Performance AXI Traffic Generator v1.0

LogiCORE IP Product Guide

Vivado Design Suite

PG381 (v1.0) February 12, 2021

Table of Contents

XILINX

Section I

IP Core Overview

XILINX

Chapter 1

Introduction

The Performance AXI Traffic Generator is intended for modeling traffic masters in Versal™ ACAP designs for performance evaluation of network on chip (NoC) based solutions. It is available in two versions: Non-Synthesizable for simulations only and Synthesizable for both simulations and running in the hardware. The traffic generator supports AXI3, AXI4, and AXI4-Stream protocols. The traffic generator supports multiple operating modes and offers high level of configuration options to support the dynamic workloads enabled by Versal ACAP devices.

Features

- Generates complex and configurable AMBA® AXI3, AXI4, and AXI4-Stream traffic patterns.
- The Non-Synthesizable Traffic Generator (TG) can be configured for a fixed operating mode through the GUI or it can use comma-separated values (CSV) file for more dynamic behaviors.
- The Synthesizable TG supports CSV file operating mode, real-time user controls through virtual input/output (VIOs), or processor control by an AXI4-Lite interface.
- The CSV format supports up to 511 instructions for the Synthesizable TG and an unlimited number of instructions for the Non-Synthesizable TG.
- Supports sending a beat in every clock cycle to maximize the bandwidth.
- Supports configurable transaction delay specified in AXI clock cycles.
- Supports configurable or infinite number of transactions of an instruction.
- Supports configurable or infinite looping of traffic patterns.
- Supports constant and incremental AXI IDs.
- Supports trigger inputs and outputs for dynamic traffic shaping.
- AXI protocol checking is available for the Non-Synthesizable TG.

AXI3/AXI4 Features

IMPORTANT! *AXI3 is only supported in the Non-Synthesizable TG. It is not supported for the Synthesizable TG when running simulations or in hardware.*

- Supports aligned or unaligned and full or narrow transfers

- Supports random address, auto increment address, and increment address by a specific value
- Supports configurable data patterns
- Supports control of start address and high address
- Supports Incremental, Wrap, and Fixed bursts
- Supports exclusive transactions
- Supports data integrity checking
- Optional performance counters to measure bandwidth and latency numbers for the synthesizable TG

AXI4-Stream Features

- Supports configurable bandwidths for the Non-Synthesizable TG
- Supports inter packet delay
- Supports configurable data patterns
- Supports generation of programmable `TLAST` with configurable packet length

IP Facts

LogiCORE™ IP Facts Table	
Core Specifics	
Supported Device Family[1]	Versal™ ACAP
Supported User Interfaces	AXI3[3], AXI4, and AXI4-Stream
Provided with Core	
Design Files	RTL
Example Design	Not Provided
Test Bench	Not Provided
Constraints File	XDC
Simulation Model	N/A
Supported S/W Driver	N/A
Tested Design Flows[2]	
Design Entry	Vivado® Design Suite
Simulation	For supported simulators, see the Xilinx Design Tools: Release Notes Guide.
Synthesis	Vivado Synthesis
Support	
Release Notes and Known Issues	Master Answer Record: 75781
All Vivado IP Change Logs	Master Vivado IP Change Logs: 72775
Xilinx Support web page	

Notes:

1. For a complete list of supported devices, see the Vivado® IP catalog.
2. For the supported versions of third-party tools, see the Xilinx Design Tools: Release Notes Guide.
3. AXI3 is only supported for the Non-Synthesizable TG. It is not supported for the Synthesizable TG.

Navigating Content by Design Process

Xilinx® documentation is organized around a set of standard design processes to help you find relevant content for your current development task. All Versal™ ACAP design process Design Hubs can be found on the Xilinx.com website. This document covers the following design processes:

- **Hardware, IP, and Platform Development:** Creating the PL IP blocks for the hardware platform, creating PL kernels, subsystem functional simulation, and evaluating the Vivado® timing, resource use, and power closure. Also involves developing the hardware platform for system integration. Topics in this document that apply to this design process include:
  - Port Descriptions
  - Register Space

- Clocking and Resets
- Customizing and Generating the Core

Licensing and Ordering

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

***Note*:** To verify that you need a license, check the License column of the IP Catalog. Included means that a license is included with the Vivado® Design Suite; Purchase means that you have to purchase a license to use the Performance AXI Traffic Generator.

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

XILINX

Section II

Synthesizable TG

Overview

Core Overview

Modes of Operation

The modes of operating the TG are the same for AXI4 and AXI4-Stream. The Synthesizable TG does not supportAXI3 protocol. When the Traffic Generator IP is generated, the instructions can be loaded in the following modes:

- CSV command format
- Dynamic operating mode

CSV Command Format

In CSV command format, all transactions are fed through a CSV file as shown in the following figures. The CSV file path is specified in the Vivado® IDE. This file is converted to MEM format and loaded in the instruction block RAM for the Synthesizable TG.

Figure 1: **CSV Format for AXI3/AXI4**

Any Line starts with # will be commented out

TG_NUM	cmd	txn_count	start_delay	inter_beat_delay	wdata_pattern	wdata_pat_value	data_integrity	dest_id	base_addr	high_addr	addr_incr_by	axi_addr	axi_len	axi_size	axi_id	axi_burst	axi_lock	axi_cache	axi_prot	axi_qos	axi_region	axi_user
1	START_LOOP	20	use_original_addr																			
1	WRITE	100	0	0	same_as_addr	0	enabled	0	5401_0000_0000	5401_FFFF_FFFF	auto_incr	0x0100_0000	7	5	0	WRAP	NORMAL	0	0	0	0	0
1	WAIT																					
1	READ	100	0				enabled	0	5401_0000_0000	5401_FFFF_FFFF	auto_incr	0x0100_0000	7	5	0	WRAP	NORMAL	0	0	0	0	0
1	WAIT																					
1	END_LOOP																					
1	START_LOOP	20	use_original_addr																			
1	WRITE	100	0	0	same_as_addr	0	enabled	0	4e_1000_0000	4e_FFFF_FFFF	auto_incr	0x0200_0000	F	5	0	WRAP	NORMAL	0	0	0	0	0
1	WAIT																					
1	READ	100	0				enabled	0	4e_1000_0000	4e_FFFF_FFFF	auto_incr	0x0200_0000	F	5	0	WRAP	NORMAL	0	0	0	0	0
1	WAIT																					
1	END_LOOP																					
1	START_LOOP	20	use_original_addr																			
1	WRITE	100	0	0	same_as_addr	0	enabled	0	9681_2000_0000	9681_FFFF_FFFF	auto_incr	0x0300_0000	1	5	0	WRAP	NORMAL	0	0	0	0	0
1	WAIT																					
1	READ	100	0				enabled	0	9681_2000_0000	9681_FFFF_FFFF	auto_incr	0x0300_0000	1	5	0	WRAP	NORMAL	0	0	0	0	0
1	WAIT																					
1	END_LOOP																					

Figure 2: **CSV Format for AXI4-Stream**

TG_NUM	cmd	pkt_count	inter_pkt_delay	inter_transfer_delay	tdata_pattern	tdata_pat_value	noc_dest_id	tdest_id	pkt_len	pkt_id	pkt_user
1	STREAM	2			random	0xabcdefab	0x4	0x8	0x1	0x8	0xF
1	WAIT										
1	START_LOOP	100									
1	STREAM	2			random	0xabcdefab	0x4	0x8	0x1	0x8	0xF
1	END_LOOP										
1	STREAM	2			random	0xabcdefab	0x4	0x8	0x1	0x8	0xF

Simulation Trigger for the NoC AXI TG

The Simulation Trigger for the NoC AXI TG is a companion IP to the Synthesizable/Non-Synthesizable TG. The Simulation Trigger IP is required to support the VIO or AXI4-Lite interface for access to the Synthesizable TG registers. The Simulation Trigger IP is also used for the traffic shaping synchronization of multiple TGs.

Dynamic Operating Mode

In this mode, the instructions can be loaded into the instruction BRAM using either the VIO interface from the Simulation Trigger for the NoC AXI TG IP core, or the AXI4-Lite interface. The addressing of the BRAM is specified in Block RAM Addressing: 0x8000 to 0xFFFC. In this mode configure the TG_START Register (0x4004) to manually start the traffic generators. The status can be monitored through the same VIO or AXI4-Lite interface.

High-Level Architecture

The Traffic Generator interface consists of a master interface, the signals of which can be connected directly to slave interface signals. The TG infrastructure has a powerful collection of functions that provide a wide range of features.

Synthesizable Traffic Generator Architecture

The Traffic Generator reads and executes instructions in the block RAM. CSV instructions can be loaded in the block RAM on the fly using VIO or the AXI4-Lite interface and can run unlimited test cases on the board using a single bit file. When a CSV file is specified in the Vivado IDE, the block RAM is already populated with the instruction information.

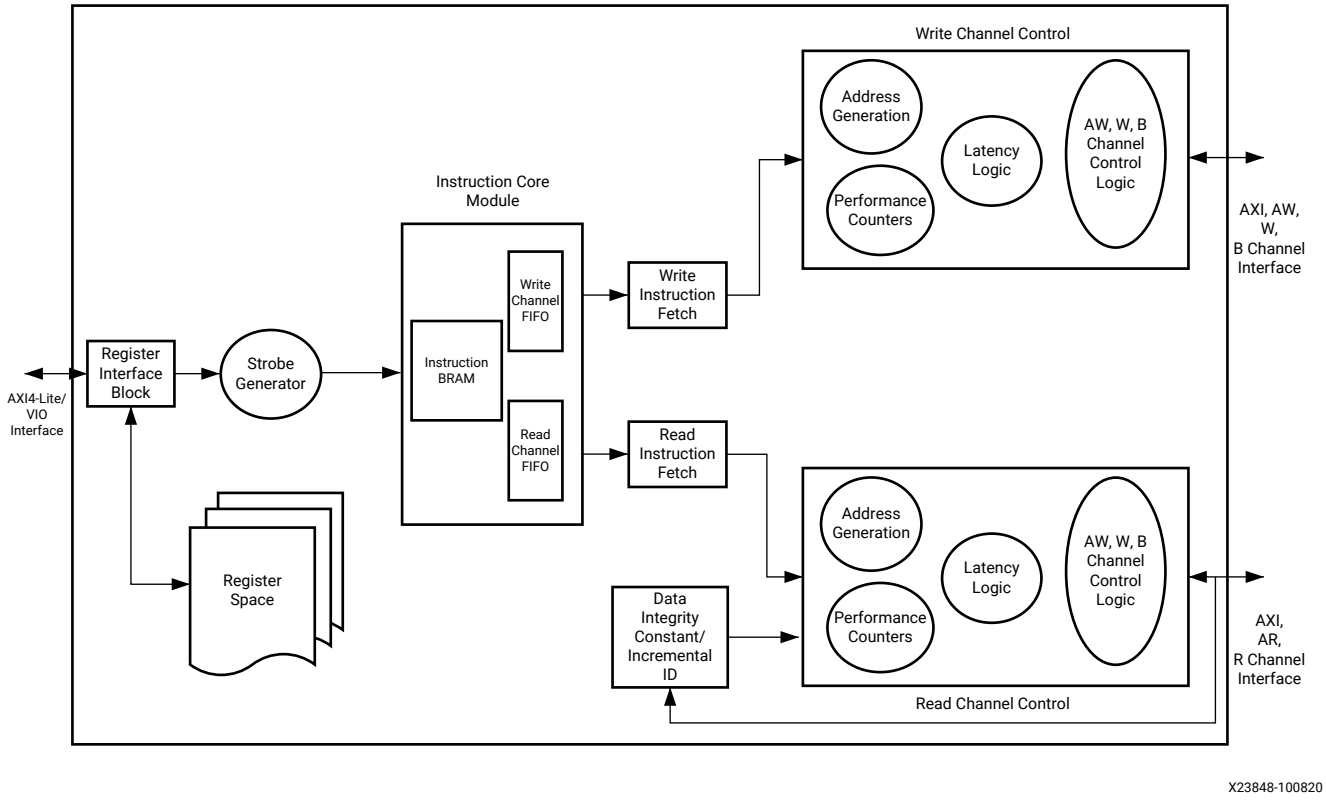
The TG has a register space with a set of configurations and status registers. Configuration registers such as soft reset or TG start are present in the register space and can be modified through VIO or AXI4-Lite interfaces.

TG status information is also stored in the status registers. You can read registers such as the bandwidth counters, transaction counters and error registers through the VIO or AXI4-Lite interfaces.

Synthesizable AXI4 Architecture

The Traffic Generator operation for AXI4 configuration is shown in the following figure.

Figure 3: **Synthesizable TG Block Diagram for AXI4**



The TG is designed to support high bandwidth bidirectional data transfers. An instruction block RAM is present in the instruction core module of the TG with a depth of 512. This means up to 512 CSV instructions can be loaded in the block RAM and each CSV instruction can have up to 65,536 Write/Read requests or a single Wait request.

When the instruction block RAM is loaded with any of the supported Modes of Operation and the TG start signal is asserted, the instruction core module reads the instruction from the block RAM and writes in the Write or Read Channel FIFO based on that instruction.

The Write Instruction Fetch module reads the data from Write Channel FIFO and gives it to the Write Channel Control module. Based on the information in the instruction fields, the AXI Write command and Write data are generated based on the data pattern selection. The second Write request is sent only after sending the data from the first Write request.

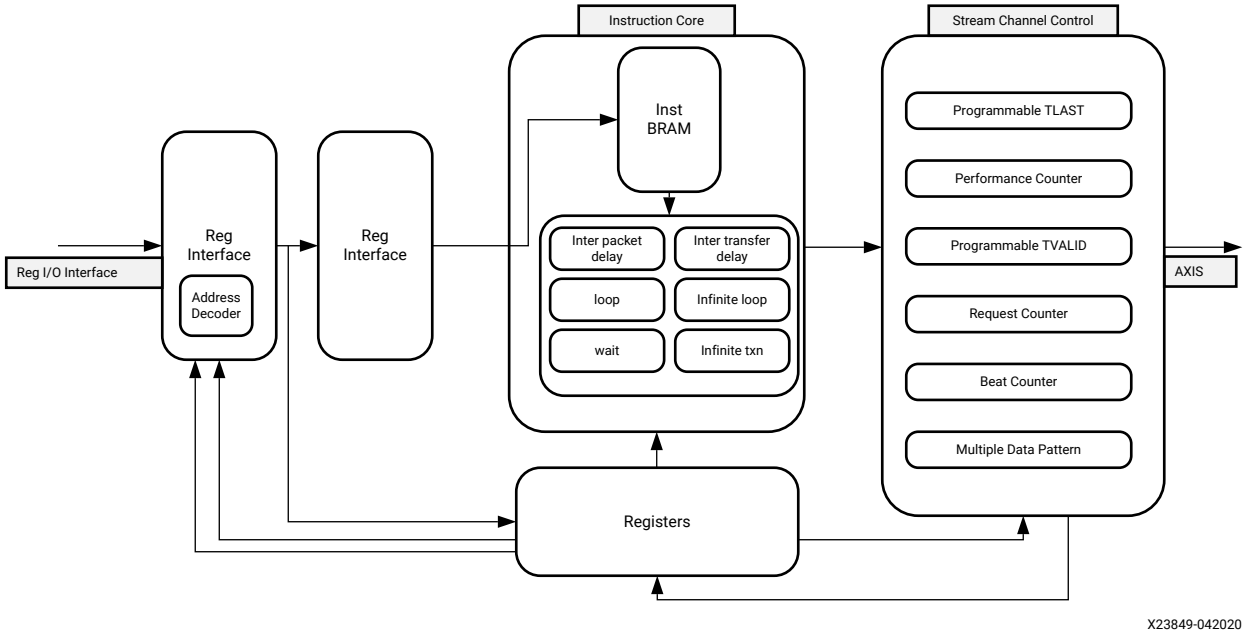
The Read Instruction Fetch module reads the data from the Read Channel FIFO and gives it to the Read Channel Control module. Based on the information in the instruction fields, the AXI Read command is generated. The Read requests are issued back to back without any Wait cycles.

Latency and data integrity modules are optional and you can enable them during IP generation. The data integrity module, if enabled, checks for the response data and reports errors to the register space when a data mismatch occurs. The TG also supports infinite transaction and infinite loop counts that allows the traffic generator to run indefinitely.

Synthesizable AXI4-Stream Architecture

The TG architecture for AXI4-Stream operation is shown in the following figure.

Figure 4: **Synthesizable TG Block Diagram for AXI4-Stream**



The TG is designed to support high bandwidth unidirectional data transfers. An instruction block RAM is present in the instruction core module of the TG with a depth of 512. This means up to 512 CSV instructions can be loaded in the block RAM and each CSV instruction can have up to 65,536 stream Write requests or a single Wait request. The Instruction Core module reads the block RAM instructions one by one, decodes them and provides the information to the Stream Channel Control module that pumps up the AXI4-Stream traffic. The AXI4-Stream Channel Control module receives control information from the Instruction Core module and generates stream traffic with multiple data pattern options. The packet length is programmable to up to 65,536 beats per request. Additionally, `TLAST` can be configured to be always one or always zero. The TG also supports infinite transaction and infinite loop counts, that allow the traffic generator to run indefinitely.

Product Specification

Port Descriptions

The Traffic Generator signals are listed and described in this section. The list of ports is the same for AXI3/AXI4 and AXI4-Stream.

Synthesizable TG

Table 1: **Synthesizable TG**

Signal	I/O	Port Width	Description
Default Signals			
clk	I	1	Traffic Generator clock (AXI clock)
pclk	I	1	Clock for instruction block RAM loading
tg_rst_n	I	1	Traffic Generator reset
axi_tg_start	I	1	Start signal indicating to start the traffic from TG
axi_tg_done	O	1	All instructions executed status
axi_tg_error	O	1	TG error status
trigger_in	I	1	Input trigger for synchronization
trigger_out	O	1	Output trigger for synchronization
nmu_wr_usr_dst	O	12	Write destination ID
nmu_rd_usr_dst	O	12	Read destination ID
VIO Interface Signals			
vio_aclk	I	1	VIO clock
vio_aresetn	I	1	VIO reset
s_axis_vio_tdata	I	32	The primary payload that is used to provide the data
s_axis_vio_tlast	I	1	Indicates the boundary of a packet
s_axis_vio_tready	O	1	Indicates that the slave can accept a transfer
s_axis_vio_tvalid	I	1	Indicates that the master is driving a valid transfer
MCS Interface Signals			
s_addr_strobe	I	1	Address strobe for address
s_address	I	30	Address
s_read_data	O	32	Read data

Table 1: **Synthesizable TG** *(cont'd)*

Signal	I/O	Port Width	Description
s_read_strobe	I	1	Read strobe for read data
s_ready	O	1	Ready signal
s_write_data	I	32	Write data
s_write_strobe	I	1	Write strobe for Write data
AXI Signals			
axi_aw*, axi_w*, axi_b*, axi_ar*, axi_r*,	I/O	Varies based on configuration	AXI3/AXI4 master interface signals. See AMBA AXI protocol specification for AXI3, AXI4
AXI4-Stream Signals			
axis_t*	I/O	Varies based on configuration	AXI4-Stream master interface signals. See AMBA AXI protocol specification for AXI4-Stream.

Data Integrity Modes

Data Integrity Check

The Synthesizable TG supports data integrity check. Data integrity refers to the checking of the read data, against the write data that was written to the slave. To use this feature, you must first write data to a particular address. If subsequent read of the address would give the same data then data integrity will pass, else it will fail.

The Read request sent out from the AXI master is stored in a FIFO. Whenever there is a response, the request information from the FIFO is retrieved and the expected data is generated. This data is compared against the received response for validation of data integrity. If data integrity fails, the error status (along with the read request address, transaction count, and beat count) is sent to the register set that can be used for further analysis. The Synthesizable TG supports AXI Incremental, Fixed, and Wrap bursts.

A maximum of eight errors can be stored in the registers. For certain bytes, data masking during the data integrity check is supported through the register interface. The Number of Data Integrity errors to stop traffic parameter in the Vivado IDE specifies the number of errors after which the traffic must be stopped. Values supported for this parameter are 0 to 256. Setting the value to 0 never stops the traffic. The data integrity check can be disabled through the register interface.

The Synthesizable TG supports two operating modes when data integrity checking is enabled; Constant ID and Incremental ID. Each TG instance can only support *one* of these operating modes when set during IP configuration. While it is possible to use Incremental ID mode with fixed AXI IDs, additional considerations must be taken in to account as described in Incremental ID (Out-of-Order) Traffic.

Constant ID (In-Order) Traffic

When the TG is configured for Constant ID mode, a fixed AXI ID is used for each traffic master in the design. The following additional conditions apply when operating in this mode:

- The traffic generator stops if the outstanding response count reaches 509.
- The `axi_id` field of the CSV file must be set to a fixed value and *not* `auto_incr`.
- Reading from unwritten locations causes the data integrity check to fail.
- Slave error or decode error responses from the AXI slave are also reported as data integrity failures.
- Although data integrity is enabled during IP generation, it can still be enabled or disabled for each instruction using the `data_integrity` field in the CSV.

Incremental ID (Out-of-Order) Traffic

In Incremental ID mode, TGs can use unique AXI IDs per transaction while managing data integrity checks. The following table demonstrates the combinations of AXI ID bus width, the number of unique AXI IDs that can be supported and the maximum number of outstanding responses per AXI ID in this operating mode. TGs can also use a fixed AXI ID but the maximum number of outstanding responses is significantly reduced when compared to Constant ID operating mode.

Table 2: **AXI ID Width and Outstanding Response Combinations**

AXI ID Width	Max. Number of IDs	Max. Outstanding Response per ID
4	16	8
3	8	16
2	4	32
1	2	64

The following additional conditions apply when operating in this mode:

- The maximum number of outstanding requests that are supported at any given time is 128.
- The traffic generated by the TG will stop if the outstanding response count per ID reaches (128/<Number of IDs>)-1.

- It is possible to use a fixed AXI ID but with reduced outstanding transaction counts when compared to Constant ID mode.
- Reading from unwritten locations causes the data integrity check to fail.

Delays Supported by the TG

AXI3/AXI4 and AXI4-Stream

- **Inter packet delay/start delay:** Represents the delay in terms of input CLK ticks between transactions. In AXI3/AXI4 this delay is called "start delay." In AXI4-Stream this delay is called "inter packet delay." Supported in the Synthesizable TG up to a 16-bit value in stream but not in AXI4.

Performance Counters

- **AXI4:**
  - **Request and Response Counters:** Request and response counters are used to calculate the bandwidth.

    Write transactions:

    - awreq_cntr: Number of Write requests issued by the TG
    - wrbeat_cntr: Number of Write beats issued by the TG
    - wlast_cntr: Number of wlast issued by the TG
    - bresp_cntr: Number of Write responses received by the TG
    - bresp_exokay_cntr: Number of Write responses received by the TG with EXOKAY response
    - bresp_slverr_cntr: Number of Write responses received by the TG with SLVERR response
    - bresp_decerr_cntr: Number of Write responses received by the TG with DECERR response
    - wbw_eff_clk_cntr : Number of cycles from the start of the TG to the wrch_done. Write Bandwidth = wrbeat_cntr/ wbw_eff_clk_cntr

    Read transactions:

    - arreq_cntr: Number of Read requests issued by the TG

- rbeat_cntr: Number of Read beats received by the TG
- rlast_cntr: Number of rlasts received by the TG
- rbw_eff_clk_cntr : Number of cycles from the start of the TG to the rdch_done. Read Bandwidth = rbeat_cntr / rbw_eff_clk_cntr

- **Latency:**

  As part of performance monitoring, latency calculation logic is also added in the TG separately for in-order (transactions with constant ID) traffic and out-of-order (transactions with incremental ID) traffic. You must select the kind of traffic (in-order or out-of-order) that is run to calculate the latency.

  For the Write channel, the latency is calculated from the start of the first request (awvalid during awready) to the reception of the first response (bvalid during bready).

  For the Read channel, latency is calculated from the start of the first request (arvalid during aready) to the reception of the first response (rvalid during rready).

  The best, worst, and average values for both Write and Read instructions are stored in the register set. For the average latency value, the TG sums up the latency values of all transactions.

- **AXI4-Stream:** Bandwidth = BEAT_COUNTER/ BANDWIDTH_COUNTER.

Synchronization of Multiple TGs

In a system with multiple TGs, it might be necessary to control a TG(s) based on the status of other TG(s). To enable this synchronization, the TG provides two signals:

- `trg_out`: This is a 1-bit output signal of the TG. Based on the instructions in the user_defined_sequence, a pulse can be generated on this output pin to indicate the status of the TG.
- `trg_in`: This is a 1-bit input signal to the TG. When the TG gets a level signal on the `trg_in` pin, it starts the next command as given by the user_defined_sequence.

Traffic Shaping

In a complex system, the `trg_in` signal to one or more TGs needs to be generated based on the status of the `trg_out` signal from multiple TGs. The Simulation Trigger module is designed for this purpose: it takes the `trg_out` from the TGs as input and generates `trg_in` to the TGs as the output. The Simulation Trigger module generates the output (connected to the `trg_in` of the TGs) when all the inputs (connected to the `trg_out` of the TGs) are received.

The following figures show the block diagrams for traffic shaping.

Figure 5: **Simulation Trigger Module with VIO Block Diagram**

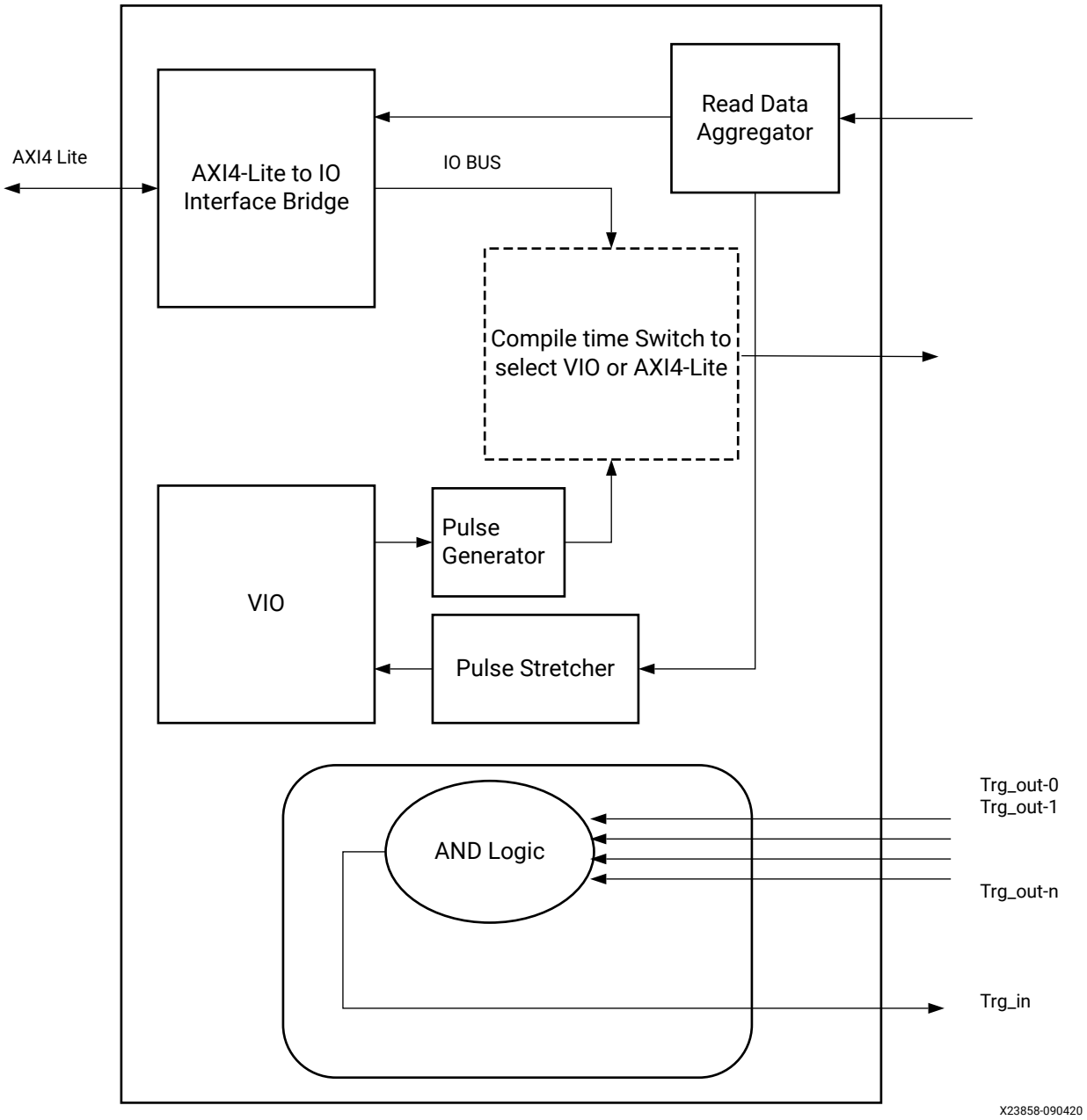
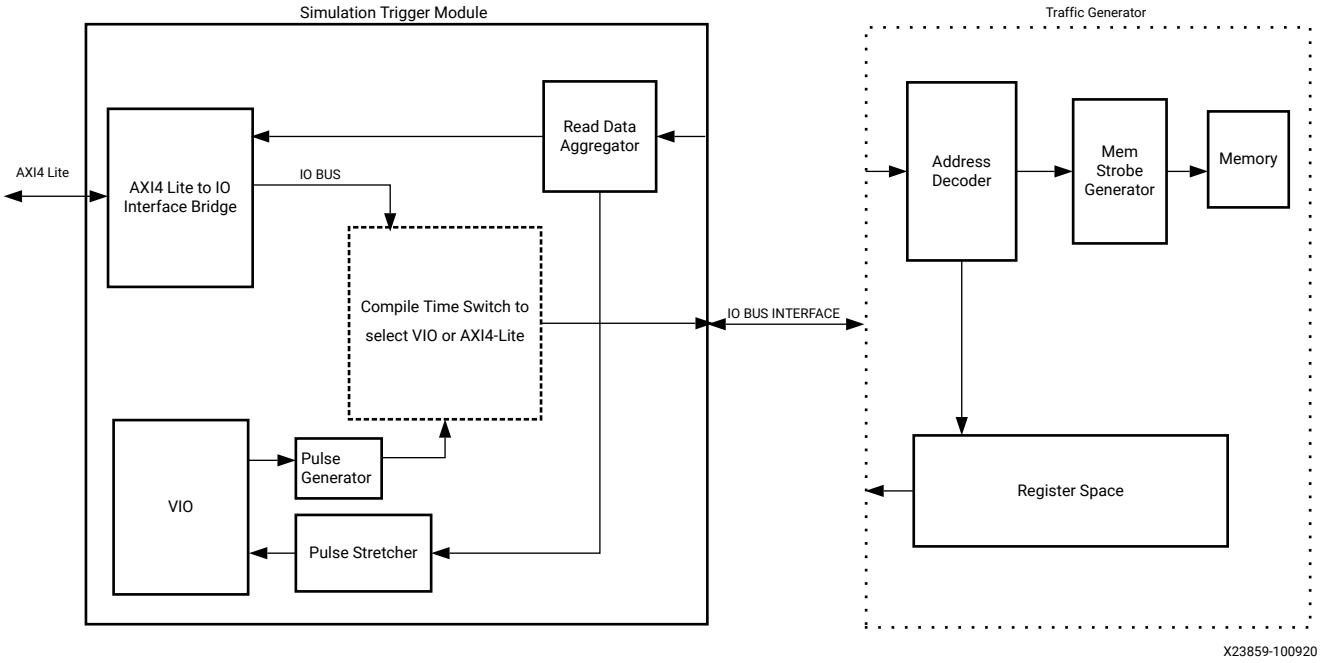


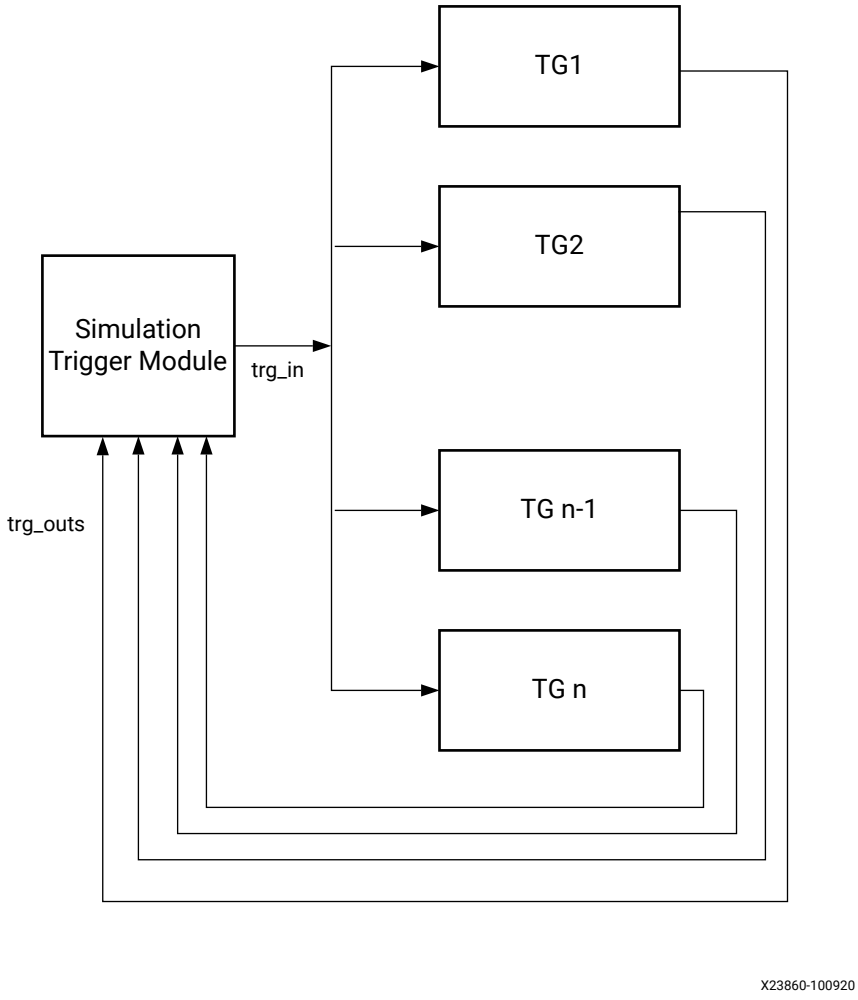
Figure 6: **Simulation Trigger Module with AXI4-Lite Block Diagram**



Simulation Trigger Module with Multiple TGs

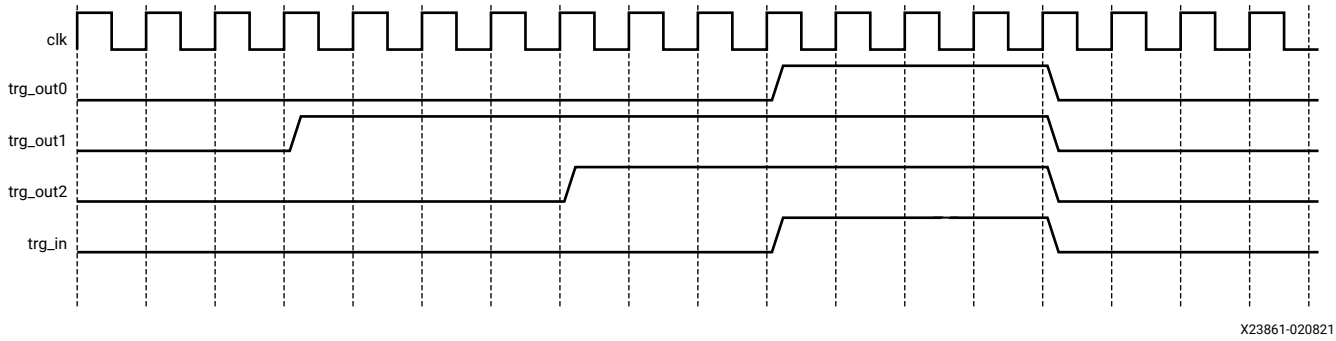
The following figure shows how multiple TGs interact with the Simulation Trigger module.

Figure 7: **Multiple TGs with Simulation Trigger Module**



The following timing diagram is an example where three TGs are instantiated. Each TG needs to execute three phases. For all TGs the phases should start at the same time but might end at different times. This requires synchronization between the TGs. The `trg_out` signals of all the TGs can be connected to the Simulation Trigger module and the `trg_in` signal can be connected to the output.

Figure 8: **Multiple TGs with trg_out and trg_in**



Hardware Access

You can load instructions and access registers through a VIO or AXI4-Lite interface. Selection between the interfaces is determined by configuration in the Vivado IDE.

AXI4-Lite Interface

The AXI4-Lite interface provides read/write access to the configuration and status registers in the Traffic Generator and also allows you to load the block RAM with instructions. All the transactions through the AXI4-Lite interface are of 32-bit data width. If you are loading an instruction into the block RAM with a width exceeding 32-bit, you must send multiple AXI4-Lite transactions, each of them 32-bit wide with the address bus value bits [5:2] incrementing each transaction starting from 4'b0000 up to 4'b1111, keeping all the other address bits constant till the complete instruction is loaded into a single memory location in the block RAM. See Running Custom Traffic for the Synthesizable TG for a programming example.

For example, if you consider the width of block RAM memory inside the TG as 416-bit and you want to load an instruction at block RAM address location 0x004 for Traffic Generator 2, the address format looks like the following table, with bits [5:2] incrementing for 4'b0000 – 4'b1011. In total 13 transactions are required to load a 416-bit wide instruction into a single block RAM location.

See User Guidelines for information on Data Integrity.

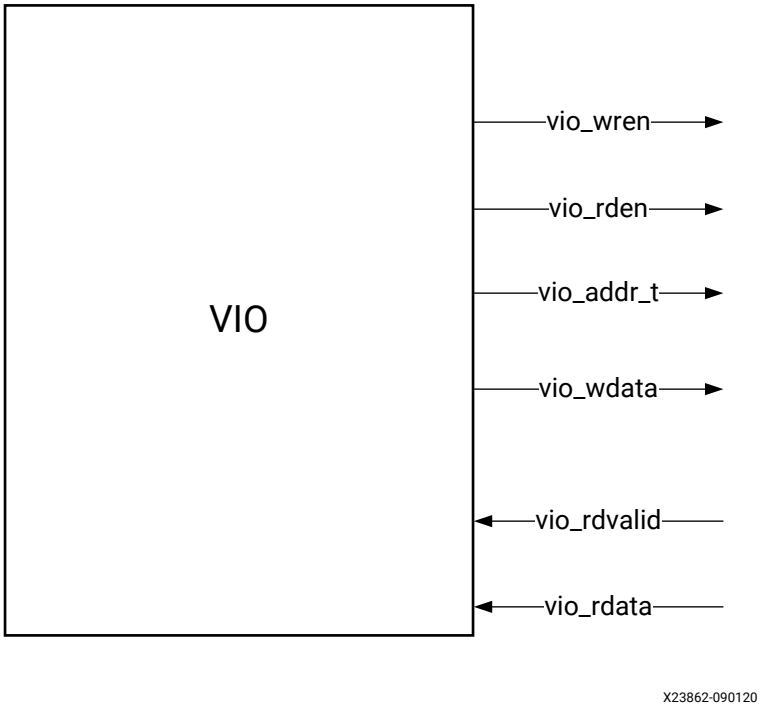
Table 3: **Example Address Format for AXI4-Lite**

[31:21]/ Reserved	[20:16]/TG Number	[15]/BRAM/ Register Space	[14:2]/BRAM Address/ Register Space Address		[1:0]
			[14:6]	[5:2]	
Reserved	5'b00001	1'b1	9'b000000100	4'b0000 – 4'b1011	2'b00

VIO Interface

The instruction block RAM must be loaded with instructions to run the traffic. These instructions can be loaded on the fly using virtual input/output (VIO). VIO is instantiated in the Simulation Trigger module. A Tcl script is used to load the instruction block RAM through VIO. The script takes the MEM file as input and generates Tcl commands that VIO can understand. Run this Tcl script in the Tcl console in the Vivado Hardware Manager to load the instruction block RAM or to read/write from the register space. A block diagram of the VIO interface is shown in the following figure.

