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

eMMC 5.0
Palladium Memory Model
User Guide

Document Version: 3.8

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

eMMC 5.0 Palladium Memory Models

1. Introduction

The Cadence Palladium MMC Model is based on data sheet specifications of the following JEDEC Specification:

Embedded MultiMediaCard(*e*•MMC), Electrical Standard (5.0)
Document Number: JEDEC JESD84-B50 (September 2013)

The model can be configured to support different configurations of sizes and other features to match real devices manufactured by various vendors.

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. Features

- JEDEC eMMC 4.3 – Support for majority of commands and functions
- JEDEC eMMC 4.4 – Support for DDR mode of operation
- JEDEC eMMC 4.5 – Support for HS200 mode of operation
- JEDEC eMMC 5.0 – Support for HS400 mode of operation
- Model can be customized with many configurable parameters
- Support for two Boot Partitions and Alternative Boot Mode
- Support for I/O Mode commands
- Support for multiple eMMC models on a single bus
- RST_N pin support
- Support for large 4KB sectors
- Packed Command support
- Discard, Trim and Sanitize commands support
- Context Management support
- Data tags support
- RPMB support (excluding HMAC check)
- Dynamic Capacity Management support
- Boot area write protection support
- HPI support
- Cache support

The following features are NOT supported in the Palladium MMC model.

- SPI model is not supported

4. Model Pin Description

In addition to the standard IO pins for the MMC as per the JEDEC specification there are several other IO pins on the MMC Palladium Memory Model that are there to provide flexibility and to support additional model features.

Typically, but not always, a controller will support pullups on the CMD and DAT buses. For this model there is a Verilog macro to add pullups for the case where a controller which DOES NOT support pullups. The Verilog macro is ***MMP_EMMC_HOST_NO_PULLUP***. If there is no pullup on CMD and DAT bus in the user's test environment (out of model), the user should define this Verilog macro.

Pin name	Direction	Description
CLK	Input	MMC CLK input
CMD	Inout	MMC CMD input
DAT0…DAT7	Inout	MMC 8bit bi-directional databus
RST_N	Input	MMC Reset pin
DATA_STROBE	Output	Strobe signal according to read out data
INT_CLK	Input	Internal clock required for MMC model operation. Frequency needs to match the parameter INT_CLK_CYCLE_TIME_IN_NS
CID_IN[127:0]	Input	Initial value for CID register. This value is latched by the model 2 clks after start of emulation
CID_OUT[127:0]	Output	Current value of the CID register
CSD_IN[127:0]	Input	Initial value for CSD register. This value is latched by the model 2 clks after start of emulation
CSD_OUT[127:0]	Output	Current value of the CSD register
PWD_IN[127:0]	Input	Password input bus
PWD_LEN_IN[7:0]	Input	Input Password length in bytes
PWD_OUT[127:0]	Output	Current value of the Password
PWD_LEN_OUT[7:0]	Output	Current length of the Password in bytes
PWD_ERASED	Output	Pulsed output indicating the Password was erased
FAST_IO_WE	Output	Fast IO write enable. Used for CMD39.
FAST_IO_RE	Output	Fast IO read enable. Used for CMD39.
FAST_IO_ADDR[6:0]	Output	Fast IO address. Used for CMD39.
FAST_IO_DATA_OUT[7:0]	Output	Fast IO write data. Used for CMD39.
FAST_IO_DATA_IN[7:0]	Input	Fast IO read data. Used for CMD39.
IRQ	Input	Used for the GO_IRQ_STATE command.
IRQ_DATA[15:0]	Input	IRQ response data field for an R5 response
IRQ_SERVED	Output	Asserted during R5 response
IRQ_REJECTED	Output	In multiple MMC environments IRQ_REJECTED indicates that the device received the IRQ signal condition but was unable to send the R5 response as either another device was already responding or the host was in the process of sending a command to abort the Wait-IRQ-State.
CS_CARD_ECC_FAILED_IN	Input	When asserted this will set the CARD_ECC_FAILED bit in the Card Status field of an R1 response.
CS_ERROR_IN	Input	When asserted this will set the ERROR bit in the Card Status field of an R1 response.

CS_UNDERRUN_IN	Input	When asserted this will set the UNDERRUN bit in the Card Status field of an R1 response.
CS_OVERRUN_IN	Input	When asserted this will set the OVERRUN bit in the Card Status field of an R1 response.
INJECT_CRC_ERROR_IN	Input	When asserted the generated CRC for a read command will have an incorrectly calculated CRC
ERASE_CLK	Input	In order to perform the erase function the model must write to each location of the internal memory. This takes one clock cycle per word. ERASE_CLK has been provided to allow the user to connect a high speed clock.

5. Model Parameter Descriptions

The following table provides details on the **user adjustable** parameters for the Palladium MMC Memory Model. These parameters may be modified when instantiating an MMC 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
ADDR_WIDTH	32	Address bit width defines memory size as (1 << ADDR_WIDTH). Should match the configuration's general purpose partition size; if not, the ADDR_WIDTH will be mismatched
GP1_ADDR_WIDTH	12	General purpose partition 1 size = (1<<GP1_ADDR_WIDTH) bytes
GP2_ADDR_WIDTH	12	General purpose partition 2 size = (1<<GP2_ADDR_WIDTH) bytes
GP3_ADDR_WIDTH	12	General purpose partition 3 size = (1<<GP3_ADDR_WIDTH) bytes
GP4_ADDR_WIDTH	12	General purpose partition 4 size = (1<<GP4_ADDR_WIDTH) bytes
MEMCORE_DATA_WIDTH_MULT_LOG2	3	log2 of memory core data width multiplier. Memory core data width = 8 * (1 << MEMCORE_DATA_WIDTH_MULT_LOG2) For example: // 0 means 8 * (1 << 0) = 8bit // 1 means 8 * (1 << 1) = 16bit // 2 means 8 * (1 << 2) = 32bit // 2 means 8 * (1 << 2) = 32bit
BOOT_SIZE_MULT_LOG2	0 in hdl; 5 in wrapper	Boot partition size = 128Kbytes * (1 << BOOT_SIZE_MULT_LOG2)
RPMB_SIZE_MULT_LOG2	0 in hdl; 2 or 5 in wrapper, dep. on size	Replay Protected Memory Block partition size = 128KBytes * (1 << RPMB_SIZE_MULT_LOG2) Note that specification indicates this value should be 2 in 4/8GB cards and 5 in 16/32/64GB.
CACHE_MEMCORE_ADDR_WIDTH	13	Shall be log2(CACHE_MEMCORE_SIZE x 1024 / MEMCORE_DATA_WIDTH)
CACHE_MEMCORE_SIZE	512	Size of cache memory is CACHE_MEMCORE_SIZE x 1kbit, shall be the same value as CACHE_SIZE of EXT_CSD[252:249]
SECTOR_BYTE_N_ACCESS	1	Selects SECTOR = 1 or BYTE = 0 access mode.
MIN_WRITE_PROTECT_GROUP_SIZE	16384	Minimum write protect group size in bytes
BLK_LEN_MAX	9	Half of buffer size for receiving data, shall be the same value as MAX_WRITE_BL_LEN in CSD
TIC	74	Required initial power-up clocks
NID	5	Command end to response R2 or R3 start in clock cycles
NCR	2	Command end to response R1 start in clock cycles
NAC	8	Read data access time in clock cycles

