Non-Confidential Proprietary Notice

This document is protected by copyright and other related rights and the practice or implementation of the information contained in this document may be protected by one or more patents or pending patent applications. No part of this document may be reproduced in any form by any means without the express prior written permission of ARM Limited (“ARM”). No license, express or implied, by estoppel or otherwise to any intellectual property rights is granted by this document unless specifically stated.

Your access to the information in this document is conditional upon your acceptance that you will not use or permit others to use the information for the purposes of determining whether implementations infringe any patents.

This document is provided “AS IS”. ARM provides no representations and no warranties, express, implied or statutory, including, without limitation, the implied warranties of merchantability, satisfactory quality, non-infringement or fitness for a particular purpose with respect to the document. For the avoidance of doubt, ARM makes no representation with respect to, and has undertaken no analysis to identify or understand the scope and content of, third party patents, copyrights, trade secrets, or other rights.

This document may include technical inaccuracies or typographical errors.

This document may be translated into other languages for convenience, and you agree that if there is any conflict between the English version of this document and any translation, the terms of the English version shall prevail.

To the extent not prohibited by law, in no event will ARM be liable for any damages, including without limitation any direct, indirect, special, incidental, punitive, or consequential damages, however caused and regardless of the theory of liability, arising out of any use of this document, even if ARM has been advised of the possibility of such damages.

This document consists solely of commercial items. You shall be responsible for ensuring that any use, duplication or disclosure of this document complies fully with any relevant export laws and regulations to assure that this document or any portion thereof is not exported, directly or indirectly, in violation of such export laws. Use of the word “partner” in reference to ARM’s customers is not intended to create or refer to any partnership relationship with any other company. ARM may make changes to this document at any time and without notice.

If any of the provisions contained in these terms conflict with any of the provisions of any signed written agreement specifically covering this document with ARM, then the signed written agreement prevails over and supersedes the conflicting provisions of these terms.

Words and logos marked with ® or ™ are registered trademarks or trademarks of ARM Limited or its affiliates in the EU and/or elsewhere. All rights reserved. Other brands and names mentioned in this document may be the trademarks of their respective owners. You must follow the ARM trademark usage guidelines http://www.arm.com/about/trademarks/guidelines/index.php.

Copyright © ARM Limited or its affiliates. All rights reserved.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

In this document, where the term ARM is used to refer to the company it means “ARM or any of its subsidiaries as appropriate”.

SoC Designer
Traffic Generator User Guide
Copyright © 2017 ARM Limited. All rights reserved.

Release Information
The following changes have been made to this document.

<table>
<thead>
<tr>
<th>Date</th>
<th>Issue</th>
<th>Confidentiality</th>
<th>Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>February 2017</td>
<td>A</td>
<td>Non-Confidential</td>
<td>Restamp/Release with 9.1.0</td>
</tr>
</tbody>
</table>
Confidentiality Status
This document is Non-Confidential. The right to use, copy and disclose this document may be subject to license restrictions in accordance with the terms of the agreement entered into by ARM and the party that ARM delivered this document to.

Product Status
The information in this document is final, that is for a developed product.

Web Address
http://www.arm.com
Chapter 1.
Overview of Traffic Generator

Features and Benefits .................................................................2

Chapter 2.
Adding and Configuring the Traffic Generator Components

Adding and Configuring the SoC Designer Plus Components .....................4
  SoC Designer Plus Component Library ..........................................4
  Adding the Cycle Model to the Component Library ................................5
  Adding the Component to the SoC Designer Canvas ..............................5

Available Ports ..............................................................................5

Setting Component Parameters ......................................................6
  General Setup Parameters ..........................................................6
    Setting the Congestion Mode ......................................................7
  Producer Profile Parameters ......................................................8
    Configuring Randomization ......................................................12
  AXI-Specific Parameters ..........................................................13

Debug Features ...............................................................................15
  Log Files ....................................................................................15
  Registers ....................................................................................16
    Producer General Registers ......................................................17
    Producer Read Registers ..........................................................17
    Producer Write Registers ..........................................................18
    Producer Pending Registers ......................................................19
    Producer Completed Read and Completed Write Registers ..............20
  Throughput Registers ..................................................................21

Memory Information ........................................................................22

Chapter 3.
Creating Traffic Profiles

Traffic Profile Development ..........................................................24
  Development Guidelines .............................................................24
  Phases of Traffic Patterns ..........................................................25
  Linking Traffic Profiles to Traffic Generator Models ............................26

Traffic Pattern Commands ............................................................26
  Control Commands .......................................................................27
  control (address_add) ...............................................................27
control (address_preset) ........................................... 29
control (iterations) ............................................. 30
control (reset_signal) .......................................... 31
control (restart) .................................................. 32
control (send_event) ........................................... 33
control (send_fifo) ................................................ 34
control (set_signal) ............................................. 35
control (traffic_replay_file) ................................. 36
control (wait) ..................................................... 38
control (wait_event) ............................................ 39
control (wait_fifo) ................................................ 40
control (wait_signal) ........................................... 41
Parameter Commands ........................................... 42
parameter (name) .................................................. 42
Example Traffic Profile Files ................................. 43
vrec_controller.xml .............................................. 43

Chapter 4.
Simulating With Traffic Replay Files

Creating a Traffic Replay File ................................. 45
Metadata Format .................................................. 45
Traffic Replay File Example ................................. 47
Implementing the Traffic Replay File ........................ 48
Chapter 1

Overview of Traffic Generator

The ARM Traffic Generator Producer and Consumer Cycle Models allow you to define specific traffic patterns and emulate them in the context of your SoC Designer system design. During simulation, SoC Designer generates performance metrics, which you can access using the System Analyzer.

Figure 1-1 illustrates the operation of the system at a high level.

Figure 1-1  Traffic Analyzer Overview
The following process explains the operation of the system as shown in Figure 1-1:

1. In an XML-based Traffic Profile, you define the traffic patterns to emulate. You can create multiple Traffic Profiles; each Traffic Generator model references one Traffic Profile per simulation. Refer to Chapter 3 for information about creating Traffic Profiles.

2. You create a system design on the SoC Designer Canvas that incorporates Traffic Generator models.

3. For each instance of a Traffic Generator model, you specify the location of the Traffic Profile to use. This is done via model parameters (see “General Setup Parameters” on page 2-6).

4. During simulation, the Traffic Profile interacts with the Traffic Generator models to generate the desired traffic. SoC Designer Plus generates and outputs CAPI and AXI4 transaction metrics.


1.1 Features and Benefits

- Traffic Generator model in the following versions: 32-bit, 64-bit, 128-bit, 256-bit, 512-bit, 1024-bit
- Simulates high-performance use cases
- Multi-threaded traffic generation
- Mobile Graphics use case included
- Traffic Replay supported
- Batch simulations supported
- .XML file-driven Traffic Pattern definition
- Traffic Pattern Signaling
- CAPI stream export to database
- AXI4 transaction stream export to database
Chapter 2

Adding and Configuring the Traffic Generator Components

This chapter describes the functionality of the Model component. It contains the following sections:

- Adding and Configuring the SoC Designer Plus Components
- Available Ports
- Setting Component Parameters
- Debug Features
2.1 Adding and Configuring the SoC Designer Plus Components

The SoC Designer User Guide (ARM DUI0956) describes how to use the component. See that guide for more information.

- SoC Designer Plus Component Library
- Adding the Cycle Model to the Component Library
- Adding the Component to the SoC Designer Canvas

2.1.1 SoC Designer Plus Component Library

The component files are the final output from the Cycle Model Studio compile and are the input to SoC Designer Plus. There are two versions of the component; an optimized release version for normal operation, and a debug version.

On Linux, the debug version of the component is compiled without optimizations and includes debug symbols for use with gdb. The release version is compiled without debug information and is optimized for performance.

On Windows, the debug version of the component is compiled referencing the debug runtime libraries so it can be linked with the debug version of SoC Designer Plus. The release version is compiled referencing the release runtime library. Both release and debug versions generate debug symbols for use with the Visual C++ debugger on Windows.

The provided component files are listed in Table 2-1 below:

<table>
<thead>
<tr>
<th>Platform</th>
<th>File Description</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux</td>
<td>maxlib.lib&lt;model_name&gt;.conf</td>
<td>SoC Designer Plus configuration file</td>
</tr>
<tr>
<td></td>
<td>lib&lt;component_name&gt;.mx.so</td>
<td>SoC Designer Plus component runtime file</td>
</tr>
<tr>
<td></td>
<td>lib&lt;component_name&gt;.mx_DBG.so</td>
<td>SoC Designer Plus component debug file</td>
</tr>
<tr>
<td>Windows</td>
<td>maxlib.lib&lt;model_name&gt;.windows.conf</td>
<td>SoC Designer Plus configuration file</td>
</tr>
<tr>
<td></td>
<td>lib&lt;component_name&gt;.mx.dll</td>
<td>SoC Designer Plus component runtime file</td>
</tr>
<tr>
<td></td>
<td>lib&lt;component_name&gt;.mx_DBG.dll</td>
<td>SoC Designer Plus component debug file</td>
</tr>
</tbody>
</table>

Additionally, this User Guide PDF file are provided with the component.
2.1.2 Adding the Cycle Model to the Component Library

The compiled Cycle Model component is provided as a configuration file (.conf). To make the component available in the Component Window in SoC Designer Canvas, use SoC Designer Canvas.

