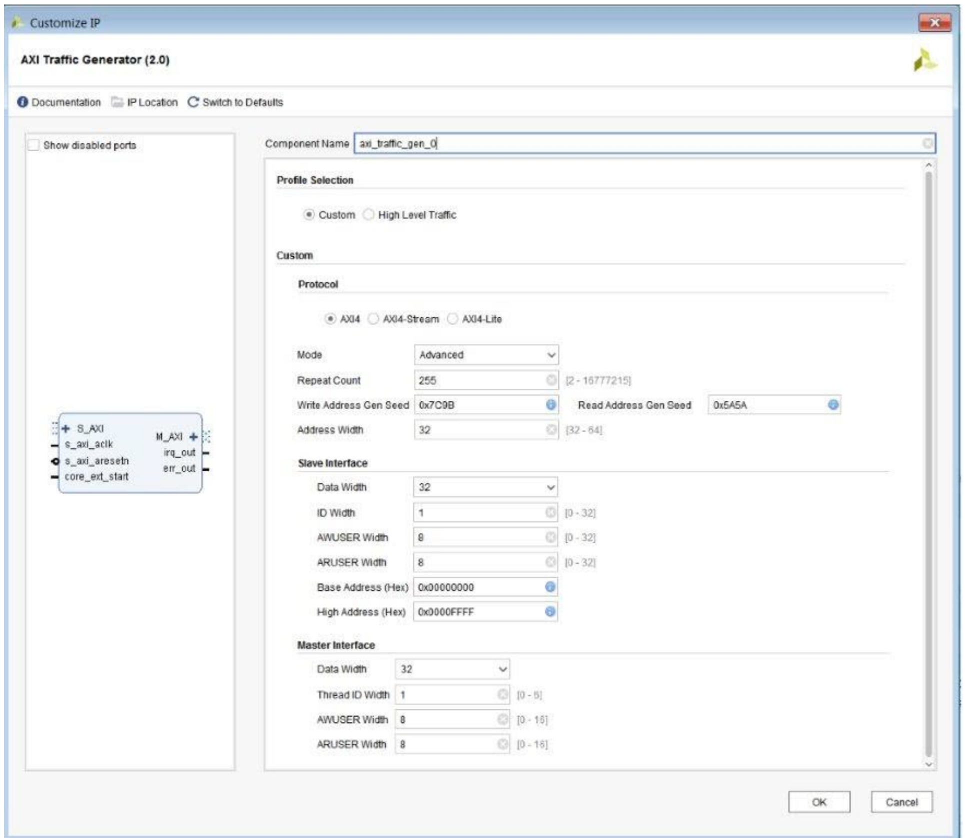
1. Select the IP from the IP catalog.
2. Double-click the selected IP or select the Customize IP command from the toolbar or right-click menu.

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

Figures in this chapter are illustrations of the Vivado IDE. The layout depicted here might vary from the current version.

The following figure shows the Customize IP window settings for the AXI Traffic Generator IP core.

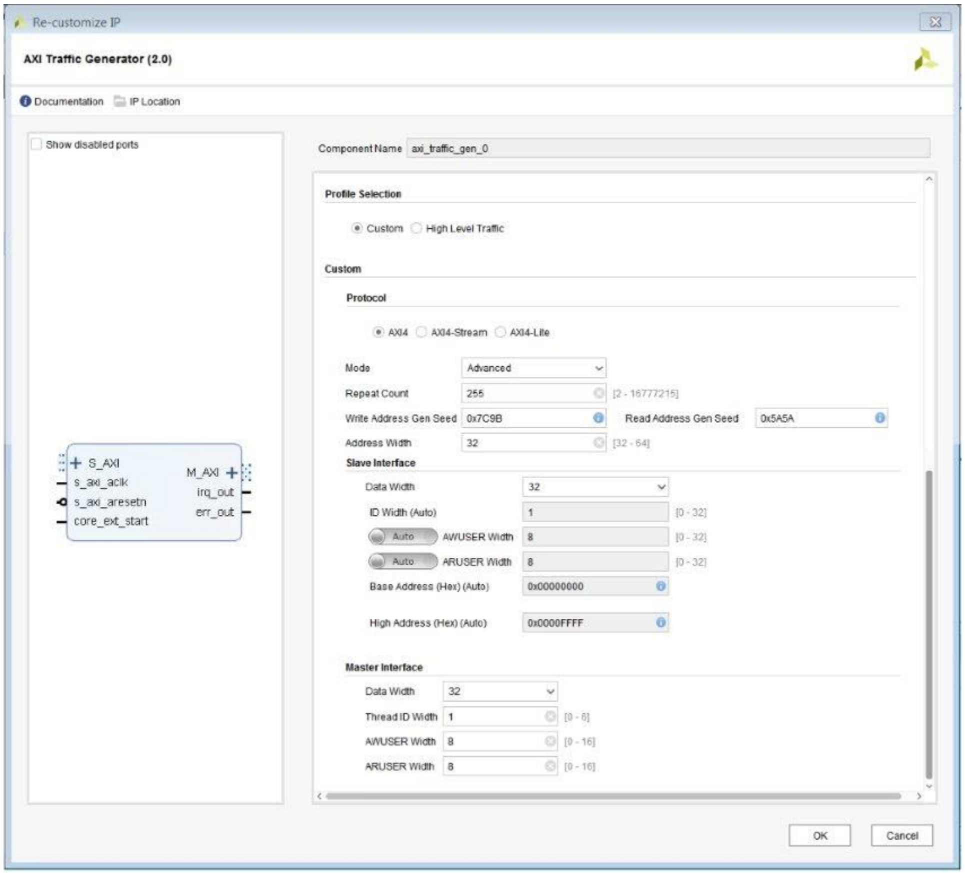
Figure 19: **Vivado Customize IP Dialog Box**



***Note*:** In the output AXI4 Master Interface, only the ports related to the configuration are exposed, the others are removed. For example, in a read-only mode, only the read ports are available.

The following figure shows the Customize IP window settings with IP integrator.

Figure 20: **Vivado Customize IP Dialog Box with IP Integrator**



- **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 "_."

Profile Selection

- **Custom or High Level Traffic:** Select the desired profile.