NCD	56	boot_operation, NCD clocks required from CMD high to next emmc command

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**
R1b_READY_COUNT	64	R1b ready time in clock cycles
BOOT_ACK_COUNT	32	Boot acknowledge time in clock cycles
MEMFILE_LITTLE_ENDIAN	1	If 1, memory contents file is little endian. Byte lane is swapped. If 0, memory contents file is big endian. Byte lane is NOT swapped
INT_CLK_CYCLE_TIME_IN_NS	1000	INT_CLK cycle time in ns
USER_WRITE_RECOVERY_TIME_IN_NS	2000	Write Recovery time for User Data area
PACKED_HEADER_WRITE_RECOVERY_ TIME_IN_NS	2000	Write Recovery time for RPMB
TEST_WRITE_RECOVERY_TIME_IN_NS	2000	Write Recovery time for Bus Test Data
BOOT1_WRITE_RECOVERY_TIME_IN_NS	2000	Write Recovery time for Boot Area 1
BOOT2_WRITE_RECOVERY_TIME_IN_NS	2000	Write Recovery time for Boot Area 2
CSD_WRITE_RECOVERY_TIME_IN_NS	2000	Write Recovery time for CSD
WP_WRITE_RECOVERY_TIME_IN_NS	2000	Write Recovery time for Write Protect bit
PWD_WRITE_RECOVERY_TIME_IN_NS	1000	Write Recovery time for Password
CMD1_RESPONSE_TIME_IN_NS	40000	CMD1 busy to ready time in ns
INI_TIMEOUT_EMU_IN_100MS	100000000	100ms unit for calculating INI_TIMEOUT_EMU
SANITIZE_TIME_IN_NS	10000000	10ms waiting for busy state when sanitizing
DEFAULT_OCR_2V7_3V	9’h1ff	OCR register default values
DEFAULT_OCR_2V0_2V6	7’h0	OCR register default values
DEFAULT_OCR_1V70_1V95	1’b1	OCR register default values
PHYSICAL_ADDRESS_BOUNDARY_READ	2048	Physical address boundary for read, must be power of 2
PHYSICAL_ADDRESS_BOUNDARY_ WRITE	2048	Physical address boundary for write, must be power of 2
EXTRA_DDR_CYCLE	1	Controls alignment of the DDR relative to the data. When EXTRA_DDR_CYCLE is set, the first data is driven on the negedge of the clock so it can be latched by the host on the posedge of the clock.
SEND_EXT_CSD_ORDER	1	SEND_EXT_CSD byte order. With value of '0', ext_csd[511] is first. With value of '1', ext_csd[0] is first.

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.

6. Verilog Macro Defines

The following table lists the Verilog macro `define(s) available in the Palladium MMC model.

Macro Name	Purpose
MMP_MMC_READ_SAMPLE_DIRECTLY	User may set macro in order to sample read out data directly with CLK or DATA_STROBE for HS400 mode, refer to Section22 Read data sampling for detailed information.

7. Commands and Responses

The model supports the following commands.

Class 0	Basic	CMD0, CMD1, CMD2, CMD3, CMD4, CMD5, CMD6, CMD7, CMD8, CMD9, CMD10, CMD12, CMD13, CMD14, CMD15, CMD19
Class 1	Stream Read	CMD11
Class 2	Block Read	CMD17, CMD18, CMD21, CMD23
Class 3	Stream Write	CMD20
Class 4	Block Write	CMD16, CMD23, CMD24, CMD25, CMD26, CDM27
Class 5	Erase	CMD35, CMD36, CMD38
Class 6	Write Protection	CMD28, CMD29, CMD30, CMD31
Class 7	Lock Card	CMD16, CMD 42
Class 9	I/O Mode	CMD39, CMD40

The model supports all response types, R1, R1b, R2, R3, R4 and R5.

The following commands are not supported:

Class 9	I/O Mode	CMD55, CMD56

8. Clocks for the Model

The model main clock is the input port CLK. This should be connected to the eMMC output clock of the DUT/MMC controller.

An additional clock is also required to be connected to the model. The port INT_CLK is required to operate some internal functions of the eMMC model. The frequency of this clock is a default of 1MHz. The user should create an additional clock source in their top level environment and assign it to this INT_CLK pin.

If an alternative frequency clock is desired then this can be used in combination with the parameter INT_CLK_CYCLE_TIME_IN_NS. The frequency defined in the Palladium database for the clock connected to the INT_CLK pin must match the period defined with this parameter.

9. Synthesis and Compilation of the Model

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

```
ixcom –ua -64bit +sv \
      +dut+<selected_wrapper> \
      cds_cva_emmc.vp \
      <selected_wrapper>.v \
      tb.v \
      -top tb \
      ${UXE_HOME}/etc/ixcom/ixc_clkgen.v \
      -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 <selected_wrapper>.vp
hdlInputFile –add cds_cva_emmc.vp
hdlImport -full -2001 -l qtref
hdlOutputFile -add -f Verilog <selected_wrapper>.vg
hdlSynthesize -memory -keepRtlSymbol -keepAllFlipFlop
<selected_wrapper>
……
```

2)

```
vavlog      cds_cva_emmc.vp \
            <selected_wrapper>.v \
vaelab      -keepRtlSymbol -keepAllFlipFlop -outputVlog
<selected_wrapper>.vg <selected_wrapper>
```

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 MMC Palladium memory model requires an internal reference clock. This needs to be provided on the INT_CLK. This should be provided using the clockSource command during the Palladium compile. .

```
clockSource -add {<Path to Model>.INT_CLK}
clockFrequency -add {<Path to Model>.INT_CLK frequency}
```

10. Memory Configuration

The array size of the model can be determined by the wrapper chosen by users.

11. Memory Arrays in the MMC Model

There are a total of ten arrays in the MMC model. The primary memory arrays are the main array and two boot arrays. There are additional arrays for HS200 tuning, modeling of the General Purpose partitions and the RPMB partition. Also, an array is used for the modeling of the Cache function. These arrays can be preloaded at runtime with the Palladium runtime environment using the memory commands if the user doesn't want to load them via eMMC commands.

