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

DDR3
Palladium Memory Model
User Guide

Document Version: 2.9

Document Date: July 2018

Contents

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

DDR3 Memory Model

1. Introduction

The Cadence Palladium DDR3 Model is based on the JEDEC specification *DDR3 SDRAM Standard JESD79-3D* (September 2009).

The model is available in several configurations with model sizes to match real devices manufactured by the following vendors: Micron, Elpida and Samsung.

Different sizes from 512Mb up to 4Gb 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. Overview of Features

The table below shows the MMP DDR3 memory model support level for the features from the JEDEC specification *DDR3 SDRAM Standard JESD79-3D* (September 2009).

Table 1: Features List of DDR3 Model

FEATURE	SUPPORT	Spec. Section + NOTE
DDR3 (JEDEC 79-3d)		
DDR3 Addressing / Configurations (2.11)		
512 Mb	Yes	See MMP catalogue for specific devices
1Gb	Yes	See MMP catalogue for specific devices
2 Gb	Yes	See MMP catalogue for specific devices
4 Gb	Yes	See MMP catalogue for specific devices
8 Gb	Yes	See MMP catalogue for specific devices
Functional (3)		
States		3.1
Power On	No	
Reset Procedure	Yes	
Initialization	Yes	
ZQ Calibration	Yes	
Idle	Yes	
MRS, MPR Write Leveling	Yes	
Self Refresh	No	
Refreshing	Yes	
Activating	Yes	
Bank Active	Yes	
Read (RD, RDS4, RDS8)	Yes	
Read A (RDA, RDAS4, RDAS8)	Yes	
Write (WR, WRS4, WRS8)	Yes	
Write A (WRA, WRAS4, WRAS8)	Yes	
Precharging	Yes	
Basic	Yes	3.2
Reset and Initialization	Yes	3.3
Registers	Yes	3.4
Mode Register MR0	Yes	3.4.2 Partial. See section 7.1 of this user guide
Mode Register MR1	Yes	3.4.3 Partial. See section 7.2 of this user guide
Mode Register MR2	Yes	3.4.4 Partial. See section 7.3 of this user guide
Mode Register MR3	Yes	3.4.5 Partial. See section 7.4 of this user guide
DDR3 SDRAM Commands (4)		
Commands	Yes	4.1 Partial. See section 8 of this user guide
CKE	Yes	4.2
No Operation Command	Yes	4.3
Deselect Command	Yes	4.4
DLL-off Mode	No	4.5
DLL on/off switching	No	4.6
Input clock frequency change	Yes	4.7
Write Leveling	Yes	4.8

FEATURE	SUPPORT	Spec. Section + NOTE
Extended Temperature	No	4.9
Multi Purpose Register	No	4.10
ACTIVE Command	Yes	4.11
PRECHARGE Command	Yes	4.12
READ Operation	Yes	4.13
WRITE Operation	Yes	4.14
Refresh Command	Yes	4.15
Self-Refresh Operation	No	4.16
Power-Down Modes	No	4.17
ZQ Calibration Commands	Yes	4.18
DD3 Specification Sections from *On-Die Termination* (5) through *Electrical Characteristics and AC Timing* (13) are not relevant to MMP emulation models, thus not supported.		

4. Configurations

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

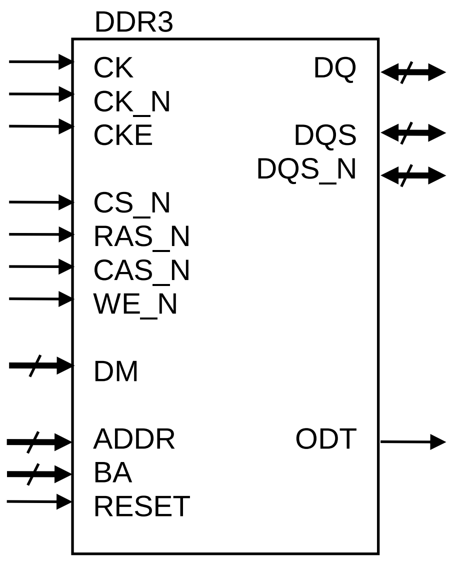
Data Width		**512Mb**	**1Gb**	**2Gb**	**4Gb**	**8Gb**
	Banks	**8**	**8**	**8**	**8**	**8**
X4	**Row Address**	**A[12:0]**	**A[13:0]**	**A[14:0]**	**A[15:0]**	**A[15:0]**
	Column Address	**A[11,9:0]**	**A[11,9:0]**	**A[11,9:0]**	**A[11,9:0]**	**A[13,11,9:0]**
X8	**Row Address**	**A[12:0]**	**A[13:0]**	**A[14:0]**	**A[15:0]**	**A[15:0]**
	Column Address	**A[9:0]**	**A[9:0]**	**A[9:0]**	**A[9:0]**	**A[11,9:0]**
X16	**Row Address**	**A[11:0]**	**A[12:0]**	**A[13:0]**	**A[14:0]**	**A[15:0]**
	Column Address	**A[9:0]**	**A[9:0]**	**A[9:0]**	**A[9:0]**	**A[9:0]**

