ZeBu® AMBA Transactor User Manual

Version Q-2020.12, December 2020

SYNOPSYS®

Contents

List of Tables

List of Figures

About This Manual

Overview

This manual describes how to use the ZeBu AMBA transactors with your design being emulated in ZeBu.

Related Documentation

For more information about the ZeBu supported features and limitations, see the ZeBu Release Notes in the ZeBu documentation package corresponding to your software version.

For more relevant information about the usage of the present transactor, see Using Transactors in the training material.

1 Introduction

1.1 AMBA Master/Slave Transactor Overview

The ARM Advanced Microcontroller Bus Architecture (AMBA) is an open-standard, on-chip interconnect specification for the connection and management of functional blocks in system-on-a-chip (SoC) designs. It facilitates development of multi-processor designs with large numbers of controllers and peripherals with a bus architecture. AMBA is widely used on a range of ASIC and SoC parts including applications processors used in modern portable mobile devices like smartphones.
The ZeBu AMBA transactor supports AXI, AXI4, ACELite & ACE protocols.
It can acts as a:

- AXI Master
- AXI Slave
- AXI4 Master
- AXI4 Slave
- ACELite Master
- ACELite Slave
- ACE Master

The ZeBu AMBA master/Slave transactor connects, with high performance, the software model.

of the AMBA Master/Slave client to the DUT mapped in ZeBu. It provides an application layer interface compatible with transaction operations on AMBA Master/Slave components.

The transactor API proposes a software layer so that you can easily re-use existing device driver application software to drive the DUT through its AMBA interface. The API supports AMBA traffic generation from the user software C++ test benches.

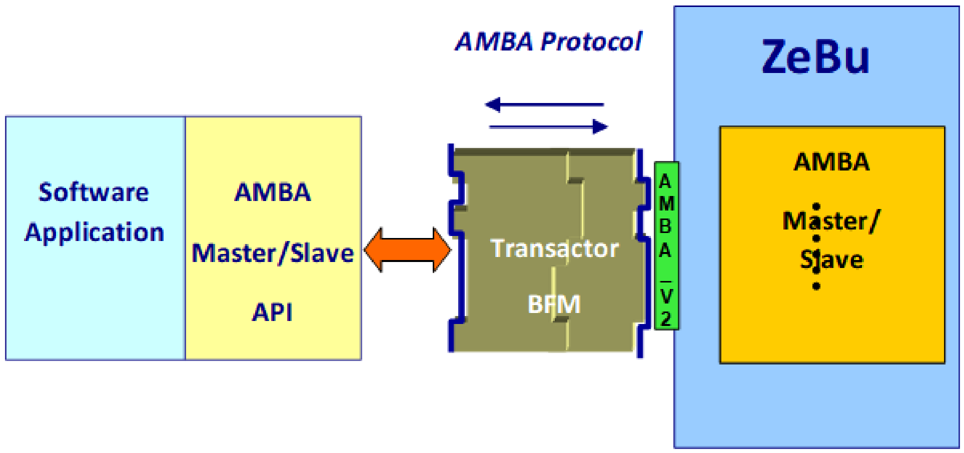

FIGURE 1. ZeBu AMBA Transactor Overview

1.2 Features

1.2.1 Supported AMBA AXI Features

- Compliant with all ARM AMBA AXI data and address widths
- Supports all protocol transfer types, burst types, burst lengths and response types
- Supports phase level APIs for separate address, data and response phases
- Support for burst-based transactions with only start address issued
- Write strobe support to enable sparse data transfer on the write data bus
- Narrow transfer support
- Unaligned address transfer support
- Ability to issue multiple outstanding transactions
- Out of order transaction completion support
- Read data interleaving and write data interleaving (AXI3) supported

1.2.2 Supported AMBA ACE transactor features

- Support for receiving snoop requests via callback
- APIs to drive snoop response
- AutoSnoopResponse feature allows transactor to drive snoop responses on its own without user intervention & also manage master level cache data & status
- Software cache model to emulate Master level cache

1.2.3 AMBA Transactor Features

- Supports multiple transactor instances.
- Easy integration with other ZeBu transactors.
- Support Blocking and Non-Blocking version of transactions
- Programmable wait states or delay insertion on different channels

- Support for bus inactivity detection and timeout (configuration parameter and dynamic change of inactivity timer)
- Support for Dynamic Asynchronous Reset
- SystemVerilog Interface/APIs
- Support for DMTCP
- Support for zRci runtime tool

1.3 Limitations

- Low power interface not supported yet
- Barrier and DVM transactions not supported yet

2 Installation

This section explains the steps to install the ZeBu AMBA Master/Slave transactor.

This section explains the following topics:

- Installing the ZeBu AMBA Master Transactor
- Installing the ZeBu AMBA Slave Transactor
- Package Description
- File Tree

2.1 Installing the ZeBu AMBA Master Transactor

2.1.1 Prerequisites

You must have write permissions to the IP directory and the current directory.

Steps

To install the ZeBu AMBA Master transactor, perform the following steps:

1. Download the AMBA Master compressed shell archive (.sh).
2. Specify the following command on the shell:

$sh xtor_amba_master_svs.<version>.sh install [ZEBU_IP_ROOT]

Note *ZEBU_IP_ROOT is the path to the Zebu IP root directory and is used automatically if no path is specified. If the path is specified and a ZEBU_IP_ROOT environment variable is set, the transactor is installed at the specified path, and the variable is ignored.*

The following message is displayed when the installation process is completed successfully:

```
xtor_amba_master_svs v.<num> has been successfully installed.
```

An error message is displayed, if there is any error during the installation. For example:

```
ERROR: /path/… is not a valid directory.
```

During installation, the symbolic links are created in the following directories for an easy access from all ZeBu tools.

- $ZEBU_IP_ROOT/include
- $ZEBU_IP_ROOT/lib
- $ZEBU_IP_ROOT/vlog

2.2 Installing the ZeBu AMBA Slave Transactor

2.2.1 Prerequisites

You must have write permissions to the IP directory and the current directory.

Steps

To install the ZeBu xtor_amba_slave_svs transactor, perform the following steps:

1. Download the xtor_amba_slave_svs compressed shell archive (.sh).
2. Specify the following command on the shell:

$sh xtor_amba_slave_svs.<version>.sh install [ZEBU_IP_ROOT]

Note *ZEBU_IP_ROOT is the path to the Zebu IP root directory and is used automatically if no path is specified. If the path is specified and a ZEBU_IP_ROOT environment variable is set, the transactor is installed at the specified path, and the variable is ignored.*

The following message is displayed when the installation process is completed successfully:

```
xtor_amba_slave_svs v.<num> has been successfully installed.
```

An error message is displayed, if there is any error during the installation. For example:

```
ERROR: /path/… is not a valid directory.
```

During installation, the symbolic links are created in the following directories for an easy access from all ZeBu tools.

- $ZEBU_IP_ROOT/include
- $ZEBU_IP_ROOT/lib
- $ZEBU_IP_ROOT/vlog

2.3 Package Description

After the ZeBu AMBA Master/Slave transactor is successfully installed, it consists of the following items:

- .so library of the ZeBu AMBA Master/AMBA Slave transactor API
- Header files of the ZeBu AMBA Master/AMBA Slave transactor API
- Verilog files for the ZeBu AMBA Master/AMBA Slave transactor
- Basic examples

