# Hardware System Verification (HSV)
# Vertical Solutions Engineering (VSE)

# DDR4 LRDIMM
# Palladium Memory Model
# User Guide

**Document Version:** 1.8

**Document Date:** July 2018

# Contents



## List of Figures



## List of Tables

# General Information

The Cadence Memory Model Portfolio provides memory device models for the Cadence Palladium XP, Palladium XP II and Palladium Z1 series systems. Optimizing the acceleration and/or emulation flow on these platforms for MMP memory models may require information outside the scope of the MMP user guides and related MMP documentation.

## 1.1 Related Publications

For basic information regarding emulation and acceleration, please refer to the following documents:

For Palladium XP and Palladium XP II:

UXE User Guide
UXE Library Developer's Guide
UXE Known Problems and Solutions
UXE Command Reference Manual
Palladium XP Planning and Installation Guide
Palladium Target System Developer's Guide
What's New in UXE

For Palladium Z1:

VXE User Guide
VXE Library Developer's Guide
VXE Known Problems and Solutions
VXE Command Reference Manual
Palladium Z1 Planning and Installation Guide
Palladium Target System Developer's Guide
What's New in VXE

# DDR4 LRDIMM Memory Model

## 1. Introduction

The Cadence Palladium DDR4 LRDIMM Model is based on the JEDEC Standard DDR4 Register DDR4RCD01 Specification (Rev 0.92 draft; APR 2013) and the JEDEC Standard DDR4 Data Buffer DDR4DB01 Specification (Rev 0.95 draft; JUL 2013). The draft DDR4 SDRAM Load Reduced DIMM Design Specification (Rev 0.81 DRAFT; APR 2013) was also consulted. Please refer to the *Cadence DDR4 Palladium Memory Model User Guide* for base device model information. This DDR4 LRDIMM Memory Model User Guide is intended to supplement the existing DDR4 User Guide, not replace it.

The DDR4 LRDIMM model is available in several configurations based on generic configurations from the JEDEC spec. As real devices become available from vendors those will be added to the catalog.

Different sizes from 4GB up to 128GB are available, please consult the memory model catalog for the current available list.

## 2. Model Release Levels

All models in the Memory Model Portfolio are graded with a release level. This release level informs users of the current maturity and status of the model. All families in the library are graded at one of these levels.

The different levels give an overall indication of the amount of testing, level of quality and feature availability in the model. For details on supported features check the User Guide for that particular model family.

There are three release levels for models in the MMP release.

| Release Level | | Model Status | Available in Release | Listed in Catalog | Requires Beta Agreement |
|---|---|---|---|---|---|
| Mainstream Release | MR | Fully released and available in the catalog for all customers to use. | Yes | Yes | No |
| Emerging Release | ER | Model has successfully completed Beta engagement(s). Most, but not all features have been tested. Documentation is available. | No | Yes | Yes |
| Initial Release | IR | Model has completed initial development and has been released to Beta customer(s). The model may have missing features, may not be fully tested, may not have documentation. Model may contain defects. | No | Yes | Yes |

Access to Initial and Emerging Release versions of the models will require a Beta Agreement to be signed before the model can be delivered.

# 3. Configurations

The following table lists possible configurations. Not all configurations are available from all vendors. Please consult the appropriate vendor site for details on the parts they offer.