NOTE : Per specification, a portion of the SDRAM address bus, in other words a slice of the column address width, is multiplexed with non-address signals. For example, in A[12:0] the following functions are shared:
A[10] serves as the auto pre-charge bit.
A[12] is BC_n

A[12:0] inputs are used for ADDR. This allocation is reflected in wrappers and core model. Please see the Input/Output Functional Description in the specification for additional detail.

5. Model Block Diagram

The widths of the Addr, Ba, Dm, Dq, and Dqs buses are dependent on the density of the part being used.



6. Model Parameter Descriptions

The following table provides details on the **user adjustable** parameters for the Palladium DDR3 Memory Model. These parameters may be modified when instantiating a DDR3 wrapper or, if necessary, by modifying the HDL parameter declarations and default values which are exposed for access and debug visibility.

User Adjustable Parameter	Default Value	Description
data_bits	16	Width of data bus DQ
addr_bits	13	Width of row address bus
bank_addr_width	3	Width of bank address bus
row_addr_width	13	Width of row address bus
col_addr_width	10	Width of column address bus
byte_width	8	Width of one byte or word in case of X4
tWLO_in_clocks	3	tWLO time in clocks
SMConfigSpecific_UserData (if defined)	0	Parameter for smart memory data passing. [2:0] bits can be used for device/channel/subpart; [63:61] bits can be used for extension value of total_addr_bits

The following table provides some information about exposed localparams that are NOT user adjustable. On rare occasion the user may find one of these localparam needs adjusting for their configuration. If this case arises, please contact Cadence emulation or MMP support.

Localparam	Default Value	Description
num_bytes	2	data_bits / byte_width
total_addr_bits	26	memory capacity: bank_addr_width + row_addr_width + col_addr_width

Note that there are additional exposed localparams in the model hdl that are not described here nor intended to be described here. These additional localparams are exposed for debugging purposes only and will not be described herein.

7. Verilog Macro Defines

The following table lists the Verilog macro defines related to the MMP DDR3 model.

As described in the section “Large Memory Support,“ each MMP DDR3 model that exceeds 30 bits of address width needs to incorporate the multiple core memory array generator (mmp_gen_mem.vp) into the memory build. In these cases, the file mmp_gen_mem.vp automatically defines Verilog macro MMP_LG_MEM_BITS with a value of ‘30’ by default. This default value may be changed within the file mmp_gen_mem.vp or overridden on the command with the define option. Users of DDR3 memories that are smaller than the 30 bit address width should NOT need to be aware of or modify this file or its Verilog macro value.

`define Macro Purpose	Optional Verilog `define	Default Value
Large memory support	MMP_LG_MEM_BITS	30 Range 25 .. 30

8. Address mapping

The array of the DDR3 model is mapped into the internal memory of the Palladium system. This array is a single two dimensional array. The mapping of bank, row and column addresses to the internal model array is as follows:

ARRAY_ADDR = {BA, ROW, COL}

This information is required if the memory needs to be preloaded with user data.

The array name in the model hierarchy is: memcore

If {row,bank,col} addressing is needed instead of the default {bank,row,col} addressing, add +define+MMP_RBC to the vlan invocation (IXCOM flow) or to the appropriate HDL-ICE synthesis (Classical flow) command. No value is required for MMP_RBC – only the compile phase define. This option is applicable when using the <model>.vp file.

