AXI to AXI Synchronous Bridge Cycle Model

Version 9.0.0

User Guide

Non-Confidential

ARM®

ARM DUI 1059A (ID110916)

AXI to AXI Synchronous Bridge Cycle Model
User Guide

Release Information

The following changes have been made to this document.

Change History

Date	Issue	Confidentiality	Change
November 2016	A	Non-Confidential	First release

Non-Confidential Proprietary Notice

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

Contents

Preface

A Cycle Model component is a library that is generated through Cycle Model Studio™. The Cycle Model then can be used within a virtual platform tool, for example, SoC Designer.

About This Guide

This guide provides all the information needed to configure and use this Cycle Model in SoC Designer.

Audience

This guide is intended for experienced hardware and software developers who create components for use with SoC Designer. You should be familiar with the following products and technology:

- SoC Designer
- Hardware design verification
- Verilog or SystemVerilog programming language

Conventions

This guide uses the following conventions:

Convention	Description	Example
`courier`	Commands, functions, variables, routines, and code examples that are set apart from ordinary text.	`sparseMem_t SparseMemCreate-New();`
italic	New or unusual words or phrases appearing for the first time.	*Transactors* provide the entry and exit points for data ...
bold	Action that the user performs.	Click **Close** to close the dialog.
<text>	Values that you fill in, or that the system automatically supplies.	<platform>/ represents the name of various platforms.
[text]	Square brackets [ ] indicate optional text.	`$CARBON_HOME/bin/modelstudio [ <filename> ]`
[text1 | text2]	The vertical bar | indicates "OR," meaning that you can supply text1 or text 2.	`$CARBON_HOME/bin/modelstudio [<name>.symtab.db | <name>.ccfg ]`

Also note the following references:

- References to C code implicitly apply to C++ as well.
- File names ending in .cc, .cpp, or .cxx indicate a C++ source file.

Further reading

This section lists related publications.

The following publications provide information that relate directly to SoC Designer:

- *SoC Designer Installation Guide*
- *SoC Designer User Guide*
- *SoC Designer Standard Component Library Reference Manual*
- *SoC Designer AHBv2 Protocol Bundle User Guide*

The following publications provide reference information about ARM® products:

- *AMBA Design Kit Technical Reference Manual*
- *AMBA Specification*
- *AMBA AHB-Lite Protocol Specification*
- *AMBA AHB Protocol Specification*
- *Architecture Reference Manual*

See http://infocenter.arm.com/help/index.jsp for access to ARM documentation.

Glossary

Table 1:

AMBA	*Advanced Microcontroller Bus Architecture*. The ARM open standard on-chip bus specification that describes a strategy for the interconnection and management of functional blocks that make up a System-on-Chip (SoC).
AHB	*Advanced High-performance Bus*. A bus protocol with a fixed pipeline between address/control and data phases. It only supports a subset of the functionality provided by the AMBA AXI protocol.
APB	*Advanced Peripheral Bus*. A simpler bus protocol than AXI and AHB. It is designed for use with ancillary or general-purpose peripherals such as timers, interrupt controllers, UARTs, and I/O ports.
AXI	*Advanced eXtensible Interface*. A bus protocol that is targeted at high performance, high clock frequency system designs and includes a number of features that make it very suitable for high speed sub-micron interconnect.
Cycle Model	A software object created by the Cycle Model Studio from an RTL design. The Cycle Model contains a cycle- and register-accurate model of the hardware design.
Cycle Model Studio	Graphical tool for generating, validating, and executing hardware-accurate software models. It creates a *Cycle Model*, and it also takes a Cycle Model as input and generates a component that can be used in SoC Designer, Platform Architect, or Accellera SystemC for simulation.
CASI	*Cycle Accurate Simulation Interface*, is based on the SystemC communication library and manages the interconnection of components and communication between components.
CADI	*Cycle Accurate Debug Interface*, enables reading and writing memory and register values and also provides the interface to external debuggers.
CAPI	*Cycle Accurate Profiling Interface*, enables collecting historical data from a component and displaying the results in various formats.
Component	Building blocks used to create simulated systems. Components are connected together with unidirectional transaction-level or signal-level connections.
ESL	*Electronic System Level*. A type of design and verification methodology that models the behavior of an entire system using a high-level language such as C or C++.
HDL	*Hardware Description Language*. A language for formal description of electronic circuits, for example, Verilog.
RTL	*Register Transfer Level*. A high-level hardware description language (HDL) for defining digital circuits.

