ZeBu®
PCIe Transactor Compliance Test Suite
User Manual

V-2024.03-SP1, February 2025

Copyright Notice and Proprietary Information

Contents

About This Manual 5

Overview 5
Related Documentation 5
Synopsys Statement on Inclusivity and Diversity 5

1 Introduction 7

1.1 Overview 7
1.2 Features 7
1.3 Requirements 7
1.3.1 FLEXlm License 7
1.3.2 ZeBu Software Compatibility 7
1.3.3 Knowledge 8
1.3.4 Software 8
1.4 Limitations 8

2 Installation 9

2.1 Installing the ZeBu PCIe CTS SVS Transactor Package 9
2.2 Package Structure 9
2.3 File Tree 10

3 Hardware Interface 11

4 Software Interface 13

Tables

No table of figures entries found.

About This Manual

Overview

This manual describes how to use the ZeBu PCI Express (PCIe) Transactor based Compliance Test Suite (CTS) with your design being emulated in ZeBu.

NOTE: The features explained in this manual are only intended for the Beta version of the transactor. They are intended to change without any prior notice.

Related Documentation

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

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

Synopsys Statement on Inclusivity and Diversity

Synopsys is committed to creating an inclusive environment where every employee, customer, and partner feels welcomed. We are reviewing and removing exclusionary language from our products and supporting customer-facing collateral. Our effort also includes internal initiatives to remove biased language from our engineering and working environment, including terms that are embedded in our software and IPs. At the same time, we are working to ensure that our web content and software applications are usable to people of varying abilities. You may still find examples of non-inclusive language in our software or documentation as our IPs implement industry-standard specifications that are currently under review to remove exclusionary language.

1 Introduction

1.1 Overview

The ZeBu PCIe SVS transactor based Compliance Test Suite provide directed scenarios of certain features of PCIe protocol which can run up to Gen5 speed. These test cases are created for EnedPoint DUT and Root Complex DUT.

These Test cases are created by using C++ APIs of PCIe Transactor. Test cases are complying with the PCI Express Base specification revision 5.0 version 1.0.

1.2 Features

Test cases are created based on following documents provided by PCI-SIG.

- PCI Express® Architecture Link Layer and Transaction Layer, (Revision 3.0)
- PCI Express® Architecture Configuration Space, (Revision 3.0)
- PCI Express® Base specification (Revision 5.0 version 1.0)

Test cases primarily cover TL,DL and Configuration space specific feature. Some of them are

- FLR (Function Level Reset)
- ECRC (End-to-End CRC)
- TLP Digest
- LCRC (Link CRC)
- AC/NAK
- Sequence Number
- Low power (ASPM L1/L0s and PCI-PM L1, L2)
- L1 Sub States (ASPM, PCI-PM)
- Configuration space related scenarios

1.3 Requirements

1.3.1 FLEXlm License

You need the following FLEXnet license features, For ZS4 platform,

- PCIE XTOR License: The license feature used is either hw_xtormm_pcie (up to Gen4) or hw_xtormm_pcie5 (for Gen5).
- PCIE XTOR CTX License: The license feature hw_xtormm_pcie_ts

NOTE: If the hw_xtormm_pcie or hw_xtormm_pcie5 license feature is not available in the server, then the `zip_ZS4XtorMemBaseLib` license feature is checked out.

1.3.2 ZeBu Software Compatibility

According to the ZeBu system you use, the PCIe SVS transactor requires the following ZeBu versions:

TABLE 1 ZeBu Software Compatibility

Environment	ZeBu Server
64-bit Linux OS	2021.09-2

1.3.3 Knowledge

You must be familiar with the ZeBu product range and have a good knowledge of ZeBu transactors' ZEMI architecture.

Ideally you previously attended Synopsys' training about *Using Transactors* and/or succeeded in *ZeBu Tutorials* concerning transactors.

You are supposed to be familiar with the PCI Express standards and protocol.