Array Name in Model	**Device Function**	**File Name in *memdata* directory**	**Purpose of Data File**
mem	Main Array	init.dat	Example
boot1mem	Boot Partition 1	boot1.hex	Example
boot2mem	Boot Partition 2	boot2.hex	Example
tune_pattern	Array containing fixed pattern for HS200 tuning sequence	tune.hex	Required use for HS200
gp1mem	General purpose partition 1	NA	NA
gp2mem	General purpose partition 2	NA	NA
gp3mem	General purpose partition 3	NA	NA
gp4mem	General purpose partition 4	NA	NA
rpmbmem	Replay protected memory block	rpmb.hex	Example; random data
cachemem	Array for cache function	NA	NA

The model is provided with example initialization files for some arrays. The main array and the two boot arrays are simple examples. The gpxmem do not have examples.

Users can use RPMB function without the hex file; this file is only used to skip the write operation to an RPMB array. The content of the provided example hex file is random data.

11.1. Address mapping to access memory arrays

Users generally need to pay attention to the address mapping when they access the main array “**mem**” as well as several other arrays with Palladium runtime load and dump commands. In the MMC model, the width of the following arrays needs consideration: mem; gp1mem; gp2mem; gp3mem; gp4mem; cachemem. The address mapping for load and dump commands can be different from that in the read/write commands. For models with programmable array widths, such as this model, the address mapping for accessing the core memory array can become complicated by the fact that even though the real memory device implements an 8bit array width, for the large memory model configurations the data need to be reformatted into 32 or 64 bit load and dump files for these runtime commands. This section explain the reason for this implementation and then suggests a procedure that alleviates the need for data modification.

There is a parameter “**MEMCORE_DATA_WIDTH_MULT_LOG2**” in this model's size/configuration wrappers which indicates the width of the real memory core implemented in Palladium. The width is:

width = 8bits x (2^ MEMCORE_DATA_WIDTH_MULT_LOG2)

For example, in the 1GB model, width = 8 x (2^3) = 64 bits. That means if users use write commands (CMD24/25) to access the memory with a logical start address like 0x1000, the actual physical address 0x200 (0x1000>>3) is accessed. Users therefore need to "dump" 0x200 to retrieve the written data instead of the expected 0x1000.

On the other hand, if the user issues mem –load for the physical address 0x200, the relevant read commands (CMD17/18) should access the core memory with the logical start address 0x1000 (0x200<<3). The data width for each address location in the preload file should be the same as the width of memcore.

Palladium runtime memory load and dump operations support various data formats which can vary the user's load/dump performance and data file content.

For programmable array widths that are larger than 8bits, the user has the option to take advantage of the Palladium utility memTran to convert a memory data file formatted in 8bits wide data—i.e., a readmemh format—to a headerless binary formatted file—i.e. raw2 format—that can be loaded via the memory –load command without the user needing to be aware of or concerned about the memory width.

The steps are as follows for an example using readmemh formatted input:

1. User creates a "readmemh" formatted, or other Palladium supported format," data file with 8bits wide data
2. User converts the readmemh file using Palladium's *memTran* utility into a file with "pd_raw2" format.
3. User loads the memory at runtime using the command 'memory –load %pd_raw2' to load the data file into memory. Because this command does not care about the memory width and simply streams the data into the assigned memory at the specified start address, the user can perform the memory access and load without modifying the data file from the standard readmemh format. The user does need to remain cognizant of the preload start address if the start address is not address 0. The start address is the physical address of the memcore that is a 32bit or 64bit memory address. The description of how to calculate the physical address for the memory was discussed above.

To look at a specific example, the user may have a memdata file called test_load.h that contains 8 bits wide data. The following memTran command converts the test_load.h file to raw2 formatted data (pd_raw2) in a file called test_load.bin and then compares the content of the two files.

```
memTran -translate %readmemh memdata/test_load.h %pd_raw2 test_load.bin -width 8 -depth 256
memTran -compare   %readmemh memdata/test_load.h %pd_raw2 test_load.bin -width 8 -depth 256
```

During runtime, the memory –load command can be used as below:

```
memory -load %pd_raw2 klm4g1yemd_b031_inst.i1.mem -file test_load.bin –nochecksize
```

And the memory –dump command as below:

```
memory -dump %pd_raw2 klm4g1yemd_b031_inst.i1.mem -file test_dump.bin
```

The dump command may be followed by the appropriate memTran commands to convert the binary formatted file back to 8-bit width format.

```
    memTran -translate %pd_raw2 test_dump.bin %readmemh test_dump.bin.postconv.h -width 64
    memTran -compare %pd_raw2 test_dump.bin %readmemh test_dump.bin.postconv.h -width 64 -depth
335
```

An additional consideration is that load/dump operations are typically faster with binary formatted files such as raw2 formats.

Please reference the UXE/VXE user guide and command reference manual for more detailed information regarding the memTran utility which translates between raw format and other file formats. Referencing the following sections, among others, may be helpful: “Compiling and Running Designs with Memories,” “Using Memory Streaming,” and “Translating Memory Format Using the memTran Utility.”

In the eMMC card model, with the exception of the model arrays mentioned above, which are **NOT** 8 bits, all other arrays are per device specification.

12. Device Registers

12.1. CSD register

The Device-Specific Data (CSD) register provides information on how to access the Device contents.

Field	***CSD-slice***	***Support***
CSD_STRUCTURE	[127:126]	Accept but ignore
SPEC_VERS	[125:122]	Accept but ignore
TAAC	[119:112]	Accept but ignore
NSAC	[111:104]	Accept but ignore
TRAN_SPEED	[103:96]	Accept but ignore
CCC	[95:84]	Accept but ignore
READ_BL_LEN	[83:80]	Support functionally
READ_BL_PARTIAL	[79:79]	Support functionally
WRITE_BLK_MISALIGN	[78:78]	Support functionally
READ_BLK_MISALIGN	[77:77]	Support functionally
DSR_IMP	[76:76]	Support functionally
C_SIZE	[73:62]	Accept but ignore
VDD_R_CURR_MIN	[61:59]	Accept but ignore
VDD_R_CURR_MAX	[58:56]	Accept but ignore
VDD_W_CURR_MIN	[55:53]	Accept but ignore
VDD_W_CURR_MAX	[52:50]	Accept but ignore
C_SIZE_MULT	[49:47]	Accept but ignore
ERASE_GRP_SIZE	[46:42]	Support functionally
ERASE_GRP_MULT	[41:37]	Support functionally
WP_GRP_SIZE	[36:32]	Support functionally
WP_GRP_ENABLE	[31:31]	Accept but ignore
DEFAULT_ECC	[30:29]	Accept but ignore
R2W_FACTOR	[28:26]	Accept but ignore
WRITE_BL_LEN	[25:22]	Support functionally

WRITE_BL_PARTIAL	[21:21]	Support functionally
CONTENT_PROT_APP	[16:16]	Accept but ignore
FILE_FORMAT_GRP	[15:15]	Accept but ignore
COPY	[14:14]	Accept but ignore
PERM_WRITE_PROTECT	[13:13]	Support functionally
TMP_WRITE_PROTECT	[12:12]	Support functionally
FILE_FORMAT	[11:10]	Accept but ignore
ECC	[9:8]	Accept but ignore
CRC	[7:1]	Accept but ignore

Note: 'Accept but ignore' means the value of the field in CSD register can be accepted and read by the MMC model but doesn't affect any function.