Table 1:

SoC Designer	The full name is *SoC Designer*. A high-performance, cycle accurate simulation framework which is targeted at System-on-a-Chip hardware and software debug, as well as architectural exploration.
SystemC	SystemC is a single, unified design and verification language that enables verification at the system level, independent of any detailed hardware and software implementation, as well as enabling co-verification with RTL design.
Transactor	*Transaction adaptors*. You add transactors to your component to connect your component directly to transaction level interface ports for your particular platform.

Chapter 1

Using the Cycle Model in SoC Designer

This chapter describes the functionality of the Cycle Model, and how to use it in SoC Designer. It contains the following sections:

1.1 Synchronous AXI to AXI Bridge Cycle Model Overview

The Synchronous AXI to AXI Bridge provides a unidirectional link between two AXIv2 domains with Synchronous clocks. Its function is to enable a master to access a slave on another bus, with the transfer initiated from one side only.

The two AXI subsystems can be running on different clock domains. The Synchronous AXI to AXI Bridge interface can be used as a *Step-Down* bridge to connect AXI buses where the originating bus (Domain 1) is running at a clock frequency which is an integer multiple of that of the target bus (Domain 2). It can also be used as a *Step-Up* bridge where the target bus is running at

a clock frequency which is an integer multiple of that of the originating bus. This is illustrated in Figure 1-1.

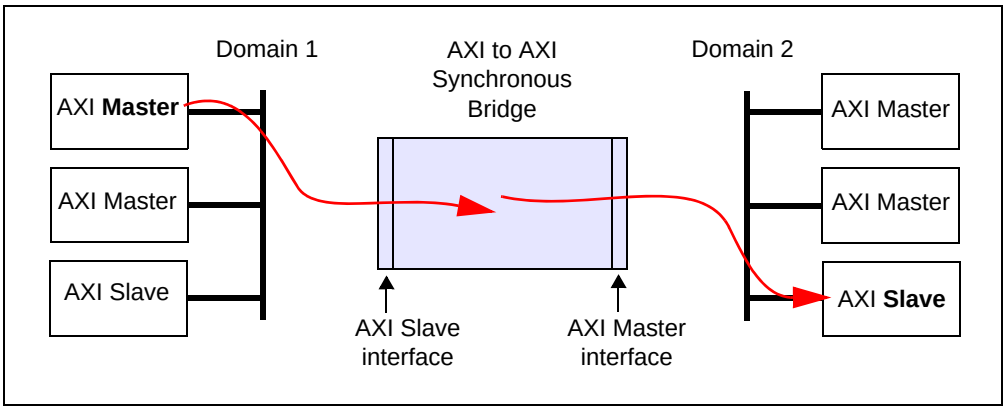

Figure 1-1 Connecting AXI Subsystems using a Synchronous AXI to AXI Bridge

The available Cycle Models are described in the next section.

1.1.1 Synchronous AXI to AXI Bridge

The AXI to AXI Synchronous Bridge is provided in three versions that support different data bus widths. All the components can be used as a Step-Down bridge, a one-to-one bridge, or a Step-Up bridge:

Table 1-1 Synchronous AXI to AXI Bridge Components

Component	Description
CM_Bridge_Axi_Axi_Sync_32	Converts transactions for a 32-bit wide data bus.
CM_Bridge_Axi_Axi_Sync_64	Converts transactions for a 64-bit wide data bus.
CM_Bridge_Axi_Axi_Sync_128	Converts transactions for a 128-bit wide data bus.

Figure 1-2 shows a simple configuration using the Synchronous AXI to AXI Bridge as a Step-Up component so a slow AXI clock domain can communicate with a faster domain.

