ZeBu™ Fast LPDDR3 SDRAM Memory Models

Document revision – a –

April 2012

Version 1.0

VSIP016

Copyright Notice
Proprietary Information

Right to Copy Documentation

Destination Control Statement

Disclaimer

Table of Contents

Figures

Tables

About This Manual

Overview

This manual describes the library of ZeBu ZLPDDR3 SDRAM synthesizable memory models (also known as Low Power DDR3 or Mobile DDR3 SDRAM), with optimized memory capacity and performance.

History

This table gives information about the content of each revision of this manual, with indication of specific applicable version.

Doc Revision	Product Version	Date	Evolution
a	1.0	Apr 12	First edition.

Related Documentation

- For details about the ZeBu supported features and limitations, you should refer to the ***ZeBu Release Notes*** in the ZeBu documentation package which corresponds to the software version you are using.
- Application Note VSAN001: ***Guidelines for the use of ZeBu DDRx Models***.

1 Introduction

1.1 ZLPDDR3 Memory Models

The ZeBu ZLPDDR3 synthesizable memory models can be used to model any Synchronous Low Power Double-Data Rate 3 (LPDDR3) DRAM.

The ZLPDDR3 library is provided as a set of IP models, with various densities and architectures listed in Section 1.3, compliant with LPDDR3 SDRAM memory devices.

These models are based on ZeBu zrm-based memory models. The type and size of zrm-based memory models depend on the ZLPDDR3 size and architecture.

The ZLPDDR3 memory models provide the usual ZeBu hardware debugging features such as runtime memory upload/download and memory cell READ/WRITE.

1.2 LPDDR3 Compliance

The ZLPDDR3 memory models are fully compliant with the LPDDR3 specifications documents issued by the JEDEC (JESD209-3) and available to JEDEC members (www.jedec.org).

These ZLPDRR3 memory models contain several extensions, listed by the JEDEC workgroup, for advanced devices modeling:

- Support of the Burst Terminate (BST) operation, compliant with the JC-42.6 Item 1790.40 extension of the JEDEC standard.
- Support of the Burst Length of 16 (BL16) as stated in the Mode Register 1 encoding (3'b100).
- Support of extended Read/Write latencies as stated in Mode Register 2, implementing a RL up to 12 and a WL up to 9, as defined in the SetA/SetB Write latencies configured by MR0[6].

1.3 ZLPDDR3 Library

ZeBu ZLPDDR3 memory models are available for 4Gb, 8Gb and 16Gb memory capacity components on ZeBu-Server and ZeBu-Blade2.
For each capacity, the ZLPDDR3 memory component comes in x16 and x32 architectures.

Table 1: ZLPDDR3 Models

	x16	x32
4 Gb	`zlpddr3_4Gb_256x16`	`zlpddr3_4Gb_128x32`
8 Gb	`zlpddr3_8Gb_512x16`	`zlpddr3_8Gb_256x32`
16 Gb	`zlpddr3_16Gb_1024x16`	`zlpddr3_16Gb_512x32`

1.4 ZLPDDR3 Capacity with ZeBu

The ZeBu-Server and ZeBu-Blade2 systems can both handle up to 8 ZLPDDR3 memory model instances.

1.5 Requirements

1.5.1 `FLEXlm` License Features

You need the appropriate `FLEXlm` license features:

- `zip_LPDDR3_All` for the entire ZLPDDR3 model library

1.5.2 Compilation Setup

The following setup is mandatory to compile the IP within the DUT:

Environment	ZeBu- Server	ZeBu- Blade2
64-bit Linux OS	6_3_1 or later	6_3_2 or later

1.5.3 Simulation Setup

The following setup is mandatory to use the LPDDR3 simulation model:

Environment	VCS	Modelsim
32-bit Linux OS	All VCS versions except: • `vcs-mx-2006.06-SP1`	All MTI versions except: • `modelsim-6.3`
64-bit Linux OS	All VCS versions except: • `vcs-mx-2006.06-SP1`	All MTI versions except: • `modelsim-6.3`

Note: The LPDDR3 simulation model can be used on any platform (32 or 64 bits) and does not require ZeBu software.

1.6 Performance

1.6.1 Logic Resources

The ZLPDDR3 synthesizable ZeBu models use the following FPGA resources:

Table 2: Logic Resources

Memory Model	Resources
`zLPDDR3_4Gb_256x16`	1174 registers / 1254 LUTs / 1 single-port zrm
`zLPDDR3_16Gb_1024x16`	1214 registers / 1302 LUTs / 1 single-port zrm

1.6.2 Operating Frequency on ZeBu

The ZLPDDR3 performance depends on the clock source provided to the clock input `Clk_t` signal: either the primary clock is directly connected to a DUT clock signal sourced from a ZeBu clock, or the derived clock is connected to a combinational DUT clock signal building a gated or divided clock from a ZeBu clock.