12.2. Extended CSD register

The Extended CSD register defines the Device properties and selected modes.

Field	***CSD-slice***	***Support***
EXT_SECURITY_ERR	[511:506]	Accept but ignore
S_CMD_SET	[504]	Accept but ignore
HPI_FEATURES	[503]	Support functionally
BKOPS_SUPPORT	[502]	Accept but ignore
MAX_PACKED_READS	[501]	Support functionally
MAX_PACKED_WRITES	[500]	Support functionally
DATA_TAG_SUPPORT	[499]	Support functionally
TAG_UNIT_SIZE	[498]	Accept but ignore
TAG_RES_SIZE	[497]	Accept but ignore
CONTEXT_CAPABILITIES	[496]	Accept but ignore
LARGE_UNIT_SIZE_M1	[495]	Accept but ignore
EXT_SUPPORT	[494]	Accept but ignore
CACHE_SIZE	[252:249]	Accept but ignore
GENERIC_CMD6_TIME	[248]	Accept but ignore
POWER_OFF_LONG_TIME	[247]	Accept but ignore
BKOPS_STATUS	[246]	Accept but ignore
CORRECTLY_PRG_SECTORS_NUM	[245:242]	Support functionally
INI_TIMEOUT_AP	[241]	Accept but ignore
PWR_CL_DDR_52_360	[239]	Accept but ignore
PWR_CL_DDR_52_195	[238]	Accept but ignore
PWR_CL_200_195	[237]	Accept but ignore
PWR_CL_200_130	[236]	Accept but ignore
MIN_PERF_DDR_W_8_52	[235]	Accept but ignore
MIN_PERF_DDR_R_8_52	[234]	Accept but ignore
TRIM_MULT	[232]	Accept but ignore
SEC_FEATURE_SUPPORT	[231]	Support functionally
SEC_ERASE_MULT	[230]	Accept but ignore
SEC_TRIM_MULT	[229]	Accept but ignore
BOOT_INFO	[228]	Support functionally
BOOT_SIZE_MULTI	[226]	Accept but ignore
ACC_SIZE	[225]	Accept but ignore
HC_ERASE_GRP_SIZE	[224]	Support functionally
ERASE_TIMEOUT_MULT	[223]	Accept but ignore
REL_WR_SEC_C	[222]	Support functionally
HC_WP_GRP_SIZE	[221]	Support functionally
S_C_VCC	[220]	Accept but ignore

S_C_VCCQ	[219]	Accept but ignore
S_A_TIMEOUT	[217]	Accept but ignore
SEC_COUNT	[215:212]	Accept but ignore
MIN_PERF_W_8_52	[210]	Accept but ignore
MIN_PERF_R_8_52	[209]	Accept but ignore
MIN_PERF_W_8_26_4_52	[208]	Accept but ignore
MIN_PERF_R_8_26_4_52	[207]	Accept but ignore
MIN_PERF_W_4_26	[206]	Accept but ignore
MIN_PERF_R_4_26	[205]	Accept but ignore
PWR_CL_26_360	[203]	Accept but ignore
PWR_CL_52_360	[202]	Accept but ignore
PWR_CL_26_195	[201]	Accept but ignore
PWR_CL_52_195	[200]	Accept but ignore
PARTITION_SWITCH_TIME	[199]	Accept but ignore
OUT_OF_INTERRUPT_TIME	[198]	Accept but ignore
DRIVER_STRENGTH	[197]	Accept but ignore
DEVICE_TYPE	[196]	Support functionally
EXT_CSD_REV	[192]	Accept but ignore
CMD_SET	[191]	Accept but ignore
CMD_SET_REV	[189]	Accept but ignore
POWER_CLASS	[187]	Accept but ignore
HS_TIMING	[185]	Support functionally
BUS_WIDTH	[183]	Support functionally
ERASED_MEM_CONT	[181]	Support functionally
PARTITION_CONFIG	[179]	Support functionally
BOOT_CONFIG_PROT	[178]	Accept but ignore
BOOT_BUS_CONDITIONS	[177]	Support functionally
ERASE_GROP_DEF	[175]	Support functionally
BOOT_WP_STATUS	[174]	Support functionally
BOOT_WP	[173]	Support functionally
USER_WP	[171]	Support functionally
FW_CONFIG	[169]	Accept but ignore
RPMB_SIZE_MULT	[168]	Accept but ignore
WR_REL_SET	[167]	Support functionally
WR_REL_PARAM	[166]	Support functionally
SANITIZE_START	[165]	Accept but ignore
BKOPS_START	[164]	Accept but ignore
BKOPS_EN	[163]	Accept but ignore
RST_n_FUNCTION	[162]	Support functionally
HPI_MGMT	[161]	Support functionally
PARTITIONING_SUPPORT	[160]	Support functionally
MAX_ENH_SIZE_MULT	[159:157]	Accept but ignore
PARTITIONS_ATTRIBUTE	[156]	Accept but ignore
PARTITION_SETTING_COMPLETED	[155]	Support functionally
GP_SIZE_MULT	[154:143]	Accept but ignore
ENH_SIZE_MULT	[142:140]	Accept but ignore
ENH_START_ADDR	[139:136]	Accept but ignore
SEC_BAD_BLK_MGMNT	[134]	Accept but ignore
TCASE_SUPPORT	[132]	Accept but ignore
PERIODIC_WAKEUP	[131]	Accept but ignore
PROGRAM_CID_CSD_DDR_SUPPORT	[130]	Accept but ignore
VENDOR_SPECIFIC_FIELD	[127:64]	Accept but ignore
NATIVE_SECTOR_SIZE	[63]	Support functionally

USE_NATIVE_SECTOR	[62]	Support functionally
DATA_SECTOR_SIZE	[61]	Support functionally
INI_TIMEOUT_EMU	[60]	Support functionally
CLASS_6_CTRL	[59]	Support functionally
DYNCAP_NEEDED	[58]	Support functionally
EXCEPTION_EVENTS_CTRL	[57:56]	Support functionally
EXCEPTION_EVENTS_STATUS	[55:54]	Support functionally
EXT_PARTITIONS_ATTRIBUTE	[53:52]	Support functionally
CONTEXT_CONF	[51:37]	Support functionally
PACKED_COMMAND_STATUS	[36]	Support functionally
PACKED_FAILURE_INDEX	[35]	Support functionally
POWER_OFF_NOTIFICATION	[34]	Accept but ignore
CACHE_CTRL	[33]	Support functionally
FLUSH_CACE	[32]	Support functionally

Note: 'Accept but ignore' means the value of the field in CSD register can be accepted and read by the MMC model but doesn't affect any function.

