ZeBu LPDDR5 Memory Model User Manual

Version V-2024.03-SP1, February 2025

Feedback

Contents

Feedback

1

About This Manual

This manual describes how to use the ZeBu ZLPDDR5 SDRAM synthesizable memory model libraries with optimized memory capacity and performance.

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.

For information about synthesis and compilation for ZeBu, see *ZeBu Compilation Manual* and the ZeBu zFAST Synthesizer Manual.

Keywords and Acronyms

In this manual:

- *<hw_ip_path>* stands for *$ZEBU_IP_ROOT/HW_IP*
- *<ip_path>* stands for *<hw_ip_path>/ZLPDRR5.<version>*
- *<model_path>* stands for *<ip_path>/<size>/<arch>*
- *<xilinx_verilog>* stands for the Xilinx Verilog source. This path depends on the ZeBu platform:
    - ZeBu Server-1, Server-2 and Blade-2: *$ZEBU_ROOT/ise13.4_patched/ISE_DS/ISE*
    - ZeBu Server-3: *$ZEBU_ROOT/ZeBu_Vivado_2014.1/data*
    - ZeBu Server-4: *$ZEBU_ROOT/ZeBU_Vivado_2017.03/data*

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.

Feedback

2

Introduction

The ZeBu ZLPDDR5 synthesizable memory models can be used to model any Synchronous Low Power Double-Data Rate 5 (LPDDR5) DRAM.

The ZLPDDR5 library is provided as a set of IP models, with various densities and architectures listed in Section 1.4, compliant with LPDDR5 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 ZLPDDR5 size and architecture.

This section explains the following topics:

- LPDDR5 Compliance
- ZLPDDR5 Features
- ZLPDDR5 Library
- Logic Resources
- Limitations

LPDDR5 Compliance

The ZLPDDR5 memory models are compatible with the JEDEC LPDDR5 specifications documents JESD209-5B. In addition, ZLPDDR5 is also compatible with LPDDR5X and LPDDR5T (9600Mbps).

ZLPDDR5 Features

The ZeBu ZLPDDR5 SDRAM memory models provide the following features:

- Cycle-accurate SDRAM device model implemented in ZeBu.
- Memory blocks and locations initialization and dump at runtime.
- Includes HDL simulation model for DUT integration testing.

- Bi-directional RTL blackbox components located in the *component* directory.
- Bi-directional Verilog or VHDL RTL wrapper files to include the ZLPDDR5 memory model within the user DUT, located in the *wrapper_rtl* directory.

ZLPDDR5 Library

ZeBu ZLPDDR5 memory models are available for 2Gb, 3Gb, 4Gb, 6Gb, 8Gb, 12Gb, 16Gb, 24Gb, and 32Gb memory capacity components on any compatible ZeBu systems.

For each capacity, the ZLPDDR5 memory component come in x16 architectures. The x8/lower byte architecture is supported in the x16 devices.

The x8/lower byte architecture is supported in the x16 device. It is configured in the zebuMR_common vector. See zebuMR_common Register.

Table 1 ZLPDDR5 Models

Total Memory Density	**Memory Model**
2 Gb	*zLPDDR5_2Gb_x16*
3 Gb	*zLPDDR5_3Gb_x16*
4 Gb	*zLPDDR5_4Gb_x16*
6 Gb	*zLPDDR5_6Gb_x16*
8 Gb	*zLPDDR5_8Gb_x16*
12 Gb	*zLPDDR5_12Gb_x16*
16 Gb	*zLPDDR5_16Gb_x16*
24 Gb	*zLPDDR5_24Gb_x16*
32 Gb	*zLPDDR5_32Gb_x16*

Logic Resources

The ZLPDDR5 synthesizable ZeBu models use the following FPGA resources:

Table 2 Logic Resources Example

Memory Model	Resources
zLPDDR5_2Gb_x16	3132 registers / 4542 LUTs / 1 single-port zrm

Table 2 Logic Resources Example (Continued)

zLPDDR5_32Gb_x16	3216 registers / 4638 LUTs / 1 single-port zrm

Limitations

The following LPDDR5 operations/features are not supported and are ignored by the current ZLPDDR5 models.

- PPR: Post Package Repair
- tWCK2CK = -1/2 tWCK
- Differential signals used in single-ended / negative only (CK_c, WCK_c, RDQS_c single_ended)
- Byte mode upper byte.

3

Installation

This section explains the installation and uninstallation procedure along with the files generated:

- Installation Procedure
- Package Description
- Uninstalling the ZLPDDR5 Package

Installation Procedure

To install the ZLPDDR5 package, perform the following steps:

1. Make sure the *ZEBU_IP_ROOT* environment variable in your shell points to your IP installation directory. Set it accordingly otherwise.
2. Launch the installation script as follows:

   ```
   ./ZLPDDR5.<VERSION>.sh install
   ```

Note:

If the ZEBU_IP_ROOT environment variable is not set, you may launch the installation script as follows:

./ZLPDDR5.<VERSION>.sh install <ZEBU_IP_ROOT>

The installation script executes the following operations:

- It creates the *$ZEBU_IP_ROOT/HW_IP/ZLPDDR5.<VERSION>* directory.
- It extracts all files of the package into this directory.
- It creates a *$ZEBU_IP_ROOT/HW_IP/ZLPDDR5* symbolic link to target the *$ZEBU_IP_ROOT/HW_IP/ZLPDDR5.<VERSION>* directory.
- It unzips the *.gz* simulation library files of the package to the *$ZEBU_IP_ROOT/HW_IP/lib* directory, and symbolic links are created to target the most recent libraries.

Package Description

Once correctly installed, the ZLPDDR5 memory models come with the following elements:

- RTL wrappers to include the component in the DUT as a Verilog RTL source file for synthesis (*wrapper_rtl* directory)
- ZLPDDR5 component blackbox, for design synthesis (*component* directory)
- Verilog RTL-level netlist, for model HDL simulation (*simu/rtl* directory)
- Simulation and emulation examples (*example* directory)
- User scripts (*script* directory)
- *.so* libraries, for model HDL simulation (*lib* directory)

Uninstalling the ZLPDDR5 Package

To uninstall the ZLPDDR5 package, launch the automatic uninstallation script as follows:

```
source ${ZEBU_IP_ROOT}/HW_IP/ZLPDDR5.<VERSION>/uninstall
```

The uninstallation script executes the following operations:

- It removes the *$ZEBU_IP_ROOT/HW_IP/ZLPDDR5.<VERSION>* directory.
- It removes the *$ZEBU_IP_ROOT/HW_IP/ZLPDDR5* symbolic link for this package version.

Feedback

4

ZLPDDR5 Memory Models Description

The ZLPDDR5 synthesizable memory models are internally configured as multi-bank DRAMs and instantiate a ZeBu memory primitive (zrm) to model the DRAM memory array.

The zrm-based memory model used depends on the LPDDR5 size and architecture (see ZLPDDR5 Library).

This section explains the following topics:

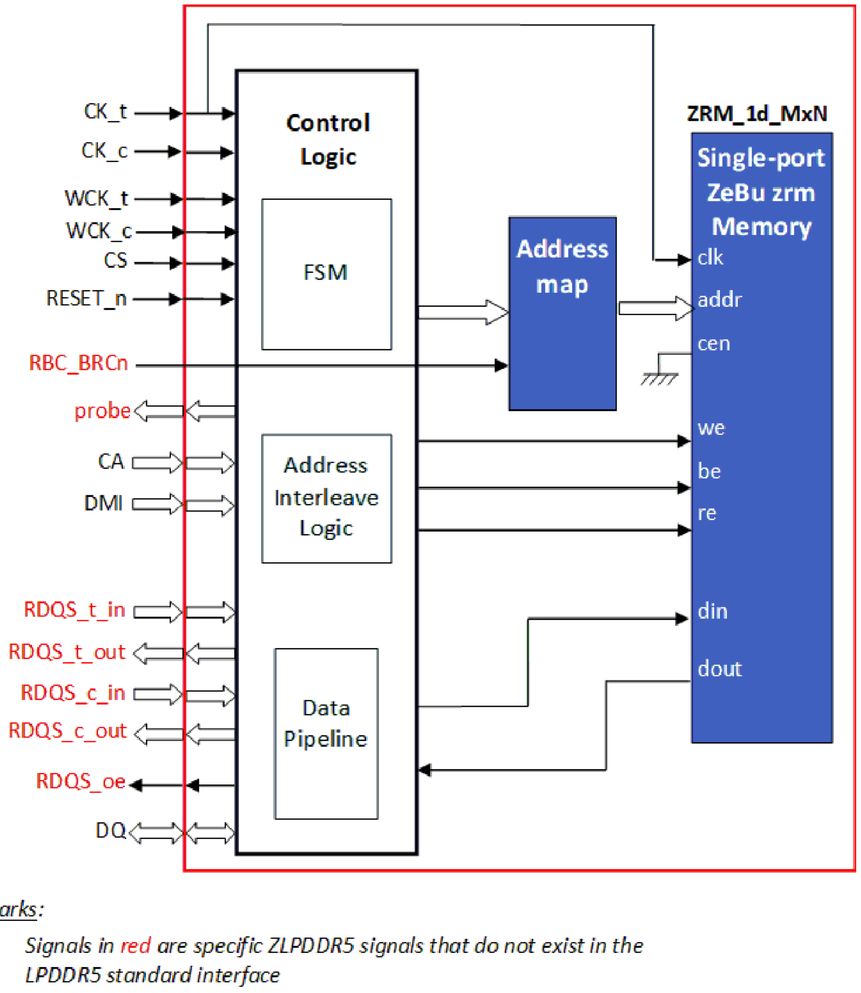
Overview

The ZLPDDR5 synthesizable memory models can be used to model any Synchronous Low Power Double-Data Rate 5 (LPDDR5) DRAM, using zrm memory resources. Such zrm-based memory models provide the usual ZeBu hardware debugging features such as runtime memory upload/download and memory cell READ/WRITE.

Functional Block Diagram

The ZLPDDR5 synthesizable ZeBu memory model is architectured as follows:

Figure 1 ZLPDDR5 Model Architecture



Interfaces of ZLPDDR5 Memory Model

The ZLPDDR5 memory interface consists of the ZLPDDR5 Verilog or VHDL wrapper that allows using the model with a JEDEC-compliant bi-directional interface. See Setting the RTL Wrapper in the Compilation Project how to use the wrapper.

