ZeBu® Transactors Compilation Application Note

Version V-2024.03-1, July 2024

SYNOPSYS®

Copyright and Proprietary Information Notice

Destination Control Statement

All technical data contained in this publication is subject to the export control laws of the United States of America. Disclosure to nationals of other countries contrary to United States law is prohibited. It is the reader's responsibility to determine the applicable regulations and to comply with them.

Disclaimer

SYNOPSYS, INC., AND ITS LICENSORS MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Trademarks

Synopsys and certain Synopsys product names are trademarks of Synopsys, as set forth at https://www.synopsys.com/company/legal/trademarks-brands.html.
All other product or company names may be trademarks of their respective owners.

Free and Open-Source Licensing Notices

If applicable, Free and Open-Source Software (FOSS) licensing notices are available in the product installation.

Third-Party Links

Any links to third-party websites included in this document are for your convenience only. Synopsys does not endorse and is not responsible for such websites and their practices, including privacy practices, availability, and content.

www.synopsys.com

Feedback

Contents

Preface

This chapter has the following topics:

About This Book

The *ZeBu Transactors Compilation Application Note* describes the steps to instantiate and compile a ZeBu transactor with an example. It also details certain guidelines that should be followed when a large number of transactors are used in a design.

Contents of This Book

The *Compiling ZeBu® Transactors Application Note* has the following chapters:

Chapter	Describes...
Introduction to ZeBu Transactors	Brief introduction to ZeBu transactors
Instantiating a Transactor	Instructions on how to instantiate a transactor
Guidelines for Mapping Transactors	The guidelines that should be followed for mapping multiple transactors
Compiling a Transactor	The steps to compile a transactor

Related Documentation

Document Name	Description
ZeBu User Guide	Provides detailed information on using ZeBu.

Document Name	Description
ZeBu Debug Guide	Provides information on tools you can use for debugging.
ZeBu Debug Methodology Guide	Provides debug methodologies that you can use for debugging.
ZeBu Unified Command-Line User Guide	Provides the usage of Unified Command-Line Interface (UCLI) for debugging your design.
ZeBu UTF Reference Guide	Describes Unified Tcl Format (UTF) commands used with ZeBu.
ZeBu Power Aware Verification User Guide	Describes how to use Power Aware verification in ZeBu environment, from the source files to runtime.
ZeBu Functional Coverage User Guide	Describes collecting functional coverage in emulation.
Simulation Acceleration User Guide	Provides information on how to use Simulation Acceleration to enable cosimulating SystemVerilog testbenches with the DUT
ZeBu Verdi Integration Guide	Provides Verdi features that you can use with ZeBu. This document is available in the Verdi documentation set.
ZeBu Runtime Performance Analysis With zTune User Guide	Provides information about runtime emulation performance analysis with zTune.
ZeBu Synthesis Verification User Guide	Provides information about the zFmCheck tool. This tool validates whether the ZeBu synthesis engine has correctly synthesized Verilog modules, VHDL entity, or architecture pairs.
ZeBu Custom DPI Based Transactors User Guide	Describes ZEMI-3 that enables writing transactors for functional testing of a design.
ZeBu LCA Features Guide	Provides a list of Limited Customer Availability (LCA) features available with ZeBu.
ZeBu Synthesis Verification User Guide	Provides a description of zFmCheck.
ZeBu Transactors Compilation Application Note	Provides detailed steps to instantiate and compile a ZeBu transactor.
ZeBu zManualPartitioner Application Note	Describes the zManualPartitioner feature for ZeBu. It is a graphical interface to manually partition a design.
ZeBu Hybrid Emulation Application Note	Provides an overview of the hybrid emulation solution and its components.

Typographical Conventions

This document uses the following typographical conventions:

To indicate	Convention Used
Program code	`OUT <= IN;`
Object names	`OUT`
Variables representing objects names	`<sig-name>`
Message	Active low signal name '<sig-name>' must end with _X
Message location	`OUT` <= IN;
Reworked example with message removed	`OUT_X` <= IN;
Important Information	**NOTE:** This rule...