Figure 9: **VIO Interface Block Diagram**



The VIO interface signals in the Simulation Trigger module are shown in the following table.

Table 4: **VIO Interface Signals**

Signal Name	Direction	Width	Description
vio_wren	O	1	Write Enable. If vio_wren is High it writes into the config register.
vio_rden	O	1	Read Enable. If vio_rden is High it reads from the config/ status register.
vio_addr_t	O	32	Register address.
vio_wdata	O	32	Write data.
vio_rdvalid	I	1	Read Valid. If vio_rdvalid is High the vio_rdata is valid.
vio_rdata	I	32	Valid Read data.

Register Space

The Performance AXI Traffic Generator has a set of registers to control its behavior, to provide status and debug information, and to control external signals.

These registers are classified into control/configuration registers and status registers. The control registers hold the control information required by the TG. This information is written through the register I/O interface (MCSIO_IN) on a slow clock (`PCLK`). The status registers hold the status information generated by the TG on a faster clock (`ACLK`) and this information can be read through the register I/O interface. The asynchronous clock domain crossing circuit ensures data integrity between the register space and the TG core. The TG registers can be accessed through the TG register interface (MCSIO_IN). By default this interface is hidden in the TG. Enable the **Traffic Reloading** option in the TG to gain access to the register interface.

The dedicated Simulation Trigger for the NoC AXI TG is a companion IP to the Performance AXI Traffic Generator and can be used to extend the register I/O interface to a typical user I/O interface such as VIO or AXI4-Lite. The Simulation Trigger IP converts the VIO/AXI4-Lite traffic into register interface traffic. The user I/O can be selected on the Traffic Reloading section of this IP. When you select one of the above options, the MCSIO_OUT register access interface appears in the Simulation Trigger IP. Connect it to the MCSIO_IN interface of the TG to send/receive register transactions.

A register space block diagram along with the interfaces to other blocks is shown in the following figure:

Figure 10: **Register Space Block Diagram**

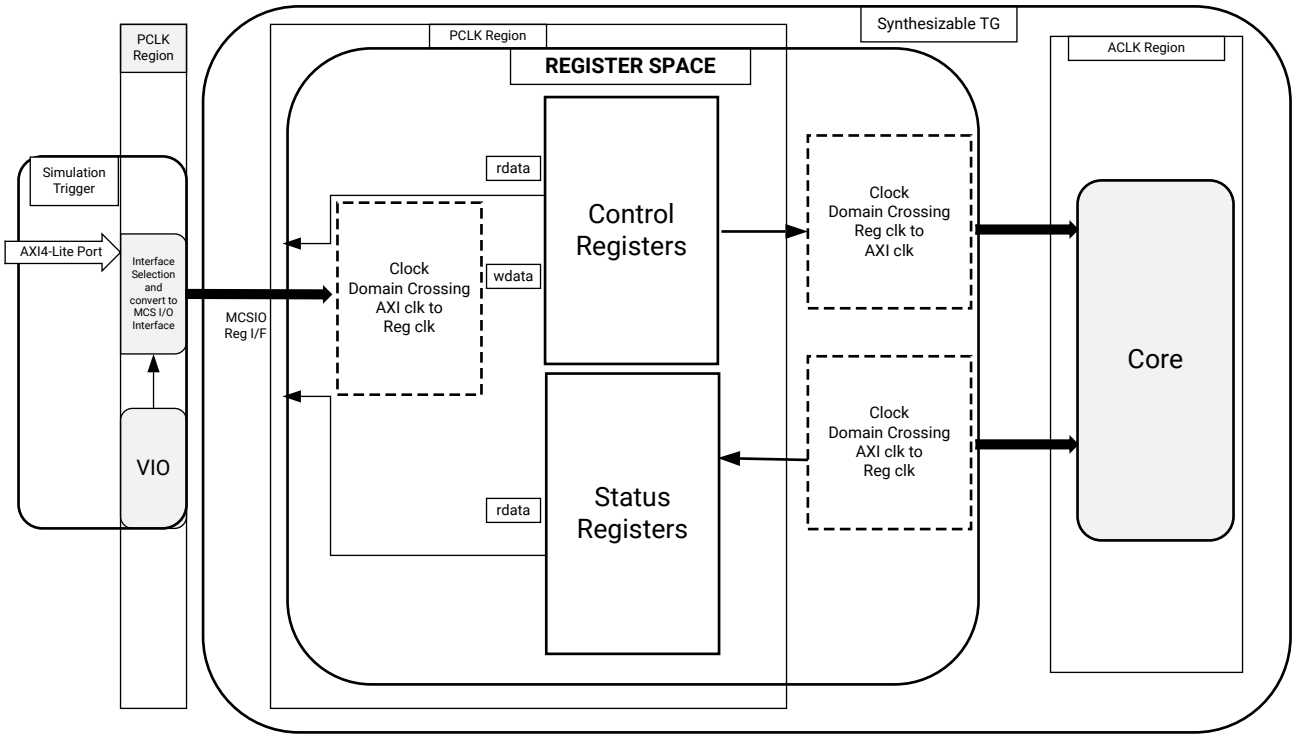


Table 5: **Register Address Space**

Address (hex)	Register
AXI3/AXI4 Configuration Registers	
0x4000	SOFT_RST
0x4004	TG_START
0x4010	OUTSTND_RESP_LIMIT_ADDR
0x41A8	DI_MASK_BYTES_1
0x41AC	DI_MASK_BYTES_2
0x41B4	AXI_RESP_CONFIG
0x42D0	CAPTURE_ENABLE
0x42D4	START_ADDR
0x42D8	END_ADDR
AXI3/AXI4 Status Registers	
0x2014	RBEAT_COUNTER
0x2018	ARREQ_COUNTER
0x201C	RLAST_CNTR
0x2020	RBW_EFF_CLK_CNTR
0x2024	RWORST_LATENCY
0x2028	RBEST_LATENCY

Table 5: **Register Address Space** *(cont'd)*

Address (hex)	Register
0x202C	RAVG_LATENCY
0x2030	AWREQ_CNTR
0x2034	WLAST_CNTR
0x2038	BRESP_CNTR
0x203C	WRBEAT_CNTR
0x2040	WBW_EFF_CLK_CNTR
0x2044	WWORST_LATENCY
0x2048	WBEST_LATENCY
0x204C	WAVG_LATENCY
0x2050	FLOW_EMPTY
0x2054	DI_ERR_COUNT
0x2058	DI_ERR_ADDR0_LSB_0
0x205C	DI_ERR_ADDR0_MSB16_0
0x2060	DI_ERR_TXN_INFO_0
0x2064	DI_ERR_ADDR0_LSB_1
0x2068	DI_ERR_ADDR0_MSB16_1
0x206C	DI_ERR_TXN_INFO_1
0x2070	DI_ERR_ADDR0_LSB_2
0x2074	DI_ERR_ADDR0_MSB16_2
0x2078	DI_ERR_TXN_INFO_2
0x207C	DI_ERR_ADDR0_LSB_3
0x2080	DI_ERR_ADDR0_MSB16_3
0x2084	DI_ERR_TXN_INFO_3
0x2088	DI_ERR_ADDR0_LSB_4
0x208C	DI_ERR_ADDR0_MSB16_4
0x2090	DI_ERR_TXN_INFO_4
0x2094	DI_ERR_ADDR0_LSB_5
0x2098	DI_ERR_ADDR0_MSB16_5
0x209C	DI_ERR_TXN_INFO_5
0x20A0	DI_ERR_ADDR0_LSB_6
0x20A4	DI_ERR_ADDR0_MSB16_6
0x20A8	DI_ERR_TXN_INFO_6
0x20AC	DI_ERR_ADDR0_LSB_7
0x20B0	DI_ERR_ADDR0_MSB16_7
0x20B4	DI_ERR_TXN_INFO_7
0x20B8	DI_ERR_ADDR0_LSB_8
0x20BC	DI_ERR_ADDR0_MSB16_8
0x20C0	DI_ERR_TXN_INFO_8
0x20C4	DI_ERR_ADDR0_LSB_9

Table 5: **Register Address Space** *(cont'd)*

Address (hex)	Register
0x20C8	DI_ERR_ADDR0_MSB16_9
0x20CC	DI_ERR_TXN_INFO_9
0x20D0	BRESP_EXOKAY_CNTR
0x20D4	BRESP_SLVERR_CNTR
0x20D8	BRESP_DECERR_CNTR
0x20DC	RCVD_RDATA_ACT_0T31
0x20E0	RCVD_RDATA_ACT_32T63
0x20E4	RCVD_RDATA_ACT_64T95
0x20E8	RCVD_RDATA_ACT_96T127
0x20EC	RCVD_RDATA_ACT_128T159
0x20F0	RCVD_RDATA_ACT_160T191
0x20F4	RCVD_RDATA_ACT_192T223
0x20F8	RCVD_RDATA_ACT_255T224
0x20FC	RCVD_RDATA_ACT_287T256
0x2100	RCVD_RDATA_ACT_319T288
0x2104	RCVD_RDATA_ACT_351T320
0x2108	RCVD_RDATA_ACT_383T352
0x210C	RCVD_RDATA_ACT_ 015T384
0x2110	RCVD_RDATA_ACT_447T416
0x2114	RCVD_RDATA_ACT_479T448
0x2118	RCVD_RDATA_ACT_511T480
0x211C	RCVD_RDATA_EXPD_0T31
0x2120	RCVD_RDATA_EXPD_32T63
0x2124	RCVD_RDATA_EXPD_64T95
0x2128	RCVD_RDATA_EXPD_96T127
0x212C	RCVD_RDATA_EXPD_128T159
0x2130	RCVD_RDATA_EXPD_160T191
0x2134	RCVD_RDATA_EXPD_192T223
0x2138	RCVD_RDATA_EXPD_255T224
0x213C	RCVD_RDATA_EXPD_287T256
0x2140	RCVD_RDATA_EXPD_319T288
0x2144	RCVD_RDATA_EXPD_351T320
0x2148	RCVD_RDATA_EXPD_383T352
0x214C	RCVD_RDATA_EXPD_415T384
0x2150	RCVD_RDATA_EXPD_447T416
0x2154	RCVD_RDATA_EXPD_479T448
0x2158	RCVD_RDATA_EXPD_511T480
0x215C	LAST_RDATA_RCVD_0T31
0x2160	LAST_RDATA_RCVD_32T63

Table 5: **Register Address Space** *(cont'd)*

Address (hex)	Register
0x2164	LAST_RDATA_RCVD_64T95
0x2168	LAST_RDATA_RCVD_96T127
0x216C	LAST_RDATA_RCVD_128T159
0x2170	LAST_RDATA_RCVD_160T191
0x2174	LAST_RDATA_RCVD_192T223
0x2178	LAST_RDATA_RCVD_255T224
0x217C	LAST_RDATA_RCVD_287T256
0x2180	LAST_RDATA_RCVD_319T288
0x2184	LAST_RDATA_RCVD_351T320
0x2188	LAST_RDATA_RCVD_383T352
0x218C	LAST_RDATA_RCVD_415T384
0x2190	LAST_RDATA_RCVD_447T416
0x2194	LAST_RDATA_RCVD_479T448
0x2198	LAST_RDATA_RCVD_511T480
0x219C	RRESP_EXOKAY_CNTR
0x21A0	RRESP_SLVERR_CNTR
0x21A4	RRESP_DECERR_CNTR
0x21B0	AXI_VLD_RDY_STATS
0x21B8	BRESP_ERR_COUNT
0x21BC	BRESP_ERR_ADDR0_LSB_0
0x21C0	BRESP_ERR_ADDR0_MSB16_0
0x21C4	BRESP_ERR_TXN_INFO_0
0x21C8	BRESP_ERR_ADDR0_LSB_1
0x21CC	BRESP_ERR_ADDR0_MSB16_1
0x21D0	BRESP_ERR_TXN_INFO_1
0x21D4	BRESP_ERR_ADDR0_LSB_2
0x21D8	BRESP_ERR_ADDR0_MSB16_2
0x21DC	BRESP_ERR_TXN_INFO_2
0x21E0	BRESP_ERR_ADDR0_LSB_3
0x21E4	BRESP_ERR_ADDR0_MSB16_3
0x21E8	BRESP_ERR_TXN_INFO_3
0x21EC	BRESP_ERR_ADDR0_LSB_4
0x21F0	BRESP_ERR_ADDR0_MSB16_4
0x21F4	BRESP_ERR_TXN_INFO_4
0x21F8	BRESP_ERR_ADDR0_LSB_5
0x21FC	BRESP_ERR_ADDR0_MSB16_5
0x2200	BRESP_ERR_TXN_INFO_5
0x2204	BRESP_ERR_ADDR0_LSB_6
0x2208	BRESP_ERR_ADDR0_MSB16_6

Table 5: **Register Address Space** *(cont'd)*

Address (hex)	Register
0x220C	BRESP_ERR_TXN_INFO_6
0x2210	BRESP_ERR_ADDR0_LSB_7
0x2214	BRESP_ERR_ADDR0_MSB16_7
0x2218	BRESP_ERR_TXN_INFO_7
0x221C	BRESP_ERR_ADDR0_LSB_8
0x2220	BRESP_ERR_ADDR0_MSB16_8
0x2224	BRESP_ERR_TXN_INFO_8
0x2228	BRESP_ERR_ADDR0_LSB_9
0x222C	BRESP_ERR_ADDR0_MSB16_9
0x2230	BRESP_ERR_TXN_INFO_9
0x2234	CAPTURE_ALL
0x2238	RBEAT_COUNTER_AL
0x223C	ARREQ_COUNTER_AL
0x2240	RLAST_CNTR_AL
0x2244	RBW_EFF_CLK_CNTR_AL
0x2248	RWORST_LATENCY_AL
0x224C	RBEST_LATENCY_AL
0x2250	RAVG_LATENCY_AL
0x2254	AWREQ_CNTR_AL
0x2258	WLAST_CNTR_AL
0x225C	BRESP_CNTR_AL
0x2260	WRBEAT_CNTR_AL
0x2264	WBW_EFF_CLK_CNTR_AL
0x2268	WWORST_LATENCY_AL
0x226C	WBEST_LATENCY_AL
0x2270	WAVG_LATENCY_AL
0x2274	FLOW_EMPTY_AL
0x2278	BRESP_EXOKAY_CNTR_AL
0x227c	BRESP_SLVERR_CNTR_AL
0x2280	BRESP_DECERR_CNTR_AL
0x2284	RRESP_EXOKAY_CNTR_AL
0x2288	RRESP_SLVERR_CNTR_AL
0x228C	RRESP_DECERR_CNTR_AL
0x2290	LAST_RDATA_RCVD_0T31_AL
0x2294	LAST_RDATA_RCVD_32T63_AL
0x2298	LAST_RDATA_RCVD_64T95_AL
0x229C	LAST_RDATA_RCVD_96T127_AL
0x22A0	LAST_RDATA_RCVD_128T159_AL
0x22A4	LAST_RDATA_RCVD_160T191_AL

Table 5: **Register Address Space** *(cont'd)*

Address (hex)	Register
0x22A8	LAST_RDATA_RCVD_192T223_AL
0x22AC	LAST_RDATA_RCVD_255T224_AL
0x22B0	LAST_RDATA_RCVD_287T256_AL
0x22B4	LAST_RDATA_RCVD_319T288_AL
0x22B8	LAST_RDATA_RCVD_351T320_AL
0x22BC	LAST_RDATA_RCVD_383T352_AL
0x22C0	LAST_RDATA_RCVD_415T384_AL
0x22C4	LAST_RDATA_RCVD_447T416_AL
0x22C8	LAST_RDATA_RCVD_479T448_AL
0x22CC	LAST_RDATA_RCVD_511T480_AL
0x22DC	CAPTR_WRBEAT
0x22E0	CAPTR_RDBEAT
AXI4-Stream Configuration Registers	
0X4004	START
0X4008	RESET
0x401C	PLAY_PAUSE
0x4020	STROBE LSB BITS
0x4024	STROBE MSB BITS
0x4028	KEEP LSB BITS
0x402C	KEEP MSB BITS
AXI4-Stream Status Registers	
0X200C	BANDWIDTH_COUNTER
0X2010	REQUEST_COUNTER
0x2014	TXN_COMPLETE
0x2018	BEAT_COUNTER

Address Decoding for Synthesizable TG

Table 6: **Address Decoding**

[31:21]	[20:16]	[15]	[14:2]	[1:0]
Reserved	TG Number	1'b0 for Register Space; 1'b1 for block RAM Space	block RAM/Register Space Address	2’b00

Register Space Addressing: 0x0000 to 0x7FFC

Table 7: **Register Space Addressing**

[31:21]	[20:16]	[15]	[14:2]	[1:0]
Reserved	TG number	1'b0	Register Address	2'b00

Block RAM Addressing: 0x8000 to 0xFFFC

Table 8: **BRAM Space Addressing**

[31:21]	[20:16]	[15]	[14:6]	[5:2]	[1:0]
Reserved	TG number	1'b1	Address per CSV instruction	Offset to load 32 bits of each block RAM instruction	2'b00

AXI3/AXI4 Configuration Registers

SOFT_RST Register (0x4000)

Table 9: **SOFT_RST Register (0x4000)**

Bit	Default Value	Access Type	Description
31:1	0	R/W	Reserved
0	1	R/W	Active-Low reset used to reset the TG logic

TG_START Register (0x4004)

Table 10: **TG_START Register (0x4004)**

Bit	Default Value	Access Type	Description
31:1	0	R/W	Reserved
0	1	R/W	When set, enables the TG to send traffic. This is a self-clearing register.

OUTSTND_RESP_LIMIT Register (0x4010)

Table 11: **OUTSTND_RESP_LIMIT Register (0x4010)**

Bit	Default Value	Access Type	Description
31:12	0x0000	R/W	Reserved
11:0	0x0000	R/W	Outstanding limit from the TG

DI_MASK_BYTES_1 Register (0x41A8)

Table 12: **DI_MASK_BYTES_1 Register (0x41A8)**

Bit	Default Value	Access Type	Description
31:0	0x00000000	R/W	Adjusted as per the strobe width. 32 LSBs of data mask.

DI_MASK_BYTES_2 Register (0x41AC)

Table 13: **DI_MASK_BYTES_2 Register (0x41AC)**

Bit	Default Value	Access Type	Description
31:0	0x00000000	R/W	Adjusted as per the strobe width. 32 MSBs of data mask. For 512-bit, the mask bytes are {DI_MASK_BYTES_2, DI_MASK_BYTES_1}

CAPTURE_ENABLE (0x42D0)

Table 14: **CAPTURE_ENABLE (0x42D0)**

Bit	Default Value	Access Type	Description
31:1	0x0000	R/W	Reserved
0	0	R/W	To enable capturing of read and write beats for calculation of average bandwidth using windowing technique. 0: Capture disabled 1: Capture enabled

START_ADDR (0x42D4)

Table 15: **START_ADDR (0x42D4)**

Bit	Default Value	Access Type	Description
31:0	0x0000_2710	R/W	It shows the starting AXI Clock cycle number for the window. It's set to some non-zero value to allow the system to settle and then start the window for capturing write/read beats to compute average bandwidth. For eg : To start the windowing from 10,000 axi clock cycles set START_ADDR as 0x 0000_2710

END_ADDR (0x42D8)

Table 16: **END_ADDR(0x42D8)**

Bit	Default Value	Access Type	Description
31:0	0x0001_ADB0	R/W	It shows the ending AXI Clock cycle number for the window. The total no. of write/ read beats captured during this period is stored to calculate average bandwidth. The size of window for calculating average is Window = END_ADDR - START_ADDR For eg : To end the windowing at 1,10,000 axi clock cycles set END_ADDR as 0x 0001_ADB0. If START_ADDR would have been 10,000 clock cycles then size of window = 1,00,000.

AXI3/AXI4 Status Registers

***Note*:** The address column denotes the address for TG0.

RBEAT_COUNTER Register (0x2014)

Table 17: **RBEAT_COUNTER Register (0x2014)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	Number of Read beats received by the TG

ARREQ_COUNTER Register (0x2018)

Table 18: **ARREQ_COUNTER Register (0x2018)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	Number of Read requests issued by the TG

RLAST_CNTR Register (0x201C)

Table 19: **RLAST_CNTR Register (0x201C)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	Number of rlast received by the TG

RBW_EFF_CLK_CNTR Register (0x2020)

Table 20: **RBW_EFF_CLK_CNTR Register (0x2020)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	Read bandwidth counter. Number of cycles from the start of the TG to the rdch_done. This is used to calculate the bandwidth. Read bandwidth = RBEAT_COUNTER/ RBW_EFF_CLK_CNTR

RWORST_LATENCY Register (0x2014)

Table 21: **RWORST_LATENCY Register (0x2014)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	Maximum latency of Read traffic

RBEST_LATENCY Register (0x2028)

Table 22: **RBEST_LATENCY Register (0x2028)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	Minimum latency of Read traffic

RAVG_LATENCY Register (0x202C)

Table 23: **RAVG_LATENCY Register (0x202C)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	Total latency of the Read traffic. This is used to calculate the average latency. Average latency = RAVG_LATENCY/ ARREQ_COUNTER

AWREQ_CNTR Register (0x2030)

Table 24: **AWREQ_CNTR Register (0x2030)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	Number of Write requests issued by the TG

WLAST_CNTR Register (0x2034)

Table 25: **WLAST_CNTR Register (0x2034)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	Number of wlast issued by the TG

BRESP_CNTR Register (0x2038)

Table 26: **BRESP_CNTR Register (0x2038)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	Number of responses received by the TG

WRBEAT_CNTR Register (0x203C)

Table 27: **WRBEAT_CNTR Register (0x203C)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	Number of Write beats issued by the TG

WBW_EFF_CLK_CNTR Register (0x2040)

Table 28: **WBW_EFF_CLK_CNTR Register (0x2040)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	Write bandwidth counter. Number of cycles from the start of the TG to the wrch_done signal (a Write Traffic Done signal mapped to Bit 8 of the FLOW_EMPTY register). This is used to calculate the bandwidth. Write bandwidth = WRBEAT_CNTR/ WBW_EFF_CLK_CNTR

WWORST_LATENCY Register (0x2044)

Table 29: **WWORST_LATENCY Register (0x2044)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	Maximum latency of Write traffic

WBEST_LATENCY Resgister (0x2048)

Table 30: **WBEST_LATENCY Resgister (0x2048)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	Minimum latency of Write traffic

WAVG_LATENCY Register (0x204C)

Table 31: **WAVG_LATENCY Register (0x204C)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	Total latency of the Read traffic. This is used to calculate the average latency. Average latency =WAVG _LATENCY/ AWREQ _COUNTER

FLOW_EMPTY Register (0x2050)

Table 32: **FLOW_EMPTY Register (0x2050)**

Bit	Default Value	Access Type	Description
31:13	0	RO	Reserved
12	0	RO	This bit is set when instruction block RAM is empty.
11	0	RO	This bit is set when Write channel instruction FIFO is empty.
10	0	RO	This bit is set when Read channel instruction FIFO is empty.
9	0	RO	If set, then the TG has completed sending both Read and Write traffic.
8	0	RO	Write traffic done. If set, then the TG has completed sending the Write traffic.
7	0	RO	Read traffic done. If set, then the TG has completed sending the Read traffic.
6	0	RO	This bit is set when Write channel FIFO is full.
5	0	RO	This bit is set when Read channel FIFO is full.
4	0	RO	This bit is set when a TG error occurs. Two things can cause the TG to update the error status bit to 1; these are Data Integrity Error and/or AXI Response Check Error.
3	0	RO	Bit set High when Read request FIFO overflows.
2	0	RO	Bit set High when Read request FIFO underflows.
1	0	RO	Bit set High when Write request FIFO overflows.
0	0	RO	Bit set High when Write request FIFO underflows.

DI_ERR_COUNT Register (0x2054)

Table 33: **DI_ERR_COUNT Register (0x2054)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	Indicates the number of data integrity (DI) errors flagged.

DI_ERR_ADDR0_LSB_0 Register (0x2058)

Table 34: **DI_ERR_ADDR0_LSB_0 Register (0x2058)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	These are the 32 LSBs of the 48-bit address at which the first DI error occurred.

DI_ERR_ADDR0_MSB16_0 Register (0x205C)

Table 35: **DI_ERR_ADDR0_MSB16_0 Register (0x205C)**

Bit	Default Value	Access Type	Description
31:16	0x0000	RO	Reserved
15:0	0x0000	RO	These are the 16 MSBs out of the 48-bit address at which the first DI error occurred.

DI_ERR_TXN_INFO_0 Register (0x2060)

Table 36: **DI_ERR_TXN_INFO_0 Register (0x2060)**

Bit	Default Value	Access Type	Description
31:30	0x0	RO	Reserved
29:28	0x0	RO	Error type information: 00 -> rresp err 01 -> data error 10 -> reserved 11 -> reserved
27:26	0x0	RO	Read response expected
25:24	0x0	RO	Read response received
23:16	0x00	RO	[23:16] Beat count at which the first DI error is flagged.
15:0	0x0000	RO	[15:0] Transaction count at which the first DI error is flagged.

DI_ERR_ADDR0_LSB_1 Register (0x2064)

Table 37: **DI_ERR_ADDR0_LSB_1 Register (0x2064)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	These are the 32 LSBs of the 48-bit address at which the second DI error occurred.

DI_ERR_ADDR0_MSB16_1 Register (0x2068)

Table 38: **DI_ERR_ADDR0_MSB16_1 Register (0x2068)**

Bit	Default Value	Access Type	Description
31:16	0x0000	RO	Reserved
15:0	0x0000	RO	These are the 16 MSBs out of the 48-bit address at which the second DI error occurred.

DI_ERR_TXN_INFO_1 Register (0x206C)

Table 39: **DI_ERR_TXN_INFO_1 Register (0x206C)**

Bit	Default Value	Access Type	Description
31:30	0x0	RO	Reserved
29:28	0x0	RO	Error type information: 00-> rresp err 01 -> data error 10 -> reserved 11 -> reserved
27:26	0x0	RO	Read response expected
25:24	0x0	RO	Read response received
23:16	0x00	RO	[23:16] Beat count at which the second DI error is flagged.
15:0	0x0000	RO	[15:0] Transaction count at which the second DI error is flagged.

DI_ERR_ADDR0_LSB_2 Register (0x2070)

Table 40: **DI_ERR_ADDR0_LSB_2 Register (0x2070)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	These are the 32 LSBs of the 48-bit address at which the third DI error occurred.

DI_ERR_ADDR0_MSB16_2 Register (0x2074)

Table 41: **DI_ERR_ADDR0_MSB16_2 Register (0x2074)**

Bit	Default Value	Access Type	Description
31:16	0x0000	RO	Reserved
15:0	0x0000	RO	These are the 16 MSBs out of the 48-bit address at which the third DI error occurred.

DI_ERR_TXN_INFO_2 Register (0x2078)

Table 42: **DI_ERR_TXN_INFO_2 Register (0x2078)**

Bit	Default Value	Access Type	Description
31:30	0x0	RO	Reserved
29:28	0x0	RO	Error type information: 00 -> rresp err 01 -> data error 10 -> reserved 11 -> reserved
27:26	0x0	RO	Read response expected
25:24	0x0	RO	Read response received
23:16	0x00	RO	[23:16] Beat count at which the third DI error is flagged.
15:0	0x0000	RO	[15:0] Transaction count at which the third DI error is flagged.

DI_ERR_ADDR0_LSB_3 Register (0x207C)

Table 43: **DI_ERR_ADDR0_LSB_3 Register (0x207C)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	These are the 32 LSBs of the 48-bit address at which the fourth DI error occurred.

DI_ERR_ADDR0_MSB16_3 Register (0x2080)

Table 44: **DI_ERR_ADDR0_MSB16_3 Register (0x2080)**

Bit	Default Value	Access Type	Description
31:16	0x0000	RO	Reserved
15:0	0x0000	RO	These are the 16 MSBs out of the 48-bit address at which the fourth DI error occurred.

DI_ERR_TXN_INFO_3 Register (0x2084)

Table 45: **DI_ERR_TXN_INFO_3 Register (0x2084)**

Bit	Default Value	Access Type	Description
31:30	0x0	RO	Reserved
29:28	0x0	RO	Error type information: 00 -> rresp err 01 -> data error 10 -> reserved 11 -> reserved
27:26	0x0	RO	Read response expected
25:24	0x0	RO	Read response received
23:16	0x00	RO	[23:16] Beat count at which the fourth DI error is flagged
15:0	0x0000	RO	[15:0] Transaction count at which the fourth DI error is flagged

DI_ERR_ADDR0_LSB_4 Register (0x2088)

Table 46: **DI_ERR_ADDR0_LSB_4 Register (0x2088)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	These are the 32 LSBs of the 48-bit address at which the fifth DI error occurred.

DI_ERR_ADDR0_MSB16_4 Register (0x208C)

Table 47: **DI_ERR_ADDR0_MSB16_4 Register (0x 208C)**

Bit	Default Value	Access Type	Description
31:16	0x0000	RO	Reserved
15:0	0x0000	RO	These are the 16 MSBs out of the 48-bit address at which the fifth DI error occurred.

DI_ERR_TXN_INFO_4 Register (0x2090)

Table 48: **DI_ERR_TXN_INFO_4 Register (0x2090)**

Bit	Default Value	Access Type	Description
31:30	0x0	RO	Reserved

Table 48: **DI_ERR_TXN_INFO_4 Register (0x2090)** *(cont'd)*

Bit	Default Value	Access Type	Description
29:28	0x0	RO	Error type information: 00 -> rresp err 01 -> data error 10 -> reserved 11 -> reserved
27:26	0x0	RO	Read response expected
25:24	0x0	RO	Read response received
23:16	0x0000	RO	[23:16] Beat count at which the fifth DI error flagged
15:0	0x0000	RO	[15:0] Transaction count at which the fifth DI error flagged

DI_ERR_ADDR0_LSB_5 Register (0x2094)

Table 49: **DI_ERR_ADDR0_LSB_5 Register (0x2094)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	These are the 32 LSBs of the 48-bit address at which the sixth DI error occurred.

DI_ERR_ADDR0_MSB16_5 Register (0x2098)

Table 50: **DI_ERR_ADDR0_MSB16_5 Register (0x2098)**

Bit	Default Value	Access Type	Description
31:16	0x0000	RO	Reserved
15:0	0x0000	RO	These are the 16 MSBs out of the 48-bit address at which the sixth DI error occurred.

DI_ERR_TXN_INFO_5 Register (0x209C)

Table 51: **DI_ERR_TXN_INFO_5 Register (0x209C)**

Bit	Default Value	Access Type	Description
31:24	0x00	RO	Reserved
23:16	0x0000	RO	[23:16] Beat count at which the sixth DI error is flagged
15:0	0x0000	RO	[15:0] Transaction count at which the sixth DI error is flagged

DI_ERR_ADDR0_LSB_6 Register (0x20A0)

Table 52: **DI_ERR_ADDR0_LSB_6 Register (0x20A0)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	These are the 32 LSBs of the 48-bit address at which the seventh DI error occurred.

DI_ERR_ADDR0_MSB16_6 Register (0x20A4)

Table 53: **DI_ERR_ADDR0_MSB16_6 Register (0x20A4)**

Bit	Default Value	Access Type	Description
31:16	0x0000	RO	Reserved
15:0	0x0000	RO	These are the 16 MSBs out of the 48-bit address at which the seventh DI error occurred.

DI_ERR_TXN_INFO_6 Register (0x20A8)

Table 54: **DI_ERR_TXN_INFO_6 Register (0x20A8)**

Bit	Default Value	Access Type	Description
31:30	0x0	RO	Reserved
29:28	0x0	RO	Error type information: 00 -> rresp err 01 -> data error 10 -> reserved 11 -> reserved
27:26	0x0	RO	Read response expected
25:24	0x0	RO	Read response received
23:16	0x0000	RO	[23:16] Beat count at which the seventh DI error flagged
15:0	0x0000	RO	[15:0] Transaction count at which the seventh DI error flagged

DI_ERR_ADDR0_LSB_7 Register (0x20AC)

Table 55: **DI_ERR_ADDR0_LSB_7 Register (0x20AC)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	These are the 32 LSBs of the 48-bit address at which the eighth DI error occurred.

DI_ERR_ADDR0_MSB16_7 Register (0x20B0)

Table 56: **I_ERR_ADDR0_MSB16_7 Register (0x20B0)**

Bit	Default Value	Access Type	Description
31:16	0x0000	RO	Reserved
15:0	0x0000	RO	These are the 16 MSBs out of the 48-bit address at which the eighth DI error occurred.

DI_ERR_TXN_INFO_7 Register (0x20B4)

Table 57: **DI_ERR_TXN_INFO_7 Register (0x20B4)**

Bit	Default Value	Access Type	Description
31:30	0x0	RO	Reserved
29:28	0x0	RO	Error type information: 00 -> rresp err 01 -> data error 10 -> reserved 11 -> reserved
27:26	0x0	RO	Read response expected
25:24	0x0	RO	Read response received
23:16	0x0000	RO	Beat count at which the eighth DI error is flagged
15:0	0x0000	RO	Transaction count at which the eighth DI error is flagged

DI_ERR_ADDR0_LSB_8 Register (0x20B8)

Table 58: **DI_ERR_ADDR0_LSB_8 Register (0x20B8)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	These are the 32 LSBs of the 48-bit address at which the ninth DI error occurred.

DI_ERR_ADDR0_MSB16_8 Register (0x20BC)

Table 59: **DI_ERR_ADDR0_MSB16_8 Register (0x20BC)**

Bit	Default Value	Access Type	Description
31:16	0x0000	RO	Reserved
15:0	0x0000	RO	These are the 16 MSBs out of the 48-bit address at which the ninth DI error occurred.

DI_ERR_TXN_INFO_8 Register (0x20C0)

Table 60: **DI_ERR_TXN_INFO_8 Register (0x20C0)**

Bit	Default Value	Access Type	Description
31:30	0x0	RO	Reserved
29:28	0x0	RO	Error type information: 00-> rresp err 01 -> data error 10 -> reserved 11 -> reserved
27:26	0x0	RO	Read response expected
25:24	0x0	RO	Read response received
23:16	0x0000	RO	Beat count at which the ninth DI error flagged
15:0	0x0000	RO	Transaction count at which the ninth DI error flagged

DI_ERR_ADDR0_LSB_9 Register (0x20C4)

Table 61: **DI_ERR_ADDR0_LSB_9 Register (0x20C4)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	These are the 32 LSBs of the 48-bit address at which the tenth DI error occurred.

DI_ERR_ADDR0_MSB16_9 Register (0x20C8)

Table 62: **DI_ERR_ADDR0_MSB16_9 Register (0x20C8)**

Bit	Default Value	Access Type	Description
31:16	0x0000	RO	Reserved
15:0	0x0000	RO	These are the 16 MSBs out of the 48-bit address at which the tenth DI error occurred.

DI_ERR_TXN_INFO_9 Register (0x20CC)

Table 63: **DI_ERR_TXN_INFO_9 Register (0x20CC)**

Bit	Default Value	Access Type	Description
31:30	0x0	RO	Reserved

Table 63: **DI_ERR_TXN_INFO_9 Register (0x20CC)** *(cont'd)*

Bit	Default Value	Access Type	Description
29:28	0x0	RO	Error type information: 00-> rresp err 01 -> data error 10 -> reserved 11 -> reserved
27:26	0x0	RO	Read response expected
25:24	0x0	RO	Read response received
23:16	0x0000	RO	[23:16] Beat count at which the tenth DI error is flagged
15:0	0x0000	RO	[15:0] Transaction count at which the tenth DI error is flagged

BRESP_EXOKAY_CNTR Register (0x20D0)

Table 64: **BRESP_EXOKAY_CNTR Register (0x20D0)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	Number of Write responses received as EXOKAY.

BRESP_SLVERR_CNTR Register (0x20D4)

Table 65: **RESP_SLVERR_CNTR Register (0x20D4)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	Number of Write responses received as SLVERR.

BRESP_DECERR_CNTR Register (0x20D8)

Table 66: **BRESP_DECERR_CNTR Register (0x20D8)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	Number of Write responses received as DECERR.

RCVD_RDATA_ACT_0T31 Register (0x20DC)

Table 67: **RCVD_RDATA_ACT_0T31 Register (0x20DC)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	[31:0] bits of first actual Read data received when DI error occurs.

RCVD_RDATA_ACT_32T63 Register (0x20E0)

Table 68: **RCVD_RDATA_ACT_32T63 Register (0x20E0)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	[63:32] bits of first actual Read data received when DI error occurs.

RCVD_RDATA_ACT_64T95 Register (0x20E4)

Table 69: **RCVD_RDATA_ACT_64T95 Register (0x20E4)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	[95:64] bits of first actual Read data received when DI error occurs.

RCVD_RDATA_ACT_96T127 Register (0x20E8)

Table 70: **RCVD_RDATA_ACT_96T127 Register (0x20E8)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	[127:96] bits of first actual Read data received when DI error occurs.

RCVD_RDATA_ACT_128T159 Register (0x20EC)

Table 71: **RCVD_RDATA_ACT_128T159 Register (0x20EC)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	[159:128] bits of first actual Read data received when DI error occurs.

RCVD_RDATA_ACT_160T191 Register (0x20F0)

Table 72: **RCVD_RDATA_ACT_160T191 Register (0x20F0)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	[191:160] bits of first actual Read data received when DI error occurs.

RCVD_RDATA_ACT_192T223 Register (0x20F4)

Table 73: **RCVD_RDATA_ACT_192T223 Register (0x20F4)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	[223:192] bits of first actual Read data received when DI error occurs.

RCVD_RDATA_ACT_255T224 Register (0x20F8)

Table 74: **RCVD_RDATA_ACT_255T224 Register (0x20F8)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	[255:224] bits of first actual Read data received when DI error occurs.

RCVD_RDATA_ACT_287T256 Register (0x20FC)

Table 75: **RCVD_RDATA_ACT_287T256 Register (0x20FC)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	[287:256] bits of first actual Read data received when DI error occurs.

RCVD_RDATA_ACT_319T288 Register (0x2100)

Table 76: **RCVD_RDATA_ACT_319T288 (Register 0x2100)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	[319:288] bits of first actual Read data received when DI error occurs.

RCVD_RDATA_ACT_351T320 Register (0x2104)

Table 77: **RCVD_RDATA_ACT_351T320 Register (0x2104)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	[351:320] bits of first actual Read data received when DI error occurs.

RCVD_RDATA_ACT_383T352 Register (0x2108)

Table 78: **RCVD_RDATA_ACT_383T352 Register (0x2108)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	[383:352] bits of first actual Read data received when DI error occurs.

RCVD_RDATA_ACT_ 015T384 Register (0x210C)

Table 79: **RCVD_RDATA_ACT_ 015T384 Register (0x210C)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	[415:384] bits of first actual Read data received when DI error occurs.

RCVD_RDATA_ACT_447T416 Register (0x2110)

Table 80: **RCVD_RDATA_ACT_447T416 Register (0x2110)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	[447:416] bits of first actual Read data received when DI error occurs.

RCVD_RDATA_ACT_479T448 Register (0x2114)

Table 81: **RCVD_RDATA_ACT_479T448 Register (0x2114)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	[479:448] bits of first actual Read data received when DI error occurs.

RCVD_RDATA_ACT_511T480 Register (0x2118)

Table 82: **RCVD_RDATA_ACT_511T480 Register (0x2118)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	[511:480] bits of first actual Read data received when DI error occurs.

RCVD_RDATA_EXPD_0T31 Register (0x211C)

Table 83: **RCVD_RDATA_EXPD_0T31 Register (0x211C)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	[31:0] bits of first expected Read data when DI error occurs.

RCVD_RDATA_EXPD_32T63 Register (0x2120)

Table 84: **RCVD_RDATA_EXPD_32T63 Register (0x2120)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	[63:32] bits of first expected Read data when DI error occurs.