For more information on SoC Designer Canvas, see the SoC Designer User Guide (ARM DUI0956).

2.1.3 Adding the Component to the SoC Designer Canvas

Locate the component named “AXI4*_TG_{bit_version}” in the Component Window and drag it out to the Canvas. It will look similar to the component shown in Figure 2-1.

Figure 2-1 Traffic Generator Component in SoC Designer Plus

2.2 Available Ports

Table 2-2 describes the ESL ports that are exposed in SoC Designer Plus.

Table 2-2 ESL Component Ports

<table>
<thead>
<tr>
<th>ESL Port</th>
<th>Description</th>
<th>Direction</th>
</tr>
</thead>
<tbody>
<tr>
<td>axi_mst_port</td>
<td>AXI master port.</td>
<td>Transaction Master</td>
</tr>
<tr>
<td>apb</td>
<td>Provides the ability to set/reset input signals to the Traffic Generator. A 32-bit word can be written with each bit representing a signal. The 32-bit word is written to the &quot;apb_base&quot; address (offset 0) parameter.</td>
<td>Transaction Slave</td>
</tr>
<tr>
<td>sig_m</td>
<td>Carries output signals in a 32-bit word.</td>
<td>Signal Master</td>
</tr>
<tr>
<td>sig_s</td>
<td>Provides the ability to set/reset input signals to the Traffic Generator. A 32-bit word can be written with each bit representing a signal. The &quot;value&quot; written to the signal port carries the 32-bit word.</td>
<td>Signal Slave</td>
</tr>
<tr>
<td>clk-in</td>
<td>Input clock.</td>
<td>Clock Slave</td>
</tr>
</tbody>
</table>
2.3 Setting Component Parameters

Using the SoC Designer Plus GUI, set the parameters described in this section to control Traffic Generator behavior.

In MxScript, set the parameter values using the `setParameter` function.

*Note:* Most parameters support runtime reconfiguration. Parameter changes during simulation take immediate effect. If you do not set parameter, its default value is applied.

This section includes specifications for:

- General Setup Parameters
- Producer Profile Parameters
- AXI-Specific Parameters

### 2.3.1 General Setup Parameters

Table 2-3 describes the General Setup parameters.

**Table 2-3 Component Parameters: General Setup**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Allowed Values</th>
<th>Default Value</th>
<th>Init/ Runtime</th>
</tr>
</thead>
<tbody>
<tr>
<td>address_width</td>
<td>Width of address signals on the AXIv2 ports.</td>
<td>32-63</td>
<td>32</td>
<td>N/A</td>
</tr>
<tr>
<td>apb_base</td>
<td>Byte-aligned address that sets the base address for the APB transaction port interface.</td>
<td>0-fffffffffffffff</td>
<td>0x1000000000</td>
<td>N/A</td>
</tr>
<tr>
<td>congestion_mode</td>
<td>Sets the desired behavior for transactions when the transmit buffer is full. See Setting the Congestion Mode for more information.</td>
<td>Drop, Suspend</td>
<td>Drop</td>
<td>N/A</td>
</tr>
<tr>
<td>protocol_variant</td>
<td>Selects the protocol for the transaction generator. Supported protocols are AXI4 and ACE-Lite.</td>
<td>AXI4, ACE-Lite</td>
<td>AXI4</td>
<td>N/A</td>
</tr>
<tr>
<td>clock_period</td>
<td>Clock period of the TG in picoseconds. The clock_period setting must match the clock connected to the clock input port of the Traffic Generator.</td>
<td>0 - UINT_Max</td>
<td>10,000</td>
<td>N/A</td>
</tr>
</tbody>
</table>
2.3.1.1 Setting the Congestion Mode

The Traffic Generator Congestion Mode algorithm simulates transaction creation when the transmit buffer is full due to bus loading or competition for shared resources. Using the parameter `congestion_mode`, you can configure the desired behavior under these conditions:

- **Drop mode** — Drops traffic created when the transmit buffer is full. This is the default setting.
- **Suspend mode** — Suspends traffic created when the transmit buffer is full; activity resumes when the blocked condition resolves. No transactions are dropped.

*Note:* Suspension time during Traffic Replay and Traffic Generation has no effect on the traffic pattern Control parameters. For example, a control wait timer specified for the Stop phase is not altered with respect to suspension times.

*Note:* The `congestion_mode` parameter is a Traffic Generator component-level parameter and may not be set via XML Traffic Parameter commands.

### Table 2-3 Component Parameters: General Setup (continued)

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Allowed Values</th>
<th>Default Value</th>
<th>Init/Runtime</th>
</tr>
</thead>
<tbody>
<tr>
<td>random_seed</td>
<td>Seed for the pseudo random number generator that is used for transaction generation. Use the same seed to produce exactly the same results in different simulation runs.</td>
<td>0 - UINT_Max</td>
<td>42</td>
<td>N/A</td>
</tr>
<tr>
<td>traffic_pattern_file</td>
<td>Identifies the Traffic Pattern input file. Environment variables may be included in the path.</td>
<td>Relative path and filename (String)</td>
<td>—</td>
<td>N/A</td>
</tr>
<tr>
<td>traffic_pattern_input</td>
<td>Enables traffic pattern inputs from traffic pattern file. Parameters that otherwise control traffic generation now serve as defaults for the traffic pattern inputs.</td>
<td>True, False</td>
<td>False</td>
<td>N/A</td>
</tr>
</tbody>
</table>
### 2.3.2 Producer Profile Parameters

Table 2-4 describes the Producer Profile parameters. All parameter values may be set using the component parameters dialog. Certain values are configurable only via the component parameter setting, not through XML Traffic Pattern commands. These cases are noted in the table.

Certain parameters support randomization by specifying a range; refer to the section Configuring Randomization.

*Note:* Reconfiguration (update to a parameter) is not supported when `traffic_pattern_input` is enabled.

#### Table 2-4 Component Parameters: Producer Profile

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Allowed Values</th>
<th>Default Value</th>
<th>Init/Runtime</th>
</tr>
</thead>
<tbody>
<tr>
<td>prod_read_tr_pause</td>
<td>Transaction issuing rate in cycles. This value determines the number of cycles from the creation of a transaction until the creation of the next transaction. If the previous transaction takes longer than this value, the next transaction overlaps the previous one. Transactions are pushed into an internal FIFO where they reside until they can be issued on the AXI interface. The FIFO dwelling time is monitored.</td>
<td>1 - UINT_Max Accepts a comma-separated range or a single value.</td>
<td>20</td>
<td>N/A</td>
</tr>
<tr>
<td>prod_write_tr_pause</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>prod_read_tr_size</td>
<td>Byte size of the generated transactions. Set to 0 to stop traffic generation. Set to another value to resume.</td>
<td>0 - 16 * Data-Size Accepts a comma-separated range or a single value.</td>
<td>0</td>
<td>N/A</td>
</tr>
<tr>
<td>prod_write_tr_size</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>prod_read_tr_bytes_per_beat(^1,2)</td>
<td>Maximum number of bytes of data transferred on each beat (data transfer).</td>
<td>1,2,4,8,16(^3) Accepts a comma-separated range or a single value.</td>
<td>Bus Byte Width</td>
<td>N/A</td>
</tr>
<tr>
<td>prod_write_tr_bytes_per_beat(^2)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Table 2-4  Component Parameters: Producer Profile (continued)