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 configurations. The available modes of operations are Advanced, Basic, and Static.

Advanced/Basic Mode

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

Repeat Count

This parameter gives the repeat count for all the transactions with Fixed-Repeat Delay mode set in PARAMRAM.

***Note*:** Earlier, the count was fixed to 255 by the IP core.

Write Address Gen Seed

When the ATG is configured to operate in Advanced mode and with Random Address generation, the provided seed determines the Random Address generated on the Write Address channel. This option is useful in cases where there are multiple ATGs in a system and you need a different address to be generated by each ATG.

This parameter can take any four-digit Hex value except `0xFFFF`. If a value of `FFFF` is set, Address would remain constant on the channel.

Read Address Gen Seed

When the ATG is configured to operate in Advanced mode and with Random Address generation, the provided seed determines the Random Address generated on the Read Address channel. This option is useful in cases where there are multiple ATGs in a system and you need a different address to be generated by each ATG.

This parameter can take any four-digit Hex value except `0xFFFF`. If a value of `FFFF` is set, Address would remain constant on the channel.

Address Width

This parameter determines the width of the read/write address ports on the Master AXI4 interface in the AXI4 mode of operation.

***Note*:** In earlier versions, this value was fixed to 32.

Slave Interface Options

- **Data Width:** Select the desired slave data width (32 and 64).
- **ID Width:** ID width of the slave. In IP integrator, it is auto-computed based on the master interface.
- **AWUSER:** Write channel user signals width. In IP integrator, it is auto-computed based on the master interface.

- **ARUSER:** Read channel user signals width. In IP integrator, it is auto-computed based on the master interface.
- **Base Address:** Base address of the core (used by the Vivado tool when creating a system using IP integrator). In IP integrator, it is auto-computed based on the master interface. This allows you to override the auto-compute values.
- **High Address:** High Address of the core (used by the Vivado tool when creating a system using IP integrator). In IP integrator, it is auto-computed based on the master interface. This allows you to override the auto-compute values.

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

- **Channel Select:** Selects desired channel on which traffic to be generated.
- **Enable Address Sweep:** When enabled, the address sweeps across the specified Base and High address.
- **Write Base Address:** Base/starting address for write transactions. This has to be configured based on the available memory slaves in the system.
- **Write High Address:** Only used when Address sweep is enabled. Write transactions generated are between Base and High address, This has to be configured based on available memory slaves in the system.
- **Read Base Address:** Base/starting address for read transactions. This has to be configured based on the available memory slaves in the system.
- **Read High Address:** Only used when Address sweep is enabled, Read transactions generated are between Base and High address, This has to be configured based on available memory slaves in the system.

- **Write Base Address (MSB):** MSB bits of base/starting address for write transactions. This has to be configured based on the available memory slaves in the system and it is only applicable when the address width is > 32.
- **Write High Address (MSB):** MSB bits are only used when address sweep is enabled. Write transactions generated are between base and high address, Only applicable when the address width is > 32.
- **Read Base Address (MSB):** MSB bits of base/starting address for read transactions. This has to be configured based on the available memory slaves in the system and it is only applicable when the address width is > 32.
- **Read High Address (MSB):** MSB bits are only used when address sweep is enabled, Read transactions generated are between base and high address, Only applicable when address width is > 32.

  Note: Though the Vivado IDE allows a complete 32-bit value for *(MSB) parameters, only the applicable bits (determined based on the address width configured) are considered and driven on the address ports.

- **Burst Length:** Burst length for read/write transactions.

AXI4-Stream Protocol

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

- **Support Sparse Strobe Keep:** Allows generation of sparse strobe and keeps on the last data beat of the streaming transaction, by picking values from the 'User STRB/TKEEP set 1 to 4' register space.
- **Channel Type:** Streaming mode selection. Allowed values are master only, master loopback, and slave loopback.
- **Data Generator Seed:** Controls the random data being generated on the TDATA channel in master only and master loopback modes. This parameter can take any four-digit Hex value except `0xFFFF`. If a value of `FFFF` is set, Data remains constant on the channel.
- **TDATA Width:** Selects the desired streaming data width 8 to 1,024 in multiples of 8 (for example, 8, 16, 32, and so on). In IP integrator when the mode is selected in slave loopback, it is auto-computed based on the master interface.
- **TUSER Width:** Width of streaming user signals. In IP integrator when the mode is selected in slave loopback, it is auto-computed based on the master interface.
- **TID Width:** Data stream identifier width. In IP integrator when the mode is selected in slave loopback, it is auto-computed based on the connected master interface.

- **TDEST Width:** Data stream routing information identifier width. In IP integrator when the mode is selected in slave loopback, it is auto-computed based on the connected master interface.
- **Burst Count Width:** Number of bits used while generating random length value when the core is register configured to generate random length.

For example with a width of 4-bit, maximum transaction length generated in 16.

AXI4-Lite Protocol

System Init/Test mode allows you to generate the AXI4 transaction on the master interface. Transactions are generated based on the configuration file you provided. When core generates all transactions, it asserts the `done` and `status` signals indicating the status of the generation.

- **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:** Loads/creates the data COE file. Contains data entries for the corresponding address in the Address COE file.
- **Address COE File:** Loads/creates the address COE file. Contains address entries for the transactions to be generated on the master interface. The end of the transaction is defined by NOP (`0xFFFFFFFF`). The core stops generating any further transactions after processing NOP.

  Note: In IP integrator, the addresses in COE files should be updated based on the address space allocated by the tool.

- **Control COE File:** Loads/creates the control COE file. Contains the control information of the type of transaction to be generated, the next COE entry to be fetched, and to count if any errors occurred.
- **Mask COE File:** Loads/creates the Mask COE File. Contains the Mask bits to be used during read data comparison.
- **Mode:**
- **System Init:** Generates write transactions.
- **System Test:** Generates write and read transactions.
- **Number of AXI Channels:** Number of AXI4-Lite interfaces available on which the transactions are generated. The core compares address entry with each channel Base/High address value and generates transactions on the matching channel.
- **CH{n}_Base Address:** Base address corresponding to channel {n}.
- **CH{n}_High Address:** High address corresponding to channel {n}.