RCVD_RDATA_EXPD_64T95 Register (0x2124)

Table 85: **RCVD_RDATA_EXPD_64T95 Register (0x2124)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	[95:64] bits of first expected Read data when DI error occurs.

RCVD_RDATA_EXPD_96T127 Register (0x2128)

Table 86: **RCVD_RDATA_EXPD_96T127 Register (0x2128)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	[127:96] bits of first expected Read data when DI error occurs.

RCVD_RDATA_EXPD_128T159 Register (0x212C)

Table 87: **RCVD_RDATA_EXPD_128T159 Register (0x212C)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	[159:128] bits of first expected Read data when DI error occurs.

RCVD_RDATA_EXPD_160T191 Register (0x2130)

Table 88: **RCVD_RDATA_EXPD_160T191 Register (0x2130)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	[191:160] bits of first expected Read data when DI error occurs.

RCVD_RDATA_EXPD_192T223 Register (0x2134)

Table 89: **RCVD_RDATA_EXPD_192T223 Register (0x2134)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	[223:192] bits of first expected Read data when DI error occurs.

RCVD_RDATA_EXPD_255T224 Register (0x2138)

Table 90: **RCVD_RDATA_EXPD_255T224 Register (0x2138)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	[255:224] bits of first expected Read data when DI error occurs.

RCVD_RDATA_EXPD_287T256 Register (0x213C)

Table 91: **RCVD_RDATA_EXPD_287T256 Register (0x213C)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	[287:256] bits of first expected Read data when DI error occurs.

RCVD_RDATA_EXPD_319T288 Register (0x2140)

Table 92: **RCVD_RDATA_EXPD_319T288 Register (0x2140)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	[319:288] bits of first expected Read data when DI error occurs.

RCVD_RDATA_EXPD_351T320 Register (0x2144)

Table 93: **RCVD_RDATA_EXPD_351T320 Register (0x2144)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	[351:320] bits of first expected Read data when DI error occurs.

RCVD_RDATA_EXPD_383T352 Register (0x2148)

Table 94: **RCVD_RDATA_EXPD_383T352 Register (0x2148)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	[383:352] bits of first expected Read data when DI error occurs.

RCVD_RDATA_EXPD_415T384 Register (0x214C)

Table 95: **RCVD_RDATA_EXPD_415T384 Register (0x214C)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	[415:384] bits of first expected Read data when DI error occurs.

RCVD_RDATA_EXPD_447T416 Register (0x2150)

Table 96: **RCVD_RDATA_EXPD_447T416 Register (0x2150)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	[447:416] bits of first expected Read data when DI error occurs.

RCVD_RDATA_EXPD_479T448 Register (0x2154)

Table 97: **RCVD_RDATA_EXPD_479T448 Register (0x2154)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	[479:448] bits of first expected Read data when DI error occurs.

RCVD_RDATA_EXPD_511T480 Register (0x2158)

Table 98: **RCVD_RDATA_EXPD_511T480 Register (0x2158)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	[511:480] bits of first expected Read data when DI error occurs.

LAST_RDATA_RCVD_0T31 Register (0x215C)

Table 99: **LAST_RDATA_RCVD_0T31 Register (0x215C)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	[31:0] bits of last Read data received.

LAST_RDATA_RCVD_32T63 Register (0x2160)

Table 100: **LAST_RDATA_RCVD_32T63 Register (0x2160)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	[63:32] bits of last Read data received.

LAST_RDATA_RCVD_64T95 Register (0x2164)

Table 101: **LAST_RDATA_RCVD_64T95 Register (0x 2164)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	[95:64] bit of last Read data received.

LAST_RDATA_RCVD_96T127 Register (0x2168)

Table 102: **LAST_RDATA_RCVD_96T127 Register (0x2168)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	[127:26] bits of last Read data received.

LAST_RDATA_RCVD_128T159 Register (0x216C)

Table 103: **LAST_RDATA_RCVD_128T159 Register (0x216C)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	[159:128] bits of last Read data received.

LAST_RDATA_RCVD_160T191 Register (0x2170)

Table 104: **LAST_RDATA_RCVD_160T191 Register (0x2170)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	[191:160] bit of last Read data received.

LAST_RDATA_RCVD_192T223 Register (0x2174)

Table 105: **LAST_RDATA_RCVD_192T223 Register (0x2174)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	[223:192] bits of last Read data received.

LAST_RDATA_RCVD_255T224 Register (0x2178)

Table 106: **LAST_RDATA_RCVD_255T224 Register (0x2178)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	[255:224] bits of last Read data received.

LAST_RDATA_RCVD_287T256 Register (0x217C)

Table 107: **LAST_RDATA_RCVD_287T256 Register (0x217C)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	[287:256] bits of last Read data received.

LAST_RDATA_RCVD_319T288 Register (0x2180)

Table 108: **LAST_RDATA_RCVD_319T288 Register (0x2180)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	[319:288] bit of last Read data received.

LAST_RDATA_RCVD_351T320 Register (0x2184)

Table 109: **LAST_RDATA_RCVD_351T320 Register (0x2184)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	[351:320] bits of last Read data received.

LAST_RDATA_RCVD_383T352 Register (0x2188)

Table 110: **LAST_RDATA_RCVD_383T352 Register (0x2188)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	[383:352] bits of last Read data received.

LAST_RDATA_RCVD_415T384 Register (0x218C)

Table 111: **LAST_RDATA_RCVD_415T384 Register (0x218C)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	[415:384] bits of last Read data received.

LAST_RDATA_RCVD_447T416 Register (0x2190)

Table 112: **LAST_RDATA_RCVD_447T416 Register (0x2190)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	[447:416] bits of last Read data received.

LAST_RDATA_RCVD_479T448 Register (0x2194)

Table 113: **LAST_RDATA_RCVD_479T448 Register (0x2194)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	[479:448] bits of last Read data received.

LAST_RDATA_RCVD_511T480 Register (0x2198)

Table 114: **LAST_RDATA_RCVD_511T480 Register (0x2198)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	[511:480] bits of last Read data received.

RRESP_EXOKAY_CNTR Register (0x219C)

Table 115: **RRESP_EXOKAY_CNTR Register (0x219C)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	Number of Read responses received as EXOKAY.

RRESP_SLVERR_CNTR Register (0x21A0)

Table 116: **RRESP_SLVERR_CNTR Register (0x21A0)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	Number of Read responses received as SLVERR.

RRESP_DECERR_CNTR Register (0x21A4)

Table 117: **RESP_DECERR_CNTR Register (0x21A4)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	Number of Read responses received as DECERR.

AXI_VLD_RDY_STATS Register (0x21B0)

Table 118: **AXI_VLD_RDY_STATS Register (0x21B0)**

Bit	Default Value	Access Type	Description
31:10	0	RO	Reserved
9	0	RO	Current status of awvalid signal. This register is used in case of hang, to check the master status.
8	0	RO	Current status of awready signal. This register is used in case of hang, to check the slave status.
7	0	RO	Current status of wvalid signal. This register is used in case of hang, to check the master status.
6	0	RO	Current status of wready signal. This register is used in case of hang, to check the slave status.
5	0	RO	Current status of bvalid signal. This register is used in case of hang, to check the slave status.
4	0	RO	Current status of bready signal. This register is used in case of hang, to check the master status.
3	0	RO	Current status of arvalid signal. This register is used in case of hang, to check the master status.
2	0	RO	Current status of arready signal. This register is used in case of hang, to check the slave status.
1	0	RO	Current status of rvalid signal. This register is used in case of hang, to check the slave status.
0	0	RO	Current status of rready signal. This register is used in case of hang, to check the master status.

BRESP_ERR_COUNT Register (0x21B8)

Table 119: **BRESP_ERR_COUNT Register (0x21B8)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	Indicates how many BSREP errors flagged

BRESP_ERR_ADDR0_LSB_0 Register (0x21BC)

Table 120: **BRESP_ERR_ADDR0_LSB_0 Register (0x21BC)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	Address at which the first BRESP error occurred. These are the 32 LSBs of the 48-bit address.

BRESP_ERR_ADDR0_MSB16_0 Register (0x21C0)

Table 121: **BRESP_ERR_ADDR0_MSB16_0 Register (0x21C0)**

Bit	Default Value	Access Type	Description
31:16	0x0000	RO	Reserved
15:0	0x0000	RO	These are the 16 MSBs of the 48-bit address at which the first BRESP error occurred.

BRESP_ERR_TXN_INFO_0 Register (0x21C4)

Table 122: **BRESP_ERR_TXN_INFO_0 Register (0x21C4)**

Bit	Default Value	Access Type	Description
31:30	0x0	RO	Reserved
29:28	0x0	RO	Error type information: 00 -> bresp_err 01 -> reserved 10 -> reserved 11 -> reserved
27:26	0x0	RO	Write response expected
25:24	0x0	RO	Write response received
23:16	0x00	RO	Reserved
15:0	0x0000	RO	Reports the transaction count value set on the Write instruction (in which the Write response error occurred) txn_count CSV field

BRESP_ERR_ADDR0_LSB_1 Register (0x21C8)

Table 123: **BRESP_ERR_ADDR0_LSB_1 Register (0x21C8)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	These are the 32 LSBs of the 48-bit address at which the second BRESP error occurred.

BRESP_ERR_ADDR0_MSB16_1 Register (0x21CC)

Table 124: **BRESP_ERR_ADDR0_MSB16_1 Register (0x21CC)**

Bit	Default Value	Access Type	Description
31:16	0x0000	RO	Reserved
15:0	0x0000	RO	These are the 16 MSBs out of the 48-bit address at which the second BRESP error occurred.

BRESP_ERR_TXN_INFO_1 Register (0x21D0)

Table 125: **BRESP_ERR_TXN_INFO_1 Register (0x21D0)**

Bit	Default Value	Access Type	Description
31:30	0x0	RO	Reserved
29:28	0x0	RO	Error type information: 00-> rresp err 01 -> data error 10 -> reserved 11 -> reserved
27:26	0x0	RO	Write response expected
25:24	0x0	RO	Write response received
23:16	0x00	RO	[23:16] Beat count at which the second BRESP error flagged.
15:0	0x0000	RO	[15:0] Transaction count at which the second BRESP error flagged.

BRESP_ERR_ADDR0_LSB_2 Register (0x21D4)

Table 126: **BRESP_ERR_ADDR0_LSB_2 Register (0x21D4)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	These are the 32 LSBs of the 48-bit address at which the third BRESP error occurred.

BRESP_ERR_ADDR0_MSB16_2 Register (0x21D8)

Table 127: **BRESP_ERR_ADDR0_MSB16_2 Register (0x21D8)**

Bit	Default Value	Access Type	Description
31:16	0x0000	RO	Reserved
15:0	0x0000	RO	These are the 16 MSBs out of the 48-bit address at which the third BRESP error occurred.

BRESP_ERR_TXN_INFO_2 Register (0x21DC)

Table 128: **BRESP_ERR_TXN_INFO_2 Register (0x21DC)**

Bit	Default Value	Access Type	Description
31:30	0x0	RO	Reserved

Table 128: **BRESP_ERR_TXN_INFO_2 Register (0x21DC)** *(cont'd)*

Bit	Default Value	Access Type	Description
29:28	0x0	RO	Error type information: 00-> rresp err 01 -> data error 10 -> reserved 11 -> reserved
27:26	0x0	RO	Write response expected
25:24	0x0	RO	Write response received
23:16	0x00	RO	[23:16] Beat count at which the third BRESP error flagged.
15:0	0x0000	RO	[15:0] Transaction count at which the third BRESP error flagged.

BRESP_ERR_ADDR0_LSB_3 Register (0x21E0)

Table 129: **BRESP_ERR_ADDR0_LSB_3 Register (0x21E0)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	These are the 32 LSBs of the 48-bit address at which the fourth BRESP error occurred.

BRESP_ERR_ADDR0_MSB16_3 Register (0x21E4)

Table 130: **BRESP_ERR_ADDR0_MSB16_3 Register (0x21E4)**

Bit	Default Value	Access Type	Description
31:16	0x0000	RO	Reserved
15:0	0x0000	RO	These are the 16 MSBs out of the 48-bit address at which the fourth BRESP error occurred.

BRESP_ERR_TXN_INFO_3 Register (0x21E8)

Table 131: **BRESP_ERR_TXN_INFO_3 Register (0x21E8)**

Bit	Default Value	Access Type	Description
31:30	0x0	RO	Reserved
29:28	0x0	RO	Error type information: 00-> rresp err 01 -> data error 10 -> reserved 11 -> reserved
27:26	0x0	RO	Write response expected

Table 131: **BRESP_ERR_TXN_INFO_3 Register (0x21E8)** *(cont'd)*

Bit	Default Value	Access Type	Description
25:24	0x0	RO	Write response received
23:16	0x00	RO	[23:16] Beat count at which the fourth BRESP error flagged.
15:0	0x0000	RO	[15:0] Transaction count at which the fourth BRESP error flagged.

BRESP_ERR_ADDR0_LSB_4 Register (0x21F0)

Table 132: **BRESP_ERR_ADDR0_LSB_4 Register (0x21F0)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	These are the 32 LSBs of the 48-bit address at which the fifth BRESP error occurred.

BRESP_ERR_ADDR0_MSB16_4 Register (0x21F4)

Table 133: **BRESP_ERR_ADDR0_MSB16_4 Register (0x21F4)**

Bit	Default Value	Access Type	Description
31:16	0x0000	RO	Reserved
15:0	0x0000	RO	These are the 16 MSBs out of the 48-bit address at which the fifth BRESP error occurred.

BRESP_ERR_TXN_INFO_4 Register (0x21F8)

Table 134: **BRESP_ERR_TXN_INFO_4 Register (0x21F8)**

Bit	Default Value	Access Type	Description
31:30	0x0	RO	Reserved
29:28	0x0	RO	Error type information: 00-> rresp err 01 -> data error 10 -> reserved 11 -> reserved
27:26	0x0	RO	Write response expected
25:24	0x0	RO	Write response received
23:16	0x0000	RO	[23:16] Beat count at which the fifth BRESP error flagged.
15:0	0x0000	RO	[15:0] Transaction count at which the fifth BRESP error flagged.

BRESP_ERR_ADDR0_LSB_5 Register (0x21FC)

Table 135: **RESP_ERR_ADDR0_LSB_5 Register (0x21FC)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	These are the 32 LSBs of the 48-bit address at which the sixth BRESP error occurred.

BRESP_ERR_ADDR0_MSB16_5 Register (0x2200)

Table 136: **BRESP_ERR_ADDR0_MSB16_5 Register (0x2200)**

Bit	Default Value	Access Type	Description
31:16	0x0000	RO	Reserved
15:0	0x0000	RO	These are the 16 MSBs out of the 48-bit address at which the sixth BRESP error occurred.

BRESP_ERR_TXN_INFO_5 Register (0x2204)

Table 137: **BRESP_ERR_TXN_INFO_5 Register (0x2204)**

Bit	Default Value	Access Type	Description
31:24	0x00	RO	Reserved
23:16	0x0000	RO	[23:16] Beat count at which the sixth BRESP error flagged.
15:0	0x0000	RO	[15:0] Transaction count at which the sixth BRESP error flagged.

BRESP_ERR_ADDR0_LSB_6 Register (0x2208)

Table 138: **BRESP_ERR_ADDR0_LSB_6 Register (0x2208)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	These are the 32 LSBs of the 48-bit address at which the seventh BRESP error occurred.

BRESP_ERR_ADDR0_MSB16_6 Register (0x220C)

Table 139: **BRESP_ERR_ADDR0_MSB16_6 Register (0x220C)**

Bit	Default Value	Access Type	Description
31:16	0x0000	RO	Reserved
15:0	0x0000	RO	These are the 16 MSBs out of the 48-bit address at which the seventh BRESP error occurred..

BRESP_ERR_TXN_INFO_6 Register (0x2210)

Table 140: **BRESP_ERR_TXN_INFO_6 Register (0x2210)**

Bit	Default Value	Access Type	Description
31:30	0x0	RO	Reserved
29:28	0x0	RO	Error type information: 00-> rresp err 01 -> data error 10 -> reserved 11 -> reserved
27:26	0x0	RO	Write response expected
25:24	0x0	RO	Write response received
23:16	0x0000	RO	[23:16] Beat count at which the seventh BRESP error flagged.
15:0	0x0000	RO	[15:0] Transaction count at which the seventh BRESP error flagged.

BRESP_ERR_ADDR0_LSB_7 Register (0x2214)

Table 141: **BRESP_ERR_ADDR0_LSB_7 Register (0x2214)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	These are the 32 LSBs of the 48-bit address at which the eighth BRESP error occurred.

BRESP_ERR_ADDR0_MSB16_7 Register (0x2218)

Table 142: **BRESP_ERR_ADDR0_MSB16_7 Register (0x2218)**

Bit	Default Value	Access Type	Description
31:16	0x0000	RO	Reserved
15:0	0x0000	RO	These are the 16 MSBs out of the 48-bit address at which the eighth BRESP error occurred

BRESP_ERR_TXN_INFO_7 Register (0x221C)

Table 143: **BRESP_ERR_TXN_INFO_7 Register (0x221C)**

Bit	Default Value	Access Type	Description
31:30	0x0	RO	Reserved

Table 143: **BRESP_ERR_TXN_INFO_7 Register (0x221C)** *(cont'd)*

Bit	Default Value	Access Type	Description
29:28	0x0	RO	Error type information: 00-> rresp err 01 -> data error 10 -> reserved 11 -> reserved
27:26	0x0	RO	Write response expected
25:24	0x0	RO	Write response received
23:16	0x0000	RO	[23:16] Beat count at which the eighth BRESP error flagged.
15:0	0x0000	RO	[15:0] Transaction count at which the eighth BRESP error flagged.

BRESP_ERR_ADDR0_LSB_8 Register (0x2220)

Table 144: **BRESP_ERR_ADDR0_LSB_8 Register (0x2220)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	These are the 32 LSBs of the 48-bit address at which the ninth BRESP error occurred.

BRESP_ERR_ADDR0_MSB16_8 Register (0x2224)

Table 145: **BRESP_ERR_ADDR0_MSB16_8 Register (0x2224)**

Bit	Default Value	Access Type	Description
31:16	0x0000	RO	Reserved
15:0	0x0000	RO	These are the 16 MSBs out of the 48-bit address at which the ninth BRESP error occurred.

BRESP_ERR_TXN_INFO_8 Register (0x2228)

Table 146: **BRESP_ERR_TXN_INFO_8 Register (0x2228)**

Bit	Default Value	Access Type	Description
31:30	0x0	RO	Reserved
29:28	0x0	RO	Error type information: 00-> rresp err 01 -> data error 10 -> reserved 11 -> reserved
27:26	0x0	RO	Write response expected
25:24	0x0	RO	Write response received

Table 146: **BRESP_ERR_TXN_INFO_8 Register (0x2228)** *(cont'd)*

Bit	Default Value	Access Type	Description
23:16	0x0000	RO	[23:16] Beat count at which the ninth BRESP error flagged.
15:0	0x0000	RO	[15:0] Transaction count at which the ninth BRESP error flagged.

BRESP_ERR_ADDR0_LSB_9 Register (0x222C)

Table 147: **BRESP_ERR_ADDR0_LSB_9 Register (0x222C)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	These are the 32 LSBs of the 48-bit address at which the tenth BRESP error occurred.

BRESP_ERR_ADDR0_MSB16_9 Register (0x2230)

Table 148: **BRESP_ERR_ADDR0_MSB16_9 Register (0x2230)**

Bit	Default Value	Access Type	Description
31:16	0x0000	RO	Reserved
15:0	0x0000	RO	These are the 16 MSBs out of the 48-bit address at which the tenth BRESP error occurred.

BRESP_ERR_TXN_INFO_9 Register (0x2234)

Table 149: **BRESP_ERR_TXN_INFO_9 Register (0x2234)**

Bit	Default Value	Access Type	Description
31:30	0x0	RO	Reserved
29:28	0x0	RO	Error type information: 00-> rresp err 01 -> data error 10 -> reserved 11 -> reserved
27:26	0x0	RO	Write response expected
25:24	0x0	RO	Write response received
23:16	0x0000	RO	[23:16] Beat count at which the tenth BRESP error flagged.
15:0	0x0000	RO	[15:0] Transaction count at which the tenth BRESP error flagged.

CAPTURE_ALL Register (0x2234)

Table 150: **CAPTURE_ALL Register (0x2234)**

Bit	Default Value	Access Type	Description
31:0		RO	For infinite transactions, send this address along with Read enable. It captures the all the *_AL counter together at that movement. (bandwidth, request counter). All the *_AL register is updated with new value until you send CAPTURE_ALL address along with Read enable. Otehrwise, it holds the previous values till reset.

RBEAT_COUNTER_AL Register (0x2238)

Table 151: **RBEAT_COUNTER_AL Register (0x2238)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	Number of Read beats received by the TG. This is used for infinite transactions. This register is updated with a new value until you send CAPTURE_ALL address along with Read enable. Otherwise, it holds the previous values till reset.

ARREQ_COUNTER_AL Register (0x223C)

Table 152: **ARREQ_COUNTER_AL Register (0x223C)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	Number of Read requests issued by the TG. This is used for infinite transactions. This register is updated with a new value until you send CAPTURE_ALL address along with Read enable. Otherwise, it holds the previous values till reset.

RLAST_CNTR_AL Register (0x2240)

Table 153: **RLAST_CNTR_AL Register (0x2240)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	Number of rlast received by the TG. This is used for infinite transactions. This register is updated with a new value until you send CAPTURE_ALL address along with Read enable. Otherwise, it holds the previous values till reset.

RBW_EFF_CLK_CNTR_AL Register (0x2244)

Table 154: **RBW_EFF_CLK_CNTR_AL Register (0x2244)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	Read bandwidth counter. Number of cycles from the start of the TG to the rdch_done signal, a Read Traffic Done signal mapped to Bit 7 of the FLOW_EMPTY register. This is used to calculate the bandwidth. Read Bandwidth =RBEAT_CNTR/ RBW_EFF_CLK_CNTR. This is used for infinite transactions. This register is updated with a new value until you send CAPTURE_ALL address along with Read enable. Otherwise, it holds the previous values till reset.

RWORST_LATENCY_AL Register (0x2248)

Table 155: **RWORST_LATENCY_AL Register (0x2248)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	Maximum latency of Read traffic. This is used for infinite transactions. This register is updated with a new value until you send CAPTURE_ALL address along with Read enable. Otherwise, it holds the previous values till reset.

RBEST_LATENCY_AL Register (0x224C)

Table 156: **RBEST_LATENCY_AL Register (0x224C)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	Minimum latency of Read traffic. This is used for infinite transactions. This register is updated with a new value until you send CAPTURE_ALL address along with Read enable. Otherwise, it holds the previous values till reset.

RAVG_LATENCY_AL Register (0x2250)

Table 157: **RAVG_LATENCY_AL Register (0x2250)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	Total latency of the Read traffic. It can be used to calculate the average latency. Average latency = RAVG_LATENCY/ ARREQ_CNTR. This is used for infinite transactions. This register is updated with a new value until you send CAPTURE_ALL address along with Read enable. Otherwise, it holds the previous values till reset.

AWREQ_CNTR_AL Register (0x2254)

Table 158: **AWREQ_CNTR_AL Register (0x2254)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	Number of Write requests issued by the TG. This is used for infinite transactions. This register is updated with a new value until you send CAPTURE_ALL address along with Read enable. Otherwise, it holds the previous values till reset.

WLAST_CNTR_AL Register (0x2258)

Table 159: **WLAST_CNTR_AL Register (0x2258)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	Number of wlast issued by the TG. This is used for infinite transactions. This register is updated with a new value until you send CAPTURE_ALL address along with Read enable. Otherwise, it holds the previous values till reset.

BRESP_CNTR_AL Register (0x225C)

Table 160: **BRESP_CNTR_AL Register (0x225C)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	Number of responses received by the TG. This is used for infinite transactions. This register is updated with a new value until you send CAPTURE_ALL address along with Read enable. Otherwise, it holds the previous values till reset.

WRBEAT_CNTR_AL Register (0x2260)

Table 161: **WRBEAT_CNTR_AL Register (0x2260)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	Number of Write beats issued by the TG. This is used for infinite transactions. This register is updated with a new value until you send CAPTURE_ALL address along with Read enable. Otherwise, it holds the previous values till reset.

WBW_EFF_CLK_CNTR_AL Register (0x2264)

Table 162: **WBW_EFF_CLK_CNTR_AL Register (0x2264)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	Write bandwidth counter. Number of cycles from the start of the TG to the wrch_done signal, a Write Traffic Done signal mapped to Bit 8 of the FLOW_EMPTY register. This is used to calculate the bandwidth. Write bandwidth = WRBEAT_CNTR/ WBW_EFF_CLK_CNTR This is used for infinite transactions. This register is updated with a new value until you send CAPTURE_ALL address along with Read enable. Otherwise, it holds the previous values till reset.

WWORST_LATENCY_AL Register (0x2268)

Table 163: **WWORST_LATENCY_AL Register (0x2268)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	Maximum latency of Write traffic .This is used for infinite transactions. This register is updated with a new value until you send CAPTURE_ALL address along with Read enable. Otherwise, it holds the previous values till reset.

WBEST_LATENCY_AL Register (0x226C)

Table 164: **WBEST_LATENCY_AL Register (0x226C)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	Minimum latency of Write traffic. This is used for infinite transactions. This register is updated with a new value until you send CAPTURE_ALL address along with Read enable. Otherwise, it holds the previous values till reset.

WAVG_LATENCY_AL Register (0x2270)

Table 165: **WAVG_LATENCY_AL Register (0x2270)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	Total latency of the Read traffic. This can be used to calculate the average latency. Average latency =WAVG _LATENCY/ AWREQ _CNTR . This is used for infinite transactions. This register is updated with a new value until you send CAPTURE_ALL address along with Read enable. Otherwise, it holds the previous values till reset.

FLOW_EMPTY_AL Register (0x2274)

Table 166: **FLOW_EMPTY_AL Register (0x2274)**

Bit	Default Value	Access Type	Description
31:13	0	RO	Reserved
12	0	RO	This bit is set when instruction BRAM is empty. This is used for infinite transactions. This register is updated with a new value until you send CAPTURE_ALL address along with Read enable. Otherwise, it holds the previous values till reset.
11	0	RO	This bit is set when Write channel instruction FIFO is empty. This is used for Infinite transactions. This register is updated with a new value until you send CAPTURE_ALL address along with Read enable. Otherwise, it holds the previous values till reset.
10	0	RO	This bit is set when Read channel instruction FIFO is empty. This is used for Infinite transactions. This register is updated with a new value until you send CAPTURE_ALL address along with Read enable. Otherwise, it holds the previous values till reset.
9	0	RO	If set, the TG has completed sending both Read and Write traffic. This is used for infinite transactions. This register is updated with a new value until you send CAPTURE_ALL address along with Read enable. Otherwise, it holds the previous values till reset.
8	0	RO	Both Read and Write traffic done. If set, then TG has completed sending both Read and Write traffic. This is used for infinite transactions. This register is updated with a new value until you send CAPTURE_ALL address along with Read enable. Otherwise, it holds the previous values till reset.
7	0	RO	Write traffic done. If set, then the TG has completed sending the Write traffic. This is used for infinite transactions. This register is updated with a new value until you send CAPTURE_ALL address along with Read enable. Otherwise, it holds the previous values till reset.
6	0	RO	Read traffic done. If set, then the TG has completed sending the Read traffic. This is used for Infinite transaction. This register is updated with a new value until you send CAPTURE_ALL address along with Read enable. Otherwise, it holds the previous values till reset.
5	0	RO	This bit is set when Read channel FIFO is full. This is used for infinite transactions. This register is updated with a new value until you send CAPTURE_ALL address along with Read enable. Otherwise, it holds the previous values till reset.
4	0	RO	This bit is set when TG error occurs. This is used for infinite transactions. This register is updated with a new value until you send CAPTURE_ALL address along with Read enable. Otherwise, it holds the previous values till reset.
3	0	RO	Register bit is set High when Read request FIFO overflows. This is used for infinite transactions. This register is updated with a new value until you send CAPTURE_ALL address along with Read enable. Otherwise, it holds the previous values till reset.
2	0	RO	Register bit is set High when Read request FIFO underflows. This is used for infinite transactions. This register is updated with a new value until you send CAPTURE_ALL address along with Read enable. Otherwise, it holds the previous values till reset.
1	0	RO	Register bit is set High when Write request FIFO overflows. This is used for infinite transactions. This register is updated with a new value until you send CAPTURE_ALL address along with Read enable. Otherwise, it holds the previous values till reset.

Table 166: **FLOW_EMPTY_AL Register (0x2274)** *(cont'd)*

Bit	Default Value	Access Type	Description
0	0	RO	Register bit is set High when Write request FIFO underflows. This is used for infinite transactions. This register is updated with a new value until you send CAPTURE_ALL address along with Read enable. Otherwise, it holds the previous values till reset.

BRESP_EXOKAY_CNTR_AL Register (0x2278)

Table 167: **BRESP_EXOKAY_CNTR_AL Register (0x2278)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	Number of Write responses received as EXOKAY. This is used for infinite transactions. This register is updated with a new value until you send CAPTURE_ALL address along with Read enable. Otherwise, it holds the previous values till reset.

BRESP_SLVERR_CNTR_AL Register (0x227c)

Table 168: **BRESP_SLVERR_CNTR_AL Register (0x227c)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	Number of Write responses received as SLVERR. This is used for infinite transactions. This register is updated with a new value until you send CAPTURE_ALL address along with Read enable. Otherwise, it holds the previous values till reset.

BRESP_DECERR_CNTR_AL Register (0x2280)

Table 169: **BRESP_DECERR_CNTR_AL Register (0x2280)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	Number of Write responses received as DECERR. This is used for infinite transactions. This register is updated with a new value until you send CAPTURE_ALL address along with Read enable. Otherwise, it holds the previous values till reset.

RRESP_EXOKAY_CNTR_AL Register (0x2284)

Table 170: **RRESP_EXOKAY_CNTR_AL Register (0x2284)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	Number of Read responses received as EXOKAY. This is used for infinite transactions. This register is updated with a new value until you send CAPTURE_ALL address along with Read enable. Otherwise, it holds the previous values till reset.

RRESP_SLVERR_CNTR_AL Register (0x2288)

Table 171: **RRESP_SLVERR_CNTR_AL Register (0x2288)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	Number of Read responses received as SLVERR. This is used for infinite transactions. This register is updated with a new value until you send CAPTURE_ALL address along with Read enable. Otherwise, it holds the previous values till reset.

RRESP_DECERR_CNTR_AL Register (0x228C)

Table 172: **RRESP_DECERR_CNTR_AL Register (0x228C)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	Number of Read responses received as DECERR. This is used for infinite transactions. This register is updated with a new value until you send CAPTURE_ALL address along with Read enable. Otherwise, it holds the previous values till reset.

LAST_RDATA_RCVD_0T31_AL Register (0x2290)

Table 173: **LAST_RDATA_RCVD_0T31_AL Register (0x2290)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	This is used for infinite transactions. This register is updated with a new value until you send CAPTURE_ALL address along with Read enable. Otherwise, it holds the previous values till reset.

LAST_RDATA_RCVD_32T63_AL Register (0x2294)

Table 174: **LAST_RDATA_RCVD_32T63_AL Register (0x2294)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	This is used for infinite transactions. This register is updated with a new value until you send CAPTURE_ALL address along with Read enable. Otherwise, it holds the previous values till reset.

LAST_RDATA_RCVD_64T95_AL Register (0x2298)

Table 175: **LAST_RDATA_RCVD_64T95_AL Register (0x2298)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	This is used for infinite transactions. This register is updated with a new value until you send CAPTURE_ALL address along with Read enable. Otherwise, it holds the previous values till reset.

LAST_RDATA_RCVD_96T127_AL Register (0x229C)

Table 176: **LAST_RDATA_RCVD_96T127_AL Register (0x229C)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	This is used for infinite transactions. This register is updated with a new value until you send CAPTURE_ALL address along with Read enable. Otherwise, it holds the previous values till reset.

LAST_RDATA_RCVD_128T159_AL Register (0x22A0)

Table 177: **LAST_RDATA_RCVD_128T159_AL Register (0x22A0)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	This is used for infinite transactions. This register is updated with a new value until you send CAPTURE_ALL address along with Read enable. Otherwise, it holds the previous values till reset.

LAST_RDATA_RCVD_160T191_AL Register (0x22A4)

Table 178: **LAST_RDATA_RCVD_160T191_AL Register (0x22A4)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	This is used for infinite transactions. This register is updated with a new value until you send CAPTURE_ALL address along with Read enable. Otherwise, it holds the previous values till reset.

LAST_RDATA_RCVD_192T223_AL Register (0x22A8)

Table 179: **LAST_RDATA_RCVD_192T223_AL Register (0x22A8)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	This is used for infinite transactions. This register is updated with a new value until you send CAPTURE_ALL address along with Read enable. Otherwise, it holds the previous values till reset.

LAST_RDATA_RCVD_255T224_AL Register (0x22AC)

Table 180: **LAST_RDATA_RCVD_255T224_AL Register (0x22AC)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	This is used for infinite transactions. This register is updated with a new value until you send CAPTURE_ALL address along with Read enable. Otherwise, it holds the previous values till reset.

LAST_RDATA_RCVD_287T256_AL Register (0x22B0)

Table 181: **LAST_RDATA_RCVD_287T256_AL Register (0x22B0)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	This is used for infinite transactions. This register is updated with a new value until you send CAPTURE_ALL address along with Read enable. Otherwise, it holds the previous values till reset.

LAST_RDATA_RCVD_319T288_AL Register (0x22B4)

Table 182: **LAST_RDATA_RCVD_319T288_AL Register (0x22B4)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	This is used for infinite transactions. This register is updated with a new value until you send CAPTURE_ALL address along with Read enable. Otherwise, it holds the previous values till reset.

LAST_RDATA_RCVD_351T320_AL Register (0x22B8)

Table 183: **LAST_RDATA_RCVD_351T320_AL Register (0x22B8)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	This is used for infinite transactions. This register is updated with a new value until you send CAPTURE_ALL address along with Read enable. Otherwise, it holds the previous values till reset.

LAST_RDATA_RCVD_383T352_AL Register (0x22BC)

Table 184: **LAST_RDATA_RCVD_383T352_AL Register (0x22BC)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	This is used for infinite transactions. This register is updated with a new value until you send CAPTURE_ALL address along with Read enable. Otherwise, it holds the previous values till reset.

LAST_RDATA_RCVD_415T384_AL Register (0x22C0)

Table 185: **LAST_RDATA_RCVD_415T384_AL Register (0x22C0)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	This is used for infinite transactions. This register is updated with a new value until you send CAPTURE_ALL address along with Read enable. Otherwise, it holds the previous values till reset.

LAST_RDATA_RCVD_447T416_AL Register (0x22C4)

Table 186: **LAST_RDATA_RCVD_447T416_AL Register (0x22C4)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	This is used for infinite transactions. This register is updated with a new value until you send CAPTURE_ALL address along with Read enable. Otherwise, it holds the previous values till reset.

LAST_RDATA_RCVD_479T448_AL Register (0x22C8)

Table 187: **LAST_RDATA_RCVD_479T448_AL Register (0x22C8)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	This is used for infinite transactions. This register is updated with a new value until you send CAPTURE_ALL address along with Read enable. Otherwise, it holds the previous values till reset.

LAST_RDATA_RCVD_511T480_AL Register (0x22CC)

Table 188: **LAST_RDATA_RCVD_511T480_AL Register (0x22CC)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	This is used for infinite transactions. This register is updated with a new value until you send CAPTURE_ALL address along with Read enable. Otherwise, it holds the previous values till reset.

CAPTR_WRBEAT (0x22DC)

Table 189: **CAPTR_WRBEAT (0x22DC)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	This is used for calculating average write bandwidth by windowing technique. It shows total number of write beats captured in a particular window of certain number of AXI clock cycles. The windowing technique is enabled using CAPTURE_ENABLE(0x42DO) and size of window is defined using START_ADDR (0x42D4) and END_ADDR(0x42D8). The write bandwidth is calculated as WRITE_BW= (No. of write beats captured/ Window size)* Total bandwidth. Total bandwidth = (AXI_DATA_WIDTH)*(AXI_CLOCK_FREQUENCY) Window size = END_ADDR - START ADDR The write efficiency is calculated as WR_EFF = (No. of write beats captured/ Window size) *100 .

CAPTR_RDBEAT (0x22E0)

Table 190: **CAPTR_RDBEAT (0x22E0)**

Bit	Default Value	Access Type	Description
31:0	0x0000_0000	RO	This is used for calculating average read bandwidth by windowing technique. It shows total number of read beats captured in a particular window of certain number of AXI clock cycles. The windowing technique is enabled using CAPTURE_ENABLE(0x42DO) and size of window is defined using START_ADDR (0x42D4) and END_ADDR(0x42D8). The read bandwidth is calculated as READ_BW= (No. of read beats captured/Window size)* Total bandwidth. Total bandwidth = (AXI_DATA_WIDTH)*(AXI_CLOCK_FREQUENCY) Window size = END_ADDR - START ADDR The read efficiency is calculated as RD_EFF = (No. of read beats captured/ Window size) *100 .

AXI4-Stream Configuration Registers

***Note*:** The address column denotes the address for TG0.

START Register (0X4004)

Table 191: **START Register (0X4004)**

Bit	Default Value	Access Type	Description
31:1	0	R/W	Reserved
0	0	R/W	When set, enables the TG to send the traffic. It is a self-clearing register.

RESET Register (0X4008)

Table 192: **RESET Register (0X4008)**

Bit	Default Value	Access Type	Description
31:1	0	R/W	Reserved
0	0	R/W	Active-Low reset

PLAY_PAUSE Register (0x401C)

Table 193: **PLAY_PAUSE Register (0x401C)**

Bit	Default Value	Access Type	Description
31:1	0x0000_0001	R/W	Reserved

Table 193: **PLAY_PAUSE Register (0x401C)** *(cont'd)*

Bit	Default Value	Access Type	Description
0	1	R/W	Write 0 to pause the traffic and write 1 to play.

STROBE LSB BITS Register (0x4020)

Table 194: **STROBE LSB BITS Register (0x4020)**

Bit	Default Value	Access Type	Description
31:1	0xFFFF_FFFF	R/W	32 LSBs of strobe signal (tstrb).

STROBE MSB BITS Register (0x4024)

Table 195: **STROBE MSB BITS Register (0x4024)**

Bit	Default Value	Access Type	Description
31:1	0xFFFF_FFFF	R/W	32 MSBs of strobe signal (tstrb)

KEEP LSB BITS Register (0x4028)

Table 196: **KEEP LSB BITS Register (0x4028)**

Bit	Default Value	Access Type	Description
31:1	0xFFFF_FFFF	R/W	32 LSBs of keep signal (tkeep)

KEEP MSB BITS Register (0x402C)

Table 197: **KEEP MSB BITS Register (0x402C)**

Bit	Default Value	Access Type	Description
31:1	0xFFFF_FFFF	R/W	32 MSBs of keep signal (tkeep)

AXI4-Stream Status Registers

BANDWIDTH_COUNTER Register (0X200C)

Table 198: **BANDWIDTH_COUNTER Register (0X200C)**

Bit	Default Value	Access Type	Description
31:1	0x0000_0000	RO	Bandwidth counter. Number of clock cycles completed to send all requests. This is used to calculate the bandwidth. Bandwidth = BEAT_COUNTER/ BANDWIDTH_COUNTER

REQUEST_COUNTER Register (0X2010)

Table 199: **REQUEST_COUNTER Register (0X2010)**

Bit	Default Value	Access Type	Description
31:1	0x0000_0000	RO	Number of requests issued by the TG

TXN_COMPLETE Register (0x2014)

Table 200: **TXN_COMPLETE Register (0x2014)**

Bit	Default Value	Access Type	Description
31:1	0	RO	Reserved
0	0	RO	If set, then the TG has completed sending the traffic.