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Allowed Values</th>
<th>Default Value</th>
<th>Init/Runtime</th>
</tr>
</thead>
<tbody>
<tr>
<td>prod_read_tr_size_auto_align</td>
<td>If true, only full-beat transactions are generated. That is, the transaction size given in prod_read_tr_size or prod_write_tr_size is adjusted to align with AXI data channel width. Set this parameter to false to generate transactions that only use part of the available data lanes. Note: If false, the bytes per beat parameter will be ignored and the full byte bus width will be used.</td>
<td>true, false</td>
<td>true</td>
<td>N/A</td>
</tr>
<tr>
<td>prod_write_tr_size_auto_align</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>prod_read_seq_length</td>
<td>Number of transactions in a sequence. Set to 0 to disable sequences.</td>
<td>0 - UINT_Max</td>
<td>0</td>
<td>N/A</td>
</tr>
<tr>
<td>prod_write_seq_length</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>prod_read_seq_pause</td>
<td>Idle cycles between sequences. This value determines the number of cycles from the creation of a sequence until the creation of the next sequence. Note that unlike transactions, sequences cannot overlap. If a sequence takes longer than the sequence pause, the next sequence is started directly thereafter.</td>
<td>1 - UINT_Max</td>
<td>50</td>
<td>N/A</td>
</tr>
<tr>
<td>prod_write_seq_pause</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>prod_read_seq_rnd_addresses</td>
<td>Use random addresses in sequences. To achieve address randomization, set address to a random number range.</td>
<td>true, false</td>
<td>false</td>
<td>N/A</td>
</tr>
<tr>
<td>prod_write_seq_rnd_addresses</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>prod_read_rnd_addresses</td>
<td>Use random addresses outside sequences. To achieve sequential addressing outside of sequences, set to false.</td>
<td>true, false</td>
<td>true</td>
<td>N/A</td>
</tr>
<tr>
<td>prod_write_rnd_addresses</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>prod_read_address</td>
<td>AXI bus address for generated transactions. In sequences with incrementing addresses, this parameter determines the sequence start address.</td>
<td>0 - UINT_Max</td>
<td>0</td>
<td>N/A</td>
</tr>
<tr>
<td>prod_write_address</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>prod_read_address_auto_align</td>
<td>By default, addresses are automatically altered to align with the AXI data channel widths. Disable this feature to generate unaligned addresses.</td>
<td>true, false</td>
<td>true</td>
<td>N/A</td>
</tr>
<tr>
<td>prod_write_address_auto_align</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
<td>Allowed Values</td>
<td>Default Value</td>
<td>Init/ Runtime</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>----------------</td>
<td>---------------</td>
<td>---------------</td>
</tr>
<tr>
<td>prod_read_id_range&lt;sup&gt;2&lt;/sup&gt;</td>
<td>ID range for generated transactions. IDs are assigned in increasing order, starting with the minimum ID and wrapping around when the maximum ID is reached.</td>
<td>0 - UINT_Max</td>
<td>(0,15)</td>
<td>N/A</td>
</tr>
<tr>
<td>prod_write_id_range&lt;sup&gt;2&lt;/sup&gt;</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>prod_read_issuing_cap&lt;sup&gt;2&lt;/sup&gt;</td>
<td>Maximum number of open read/write transactions on the AXI interface. These values throttle how many overlapping transactions may be started on the AXI interface. However, the value has no influence on the transaction generation rate (see prod_[read</td>
<td>write]_tr_pause parameters): transactions are held back in a FIFO when the number of open transactions on the AXI interface has reached the issuing capability limit.</td>
<td>1 - UINT_Max</td>
<td>100</td>
</tr>
<tr>
<td>prod_write_issuing_cap&lt;sup&gt;2&lt;/sup&gt;</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Note:** These parameters may not be set via XML Traffic Parameter commands.

<table>
<thead>
<tr>
<th>prod_read_data_accept_ws</th>
<th>Wait cycles in the Read Data phase (RVALID to RREADY latency). The wait cycles are distributed uniformly among the data beats.</th>
<th>1 - UINT_Max</th>
<th>0</th>
<th>N/A</th>
</tr>
</thead>
<tbody>
<tr>
<td>prod_write_data_issuing_ws</td>
<td>Wait cycles in the Write Data phase (latency until WVALID is issued). The wait cycles are distributed uniformly among the data beats.</td>
<td>1 - UINT_Max</td>
<td>0</td>
<td>N/A</td>
</tr>
<tr>
<td>--------------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>prod_write_resp_accept_ws</td>
<td>Wait cycles until the acceptance of incoming write response (BVALID to BREADY latency).</td>
<td>1 - UINT_Max</td>
<td>0</td>
<td>N/A</td>
</tr>
<tr>
<td>--------------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Note:** This parameter may not be set via XML Traffic Parameter commands.
### Table 2-4  Component Parameters: Producer Profile (continued)

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Allowed Values</th>
<th>Default Value</th>
<th>Init/Runtime</th>
</tr>
</thead>
<tbody>
<tr>
<td>prod_limit_outstanding_transactions²</td>
<td>Maximum number of outstanding transactions. Outstanding transactions include transaction open on the AXI interface and generated transactions waiting to be placed on the AXI interface. If the limit is reached on the Read or Write channel, additional transactions are dropped.</td>
<td>1 - UINT_Max</td>
<td>1000</td>
<td>N/A</td>
</tr>
</tbody>
</table>

*Note: This parameter may not be set via XML Traffic Parameter commands.*

| prod_read_nop_cycles² | prod_write_nop_cycles² | Insert NOP cycles. No read or write traffic is generated until the NOP cycles have passed. NOP cycles are applied at the start of the simulation, and you can insert additional NOP cycles anytime during simulation by setting the parameter value. These parameters offer a convenient way to temporarily suspend traffic generation without having to change the traffic profile. For example, this feature can be used in combination with throughput/latency monitoring to program the behavior of a dynamic traffic/QoS regulator. | 1 - UINT_Max     | 0            | N/A          |

| prod_read_total_bytes² | prod_write_total_bytes² | Sets a limit on the number of bytes to read/write. Transaction generation will stop when this limit is reached. A zero (0) value indicates no limit. | 0 - UINT_Max | 0            | N/A          |

1. If either the producer read/write_tr_size_auto_align parameter is false, or the producer read/write_address_auto_align parameter is false, the producer read/write_tr_bytes_per_beat parameter value is ignored. For these unaligned transactions, the maximum number of bytes per beat for the generated transaction equals the AXI bus byte width.

2. Randomization not supported (ranges are not accepted).

3. The producer read/write_tr_bytes_per_beat parameter values cannot exceed the AXI bus byte width. For example, for a CAXITG_32 component, you may not set this parameter to greater than 4 bytes (32 bits).
2.3.2.1 Configuring Randomization

Certain Producer Profile parameters (Table 2-4) support randomization through the specification of a range; values may be entered in decimal or hexadecimal format. For example, the following XML parameter commands specify randomization using ranges:

```
<parameter name="prod_read_tr_pause">(0x20, 0x40)</parameter>
<parameter name="prod_read_tr_pause">(32, 64)</parameter>
```

These commands specify that the wait time between the creation of one transaction and the creation of the subsequent transaction will be a random number of cycles between 0x20 and 0x40 (hexadecimal), or between 32 and 64 (decimal).

Note that creating a range does not trigger randomization for all parameters. For the following parameters, ranges may specify sequences:

- prod_read_address
- prod_write_address
- prod_read_id_range
- prod_write_id_range
### 2.3.3 AXI-Specific Parameters

Table 2-5 describes the Producer Profile parameters that are specific to AXI.

Unless otherwise noted, all parameters support randomization, as well as reconfiguration at any time during simulation. Changes take immediate effect.