- **Maximum Command Retry Count:** Allows you to limit the Maximum number of times the same transaction can be issued without other transactions in between.
- **Maximum Clocks to Run:** Maximum number of clocks after coming out of reset the total sequence can take after which it is declared Hang.

High Level Traffic

For High Level Traffic mode, there are two options:

- **AXI Options:** Allows you to configure AXI options like AXI master data width.
- **Profile Specific Options:** Allows you to configure profile specific options like the number of lanes in a PCIe® profile.

AXI Options

- **AXI Master Width:** Selects the desired master data width (32, 64, 128, 256, and 512).
- **Enable Address Sweep:** Enables address incrementation based on burst length for consecutive transactions generated.
- **AXI Base Address:** Base address for the master.
- **AXI High Address:** High address for the master.
- **AXI Base Address (MSB):** MSB bits of the base address for the master.
- **AXI High Address (MSB):** MSB bits of the high address for the master.

  ***Note*:** Though the Vivado IDE allows a complete 32-bit value for *(MSB) parameters, only the applicable bits (determined based on address width configured) are considered and driven on the address ports.

- **Burst Length:** Burst length of AXI transactions to be generated.
- **Channel Select:** Selects the desired channel on which traffic is to be generated.

Profile Specific Options

In the Profile Specific Options, you can configure five modes.

Video Mode

- **Hsize:** Hsize of the Video: 640 to 1,920.
- **Vsize:** Vsize of the video: 480 to 1,080.
- **Frame Rate:** Number of frames per second: 60,75.
- **Pixel Bits:** Number of pixel bits: 8, 10, 12.

- **Format:** Video format: RGB, YUV: 4:4:4, YUV: 4:2:2, YUV: 4:2:0.

PCIe Mode

- **PCIe Lanes:** Number of PCIe Lanes: 1, 2, 4, 8.
- **PCIe Lane Rate (GTps):** Lane rate of each lane in GT/s: 2.5, 5, 8, 16.
- **PCIe Load (%):** Percentage of PCIe traffic load on the bus: 1 to 100.

Ethernet Mode

- **Ethernet Speed (Mbps):** Selects Ethernet speed: 10, 100, 1,000.
- **Ethernet Load (%):** Percentage of Ethernet traffic load on the bus: 1 to 100.

USB Mode

- **USB Mode:** Mode of USB: ISOC, BULK.
- **USB Load:** Percentage of USB traffic load on the bus: 1 to 100.
  - Throughput generated with a granularity of MBps.
  - Throughput is rounded to a lower integer number.

Data Mode

This is a special case, where you can configure different AXI options to generate the desired AXI traffic for custom protocol.

- **Traffic Gen:** Generates one set of traffic or repetitive traffic (One-shot, Repetitive)

  Based on your settings, the core internally creates a command set to meet your requirements.

  - **One-shot:** Internally created command set is executed once.
  - **Repetitive:** Internally created command set is executed repeatedly until the core receives a stop pulse through `core_ext_stop`.

- **Traffic Pattern:** Length of AXI transactions, Random lengths with a minimum, maximum, and average length value or Fixed length transactions.

  When Address sweep is enabled, only a fixed traffic pattern is supported and all transactions are generated with this burst length.

  The core tries to achieve the average length on a channel as the percentage of traffic share on that channel increases.

For example, the average length of transactions generated closely matches with the Vivado IDE setting when a percentage of traffic share is 50%. Rather than with a case where a traffic share is 5%.

- **Transaction Type:** Read Only, Write Only or Read Write transactions.
- **Inter Transfer Gap (ITG) Type:** Type of the gap between issuance of two AXI transactions. Fixed (Minimum Fixed Gap in Clocks) or Random gap with a range of 0 to 1,024.

  ***Note*:** The IP in this mode always has a minimum six cycles delay between transactions, so the values from zero to six have no effect.

- **Transaction Seed:** Seed value for the traffic generation. This allows you to regenerate the same traffic with the same seed.

User Parameters

The following table shows the relationship between the fields in the AMD Vivado™ IDE and the user parameters (which can be viewed in the Tcl Console).

Table 37: **Vivado IDE Parameter to User Parameter Relationship**

Vivado IDE Parameter/Value[1]	User Parameter/Value[1]	Default Value[1]
Profile Selection Allowed value are Custom and High Level Traffic.	ATG_OPTIONS	Custom
Protocol Allowed values are AXI, AXI4-Lite, and AXI4-Stream.	C_ATG_MODE	AXI4
Mode Allowed values are Advanced, Basic, and Static.	C_ATG_MODE_L2	Advanced
Repeat Count	C_REPEAT_COUNT	255
Slave Interface Data Width Allowed values are 32 and 64.	C_S_AXI_DATA_WIDTH	32
Slave Interface ID Width Allowed values are from 0 to 32.	C_S_AXI_ID_WIDTH	1
Slave Interface AWUSER Width Allowed values range from 0 to 8.	C_S_AXI_AWUSER_WIDTH	8
Slave Interface ARUSER Width Allowed values range from 0 to 8.	C_S_AXI_ARUSER_WIDTH	8
Base Address Valid HEX Address value for slave interface.	C_BASEADDR	0x00000000
High Address Valid HEX High Address value for slave interface.	C_HIGHADDR	0x0000FFFF
Master Interface Data Width Allowed values are 32, 64, 128, 256, and 512.	C_M_AXI_DATA_WIDTH	32