BEAT_COUNTER Register (0x2018)

Table 201: **BEAT_COUNTER Register (0x2018)**

Bit	Default Value	Access Type	Description
31:1	0x0000_0000	RO	Number of beats issued by the TG.

Designing with the Core

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

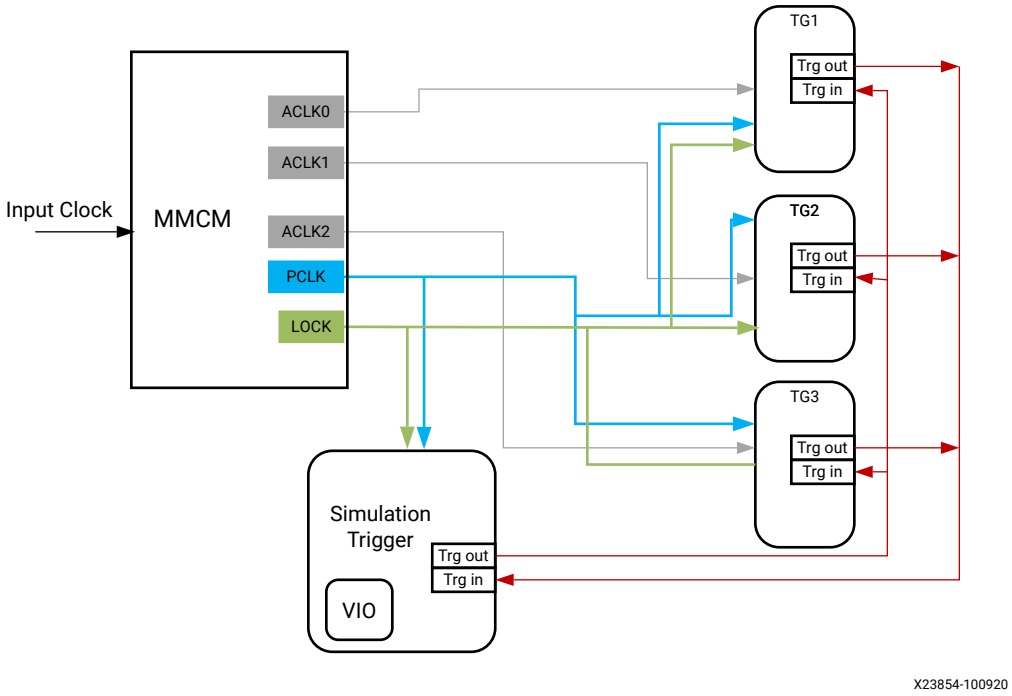
Clocking

The mixed-mode clock manager (MMCM) block has a input clock that is used to generate the AXI clocks for different TGs and a `pclk` signal for instruction block RAM loading, the register space, and Simulation Trigger. You can run multiple TGs at the same or at a different frequency. You can use an MMCM to generate multiple clock frequencies.

See *Versal ACAP Clocking Resources Architecture Manual* (AM003).

In the following figure, the MMCM is generating multiple AXI clock frequencies using a single input clock. Use the generated clocks to run TGs at independent AXI clock frequencies. The TG also needs a low frequency `pclk`clock along with a high frequency AXI clock for loading the instruction block RAM and accessing the register space. You can connect a `LOCK` signal from the MMCM to the `RESET` port of the TG.

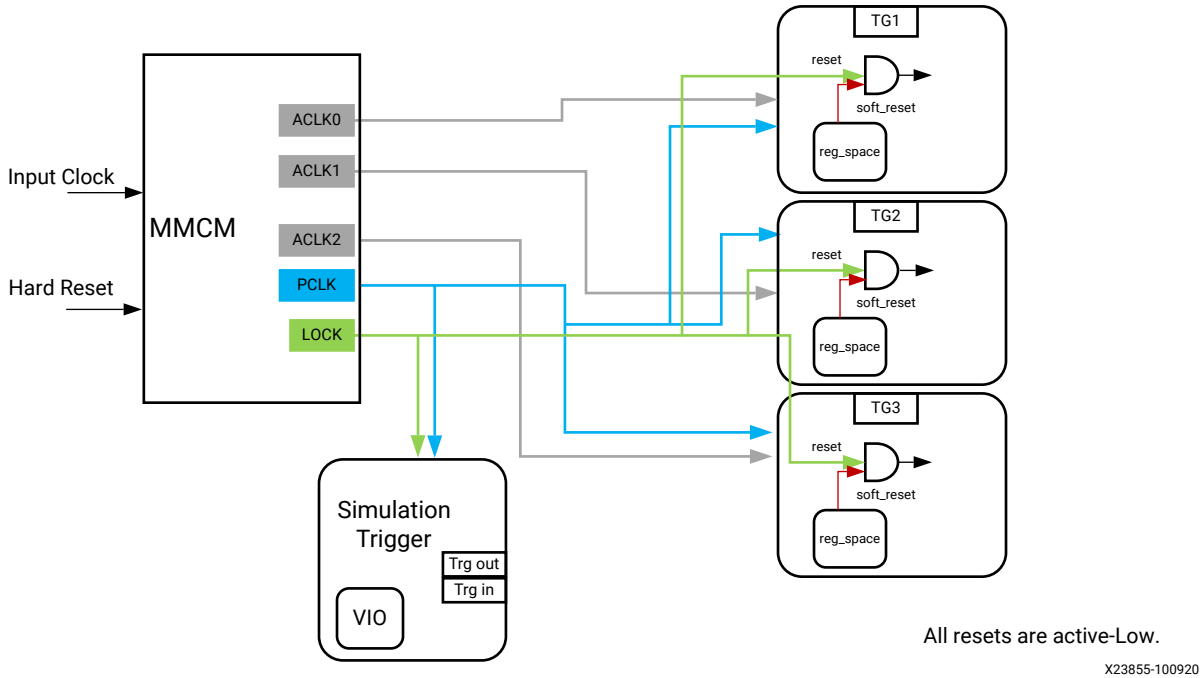
Figure 11: **Clock Architecture Block Diagram**



Resets

The TG has one external asynchronous hard reset and a soft reset from the register space.

Figure 12: **Reset Architecture Block Diagram**



User Guidelines

- Traffic is specified through the CSV file in the Vivado IDE.
- In designs with multiple TGs, if only one CSV is used, the same CSV must be given to all TGs.
- In designs with multiple TGs, if traffic shaping is used (traffic is defined in phases using the `PHASE_DONE` command, the same CSV must be given to all TGs, else, simulation might give incorrect results.
- The source ID in the Vivado IDE and the `tg_num` in the CSV must be the same.
- Nested loops are not supported.
- The first instruction cannot be WAIT.
- Consecutive WAIT instructions are not supported and might result in unpredictable behavior.
- The `txn_count` or `loop_count` in the CSV cannot be zero.
- A LOOP command should contain at least one instruction.
- CSV fields should not contain any extra space (such as `" WRITE"` instead of `"WRITE"`).
- The AXI clock should always be greater than or equal to the `sim_trig` clock (`pclk`).
- CSV hex fields must have a "0x" before the rest of the value; for example, 0x12345678 for a 32-bit hex value.

- In the case of an incremental address pattern (that is, an `auto_incr` or `addr_incr_by` value address pattern):
  - When generating the AXI address, if the TG is asked to generate an address that crosses the 4K boundary, the TG splits that single transaction into two transactions at the 4K boundary by adjusting the AXI length. For example, if the TG is asked to generate `axi_address` = 0x0201_0000_0F80, `axi_size` = 6, and `axi_len` = 7, this single transaction crosses the 4K boundary. The TG splits this transaction into two transactions at the 4K boundary to avoid the 4K boundary crossing issue. The split transactions are:
    - TXN1: `axi_address` = 0x0201_0000_0F80, `axi_size` = 6 and `axi_len` = 1
    - TXN2: `axi_address` = 0x0201_0000_1000, `axi_size` = 6 and `axi_len` = 5.
  - When generating the AXI address, if the TG is asked to generate an out-of-range address (that is, an address that does not fall within `base_addr` and `high_addr`), the TG restarts the transaction from `base_addr`.

User Guidelines Specific to AXI4

- Read, Write, and Wait are each considered to be one instruction.
- If the transaction count per CSV command is less than 8 maximum bandwidth is not guaranteed from the TG.
- Supported axi_size and axi_len combinations are shown in the following table.

Table 202: **Supported Combinations of AXI Length and AXI Size**

axi_size in Bytes	Supported Number of Beats per Transaction (axi_len + 1)
128	≤32
64	≤64
32	≤128
16	≤256
8	Any value supported by AXI protocol
4	Any value supported by AXI protocol
2	Any value supported by AXI protocol
1	Any value supported by AXI protocol

- If data integrity is selected as Constant Id in the Vivado IDE, then CSV field for ID selection (axi_id) is ignored. In this case, transactions are sent out with the ID value as 0.
- When Data Integrity checks are enabled with mixed write read traffic patterns, the user needs to take care that the memory is written before it is read. This can be done by running simulations and observing how long it takes to get BRESPs and adjust the CSV start_delay value for the Read traffic accordingly.

- If data integrity is enabled, Write/Read coherency must be maintained in the traffic specification itself. Writes must happen before reads and a data integrity error results if this does not happen. A Wait instruction must be used after Write instructions and before Read instructions.
- If data integrity is enabled Write and Read requests must have the same data pattern selected.
- Slave error or decode error responses from the AXI slave are also reported as a data integrity failure.
- Latency is expected for reads from the TG if the CSV has a few Write instructions followed by Read instructions.

User Guidelines Specific to AXI4-Stream

- If the transaction count per CSV command is less than five, maximum bandwidth is not guaranteed from the TG.
- Stream and Wait are each considered as one instruction.
- Inter packet delay should be zero when the packet length is zero.
- Loop count one is unsupported.
- The minimum AXI4-Stream data width for the 16byte_incr data pattern is 128-bit. An AXI4-Stream data width less than 128-bit does not support the 16byte_incr data type.
- The maximum inter transfer delay count is 65530.

Using the Traffic Generators

CSV File Run for the Synthesizable TG

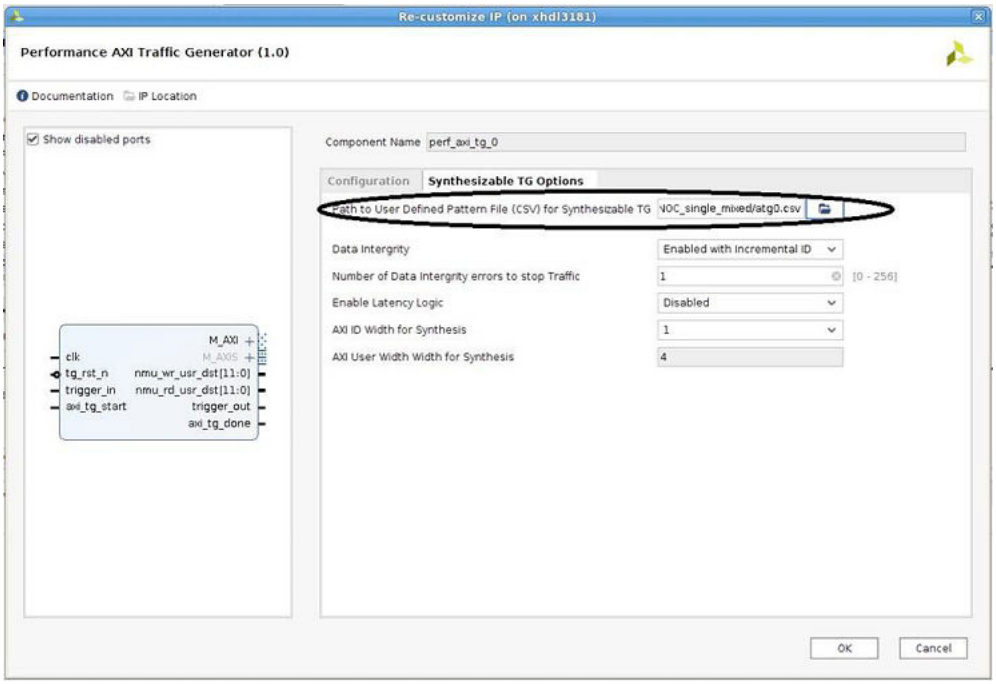
In a default run, traffic can be run by linking to the path of the CSV file in the Vivado IDE. This can be done both in simulation and hardware.

See Common CSV Input for All TGs.

Simulation

When the design is generated with the synthesizable configuration, simulation can be run without following any sequence of steps. A CSV file can be attached in the Vivado IDE as shown below and you can run the simulation. Internal scripts generate the MEM file and load it for simulation.

Figure 13: **Path for CSV file**



Hardware

When the bit file is programmed, default traffic can be run on hardware without loading the instruction into the block RAM or following any sequence of steps. When the CSV file is attached in the Vivado IDE, the tools automatically generate the bit file and program it on the hardware. Traffic starts automatically and the traffic pattern mentioned in the CSV is driven by the TG. This can be done only once on hardware; that is, programming the same bit stream (or PDI) again runs the same traffic with which the bit stream (or PDI) was generated. If a different traffic pattern needs to be generated, attach a new CSV file in the IDE, generate the bit file again, and program it on the hardware.

Running Custom Traffic for the Synthesizable TG

The TG has been equipped with a register interface to load the traffic pattern into the block RAM and run the custom traffic. This can be done through the AXI4-Lite and VIO interfaces.

Hardware

After running the default traffic, the VIO interface can be used to load the block RAM with a new set of instructions to generate new traffic. The following sequence must be followed to load the block RAM and run the new traffic pattern. The addresses of the corresponding registers as well as sample addresses and data for two CSV instructions to load into the block RAM are provided.

1. Write 0 to the SOFT_RST register (active-Low register):

address: 0000_4000 Data: 0000_0000

2. Write 1 to the SOFT_RST register:

   address: 0000_4000 Data: 0000_0001

3. Load the block RAM with instructions. The addresses to load the block RAM are as follows:

   Instruction = 1;

   address = 32'h00008030; data = instruction1[12*32+:32]

   address = 32'h0000802c; data = instruction1[11*32+:32]

   address = 32'h00008028; data = instruction1[10*32+:32]

   address = 32'h00008024; data = instruction1[9*32+:32]

   address = 32'h00008020; data = instruction1[8*32+:32]

   address = 32'h0000801c; data = instruction1[7*32+:32]

   address = 32'h00008018; data = instruction1[6*32+:32]

   address = 32'h00008014; data = instruction1[5*32+:32]

   address = 32'h00008010; data = instruction1[4*32+:32]

   address = 32'h0000800c; data = instruction1[3*32+:32]

   address = 32'h00008008; data = instruction1[2*32+:32]

   address = 32'h00008004; data = instruction1[1*32+:32]

   address = 32'h00008000; data = instruction1[0*32+:32]

   Instruction =2

   address = 32'h00008070; data = instruction2[12*32+:32]

   address = 32'h0000806c; data = instruction2[11*32+:32]

   address = 32'h00008068; data = instruction2[10*32+:32]

   address = 32'h00008064; data = instruction2[9*32+:32]

   address = 32'h00008060; data = instruction2[8*32+:32]

   address = 32'h0000805c; data= instruction2[7*32+:32]

   address = 32'h00008058; data = instruction2[6*32+:32]

   address = 32'h00008054; data = instruction2[5*32+:32]

   address = 32'h00008050; data = instruction2[4*32+:32]

   address = 32'h0000804c; data = instruction2[3*32+:32]

   address = 32'h00008048; data = instruction2[2*32+:32]

address = 32'h00008044; data = instruction2[1*32+:32]

address = 32'h00008040; data = instruction2[0*32+:32]

4. Write 1 to the TG_START register to start the traffic:

   Address: 00004004; Data: 0000_0001

Refer to Address Decoding for Synthesizable TG for more information on these registers

Design Flow Steps

This section describes customizing and generating the Performance AXI Traffic Generator, constraining the Performance AXI Traffic Generator, and the simulation, synthesis, and implementation steps that are specific to this IP Performance AXI Traffic Generator. More detailed information about the standard 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 Xilinx® tools to customize and generate the Performance AXI Traffic Generator and the Simulation Trigger for NoC AXI TG in the Vivado® Design Suite.

If you are customizing and generating the Performance AXI Traffic Generator 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 Performance AXI Traffic Generator using the following steps:

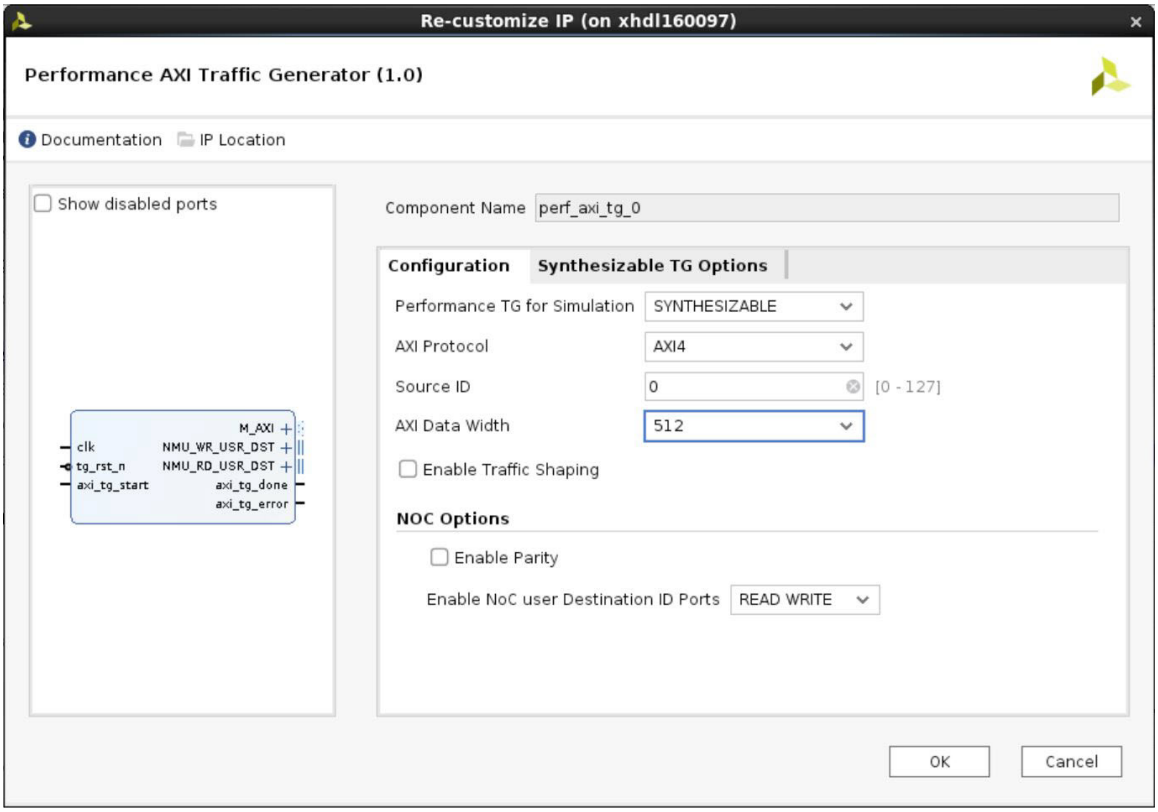
1. Select the IP from the IP catalog.
2. Double-click the selected IP or select the Customize IP command from the toolbar or right-click menu.

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.

Configuration Tab

Figure 14: **Configuration Tab**



- **Component Name:**
- **Performance TG for Simulation:**

  Synthesizable

  Non-synthesizable
- **AXI protocol:**

  AXI3

  AXI4

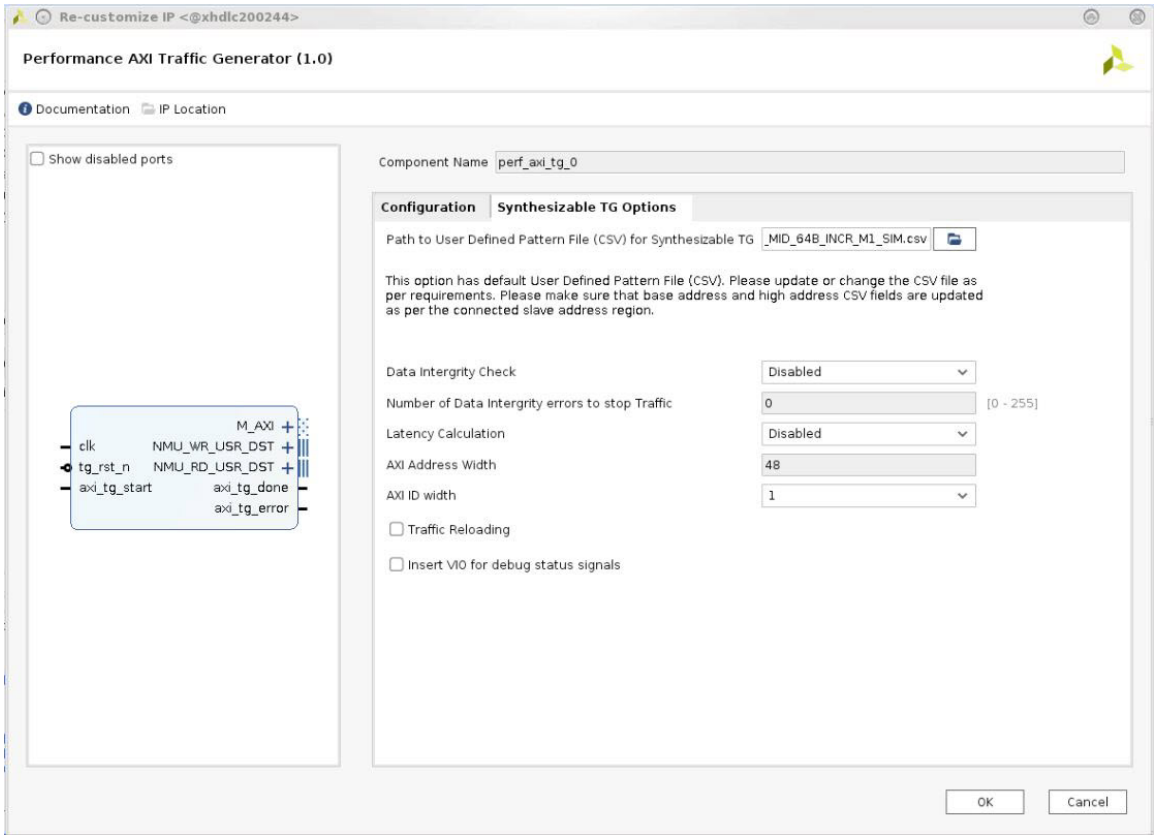
  AXI4-Stream

  Note: AXI3 is only available for the Non-Synthesizable TG.

- **Source ID:** The TG number must be specified here. Supports 0 to 127.
- **AXI Data Width:** Supported widths are 32, 64, 128, 256, and 512.
- **Enable Traffic Shaping:** Enables the traffic shaping module in the TG. The `trigger_out` and `trigger_in` ports appear on the TG interface.
- **Enable Even Parity:** When enabled, even parity for AXI address and Write data is generated and sent on the `AxUSER` and `WUSER` signals respectively.
- **Enable NoC User Destination ID Ports:** NONE, READ, WRITE, READ_WRITE

Synthesizable TG Options Tab

Figure 15: **Synthesizable TG Options Tab**



- **Path to User Defined Pattern File (CSV) for Synthesizable TG:** The location of the CSV file must be added here.
- **Data Integrity:**

  Disabled

  Enabled with Constant ID

Enabled with Incremental ID

- **Number of Data Integrity errors to stop Traffic:** Values supported are 0 to 256 in which 0 means never stop traffic.
- **Latency Calculation:**

  Disabled

  Enabled with Constant ID

  Enabled with Incremental ID

- **AXI Address Width:** 48 bits
- **AXI ID Width:**

  Up to 16 when Data Integrity is disabled

  Up to 4 with Incremental ID

  1 for Constant ID

- **Traffic Reloading:** Enables the register I/O interface to access the TG register space and BRAM instruction loading. Enabling this feature requires a Simulation Trigger IP in your design.
- **Debug core:** VIO for debug status signals. Enables VIO debug monitoring for status signals.

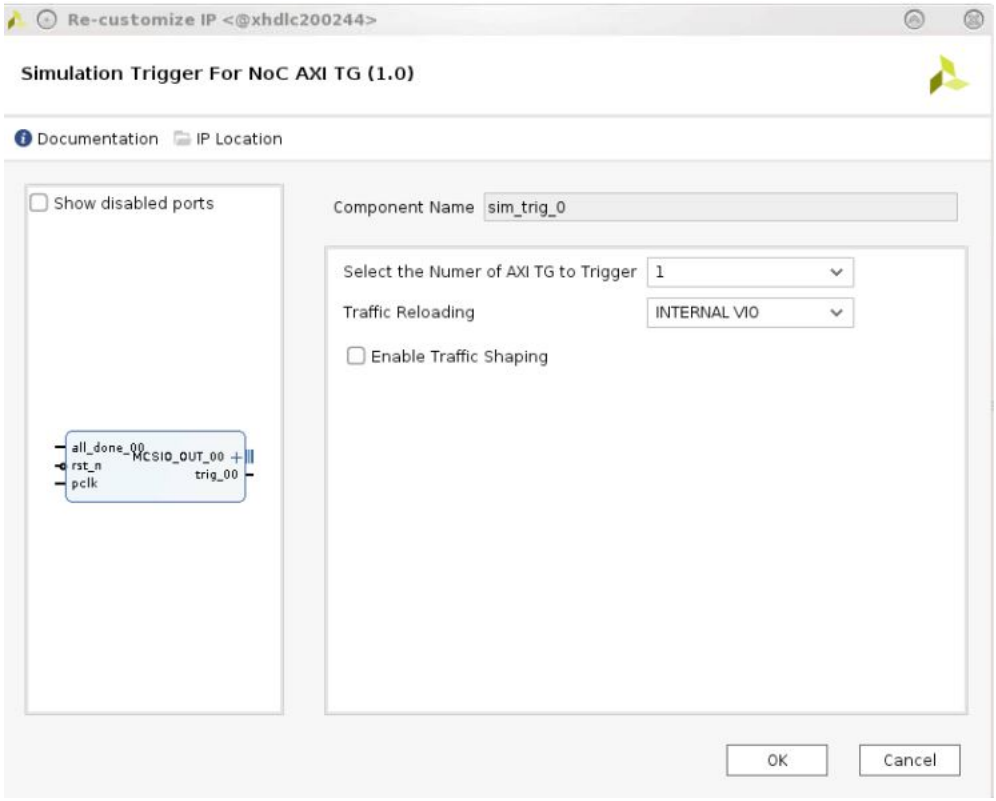
Simulation Trigger for the NoC AXI TG IP

This section includes information about using Xilinx tools to customize and generate the simulation trigger for the NoC AXI TG IP in the Vivado Design Suite. The Simulation Trigger IP can be used with both the Synthesizable and Non-Synthesizable TG. To customize the simulation trigger for the NoC AXI TG IP, perform the following initial steps:

1. Select the IP from the IP catalog in the IP integrator window.
2. Double-click on the selected IP.

Simulation Trigger Component Configuration Tab

Figure 16: **Simulation Trigger Component Configuration Tab**



- **Select the Number of AXI TG to trigger:** Set this value to the number of TGs used in the system design.
- **Traffic Reloading:** To reload the traffic on hardware after running default traffic, this option is required. Traffic reloading for thePerformance AXI Traffic Generator is possible only with this option. The available settings are as follows:
  - **NONE:** Select this option if there is no traffic reloading for the TG.
  - **INTERNAL_VIO:** Select this option to reload traffic for TG with the help of the VIO debug core.
  - **EXTERNAL_AXI4_LITE:** Select this option to reload traffic across the TG with the help of AXI4-Lite.

    In a system-level design, if the Traffic Reloading option selected is INTERNAL_VIO or EXTERNAL_AXI4_LITE in Simulation Trigger for the NoC AXI TG IP, you should also enable the Traffic Reloading option in Performance AXI Traffic Generator cores. Refer to Hardware Access for more information about traffic reloading.

- **Enable Traffic Shaping:** Select this option for doing traffic shaping between Performance AXI Traffic Generator cores in a system design. In a system-level design, if this option is selected in Simulation Trigger for NoC AXI TG IP, you should also enable it in Performance AXI Traffic Generator cores. Refer to Synchronization of Multiple TGs for more information.

Output Generation

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

CSV Format for Multiple TGs

There are two methods for providing a CSV file input to the TG when there is more than one TG in the system:

- A dedicated CSV file as input for each TG
- A common CSV file as input for all TGs

Dedicated CSV for Each TG

In this scenario, there are two TGs (AXI masters), and each is assigned a CSV file in its configuration GUI. TG1 has a source ID of 1 and TG2 has a source ID of 2..

Figure 17: **Dedicated CSV for Each TG**

TG1 executes the CSV lines where `TG_NUM` is 1. TG1 also executes the CSV lines where `TG_NUM` is left empty.

TG2 executes the CSV lines where `TG_NUM` is 2. TG2 also executes the CSV lines where `TG_NUM` is left empty.

Figure 18: **axi_design_a_tg1.csv (TG_NUM = 1)**

TG_NUM	cmd	txn_count	start_delay	inter_beat_delay	wdata_pattern	wdata_pat_value	data_integrity	dest_id	base_addr	high_addr	addr_inc_by	axi_addr_offset
1	WRITE	5	0	0	random	0	enabled		0000_0000	0000_FFFF	auto_incr	0
1	READ	10	0	0	random	0	enabled		0000_0000	0001_FFFF	auto_incr	0

Figure 19: **axi_design_a_tg1.csv (TG_NUM Empty)**

TG_NUM	cmd	txn_count	start_delay	inter_beat_delay	wdata_pattern	wdata_pat_value	data_integrity	dest_id	base_addr	high_addr	addr_inc_by	axi_addr_offset
	WRITE	5	0	0	random	0	enabled		0000_0000	0000_FFFF	auto_incr	0
	READ	10	0	0	random	0	enabled		0000_0000	0001_FFFF	auto_incr	0

Figure 20: **axi_design_a_tg2.csv (TG_NUM = 2)**

TG_NUM	cmd	txn_count	start_delay	inter_beat_delay	wdata_pattern	wdata_pat_value	data_integrity	dest_id	base_addr	high_addr	addr_inc_by	axi_addr_offset
2	WRITE	5	0				enabled		0001_0000	0001_FFFF	auto_incr	0
2	READ	10	0	0	random	0	enabled		0000_0000	0001_FFFF	auto_incr	0

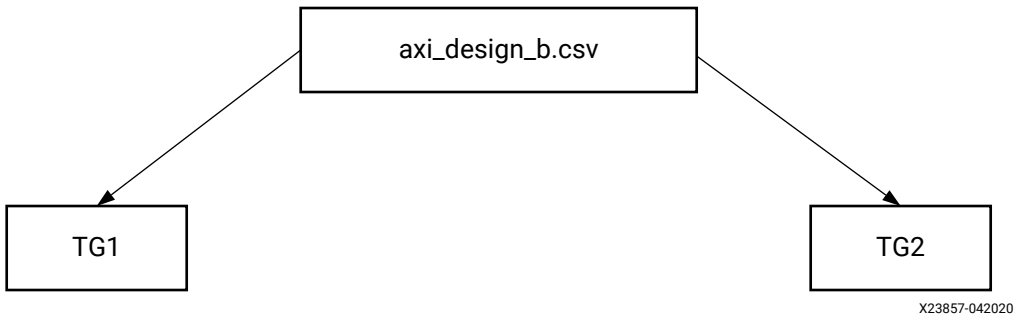
Figure 21: **axi_design_a_tg2.csv (TG_NUM Empty)**

TG_NUM	cmd	txn_count	start_delay	inter_beat_delay	wdata_pattern	wdata_pat_value	data_integrity	dest_id	base_addr	high_addr	addr_inc_by	axi_addr_offset
	WRITE	5	0				enabled		0001_0000	0001_FFFF	auto_incr	0
	READ	10	0	0	random	0	enabled		0000_0000	0001_FFFF	auto_incr	0

Common CSV Input for All TGs

In this scenario, there are two Traffic Generators (AXI masters) using a common CSV file as input. TG1 has a source ID of 1 and TG2 has a source ID of 2. The traffic is generated by the TG by using `TG_NUM` value field.

Figure 22: **Common CSV Input for All TGs**



Both TG1 and TG2 read the CSV file in the following figure. The TG parser reads the `TG_NUM` value on each CSV line and matches it with the source ID of the TG. In a given line, if the `TG_NUM` matches with source ID or the `TG_NUM` field is left empty, the CSV line is executed by the TG.

Figure 23: **Common CSV Example**

TG_NUM	cmd	txn_count	start_delay	inter_beat_delay	wdata_pattern	wdata_pat_value	data_integrity	dest_id	base_addr	high_addr	addr_inc_by	axi_addr_offset
1	WRITE	5	0	0	random	0	enabled		0000_0000	0000_FFFF	auto_incr	0
2	WRITE	5	0				enabled		0001_0000	0001_FFFF	auto_incr	0
	READ	10	0	0	random	0	enabled		0000_0000	0001_FFFF	auto_incr	0

- CSV line 2 (`TG_NUM` = 1) is executed only by TG1. TG2 skips this line from execution because it does not match the source ID of TG2.

- CSV line 3 (`TG_NUM` = 2) is executed only by TG2. TG1 skips this line from execution because it does not match the source ID of TG1.
- CSV line 4 (`TG_NUM` left empty) is executed by both TG1 and TG2 because the `TG_NUM` field is left empty

The following lines are executed by TG1.

Figure 24: **TG1 Parser Output**

TG_NUM	cmd	txn_count	start_delay	inter_beat_delay	wdata_pattern	wdata_pat_value	data_integrity	dest_id	base_addr	high_addr	addr_inc_by	axi_addr_offset
1	WRITE	5	0	0	random	0	enabled		0000_0000	0000_FFFF	auto_incr	0
1	READ	10	0	0	random	0	enabled		0000_0000	0000_FFFF	auto_incr	0

The following lines are executed by TG2.

Figure 25: **TG2 Parser Output**

TG_NUM	cmd	txn_count	start_delay	inter_beat_delay	wdata_pattern	wdata_pat_value	data_integrity	dest_id	base_addr	high_addr	addr_inc_by	axi_addr_offset
2	WRITE	5	0				enabled		0001_0000	0001_FFFF	auto_incr	0
2	READ	10	0	0	random	0	enabled		0000_0000	0001_FFFF	auto_incr	0

Creating a custom CSV

The CSV file provides the description of the traffic that the performance AXI Traffic generator must generate. One default CSV is packaged as a part of the IP. After creating a functional block design and generating output products, you can locate the default CSV file in the following directory

```
./<project_name>.ip_user_files/mem_init_files/
default_aximm_synth_tg.csv.
```

You can copy this file to a custom location and modify it according to the type of traffic you want to generate. As a user, you can now use the custom CSV. Specify the custom CSV's path in the **Path to user defined pattern file (CSV)** option of the **Synthesizable TG option** tab in the GUI.

See AXI4 Traffic Pattern CSV Format for more information on CSV command format.

AXI4 Traffic Pattern CSV Format

WRITE Command

Table 203: **WRITE CSV Command Format for AXI4**

Command Input Field	Width	Synthesizable TG Options
txn_count	16	• **<value in decimal>:** Transaction count. Repeat the current command <N> times, where N = value. • **INF:** Repeat the current command infinitely.
start_delay	16	• **<value in decimal>:** Value in number of clocks. This specifies the delay between transactions.
inter_beat_delay	-	Not supported. This field is kept empty in the csv file.
wdata_pattern	STRING	• constant • same_as_addr • same_as_addr_xor • hammer ***Note*:** This field is "don't care" for the Synthesizable TG. To select the required data pattern the wdata_pat_value field must be set. See Table 211 for details about the patterns.
wdata_pat_value	9	• **<value in hex>:** If the most significant bit is 0 the next eight bits will be sent as data byte (for example 0x032, 0x0F2). Otherwise the three least signficant bits specify the data pattern (wdata_pattern: constant). • **0x100:** Address as data. (wdata_pattern: same_as_addr) • **0x101:** Byte XOR of address as data. (wdata_pattern: same_as_addr_xor) • **0x102:** hammer data (wdata_pattern: hammer)
data_integrity	STRING	Not supported for Writes.
dest_id	12	• **<value in hex>:** The targeted NoC slave destination ID value (12-bit value) can be given in this field for the transaction. The dest_id value will be sent on the nmu_wr_usr_dst signal along with AXI Write requests.

Table 203: **WRITE CSV Command Format for AXI4** *(cont'd)*

Command Input Field	Width	Synthesizable TG Options
base_addr	48	• **<value in hex>:** The base_addr can be specified in this field. The incremented address for the transactions will be looped back to the base address when the high address boundary is reached. In case of randomized address, the TG will generate the address between base_addr and high_addr.
high_addr	48	• **<value in hex>:** The high address can be specified here. Up to this value the address will be incremented by the TG. When the incremented address reaches the high address boundary, the next transaction start address will be the base_addr. In case of a randomized address, TG will generate the address between base_addr and high_addr.
addr_incr_by	48/STRING	• **<value in hex> :** This field value is considered as “value in hex” when the axi_addr_offset field is set to a start address offset value. The first transaction start address (SA) will be picked from the axi_addr_offset and base_addr fields (SA = base_addr + axi_addr_offset). From the second transaction onwards for each AXI transaction, the start address is incremented by the specified addr_incr_by value. For example: txn_count=4, base_addr=0x0_0000, high_addr=0xF_FFFFF, addr_incr_by=0x41, axi_addr_offset=0x01, axi_len=0, axi_size=4. 1st transaction: AWADDR=0x0_0001 2nd transaction: AWADDR=0x0_0042 3rd transaction: AWADDR=0x0_0083 4th transaction: AWADDR=0x0_00C4 • **<seed_value in hex>:** This field value is considered as seed for the PRBS module to generate random addresses when the axi_addr_offset field is set to random address options. • **auto_incr:** The first transaction start address (SA) will be picked from the axi_addr_offset and base_addr fields (SA = base_addr + axi_addr_offset). From the second transaction onwards for each AXI transaction issued, the start address is calculated by using the expression described below.