1.3.4 Software

You need the following software elements with appropriate licenses (if required):

- 64-bit Linux OS with RHEL 6.
- ZeBu software correctly installed.
- C/C++ compiler: GCC 7.3 (32-bit or 64-bit environment)
- Requires -lZebuXtor additional dynamic library during runtime.

1.4 Limitations

There are some Test cases which has DUT dependencies and to execute them, user has to develop some method "example_test_functions_dut.hh"

There are some cases which are not qualified in Back-Back mode, because of capabilities are not present in XTOR, but if DUT support them then these can run completely.

2 Installation

This section explains the steps to install the *ZeBu PCIe CTS SVS* transactor, under the following topics:

- ❖ Installing the ZeBu PCIe CTS SVS Transactor
- ❖ Package Description
- ❖ File Tree

2.1 Installing the ZeBu PCIe CTS SVS Transactor Package

Prerequisites

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

To install the *ZeBu PCIe CTS* transactor, perform the following steps:

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

```
xtor_pcie_cts_svs.<version>.sh install [options] installed_path
```

where:

- o [options] defines the working environment, which is ZeBu Server-4 environment for the 64-bit Linux OS.
- o Installed_path is the path to your ZeBu IP directory:
  - ♦ If no path argument is specified, the ZEBU_IP_ROOT environment variable is used automatically.
  - ♦ If the path is specified and a ZEBU_IP_ROOT environment variable is also set, the transactor is installed at the defined path and the environment variable is ignored.

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

```
xtor_pcie_cts_svs v.<version_num> has been successfully installed.
```

If an error occurred during the installation, a message is displayed to point out the error. Here is an error message example:

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

2.2 Package Structure

After the ZeBu PCIe transactor has been correctly installed, it comes with the following elements:

.so shared library of the PCIe CTS transactor (libxtor_pcie_cts_svs.so).

Header files of the PCIe Transactor (xtor_cxl_cts_svs.hh).

2.3 File Tree

The following is the transactor file tree after package installation:

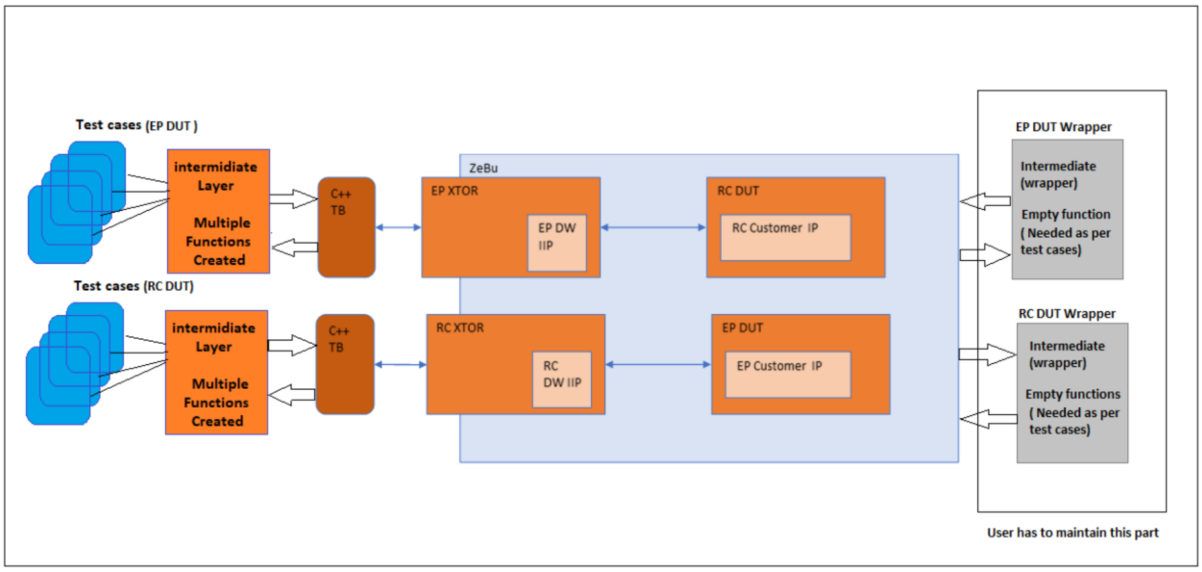
```
$ZEBU_IP_ROOT

`--XTOR

`-- xtor_pcie_svs.<version>

|-- doc

| |-- ZeBu_XTOR_PCIe_CTS_svs_UM.pdf

| |-- foss

| | |-- ZX-XTOR-Library_<ver>_FOSS.PDF

|-- example

| |-- src

| | |-- test_suite_bench <All test cases are present here>

| |-- zebu

|-- include

| |-- xtor_pcie_cts_svs.hh

|-- lib

|-- lib64

| |-- libxtor_pcie_cts_svs.<version>.solf

| `-- libxtor_pcie_cts_svs.so

|-- misc

| |-- pcie_pa_extension

|-- vlog

| |-- vcs

| | |-- xtor_pcie_svs_lanes_model.sv

| | |-- xtor_pcie_svs.sv
```