2.4 File Tree

The following is an sample file tree of the ZeBu AMBA Master/Slave transactors after package installation:

```
./├── bin
├── bin64 -> bin
├── include
│   ├── amba_defines.hh -> ../XTOR/xtor_amba_slave_svs.$(RELEASE_STRING)/
include/amba_defines.$(RELEASE_STRING).hh
│   ├── AXIDefines.hh -> ../XTOR/xtor_amba_slave_svs.$(RELEASE_STRING)/
include/AXIDefines.$(RELEASE_STRING).hh
│   ├── CoRoutine.hh -> ../XTOR/xtor_amba_slave_svs.$(RELEASE_STRING)/
include/CoRoutine.12.0.hh
│   ├── MPXtorBase.hh -> ../XTOR/xtor_amba_slave_svs.$(RELEASE_STRING)/
include/MPXtorBase.12.0.hh
│   ├── svt_cr_threading.hh -> ../XTOR/
xtor_amba_slave_svs.$(RELEASE_STRING)/include/svt_cr_threading.12.0.hh
│   ├── svt_c_runtime_cfg.hh -> ../XTOR/
xtor_amba_slave_svs.$(RELEASE_STRING)/include/svt_c_runtime_cfg.12.0.hh
│   ├── svt_c_threading.hh -> ../XTOR/
xtor_amba_slave_svs.$(RELEASE_STRING)/include/svt_c_threading.12.0.hh
│   ├── svt_hw_platform.hh -> ../XTOR/
xtor_amba_slave_svs.$(RELEASE_STRING)/include/svt_hw_platform.12.0.hh
│   ├── svt_message_port.hh -> ../XTOR/
xtor_amba_slave_svs.$(RELEASE_STRING)/include/svt_message_port.12.0.hh
│   ├── svt_pthread_threading.hh -> ../XTOR/
xtor_amba_slave_svs.$(RELEASE_STRING)/include/
svt_pthread_threading.12.0.hh
│   ├── svt_report.hh -> ../XTOR/xtor_amba_slave_svs.$(RELEASE_STRING)/
include/svt_report.12.0.hh
│   ├── svt_report_uvm.hh -> ../XTOR/
xtor_amba_slave_svs.$(RELEASE_STRING)/include/svt_report_uvm.12.0.hh
│   ├── svt_simulator_platform.hh -> ../XTOR/
xtor_amba_slave_svs.$(RELEASE_STRING)/include/
svt_simulator_platform.12.0.hh
```

```
│   ├── svt_systemc_threading.hh -> ../XTOR/
xtor_amba_slave_svs.$(RELEASE_STRING)/include/
svt_systemc_threading.12.0.hh
│   ├── svt_systemverilog_threading.hh -> ../XTOR/
xtor_amba_slave_svs.$(RELEASE_STRING)/include/
svt_systemverilog_threading.12.0.hh
│   ├── svt_zebu_platform.hh -> ../XTOR/
xtor_amba_slave_svs.$(RELEASE_STRING)/include/svt_zebu_platform.12.0.hh
│   ├── TopCoRoutine.hh -> ../XTOR/xtor_amba_slave_svs.$(RELEASE_STRING)/
include/TopCoRoutine.12.0.hh
│   ├── TopScheduler.hh -> ../XTOR/xtor_amba_slave_svs.$(RELEASE_STRING)/
include/TopScheduler.12.0.hh
│   ├── xtor_amba_defines_svs.hh -> ../XTOR/
xtor_amba_slave_svs.$(RELEASE_STRING)/include/
xtor_amba_defines_svs.$(RELEASE_STRING).hh
│   ├── xtor_amba_master_svs.hh -> ../XTOR/
xtor_amba_master_svs.$(RELEASE_STRING)/include/
xtor_amba_master_svs.$(RELEASE_STRING).hh
│   ├── xtor_amba_slave_svs.hh -> ../XTOR/
xtor_amba_slave_svs.$(RELEASE_STRING)/include/
xtor_amba_slave_svs.$(RELEASE_STRING).hh
│   ├── Xtor_defines.hh -> ../XTOR/xtor_amba_slave_svs.$(RELEASE_STRING)/
include/Xtor_defines.12.0.hh
│   ├── Xtor.hh -> ../XTOR/xtor_amba_slave_svs.$(RELEASE_STRING)/include/
Xtor.12.0.hh
│   ├── XtorScheduler.hh -> ../XTOR/xtor_amba_slave_svs.$(RELEASE_STRING)/
include/XtorScheduler.12.0.hh
│   ├── ZebuIpRoot.hh -> ../XTOR/xtor_amba_slave_svs.$(RELEASE_STRING)/
include/ZebuIpRoot.12.0.hh
│   └── ZFSDB.hh -> ../XTOR/xtor_amba_slave_svs.$(RELEASE_STRING)/include/
ZFSDB.12.0.hh
├── lib
│   ├── libxtor_amba_master.so -> ../XTOR/
xtor_amba_master_svs.$(RELEASE_STRING)/lib64/
libxtor_amba_master.$(RELEASE_STRING).so
│   ├── libxtor_amba_slave.so -> ../XTOR/
xtor_amba_slave_svs.$(RELEASE_STRING)/lib64/
libxtor_amba_slave.$(RELEASE_STRING).so
```

```
│   ├── libZebuXtorSim.so -> ../XTOR/
xtor_amba_slave_svs.$(RELEASE_STRING)/lib64/libZebuXtorSim.12.0.so
│   ├── libZebuXtor.so -> ../XTOR/xtor_amba_slave_svs.$(RELEASE_STRING)/
lib64/libZebuXtor.12.0.so
│   └── libZebuXtorUVM.so -> ../XTOR/
xtor_amba_slave_svs.$(RELEASE_STRING)/lib64/libZebuXtorUVM.12.0.so
├── lib64 -> lib
├── treee
├── version
│   └── Banner
├── vlog
│   ├── gtech
│   ├── gtech_lib.v -> ../XTOR/xtor_amba_slave_svs.$(RELEASE_STRING)/vlog/
gtech_lib.12.0.v
│   ├── svt_dpi_globals.sv -> ../XTOR/
xtor_amba_slave_svs.$(RELEASE_STRING)/vlog/svt_dpi_globals.12.0.sv
│   ├── svt_dpi_report_uvm.sv -> ../XTOR/
xtor_amba_slave_svs.$(RELEASE_STRING)/vlog/svt_dpi_report_uvm.12.0.sv
│   ├── svt_dpi.sv -> ../XTOR/xtor_amba_slave_svs.$(RELEASE_STRING)/vlog/
svt_dpi.12.0.sv
│   ├── svt_systemverilog_threading.sv -> ../XTOR/
xtor_amba_slave_svs.$(RELEASE_STRING)/vlog/
svt_systemverilog_threading.12.0.sv
│   └── vcs
│       ├── vs_fifo_udpi.sv -> ../../XTOR/
xtor_amba_slave_svs.$(RELEASE_STRING)/vlog/vcs/vs_fifo_udpi.12.0.sv
│       ├── vs_memory.sv -> ../../XTOR/
xtor_amba_slave_svs.$(RELEASE_STRING)/vlog/vcs/vs_memory.12.0.sv
│       ├── xtor_amba_master_acelite_svs.sv -> ../../XTOR/
xtor_amba_master_svs.$(RELEASE_STRING)/vlog/vcs/
xtor_amba_master_acelite_svs.sv
│       ├── xtor_amba_master_ace_svs.sv -> ../../XTOR/
xtor_amba_master_svs.$(RELEASE_STRING)/vlog/vcs/
xtor_amba_master_ace_svs.sv
│       ├── xtor_amba_master_axi3_svs.sv -> ../../XTOR/
xtor_amba_master_svs.$(RELEASE_STRING)/vlog/vcs/
xtor_amba_master_axi3_svs.sv
```

```
│       ├── xtor_amba_master_axi4_svs.sv -> ../../XTOR/
xtor_amba_master_svs.$(RELEASE_STRING)/vlog/vcs/
xtor_amba_master_axi4_svs.sv
│       ├── xtor_amba_slave_acelite_svs.sv -> ../../XTOR/
xtor_amba_slave_svs.$(RELEASE_STRING)/vlog/vcs/
xtor_amba_slave_acelite_svs.sv
│       ├── xtor_amba_slave_axi3_svs.sv -> ../../XTOR/
xtor_amba_slave_svs.$(RELEASE_STRING)/vlog/vcs/
xtor_amba_slave_axi3_svs.sv
│       ├── xtor_amba_slave_axi4_svs.sv -> ../../XTOR/
xtor_amba_slave_svs.$(RELEASE_STRING)/vlog/vcs/
xtor_amba_slave_axi4_svs.sv
│       ├── zebu_vs_apb_master_udpi.sv -> ../../XTOR/
xtor_amba_slave_svs.$(RELEASE_STRING)/vlog/vcs/
zebu_vs_apb_master_udpi.12.0.sv
│       ├── zebu_vs_complex_hwtosw_fifo_udpi.sv -> ../../XTOR/
xtor_amba_slave_svs.$(RELEASE_STRING)/vlog/vcs/
zebu_vs_complex_hwtosw_fifo_udpi.12.0.sv
│       ├── zebu_vs_complex_rst_and_clk_udpi.sv -> ../../XTOR/
xtor_amba_slave_svs.$(RELEASE_STRING)/vlog/vcs/
zebu_vs_complex_rst_and_clk_udpi.12.0.sv
│       ├── zebu_vs_complex_swtohw_fifo_udpi.sv -> ../../XTOR/
xtor_amba_slave_svs.$(RELEASE_STRING)/vlog/vcs/
zebu_vs_complex_swtohw_fifo_udpi.12.0.sv
│       ├── zebu_vs_from_sw_fifo.sv -> ../../XTOR/
xtor_amba_slave_svs.$(RELEASE_STRING)/vlog/vcs/
zebu_vs_from_sw_fifo.12.0.sv
│       ├── zebu_vs_simple_hwtosw_fifo_udpi.sv -> ../../XTOR/
xtor_amba_slave_svs.$(RELEASE_STRING)/vlog/vcs/
zebu_vs_simple_hwtosw_fifo_udpi.12.0.sv
│       ├── zebu_vs_simple_rst_and_clk_udpi.sv -> ../../XTOR/
xtor_amba_slave_svs.$(RELEASE_STRING)/vlog/vcs/
zebu_vs_simple_rst_and_clk_udpi.12.0.sv
│       ├── zebu_vs_simple_swtohw_fifo_udpi.sv -> ../../XTOR/
xtor_amba_slave_svs.$(RELEASE_STRING)/vlog/vcs/
zebu_vs_simple_swtohw_fifo_udpi.12.0.sv
│       └── zebu_vs_to_sw_fifo.sv -> ../../XTOR/
xtor_amba_slave_svs.$(RELEASE_STRING)/vlog/vcs/zebu_vs_to_sw_fifo.12.0.sv
├── XTOR
```

```
|   ├── xtor_amba_master_svs.$(RELEASE_STRING)
|   |   ├── doc
|   |   |   └── foss
|   |   |       └── ZX-XTOR-Library_20180911_FOSS.PDF
|   |   ├── example
|   |   |   ├── acelite_master_dut_slave
|   |   |   |   ├── src
|   |   |   |   |   ├── bench
|   |   |   |   |   |   ├── tb_common.hh
|   |   |   |   |   |   ├── tb_top.cc
|   |   |   |   |   |   └── tests
|   |   |   |   |   |       ├── ts.channel_delay_test.hh
|   |   |   |   |   |       ├── ts.trivial_test.hh
|   |   |   |   |   |       ├── ts.trivial_test_nb.hh
|   |   |   |   |   |       └── ts.trivial_test_phase.hh
|   |   |   |   |   ├── dut
|   |   |   |   |   |   ├── axi_slave_dut.v
|   |   |   |   |   |   ├── axi_slave_memory_ctrl.v
|   |   |   |   |   |   ├── hw_top.sv
|   |   |   |   |   |   ├── tb_top.sv
|   |   |   |   |   |   └── zebu_sram2Mx.v
|   |   |   |   |   └── env
|   |   |   |   |       ├── axi.rc
|   |   |   |   |       ├── designFeatures
|   |   |   |   |       ├── designFeatures_sem
|   |   |   |   |       ├── project_sem.utf
|   |   |   |   |       └── project.utf
|   |   |   |   └── zebu
|   |   |   |       └── Makefile
|   |   |   ├── axi4_master_dut_slave
|   |   |   |   ├── src
|   |   |   |   |   ├── bench
|   |   |   |   |   |   ├── tb_common.hh
```

```
│   │   │   │   │   │   ├── tb_top.cc
│   │   │   │   │   │   └── tests
│   │   │   │   │   │       ├── ts.channel_delay_test.hh
│   │   │   │   │   │       ├── ts.trivial_test.hh
│   │   │   │   │   │       ├── ts.trivial_test_nb.hh
│   │   │   │   │   │       └── ts.trivial_test_phase.hh
│   │   │   │   │   ├── dut
│   │   │   │   │   │   ├── axi_slave_dut.v
│   │   │   │   │   │   ├── axi_slave_memory_ctrl.v
│   │   │   │   │   │   ├── hw_top.sv
│   │   │   │   │   │   ├── tb_top.sv
│   │   │   │   │   │   └── zebu_sram2Mx.v
│   │   │   │   │   └── env
│   │   │   │   │       ├── axi.rc
│   │   │   │   │       ├── designFeatures
│   │   │   │   │       ├── designFeatures_sem
│   │   │   │   │       ├── project_sem.utf
│   │   │   │   │       └── project.utf
│   │   │   │   └── zebu
│   │   │   │       └── Makefile
│   │   │   ├── axi_master_dut_slave
│   │   │   │   ├── src
│   │   │   │   │   ├── bench
│   │   │   │   │   │   ├── tb_common.hh
│   │   │   │   │   │   ├── tb_top.cc
│   │   │   │   │   │   └── tests
│   │   │   │   │   │       ├── ts.channel_delay_test.hh
│   │   │   │   │   │       ├── ts.randomize_test.hh
│   │   │   │   │   │       ├── ts.trivial_test.hh
│   │   │   │   │   │       ├── ts.trivial_test_nb.hh
│   │   │   │   │   │       └── ts.trivial_test_phase.hh
│   │   │   │   │   ├── dut
│   │   │   │   │   │   ├── axi_slave_dut.v
│   │   │   │   │   │   ├── axi_slave_memory_ctrl.v
│   │   │   │   │   │   ├── hw_top.sv
│   │   │   │   │   │   ├── tb_top.sv
│   │   │   │   │   │   └── zebu_sram2Mx.v
│   │   │   │   │   └── env
│   │   │   │   │       ├── axi.rc
│   │   │   │   │       ├── designFeatures
│   │   │   │   │       ├── designFeatures_sem
│   │   │   │   │       ├── project_sem.utf
│   │   │   │   │       └── project.utf
│   │   │   │   └── zebu
│   │   │   │       └── Makefile
│   │   │   ├── axi_master_dut_slave_exe
│   │   │   │   ├── src
│   │   │   │   │   ├── bench
│   │   │   │   │   │   └── testbench.cc
│   │   │   │   │   ├── dut
│   │   │   │   │   │   ├── axi_slave_dut.v
│   │   │   │   │   │   ├── axi_slave_memory_ctrl.v
│   │   │   │   │   │   ├── hw_top.sv
│   │   │   │   │   │   ├── tb_top.sv
│   │   │   │   │   │   └── zebu_sram2Mx.v
│   │   │   │   │   └── env
│   │   │   │   │       ├── axi.rc
│   │   │   │   │       ├── designFeatures
│   │   │   │   │       └── project.utf
│   │   │   │   └── zebu
│   │   │   │       └── Makefile
│   │   │   ├── axi_zRci
│   │   │   │   ├── src
│   │   │   │   │   ├── bench
│   │   │   │   │   │   ├── tb_common.hh
│   │   │   │   │   │   ├── tb_top_zRci.cc
│   │   │   │   │   │   └── tests
│   │   │   │   │   │       ├── ts.channel_delay_test.hh
│   │   │   │   │   │       ├── ts.randomize_test.hh
│   │   │   │   │   │       ├── ts.trivial_test.hh
│   │   │   │   │   │       ├── ts.trivial_test_nb.hh
│   │   │   │   │   │       └── ts.trivial_test_phase.hh
│   │   │   │   │   ├── dut
│   │   │   │   │   │   ├── axi_slave_dut.v
│   │   │   │   │   │   ├── axi_slave_memory_ctrl.v
│   │   │   │   │   │   ├── hw_top.sv
│   │   │   │   │   │   └── zebu_sram2Mx.v
│   │   │   │   │   └── env
│   │   │   │   │       ├── axi.rc
│   │   │   │   │       ├── designFeatures
│   │   │   │   │       ├── project.utf
│   │   │   │   │       └── zRci.tcl
│   │   │   │   └── zebu
│   │   │   │       └── Makefile
│   │   │   └── README
│   │   ├── include
│   │   │   ├── AXIDefines.hh -> AXIDefines.$(RELEASE_STRING).hh
│   │   │   ├── AXIDefines.$(RELEASE_STRING).hh
│   │   │   ├── CoRoutine.12.0.hh
│   │   │   ├── CoRoutine.hh -> CoRoutine.12.0.hh
│   │   │   ├── MPXtorBase.12.0.hh
│   │   │   ├── MPXtorBase.hh -> MPXtorBase.12.0.hh
│   │   │   ├── svt_cr_threading.12.0.hh
│   │   │   ├── svt_cr_threading.hh -> svt_cr_threading.12.0.hh
│   │   │   ├── svt_c_runtime_cfg.12.0.hh
│   │   │   ├── svt_c_runtime_cfg.hh -> svt_c_runtime_cfg.12.0.hh
│   │   │   ├── svt_c_threading.12.0.hh
│   │   │   ├── svt_c_threading.hh -> svt_c_threading.12.0.hh
│   │   │   ├── svt_hw_platform.12.0.hh
```

```
|   |   |   ├── svt_hw_platform.hh -> svt_hw_platform.12.0.hh
|   |   |   ├── svt_message_port.12.0.hh
|   |   |   ├── svt_message_port.hh -> svt_message_port.12.0.hh
|   |   |   ├── svt_pthread_threading.12.0.hh
|   |   |   ├── svt_pthread_threading.hh -> svt_pthread_threading.12.0.hh
|   |   |   ├── svt_report.12.0.hh
|   |   |   ├── svt_report.hh -> svt_report.12.0.hh
|   |   |   ├── svt_report_uvm.12.0.hh
|   |   |   ├── svt_report_uvm.hh -> svt_report_uvm.12.0.hh
|   |   |   ├── svt_simulator_platform.12.0.hh
|   |   |   ├── svt_simulator_platform.hh ->
svt_simulator_platform.12.0.hh
|   |   |   ├── svt_systemc_threading.12.0.hh
|   |   |   ├── svt_systemc_threading.hh -> svt_systemc_threading.12.0.hh
|   |   |   ├── svt_systemverilog_threading.12.0.hh
|   |   |   ├── svt_systemverilog_threading.hh ->
svt_systemverilog_threading.12.0.hh
|   |   |   ├── svt_zebu_platform.12.0.hh
|   |   |   ├── svt_zebu_platform.hh -> svt_zebu_platform.12.0.hh
|   |   |   ├── TopCoRoutine.12.0.hh
|   |   |   ├── TopCoRoutine.hh -> TopCoRoutine.12.0.hh
|   |   |   ├── TopScheduler.12.0.hh
|   |   |   ├── TopScheduler.hh -> TopScheduler.12.0.hh
|   |   |   ├── Xtor.12.0.hh
|   |   |   ├── xtor_amba_defines_svs.hh ->
xtor_amba_defines_svs.$(RELEASE_STRING).hh
|   |   |   ├── xtor_amba_defines_svs.$(RELEASE_STRING).hh
|   |   |   ├── xtor_amba_master_svs.hh ->
xtor_amba_master_svs.$(RELEASE_STRING).hh
|   |   |   ├── xtor_amba_master_svs.$(RELEASE_STRING).hh
|   |   |   ├── Xtor_defines.12.0.hh
|   |   |   ├── Xtor_defines.hh -> Xtor_defines.12.0.hh
|   |   |   ├── Xtor.hh -> Xtor.12.0.hh
|   |   |   ├── XtorScheduler.12.0.hh
```

```
│   │   │   ├── XtorScheduler.hh -> XtorScheduler.12.0.hh
│   │   │   ├── ZebuIpRoot.12.0.hh
│   │   │   ├── ZebuIpRoot.hh -> ZebuIpRoot.12.0.hh
│   │   │   ├── ZFSDB.12.0.hh
│   │   │   └── ZFSDB.hh -> ZFSDB.12.0.hh
│   │   ├── lib -> lib64
│   │   ├── lib64
│   │   │   ├── libxtor_amba_master.so ->
libxtor_amba_master.$(RELEASE_STRING).so
│   │   │   ├── libxtor_amba_master.$(RELEASE_STRING).so
│   │   │   ├── libZebuXtor.12.0.so
│   │   │   ├── libZebuXtorSim.12.0.so
│   │   │   ├── libZebuXtorSim.so -> libZebuXtorSim.12.0.so
│   │   │   ├── libZebuXtor.so -> libZebuXtor.12.0.so
│   │   │   ├── libZebuXtorUVM.12.0.so
│   │   │   └── libZebuXtorUVM.so -> libZebuXtorUVM.12.0.so
│   │   └── vlog
│   │       ├── gtech
│   │       ├── gtech_lib.12.0.v
│   │       ├── gtech_lib.v -> gtech_lib.12.0.v
│   │       ├── svt_dpi.12.0.sv
│   │       ├── svt_dpi_globals.12.0.sv
│   │       ├── svt_dpi_globals.sv -> svt_dpi_globals.12.0.sv
│   │       ├── svt_dpi_report_uvm.12.0.sv
│   │       ├── svt_dpi_report_uvm.sv -> svt_dpi_report_uvm.12.0.sv
│   │       ├── svt_dpi.sv -> svt_dpi.12.0.sv
│   │       ├── svt_systemverilog_threading.12.0.sv
│   │       ├── svt_systemverilog_threading.sv ->
svt_systemverilog_threading.12.0.sv
│   │       └── vcs
│   │           ├── vs_fifo_udpi.12.0.sv
│   │           ├── vs_fifo_udpi.sv -> vs_fifo_udpi.12.0.sv
│   │           ├── vs_memory.12.0.sv
│   │             ├── vs_memory.sv -> vs_memory.12.0.sv
│   │             ├── xtor_amba_master_acelite_svs.sv
│   │             ├── xtor_amba_master_ace_svs.sv
│   │             ├── xtor_amba_master_axi3_svs.sv
│   │             ├── xtor_amba_master_axi4_svs.sv
│   │             ├── zebu_vs_apb_master_udpi.12.0.sv
│   │             ├── zebu_vs_apb_master_udpi.sv ->
zebu_vs_apb_master_udpi.12.0.sv
│   │             ├── zebu_vs_complex_hwtosw_fifo_udpi.12.0.sv
│   │             ├── zebu_vs_complex_hwtosw_fifo_udpi.sv ->
zebu_vs_complex_hwtosw_fifo_udpi.12.0.sv
│   │             ├── zebu_vs_complex_rst_and_clk_udpi.12.0.sv
│   │             ├── zebu_vs_complex_rst_and_clk_udpi.sv ->
zebu_vs_complex_rst_and_clk_udpi.12.0.sv
│   │             ├── zebu_vs_complex_swtohw_fifo_udpi.12.0.sv
│   │             ├── zebu_vs_complex_swtohw_fifo_udpi.sv ->
zebu_vs_complex_swtohw_fifo_udpi.12.0.sv
│   │             ├── zebu_vs_from_sw_fifo.12.0.sv
│   │            ├── zebu_vs_from_sw_fifo.sv -> zebu_vs_from_sw_fifo.12.0.sv
│   │             ├── zebu_vs_simple_hwtosw_fifo_udpi.12.0.sv
│   │             ├── zebu_vs_simple_hwtosw_fifo_udpi.sv ->
zebu_vs_simple_hwtosw_fifo_udpi.12.0.sv
│   │             ├── zebu_vs_simple_rst_and_clk_udpi.12.0.sv
│   │             ├── zebu_vs_simple_rst_and_clk_udpi.sv ->
zebu_vs_simple_rst_and_clk_udpi.12.0.sv
│   │             ├── zebu_vs_simple_swtohw_fifo_udpi.12.0.sv
│   │             ├── zebu_vs_simple_swtohw_fifo_udpi.sv ->
zebu_vs_simple_swtohw_fifo_udpi.12.0.sv
│   │             ├── zebu_vs_to_sw_fifo.12.0.sv
│   │             └── zebu_vs_to_sw_fifo.sv -> zebu_vs_to_sw_fifo.12.0.sv
│   └── xtor_amba_slave_svs.$(RELEASE_STRING)
│       ├── doc
│       │   └── foss
│       │       └── ZX-XTOR-Library_20180911_FOSS.PDF
```

```
|         ├── example
|         |    ├── acelite_master_slave
|         |    |    ├── src
|         |    |    |    ├── bench
|         |    |    |    |    ├── tb_common.hh
|         |    |    |    |    ├── tb_top.cc
|         |    |    |    |    └── tests
|         |    |    |    |         ├── ts.multi_txn_test.hh
|         |    |    |    |         └── ts.trivial_test.hh
|         |    |    |    ├── dut
|         |    |    |    |    ├── hw_top.sv
|         |    |    |    |    └── tb_top.sv
|         |    |    |    └── env
|         |    |    |         ├── designFeatures
|         |    |    |         ├── designFeatures_sem
|         |    |    |         ├── project_sem.utf
|         |    |    |         └── project.utf
|         |    |    └── zebu
|         |    |         └── Makefile
|         |    ├── axi4_master_slave
|         |    |    ├── src
|         |    |    |    ├── bench
|         |    |    |    |    ├── tb_common.hh
|         |    |    |    |    ├── tb_top.cc
|         |    |    |    |    └── tests
|         |    |    |    |         ├── ts.multi_txn_test.hh
|         |    |    |    |         └── ts.trivial_test.hh
|         |    |    |    ├── dut
|         |    |    |    |    ├── hw_top.sv
|         |    |    |    |    └── tb_top.sv
|         |    |    |    └── env
|         |    |    |         ├── designFeatures
|         |    |    |         ├── designFeatures_sem
```

```
│         │    │    │         ├── project_sem.utf
│         │    │    │         └── project.utf
│         │    │    └── zebu
│         │    │         └── Makefile
│         │    └── axi_master_slave
│         │         ├── src
│         │         │    ├── bench
│         │         │    │    ├── tb_common.hh
│         │         │    │    ├── tb_top.cc
│         │         │    │    └── tests
│         │         │    │         ├── ts.multi_txn_test.hh
│         │         │    │         ├── ts.out_of_order_txn_test.hh
│         │         │    │         ├── ts.random_txn_test.hh
│         │         │    │         └── ts.trivial_test.hh
│         │         │    ├── dut
│         │         │    │    ├── hw_top.sv
│         │         │    │    └── tb_top.sv
│         │         │    └── env
│         │         │         ├── designFeatures
│         │         │         ├── designFeatures_sem
│         │         │         ├── project_sem.utf
│         │         │         └── project.utf
│         │         └── zebu
│         │              └── Makefile
│         ├── include
│         │    ├── AXIDefines.hh -> AXIDefines.$(RELEASE_STRING).hh
│         │    ├── AXIDefines.$(RELEASE_STRING).hh
│         │    ├── CoRoutine.12.0.hh
│         │    ├── CoRoutine.hh -> CoRoutine.12.0.hh
│         │    ├── MPXtorBase.12.0.hh
│         │    ├── MPXtorBase.hh -> MPXtorBase.12.0.hh
│         │    ├── svt_cr_threading.12.0.hh
│         │    ├── svt_cr_threading.hh -> svt_cr_threading.12.0.hh
│       │   ├── svt_c_runtime_cfg.12.0.hh
│       │   ├── svt_c_runtime_cfg.hh -> svt_c_runtime_cfg.12.0.hh
│       │   ├── svt_c_threading.12.0.hh
│       │   ├── svt_c_threading.hh -> svt_c_threading.12.0.hh
│       │   ├── svt_hw_platform.12.0.hh
│       │   ├── svt_hw_platform.hh -> svt_hw_platform.12.0.hh
│       │   ├── svt_message_port.12.0.hh
│       │   ├── svt_message_port.hh -> svt_message_port.12.0.hh
│       │   ├── svt_pthread_threading.12.0.hh
│       │   ├── svt_pthread_threading.hh -> svt_pthread_threading.12.0.hh
│       │   ├── svt_report.12.0.hh
│       │   ├── svt_report.hh -> svt_report.12.0.hh
│       │   ├── svt_report_uvm.12.0.hh
│       │   ├── svt_report_uvm.hh -> svt_report_uvm.12.0.hh
│       │   ├── svt_simulator_platform.12.0.hh
│       │   ├── svt_simulator_platform.hh ->
svt_simulator_platform.12.0.hh
│       │   ├── svt_systemc_threading.12.0.hh
│       │   ├── svt_systemc_threading.hh -> svt_systemc_threading.12.0.hh
│       │   ├── svt_systemverilog_threading.12.0.hh
│       │   ├── svt_systemverilog_threading.hh ->
svt_systemverilog_threading.12.0.hh
│       │   ├── svt_zebu_platform.12.0.hh
│       │   ├── svt_zebu_platform.hh -> svt_zebu_platform.12.0.hh
│       │   ├── TopCoRoutine.12.0.hh
│       │   ├── TopCoRoutine.hh -> TopCoRoutine.12.0.hh
│       │   ├── TopScheduler.12.0.hh
│       │   ├── TopScheduler.hh -> TopScheduler.12.0.hh
│       │   ├── Xtor.12.0.hh
│       │   ├── xtor_amba_defines_svs.hh ->
xtor_amba_defines_svs.$(RELEASE_STRING).hh
│       │   ├── xtor_amba_defines_svs.$(RELEASE_STRING).hh
│       │   ├── xtor_amba_slave_svs.hh ->
xtor_amba_slave_svs.$(RELEASE_STRING).hh
```

```
|          |    ├── xtor_amba_slave_svs.$(RELEASE_STRING).hh
|          |    ├── Xtor_defines.12.0.hh
|          |    ├── Xtor_defines.hh -> Xtor_defines.12.0.hh
|          |    ├── Xtor.hh -> Xtor.12.0.hh
|          |    ├── XtorScheduler.12.0.hh
|          |    ├── XtorScheduler.hh -> XtorScheduler.12.0.hh
|          |    ├── ZebuIpRoot.12.0.hh
|          |    ├── ZebuIpRoot.hh -> ZebuIpRoot.12.0.hh
|          |    ├── ZFSDB.12.0.hh
|          |    └── ZFSDB.hh -> ZFSDB.12.0.hh
|          ├── lib -> lib64
|          ├── lib64
|          |    ├── libxtor_amba_slave.so ->
libxtor_amba_slave.$(RELEASE_STRING).so
|          |    ├── libxtor_amba_slave.$(RELEASE_STRING).so
|          |    ├── libZebuXtor.12.0.so
|          |    ├── libZebuXtorSim.12.0.so
|          |    ├── libZebuXtorSim.so -> libZebuXtorSim.12.0.so
|          |    ├── libZebuXtor.so -> libZebuXtor.12.0.so
|          |    ├── libZebuXtorUVM.12.0.so
|          |    └── libZebuXtorUVM.so -> libZebuXtorUVM.12.0.so
|          └── vlog
|               ├── gtech
|               ├── gtech_lib.12.0.v
|               ├── gtech_lib.v -> gtech_lib.12.0.v
|               ├── svt_dpi.12.0.sv
|               ├── svt_dpi_globals.12.0.sv
|               ├── svt_dpi_globals.sv -> svt_dpi_globals.12.0.sv
|               ├── svt_dpi_report_uvm.12.0.sv
|               ├── svt_dpi_report_uvm.sv -> svt_dpi_report_uvm.12.0.sv
|               ├── svt_dpi.sv -> svt_dpi.12.0.sv
|               ├── svt_systemverilog_threading.12.0.sv
```

```
│           ├── svt_systemverilog_threading.sv ->
svt_systemverilog_threading.12.0.sv
│           └── vcs
│               ├── vs_fifo_udpi.12.0.sv
│               ├── vs_fifo_udpi.sv -> vs_fifo_udpi.12.0.sv
│               ├── vs_memory.12.0.sv
│               ├── vs_memory.sv -> vs_memory.12.0.sv
│               ├── xtor_amba_slave_acelite_svs.sv
│               ├── xtor_amba_slave_axi3_svs.sv
│               ├── xtor_amba_slave_axi4_svs.sv
│               ├── zebu_vs_apb_master_udpi.12.0.sv
│               ├── zebu_vs_apb_master_udpi.sv ->
zebu_vs_apb_master_udpi.12.0.sv
│               ├── zebu_vs_complex_hwtosw_fifo_udpi.12.0.sv
│               ├── zebu_vs_complex_hwtosw_fifo_udpi.sv ->
zebu_vs_complex_hwtosw_fifo_udpi.12.0.sv
│               ├── zebu_vs_complex_rst_and_clk_udpi.12.0.sv
│               ├── zebu_vs_complex_rst_and_clk_udpi.sv ->
zebu_vs_complex_rst_and_clk_udpi.12.0.sv
│               ├── zebu_vs_complex_swtohw_fifo_udpi.12.0.sv
│               ├── zebu_vs_complex_swtohw_fifo_udpi.sv ->
zebu_vs_complex_swtohw_fifo_udpi.12.0.sv
│               ├── zebu_vs_from_sw_fifo.12.0.sv
│              ├── zebu_vs_from_sw_fifo.sv -> zebu_vs_from_sw_fifo.12.0.sv
│               ├── zebu_vs_simple_hwtosw_fifo_udpi.12.0.sv
│               ├── zebu_vs_simple_hwtosw_fifo_udpi.sv ->
zebu_vs_simple_hwtosw_fifo_udpi.12.0.sv
│               ├── zebu_vs_simple_rst_and_clk_udpi.12.0.sv
│               ├── zebu_vs_simple_rst_and_clk_udpi.sv ->
zebu_vs_simple_rst_and_clk_udpi.12.0.sv
│               ├── zebu_vs_simple_swtohw_fifo_udpi.12.0.sv
│               ├── zebu_vs_simple_swtohw_fifo_udpi.sv ->
zebu_vs_simple_swtohw_fifo_udpi.12.0.sv
│               ├── zebu_vs_to_sw_fifo.12.0.sv
│               └── zebu_vs_to_sw_fifo.sv -> zebu_vs_to_sw_fifo.12.0.sv
```

```
├── xtor_amba_master_svs -> XTOR/xtor_amba_master_svs.$(RELEASE_STRING)
└── xtor_amba_slave_svs -> XTOR/xtor_amba_slave_svs.$(RELEASE_STRING)
```