Table 37: **Vivado IDE Parameter to User Parameter Relationship** *(cont'd)*

Vivado IDE Parameter/Value[1]	User Parameter/Value[1]	Default Value[1]
Master Interface Thread ID Width Allowed values range from 0 to 6.	C_M_AXI_THREAD_ID_WIDTH	1
Master Interface AWUSER Width Allowed values range from 0 to 8.	C_M_AXI_AWUSER_WIDTH	8
Master Interface ARUSER Width Allowed values range from 0 to 8.	C_M_AXI_ARUSER_WIDTH	8
Static Mode Channel Select Allowed values are Read_Only, Write_Only, and Read_Write.	C_ATG_STATIC_CH_SELECT	Read_Write
Enable Address Sweep Allowed values are TRUE and FALSE.	C_ATG_STATIC_INCR	0
Write Base Address Valid HEX base address for Write channel.	C_ATG_STATIC_WR_ADDRESS	0x12A00000
Write High Address Valid HEX High base address for Write channel.	C_ATG_STATIC_WR_HIGH_ADDRESS	0x12A00FFF
Read Base Address Valid HEX base address for Read channel	C_ATG_STATIC_RD_ADDRESS	0x13A00000
Read High Address Valid HEX High base address for Read channel.	C_ATG_STATIC_RD_HIGH_ADDRESS	0x13A00FFF
Burst Length Allowed values range from 1 to 256.	C_ATG_STATIC_LENGTH	16
Support Sparse Strb Keep Boolean values TRUE or FALSE allowed.	C_AXIS_SPARSE_EN	1
Channel Type Allowed values are Master Only, Master Loopback, and Slave Loopback.	C_AXIS_MODE	Master Only
Burst Count Width Allowed values range from 1 to 16.	C_ATG_STREAMING_MAX_LEN_BITS	16
TDATA Width Allowed values are 8, 16, 24, 32, 40, 48, 56, 64, 72, 80, 88, 96, 104, 112, 120, 128, 136, 144, 152, 160, 168, 176, 184, 192, 200, 208, 216, 224, 232, 240, 248, 256, 264, 272, 280, 288, 296, 304, 312, 320, 328, 336, 344, 352, 360, 368, 376, 384, 392, 400, 408, 416, 424, 432, 440, 448, 456, 464, 472, 480, 488, 496, 504, 512, 520, 528, 536, 544, 552, 560, 568, 576, 584, 592, 600, 608, 616, 624, 632, 640, 648, 656, 664, 672, 680, 688, 696, 704, 712, 720, 728, 736, 744, 752, 760, 768, 776, 784, 792, 800, 808, 816, 824, 832, 840, 848, 856, 864, 872, 880, 888, 896, 904, 912, 920, 928, 936, 944, 952, 960, 968, 976, 984, 992, 1000, 1008, 1016, and 1024.	C_AXIS_DATA_WIDTH	32
TUSER Width Allowed values range from 0 to 256.	C_AXIS_TUSER_WIDTH	8
TID Width Allowed values range from 0 to 16.	C_AXIS_TID_WIDTH	8

Table 37: **Vivado IDE Parameter to User Parameter Relationship** *(cont'd)*

Vivado IDE Parameter/Value[1]	User Parameter/Value[1]	Default Value[1]
TDEST Width Allowed values range from 0 to 8.	C_AXIS_TDEST_WIDTH	8
AXI4-Lite Mode Allowed values are System_Init and System_Test.	C_ATG_SYSINIT_MODES	System_Init
Transaction Depth Allowed values are 16, 32, 64, 128, and 256.	C_ATG_MIF_DATA_DEPTH	16
Number of AXI Channels Allowed values range from 1 to 5.	C_ATG_SYSTEM_MAX_CHANNELS	1
CH*- Base Address Valid HEX strings for all the channels selected with no overlaps.	C_ATG_SYSTEM_CH*_LOW	0x00000*00 "*" depends on the channel dealt with; varies from 0 to 4 based on channel number.
CH*- High Address Valid HEX strings for all the channels selected with no overlaps.	C_ATG_SYSTEM_CH*_HIGH	0x00000*FF "*" depends on the channel dealt with; varies from 0 to 4 based on channel number.
Maximum Command Retry Count Allowed values range from 1 to 4294967295.	C_ATG_SYSTEM_CMD_MAX_RETRY	256
Maximum Clocks to Run Allowed values range from 15 to 4294967295.	C_ATG_SYSTEM_TEST_MAX_CLKS	5000
Address COE file should have the complete path of Address COE file to be used by IP.	C_ATG_SYSTEM_INIT_ADDR_MIF	no_coe_file_loaded
Data COE file should have the complete path of Data COE file to be used by IP.	C_ATG_SYSTEM_INIT_DATA_MIF	no_coe_file_loaded
Mask COE file should have the complete path of Address COE file to be used by IP.	C_ATG_SYSTEM_INIT_MASK_MIF	no_coe_file_loaded
Ctrl COE file should have the complete path of Address COE file to be used by IP.	C_ATG_SYSTEM_INIT_CTRL_MIF	no_coe_file_loaded
Traffic Profile Allowed values are Video, PCIe, Ethernet, USB, and Data.	TRAFFIC_PROFILE	Video
AXI Master Width Data width of Master AXI interface and allowed values are 32, 64, 128, 256, and 512.	MASTER_AXI_WIDTH	32
Enable Address Sweep in High Level Traffic Profile (HLTP) Mode Allowed values are boolean TRUE or FALSE.	C_ATG_STATIC_HLTP_INCR	0
AXI Base Address Valid HEX string values for Address.	MASTER_BASE_ADDRESS	0x00000000
AXI High Address Valid HEX string values for Address.	MASTER_HIGH_ADDRESS	0xFFFFFFFF
Channel Select Allowed values are Read_Only, Read_Write, and Write_Only.	ATG_HLT_CH_SELECT	Read_Write