3 Hardware Interface

Same as PCIE XTOR. For more please refer to "`ZeBu_XTOR_PCIe_svs_UM.pdf`" for Hardware section part.

4 Software Interface

Following is the architecture of PCIE XTOR CTS.



There are following primary functions are created as part of Intermediate Layer (xtor_pcie_cts_svs.hh) of CTS

Method	Description
ep_enumeration_function	This function will perform enumeration for EP DUT and program the BAR, it also checks the memory pointed BAR should not be overlapped, also it provides whether EP DUT support Mem 32/64 and IO information. `ep_enumeration_function(uint32_t* MEM32_SUPPORT,uint32_t*MEM64_SUPPORT,uint32_t* IO_SUPPORT);` * param *MEM32_SUPPORT :During enumeration this value sets if MEM32 is supported else 0 if not supported. * param *MEM64_SUPPORT:During enumeration this value sets if MEM64 is supported else 0 if not supported. * param *IO_SUPPORT:During enumeration this value sets if IO operation is supported else 0 if not supported
rc_enumeration_function	This function will perform enumeration for RC DUT and program the BAR, based on certain input which user must provide.
read_pcie_configuration	This method is used to generate Cfg read request. `uint32_t pcie_xtor_tb::read_pcie_configuration(unsigned cfg_num, int regNum, uint8_t busNum, uint8_t devNum,uint8_t funcNum)`

	param cfg_num for type0 its value =0 & for type 1 its value =1 param regnum contains offset of a target register. param busNum it indicates BUS number of devices to be connected. param devNum it indicates targeted Device number. param funcNum it indicates targeted function number of devices. ret val uint32 int type return read value of targeted register.
write_pcie_configuration	This method is used to generate Cfg write request. `void pcie_xtor_tb::write_pcie_configuration(unsigned cfg_num,uint32_t _data, int regNum, uint8_t busNum, uint8_t devNum, uint8_t funcNum,uint8_t be)` param cfg_num for type0 its value =0 & for type 1 its value =1 param _data modified value for targeted register. param regnum contains offset of a target register. param busNum it indicates BUS number of devices to be connected. param devNum it indicates targeted Device number. param funcNum it indicates targeted function number of devices.
ltssm_prints	This will print LTSSM state of XTOR
l1sub_prints	This will print L1 Sub states of XTOR
test_error_exit	This method calls when test wants to exit with error.
initiate_exit_hotreset	Initiate hot reset through software by setting secondary reset field in bridge control register
wait_entry_exit_hotreset	Wait and trigger exit from HOT reset state
entry_D0_uninitialized_state	This method enters DO uninitialized state. `void pcie_xtor_tb::entry_D0_uninitialized_state(float wait_time,float time_periodGen1,float scale_factor,unsigned pcie_base_capability_baseaddr)` * param wait_time equivalent time to wait for a process to be completed as per spec. * param time_periodGen1 Equivalent to time period of pcie gen 1. * param scale_factor SNPS DWC IIP core fast simulation scaling factor. * param pcie_base_capability_baseaddr denotes base address for pcie base capability.
set_rc_baraddr	This method returns an initial start address for RC BAR register `/*  void set_rc_baraddr( uint32_t *BAR0_RC_SET32,uint32_t* BAR1_RC_SET32);` * param BAR0_RC_SET32 user defined start address for BAR0. * param BAR1_RC_SET32,user defined start address for BAR1.