3 Hardware Interface

The ZeBu AMBA transactor connects to the DUT port through a directional interface.

3.1 AXI Master Interface Description

The following figure illustrates an example of ZeBu AMBA AXI Master connection:

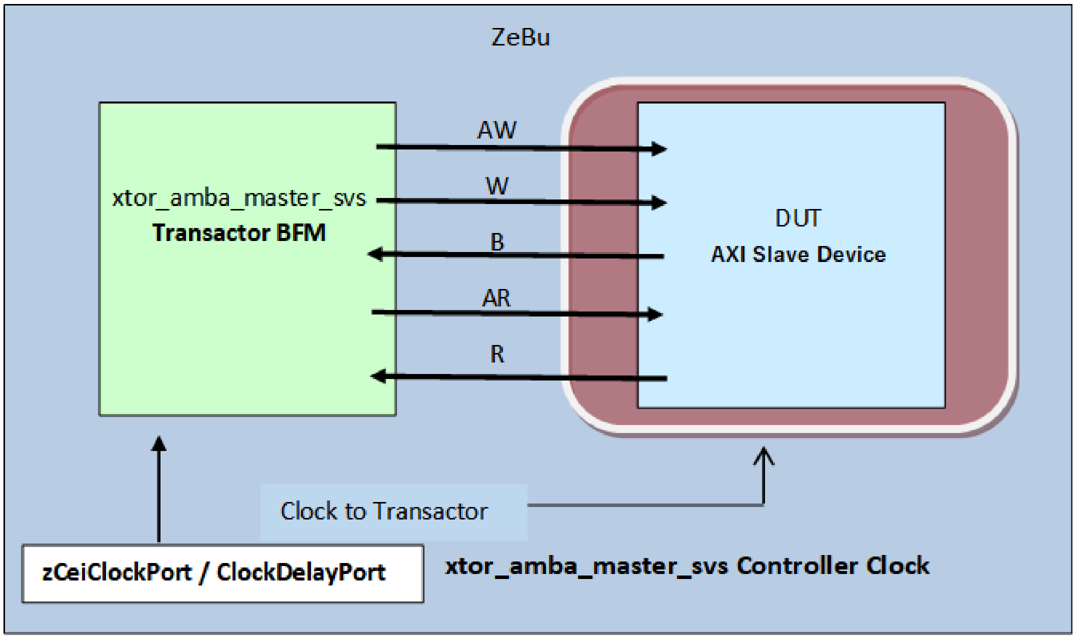

FIGURE 2. ZeBu AMBA AXI Master connection Example

3.2 AXI Slave Interface Description

The following figure illustrates an example of ZeBu AMBA AXI Slave connection:

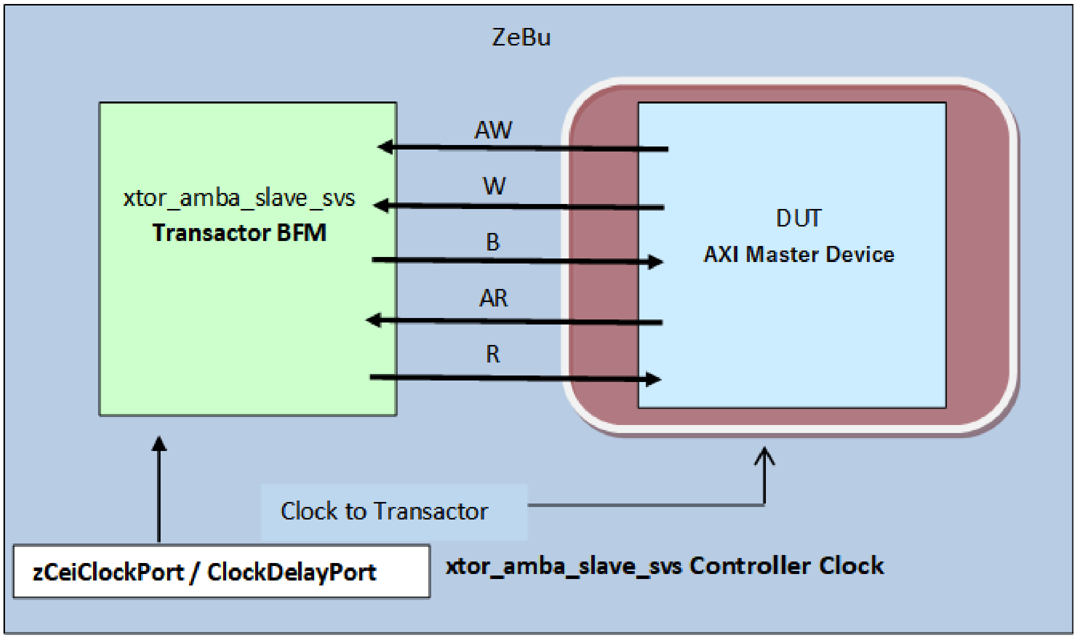

FIGURE 3. ZeBu AMBA AXI Slave connection Example

3.3 ACE Master Description

The following figure illustrates an example of ZeBu ACE Master connection:

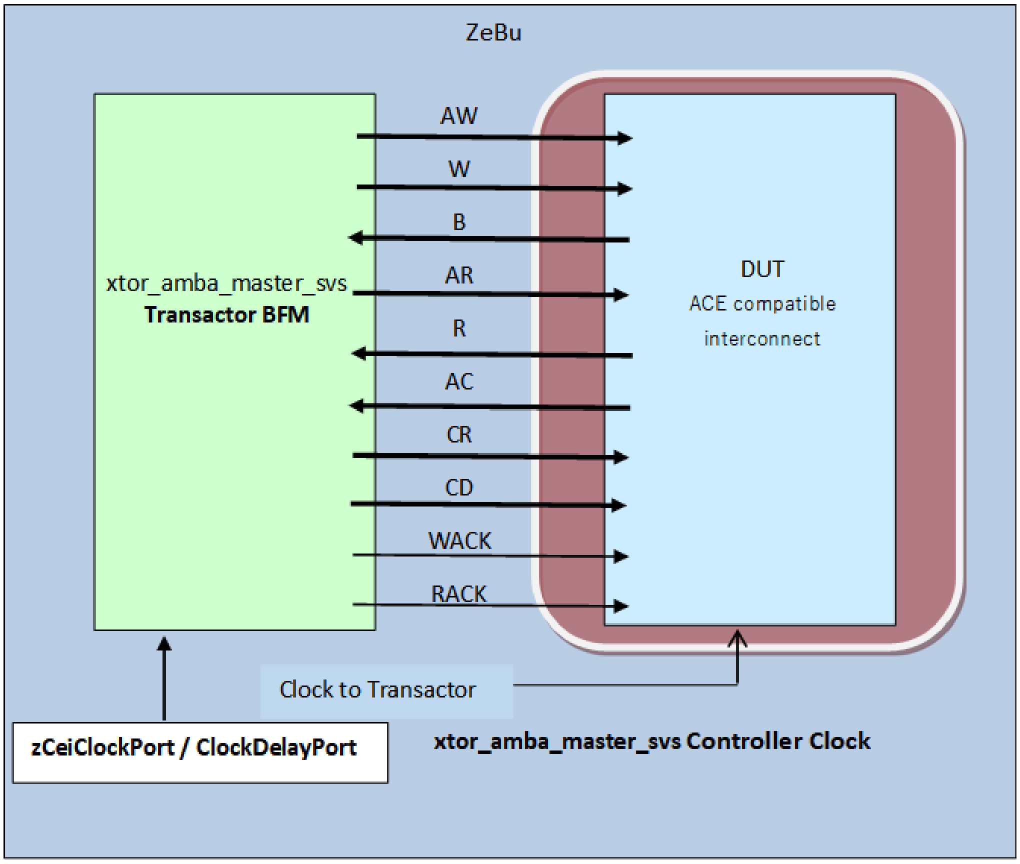

FIGURE 4. ZeBu AMBA ACE Master Connection Example

3.4 ZeBu AMBA Transactor Clock Distribution

The ZeBu AMBA Master/AMBA Slave controller of the DUT uses a clock source that must be the same as the one used by the transactor reference clock. This clock must be defined as the controlled clock for the transactor.

3.5 ZeBu AMBA Interface Description

3.5.1 Signal List

The following table lists the signals for the ZeBu `AMBA Master` Transactor.

TABLE 1 ZeBu `AMBA Master` Transactor Signal List

Symbol	Size in bits	Transactor Type	DUT Type	Description
ACLK	1	Input	Output or Input	AXI clock
ARESETN	1	Input	Output	AXI reset signal driven by DUT
AWADDR	ADDR_WIDTH	Output	Input	AW address
AWVALID	1	Output	Input	AW valid
AWID, AWLEN, AWSIZE, AWBURST, AWLOCK, AWCACHE, AWPROT	ID_WIDTH, BLEN_WIDTH , 3 , 2, 4 ,3	Output	Input	AW control signals:
AWREADY	1	Input	Output	Slave AWREADY signal. Indicates that the Slave is ready to receive AW channel
ARADDR	ADDR_WIDTH	Output	Input	AR address
ARVALID	1	Output	Input	AR valid

TABLE 1 ZeBu `AMBA Master` Transactor Signal List

ARID, ARLEN, ARSIZE, ARBURST, ARLOCK, ARCACHE, ARPROT	ID WIDTH, BLEN_WIDTH , 3 , 2, 4 ,3	Output	Input	AR control signals
ARREADY	1	Input	Output	Slave ARREADY signal. Indicates that the Slave is ready to receive AR channel
WDATA	DATA_WIDTH	Output	Input	Data to write
WID	ID_WIDTH	Output	Input	AXI3 only. ID value
WSTRB	WSTRB_WIDTH/8	Output	Input	Byte-lane of the data to write
WVALID	1	Output	Input	W valid
WLAST	1	Output	Input	Indicates last data of a burst.
WREADY	1	Input	Output	Indicates that the Slave is ready to receive data
BRESP	2	Input	Output	Response from the Slave
BID	ID_WIDTH	Input	Output	ID value of the response
BVALID	1	Input	Output	W valid
BREADY	1	Output	Input	Master indicates if ready to get response
RDATA	DATA_WIDTH	Input	Output	Read data

TABLE 1 ZeBu `AMBA Master` Transactor Signal List

RID	ID_WIDTH	Input	Output	ID value of the response
RRESP	2	Input	Output	Response from the Slave
RVALID	1	Input	Output	R valid
RLAST	1	Input	Output	Indicates last data of a burst
RREADY	1	Output	Input	Master indicates if ready to get response
AWQOS,AWREGIO N	4,4	Output	Input	AXI4 only. Read address channel QOS & Region control
ARQOS,ARREGION	4,4	Output	Input	AXI4 only. Write address channel QOS & Region control
AWUSER,ARUSER, WUSER	USER_WIDTH	Output	Input	AXI4 only. User signal for AW, AR & W channel respectively.
BUSER, RUSER	USER_WIDTH	Input	Output	AXI4 only. User signal for B & R channel respectively
AWSNOOP,AWDO MAIN,AWBAR	3,2,2	Output	Input	ACELite/ACE only. Write address channel Snoop, Domain & Barrier control
ARSNOOP,ARDOM AIN,ARBAR	4,2,2	Output	Input	ACELite/ACE only. Read address channel Snoop, Domain & Barrier control

TABLE 1 ZeBu `AMBA Master` Transactor Signal List

ACREADY	1	Output	Input	ACE only. ACE interconnect ACREADY signal. Indicates that the Master is ready to receive AW channel
AWUNIQUE	1	Output	Input	ACE only. Can be used with various write transactions to improve the operation of lower levels of the cache hierarchy.
ACADDR	SNOOP_ ADDR_ WIDTH	Input	Output	ACE only. Snoop address
ACVALID	1	Input	Output	ACE only. AC valid
CRREADY	1	Input	Output	ACE only. Slave CRREADY signal. Indicates that the interconnect is ready to receive AW channel
CRRESP	ADDR_WIDTH	Output	Input	ACE only. Snoop Response
CRVALID	1	Output	Input	ACE only. CR valid
CDREADY	1	Input	Output	ACE only. Slave CDREADY signal. Indicates that the Slave is ready to receive AW channel
CDDATA	ADDR_WIDTH	Output	Input	ACE only. Snoop data
CDVALID	1	Output	Input	ACE only. CD valid
RACK	1	Output	Input	ACE only. Read acknowledge from Master

TABLE 1 ZeBu `AMBA Master` Transactor Signal List

WACK	1	Output	Input	ACE only. Write acknowledge from Master
xtor_info_port	32	Output	NA	Transactor info port for master,this port is used for debugging

The following table lists the signals for the xtor_amba_slave_svs Transactor:

TABLE 2 ZeBu xtor_amba_slave_svs Transactor Signal List

Symbol	**Size in bits**	**Transactor Type**	**DUT Type**	**Description**
ACLK	1	Input	Output or Input	AXI clock
ARESETN	1	Input	Output	AXI reset signal driven by DUT
AWADDR	ADDR_WIDTH	Input	Output	AW address
AWVALID	1	Input	Output	AW valid
AWID, AWLEN, AWSIZE, AWBURST, AWLOCK, AWVACHE, AWPROT	ID_WIDTH,BLEN_WIDTH,3, 2,4,3	Input	Output	AW control signals:
AWREADY	1	Output	Input	Slave AWREADY signal. Indicates that the Slave is ready to receive AW channel
ARADDR	ADDR_WIDTH	Input	Output	AR address