*Note: Reconfiguration is not supported when traffic_pattern_input is enabled*

*Note: AXI-specific Producer Profile parameters do not support randomization.*

**Table 2-5 Component Parameters: AXI-Specific**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Allowed Values</th>
<th>Default Value</th>
<th>Init/Runtime</th>
</tr>
</thead>
<tbody>
<tr>
<td>axi4_protocol</td>
<td>Sets the protocol variant.</td>
<td>AXI4, ACE-Lite</td>
<td>AXI4</td>
<td>N/A</td>
</tr>
<tr>
<td>axi4_user_width</td>
<td>Establishes the width (in bits) of the user signal (aruser, awuser, ruser, buser)</td>
<td>0 - 64</td>
<td>0</td>
<td>N/A</td>
</tr>
<tr>
<td>prod_axi_arburst_value</td>
<td>ARBURST signal value. See table 4-3 in AMBA AXI Specification v1.0.</td>
<td>0 - 2</td>
<td>1</td>
<td>N/A</td>
</tr>
<tr>
<td>prod_axi_arcache_value</td>
<td>ARCACHE signal value. See table 5-1 in AMBA AXI Specification v1.0</td>
<td>0 - 15</td>
<td>3</td>
<td>N/A</td>
</tr>
<tr>
<td>prod_axi_arprot_value</td>
<td>ARPROT signal value. See table 5-2 in AMBA AXI Specification v1.0</td>
<td>0 - 7</td>
<td>0</td>
<td>N/A</td>
</tr>
<tr>
<td>prod_axi_aruser_value</td>
<td>ARUSER signal value. The AMBA AXI specification does NOT define the functionality of the ARUSER signal</td>
<td>0 - 511</td>
<td>0</td>
<td>N/A</td>
</tr>
<tr>
<td>prod_axi_awburst_value</td>
<td>AWBURST signal value. See table 4-3 in AMBA AXI Specification v1.0.</td>
<td>0 - 2</td>
<td>1</td>
<td>N/A</td>
</tr>
<tr>
<td>prod_axi_awcache_value</td>
<td>AWCACHE signal value. See table 5-1 in AMBA AXI Specification v1.0</td>
<td>0 - 15</td>
<td>3</td>
<td>N/A</td>
</tr>
<tr>
<td>prod_axi_awprot_value</td>
<td>AWPROT signal value. See table 5-2 in AMBA AXI Specification v1.0</td>
<td>0 - 7</td>
<td>0</td>
<td>N/A</td>
</tr>
<tr>
<td>prod_axi_awuser_value</td>
<td>AWUSER signal value. The AMBA AXI specification does NOT define the functionality of the ARUSER signal</td>
<td>0 - 511</td>
<td>0</td>
<td>N/A</td>
</tr>
<tr>
<td>prod_axi_ruser_value</td>
<td>RUSER signal value. The AMBA AXI specification does NOT define the functionality of the ARUSER signal</td>
<td>0 - 511</td>
<td>0</td>
<td>N/A</td>
</tr>
<tr>
<td>prod_axi_wuser_value</td>
<td>WUSER signal value. The AMBA AXI specification does NOT define the functionality of the ARUSER signal</td>
<td>0 - 511</td>
<td>0</td>
<td>N/A</td>
</tr>
<tr>
<td>prod_axi_arqos</td>
<td>ARQOS signal value See AMBA AXI and ACE Protocol Specification Issue D.</td>
<td>0-15</td>
<td>0</td>
<td>N/A</td>
</tr>
<tr>
<td>prod_axi_awqos</td>
<td>AWQOS signal value See AMBA AXI and ACE Protocol Specification Issue D.</td>
<td>0-15</td>
<td>0</td>
<td>N/A</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
<td>Allowed Values</td>
<td>Default Value</td>
<td>Init/Runtime</td>
</tr>
<tr>
<td>------------------</td>
<td>------------------------------------------------------------------------------</td>
<td>----------------</td>
<td>---------------</td>
<td>--------------</td>
</tr>
<tr>
<td>prod_axi_arregion</td>
<td>ARREGION signal value See AMBA AXI and ACE Protocol Specification Issue D.</td>
<td>0-15</td>
<td>0</td>
<td>N/A</td>
</tr>
<tr>
<td>prod_axi_awregion</td>
<td>AWREGION signal value See AMBA AXI and ACE Protocol Specification Issue D.</td>
<td>0-15</td>
<td>0</td>
<td>N/A</td>
</tr>
<tr>
<td>prod_axi_arsnoop</td>
<td>ARSNOOP signal value See AMBA AXI and ACE Protocol Specification Issue D.</td>
<td>0-15</td>
<td>0</td>
<td>N/A</td>
</tr>
<tr>
<td>prod_axi_awsnnoo</td>
<td>AWSNOOP signal value See AMBA AXI and ACE Protocol Specification Issue D.</td>
<td>0-7</td>
<td>0</td>
<td>N/A</td>
</tr>
<tr>
<td>prod_axi_ardomain</td>
<td>ARDOMAIN signal value See AMBA AXI and ACE Protocol Specification Issue D.</td>
<td>0-3</td>
<td>3</td>
<td>N/A</td>
</tr>
<tr>
<td>prod_axi_awdomain</td>
<td>AWDOMAIN signal value See AMBA AXI and ACE Protocol Specification Issue D.</td>
<td>0-3</td>
<td>3</td>
<td>N/A</td>
</tr>
<tr>
<td>prod_axi_arbar</td>
<td>ARBAR signal value See AMBA AXI and ACE Protocol Specification Issue D.</td>
<td>0-1</td>
<td>0</td>
<td>N/A</td>
</tr>
<tr>
<td>prod_axi_awbar</td>
<td>ARBAR signal value See AMBA AXI and ACE Protocol Specification Issue D.</td>
<td>0-1</td>
<td>0</td>
<td>N/A</td>
</tr>
</tbody>
</table>

1. Setting parameter value does not impact Traffic Generator functionality beyond placing the specified value on the AW or AW channel signal. For example, Traffic Generator always uses increment burst functionality.
2.4 Debug Features

This section describes:

- Log Files
- Registers
- Memory Information

2.4.1 Log Files

During simulation, Traffic Generator generates a log file for each Traffic Pattern. The log file lists:

- the time (in nanoseconds) at which each command is executed
- the name of the command executed
- the phase at which the command is run
- the action taken

Note: In the log file, "eTP_PHASE_RUN" represents the "run" or "active" period between the Start phase and the Stop phase.

The following is sample log file output:

```
1000004 ns, name: camera_read_last_frame.ping, phase: eTP_PHASE_START, action: ACTIVATE
1000004 ns, name: camera_read_last_frame.ping, phase: eTP_PHASE_RUN, action: WAIT_EVENT
1000006 ns, name: camera_read_ping_buffer, phase: eTP_PHASE_START, action: ACTIVATE
1000006 ns, name: camera_read_ping_buffer, phase: eTP_PHASE_RUN, action: WAIT_TIME
1010006 ns, name: camera_read_ping_buffer, phase: eTP_PHASE_RUN, write_bytes: 0
1010006 ns, name: camera_read_ping_buffer, phase: eTP_PHASE_RUN, read_bytes: 1440
1010006 ns, name: camera_read_ping_buffer, phase: eTP_PHASE_RUN, action: SEND_EVENT
1010006 ns, name: camera_read_ping_buffer, phase: eTP_PHASE_RUN, action: DEACTIVATE
1010006 ns, name: camera_read_ping_buffer, phase: eTP_PHASE_RUN, action: ITERATION COMPLETE
1010006 ns, name: camera_read_ping_buffer, phase: eTP_PHASE_START, action: ACTIVATE
```

The log file is:

- Output to the directory that houses the Traffic Pattern file.
- Given the same name as the Traffic Pattern file, with the extension .log instead of .xml.

You specify the Traffic Pattern name and location using the component parameter `traffic_pattern_file` (see “General Setup Parameters” on page 2-6).
2.4.2 Registers

The Traffic Generator component has a debug interface (CADI) that allows you to view, manipulate, and control its registers and memories in the SoC Designer Simulator. To access the register and memory views in SoC Designer Plus, right-click on the component and choose the appropriate menu entry.

You can view transactions using the transaction monitors attached to connections. By right-clicking on any of the AXI connections in SoC Designer Plus, you can attach a transaction monitor probe.

The Traffic Generator registers provide information about the current status of transaction generation. The registers are organized as follows:

- **Producer General** — General producer information. See Section 2.4.2.1.
- **Producer Read** and **Producer Write** — Statistics about completed read and write transactions in Producer mode. See Section 2.4.2.2 and Section 2.4.2.3.
- **Producer Pending** — Statistics about transactions that are still in progress. See Section 2.4.2.4.
- **Producer Completed Read** and **Producer Completed Write** — Statistics about the most recent completed read / write transaction in Producer mode, including detailed latency measurements. See Section 2.4.2.5.
- **Throughput** — Throughput statistics for both AXI master interface (this is disabled if `enable_throughput_monitoring` is set to False). See Section 2.4.2.6.

To observe the registers in the SoC Designer Simulator GUI:

1. Open the register window for the component.
2. Select the register group tab with the appropriate name.

To query register values in MxScript, use the function CADIRegRead, and prefix the register name with the appropriate group name, followed by a dot. For example:

```plaintext
message("INFO", "M0: Write id=%d complete. FIFO time: %d cycles. AXI duration: %d cycles.",
CADIRegRead("M0", "P completed read.ID"), CADIRegRead("M0", "P completed read.Req start cycle")
- CADIRegRead("M0", "P completed read.Start cycle"),
CADIRegRead("M0", "P completed read.Finish cycle"
- CADIRegRead("M0", "P completed read.Req start cycle"));
```
2.4.2.1 Producer General Registers

Table 2-6 describe the Producer General Transaction registers

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Update Occurs When</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random seed</td>
<td>Random seed of the Traffic Generator.</td>
<td>Random seed parameter has been changed</td>
</tr>
<tr>
<td>Read bytes issued</td>
<td>Total number of bytes read so far.</td>
<td>Transaction is completed</td>
</tr>
<tr>
<td>Write bytes issued</td>
<td>Total number of bytes written so far.</td>
<td>Transaction is completed</td>
</tr>
</tbody>
</table>

2.4.2.2 Producer Read Registers

Table 2-7 describe the Producer Read Transaction registers. Updates occur when the transaction is completed.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bytes</td>
<td>Total number of bytes read so far.</td>
</tr>
<tr>
<td>Transactions</td>
<td>Total number of read transactions completed so far.</td>
</tr>
<tr>
<td>Reqdur last</td>
<td>Duration of ARVALID/ARREADY handshake in cycles for last completed read transaction.</td>
</tr>
<tr>
<td>Reqdur avg</td>
<td>Average duration of the ARVALID/ARREADY handshake in cycles.</td>
</tr>
<tr>
<td>Reqdur min</td>
<td>Minimum duration of the ARVALID/ARREADY handshake in cycles.</td>
</tr>
<tr>
<td>Reqdur max</td>
<td>Maximum duration of the ARVALID/ARREADY handshake in cycles.</td>
</tr>
<tr>
<td>Busdur last</td>
<td>Duration of last completed read transaction; measured in cycles from issuance of ARVALID unit reception of RLAST.</td>
</tr>
<tr>
<td>Busdur avg</td>
<td>Average total duration of the completed transactions; this is measured in cycles from issuance of ARVALID until reception of RLAST.</td>
</tr>
<tr>
<td>Busdur min</td>
<td>Minimum total duration of the completed transactions; this is measured in cycles from issuance of ARVALID until reception of RLAST.</td>
</tr>
<tr>
<td>Busdur max</td>
<td>Maximum total duration of the completed transactions; this is measured in cycles from issuance of ARVALID until reception of RLAST.</td>
</tr>
<tr>
<td>Fifodur last</td>
<td>Dwelling time (in cycles) of the last completed read transaction in the internal FIFO until ARVALID could be issued. This parameter is a measurement of the back-pressure on the AXI interface.</td>
</tr>
<tr>
<td>Fifodur avg</td>
<td>Average dwelling time (in cycles) of the completed read transactions in the internal FIFO until ARVALID could be issued. This parameter is a measurement of the back-pressure on the AXI interface.</td>
</tr>
</tbody>
</table>
Table 2-8 describe the Producer Write Transaction registers. Updates occur when the transaction is completed.

