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

Palladium Memory Models:
Managing Large Memory Models
User Guide

Document Version: 1.4

Document Date: July 2016

Contents

1. 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

2. Managing Large Memory Models

1. Introduction

The Cadence Palladium Memory Model Portfolio has several model families that include large memory configurations that can interact with Palladium and IES address bit limits. Since these interactions elicit standard warning and error messages, have platform wide limits, and enjoy standard sparse memory approaches for managing these MMP model configurations, this standalone user guide provides some details in this area.

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 and may not have documentation. Model may contain defects.	No	Yes	Yes

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

3. Current Address Bit Limits and Sparse Mode Approaches

The Cadence Palladium Memory Model Portfolio has several model families that include large memory configurations that can interact with Palladium and IES address bit limits. These limits depend on flow and tool (Palladium acceleration or IXCOM, Palladium emulation or Classic ICE, simulation with IES). A sparse memory, or sparsely used memory, is a memory instance declared with a large/deep address range but where only a small percentage of the address range is accessed during simulation or emulation. Both Palladium flows and the IES flow support sparse memory. Enabling a large MMP memory configuration as a sparse memory, if appropriate to user's goals, may reduce the amount of memory required and thus constrain that large memory to fit within available resources and limits.

3.1. Classic ICE Flow

In Classic ICE mode, a maximum of 63 address bits are supported at compile time and the limit of memory support is defined by the available emulator resources during the run time. The synthesis phase address bit limit of the Classic ICE flow is 2^31. In ICE flow, if you know that certain memories in your design are sparsely used, you can use the *memoryTransform* command at compile time to declare these memories as "sparse". Sparsely-used memories must be declared in this flow using the XEL based *memoryTransform* command as follows:

```
memoryTransform -add {<name> SPARSE <n> <p>}
```

where,
<name> is a hierarchical name of the memory
<n> is used to calculate the number of physical pages. Number of physical pages = 2n
<p> is used to calculate the size of each page. Number of words per page = 2p

For additional details please refer to the *UXE User Guide* in the section "Handling Sparse Memories" and subsection of "Modeling Sparsely-Used Memories at Compile Time in ICE Model" where the ICE approach to sparse memory is detailed.

3.2. IXCOM (Simulation Acceleration) Flow

Since the *memoryTransform* command discussed above for ICE flow implements sparse support only in the hardware emulator but IXCOM (simulation acceleration) requires sparse support in both the hardware emulator and in the software simulator, a different approach to managing large memories is implemented here.

Additionally, for IXCOM, or SA mode, the MMP models are constrained by both an IES address bit limit and an IXCOM address bit limit since IXCOM leverages the IES simulator. The IES address bit limit is currently 2^27 bits under standard conditions and 2^31 with a sparse memory flow.

The IES sparse memory support for Verilog is based upon the Verilog /*sparse*/ pragma. Additionally the user can specify that all arrays over a specified size are to be treated as sparse by IES by using the *–sparsearray* option when invoking the elaborator (*ncelab* –

sparsearray) where the option takes an positive number argument indicating the number of array elements as shown below:

```
% ncelab -sparsearray 1000000 worklib.top
```

Please see the IES user guide on Verilog modeling for additional details.

The IES sparse memory support for VHDL is based upon specifying that all arrays over a specified size are to be treated as sparse by IES by using the –vhdlsparsearray option when invoking BOTH the compiler and elaborator. Again, the option takes a positive number argument indicating the number of array elements as shown below. Currently this vhdl sparse array option is only required for QSPI memory model n25q00a, which is 1Gb in size.

```
ixcom -ua -top tb +dut+n25q00a –ncelabargs "-
vhdlsparsearray 1000000" –ncvhdlargs "-vhdlsparsearray
1000000"
```

The IXCOM address bit limit is currently 2^29 bits under standard conditions and 2^30 with a sparse memory flow.