13. Timing values

The table below shows the supported timing and its default value in the MMC model.

Symbol	***Default value***	***Description***
TIC	74	Required initial power-up clocks
NID	5	Command end to response R2 or R3 start in clock cycles, the value range defined in the spec is (Min=5, Max=5). The user can change the value by changing the parameter NID in the MMC model.
NCR	2	Command end to response R1 start in clock cycles, the value range defined in the spec is (Min=2, Max=64). The user can change the value by changing the parameter NCR in the MMC model.
NAC	8	Read data access time in clock cycles, the value range defined in the spec is (Min=2 for Normal mode and Min=8 for HS mode). The user can change the value by changing the parameter NAC in the MMC model.
NCD	56	Boot operation, NCD clocks required from CMD high to next mmc command, the value range defined in the spec is (Min=56). The user can change the value by changing the parameter NCD in the MMC model.

14. Special Notes for EXT_CSD_reg initialization

1) Customers MUST source "keepnet_FF.qel" as part of the emulation compile script to ensure that the necessary nets of the EXT_CSD_reg are available for forcing at runtime.
2) Customers shall run the runtime force script "xxx_EXT_CSD_reg.fs" prior to running Palladium to initialize EXT_CSD registers. The user can source the force script at run time.

15. HS200 Tuning Sequence

The HS200 tuning sequence pattern is stored in the Palladium MMC model in the array tune_pattern. If the user needs this feature, this array must be loaded using the Palladium memory command with the provided tune.hex memory initialization file before CMD21 is issued.

16. Initialization Sequence

1) Users must follow the right steps to boot up the model, all the state paths which allow the model enter into IDLE state are shown below.
2) Initialization sequence is issued after power on or hardware reset or CMD0 with argument=0xF0F0F0F0.
3) BOOT_PARTITION_ENABLE is set in EXT_CSD[179];
4) Alternative boot mode should be mandatorily enabled in EXT_CSD[228] of the devices who follow v4.4 and after.

16.1. BOOT_PARTITION_ENABLE = 0

If BOOT_PARTITION_ENABLE bits in EXT_CSD[179] are set to 0, device will directly enter into IDLE state and wait for CMD1.

16.2. BOOT_PARTITION_ENABLE = 1

Users can follow any of the following ways to finish the initialization sequence. Any other ways may cause error and make the initialization sequence failed.
In the description below, PRE-IDLE state is the state after power on or hardware reset or CMD0 reset.

Initialization ways:

1) Issue CMD1 directly when device is in PRE-IDLE state, in this case, device will directly enter into STANDBY state and wait for CMD2.
2) Issue CMD0 with arg=FFFFFFFA directly when device is in PRE-IDLE state, in this case, device will enter into alternative boot mode if alternative boot mode bit is enabled in EXT_CSD[228], this is mandatory in the devices who follow v4.4 and after. Users should notice that CMD line can't be pulled low and other commands should not be issued before CMD0 with arg=FFFFFFFA being issued. After reading boot data is finished or terminated by CMD0 reset, device will enter IDLE state and wait for CMD1.
3) Pull CMD line low and hold it less than 74 clock cycles before issuing commands. In this case, device will enter IDLE state and wait for CMD1. If the command is CMD1, device will enter STAND-BY state and wait for CMD2.
4) Pull CMD line low and hold it no less than 74 clock cycles. In this case, device will enter original boot mode. After reading boot data is finished or terminated by CMD0 reset or terminated by pull CMD line high, device will enter IDLE state and wait for CMD1.

17. Use Large 4KB sectors

Follow the steps below, users can enable large 4KB sectors.

1) Write 0x01 to the USE_NATIVE_SECTOR field in EXT_CSD[62] by CMD6.
2) Hardware reset or CMD0 reset.
3) Issue CMD1 before reaching the time out limit, the time out is aligned to INI_TIMEOUT_EMU field in EXT_CSD[60].
4) After CMD1, device will be in large 4KB sectors mode.

Users should notice that only multiple-block read/write can be supported in 4KB sectors mode, single-block read/write are illegal. Sector counts shall be multiples of 8(4KB).

18. RPMB

We don't support HMAC checking in current module, so we suggest customers to turn off the HMAC checking in the host for the data read out from memory.

19. HPI

Only the following commands are affected by HPI:

1) Single block write
2) Multiple blocks write
3) Erase (including Trim and Discard)
4) Sanitize
5) CACHE_CTRL when used for turning the cache OFF
6) CACHE_FLUSH

20. CACHE

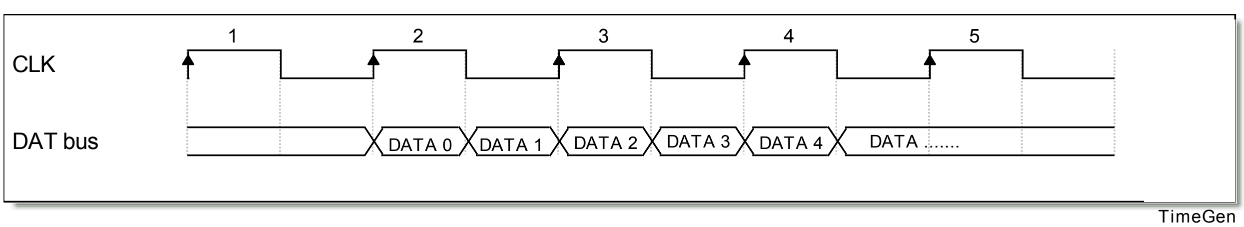
1) Customers shall correctly set the parameters CACHE_MEMCORE_SIZE and CACHE_MEMCORE_ADDR_WIDTH, also CACHE_SIZE in EXT_CSD[252:249].
2) Sanitizing/hardware reset/CMD0 reset will clear the Cache; also after flushing all the data in cache, it will be cleared.
3) If write reliable is enabled or force programming bit is set in CMD23 or customer is accessing BOOT area, data will be directly written into non-volatile storage even if Cache function is ON.
4) Cache flushing can be interrupted by HPI.