Table 2-8  Producer Write Registers

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bytes</td>
<td>Total number of bytes written so far.</td>
</tr>
<tr>
<td>Transactions</td>
<td>Total number of write transactions completed so far.</td>
</tr>
<tr>
<td>Reqdur last</td>
<td>Duration of the AWVALID/AWREADY handshake in cycles for last completed write transaction.</td>
</tr>
<tr>
<td>Reqdur avg</td>
<td>Average duration of the AWVALID/AWREADY handshake in cycles.</td>
</tr>
<tr>
<td>Reqdur min</td>
<td>Minimum duration of the AWVALID/AWREADY handshake in cycles.</td>
</tr>
<tr>
<td>Reqdur max</td>
<td>Maximum duration of the AWVALID/AWREADY handshake in cycles.</td>
</tr>
<tr>
<td>Busdur last</td>
<td>Duration of the last completed write transaction; this is measured in cycles from issuance of AWVALID until completion of BVALID/BREADY handshake.</td>
</tr>
<tr>
<td>Busdur avg</td>
<td>Average total duration of the completed transactions; this is measured in cycles from issuance of AWVALID until completion of BVALID/BREADY handshake.</td>
</tr>
<tr>
<td>Busdur min</td>
<td>Minimum total duration of the completed transactions; measured in cycles from issuance of AWVALID until completion of BVALID/BREADY handshake.</td>
</tr>
<tr>
<td>Busdur max</td>
<td>Maximum total duration of the completed transactions; measured in cycles from issuance of AWVALID until completion of BVALID/BREADY handshake.</td>
</tr>
<tr>
<td>Fifodur last</td>
<td>Dwelling time (in cycles) of the last completed write transaction in the internal FIFO until AWVALID could be issued. This parameter is a measurement of the back-pressure on the AXI interface.</td>
</tr>
</tbody>
</table>
2.4.2.4 Producer Pending Registers

Table 2-9 describe the Producer Pending Transaction registers.

*Note:* The pending transaction and pending bytes counters include both transactions in progress on the AXI, and transactions that have been generated but are waiting in the Traffic Generator queues.

### Table 2-9 Producer Pending Registers

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Update Occurs When</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fifodur avg</td>
<td>Average dwelling time (in cycles) of the completed write transactions in the internal FIFO until AWVALID could be issued. This parameter is a measurement of the back-pressure on the AXI interface.</td>
<td>Read transaction is created</td>
</tr>
<tr>
<td>Fifodur max</td>
<td>Maximum dwelling time (in cycles) of the completed write transactions in the internal FIFO until AWVALID could be issued. This parameter is a measurement of the back-pressure on the AXI interface.</td>
<td>Write transaction is created</td>
</tr>
<tr>
<td>Outstanding avg</td>
<td>Average number of concurrently-active write transactions so far.</td>
<td>Transaction is created or completed</td>
</tr>
<tr>
<td>Outstanding max</td>
<td>Maximum number of concurrently-active write transactions so far.</td>
<td>Transaction is created or completed</td>
</tr>
</tbody>
</table>

### Table 2-8 Producer Write Registers (continued)

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Update Occurs When</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read ID</td>
<td>ID of the last created read transaction.</td>
<td>Read transaction is created</td>
</tr>
<tr>
<td>Write ID</td>
<td>ID of the last created write transaction.</td>
<td>Write transaction is created</td>
</tr>
<tr>
<td>Read transactions</td>
<td>Number of generated but not completed read transactions.</td>
<td>Transaction is created or completed</td>
</tr>
<tr>
<td>Write transactions</td>
<td>Number of generated but not completed write transactions.</td>
<td>Transaction is created or completed</td>
</tr>
<tr>
<td>Read bytes</td>
<td>Number of bytes still to read with pending transactions.</td>
<td>Read transaction is created or completed</td>
</tr>
<tr>
<td>Write bytes</td>
<td>Number of bytes still to write with pending transactions.</td>
<td>Write transaction is created or completed</td>
</tr>
<tr>
<td>Dropped read transactions</td>
<td>Number of dropped read transactions (see parameter prod_limit_outstanding_transactions).</td>
<td>Read transaction has been dropped</td>
</tr>
<tr>
<td>Dropped write transactions</td>
<td>Number of dropped write transactions (see parameter prod_limit_outstanding_transactions).</td>
<td>Write transaction has been dropped</td>
</tr>
</tbody>
</table>
2.4.2.5 Producer Completed Read and Completed Write Registers

Table 2-10 describe the Producer Completed Read and Completed Write Transaction registers.

**Table 2-10 Producer Completed Read and Completed Write Registers**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>ID of the last completed transaction.</td>
</tr>
<tr>
<td>Bytes</td>
<td>Size of the last completed transaction.</td>
</tr>
<tr>
<td>Address</td>
<td>Address of the last completed transaction.</td>
</tr>
<tr>
<td>Read/Write buffer address</td>
<td>Read/Write buffer address of the last completed transaction.</td>
</tr>
<tr>
<td>Start cycle</td>
<td>Cycle in which the transaction was generated.</td>
</tr>
<tr>
<td>Req start cycle</td>
<td>Request start on bus time point (AXI: set AxVALID).</td>
</tr>
<tr>
<td>Req grant cycle</td>
<td>Request granted on bus time point (AXI: get AxREADY).</td>
</tr>
<tr>
<td>Data n start cycle n is</td>
<td>Data beat n start on bus time point (AXI: set WVALID / get RVALID). Shows 0</td>
</tr>
<tr>
<td>element of {1,2,…,16}</td>
<td>for unused data beats.</td>
</tr>
<tr>
<td>Data n grant cycle n is</td>
<td>Data beat n grant on bus time point (AXI: get WREADY / set RREADY). Shows 0</td>
</tr>
<tr>
<td>element of {1,2,…,16}</td>
<td>for unused data beats.</td>
</tr>
<tr>
<td>Resp start cycle</td>
<td>Response start on bus time point (AXI: get BVALID).</td>
</tr>
<tr>
<td>Resp grant cycle</td>
<td>Response granted on bus time point (AXI: set BREADY).</td>
</tr>
<tr>
<td>Finish cycle</td>
<td>Cycle in which the transaction was completed.</td>
</tr>
</tbody>
</table>

Updates occur when the Read/Write transaction is completed.
2.4.2.6 Throughput Registers

Table 2-11 describe the Throughput registers. Note the following regarding the content of this table:

- Current throughput — The average of the throughput over the last 10 intervals.
- Average throughput — The average of the throughput over the last 100 intervals.
- The number of cycles in an interval is specified using the `calc_interval_throughput` parameter.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Update Occurs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base Address</td>
<td>Base address for producer and consumer CAXITG_BUFFER memory.</td>
<td>During component initial-ization</td>
</tr>
<tr>
<td>Producer 1 interval write</td>
<td>Producer write throughput (Kbytes/Sec) during the last interval.</td>
<td>After throughput measurement</td>
</tr>
<tr>
<td>Producer current write</td>
<td>Producer current write throughput (KBytes/Sec).</td>
<td></td>
</tr>
<tr>
<td>Producer average write</td>
<td>Producer average write throughput (KBytes/Sec).</td>
<td></td>
</tr>
<tr>
<td>Producer 1 interval read</td>
<td>Producer read throughput (KBytes/Sec) during the last interval.</td>
<td></td>
</tr>
<tr>
<td>Producer current read</td>
<td>Producer current read throughput (KBytes/Sec).</td>
<td></td>
</tr>
<tr>
<td>Producer average read</td>
<td>Producer average write throughput (KBytes/Sec).</td>
<td></td>
</tr>
</tbody>
</table>
2.4.3 Memory Information

The Traffic Generator contains the CAXITG_BUFFER memory, which is used as follows:

- In Producer Write mode, data sent with Write transactions is taken from the CAXITG_BUFFER.
- In Producer Read mode, data received with Read transactions is written to the CAXITG_BUFFER.