Table 203: **WRITE CSV Command Format for AXI4** *(cont'd)*

Command Input Field	Width	Synthesizable TG Options

addr_incr_by (continued)	48/STRING	• **For INCR_WRAP bursts:**
		  • **auto_incr:** The first transaction start address (SA) will be picked from the axi_addr_offset and base_addr fields (SA = base_addr + axi_addr_offset). From the second transaction onwards for each AXI transaction issued, the start address is calculated by using the following expression:
		  start_address (from second transaction onwards) = previous_start_address + ((2^axi_size) * (axi_len+1))
		  For example: txn_count= 3, base_addr= 0x0000_0000, high_addr= 0xFFFF_FFFF, axi_addr_offset= 0x0000_0000, axi_size= 6, axi_len= 0, axi_burst= 1(INCR).
		  The start addresses of all three transactions are as follows:
		  1st transaction start address: 0x0000_0000
		  2nd transaction start address: 0x0000_0000 + ((2^6)*(0+1)) = 0x0000_0040
		  3rd transaction start address: 0x0000_0040 + ((2^6)*(0+1)) = 0x0000_0080

addr_incr_by (continued)	48/STRING	• **For FIXED bursts:**
		  • **auto_incr:** The first transaction start address (SA) will be picked from the axi_addr_offset and base_addr fields (SA = base_addr + axi_addr_offset). From the second transaction onwards for each AXI transaction issued, the start address is calculated by using the following expression:
		  start_address (from second transaction onwards) = previous_start_address + (2^axi_size)
		  For example, txn_count= 3, base_addr= 0x0000_0000, high_addr= 0xFFFF_FFFF, axi_addr_offset= 0x0000_0000, axi_size= 5, axi_len= 4, axi_burst= 0(FIXED).
		  The start addresses of all three transactions are as follows:
		  1st transaction start address: 0x0000_0000
		  2nd transaction start address: 0x0000_0000 + (2^5) = 0x0000_0020
		  3rd transaction start address: 0x0000_0020 + (2^5) = 0x0000_0040

Table 203: **WRITE CSV Command Format for AXI4** *(cont'd)*

Command Input Field	Width	Synthesizable TG Options
axi_addr_offset	48/STRING	• **<value in hex>:** This is the offset value added to base_addr to calculate the start address of the first transaction. From the second transaction onwards, the start address is calculated based on the addr_incr_by field option. If the next transaction address reaches the high_addr boundary it will be looped back to base_addr. • **random:** Any random address is generated between base_addr and high_addr. The addr_incr_by field value is set as seed to the PRBS module. • **random_aligned:** Any random aligned address is generated between base_addr and high_addr. The addr_incr_by field value is set as seed to the PRBS module.
axi_len	8	• **<value in hex>:** Write transactions will be sent with the specified AXI length value. Width: 4 for AXI3, 8 for AXI4. ***Note*:** For AXI3, MAX_LEN = 0xF. For AXI4, MAX_LEN varies depending on AWSIZE due to the 4k Boundary limit: • When AWSIZE is 64B, MAX_LEN is 0x3F. • When AWSIZE is 32B, MAX_LEN is 0x7F. • When AWSIZE is 16, 8, 4, 2 or 1B, MAX_LEN is 0xFF.
axi_size	3	• **<value in hex>:** Write transactions will be sent with the specified AXI size value.
axi_id	16/STRING	• **<value in hex>:** Write transactions will be sent with the specified AXI ID value. • **auto_incr:** The first Write transaction will be sent out with an ID of 0x0 and each consecutive transaction ID will be incremented by 0x1. When the ID reaches MAX_ID value, it will start again from 0x0 and increment until MAX_ID is reached again, and so on. Width: 1 to 4 if data integrity is enabled. 1 to 16 if data integrity is disabled. ***Note*:** MAX_ID = 2^AXI ID Width - 1
axi_burst	2/STRING	• **<value in hex>:** Supported values are 0x0, 0x1, 0x2. • **<value in string>:** Supported values are FIXED, INCR, WRAP.

Table 203: **WRITE CSV Command Format for AXI4** *(cont'd)*

Command Input Field	Width	Synthesizable TG Options
axi_lock	2/STRING	• **<value in hex>:** Supported values are 0x0, 0x1. • **<value in string>:** Supported values are NORMAL, EXCLUSIVE.
axi_cache	4	<value in hex>
axi_prot	3	<value in hex>
axi_qos	4	<value in hex>
axi_region	4	<value in hex>
axi_user (for AWUSER)	10	<value in hex>
exp_resp	STRING	• OKAY • EXOKAY • SLVERR • DECERR • Blank/empty Field

READ Command

Table 204: **READ CSV Command Format for AXI4**

Command Input Field	Width	Synthesizable TG Options
txn_count	16	• **<value in decimal>:** Transaction count. Repeat the current command <N> times, where N = value. • **INF:** Repeat the current command infinitely.
start_delay	16	• **<value in decimal>:** Value in number of clocks. This specifies the delay between transactions.
inter_beat_delay	N/A	N/A. This field is kept empty in the CSV .
wdata_pattern	STRING	This field is "don’t care" for the Synthesizable TG. The synthesizable TG uses wdata_pat_value for calculating expected read data

Table 204: **READ CSV Command Format for AXI4** *(cont'd)*

Command Input Field	Width	Synthesizable TG Options
wdata_pat_value	9	• **<value in hex>:** If the most significant bit is 0 the next eight bits is sent as data byte (for example 0x032, 0x0F2). Otherwise the three least signficant bits specify the data pattern (wdata_pattern: constant). • **0x100:** Address as data. (wdata_pattern: same_as_addr) • **0x101:** Byte XOR of address as data. (wdata_pattern: same_as_addr_xor) • **0x102:** hammer data (wdata_pattern: hammer) ***Note*:** This field should be set to the expected Read data pattern when data_integrity is enabled.
data_integrity	STRING	• **enabled:** Data integrity checks are enabled for the Read transaction. Whenever Read data arrives the TG compares the read data with the expected data that is internally generated based on the data_pattern set on wdata_pat_value. • **disabled:** Data integrity checks are disabled for this Read transaction.
dest_id	12	• **<value in hex>:** The targeted NoC slave destination ID value (12-bit value) can be given in this field for the transaction. The dest_id value is sent on the nmu_rd_usr_dst signal along with AXI Read requests.
base_addr	48	• **<value in hex>:** The base_addr can be specified in this field. The incremented address for the transactions are looped back to the base address when the high address boundary is reached. In case of randomized address, the TG is generated the address between base_addr and high_addr.
high_addr	48	• **<value in hex>:** The high address can be specified here. Up to this value the address is incremented by the TG. When the incremented address reaches the high address boundary, the next transaction start address is the base_addr. In case of a randomized address, TG are generated the address between base_addr and high_addr.

Table 204: **READ CSV Command Format for AXI4** *(cont'd)*

Command Input Field	Width	Synthesizable TG Options
addr_incr_by	48	• **<value in hex> :** This field value is considered as "value in hex" when the axi_addr_offset field is set to a start address offset value. The first transaction start address (SA) is picked from the axi_addr_offset and base_addr fields (SA = base_addr + axi_addr_offset). From the second transaction onwards for each AXI transaction, the start address is incremented by the specified addr_incr_by value. For example: txn_count=4, base_addr=0x0_0000, high_addr=0xF_FFFFF, addr_incr_by=0x41, axi_addr_offset=0x01, axi_len=0, axi_size=4. 1st transaction: ARADDR=0x0_0001 2nd transaction: ARADDR=0x0_0042 3rd transaction: ARADDR=0x0_0083 4th transaction: ARADDR=0x0_00C4 • **<seed_value in hex>:** This field value is considered as seed for the PRBS module to generate random addresses when the axi_addr_offset field is set to random address options. • **auto_incr:** The first transaction start address (SA) is picked from the axi_addr_offset and base_addr fields (SA = base_addr + axi_addr_offset). From the second transaction onwards for each AXI transaction issued, the start address is calculated by using the expression described below.
addr_incr_by (continued)	48	• **INCR/WRAP Burst:** • **auto_incr:** The first transaction start address (SA) will be picked from the axi_addr_offset and base_addr fields (SA = base_addr + axi_addr_offset). From second transaction onwards for each AXI transaction issued, the start address is calculated using the following expression: start_address (from second transaction onwards) = previous_start_address + ((2^axi_size) * (axi_len+1)) For example: txn_count= 3, base_addr= 0x0000_0000, high_addr= 0xFFFF_FFFF, axi_addr_offset= 0x0000_0000, axi_size= 6, axi_len= 0, axi_burst= 1(INCR). The start addresses of all three transactions are as follows: 1st transaction start address: 0x0000_0000 2nd transaction start address: 0x0000_0000 + ((2^6)*(0+1)) = 0x0000_0040 3rd transaction start address: 0x0000_0040 + ((2^6)*(0+1)) = 0x0000_0080

Table 204: **READ CSV Command Format for AXI4** *(cont'd)*

Command Input Field	Width	Synthesizable TG Options
addr_incr_by (continued)	48	• **FIXED Burst:** • **auto_incr:** The first transaction start address (SA) is picked from the axi_addr_offset and base_addr fields (SA = base_addr + axi_addr_offset). From second transaction onwards for each AXI transaction issued, the start address is calculated using the following expression: start_address (from second transaction onwards) = previous_start_address + (2^axi_size) For example, txn_count= 3, base_addr= 0x0000_0000, high_addr= 0xFFFF_FFFF, axi_addr_offset= 0x0000_0000, axi_size= 5, axi_len= 4, axi_burst= 0(FIXED). The start addresses of all three transactions are as follows: 1st transaction start address: 0x0000_0000 2nd transaction start address: 0x0000_0000 + (2^5) = 0x0000_0020 3rd transaction start address: 0x0000_0020 + (2^5) = 0x0000_0040
axi_addr_offset	48	• **<value in hex>:** This is the offset value added to base_addr to calculate the start address of the first transaction. From the second transaction onwards, the start address is calculated based on the addr_incr_by field option. If the next transaction address reaches the high_addr boundary it is looped back to base_addr. • **random:** Any random address is generated between base_addr and high_addr. The addr_incr_by field value is set as seed to the PRBS module. • **random_aligned:** Any random aligned address is generated between base_addr and high_addr. The addr_incr_by field value is set as seed to the PRBS module.
axi_len	8	• **<value in hex>:** Read transactions are sent with the specified AXI length value. Width: 4 for AXI3, 8 for AXI4. ***Note*:** For AXI3, MAX_LEN = 0xF. For AXI4, MAX_LEN varies depending on ARSIZE due to the 4k Boundary limit: • When ARSIZE is 64B, MAX_LEN is 0x3F. • When ARSIZE is 32B, MAX_LEN is 0x7F. • When ARSIZE is 16, 8, 4, 2 or 1B, MAX_LEN is 0xFF.
axi_size	3	• **<value in hex>:** Read transactions are sent with the specified AXI size value.

Table 204: **READ CSV Command Format for AXI4** *(cont'd)*

Command Input Field	Width	Synthesizable TG Options
axi_id	16/STRING	• **<value in hex>:** Read transactions are sent with the specified AXI ID value. • **auto_incr:** The first Read transaction is sent out with an ID of 0x0 and each consecutive transaction ID is incremented by 0x1. When the ID reaches MAX_ID value, it starts again from 0x0 and increment until MAX_ID is reached again, and so on. Width: 1 to 4 if data integrity is enabled. 1 to 16 if data integrity is disabled. ***Note*:** MAX_ID = 2^AXI ID Width - 1
axi_burst	2/STRING	• **<value in hex>:** Supported values are 0x0, 0x1, 0x2. • **<value in string>:** Supported values are FIXED, INCR, WRAP.
axi_lock	2/STRING	• **<value in hex>:** Supported values are 0x0, 0x1. • **<value in string>:** Supported values are NORMAL, EXCLUSIVE.
axi_cache	4	<value in hex>
axi_prot	3	<value in hex>
axi_qos	4	<value in hex>
axi_region	4	<value in hex>
axi_user	10	<value in hex>
exp_resp	STRING	• OKAY • EXOKAY • SLVERR • DECERR • Blank/Empty Field

WAIT Command

Table 205: **Wait Command Format for AXI4**

Command Input Field	Width	Synthesizable TG Options
wait_option	STRING	• **all_wr_rd_resp:** Wait till all Write and Read responses are received.
wait_unit	-	Not Supported

START_LOOP/END_LOOP

Table 206: **LOOP CSV Command Format for AXI4**

Command Input Field	Width	Synthesizable TG Options
loop_count	16	• **<value in decimal>:** Repeats the loop the specified number of times.
loop_operation	STRING	• **incr_original_addr:** While executing the loop each time, the address offset of the WRITE and READ instructions inside the LOOP command is incremented by the specified value in the loop_addr_incr_by field. When the incremented value goes out of range, the TG restarts the address from base_addr.
loop_addr_incr_by	16	• **<value in hex>:** Specifies the address offset increment by value for each loop.

PHASE_DONE Command

The phase done command is used for traffic shaping across multiple TG. PHASE_DONE command execution occurs in following steps

1. Waits to receive all write and read responses.
2. Asserts the trigger_out signal.
3. Waits for trigger_in assertion by the simulation Trigger IP. The trigger_in is asserted when trigger_out from all the TGs is received.
4. De-asserts trigger_out signal.

The PHASE_DONE command, upon its execution, ensures the traffic across multiple TG gets synchronized.

***Note*:** The PHASE_DONE command does not have any command input fields.

SET_DEFAULT and DISPLAY Commands

***Note*:** The SET_DEFAULT and DISPLAY commands are not supported in the Synthesizable TG.

AXI4-Stream Traffic Pattern CSV

STREAM Command

Table 207: **STREAM CSV Command Format for AXI4-Stream**

Command Input Field	Width	Synthesizable TG Options
pkt_count	16	• **<value in decimal>:** Stream packet count. Repeat the current command <N> times, where N = value. • **INF:** Repeat the current command infinitely.
inter_pkt_delay	16	• **<value in decimal>:** Value in number of clocks. Issue transaction after specified number of clocks.
inter_transfer_delay	16	• **<value in decimal>:** Value in number of clocks. Adds specified number of clocks as delay between two transfers in a stream packet. ***Note*:** The maximum ax delay value supported is 65530.
tdata_pattern	STRING	• constant • random • hammer • byte_incr • 16byte_incr
tdata_pat_value	32	• **<value in hex>:** When the tdata_pattern is selected as constant, this field value is used as constant data which is sent on all Stream packet data transfers (tdata_pattern: constant). • **<seed_value in hex>:** When the tdata_pattern is selected as random, this field value is used as seed for the PRBS (pseudorandom binary sequence) module (tdata_pattern: random).
Reserved	-	-
noc_dest_id	12	• **<value in hex>:** The targeted NoC slave destination ID value (12-bit value) can be given in this field for the Stream packet. ***Note*:** The noc_dest_id value is sent on the nmu_wr_usr_dst signal along with AXI4-Stream packets.
Reserved	-	-

Table 207: **STREAM CSV Command Format for AXI4-Stream** *(cont'd)*

Command Input Field	Width	Synthesizable TG Options
Reserved	-	-
Reserved	-	-
tdest_id	12	• **<value in hex>:** This value is sent on the axis_tdest signal for all Stream packets.
pkt_len	16	• **<value in hex>:** Stream packet transactions is sent with N transfer where, N= length value in this field. ***Note*:** If pkt_len = 0, axis_tlast is always zero. Otherwise the axi_tlast signal is asserted High for every final transfer in the packet.
Reserved	-	-
pkt_id	STRING/16	• **<value in hex>:** All Stream packet transactions is sent with this ID. • **auto_incr:** The first Stream packet transaction is sent out with an ID of 0 and each consecutive transaction packet ID is incremented by 1. When the packet ID reaches the MAX_ID, the next transaction ID starts from 0 and increments again. ***Note*:** MAX_ID = 2^ AXI ID Width) – 1
Reserved	-	-
Reserved	-	-
Reserved	-	-
Reserved	-	-
Reserved	-	-
Reserved	-	-
pkt_user	16	• **<value in hex>:** This value is sent in the axis_tuser signal for all Stream packets.

WAIT Command

Table 208: **WAIT CSV Command Format for AXI4-Stream**

Command Input Field	Width	Synthesizable TG Options
wait_option	16	• **<wait_period by clocks, value in decimal>:** Wait by specified number of AXI clocks.. For example, if you want to wait for 50 AXI clock cycles, set 50 in this field.
wait_unit	-	Not supported

START_LOOP/END_LOOP

Table 209: **LOOP CSV Command Format for AXI4-Stream**

Command Input Field	Width	Synthesizable TG Options
loop_count	16	• **<value in decimal>:** Repeats the loop the specified number of times.

PHASE_DONE Command

***Note*:** The PHASE_DONE command does not have any command input fields.

SET_DEFAULT and DISPLAY Commands

***Note*:** The SET_DEFAULT and DISPLAY commands are not supported in the Synthesizable TG.

CSV Command Usage Examples forAXI4

Wait Command

This CSV has Write, Wait, and Read commands with a transaction count of 1000 for Write and Read. The start delay is 0 . The field `wdata_pat_value` is 0, that means user data 00000000. Data integrity is enabled. The base address and high address are `0xbc16_1000_0000` and `0xbc1e_FFFF_FFFF` respectively. The field `addr_incr_by` is `auto_incr` which indicates auto increment of address.

The Wait command after the Write command holds the next instruction until all the Write instructions are finished and responses are received. The next instruction is then issued, which is Read in this example.

Figure 26: **AXI3/AXI4 Wait Command CSV Example**

#Any line starting with # will be commented out

TG_NUM	cmd	txn_count	start_dela	inter_bea	wdata_pa	wdata_	data_inte	dest_id	base_addı	high_addr	addr_inc_	axi_addr	axi_len	axi_bur	axi_lock	axi_ca	axi_pr	axi_qc	axi_regi	axi_user
1	WRITE	1000	0	0	same_as_	0	enabled	0	bc1e_100(	bc1e_FFFF	auto_incr	0x0001_00(	A	FIXED	NORMAL	0	0	0	0	0
1	WAIT																			
1	READ	1000	0	0	same_as_	0	enabled	0	bc1e_100(	bc1e_FFFF	auto_incr	0x0001_00(	A	FIXED	NORMAL	0	0	0	0	0

Write Command

This CSV has a single Write command with a transaction count of 1000. The start delay is 0 . The field `wdata_pat_value` is 0 which means user data 00000000. The base address and high address are `0xbc16_1000_0000` and `0xbc1e_FFFF_FFFF` respectively. The field `addr_incr_by` is `auto_incr` which indicates auto increment of address.

Figure 27: **AXI3/AXI4 Write Command CSV Example**

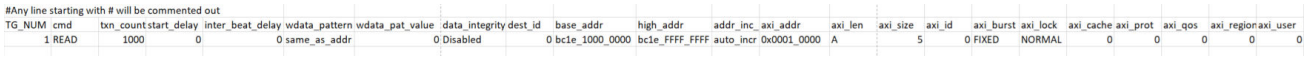
#Any line starting with # will be commented out

TG_NUM	cmd	txn_count	start_dela	inter_beat_	wdata_pattern	wdata_pat_value	data_integ	dest_id	base_addr	high_addr	addr_inc_by	axi_addr	axi_len	axi_size	axi_id	axi_burst	axi_lock	axi_cache	axi_prot	axi_qos	axi_region	axi_user
1	WRITE	1000	0	0	same_as_addr	0	enabled	0	bc1e_1000_0000	bc1e_FFFF_FFFF	auto_incr	0x0001_0000	A	5	0	FIXED	NORMAL	0	0	0	0	0

Read Command

This CSV has a single Read command with a transaction count of 1000. The start delay is 0. The field `wdata_pat_value` is 0 which means user data 00000000. Data integrity is disabled. The base address and high address are `0xbc16_1000_0000` and `0xbc1e_FFFF_FFFF` respectively. The field `addr_incr_by` is `auto_incr` which indicates auto increment of address.

Figure 28: **AXI3/AXI4 Read Command Example**



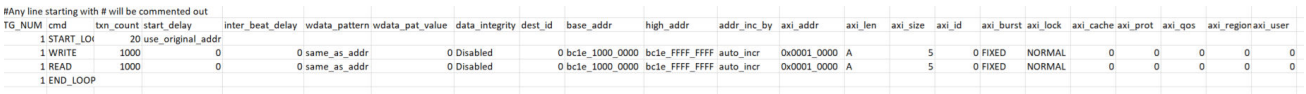
#Any line starting with # will be commented out

TG_NUM	cmd	txn_count	start_delay	inter_beat_delay	wdata_pattern	wdata_pat_value	data_integrity	dest_id	base_addr	high_addr	addr_inc_	axi_addr	axi_len	axi_size	axi_id	axi_burst	axi_lock	axi_cache	axi_prot	axi_qos	axi_region	axi_user
1	READ	1000	0	0	same_as_addr	0	Disabled	0	bc1e_1000_0000	bc1e_FFFF_FFFF	auto_incr	0x0001_0000	A	5	0	FIXED	NORMAL	0	0	0	0	0

Loop Command (START_LOOP/END_LOOP)

This CSV has Write and Read commands with a transaction count of 1000 each and a Loop command with a loop count of 20. The start delay is 0. The START_LOOP/END_LOOP instruction does the loop operation. The instructions that are defined between START_LOOP and END_LOOP instruction are executed in a given order in loop by the number of times mentioned on loop_count variable. On each loop, the command input field value of associated instructions are updated based on other loop variables, if applicable. This CSV has Write and Read commands with a transaction count of 1000 each and a Loop command with a loop count of 20. So, 1000 write transaction and 1000 read transaction are carried out 20 times.

Figure 29: **AXI3/AXI4 Loop Command CSV Example**



#Any line starting with # will be commented out

TG_NUM	cmd	txn_count	start_delay	inter_beat_delay	wdata_pattern	wdata_pat_value	data_integrity	dest_id	base_addr	high_addr	addr_inc_by	axi_addr	axi_len	axi_size	axi_id	axi_burst	axi_lock	axi_cache	axi_prot	axi_qos	axi_region	axi_user
1	START_LO	20	use_original_addr																			
1	WRITE	1000	0	0	same_as_addr	0	Disabled	0	bc1e_1000_0000	bc1e_FFFF_FFFF	auto_incr	0x0001_0000	A	5	0	FIXED	NORMAL	0	0	0	0	0
1	READ	1000	0	0	same_as_addr	0	Disabled	0	bc1e_1000_0000	bc1e_FFFF_FFFF	auto_incr	0x0001_0000	A	5	0	FIXED	NORMAL	0	0	0	0	0
1	END_LOOP																					

Infinite Loop

This CSV has Write and Read commands, each with a transaction count of 1000, with infinite loop. The start delay is 0. The field `wdata_pat_value` is 100, that means the user data is the same as the address. Data integrity is enabled. The base address and high address are `0x208_0000_0000` and `0x208_FFFF_FFFF` respectively. The field `addr_incr_by` is `auto_incr` which indicates auto increment of address. The Write and Read commands (each with a transaction count of 1000 for each run) repeat infinitely.

Figure 30: **AXI3/AXI4 Infinite Command CSV Example**

TG_NMU	cmd	txn_count	start_dela	inter_beat	wdata_pat	wdata_pat	data_integ	dest_id	base_addr	high_addr	addr_incr_	axi_addr	axi_len	axi_size	axi_id	aci_burst	axi_lock	axi_cache	axi_prot	axi_qos	axi_region	axi_user	exp_resp
0	START_LO(	INF																					
0	WRITE	1000	0	0	ADDR_AS_	100	enabled	0	0x2080000	0x2080000	auto_incr	0x0	0xff	0x20	0x0	INCR	NORMAL	0x2	0x0	0x0	0x0	0x0	OKAY
0	READ	1000	0	0	ADDR_AS_	100	enabled	0	0x2080000	0x2080000	auto_incr	0x0	0xff	0x20	0x1	INCR	NORMAL	0xe	0x7	0xa	0x1	0x5ea1	OKAY
0	WAIT	all_wr_rd_resp																					
0	END_LOOP																						

CSV Command Usage Examples for AXI4-Stream

Stream Command

- Issues 800 stream packets (`pkt_count` = 800), each with eight transfers (`pkt_len` = 8), containing HAMMER data to the slave that has a destination ID of 'h08 (`tdest_id` = 8).
- The packet user value for each transfer is 'hF (`pkt_user` =F).
- The packet ID value for each transfer is 'h8 (`pkt_user` = 8).

Figure 31: **AXI4-Stream Stream Command CSV Example**

#pkt_count -> decimal, inter_pkt_delay-> decimal, inter_transfer_delay -> decimal, #tdtestid-> in hex, pkt_len-> in hex, pkt_id -> in hex, pkt_usr -> hex

TG_NUM	cmd	pkt_count	inter_pkt_delay	inter_transfer_delay	tdata_pattern	tdata_pat_value	tdest_id	pkt_len	pkt_id	pkt_user
1	STREAM	800	0	0	HAMMER		0x8	0x8	0x8	0xf

Wait Command

The WAIT command inserts 10 clocks between the TLAST of the 800th packet of the command in line 1, and the assertion of TVALID for the first packet of the command in line 3.

Figure 32: **AXI4-Stream Wait Command CSV Example**

#pkt_count -> decimal, inter_pkt_delay-> decimal, inter_transfer_delay -> decimal, #tdestid-> in hex, pkt_len-> in hex, pkt_id -> in hex, pkt_usr -> hex

TG_NUM	cmd	pkt_count	inter_pkt_delay	inter_transfer_delay	tdata_pattern	tdata_pat_value	tdest_id	pkt_len	pkt_id	pkt_user
1	STREAM	800	0	0	hammer		0x8	0x8	0x8	0xF
1	WAIT	10								
1	STREAM	800	0	0	hammer		0x8	0x8	0x8	0xF

Loop Command (START_LOOP/END_LOOP)

The following figure shows 800 stream packets that is looped 100 times (`loop_count` = 100).

Figure 33: **AXI4-Stream Loop Command CSV Example**

TG_NUM	cmd	pkt_count	inter_pkt_delay	inter_transfer_delay	tdata_pattern	tdata_pat_value	tdest_id	pkt_len	pkt_id	pkt_user
1	START_LOOP	100								
1	STREAM	800	0	0	hammer		0x8	0x8	0x8	0xF
1	END_LOOP									

Data Patterns

Data patterns for AXI4

The following data patterns are supported by the Synthesizable TG for AXI4.

Table 210: **Data Patterns for the Synthesizable TG for AXI4**

Data Format	Options
Constant	• The user-defined data byte is constantly sent on all bytes of the Write data beat. • You can set the data byte value in the `wdata_pat_value` CSV field. • For example, if `wdata_pat_value` = 0x032, `AXI Data Width` = 64, and `axi_size` = 3, all the `data_beat` values of the given instruction will be sent as 0x3232_3232_3232_3232.
SAME_AS_ADDR	• The corresponding AXI beat byte address (LSB 8 bits) is sent as data in the bytes of the Write beat. • You can select the `same_as_addr` data pattern by setting the `wdata_pat_value` CSV field to 0x100. • For example, if `wdata_pat_value` = 0x100, `AXI Data Width` = 64, `axi_addr` = 0x0200_0000_11A0, `axi_burst` = INCR, `axi_len` = 3, and `axi_size` = 3, data beats are generated as follows: ◦ Byte_Lane0 address on Beat0= 0x0200_0000_11A0 --> LSB 8 bits are A0 which are sent as Write data on byte_lane0 of beat0. ◦ Byte_Lane1 address on Beat0= 0x0200_0000_11A1 --> LSB 8 bits are A1 which are sent as Write data on byte_lane1 of beat0. ◦ Beat0 = 0xA7A6_A5A4_A3A2_A1A0 ◦ Beat1 = 0xAFAE_ADAC_ABAA_A9A8 ◦ Beat2 = 0xB7B6_B5B4_B3B2_B1B0 ◦ Beat3 = 0xBFBE_BDBC_BBBA_B9B8

Table 210: **Data Patterns for the Synthesizable TG for AXI4** *(cont'd)*

Data Format	Options
SAME_AS_ADDR_XOR	• The corresponding AXI beat byte address is bytewise folded, XORed, and sent as data in the bytes of the Write beat. • You can select the `same_as_addr_xor` data pattern by setting the `wdata_pat_value` CSV field to 0x101. • For example, if `wdata_pat_value` = 0x101, `AXI Data Width` = 64, `axi_addr` = 0x0200_0000_11A0, `axi_burst` = INCR, `axi_len` = 3, and `axi_size` = 3, data beats are generated as follows: ◦ Byte_Lane0 address on Beat0= 0x0200_0000_11A0 --> 0x02^0x00^0x00^0x00^0x00^0x11^0xA0= 0x B3. It is sent as Write data on byte_lane0 of beat0. ◦ Byte_Lane1 address on Beat0= 0x0200_0000_11A1 --> 0x02^0x00^0x00^0x00^0x00^0x11^0xA1= 0x B2. It is sent as Write data on byte_lane1 of beat0. ◦ Beat0 = 0xB4B5_B6B7_B0B1_B2B3 ◦ Beat1 = 0xBCBD_BEBF_B8B9_BABB ◦ Beat2 = 0xA4A5_A6A7_A0A1_A2A3 ◦ Beat3 = 0xACAD_AEAF_A8A9_AAAB

Table 210: **Data Patterns for the Synthesizable TG for AXI4** *(cont'd)*

Data Format	Options
Hammer	***Note*:** Hammer data is only supported on full transfer AXI transactions for the Synthesizable TG. Selecting hammer data on a narrow transfer might result in unpredictable behavior. • Hammer data has a long number of tail bits (MSBs) with value 0 or 1 and a short number of header bits (LSBs) with an inverted bit value in a data beat. The values of the tail bits (width of ¾ of `axi_size_in_bits`) and header bits (width of ¼ of `axi_size_in_bits`) are generated based on the corresponding AXI beat address. If the resulting value of `beat_start_address` divided by `axi_size_in_bytes` is even the header bits are 1 and the tail bits are 0. If the resulting value is odd the header bits are 0 and tail bits are 1. • You can select the hammer data pattern by setting the `wdata_pat_value` CSV field to 0x102. For example, if `wdata_pat_value` = 0x102, `AXI Data Width` = 64, `axi_addr` = 0x0000_0000_11A5, `axi_burst` = INCR, `axi_len` = 3, and `axi_size` = 3, data beats are generated as follows: • `header_width` = `axi_size_in_bits`/4 = 64/4 = 16 • `tail_width` = (axi_size_in_bits*3)/4 = (64*3)/4 = 48 • Beat0 calculation: Aligned(`beat_start_address`)/`axi_size_in_bytes` = 0x0000_0000_11A0/8 = 0x234(even value) So, header bits are 16{{1'b1}} and tail bits are 48{{1'b0}}. Beat0 = 0x0000_0000_0000_FFFF • Beat1 calculation: Aligned(`beat_start_address`)/`axi_size_in_bytes` = 0x0000_0000_11A8/8 = 0x235(odd value) So, header bits are 16{{1'b0}} and tail bits are 48{{1'b1}}. Beat1 = 0xFFFF_FFFF_FFFF_0000 • Beat2 calculation: Aligned(`beat_start_address`)/`axi_size_in_bytes` = 0x0000_0000_11B0/8 = 0x236(even value) So, header bits are 16{{1'b1}} and tail bits are 48{{1'b0}}. Beat2 = 0x0000_0000_0000_FFFF • Beat3 calculation: Aligned(`beat_start_address`)/`axi_size_in_bytes` = 0x0000_0000_11B8/8 = 0x237(odd value) So, header bits are 16{{1'b0}} and tail bits are 48{{1'b1}}. Beat3 = 0xFFFF_FFFF_FFFF_0000

Data patterns for AXI4-Stream

The following data patterns are supported by the Synthesizable TG for AXI4-Stream:

Table 211: **Data Patterns for Synthesizable TG for AXI4-Stream**

Data Pattern	Option
CONSTANT DATA	• The user defined data bytes (max 32-bit wide) is constantly sent on all stream transfers. • The user can select the constant data pattern by setting tdata_pattern csv field to constant and can set the data byte value on tdata_pat_value csv field. • For example, if the tdata_pattern= constant, tdata_pat_value= 0x5556_5758 and AXI Data Width= 64, all the stream transfer values of the given instruction will be sent as 0x0000_0000_5556_5758.
RANDOM DATA	• The random data is sent on each stream transfer. The seed value can be set to generate different random patterns. • The user can select the random data pattern by setting tdata_pattern csv field to random and can set the seed value on tdata_pat_value csv field.
HAMMER DATA	• The hammer data has long number of tail bits (MSbits) with value of 0 or 1 and short number of header bits (LSbits) with inverted bit value in a stream transfer. In each stream packet, the value of tail bits (wide of ¾th of data_width_bits) and header bits (wide of ¼th of data_width_bits) of first stream transfer are 0 and 1 respectively. From the second transfer onwards, each stream transfer value is generated as inverse of previous stream transfer. • The user can select the hammer data pattern by setting the tdata_pattern csv field to hammer. • For example, if AXI Data Width= 64, pkt_cnt=2 and pkt_len=0x5, stream transfers are generated as below: ◦ header_width = data_width_bits/4 = 64/4 = 16 ◦ tail_width = (data_width_bits*3)/4 = (64*3)/4 = 48 • In each stream packet, the 1st stream transfer is calculated as header bits of 16{{1'b1}} and tail bits of 48{{1'b0}}. From 2nd transfer onwards, the previous tdata value are inverted on each transfer. TDATA = 0x0000_0000_0000_FFFF, TLAST = 0 TDATA = 0xFFFF _ FFFF _FFFF_0000, TLAST = 0 TDATA = 0x0000_0000_0000_FFFF, TLAST = 0 TDATA = 0xFFFF _ FFFF _FFFF_0000, TLAST = 0 TDATA = 0x0000_0000_0000_FFFF, TLAST = 1 TDATA = 0x0000_0000_0000_FFFF, TLAST = 0 TDATA = 0xFFFF _ FFFF _FFFF_0000, TLAST = 0 TDATA = 0x0000_0000_0000_FFFF, TLAST = 0 TDATA = 0xFFFF _ FFFF _FFFF_0000, TLAST = 0 TDATA = 0x0000_0000_0000_FFFF, TLAST = 1

Table 211: **Data Patterns for Synthesizable TG for AXI4-Stream** *(cont'd)*

Data Pattern	Option
BYTE_INCR DATA	• Every byte in the stream transfer has an incremented value. In each stream packet, the LSbyte value of first transfer is 0x00 and every other byte of transfers has an incremented value. • The user can select the byte_incr data pattern by setting tdata_pattern csv field to byte_incr. • For example, if AXI Data Width= 128, pkt_cnt=2 and pkt_len=0x3, stream transfers are generated as below: TDATA= 0x0F0E_0D0C_0B0A_0908_0706_0504_0302_0100, TLAST= 0 TDATA= 0x1F1E_1D1C_1B1A_1918_1716_1514_1312_1110, TLAST= 0 TDATA= 0x2F2E_2D2C_2B2A_2928_2726_2524_2322_2120, TLAST= 1 TDATA= 0x0F0E_0D0C_0B0A_0908_0706_0504_0302_0100, TLAST= 0 TDATA= 0x1F1E_1D1C_1B1A_1918_1716_1514_1312_1110, TLAST= 0 TDATA= 0x2F2E_2D2C_2B2A_2928_2726_2524_2322_2120, TLAST= 1

Table 211: **Data Patterns for Synthesizable TG for AXI4-Stream** *(cont'd)*

Data Pattern	Option
16BYTE_INCR DATA	• Every 16 byte in the stream transfer has an incremented value. In each stream packet, the value of Least significant 16 bytes of first transfer is 0x0 and every other set of 16 bytes of transfers has an incremented value. ***Note*:** 16byte_incr data pattern is only supported on 128-bit, 256-bit and 512-bit data width designs. • The user can select the 16byte_incr data pattern by setting tdata_pattern csv field to 16byte_incr. • Example1, if AXI Data Width= 128, pkt_cnt=2 and pkt_len=0x4, stream transfers are generated as below: TDATA= 0x0000_0000_0000_0000_0000_0000_0000_0000, TLAST= 0 TDATA= 0x0000_0000_0000_0000_0000_0000_0000_0001, TLAST= 0 TDATA= 0x0000_0000_0000_0000_0000_0000_0000_0002, TLAST= 0 TDATA= 0x0000_0000_0000_0000_0000_0000_0000_0003, TLAST= 1 TDATA= 0x0000_0000_0000_0000_0000_0000_0000_0000, TLAST= 0 TDATA= 0x0000_0000_0000_0000_0000_0000_0000_0001, TLAST= 0 TDATA= 0x0000_0000_0000_0000_0000_0000_0000_0002, TLAST= 0 TDATA= 0x0000_0000_0000_0000_0000_0000_0000_0003, TLAST= 1 • Example2, if AXI Data Width= 256, pkt_cnt=2 and pkt_len=0x4, stream transfers are generated as below: TDATA= 0x0000_0000_0000_0000_0000_0000_0000_0001_0000_0000_0000_0000_0000_0000_0000_0000, TLAST= 0 TDATA= 0x0000_0000_0000_0000_0000_0000_0000_0003_0000_0000_0000_0000_0000_0000_0000_0002, TLAST= 0 TDATA= 0x0000_0000_0000_0000_0000_0000_0000_0005_0000_0000_0000_0000_0000_0000_0000_0004, TLAST= 0 TDATA= 0x0000_0000_0000_0000_0000_0000_0000_0007_0000_0000_0000_0000_0000_0000_0000_0006, TLAST= 1 TDATA= 0x0000_0000_0000_0000_0000_0000_0000_0001_0000_0000_0000_0000_0000_0000_0000_0000, TLAST= 0 TDATA= 0x0000_0000_0000_0000_0000_0000_0000_0003_0000_0000_0000_0000_0000_0000_0000_0002, TLAST= 0 TDATA= 0x0000_0000_0000_0000_0000_0000_0000_0005_0000_0000_0000_0000_0000_0000_0000_0004, TLAST= 0 TDATA= 0x0000_0000_0000_0000_0000_0000_0000_0007_0000_0000_0000_0000_0000_0000_0000_0006, TLAST= 1

Constraining the Core

Required Constraints

This section is not applicable for this IP Performance AXI Traffic Generator.

Device, Package, and Speed Grade Selections

This section is not applicable for this IP Performance AXI Traffic Generator.

Clock Frequencies

This section is not applicable for this IP Performance AXI Traffic Generator.

Clock Management

This section is not applicable for this IP Performance AXI Traffic Generator.

Clock Placement

This section is not applicable for this IP Performance AXI Traffic Generator.

Banking

This section is not applicable for this IP Performance AXI Traffic Generator.

Transceiver Placement

This section is not applicable for this IP Performance AXI Traffic Generator.

I/O Standard and Placement

This section is not applicable for this IP Performance AXI Traffic Generator.

Simulation

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

The MEM file is automatically loaded to the block RAM during simulation. There is no need to load the MEM files manually.

Synthesis and Implementation

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

XILINX

Section III

Non-Synthesizable TG

Overview

The Non-Synthesizable Traffic generator is a simulation only traffic generator to generate AXI based traffic. It supports AXI3, AXI4, AXI4-Stream protocols. It generates data with higher configurability when compared to the synthesizable version. The Non-Synthesizable Traffic Generator supports additional features like protocol checker and bandwidth controller.

Modes of Operation

The modes of operating the TG are the same for AXI3, AXI4, and AXI4-Stream. When the Traffic Generator IP is generated the instructions can be loaded in following modes :

- CSV command format
- Fixed operating mode

CSV Command Format

In CSV command format all transactions are fed through a CSV file as shown in the following figures. The CSV file path is specified in the Vivado® IDE.