**Table 1: DDR4 LRDIMM Configurations**

| Memory Size | Data Width | Ranks | Model Name | Base Device Model |
|---|---|---|---|---|
| 4GB | 64 | 1 | jedec_ddr4_4GB_64_lrdimm | jedec_ddr4_2gb_4 |
| 4GB | 72 | 1 | jedec_ddr4_4GB_72_lrdimm | jedec_ddr4_2gb_4 |
| 8GB | 64 | 1 | jedec_ddr4_8GB_64_lrdimm | jedec_ddr4_4gb_4 |
| 8GB | 72 | 1 | jedec_ddr4_8GB_72_lrdimm | jedec_ddr4_4gb_4 |
| 16GB | 64 | 2 | jedec_ddr4_16GB_2r_64_lrdimm | jedec_ddr4_4gb_4 |
| 16GB | 72 | 2 | jedec_ddr4_16GB_2r_72_lrdimm | jedec_ddr4_4gb_4 |
| 32GB | 64 | 4 | jedec_ddr4_32GB_4r_64_lrdimm | jedec_ddr4_4gb_4 |
| 32GB | 72 | 4 | jedec_ddr4_32GB_4r_72_lrdimm | jedec_ddr4_4gb_4 |
| 32GB | 72 | 4 | m386a4g40dm0 | k4a4g045wd |
| 32GB | 72 | 4 | m386a4g40dm1 | k4a4g045wd |
| 32GB | 72 | 2 | m386a4k40bb0 | k4a8g045wb |
| 64GB | 72 | 4 | m386a8k40bm1 | k4a8g045wb |
| 128GB | 72 | 8 | m386aak40b40 | k4a8g045wb |

## 4. Model Block Diagram

The following block diagram in Figure 1 shows a single rank 64-bit wide LRDIMM model. The device consists of 16 x4 DDR4 SDRAMs, 8 Data Buffers, and a Registering Clock Driver. These components are briefly described below.

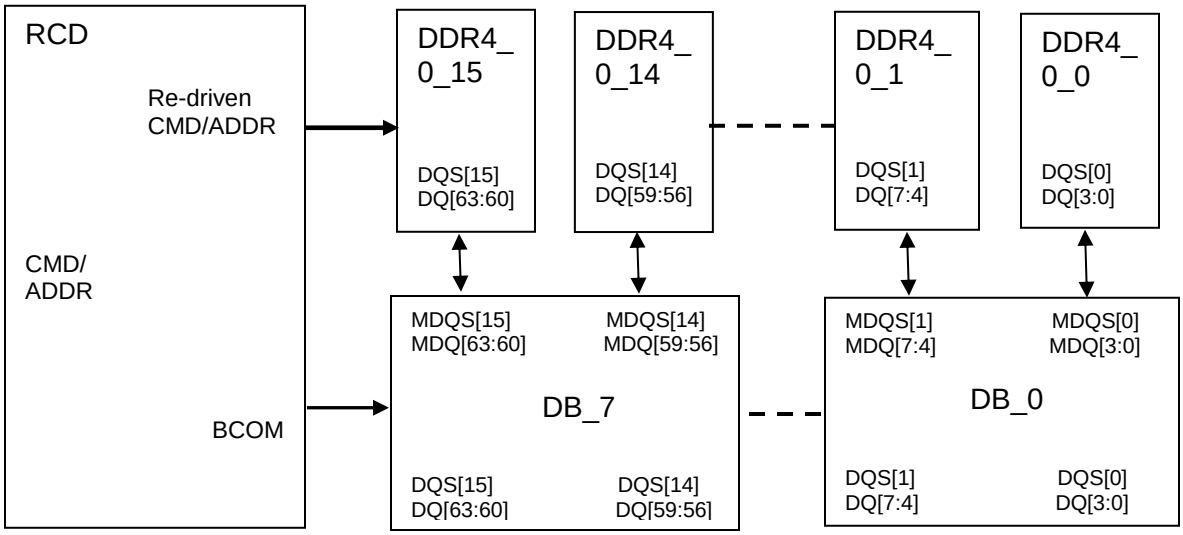



**Figure 1: DDR4 LRDIMM Block Diagram**

## 5. Registering Clock Driver

The registering clock driver, also called register, re-drives command, address, and clock signals from the host to the SDRAMs. It consists of 16 4-bit and 15 8-bit control words that can be configured by the host to enable specific features and to enable or disable output drivers and clocks. It communicates with the Data Buffers through the BCOM bus as commands are received from the host. It informs the Data Buffers the current command such as Write, Read, Mode Register settings, as well as writing and reading control words within the Data Buffers.