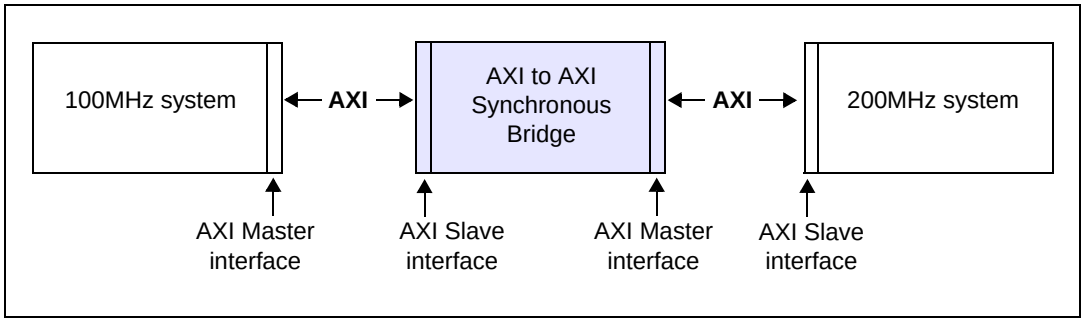

Figure 1-2 Simple AXI to AXI Step-Up Bridge Configuration

Figure 1-3 shows a simple configuration using the Synchronous AXI to AXI Bridge as a Step-Down component so a fast AXI clock domain can communicate with a slow domain.

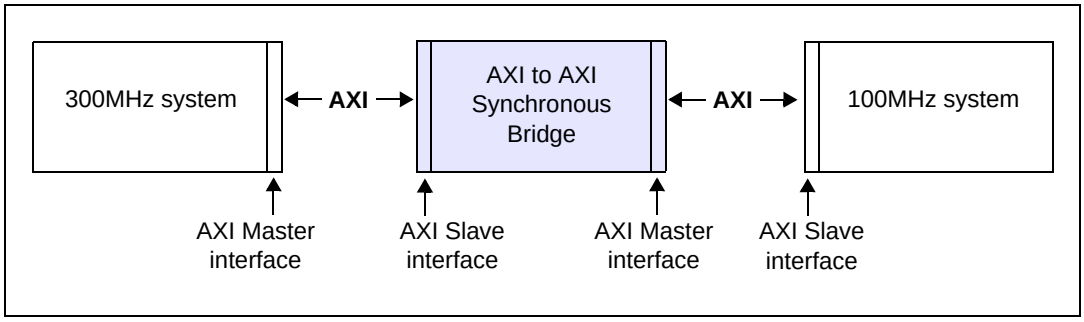

Figure 1-3 Simple AXI to AXI Step-Down Bridge Configuration

1.1.2 Cycle Model Features

The Synchronous AXI to AXI Bridges provide the following features:

- Frequency domain: The bridge can be used as a Step-Down bridge (N:1), a one-to-one bridge (1:1), or a Step-Up bridge (1:N). This is controlled by the *Integer clock ratio m:n* component parameter.
- Support for 32-, 64-, and 128-bit data bus widths.
- Bursts are preserved across the bridge.
- Buffered synchronization of the write address, write data, and read address channels for the Step-Up configuration.
- Buffered synchronization of the write response and read channels for the Step-Down configuration.

The different bridge types are described in Table 1-2.

Table 1-2 Synchronous Bridge Types

Type	Description
Step-Down	The Synchronous Step-Down bridge is used to connect AXI buses where the originating bus (Domain1) is running at a clock frequency greater than the target bus (Domain2). Domain1 is connected to the slave side of the bridge, and Domain2 is connected to the Master side. The clock ratio slave:master is specified in the form *m*:1 where *m* is an integer > 1.
Step-Up	The Synchronous Step-Up bridge is used to connect AXI buses where the target bus is running at a clock frequency greater than the originating bus. The clock ratio slave:master is specified in the form 1:*n* where *n* is an integer > 1.
One-to-One	The One-to-One bridge is used to connect AXI buses where the originating bus and the target bus are running at the same clock frequency. Note that this configuration of the component can be used to replace the *Fully registered* version of the RVML *AXI_RegSlice* v1 component.

1.1.3 Connecting with AXIv1 Components

When connecting with AXIv1 components you will need to use additional Cycle Models from the AXI Protocol Bundle. However, it is recommended that you upgrade to use all AXIv2 components as they provide enhanced functionality and they are easier to use.

1.1.3.1 Connecting from an AXIv1 Master Port

If you are making a connection from an AXIv1 master component, or replacing an ARM RVML Synchronous AXI To AXI Bridge Cycle Accurate model, you will need to use an AXIv1 to AXIv2 Bridge Cycle Model (AxiToAxi2).