The following table describes the uni-directional ZLPDDR5 interfaces. The gray rows indicate specific ZLPDDR5 signals that do not exist in the LPDDR5 standard interface.

Table 3 ZLPDDR5 Unidirectional Interface

Name	**Type**	**Description**
CK_tCK_c	Input	*Clock*: CK_t and CK_c_ are differential clock inputs. All address command and control input signals are sampled on positive edges of CK_t.CK_c (negative) is not used in the ZLPDDR5 model.
WCK_tWCK_c	Input	WCK_t and WCK_c are differential clocks used for WRITE data capture and READ data output.
CS	Input	*Chip Select*: is considered part of the command code.
RESET_n	Input	*Reset*: when asserted LOW, this resets the device.
CA	Input	Command/Address Inputs
DMI	I/O	Data Mask (DM) or Data Bus Inversion (DBI) according to the mode register configuration
DQ	I/O	Data Input/Output: Bi-directional data bus.
RDQS_t_inRDQS_c_in	Input	Data Strobe Input (Differential, RDQS_t and RDQS_c). RDQS_c_in is not used in the ZLPDDR5 model.
RDQS_t_outRDQS_c_out	Output	Data Strobe Output (Differential: R*DQS_t* and R*DQS_c*. It is output with READ data from the memory.*RDQS_t_out* is edge-aligned to READ data.*RDQS_c_out* (negative) is edge-aligned to READ data.
RDQS_oeRDQS_oe	Output	RDQS output enable signal.Refer to Section 3.4.1 for further details.
probe	Output	Debug probe.Please refer to Chapter 6 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.For more information, see zrm Address Translation.

The following table describes the bidirectional ZLPDDR5 interfaces.

Table 4 ZLPDDR5 Bi-directional Interface

Name	Type	Description
CK_tCK_c	Input	Clock: *CK_t* and *CK_c* are differential clock inputs. All address command and control input signals are sampled on positive edges of *CK_t*. *CK_c* (negative) is not used in the ZLPDDR5 model.
WCK_tWCK_c	Input	WCK_t and WCK_c are differential clocks used for WRITE data capture and READ data output.
CS	Input	Chip Select: is considered part of the command code.
RESET_n	Input	This resets the memory.
CA	Input	Command/Address Inputs.
DQ	I/O	Data Input/Output: Bi-directional data bus.
RDQS_tRDQS_c	I/O	Data Strobe (Bi-directional, Differential): The data strobe is bi-directional (used for READ and WRITE data) and differential (*RDQS_t* and *RDQS_c*).It is output with READ data and input with WRITE data. *RDQS_t* is edge-aligned to READ data and centered with WRITE data.
DMI	I/O	Data Mask(DM) and/or Data Bus Inversion (DBI) according to the mode register configuration for READ/WRITE operations.
ZQ	Input	Calibration reference: Used to calibrate the output drive strength and termination resistance.ZQ is not used in the ZLPDDR5 model

Note:

All differential signals are modeled as single-ended signals. Therefore on all differential signals available at a component's pinout, only xxx_t signals are really connected inside the memory model.

Key Differentiators from LPDDR5 SDRAM Models

This section explains the following key differences in ZLPDDR5 memory models with respect to the LPDDR5 SDRAM memory models:

- LPDDR5 Device Interface Modifications
- LPDDR5 Operations
- LPDDR5 Timing Modeling in ZLPDDR5

LPDDR5 Device Interface Modifications

RDQS_t_in/RDQS_c_in and RDQS_t_out/RDQS_c_out Ports

The *RDQS_t* and *RDQS_c* bi-directional differential ports have been replaced by 4 unidirectional ports:

- *RDQS_t_in* and *RDQS_c_in*: inputs to the ZLPDDR5 model
- *RDQS_t_out* and *RDQS_c_out*: outputs from the ZLPDDR5 model

Since the R*DQS* port is used to latch data, this modification allows avoiding gated clocks. For proper use, the original *RDQS* bidirectional signal should be split into four unidirectional ports inside the LPDDR5 controller mapped in your design.

LPDDR5 Operations

The ZLPDDR5 model is functionally equivalent to the LPDDR5 memory device, but timing requirements are not applicable. The ZLPDDR5 model is accurate up to a half cycle but cannot take into account setup and hold time for example.

All commands and operating modes are accepted by the ZLPDDR5 model, including the programming of mode registers defining Read/Write latencies and burst lengths.

Refresh and self-refresh commands are ignored.

See reference LPDDR5 device datasheets for descriptions of correct operations of the LPDDR5 SDRAM.

LPDDR5 Timing Modeling in ZLPDDR5

Read Operations

The real Read latency seen on the component interface can be different from the Read latency value (*RLmrs*) set in the mode registers. The difference is caused by the *RDQS* output access time from *CK_t* (*tDQS2DQO*), which is device-dependent:

Data Read Latency = RLmrs + tDQS2DQO

In order to model the behavior with DDR cycle accuracy, ZLPDDR5 models contain a specific mode register named *zebuReg[39:36]* (possible values: *0* to *15*) to control the *tDQS2DQO* timing delay. A *tDQS2DQO* unit is equivalent to a half-cycle of W*CK_t clock, in other words 0* means a *0 ns* delay and *3* means a delay equivalent to *1.5*WCK_t* periods.

Example:

With a W*CK_t* running at 800 MHz (1.25 ns) for the memory device, the *tDQS2DQO* must be programmed as *zebuReg[39:36]=4h'3* to model a *tDQS2DQO* equal to 1.875 ns.

Corresponding Tcl script for modification of the register from *zRci*:

```
force top.zlpDDR5.rank_0_ins_zebuMR.zebuReg 0x3XXXX -radix hexa -deposit
force top.zlpDDR5.rank_1_ins_zebuMR.zebuReg 0x3XXXX -radix hexa -deposit
```

Write operations

The LPDDR5 uses an unmatched *RDQS DQ* path for lower power, so the first *DQ* data seen on the component interface is different from the Write latency value (*WLmrs*) set in the mode registers. The difference is caused by the *RDQS* to *DQ* delay time (*tDQS2DQI*), which is device-dependent:

Data Write Latency = *WLmrs* + *tDQS2DQI*

In order to model the behavior with DDR cycle accuracy, ZLPDDR5 models contain a specific mode register named *zebuReg[35:32]* (possible values: *0* to *15*) to control the *tDQS2DQI* timing delay. This indicates first valid edge (rising or falling) of W*CK_t* for sampling first data *DQ*.

A *tDQS2DQI* unit is equivalent to a half-cycle of W*CK_t* clock, in other words 0 means a *0 ns* delay and *4* means a delay equivalent to *2*WCK_t* periods.

Example:

With a W*CK_t* running at 800 MHz (1.25 ns) for the memory device, the *tDQS2DQI* must be programmed as *zebuReg[35:32]=3b'001* to model a *tDQS2DQI* equal to 625 ps.

Corresponding Tcl script for modification of the register from *zRci*:

```
force top.lpDDR4.rank_0_ins_zebuMR.zebuReg 0x1XXXXX -radix hexa -deposit
force top.lpDDR4.rank_1_ins_zebuMR.zebuReg 0x1XXXXX -radix hexa -deposit
```

tWCK2CK

This delay is the possible delay between CK and WCK. This delay can be positive or negative. In 1:2 clock ratio tWCK2CK must remain strictly between -0.25 tWCK and +0.25 tWCK. Any delay in this space is natively supported by the memory without any action from the user

In 1:4 clock ratio tWCK2CK must remain strictly between -0.5 tWCK and +0.5 tWCK.

For delays between -025 tWCK and +0.25tWCK, no action is required from the user as the shift between CK and WCK is natively supported by the model.

In order to model the behavior with DDR cycle accuracy, ZLPDDR5 models contain a specific mode register named *zebuReg[31:30]* (possible values: 2'b0*0, 2'b01,2'b10. 2'b11 is a reserved value*) to control the *tWCK2DQ* timing delay.

For -0.5*tWCK < tWCK2CK < -0.25 tWCK use zebuReg[31:30] = 2'b10;

Note:

ZebuReg[31:30] = 2'b10 is not yet supported by the model.

For -0.25*tWCK < tWCK2CK < +0.25 tWCK use zebuReg[31:30] = 2'b00;

For +0.25*tWCK < tWCK2CK < +0.5 tWCK use zebuReg[31:30] = 2'b01;

Configurable Mode Register

The ZLPDDR5 memory models have a common register (*zebuMR_common*) and a specific register (zebuMR) for each channel. This architecture is common to all ZeBu memory IPs. However, ZeBu LPDDR5 memory models have only one channel.

zebuMR_common Register

Each instance of the ZLPDDR5 model has a *zebuMR_common* register of this instance. It is divided into several Mode Registers (*MR*), each one corresponding to the specific information.

The following figure describes the common *zebuMR_common* register content:

Table 5 zebuMR_common Mapping with Default Values

Bits	*63:28*	*27*	*26*			*31 26 25 24 23 16 15 8 7 0*	
MR				MR8[7:6]	MR7[7:0]	MR6[7:0]	MR5[7:0]
Default Value		0		0	0	0x0	0xff
Information	RFU	Analyzer display On/Off	RFU	Type	Revision ID2	Revision ID1	Manufacturer ID

The following is the path to the *zebuMR_common* register is:

```
<path_to_zLPDDR5_inst>.ins_zebuMR_common.zebuReg[23:0]
```

zeBuMR Register

Each instance of the ZLPDDR5 model has one *zebuMR* register.

The *zebuMR* register is divided into several Mode Registers (*MR*), each one corresponding to specific information.

The following table describes the *zebuMR* register content:

Table 6 specific zebuMR mapping with default values

bits	127-57	56-55	54-40	369:36	35:32	31:30	29	28	27	26	25					3 0
		MR20[1:0]		tWCK2DQO	tWCK2DQI	tWCK2CK		MR22[6]	MR22[4]	MR18[7]	MR13[5]	MR10[7:4]	MR4[7:0]	MR3[7:3]	MR2[3:0]	MR1[7:4]
default value		0		0	0	0		0	0	0	0	0	0x9	0x0	0x0	0x0
information	RFU	Read DQS	RFU				R	R ECC	W ECC	CKR	DM	RDQS_PST , RDQS_PRE	Refresh Rate	DBI, WLS, BK/BG ORG	Read Latency	Write Latency

The tWCK2DQO, tWCK2DQI and tWCK2CK delays are not supported by the zDFI. When using with zDFI, do not change any default timing in ZLPDDR5.