Table 3: Operating Frequency

Memory Model	ZeBu-Server DDR3 Mem Server	ZeBu-Blade2 DDR3 Mem Server
ZLPDDR3 (Prim. clock)	12.5 MHz	12.5 MHz
ZLPDDR3 (Div. clock)	8.0 MHz	8.0 MHz

Note: These figures may drop by 10 to 20% if the memory server is shared by several large design memories.

1.7 Limitations

The following LPDDR3 operations are not supported by the current ZLPDDR3 models:

- Write Leveling feature of the LPDDR3 device
- CA Training sequence
- The Refresh and Self Refresh operations are ignored by the ZLPDDR3 model

1.8 ZLPDDR3 DIMM Memory Models

ZeBu ZLPDDR3 memory models can be used to build UDIMM (unbuffered DIMM) and RDIMM (registered DIMM) memory models.

ZLPDDR3_UDIMM and ZLPDDR3_RDIMM memory models are customized and supplied by EVE Solutions group, upon request, according to detailed user requirements.

2 Package Content

2.1 Package Description

The ZLPDDR3 memory models come with the following elements:

- Encrypted ZeBu models, EDIF netlists, for implementation (`edif` directory)
- ZLPDDR3 component blackbox, for design synthesis (`component` directory)
- ZLPDDR3 wrapper with bi-directional DQS, for full synthesis compatibility with JEDEC standard LPDDR3 component (`wrapper_rtl` directory)
- Verilog gate-level netlist, for model HDL simulation (`simu`)
- `.so` libraries, for model HDL simulation (`libIpSimu`)

2.2 Installation

To install the ZLPDDR3 package, you must first unpack it using:

```
$ tar -xcvf ZLPDDR3.<version>.tgz
```

You must also set the `ZEBU_IP_ROOT` environment variable in your shell to point to your IP installation directory.

Then launch the install script from unpacked directory:

```
./install
```

2.3 File Tree

```
ZLPDDR3.<version>
  |-- <size>
  |    `-- <arch>
  |         |-- component
  |         |    |-- <model_name>_bidir.v
  |         |    |-- <model_name>_bidir.vhd
  |         |    |-- <model_name>.v
  |         |    `-- <model_name>.vhd
  |         |-- edif
  |         |    `-- <model_name>.edf
  |         |-- wrapper_rtl
  |         |    `-- <model_name>_bidir.v
  |         `-- simu
  |              |-- mti
  |              |    `-- <model_name>.vp
  |              |-- nc
  |              `-- VCS
  |                   `-- <model_name>.vp
  |-- doc
  |-- install
  |-- libIpSimu
  |    |-- libIpSimu_32.so
  |    |-- libIpSimu_64.so
  `-- script
       `-- Install_import_file.sh
```

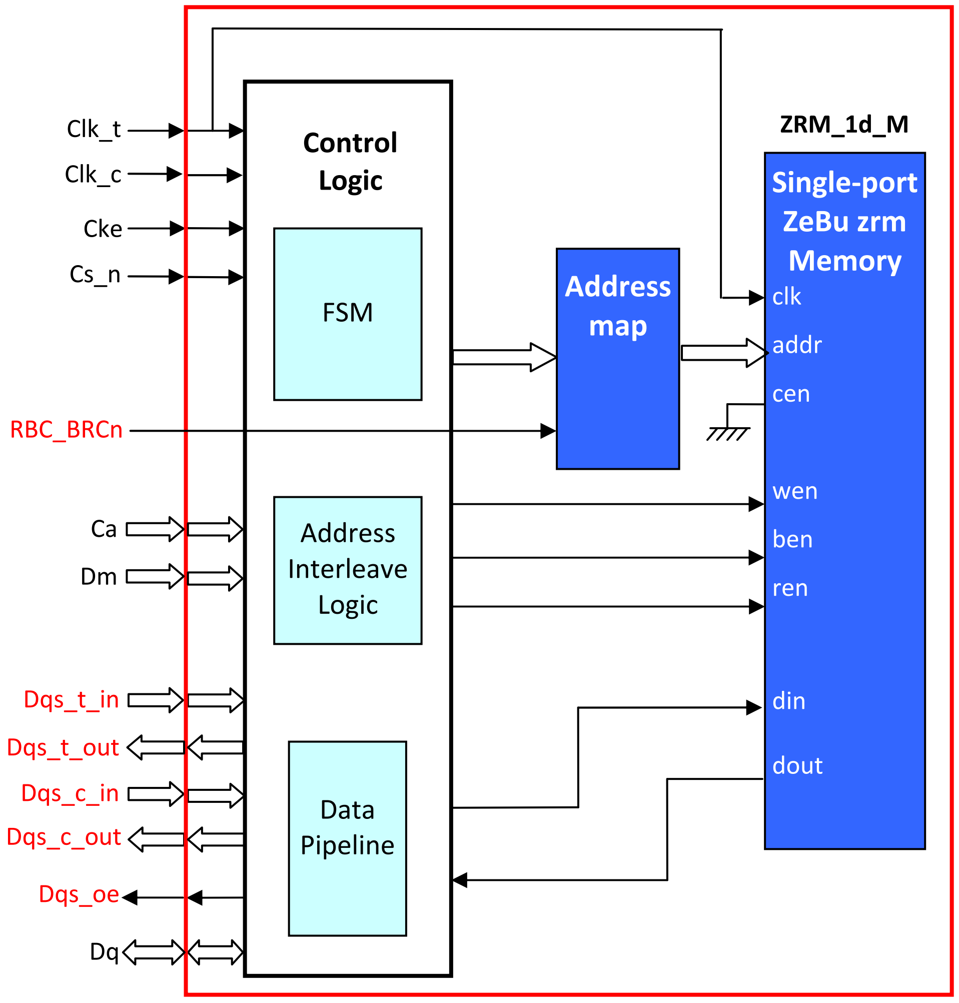
Where:

- `<size>`=`4Gb/8Gb/16Gb`
- `<arch>`=`x16/x32` architecture
- `<model_name>`=any of the memory models listed in Section 1.3.

3 ZLPDDR3 Memory Model Description

3.1 Functional Block Diagram

The ZLPDDR3 synthesizable ZeBu memory models are architectured as follows:



Remark: Signals in red are specific ZLPDDR3 signals that do not exist in the LPDDR3 standard interface (see Section 3.3.2 for further details.).

Figure 1: ZLPDDR3 Model Architecture

3.2 Interfaces of ZLPRR3 Memory Model

The ZLPDDR3 memory Verilog or VHDL model has a unidirectional interface. This model is provided with the ZLPDDR3 Verilog wrapper that allows to use the model with a JEDEC-compliant bi-directional interface.

Both types of interface are described hereafter.

Table 4: ZLPDDR3 Unidirectional Interface

Name	Type	Description
`Clk_t` `Clk_c`	Input	Clock: `Clk_t` and `Clk_c` are differential clock inputs. All Double Data Rate (DDR) `CA` inputs are sampled on both positive and negative edges of `Clk_t`. *`Clk_c (negative)` is not used in the ZLPDDR3 model.*
`Cke`	Input	Clock Enable: `Cke` HIGH activates and `Cke` LOW de-activates internal clock signals and therefore device input buffers and output drivers.
`Cs_n`	Input	Chip Select: `Cs_n` is considered part of the command code.
`Ca`	Input	DDR Command/Address Inputs: Unidirectional command/address bus inputs
`Dq`	I/O	Data Input/Output: Bi-directional data bus.
`Dqs_t_in` `Dqs_c_in`	Input	Data Strobe Input (Differential, `Dqs_t` and `Dqs_c`). It is used to sample WRITE data. `Dqs_t_in` is centered with WRITE data. *`Dqs_c_in (negative)` is not used in the ZLPDDR3 model.*
`Dqs_t_out` `Dqs_c_out`	Output	Data Strobe Output (Differential: `Dqs_t` and `Dqs_c`). It is output with READ data from the memory. `Dqs_t_out` is edge-aligned to READ data. `Dqs_c_out` (negative) is edge-aligned to READ data.
`Dm`	Input	Input Data Mask: `Dm` is the input mask signal for WRITE data. Input data is masked when `Dm` is sampled HIGH. `Dm[0]` is the input data mask signal for the data on `DQ0-7`.
`Dqs_oe`	Output	Output enable signal. Refer to Section 3.3.2 for further details.
`RBC_BRCn`	Input	Addressing mode: it is defined as a parameter (`USER_RBC_BRCn`) for the component instance. The parameter value could be 0 for BRC mode and 1 for RBC mode. Default value is 0.

Note: Grayed rows indicate specific ZLPDDR3 signals that do not exist in the LPDDR3 interface.