Figure 34: **CSV Format for AXI3/AXI4**

Any Line starts with # will be commented out

TG_NUM	cmd	txn_count	start_delay	inter_beat_delay	wdata_pattern	wdata_pat_value	data_integrity	dest_id	base_addr	high_addr	addr_incr_by	axi_addr	axi_len	axi_size	axi_id	axi_burst	axi_lock	axi_cache	axi_prot	axi_qos	axi_region	axi_user
1	START_LOOP	20	use_original_addr																			
1	WRITE	100	0	0	same_as_addr	0	enabled	0	5401_0000_0000	5401_FFFF_FFFF	auto_incr	0x0100_0000	7	5	0	WRAP	NORMAL	0	0	0	0	0
1	WAIT																					
1	READ	100	0				enabled	0	5401_0000_0000	5401_FFFF_FFFF	auto_incr	0x0100_0000	7	5	0	WRAP	NORMAL	0	0	0	0	0
1	WAIT																					
1	END_LOOP																					
1	START_LOOP	20	use_original_addr																			
1	WRITE	100	0	0	same_as_addr	0	enabled	0	4e_1000_0000	4e_FFFF_FFFF	auto_incr	0x0200_0000	F	5	0	WRAP	NORMAL	0	0	0	0	0
1	WAIT																					
1	READ	100	0				enabled	0	4e_1000_0000	4e_FFFF_FFFF	auto_incr	0x0200_0000	F	5	0	WRAP	NORMAL	0	0	0	0	0
1	WAIT																					
1	END_LOOP																					
1	START_LOOP	20	use_original_addr																			
1	WRITE	100	0	0	same_as_addr	0	enabled	0	9681_2000_0000	9681_FFFF_FFFF	auto_incr	0x0300_0000	1	5	0	WRAP	NORMAL	0	0	0	0	0
1	WAIT																					
1	READ	100	0				enabled	0	9681_2000_0000	9681_FFFF_FFFF	auto_incr	0x0300_0000	1	5	0	WRAP	NORMAL	0	0	0	0	0
1	WAIT																					
1	END_LOOP																					

Figure 35: **CSV Format for AXI4-Stream**

TG_NUM	cmd	pkt_count	inter_pkt_delay	inter_transfer_delay	tdata_pattern	tdata_pat_value	noc_dest_id	tdest_id	pkt_len	pkt_id	pkt_user
1	STREAM	2			random	0xabcdefab	0x4	0x8	0x1	0x8	0xF
1	WAIT										
1	START_LOOP	100									
1	STREAM	2			random	0xabcdefab	0x4	0x8	0x1	0x8	0xF
1	END_LOOP										
1	STREAM	2			random	0xabcdefab	0x4	0x8	0x1	0x8	0xF

Fixed Operating Mode

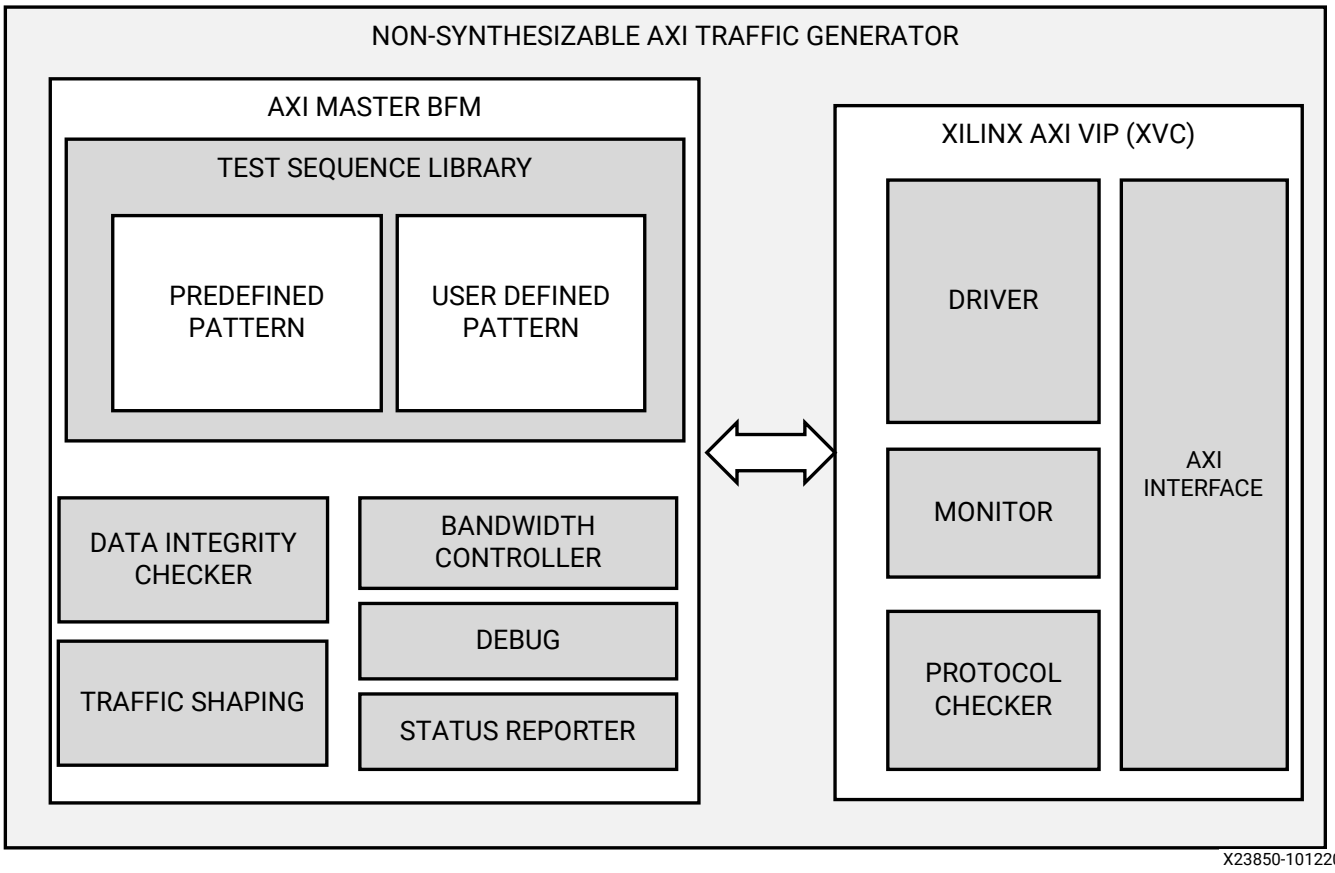
The Non-Synthesizable TG has a set of predefined patterns that can be selected and configured through the Vivado IDE. The basic patterns include continuous reads, continuous writes, writes followed by reads, writes and reads in parallel, and interleaved writes and reads. In this mode you can configure the number of transactions, start address, end address, axi_id, axi_burst, axi_len, axi_size, bandwidth, data pattern, and data integrity checks.

***Note*:** The Synthesizable TG does not support Fixed Operating Mode.

High-Level Architecture

The high level block diagram for the Simulation Traffic Generator is shown in the following figure.

Figure 36: **Non-Synthesizable TG Block Diagram**



- **Test Sequence Library:** This module generates the AXI request transaction objects based on the user configuration from the Vivado IDE.
  - **Predefined Pattern:** The Predefined Pattern allows you to generate simple custom traffic that can be configured in the Vivado® IDE.
  - **User Defined Pattern:** The User Defined Pattern allows you to write your own sequences to generate complex custom traffic using CSV input format.
- **Bandwidth Controller:** This module calculates the transaction start delay based on the configured bandwidth settings. It is set on the transaction object. The updated transaction object is sent to the XVC module from the AXI Master BFM module.
- **Driver:** This module uses the BFM transaction object to drive the AXI request on the AXI interface.
- **Monitor:** The Monitor module collects the AXI request and AXI response details from the interface to create the complete transaction object. For each AXI request, the monitor creates a transaction object that has request and corresponding response information. These transaction objects are used on the Data Integrity Checker module to do data comparison.

- **Protocol Checker:** The Protocol Checker module checks the protocol violation of AXI requests and AXI responses on the interface.
- **Data Integrity Checker:** The Data Integrity Checker module collects the complete write transaction object from the monitor and stores the Write data into the internal sparse memory (associative array) with the AXI address as array index. When the complete read transaction object is received, the data integrity is evaluated by comparing the read data against the written data.
- **Traffic Shaping:** The Traffic Shaping module is used to synchronize the traffic across the TGs in the design.
- **Debug:** For ease of debugging, the Debug module prints information such as the sent request count, received response count, and error/warning condition messages.
- **Status Reporter:** The Status Reporter module generates the test status report that consists of test configuration details and test pass/fail information. It is printed in the simulation log at end of simulation.

***Note*:** For the AXI4-Stream protocol the Data Integrity Checker and XVC monitor modules are not used.

Product Specification

Port Description

Table 212: **Non-Synthesizable TG**

Signal	I/O	Port Width	Description
clk	I	1	Traffic Generator clock (AXI clock)
tg_rst_n	I	1	Traffic Generator reset
TG Control Signals			
axi_tg_start	I	1	The execution of AXI-TG starts when this signal is asserted. If no synchronization with init_calib_complete or other TGs is required, tie this signal to tg_rst_n.
axi_tg_done	O	1	When asserted, it indicates that all Write/Read transactions are completed
axi_tg_error	O	1	When asserted, indicates that the TG is encountered an error condition
trigger_in	I	1	Input trigger for synchronization
trigger_out	O	1	Output trigger for synchronization
AXI Signals			
axi_aw*, axi_w*, axi_b*, axi_ar*, axi_r*,	I/O	Varies based on configuration	AXI3/AXI4 master interface signals. See AMBA AXI protocol specification for AXI3, AXI4.
AXI4-Stream Signals			
axis_t*	I/O	Varies based on configuration	AXI4-Stream master interface signals. See AMBA AXI protocol specification for AXI4-Stream.
NoC IP Specific Signals			
nmu_wr_usr_dst	O	12	Sends the destination slave ID for Write/Stream transactions. Only used when the TG is connected to Versal™ NoC NMU (NoC Master Unit).
nmu_rd_usr_dst	O	12	Sends the destination slave ID for Read transactions. Only used when the TG is connected to Versal NoC NMU (NoC Master Unit).

Data Integrity Modes

Data integrity checking can be enabled or disabled by selecting or deselecting the Data Integrity Check option in the Non-Synthesizable TG Options tab in the Vivado IDE. While in this mode, you can select predefined data patterns or when using a CSV stimulus it can be controlled by the data_integrity field of the CSV file.

The data integrity check is performed by the TG when the Read data is received on the RDATA channel. It is evaluated with byte-level (AXI byte lane) granularity. The Data Integrity Checker module of each TG on the system shares the common memory array across the entire slave address space. This makes it possible to check the data integrity across multiple masters in a system. For example, Master A can write data to Slave X and then Master B can read the same data from the slave and check if the resulting data matches the expected data.

- **Error Conditions:**
  - If the Read response is SLVERR or DECERR, then the data integrity check does not happen for the respective Read data beat. The TG asserts an error based on the `exp_resp` settings.
  - If a data mismatch occurs, the TG reports an error:

```
# ============================================================
# >>>>>> SRC_ID 1 :: DATA MISMATCH ERROR >>>>>>
# XIL_AXI_READ (3133) A:0x0000000100000000 ID:0x00000000 len:0x0f
XIL_AXI_SIZE_32BYTE XIL_AXI_BURST_TYPE_INCR C:0b0000
L:XIL_AXI_ALOCK_NOLOCK P:0b000
# BEAT = 3, BYTELANE = 3, BYTEADDR = 100000063
# wr_data = 78
# rd_data = bb
# ============================================================
```

    Note: If data integrity is enabled in the Vivado IDE and a mismatch occurs, the same type of error message is also be printed in non-CSV mode.

- **Warning Conditions:**
  - If the Read is to an unwritten location, the data integrity check does not happen to that byte lane and the TG reports a warning in the test status report.
  - If a Write happens to the same location twice without reading that location at least once (meaning data was written and then overwritten without a Read in between), the TG gives a warning in the test status report that is automatically printed in the log at end of the simulation.

Additional Considerations for Non-Synthesizable TG Data Integrity Checks

- Data integrity checking is not used for AXI4-Stream because it is a one-way traffic (Write only).
- Enabling data integrity check when doing parallel Read/Write traffic to the same slave location is not recommended because it may give false data mismatch errors.
- Enabling data integrity check when doing parallel Write traffic from multiple TGs to the same slave location is not recommended because it may give false data mismatch errors.

Delays supported by the TG

- **Inter transfer delay/inter beat delay:** Represents the delay in terms of input CLK ticks between the data transfers (that is to say, data beats) in a transaction. In AXI3/AXI4, this delay is called "inter beat delay." In AXI4-Stream, this delay is called "inter transfer delay." For the Non-Synthesizable TG, the delay is supported up to a 32-bit value.
- **Inter packet delay/start delay:** Represents the delay in terms of input CLK ticks between transactions. In AXI3/AXI4 this delay is called "start delay." In AXI4-Stream this delay is called "inter packet delay." For the Non-Synthesizable TG, the delay is supported up to a 32-bit value.

Performance Monitoring

To get the bandwidth and latency values, connect the NoC AXI Performance Monitor (axi_pmon) IP to the TG in Vivado IP integrator. The NoC AXI Performance Monitor is a non-synthesizable IP that replaces the performance monitoring counters found in the Synthesizable Performance AXI Traffic Generator.

Minimum and maximum latency values are calculated in the same way as in the Synthesizable TG. The average latency, however, is calculated as the sum of the latency values of all transactions divided by the number of transactions. In Synthesizable TG, only the summation of latency values are provided and it is a user's responsibility to divide it by the number of transactions to get the actual average latency.

The Read bandwidth measurement starts when the first Read request is issued and stops when the response to the last Read request is received. The Write bandwidth measurement starts when the first Write request is issued and stops when the response to the last Write request is received.

- **For AXI3/AXI4:** Bandwidth (in MB/s) = (Total_Number_of_Bytes_Transferred*1000_000)/ (Last_Response_Received_Time_in_ps - First_Request_Sent_Time_in_ps)
- **For AXI4-Stream:** Bandwidth (in MB/s) = (Total_Number_of_Beats_Transferred*Data_Width_in_Bytes*1000_000)/ (Last_Beat_Sent_Time_in_ps-First_Beat_Sent_Time_in_ps)

***Note*:** The Sent_Time and Received_Time are captured when the corresponding AXI channels' valid and ready handshake is done.

The bandwidth figures are printed in the log at end of the simulation, as given below.

For AXI3/AXI4:

```
# ===========================================================
# >>>>>> SRC_ID 0 :: AXI_PMON :: BW ANALYSIS >>>>>>
# ===========================================================
# AXI Clock Period = 3332 ps
# Min Write Latency = 60 axi clock cycles
# Max Write Latency = 66 axi clock cycles
# Avg Write Latency = 62 axi clock cycles
# Actual Achieved Write Bandwidth = 560.590030 MBps
# ***********************************************
# Min Read Latency = 26 axi clock cycles
# Max Read Latency = 32 axi clock cycles
# Avg Read Latency = 29 axi clock cycles
# Actual Achieved Read Bandwidth = 605.347717 MBps
# ===========================================================
```

For AXI4-Stream:

```
# ===========================================================
# >>>>>> SRC_ID 0 :: AXIS_PMON :: BW ANALYSIS >>>>>>
# ===========================================================
# AXI Clock Period = 3332 ps
# Actual Achieved Stream Bandwidth = 560.590030 MBps
# ===========================================================
```

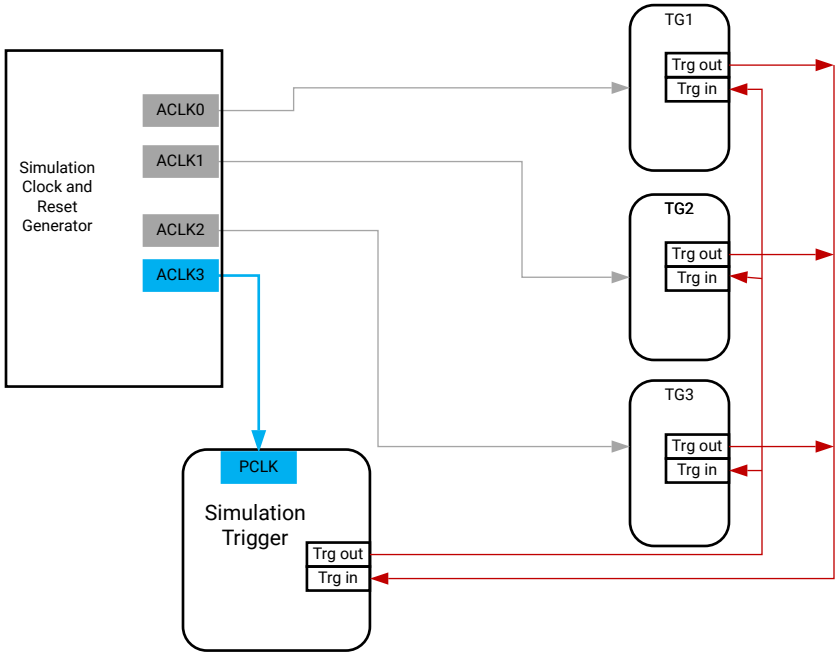
Designing with the Core

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

Clocking

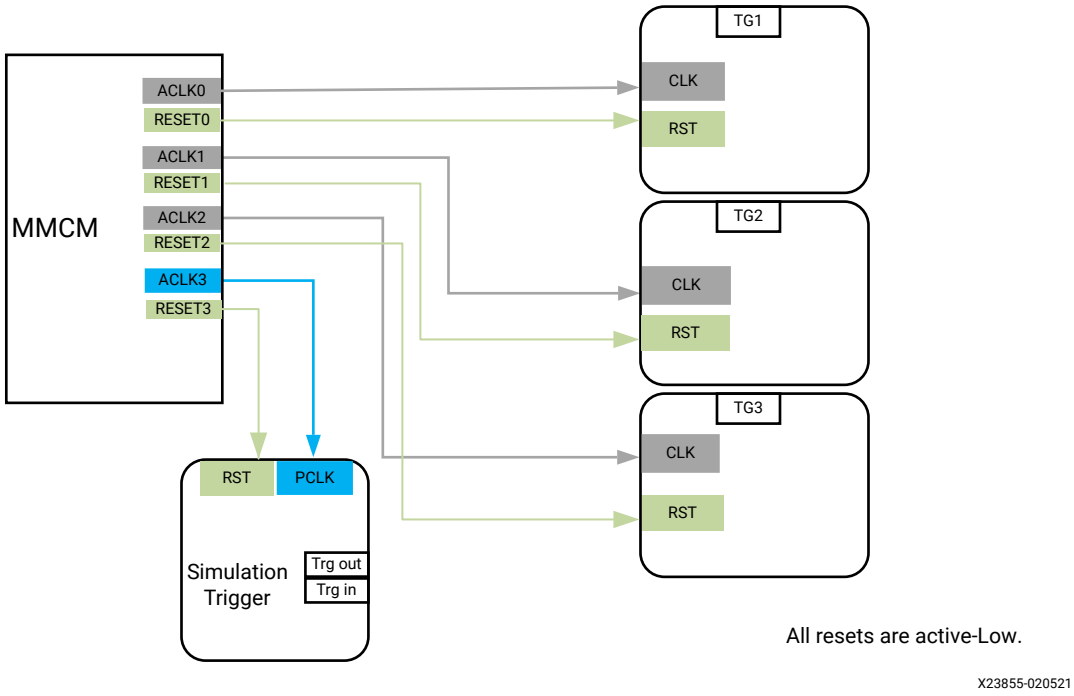
The Simulation Clock and Reset Generator block is used to generate the AXI clocks for different TGs and a `pclk` signal for the Simulation Trigger.

You can run Multiple TGs at the same or at a different frequency. A Simulation Clock and Reset Generator generates multiple clock frequencies during simulation. Generated clocks can be used to run TGs at independent AXI clock frequencies.



Resets

The Reset to every individual TG is generated from Simulation Clock and Reset Generator IP.



User Guidelines

- In the user defined pattern test:
  - Nested loop is not supported.
  - The `START_LOOP` and `END_LOOP` commands must be used as pair. There should never be an uneven number of `START_LOOP` or `END_LOOP` commands present in the CSV file.
  - CSV hex fields must have a "0x" prefix before the rest of the value; for example, 0x12345678 for a 32-bit hex value.
- In the predefined pattern test:
  - The NoC destination ID value cannot be customized and the value generated by the Vivado software is sent on NoC signals (`nmu_wr_usr_dst/nmu_rd_usr_dst`).

- You can only configure one set of AXI transaction details, even when multiple slaves are connected to the TG using interconnect. If the multiple slaves are connected to a TG, the same set of traffic is sent to each slave with unique `axi_addr` addresses (in the case of AXI3/AXI4) and NoC destination ID values. These values are selected from the Vivado software. The TG drives the set number of transactions to the first connected slave and repeats the same number of transactions to the second connected slave and so on.

User Guidelines Specific to AXI3/AXI4

- In the predefined pattern test:
  - The `AXI_AxCACHE`, `AXI_AxLOCK`, `AXI_AxQOS`, `AXI_AxPROT`, `AXI_AxREGION`, and `AXI_AxUSER` AXI fields cannot be customized. These fields have a default value of 0 except `AXI_CACHE`, that has a default value of 2.
  - The `start_delay`, `inter_beat_delay`, and `exp_resp` values cannot be customized. A default value of 0 for `inter_beat_delay` is used. The `start_delay` value is calculated based on the bandwidth settings. The `exp_resp` is empty by default.
  - The address pattern cannot be configured. By default, the address is automatically incremented on each transaction. This is equivalent to the `auto_incr` address pattern in the user defined pattern test.
  - The AXI address offset cannot be customized. The default value is 0. The first transaction is always from the programmed base address value.
  - The seed value for the Write data pattern of `RANDOM_DATA` cannot be customized. The default value of 0 is used to generate random data.
- AXI WUSER customization is not supported. The default value of 0 is used.
- In the case of an incremental address pattern (that is to say, an `auto_incr` or `addr_incr_by` value address pattern):
  - When generating the AXI address, if the TG is asked to generate an address that crosses the 4K boundary, the TG splits that single transaction into two transactions at the 4K boundary by adjusting the AXI length. For example, if the TG is asked to generate `axi_address` = 0x0201_0000_0F80, `axi_size` = 6, and `axi_len` = 7, this single transaction crosses the 4K boundary. The TG splits this transaction into two transactions at the 4K boundary to avoid the 4K boundary crossing issue. The split transactions are:
    - TXN1: `axi_address` = 0x0201_0000_0F80, `axi_size` = 6 and `axi_len` = 1
    - TXN2: `axi_address` = 0x0201_0000_1000, `axi_size` = 6 and `axi_len` = 5.
  - When generating the AXI address, if the TG is asked to generate an out-of-range address (that is, an address that does not fall within `base_addr` and `high_addr`), the TG restarts the transaction from `base_addr`.

User Guidelines Specific to AXI4-Stream

- In the predefined pattern test:
  - The values for `inter_pkt_delay` and `inter_transfer_delay` cannot be customized. The default value of 0 for `inter_transfer_delay` is used. The `inter_pkt_delay` is calculated based on the bandwidth settings.
  - The `TDEST_ID`, `TID`, and `TUSER` AXI fields cannot be customized. These fields have a default value of 0.

Using the Traffic Generator

Test Status Report

AXI3/AXI4 Test

Any test provides a status report at the end of simulation. The status report comprises the test pass/fail status, the total number of sent requests, the total number of received responses, and data integrity status (enabled/disabled).

```
# ===========================================================
# >>>>>> SRC ID 0 :: TEST REPORT >>>>>>
# ===========================================================
# [INFO] SRC ID = 0 ::: TG_HIERARCHY =
design_1_wrapper_sim_wrapper.design_1_wrapper_i.design_1_i.perf_axi_tg_0.ins
t.u_top_axi_mst
# [INFO] SRC ID = 0 ::: AXI_PROTOCOL = AXI4
# [INFO] SRC ID = 0 ::: AXI_CLK_PERIOD = 3332ps, AXI_DATAWIDTH = 512bit
# [INFO] SRC ID = 0 ::: TEST_NAME = write_read_interleaved
# [INFO] SRC ID = 0 ::: TOTAL_WRITE_REQ_SENT = 100,
TOTAL_WRITE_RESP_RECEIVED = 100
# [INFO] SRC ID = 0 ::: TOTAL_READ_REQ_SENT = 100, TOTAL_READ_RESP_RECEIVED
= 100
# [INFO] SRC ID = 0 ::: DATA_INTEGRITY_CHECK = ENABLED
# [INFO] SRC ID = 0 ::: TEST_STATUS = TEST PASSED
# ===========================================================
```

- **Errors:** The TEST FAILED status is reported in the following cases:
  - All Read/Write transactions are not completed (that is, all responses to the sent requests have *not* been received by the sender TG).
  - If the TG receives the response SLVERR/DECERR to the check for response correctness (BRESP/RRESP).
    - **Exceptions:**
      . If the exp_resp field is set as SLVERR in the CSV test, OKAY, EXOKAY, or DECERR responses from the slave are considered as failed.

. If the exp_resp field is set as DECERR in the CSV test, OKAY, EXOKAY, or SLVERR responses from the slave is considered as failed.

- Data integrity check should pass if enabled.
- Any test sequence error occurring due to the input to the TG causes the TG to update the status to failed.

- **Warnings:**
  - Write data has been overwritten without reading it back.
  - Read occurred from an unwritten location.
  - The AXI Address 4K boundary is crossed while generating AXI transactions. Based on the input to the TG, and the address/length of these transactions have been adjusted to avoid 4K boundary crossing.
  - The TG is idle throughout the simulation if no traffic is sent from the TG. This warning occurs when the txn_cnt is set as 0 or when an empty CSV without any Write/Read commands specified is given to the TG.
  - Simulation ran for fewer cycles (<6000 AXI clock cycles), if the simulator run time is set as less than a 6000 `axi_clock` cycle period. This warning can be ignored if the simulation is terminated early due to any other fatal error.

AXI4-Stream Test

Any test provides a status report at the end of simulation. The status report comprises the test pass/fail status and the total number of sent packets.

```
# ===========================================================
# >>>>>> SRC ID 0 :: TEST REPORT >>>>>>
# ===========================================================
# [INFO] SRC ID = 0 ::: TG_HIERARCHY =
design_1_wrapper_sim_wrapper.design_1_wrapper_i.design_1_i.perf_axi_tg_0.ins
t.u_top_axi_mst
# [INFO] SRC ID = 0 ::: AXI_PROTOCOL = AXI4_STREAM
# [INFO] SRC ID = 0 ::: AXI_CLK_PERIOD = 3332ps, AXI_DATAWIDTH = 512bit
# [INFO] SRC ID = 0 ::: TEST_NAME = write_only
# [INFO] SRC ID = 0 ::: TOTAL_PACKET_SENT = 100
# [INFO] SRC ID = 0 ::: TEST_STATUS = TEST PASSED
# ===========================================================
```

- **Errors:** The TEST FAILED status is reported in the following cases:
  - If not all sent packet transactions have been accepted by the slave.
  - If any test sequence error occurred due to the input to the TG.
- **Warnings:**
  - The TG is idle throughout the simulation if no traffic is sent from the TG. This warning occurs when the pkt_cnt is set as 0 or when an empty CSV without any Stream commands specified is given to the TG.

- If the simulator run time set on the Vivado IDE is less than 6000 AXI cycles, the TG issues a warning. This warning can be ignored if the simulation is terminated early due to any other fatal error.

Design Flow Steps

Customizing and Generating the Core

This section includes information about using Xilinx® tools to customize and generate the Performance AXI Traffic Generator and the Simulation Trigger for NoC AXI TG in the Vivado® Design Suite.

If you are customizing and generating the Performance AXI Traffic Generator 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 Performance AXI Traffic Generator using the following steps:

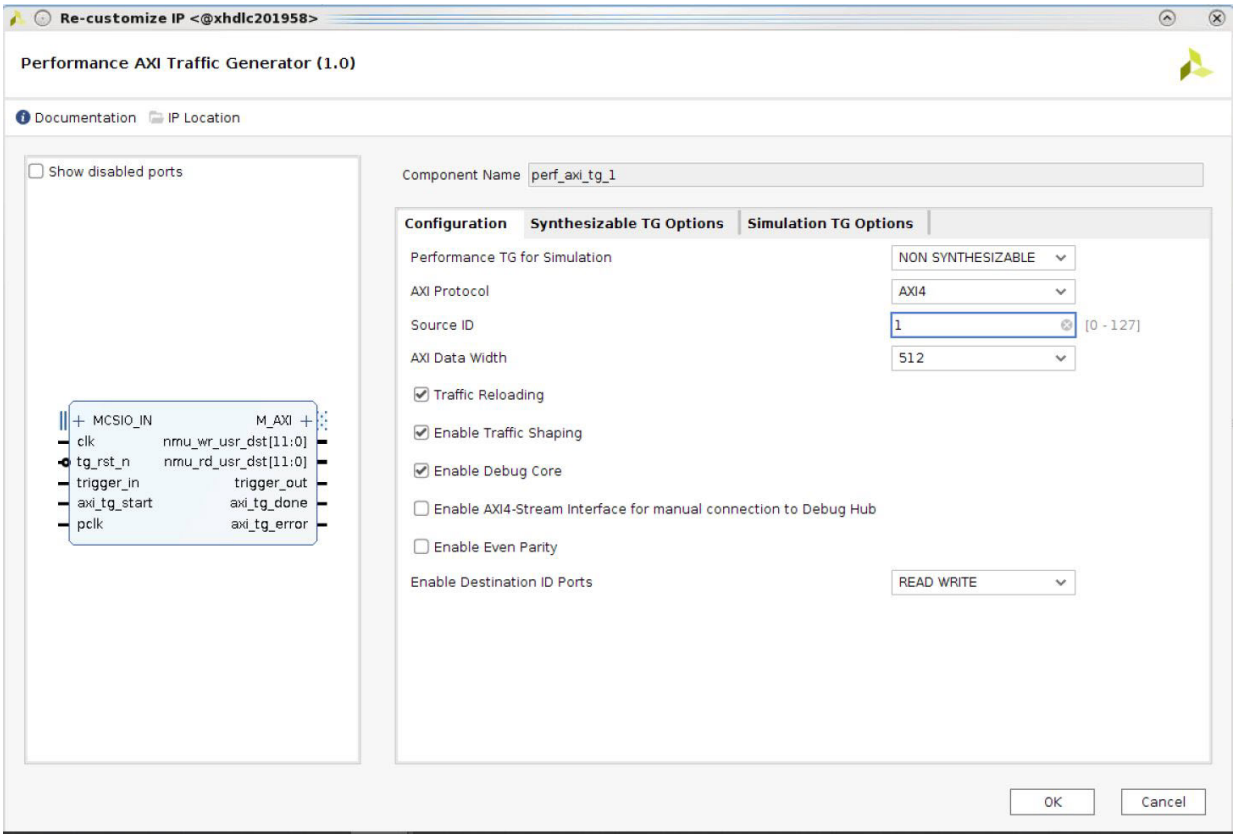
1. Select the IP from the IP catalog.
2. Double-click the selected IP or select the Customize IP command from the toolbar or right-click menu.

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.

Configuration Tab

Figure 37: **Configuration Tab**



- **Component Name:**
- **Performance TG for Simulation:**

  Synthesizable

  Non-synthesizable
- **AXI protocol:**

  AXI3

  AXI4

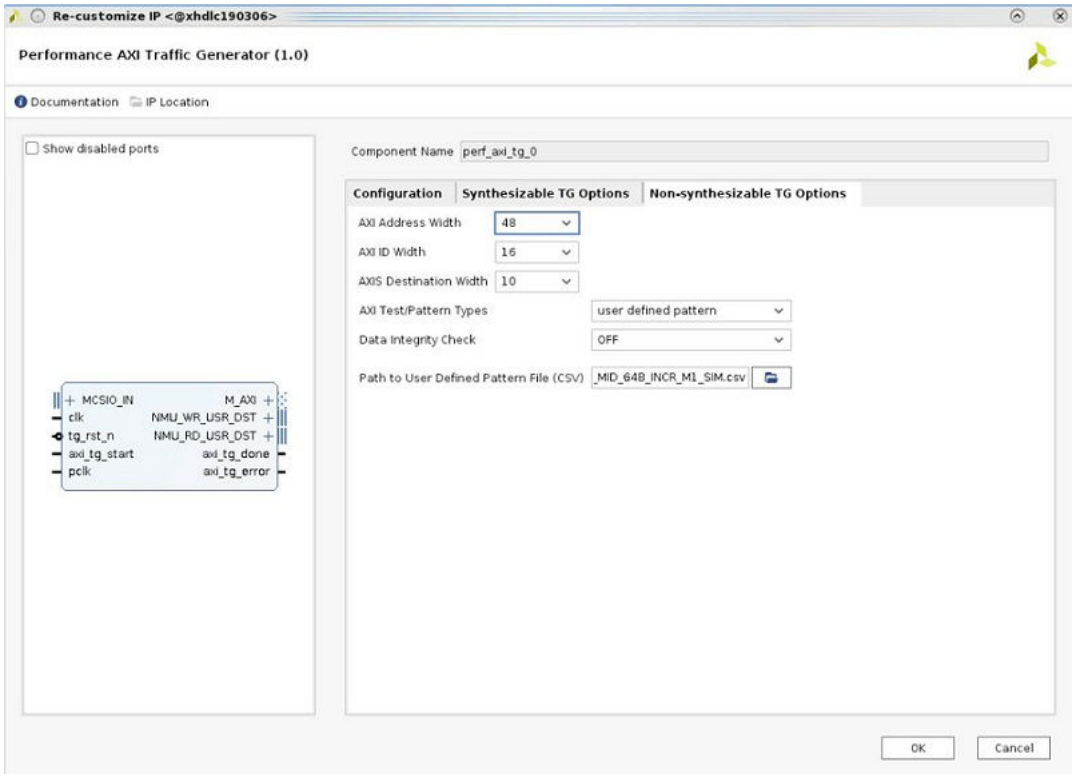
  AXI4-Stream

  Note: AXI3 is only available for the Non-Synthesizable TG.
- **Source ID:** The TG number must be specified here. Supports 0 to 127.
- **AXI Data Width:** Supported widths are 32, 64, 128, 256, and 512.

- **Enable Traffic Shaping:** Enables the traffic shaping module in the TG. The `trigger_out` and `trigger_in` ports will appear on the TG interface.
- **Enable Even Parity:** When enabled, even parity for AXI address and Write data is generated and sent on the `AxUSER` and `WUSER` signals respectively.
- **Enable NoC User Destination ID Ports:** NONE, READ, WRITE, READ_WRITE

Non-Synthesizable TG Options Tab

Figure 38: **Non-Synthesizable TG Options Tab**



- **AXI Address Width for Simulation:** 12 to 64
- **AXI ID Width for Simulation:** 1 to 16
- **AXI Test/Pattern Types:**
  - **Write only:** This test issues a continuous (that is, back to back) number (<N>) of Write transactions with an address delay of X. The value of X is calculated from the requested bandwidth. You can configure all AXI fields and the start/end address. This test does not support data integrity checking.

- **Read only:** This test issues a continuous number (<N>) of Read transactions with an address delay of X (the value of X is calculated from the requested bandwidth). You can configure all AXI fields and the start/end address. This test does not support data integrity checking.
- **Writes and reads in parallel:** This test issues a continuous number (<N>) of Write and Read transactions in parallel with an address delay of X. The value of X for the Read and Write channel is calculated from the requested Read and Write bandwidth respectively. You can configure all AXI fields and the start/end address. This test does not support data integrity checking.
- **Writes followed by reads:** A number (<N>) of continuous Writes is issued and waits for all Write responses to be received. It then issues the N number of continuous Reads to the already written address. This test supports data integrity checking.
- **Write read interleaved:** This test issues a number (<N>) of continuous Writes to the connected slave and issues Reads to the written address as soon as Write response is received. Each Read request is issued after receiving the Write response from the slave (it ensures that the slave has written the data). The Read request is sent to the written address of the corresponding BID. This test supports data integrity checking.
- **User defined pattern:** Through the CSV file.
- **Video pattern:** This test sequence generates a video pattern based on video format inputs from the Vivado IDE. To send a frame, the rows are sent one by one. If the Number of frame buffers is more than 1, consecutive frames are sent to a different address provided by Offset of Next Frame in Bytes. The total number of frames sent is controlled by the Number of frames to send parameter. The bandwidth is determined by the Frame rate parameter. There are two transaction options: Write only and Read only. This test does not support data integrity checking. Only RGB format is currently supported.

- **Data Integrity Check:**

  ON, OFF

- **AXI Read Burst:** INCR, WRAP, FIXED
- **AXI Read Size:** 1, 2, 4, 8, 16, 32, 64 (in bytes), and Exercise all Read Size
- **AXI Read Length:** 0-255, Exercise all Read Length
- **AXI Read Bandwidth (MBPS):** 10-19200
- **AXI Read Base Address:** Slave Base Address
- **AXI Read High Address:** Slave High Address
- **Number of read transactions:** 1-1000000
- **Write Data Pattern Types:**

  See Table 213 for details on Data Patterns for the Non-Synthesizable TG.

Constant Data

Random Data

Walking 1 Data

Walking 0 Data

Hammer Data

SRC ID as Data

ADDR as Data

ADDR as Data XOR

AXI ID as Data

AXI Burst as Data

AXI Length as Data

AXI Size as Data

AXI Cache as Data

- **AXI Write Burst:**

  INCR

  WRAP

  FIXED

- **AXI Write Size:** 1, 2, 4, 8, 16, 32, 64 (in Bytes) and Exercise all Write Size
- **AXI Write Length:** 0–255, Exercise all Write Length
- **AXI Write Bandwidth (MBPS):** 10–19200
- **Number of Write transactions:** 1–1000000
- **Path to user Defined Pattern File (CSV):** The location of the CSV file must be added here.

When video pattern is selected for the AXI Test/Pattern types:

- **Video Traffic Type:** Write, Read
- **Samples per clock:** 1, 2, 4, 8, 16
- **Horizonal Pixels Size:** 64–1920
- **Vertical Pixels Size:** 64–1080

- **Frame rate:** 10–100
- **Number of bits per component:** 1-16
- **Color format:** RGB
- **Number of frame buffers:** 1–100
- **Number of frames to send:** 1–100000

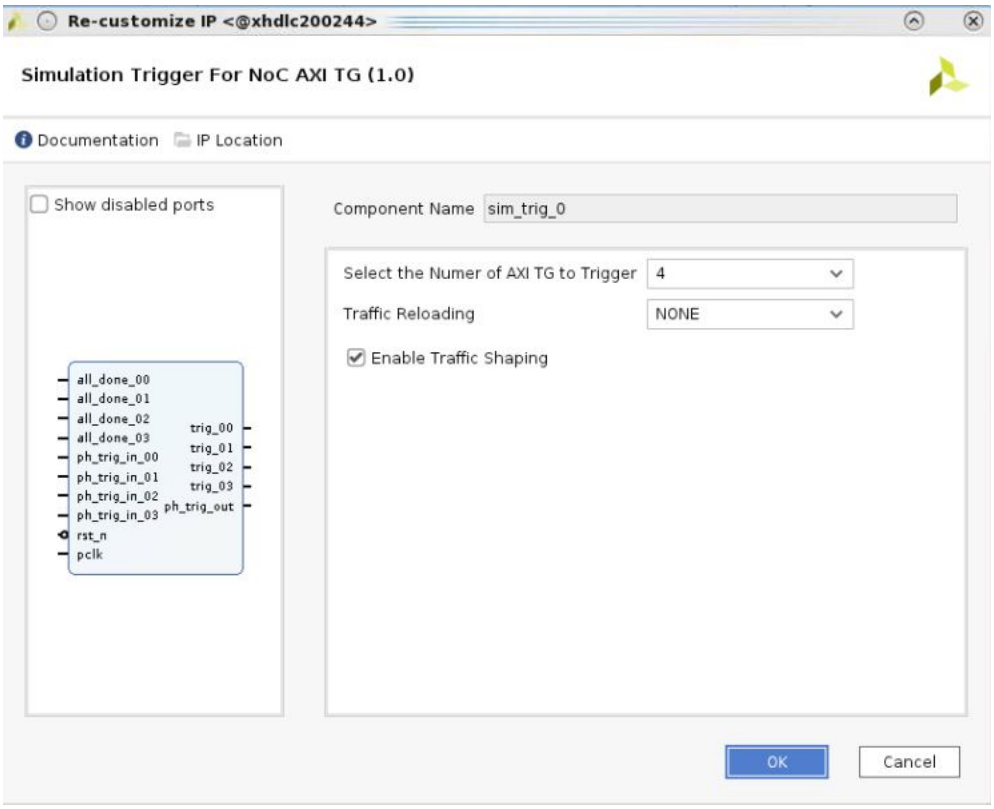
Simulation Trigger for the NoC AXI TG IP

This section includes information about using Xilinx tools to customize and generate the simulation trigger for the NoC AXI TG IP in the Vivado Design Suite. The Simulation Trigger IP can be used with Non-Synthesizable TG for traffic shaping between multiple TGs. To customize the simulation trigger for the NoC AXI TG IP, perform the following initial steps:

1. Select the IP from the IP catalog in the IP integrator window.
2. Double-click on the selected IP.

Simulation Trigger Component Configuration Tab

Figure 39: **Simulation Trigger Component Configuration Tab**



- **Select the Number of AXI TG to trigger:** Set this value to the number of TGs used in the system design.
- **Traffic Reloading:** This feature is not supported by Non-Synthesizable TG . In the GUI, select the traffic reloading option as NONE to disable this feature.
- **Enable Traffic Shaping:** Select this option for doing traffic shaping between Performance AXI Traffic Generator cores in a system design. In a system-level design, if this option is selected in simulation trigger for NoC AXI TG IP, you should also enable it in Performance AXI Traffic Generator cores. Refer to Synchronization of Multiple TGs for more information.

CSV Format for Multiple TGs

There are two methods for providing a CSV file input to the TG when there is more than one TG in the system:

- A dedicated CSV file as input for each TG
- A common CSV file as input for all TGs

Dedicated CSV for Each TG

In this scenario, there are two TGs (AXI masters), and each is assigned a CSV file in its configuration GUI. TG1 has a source ID of 1 and TG2 has a source ID of 2..