The following table describes the syntax used in this document:

Syntax	Description
[ ] (Square brackets)	An optional entry
{ } (Curly braces)	An entry that can be specified once or multiple times
| (Vertical bar)	A list of choices out of which you can choose one
... (Horizontal ellipsis)	Other options that you can specify

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 to ZeBu Transactors

ZeBu transactors are a family of protocol-specific transaction-based verification solutions that enable you to quickly build a complete system-level test environment for your system-on-chip (SoC) designs. ZeBu transactors support common protocols and standards, such as PCI Express 3.0, AMBA, USB, MIPI CSI-2 and MIPI DSI, I2C, I2S, Gigabit Ethernet and 10 Gigabit Ethernet, Digital Video, and JTAG.

These transactors include Bus Functional Models (BFMs) that are mapped into the ZeBu RTB hardware. These BFMs provide maximum performance and ensure that a transactor is synchronized to an emulated design.

Each transactor includes C/C++ APIs to quickly create testbenches and drivers that generate real-world traffic, or to link to virtual platforms. ZeBu transactors are complemented by the ZEMI-3 transactor compiler, so you can generate your own ZeBu-compatible custom transactors. ZeBu transactors are fully compatible with ZeBu's interactive design debugging, supporting single-stepping of the clocks, and waveform dumping with Dynamic-Probes.

2

Instantiating a Transactor

To instantiate a transactor wrapper, perform the following steps:

1. Connect the controlled clock (`.xtor_cclock0`) generated by the `zceiClockPort` as an input to the transactor.

For example, if you are using a CGMAC transactor, the `ref_clk_3125` clock must be connected to the `input .xtor_cclock0`.

1. Instantiate the transactor black box in the desired location. This black box can be found in the `$ZEBU_IP_ROOT/uc_xtor` directory.
2. Configure the `zceiClockPort` in the top level Verilog wrapper.

The transactor can be instantiated at any level of the hierarchy.

The following is an example of instantiating a CGMAC transactor wrapper in the top-level Verilog wrapper.

```
module example_cgmac_wrapper;

`ifdef MII_SDR128b

wire            ref_clk_3125;
wire            ref_clk_1562_5;
wire            cgmii_rx_clk1;
wire [127:0]    cgmii_rxd1;
wire [15:0]     cgmii_rxc1;
wire            cgmii_tx_clk1;
wire [127:0]    cgmii_txd1;
wire [15:0]     cgmii_txc1;
wire            cgmii_rx_clk2;
wire [127:0]    cgmii_rxd2;
wire [15:0]     cgmii_rxc2;
wire            cgmii_tx_clk2;

wire [127:0]    cgmii_txd2;
wire [15:0]     cgmii_txc2;
wire [31:0]     timestamp;
  cgmac_sdr128_driver cgmac_driver_1 (
  .ref_clk_3125           (ref_clk_3125),
  .ref_clk_3125_primary   (1'b1),
  .ref_clk_1562_5         (ref_clk_1562_5),
  .ref_clk_1562_5_primary (1'b1),
```

```
  .cgmii_rx_clk           (cgmii_rx_clk1),
  .cgmii_rxd              (cgmii_rxd1[127:0]),
  .cgmii_rxc              (cgmii_rxc1[15:0]),
  .cgmii_tx_clk           (cgmii_tx_clk1),
  .cgmii_txd              (cgmii_txd1[127:0]),
  .cgmii_txc              (cgmii_txc1[15:0]),
  .xtor_cclock0           (ref_clk_3125)
  );
  cgmac_sdr128_driver cgmac_driver_2 (
  .ref_clk_3125           (ref_clk_3125),
  .ref_clk_3125_primary   (1'b1),
  .ref_clk_1562_5         (ref_clk_1562_5),
  .ref_clk_1562_5_primary (1'b1),
  .cgmii_rx_clk           (cgmii_rx_clk2),
  .cgmii_rxd              (cgmii_rxd2[127:0]),
  .cgmii_rxc              (cgmii_rxc2[15:0]),
  .cgmii_tx_clk           (cgmii_tx_clk2),
  .cgmii_txd              (cgmii_txd2[127:0]),
  .cgmii_txc              (cgmii_txc2[15:0]),
  .xtor_cclock0           (ref_clk_3125)
  );
  zceiClockPort ref_clk_1 ( .cclock  (ref_clk_1562_5));
  zceiClockPort ref_clk_2 ( .cclock  (ref_clk_3125));

dut dut(
  .ref_clk_1562_5       (ref_clk_1562_5),
  .timestamp            (timestamp),
  .cgmii_rx_clk1        (cgmii_rx_clk1),
  .cgmii_rxd1           (cgmii_rxd1),
  .cgmii_rxc1           (cgmii_rxc1),
  .cgmii_tx_clk1        (cgmii_tx_clk1),
  .cgmii_txd1           (cgmii_txd1),
  .cgmii_txc1           (cgmii_txc1),
  .cgmii_rx_clk2        (cgmii_rx_clk2),
  .cgmii_rxd2           (cgmii_rxd2),
  .cgmii_rxc2           (cgmii_rxc2),
  .cgmii_tx_clk2        (cgmii_tx_clk2),
  .cgmii_txd2           (cgmii_txd2),
  .cgmii_txc2           (cgmii_txc2)
  );

`endif

endmodule
```