9. Register Definitions

In the DDR3 there are four registers, the Mode Register and three Extended Mode Register. The Palladium DDR3 model implements the Mode Register and all three Extended Mode Registers.

9.1. Mode Register MR0

The mode register is implemented in the DDR3 model. The model supports the parameter values as described in the tables and sub tables below. Power Down, Write Recovery, and DLL Reset are not supported in this memory model.

15-13	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
0	PD	Write Recovery			DLL Reset	Mode	CAS Latency			Burst Type	CAS Latency	Burst Length	

Bit 12	**Power Down**	
0	Fast Exit	Not Supported
1	Slow Exit	Not supported

Bit 11	**Bit 10**	**Bit 9**	**Write Recovery**
X	X	X	Not supported

Bit 8	**DLL Reset**	
0	No	Not Supported
1	Yes	Not supported

Bit 7	**Mode**	
0	Normal	Supported
1	Test	Not supported

Bit 6	**Bit 5**	**Bit 4**	**Bit 2**	**CAS Latency**
0	0	0	0	Reserved
0	0	1	0	5
0	1	0	0	6
0	1	1	0	7
1	0	0	0	8
1	0	1	0	9
1	1	0	0	10
1	1	1	0	11
0	0	0	1	12
0	0	1	1	13
0	1	0	1	14
0	1	1	1	15
1	0	0	1	16
1	0	1	1	Reserved
1	1	0	1	Reserved
1	1	1	1	Reserved

Bit 3	**Burst Type**	
0	Nibble Sequential	Supported
1	Interleaved	Supported

Bit 1	**Bit 0**	**Burst Length**
0	0	8 Fixed
0	1	BC4 or 8 On the Fly
1	0	BC4 Fixed
1	1	Reserved

9.2. Mode Register MR1

The mode register MR1 is implemented in the DDR3 model. The model supports the following parameter values.

15-13	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
0	QOFF	TDQS Enable	0	RTT3	0	Write Leveling	RTT2	DIC2	Additive Latency		RTT1	DIC1	DLL Enable

TDQS Enable, RTT3, RTT2, RTT1, DIC2, DIC1 and DLL Enable are not applicable to the DDR3 Palladium Model.

Bit 4	**Bit 3**	**Additive Latency**
0	0	0
0	1	CL-1
1	0	CL-2
1	1	Reserved

Bit 7	**Write Leveling 1**
0	Write Leveling Disabled
1	Write Leveling Enabled

Bit 12	**QOFF 1**
0	Output Buffer Enabled
1	Output Buffer Disabled

9.3. Mode Register MR2

The mode register MR2 is implemented in the DDR3 model. The model supports the following parameter values.

15-13	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
0	0	0	RTT_WR		0	SRT	ASR	CAS Write Latency			PASR		

RTT_WR, SRT, ASR and PASR are not applicable to the DDR3 Palladium Model.

Bit 5	Bit 4	Bit 3	CAS Write Latency
0	0	0	5
0	0	1	6
0	1	0	7
0	1	1	8
1	0	0	9
1	0	1	10
1	1	0	11
1	1	1	12

9.4. Mode Register MR3

The mode register MR3 is implemented in the DDR3 model. The model supports the following parameter values.

15-3	**2**	**1**	**0**
0	**MPR**	**MPR Location**	

Bit 2	MPR 1
0	Normal Operation
1	Dataflow from MPR

Bit 1	Bit 0	MPR Location 1
0	0	Predefined Pattern
0	1	Reserved
1	0	Reserved
1	1	Reserved

10. Commands

The DDR3 model provides command support as indicated in the table below.

