HB0510

Hand book

CoreMMC v3.0

June 2018

Microsemi

a Microchip company

Contents

1 Revision History

The revision history describes the changes that were implemented in the document. The changes are listed by revision, starting with the most current publication.

1.1 Revision 4.0

Revision 4.0 was published in June 2018. The following edits are done in this revision of the document.

- Added a new chapter, Design Constraints (see page 42).
- CLK_OE and CLKI ports are removed from the figure, CoreMMC I/O Signal Diagram (see page 2) and table, CoreMMC I/O Signal Descriptions (see page 11).
- Replaced the figures, CoreMMC System Integration (see page 39), CoreMMC SmartDesign Configuration (see page 37), and CoreMMC Full I/O View (see page 36).
- Added two new figures, CoreMMC Configuration (see page 40), and Place and Route Configuration (see page 41).

1.2 Revision 3.0

Revision 3.0 was published in May 2018. The following edits are done in this revision of the document.

- Added details of additional Multiple Block support registers.
- Updated user testbench information to reflect new BFM based testbench.
- Modified prefixes of all interrupt related bits to prevent naming clashes with single & multiple block prefix convention.
- Additional information provided for Clock Register description.
- Renamed DMA control signals.
- Updated utilization data including information on achieving >104 MHz timing
- Added VHDL support
- Width of Write & Read Data FIFOs updated from 8- to 32-bits

1.3 Revision 2.0

Revision 2.0 was published in Oct 2013. It was the first publication of this document.

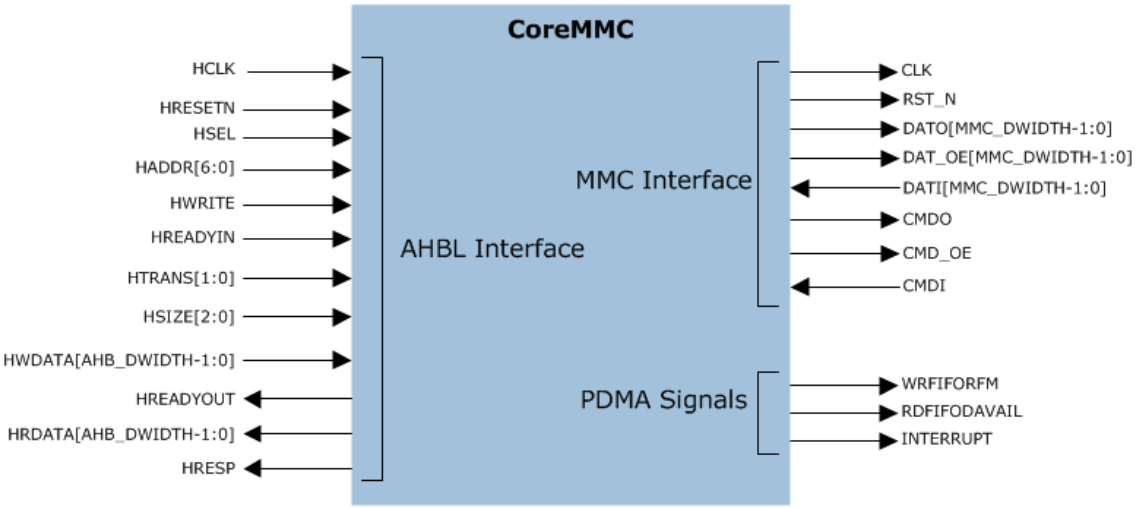
2 Introduction

CoreMMC provides an embedded MultiMedia Card (eMMC) controller to interface with eMMC devices. eMMC devices can be utilized to expand the on-board storage capacity for larger applications or to store user data. The core is parameterized to allow user specifications of MMC data bus width and FIFO depth and is conformant to the JEDEC eMMC 4.41 Standard Specification.

CoreMMC consists of an AMBA High-performance Bus-Lite (AHB-Lite) slave interface designed to connect to an AHB-Lite bus. Block write and read data FIFOs can be accessed by an AHB-Lite master along with the CoreMMC registers. eMMC is a version of the MMC standard which is intended to provide a unified command or control interface for various memory types, mostly for high-capacity, high-performance, low-cost multimedia storage purposes. Communication is achieved using a (up-to) 10-bit bus and a control protocol that is defined as part of the JEDEC eMMC standard. For more information, refer to [R1].

eMMC abstracts the increasingly complex and divergent Flash interface of various vendors and allows for better design modularity and portability.

Figure 1 • CoreMMC I/O Signal Diagram



2.1 Supported Families

The following list of families support CoreMMC v3.0.

- SmartFusion®2
- IGLOO®2
- PolarFire®

2.2 Key Features

The following list describes the key features of CoreMMC.