TABLE 2 ZeBu xtor_amba_slave_svs Transactor Signal List

Symbol	Size in bits	Transactor Type	DUT Type	Description
ARVALID	1	Input	Output	AR valid
ARID, ARLEN, ARSIZE, ARBURST, ARLOCK, ARVACHE, ARPROT	ID_WIDTH,BLEN_WIDTH,3,2,4,3	Input	Output	AR control signals:
ARREADY	1	Output	Input	Slave ARREADY signal. Indicates that the Slave is ready to receive AR channel
WDATA	DATA_WIDTH	Input	Output	Data to write
WID	ID_WIDTH	Input	Output	AXI3 only. ID value
WSTRB	DATA_WIDTH/8	Input	Output	Byte-lane of the data to write
WVALID	1	Input	Output	W valid
WLAST	1	Input	Output	Indicates last data of a burst
WREADY	1	Output	Input	Indicates that the Slave is ready to receive data
BRESP	2	Output	Input	Response from the Slave
BID	ID_WIDTH	Output	Input	ID value of the response
BVALID	1	Output	Input	W valid
BREADY	1	Input	Output	Master indicates if ready to get response
RDATA	DATA_WIDTH	Output	Input	Read data

TABLE 2 ZeBu xtor_amba_slave_svs Transactor Signal List

Symbol	Size in bits	Transactor Type	DUT Type	Description
RID	ID_WIDTH	Output	Input	ID value of the response
RRESP	2	Output	Input	Response from the Slave
RVALID	1	Output	Input	R valid
RLAST	1	Output	Input	Indicates last data of a burst
AWUSER,ARUSER,WUSER	USER_WIDTH	Input	Output	AXI4 only. User signal for AW, AR & W channel respectively.
BUSER, RUSER	USER_WIDTH	Output	Input	AXI4 only. User signal for B & R channel respectively
xtor_info_port	32	Output	NA	Transactor info port for slave ,this port is used for debugging

3.5.2 AXI Master/Slave Driver Parameters

The following table lists the parameters for the AXI Master/Slave Driver:

TABLE 3 ZeBu AMBA Master Transactor Driver Parameters

Driver Parameter	Allowed values	Scope	Description
DATA_WIDTH	32,64,128,256,512, 1024	All	Width of WDATA/RDATA
ADDR_WIDTH	32,64	All	Width of AWADDR/ARADDR
ID_WIDTH	4,8,16	All	Width of ARID,AWID,WID,BID,RID
WSTRB_WIDTH	4,8,16,32,64,128	All	Width of WSTRB

TABLE 3 ZeBu AMBA Master Transactor Driver Parameters

BLEN_WIDTH	4,8	All	Width of AWLEN/ARLEN
AXI_INTERLEAVE_DEPTH	1,2,4,8,16	AXI3 Master All Slave	Depth of Write Data Interleaving Depth of Read Data Interleaving
SNOOP_ADDR_WIDTH	32,64	ACE Master	Width of ACADDR
SNOOP_DATA_WIDTH	32,64,128,256,512,1024	ACE Master	Width of CDDATA
XTOR_DEBUG_WIDTH	32	All	Width of xtor_info_port

3.5.3 Connecting the signals to the Design

Here is a typical Verilog example showing how to connect an `AMBA Master` transactor to a `xtor_amba_slave_svs` transactor. Any of these transactors can be replaced by a DUT:

```
module hw_top();
// AXI Bus Signals
// Write Data Channel
wire   [DATA_WIDTH-1:0]      AXI_WDATA;
wire   [ID_WIDTH-1:0]        AXI_WID;
wire   [WSTRB_WIDTH-1:0]     AXI_WSTRB;
wire                         AXI_WLAST;
wire                         AXI_WVALID;
wire                         AXI_WREADY;

// Write Addr Channel
wire   [ADDR_WIDTH-1:0]      AXI_AWADDR;
wire   [ID_WIDTH-1:0]        AXI_AWID;
wire   [3:0]                 AXI_AWLEN;
wire  [2:0]                 AXI_AWSIZE;
wire  [2:0]                 AXI_AWPROT;
wire  [1:0]                 AXI_AWBURST;
wire  [1:0]                 AXI_AWLOCK;
wire  [3:0]                 AXI_AWCACHE;
wire                        AXI_AWVALID;
wire                        AXI_AWREADY;

// Write Resp channel
wire  [ID_WIDTH-1:0]        AXI_BID;
wire  [1:0]                 AXI_BRESP;
wire                        AXI_BVALID;
wire                        AXI_BREADY;

// Read Addr Channel
wire  [ADDR_WIDTH-1:0]      AXI_ARADDR;
wire  [ID_WIDTH-1:0]        AXI_ARID;
wire  [3:0]                 AXI_ARLEN;
wire  [2:0]                 AXI_ARSIZE;
wire  [2:0]                 AXI_ARPROT;
wire  [1:0]                 AXI_ARBURST;
wire  [1:0]                 AXI_ARLOCK;
wire  [3:0]                 AXI_ARCACHE;
wire                        AXI_ARVALID;
wire                        AXI_ARREADY;

// Read Resp Channel
wire  [DATA_WIDTH-1:0]      AXI_RDATA;
wire  [ID_WIDTH-1:0]        AXI_RID;
wire  [1:0]                 AXI_RRESP;
wire                        AXI_RLAST;
wire                        AXI_RVALID;
wire                        AXI_RREADY;
```

```
// Clock and reset handling
reg                         reset;
    wire                    clk_local;
    wire                    rst_local;
    wire                    resetcounter;
    reg [15:0]              resetcnt;

    zceiClockPort zceiClockPort_clk_i (
        .cclock(clk_local),
        .cresetn(rst_local)
      );

    assign resetcounter = ~ rst_local;
    always@ (posedge clk_local or posedge resetcounter)
    begin
        if(resetcounter) begin
            resetcnt <= 16'h0000;
            reset <= 1'b0;
        end
        else if (resetcnt[6] == 1'b1) begin
            reset <= 1'b1;
        end
        else
        begin
            resetcnt <= resetcnt + 1;
            reset <= 1'b0;
        end
    end

bit [XTOR_DEBUG_WIDTH_MASTER-1 :0] xtor_info_master;
bit [XTOR_DEBUG_WIDTH_SLAVE-1 :0]  xtor_info_slave;
xtor_amba_master_axi3_svs #(  .DATA_WIDTH (DATA_WIDTH),
                        .ADDR_WIDTH (ADDR_WIDTH),
                        .ID_WIDTH   (ID_WIDTH),
                        .WSTRB_WIDTH(WSTRB_WIDTH),
                        .BLEN_WIDTH (BLEN_WIDTH),
                        .AXI_INTERLEAVE_DEPTH(AXI_INTERLEAVE_DEPTH),
                       .XTOR_DEBUG_WIDTH_MASTER(XTOR_DEBUG_WIDTH_MASTER))

                                  master_node_i (
                                 .ACLK(clk_local),
                                 .ARESETn(reset),
                                 .WDATA(AXI_WDATA),
                                 .WSTRB(AXI_WSTRB),
                                 .WLAST(AXI_WLAST),
                                 .WVALID(AXI_WVALID),
                                 .WID(AXI_WID),
                                 .WREADY(AXI_WREADY),
                                 .AWADDR(AXI_AWADDR),
                                 .AWID(AXI_AWID),
                                 .AWLEN(AXI_AWLEN),
                                 .AWSIZE(AXI_AWSIZE),
                                 .AWPROT(AXI_AWPROT),
                                 .AWBURST(AXI_AWBURST),
                                 .AWLOCK(AXI_AWLOCK),
                                 .AWCACHE(AXI_AWCACHE),
                                 .AWVALID(AXI_AWVALID),
                                 .AWREADY(AXI_AWREADY),
                                 .BID(AXI_BID),
                                 .BRESP(AXI_BRESP),
                                 .BVALID(AXI_BVALID),
                                 .BREADY(AXI_BREADY),
                                 .ARADDR(AXI_ARADDR),
                                 .ARID(AXI_ARID),
                                 .ARLEN(AXI_ARLEN),
                                  .ARSIZE(AXI_ARSIZE),
                                  .ARPROT(AXI_ARPROT),
                                  .ARBURST(AXI_ARBURST),
                                  .ARLOCK(AXI_ARLOCK),
                                  .ARCACHE(AXI_ARCACHE),
                                  .ARVALID(AXI_ARVALID),
                                  .ARREADY(AXI_ARREADY),
                                  .RDATA(AXI_RDATA),
                                  .RID(AXI_RID),
                                  .RRESP(AXI_RRESP),
                                  .RLAST(AXI_RLAST),
                                  .RVALID(AXI_RVALID),
                                  .RREADY(AXI_RREADY),
                                  .xtor_info_port(xtor_info_master)
                              );

xtor_amba_slave_axi3_svs  #(  .DATA_WIDTH (DATA_WIDTH),
                        .ADDR_WIDTH (ADDR_WIDTH),
                        .ID_WIDTH   (ID_WIDTH),
                        .WSTRB_WIDTH(WSTRB_WIDTH),
                        //.BLEN_WIDTH (BLEN_WIDTH),
                        .AXI_INTERLEAVE_DEPTH(AXI_INTERLEAVE_DEPTH),
                        .XTOR_DEBUG_WIDTH_SLAVE(XTOR_DEBUG_WIDTH_SLAVE))

                                   slave_node_i (
                                  .ACLK(clk_local),
                                  .ARESETn(reset),
                                  .WDATA(AXI_WDATA),
                                  .WSTRB(AXI_WSTRB),
                                  .WLAST(AXI_WLAST),
                                  .WVALID(AXI_WVALID),
                                  .WID(AXI_WID),
                                  .WREADY(AXI_WREADY),
                                  .AWADDR(AXI_AWADDR),
                                  .AWID(AXI_AWID),
                                  .AWLEN(AXI_AWLEN),
                                  .AWSIZE(AXI_AWSIZE),
                                  .AWPROT(AXI_AWPROT),
                                  .AWBURST(AXI_AWBURST),
                                  .AWLOCK(AXI_AWLOCK),
                                  .AWCACHE(AXI_AWCACHE),
                                  .AWVALID(AXI_AWVALID),
                                  .AWREADY(AXI_AWREADY),
                                  .BID(AXI_BID),
                                  .BRESP(AXI_BRESP),
                                  .BVALID(AXI_BVALID),
                                  .BREADY(AXI_BREADY),
                                  .ARADDR(AXI_ARADDR),
                                  .ARID(AXI_ARID),
                                  .ARLEN(AXI_ARLEN),
                                  .ARSIZE(AXI_ARSIZE),
                                  .ARPROT(AXI_ARPROT),
                                  .ARBURST(AXI_ARBURST),
                                  .ARLOCK(AXI_ARLOCK),
                                  .ARCACHE(AXI_ARCACHE),
                                  .ARVALID(AXI_ARVALID),
                                  .ARREADY(AXI_ARREADY),
                                  .RDATA(AXI_RDATA),
                                  .RID(AXI_RID),
                                  .RRESP(AXI_RRESP),
                                  .RLAST(AXI_RLAST),
                                  .RVALID(AXI_RVALID),
                                  .RREADY(AXI_RREADY),
                                  .xtor_info_port(xtor_info_slave)
                              );
```

```
endmodule
```

4 Software Interface

The ZeBu AMBA transactor provides C++ API’s. They are included among the header files located in $ZEBU_IP_ROOT/include.

The following table describes the header files for the ZeBu AMBA API:

TABLE 4 Header Files for ZeBu AMBA transactors

Header Files	**Description**
xtor_amba_defines_svs.hh	Contains C++ enumerations, structs and defines used in AMBA transactors.
xtor_amba_master_svs.hh	Contains C++ API’s that are used for xtor_amba_master_svs_svs.
xtor_amba_slave_svs.hh	Contains C++ API’s that are used for xtor_amba_slave_svs.

This section explains the following topics:

- *ZeBu AMBA Master Classes*
- *ZeBu AMBA Slave Classes*
- *ZeBu AMBA AXIData Class*
- *ZeBu AMBA AXIResp Class*
- *ZeBu AMBA xtor_ace_cache Class*
- *ZeBu AMBA xtor_ace_cache_line Class*
- *Enumerations and Structures*
- *Transactor Connection and Initialization*
- *Creating AMBA Bus Transactions*

4.1 ZeBu AMBA Master Classes

You can configure and transmit/receive on the AMBA bus using the following methods. The class objects are contained in the XTOR_AMBA_DEFINES_SVS namespace that is automatically invoked through the xtor_amba_master_svs.hh and xtor_amba_slave_svs.hh file. Therefore, there is no need to declare this namespace in your testbench.

The following table lists the APIs for AMBA Master:

TABLE 5 AMBA Master API List

Function	Description
~xtor_amba_master_svs ()	Destructor of the transactor.
static xtor_amba_master_svs* getInstance(const char* hdl_path, svt_c_runtime_cfg* runtime)	Get the AMBA Master transactor instance for transactor located at the specified path in the hardware platform. It is created using the specified runtime configuration. **Parameters:** hdl_path: Full HDL path of transactor HW instance runtime: Runtime environment descriptor **Returns**: Instance of the amba master transactor
bool setParam (XtorParam param, uint32_t val)	Set transactor parameters to a specified value **Parameters**: param Name of parameter to be set. val value that is to be assigned to above parameter. **Returns** true: successful, false: failure
bool getParam (XtorParam param, uint32_t &val)	get value of transactor parameter **Parameters**: param: name of parameter whose value is required. val: reference to uint32_t that will be updated with required value **Returns** true: successful, false: failure
bool runUntilReset ()	Blocking call to wait for transactor to come out of reset. **Returns** true: successful false: failure

TABLE 5 AMBA Master API List

Function	Description
bool runClk (uint32_t num_clock)	Run controlled clock cycles for defined numCycles. **Parameters**: *numCycles:* Number of Xtor clock cycle without activity allowed. **Returns** true if ok false if error
bool setDebugLevel (uint8_t level)	Set the verbosity of the display messages to be displayed runtime. **Parameters**: level: Verbosity of the messages 4 - Full 3 - High 2 - Medium 1 - Low 0 - NONE Returns true:successful false: failure
bool setDebugFile (FILE *file_name, int file_control)	Controls the log files to be used for dumping debug messages. **Parameters**: file_name: file pointer file_control: Control of File 0 - Create a new file 1 - Append to file 2 - Close the file **Returns** true: successful false: failure
void enableTxnDump (bool enable)	Enable transaction dumping on terminal on successful Txn completion. **Parameters:** enable: 1: enable dumping 0: disable dumping

TABLE 5 AMBA Master API List

Function	Description
void registerBrespCB(void(*BrespCB)(void *ctxt, AXIResp &bresp), void* context = NULL)	Callback for reception of Write Response. **Parameters**: ● **ctxt**: pointer to the transactor class object, if the context argument is not set. Else, set to context specified using the context parameter. ● **bresp**: reference to AXIResp class that contains BID, BRESP & BUSER ● **context**: Reference to context that you can set (optional). Will be returned in the ctxt if set.
void registerRrespCB(void(*RrespCB)(void *ctxt, const AXIData& data, AXIResp &rresp), void* context = NULL)	Callback for reception of Read Response. **Parameters**: ● **ctxt**: Pointer to xtor class object if "context" argument is not set. Else, set to "context" provided. ● **data**: reference to AXIData class object holding data received for Read. ● **rresp**: reference to AXIResp class that contains RID, RRESP, and RUSER. ● **context**: Reference to context that you can set (optional). Will be returned in the ctxt as is if set
void registerACreqCB(void(*ACreqCB)(void *ctxt, uint64_t acaddr, int snoop, int prot), void* context = NULL)	Callback for reception of Snoop request. **Parameters**: ● **ctxt**: Pointer to transactor class object if context argument is not set. Else, set to "context" provided ● **acaddr**: ACADDR received with snoop request. ● **snoop**: ACSNOOP received with snoop request. ● **prot**: ACPROT received with snoop request. ● **context**: Reference to context that you can set (optional). Will be returned in the ctxt if set.
void AutoSnoopResponse ()	Handles snoop response, can be used only when USE_C_CACHE parameter is set high.

TABLE 5 AMBA Master API List

Function	Description
int wrTxnNB (AXIChanInfo addr, AXIData *data)	API to send write transaction in non-blocking manner. **Parameters**: addr: Address struct for write transaction, refer to xtor_amba_defines_svs.hh for struct definition. data: AXIData class pointer holding data to be sent for Write transaction. **Returns:** ● 0, if transaction was sent ● non-zero integer, if transaction was not sent due to some reason. See warning printed on stdout.
int rdTxnNB (AXIChanInfo addr)	API to send read transaction in non-blocking manner,response is received by callback. **Parameters**: **addr**: Address struct for read transaction. See xtor_amba_defines_svs.hh for struct definition. **Returns:** ● 0, if transaction was sent ● non-zero integer, if transaction was not sent due to some reason. See warning printed on stdout.
int wrTxn (AXIChanInfo addr, AXIData *data, AXIResp *bresp)	API to send write transaction in a blocking manner. **Parameters:** ● **addr**: Address struct for write transaction. See xtor_amba_defines_svs.hh for struct definition. ● **data**: AXIData class pointer holding data to be sent for Write transaction. ● **bresp**: response received. See xtor_amba_defines_svs.hh for class definition. **Returns:** ● 0, if transaction was sent ● non-zero integer, if transaction was not sent due to some reason. See warning printed on stdout.

TABLE 5 AMBA Master API List

Function	Description
int rdTxn (AXIChanInfo addr, AXIData *rdata, AXIResp *rresp)	API to send read transaction in blocking manner. **Parameters**: *addr*: Address struct for read transaction, refer to xtor_amba_defines_svs.hh for struct definition *data:* data received, AXIData class pointer holding data received for Write transaction *resp:* response received, refer to xtor_amba_defines_svs.hh for enum definition. **Returns:** ● 0, if transaction was sent ● non-zero integer, if transaction was not sent due to some reason. See warning printed on stdout.
int sendWrAddr (AXIChanInfo addr)	Phase API to send write address transaction in non-blocking manner, response is received by callback. **Parameters**: addr: Address struct for write transaction, refer to xtor_amba_defines_svs.hh for struct definition. **Returns:** ● 0, if transaction was sent ● non-zero integer, if transaction was not sent due to some reason. See warning printed on stdout.
int sendWrData (AXIData *data)	API to send write data transaction in non-blocking manner,response is received by callback. **Parameters**: *data:* AXIData class pointer holding data,burst_length & id to be sent for Write transaction. **Returns:** ● 0, if transaction was sent ● non-zero integer, if transaction was not sent due to some reason. See warning printed on stdout.