To access the contents of CAXITG_BUFFER memory, use the CADI direct memory interface.

The address of the buffer and the size of the buffer are set by parameters (base_address, buffer_size). If the Write/Read address falls in this range, the data is fetched directly according to the address. If the Write/Read address falls outside this range, then the offset into the buffer is determined by: AXI address % buffer_size. If the Write/Read starts within but extends beyond the range of the buffer, the access wraps to the start of the buffer.

To inspect and modify the contents of the CAXITG_BUFFER memory, use the Memory view in SoC Designer Plus. In MxScript, use the functions:

- CADIMemRead and CADIMemWrite to enable reading and writing the memory.
- CADIMemLoadFile and CADIMemStoreFile to load and store the memory contents using a file on the hard drive.

Memory Buffer size
The default size of the CAXITG_BUFFER is 4 GBytes (0x100000000). To change the buffer size, use the buffer_size parameter. buffer_size values must be non-zero increments of 4096. The CAXITG_BUFFER uses a sparse memory implementation.
Chapter 3

Creating Traffic Profiles

Traffic Generator accepts a Traffic Profile input file that emulates traffic patterns, which are pre-defined groups of parameter settings with triggering and timing definitions. You can associate traffic patterns with any Traffic Generator in your system design, link traffic patterns to each other in the system, and run them on a threaded basis (concurrently).

This section summarizes the approach to creating Traffic Profile files, and provides a reference to the XML-based API:

- Traffic Profile Development
- Traffic Pattern Commands
- Example Traffic Profile Files
3.1 Traffic Profile Development

This section includes:

- Development Guidelines
- Phases of Traffic Patterns
- Linking Traffic Profiles to Traffic Generator Models

3.1.1 Development Guidelines

To get started developing Traffic Profiles, review the guidelines in this section:

- Develop one XML-based Traffic Profile file for each Traffic Generator model.
- Review the structure indicated in the section “Phases of Traffic Patterns” on page 3-25, below. Each traffic pattern in a Traffic Profile file should define the desired actions for each phase of execution.
- Using the Control commands described in the section “Linking Traffic Profiles to Traffic Generator Models” on page 3-26, define patterns with specific timed, scheduled events, and the desired number of iterations. A traffic pattern may link to other patterns via events and FIFO triggers.

Note: XML is a markup specification language rather than a programming language; it does not implement a line-by-line execution sequence. Traffic pattern execution is handled by SoC Designer Plus during simulation, based on your Control and Parameter definitions. Therefore the commands can be written in any order.

- Use the examples found in the Traffic Generator installation directory as a reference, as well as the examples in “Example Traffic Profile Files” on page 3-43.
3.1.2 Phases of Traffic Patterns

Figure 3-1 illustrates the structure of a Traffic Pattern, which consists of five phases. Each Traffic Pattern should define the desired actions for each phase of execution.

![Diagram of Traffic Pattern Phases](image)

As shown in Figure 3-1, Traffic Patterns proceed through the following phases:

1. **Initial** — Traffic Patterns always start at the Initial phase when a simulation is begun or restarted. When the Initial phase ends, the Start phase begins.

2. **Start** — The actual work of the Traffic Pattern occurs between the Start and Stop phases. The Start/Stop loop executes based on the number of iterations you specify using the `control(iterations)` command.

3. **Stop** — This phase marks the end of the Traffic Pattern execution phase. The Final phase begins when the Start/Stop loop has executed *either*:
   - The number of times specified by the `control(iterations)` command, or
   - The number of times required to reach the traffic threshold specified by the `prod_read_total_bytes` or `prod_write_total_bytes` parameter.

   If both `control(iterations)` and a byte threshold are specified, the Start/Stop loop exits when one of these conditions is met.

4. **Final** — The Traffic Pattern enters the Final phase after all iterations of the Start/Stop loop have completed.

5. **Restart** — This is an optional phase directing the Traffic Pattern to return to the Initial phase.
3.1.3 Linking Traffic Profiles to Traffic Generator Models

When your Traffic Profile files are complete, you are ready to link them to the corresponding Traffic Generator model. Use SoC Designer Canvas to set the following parameters on a per-Traffic Generator Model basis (refer to Table 2-3 on page 6 for more information about setting these parameters):

- traffic_pattern_input — Enables traffic pattern inputs on the model.
- traffic_pattern_file — Specifies the name of the traffic profile file associated with the model.

SoC Designer Plus loads the associated traffic profiles when it reads in the system (.MXP) file.

3.2 Traffic Pattern Commands

This section discusses:

- Control Commands
- Parameter Commands

Generally speaking, Control commands dictate the behavior of the traffic pattern prior to and after actual traffic generation, which occurs between the Start and Stop phases. Parameter commands dictate the nature of the traffic generated during traffic generation.

Note: XML is a markup specification language rather than a programming language; it does not implement a line-by-line execution sequence. Traffic Pattern execution is handled by SoC Designer Plus during simulation, based on your Control and Parameter definitions. Therefore the commands can be written in any order.
3.2.1 Control Commands

Control commands include:

- control (address_add)
- control (address_preset)
- control (iterations)
- control (reset_signal)
- control (restart)
- control (send_event)
- control (send_fifo)
- control (set_signal)
- control (traffic_replay_file)
- control (wait)
- control (wait_event)
- control (wait_fifo)
- control (wait_signal)

3.2.1.1 control (address_add)

Description

Adds a value to an address at the completion of the phase; or, in the case of transactions, after the specified transaction. Operation is as follows:

- For the Stop phase, the specified value is added to the address value that existed at the associated Start phase,
- For the Final phase, the specified value is added to the address value that existed at the associated Initial phase,
- For Transaction events, the specified value is added to the base address of the previous transaction.

Note: For Transaction events, address_add has no effect for traffic replay files.

Syntax

<control type = “address_add” phase = <“phase” | “event”> access = <“access”>>input> </control>

where:

<phase | event> is a valid phase or event from the Attributes table below

<access> is a valid value from the Attributes table below. This is an optional attribute.
<input> is the value to add, entered as a numeral. Address_add values can be negative or positive.

Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
<th>Mandatory/Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>access</td>
<td>{read, write}</td>
<td>O</td>
</tr>
<tr>
<td>phase</td>
<td>{stop, final, transaction}</td>
<td>M</td>
</tr>
<tr>
<td>type</td>
<td>address_add</td>
<td>M</td>
</tr>
</tbody>
</table>

Example

The example below does the following: At the end of the Stop phase, it adds the value 32 to the address that existed at the beginning of the Start/Stop loop:

```xml
<control type="address_add" event="stop">32</control>
```
3.2.1.2 control (address_preset)

Description
Presets the read/write address to a value other than the minimum value for address ranges.

Note: When used for traffic replay files, address_preset sets a base address for the traffic file transactions. Subsequently, the address for the transaction is an offset to that base address.

Syntax
<control type = “address_preset” access = <“access value”>> <input> </control>

where:

<access> is a valid access value from the Attributes table below. This is an optional attribute. If not specified, both Read and Write are set to the specified address.

<input> is the address to preset in decimal or hexadecimal format.

Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
<th>Mandatory/Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>access</td>
<td>{read, write}</td>
<td>O</td>
</tr>
<tr>
<td>type</td>
<td>address_preset</td>
<td>M</td>
</tr>
</tbody>
</table>

Example
The example below does the following: Presets the Read and Write address to 0x2394FE0.

<control type="address_preset">0x2394FE0</control>
3.2.1.3 control (iterations)

Description
Specifies the number of iterations to run the Start/Stop loop. If unspecified, Traffic Generator runs a single iteration.

Note: Setting iterations to zero (0) results in infinite iterations.

Syntax
<control type = "iterations"> <input> </input> </control>

where:
<input> is the number of iterations to run, entered as a numeral

Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
<th>Mandatory/Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>iterations</td>
<td>M</td>
</tr>
</tbody>
</table>

Example
The example below sets the number of Start/Stop loop iterations to 36:

<control type="iterations">36</control>
3.2.1.4 control (reset_signal)

Description
Resets the specified signal or signals prior to initiating the specified phase. When multiple signals are specified, all signals are reset.

Syntax
<control type = “reset_signal” phase = “<phase>”><signal(s)> </control>

where:

<signal(s)> is the signal or signals to reset.

<phase> is the phase to initiate after resetting the signal.

Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
<th>Mandatory/Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>phase</td>
<td>{initial, start, stop, final}</td>
<td>M</td>
</tr>
<tr>
<td>type</td>
<td>reset_signal</td>
<td>M</td>
</tr>
</tbody>
</table>

Example
The example below resets bit 3 and bit 1 to inactive ("0"), then initiates the Final phase.