Since the IXCOM address bit limits sit between the IES address bit limit range for normal and sparse flows careful management of the larger MMP configurations (especially, but not exclusively, those models configured in the 16GB and 32GB or larger sizes). Note that the IXCOM compile for these larger MMP model configurations may need sparse support to successfully complete the initial IES compile but may then need, later in the same build, independently and differently enabled support for sparse memory within IXCOM flow.

To ease the user's path in this area, the MMP model deliverables for large configurations are already generated with a core memory declaration that uses the Verilog, and IES supported, /* sparse */ pragma. An example is shown below:

```
reg [MEMCORE_DATA_HIGH:0]   mem [0:MEMCORE_END_ADDR];   /* sparse */
```

It is expected that this IES pragma support will support the user up to the IES spare limit of 2^31 bits. What the embedded /*sparse */ pragma does NOT currently do is support the user up to the IXCOM limit of 2^30 bits. If the user hits the standard IXCOM address bit limit of 2^29 the IXCOM compile will elicit a *Error* message stating the memory width is too wide. An example error is shown below. At this point, the user will want to consider if the IXCOM sparse memory approach is useful for their design and validation context.

```
Error! in file <path>/<model_name>.vp at line #      [WIDTH_TOO_LARGE]
     Size of vector greater than (2147483647) is not supported
```

The IXCOM approach to sparse support is based on the $ixc_ctrl system task and has the following syntax:

```
$ixc_ctrl("sparse_mem", <memory_name>, <num_pages>, <page_size>);
```

where the following definitions apply:
<*memory_name*> is the reference to memory to be implemented as sparse.
<*num_pages*> is the number of pages of memory accessed by the testcase.
<*page_size*> is the size of each memory page (unit: memory word / address).
Note: <*num_pages*> and <*page_size*> are rounded up the nearest power of 2.

Because the IXCOM sparse implementation has limitations and requires careful calculation of inputs, the user should consult the *UXE User Guide* in section “Support for Sparse Memories in SA Mode” and review the details of this implementation before applying this command to an MMP model.

The $ixc_ctrl (sparse_mem” …) command supports hierarchical memory names so it is also recommended that the user place this system task in a wrapper to the core memory; however, placing the command within an unprotected area of the core memory model deliverable will also work.

The user may also want to note, in case this utility is being used, that the UXE User Guide describes IRUN support for the /* sparse */ pragma which the IXCOM compiler recognizes.

There is currently nothing in place within IXCOM that will support the user beyond the 2^30 bit limit.

4. Controlling IXCOM Sparse Memory Usage

The MMP user may desire more control over sparse memory usage than is available with the simple command usage suggested above. For example, the user may have multiple flows and want to control when the IXCOM system task $ixc_ctrl (sparse_mem” …) comes into play. This may be accomplished using some combination of Verilog macro `ifdef structuring. As an example that takes advantage of the pre-defined Verilog macro IXCOM_UXE:

```
`ifdef IXCOM_UXE
`ifdef MT9JSF25672AZ_UDIMM_SPARSE
initial begin
  `ifndef MT9JSF25672AZ_UDIMM_NUM_PAGES
    `define MT9JSF25672AZ_UDIMM_NUM_PAGES 32
  `endif
  `ifndef MT9JSF25672AZ_UDIMM_PAGE_SIZE
    `define MT9JSF25672AZ_UDIMM_PAGE_SIZE 1024
  `endif
  $ixc_ctrl(?sparse_mem?, memcore, `MT9JSF25672AZ_UDIMM_NUM_PAGES,
`MT9JSF25672AZ_UDIMM_PAGE_SIZE);
end
`endif
`endif
```

Here, since IXCOM_UXE only takes effect for the IXCOM flow, the user can make the model sparse but override the default with their own values.

3. Revision History

The following table shows the revision history for this document

Date	Version	Revision
September 2014	1.0	Initial release
March 2015	1.1	Update related publications list.
July 2015	1.2	Update Cadence naming on front page
January 2016	1.3	Update for Palladium-Z1 and VXE
July 2016	1.4	Remove hyphen in Palladium naming