# 6. Data Buffer

The Data Buffers re-drives DQ and DQS signals from the host to reduce loading. It receives commands from the register to determine the direction of data flow at specific latency delays. Precise timing delays may be configured by the host via the control words. However MMP models are cycle based, therefore, cannot support fractional delays. There are 8 function spaces in the data buffer for storing control words. Function space 0 consists of 16 4-bit control words and 15 8-bit control words. Function spaces 1 to 7 each consists of 16 8-bit control words.

# 7. Data Buffer Commands—BCOM Commands

The Data Buffer model accepts the following commands:

- Write
- Read
- MRS Write
- BCW Write
- BCW Read
- NOP

# 8. DDR4 SDRAMs

The DDR4 LRDIMM model instantiates sixteen of the DDR4 SDRAM memory model. Please consult the *Cadence DDR4 Palladium Memory Model User Guide* for information about the DDR4 SDRAM model.

# 9. Initialization Sequence

The DDR4 LRDIMM and base device models require that the memory controller follow the initialization sequence as documented in the specification. The sequence basically entails the following steps:

1. Assert RESET
2. De-assert RESET
3. Start clocks
4. Wait for CKE to asserted
5. Write to all seven Mode Registers
6. Issue ZQC command

There is no ordering required in the specification for Step 5 (Write to all seven Mode Registers), however, all seven mode registers must be written. The model requires that these steps be performed in the correct sequence in order to complete initialization. The model will not respond to any others commands unless this sequence is completed. Please consult the JEDEC DDR4 LRDIMM specification for additional detail about the initialization sequence and its requirements.

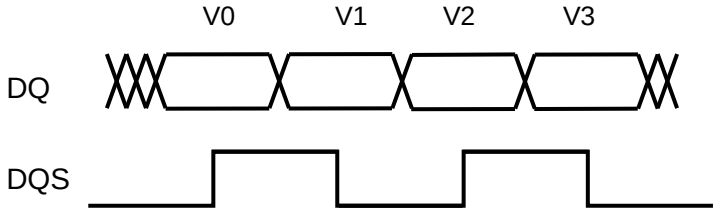
# 10. Limitations

Currently the DDR4 LRDIMM model does not support the following features:

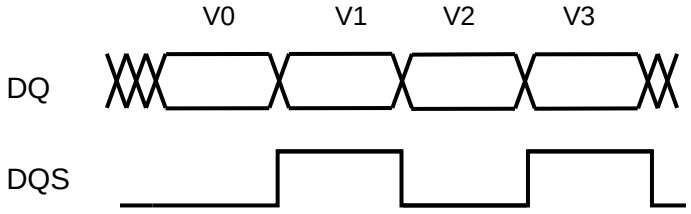
Per Buffer Addressability
Per DRAM Addressability
Host_Read/MREP/MRD/MWD training
DRAM Write Leveling/Host Interface Write Leveling
Transparent Mode
Power Down Modes
Frequency Change
Dual Frequency
Address Mirroring
SPD

# 11. Handling DQS in Palladium Memory Models

For writes to a DDR memory, industry datasheets show each DQS edge centered within the corresponding valid period (v0, v1, v2, etc.) of DQ, as in the following diagram.

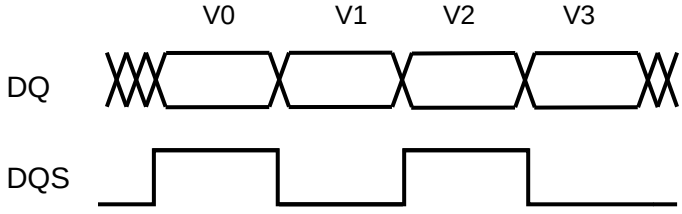



For DDR models provided by Cadence for Palladium, if the design drives DQ and DQS signals with the above timing, the DDR memory will behave correctly. However, to obtain this timing in Palladium, the fastest design clock must toggle twice as frequently as the DQS signal. If this faster clock is not needed for any other reason, the presence of the faster clock will usually cause an unnecessary 2X slowdown in emulation speed. To eliminate the need for a faster clock, you can have the design generate each DQS edge at the end of the corresponding DQ valid period (rather than the middle), as in the following diagram:




Note that the first DQS edge is at the *end* of first valid DQ, not at the beginning.