<control type="reset_signal" phase="final">0xA</control>
3.2.1.5 control (restart)

**Description**
Specifies whether to restart the Traffic Pattern upon completion of the Final phase. By default, restart is set to False.

**Syntax**

```
<control type = “restart”><input> </control>
```

where:

- `<input>` is either `true` or `false`.

**Attributes**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
<th>Mandatory/Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>restart</td>
<td>M</td>
</tr>
</tbody>
</table>

**Example**
The example below sets Restart to True:

```
<control type="restart">true</control>
```
3.2.1.6 control (send_event)

**Description**
Sends an event to the specified traffic pattern upon initiation of the specified phase.

**Syntax**
<control type = “send_event” name = “<traffic pattern>,<event name>” phase = “<phase>”/>

where:

<traffic pattern> is the traffic pattern to which to send the event.

<event name> is the name of the event to send.

<phase> is the phase whose occurrence initiates the sending of the event.

**Attributes**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
<th>Mandatory/Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>phase</td>
<td>{initial, start, stop, final}</td>
<td>M</td>
</tr>
<tr>
<td>name</td>
<td>&lt;traffic pattern&gt;,&lt;event name&gt;</td>
<td>M</td>
</tr>
<tr>
<td>type</td>
<td>send_event</td>
<td>M</td>
</tr>
</tbody>
</table>

**Example**
The example below does the following: At the beginning of the Final phase, sends an event called vrotate_done to the traffic pattern uc_control.

<control type="send_event" phase="final" name="uc_control.vrotate_done"/>
3.2.1.7 control (send_fifo)

**Description**
Executes the specified number of writes to the specified fifo upon initiation of the specified phase.

**Syntax**
```xml
<control type="send_fifo" phase="<phase>" depth="<fifo depth>" name="<traffic_pattern>.<fifo name>" input=""/></control>
```

where:

- `<traffic pattern>` is the traffic pattern to which to send the event.
- `<fifo name>` is the fifo to which to write.
- `<phase>` is the phase whose occurrence initiates the send_fifo.
- `<fifo depth>` is the number of queued Writes that the fifo accepts. When the fifo is full, and a Write is executed, the Write is blocked until Read is executed and the fifo can accept additional data.

**Note:** The sender and receiver depth attributes must match. If they don't, the depth to use cannot be determined.

- `<input>` is the number of writes to execute.

**Attributes**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
<th>Mandatory/Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>depth</td>
<td><code>&lt;fifo depth&gt;</code></td>
<td>O</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If not specified, defaults to 16.</td>
</tr>
<tr>
<td>phase</td>
<td>{initial, start, stop, final}</td>
<td>M</td>
</tr>
<tr>
<td>name</td>
<td><code>&lt;traffic pattern&gt;.&lt;fifo name&gt;</code></td>
<td>M</td>
</tr>
<tr>
<td>type</td>
<td>send_fifo</td>
<td>M</td>
</tr>
</tbody>
</table>

**Example**
The example below does the following: At the beginning of the Stop phase, it executes one fifo write to the block_store fifo in the vrotate_store traffic pattern, which has a depth of 2.

```xml
<control type="send_fifo" phase="stop" depth="2" name="vrotate_store.block_store">1</control>
```
3.2.1.8 control (set_signal)

**Description**
Sets the specified signal or signals prior to initiating the specified phase. When multiple signals are specified, all signals are set.

**Syntax**
<control type = “set_signal” phase = “<phase>”><signal(s)> </control>

where:

<signal(s)> is the signal or signals to set.

<phase> is the phase to initiate after setting the signal.

**Attributes**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
<th>Mandatory/Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>phase</td>
<td>{initial, start, stop, final}</td>
<td>M</td>
</tr>
<tr>
<td>type</td>
<td>set_signal</td>
<td>M</td>
</tr>
</tbody>
</table>

**Example**
The example below sets bit 2 and bit 0 to active ("1"), then initiates the Initial phase.

```
<control type="set_signal" phase="initial">0x5</control>
```
3.2.1.9 control (traffic_replay_file)

Description

If the traffic_replay_file command is present in the input file, then the Traffic Generator reads from the Traffic Replay file instead of generating traffic according to the Producer Profile parameter settings (see “Producer Profile Parameters” on page 2-8).

Traffic generation occurs between the Start and Stop phases. All other Control commands remain in effect (events, fifos, waits, phases, total bytes settings, etc.).

Note: Using the traffic_replay_file command automatically sets the prod_read_rnd_addresses and prod_write_rnd_address parameters to false.

Syntax

<control type = “traffic_replay_file” loop = ”<loop>” cycle_adder = ”<cycle_adder>” cycle_multiplier = ”<cycle_multiplier>” <filename> </control>

where:

<loop> indicates whether looping is:

- **Enabled (true)** — If the end of the traffic replay file is reached before other Stop conditions are met, the reader loops to the top of the file to get more transactions.

- **Disabled (false)** — Traffic Generator ceases producing transactions until the Stop conditions are met, or completes the iteration if no Stop conditions have been defined.

By default, looping is set to false.

<cycle_adder> adjusts the transaction cycle numbers in the input file in order to shift the transactions forward or backward. The cycle_adder value is applied to all transactions in the file.

<cycle_multiplier> adjusts the transaction cycle number in the input file to slow down or speed up the pacing of the transactions. The multiplier is applied to all transactions in the file.

Note: There is no need to adjust for differences in cycle period in the file and cycle period of the simulation. This is done automatically. For instance, if the cycle period of the input file is 10ns and the "clock_period" of the Traffic Generator is 5 ns, the cycle number in the input file is automatically adjusted by a factor of two to synchronize the traffic file with the system.

<filename> is the name of the traffic file from which to read. By default, the system expects the traffic replay file to be relative to the location of the traffic pattern input file. You may also specify an explicit path.
### Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
<th>Mandatory/Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>traffic_input_file</td>
<td>M</td>
</tr>
<tr>
<td>loop</td>
<td>{true, false}</td>
<td>O</td>
</tr>
<tr>
<td>cycle_adder</td>
<td>{integer}</td>
<td>O</td>
</tr>
<tr>
<td>cycle_multiplier</td>
<td>{float}</td>
<td>O</td>
</tr>
</tbody>
</table>

### Example

The example below enables looping of the traffic replay file called `transTrace.CAXITG_64_7i.log`. It shifts the transactions backward by two, and speeds up the pacing of the transactions by 1.5.

```
<control type="traffic_replay_file" loop="true" cycle_adder="-2"
cycle_multiplier ="1.5">transTrace.CAXITG_64_7i.log</control>
```
3.2.1.10 control (wait)

Description
Waits for a specified amount of time before initiating the specified phase. If a wait_time is specified along with a wait_event, then the phase is initiated when one of these conditions is first met.

Syntax
<control type = “wait” unit = “<unit>” phase = “<phase>” ><input></control>

where:

<unit> is the unit of measure for the time being input.

<phase> is the phase to initiate.

<input> is the amount of time to wait.

Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Values</th>
<th>Mandatory/Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>phase</td>
<td>{initial, start, stop}</td>
<td>M</td>
</tr>
<tr>
<td>type</td>
<td>wait</td>
<td>M</td>
</tr>
<tr>
<td>unit</td>
<td>{ps, ns, us, ms, sec, cycles}</td>
<td>M</td>
</tr>
</tbody>
</table>

Example
The example below waits 16 ms before initiating the Stop phase.

<control type="wait" unit="ms" phase="stop">16</control>
3.2.1.11 control (wait_event)

Description
Waits for a specified event prior to initiating the specified phase. When multiple events are specified, all events must be received. Specify multiple events using separate lines.

Syntax
<control type = “wait_event” name = "<name>" phase = "<phase"/>

where:

   <name> is the event or events whose occurrence initiates the phase.

   <phase> is the phase to initiate.

Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
<th>Mandatory/Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>phase</td>
<td>{initial, start, stop}</td>
<td>M</td>
</tr>
<tr>
<td>name</td>
<td>&lt;name&gt;</td>
<td>M</td>
</tr>
<tr>
<td>type</td>
<td>wait_event</td>
<td>M</td>
</tr>
</tbody>
</table>

Example
The following example waits for an interrupt event before initiating the Initial phase:

<control type="wait_event" name="interrupt" phase="initial"/>
3.2.1.12 control (wait_fifo)

Description

Waits for the specified number of reads from the specified fifo prior to initiating the specified phase. Fifo waits always occur after Time and Event waits when specified for the same phase.

When you specify multiple fifo waits for the same phase, they are executed sequentially. All fifo waits must be satisfied to progress to the specified phase.

Syntax

<control type = "wait_fifo" name = "<fifo name>" phase = "<phase>" depth = "<fifo depth>"><input></control>

where:

<fifo name> is the name of the fifo sending Reads.

<phase> is the phase to initiate.

<fifo depth> is the depth of the fifo sending Reads.

Note: The sender and receiver depth attributes must match. If they don't, the depth to use cannot be determined.