mem32_addr	Generate random address for targeted Bar with valid range based on length field `uint32_t mem32_addr (unsigned length);`
mem64_addr	Generate random address for targeted Bar with valid range based on length field `long long int mem64_addr (unsigned length);`
enable_ecrc_checking	It sets ECRC check enable bit if ECRC check support bit is set. `bool pcie_xtor_tb::enable_ecrc_checking(unsigned AER_capability_baseaddress,uint8_t *ex_error_cnt)` * param AER_capability_baseaddress denotes base address for AER capability. * param ex_error_cnt its a counter that gets incremented as an error occur. * retval is bool that indicates whether ecrc support and ecrc enable has been set or not.
ecrc_error_reporting_config	Setting ECRC reporting, severity, Masking, SERR. `void pcie_xtor_tb::ecrc_error_reporting_config(uint32_t serr, uint32_t reporting, uint32_t mask, uint32_t severity,unsigned` AER_capability_baseaddress,unsigned pcie_base_capability_baseaddr) * param serr sets or resets serr#enable bit in type 0 command register. * param ecrc_reporting enables or disables error reporting bit in device control register. * param mask masks or unmasks ecrc error bit in uncorrectable error masking register. * param severity sets ecrc error severity as fatal or nonfatal * param AER_capability_baseaddress denotes base address for AER capability. * param pcie_base_capability_baseaddr denotes base address for pcie base capability.
crc_error_injection	To enable CRC error injection for tx/Rx Tlp/Dllp * param crc_type defines type of crc encoded as follow. * Tx Path `void pcie_xtor_tb::crc_error_injection (uint8_t crc_type, uint8_t num, unsigned base_addr,int ex_error_cnt)` 0000b: New TLP's LCRC error injection 0001b: 16bCRC error injection of ACK/NAK DLLP 0010b: 16bCRC error injection of Update-FC DLLP 0011b: New TLP's ECRC error injection 0100b: TLP's FCRC error injection (128b/130b)

	0101b: Parity error of TSOS (128b/130b) 0110b: Parity error of SKPOS (128b/130b) Rx Path 1000b: LCRC error injection 1011b: ECRC error injection Others: Reserved * param tlp_num no of Tlp/Dllp to be corrupted. * param ex_error_cnt: if 1 or more then test fails. */
tx_tlp_dllp_seqnum_change	Enable Seq num error injection for tx Tlp/Dllp. `void pcie_xtor_tb::tx_tlp_dllp_seqnum_change (uint8_t einj_seqnum_type, int seqnum_change, uint8_t num,unsigned base_addr,uint8_t *ex_error_cnt)` * param tlp_num no of Tlp/Dllp to be corrupted. * param ex_error_cnt: if 1 or more then test fails.
initiate_duplicate_tlp	To generate duplicate/ nulliifed TLP `void pcie_xtor_tb::initiate_duplicate_tlp (uint8_t einj5_specified_tlp, uint8_t num,unsigned base_addr,uint8_t *ex_error_cnt)` * param einj5_specified_tlp indicates whether tlp will be duplicate or nullified. * num = no of tlp get effected. * param ex_error_cnt: if 1 or more then test fails.
flr_support_triggering	To check support and trigger FLR for EP function. `void flr_support_triggering(int ncyc);` * param ncyc no of wait cycle to finish FLR at current rate . * param ex_error_cnt: if 1 or more then test fails. * param pcie_base_capability_baseaddr : provides base address for Pcie base capability.
	This function will catch current sequence number for TX and Rx TLP `void get_curnt_tx_rx_tlp_seq_num(uint32_t* tx_seq_num_tx_tlp,uint32_t* tx_seq_num_rx_tlp);` * param * tx_seq_num_tx_tlp: this pointer will hold the current sequence * number of Transmitted TLP. * param * tx_seq_num_rx_tlp: this pointer will hold the current sequence * number of received TLP.
get_curnt_tx_rx_acknack_seq_num	This function will catch current sequence number for TX and Rx ACK/NAK DLLP.