Table 5: ZLPDDR3 Bi-directional Interface

Name	Type	Description
`Clk_t` `Clk_c`	Input	Clock: `Clk_t` and `Clk_c` are differential clock inputs. All Double Data Rate (DDR) `CA` inputs are sampled on both positive and negative edges of `Clk_t`. *`Clk_c (negative)` is not used in the ZLPDDR3 model.*
`Cke`	Input	Clock Enable: `Cke` HIGH activates and `Cke` LOW de-activates internal clock signals and therefore device input buffers and output drivers.
`Cs_n`	Input	Chip Select: `Cs_n` is considered part of the command code.
`Ca`	Input	DDR Command/Address Inputs: Unidirectional command/address bus inputs.
`Dq`	I/O	Data Input/Output: Bi-directional data bus.
`Dqs_t` `Dqs_c`	I/O	Data Strobe (Bi-directional, Differential): The data strobe is bi-directional (used for READ and WRITE data) and differential (`Dqs_t` and `Dqs_c`). It is output with READ data and input with WRITE data. `Dqs_t` is edge-aligned to READ data and centered with WRITE data.
`Dm`	Input	Input Data Mask: `Dm` is the input mask signal for WRITE data. Input data is masked when `Dm` is sampled HIGH. `Dm[0]` is the input data mask signal for the data on `DQ0-7`.

3.3 Differences with LPDDR3 Models

3.3.1 LPDRR3 Standard Interface Description

Table 6: LPDDR3 Standard Interface (JEDEC)

Name	Type	Description
`Clk_t` `Clk_c`	Input	Clock: `Clk_t` and `Clk_c` are differential clock inputs. All Double Data Rate (DDR) `CA` inputs are sampled on both positive and negative edges of `Clk_t`.
`Cke`	Input	Clock Enable: `Cke` HIGH activates and `Cke` LOW de-activates internal clock signals and therefore device input buffers and output drivers.
`Cs_n`	Input	Chip Select: `Cs_n` is considered part of the command code.
`Ca`	Input	DDR Command/Address Inputs: Uni-directional command/address bus inputs.
`Dq`	I/O	Data Inputs/Output: Bi-directional data bus.
`Dqs_t` `Dqs_c`	I/O	Data Strobe (Bi-directional, Differential): The data strobe is bi-directional (used for READ and WRITE data) and differential (`Dqs_t` and `Dqs_c`). It is output with READ data and input with WRITE data. `Dqs_t` is edge-aligned to READ data and centered with WRITE data.
`Dm`	Input	Input Data Mask: `Dm` is the input mask signal for WRITE data. Input data is masked when `Dm` is sampled HIGH. `Dm[0]` is the input data mask signal for the data on `DQ0-7`.
`odt`	Input	On Die Termination.

3.3.2 LPDDR3 Interface Modifications for the ZLPDDR3 Unidirectional Model

The `Dqs_t` and `Dqs_c` bi-directional differential ports have been replaced by 4 mono-directional ports: `Dqs_t_in`, `Dqs_t_out`, `Dqs_c_in` and `Dqs_c_out`.

- `Dqs_t_in` and `Dqs_c_in`: inputs to the ZLPDDR3 model
- `Dqs_t_out` and `Dqs_c_out`: outputs from the ZLPDDR3 model

Since the `Dqs` port is used to latch data, this modification was done to avoid gated clocks. For proper use, the original `Dqs` bi-directional signal should be split into four mono-directional ports inside the LPDDR3 controller mapped in your design.

An additional output enable port, `Dqs_oe`, is available to manage the direction of `Dq` and `Dqs` bi-directional signals:

- `Dqs_oe=1` when `Dq` and `Dqs` buses are driven by the memory

Finally, the `odt` input signal is not used in the ZLPDDR3 interface.

Application Note VSAN001, ***Guidelines for the use of ZDDRx Models***, provides more detailed information about this use-model.

3.3.3 LPDDR3 Operations

The ZLPDDR3 model is functionally equivalent to the LPDDR3 memory device, but timing requirements are not applicable. The ZLPDDR3 model is accurate up to a half cycle but cannot take into account setup and hold time for example.

All commands and operating modes (except Write Leveling and CA training) are accepted by the ZLPDDR3 model, including the programming of mode registers defining Read/Write latencies and burst lengths.

Auto-refresh and self-refresh commands are ignored.
Refer to the reference LPDDR3 device datasheets for descriptions of correct operations of the LPDDR3 SDRAM.

3.3.4 LPDDR3 Timing Modeling in ZLPDDR3

The real Read latency seen on the component interface is different from the Read latency value (`RLmrs`) set in the mode registers. The difference is caused by the `DQS` output access time from `Clk_t` (t_{DQSCK}), which is device dependent:

Data Read Latency = `RLmrs` + t_{DQSCK}

In order to model the behavior with DDR cycle accuracy, ZLPDDR3 models contain a specific mode register named `zebuMR[2:0]` (possible values: `0` to 5) to control the t_{DQSCK} timing delay. A t_{DQSCK} unit is equivalent to a ½ cycle of `Clk_t` clock, in other words `0` means a `0 ns` delay and 4 means a delay equivalent to `2*Clk_t` periods.