TABLE 5 AMBA Master API List

Function	Description
int sendRdAddr (AXIChanInfo addr)	API to send read transaction in non-blocking manner, response is received by callback. **Parameters**: *addr:* Address struct for read transaction, refer to xtor_amba_defines_svs.hh for struct definition. **Returns:** ● 0, if transaction was sent ● non-zero integer, if transaction was not sent due to some reason. See warning printed on stdout.
void driveRready (bool value)	API to drive ready signal for Read Response Channel. **Parameters**: value true: drive RREADY to high, 0: drive RREADY to low.
void driveBready (bool value)	API to drive ready signal for Write Response Channel. **Parameters**: value true: drive BREADY to high, 0: drive BREADY to low
int sendSnpResp (ACESnpResp snoopresp)	API to drive CRRESP (snoop response). **Parameters** snoopresp structure containing values to drive CRRESP. **Returns:** ● 0, if transaction was sent ● non-zero integer, if transaction was not sent due to some reason. See warning printed on stdout.
int sendSnpData (AXIData *snoopdata)	API to drive CDDATA (snoop data). **Parameters:** value true: drive BREADY to high, 0: drive BREADY to low. **Returns:** ● 0, if transaction was sent ● non-zero integer, if transaction was not sent due to some reason. See warning printed on stdout.

4.2 ZeBu AMBA Slave Classes

The following table lists the APIs for AMBA Slave:

TABLE 6 AMBA Slave API List

Function	Description
~xtor_amba_slave_svs ()	Destructor of the transactor.
static xtor_amba_slave_svs* getInstance(const char* hdl_path, svt_c_runtime_cfg* runtime)	Get the xtor_amba_slave_svs transactor instance for transactor located at the specified path in the hardware platform. It is created using the specified runtime configuration. **Parameters:** hdl_path: Full HDL path of transactor HW instance runtime: Runtime environment descriptor **Returns**: Instance of the amba slave transactor
bool setParam (XtorParam param, uint32_t val)	Set transactor parameters to a specified value **Parameters**: *param*: Name of parameter to be set. val value that is to be assigned to above parameter. **Returns** true: successful, false: failure
bool getParam (XtorParam param, uint32_t &val)	get value of transactor parameter **Parameters**: *param*: name of parameter whose value is required. *val*: reference to uint32_t that will be updated with required value **Returns** true: successful, false: failure
bool runUntilReset ()	Blocking call to wait for transactor to come out of reset. **Returns** true: successful false: failure

TABLE 6 AMBA Slave API List

Function	**Description**
bool runClk (uint32_t num_clock)	Run controlled clock cycles for defined numCycles. **Parameters**: *numCycles:* Number of Xtor clock cycle without activity allowed. **Returns** true if ok false if error
bool setDebugLevel (uint8_t level)	Set the verbosity of the display messages to be displayed runtime. **Parameters**: level: Verbosity of the messages 4 - Full 3 - High 2 - Medium 1 - Low 0 - NONE Returns true:successful false: failure
bool setDebugFile (FILE *file_name, int file_control)	Controls the log files to be used for dumping debug messages. **Parameters**: file_name: file pointer file_control: Control of File 0 - Create a new file 1 - Append to file 2 - Close the file **Returns** true: successful false: failure
void registerReadAddrCB(void(*ReadAddrCB)(void* ctxt,AXIChanInfo addr), void* context = NULL)	Callback for reception of Read request. **Parameters:** ● **ctxt**: Pointer to xtor class object if "context" argument is not set. Otherwise set to "context" provided. ● **addr**: channel info received. See xtor_amba_defines_svs.hh for struct definition. ● **context**: Reference to context that user can set (optional). Will be returned in the ctxt as is if set.

TABLE 6 AMBA Slave API List

Function	Description
void registerWriteTxnCB(void(*WriteTxnCB)(void* ctxt,AXIChanInfo addr, AXIData* data), void* context = NULL)	Callback for reception of Write Data request. **Parameters**: ● **ctxt**: Pointer to xtor class object if context argument is not set. Otherwise set to context provided. ● **addr**: Channel info received. See xtor_amba_defines_svs.hh for enum definition. ● **data**: AXIData holding data received for Write transaction. ● **context**: Reference to context that you can set (optional). Will be returned in the ctxt as is if set
void registerRstAssertCB(void(*rstAssertCB)(void *ctxt), void* context = NULL)	Callback for reception of reset assertion Parameters: ● **ctxt**: pointer to xtor class object if context argument is not set. Otherwise set to context provided. ● **context**: Reference to context that you can set (optional). The value of this argument is returned in ctxt, if set.
void registerRstDeAssertCB(void(*rstDeAssertCB)(void *ctxt), void* context = NULL)	Callback for reception of reset deassertion. ● **ctxt**: Pointer to xtor class object, if "context" argument is not set. Else, use the value specified using the context argument. ● **context**: Reference to context that user can set (optional). The value of this argument is returned in ctxt, if set.
bool sendWriteResponse ()	API to send write response, sends the responses that were recorded in WriteTxnCB, Responses are sent in First in First out manner. **Returns** True: if response was sent False: if no response was stored in queue or hardware FIFO is full
bool sendReadResponse ()	API to send Read response, sends the responses that were recorded in ReadReqCB, Responses will be sent in First in First out manner. **Returns** True: if response was sent False: if no response was stored in queue or hardware FIFO is full

TABLE 6 AMBA Slave API List

Function	Description
bool sendResponse ()	API to send Read & Write responses, sends the responses that were recorded in ReadReqCB/WriteTxnCB, Responses will be sent in First in First out manner. Needs to be called outside of ReadReqCB/WriteTxnCB, preferably in a separate thread. **Returns** True: if response was sent False: if no response was stored in queue or hardware FIFO is full.
driveARready (bool value)	API to drive ready signal for Read Address Channel. **Parameters:** value: true: drive ARREADY to high, 0: drive ARREADY to low
driveAWready (bool value)	API to drive ready signal for Write Address Channel. **Parameters:** value: true: drive AWREADY to high 0: drive AWREADY to low
driveWready (bool value)	API to drive ready signal for Write Data Channel. **Parameters:** value: true: drive WREADY to high 0: drive WREADY to low
printStats ()	API to print statistics of transactions.

TABLE 6 AMBA Slave API List

Function	Description
storeWriteResponse (AXIResp *B_resp)	API to store Write responses in the software FIFO, to be sent later by calling sendWriteResponse API. Responses are sent in First in First out manner. **Parameters**: B_resp: write response to store **Returns** true if successful
storeReadResponse (AXIResp *R_resp, AXIData *data)	API to store Read responses in software FIFO, to be sent later by calling sendReadResponse API. Responses are sent in First in First out manner. **Parameters**: R_resp:read response to store data: axidata to store **Returns** true if successful

4.3 ZeBu AMBA AXIData Class

The following table lists the APIs for the AXIData class:

TABLE 7 AXIData Class API List

Function	Description
uint32_t GetLengthByte () const	Function returning the data size in byte. **Returns** size in bytes
void SetId (uint32_t id)	Function Set the id. **Parameters**: id value of id to be set.
uint32_t GetId ()	Function returning the id. **Returns** size in bytes
void SetBurstLength (uint32_t len)	Function Set the BurstLength.
uint32_t GetBurstLength ()	Function Set the BurstLength. **Parameters**: len value of BurstLength to be set
uint32_t GetLengthWord ()	Function returning the data size in 32bits word. **Returns** size in words
bool FillByte (const uint8_t byte, const axiBE_t be=AXI_BYTE_ENABLED)	Function filling AXIData with dataBlock. **Parameters**: byte the data to fill in byte be byte enable value, default is AXI_BYTE_ENABLED **Returns** true if ok false if axiData maximum size is reached

TABLE 7 AXIData Class API List

Function	**Description**
bool FillByteArray (const uint8_t *dataBlock, uint32_t numBytes, const axiBE_t *be=0)	Function filling AXIData with dataBlock. **Parameters**: dataBlock: the data to fill in byte numBytes: Number of bytes to fill be array representing byte enable for each byte of dataBlock **Returns** true if ok false if axiData maximum size is reached
bool FillByteAtIndex (const uint8_t byte, uint32_t index, const axiBE_t be=AXI_BYTE_ENABLED)	Function filling AXIData with dataBlock. **Parameters** byte: the data to fill in byte index: index of the data byte be: array giving byte enable values **Returns** true if ok, false if axiData maximum size is reached
bool FillWord32 (const uint32_t word, const axiBE_t *be=0)	Function filling AXIData with dataBlock with a 32 bits word. **Parameters** word: the data to fill be: array representing byte enable for each byte of the 32bits word Returns true if ok, false if axiData maximum size is reached
bool FillWord64 (const uint64_t word, const axiBE_t *be=0)	Function filling AXIData with dataBlock with 64 bits word. **Parameters** word: the data to fill be: array representing byte enable for each byte of the 64bits word **Returns** true if ok, false if axiData maximum size is reached

TABLE 7 AXIData Class API List

Function	Description
bool FillWord32Array (const uint32_t *dataBlock, uint32_t numWord, const axiBE_t *be=0)	Function filling AXIData with dataBlock. **Parameters** dataBlock: the data to fill in word numWord: number of 32bits word to fill from the array be: array representing byte enable for each byte of dataBlock **Returns** true if ok, false if axiData maximum size is reached
bool FillWord64Array (const uint64_t *dataBlock, uint32_t numWord, const axiBE_t *be=0)	Function filling AXIData with dataBlock. **Parameters** dataBlock: the data to fill in word numWord number of 64bits word to fill from the array be: array representing byte enable for each byte of dataBlock **Returns** true if ok, false if axiData maximum size is reached
bool FillWord32AtIndex (const uint32_t dataWord, uint32_t index, const axiBE_t *be=0)	Function filling AXIData with dataBlock. **Parameters** dataWord: the data to fill in byte index: index of the 32bits data word be: array representing byte enable for each byte of the 32bits word **Returns** true if ok, false if axiData maximum size is reached.

TABLE 7 AXIData Class API List

Function	Description
bool FillWord64AtIndex (const uint64_t dataWord, uint32_t index, const axiBE_t *be=0)	Function filling AXIData with dataBlock. **Parameters** dataWord: the data to fill in byte index: index of the 64bits data word be: array representing byte enable for each byte of the 64bits word **Returns** true if ok, false if axiData maximum size is reached
uint32_t GetBeat (uint32_t sizeBytes) const	Function returning the number of AXI bus beat needed to transfer this axi data if data is placed at an AXI address boundary. **Parameters**: sizeBytes: Number of databytes per chunk **Returns** Number of AXI burst beat
uint32_t GetBeatFromAddr (axi_addr_t addr, uint32_t sizeBytes) const	Function returning the number of AXI bus beat needed to transfer this axi data if data is placed at the given address. **Parameters**: addr: address from which data is written/read sizeBytes: Number of databytes per chunk Returns:Number of AXI burst beat
uint8_t IndexByte (uint32_t index) const	Function returning the byte at the specified index. **Parameters**: index: index on the data block **Returns** byte pointer on the data block

TABLE 7 AXIData Class API List

Function	Description
uint8_t * GetDataPt (int32_t index=-1) const	Function returning a pointer to the data at specified index. **Parameters**: index: index on the data block, if -1, return: pointer at current index **Returns** byte pointer on the data block
uint32_t IndexWord32 (uint32_t index) const	Function returning the value at the specified index. **Parameters**: index: index of the 32bits data word **Returns** word pointer on the data block
uint64_t IndexWord64 (uint32_t index) const	Function returning the value at the specified index. **Parameters:** index: index of the 64bits data word **Returns** word pointer on the data block
axiBE_t IndexByteValue (uint32_t index, uint8_t &busvalue) const	Function returning the byte at the specified index. **Parameters** index: index on the data block busvalue: the byte value at the specified index **Returns** byte enable value at the specified index

TABLE 7 AXIData Class API List

Function	Description
axiBE_t * IndexWordValue32 (uint32_t index, uint32_t &busvalue) const	Function returning the value at the specified index. **Parameters**: index: index of the 32bits data word busvalue: the 32bits value at the specified index **Returns** pointer to a byte enable array representing byte enables for the 32bits word
axiBE_t * IndexWordValue64 (uint32_t index, uint64_t &busvalue) const	Function returning the value at the specified index.
axiBE_t ByteEnableIndex (uint32_t index) const	Function returning the value of the byte enable at the specified index.
const axiBE_t * ByteEnableArray () const	Function returning a pointer to the byte enable array.
void Clear ()	Function clearing the AXI data content.
void Dump (uint32_t sizeBytes=4) const	Function printing data.
void DumpAtAddr (axi_addr_t addr, uint32_t sizeBytes=4) const	Function printing data if placed at the specified address.
bool FillByteUser (const uint8_t byte)	Function filling AXIData with dataBlockUser.
uint8_t IndexByteUser (uint32_t index) const	Function returning the USER byte at the specified index.
uint32_t GetNumBytesUser	Function returning number of bytes in USER data.
static void DumpDataBlockByte (const uint8_t *dataBlock, const uint32_t size)	Function printing data as bytes.

TABLE 7 AXIData Class API List

Function	Description
static void DumpDataBlockWord (const uint32_t *dataBlock, const uint32_t size)	Function printing data as bytes.
static void DumpDataBlockByteIndex (const uint8_t *dataBlock, const uint32_t index)	Function printing data if placed at the specified address.
static void DumpDataBlockWordIndex (const uint32_t *dataBlock, const uint32_t index)	Function printing data if placed at the specified address.

4.4 ZeBu AMBA AXIResp Class

The following table lists the APIs for the AXIResp class:

TABLE 8 AXIResp class

Function	Description
void SetId (axi_id_t id)	Function used to set RID. **Parameters:** id: ID to set
axi_id_t GetId ()	Function returning RID. **Returns**:RID
void SetACEResp (bool isshared, bool passdirty)	Function used to set ACEResp. **Parameters:** isshared ACEResp IsShared to set. passdirty ACEResp PassDirty to set.
void GetACEResp (bool &isshared, bool &passdirty)	Function to get ACEResp. **Parameters:** isshared: bool pointer to get ACEResp IsShared passdirty: bool pointer to get ACEResp PassDirty **Returns**:void
bool AddResp (enAXIResp resp)	Function adding an element in the array of AXI RRESP. **Parameters:** resp: the AXI RRESP to add **Returns**:true if ok, else false
bool FillByteUser (const uint8_t byte)	Function filling AXIData with dataBlockUser. **Parameters**: **byte**: the data to fill in byte Returns:true if ok, false if maximum size is reached.
uint32_t GetNumBytesUser () const	Function returning the number of USER data bytes in the array. **Returns**:number of element

TABLE 8 AXIResp class

Function	Description
uint32_t GetNumResp () const	Function returning the number of RRESP element in the array. **Returns**:number of element
enAXIResp * GetResp () const	Function returning the array of AXI RRESP field (one per beat of DATA). **Returns**:Pointer to the AXI RResp field
uint8_t IndexByteUser (uint32_t index)	Function returning the USER byte at the specified index. **Parameters**: index: index on the data block **Returns**:byte pointer on the data block
void SetDuration (uint32_t duration)	Function used to set the duration of the transaction.
uint32_t GetDuration () const	Function used to set the duration of the transaction. **Parameters:** duration: Duration of the transaction in number of clock cycles
void Dump () const	Function dumping AXIResp information.

4.5 ZeBu AMBA xtor_ace_cache Class

The following table lists the APIs for the xtor_ace_cache class:

TABLE 9 xtor_ace_cache class

Function	Desc
bool read_by_index (uint32_t index, axi_addr_t &addr, std::deque< uint32_t > &data, bool &is_unique, bool &is_clean, uint64_t &age)	Sparse array of xtor_ace_cache_line objects, used to model the physical storage of this cache. The array is to be indexed only by an expression that evaluates to an address within the bounds of the maximum number of cache lines in the cache. Returns the contents of the cache line at the given index. If the index does not exist this function returns 0, else it returns 1. **Parameters**: ● **index**: The index of the cache line to be read. ● **addr**: Assigned with the address stored in the given index. Please note that if the address specified is unaligned to cache line size, an aligned address is computed internally, before doing the operation. ● **data**: Assigned with the data stored in the given index. ● **is_unique**: Returns 1 if the cache line at the given index is not shared with any other cache, else returns 0. ● **is_clean**: Returns 1 if the cache line at the given index is updated relative to main memory, else returns 0. **Returns**: If the read is successful, that is, if the given index has an entry in the cache 1 is returned, else returns 0.
bool read_line_by_index (uint32_t index, axi_addr_t &addr, std::deque< uint32_t > &data, std::deque< bool > &dirty_byte_flag, bool &is_unique, bool &is_clean, uint64_t &age)	Returns the contents of the cache line at the given index. If the index does not exist this function returns 0, else it returns 1. **Parameters**: ● index: The index of the cache line to be read. ● addr: Assigned with the address stored in the given index. Please note that if the address specified is unaligned to cache line size, an aligned address is computed internally, before doing the operation. ● data: Assigned with the data stored in the given index. ● dirty_byte_flag: Returns dirty flag associated with each byte in the cache line associated with the given address. '1' indicates data byte is dirty '0' indicates data byte id clean ● is_unique: Returns 1 if the cache line at the given index is not shared with any other cache, else returns 0. ● is_clean: Returns 1 if the cache line at the given index is updated relative to main memory, else returns 0. **Returns**: If the read is successful, that is, if the given index has an entry in the cache 1 is returned, else returns 0.