The mode registers with multiple frequency set points are all initialized to the same value. However, multiple frequency set points (FSP) are not supported in the zDFI.

The path to the *zebuMR* register is *<path_to_zLPDDr5_inst>.ins_zebuMR.zebuReg[127:0]*

zrm Address Translation

The memory space of the LPDDR5 device is three-dimensional and is organized in banks, rows and columns. In the ZLPDDR5 memory model, the memory space is flat (bank*row*column depth array of words).

The zrm address is decoded from *Bank*, *Row* and *Column* addressing of LPDDR5.

The *RBC_BRCn* input is available at the ZLPDDR5 interface to select the zrm memory array addressing mode at compilation time. The following modes are available:

- BRC for {Bank,Row,Column} addressing: In BRC mode, the starting address for zrm will be transformed into {Bank, Row, Column} where Bank is the most significant address bit
- RBC for {Row,Bank,Column} addressing: In RBC mode, the starting address for zrm will be transformed into {Row, Bank, Column} where Row is the most significant address bit.
- Bank is defined as:
    - BK[2:0] in 8 bank mode,
    - BK[3:0] in 16 bank mode,
    - {BG[1:0],BK[1:0]} in bank group mode.
- The Row vector size depends on the memory configuration.
- Column is defined as
    - {C[5:0],B4,B3,3'b000} in 8 bank mode,
    - {C[5:0],B3,3'b000} in 16bank or bank group mode.

To change zrm addressing:

- *RBC_BRCn = 0* for *BRC* mode (default)
- *RBC_BRCn = 1* for *RBC* mode

For Binary Density Memory Models (2Gb, 4Gb, 8Gb, 16Gb, 32 Gb Memory Models)

For example if you want to read your memory in a design using the 2Gbx16, 16 bank mode ZLPDDR5 model:

- *Bank=0x1 (4 bits)*
- *Row = 0x2 (13 bits for 2Gb)*
- *Col = 0x40 (10 bits = {C5-C0, B3, 3'b000})*

then the starting address for zrm (in hexadecimal) will be as follows:

- *BRC Mode*:

zrm_addr[26:0] = {4'b0001,13'b0000000000010,10'b0001000000} = 27'h0800840

- *RBC Mode*:

zrm_addr[26:0] = {13'b0000000000010,4'b0001,10'b0001000000} = 27'h0008440

However, if you want to read your memory in a design using the 8Gbx16, 8 bank mode ZLPDDR5 model:

- Bank=0x7 (3 bits)
- Row = 0x20A (15 bits for 2Gb)
- Col = 0x60 (11 bits = {C5-C0, B4, B3, 3'b000})

then the starting address for zrm (in hexadecimal) will be as follows:

- BRC mode:

zrm_addr[28:0] = {3'b1 11,15'b00 0001 0000 0101 0,11'b000 0110 0000} = 29'h13105060

- *RBC mode*:

zrm_addr[28:0] = {15'b0 0000 1000 0010 10,3'b11 1,11'b000 0110 0000} = 29'h0082B860

For Non-Binary Density Memory Models (3 Gb, 6 Gb, 12 Gb, 24 Gb Memory Models)

The 6Gb, 12Gb and 24Gb ZeBu ZLPDDR5 memory models are non-binary density devices. As a consequence, only three quarters of the row address space is valid. When the MSB of *row* address bit is HIGH, the MSB-1 address bit must be LOW.

This has no impact in RBC mode. However in BRC mode, the zrm address will be converted to have a linear addressing as follows:

$$\begin{aligned} zrm\ address = {} & (bank\_addr \times MAX\_ROW\_SIZE \times MAX\_COL\_SIZE) \\ & + (row\_addr \times MAX\_COL\_SIZE) + column\_addr \end{aligned}$$

Example

Assume that you want to read your memory in a design using the 6Gbx16 16 bank mode, ZLPDDR5 model with:

- *Bank=1 (4 bits)*
- *Row=2 (15 bits)*
- *Column=4 (7 bits)*

- *MAX_ROW_SIZE* = 2^15*3/4 = 24576 for this model
- *MAX_COL_SIZE* = 2^7 = 128 for this model

In this case, the starting address for zrm (in hexadecimal) should be as follows:

- *BRC Mode*:

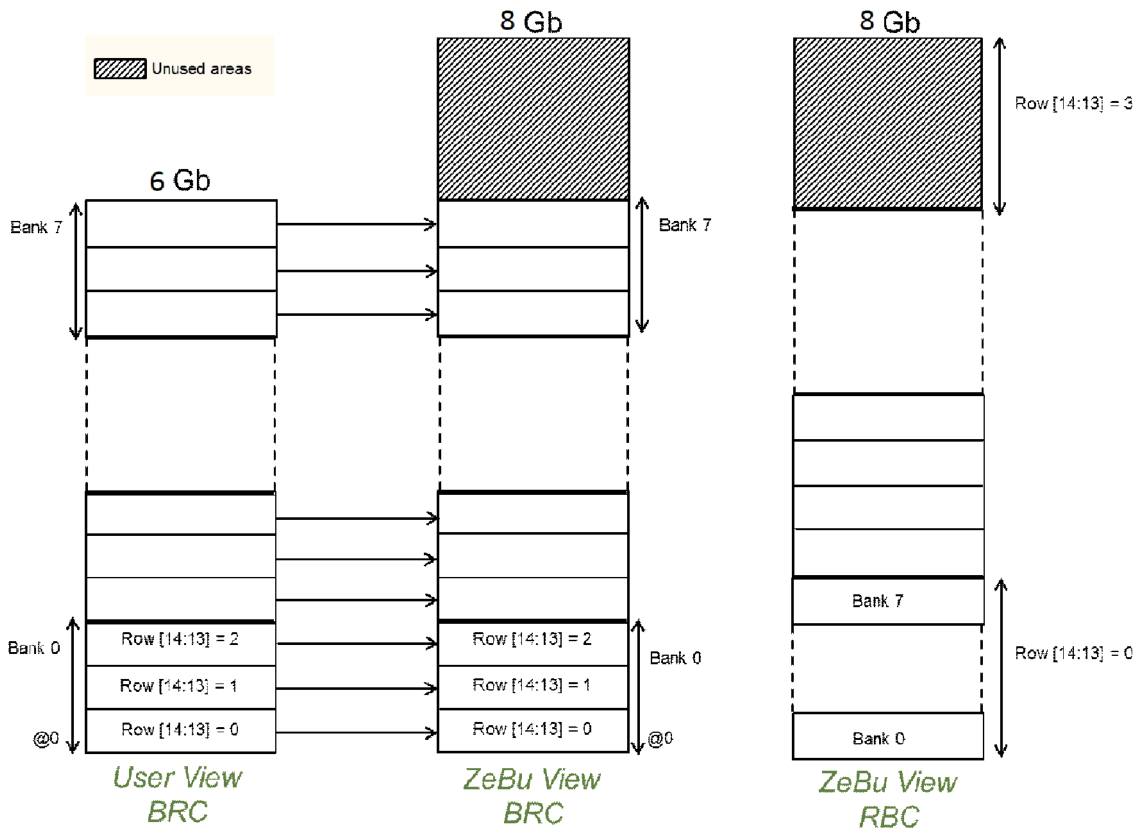
```
zrm_addr[25:0] = (1*24576*128) + 2*128 +4 = 26'h300104
```

- *RBC Mode*:

```
zrm_addr[25:0] = {000000000000010,0001,0000100} = 16'h0001084
```

The following figure illustrates a 6Gb ZLPDDR5 Models with 15-bit Row Address

Figure 2 6Gb ZLPDDR5 Models with 15-bit Row Address



5

Integration with the DUT

This section describes how to integrate the ZLPDDR5 memory model with the DUT, under the following sections:

- Prerequisite
- Setting the RTL Wrapper in the Compilation Project
- Synthesizing and Compiling for ZeBu
- Simulation

Prerequisite

You should first have properly set the RTL wrapper in your compilation project, as described in Setting the RTL Wrapper in the Compilation Project.

Setting the RTL Wrapper in the Compilation Project

Purpose of the RTL Wrapper

The ZLPDDR5 memory has a unidirectional interface with additional ports which are not part of the JEDEC standard. Then, in order to substitute a standard DRAM memory with a JEDEC-compliant bidirectional interface, the ZLPDDR5 memory model should be integrated with the DUT using a dedicated RTL wrapper.

The RTL wrapper for bidirectional *RDQS* signal has the following parameters:

- *USER_RBC_BRCn*: addressing mode. The parameter value should be 0 for BRC mode and 1 for RBC mode. Default value is 0.
- *USER_RDQS_DELAY*: additional delay on *RDQS* signal.

Wrappers and Associated Model Files Locations

Source code files for RTL wrappers of the ZLPDDR5 interface are available at *<model_path>/wrapper_rtl*.

Associated blackbox description Verilog and VHDL files are available at *<model_path>/ component*: *<model_name>_bidir.<v/vhd>*.

Synthesizing and Compiling for ZeBu

Use the following commands to add the RTL bidirectional wrapper and the RTL model to an existing zCui compilation UTF project.

In VCS compilation command, add following option and files:

```
+define+ZEBU_SYNTH
$ZEBU_IP_ROOT/HW_IP/wrapper_rtl/<model_name>_bidir.v
$ZEBU_IP_ROOT/HW_IP/vlog/vcs/<model_name>.vp
```

Alternatively, you can use -y VCS option, as shown below:

```
+define+ZEBU_SYNTH +libext+.v+.vp+.sv
-y $ZEBU_IP_ROOT/HW_IP/wrapper_rtl
-y $ZEBU_IP_ROOT/HW_IP/vlog/vcs
```

To enable the DPI synthesis, if analyzer is needed, use the following command:

dpi_synthesis -enable ALL

For ZS5/EP1 following additional define need to be added in vcs command line to ensure the ZRM latency encoding matching with ZeBu firmware:

```
+define+ZRM_NEW_LATENCY
```

Simulation

The ZLPDDR5 package is supplied with:

- a *libIpSimu_<32/64>.so* library in the *<hw_ip_path>/lib* directory for simulation purposes
- an RTL-level Verilog simulation model in an encrypted format with encryption depending on the target HDL simulator. It is available in the *<model_path>/simu/rtl* directory.