<input> is the number of reads for which to wait.

Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
<th>Mandatory/Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>wait</td>
<td>M</td>
</tr>
<tr>
<td>phase</td>
<td>{initial, start, stop}</td>
<td>M</td>
</tr>
<tr>
<td>name</td>
<td>&lt;fifo name&gt;</td>
<td>M</td>
</tr>
<tr>
<td>depth</td>
<td>&lt;fifo depth&gt;</td>
<td>O</td>
</tr>
</tbody>
</table>

Examples

The example below waits for 16 fifo reads, at a depth of 50, from the row fifo before initiating the Start phase:

<control type="wait_fifo" name="row" phase="start" depth="50">16
</control>
3.2.1.13 control (wait_signal)

Description
Waits for a specified signal or signals prior to initiating the specified phase. When multiple signals are specified, all signals must be in active ("1") state simultaneously.

Syntax
<control type = “wait_signal” phase = “<phase>”><signal(s)> </control>

where:

<signal(s)> is the signal or signals to set.

<phase> is the phase to initiate after receiving the signal.

Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
<th>Mandatory/Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>phase</td>
<td>{initial, start, stop, final}</td>
<td>M</td>
</tr>
<tr>
<td>type</td>
<td>wait_signal</td>
<td>M</td>
</tr>
</tbody>
</table>

Example
The example below waits for bit 2 to be active ("1"), then initiates the Start phase.

<control type="wait_signal" phase="start">0x4</control>
3.2.2 Parameter Commands

This section describes the syntax for entering parameter commands. For a list of the parameters and attributes, refer to the sections “Producer Profile Parameters” on page 2-8 and “AXI-Specific Parameters” on page 2-13.

Note: Certain parameters may not be set using XML-based Traffic Parameter commands; rather, you must use the component parameter setting. These cases are noted in the parameter descriptions.

3.2.2.1 parameter (name)

Description
Sets the specified parameter to the specified parameter value.

Syntax

<parameter name="<parameter name>">><parameter value></parameter>

where:

<parameter name> is a valid Producer Profile or AXI-specific parameter.

<parameter value> is a value that is valid for that parameter.

Attributes
Refer to “Producer Profile Parameters” on page 2-8 and “AXI-Specific Parameters” on page 2-13.

Examples

<parameter name="prod_read_tr_size">32</parameter>
<parameter name="prod_read_id_range">0</parameter>
<parameter name="prod_read_address">(0x2000000, 0x21FFFFF)</parameter> <parameter name="prod_read_total_bytes">512</parameter>
<parameter name="prod_read_rnd_addresses">0</parameter>
3.3 Example Traffic Profile Files

The following example is the vrec_controller.xml Traffic Profile file. This is part of the set of traffic pattern files that ship with the video record use case included in the Cortex-A15 Traffic Generator and Baremetal CPAK, available on ARM System Exchange (armipexchange.com).

3.3.1 vrec_controller.xml

The following example is the vrec_controller.xml Traffic Profile. This Traffic Profile file includes the following traffic patterns: uc_control and uc_control_int. Refer to Figure 3-2 (following the example) for a graphical view of its execution.

```xml
<?xml version="1.0"?>
<traffic_patterns>
  <traffic_pattern name="uc_control">
  <!-- wait 1ms for start of use case simulation -->
  <control unit="ms" phase="initial" type="wait">1</control>
  <!-- enable interrupt controller pattern -->
  <control name="uc_control_int.start" phase="initial" type="send_event"/>
  <!-- wait for frame interrupt -->
  <control name="frame" phase="start" type="wait_event"/>
  <!-- kick off frame processing -->
  <!-- -->
  <control name="camera_control.interrupt" phase="start" type="send_event"/>
  <control name="vencoder_control.interrupt" phase="start" type="send_event"/>
  <control name="vdecoder_control.interrupt" phase="start" type="send_event"/>
  <control name="vrotate_control.interrupt" phase="start" type="send_event"/>
  <control name="display_dma_control.interrupt" phase="start" type="send_event"/>
  <!-- wait for frame completion -->
  <control name="camera_done" phase="stop" type="wait_event"/>
  <control name="vencoder_done" phase="stop" type="wait_event"/>
  <control name="vdecoder_done" phase="stop" type="wait_event"/>
  <control name="vrotate_done" phase="stop" type="wait_event"/>
  <control name="display_dma_done" phase="stop" type="wait_event"/>
  <!-- signal one frame to be stored to memory -->
  <control name="memory_dma_control.frame" phase="stop" type="send_fifo">1</control>
  <!-- write frames to memory -->
  <control name="memory_dma_control.start" phase="final" type="send_event"/>
  <!-- run for 3 frames -->
  <!-- no time duration - wait for events -->
  <control type="iterations">3</control>
  <control type="restart">false</control>
  </traffic_pattern>

<traffic_pattern name="uc_control_int" type="control">
  <!-- write for start of sim -->
  <control name="start" phase="initial" type="wait_event"/>
  <!-- generate frame interrupts at 60fps ... forever -->
  <control name="uc_control.frame" phase="start" type="send_event"/>
  <control unit="ms" phase="stop" type="wait">16</control>
</traffic_pattern>
</traffic_patterns>
```
Figure 3-2 shows a graphical view of the execution flow of `vrec_controller.xml`.

Figure 3-2 uc_control flow
A Traffic Replay file describes the traffic patterns to be simulated. The traffic data is rendered in a space-delimited text file that conforms to the metadata format described in this chapter.

You can configure Traffic Replay files to loop during simulation, shift the transaction cycle numbers, and slow down or speed up the pace of transaction execution. Refer to “control (traffic_replay_file)” on page 3-36.

4.1 Creating a Traffic Replay File

This section describes:

- Metadata Format
- Traffic Replay File Example

4.1.1 Metadata Format

The file format of the Traffic Replay file is space-delimited. It must include a first line with meta data input, followed by subsequent lines, each of which describes a single transaction:

**Line 1 (Traffic Replay Metadata) takes the following form:**

```
```

where:

Version Info is the Traffic Generator file format version; for example, 1.0.
Protocol is the name of the AXI4 protocol. Valid options are AXI4 and ACE-Lite.

Cycle Period Value is the time of each cycle; for example, 6666.

Cycle Period Units is the unit in which the Cycle Period Value is measured. Valid options are: ps, ns, us, and ms.

Lines 2 through Last (Transaction Information):

There are four types of transactions, and the required properties included in the metadata text file are different for each type. Table 4-1 describes the required properties for each transaction type.

<table>
<thead>
<tr>
<th>Table 4-1 Transaction Metadata</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transaction Type</td>
</tr>
<tr>
<td>Generic</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>AXI4</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>ACE-Lite</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
4.1.2 Traffic Replay File Example

The following example shows part of a Traffic Replay file:

- The first line indicates that the Traffic Generator file format is version 0.1, the protocol is ACE-Lite, and the cycle period is 6666 ps.
- The second line begins the transaction section, specifying a write to address 10,000,000 at cycle 102.

```
0.1 ACE-Lite 6666 ps
102 w 10000000 4 8 1 1 0 3 0 1 0 0 0 0 0 0
122 w 10000020 4 8 1 1 0 3 0 1 0 0 0 0 0 0
142 w 10000040 4 8 1 1 0 3 0 1 0 0 0 0 0 0
162 w 10000060 4 8 1 1 0 3 0 1 0 0 0 0 0 0
182 w 10000080 4 8 1 1 0 3 0 1 0 0 0 0 0 0
183 w 10000000 4 8 1 1 0 3 0 1 0 0 0 0 0 0
203 w 10000020 4 8 1 1 0 3 0 1 0 0 0 0 0 0
223 w 10000040 4 8 1 1 0 3 0 1 0 0 0 0 0 0
243 w 10000060 4 8 1 1 0 3 0 1 0 0 0 0 0 0
252 r 0 4 8 0 1 0 3 0 1 0 0 0 0 0
252 w 0 4 8 0 1 0 3 0 1 0 0 0 0 0 0
257 r 20 4 8 0 1 0 3 0 1 0 0 0 0 0 0
253 w 20000000 4 8 2 1 0 3 0 1 0 0 0 0 0 0
```
• The single digits represent aspects of ACE, AXI4, or AXI behavior (refer to the AMBA® AXI™ and ACE™ Protocol Specification for details). The following figure uses the Cycle 102 line to show the characteristic to which each digit maps:

![Traffic Replay Transaction Mappings Diagram]

**Figure 4-1** Traffic Replay Transaction Mappings

### 4.2 Implementing the Traffic Replay File

The Traffic Generator component executes the Traffic Replay file when you include the `traffic_replay_file` command in your Traffic Profile input file (see “control (traffic_replay_file)” on page 3-36 and “Linking Traffic Profiles to Traffic Generator Models” on page 3-26). The Traffic Generator then reads from the Traffic Replay file instead of generating traffic according to the Producer Profile parameter settings.