Command	Abbreviation	Command Accepted	Functionality Supported
Mode Register Set	MRS	Yes	Yes
Refresh	REF	Yes	Yes
Self Refresh Entry	SRE	Yes	No
Self Refresh Exit	SRX	Yes	No
Single Bank Precharge	PRE	Yes	Yes
Precharge all Banks	PREA	Yes	Yes
Bank Activate	ACT	Yes	Yes
Write (Fixed BL8 or BC4)	WR	Yes	Yes
Write (BC4, on the Fly)	WRS4	Yes	Yes
Write (BL8, on the Fly)	WRS8	Yes	Yes
Write with Auto Precharge (Fixed BL8 or BC4)	WRA	Yes	Yes
Write with Auto Precharge (BC4, on the Fly)	WRAS4	Yes	Yes
Write with Auto Precharge (BL8, on the Fly)	WRAS8	Yes	Yes
Read (Fixed BL8 or BC4)	RD	Yes	Yes
Read (BC4, on the Fly	RDS4	Yes	Yes
Read (BL8, on the Fly)	RDS8	Yes	Yes
Read with Auto Precharge (Fixed BL8 or BC4)	RDA	Yes	Yes
Read with Auto Precharge (BC4, on the Fly)	RDAS4	Yes	Yes
Read with Auto Precharge (BL8, on the Fly)	RDAS8	Yes	Yes
No Operation	NOP	Yes	Yes
Device Deselected	DES	Yes	Yes
Power Down Entry	PDE	Yes	No
Power Down Exit	PDX	Yes	No
ZQ Calibration Long	ZQCL	Yes	No
ZQ Calibration Short	ZQCS	Yes	No

11. Initialization Sequence

The DDR3 model requires that the memory controller follows 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 Mode Register 2
6. Write to Mode Register 3
7. Write to Mode Register 1
8. Write to Mode Register 0
9. Issue ZQC command

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.

12. Compile and Emulation

The model is provided as protected RTL files (*.vp). The protected RTL file needs to be synthesized prior to the back-end Palladium compile. An example of the command for compilation (including synthesis) of this model in the IXCOM flow is shown below.

```
ixcom -64 +sv –ua \
      +dut+<mod_name> \
      <mod_name>.vp \
      tb.v \
      -top tb \
      -incdir ../../../utils/cdn_mmp_utils/sv \
      ../../../utils/cdn_mmp_utils/sv/cdn_mmp_utils.sv \
      ……

xeDebug -64 --ncsim \
  -sv_lib ../../../utils/cdn_mmp_utils/lib/64bit/libMMP_utils.so -- \
 -input auto_xedebug.tcl
```

The scripts below show two examples for Palladium classic ICE synthesis:

1)
```
hdlInputFile -add <mod_name>.vp
hdlImport -full -2001 –l qtref
hdlOutputFile -add -f Verilog <mod_name>.vg
hdlSynthesize -memory -keepRtlSymbol -keepAllFlipFlop <mod_name>
……
```

2)
```
vavlog      <mod_name>.vp \
vaelab      -keepRtlSymbol -keepAllFlipFlop -outputVlog <mod_name>.vg
<mod_name>
```

13. SWI Smart Memory Interface

This DDR3 core model (ddr3_pd.vp) supports a fixed, Verilog macro controlled interface, or “hooks”, to one of the Smart Memory components in Cadence’s Software Integrator (SWI) product. Software Integrator (SWI) is a support library that is part of Cadence’s Hybrid Solution-- a multi-tool system level solution that runs in a hybrid PXP + IES simulation mode and targets the increasing number of SoC performance conscious projects requiring the booting of commercial operating systems and the running of complex software-driven tests against an accurate model of the SoC prior to tapeout.

For additional details about the SWI product at large please consult the SWI product documentation which includes a user guide. This documentation can be accessed via support.cadence.com and is located on the Product Pages/Product Manuals link where SWI 13.1 is located with other Functional Verification products.

The user of the SWI solution who is integrating this core model to the corresponding SWI Smart Memory component side should define the Verilog macro “MMP_SM” to enable the Smart Memory interface. This will enable the portion of the interface that resides in the MMP model core, thus completing access to the implemented SWI Smart Memory functionality.

Table 2 : SWI Smart Memory Verilog Define

`define Macro purpose	Possible `define Values
Set the Smart Memory interface to compile “in” as part of the memory model	MMP_SM

The SWI Smart Memory interface includes the signals shown in Table 3: SWI Smart Memory Interface Signals. It is outside the MMP scope to treat the integration of the MMP model into a hybrid solution. For additional details, please consult the SWI documentation and other Hybrid Solution documentation.