  Note:

  Do not use +define+ZEBU_SYNTH in simulation mode, the model is not functional.

Simulation with RTL-Level Model

To simulate your design with the ZLPDDR5 component, compile the provided ZLPDDR5 RTL-level simulation model found in *<model_path>/simu/rtl*.

RTL-Level Simulation with Synopsys VCS

If your design includes SystemVerilog source files, it is recommended to add the following lines mentioning ZLPDDR5 at the end of the script:

```
vcs <my_options> <file_list> -sverilog
 <model_path>/simu/rtl/vcs/<model_name>.vp
 <hw_ip_path>/lib/libIpSimu /libIpSimu_<32/64>.so
```

If you need to use several different ZLPDDR5 models, you should compile each one separately as VCS does not support multiple compilations of the same sub-modules in the same command line.

Example:

```
 <model_path>/simu/rtl/vcs/<model_name>.vp

 <model_path>/simu/rtl/vcs/<model_name>.vp

vcs <my_options> <my_top_level_name>
 <hw_ip_path>/lib/libIpSimu_<32/64>.so
```

Otherwise, you would get the following error message:

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

Result of ZeBu IP License Checking

For simulation with VCS, you should get the following (according to the model) when the license check is successful:

```
---------- At time 5.0 ps Testing license ----------
##############################################
#         Copyright (c) 2005-2014            #
# Emulation and Verification Engineering  SA #
#--------------------------------------------#
# ZeBu libIpSimu                             #
# revision : 2.2 64 bit                      #
# date : Fri 28 3 2014 - 12:50:08            #
##############################################
Testing ZLPDDR5 8192Mb ZeBu IP license
```

```
Checking out ZLPDDR5 8192Mb license

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

Otherwise, contact your local representative.

Feedback

6

Accessing ZLPDDR5 Models (zrm Load & Dump)

To access the content of the memory at runtime, use the standard way to read from or write to ZeBu memories with its appropriate hierarchical path. This path to the memory is like:

<path_to_zlpddr5_mem>=<path_to_zlpddr5_inst>.mem_core_sp.mem_logic

This section explains the following topics:

- Example
- Initializing Memories for Simulation

Example

You can initialize a memory in a design instancing a LPDDR5 model using the 'memory.init' content file.

If the path to the zLPDDR5 instance is 'top_dut.my_zlpddr5_inst', then the full path to the ZRM memory would be as follows:

```
<path_to_zlpddr5_mem>=top_dut.my_zlpddr5_inst.mem_core_sp.mem_logic
```

Therefore, specify the following line 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 path and the corresponding content file name. In this example, the content of the init_mem file is:

In a C++ co-simulation testbench, specify the following:

```
my_memory = my_board->getMemory("<path_to_zlpddr5_mem>");
 my_memory->loadFrom("memory.init");
```

With zRci tcl backdoor access:

memory -load "<path_to_zlpddr5_mem>" -file data_load.hex -radix hexa

memory -store "<path_to_zlpddr5_mem>" -file data_dump.hex -radix hexa

Initializing Memories for Simulation

In a Verilog simulation testbench, the following methods are used for memory initialization.

These methods can only be applied to zLPDDR5 models simulated at RTL level.

The following message displayed at the beginning of the simulation will be helpful to retrieve the exact memory path:

```
---------- The logical memory array path is
 tb.ins_zlpddr5_tb.zip0.zlpddr5.zlpddr5.mem_core_sp.mem_logic (width =
 16, depth = 536870912, size = 8Gb)
```

Using Verilog System Tasks

Verilog system tasks, $readmemh and $writememh are used to initialize and dump memory data respectively.

- $readmemh system task reads address and corresponding data (in hexadecimal format) from a file and loads it into the memory.

  ```
  $readmemh("load_data.hex",
  <path_to_zlpddr5_inst>.mem_core_sp.mem_logic[start@,stop@]);
  ```

- $writememh system task dumps address and corresponding data (in hexadecimal format) to a file from the memory.

  ```
  $writememh("dump_data.hex",<path_to_zlpddr5_inst>.mem_core_sp.mem
   _logic[start@,stop@]);
  ```

Using Specific Verilog Tasks

Specific Verilog tasks are used to initialize the memory array to all 0 or 1:

```
<path_to_zlpddr5_inst>.mem_core_sp.zip_init_mem_all(0|1)
```

7

Debug Information

For debugging purpose, a list of signals is available at runtime to trace the internal behavior of ZLPDDR5 models. There are two vectors:

- probe, showing various internal signals
- probe_zrm, showing the zrm interface

Probe Signals

A trace vector called, probe[91:0] is, available on interface and you can access it with dynamic probes at ZeBu runtime. It contains the necessary information to analyze the behavior of your memory models:

Table 7 Probe Signals

Signal	Description
probe[91]	read data copy enabled
probe[90]	write data copy enabled
probe[89]	zrm_latency_encoding
probe[88]	cfg_rd_wr_rdqs_training(MR[26][7])
probe[87]	cfg_cbt_training(|MR[16][5:4])
probe[86]	cfg_wr_leveling(MR[18][6])
probe[85]	cfg_rdqs_toggle_mode(|MR[46][1:0])
probe[84:82]	CBT phase, write ecc enable,read ecc enable
probe[81:66]	Bank activated
probe[65]	Reserve
probe[64]	Illegal commands detected and ignored
probe[63]	Illegal burst length 32 in current bank mode

Table 7 Probe Signals (Continued)

probe[62]	Reserve
probe[61]	column address bit (B3, C0 in 8B mode) high (will be ignored)
probe [60]	command unsupported
probe[59:52]	Reserve
probe[51:46]	Read latency
probe[45:40]	Write latency
probe[39:36]	tWCQ2DQO
probe[35:32]	tWCK2DQI
probe[31]	Burst length 32
probe[30:29]	fsp_op
probe[28:27]	fsp_wr
probe[26]	dm_disable
probe[25]	dbi_wr
probe[24]	dbi_rd
probe[23:22]	rd_pre(MR[10][5:4])
probe[21:20]	rd_post (MR[10][7:6])
probe[19]	8 BK mode
probe[18]	16 BK mode
probe[17]	4BK/4BG mode
probe[16]	wls_set_B
probe[15]	cbt mode
probe[14]	Clock ratio 1:2
probe[13]	command READ (all types)
probe[12]	Command WRITE (all types)
probe[11]	Command ACT (ACT1 + ACT2 sequence)

Table 7 Probe Signals (Continued)

probe[10]	Command MRR
probe[9]	Command MRW
probe[8]	Command PRE
probe[7]	Command MPC
probe[6]	Command RFF
probe[5]	Command WFF
probe[4]	Command RDC
probe[3:0]	Current state of ZLPDDR5IDLE = 0MRR = 1MRW = 2ACT =3 (ACT means ACT2 following ACT1, no specific state for ACT1)READ = 4WRITE = 5TRAIN = 6 (CBT or WR leveling)RDC = 7RFF = 8WFF = 9

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_zLPDDR5_mem> = top_dut.my_zLPDDR5_ins.mem_core_sp.mem_logic
```

Note:

The probe[] signals and the zrm memory waveforms provide enough information for efficient support of ZLPDDR5 behavior and integration issues.

Alias Files for Verdi™

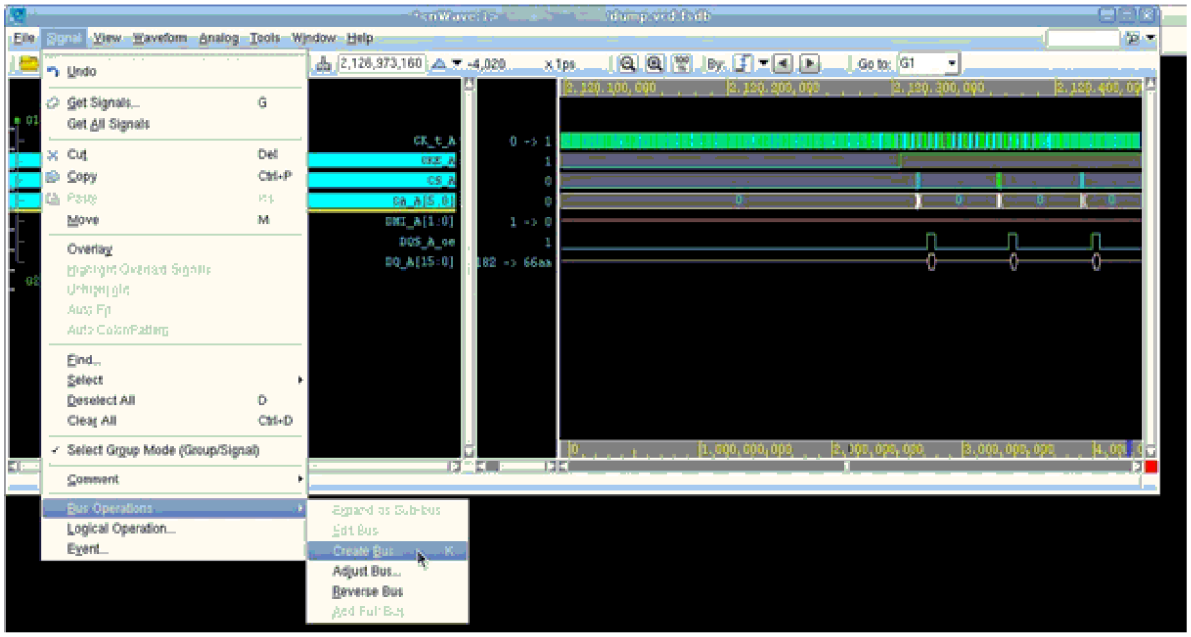
The ZLPDDR5 package provides the following two alias files in the *script/nWave* directory to facilitate debug in Verdi:

- *cmd.alias*
- *probe.alias*
- probe_zrm.tcl

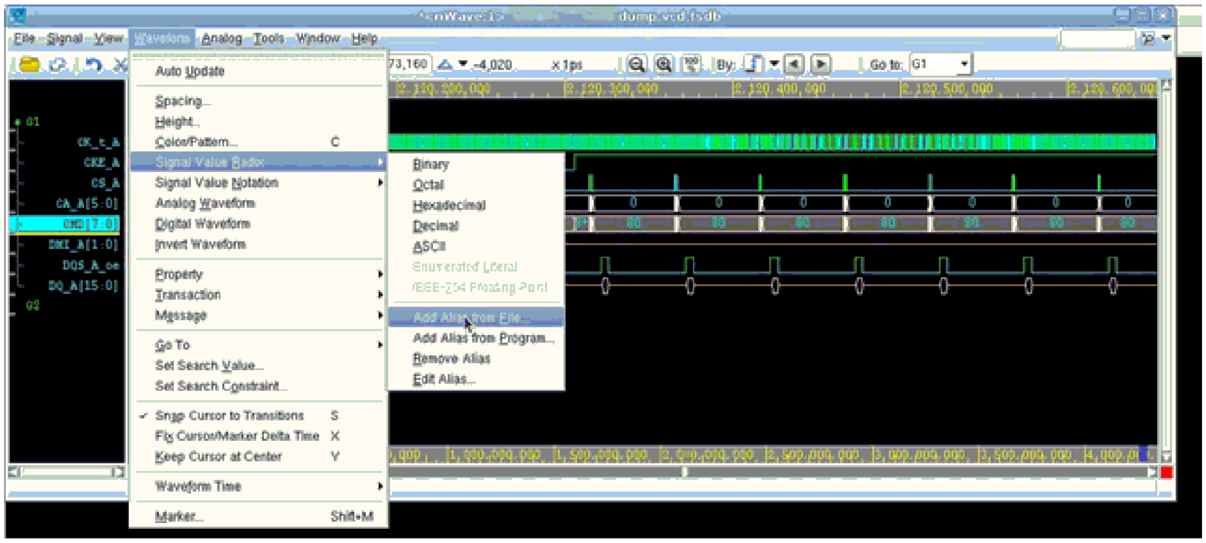
cmd.alias

To use the cmd.alias file, perform the following steps:

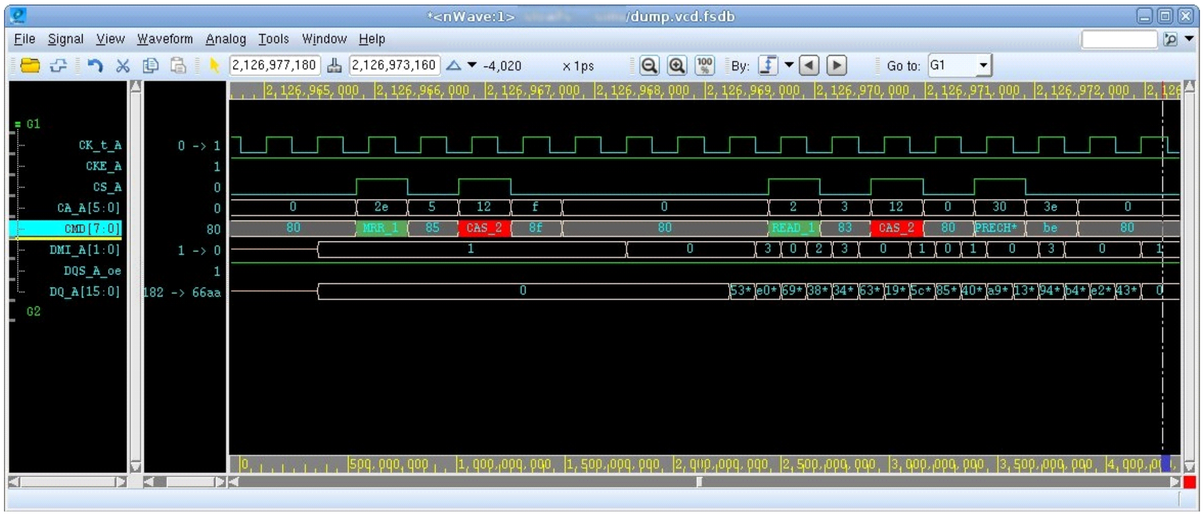
1. In *nWave*, select *Signal* > *Bus Operations* > *Create Bus* and create a new 8-bit width bus:
   - from CK_t, *CS* and *CA[6:0]* in MSB to LSB
   - name it *CMD* for example



1. Select the created bus vector (named *CMD* in the previous step).
2. Assign the *cmd.alias* file to it by selecting *Waveforms* > *Signal Value Radix* > *Add Alias from File*:



The following figure should get a display result similar to the one below:

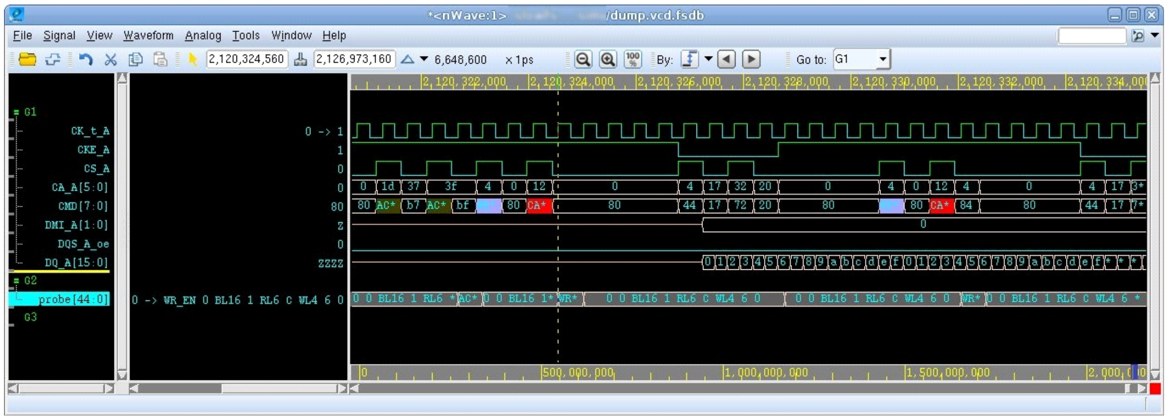


probe.alias

To use the probe.alias file, perform the following steps:

1. In *nWave*, select *Signal* > *Bus Operations* > *Create Bus* and create a new 45-bit width bus from *probe[76:0]*:
2. Select the created bus vector.
3. Assign the
4. *probe.alias* file to it by selecting *Waveforms* > *Signal Value Radix* > Add Alias from File.

You should get a display result similar to the one below:


File Signal View Waveform Analog Tools Window Help
G1
CK_t_A
CKE_A
CS_A
CA_A[5:0]
CMD[7:0]
DMI_A[1:0]
DQS_A_oe
DQ_A[15:0]
G2
probe[44:0]
G3
0 -> WR_EN 0 BL16 1 RL6 C WL4 6 0

probe_zrm.tcl

This probe gives access to the ZRM interface. To speed up the memory model, a folding mechanism concatenates and reads/writes the chunk of 8 (for x16bit mode) or 16 (for x8bit mode) consecutive words in the zrm at a single location.

The results is

zrm_address_size = zLPDDR5_address_size-3.

zrm_data_width = (8*16 bits OR 16*8 bits) =128 bits)

The following table lists the zrm_probe signals:

Table 8 zrm_probe signals

ZRM signal	size	zrm_probe
mem_dout_ack	1	zrm_probe[306]
mem_be	16	zrm_probe[305:290]

Table 8 zrm_probe signals (Continued)

mem_address	32	zrm_probe[289:258]
mem_we	1	zrm_probe[257]
mem_din	128	zrm_probe[256:129]
mem_re	1	zrm_probe[128]
mem_dout	128	zrm_probe[127:0]

To make it easier to use the zrm_probe always displays the address on 32bits. That way, the zrm_probe size is not dependent on the model density. In order to split the zrm_probe into human-readable fields, there is a utility script named probe_zrm.tcl.

It is available in *$ZEBU_IP_ROOT/HW_IP/ZLPDDR5/script/nWave/probe_zrm.tcl*

Copy probe_zrm.tcl locally and update ZRM_PROBE_SCOPE valiable with currect path to zrm_probe For example:

set *ZRM_PROBE_SCOPE <path_to_zLPDDR5_instance>/zlpddr5/zrm_probe*

1. Before using this script, you must define the following environment variable:

- ZRM_PROBE_SCOPE: scope where the zrm_probe is available. For example:

  ```
  <path_to_zLPDDR5_instance>/zlpddr5/zrm_probe
  ```

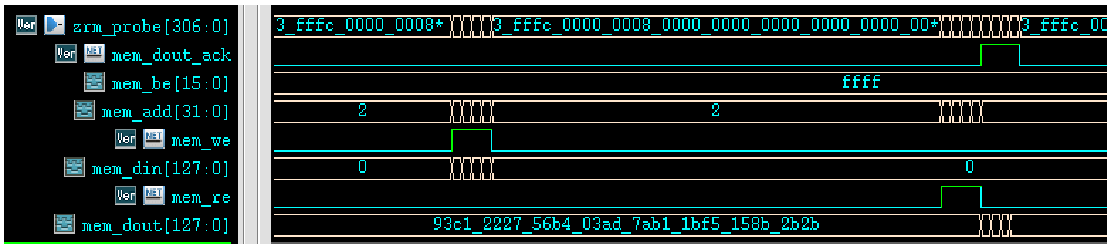
-

- ZRM_PROBE_SIZE: size of the zrm_probe signal.

  1. In nWave, menu Tools->Preferences->Waveform: Scroll down and tick the "Enable Command Entry Form" option -> OK.
  2. In Verdi, menu *Tools->Preferences->General*: tick the "*Enable TCL Command Entry*" option -> OK
  3. A Command Entry window will then be opened.
  4. Source the probe_zrm.tcl script at tcl command line.

Following signals will then appear in your waveform window:

Figure 3 Waveform probe zrm



8

Using Analyzer feature

Setup

For high level debugging a ZLPDDR5 analyzer feature is embedded in the memory model. This feature reports DRAM transactions, with data payload.

It also provides memory usage statistics such as data bandwidth, command read and command write miss, hit, empty and command counter.

This section explains the following topics:

- Package Content
- Feature
- Analyzer
- Statistics
- Using Analyzer feature
- Runtime
- Example

Package Content

The existing package contains an additional library in the $*ZEBU_IP_ROOT/HW_IP/lib* directory for the Analyzer feature:

- *For ZeBu*: libzlpddr5_analyzer_64.so
- *For Simulation*: libzlpddr5_analyzer_simu_64.so (or _32.so depending on the architecture).