Table 37: **Vivado IDE Parameter to User Parameter Relationship** *(cont'd)*

Vivado IDE Parameter/Value[1]	User Parameter/Value[1]	Default Value[1]
Burst Length in HLTP Mode This parameter is present when Enable Address Sweep is disabled. Allowed values range from 1 to 256.	ATG_HLT_STATIC_LENGTH	16
Burst Length in HLTP Mode This parameter is present when Enable Address Sweep is enabled. Allowed values are 1, 2, 4, 8, 16, 32, 64, 128, and 256.	ATG_HLT_STATIC_LENGTH_INCR	16
HSIZE Enabled in VIDEO mode of HLTP. Allowed values range from 640 to 1920.	VIDEO_HSIZE	1920
VSIZE Enabled in VIDEO mode of HTLP. Allowed values range from 480 to 1080.	VIDEO_VSIZE	1080
Frame Rate Enabled in VIDEO mode of HLTP. Allowed values are 60 and 75.	VIDEO_FRAME_RATE	60
Pixel Bits Enabled in VIDEO mode of HLTP. Allowed values are 8, 10, and 12.	VIDEO_PIXEL_BITS	8
Format Enabled in VIDEO mode of HLTP. Allowed values are: 6 – to represent RGB 4 – to represent YUV-4:4:4 2 – to represent YUV-4:2:2 0 – to represent YUV-2:2:0	VIDEO_FORMAT	6
PCIe Lanes Enabled in PCIe mode of HLTP. Allowed values are 1, 2, 4, and 8.	PCIE_LANES	1
PCIe Lane Rate Enabled in PCIe mode of HLTP. Allowed values are 2.5, 5, 8, and 16.	PCIE_LANE_RATE	2.5
PCIe Load Enabled in PCIe mode of HLTP. Allowed values range from 1 to 100.	PCIE_LOAD	50
Ethernet Speed Enabled in Ethernet mode of HLTP. Allowed values are 10, 100, and 1000.	ETHERNET_SPEED	1000
Ethernet Load Enabled in Ethernet mode of HLTP. Allowed values range from 1 to 100.	ETHERNET_LOAD	50
USB Mode Enabled in USB mode of HLTP. Allowed values are ISOC and BULK.	USB_MODE	ISOC
USB Load Enabled in USB mode of HLTP. Allowed values range from 1 to 100.	USB_LOAD	50

Table 37: **Vivado IDE Parameter to User Parameter Relationship** *(cont'd)*

Vivado IDE Parameter/Value[1]	User Parameter/Value[1]	Default Value[1]
Traffic Gen Enabled in Data mode of HLTP. Allowed values are One_Shot and Repetitive.	C_ATG_REPEAT_TYPE	One_Shot
Traffic Pattern Enabled in Data mode of HLTP. Allowed values are Fixed and Random.	DATA_TRAFFIC_PATTERN	Random
Transfer Length (Minimum) Enabled in Data mode of HLTP. Allowed values are 1, 2, 4, 8, 16, 32, 64, 128, and 256.	DATA_SIZE_MIN	1
Transfer Length (Maximum) Enabled in Data mode of HLTP. Allowed values are 1, 2, 4, 8, 16, 32, 64, 128, and 256.	DATA_SIZE_MAX	256
Transfer Length (Average) Enabled in Data mode of HLTP. Allowed values are 1, 2, 4, 8, 16, 32, 64, 128, and 256.	DATA_SIZE_AVG	32
Transaction Type Enabled in Data mode of HLTP. Allowed values are Read_Only, Write_Only, and Read_Write.	DATA_TRANS_TYPE	Read_Write
Read Share Enabled in Data mode of HLTP. Allowed values range from 0 to 100.	DATA_READ_SHARE	50
Write Share Enabled in Data mode of HLTP. Allowed values range from 0 to 100.	DATA_WRITE_SHARE	50
ITG Type Enabled in Data mode of HLTP. Allowed values are Fixed and Random.	DATA_TRANS_GAP	Fixed
ITG (Clocks) Enabled in Data mode of HLTP. Allowed values range from 0 to 1023.	DATA_ITG_GAP	5
Transaction Seed Enabled in Data mode of HLTP. Allowed values range from 1 to 100.	DATA_TRANS_SEED	1
Write Base Address (MSB)	C_ATG_STATIC_WR_ADDRESS_EXT	0x00000000
Write High Address (MSB)	C_ATG_STATIC_WR_HIGH_ADDRESS_EXT	0x00000000
Read Base Address (MSB)	C_ATG_STATIC_RD_ADDRESS_EXT	0x00000000
Read High Address (MSB)	C_ATG_STATIC_RD_HIGH_ADDRESS_EXT	0x00000000
AXI Base Address (MSB)	MASTER_BASE_ADDRESS_EXT	0x00000000
AXI High Address (MSB)	MASTER_HIGH_ADDRESS_EXT	0x00000000
Write Address Gen Seed	AXI_WR_ADDR_SEED	0x7C9B
Read Address Gen Seed	AXI_RD_ADDR_SEED	0x5A5A
Data Generator Seed	STRM_DATA_SEED	0xABCD

Notes:

1. Parameter values are listed in the table where the Vivado IDE parameter value differs from the user parameter value. Such values are shown in this table as indented below the associated parameter.

Output Generation

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

Constraining the Core

This section contains information about constraining the core in the Vivado Design Suite. There are no IP-specific constraints other than the AXI clock constraint and the necessary constraints delivered when IP is generated. This core generates the out-of-context (OOC) XDCs.

Required Constraints

This section is not applicable for this IP core.

Device, Package, and Speed Grade Selections

This section is not applicable for this IP core.

Clock Frequencies

This section is not applicable for this IP core.

Clock Management

This section is not applicable for this IP core.

Clock Placement

This section is not applicable for this IP core.

Banking

This section is not applicable for this IP core.

Transceiver Placement

This section is not applicable for this IP core.

I/O Standard and Placement

This section is not applicable for this IP core.

Simulation

For comprehensive information about AMD Vivado™ simulation components, as well as information about using supported third-party tools, see the *Vivado Design Suite User Guide: Logic Simulation* (UG900).

***Note*:** For cores targeting 7 series FPGAs or Zynq 7000 SoC devices, UNIFAST libraries are not supported. AMD IP is tested and qualified with UNISIM libraries only.

Synthesis and Implementation

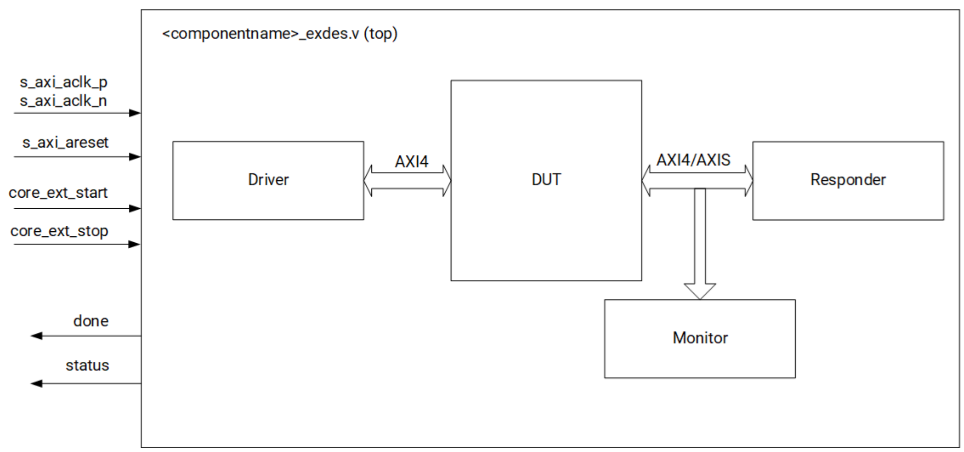
For details about synthesis and implementation, see the *Vivado Design Suite User Guide: Designing with IP* (UG896).