21. HS400 mode

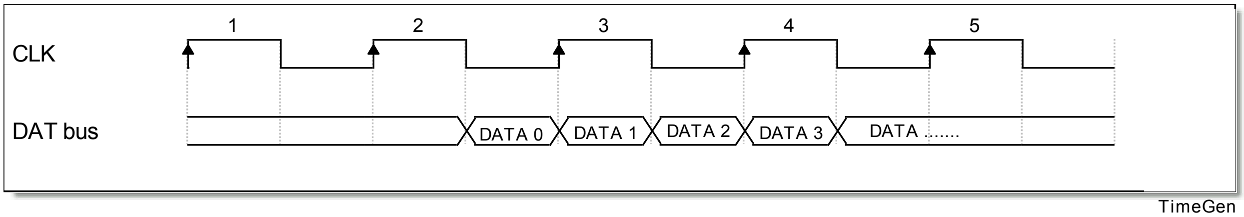
DDR mode shall be enabled (EXT_CSD[183]) after HS_TIMING (EXT_CSD[185]) is set for HS400, this is similar with the sequence of enabling high speed mode.

22. Read data sampling

In DDR mode, by default, the read out data alignment with CLK is as shown below. Users can sample the data as they expect. However, users can NOT sample the read out data directly with CLK.



The MMP model provides a macro --- **MMP_MMC_READ_SAMPLE_DIRECTLY** — for users to sample read out data directly with CLK or DATA_STROBE when in HS400 mode. Users need to add/set this Verilog macro when they compile the model. After doing so, the clock and data alignment is modified as shown in below diagram.



23. Mandatory functions in V4.5 and V5.0

Here are some functions which shall be mandatorily enabled in the devices which follow V4.5 and V5.0.

1) Boot function, bit0 of EXT_CSD[228] shall be set 1.
2) HPI feature, HPI_SUPPORT bit of EXT_CSD[503].
3) MAX_PACKED_READS of EXT_CSD[501] shall be 5 or higher.
4) MAX_PACKED_WRITE of EXT_CSD[500] shall be 3 or higher.
5) MAX_CONTEXT_ID of EXT_CSD[496] shall be 5 or higher.
6) REL_WR_SEC_C of EXT_CSD[222] shall be fixed to 1.

24. Handling large capacity memory sizes

Due to limitation on UXE and IUS software the maximium size model that can be directly supported is 16GB. This max configuration uses the 64bit wide core memory array (mem) and has a depth of 2^31 which is the largest supported.

In order to support large array sizes sparse memory modeling methodologies must be used on the main array of the model, mem.

For ICE mode this is achieved with the command memoryTransform. Example command syntax is given below. Please consult the UXE User documentation for details on the memoryTransform command and the associate page number and page size parameters.

```
memoryTransform –add {i1.mem SPARSE 8 9}
```

For IXCOM mode a system task is used to enable sparse memory

```
$ixc_ctrl("sparse_mem", i1.mem, 8, 9);
```

25. eMMC Debugging

The eMMC model is complex and therefore the associated problem of debugging issues is likewise complex. Below is a list of recommended debugging techniques and tips that the user may use in isolating a problem.

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

- **Waveform debugging:** signal and sequences
  - Check that the clock and reset signals are correctly driven.
  - Check that the EXT_CSD_reg is correctly initialized, refer to the section11 *Special Notes for EXT_CSD_reg Initialization*.
  - Check that the eMMC initialization command followed the initialization sequence required in Section13.
  - Check that the commands are issued at correct device state (*refer to JESD84-B50 Table51- Device state transitions for detail information*), otherwise the command will be taken as illegal command and there will be no response.
    Related signals for checking:
    CST_ILLEGAL_COMMAND_reg: asserted when illegal command received.
    MMC_state: indicates the current state of eMMC device.

Value of MMC_state	*MMC State Name*
0	*MMC_ST_UNINIT*
1	*MMC_ST_IDLE*
2	*MMC_ST_READY*
3	*MMC_ST_IDENT*
4	*MMC_ST_STBY*
5	*MMC_ST_TRAN*
6	*MMC_ST_DATA*
7	*MMC_ST_RCV*
8	*MMC_ST_PRG*
9	*MMC_ST_DIS*
10	*MMC_ST_BTST*
11	*MMC_ST_SLP*
12	*MMC_ST_INA*
13	*MMC_ST_WIRQ*
14	*MMC_ST_PREIDLE*
15	*MMC_ST_PREBOOT*
16	*MMC_ST_BOOT*

CMD_start, CMD_end: indicates the start and end point on the CMD bus.
CMD_index: indicates which command received.
CMD_arg: indicates the argument of received command.

- **Golden waveform:** A package with a reference waveform is available which shows the following command sequence:
  1) skip_boot_mode:
     Directly issue CMD1 to initialize eMMC whatever BOOT_PARTITION_ENABLE is;
     Issue CMD2 -> CMD3 -> CMD7 to enter TRANS mode;
     Issue CMD6 to change bus width;
     Do normal read/write;
     Do ddr read/write;

Do HS400 read/write.

2) boot_mode1:
   Issue CMD0 with arg=FFFFFFFA to enter boot mode;
   After boot is finished, issue CMD1 -> CMD2 -> CMD3 -> CMD7 to enter TRANS mode;
   Issue CMD6 to change bus width;
   Do normal read/write;
   Do ddr read/write;
   Do HS400 read/write.

3) boot_mode2:
   Keep CMD line LOW more than 74 clock cycles to enter boot mode;
   After boot is finished, issue CMD1 -> CMD2 -> CMD3 -> CMD7 to enter TRANS mode;
   Issue CMD6 to change bus width;
   Do normal read/write;
   Do ddr read/write;
   Do HS400 read/write.

- **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.