Feature

The Analyzer feature provides the following information:

- Reporting of the ZLPDDR5 activity, called analyzer,
- Reporting of several metrics (called statistics) since the beginning of the run and during each frame period.

Communication between the ZLPPDR5 memory model and the software library that generates both the analyzer and statistics relies on the ZDPI feature.

Analyzer

Analyzer feature provides relevant information on each command that the memory model receives, including read and write data payload.

This section explains the following topics:

- Controlling the Analyzer feature
- Output
- Log file content
- Read and Write Command Displayed

Controlling the Analyzer feature

By default, the Analyzer feature is disabled.

To enable it, set the value of the USER_analyzer_en environment variable to 1 when instantiating the ZLPDDR5 memory wrapper (in *$ZEBU_IP_ROOT/HW_IP/wrapper_rtl*).

Output

You can configure the display for the Analyzer messages to display these in a file or a terminal, using the SNPS_VS_ZIP_TRACER_OUTPUT environment variable.

The following are the permissible values for the *SNPS_VS_ZIP_TRACER_OUTPUT* environment variable and the tool behavior:

- *Not defined*: Analyzer messages are displayed in a log file.
- *TERM*: Analyzer messages are displayed in the terminal.
- *FILE*: Analyzer messages are displayed in a log file.

Path

You can define the path to the Analyzer log file using the *SNPS_VS_ZIP_TRACER_FILE_PATH* environment variable.

If not defined, default path is the current directory from where the run is launched.

If defined to a specific path, the log file is created in specified path.

Filename

The filename for a specific instance of ZLPDDR5 is the path of this instance in the design followed by *analyzer.txt* suffix.

For example, tb.lpddr5.analyzer.txt.

Note:

The filename cannot be changed.

Log file control

To minimize the size of the analyzer log file and to store only relevant information, a dedicated Zebu register is provided to enable or disable the writing of data in the analyzer log file. For example, the user can enable the analyzer log file writing during a specific timing window.

The zebu register location for LPDDR5 memory model:

- For LPDDR5:
  - path_to_ZLPDDR5_inst>.ins_zebuMR.zebuReg[27]

Setting this register to 1 will disable the log file writing

Setting this register to 0 will enable the log file writing (default behaviour)

Log file content

The Analyzer log file consists of the following sections:

- Header
- Table Column

Header

The header of the Analyzer log file consists of:

- Scope, which signifies the ZLPDDR5 analyzer instance name on which the analyzer is reporting information.

Example:

```
   Scope : tb.lpddr5.analyzer
```

- Compile date when analyzer library was been compiled.

Example

```
 Current analyzer version date : Thu 24 10 2019 - 07:45:03
```

- General information about the ZLPDDR5 memory model, such as, dq_width of the memory model.

Example:

```
LPDDR5 model : dq_width = 16
```

Table Column

The following are the columns in the Analyzer log file:

```
|GLOBAL   |   MEM   | BK GRP   |   COMMAND | LOGICAL   | BA| ROW| COL | DM |
 DATA
 STAMP      STAMP    CKRATIO              ADDRESS
```

This section explains the columns in the Analyzer log file:

- Global Stamp
- Mem Stamp
- BK GRP CKRATIO
- Command
- Logical Address
- DM
- Data

Global Stamp

Time stamp reported by *svGetTimeFromScope()*

Mem Stamp

Reports the *ZLPDDR5 CK_t* clock cycle counter.

BK GRP CKRATIO

Reports the following information when a read or write data command is received:

- *Bank modes*: Allowed values are BG, 8B or 16B.
- *WCK*:*CK ratio* : Allowed values are: 2:1 or 4:1.

Command

Name of the reported command. Name displayed on the log file is the same as on the JEDEC specification.

Example

```
|     0|     22 |    - | PREpb    |   -   |    1|    - |    - |    - | -      |
|     0|      43|     - | PREab    |    - |  - |    - |    - |    - | -      |
```

Precharge per bank and precharge all bank commands.

Logical Address

The reported Read and Write address is build based on bank address (BA), row (R) and column (C) in the following order:

```
{BA,Row,Col}
```

Each width may vary depending on the memory model (see ZLPDDR5 SDRAM addressing in the ZLPDDR5 SDRAM specification).

- *BA*: Reports the Bank Address decoded from the related command.
- *ROW*: Reports the Row Address decoded from the related command.
- *COL*: Reports the Column Address decoded from the related command.

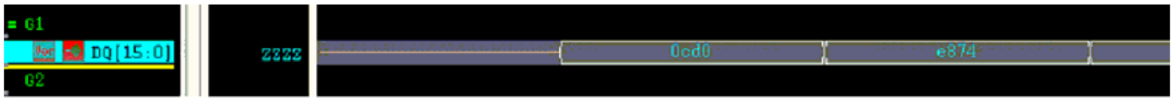
DM

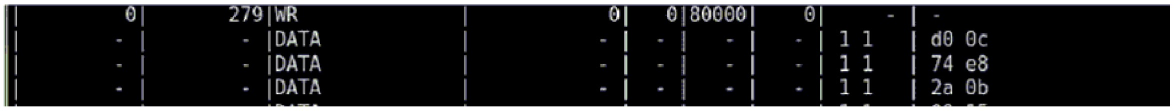
Reports byte enable. This field is relevant for mask write command. A 0 means that corresponding data byte is masked and not written in the memory. Else, the corresponding data byte is written in the memory.

For non-masked command, byte enable is displayed at 1.

Data

Reports the MRW/MRR and Read/Write/Mask-Write data from/to memory.





```
||          0|          279|WR              |         0|     0|80000|      0|      - | -
|          - |            - |DATA            |         - |    - |     - |      - | 1 1     | d0 0c
|          - |            - |DATA            |         - |    - |     - |      - | 1 1     | 74 e8
|          - |            - |DATA            |         - |    - |     - |      - | 1 1     | 2a 0b
```

First line is reporting the first data of the burst, second line is reporting the second data of the burst, and so on.

Read and Write Command Displayed

Since Read/Write command (RD/WR) and data payload (DATA) arrives separately on the bus, it is difficult to find the related command for a specific data payload. To ease the understanding of the analyzer log file, Read and Write Command and Data Payload are reported in a single block in a log file with a single timestamp. This is the timestamp where the command is issued.

On the above snapshot a write command arrives at Mem Stamp 279, and data arrives to the ZLPDDR5 after the write latency. Data command and Data payload are displayed in the log file when the last Data is received, in a single block.

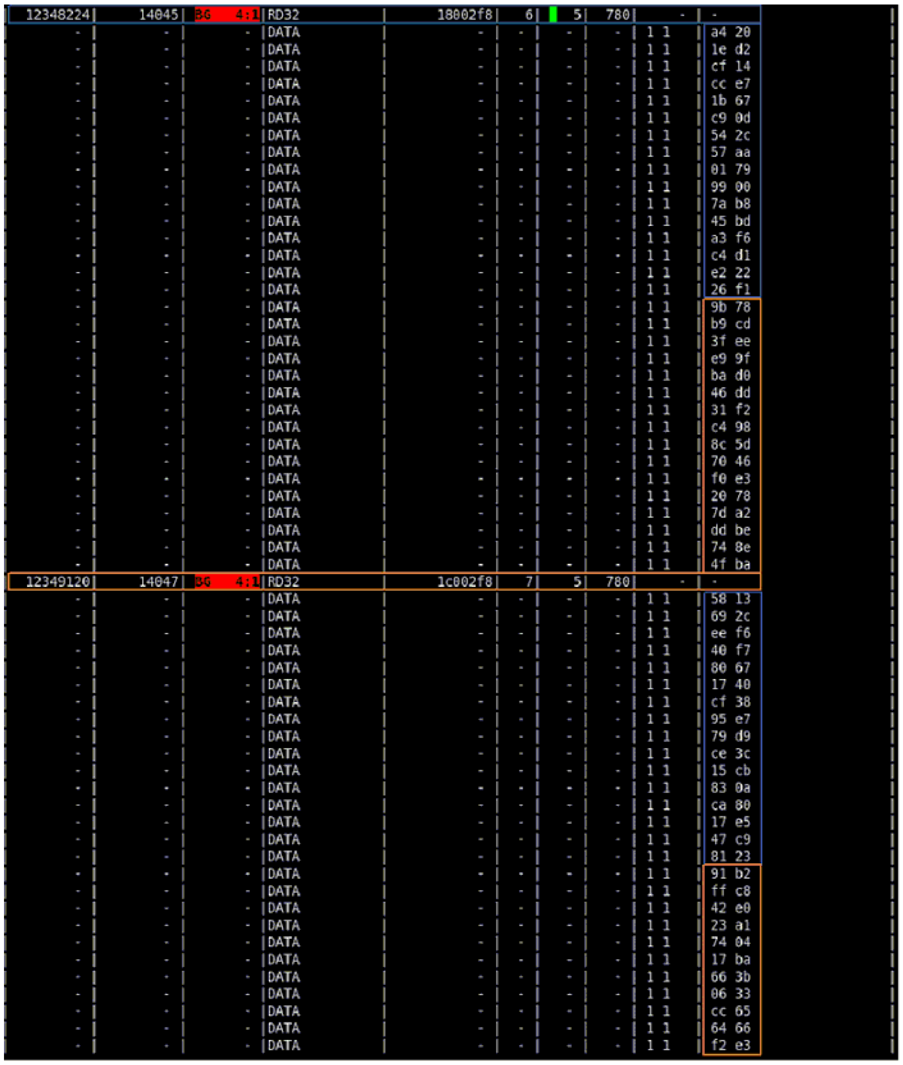
Interleaved Burst

When the ZLPDDR5 is configured in the BG mode, CK ratio 4:1 and BL 32, it generates a configuration where a burst of data command makes the data payload for each of the command interleaved with the other command. See *Burst Operation in the BG mode, WCK vs CK = 4:1* in the JESD209-5 specification.

In this case, two consecutive Read 32 are sent by the host. As a result, for a burst of data command, 1 clock period between the first command and the second command is generated, as reported by the mem stamp.