Figure 1-4 shows a simple configuration using both an *AXIv1 to AXIv2 Bridge* and an *AXI to AXI Synchronous Bridge*.

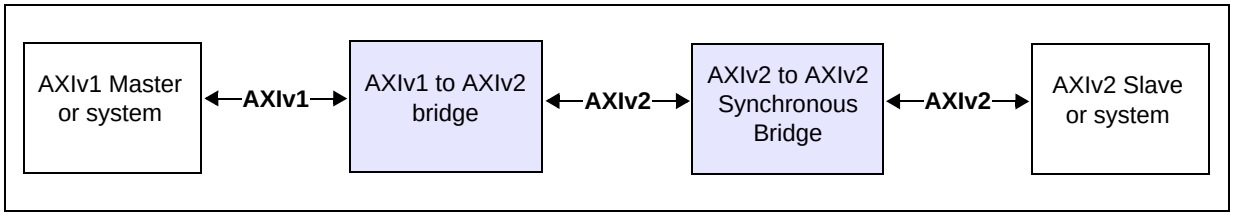

Figure 1-4 Connecting Synchronous AXIv1 and AXIv2 Components

1.1.3.2 Connecting to an AXIv1 Slave Port

If you are making a connection to an AXIv1 slave component, or replacing an ARM RVML Synchronous AXI To AXI Bridge Cycle Accurate model, you will need to use an AXIv2 to AXIv1 Bridge Cycle Model (Axi2ToAxi).

Figure 1-5 shows a simple configuration using both an *AXIv2 to AXIv1 Bridge* and an *AXI to AXI Synchronous Bridge*.

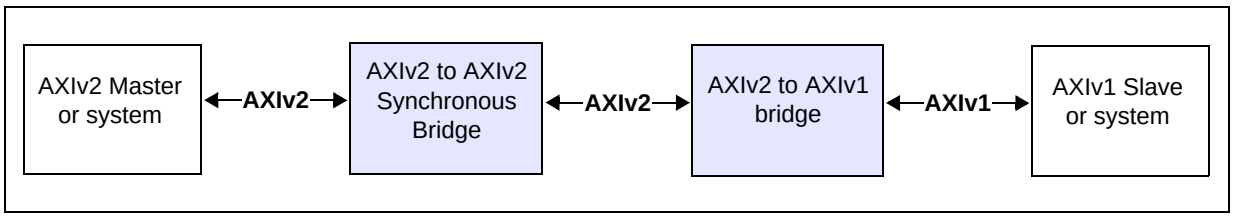

Figure 1-5 Connecting Synchronous AXIv2 and AXIv1 Components

1.2 Component Ports

Table 1-3 describes the ESL ports that are exposed in SoC Designer for the Synchronous AXI to AXI Bridge.

Table 1-3 ESL Component Ports

ESL Port	Description	Direction	Type
ARESETn	Reset port for resetting the Cycle Model.	Input	Signal slave
axi_s	AXIv2 transaction slave port.	Input	Transaction slave
clk-in	Input clock. The component is clocked at the frequency of the clock connected to the *clk-in* port. If the *clk-in* port is not connected, clocking is taken from SoC Designer System Properties. This must be connected to the faster of the two clocks. When the faster clock is the global clock, the clock port does not need to be connected.	Input	Clock slave
axi_m	AXIv2 transaction master port.	Output	Transaction master

1.3 Component Parameters

You can change the settings of all the component parameters in SoC Designer Canvas, and of some of the parameters in SoC Designer Simulator.

Table 1-4 describes the component parameters that are available for the Synchronous AXI to AXI Bridge. Parameters that may be modified at runtime are identified with Yes in the Runtime column, otherwise the parameter values are fixed and must be set before the start of simulation.