	`void get_curnt_tx_rx_acknack_seq_num(uint32_t* tx_seq_num_tx_acknack,uint32_t* tx_seq_num_rx_acknack);` * param * tx_seq_num_tx_acknack: this pointer will hold the current sequence * number of Transmitted ACK/NAK DLLP. * param * tx_seq_num_rx_acknack: this pointer will hold the current sequence * number of received ACK/NAK DLLP.
hold_ack_nak_dllp	This function will enable register for blocking ACK/NAK DLLP transmission. `void pcie_xtor_tb::hold_ack_nak_dllp(unsigned addr,int ex_error_cnt)` * param addr: this represents base address of RAS_DES_CAPABILITY base * address.
aspm_nak_msg_l1_exit	This function will enable for ASPM L1 entry and then will send PM NAK message so that L1 entry request will be dropped... `void aspm_nak_msg_l1_exit(unsigned xtor_pcie_capability_baseaddr,unsigned dut_pcie_capability_baseaddr);` * param dut_pcie_capability_baseaddr :dut PCIe capability Base address. * param xtor_pcie_capability_baseaddr :xtor PCIe capability Base * address. * param dut_pcie_capability_baseaddr :Dut PCIe capability Base * address.
initiate_l1_1_entry_ep_dut	This function is used to initiate L1_1 entry to EP DUT . It also provides information whether EP DUT supported L1 Sub State feature or not. `void initiate_l1_1_entry_ep_dut(unsigned xtor_ext_cap_base_addr,unsigned dut_ext_cap_base_addr, unsigned xtor_base_cap_addr,unsigned dut_base_cap_addr)`
initiate_l1_1_entry_rc_dut	This function is used to initiate L1_1 entry to EP DUT . It also provides information whether EP DUT supported L1 Sub State feature or not. `void initiate_l1_1_entry(unsigned xtor_ext_cap_base_addr,unsigned dut_ext_cap_base_addr, unsigned xtor_base_cap_addr,unsigned dut_base_cap_addr,unsigned is_aspm)`
initiate_l1_2_entry_ep_dut	This function is used to initiate L1_2 entry to EP DUT .

	It also provides information whether EP DUT supported L1 Sub State feature or not. `void initiate_l1_2_entry_ep_dut(unsigned xtor_l1_ss_baseaddr, unsigned dut_l1_ss_baseaddr, unsigned xtor_pcie_baseaddr, unsigned dut_pcie_baseaddr)`
initiate_l1_2_entry_rc_dut	This function is used to initiate L1_2 entry to EP DUT . It also provides information whether EP DUT supported L1 Sub State feature or not. `void initiate_l1_2_entry_rc_dut(unsigned xtor_base_addr, unsigned dut_base_addr)`
initiate_l1_ss_exit	This function is used to initiate L1_1 exit by deassertion of CLKREQ# from RC XTOR side
cycle_calculator	Convert require wait time to no of cycle. `int cycle_calculator(float wait_time, float genclk_tp,float scale_factor);` * param wait_time require wait time for a process to be completed. * param genclk_tp time period of active device clk. * param scale factor a constant value = 10e-3. */