Consider the following figure:

Figure 4 Interleaved Burst



```
| 12348224|    14045|BG    4:1|RD32     |   18002f8|  6| ▮  5|  780|      - | -
|        -|        -|        -|DATA     |         -|  -|    -|    -| 1 1    | a4 20
|        -|        -|        -|DATA     |         -|  -|    -|    -| 1 1    | 1e d2
|        -|        -|        -|DATA     |         -|  -|    -|    -| 1 1    | cf 14
|        -|        -|        -|DATA     |         -|  -|    -|    -| 1 1    | cc e7
|        -|        -|        -|DATA     |         -|  -|    -|    -| 1 1    | 1b 67
|        -|        -|        -|DATA     |         -|  -|    -|    -| 1 1    | c9 0d
|        -|        -|        -|DATA     |         -|  -|    -|    -| 1 1    | 54 2c
|        -|        -|        -|DATA     |         -|  -|    -|    -| 1 1    | 57 aa
|        -|        -|        -|DATA     |         -|  -|    -|    -| 1 1    | 01 79
|        -|        -|        -|DATA     |         -|  -|    -|    -| 1 1    | 99 00
|        -|        -|        -|DATA     |         -|  -|    -|    -| 1 1    | 7a b8
|        -|        -|        -|DATA     |         -|  -|    -|    -| 1 1    | 45 bd
|        -|        -|        -|DATA     |         -|  -|    -|    -| 1 1    | a3 f6
|        -|        -|        -|DATA     |         -|  -|    -|    -| 1 1    | c4 d1
|        -|        -|        -|DATA     |         -|  -|    -|    -| 1 1    | e2 22
|        -|        -|        -|DATA     |         -|  -|    -|    -| 1 1    | 26 f1
|        -|        -|        -|DATA     |         -|  -|    -|    -| 1 1    | 9b 78
|        -|        -|        -|DATA     |         -|  -|    -|    -| 1 1    | b9 cd
|        -|        -|        -|DATA     |         -|  -|    -|    -| 1 1    | 3f ee
|        -|        -|        -|DATA     |         -|  -|    -|    -| 1 1    | e9 9f
|        -|        -|        -|DATA     |         -|  -|    -|    -| 1 1    | ba d0
|        -|        -|        -|DATA     |         -|  -|    -|    -| 1 1    | 46 dd
|        -|        -|        -|DATA     |         -|  -|    -|    -| 1 1    | 31 f2
|        -|        -|        -|DATA     |         -|  -|    -|    -| 1 1    | c4 98
|        -|        -|        -|DATA     |         -|  -|    -|    -| 1 1    | 8c 5d
|        -|        -|        -|DATA     |         -|  -|    -|    -| 1 1    | 70 46
|        -|        -|        -|DATA     |         -|  -|    -|    -| 1 1    | f0 e3
|        -|        -|        -|DATA     |         -|  -|    -|    -| 1 1    | 20 78
|        -|        -|        -|DATA     |         -|  -|    -|    -| 1 1    | 7d a2
|        -|        -|        -|DATA     |         -|  -|    -|    -| 1 1    | dd be
|        -|        -|        -|DATA     |         -|  -|    -|    -| 1 1    | 74 8e
|        -|        -|        -|DATA     |         -|  -|    -|    -| 1 1    | 4f ba
| 12349120|    14047|BG    4:1|RD32     |   1c002f8|  7|    5|  780|      - | -
|        -|        -|        -|DATA     |         -|  -|    -|    -| 1 1    | 58 13
|        -|        -|        -|DATA     |         -|  -|    -|    -| 1 1    | 69 2c
|        -|        -|        -|DATA     |         -|  -|    -|    -| 1 1    | ee f6
|        -|        -|        -|DATA     |         -|  -|    -|    -| 1 1    | 40 f7
|        -|        -|        -|DATA     |         -|  -|    -|    -| 1 1    | 80 67
|        -|        -|        -|DATA     |         -|  -|    -|    -| 1 1    | 17 40
|        -|        -|        -|DATA     |         -|  -|    -|    -| 1 1    | cf 38
|        -|        -|        -|DATA     |         -|  -|    -|    -| 1 1    | 95 e7
|        -|        -|        -|DATA     |         -|  -|    -|    -| 1 1    | 79 d9
|        -|        -|        -|DATA     |         -|  -|    -|    -| 1 1    | ce 3c
|        -|        -|        -|DATA     |         -|  -|    -|    -| 1 1    | 15 cb
|        -|        -|        -|DATA     |         -|  -|    -|    -| 1 1    | 83 0a
|        -|        -|        -|DATA     |         -|  -|    -|    -| 1 1    | ca 80
|        -|        -|        -|DATA     |         -|  -|    -|    -| 1 1    | 17 e5
|        -|        -|        -|DATA     |         -|  -|    -|    -| 1 1    | 47 c9
|        -|        -|        -|DATA     |         -|  -|    -|    -| 1 1    | 81 23
|        -|        -|        -|DATA     |         -|  -|    -|    -| 1 1    | 91 b2
|        -|        -|        -|DATA     |         -|  -|    -|    -| 1 1    | ff c8
|        -|        -|        -|DATA     |         -|  -|    -|    -| 1 1    | 42 e0
|        -|        -|        -|DATA     |         -|  -|    -|    -| 1 1    | 23 a1
|        -|        -|        -|DATA     |         -|  -|    -|    -| 1 1    | 74 04
|        -|        -|        -|DATA     |         -|  -|    -|    -| 1 1    | 17 ba
|        -|        -|        -|DATA     |         -|  -|    -|    -| 1 1    | 66 3b
|        -|        -|        -|DATA     |         -|  -|    -|    -| 1 1    | 06 33
|        -|        -|        -|DATA     |         -|  -|    -|    -| 1 1    | cc 65
|        -|        -|        -|DATA     |         -|  -|    -|    -| 1 1    | 64 66
|        -|        -|        -|DATA     |         -|  -|    -|    -| 1 1    | f2 e3
```

The above figure illustrates the following information:

1. memstamp 14045 for the first Read32 and memstamp 14047 for the second Read32.
2. Read data payload of the first Read command and the second Read command are interleaved on the DQ bus.
3. The analyzer reports the first Read32 command with 32 data payloads:
   - First 16 data payloads are related to the first Read32 command
   - Last 16 data are related to the second Read32 command

4. Then the analyzer reports the second Read32 command with 32 data payloads:
   - First 16 data payloads are related to the first Read32 command
   - Last 16 data payloads are related the second Read32 command
5. burst of 2 Read32 and highlights the above explanation.

The sequence described above is valid for a burst of Write32 command too.

Statistics

Statistics feature provide metrics related to the ZLPDDR5 activity. You can use this feature only when the Analyzer feature is enabled. If the Analyzer feature is disabled at the compilation step or at runtime, statistics are not be generated. For more information, see Controlling the Analyzer feature.

This section explains the following topics:

- Output
- Controlling statistics feature
- Time frame
- Log file content

Output

All statistics are generated in a log file.

The filename for a specific instance of ZLPDDR5 is the path of this instance in the design followed by analyzer_stat.txt, as shown in the following example:

```
tb.lpddr5.analyzer_stat.txt
```

Controlling statistics feature

You can control the information generated by the Statistic feature using the SNPS_VS_ZIP_TRACER_STAT environment variable.

The following explains the permissible values for the SNPS_VS_ZIP_TRACER_STAT environment variable

- *Unset or equal to 0*: Statistics are not generated during runtime
- *Set to 1*: Only bandwidth and bank activity is generated.
- *Set to 2*: In addition to the bandwidth and bank activity, the number of each command received by ZLPDDR5, is also reported.

The Statistics feature calculates the metrics from the beginning of the simulation and during a specific time frame.

Time frame

Metrics are reported in the file after every time frame period.

You can adjust the time frame period by using the following zebuMR_common registers:

- *Frequency*: zebuMR_common[31:28]
- *Multiplier*: zebuMR_common[35:32]
- *Interval*: zebuMR_common[37:36]

Interval value defines the base frequency of the report. The following are the permissible values:

- 0: every 1024 clock period (2^10)
- 1: every 16384 clock period (2^14)
- 2: every 262144 clock period (2^18)
- 3: every 4194304 clock period (2^22)

Multiplier value allows to adjust the base frequency. Here, frequency signifies the frequency at which the memory model runs. This is used by the Statistics feature.

The following is the formula for calculating the report frequency

```
report frequency = (multiplier_value +1) * (interval clock period)
```

For example, if the interval value is set at 0 and the multiplier is set at 2, then frequency report is (2+1) * 1024 = 3072 clock period.

The following table lists the zebuMR register and the corresponding frequency:

Table 9 Register Frequency

zebuMR_common[31:28]	Frequency (MHz)
0	67

Table 9 Register Frequency (Continued)

1	133
2	200
3	267
4	344
5	400
6	467
7	533
8	600
9	688
10	750
11	800

Note:

This frequency value is not related to the ZeBu frequency.

It is just available to translate frame period into in μs.

Log file content

The following information is displayed for the every frame period:

```
Statistics RANK 0 at time Global stamp  :    0   -- Mem stamp   :      0
Frequency:  267  MHz -- Clock ratio :  4:1 -- Nb_clock_cycle_per_frame :
 1024 -- Frame period : 3.835206 us
```

In this example the frequency is set at 267 MHz and the number of clock cycle per frame is 1024, so a report will be written in the file every 3.8 μs.