For reads from the DDR model, the DDR model will drive DQ and DQS with the first DQS edge at the *beginning* of the first valid data, not at the end:



The DDR model behaves this way to conform with industry datasheets for DDR memories. The design reading the data from the DDR model must delay the DQS signal, and use the delayed-DQS signal to sample the DQ. A delay of one Q_FDP0B should work fine, even in CAKE 1X mode. If you are using CAKE 1X mode and the DDR clock is the fastest design clock, the DQ signal will change twice per FCLK, and the Q_FDP0B delaying DQS will provide one-half FCLK delay, so that each delayed-DQS edge is at the end of the corresponding data valid period.

To delay the DQS signal, a commonly used approach is to create a special pad cell for DQS, that has a Q_FDP0B delay cell inserted on the path that leads from the DDR memory into the design.

The user may insert delays into pad cells (or elsewhere in the design) using the below code example which leverages ixc_pulse, an internal primitive that can be used to access FCLK and to create controlled delay, for IXCOM flow and leverages the Q_FDP0B primitive for delay generation in the Classic ICE flow. For more detailed information about ixc_pulse please reference the *UXE User Guide* section called *Generating Pulses*. There is no need for the user to define IXCOM_UXE for the Verilog macro; it is predefined for the user in IXCOM flow. Note that in UXE 13.1.0 and prior the equivalent pulse generating function was named axis_pulse.

```
// Flow independent delay cell
module pxp_fclk_delay (in, out_delay);
input in;
output out_delay;

reg out_delay;

`ifdef IXCOM_UXE
  wire VCC=1'b1;
  ixc_pulse #(1)(Fclk,VCC);
  always @(posedge Fclk)
    out_delay <= in;
`else
  Q_FDP0B fclk_dly (.D(in), .Q(out_delay));
`endif

endmodule
```

## 12. User Adjustable Parameters

There are no additional user adjustable parameters for the DDR4 LRDIMM model above and beyond the ones described for the DDR4 SDRAM in the DDR4 SDRAM User Guide.

## 13. Verilog Macro Defines

The following table () lists the Verilog macro `define(s) required by the Palladium DDR4 LRDIMM model.

**Table 2: Optional Verilog Defines**

| **`define Macro purpose** | **Optional Verilog `define** | **Default Value** |
|---|---|---|
| **Set addressing to {row,bank,col} instead of the default {bank,row,col}.** | MMP_RBC | Undefined by default; no value needed |
| **Set addressing to {ba,bg,row,col} instead of the default {bg,ba,row,col}.** | MMP_BA_BG | Undefined by default; no value needed |

A Verilog define requires that a condition/configuration be defined by passing a value to a Verilog `*define* macro with the correct string. This define operation should be located in the user's runtime script; there is no need for the user to modify the model code to effect this definition.

As an example, consider the setting of the memory model type. For the irun flow the user may add "-define < memory_type >" to the irun command in the runtime scripting where <memory_type> is one of the above supported values. For ixcom flow the user may add "+define+< memory_type>" to the ixcom command in the runtime scripting, where <memory_type> is one of the above supported values. Below is an example of such a command for the irun flow where MMP_DFI_DDR4 is the user's selected define value.:

```
irun    -64bit -c -sv \
        <file list for model  *.vp> . . .
        <path to testbench.v> \
        -incdir ../../../utils/cdn_mmp_utils/sv \
        ../../../utils/cdn_mmp_utils/sv/cdn_mmp_utils.sv \
        -sv_lib \
        ../../../utils/cdn_mmp_utils/lib/64bit/libMMP_utils.so \
        -define MMP_RBC
```