TABLE 9 xtor_ace_cache class

Function	Desc
bool read_by_addr (axi_addr_t addr, uint32_t &index, std::deque< uint32_t > &data, bool &is_unique, bool &is_clean, uint64_t &age)	Returns the contents of the cache line at the given address. If there is no entry corresponding to this address returns 0, else it returns 1. **Parameters:** addr: The address associated with the cache line that needs to be read. Please note that if the address specified is unaligned to cache line size, an aligned address is computed internally, before doing the operation. index: Assigned with the index of the cache line associated with the given address. data: Assigned with the data stored in the cache line associated with the given address. is_unique: Returns 1 if the cache line for given address is not shared with any other cache, else returns 0. is_clean: Returns 1 if the cache line for the given address is updated relative to main memory, else returns 0. **Returns**: If the read is successful, that is, if the given address has an associated entry in the cache 1 is returned, else returns 0.
bool read_only_data (axi_addr_t addr, std::deque< uint32_t > &data_o)	Returns only the data(not status) of the cache line at the given address. If there is no entry corresponding to this address returns 0, else it returns 1. **Parameters**: addr: The address associated with the cache line that needs to be read. Please note that if the address specified is unaligned to cache line size, an aligned address is computed internally, before doing the operation. data: Assigned with the data stored in the cache line associated with the given address. **Returns**: If the read is successful, that is, if the given address has an associated entry in the cache 1 is returned, else returns 0.

TABLE 9 xtor_ace_cache class

Function	Desc
bool read_line_by_addr (axi_addr_t addr, uint32_t &index, std::deque< uint32_t > &data, std::deque< bool > &dirty_byte_flag, bool &is_unique, bool &is_clean, uint64_t &age)	Returns the contents of the cache line at the given address. If there is no entry corresponding to this address returns 0, else it returns 1. **Parameters**: addr: The address associated with the cache line that needs to be read. Please note that if the address specified is unaligned to cache line size, an aligned address is computed internally, before doing the operation. index: Assigned with the index of the cache line associated with the given address. data: Assigned with the data stored in the cache line associated with the given address. dirty_byte_flag: Returns dirty flag associated with each byte in the cache line associated with the given address. '1' indicates data byte is dirty '0' indicates data byte id clean is_unique: Returns 1 if the cache line for given address is not shared with any other cache, else returns 0. is_clean: Returns 1 if the cache line for the given address is updated relative to main memory, else returns 0. **Returns**: If the read is successful, that is, if the given address has an associated entry in the cache 1 is returned, else returns 0.

TABLE 9 xtor_ace_cache class

Function	Desc
bool write (uint32_t index, axi_addr_t addr, std::deque< uint32_t > data, std::deque< bool > byteen, int8_t is_unique, int8_t is_clean, bool age)	Writes cache line information into a cache line. If the index is a positive value, the addr, data and status information is written in the particular index. Any existing information will be overwritten. If the value of index is passed as -1, the data will be written into any available index based on the cache structure. If there is no available index, data is not written and a failed status (-1) is returned. **Parameters**: index: The index to which the cache line information needs to be written. The range of index is between 0 to (svt_axi_port_configuration::num_cache_lines - 1). addr: The main memory address to which the cache line is to be associated. Please note that if the address specified is unaligned to cache line size, an aligned address is computed internally, before doing the operation. data: The data to be written into this cache line. byteen (Optional): The byte-enables to be applied to this write. A 1 in a given bit position enables the byte in the data corresponding to that bit position. is_unique (Optional): The shared status to be stored in the cache line. If not passed, the status of the line is not changed/ updated. is_clean (Optional): The clean status to be stored in the cache line. If not passed, the status of the line is not changed/ updated. **Returns**: 1 if the write was successful, or 0 if it was not successful.
int32_t get_index_for_addr (axi_addr_t addr)	Gets the index of a cache line corresponding to the given address If there is no corresponding entry returns -1. **Parameters:** addr: The address whose index is required. **Returns**: Returns the index corresponding to the given address. If there is no such entry, returns -1.

TABLE 9 xtor_ace_cache class

Function	Desc
bool get_addr_at_index (uint32_t index, axi_addr_t &addr)	Gets the address stored in the cache line at given index. If there is no corresponding entry returns -1. **Parameters**: index: The index of the cache line addr: Assigned with the address stored at given index. **Returns**: Returns 1 if there is a cache line at given index, else returns 0.
bool invalidate_addr (axi_addr_t addr)	Invalidates the cache line entry for a given addr. This method removes the cache line from the cache for the given address. **Parameters**: addr: The address that needs to be invalidated. **Returns**: Returns 1 if the operation is successful, that is, if there is a corresponding entry. Else, returns 0.
bool invalidate_index (uint32_t index)	Invalidates the cache line entry at the given index. This method removes the cache line from the cache for the given index. **Parameters:** index The index that needs to be invalidated. **Returns**: Returns 1 if the operation is successful, that is, if there is a corresponding entry. Else, returns 0.
bool update_status (axi_addr_t addr, int8_t is_unique, int8_t is_clean)	Updates the status of the cache line for the given address **Parameters**: addr: The address that needs to be updated. is_unique: The shared status to which the cache line associated with the addres needs to be updated. A value of 1 indicates that the corresponding cache line is not shared with other caches A value of 0 indicates that the corresponding cache line is shared with other caches A value of -1 indicates that the current status need not be changed. is_clean: The clean status to which the cache line associated with the addres needs to be updated. A value of 1 indicates that the corresponding cache line is updated relative to main memory. A value of 0 indicates that the corresponding cache line is not updated with respect to main memory. A value of -1 indicates that the current status need not be changed. **Returns**: Returns 1 if the operation is successful, that is, there is a corresponding entry in cache, otherwise returns 0.

TABLE 9 xtor_ace_cache class

Function	Desc
bool get_status (axi_addr_t addr, bool &is_unique, bool &is_clean)	Gets the status of the cache line for the given address Parameters: addr: The address whose status is required. is_unique: The shared status of the cache line associated with the address. A value of 1 indicates that the corresponding cache line is not shared with other caches A value of 0 indicates that the corresponding cache line is shared with other caches is_clean: The clean status of the cache line associated with the address. A value of 1 indicates that the corresponding cache line is updated relative to main memory. A value of 0 indicates that the corresponding cache line is not updated with respect to main memory. **Returns**: Returns 1 if the operation is successful, that is, there is a corresponding entry in cache, otherwise returns 0.

TABLE 9 xtor_ace_cache class

Function	Desc
int32_t get_any_index (uint32_t low_index, uint32_t high_index)	This method allows the user to get an index in the cache which may either be allocated for a cache line, or not allocated. If user specifies the values of arguments is_unique and is_clean as -1, then an index is returned where no cache line has been allocated. If user specifies the values of arguments is_unique and is_clean as 0 or 1, index is returned where a cache line is allocated, and the cache line state is as specified. The returned index is between values specified by low_index and high_index. **Parameters**: is_unique The shared status of the cache line which needs to be retreived. A value of 1 indicates that the cache line must not be shared with other caches. A value of 0 indicates that the cache line could be shared. A value of -1 indicates that the shared status of the cache line could be in any state. is_clean The clean status of the cache line that needs to be retreived. A value of 1 indicates that the cache line must be updated relative to main memory. A value of 0 indicates that the cache line must not be updated with respect to main memory. A value of -1 indicates that the clean status of the cache line could be in any state. low_index The first index at which the operation is to be done. The default value is 0. high_index The last index at which the operation is to be done. The default value is (svt_axi_port_configuration::num_cache_lines - 1). **Returns**: Returns a positive integer if the operation is successful, that is, a cache line satisfying the requirements is found, otherwise returns -1.
int32_t get_least_recently_used (int32_t low_index, int32_t high_index)	Get least recently used cache line. **Parameters**: low_index lower index bound high_index higher index bound **Returns**: index of least recently used cache

TABLE 9 xtor_ace_cache class

Function	Desc
bool update_age (axi_addr_t addr, uint64_t age)	Update age of cache line at specified address **Parameters**: addr: address of cache line age: age to be updated **Returns**: true:successfull false:cache line does not exist
std::string get_cmd (bool txn, uint32_t snoop, uint32_t domain, uint32_t bar)	Get name of ACE txn name based on address attributes **Parameters**: txn true:read false:write snoop AxSNOOP attribute domain AxDOMAIN attribute bar AXBAR attribute **Returns**: string of ACE txn name
void invalidate_all ()	Clears the contents of the cache.
void print_all_cache ()	Returns the contents of the cache line at the given address. If there is no entry corresponding to this address returns 0, else it returns 1. **Parameters:** addr: The address associated with the cache line that needs to be read. Please note that if the address specified is unaligned to cache line size, an aligned address is computed internally, before doing the operation. index: Assigned with the index of the cache line associated with the given address. data: Assigned with the data stored in the cache line associated with the given address. is_unique: Returns 1 if the cache line for given address is not shared with any other cache, else returns 0. is_clean: Returns 1 if the cache line for the given address is updated relative to main memory, else returns 0. **Returns**: If the read is successful, that is, if the given address has an associated entry in the cache 1 is returned, else returns 0.

TABLE 9 xtor_ace_cache class

Function	Desc
xtor_ace_cache_line get_cache_line (axi_addr_t addr)	Get the cacheline stored in the specified address. **Parameters**: addr: cache line address **Returns**: Return the object of single cache line, if there is no such cacheline a null is returned
axi_addr_t get_aligned_addr (axi_addr_t addr)	Get the aligned address computed from input address. **Parameters**: addr: cache line address **Returns**: Return the aligned address
void set_log_base_2 (uint32_t cls)	Set the log base 2 value of input argument. **Parameters:** **cls**: input whose log base 2 is to be computed.
bool cache_lines_exists (axi_addr_t addr)	Check if given address has cache line allocated. **Parameters**: addr: cache line address **Returns**: true:cache line exits false: cache line not present
uint32_t cache_line_index (axi_addr_t addr)	Get index of cache line of specified address. **Parameters:** addr: cache line address **Returns**: index of cache line
bool addr_mapped_to_indices_exists (uint32_t index)	Check if specified index exists. **Parameters:** addr: cache line index **Returns:** true:cache line exits false: cache line not present
uint32_t addr_mapped_to_indices_next (uint32_t index)	Get next valid index of cache line . **Parameters**: addr: cache line address **Returns**: index of next cache line

TABLE 9 xtor_ace_cache class

Function	Desc
bool cache_lines_delete (axi_addr_t addr)	Delete cache line at specified address. **Parameters**: addr: cache line address **Returns:** true:cache line deleted false: cache line not present
bool addr_mapped_to_indices_ delete (uint32_t index)	Delete cache line at specified index. **Parameters**: index: cache line index **Returns**: true:cache line deleted false: cache line not present
void merge_data (uint32_t &merged_data, uint32_t first, uint32_t second, uint32_t be)	Merge two data value into single value. **Parameters**: merged_data: reference of merged data first: first data value second: second data value be: byte enables

4.6 ZeBu AMBA xtor_ace_cache_line Class

The following table lists the xtor_ace_cache_line class:

TABLE 10 xtor_ace_cache_line class

Function	Desc
void do_new (uint32_t index, uint32_t cache_line_size, axi_addr_t addr, std::deque< uint32_t > init_data, bool is_unique=0, bool is_clean=0)	Function to initialize new cache line. **Parameters**: cache_line_size size of cache line in bytes. addr: address of cache line init_data: data to initialize cache line /param: is_unique pass true if cache line is unique /param is_clean pass true if cache line is clean.
void set_index (uint32_t index_i)	Function to set index /param index_i index to be set.
bool read (std::deque< uint32_t > &data_o)	Function to get the value of the data word stored in this cacheline /param data_o reference to deque of uint32_t where data is to stored.
bool write (std::deque< uint32_t > data_i, axi_addr_t addr_i, std::deque< bool > byteen_i)	Returns the status of this cache line **Parameters**: is_unique: Indicates if this line may be shared with other caches. is_clean: Indicates if this line is updated relative to main memory.
void set_status (int8_t is_unique_i, int8_t is_clean_i)	Sets the status of this cache line **Parameters**: is_unique Sets the shared status of this line A value of -1 indicates that the shared status need not be changed. A value of 0 indicates that this line may be shared with other caches. A value of 1 indicates that this line is not shared with other caches. is_clean Sets the clean/dirty status of this line that indicates whether this line is updated compared to the memory A value of -1 indicates that the is_clean status need not be changed. A value of 0 indicates that this line is dirty relative to the memory. A value of 1 indicates that this line is clean relative to the memory.
axi_addr_t get_addr ()	Returns the address associated with this cache line.

TABLE 10 xtor_ace_cache_line class

Function	Desc
uint64_t get_age ()	Function to get age of cache line.
void set_age (uint64_t new_age)	Function to set age of cache line. **Parameters**: new_age: age to be set.
void incr_age ()	Function to increment age of cache line.

4.7 Enumerations and Structures

This section explains the enumerations and structures available in the C++ Software Interface.

4.7.1 Enumerations

Use the setParam API to use the enums.

The following enums are supported for the SV Interface:

- *XtorParam*
- *enBurstType*

4.7.1.1 XtorParam

Used to configure parameters/features in AMBA transactors.

TABLE 11 Enumerations, Structures used in ZeBu AMBA transactors

Enumeration members	Scope	Description
AWREADY_TIMEOUT, ARREADY_TIMEOUT, WREADY_TIMEOUT	Master	Sets timeout for specified ready signal
INACTIVITY_TIMEOUT	Master	Sets timeout for inactivity on AXI bus
AWVALID_DELAY, ARVALID_DELAY, WVALID_DELAY, AWVALID_TO_WVALID_DELAY	Master	Sets delay while driving transactions
IGNORE_BRESP	Master	Write response (BRESP) is ignored if set to 1
BVALID_DELAY, RVALID_DELAY	Slave	Sets timeout for specified ready signal
CACHE_LINE_SIZE	ACE Master	Sets size of cache line of ACE Master in bytes

TABLE 11 Enumerations, Structures used in ZeBu AMBA transactors

USE_C_CACHE	ACE Master	Snoop response is automatically driven by xtor if set to 1, else it is driven by testbench
NUM_CACHE_LINES	ACE Master	Number of cache lines in ACE Master cache
CRREADY_TIMEOUT,CDREADY_TIMEOUT	ACE Master	Sets timeout value for snoop response and data channel
CRVALID_DELAY, CDVALID_DELAY	ACE Master	sets delay while driving transactions on snoop response and data channel
DISABLE_CACHE_CHECK	ACE Master	Disable sending WriteBack transaction (due to Dirty Cache) in case of CleanInvalid / CleanShared txn if set to 1

Contains AXI response type for BRESP/RRESP. The following table lists the enumeration members for the `enAXIResp_e` enum.

TABLE 12 enAXIResp_e Enumeration Members

Enumeration members	Scope	Description
`AXI_RESP_OKAY`	Master/Slave	Use for Okay Response.
`AXI_RESP_EXOKAY`	Master/Slave	Use for ExOkay Response.
`AXI_RESP_SLVERR`	Master/Slave	Use for Slave Error Response.
`AXI_RESP_DECERR`	Master/Slave	Use for Decode Error Response.

4.7.1.2 enBurstType

Contains AXI Burst type. The following table lists the enumeration members for the `enBurstType` enum.

TABLE 13 enAXIResp_e Enumeration Members

Enumeration members	Scope	Description
`AXI_BURST_FIXED`	Master/Slave	Use for Fixed type Burst.
`AXI_BURST_INCR`	Master/Slave	Use for Increment type Burst.
`AXI_BURST_WRAP`	Master/Slave	Use for Wrap type Burst.
`AXI_BURST_UNUSED`	Master/Slave	Not Applicable.

4.7.2 Structures

The following structures are supported by the transactor C++ software interface:

- *AXIChanInfo*
- *ACESnpResp*

4.7.2.1 AXIChanInfo

Stores address information for driving AW or AR transaction and is used in transaction APIs, wrTxnNB, rdTxnNB, wrTxn, and rdTx.

This structure is used in the transaction APIs and is present in xtor_amba_defines.hh.

TABLE 14 AXIChanInfo: Struct Members

Struct members	**Scope**	**Description**

axi_id_t id;
axi_addr_t addr;
uint16_t size;
uint8_t len;
uint8_t lock;
uint8_t cache;
uint8_t prot;
uint8_t burst;
	Master/Slave	Can be used for AXI3/AXI4/ ACELite/ACE.

uint8_t qos ; uint8_t region ; uint64_t user ;	Master/Slave	Can be used for AXI4/ ACELite/ACE.
uint8_t snoop ; uint8_t domain ; uint8_t bar ; boo awunique	Master/Slave	Can be used for ACELite/ACE.

4.7.2.2 ACESnpResp

Stores data information for W or R transactions.

TABLE 15

Structure members	**Scope**

byte unsigned len;
int unsigned id;
byte unsigned strb[];
byte unsigned data[];
int unsigned num_data_bytes;
	All

byte unsigned user[];
int unsigned num_user_bytes;
	AXI4/ACELite/ ACE

4.8 Transactor Connection and Initialization

Perform the following steps to initialize and connect a transactor:

- *Starting a Testbench*
- *Configuring the AMBA Transactors*

4.8.1 Starting a Testbench

A typical testbench starts with the following lines:

```
#include <stdio.h>
#include <stdlib.h>
#include <iostream>
#include <string.h>
#include <sstream>
#include <queue>
#include "xtor_amba_slave_svs.hh"
#include "xtor_amba_master_svs.hh"
using namespace ZEBU_IP;
using namespace XTOR_AMBA_MASTER_SVS;
using namespace XTOR_AMBA_SLAVE_SVS;
using namespace ZEBU;
class trivial_test
{
public:
    svt_c_threading *m_threading;
    svt_hw_platform *platform;
    xtor_amba_master_svs   *master;
    xtor_amba_slave_svs    *slave;
    trivial_test()
    {
        svt_c_runtime_cfg* runtime = svt_c_runtime_cfg::get_default();
        m_report = svt_report::get_global_api();
        m_threading = runtime->get_threading_api();
        platform = runtime->get_platform();
        m_report->set_verbosity(svt_report::FULL);
        if (!platform->post_elaboration())
        {
            m_report->fatal("TEST/HW/ELAB", "Platform error in post-
elaboration");
        }

        master       =
xtor_amba_master_svs::getInstance("hw_top.master_node_i",runtime);
        slave          =
xtor_amba_slave_svs::getInstance("hw_top.slave_node_i",runtime);
    }
```

4.8.2 Configuring the AMBA Transactors

ZeBu AMBA transactors can be configured by using configuration API’s provided. An example of configuring parameters and registering callbacks is as follows:

Example 1: Configuring the transactor and registering callbacks

```
            master->enableTxnDump(true);
            master->setDebugLevel(0);
            master->runUntilReset();
            master->runClk(10);

            master->setParam(AWVALID_DELAY, 100);
            master->setParam(ARVALID_DELAY, 100);
            master->setParam(WVALID_DELAY, 20);
            master->setParam(AWVALID_TO_WVALID_DELAY, 50);

           //register the callbacks
```

```
          master->registerBrespCB(&cb_wresp);
          master->registerRrespCB(&cb_rresp);
          slave->registerReadAddrCB(&cb_raddr);
          slave->registerWriteTxnCB(&cb_wrtxn);
```

4.9 Creating AMBA Bus Transactions

The methods described hereafter are used to send and receive transactions on the AMBA Bus.