Figure 40: **Dedicated CSV for Each TG**

TG1 executes the CSV lines where `TG_NUM` is 1. TG1 also executes the CSV lines where `TG_NUM` is left empty.

TG2 executes the CSV lines where `TG_NUM` is 2. TG2 also executes the CSV lines where `TG_NUM` is left empty.

Figure 41: **axi_design_a_tg1.csv (TG_NUM = 1)**

TG_NUM	cmd	txn_count	start_delay	inter_beat_delay	wdata_pattern	wdata_pat_value	data_integrity	dest_id	base_addr	high_addr	addr_inc_by	axi_addr_offset
1	WRITE	5	0	0	random	0	enabled		0000_0000	0000_FFFF	auto_incr	0
1	READ	10	0	0	random	0	enabled		0000_0000	0001_FFFF	auto_incr	0

Figure 42: **axi_design_a_tg1.csv (TG_NUM Empty)**

TG_NUM	cmd	txn_count	start_delay	inter_beat_delay	wdata_pattern	wdata_pat_value	data_integrity	dest_id	base_addr	high_addr	addr_inc_by	axi_addr_offset
	WRITE	5	0	0	random	0	enabled		0000_0000	0000_FFFF	auto_incr	0
	READ	10	0	0	random	0	enabled		0000_0000	0001_FFFF	auto_incr	0

Figure 43: **axi_design_a_tg2.csv (TG_NUM = 2)**

TG_NUM	cmd	txn_count	start_delay	inter_beat_delay	wdata_pattern	wdata_pat_value	data_integrity	dest_id	base_addr	high_addr	addr_inc_by	axi_addr_offset
2	WRITE	5	0				enabled		0001_0000	0001_FFFF	auto_incr	0
2	READ	10	0	0	random	0	enabled		0000_0000	0001_FFFF	auto_incr	0

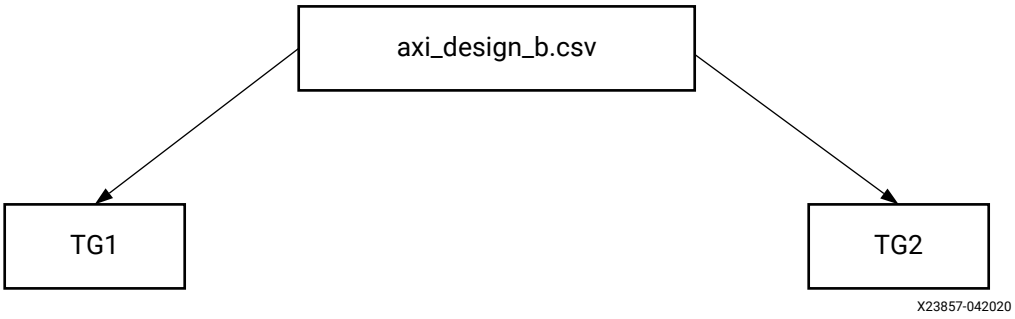
Figure 44: **axi_design_a_tg2.csv (TG_NUM Empty)**

TG_NUM	cmd	txn_count	start_delay	inter_beat_delay	wdata_pattern	wdata_pat_value	data_integrity	dest_id	base_addr	high_addr	addr_inc_by	axi_addr_offset
	WRITE	5	0				enabled		0001_0000	0001_FFFF	auto_incr	0
	READ	10	0	0	random	0	enabled		0000_0000	0001_FFFF	auto_incr	0

Common CSV Input for All TGs

In this scenario, there are two Traffic Generators (AXI masters) using a common CSV file as input. TG1 has a source ID of 1 and TG2 has a source ID of 2. The traffic is generated by the TG by using `TG_NUM` value field.

Figure 45: **Common CSV Input for All TGs**



Both TG1 and TG2 read the CSV file in the following figure. The TG parser reads the `TG_NUM` value on each CSV line and matches it with the source ID of the TG. In a given line, if the `TG_NUM` matches with source ID or the `TG_NUM` field is left empty, the CSV line is executed by the TG.

Figure 46: **Common CSV Example**

TG_NUM	cmd	txn_count	start_delay	inter_beat_delay	wdata_pattern	wdata_pat_value	data_integrity	dest_id	base_addr	high_addr	addr_inc_by	axi_addr_offset
1	WRITE	5	0	0	random	0	enabled		0000_0000	0000_FFFF	auto_incr	0
2	WRITE	5	0				enabled		0001_0000	0001_FFFF	auto_incr	0
	READ	10	0	0	random	0	enabled		0000_0000	0001_FFFF	auto_incr	0

- CSV line 2 (`TG_NUM` = 1) is executed only by TG1. TG2 skips this line from execution because it does not match the source ID of TG2.
- CSV line 3 (`TG_NUM` = 2) is executed only by TG2. TG1 skips this line from execution because it does not match the source ID of TG1.
- CSV line 4 (`TG_NUM` left empty) is executed by both TG1 and TG2 because the `TG_NUM` field is left empty

The following lines are executed by TG1.

Figure 47: **TG1 Parser Output**

TG_NUM	cmd	txn_count	start_delay	inter_beat_delay	wdata_pattern	wdata_pat_value	data_integrity	dest_id	base_addr	high_addr	addr_inc_by	axi_addr_offset
1	WRITE	5	0	0	random	0	enabled		0000_0000	0000_FFFF	auto_incr	0
1	READ	10	0	0	random	0	enabled		0000_0000	0000_FFFF	auto_incr	0

The following lines are executed by TG2.

Figure 48: **TG2 Parser Output**

TG_NUM	cmd	txn_count	start_delay	inter_beat_delay	wdata_pattern	wdata_pat_value	data_integrity	dest_id	base_addr	high_addr	addr_inc_by	axi_addr_offset
2	WRITE	5	0				enabled		0001_0000	0001_FFFF	auto_incr	0
2	READ	10	0	0	random	0	enabled		0000_0000	0001_FFFF	auto_incr	0

AXI3/AXI4 Traffic Pattern CSV Format

WRITE Command

Table 213: **WRITE CSV Command Format for AXI3/AXI4**

Command Input Field	Width	Non-Synthesizable TG Options
txn_count	32/STRING	• **<value in decimal>:** Transaction count. Repeat the current command <N> times, where N = value. • **<value in decimal>KB:** Amount of data to be transferred in KB, for example 50 KB, 100 KB. • **<value in decimal>MB:** Amount of data to be transferred in MB, for example 5 MB, 10 MB. • **<value in decimal>GB:** Amount of data to be transferred in GB, for example 1 GB, 2 GB. • **INF:** Repeat the current command infinitely.
start_delay	32/STRING	• **<value in decimal>:** Value in number of clocks. Issue transaction after specified number of clocks • **<value in decimal>MBps:** Bandwidth value for Write transactions, for example 12500 MB/s. ***Note*:** 1 MB/s = 1,000,000 Bps (Bytes per second).
inter_beat_delay	32	• **<value in decimal>:** Value in number of clocks. Adds specified number of clocks between two Write data beats in a transaction

Table 213: **WRITE CSV Command Format for AXI3/AXI4** *(cont'd)*

Command Input Field	Width	Non-Synthesizable TG Options
wdata_pattern	STRING	• constant • random • walking_0 • walking_1 • hammer • same_as_src • same_as_addr • same_as_addr_xor • same_as_id • same_as_burst • same_as_len • same_as_size • same_as_cache ***Note*:** Data pattern details are described in the Data Patterns section.
wdata_pat_value	512	• **<value in hex>:** When the wdata_pattern field is set to constant, this field value is used as constant data which is sent on all Write data beats (wdata_pattern: constant). • **<seed_value in hex>:** When wdata_pattern field is selected as “random”, this field value is used as seed for RNG (wdata_pattern: random).
data_integrity	STRING	• **enabled:** Data integrity checks will be enabled for the Write transaction. The Write transaction details are stored in the AXI TG sparse memory to do comparison later when a Read occurs to this location. • **disabled:** Data integrity checks will be disabled for the Write transaction, and details arel *not* stored in the AXI TG sparse memory.
dest_id	12	• **<value in hex>:** The targeted NoC slave destination ID value (12-bit value) can be given in this field for the transaction. The dest_id value is sent on the nmu_wr_usr_dst signal along with AXI Write requests.

Table 213: **WRITE CSV Command Format for AXI3/AXI4** *(cont'd)*

Command Input Field	Width	Non-Synthesizable TG Options
base_addr	64	• **<value in hex>:** The base_addr can be specified in this field. The incremented address for the transactions is looped back to the base address when the high address boundary is reached. In case of randomized address, the TG generates the address between base_addr and high_addr.
high_addr	64	• **<value in hex>:** The high address can be specified here. Up to this value the address is incremented by the TG. When the incremented address reaches the high address boundary, the next transaction start address is the base_addr. In case of a randomized address, TG generates the address between base_addr and high_addr.
addr_incr_by	64/STRING	• **<value in hex>:** This field value is considered as "value in hex" when the axi_addr_offset field is set to a start address offset value. The first transaction start address (SA) will be picked from the axi_addr_offset and base_addr fields (SA = base_addr + axi_addr_offset). From the second transaction onwards for each AXI transaction, the start address is incremented by the specified addr_incr_by value. For example: txn_count=4, base_addr=0x0_0000, high_addr=0xF_FFFFF, addr_incr_by=0x41, axi_addr_offset=0x01, axi_len=0, axi_size=4. 1st transaction: AWADDR=0x0_0001 2nd transaction: AWADDR=0x0_0042 3rd transaction: AWADDR=0x0_0083 4th transaction: AWADDR=0x0_00C4 • **<seed_value in hex>:** This field value is considered as seed for RNG to generate random addresses when the axi_addr_offset field is set to random address options. • **auto_incr:** The first transaction start address (SA) is picked from the axi_addr_offset and base_addr fields (SA = base_addr + axi_addr_offset). From second transaction onwards for each AXI transaction issued, the start address is calculated using the expression described below.

Table 213: **WRITE CSV Command Format for AXI3/AXI4** *(cont'd)*

Command Input Field	Width	Non-Synthesizable TG Options
addr_incr_by (continued)	64/STRING	• **INCR/WRAP Burst:** • **auto_incr:** The first transaction start address (SA) will be picked from the axi_addr_offset and base_addr fields (SA = base_addr + axi_addr_offset). From second transaction onwards for each AXI transaction issued, the start address is calculated using the following expression: start_address (from second transaction onwards) = previous_start_address + ((2^axi_size) * (axi_len+1)) For example: txn_count= 3, base_addr= 0x0000_0000, high_addr= 0xFFFF_FFFF, axi_addr_offset= 0x0000_0000, axi_size= 6, axi_len= 0, axi_burst= 1(INCR). The start addresses of all three transactions are as follows: 1st transaction start address: 0x0000_0000 2nd transaction start address: 0x0000_0000 + ((2^6)*(0+1)) = 0x0000_0040 3rd transaction start address: 0x0000_0040 + ((2^6)*(0+1)) = 0x0000_0080
addr_incr_by (continued)	64/STRING	• **FIXED Burst:** • **auto_incr:** The first transaction start address (SA) is picked from the axi_addr_offset and base_addr fields (SA = base_addr + axi_addr_offset). From second transaction onwards for each AXI transaction issued, the start address is calculated using the following expression: start_address (from second transaction onwards) = previous_start_address + (2^axi_size) For example, txn_count= 3, base_addr= 0x0000_0000, high_addr= 0xFFFF_FFFF, axi_addr_offset= 0x0000_0000, axi_size= 5, axi_len= 4, axi_burst= 0(FIXED). The start addresses of all three transactions are as follows: 1st transaction start address: 0x0000_0000 2nd transaction start address: 0x0000_0000 + (2^5) = 0x0000_0020 3rd transaction start address: 0x0000_0020 + (2^5) = 0x0000_0040

Table 213: **WRITE CSV Command Format for AXI3/AXI4** *(cont'd)*

Command Input Field	Width	Non-Synthesizable TG Options
axi_addr_offset	64/STRING	• **<value in hex>:** This is the offset value added to base_addr to calculate the start address of the first transaction. From the second transaction onwards, the start address is calculated based on the addr_incr_by field option. If the next transaction address reaches the high_addr boundary it is looped back to base_addr. • **random:** Any random address is generated between base_addr and high_addr. The addr_incr_by field value is set as seed to RNG. • **random_aligned:** Any random aligned address is generated between base_addr and high_addr. addr_incr_by field value is set as seed to RNG. • **random_unaligned:** Any random unaligned address is generated between base_addr and high_addr.The addr_incr_by field value is set as seed to RNG. ***Note*:** To generate an address that is aligned to the transaction size (that is, (len+1) * 2^size), use the SET_DEFAULT command addr_mask option to set the required LSB address bits to 0.
axi_len	8/STRING	• **<value in hex>:** Write transactions is sent with the specified AXI length value. Width: 4 for AXI3, 8 for AXI4. • **all:** Generates all length values in transactions. The first transaction is sent with an axi_len of 0x0 and then incremented on each consecutive AXI transaction. When MAX_LEN is reached, the transaction starts again from an axi_len of 0x0 and increments until MAX_LEN is reached, and so on. ***Note*:** For AXI3, MAX_LEN = 0xF. For AXI4, MAX_LEN varies depending on AWSIZE due to the 4k Boundary limit: • When AWSIZE is 64B, MAX_LEN is 0x3F. • When AWSIZE is 32B, MAX_LEN is 0x7F. • When AWSIZE is 16, 8, 4, 2 or 1B, MAX_LEN is 0xFF.

Table 213: **WRITE CSV Command Format for AXI3/AXI4** *(cont'd)*

Command Input Field	Width	Non-Synthesizable TG Options
axi_size	3/STRING	• **<value in hex>:** Write transactions is sent with the specified AXI size value. • **all:** Generates all the size values in transactions. The first AXI transaction is sent with MAX_SIZE and the size is decremented with each consecutive transaction. When it reaches 0x0, transactions start again from MAX_SIZE and decrements until 0x0 is reached again and so on. ***Note*:** MAX_SIZE = $clog2(AXI Data Width/8).
axi_id	16/STRING	• **<value in hex>:** Write transactions are sent with the specified AXI ID value. • **auto_incr:** The first Write transaction is sent out with an ID of 0x0 and each consecutive transaction ID is incremented by 0x1. When the ID reaches MAX_ID value, it starst again from 0x0 and increments until MAX_ID is reached again, and so on. ***Note*:** MAX_ID = 2^AXI ID Width - 1
axi_burst	2/STRING	• **<value in hex>:** Supported values are 0x0, 0x1, 0x2. • **<value in string>:** Supported values are FIXED, INCR, WRAP.
axi_lock	2/STRING	• **<value in hex>:** Supported values are 0x0, 0x1. • **<value in string>:** Supported values are NORMAL, EXCLUSIVE.
axi_cache	4/STRING	• **<value in hex>:** • **<value in string>:** Supported values are MOD, NON_MOD. ***Note*:** In the <value in string> option, MOD denotes that the AXI Write transaction is modifiable (AWCACHE = 0x2) by the slave/interconnect. NON_MOD denotes that the AXI Write transaction is non-modifiable (AWCACHE = 0x0) by the slave/interconnect.
axi_prot	3	<value in hex>
axi_qos	4	<value in hex>
axi_region	4	<value in hex>
axi_user (for AWUSER)	16	<value in hex>

Table 213: **WRITE CSV Command Format for AXI3/AXI4** *(cont'd)*

Command Input Field	Width	Non-Synthesizable TG Options
exp_resp	STRING	• **OKAY:** AXI Write responses should come as expected as OKAY, otherwise the TG will error out. • **EXOKAY:** AXI Write responses should come as expected as EXOKAY, otherwise the TG will error out. • **SLVERR:** AXI Write responses should come as expected as SLVERR, otherwise the TG will error out. • **DECERR:** AXI Write responses should come as expected as DECERR, otherwise the TG will error out. • **<Blank/Empty_Field>:** Leaving the exp_resp field Empty/Blank/Unfilled. In this case, TG does not do any expected response check. Instead, it treats the AXI responses in a standard manner; that is, if any responses are coming as SLVERR or DECERR, the TG will error out.

READ Command

Table 214: **READ CSV Command Format for AXI3/AXI4**

Command Input Field	Width	Non-Synthesizable TG Options
txn_count	32/STRING	• **<value in decimal>:** Transaction count. Repeat the current command <N> times, where N = value. • **<value in decimal>KB:** Amount of data to be transferred in KB, for example 50 KB, 100 KB. • **<value in decimal>MB:** Amount of data to be transferred in MB, for example 5 MB, 10 MB. • **<value in decimal>GB:** Amount of data to be transferred in GB, for example 1 GB, 2 GB. • **INF:** Repeat the current command infinitely.
start_delay	32/STRING	• **<value in decimal>:** Value in number of clocks. Issue transaction after specified number of clocks • **<value in decimal>MBps:** Bandwidth value for Read transactions, for example 12500 MB/s. ***Note*:** 1 MB/s = 1,000,000 Bps (Bytes per second).
inter_beat_delay	N/A	N/A
wdata_pattern	N/A	N/A

Table 214: **READ CSV Command Format for AXI3/AXI4** *(cont'd)*

Command Input Field	Width	Non-Synthesizable TG Options
wdata_pat_value	N/A	N/A
data_integrity	STRING	• **enabled:** Data integrity checks are enabled for the Read transaction. Whenever read data arrives, the TG compares it with the stored sparse memory data if the read address exists in the sparse memory. If the read address does not exist in the sparse memory, the TG skips the comparison and issues a “read from unwritten location” warning based on the VERBOSITY settings. • **disabled:** Data integrity checks are disabled for this Read transaction. The Read data is *not* compared with the stored sparse MEM data.
dest_id	12	• **<value in hex>:** The targeted NoC slave destination ID value (12-bit value) can be given in this field for the transaction. The dest_id value is sent on the nmu_rd_usr_dst signal along with AXI Read requests.
base_addr	64	• **<value in hex>:** The base_addr can be specified in this field. The incremented address for the transactions is looped back to the base address when the high address boundary is reached. In case of randomized address, the TG generates the address between base_addr and high_addr.
high_addr	64	• **<value in hex>:** The high address can be specified here. Up to this value the address is incremented by the TG. When the incremented address reaches the high address boundary, the next transaction start address is the base_addr. In case of a randomized address, TG generates the address between base_addr and high_addr.

Table 214: **READ CSV Command Format for AXI3/AXI4** *(cont'd)*

Command Input Field	Width	Non-Synthesizable TG Options
addr_incr_by	64/STRING	• **<value in hex>:** This field value is considered as "value in hex" when the axi_addr_offset field is set to a start address offset value. The first transaction start address (SA) is picked from the axi_addr_offset and base_addr fields (SA = base_addr + axi_addr_offset). From the second transaction onwards for each AXI transaction, the start address is incremented by the specified addr_incr_by value. For example: txn_count=4, base_addr=0x0_0000, high_addr=0xF_FFFFF, addr_incr_by=0x41, axi_addr_offset=0x01, axi_len=0, axi_size=4. 1st transaction: ARADDR=0x0_0001 2nd transaction: ARADDR=0x0_0042 3rd transaction: ARADDR=0x0_0083 4th transaction: ARADDR=0x0_00C4 • **<seed_value in hex>:** This field value is considered as seed for RNG to generate random addresses when the axi_addr_offset field is set to random address options. • **auto_incr:** The first transaction start address (SA) is picked from the axi_addr_offset and base_addr fields (SA = base_addr + axi_addr_offset). From second transaction onwards for each AXI transaction issued, the start address is calculated using the expression described below.
addr_incr_by (continued)	64/STRING	• **INCR/WRAP Burst:** • **auto_incr:** The first transaction start address (SA) is picked from the axi_addr_offset and base_addr fields (SA = base_addr + axi_addr_offset). From second transaction onwards for each AXI transaction issued, the start address is calculated using the following expression: start_address (from second transaction onwards) = previous_start_address + ((2^axi_size) * (axi_len+1)) For example: txn_count= 3, base_addr= 0x0000_0000, high_addr= 0xFFFF_FFFF, axi_addr_offset= 0x0000_0000, axi_size= 6, axi_len= 0, axi_burst= 1(INCR). The start addresses of all three transactions are as follows: 1st transaction start address: 0x0000_0000 2nd transaction start address: 0x0000_0000 + ((2^6)*(0+1)) = 0x0000_0040 3rd transaction start address: 0x0000_0040 + ((2^6)*(0+1)) = 0x0000_0080

Table 214: **READ CSV Command Format for AXI3/AXI4** *(cont'd)*

Command Input Field	Width	Non-Synthesizable TG Options

addr_incr_by (continued)	64/STRING	• **FIXED Burst:**
		  • **auto_incr:** The first transaction start address (SA) is picked from the axi_addr_offset and base_addr fields (SA = base_addr + axi_addr_offset). From second transaction onwards for each AXI transaction issued, the start address is calculated using the following expression:
		  start_address (from second transaction onwards) = previous_start_address + (2^axi_size)
		  For example, txn_count= 3, base_addr= 0x0000_0000, high_addr= 0xFFFF_FFFF, axi_addr_offset= 0x0000_0000, axi_size= 5, axi_len= 4, axi_burst= 0(FIXED).
		  The start addresses of all three transactions are as follows:
		  1st transaction start address: 0x0000_0000
		  2nd transaction start address: 0x0000_0000 + (2^5) = 0x0000_0020
		  3rd transaction start address: 0x0000_0020 + (2^5) = 0x0000_0040

axi_addr_offset	64/STRING	• **<value in hex>:** This is the offset value added to base_addr to calculate the start address of the first transaction. From the second transaction onwards, the start address is calculated based on the addr_incr_by field option. If the next transaction address reaches the high_addr boundary is looped back to base_addr. • **random:** Any random address is generated between base_addr and high_addr. The addr_incr_by field value is set as seed to RNG. • **random_aligned:** Any random aligned address is generated between base_addr and high_addr. addr_incr_by field value is set as seed to RNG. • **random_unaligned:** Any random unaligned address is generated between base_addr and high_addr.The addr_incr_by field value is set as seed to RNG. ***Note*:** To generate an address that is aligned to the transaction size (that is, (len+1) * 2^size), use the SET_DEFAULT command addr_mask option to set the required LSB address bits to 0.

Table 214: **READ CSV Command Format for AXI3/AXI4** *(cont'd)*

Command Input Field	Width	Non-Synthesizable TG Options
axi_len	8/STRING	• **<value in hex>:** Read transactions are sent with the specified AXI length value. Width: 4 for AXI3, 8 for AXI4. • **all:** Generates all length values in transactions. The first transaction is sent with an axi_len of 0x0 and then incremented on each consecutive AXI transaction. When MAX_LEN is reached, the transaction starts again from an axi_len of 0x0 and is incremented until MAX_LEN is reached, and so on. ***Note*:** For AXI3, MAX_LEN = 0xF. For AXI4, MAX_LEN varies depending on ARSIZE due to the 4k Boundary limit: • When ARSIZE is 64B, MAX_LEN is 0x3F. • When ARSIZE is 32B, MAX_LEN is 0x7F. • When ARSIZE is 16, 8, 4, 2 or 1B, MAX_LEN is 0xFF.
axi_size	3/STRING	• **<value in hex>:** Read transactions are sent with the specified AXI size value. • **all:** Generates all the size values in transactions. The first AXI transaction is sent with MAX_SIZE and the size is decremented with each consecutive transaction. When it reaches 0x0, transactions will start again from MAX_SIZE and is decremented until 0x0 is reached again and so on. ***Note*:** MAX_SIZE = $clog2(AXI Data Width/8).
axi_id	16/STRING	• **<value in hex>:** Read transactions are sent with the specified AXI ID value. • **auto_incr:** The first Read transaction is sent out with an ID of 0x0 and each consecutive transaction ID is incremented by 0x1. When the ID reaches MAX_ID value, it starts again from 0x0 and increment until MAX_ID is reached again, and so on. ***Note*:** MAX_ID = 2^AXI ID Width - 1
axi_burst	2/STRING	• **<value in hex>:** Supported values are 0x0, 0x1, 0x2. • **<value in string>:** Supported values are FIXED, INCR, WRAP.
axi_lock	2/STRING	• **<value in hex>:** Supported values are 0x0, 0x1. • **<value in string>:** Supported values are NORMAL, EXCLUSIVE.

Table 214: **READ CSV Command Format for AXI3/AXI4** *(cont'd)*

Command Input Field	Width	Non-Synthesizable TG Options
axi_cache	4/STRING	• **<value in hex>:** • **<value in string>:** Supported values are MOD, NON_MOD. ***Note*:** In the <value in string> option, MOD denotes that the AXI Read transaction is modifiable (ARCACHE = 0x2) by the slave/interconnect. NON_MOD denotes that the AXI Read transaction is non-modifiable (ARCACHE = 0x0) by the slave/interconnect.
axi_prot	3	<value in hex>
axi_qos	4	<value in hex>
axi_region	4	<value in hex>
axi_user	16	<value in hex>
exp_resp	STRING	• **OKAY:** AXI Read responses should come as expected as OKAY, otherwise the TG will error out. • **EXOKAY:** AXI Read responses should come as expected as EXOKAY, otherwise the TG will error out. • **SLVERR:** AXI Read responses should come as expected as SLVERR, otherwise the TG will error out. • **DECERR:** AXI Read responses should come as expected as DECERR, otherwise the TG will error out. • **<Blank/Empty_Field>:** Leaving the exp_resp field Empty/Blank/Unfilled. In this case, TG does not do any expected response check. Instead, it treats the AXI responses in a standard manner; that is, if any responses are coming as SLVERR or DECERR, the TG will error out.

WAIT Command

Table 215: **Wait Command Format for AXI3/AXI4**

Command Input Field	Width	Non-Synthesizable TG Options
wait_option	32/STRING	• **all_rd_resp:** Wait till all Read responses are received. • **all_wr_resp:** Wait till all Write responses are received. • **all_wr_rd_resp :** Wait till all Write and Read responses are received. • **<wait_period by clocks, value in decimal>:** Wait by specified number of AXI clocks. The wait_unit "clk" must be specified in next column. For example, if you want to wait for 50 AXI clock cycles, set 50 in this field and set "clk" in the wait_unit field. • **<wait_period by time scale, value in decimal>:** Wait by specified time period. The required wait_unit (ps/ns/us/ms) must be specified in next column. For example, if you want to wait for 10 milliseconds, set 10 in this field and set "ms" in the wait_unit field.
wait_unit	STRING	• **clk:** Wait unit to specify the wait_period by clocks. • **ps, ns, us, ms:** Wait unit to specify the wait_period by time scale.

START_LOOP/END_LOOP

Table 216: **LOOP CSV Command Format for AXI3/AXI4**

Command Input Field	Width	Non-Synthesizable TG Options
loop_count	32	• **<value in decimal>:** Repeats the loop the specified number of times.
loop_operation	STRING	• **use_original_addr:** While executing the loop each time, the address offset of the WRITE and READ instructions inside the LOOP command is reset to the original value. • **incr_original_addr:** While executing the loop each time, the address offset of the WRITE and READ instructions inside the LOOP command is incremented by the specified value in the loop_addr_incr_by field. When the incremented value goes out of range, the TG restarts the address from base_addr.
loop_add_incr_by	64	• **<value in hex>:** Specifies the address offset increment by value for each loop.

PHASE_DONE Command

The phase done command is used for traffic shaping across multiple TG. PHASE_DONE command execution occurs in following steps

1. Waits to receive all write and read responses.
2. Asserts the trigger_out signal.
3. Waits for trigger_in assertion by the simulation Trigger IP. The trigger_in is asserted when trigger_out from all the TGs is received.
4. De-asserts trigger_out signal.

The PHASE_DONE command, upon its execution, ensures the traffic across multiple TG gets synchronized.

***Note*:** The PHASE_DONE command does not have any command input fields.

SET_DEFAULT Command

The SET_DEFAULT command facilitates providing default values to all READ/WRITE commands to be executed. The default values for different fields of READ/WRITE command like txn_count, burst_length, burst_size, etc., can be set using SET_DEFAULT command. It saves the user from writing repetitive information in the CSV.

Table 217: **SET DEFAULT CSV Command Format for AXI3/AXI4**

Command Input Field	Width	Non-Synthesizable TG Options
default_cmd	STRING	• **READ:** Default values are applied to all READ commands. • **WRITE:** Default values are applied to all WRITE commands. • **WRRD:** Default values are applied to all WRITE and READ commands.

Table 217: **SET DEFAULT CSV Command Format for AXI3/AXI4** *(cont'd)*

Command Input Field	Width	Non-Synthesizable TG Options

default_field	STRING	• **Sets the default value for the following fields of all specified default commands:**
		  • txn_count
		  • start_delay
		  • inter_beat_delay
		  • wdata_pattern
		  • wdata_pat_value
		  • data_integrity
		  • dest_id
		  • base_addr
		  • high_addr
		  • addr_incr_by
		  • axi_addr_offset
		  • axi_len
		  • axi_size
		  • axi_id
		  • axi_burst
		  • axi_lock
		  • axi_cache
		  • axi_prot
		  • axi_qos
		  • axi_region
		  • axi_user
		  • exp_resp
		  • bandwidth
		  • addr_mask
		Note: You can specify addr_mask_en by bit position in the default_value field. You can specify addr_mask_value by bit position in the default_value_2 field.

Table 217: **SET DEFAULT CSV Command Format for AXI3/AXI4** *(cont'd)*

Command Input Field	Width	Non-Synthesizable TG Options
default_value	STRING/32	• <txn_count value in decimal> • <start_delay value in number of clocks> • <inter_beat_delay value in number of clocks> • <wdata_pattern value in string> • <wdata_pat_value value in hex> • <data_integrity value in string> • <dest_id value in hex> • <base_addr value in hex • <high_addr value in hex> • <addr_incr_by value in hex> • <axi_addr_offset value in hex> • <axi_len value in hex> • <axi_size value in hex> • <axi_id value in hex> • <axi_burst value in hex> • <axi_ lock value in hex> • <axi_cache value in hex> • <axi_prot value in hex> • <axi_qos value in hex • <axi_region value in hex> • <axi_user value in hex> • <exp_resp value in string> • <bandwidth value in MB/s> • <addr mask enable by bit position, value in hex>: Set 1 in the bit position of address to be masked (bitwise granularity). The mask value to this address bit position can be mentioned in the next column (default_value_2). For example, if bit[8] of axi_addr needs to be masked always with the value of 0 and bit[7] of axi_addr needs to be masked always with the value of 1, set this field as 0x0000_0000_0000_0180 and set the next column as 0x0000_0000_0000_0080

Table 217: **SET DEFAULT CSV Command Format for AXI3/AXI4** *(cont'd)*

Command Input Field	Width	Non-Synthesizable TG Options
default_value_2	STRING/64	• **<addr mask value by bit position, value in hex>:** The address mask value can be mentioned in this field (bitwise granularity). For example, if bit[8] of axi_addr needs to be masked for all transactions with the value of 1 and bit [16] needs to be masked with 0, set this field as 0x0000_0000_0000_0100 and set the previous column (default_value) as 0x0000_0000_0001_0100.

DISPLAY Command

Command Input Field	Width	Non-Synthesizable TG Options
display_message	STRING	• **<Message>:** The specified message is displayed in the simulation log when this command is executed.

AXI4 Stream Traffic Pattern CSV Format

STREAM Command

Table 218: **STREAM CSV Command Format for AXI4-Stream**

Command Input Field	Width	Non-Synthesizable TG Options
pkt_count	32/STRING	• **<value in decimal>:** Stream packet count. Repeat the current command <N> times, where N = value. • **<value in decimal>KB:** Amount of data to be transferred in KB, for example 50 KB, 100 KB. • **<value in decimal>MB:** Amount of data to be transferred in MB, for example 5 MB, 10 MB. • **<value in decimal>GB:** Amount of data to be transferred in GB, for example 1 GB, 2 GB. • **INF:** Repeat the current command infinitely.

Table 218: **STREAM CSV Command Format for AXI4-Stream** *(cont'd)*

Command Input Field	Width	Non-Synthesizable TG Options
inter_pkt_delay	32/STRING	• **<value in decimal>:** Value in number of clocks. Issue transaction after specified number of clocks. • **<value in decimal>MBps:** Bandwidth value for Stream transactions, for example 12500 MB/s. ***Note*:** 1 MB/s = 1,000,000 Bps (bytes per second).
inter_transfer_delay	32	• **<value in decimal>:** Value in number of clocks. Adds specified number of clocks as delay between two transfers in a Stream packet.
tdata_pattern	STRING	• constant • random • hammer • byte_incr • 16byte_incr • walking_1 • walking_0 • same_as_src • same_as_len • same_as_id
tdata_pat_value	512	• **<value in hex>:** When the tdata_pattern is selected as constant, this field value is used as constant data that is sent on all Stream packet data transfers (tdata_pattern: constant). • **<seed_value in hex>:** When the tdata_pattern is selected as random, this field value is used as seed for RNG (random number generator) (tdata_pattern: random).
Reserved	-	-
noc_dest_id	12/STRING	• **<value in hex>:** The targeted slave destination ID value (12-bit value) can be given in this field for the Stream packet. • **<Slave Logical Name> :** The slave logical name is the AXI stream slave interface name generated by the Vivado IDE. (for example, M00_AXIS, M01_AXIS, M02_AXIS, ...). The TG internally maps the logical name with the corresponding NoC slave destination ID. ***Note*:** The noc_dest_id value will be sent on the NoC signal (nmu_wr_usr_dst) along with AXI4-Stream packets.

Table 218: **STREAM CSV Command Format for AXI4-Stream** *(cont'd)*

Command Input Field	Width	Non-Synthesizable TG Options
Reserved	-	-
Reserved	-	-
Reserved	-	-
tdest_id	12	• **<value in hex> :** This value will be sent on the axis_tdest signal for all Stream packets.
pkt_len	16	• **<value in hex>:** Stream packet transactions are sent with N+1 transfer where N = length value in this field. ***Note*:** The axi_tlast signal is asserted High for every final transfer in the packet.
Reserved	-	-
pkt_id	STRING/16	• **<value in hex>:** All Stream packet transactions is sent with this ID. • **auto_incr:** The first Stream packet transaction nt out with an ID of 0 and each consecutive transaction packet ID is incremented by 1. When the packet ID reaches the MAX_ID, the next transaction ID starts from 0 and increments again. ***Note*:** MAX_ID = 2^ AXI ID Width) – 1
Reserved	-	-
pkt_user	64	• **<value in hex>:** This value is transferred in the axis_tuser signal for all Stream transfers.

WAIT Command

Table 219: **WAIT CSV Command Format for AXI4-Stream**

Command Input Field	Width	Non-Synthesizable TG Options
wait_option	STRING/32	• **all_req_sent :** Wait till all Stream requests are accepted by the slave. • **<wait_period by clocks, value in decimal>:** Wait by specified number of AXI clocks. The wait_unit "clk" must be specified in next column. For example, if you want to wait for 50 AXI clock cycles, set 50 in this field and set "clk" in the wait_unit field. • **<wait_period by time scale, value in decimal> :** Wait by specified time period. The required wait_unit (ps/ns/us/ms) must be specified in next column. For example, if you want to wait for 10 milliseconds, set 10 in this field and set "ms" in the wait_unit field.

Table 219: **WAIT CSV Command Format for AXI4-Stream** *(cont'd)*

Command Input Field	Width	Non-Synthesizable TG Options
wait_unit	STRING	• **clk:** Wait unit to specify the wait_period in clocks. • **ps, ns, us, ms:** Wait unit to specify the wait_period in time scale.

START_LOOP/END_LOOP

Table 220: **LOOP CSV Command Format for AXI4-Stream**

Command Input Field	Width	Non-Synthesizable TG Options
loop_count	32	• **<value in decimal>:** Repeats the loop the specified number of times.

PHASE_DONE Command

***Note*:** The PHASE_DONE command does not have any command input fields.

SET_DEFAULT Command

Table 221: **SET DEFAULT CSV Command Format for AXI4-Stream**

Command Input Field	Width	Non-Synthesizable TG Options
default_cmd	STRING	• **STREAM:** Default values are applied to all STREAM commands.

Table 221: **SET DEFAULT CSV Command Format for AXI4-Stream** *(cont'd)*

Command Input Field	Width	Non-Synthesizable TG Options

default_field	STRING	• **Sets the default value for the following fields of all STREAM commands:**
		  • pkt_count
		  • inter_pkt_delay
		  • inter_transfer_delay
		  • tdata_pattern
		  • tdata_pat_value
		  • noc_dest_id
		  • test_id
		  • pkt_id
		  • pkt_len
		  • pkt_user
		  • bandwidth
		  .

default_value	STRING/32	• <pkt_count value in decimal> • <inter_pkt_delay value in number of clocks> • <inter_transfer_delay value in number of clocks> • <tdata_pattern value in string> • <tdata_pat_value value in hex> • <noc_dest_id value in hex> • <tdest_id value in hex> • <pkt_id value in hex> • <pkt_len value in hex> • <pkt_user value in hex> • <bandwidth value in MB/s> • <in decimal> • <in hex>

DISPLAY Command

Table 222: **DISPLAY CSV Command Format for AXI4-Stream**

Command Input Field	Width	Non-Synthesizable TG Options
Display_message	STRING	• **<Message>:** The specified message is displayed in the simulation log when this command is executed.

CSV Command Usage Examples for AXI3/AXI4

Wait Command

This CSV has Write, Wait, and Read commands with a transaction count of 1000 for Write and Read. The start delay is 0 . The field `wdata_pat_value` is 0, that means user data 00000000. Data integrity is enabled. The base address and high address are `0xbc16_1000_0000` and `0xbc1e_FFFF_FFFF` respectively. The field `addr_incr_by` is `auto_incr` which indicates auto increment of address.

The Wait command after the Write command holds the next instruction until all the Write instructions are finished and responses are received. The next instruction is then issued, which is Read in this example.

Figure 49: **AXI3/AXI4 Wait Command CSV Example**

#Any line starting with # will be commented out																					
TG_NUM	cmd	txn_count	start_dela	inter_bea	wdata_pa	wdata_	data_inte	dest_id	base_addi	high_addr	addr_inc_	axi_addr	axi_len	axi_bur	axi_lock	axi_ca	axi_pr	axi_qc	axi_regi	axi_user
1	WRITE	1000	0	0	same_as_	0	enabled	0	bc1e_100(	bc1e_FFFF	auto_incr	0x0001_00(	A	FIXED	NORMAL	0	0	0	0	0
1	WAIT																			
1	READ	1000	0	0	same_as_	0	enabled	0	bc1e_100(	bc1e_FFFF	auto_incr	0x0001_00(	A	FIXED	NORMAL	0	0	0	0	0

Write Command

This CSV has a single Write command with a transaction count of 1000. The start delay is 0 . The field `wdata_pat_value` is 0 which means user data 00000000. The base address and high address are `0xbc16_1000_0000` and `0xbc1e_FFFF_FFFF` respectively. The field `addr_incr_by` is `auto_incr` which indicates auto increment of address.

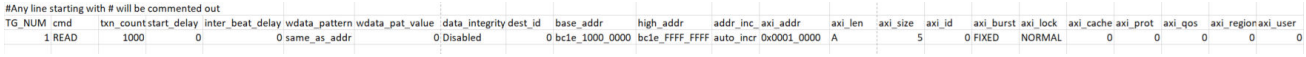
Figure 50: **AXI3/AXI4 Write Command CSV Example**

#Any line starting with # will be commented out																				
TG_NUM	cmd	txn_count	start_dela	inter_beat_	wdata_pattern	wdata_pat_value	data_inte	dest_id	base_addr	high_addr	addr_inc_by	axi_addr	axi_len	axi_size	axi_id	axi_burst	axi_lock	axi_cache	axi_prot	axi_qos	axi_region	axi_user
1	WRITE	1000	0	0	same_as_addr	0	enabled	0	bc1e_1000_0000	bc1e_FFFF_FFFF	auto_incr	0x0001_0000	A	5	0	FIXED	NORMAL	0	0	0	0	0

Read Command

This CSV has a single Read command with a transaction count of 1000. The start delay is 0. The field `wdata_pat_value` is 0 which means user data 00000000. Data integrity is disabled. The base address and high address are `0xbc16_1000_0000` and `0xbc1e_FFFF_FFFF` respectively. The field `addr_incr_by` is `auto_incr` which indicates auto increment of address.