Example Design

This chapter contains information about the example design provided in the AMD Vivado™ Design Suite.

The top module instantiates all components of the core and example design that are needed to implement the design in hardware, as shown in the following figure. This includes the driver, responder, and monitor modules.

Figure 21: **AXI Traffic Generator Example Design Block Diagram**



This example design demonstrates the transactions on AXI4 and AXI4-Stream interfaces of the Device Under Test (DUT) based on the mode in which the DUT is configured.

- **Driver:** AXI Traffic Generator in System Test is used as a Driver to configure the DUT and checks for the pass/fail condition.
- **Responder:** AXI block RAM (BRAM) Controller is used to respond to AXI4 transactions generated by the DUT in the applicable modes.
- **Monitor:** AXI Performance Monitor is used to monitor the AXI4 transactions generated by the DUT in High Level Traffic modes.

Implementing the Example Design

After following the steps described in Customizing and Generating the Core to generate the core, implement the example design as follows:

1. Right-click the core in the Hierarchy window, and select **Open IP Example Design**.
2. A new window pops up, asking you to specify a directory for the example design. Select a new directory or keep the default directory.

   A new project is automatically created in the selected directory and it is opened in a new Vivado window.
3. In the Flow Navigator (left-side pane), click **Run Implementation** and follow the directions.

Example Design Directory Structure

In the current project directory, a new project with the name `<componentname>_example` is created. This directory and its subdirectories contain all the source files that are required to create the AXI Traffic Generator example design.

The following table shows the files delivered in the `<componentname>_example/<componentname>_example.srcs/` directory. This `<component_name>/example_design` directory contains the generated example design top files.

Table 38: **Example Design Directory**

Name	Description
<component_name>_exdes.xdc	Top-level constraints file for the example design.
<componentname>_exdes.v	Top-level HDL file for the example design.

Simulating the Example Design

For more information on Simulation, refer to *Vivado Design Suite User Guide: Logic Simulation* (UG900).

Simulation Results

The simulation script compiles the AXI Traffic Generator example design and supporting simulation files. It then runs the simulation and checks to ensure that it is completed successfully.

If the test passes, then the following message is displayed:

```
Test Completed Successfully
```

If the test fails, then the following message is displayed:

```
ERROR: Test Failed
```

If the test hangs, then the following message is displayed:

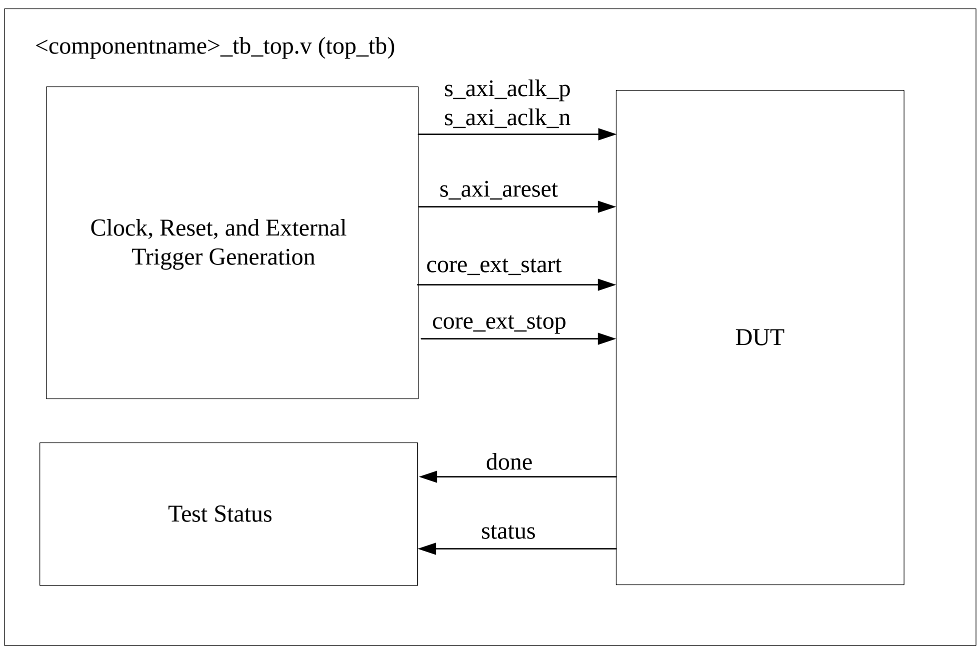
```
ERROR: Test did not complete (timed-out)
```

Test Bench

This chapter contains information about the test bench provided in the AMD Vivado™ Design Suite.

The following figure shows the test bench for the AXI Traffic Generator example design. The top-level test bench generates a 100 MHz clock, external start/stop triggers pulses based on the mode, and drives an initial reset to the example design.

Figure 22: **AXI Traffic Generator Example Design Test Bench**



Example Sequences for Custom Mode

Protocol – AXI4

Advanced/Basic

1. Configure CMDRAM/MSTRAM with a write and read transactions.

2. Enable interrupt generation for transfer completion.
3. Enable core to start generating traffic.
4. Poll for transfer completion status.

Static

1. Configure Length (`0x8`) to Static Length register.
2. Enable to start generating traffic.
3. Perform a dummy read to the Static Length register before disabling the core.
4. Disable the core to stop the generation of traffic.
5. Poll for a done bit in Static Control register.

Protocol – AXI4-Stream

Master Only/Master Loopback

1. Configure Transfer Length (`0x40009`) to Transfer Length register.
2. Enable the core to start generating streaming traffic.
3. Poll for done bit in Streaming Control register.
4. Read the Status register to compare the number of transactions generated versus programed value.