3

Guidelines for Mapping Transactors

When a large number of transactors is used, it is necessary to map these transactors to different FPGAs. Such an environment is called multi-RTB environment.

You can declare a script for mapping a transactor in a multi-RTB environment using the following command:

```
xtors -mapping_script <transactor_mapping_file>.tcl
```

The transactor mapping script is a tcl file, containing the following commands: `defmapping -rtlname <xtor_name> <module_loc>`

where,

- `<xtor_name>`:
  - Specifies the name of the instantiated transactor.
  - Supports wildcards in defmapping with RTL names.
- `<module_loc>`: Specifies the ZeBu module to which the transactor is mapped. The format for `<module_loc>` is `U0_<module_ID>`, for example, `U0_M0`.

4

Compiling a Transactor

This chapter lists the prerequisites for compiling a ZeBu transactor. It also provides two examples: one to explain the structure of a UTF file and the other to explain compilation of a ZeBu transactor.

Prerequisites

Ensure that you have created the UTF file and the top-level Verilog wrapper.

Steps to Compile Transactors

To compile the transactors, perform the following steps:

1. Point the `$ZEBU_IP_ROOT` environment variable to the location where your ZeBu transactors are installed.

For example, using a `csh` shell: `setenv ZEBU_IP_ROOT<installation_path>/zebu_ip`

1. Add the following command to the ZeBu UTF project file: `xtors -use_zebu_ip_root true`
2. Instantiate the transactor instance in the hardware top. The module description can be found in the following Verilog file: (`ZEBU_IP_ROOT/uc_xtot/xtrodrivername.v`).

ZeBu performs the black-box testing of the transactor module at the front-end compilation stage.

After the front-end compilation, ZeBu collects the transactor's source from the `$ZEBU_IP_ROOT` path and proceeds to with back-end compilation.

Example of a .utf file

The following is an example of a `.utf` file for CGMAC.

```
vcs_exec_command { vcs -full64  \
             -hw_top=example_cgmac_wrapper \
             +libext+.v \
             -y $ZEBU_IP_ROOT/uc_xtor \
```

Feedback

```
              +define+$CGMAC_ITF \
              ../src/dut/example_cgmac_wrapper.v \

              ../src/dut/dut.v }
architecture_file -filename "$env(FILE_CONF)"
grid_cmd -submit $env(REMOTECMD) -delete  {}
grid_cmd -queue ZebuIse -submit $env(REMOTECMD) -delete {} -njobs
 $env(MAXISEJOBS)
xtors -use_zebu_ip_root true
```

If you want to use Xilinx primitives, include the path of these primitives in the example transactor, as shown below:

```
-y $ZEBU_XIL/ISE/verilog/src/unisims
```

For more information about the example `.utf` file, see

`example/src/env/<example_transactorname_wrapper>.utf`.

Example of Transactor Compilation

In this section, the CGMAC transactor is used as an example to explain transactor compilation in ZeBu.

To compile the CGMAC transactor, perform the following steps:

1. Set the `$ZEBU_IP_ROOT` environment variable to the installation path of the transactor.
2. Navigate to the `example/Zebu` directory.
3. Launch the compilation using `% make compil`.