Table 3: SWI Smart Memory Interface Signals

NAME	DECLARATION	DESCRIPTION
sm_raddr	output [(total_addr_bits-1):0] sm_raddr	Smart Memory read address
sm_re	output sm_re	Smart Memory read enable
sm_raddr_en	output sm_raddr_en	Smart Memory read address enable
sm_dout	input [data_bits-1:0] sm_dout	Smart Memory read data
sm_waddr	output [(total_addr_bits-1):0] sm_waddr	Smart Memory write address
sm_we	output sm_we	Smart Memory write enable
sm_waddr_en	output sm_waddr_en	Smart Memory write address enable
sm_din	output [data_bits-1:0] sm_din	Smart Memory write data
sm_bw	output [data_bits-1:0] sm_bw	Smart Memory data mask
verbosity	input [3:0] verbosity	Smart Memory debug info display level control

The Smart Memory interface includes a user adjustable parameter that passes user data from wrapper layers that are external to this MMP model into MMP model. The single 64-bit parameter is subdivided by field to accommodate data as shown in Table 4: SWI Smart Memory User Adjustable Parameters below. This parameter defaults to a value of ‘0’ and is managed by the SWI product Smart Memory component. For additional details, please consult the SWI documentation and other Hybrid Solution documentation.

Table 4: SWI Smart Memory User Adjustable Parameters

PARAMETER	DESCRIPTION
parameter [63:0] SMConfigSpecific_UserData	Smart Memory User Data Passing
FIELD	**DESCRIPTION**
[2:0]	Used for specifying device/channel/subpart
[63:61]	Extension value of total_addr_bits

The Smart Memory interface includes non-user adjustable localparams and parameters as shown in Table 5: SWI Smart Memory Non-User Adjustable Parameters below. Note that a localparam of the same name (total_addr_bits) but of different size is part of the standard MMP core model.

Table 5: SWI Smart Memory Non-User Adjustable Parameters

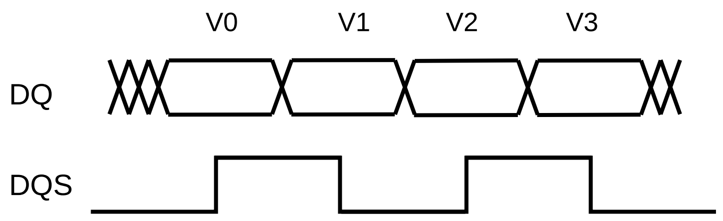
LOCALPARAM	VALUE	DESCRIPTION
total_addr_bits	bank_addr_width+row_addr_width +col_addr_width + SMConfigSpecific_UserData[63:61]	Memory capacity width

The SWI Smart Memory interface has dependencies on the inclusion of external file(s). For additional details about purpose and content of any Smart Memory related external files, please consult the SWI documentation and other Hybrid Solution documentation.

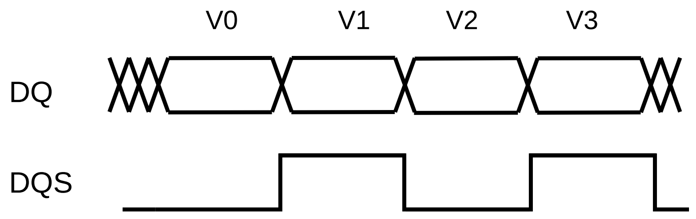
```
`ifdef MMP_SM
   `include "cdn_sm_mapDRBCToLinAdr.vh"
`endif
```

14. 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.

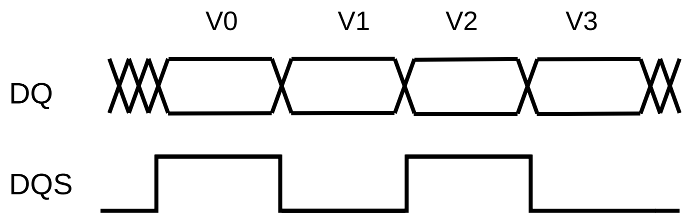


For DDR models provided by Cadence for Palladium, if the design drives DQ and DQS signals with the above timing, the DDR memory will behave correctly. However, to obtain this timing in Palladium, the fastest design clock must toggle twice as frequently as the DQS signal. If this faster clock is not needed 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
```

15. Configuration Parameters

There are a few parameters that can be set to adjust the configuration of a model. One limitation is that the sum of bank_addr_width, row_addr_width, and col_addr_width should be less than 32 bits due to the 2G address limit in HDL-ICE. The following parameters are available in the protected rtl version (<model_name>.vp file) of a model:

```
parameter    data_bits = 16;  // width of DQ bus
parameter    addr_bits = 13;  // row address width
parameter    bank_addr_width = 3;  // bank address width
parameter    row_addr_width = addr_bits;  // row address width
parameter    col_addr_width = 10;  // column address width
parameter    byte_width = 8;   // byte_width = 4 if DQ width is 4; otherwise byte_width = 8
parameter    tWL0_in_clocks = 3;   // tWL time in number of clocks
```

16. Protected RTL (*.vp) COMPILE of RDIMMs and UDIMMs

The RDIMM models and UDIMM models are wrapper models that typically instantiate one or more submodules. For example, the model *mt9jsf12872pz_rdimm,* which is listed in the MMP catalogue in the Micron DDR3 RDIMM list, instantiates modules *mt9jsf12872pz* and *sste32882*.However, for the protected RTL version of the models (*.vp), the user must determine which submodules are instantiated (by reviewing the model RTL (sub modules are instantiated at the bottom of the file) and then script the compile to include these instantiated modules. The user may find the requisite sub modules in the same release directory as the protected RTL (*.vp) for the primary model.

17. MMP and ECC (Error Correcting Code)

MMP models do not support Error Correcting Code (ECC) functionality. ECC functions, if they are present in a memory device, are typically found in the NAND and DDRx families. The MMP product does not have any plans to provide such functions in the models. MMP models are provided as system level emulation models and not as verification IP. The below sections discuss work-arounds that enable the user to deal with some ECC scenarios. Note that ECC means different things to different device families.

17.1. DDRx and ECC (Error Correcting Code)

Error Correction Code (ECC) allows single bit errors to be corrected and other bit errors to be detected, thus improving high frequency operation reliability and data accuracy. While the MMP DDRx models do not include any ECC functionality or handling, the user can arrange to “support” ECC on the DDR model first by using a model with a 72bit datapath and, additionally, by artificially injecting errors using some external mechanism, if such is needed. To word this differently, MMP models can often be found that interface with the memory controller data path, thus satisfying connectivity requirements. This arrangement might support a user who needs to test ECC related error conditions in emulation. The following paragraphs provide some details for consideration.

For DDRx models, ECC is performed by the controller. The controller calculates an ECC value and writes it to the upper 8 bits of a 72bit DIMM. When the controller writes data or when it reads data, it re-calculates the ECC based on current read data then checks the resulting value against the read ECC stored in the upper 8 bits. If the re-calculated value is equivalent to the stored value, then there is no error.

So, to support ECC in an MMP model during acceleration or emulation, the user first needs a 72bit data path to provide the added ports and data path for handling the upper 8 bits of ECC above and beyond the standard 64bit data path. In other words, a normal DDR memory has a 64bit data bus where the ECC memory will have a 72bit data bus and 9 bits for DQS and DM. For most scenarios, ones where the user will not enable ECC function in their controller and will not inject errors, this operation is compatible for use with MMP models. The expanded ports and data path constitutes all of the support that MMP models can provide toward ECC handling.

There are two potential paths for achieving the 72bit data bus. The first is to select and use a 72bit DIMM. Please review the DIMM pages of the MMP catalogue for an appropriate 72bit DIMM. If an appropriate 72bit part cannot be found, the user may contact Cadence support to initiate a request for a 72bit data path version of the model of interest. A second approach to consider for some models is to expand the data path of a smaller width data bus to 72bits. The safest way to expand the data path to 72bits is to modify the data width parameter to 72 in the standard 64bit model. The user can set the parameter *data_bits* to 72 when instantiating the model. Not all models, however, have a *data_bits* parameter available for configuration.

Next, the user considers the error injection aspect. There is no capability in MMP models for error injection. MMP models are provided as system level emulation models and not as verification IP. For specifically testing error conditions, the user needs to work around the gap. One alternative that the user might consider is to corrupt memory data in order to mimic data corruption. In this scenario, the user can execute the xeDebug memory command to write some incorrect data to the array. For example, the following command will corrupt one location in an array:

```
        memory -setvalue ddr3_inst -value 0x01234567 -start 0x10000100 -end
0x10000100
```

Another alternative is to consider using MMAV/Denali simulation models in IUS.

18. Large Memory Support

MMP model capacity has been limited to 1G words due to the current 30 bit address width limitation in IXCOM. To work around this constraint, a multiple core memory array generator (mmp_gen_mem.vp) has been incorporated into these large sized memory models that splits larger core memory arrays into multiple 1G arrays. In these cases, the file mmp_gen_mem.vp automatically defines Verilog macro MMP_LG_MEM_BITS with a value of ‘30’ by default. Files mmp_gen_mem.vp and mmp_submem.vp are located in the sdram/common sub-directory under the MMP installation.

18.1 Preloading Multiple Arrays

With a single array of 30 bits or less the address mapping is simply:array_addr = {BA,ROW,COL}. In this case the hierarchical memory array name appears thusly:

```
tb_top.rtl_module.ddr3_i0.memcore
```

In scenarios with array address space that is greater than 30 bits, there are multiple arrays which are mapped using distinctly different hierarchical naming. The hierarchical path for the array names associated with each core memory array of the large memory are reported as output to the "memory –list" command or can be viewed in the dbFiles/xcva_top_et5mpart file. For example, with a 32 bit address the user will see 4 arrays with naming as follows:

```
tb_top.rtl_module.ddr3_i0.mem1.\multiple.array_0_.u1 .memcore addresses 0  ..  1G - 1
tb_top.rtl_module.ddr3_i0.mem1.\multiple.array_1_.u1 .memcore addresses 1G .. 2G - 1
tb_top.rtl_module.ddr3_i0.mem1.\multiple.array_2_.u1 .memcore addresses 2G .. 3G - 1
tb_top.rtl_module.ddr3_i0.mem1.\multiple.array_3_.u1 .memcore addresses 3G .. 4G – 1
```

Multiple data files for preloading each separate memory array are also required. In the example above, there will be four data files needed to preload the entire large memory.

Likewise, multiple memory –load commands are needed to preload the large memory of this example. An xeDebug preload example for the case above will look as follows:

```
memory -load %readmemh
tb_top.rtl_module_w.rtl_module.ddr3_i0.mem1.multiple.array_0_.u1.memcore -file mem0.dat

memory -load %readmemh
tb_top.rtl_module_w.rtl_module.ddr3_i0.mem1.multiple.array_1_.u1.memcore -file mem1.dat

memory -load %readmemh
tb_top.rtl_module_w.rtl_module.ddr3_i0.mem1.multiple.array_2_.u1.memcore -file mem2.dat

memory -load %readmemh
tb_top.rtl_module_w.rtl_module.ddr3_i0.mem1.multiple.array_3_.u1.memcore -file mem3.dat
```

18.2 Models with Multiple Arrays

Not every DDR3 model exceeds the 30 bit IXCOM memory limit. Below is a list of models with address width exceeding 30 bits and the number of arrays generated in the core memory.

Model Name	BA	ROW	COL	Total bits	Arrays
mt41k2g4	3	16	12	31	2

18.3 IXCOM Compilation

For large memory models that exceed 30 bits of address width, include the files mmp_gen_mem.vp and mmp_submem.vp in the build as shown below. This file will incorporate the multiple core memory array generator into the build and the Verilog macro MMP_LG_MEM_BITS will automatically be defined.