4.9.1 Seding Read and Write Transactions

After configuring ZeBu xtor_amba_master_svs transactor, create and send transactions as shown below:

```
        AXIChanInfo axi_addr;
        AXIData    *axi_data;

        axi_addr.id      = 0x1;
        axi_addr.len     = 0x2;
        axi_addr.size    = 0x1;
        axi_addr.addr    = 0x2;
        axi_addr.prot    = 0x0;
        axi_addr.lock    = 0x0;
        axi_addr.cache   = 0x0;
        axi_addr.burst   = AXI_BURST_INCR;

        uint32_t total_num_bytes     = (axi_addr.len + 1)* (1 <<
axi_addr.size);
        axi_data                     = new AXIData();
        axiBE_t byte_en              = 0xFF;

        for(int i = 0; i < total_num_bytes; i++)
        {
            uint8_t data      = i;
            axi_data->FillByteArray( &data, 1, &byte_en);
        }
        axi_sent_map[axi_addr.id].push(axi_data);
        master->wrTxnNB(axi_addr, axi_data);
        master->rdTxnNB(axi_addr);
```

4.9.2 Example: Master And Slave Callback

After a read address is sent from master, slave gets this transaction in it’s callback and prepares the response accordingly and calls storeResponse API to push the response in a s/w queue. Send response can then be called from main thread of a testbench or by spawning a pthread. Similar process is followed when slave receives a write transaction and wants to respond to the same. Callbacks are also provided in master to capture the response values in testbench. Following example shows callback usage:

```
// Callback functions to handle responses for Non-blocking and phase APIs
    void cb_wresp(void *xtor, AXIResp &resp)
    {
      enAXIResp *axi_bresp;
        axi_bresp = resp.GetResp();
        if (axi_bresp!=NULL)
        {
            if(axi_bresp[0] == AXI_RESP_OKAY)
                std::cout << "TEST :: OKAY: wresp received resp = " <<
axi_bresp[0] << " for id = " << resp.GetId() << "\n";
            else
                std::cout << "TEST :: ERROR: wresp received resp = " <<
axi_bresp[0] << " for id = " << resp.GetId() << "\n";
        }
        else
            std::cout << "TEST :: ERROR: enAXIResp pointer returned by
GetResp API is NULL \n";
    }
    void cb_rresp(void *xtor,  const AXIData& data , AXIResp &resp)
    {
        AXIData* axi_data_sent = axi_sent_map[resp.GetId()].front();
        axi_sent_map[resp.GetId()].pop();

        bool matched = true;
        uint32_t num_bytes = data.GetLengthByte();
        for(int i = 0; i < num_bytes; i++)
        {
            if(axi_data_sent->ByteEnableIndex(i) != 0)
                if(data.IndexByte(i) != axi_data_sent->IndexByte(i))
                {
                    printf("TEST :: ERROR: comparision failed at location
= %d ,data rx is %x ,data saved is %x
\n",i,data.IndexByte(i),axi_data_sent->IndexByte(i));
                    matched = false;
                }
        }
        if( matched == false)
        {
            all_data_matched = false;
            printf("TEST :: ERROR: rresp received with read data not
matching with the write data for id = 0x%x \n",resp.GetId());
        }
        else
            printf("TEST :: OKAY: rresp received with read data matching
with the write data for id = 0x%x \n ",resp.GetId());
    }
    //slave related callbacks
    void cb_raddr(void* xtor,AXIChanInfo addr )
    {
        xtor_amba_slave_svs* xtor_a = (xtor_amba_slave_svs*) xtor;
        m_report->info( "TEST","Read request received for slave",
svt_report::HIGH);
        printf("ID                    = 0x%x \n", addr.id);
        printf("ADDR                    = 0x%lx \n", addr.addr);
        printf("SIZE                    = 0x%x \n", addr.size);
        printf("LEN                     = 0x%x \n", addr.len);
        printf("LOCK                    = 0x%x \n", addr.lock);

        //send response here
        AXIData *axi_rdata = new AXIData();
        AXIResp* response = new AXIResp;
        response->SetId(addr.id);
        axi_rdata->SetId(addr.id);

        AXIData *axi_saved_data ;
        axi_saved_data  = axi_slave_data_map[addr.addr];
        uint32_t total_num_bytes     = (addr.len + 1)* (1 << addr.size);
        axiBE_t byte_en              = 0xFF;
        if(axi_saved_data != NULL)
        {
            for(int i = 0; i < total_num_bytes; i++)
            {
                uint8_t data  = axi_saved_data->IndexByte(i) ;
                axi_rdata->FillByteArray( &data, 1, &byte_en);
                //printf("TEST :: data being sent is %x \n",data);
            }
            for (int i=0; i<(addr.len+1);i++)
            {
                enAXIResp rresp = AXI_RESP_OKAY;
                response->AddResp(rresp);
            }
                xtor_a->storeReadResponse(response, axi_rdata);
                delete response;
                delete axi_rdata;
}
}
```

```
    void cb_wrtxn(void* xtor,AXIChanInfo addr, AXIData* data)
    {
        xtor_amba_slave_svs* xtor_a = (xtor_amba_slave_svs*) xtor;
        m_report->info( "TEST","Write Txn received for slave",
svt_report::HIGH);
        printf("ID                          = 0x%x \n", addr.id);
        printf("ADDR                        = 0x%lx \n", addr.addr);
        printf("SIZE                        = 0x%x \n", addr.size);
        printf("LEN                         = 0x%x \n", addr.len);
        printf("LOCK                        = 0x%x \n", addr.lock);
        uint8_t* data_ptr        = data->GetDataPt(0);
        const uint8_t* be_ptr    = data->ByteEnableArray();
        uint32_t num_bytes       = data->GetLengthByte();

        printf("Data ::                     =   \t");
        for(uint32_t i = 0 ; i < num_bytes ; i++ )
            printf("0x%x \t", data_ptr[i]);
        printf("\n");
        printf("Byte Enables ::             =   \t");
        for(uint32_t i = 0 ; i < num_bytes ; i++ )
            printf("0x%x \t", be_ptr[i]);
        printf("\n");
        //send response here
        AXIResp* response = new AXIResp;
        response->SetId(addr.id);
        axi_slave_data_map[addr.addr] = data;
        enAXIResp bresp = AXI_RESP_OKAY;
        response->AddResp(bresp);
        xtor_a->storeWriteResponse(response);
        delete response;
}
// Spawn a thread to send responses from Slave xtor
void response_thread ()
```

```
{
    while (!TEST_END)
 {
       slave->sendResponse();  // sends wr/rd responses stored in software
queue
  }
}
```

5 SV Interface Support

The ZeBu AMBA transactor provides SystemVerilog Interface (SVI) API's. They are included in the header files located in the `$ZEBU_IP_ROOT/include`.

The following table describes the header files for the ZeBu AMBA SVI API:

TABLE 16 Header Files for ZeBu AMBA transactors

Header Files	Description
xtor_amba_defines_svs.hh	Contains SV enumerations, structs and defines used in AMBA transactors.
xtor_amba_master_svs.hh	Contains SV API's that are used for xtor_amba_master_svs.
xtor_amba_slave_svs.hh	Contains SV API's that are used for xtor_amba_slave_svs.

This section explains the following topics:

- *ZeBu AMBA Master SVI Classes*
- *ZeBu AMBA Slave SVI Classes*
- *Enumerations and Structures*
- *Transactor Connection and initialization*

5.1 ZeBu AMBA Master SVI Classes

You can configure and transmit/receive on the AMBA bus using the following methods.

TABLE 17 AMBA Master and Slave SVI Classes

Function	Description
function new(input string path)	Construct object of SVI class, xtor_amba_master_if_svs **Parameters**: ● path: Full HDL path of transactor HW instance
task runUntilReset()	Blocking call to wait until reset is complete.
task isResetActive(ref bit status)	Non-blocking call to check if the reset is complete. **Parameters**: ● status: 1: Xtor is in Reset; 0: Xtor is out of reset
function bit setDebugLevel(input byte unsigned level)	Set the verbosity of the messages to be displayed. **Parameters**: level: Verbosity of the messages 4 - Full 3 - High 2 - Medium 1 - Low 0 - NONE
task runClk(int unsigned numclocks)	Blocking call to wait for specified number of clock cycles **Parameters**: ● Numclocks: number of cycles to wait
function void setDebugFile (string filename, int unsigned mode)	Set the file to be used for logging the System verilog Messages **Parameters**: ● mode: Control mode of File 0 - Create a new file 1 - Append to file 2 - Close the file

TABLE 17 AMBA Master and Slave SVI Classes

Function	**Description**
task setParam (AMBAXtorParam_e param, int unsigned value)	set transactor parameters to a specified value **Parameters**: ● **param**: Name of parameter to be set, refer to xtor_amba_defines_svs.svi for definition of enum ● **val**: value that is to be assigned to above parameter
task getParam(AMBAXtorParam_e param, ref int unsigned value);	get value of a specific transactor parameter **Parameters**: ● **param**: Name of parameter to be retrieved, refer to xtor_amba_defines_svs.svi for definition of enum ● **val**: value retrieved
task getHwConfig(ref AMBAXtorHwConfig_s hwconfig)	get value of HW parameters of transactor **Parameters**: ● **hwconfig**: Struct containing HW config values, refer to xtor_amba_defines_svs.svi for definition of struct
task enableTxnDump(bit enable)	Enable transaction dumping on terminal on successful Txn completion **Parameters**: ● Enable: 1: enable dumping, 0: disable dumping
task wrTxnNB(AMBAChanInfo_s addr, AMBAData_s data);	API to send write transaction in non-blocking manner,response is received by callback **Parameters**: ● **Addr**: Address struct for write transaction, refer to xtor_amba_defines_svs.svi for struct definition ● **Data**: struct for data to be sent for Write transaction, refer to xtor_amba_defines_svs.svi for struct definition
task rdTxnNB(AMBAChanInfo_s addr);	API to send read transaction in non-blocking manner,response is received by callback **Parameters**: ● **Addr**: Address struct for read transaction, refer to xtor_amba_defines_svs.svi for struct definition

TABLE 17 AMBA Master and Slave SVI Classes

Function	Description
task wrTxn(AMBAChanInfo_s addr, AMBAData_s data, ref AMBAResp_s bresp);	API to send write transaction in blocking manner **Parameters**: ● **Addr**: Address struct for write transaction, refer to xtor_amba_defines_svs.svi for struct definition ● **Data**: struct for write transaction data, refer to xtor_amba_defines_svs.svi for struct definition ● **Resp**: response received, refer to xtor_amba_defines_svs.svi for struct definition
task rdTxn(AMBAChanInfo_s addr, ref AMBAData_s data, ref AMBAResp_s rresp);	API to send read transaction in blocking manner. **Parameters**: ● **Addr**: Address struct for read transaction, refer to xtor_amba_defines_svs.svi for struct definition. ● **data**: data received for read transaction, refer to xtor_amba_defines_svs.svi for struct definition ● **rresp**: response received, refer to xtor_amba_defines_svs.svi for struct definition
task sendWrAddr(AMBAChanInfo_s addr);	Phase API to send write address transaction in non-blocking manner, response is received by callback. **Parameters**: ● **Addr**: Address struct for write transaction, refer to xtor_amba_defines_svs.svi for struct definition
task sendWrData(AMBAData_s data);	API to send write data transaction in non-blocking manner, response is received by callback. **Parameters**: ● **Data**: AMBAData_s struct holding data channel related information, refer to xtor_amba_defines_svs.svi for struct definition.
task sendRdAddr(AMBAChanInfo_s addr);	API to send read transaction in non-blocking manner, response is received by callback. **Parameters**: ● **addr** Address struct for read transaction, refer to xtor_amba_defines_svs.svi for struct definition
task driveRready(bit value);	API to drive ready signal for Read Response Channel. **Parameters**: ● Value : 1: drive RREADY to high, 0: drive RREADY to low.

TABLE 17 AMBA Master and Slave SVI Classes

Function	Description
task driveBready(bit value);	API to drive ready signal for Write Response Channel **Parameters**: ● **Value**: 1: drive BREADY to high, 0: drive BREADY to low
function bit CheckProtocolConsistency(AMBAChanInfo_s addr, bit is_write);	API to check if AXI transaction is valid as per protocol **Parameters**: ● **addr**: Address struct for AXI transaction, refer to xtor_amba_defines_svs.svi for struct definition. ● **is_write**: 1 if write txn, 0 if read txn.
task AutoSnoopResponse()	API to automatically handle snoop response, can be used only when USE_C_CACHE parameter is set high.
task wresp_callback(ref AMBAResp_s resp)	Callback for reception of write response **Parameters**: ● **Resp**: response received, refer to xtor_amba_defines_svs.svi for struct definition
task rresp_callback(ref AMBAData_s data, ref AMBAResp_s resp)	Callback for reception of read response **Parameters**: ● **data**: data received for read transaction, refer to xtor_amba_defines_svs.svi for struct definition ● **resp**: response received, refer to xtor_amba_defines_svs.svi for struct definition
task rst_assert_callback()	Callback for reception of reset assertion
task rst_deassert_callback()	Callback for reception of reset deassertion
task AC_req_callback(bit [63:0] acaddr, int snoop, int prot)	Callback for reception of Snoop request **Parameters**: ● acaddr ACADDR received with snoop request ● snoop ACSNOOP received with snoop request ● prot ACPROT received with snoop request

TABLE 17 AMBA Master and Slave SVI Classes

Function	**Description**
task sendSnpResp(AMBAXtorSnpResp_s snoopresp)	API to drive CRRESP (snoop response) **Parameters**: ● snoopresp structure containing values to drive CRRESP
task sendSnpData(AMBAData_s snoopdata)	API to drive CDDATA (snoop data) **Parameters**: ● **snoopdata**: data struct to be driven, refer to xtor_amba_defines_svs.svi for information on struct

5.2 ZeBu AMBA Slave SVI Classes

The following table lists the AMBA Slave SVI APIs:

TABLE 18 AMBA Slave SVI API list

Function	Description
function new(input string path)	Construct object of SVI class, xtor_amba_slave_if_svs **Parameters**: ● **path**: Full HDL path of transactor HW instance
task runUntilReset()	Blocking call blocking call to wait for transactor to come out of reset.
task isResetActive(ref bit status)	Non-blocking call to check if transactorhas come out of reset **Parameters**: ● **status**: 1-Xtor is in Reset, 0: Xtor is out of reset
function bit setDebugLevel(input byte unsigned level)	Set the verbosity of the display messages to be displayed **Parameters**: ● **level**: Verbosity of the messages 4 - Full 3 - High 2 - Medium 1 - Low 0 - NONE
task runClk(int unsigned numclocks)	blocking call to wait for specified number of clock cycles **Parameters**: ● **Numclocks**: number of cycles to wait
function void setDebugFile (string filename, int unsigned mode)	set the file to be used for logging the System Verilog messages. **Parameters**: ● **mode**: Control mode of File 0 - Create a new file 1 - Append to file 2 - Close the file

TABLE 18 AMBA Slave SVI API list

Function	Description
task setParam(AMBAXtorParam_e param, int unsigned value)	set transactor parameters to a specified value. **Parameters**: ● param: Name of parameter to be set, refer to xtor_amba_defines_svs.svi for definition of enum ● val: value that is to be assigned to above parameter
task getParam(AMBAXtorParam_e param, ref int unsigned value);	get value of a specific transactor parameter. **Parameters**: ● **param**: Name of parameter to be retrieved, refer to xtor_amba_defines_svs.svi for definition of enum. ● **val**: value retrieved
task getHwConfig(ref AMBAXtorHwConfig_s hwconfig)	get value of HW parameters of transactor. **Parameters**: **hwconfig**: Struct containing HW config values, refer to xtor_amba_defines_svs.svi for definition of struct.
task rst_assert_callback()	Callback for reception of reset assertion.
task rst_deassert_callback()	Callback for reception of reset deassertion.
task sendWriteResponse()	API to send write response, sends the responses that were recorded in WriteTxnCB. Responses are sent in First in First out manner. return true if response was sent, false if no response was stored in queue
task sendReadResponse()	API to send Read & Write responses, sends the responses that were recorded in ReadReqCB/WriteTxnCB, Responses will be sent in First in First out manner. Needs to be called outside of ReadReqCB/ WriteTxnCB, preferably in a separate thread. **Returns** true if response was sent , false if no response was stored in queue

TABLE 18 AMBA Slave SVI API list

Function	Description
task sendResponse()	API to send write and read responses that were recorded in WriteTxnCB and ReadReqCB ,Write responses will be sent first, Read responses will be sent once the write response fifo goes empty **return** true if response was sent , false if no response was stored in Read/Write queue
task driveARready(bit value)	API to drive ready signal for Read Address Channel Parameters: **value** : 1: drive ARREADY to high, 0: drive ARREADY to low
task driveAWready(bit value)	API to drive ready signal for Write Address Channel **Parameters:** **value** : 1: drive ARREADY to high, 0: drive ARREADY to low
task driveWready(bit value)	API to drive ready signal for Write Data Channel **Parameters:** **value** : 1: drive ARREADY to high, 0: drive ARREADY to low
task printStats()	API to print statistics of transactions
task storeWriteResponse(ref AMBAResp_s B_resp)	API to store Write responses in s/w fifo , to be sent later by calling sendWriteResponse API ,Responses will be sent in First in First out manner **Parameters**: **B_resp**: write response to store return true if successful
task storeReadResponse(ref AMBAResp_s R_resp, ref AMBAData_s data)	API to store Read responses in s/w fifo , to be sent later by calling sendReadResponse API,Responses will be sent in First in First out manner **Parameters**: ● R_resp: read response to store ● data: axidata to store, **return** true if successful

TABLE 18 AMBA Slave SVI API list

Function	Description
task rd_addr_callback(ref AMBAChanInfo_s addr)	Callback to receive Read address request **Parameters**: ● **addr**: Address struct for read transaction, refer to xtor_amba_defines_svs.svi for struct definition ● **data**: data received for read transaction, refer to xtor_amba_defines_svs.svi for struct definition
task wr_txn_callback(ref AMBAChanInfo_s addr, ref AMBAData_s data)	Callback to receive write address request & write data **Parameters**: ● **addr**: Address struct for write transaction, refer to xtor_amba_defines_svs.svi for struct definition ● **data**: data received for write transaction, refer to xtor_amba_defines_svs.svi for struct definition

5.3 Enumerations and Structures

This section explains the enumerations and structures available in the SV Interface.