The following are the command bus and data metrics:

```
CMD  : Nb clock cycles since beginning   :   716  -- Nb clock cycles in
 frame  : 82
CMD bandwidth :   16.97 Mbps -- CMD bandwidth in frame    :      21.38
 Mbps
DATA : Data exchanged since beginning: 68096 -- Data exchanged in frame:
 7168
```

```
Data bandwidth: 1614.14 Mbps -- Data bandwidth since last : 1869.00 Mbps
Data rate : 100.88 Mbps -- Theorical Data rate :    2136.00 Mbps
```

The bank statistics are reported since the beginning and during the frame period (in parenthesis).

```
Bank  0 RD hit ratio: 1.00 (1.00) RD miss ratio: 0.00 (0.00) RD empty:
  0 (  0) WR hit ratio: 0.50 (0.50) WR miss ratio: 0.19 (0.19) WR empty:
    5 ( 5) Without_rd_wr     0 ( 0)
   Bank  1 RD hit ratio: 1.00 (1.00) RD miss ratio: 0.00 (0.00) RD empty:
     0 (    0) WR hit ratio: 0.00 (0.00) WR miss ratio: 0.25 (0.25) WR
 empty:     3 (    3) Without_rd_wr     0 (0)
   Bank  2 RD hit ratio: 1.00 (1.00) RD miss ratio: 0.00 (0.00) RD empty:
     0 (    0) WR hit ratio: 0.00 (0.00) WR miss ratio: 0.00 (0.00) WR
 empty:     3 (    3) Without_rd_wr     0 (0)
Additional metrics are available when SNPS_VS_ZIP_TRACER_STAT is set at 2
- Number of Activate          :         40      -- Since last          9
- Number of Masked Write      :          0      -- Since last          0
- Number of Masked Write 32   :         27      -- Since last          2
- number of write x           :          0      -- since last          0
- number of cas               :         44      -- since last          0
- number of cas write         :          0      -- since last          0
- number of cas read          :          0      -- since last          0
- number of cas fs            :          0      -- since last          0
- number of cas off           :          0      -- since last          0
- number of cas wr x          :          0      -- since last          0
- number of cas b3            :          0      -- since last          0
- number of cas dc        :          0      -- since last          0
- number of write fifo    :          0      -- since last          0
- number of read fifo     :          0      -- since last          0
- number of read dq cal   :          0      -- since last          0
- number of mpc start WCK2DQI  :    0      -- since last          0
- number of mpc stop WCK2DQI   :    0      -- since last          0
- number of mpc start WCK2DQO  :          0      -- since last          0
- number of mpc stop WCK2DQO   :          0      -- since last          0
- number of mpc ZQ cal start   :          0      -- since last          0
- number of mrw                 :          8      -- since last
 0
- number of mrr                 :          3      -- since last
 0
- number of pde                 :         53      -- since last
 6
- number of pdx                 :          0      -- since last
 0
- number of precharge all bank  :         12      -- since last
 3
- number of precharge per bank  :         17      -- since last
 0
- number of read 16              :          0      -- since last
 0
- number of read 32              :         65      -- since last
 6
```

```
- number of refresh per bank     :          0      -- since last
 0
- number of refresh all bank     :          0      -- since last
 0
- number of self refresh enter   :          0      -- since last
 0
- number of self refresh exit    :          0      -- since last
 0
- number of self refresh power down :      0      -- since last
 0
- number of deep sleep         :          0      -- since last         0
- number of write 16           :          0      -- since last         0
- number of write 32           :         41      -- since last         6
```

Before design compilation

Before compiling your design, enable the Analyzer feature by setting *USER_analyzer_en* parameter to 1 in the wrapper file. By default, the Analyzer feature is disabled.

Also, update the LD_LIBRARY_PATH to add the directory where the ZLPDDR5 analyzer library (libzlpddr5_analyzer_64.so) is located.

In addition, in the project UTF file, enable the DPI feature by specifying the following command:

```
dpi_synthesis -enable ALL
```

Runtime

To emulate the ZLPDDR5 memory model on ZeBu with the Analyzer/Statistics feature, load the analyzer shared library.

Specify the following in the zRci Tcl script using zRci:

```
ccall -load  $::env(ZEBU_IP_ROOT)/HW_IP/lib/libzlpddr5_analyzer_64.so
ccall -enable
```

Example

An example for the Analyzer and Statistics feature is available in the following directory:

```
It is available here: $ZEBU_IP_ROOT/HW_IP/ZLPDDR5/example
```

Feedback

9

Examples

The ZLPDDR5 package provides a waveform example and a tutorial, which are explained in the following sections:

- Waveform Example
- Tutorial

Waveform Example

The memory model package provides an example of waveform file in the *.fsdb* format. It can be open with *nWave*.

This waveform file comes from the simulation executed as an example in the Tutorial section.

The waveform file provided in the *example/waveform* directory.

Tutorial

This tutorial explains how to use the ZLPDDR5 Memory Models in the following situations:

- HDL Simulation
- Emulation on Zebu with
- *zRci* and user-defined Tcl script

Files Used for HDL Simulation

The following files of the *example* directory shown above are used in case of HDL Simulation:

- testbench files: *example/dut/*.v* and *example/dut/*.sv*
- run files: *example/run/pattern.txt* (memory settings)
- automatic flow: *Makefile*

Files Used for Emulation on ZeBu with *zRci* and User-Defined Tcl Script

The following files of the *example/zebu* directory shown above are used in case of emulation on ZeBu with *zRci* and user-defined Tcl script:

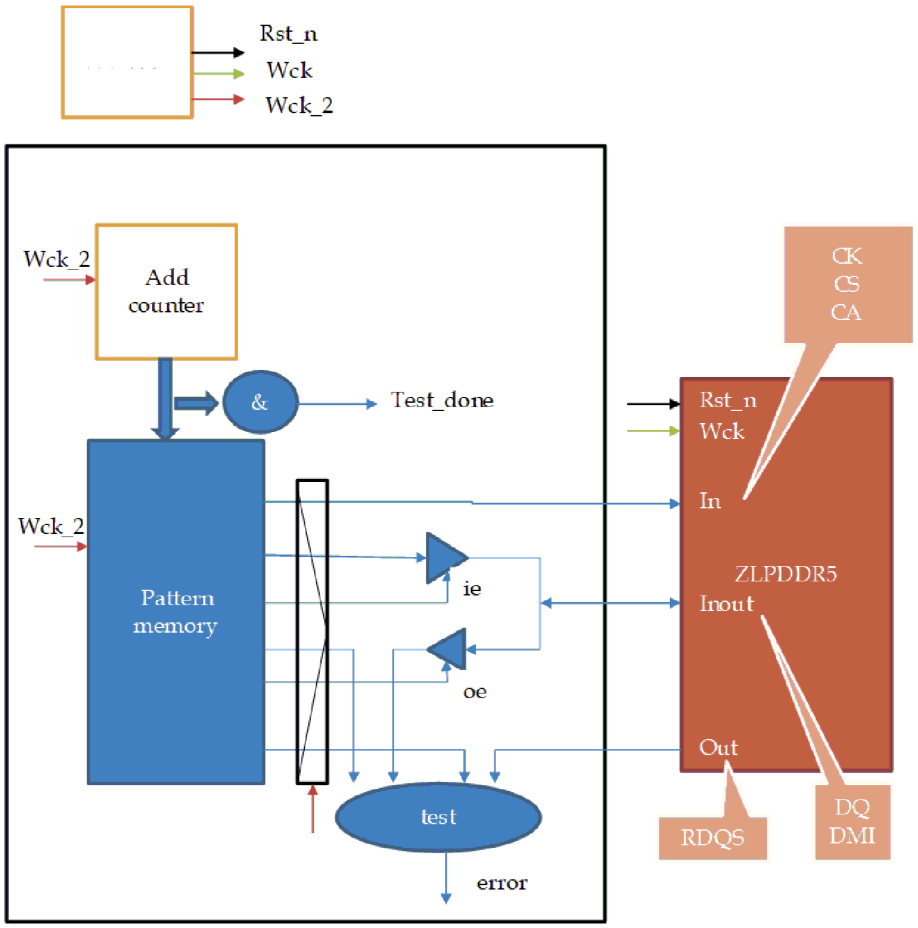
- compilation files:
  - *zebu/src/compil/LPDDR5.utf*
- run files:
  - *zebu/src/run/designFeatures*
  - *simu/src/run/pattern.txt (memory settings)*
  - *zebu/src/run/pattern.mem (memory settings)*
  - *zebu/src/run/zRci_test.tcl*
- automatic flow: *Makefile*

Feedback

Description

The following figure provides an overview of the DUT:

Figure 5 Tutorial DUT Overview



In the figure above, the DUT instantiates:

- a ZLPDDR5 Memory Models (*ins_zLPDDR5*)
- a ZeBu Memory (*pattern*) which contains commands and data for the ZLPDDR5 Memory.

To validate the correctness of the ZLPDDR5 instance, read data are compared to expected data stored in the *pattern* memory.

The following are the output files generated by the DUT:

- *test_done* is set when the end of the tutorial is reached (address counter reaches max value)
- *test_error* is set when ZLPDDR5 outputs differ from reference values in pattern memory.

Content

The pattern performs the following operations:

1. It configures the ZLPDDR5 model with typical latencies and no Data Bus Inversion (Mode Register 1, 2 and 3), clock ratio 4:1, 16 bank mode
2. It executes the Active 4 different bank rows, ACT-WR, 1000 gapless WRITE32, changing bank each time
3. It executes 1000 gapless READ32

Running the Tutorial Examples

Setting the Environment

Make sure the following environment variables are set correctly before running the tutorial examples:

- *ZEBU_ROOT* must be set to a valid ZeBu installation.
- *ZEBU_IP_ROOT* must be set to the package installation directory.
- *FILE_CONF* must be set to your system architecture file.
- *REMOTECMD* can be specified if you want to use remote synthesis and remote ZeBu jobs.
- *ZEBU_XIL* or *ZEBU_XIL_VIVADO* must be set to a valid ISE installation (the script default value for the ISE installation directory is *$ZEBU_ROOT/zebu_env.bash*)

Running HDL Simulation

The HDL simulation is performed by Synopsys VCS simulator.

To compile and run the example, perform the following steps:

1. Go to the *example* directory.
2. Launch the

3. compilation flow with the *Makefile*:

   ```
   make simu
   ```

4. Optionally, clean the compilation and run directory (the working directory automatically created at compilation and run is removed):

   ```
   make clean_simu
   ```

Running Emulation with *zRci* and User-Defined Tcl Script

To compile and run the example, perform the following steps:

1. Select the synthesis tool of your choice by either:
   - changing the selected synthesis tool in *zCui* graphical interface
   - changing the
   - *SYNTH_TOOLS* environment variable
2. Go to the *example/zebu* directory
3. Launch the compilation flow with the *Makefile*:

make compile	without *zCui* Graphical User Interface
make compile_gui	with *zCui* Graphical User Interface

4. Launch the emulation flow with the *Makefile*:

make run	without *zRci* Graphical User Interface
make run_gui	with *Verdi* Graphical User Interface

5. Optionally, clean the compilation and run directory (the working directory automatically created at compilation and run is removed):

   ```
   make clean_compile
   make clean_run
   ```