Table 1-4 Component Parameters

Name	Description	Allowed Values	Default Value	Runtime
Align Waveforms	When set to *true*, waveforms dumped from Cycle Model components are aligned with the SoC Designer simulation time. The reset sequence, however, is not included in the dumped data. When set to *false*, the reset sequence is dumped to waveform data, however, the component time is not aligned with the SoC Designer time.	true, false	true	No
axi_m Enable Debug Messages	When set to *true*, writes debug messages to the SoC Designer output window for the AXI master.	true, false	false	Yes
axi_s axi_size[0-5] [1]	These parameters are obsolete and should be left at their default values.[2]	0x0 - 0xFFFFFFFF	size 0 default is 0x100000000, size 1-5 default is 0x0	No
axi_s axi_start[0-5]		0x0 - 0xFFFFFFFF	0x0	No
axi_s Enable Debug Messages	When set to *true*, writes debug messages to the SoC Designer output window for the AXI slave.	true, false	false	Yes
Carbon DB Path	Sets the directory path to the database file.	Not Used	empty	No
Integer clock ratio m:n	Set this to the ratio of the two clocks. For a Step-Down bridge, for example, if the originating domain is 3 times faster than the target domain, enter 3:1. For a Step-Up bridge, for example, if the target domain is 2 times faster than the originating domain, enter 1:2. For a one-to-one bridge, leave this set to the default 1:1.	*m:n* one value *must be* 1	1:1	Yes
Dump Waveforms	Whether SoC Designer dumps waveforms for this component.	true, false	false	Yes
Enable Debug Messages	Enable or disable the capture of debug messages for the component.	true, false	false	Yes
Waveform File [3]	Name of the waveform file.	*string*	arm_CM_Bridge_Axi_Axi_Sync_<*data_width*>.vcd	No
Waveform Format	The format of the waveform dump file.	VCD, FSDB	VCD	No
Waveform Timescale	Sets the timescale to be used in the waveform.	Many values in drop-down	1 ns	No

1. The square brackets specify that a range of numbers that are available. For example, the parameter name for the start addresses “axi_s axi_start[0-5]” will be expanded to 6 parameter name combinations that range from “axi_s axi_start 0” to “axi_s axi_start 5”. The size of a memory region depends on the “axi_s axi_start” and “axi_s axi_size” parameters. The end address is calculated as Start + Size -1. The size of the memory region must not exceed the value of 0x100000000. If the sum of Start+Size is greater than 0x100000000, the size of the memory region is reduced to the difference: 0x100000000-Start.
2. ARM recommends using the Memory Map Editor (MME) in SoC Designer, which provides centralized viewing and management of the memory regions available to the components in a system. For information about migrating existing systems to use the MME, refer to Chapter 9 of the *SoC Designer User Guide*.
3. When enabled, SoC Designer writes accumulated waveforms to the waveform file in the following situations: when the waveform buffer fills, when validation is paused and when validation finishes, and at the end of each validation run.

Up to six non-overlapping, non-contiguous memory regions can be defined that the AXI slave port registers with the connected AXI master port. A memory region is defined by *axi_s axi_startN* and *axi_s axi_sizeN*. The memory region must lie in an overall 32-bit address space. A size of 0 disables a region. The start address and the size should be a multiple of the block size given by an AXI master port to which the component is connected. This block size is typically a minimum value of 4KB (0x1000).

1.4 Debug Features

The Synchronous AXI to AXI Bridge has a debug interface (CADI) that allows you to track the values of the applicable AXI signals. A view can be accessed in SoC Designer by right-clicking on the component and choosing the appropriate menu entry.

Transactions can be visualized using the transaction monitors attached to connections. By right clicking on any of the connections in SoC Designer, a transaction monitor probe can be attached.

1.4.1 Register Information

The available signals are listed in the following sections:

- AXI Slave Port Signals Registers
- AXI Master Port Signals Registers

1.4.1.1 AXI Slave Port Signals Registers

Table 1-6 shows the AXI slave port signals. See the *ARM AMBA AXI Protocol Specification* for more information about these signals.