Programming information for `zebuMR` register is available in Section 3.4.

Example:

With a `Clk_t` running at 333 MHz (3 ns) for the memory device, the t_{DQSCK} must be programmed as `zebuMR[2:0]=3b'001` to model a `tQDSCK` equal to 1.5 ns.
Corresponding Tcl script for modification of the register from **`zRun`**:

```
ZEBU_Signal_write  top.mobileDDR3.zebuMR_0.zebuMR 001 %b
ZEBU_Monitor_flush
```

3.4 Configurable Mode Register

The ZLPDDR3 memory models have an additional register (zebuMR) for runtime control of some key parameters of the LPDRR3 device:

- The value of t_{DQSCK}
- Mode Register 0 (MR0): Write Latency SetA/SetB support
- Mode Register 1 (MR1): Burst Length value
- Mode Register 2 (MR2): Read and Write Latencies

	15 … 8	7 … 4	3	2 … 0
	MR2[7:0]	MR1[3:0]	MR0[6]	t_{DQSCK}
default	0x3	0x3	0x0	0

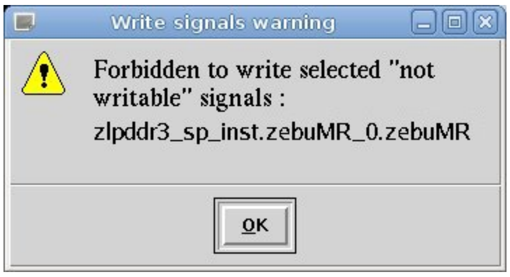
Figure 2: zebuMR mapping with default values

The path to the zebuMR register is:
`<path_to_zlpddr3_inst>.zebuMR_0.zebuMR[15:0]`

Note:
The zebuMR register can be written at runtime only when both of the following conditions are met:

- The **Enable BRAM Read&Write / Write Register / Save&Restore** item in the **Debugging** tab of **zCui** was enabled when compiling the design.
- The ZeBu ZLPDDR3 memory model logic is mapped on an FPGA where there is no RLDRAM instantiated (RLDRAM memories are present on 4C and 8C FPGA modules only).

  When the design instantiates RLDRAM memories, no message is displayed during compilation but the register write operation is not possible at runtime. The following message is displayed at runtime when a register write operation is not possible:



Forbidden to write selected "not writable" signals :
zlpddr3_sp_inst.zebuMR_0.zebuMR

Figure 3: Error message in zRun for forbidden register-write

For a design instantiating RLDRAM memories in a 4C or 8C module, it is highly recommended to map the ZLPDDR3 in the ZeBu memory server FPGA (F4 FPGA for the 4C module, F8 FPGA for the 8C module). For that purpose, manual mapping commands should be added for the design compilation in **zCui**.

3.5 zrm Address Translation

An input, `RBC_BRCn`, is available at the ZLPDDR3 interface to select the zrm memory array addressing mode at compilation time. Two modes are available:

- `BRC` for `{Bank,Row,Column}` addressing
- `RBC` for `{Row,Bank,Column}` addressing

To change zrm addressing:

- `RBC_BRCn = 0` Standard BRC mode
- `RBC_BRCn = 1`

The zrm address is decoded from `Bank`, `Row` and `Column` addressing of LPDDR3. In BRC mode, the starting address for zrm will be transformed into `{Bank, Row, Column}` where `Bank` is the most significant address bit.

For example, if you want to read your memory in a design using the 4Gb(x16) ZLPDDR3 model at `Bank=1`, `Row=1`, `Column=4`, then the starting address for zrm (in hexadecimal) will be as follows:

- BRC Mode: (`RBC_BRCn = 0`)
  `zrm_addr[27:0] = {001,00000000000001,00000000100} = 0xh2000804`
- RBC Mode: (`RBC_BRCn = 1`)
  `zrm_addr[27:0] = {00000000000001,001,00000000100} = 0xh0004804`

4 Integration with the DUT

4.1 Simulation

The ZLPDDR3 package is supplied with a `libIpSimu.so` library in the `libIpSimu` directory for simulation purposes; a gate-level simulation model is also provided in an encrypted format with encryption depending of the target HDL simulator. In the current release, Synopsys VCS and ModelSim models are provided for simulation. For that purpose, you must link with the `libIpSimu.so` library at runtime and enable SystemVerilog construct support in the HDL simulator.