Slave Loopback

1. Program streaming source (AXI Traffic Generator in Master Only mode used) to generate four streaming transactions of length nine.
2. Poll for a done bit in the streaming source.
3. Read the Status register of DUT to compare the number of transactions generated versus the programmed value.

Protocol – AXI4-Lite

System Init/System Test

1. Poll for a done bit from the DUT and report status of AXI4-Lite traffic generation sequence programmed through COE file.

Example Sequences for High Level Traffic (Video,PCIe, Ethernet, USB) Modes

1. Program AXI Performance Monitor to count the number of bytes transferred on AXI4 Channels.
2. Generate a start pulse to the core through `core_ext_start`.
3. Wait for 1 ms.
4. Generate a stop pulse to core through `core_ext_stop`.
5. Poll metric counters of AXI Performance monitor to compare the number of bytes generated versus programmed value based on the throughput required.

Example Sequences for High Level Traffic (Data)

1. Program AXI Performance Monitor to count the number of bytes transferred on AXI4 Channels.
2. Generate a start pulse to the core through `core_ext_start`.
3. Wait for 1 ms.
4. Poll metric counters of AXI Performance monitor to compare the number of bytes generated versus programmed value based on the percentage of traffic share on each channel.

Upgrading

This appendix contains information about upgrading to a more recent version of the IP core.

Migrating to the Vivado Design Suite

For information on migrating to the AMD Vivado™ Design Suite, see the *ISE to Vivado Design Suite Migration Guide* (UG911).

Upgrading in the Vivado Design Suite

This section provides information about any changes to the user logic or port designations between core versions.

Port Changes

The following ports are added in this release:

- `core_ext_start`
- `core_ext_stop`
- `s_axis_master` - Streaming interface
- `axis_err_count`
- `m_axi_lite` – Four new AXI4-Lite interfaces added
- `done`
- `status`

***Note*:** For more information on port descriptions, see Port Descriptions.

Appendix B

Debugging

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

If the IP requires a license key, the key must be verified. The AMD Vivado™ design tools have several license checkpoints for gating licensed IP through the flow. If the license check succeeds, the IP can continue generation. Otherwise, generation halts with an error. License checkpoints are enforced by the following tools:

- 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.*

Finding Help with AMD Adaptive Computing Solutions

To help in the design and debug process when using the core, the Support web page contains key resources such as product documentation, release notes, answer records, information about known issues, and links for obtaining further product support. The Community Forums are also available where members can learn, participate, share, and ask questions about AMD Adaptive Computing solutions.

Documentation

This product guide is the main document associated with the core. This guide, along with documentation related to all products that aid in the design process, can be found on the Support web page or by using the AMD Adaptive Computing Documentation Navigator. Download the Documentation Navigator from the Downloads page. For more information about this tool and the features available, open the online help after installation.

Answer Records

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

Answer Records for this core can be located by using the Search Support box on the main Support web page. To maximize your search results, use 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 Core

AR: 54426.

Technical Support

AMD Adaptive Computing provides technical support on the Community Forums for this AMD LogiCORE™ IP product when used as described in the product documentation. AMD Adaptive Computing cannot guarantee timing, functionality, or support if you do any of the following:

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

To ask questions, navigate to the Community Forums.

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 Design Suite Debug Feature

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

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)

See the *Vivado Design Suite User Guide: Programming and Debugging* (UG908).

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 AMD Vivado™ debug feature is a valuable resource to use in hardware debug. The signal names mentioned in the following individual sections can be probed using the debug feature for debugging the specific problems.

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:
  1. Ensure `valid_cmd` bits are set properly while loading commands to command RAM.
  2. Check `My_depend`, Other depend fields are set correctly so as not to cause a dead-lock situation.
  3. 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:
  1. Ensure `valid_cmd` bits are set properly while loading commands to command RAM.
  2. Check if `Reg0_master_control` [18:0] is set to 0.
  3. Check delay values programmed to PARAMRAM and wait for sufficient time for the core to insert these delays while generating the transactions.
- In Streaming mode, the core is generating a random length of transactions instead of the programed length.
  1. Ensure RANLEN is set to 0 in the Streaming Config register before enabling the core.

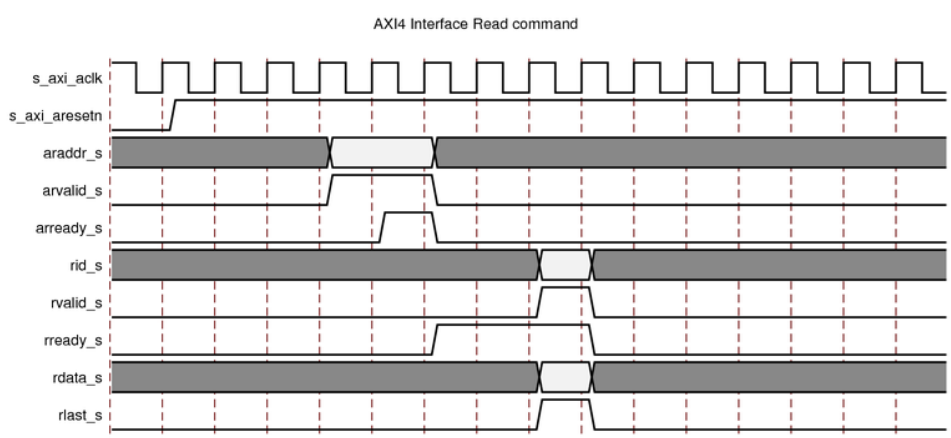
Interface Debug

AXI4 Interfaces

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 the following figure 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 the enables are also asserted.
- If the simulation has been run, verify in simulation and/or a Vivado Design Suite debug feature capture that the waveform is correct for accessing the AXI4 interface.

Figure 23: **AXI4 Read Timing Diagram**



Appendix C

Additional Resources and Legal Notices

Finding Additional Documentation

Documentation Portal

The AMD Adaptive Computing Documentation Portal is an online tool that provides robust search and navigation for documentation using your web browser. To access the Documentation Portal, go to https://docs.xilinx.com.