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>.ext_csd_ext_security_err	EXT_CSD[505]	R
<model_name>.ext_csd_s_cmd_set	EXT_CSD[504]	R
<model_name>.ext_csd_hpi_features	EXT_CSD[503]	R
<model_name>.ext_csd_bkops_support	EXT_CSD[502]	R
<model_name>.ext_csd_max_packed_reads	EXT_CSD[501]	R
<model_name>.ext_csd_max_packed_writes	EXT_CSD[500]	R
<model_name>.ext_csd_data_tag_support	EXT_CSD[499]	R
<model_name>.ext_csd_tag_unit_size	EXT_CSD[498]	R
<model_name>.ext_csd_tag_res_size	EXT_CSD[497]	R
<model_name>.ext_csd_context_capabilities	EXT_CSD[496]	R
<model_name>.ext_csd_large_unit_size	EXT_CSD[495]	R
<model_name>.ext_csd_ext_support	EXT_CSD[494]	R
<model_name>.ext_csd_cache_size_3	EXT_CSD[252]	R
<model_name>.ext_csd_cache_size_2	EXT_CSD[251]	R
<model_name>.ext_csd_cache_size_1	EXT_CSD[250]	R
<model_name>.ext_csd_cache_size_0	EXT_CSD[249]	R
<model_name>.ext_csd_generic_cmd6_time	EXT_CSD[248]	R
<model_name>.ext_csd_power_off_long_time	EXT_CSD[247]	R
<model_name>.ext_csd_bkops_status	EXT_CSD[246]	R
<model_name>.ext_csd_correctly_prg_sectors_num_3	EXT_CSD[245]	R
<model_name>.ext_csd_correctly_prg_sectors_num_2	EXT_CSD[244]	R
<model_name>.ext_csd_correctly_prg_sectors_num_1	EXT_CSD[243]	R
<model_name>.ext_csd_correctly_prg_sectors_num_0	EXT_CSD[242]	R
<model_name>.ext_csd_ini_timeout_pa	EXT_CSD[241]	R
<model_name>.ext_csd_pwr_cl_ddr_52_360	EXT_CSD[239]	R
<model_name>.ext_csd_pwr_cl_ddr_52_195	EXT_CSD[238]	R
<model_name>.ext_csd_pwr_cl_200_360	EXT_CSD[237]	R
<model_name>.ext_csd_pwr_cl_200_195	EXT_CSD[236]	R
<model_name>.ext_csd_min_perf_ddr_w_8_52	EXT_CSD[235]	R
<model_name>.ext_csd_min_perf_ddr_r_8_52	EXT_CSD[234]	R
<model_name>.ext_csd_trim_mult	EXT_CSD[232]	R
<model_name>.ext_csd_sec_feature_support	EXT_CSD[231]	R
<model_name>.ext_csd_sec_erase_mult	EXT_CSD[230]	R
<model_name>.ext_csd_sec_trim_mult	EXT_CSD[229]	R
<model_name>.ext_csd_boot_info	EXT_CSD[228]	R
<model_name>.ext_csd_boot_size_mult	EXT_CSD[226]	R
<model_name>.ext_csd_acc_size	EXT_CSD[225]	R
<model_name>.ext_csd_hc_erase_grp_size	EXT_CSD[224]	R
<model_name>.ext_csd_erase_timeout_mult	EXT_CSD[223]	R
<model_name>.ext_csd_rel_wr_sec_c	EXT_CSD[222]	R
<model_name>.ext_csd_hc_wp_grp_size	EXT_CSD[221]	R
<model_name>.ext_csd_s_c_vcc	EXT_CSD[220]	R
<model_name>.ext_csd_s_c_vccq	EXT_CSD[219]	R
<model_name>.ext_csd_s_a_timeout	EXT_CSD[217]	R
<model_name>.ext_csd_sec_count_3	EXT_CSD[215]	R

<model_name>.ext_csd_sec_count_2	EXT_CSD[214]	R
<model_name>.ext_csd_sec_count_1	EXT_CSD[213]	R
<model_name>.ext_csd_sec_count_0	EXT_CSD[212]	R
<model_name>.ext_csd_min_perf_w_8_52	EXT_CSD[210]	R
<model_name>.ext_csd_min_perf_r_8_52	EXT_CSD[209]	R
<model_name>.ext_csd_min_perf_w_8_26_4_52	EXT_CSD[208]	R
<model_name>.ext_csd_min_perf_r_8_26_4_52	EXT_CSD[207]	R
<model_name>.ext_csd_min_perf_w_4_26	EXT_CSD[206]	R
<model_name>.ext_csd_min_perf_r_4_26	EXT_CSD[205]	R
<model_name>.ext_csd_pwr_cl_26_360	EXT_CSD[203]	R
<model_name>.ext_csd_pwr_cl_52_360	EXT_CSD[202]	R
<model_name>.ext_csd_pwr_cl_26_195	EXT_CSD[201]	R
<model_name>.ext_csd_pwr_cl_52_195	EXT_CSD[200]	R
<model_name>.ext_csd_partition_switch_time	EXT_CSD[199]	R
<model_name>.ext_csd_out_of_interrupt_time	EXT_CSD[198]	R
<model_name>.ext_csd_driver_strength	EXT_CSD[197]	R
<model_name>.ext_csd_device_type	EXT_CSD[196]	R
<model_name>.ext_csd_csd_structure	EXT_CSD[194]	R
<model_name>.ext_csd_ext_csd_rev	EXT_CSD[192]	R
<model_name>.ext_csd_cmd_set	EXT_CSD[191]	R/W
<model_name>.ext_csd_cmd_set_rev	EXT_CSD[189]	R
<model_name>.ext_csd_power_class	EXT_CSD[187]	R/W
<model_name>.ext_csd_hs_timing	EXT_CSD[185]	R/W
<model_name>.ext_csd_bus_width	EXT_CSD[183]	R/W
<model_name>.ext_csd_erased_mem_count	EXT_CSD[181]	R
<model_name>.ext_csd_partition_config	EXT_CSD[179]	R/W
<model_name>.ext_csd_boot_config_prot	EXT_CSD[178]	R/W
<model_name>.ext_csd_boot_bus_width	EXT_CSD[177]	R/W
<model_name>.ext_csd_erase_group_def	EXT_CSD[175]	R/W
<model_name>.ext_csd_boot_wp_status	EXT_CSD[174]	R
<model_name>.ext_csd_boot_wp	EXT_CSD[173]	R/W
<model_name>.ext_csd_user_wp	EXT_CSD[171]	R/W
<model_name>.ext_csd_fw_config	EXT_CSD[169]	R/W
<model_name>.ext_csd_rpmb_size_mult	EXT_CSD[168]	R
<model_name>.ext_csd_wr_rel_set	EXT_CSD[167]	R/W
<model_name>.ext_csd_wr_rel_param	EXT_CSD[166]	R
<model_name>.ext_csd_sanitize_start	EXT_CSD[165]	R/W
<model_name>.ext_csd_bkops_start	EXT_CSD[164]	R/W
<model_name>.ext_csd_bkops_en	EXT_CSD[163]	R/W
<model_name>.ext_csd_rst_function	EXT_CSD[162]	R/W
<model_name>.ext_csd_hpi_mgmt	EXT_CSD[161]	R/W
<model_name>.ext_csd_part_support	EXT_CSD[160]	R
<model_name>.ext_csd_max_enh_size_mult_2	EXT_CSD[159]	R
<model_name>.ext_csd_max_enh_size_mult_1	EXT_CSD[158]	R
<model_name>.ext_csd_max_enh_size_mult_0	EXT_CSD[157]	R
<model_name>.ext_csd_partitions_attr	EXT_CSD[156]	R/W
<model_name>.ext_csd_part_setting_comp	EXT_CSD[155]	R/W
<model_name>.ext_csd_gp_size_mult_4_2	EXT_CSD[154]	R/W
<model_name>.ext_csd_gp_size_mult_4_1	EXT_CSD[153]	R/W
<model_name>.ext_csd_gp_size_mult_4_0	EXT_CSD[152]	R/W
<model_name>.ext_csd_gp_size_mult_3_2	EXT_CSD[151]	R/W
<model_name>.ext_csd_gp_size_mult_3_1	EXT_CSD[150]	R/W
<model_name>.ext_csd_gp_size_mult_3_0	EXT_CSD[149]	R/W
<model_name>.ext_csd_gp_size_mult_2_2	EXT_CSD[148]	R/W
<model_name>.ext_csd_gp_size_mult_2_1	EXT_CSD[147]	R/W
<model_name>.ext_csd_gp_size_mult_2_0	EXT_CSD[146]	R/W