To simulate your design with the ZLPDDR3 component you must compile the provided ZLPDDR3 gate-level simulation model. You must then include the following Xilinx HDL simulation model library in your simulation environment:

- Unisims for ISE 9.1 and later

A gate-level Verilog simulation model of ZLPDDR3 is available at this location:

```
$ZEBU_IP_ROOT/HW_IP/ZLPDDR3.<version>/<size>/<arch>/simu/
```

4.1.1 Synopsys VCS

All ZLPDDR3 models were tested with VCS version `vcs-mx-2006.06-SP2`.

Notes:

- If your design includes SystemVerilog source files, it is recommended to add the lines mentioning ZLPDDR3 at the end of the script:

  ```
  vcs <my_options> <file_list> -sverilog
  $ZEBU_IP_ROOT/HW_IP/ZLPDDR3.<version>/simu/vcs/<model_name>.vp
  $ZEBU_IP_ROOT/HW_IP/ZLPDDR3.<version>/libIpSimu
  /libIpSimu_<XX>.so $ZEBU_ROOT/ise_x_y/verilog/src/glbl.v -y
  $ZEBU_ROOT/ise_x_y/verilog/src/unisims +libext+.v
  ```

  Where <XX> is `32` or `64` according to your environment (32- or 64–bit OS).

- If you need to use several different ZLPDDR3 models, you must compiled them in separate command lines as VCS does not support multiple compilations of the same sub-modules in the same command line. Compiling in a single command line would cause the following error message:

  ```
  Error-[MPD] Module previously declared
  ```

If the license check is successful, you should get the following (according to model):

```
---------- At time 5.0 ps Testing license ----------
      ###############################################
      #         Copyright (c) 2005-2009             #
      # Emulation and Verification Engineering  SA #
      #--------------------------------------------#
      # ZeBu libIpSimu                              #
      # revision : 1.4 64 bit                       #
      # date : Thu 9 2 2012 - 12:50:08              #
      ###############################################
Testing ZLPDDR3 4Gb ZeBu IP license

Checking out ZLPDDR3 4Gb license

---------- At time 5.0 ps check license OK ----------
```

4.1.2 MTI ModelSim

All ZLPDDR3 models were tested with MTI version `modelsim-6.2f`.

```
vlog -sv
 $ZEBU_IP_ROOT/HW_IP/ZLPDDR3.<version>/simu/mti/<model_name>.vp
 $ZEBU_ROOT/ise_x_y/verilog/src/glbl.v -y
 $ZEBU_ROOT/ise_x_y/verilog/src/unisims +libext+.v

vsim -sv_lib
 $ZEBU_IP_ROOT/HW_IP/ZLPDDR3.<version>/libIpSimu/libIpSimu_<XX>
 <my_options>
```

Where <XX> is `32` or `64` according to your environment (32- or 64–bit OS).

If the license check is successful, you should get the following (according to model):

```
# ---------- At time 5.0 ps Testing license ----------
#
#     ##############################################
#     #         Copyright (c) 2005-2009            #
#     # Emulation and Verification Engineering  SA #
#     #------------------------------------------#
#     # ZeBu libIpSimu                             #
#     # revision : 1.4 64 bit                      #
#     # date : Thu 9 2 2012 - 12:50:08             #
#     #------------------------------------------#
#     ##############################################
#
# Testing ZLPDDR3 4Gb ZeBu IP license
#
# Checking out ZLPDDR3 4Gb license
#
# ---------- At time 5.0 ps check license OK ----------
```

4.2 Synthesis

During your design synthesis you must use a blackbox for the ZLPDDR3 component. You can find Verilog and VHDL files for the blackbox components at the following location:

```
$ZEBU_IP_ROOT/HW_IP/ZLPDDR3.<version>/<size>/<arch>/component/
```

4.3 Compilation

To compile the design for use with ZLPDDR3 memory models, you must add the relevant encrypted ZLPDDR3 EDIF netlist in your system-level compiler script. Please find the encrypted netlist you need at the following location:

```
$ZEBU_IP_ROOT/HW_IP/ZLPDDR3.<version>/<size>/<arch>/edif/ <model_name>.edf
```

5 Accessing ZLPDDR3 Models (zrm Load & Dump)

5.1 Running on ZeBu

To access the content of the memory at runtime, you can use the standard way to read from or write to ZeBu memories:

- V6_3_1 patch A_02 or later:
  Use the appropriate hierarchical path to memory:
  ```
  <path_to_zLPDDR3_mem> = <path_to_zlpddr3_inst>.mem_core_logic
  ```
- V6_2_0B_06 or later:
  Use `change_mem` and the same path.
  Read Chapter 7 before you perform a zrm Load&Dump operation.