5.3.1 Enumerations

Use the setParam API to use the enums.

The following enums are supported for the SV Interface:

- *AMBAXtorParam_e*
- *enAXIResp_e*

5.3.1.1 AMBAXtorParam_e

Used to configure parameters/features in AMBA transactors. The following table lists the enumeration members for the AMBAXtorParam_e enum.

TABLE 19 Enumerations, Structures used in ZeBu AMBA transactors

Member of the Enum	**Scope**	**Description**
AWREADY_TIMEOUT, ARREADY_TIMEOUT, WREADY_TIMEOUT	Master	Sets timeout for specified ready signal
INACTIVITY_TIMEOUT	Master	Sets timeout for inactivity on AXI bus
AWVALID_DELAY, ARVALID_DELAY, WVALID_DELAY, AWVALID_TO_WVALID_DELAY	Master	Sets delay while driving transactions
IGNORE_BRESP	Master	Write response (BRESP) is ignored if set to 1
BVALID_DELAY, RVALID_DELAY	Slave	Sets timeout for specified ready signal
CACHE_LINE_SIZE	ACE Master	Sets size of cache line of ACE Master in bytes

TABLE 19 Enumerations, Structures used in ZeBu AMBA transactors

USE_C_CACHE	ACE Master	Snoop response is automatically driven by xtor if set to 1, else it is driven by testbench
NUM_CACHE_LINES	ACE Master	Number of cache lines in ACE Master cache
CRREADY_TIMEOUT,CDREADY_TIMEOUT	ACE Master	Sets timeout value for snoop response and data channel
CRVALID_DELAY, CDVALID_DELAY	ACE Master	sets delay while driving transactions on snoop response and data channel
DISABLE_CACHE_CHECK	ACE Master	Disable sending WriteBack transaction (due to Dirty Cache) in case of CleanInvalid / CleanShared txn if set to 1

5.3.1.2 enAXIResp_e

Contains AXI response type for BRESP/RRESP. The following table lists the enumeration members for the `enAXIResp_e` enum.

TABLE 20 enAXIResp_e Enumeration Members

Enumeration members	**Scope**	**Description**
`AXI_RESP_OKAY`	Master/Slave	Use for Okay Response.
`AXI_RESP_EXOKAY`	Master/Slave	Use for ExOkay Response.
`AXI_RESP_SLVERR`	Master/Slave	Use for Slave Error Response.
`AXI_RESP_DECERR`	Master/Slave	Use for Decode Error Response.

5.3.2 Structures

The following structures are supported for the SV Interface:

- *AXIChanInfo*
- *AMBAData_s*
- *AMBAResp_s*
- *AMBAXtorSnpResp_S*
- *AMBAXtorHwConfig_S*

5.3.2.1 AXIChanInfo

Stores address information for driving AW or AR transaction and is used in transaction APIs, wrTxnNB, rdTxnNB, wrTxn, and rdTx.

This structure is used in the transaction APIs and is present in xtor_amba_defines.hh.

TABLE 21 AXIChanInfo: Struct Members

Struct members	Scope	Description

axi_id_t id;
axi_addr_t addr;
uint16_t size;
uint8_t len;
uint8_t lock;
uint8_t cache;
uint8_t prot;
uint8_t burst;
	Master/Slave	Can be used for AXI3/AXI4/ ACELite/ACE.

uint8_t qos ; uint8_t region ; uint64_t user ;	Master/Slave	Can be used for AXI4/ ACELite/ACE.
uint8_t snoop ; uint8_t domain ; uint8_t bar ; boo awunique	Master/Slave	Can be used for ACELite/ACE.

5.3.2.2 AMBAData_s

Stores data information for W or R transactions.

TABLE 22 AMBAData_s: Struct Members

Structure members	Scope

byte unsigned len;
int unsigned id;
byte unsigned strb[];
byte unsigned data[];
int unsigned num_data_bytes;
	All

byte unsigned user[];
int unsigned num_user_bytes;
	AXI4/ACELite/ ACE

5.3.2.3 AMBAResp_s

Stores response information for W or R transactions.

TABLE 23 AMBAResp_s: Struct Members

Structure members	Scope

int unsigned id;
AMBAResp_e resp[];
int unsigned num_valid_resp;
byte unsigned user[]; /
int unsigned num_user_bytes;
	All

5.3.2.4 AMBAXtorSnpResp_S

Stores address information for driving CR transactions.

TABLE 24 AMBAXtorSnpResp_S: Struct Members

Structure members	Scope

byte unsigned dataTransfer;
byte unsigned error;
byte unsigned passDirty;
byte unsigned isShared;
byte unsigned wasUnique;
	ACE

5.3.2.5 AMBAXtorHwConfig_S

Stores address information for driving CR transactions.

TABLE 25 AMBAXtorHwConfig_S: Struct Members

Structure members	Scope

byte is_master; // slave:0, master:>0
int protocol_type; // 0:AXI3, 1:AXI4, 2:ACELITE 3:ACE
int data_width;
int addr_width;
int id_width;
int blen_width;
int interleave_depth;
	ACE

int user_width;	AXI4/ACELite only

int snoop_data_width;
int snoop_addr_width;
	ACE only

5.4 Transactor Connection and initialization

5.4.1 Starting a typical testbench

A typical testbench starts with the following lines:

```
`include "xtor_amba_master_svs.svi"
`include "xtor_amba_slave_svs.svi"
module tb_top ;

  event rd_resp_avail;
  event wr_resp_avail;

  bit[7:0] memory[1023:0];

  // All Instances
  hw_top                    hw_top_inst();
  xtor_amba_master_if_svs   amba_m_if;
  xtor_amba_slave_if_svs    amba_s_if;
```

5.4.2 Master And Slave Callback Usage

Master & Slave callbacks must be extended from xtor_amba_master_cb_svs & xtor_amba_slave_cb_svs classes respectively.

```
  // Callback Class
  class master_callback extends xtor_amba_master_cb_svs;
    task wresp_cb(ref AMBAResp_s resp);
        $display("xtor_amba_master_cb_svs :: Received write resp
callback");
    endtask
    task rresp_cb(ref AMBAData_s data, ref AMBAResp_s resp);
```

```
        $display("xtor_amba_master_cb_svs :: Received read resp callback");
         //for(int unsigned i = 0; i < data.num_data_bytes; i++)
             //$display("Byte %d is 0x%h", i, data.data[i]);
     endtask
   endclass

   class slave_callback extends xtor_amba_slave_cb_svs;
       virtual task rd_addr_cb (ref AMBAChanInfo_s addr);
           AMBAData_s data;
           AMBAResp_s resp;
           $display("xtor_amba_slave_cb_svs :: Received read addr 
callback");
           data.id = addr.id;
           data.num_data_bytes = (addr.len +1) * (1 << addr.size);
           data.data = new[data.num_data_bytes];
           data.num_user_bytes = (addr.len + 1) * 4; //User_width is 32 bit
           data.user = new[data.num_user_bytes];
           for(int unsigned i = 0; i < data.num_data_bytes; i++)
           begin
               data.data[i] = memory[addr.addr + i];
              //$display("TEST:: Packing byte = %h \n", memory[addr.addr + 
i]);
           end
           for(int unsigned i = 0; i < data.num_user_bytes; i++)
           begin
               data.user[i] = memory[addr.addr + i] + 5;
           end
           resp.num_valid_resp = addr.len + 1;
           resp.resp           = new [resp.num_valid_resp ];
           for(int unsigned i = 0; i <resp.num_valid_resp ; i++)
           begin
               resp.resp[i] = AMBA_RESP_OKAY;
           end
```

```
          resp.id = addr.id;
          amba_s_if.storeReadResponse(resp, data);
          ->rd_resp_avail;
      endtask

     virtual task wr_txn_cb(ref AMBAChanInfo_s addr, ref AMBAData_s data);
          AMBAResp_s resp;
          $display("xtor_amba_slave_cb_svs :: Received write txn 
callback");
          for(int unsigned i = 0; i < data.num_data_bytes; i++)
          begin
              //$display("TEST:: Received byte = %h \n", data.data[i]);
              memory[addr.addr + i] = data.data[i];
          end
          resp.num_valid_resp = addr.len + 1;
          resp.resp           = new [resp.num_valid_resp ];
          for(int unsigned i = 0; i <resp.num_valid_resp ; i++)
          begin
              resp.resp[i] = AMBA_RESP_OKAY;
          end
          resp.id = addr.id;
          amba_s_if.storeWriteResponse(resp);
          ->wr_resp_avail;
      endtask
  endclass

  task send_rd_responses(ref xtor_amba_slave_if_svs s_if);
      while(1)
      begin
          @(rd_resp_avail);
          s_if.sendReadResponse();
      end
  endtask
```

```
task send_wr_responses(ref xtor_amba_slave_if_svs s_if);
    while(1)
    begin
        @(wr_resp_avail);
        s_if.sendWriteResponse();
    end
endtask

master_callback          master_cb;
slave_callback           slave_cb;

AMBAXtorHwConfig_s       hw_config;
AMBAChanInfo_s           info;
AMBAData_s               wdata;
AMBAData_s               rdata;
AMBAResp_s               wresp;
AMBAResp_s               rresp;
int unsigned             num_bytes;
int unsigned             num_bytes_user;

initial begin

  $display("TEST:: Starting Testbench");
  master_cb             = new();
  slave_cb              = new();
  amba_m_if             = new ("tb_top.hw_top_inst.master_node_i");
  amba_s_if             = new ("tb_top.hw_top_inst.slave_node_i");
```

5.4.3 Configuring the AMBA Transactors

ZeBu AMBA transactors can be configured by using configuration API’s provided. An example of configuring parameters and registering callbacks is as follows:

```
amba_m_if.callback     = master_cb;
  amba_s_if.callback     = slave_cb;
  amba_m_if.setDebugLevel(0);
  amba_m_if.enableTxnDump(1);
  amba_m_if.runUntilReset();
  amba_s_if.runUntilReset();
  amba_s_if.setDebugLevel(0);
  amba_s_if.runClk(10);
  amba_s_if.getHwConfig(hw_config);

  // spawn threads to take care of slave responses
  fork
      send_rd_responses(amba_s_if);
      send_wr_responses(amba_s_if);
  join_none
```

5.4.4 Creating AMBA Bus Transactions

The methods described hereafter are used to send and receive transactions on the AMBA Bus.

```
  $display("TEST::Starting_transactions \n");

  repeat(50)
  begin
      // Fill the address struct
      info.addr     = 'h10;
      info.len      = 'h4;
      info.size     = 'h2;
      info.burst    = 'h1;
      info.id       = 'h6;
```

```
        // Fill the Data struct
        num_bytes            = (info.len + 1) * (1 << info.size);
       num_bytes_user       = (info.len + 1) * ((hw_config.user_width)/8);
        wdata.data           = new[num_bytes];
        wdata.strb           = new[num_bytes];
        wdata.user           = new[num_bytes_user];
        wdata.id             = info.id;
        wdata.num_data_bytes = num_bytes;
        wdata.num_user_bytes = num_bytes_user;

        for(int i = 0; i < num_bytes; i++)
        begin
            wdata.data[i]    = i;
            wdata.strb[i]    = 'hFF;
        end
        for(int i = 0; i < num_bytes_user; i++)
            wdata.user[i]    = i + 5;

        // Initialize the wresp
        wresp.resp = new[info.len + 1];
        wresp.user = new[num_bytes_user];
        amba_m_if.wrTxn(info, wdata, wresp);
        rdata.data           = new[num_bytes];
        rdata.strb           = new[num_bytes];
        rdata.user           = new[num_bytes_user];
        rresp.resp           = new[info.len + 1];
        rresp.user           = new[num_bytes_user];
        amba_m_if.rdTxn(info, rdata, rresp);

       $display("TEST:: Starting with Non Blocking Txn \n");
       amba_m_if.wrTxnNB(info,wdata);
       amba_m_if.rdTxnNB(info);
```

```
        //amba_m_if.runClk(500);
        wdata.data.delete;
        wdata.strb.delete;
        wdata.user.delete;
        rdata.user.delete;
        rdata.data.delete;
        rdata.strb.delete;
        rresp.user.delete;
        rresp.resp.delete;
    end
    amba_m_if.runClk(5000);
    amba_s_if.printStats();
    $display("TEST:: Testbench PASSED ");
    $display("TEST:: Finishing Testbench");
    $finish;

  end
endmodule
```

6 Tutorial

The transactor package is delivered with several examples, which describe how to use the ZeBu AMBA transactors in conjunction with a DUT. The examples instantiate master and slave node transactors. The testbench is a c++ program driving the transactor which:

- Create ZeBu AMBA by instantiating ZeBu AMBA Master/AMBA Slave objects.
- Configures the transactors
- Sends and receives read and write transactions

This section explains the following topics:

- *Example Structure*
- *Running on ZeBu*
- *Test Cases on ZeBu*

6.1 Example Structure

The following illustrates the Example structure followed in ZeBu AMBA Master/AMBA Slave transactor.

- Examples can be found for each protocol in their respective directories. Following are the example directories available:

  Master:

  - axi_master_dut_slave
  - axi_zrci
  - axi_master_dut_exe
  - axi4_master_dut_slave
  - acelite_master_dut_slave

  Slave:

  - axi_master_slave
  - axi4_master_slave
  - acelite_master_slave

- Each directory further contains the following eg axi_master_slave contains following directories.

  - **src**:
    - **dut** : contains test hardware top files
    - **bench**: contains test bench top and common files used by testbench
    - **tests**: contains all tests case files
    - **env**: contains the environment files needed for compile
  - **ZeBu:** contains Makefile

6.2 Running on ZeBu

To use the example, ensure that your transactor is installed correctly and the ZEBU_IP_ROOT environment variable is set accordingly.

Also, you can compile and execute the example on any ZeBu Server system.

6.2.1 Compiling for ZeBu

The compilation flow is available in the Makefile provided in the example/axi_master_slave/zebu directory. You can launch the compilation using the following compilation target:

```
$ make ZEBU=1 compile
```

The project files are available in the example/axi_master_slave/src/env directory.

6.2.2 Running the Test Case

After successful compilation, use the following command in example/axi_master_slave/zebu directory.

```
$ make ZEBU=1 run TESTNAME=<test case name>
```

Example: $ make ZEBU=1 run TESTNAME=random_txn_test

6.3 Test Cases on ZeBu

The directory example/<example_dir>/src/bench contains test cases of ZeBu AMBA transactors. Following is the description of test cases:

TABLE 26 AMBA Master Test Cases on ZeBu

Test Case	**Description**
axi_master_dut_slave	
channel_delay_test	This test case involves axi master instance and DUT acting as slave. It involves sending write and read transactions from master. Various channel delay values are configured from testbench.
randomize test	This test case involves axi master instance and DUT acting as slave. It involves sending write and read transactions from master having randomized values of id, len ,size,cache ,address etc
trivial_test	This trivial test case involves sending write and read transactions from master. Read data received is compared with data that was sent in write transaction.
trivial_test_nb	This test case involves transmission of non blocking transactions from master
trivial_test_phase	This test involves sending transactions from master in a phased manner
axi4_master_dut_slave	
channel_delay_test	This test case involves axi 4 master instance and DUT acting as slave. It involves sending write and read transactions from master. Various channel delay values are configured from testbench.
trivial_test	This trivial test case involves sending write and read transactions from master. Read data received is compared with data that was sent in write transaction.
trivial_test_nb	This test case involves transmission of non blocking transactions from master
trivial_test_phase	This test involves sending transactions from master in a phased manner

TABLE 26 AMBA Master Test Cases on ZeBu

trivial_test_phase	This test involves sending transactions from master in a phased manner
acelite_master_dut_slave	
channel_delay_test	This test case involves acelite master instance and DUT acting as slave. It involves sending write and read transactions from master. Various channel delay values are configured from testbench.
trivial_test	This trivial test case involves sending write and read transactions from master. Read data received is compared with data that was sent in write transaction.
trivial_test_nb	This test case involves transmission of non blocking transactions from master
trivial_test_phase	This test involves sending transactions from master in a phased manner

TABLE 27 xtor_amba_slave_svs Test Cases on ZeBu

Test Case	**Description**
axi_master_slave	
multi_txn_test	This test case involves sending multiple read/write transactions from master, a separate thread is used to send responses from slave
random_txn_test	This test case involves sending read/write transactions with random attributes from master, a separate thread is used to send responses from slave. Comparison of data sent in write transaction and data received for read transaction is done in master through callback.
trivial_test	This trivial test case involves sending write and read transactions from master. Responses are stored in callback for slave and sent through a separate thread.
out_of_order_txn_test	This test case involves transmission of read data in an out of order manner from slave. The order in which responses are sent is controlled by calling store response in the same manner.
axi_master_slave	

TABLE 27 xtor_amba_slave_svs Test Cases on ZeBu

multi_txn_test	This test case involves sending multiple read/write transactions from master, a separate thread is used to send responses from slave
random_txn_test	This test case involves sending read/write transactions with random attributes from master, a separate thread is used to send responses from slave. Comparison of data sent in write transaction and data received for read transaction is done in master through callback.
trivial_test	This trivial test case involves sending write and read transactions from master. Responses are stored in callback for slave and sent through a separate thread.
out_of_order_txn_test	This test case involves transmission of read data in an out of order manner from slave. The order in which responses are sent is controlled by calling store response in the same manner.
multi_txn_test	This test case involves sending multiple read/write transactions from master, a separate thread is used to send responses from slave
acelite_master_slave	
multi_txn_test	This test case involves sending multiple read/write transactions from master, a separate thread is used to send responses from slave
random_txn_test	This test case involves sending read/write transactions with random attributes from master, a separate thread is used to send responses from slave. Comparison of data sent in write transaction and data received for read transaction is done in master through callback.
trivial_test	This trivial test case involves sending write and read transactions from master. Responses are stored in callback for slave and sent through a separate thread.
out_of_order_txn_test	This test case involves transmission of read data in an out of order manner from slave. The order in which responses are sent is controlled by calling store response in the same manner.