Figure 51: **AXI3/AXI4 Read Command Example**



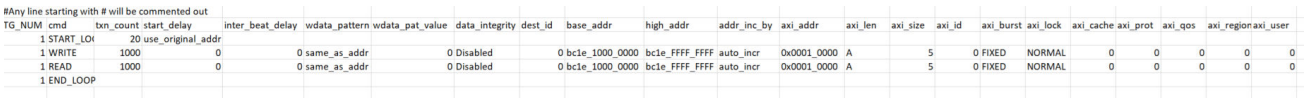
#Any line starting with # will be commented out

TG_NUM	cmd	txn_count	start_delay	inter_beat_delay	wdata_pattern	wdata_pat_value	data_integrity	dest_id	base_addr	high_addr	addr_inc_	axi_addr	axi_len	axi_size	axi_id	axi_burst	axi_lock	axi_cache	axi_prot	axi_qos	axi_region	axi_user
1	READ	1000	0	0	same_as_addr	0	Disabled	0	bc1e_1000_0000	bc1e_FFFF_FFFF	auto_incr	0x0001_0000	A	5	0	FIXED	NORMAL	0	0	0	0	0

Loop Command (START_LOOP/END_LOOP)

This CSV has Write and Read commands with a transaction count of 1000 each and a Loop command with a loop count of 20. The start delay is 0. The START_LOOP/END_LOOP instruction does the loop operation. The instructions that are defined between START_LOOP and END_LOOP instruction are executed in a given order in loop by the number of times mentioned on loop_count variable. On each loop, the command input field value of associated instructions are updated based on other loop variables, if applicable. This CSV has Write and Read commands with a transaction count of 1000 each and a Loop command with a loop count of 20. So, 1000 write transaction and 1000 read transaction are carried out 20 times.

Figure 52: **AXI3/AXI4 Loop Command CSV Example**



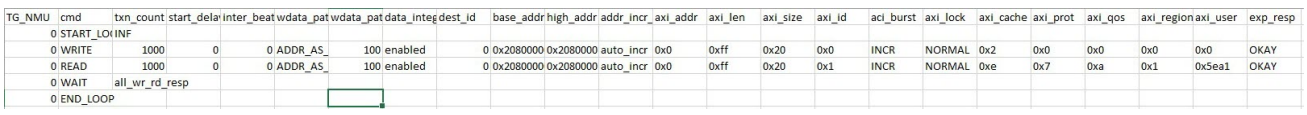
#Any line starting with # will be commented out

TG_NUM	cmd	txn_count	start_delay	inter_beat_delay	wdata_pattern	wdata_pat_value	data_integrity	dest_id	base_addr	high_addr	addr_inc_by	axi_addr	axi_len	axi_size	axi_id	axi_burst	axi_lock	axi_cache	axi_prot	axi_qos	axi_region	axi_user
1	START_LO		20	use_original_addr																		
1	WRITE	1000	0	0	same_as_addr	0	Disabled	0	bc1e_1000_0000	bc1e_FFFF_FFFF	auto_incr	0x0001_0000	A	5	0	FIXED	NORMAL	0	0	0	0	0
1	READ	1000	0	0	same_as_addr	0	Disabled	0	bc1e_1000_0000	bc1e_FFFF_FFFF	auto_incr	0x0001_0000	A	5	0	FIXED	NORMAL	0	0	0	0	0
1	END_LOOP																					

Infinite Loop

This CSV has Write and Read commands, each with a transaction count of 1000, with infinite loop. The start delay is 0. The field `wdata_pat_value` is 100, that means the user data is the same as the address. Data integrity is enabled. The base address and high address are `0x208_0000_0000` and `0x208_FFFF_FFFF` respectively. The field `addr_incr_by` is `auto_incr` which indicates auto increment of address. The Write and Read commands (each with a transaction count of 1000 for each run) repeat infinitely.

Figure 53: **AXI3/AXI4 Infinite Command CSV Example**



TG_NMU	cmd	txn_count	start_dela	inter_beat	wdata_pat	wdata_pat	data_integ	dest_id	base_addr	high_addr	addr_incr_	axi_addr	axi_len	axi_size	axi_id	axi_burst	axi_lock	axi_cache	axi_prot	axi_qos	axi_region	axi_user	exp_resp
0	START_LO	INF																					
0	WRITE	1000	0	0	ADDR_AS_	100	enabled	0	0x2080000	0x2080000	auto_incr	0x0	0xff	0x20	0x0	INCR	NORMAL	0x2	0x0	0x0	0x0	0x0	OKAY
0	READ	1000	0	0	ADDR_AS_	100	enabled	0	0x2080000	0x2080000	auto_incr	0x0	0xff	0x20	0x1	INCR	NORMAL	0xe	0x7	0xa	0x1	0x5ea1	OKAY
0	WAIT	all_wr_rd_resp																					
0	END_LOOP																						

CSV Command Usage Examples for AXI4-Stream

Stream Command

- Issues 800 stream packets (`pkt_count` = 800), each with eight transfers (`pkt_len` = 8), containing HAMMER data to the slave that has a destination ID of 'h08 (`tdest_id` = 8).
- The packet user value for each transfer is 'hF (`pkt_user` =F).
- The packet ID value for each transfer is 'h8 (`pkt_user` = 8).

Figure 54: **AXI4-Stream Stream Command CSV Example**

#pkt_count -> decimal, inter_pkt_delay-> decimal, inter_transfer_delay -> decimal, #tdtestid-> in hex, pkt_len-> in hex, pkt_id -> in hex, pkt_usr -> hex

TG_NUM	cmd	pkt_count	inter_pkt_delay	inter_transfer_delay	tdata_pattern	tdata_pat_value	tdest_id	pkt_len	pkt_id	pkt_user
1	STREAM	800	0	0	HAMMER		0x8	0x8	0x8	0xf

Wait Command

The WAIT command inserts 10 clocks between the TLAST of the 800th packet of the command in line 1, and the assertion of TVALID for the first packet of the command in line 3.

Figure 55: **AXI4-Stream Wait Command CSV Example**

#pkt_count -> decimal, inter_pkt_delay-> decimal, inter_transfer_delay -> decimal, #tdestid-> in hex, pkt_len-> in hex, pkt_id -> in hex, pkt_usr -> hex

TG_NUM	cmd	pkt_count	inter_pkt_delay	inter_transfer_delay	tdata_pattern	tdata_pat_value	tdest_id	pkt_len	pkt_id	pkt_user
1	STREAM	800	0	0	hammer		0x8	0x8	0x8	0xF
1	WAIT	10								
1	STREAM	800	0	0	hammer		0x8	0x8	0x8	0xF

Loop Command (START_LOOP/END_LOOP)

The following figure shows 800 stream packets that is looped 100 times (`loop_count` = 100).

Figure 56: **AXI4-Stream Loop Command CSV Example**

TG_NUM	cmd	pkt_count	inter_pkt_delay	inter_transfer_delay	tdata_pattern	tdata_pat_value	tdest_id	pkt_len	pkt_id	pkt_user
1	START_LOOP	100								
1	STREAM	800	0	0	hammer		0x8	0x8	0x8	0xF
1	END_LOOP									

Data Patterns

Data patterns for AXI3/AXI4

The following data patterns are supported by the Non-Synthesizable TG for AXI3/AXI4.

Table 223: **Data Patterns for the Non-Synthesizable TG**

Data Format	Options
Constant	• The user-defined data beat value is constantly sent on all Write data beats. • In the case of `user_defined_pattern_test`, you can select the constant data pattern by setting the `wdata_pattern` CSV field to constant and setting the data value of the `wdata_pat_value` field. • In the case of `pre_defined_pattern_test`, you can select the constant data pattern by setting the Write Data Pattern Type GUI option to `CONSTANT_DATA` and setting the data value on the Write Data Pattern Value GUI option. • For example, if Write Data Pattern Value = 0xA5A6_A7A8_B5B6_B7B8, `AXI Data Width` = 64, and assuming it is full transfer, all the values of the given instruction will be sent as 0xA5A6_A7A8_B5B6_B7B8.
Random	• The random data is sent on each Write data beat. The default seed value of 0 is used to generate random data. • In the case of `user_defined_pattern_test`, you can select the random data pattern by setting the `wdata_pattern` CSV field to random and setting the seed value on the `wdata_pat_value` field. • In the case of `pre_defined_pattern_test`, you can select the random data pattern by setting the Write Data Pattern Type GUI option to `RANDOM_DATA`. The default seed value of 0 is used to generate random data.

Table 223: **Data Patterns for the Non-Synthesizable TG** *(cont'd)*

Data Format	Options
WALKING_0	• Any one bit on the data beat has a value of 0 and the rest have a value of 1. The value 0 walks through (left in a circular fashion shifted by 1) bitwise in each beat. The LSB of the first data beat on the first transaction has a value of 0. From the second beat onwards, the previous data beat is shifted left in a circular fashion. • You can select the walking_0 data pattern by setting the `wdata_pattern` CSV field to `walking_0` in the case of `user_defined_pattern_test` and by setting the Write Data Pattern Type GUI option to `WALKING_0_DATA` in the case of pre_defined_pattern_test. • For example, if `axi_len` = 8 and `axi_size` = 0, the `walking_0` data is sent as follows (shown below the data for the valid byte lane): ◦ Beat0 = 8'b1111_1110 ◦ Beat1 = 8'b1111_1101 ◦ Beat2 = 8'b1111_1011 ◦ Beat3 = 8'b1111_0111 ◦ Beat4 = 8'b1110_1111 ◦ Beat5 = 8'b1101_1111 ◦ Beat6 = 8'b1011_1111 ◦ Beat7 = 8'b0111_1111 ◦ Beat8 = 8'b1111_1110

Table 223: **Data Patterns for the Non-Synthesizable TG** *(cont'd)*

Data Format	Options
WALKING_1	• Any one bit on the data beat has a value of 1 and the rest have a value of 0. The value 0 walks through (left in a circular fashion shifted by 1) bitwise in each beat. The LSB of the first data beat on the first transaction has a value of 1. From the second beat onwards, the previous data beat is shifted left in a circular fashion. • You can select the walking_1 data pattern by setting the wdata_pattern CSV field to `walking_1` in the case of `user_defined_pattern_test` and by setting the Write Data Pattern Type GUI option to `WALKING_1_DATA` in the case of `pre_defined_pattern_test`. • For example, if `axi_len` = 8 and `axi_size` = 0, the `walking_1` data is sent as follows (shown below the data for the valid bytelane): ◦ Beat0 = 8'b0000_0001 ◦ Beat1 = 8'b0000_0010 ◦ Beat2 = 8'b0000_0100 ◦ Beat3 = 8'b0000_1000 ◦ Beat4 = 8'b0001_0000 ◦ Beat5 = 8'b0010_0000 ◦ Beat6 = 8'b0100_0000 ◦ Beat7 = 8'b1000_0000 ◦ Beat8 = 8'b0000_0001
Hammer	• The hammer data has a long number of tail bits (MSBs) with a value of 0 or 1 and a short number of header bits (LSBs) with an inverted bit value in a data beat. The value of the tail bits (width of ¾ of `axi_size_in_bits`) and header bits (width of ¼ of `axi_size_in_bits`) is generated based on the corresponding AXI beat address. If the resulting value of `beat_start_address` divided by axi_size_in_bytes is even, the header bits are 1 and the tail bits are 0. If the resulting value is odd, the header bits are 0 and the tail bits are 1. • You can select the hammer data pattern by setting the `wdata_pattern` CSV field to hammer in the case of `user_defined_pattern_test` and by setting Write Data Pattern Type GUI option to `HAMMER_DATA` in the case of `pre_defined_pattern_test`.

Table 223: **Data Patterns for the Non-Synthesizable TG** *(cont'd)*

Data Format	Options
Hammer (continued)	For example, if AXI Data Width = 64, `axi_addr` = 0x0000_0000_A11A, `axi_burst` = INCR, `axi_len` = 3, and `axi_size` = 3, data beats are generated as follows: • `header_width` = `axi_size_in_bits`/4 = 64/4 = 16 • `tail_width` = (`axi_size_in_bits`*3)/4 = (64*3)/4 = 48 • Beat0 calculation: Aligned(`beat_start_address`)/`axi_size_in_bytes`= 0x0000_0000_A118/8 = 0x1423(odd value) So, header bits are 16{{1'b0}} and tail bits are 48{{1'b1}}. Beat0 = 0xFFFF_FFFF_FFFF_0000 • Beat1 calculation: Aligned(`beat_start_address`)/`axi_size_in_bytes`= 0x0000_0000_A120/8 = 0x1424(even value) So, header bits are 16{{1'b1}} and tail bits are 48{{1'b0}}. Beat1 = 0x0000_0000_0000_FFFF • Beat2 calculation: Aligned(`beat_start_address`)/`axi_size_in_bytes`= 0x0000_0000_A128/8 = 0x1425(odd value) So, header bits are 16{{1'b0}} and tail bits are 48{{1'b1}}. Beat2 = 0xFFFF_FFFF_FFFF_0000 • Beat3 calculation: aligned(`beat_start_address`)/`axi_size_in_bytes`= 0x0000_0000_1130/8 = 0x1426(even value) So, header bits are 16{{1'b1}} and tail bits are 48{{1'b0}}. Beat3 = 0x0000_0000_0000_FFFF
SAME_AS_SRC	• The TG source ID value is sent as data on all Write data beats. • You can select the `same_as_src` data pattern by setting the `wdata_pattern` CSV field to `same_as_src` in the case of `user_defined_pattern_test` and by setting the Write Data Pattern Type GUI option to `SRC_ID_AS_DATA` in the case of `pre_defined_pattern_test`.

Table 223: **Data Patterns for the Non-Synthesizable TG** *(cont'd)*

Data Format	Options
SAME_AS_ADDR	• The corresponding AXI beat byte address (LSB 8 bits) is sent as data in the bytes of the Write beat. • You can select the `same_as_addr` data pattern by setting the `wdata_pattern` CSV field to `same_as_addr` in the case of `user_defined_pattern_test` and by setting the Write Data Pattern Type GUI option to `ADDR_AS_DATA` in the case of `pre_defined_pattern_test`. • For example, if AXI Data Width = 64, `axi_addr` = 0x0200_0000_11A0, `axi_burst` = INCR, `axi_len` = 3, and `axi_size` = 3, data beats are generated as follows: ◦ Byte_Lane0 address on Beat0= 0x0200_0000_11A0 --> LSB 8 bits are A0 which are sent as Write data on byte_lane0 of beat0 ◦ Byte_Lane1 address on Beat0= 0x0200_0000_11A1 --> LSB 8 bits are A1 which are sent as Write data on byte_lane1 of beat0 ◦ Beat0 = 0xA7A6_A5A4_A3A2_A1A0 ◦ Beat1 = 0xAFAE_ADAC_ABAA_A9A8 ◦ Beat2 = 0xB7B6_B5B4_B3B2_B1B0 ◦ Beat3 = 0xBFBE_BDBC_BBBA_B9B8
SAME_AS_ADDR_XOR	• The corresponding AXI beat byte address is bytewise folded, XORed, and sent as data in the bytes of the Write beat. • You can select the `same_as_addr_xor` data pattern by setting the `wdata_pattern` CSV field to `same_as_addr_xor` in the case of `user_defined_pattern_test` and by setting the Write Data Pattern Type GUI option to `ADDR_XOR_AS_DATA` in the case of `pre_defined_pattern_test` . • For example, if `AXI Data Width` = 64, `axi_addr` = 0x0200_0000_11A0, `axi_burst` = INCR, `axi_len` = 3, and `axi_size` =3, data beats are generated as follows: ◦ Byte_Lane0 address on Beat0= 0x0200_0000_11A0 --> 0x02^0x00^0x00^0x00^0x00^0x11^0xA0= 0x B3. It will be sent as Write data on byte_lane0 of beat0. ◦ Byte_Lane1 address on Beat0= 0x0200_0000_11A1 --> 0x02^0x00^0x00^0x00^0x00^0x11^0xA1= 0x B2. It will be sent as Write data on byte_lane1 of beat0. ◦ Beat0 = 0xB4B5_B6B7_B0B1_B2B3 ◦ Beat1 = 0xBCBD_BEBF_B8B9_BABB ◦ Beat2 = 0xA4A5_A6A7_A0A1_A2A3 ◦ Beat3 = 0xACAD_AEAF_A8A9_AAAB

Table 223: **Data Patterns for the Non-Synthesizable TG** *(cont'd)*

Data Format	Options
SAME_AS_ID	• The corresponding AXI transaction ID value (AWID) is sent as data on all Write data beats. • You can select the same_as_id data pattern by setting the wdata_pattern CSV field to same_as_id in the case of user_defined_pattern_test and by setting the Write Data Pattern Type GUI option to AXI_ID_AS_DATA in the case of pre_defined_pattern_test.
SAME_AS_BURST	• The corresponding AXI transaction burst value (AWBURST) is sent as data on all Write data beats. • You can select the same_as_burst data pattern by setting the wdata_pattern CSV field to same_as_burst in the case of user_defined_pattern_test and by setting the Write Data Pattern Type GUI option to AXI_BURST_AS_DATA in the case of pre_defined_pattern_test.
SAME_AS_LEN	• The corresponding AXI transaction length value (AWLEN) is sent as data on all Write data beats. • You can select the same_as_len data pattern by setting the wdata_pattern CSV field to same_as_len in the case of user_defined_pattern_test and by setting the Write Data Pattern Type GUI option to AXI_LEN_AS_DATA in the case of pre_defined_pattern_test.
SAME_AS_SIZE	• The corresponding AXI transfer size value (AWSIZE) is sent as data on all Write data beats. • You can select the same_as_size data pattern by setting the wdata_pattern CSV field to same_as_size in the case of user_defined_pattern_test and by setting the Write Data Pattern Type GUI option to AXI_SIZE_AS_DATA in the case of pre_defined_pattern_test.
SAME_AS_CACHE	• The corresponding AXI transaction cache value (AWCACHE) is sent as data on all Write data beats. • You can select the same_as_cache data pattern by setting the wdata_pattern CSV field to same_as_cache in the case of user_defined_pattern_test and by setting the Write Data Pattern Type GUI option to AXI_CACHE_AS_DATA in the case of pre_defined_pattern_test.

Data Patterns for AXI4-Stream

The following data patterns are supported by the Non-Synthesizable TG for AXI4-Stream.

Table 224: **Data Patterns for Non-Synthesizable TG AXI4-Stream**

Data Pattern	Options
CONSTANT DATA	• The user defined data bytes (max 512-bit wide) is constantly sent on all stream transfer. • In case of user_defined_pattern_test, the user can select the constant data pattern by setting the tdata_pattern csv field to constant and can set the data value on tdata_pat_value field. • In case of pre_defined_pattern_test, the user can select the constant data pattern by setting the GUI parameter USER_C_AXI_TDATA_PATTERN to CONSTANT_DATA and can set the data value on USER_C_AXI_TDATA_VALUE GUI parameter. • For example, if tdata_pat_value= 0x3637_3839_4041_4243_4445_4647 and AXI Data Width= 128, all the stream transfer values of the given instruction is sent as 0x0000_0000_3637_3839_4041_4243_4445_4647.
RANDOM DATA	• The random data is sent on each stream transfer. The seed value can be set to generate different random patterns. • In case of user_defined_pattern_test, the user can select the random data pattern by setting the tdata_pattern csv field to random and can set the seed value on tdata_pat_value field. • In case of pre_defined_pattern_test, the user can select the random data pattern by setting the GUI parameter USER_C_AXI_TDATA_PATTERN to “RANDOM_DATA” and can set the seed value on USER_C_AXI_TDATA_VALUE GUI parameter.

Table 224: **Data Patterns for Non-Synthesizable TG AXI4-Stream** *(cont'd)*

Data Pattern	Options
HAMMER DATA	• The hammer data has long number of tail bits (MSbits) with value of 0 or 1 and short number of header bits (LSbits) with inverted bit value in a transfer. In a first stream packet, the value of tail bits (wide of ¾th of data_width_bits) and header bits (wide of ¼th of data_width_bits) of first stream transfer are 0 and 1 respectively. From the second transfer onwards, each stream transfer value is generated as inverse of previous stream transfer irrespective of TLAST value. • The user can select the hammer data pattern by setting the tdata_pattern csv field to hammer in case of user_defined_pattern_test and by setting GUI parameter USER_C_AXI_TDATA_PATTERN to HAMMER_DATA in case of pre_defined_pattern_test. • For example, if AXI Data Width= 128, pkt_cnt=2 and pkt_len=0x4, stream transfers are generated as below: header_width = data_width_bits/4 = 128/4 = 32 tail_width = (data_width_bits*3)/4 = (128*3)/4 = 96 In a first stream packet, the value of 1st stream transfer is calculated as header bits of 32{{1'b1}} and tail bits of 96{{1'b0}}. From 2nd transfer onwards, the previous tdata values are inverted on each transfer irrespective of TLAST value. TDATA = 0x0000_0000_0000_0000_FFFF_FFFF, TLAST = 0 TDATA = 0xFFFF _FFFF_FFFF _ FFFF _0000_0000, TLAST = 0 TDATA = 0x0000_0000_0000_0000_ FFFF _FFFF, TLAST = 0 TDATA = 0xFFFF _FFFF_FFFF _ FFFF _0000_0000, TLAST = 0 TDATA = 0x000_0000_0000_0000_ FFFF _FFFF, TLAST = 1 TDATA = 0xFFFF _FFFF_FFFF _ FFFF _0000_0000, TLAST = 0 TDATA = 0x000_0000_0000_0000_ FFFF _FFFF, TLAST = 0 TDATA = 0xFFFF _FFFF_FFFF _ FFFF _0000_0000, TLAST = 0 TDATA = 0x000_0000_0000_0000_ FFFF _FFFF, TLAST = 0 TDATA = 0xFFFF _FFFF_FFFF _ FFFF _0000_0000, TLAST = 1
BYTE_INCR DATA	• Every byte in the stream transfer has an incremented value. In each stream packet, the LSbyte value of first transfer is 0x00 and every other byte of transfers has an incremented value. • The user can select the byte_incr data pattern by setting the tdata_pattern csv field to byte_incr in case of user_defined_pattern_test and by setting GUI parameter USER_C_AXI_TDATA_PATTERN to BYTE_INCR_DATA in case of pre_defined_pattern_test. • For example, if AXI Data Width= 128, pkt_cnt=2 and pkt_len=0x2, stream transfers are generated as below: TDATA= 0x0F0E_0D0C_0B0A_0908_0706_0504_0302_0100, TLAST= 0 TDATA= 0x1F1E_1D1C_1B1A_1918_1716_1514_1312_1110, TLAST= 0 TDATA= 0x2F2E_2D2C_2B2A_2928_2726_2524_2322_2120, TLAST= 1 TDATA= 0x0F0E_0D0C_0B0A_0908_0706_0504_0302_0100, TLAST= 0 TDATA= 0x1F1E_1D1C_1B1A_1918_1716_1514_1312_1110, TLAST= 0 TDATA= 0x2F2E_2D2C_2B2A_2928_2726_2524_2322_2120, TLAST= 1

Table 224: **Data Patterns for Non-Synthesizable TG AXI4-Stream** *(cont'd)*

Data Pattern	Options
16BYTE_INCR DATA	• Every 16 byte in the stream transfer has an incremented value. In each stream packet, the value of Least significant 16 bytes of first transfer is 0x0 and every other set of 16 bytes of transfers has an incremented value. ***Note***: 16byte_incr data pattern is only supported on 128-bit, 256-bit and 512-bit data width designs. • The user can select the 16byte_incr data pattern by setting the tdata_pattern csv field to 16byte_incr in case of user_defined_pattern_test and by setting GUI parameter USER_C_AXI_TDATA_PATTERN to 16BYTE_INCR_DATA in case of pre_defined_pattern_test. • Example1, if AXI Data Width= 128, pkt_cnt=2 and pkt_len=0x3, stream transfers will be generated as below: TDATA= 0x0000_0000_0000_0000_0000_0000_0000_0000, TLAST= 0 TDATA= 0x0000_0000_0000_0000_0000_0000_0000_0001, TLAST= 0 TDATA= 0x0000_0000_0000_0000_0000_0000_0000_0002, TLAST= 0 TDATA= 0x0000_0000_0000_0000_0000_0000_0000_0003, TLAST= 1 TDATA= 0x0000_0000_0000_0000_0000_0000_0000_0000, TLAST= 0 TDATA= 0x0000_0000_0000_0000_0000_0000_0000_0001, TLAST= 0 TDATA= 0x0000_0000_0000_0000_0000_0000_0000_0002, TLAST= 0 TDATA= 0x0000_0000_0000_0000_0000_0000_0000_0003, TLAST= 1 • Example2, if USER_C_AXI_DATA_WIDTH= 256, pkt_cnt=2 and pkt_len=0x3, stream transfers will be generated as below: TDATA= 0x0000_0000_0000_0000_0000_0000_0000_0001_0000_0000_0000_0000_0000_0000_0000_0000, TLAST= 0 TDATA= 0x0000_0000_0000_0000_0000_0000_0000_0003_0000_0000_0000_0000_0000_0000_0000_0002, TLAST= 0 TDATA= 0x0000_0000_0000_0000_0000_0000_0000_0005_0000_0000_0000_0000_0000_0000_0000_0004, TLAST= 0 TDATA= 0x0000_0000_0000_0000_0000_0000_0000_0007_0000_0000_0000_0000_0000_0000_0000_0006, TLAST= 1 TDATA= 0x0000_0000_0000_0000_0000_0000_0000_0001_0000_0000_0000_0000_0000_0000_0000_0000, TLAST= 0 TDATA= 0x0000_0000_0000_0000_0000_0000_0000_0003_0000_0000_0000_0000_0000_0000_0000_0002, TLAST= 0 TDATA= 0x0000_0000_0000_0000_0000_0000_0000_0005_0000_0000_0000_0000_0000_0000_0000_0004, TLAST= 0 TDATA= 0x0000_0000_0000_0000_0000_0000_0000_0007_0000_0000_0000_0000_0000_0000_0000_0006, TLAST= 1

Table 224: **Data Patterns for Non-Synthesizable TG AXI4-Stream** *(cont'd)*

Data Pattern	Options
WALKING_0 DATA	• Any one bit on stream transfer has the value as 0 and rest all bits have the value as 1. The value 0 will walk through (that is, circularly left shifted by 1 position) bit wise in each transfer. The LSbit of first stream transfer on first stream packet has the value as 0. From the second transfer onwards, previous stream transfer is circularly left shifted irrespective of TLAST value. • The user can select the walking_0 data pattern by setting the “tdata_pattern” csv field to “walking_0” in case of user_defined_pattern_test and by setting GUI parameter USER_C_AXI_TDATA_PATTERN to “WALKING_0_DATA” in case of pre_defined_pattern_test. • For example, if AXI Data Width= 32, pkt_cnt=9 and pkt_len=0x3, stream transfers are generated as below: TDATA = 32‘b1111_1111_1111_1111_1111_1111_1111_1110, TLAST = 0 TDATA = 32‘b1111_1111_1111_1111_1111_1111_1111_1101, TLAST = 0 TDATA = 32‘b1111_1111_1111_1111_1111_1111_1111_1011, TLAST = 0 TDATA = 32‘b1111_1111_1111_1111_1111_1111_1111_0111, TLAST = 1 TDATA = 32‘b1111_1111_1111_1111_1111_1111_1110_1111, TLAST = 0 TDATA = 32‘b1111_1111_1111_1111_1111_1111_1101_1111, TLAST = 0 TDATA = 32‘b1111_1111_1111_1111_1111_1111_1011_1111, TLAST = 0 TDATA = 32‘b1111_1111_1111_1111_1111_1111_0111_1111, TLAST = 1 TDATA = 32‘b1111_1111_1111_1111_1111_1110_1111_1111, TLAST = 0 TDATA = 32‘b1111_1111_1111_1111_1111_1101_1111_1111, TLAST = 0 TDATA = 32‘b1111_1111_1111_1111_1111_1011_1111_1111, TLAST = 0 TDATA = 32‘b1111_1111_1111_1111_1111_0111_1111_1111, TLAST = 1 TDATA = 32‘b1111_1111_1111_1111_1110_1111_1111_1111, TLAST = 0 TDATA = 32‘b1111_1111_1111_1111_1101_1111_1111_1111, TLAST = 0 TDATA = 32‘b1111_1111_1111_1111_1011_1111_1111_1111, TLAST = 0 TDATA = 32‘b1111_1111_1111_1111_0111_1111_1111_1111, TLAST = 1 TDATA = 32‘b1111_1111_1111_1110_1111_1111_1111_1111, TLAST = 0 TDATA = 32‘b1111_1111_1111_1101_1111_1111_1111_1111, TLAST = 0 TDATA = 32‘b1111_1111_1111_1011_1111_1111_1111_1111, TLAST = 0 TDATA = 32‘b1111_1111_1111_0111_1111_1111_1111_1111, TLAST = 1 TDATA = 32‘b1111_1111_1110_1111_1111_1111_1111_1111, TLAST = 0 TDATA = 32‘b1111_1111_1101_1111_1111_1111_1111_1111, TLAST = 0 TDATA = 32‘b1111_1111_1011_1111_1111_1111_1111_1111, TLAST = 0 TDATA = 32‘b1111_1111_0111_1111_1111_1111_1111_1111, TLAST = 1 TDATA = 32‘b1111_1110_1111_1111_1111_1111_1111_1111, TLAST = 0 TDATA = 32‘b1111_1101_1111_1111_1111_1111_1111_1111, TLAST = 0 TDATA = 32‘b1111_1011_1111_1111_1111_1111_1111_1111, TLAST = 0 TDATA = 32‘b1111_0111_1111_1111_1111_1111_1111_1111, TLAST = 1 TDATA = 32‘b1110_1111_1111_1111_1111_1111_1111_1111, TLAST = 0 TDATA = 32‘b1101_1111_1111_1111_1111_1111_1111_1111, TLAST = 0 TDATA = 32‘b1011_1111_1111_1111_1111_1111_1111_1111, TLAST = 0 TDATA = 32‘b0111_1111_1111_1111_1111_1111_1111_1111, TLAST = 1 TDATA = 32‘b1111_1111_1111_1111_1111_1111_1111_1110, TLAST = 0 TDATA = 32‘b1111_1111_1111_1111_1111_1111_1111_1101, TLAST = 0 TDATA = 32‘b1111_1111_1111_1111_1111_1111_1111_1011, TLAST = 0 TDATA = 32‘b1111_1111_1111_1111_1111_1111_1111_0111, TLAST = 1

Table 224: **Data Patterns for Non-Synthesizable TG AXI4-Stream** *(cont'd)*

Data Pattern	Options
WALKING_1 DATA	• Any one bit on stream transfer has the value as 1 and rest all bits have the value as 0. The value 1 walks through (that is, circularly left shifted by 1 position) bit wise, in each transfer. The LSbit of first stream transfer on first stream packet has the value as 1. From the second transfer onwards, previous stream transfer is circularly left shifted irrespective of TLAST value. • The user can select the walking_1 data pattern by setting the tdata_pattern csv field to walking_1 in case of user_defined_pattern_test and by setting GUI parameter USER_C_AXI_TDATA_PATTERN to WALKING_1_DATA in case of pre_defined_pattern_test. • For example, if USER_C_AXI_DATA_WIDTH= 32, pkt_cnt=9 and pkt_len=0x3, stream transfers are generated as below: TDATA = 32'b0000_0000_0000_0000_0000_0000_0000_0001, TLAST = 0 TDATA = 32'b0000_0000_0000_0000_0000_0000_0000_0010, TLAST = 0 TDATA = 32'b0000_0000_0000_0000_0000_0000_0000_0100, TLAST = 0 TDATA = 32'b0000_0000_0000_0000_0000_0000_0000_1000, TLAST = 1 TDATA = 32'b0000_0000_0000_0000_0000_0000_0001_0000, TLAST = 0 TDATA = 32'b0000_0000_0000_0000_0000_0000_0010_0000, TLAST = 0 TDATA = 32'b0000_0000_0000_0000_0000_0000_0100_0000, TLAST = 0 TDATA = 32'b0000_0000_0000_0000_0000_0000_1000_0000, TLAST = 1 TDATA = 32'b0000_0000_0000_0000_0000_0001_0000_0000, TLAST = 0 TDATA = 32'b0000_0000_0000_0000_0000_0010_0000_0000, TLAST = 0 TDATA = 32'b0000_0000_0000_0000_0000_0100_0000_0000, TLAST = 0 TDATA = 32'b0000_0000_0000_0000_0000_1000_0000_0000, TLAST = 1 TDATA = 32'b0000_0000_0000_0000_0001_0000_0000_0000, TLAST = 0 TDATA = 32'b0000_0000_0000_0000_0010_0000_0000_0000, TLAST = 0 TDATA = 32'b0000_0000_0000_0000_0100_0000_0000_0000, TLAST = 0 TDATA = 32'b0000_0000_0000_0000_1000_0000_0000_0000, TLAST = 1 TDATA = 32'b0000_0000_0000_0001_0000_0000_0000_0000, TLAST = 0 TDATA = 32'b0000_0000_0000_0010_0000_0000_0000_0000, TLAST = 0 TDATA = 32'b0000_0000_0000_0100_0000_0000_0000_0000, TLAST = 0 TDATA = 32'b0000_0000_0000_1000_0000_0000_0000_0000, TLAST = 1 TDATA = 32'b0000_0000_0001_0000_0000_0000_0000_0000, TLAST = 0 TDATA = 32'b0000_0000_0010_0000_0000_0000_0000_0000, TLAST = 0 TDATA = 32'b0000_0000_0100_0000_0000_0000_0000_0000, TLAST = 0 TDATA = 32'b0000_0000_1000_0000_0000_0000_0000_0000, TLAST = 1 TDATA = 32'b0000_0001_0000_0000_0000_0000_0000_0000, TLAST = 0 TDATA = 32'b0000_0010_0000_0000_0000_0000_0000_0000, TLAST = 0 TDATA = 32'b0000_0100_0000_0000_0000_0000_0000_0000, TLAST = 0 TDATA = 32'b0000_1000_0000_0000_0000_0000_0000_0000, TLAST = 1 TDATA = 32'b0001_0000_0000_0000_0000_0000_0000_0000, TLAST = 0 TDATA = 32'b0010_0000_0000_0000_0000_0000_0000_0000, TLAST = 0 TDATA = 32'b0100_0000_0000_0000_0000_0000_0000_0000, TLAST = 0 TDATA = 32'b1000_0000_0000_0000_0000_0000_0000_0000, TLAST = 1 TDATA = 32'b0000_0000_0000_0000_0000_0000_0000_0001, TLAST = 0 TDATA = 32'b0000_0000_0000_0000_0000_0000_0000_0010, TLAST = 0 TDATA = 32'b0000_0000_0000_0000_0000_0000_0000_0100, TLAST = 0 TDATA = 32'b0000_0000_0000_0000_0000_0000_0000_1000, TLAST = 1

Table 224: **Data Patterns for Non-Synthesizable TG AXI4-Stream** *(cont'd)*

Data Pattern	Options
SAME_AS_SRC DATA	• The TG source ID value is sent as data on all stream transfers. • The user can select the same_as_src data pattern by setting the tdata_pattern csv field to same_as_src in case of user_defined_pattern_test and by setting GUI parameter USER_C_AXI_TDATA_PATTERN to SRC_ID_AS_DATA in case of pre_defined_pattern_test. • For example, if TG Source ID value= 18 and AXI Data Width= 32, all the stream transfer values of the given instruction are sent as 0x0000_0012.
SAME_AS_ID DATA	• The corresponding stream packet ID value (TID) is sent as data on all stream transfers. • The user can select the same_as_id data pattern by setting the tdata_pattern csv field to same_as_id in case of user_defined_pattern_test and by setting GUI parameter USER_C_AXI_TDATA_PATTERN to “PKT_ID_AS_DATA” in case of pre_defined_pattern_test. • For example, if Stream Packet ID(TID)= 0x1E and AXI Data Width= 32, all the stream transfer values of the given instruction are sent as 0x0000_001E.
SAME_AS_LEN DATA	• The corresponding stream packet length value is sent as data on all stream transfers. • The user can select the same_as_len data pattern by setting the tdata_pattern csv field to same_as_len, in case of user_defined_pattern_test and by setting GUI parameter USER_C_AXI_TDATA_PATTERN to PKT_LEN_AS_DATA in case of pre_defined_pattern_test. • For example, if Stream Packet Length set as 0xFFFF and AXI Data Width= 32, all the stream transfer values of the given instruction are sent as 0x0000_FFFF.

Simulation

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

The MEM file is automatically loaded to the block RAM during simulation. There is no need to load the MEM files manually.

Appendix A

Debugging

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

If the IP requires a license key, the key must be verified. The 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 on Xilinx.com

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

Documentation

This product guide is the main document associated with the Performance AXI Traffic Generator. This guide, along with documentation related to all products that aid in the design process, can be found on the Xilinx Support web page or by using the Xilinx® Documentation Navigator. Download the Xilinx 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 a Xilinx product. Answer Records are created and maintained daily ensuring that users have access to the most accurate information available.

Answer Records for this Performance AXI Traffic Generator can be located by using the Search Support box on the main Xilinx 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 Traffic Generator

AR 75781.

Technical Support

Xilinx provides technical support on the Xilinx Community Forums for this LogiCORE™ IP product when used as described in the product documentation. Xilinx 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 Xilinx Community Forums.

XILINX

Appendix B

Additional Resources and Legal Notices

Xilinx Resources

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

Documentation Navigator and Design Hubs

Xilinx® Documentation Navigator (DocNav) provides access to Xilinx documents, videos, and support resources, which you can filter and search to find information. To open DocNav:

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

Xilinx 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.
- On the Xilinx website, see the Design Hubs page.

Note: For more information on DocNav, see the Documentation Navigator page on the Xilinx website.

References

These documents provide supplemental material useful with this guide:

1. *AMBA AXI and ACE Protocol Specification* (ARM IHI0022E)
2. *AMBA AXI4-Stream Protocol Specification* (ARM IHI 0051A)
3. *Versal ACAP Programmable Network on Chip and Integrated Memory Controller LogiCORE IP Product Guide* (PG313)

Revision History

The following table shows the revision history for this document.

Section	Revision Summary
02/12/2021 Version 1.0	
Initial release.	N/A

Please Read: Important Legal Notices

This document contains preliminary information and is subject to change without notice. Information provided herein relates to products and/or services not yet available for sale, and provided solely for information purposes and are not intended, or to be construed, as an offer for sale or an attempted commercialization of the products and/or services referred to herein.

AUTOMOTIVE APPLICATIONS DISCLAIMER

AUTOMOTIVE PRODUCTS (IDENTIFIED AS "XA" IN THE PART NUMBER) ARE NOT WARRANTED FOR USE IN THE DEPLOYMENT OF AIRBAGS OR FOR USE IN APPLICATIONS THAT AFFECT CONTROL OF A VEHICLE ("SAFETY APPLICATION") UNLESS THERE IS A SAFETY CONCEPT OR REDUNDANCY FEATURE CONSISTENT WITH THE ISO 26262 AUTOMOTIVE SAFETY STANDARD ("SAFETY DESIGN"). CUSTOMER SHALL, PRIOR TO USING OR DISTRIBUTING ANY SYSTEMS THAT INCORPORATE PRODUCTS, THOROUGHLY TEST SUCH SYSTEMS FOR SAFETY PURPOSES. USE OF PRODUCTS IN A SAFETY APPLICATION WITHOUT A SAFETY DESIGN IS FULLY AT THE RISK OF CUSTOMER, SUBJECT ONLY TO APPLICABLE LAWS AND REGULATIONS GOVERNING LIMITATIONS ON PRODUCT LIABILITY.

Copyright