Example: With V6_3_1 or later, you can initialize a memory in a design using a 4Gb(x16) ZLPDDR3 model with the `memory.init` content file. If the path to the ZLPDDR3 instance is `Top_dut.my_zlpddr3_instance`, then the full path to the zrm memory would be:

```
<pathto_zlpddr3_mem>=Top_dut.my_zlpddr3_instance.mem_core_logic
```

- In a `designFeatures` file (default mode for synthesizable testbenches):
  ```
  $memoryInitDB = "init_mem"
  ```
  where `init_mem` is a file consisting of a collection of lines, with each line listing a memory and the corresponding content file name. In this example, the content of the `init_mem` file is:
  ```
  <path_to_zlpddr3_mem> memory.init
  ```
- In a C++ co-simulation testbench:
  ```
  my_memory = my_board->getMemory("<path_to_zlpddr3_mem>");
  my_memory->loadFrom("memory.init");
  ```
- In a Verilog HDL co-simulation testbench:
  ```
  $ZEBU_readmem("memory.init", "<path_to_zlpddr3_mem>");
  ```

Note: The actual path to memory may differ from version to version.

5.2 Simulating ZLPDDR3 gate-level model with an HDL simulator

In a Verilog simulation testbench, there are two methods for memory initialization. These methods can only be applied to ZLPDDR3 models simulated at gate level.

- The first method uses Verilog system functions to access the physical view of the ZLPDDR3 memory model:

```
$readmemh("data.hex","<path_to_zlpddr3_inst>.
 mem_core_sp.mem", start@, stop@);
$writememh("dumpdata.hex","<path_to_zlpddr3_inst>.
 mem_core_sp.mem", start@, stop@);
```

  Note: With this method, you can load and dump the physical views of memory arrays using multiple files (by calling `$readmemh` and `$writememh` system functions with different file names).

- The second method uses specific Verilog tasks (defined in the encrypted ZLPDDR3 model) to access the logical view of the ZLPDDR3 memory model:
  - `zip_readmemh(startaddr , endaddr)`
  - `zip_writememh(startaddr , endaddr)`

  These tasks do not contain the definitions of the memory array content files. Hence, file names must be defined statically at the top level of the testbench.

  - The file names are specified with the load file (`freadname`) and dump file (`fwritename`) parameters attached to the ZLPDDR3 instance:

```
defparam <path_to_zlpddr3_inst>.mem_core_sp.freadname
  = "zlpddr3_0_memory.init";
defparam <path_to_zlpddr3_inst>.mem_core_sp.fwritename
  = "zlpddr3_0_memory.dump";
```

  - The ZLPDDR3 memory array can then be accessed via its logical view:

```
<path_to_zlpddr3_inst>.mem_core_sp.zip_readmemh
  (start@, stop@);
<path_to_zlpddr3_inst>.mem_core_sp.zip_writememh
  (start@, stop@);
```

  Note: With this method, `freadname` and `fwritename` cannot be changed by the testbench during its execution.

6 Debug Information

For debugging purposes, a list of signals is available at runtime to trace the internal behavior of ZLPDDR3 models. A trace vector called `probe[20:0]` can be accessed with dynamic probes at ZeBu runtime. It contains the necessary information to analyze the behavior of your memory models.

Signal	Description
LPDDR3 Protocol Extension	
`probe[26]`	`setA_B` signal: set to `1` when the model is using the setB WriteLatency values
`probe[25]`	`cmd_BL16` signal: set to `1` when the BL value is equal to 16
`probe[24]`	`cmd_bst` signal: set to `1` when a burst stop operation is in progress
ZLPDDR3 Debug Information	
`probe[22]`	`cmd_CAtraining` signal: set to `1` when `a CA training setup` is received
`probe[21]`	`cmd_writeLvl` signal: set to `1` when `a Write_leveling command` is received
`probe[20]`	`cmd_read` signal: set to `1` when a READ operation is in progress
`probe[19]`	`cmd_write` signal: set to `1` when a WRITE operation is in progress
`probe[18]`	`cmd_mrw` (mode register WRITE)
`probe[17]`	`cmd_mrr` (mode register READ)
`probe[16]`	`cmd_active` signal: set to `1` when `cmd_active` is activated
`probe[15]`	`cmd_precharge` signal: set to `1` when `cmd_precharge` is activated
`probe[14]`	`cmd_refresh` signal: set to `1` when `a Refresh` command is received
`probe[13:11]`	`current_state` of ZLPDDR3 wrapper: Reserved
`probe[10:8]`	Corresponds to MR1[2:0] (burst length)
`probe[7:4]`	Corresponds to MR2[3:0] (read/write latency)
ZLPDDR3 Protocol checker	
`probe[3]`	Set to 1 when row addressing is out of range
`probe[2]`	Set to 1 when column addressing is out of range
`probe[1]`	Set to 1 when the MR2 register is configured on a reserved value
`probe[0]`	Set to 1 when detecting Dqs toggle outside of the expected writing time

In conjunction with this probe vector, the whole interface of the zrm memories is fully visible at runtime to trace the real operations performed on the memory array. The full pathname of the memory to trace should be similar to:

```
<path_to_zlpddr3_mem> = top_dut.my_zlpddr3_instance.mem_core_sp
```

Note: The `probe[]` signals and the zrm memory waveforms provide enough information for efficient support of ZLPDDR3 behavior and integration issues.