<model_name>.ext_csd_gp_size_mult_1_2	EXT_CSD[145]	R/W
<model_name>.ext_csd_gp_size_mult_1_1	EXT_CSD[144]	R/W
<model_name>.ext_csd_gp_size_mult_1_0	EXT_CSD[143]	R/W
<model_name>.ext_csd_enh_size_mult_2	EXT_CSD[142]	R/W
<model_name>.ext_csd_enh_size_mult_1	EXT_CSD[141]	R/W
<model_name>.ext_csd_enh_size_mult_0	EXT_CSD[140]	R/W
<model_name>.ext_csd_enh_start_addr_3	EXT_CSD[139]	R/W
<model_name>.ext_csd_enh_start_addr_2	EXT_CSD[138]	R/W
<model_name>.ext_csd_enh_start_addr_1	EXT_CSD[137]	R/W
<model_name>.ext_csd_enh_start_addr_0	EXT_CSD[136]	R/W
<model_name>.ext_csd_sec_bad_blk_mgmnt	EXT_CSD[134]	R/W
<model_name>.ext_csd_tcase_support	EXT_CSD[132]	R/W
<model_name>.ext_csd_period_wakeup	EXT_CSD[131]	R/W
<model_name>.ext_csd_program_cid_csd_ddr_support	EXT_CSD[130]	R
<model_name>.ext_csd_native_sector_size	EXT_CSD[63]	R
<model_name>.ext_csd_use_native_sector	EXT_CSD[62]	R/W
<model_name>.ext_csd_data_sector_size	EXT_CSD[61]	R
<model_name>.ext_csd_ini_timeout_emu	EXT_CSD[60]	R
<model_name>.ext_csd_class_6_ctrl	EXT_CSD[59]	R/W
<model_name>.ext_csd_dyncap_needed	EXT_CSD[58]	R
<model_name>.ext_csd_exception_events_ctrl_1	EXT_CSD[57]	R/W
<model_name>.ext_csd_exception_events_ctrl_0	EXT_CSD[56]	R/W
<model_name>.ext_csd_exception_events_status_1	EXT_CSD[55]	R
<model_name>.ext_csd_exception_events_status_0	EXT_CSD[54]	R
<model_name>.ext_csd_ext_part_attr_3_4	EXT_CSD[53]	R/W
<model_name>.ext_csd_ext_part_attr_1_2	EXT_CSD[52]	R/W
<model_name>.ext_csd_context_conf_01	EXT_CSD[51]	R/W
<model_name>.ext_csd_context_conf_02	EXT_CSD[50]	R/W
<model_name>.ext_csd_context_conf_03	EXT_CSD[49]	R/W
<model_name>.ext_csd_context_conf_04	EXT_CSD[48]	R/W
<model_name>.ext_csd_context_conf_05	EXT_CSD[47]	R/W
<model_name>.ext_csd_context_conf_06	EXT_CSD[46]	R/W
<model_name>.ext_csd_context_conf_07	EXT_CSD[45]	R/W
<model_name>.ext_csd_context_conf_08	EXT_CSD[44]	R/W
<model_name>.ext_csd_context_conf_09	EXT_CSD[43]	R/W
<model_name>.ext_csd_context_conf_10	EXT_CSD[42]	R/W
<model_name>.ext_csd_context_conf_11	EXT_CSD[41]	R/W
<model_name>.ext_csd_context_conf_12	EXT_CSD[40]	R/W
<model_name>.ext_csd_context_conf_13	EXT_CSD[39]	R/W
<model_name>.ext_csd_context_conf_14	EXT_CSD[38]	R/W
<model_name>.ext_csd_context_conf_15	EXT_CSD[37]	R/W
<model_name>.ext_csd_packed_command_status	EXT_CSD[36]	R
<model_name>.ext_csd_packed_failure_index	EXT_CSD[35]	R
<model_name>.ext_csd_power_off_notification	EXT_CSD[34]	R/W
<model_name>.ext_csd_cache_ctrl	EXT_CSD[33]	R/W
<model_name>.ext_csd_flush_cache	EXT_CSD[32]	R/W

26. Revision History

The following table shows the revision history for this document

Date	Version	Revision
Feb 2013	1.0	Initial release
April 2011	1.1	Added RST_N support
May 2011	1.2	Added GO_PRE_IDLE_STATE
June 2011	1.3	Added MMP release level info
Jan 2012	1.4	eMMC 4.5 support added including HS200 mode
March 2013	1.5	4kB sector support, General Purpose partitions and more EXT_CSD registers added.
May 2013	1.6	Fixed boot sequence, support packed command, context management, trim/discard and sanitize
July 2013	1.7	RPMB support; HPI support; Dynamic Capacity Management support; Improve protection features
Nov 2013	1.8	Support HS400 mode which is for eMMC 5.0 draft version; Support Cache
Nov 2013	1.9	Cleanup for eMMC 5.0 version
April 2014	2.0	Large capacity updates
May 2014	2.1	Repair minor typos and finish IRQ_REJECTED descriptor
June 2014	2.2	Added section on INT_CLK and IXCOM
July 2014	2.3	Simplified calculation of GP area size. Repaired doc property title. Modified user adjustable parameter and localparam tables to align with changes being made to model and wrapper HDL.
September 2014	2.4	Remove version from UG file name.
January 2015	2.5	Add some info to Memory Arrays section. Update related publications list.
July 2015	2.51	Add description for pullups and ***HOST_NO_PULLUP*** macro
July 2015	2.6	Update Cadence naming on front page
September 2015	2.7	Modify compile notes to reflect *.vp as sole model format. Add note about synthesis options.
November 2015	2.8	Add description for read data sampling and MMP_MMC_READ_SAMPLE_DIRECTLY macro
December 2015	2.9	Add introduction of address mapping when accessing memory arrays
January 2016	3.0	Update for Palladium-Z1 and VXE. Expand section “Address mapping to access memory arrays” Adding comment for not sourcing the force script at time 0 for EXT_CSD register
May 2016	3.1	Adding eMMC Debugging section
July 2016	3.2	Adding device register and timing values sections. Removed hyphen from Palladium naming.
October 2016	3.3	Adding Verilog Macro Defines section.
September 2017	3.4	Adding Debug Display feature Adding reference specification version date Update macro name to MMP_EMMC_HOST_NO_PULLUP
January 2018	3.5	Modify header and footer

April 2018	3.6	Remove the limitation for not source xxx_EXT_CSD_reg.fs at time 0
June 2018	3.7	Add section for Manual configuration
July 2018	3.8	Update for new utility library