```
vlan      -64bit +sv tb.v \
              <model_name>.vp \
              mmp_gen_mem.vp mmp_submem.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
```

Note: Files mmp_gen_mem.vp and mmp_submem.vp are located in the sdram/common sub-directory under the MMP installation.

18.4 IUS Compilation

In IUS the address width limit is less than 30 bits. Even when the Verilog sparse directive is used in the model sometimes large data width may cause an error. If this data width error occurs, the user may override the default array generator bit setting to a smaller number to work around this issue. The macro named MMP_LG_MEM_BITS is set to 30 by default. It may be changed within the file mmp_gen_mem.vp or overridden on the command with the define option. The user should note that more core memory arrays will be generated when the MMP_LG_MEM_BITS bit setting is reduced, which may then require additional data files for preloading the memory.

19. Debugging

This model has several debugging options, techniques and tips that may assist the user may use in isolating a problem.

- For issues that may not be DDR4 specific please review the *Memory Model Portfolio FAQ for All Models User Guide.*

- **Debug signals:** The following key signals can be monitored to help detect incorrect sequence:
  - key signals:

cmd	Current commands
WL	Write latency
RL	Read latency
init_done	Indicates whether initialization is completed
mwe	Window of the written data
dout_window	Window of the read out data

- **Golden waveform:** A package with a reference waveform is available which shows the following command sequence(s):

(1) command sequence:
   setting MR2
   setting MR3

setting MR1
setting MR0 // initialization complete
-> normal READ and WRITE on FLY mode

- **Debug Display:** This MMP memory model has available a built-in debug methodology called MMP Debug Display that is based on the Verilog system task $display. Please see the *Palladium Memory Model Debug Display User Guide* in the release docs directory for additional information.

- **Manual Configuring of this MMP Model Family**

  This MMP model supports manual configuration by accompanying the model mode register or configuration register declarations with synthesis directives, such as keep_net directives, that instruct the compiler to ensure that the relevant nets remain available for runtime forcing. For a general description of this support please see the user guide in the MMP release with path and filename *docs/MMP_FAQ_for_All_Models.pdf*.

  While MMP strongly recommends following protocol based commands to configure MMP models, MMP recognizes that the design test environment may desire to trade off the risks inherent in streamlining or circumventing the initialization sequence part of the protocol in order to better support some testing environments.

  The following table lists the internal register path and naming along with the specification or datasheet naming for model mode registers or configuration registers that are accompanied by keep_net synthesis directives in support of such manual configuration. ONLY writeable configuration registers or fields are supported thusly. Please read the relevant datasheet for details about individual register behavior and mapping to fields.

Table: Writeable Mode Register / Configuration Register Info

Hierarchical RTL Naming for Writeable Configuration Related Registers & Signals	Specification or Vendor Datasheet Naming for Configuration Related Registers	Access
<model_name>.MR0	MR0	W
<model_name>.MR1	MR1	W
<model_name>.MR2	MR2	W
<model_name>.MR3	MR3	W
<model_name>.init_done	[Not Applicable]	1’b1 indicates initialization is complete

20. Revision History

The following table shows the revision history for this document.

Date	Version	Revision
March 2010	1.0	Initial Release
July 2010	1.1	Corrected the initialization sequence description
June 2011	1.2	Added release levels description
March 2012	1.3	Added all supported CAS latency, fixed typos
April 2014	1.4	Added a section on Configuration Parameters.
May 2014	1.5	Added +define+MMP_RBC to select {row,bank,col} addressing.
June 2014	1.6	Added Write Leveling to list of params that are “not applicable” to DDR3 model.
July 2014	1.7	Added info about .vp file compile (usually IXCOM) with sub modules
July 2014	1.8	Repaired doc property title.
September 2014	1.9	Remove version from UG file name. Update UXE / IXE documentation reference titles. Updated Configuration section with info about multiplexed bits with non-address functions. Added paragraph about flow independent delay cell in “Handling DQS …” section.
November 2014	1.10	Remove emulation capacity info.
December 2014	1.11	Added Write Leveling, Output buffer disable (MR #1). Added Read Leveling (MR #3 MPR, MPR Location). Added tWL0_in_clocks parameter description
January 2015	1.12	Added MMP and ECC section. Added tables describing support of features, commands more clearly. Update related publications list.
March 2015	1.13	Add section on SWI Smart Memory interface
April 2015	1.14	Add section with tables for adjustable parameters and non-adjustable localparams for debug
July 2015	1.15	Document large memory support.
July 2015	1.16	Update Cadence naming on front page
August 2015	1.17	Replaced gen_mem.vp with mmp_gen_mem.vp and added mmp_submem.vp.
September	1.18	Modify compile notes to reflect *.vp as sole model format
January 2016	2.0	Update for Palladium-Z1 and VXE
April 2016	2.1	Update SM interface sm_we
July 2016	2.2	Remove hyphen in Palladium naming
December 2016	2.3	Add “Manual Configuring of DDRx/LPDDRx” section
April 2017	2.4	Update for new SM interface

Date	Version	Revision
May 2017	2.5	Move mmp_gen_mem.vp and mmp_submem.vp files to sdram/common sub-directory under MMP installation
October 2017	2.6	Modify “bank” vs “ba” references in Address Mapping section.
January 2018	2.7	Modify header and footer
June 2018	2.8	Update Manual Configuring description in Debugging section
July 2018	2.9	Update for new utility library