7 Appendix: Memory Physical Mapping Translation

Note: As of V6_3_1 patch A_02, the physical mapping translation of the memory IP is handled automatically by the ZeBu compiler. Skip this section if you have V6_3_1 patch A_02 or above.

This section describes the use of `change_mem` in **`zDbPostProc`** to perform a zrm Load&Dump operation, as described in Chapter 5. Please read it beforehand.
The `change_mem` command can only be used with patch V6_2_0B_06 or later for the ZeBu software.

The information below shows how to create a memory map for physical-to-logical memory translation in the ZeBu runtime database.

1. Identify the path to the ZLPDDR3 instance in your design:
   ```
   zDbPostProc –p <zebu.work>
   show –memories -v
   ```
2. In the screen output, look for a memory containing a `mem_core_sp` string:
   ```
   zebu.work]$zDbPostProc -p .

                        ##############################################
                        #                     Copyright (c) 2002-2010 #
                        # Emulation and Verification Engineering  SA #
                        #--------------------------------------------#
                        # zDbPostProc                                #
                        # revision : V6_2_0B_06                      #
                        # date :     25 Mar 2010 - 11:06:23          #
                        #--------------------------------------------#
                        ##############################################

   # zDbPostProc -p .
   # start time is Mon Feb  13 15:39:23 2012


   #   step open DB : Loading ./ZebuDB.zdb
   #   step open DB : Loaded ...

   zDbPostProc [zebu.work]show -memories -v
   top.ins_zlpddr3__0.mem_core_sp 67108864 x 64
   ```
3. From the `zebu.work` directory, issue a `change_mem` command to import and update the ZIP memory in the ZeBu database:
   ```
   zDbPostProc –p <zebu.work>
   change_mem –m <mem_path> -c <scaleFactor>
   save_db
   ```
   Where:
   - `<mem_path>`: path to ZLPDDR3 instance found in previous step
   - `<scaleFactor>`: memory reduction factor (factor 4 for ZLPDDR3)

The screen output should look as follows:

```
zDbPostProc [zebu.work]change_mem -m top.ins_zlpddr3__0.mem_core_sp -c 4

zDbPostProc [zebu.work]show -memories –v

top.ins_zlpddr3__0.mem_core_sp 268435456 x 16

zDbPostProc [zebu.work]save_db
#   step save_db : Saving ./ZebuDB.zdb
#   step save_db : Saved ...
#   step save_db : Saving ZebuPrb.tcl
#   step save_db : Saved ...
#   step save DB : Saving ZebuPrb.lst
#   step save DB : Saved ...
```

Failing to follow the previous steps will lead to incorrect memory locations initialized or dumped at runtime.

8 EVE Contacts

For product support, contact: support@eve-team.com.

For general information, visit our company web-site: http://www.eve-team.com

Europe Headquarters	EVE SA 3, avenue Jeanne Garnerin Air Park Paris Sud 91320 WISSOUS FRANCE Tel: +33-1-64 53 27 30
US Headquarters	EVE USA, Inc. 2290 N. First Street, Suite 304 San Jose, CA 95054 USA Tel: 1-888-7EveUSA (+1-888-738-3872)
Japan Headquarters	Nihon EVE KK KAKiYA Building 4F 2-7-17, Shin-Yokohama Kohoku-ku, Yokohama-shi, Kanagawa 222-0033 JAPAN Tel: +81-45-470-7811
Korea Headquarters	EVE Korea, Inc. 804 Kofomo Tower, 16-3, Sunae-Dong, Bundang-Gu, Sungnam City, Kyunggi-Do, 463-825, KOREA Tel: +82-31-719-8115
India Headquarters	EVE Design Automation Pvt. Ltd. #143, First Floor, Raheja Arcade, 80 Ft. Road, 5th Block, Koramangala Bangalore - 560 095 Karnataka INDIA Tel: +91-80-41460680/30202343
Taiwan Headquarters	EVE-Taiwan Branch Room 806 8F, No. 20 GuanChian Road Taipei, Taiwan 100 Tel: +886 2 2375 9275