If the user desires to set the Verilog define during compile time vlan script snippet below may serve as an example:

```
vlan   -64bit +sv tb.v \
       <model_name>.vp \
       ddr4rcd01.vp \
       ddr4db01.vp \
       <base_device_model>.vp \
       -incdir ../../../utils/cdn_mmp_utils/sv \
       ../../../utils/cdn_mmp_utils/sv/cdn_mmp_utils.sv \
       ${UXE_HOME}/etc/ixcom/IXCclkgen.sv \
       -vlog_ext .vp \
       -v ${UXE_HOME}/etc/ixcom/ixcom_hdlice_ref.vp \
       +define+MMP_RBC
```

## 14. IXCOM Compilation

The memory models are currently provided in one format: an encrypted RTL file(s) (*.vp) that targets use in either the IXCOM flow or in the ICE flow. The encrypted RTL (*.vp) file(s) must be synthesized along with other design code prior to acceleration / emulation.

Here are some simple commands for compiling the DDR4 LRDIMM models in the IXCOM flow and ICE flow.

```
vlan   -64bit +sv tb.v \
       <model_name>.vp \
       ddr4rcd01.vp \
       ddr4db01.vp \
       <base_device_model>.vp \
       -incdir ../../../utils/cdn_mmp_utils/sv \
       ../../../utils/cdn_mmp_utils/sv/cdn_mmp_utils.sv \
       ${UXE_HOME}/etc/ixcom/IXCclkgen.sv \
       -vlog_ext .vp \
       -v ${UXE_HOME}/etc/ixcom/ixcom_hdlice_ref.vp

ixcom –ua +dut+<model_name> -top tb

xeDebug -64 --ncsim \
   -sv_lib ../../../utils/cdn_mmp_utils/lib/64bit/libMMP_utils.so -- \
   hw.i
```

The content of hw.i may have the following commands:

```
debug .
host .
xc xt0 zt0 run
run
```

```
exit
```

ICE flow synthesis commands:
vavlog ../src/<model_name>.vp ddr4rcd01.vp ddr4db01.vp <base_device_model>.vp

vaelab –keepRtlSymbol –keepAllFlipFlop –outputVlog <model_name>.vgp <model_name>

**NOTE:** It is common for Palladium flows to require –keepallFlipFlop since it removes optimizations that are in place by default. For example, without –keepAllFlipFlop, HDL-ICE can remove flops with constant inputs and merge equivalent FF. The picture above is modified a bit when ICE ATB mode ( –atb) is used since then a constant input FF is only optimized out when there is no initial value for it or the initial value is the same as the constant input value.

It is also common for Palladium flows to require –keepRtlSymbol. This option enables the HDL Compiler to keep original VHDL RTL symbols, such as ".", whenever possible. In other words, it maps VHDL RTL signal name a.b to the netlist entry, \a.b. Without this modifier, the signal name would otherwise be converted to a_b in the netlist.

If the recommended compile script includes the aforementioned options, the user must include them to avoid affecting functionality of the design.
The above examples are intended to show the difference between compiling DDR4 models and DDR4 DIMM models. Please see the UXE or VXE user guide for more details on the IXCOM flow.

# 15. Revision History

The following table shows the revision history for this document

| Date | Version | Revision |
|---|---|---|
| December 2014 | 0.1 | Initial Release |
| March 2015 | 1.0 | Update related publications list. Update revision for official release. |
| July 2015 | 1.1 | Update Cadence naming on front page |
| September 2015 | 1.2 | Added MMP_BA_BG macro for {ba,bg,row,col} address mapping. Removed references to <model_name>.vgp as an input file. |
| January 2016 | 1.3 | Update for Palladium-Z1 and VXE |
| February 2016 | 1.4 | Added Samsung models. |
| July 2016 | 1.5 | Remove hyphen in Palladium naming |
| January 2018 | 1.6 | Modify header and footer |
| June 2018 | 1.7 | Remove Beta watermark. DDR4 LRDIMM at non-Beta level. |
| July 2018 | 1.8 | Update for new utility library |