Documentation Navigator

Documentation Navigator (DocNav) is an installed tool that provides access to AMD Adaptive Computing documents, videos, and support resources, which you can filter and search to find information. To open DocNav:

- From the AMD Vivado™ IDE, select **Help → Documentation and Tutorials**.
- On Windows, click the **Start** button and select **Xilinx Design Tools → DocNav**.
- At the Linux command prompt, enter `docnav`.

***Note*:** For more information on DocNav, refer to the *Documentation Navigator User Guide* (UG968).

Design Hubs

AMD Design Hubs provide links to documentation organized by design tasks and other topics, which you can use to learn key concepts and address frequently asked questions. To access the Design Hubs:

- In DocNav, click the **Design Hubs View** tab.
- Go to the Design Hubs web page.

Support Resources

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

References

These documents provide supplemental material useful with this guide:

1. *Vivado Design Suite User Guide: Designing with IP* (UG896)
2. *Vivado Design Suite: AXI Reference Guide* (UG1037)
3. *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994)
4. *Vivado Design Suite User Guide: Getting Started* (UG910)
5. *Vivado Design Suite User Guide: Logic Simulation* (UG900)
6. *ISE to Vivado Design Suite Migration Guide* (UG911)
7. *Vivado Design Suite User Guide: Programming and Debugging* (UG908)
8. *Vivado Design Suite User Guide: Implementation* (UG904)
9. Arm AMBA AXI Protocol Specification, version 2.0 (Arm IHI 0022C)
10. AMBA AXI4-Stream Protocol Specification

Revision History

The following table shows the revision history for this document.

Section	Revision Summary
10/18/2023 Version 3.0	
Streaming Mode with Processor	Addded a note.
02/11/2019 Version 3.0	
N/A	Updated Streaming Mode Register Map section.
04/04/2018 Version 3.0	
N/A	• Updated the invalid command information in Note 1 of Table 1-2. • Added one Note under Table 2-3.

Section	Revision Summary
10/04/2017 Version 3.0	
N/A	• Revision number advanced to 3.0 to align with the core version number. • Updated the Output AXI4 Master Interface ports.
04/05/2017 Version 2.0	
N/A	• Updated the figures to customize the core. • Updated the User Parameters table. • Added the LFSR implementation.
04/06/2016 Version 2.0	
N/A	• Updated Master RAM section. • Updated figures in Design Flow Steps chapter. • Added Write Address Gen Seed and Data Generator Seed descriptions. • Updated User Parameters table.
11/18/2015 Version 2.0	
N/A	Added support for UltraScale+ families.
09/30/2015 Version 2.0	
N/A	• Updated Features description in IP Facts. • Updated AXI4 Traffic Generator Block Diagram. • Added Address RAM section. • Moved Performance and Resource Utilization to HTML. • Updated Slave-Write/Slave-Read Address Map. • Added description to Static Mode. • Added description in High Level Traffic. • Added Extended Transfer Length register to Streaming Mode Register Map. • Updated figures in Design Flow Steps chapter. • Added Address Width section in AXI4 Protocol section. • Updated description in Static Mode Options section. • Updated description in AXI Options section. • Added parameters to Vivado IDE Parameter to User Parameter Relationship.
04/01/2015 Version 2.0	
N/A	• Updated description in 011 in PARAMRAM Entry Control Signals table. • Added description in PARAMRAM Opcodes section. • Updated GUIs in Customizing and Generating the Core section. • Added Repeat Count section. • Added User Parameters section. • Added UNISIM important note in Simulation section.

Section	Revision Summary
10/01/2014 Version 2.0	
N/A	• Document updates only for revision change. • Added note #4 in Table 1-2: CMDRAM Memory Format. • Added notes to Table 1-3: PARAMRAM Entry Control Signals. • Added note #2 to Table 1-6: OP_DELAY. • Updated description in Static Mode section. • Added Note and Important note in High Level Traffic section. • Added note #3 to Table 2-3: AXI Traffic Generator I/O Signals. • Added note to Table 2-19: Static Length. • Updated Static Mode Options section. • Updated AXI Options section.
04/02/2014 Version 2.0	
N/A	• Updated Flexible data width capabilities and initialization support in Features section. • Updated Basic description in AXI Traffic Generator Modes table. • Updated AXI4-Stream Traffic Generator Block Diagram figure. • Updated description in Command RAM section. • Updated Bits[31:29] descriptions in PARAMRAM Entry Control Signals table. • Updated Mstram_index offsets in Address Generation section. • Updated CMDRAM and MSTRAM region in Slave-Write/ Slave-Read Address Map table and description • Added entries are 32-bit and Cascading ATGs description in System Init/ Test Mode section. • Added High Level Traffic section and updated System Init/Test Mode status description in Programming Sequence section. • Updated description in Advanced/Basic Mode without Processor Intervention section. • Updated Resource Utilization section. • Updated TDATA Width description in AXI4-Stream Protocol section. • Updated Bit[19] descriptions in Master Control (0x00) register. • Added Bit[2] in Streaming Config (0x34) register. • Added User STRB/TKEEP Set 1 to 4 (0x40 to 0x4C) registers. • Added traffic generation note to Streaming Control (0x30) and Static Mode Control (0x60) registers. • Added IP integrator note in AXI4-Lite Protocol section. • Added One-shot and Repetitive descriptions in Data Mode section.

Section	Revision Summary
12/18/2013 Version 2.0	
N/A	• Added UltraScale support. • Added loop enable Bit[19] to Master Control register 0x0.
10/02/2013 Version 2.0	
N/A	Revision number advanced to 2.0 to align with core version number. • Added new features in IP Facts. • Added IP Integrator. • Updated Overview chapter. • Updated Resource Utilization in Product Specification chapter. • Updated Streaming Config register. • Updated Resets section. • Updated Generating and Customizing the Core chapter. • Added Simulation, Synthesis, Example Design, and Test Bench chapters. • Added Port Changes in Migrating Appendix. • Added Streaming mode to General Checks in Debug Appendix.
03/20/2013 Version 1.0	
Initial Xilinx release.	This Product Guide replaces PG094 AXI Exerciser.

Please Read: Important Legal Notices

AUTOMOTIVE APPLICATIONS DISCLAIMER

Copyright