Table 1-5 AXI Slave Port Signal Registers

Name	Description	Type
ARIDS	The read address ID.	read-only
ARADDRS	The read address.	read-only
ARVALIDS	Indicates whether the read address is available.	read-only
ARREADYS	Indicates whether the slave is ready to accept the read address.	read-only
ARLENS	The burst length.	read-only
ARSIZES	The burst size.	read-only
ARBURSTS	The burst type.	read-only
ARLOCKS	The lock type.	read-only
ARCACHES	The cache type.	read-only
ARPROTS	The protection type.	read-only
AWIDS	The write address ID.	read-only
AWADDRS	The write address.	read-only
AWVALIDS	Indicates whether the write address is available.	read-only
AWREADYS	Indicates whether the slave is ready to accept the write address.	read-only
AWLENS	The burst length.	read-only
AWSIZES	The burst size.	read-only
AWBURSTS	The burst type.	read-only
AWLOCKS	The lock type.	read-only
AWCACHES	The cache type.	read-only
AWPROTS	The protection type.	read-only
WIDS	The write ID tag.	read-only
WDATAS	The write data.	read-only
WSTRBS	The write strobes.	read-only
WLASTS	The last transfer in a write burst.	read-only
WVALIDS	Indicates whether the write data and strobes are available.	read-only
WREADYS	Indicates whether the slave is ready to accept the write data.	read-only
RIDS	The read ID tag.	read-only
RDATAS	The read data.	read-only
RLASTS	The last transfer in a read burst.	read-only
RVALIDS	Indicates whether the read data is available.	read-only

Table 1-5 AXI Slave Port Signal Registers (continued)

Name	Description	Type
RREADYS	Indicates whether the master is ready to accept the read data and response information.	read-only
RRESPS	The read response.	read-only
BIDS	The response ID.	read-only
BRESPS	The write response.	read-only
BVALIDS	Indicates whether the write response is available.	read-only
BREADYS	Indicates whether the master is ready to accept the response information.	read-only
AWUSERS	Additional master interface signals for the write address channel.	read-only
WUSERS	Additional master interface signals for the write data channel.	read-only
BUSERS	Additional master interface signals for the write response channel.	read-only
ARUSERS	Additional master interface signals for the read address channel.	read-only
RUSERS	Additional master interface signals for the read data channel.	read-only

1.4.1.2 AXI Master Port Signals Registers

Table 1-6 shows the AXI master port signals. See the *ARM AMBA AXI Protocol Specification* for more information about these signals.

Table 1-6 AXI Master Port Signal Registers

Name	Description	Type
ARIDM	The read address ID.	read-only
ARADDRM	The read address.	read-only
ARVALIDM	Indicates whether the read address is available.	read-only
ARREADYM	Indicates whether the slave is ready to accept the read address.	read-only
ARLENM	The burst length.	read-only
ARSIZEM	The burst size.	read-only
ARBURSTM	The burst type.	read-only
ARLOCKM	The lock type.	read-only
ARCACHEM	The cache type.	read-only
ARPROTM	The protection type.	read-only
AWIDM	The write address ID.	read-only
AWADDRM	The write address.	read-only
AWVALIDM	Indicates whether the write address is available.	read-only

Table 1-6 AXI Master Port Signal Registers (continued)

Name	Description	Type
AWREADYM	Indicates whether the slave is ready to accept the write address.	read-only
AWLENM	The burst length.	read-only
AWSIZEM	The burst size.	read-only
AWBURSTM	The burst type.	read-only
AWLOCKM	The lock type.	read-only
AWCACHEM	The cache type.	read-only
AWPROTM	The protection type.	read-only
WIDM	The write ID tag.	read-only
WDATAM	The write data.	read-only
WSTRBM	The write strobes.	read-only
WLASTM	The last transfer in a write burst.	read-only
WVALIDM	Indicates whether the write data and strobes are available.	read-only
WREADYM	Indicates whether the slave is ready to accept the write data.	read-only
RIDM	The read ID tag.	read-only
RDATAM	The read data.	read-only
RLASTM	The last transfer in a read burst.	read-only
RVALIDM	Indicates whether the read data is available.	read-only
RREADYM	Indicates whether the master is ready to accept the read data and response information.	read-only
RRESPM	The read response.	read-only
BIDM	The response ID.	read-only
BRESPM	The write response.	read-only
BVALIDM	Indicates whether the write response is available.	read-only
BREADYM	Indicates whether the master is ready to accept the response information.	read-only
AWUSERM	Additional master interface signals for the write address channel.	read-only
WUSERM	Additional master interface signals for the write data channel.	read-only
BUSERM	Additional master interface signals for the write response channel.	read-only
ARUSERM	Additional master interface signals for the read address channel.	read-only
RUSERM	Additional master interface signals for the read data channel.	read-only

1.5 Available Profiling Data

This Cycle Model does not provide profiling streams, and hence, does not provide profiling information. Transaction related information can be retrieved from the respective bus components.