- Up to 52 MHz MMC clock rate, making for a maximum of ~52 MBs throughput at 8-bit data width (Theoretical throughput of CoreMMC. Actual throughput will be affected by eMMC device throughput and AHB bandwidth)
- Configurable data bus widths

–1, 4 or 8 bit

- Supports Block mode transfer
  - Single block write
  - Multiple block write
  - Single block read
  - Multiple block read
- Cyclic Redundancy Check (CRC) protection for both commands and data transfers
- AHB-Lite compliant
  - Command, Response, Write, and Read Data registers for indirect access to MMC part
  - 8-/16-/32-bit AHB transfers
- Write DATA FIFO
  - To decouple AHB from MMC bus for write data transfers
  - Depth configurable from 512 Bytes to 32 KBs
  - Overrun and Underrun error flags
- Read DATA FIFO
  - To decouple AHB from MMC bus for read data transfers
  - Depth configurable from 512 Bytes to 32 KBs
  - Overrun and Underrun error flags
- Interrupt generation
  - Command Sent and Response Received interrupts
  - Error flag interrupts
  - Single block write and read done interrupts
  - Multiple block write and read done interrupts
- Automatic Sleep mode for eMMC when clock pulled Low

2.3 Limitations

The following list gives the limitations of CoreMMC in this version.

- Sequential mode transfers
- ECC error correction (handled on the device side instead)
- Stream Read and Stream Write
- Double Data Rate (DDR)

2.4 Core Version

This handbook supports CoreMMC version 3.0.

2.5 Supported Interfaces

CoreMMC is available with the following interfaces.

- AHB-Lite slave interface
- Interrupt request interface
- MMC master interface
- Peripheral Direct Memory Access (PDMA) interface

For more information on these interfaces, see Table 7 (see page 11).

2.6 Device Utilization and Performance

CoreMMC has been implemented in several devices of Microsemi using standard speed grades. The following tables list a summary of implementation data for different product families.

Table 1 • CoreMMC Device Utilization and Performance (Minimum Configuration)

Family	Logic Elements			Utilization		Performance (MHz)	RAM
	Sequential	Combinatorial	Total	Device	Total %		
SmartFusion2	1005	1721	2726	M2S050	4.83	140	8 Blocks of RAM64X18
IGLOO2	1005	1725	2730	M2GL050	4.84	139	8 Blocks of RAM64X18
PolarFire	776	1546	2322	MPF100T	2.13	157	2 Blocks of LSRAM

Note: Top-level parameters or generics were set as – MMC_DWIDTH = 1-bit, FIFO_DEPTH = 512 bytes.

Table 2 • CoreMMC Device Utilization and Performance (Maximum Configuration)

Family	Logic Elements			Utilization		Performance (MHz)	RAM
	Sequential	Combinatorial	Total	Device	Total %		
SmartFusion2	2028	2937	4965	M2S050	8.81	123	32 Blocks of RAM1Kx18
IGLOO2	2031	2920	4951	M2GL050	8.78	125	32 Blocks of RAM1Kx18
PolarFire	2022	3027	5049	MPF100T	4.65	146	32 Blocks of LSRAM

Note: Top-level parameters or generics were set as – MMC_DWIDTH = 8-bit, FIFO_DEPTH = 32 KB.

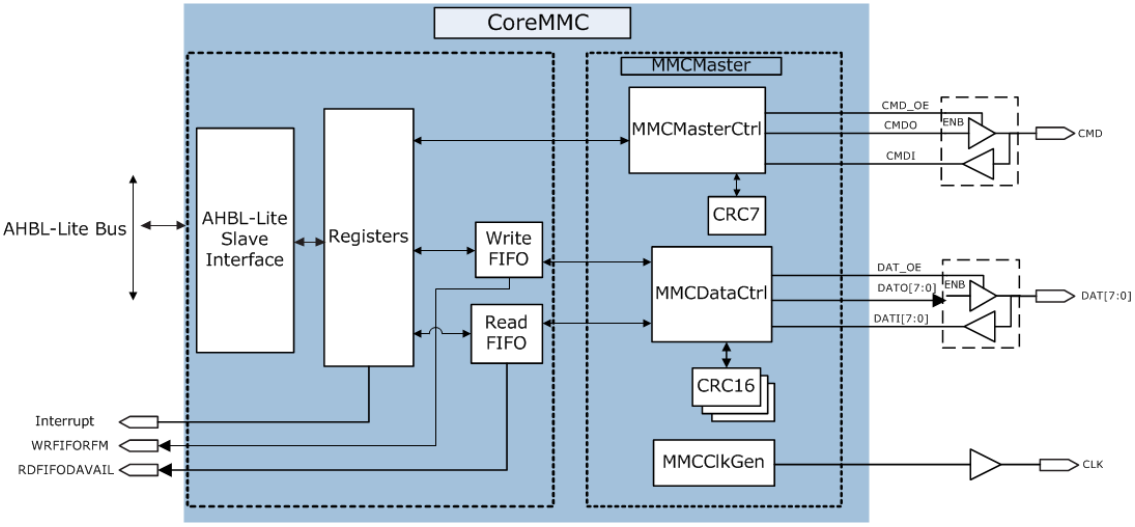
3 Functional Description

The CoreMMC soft IP provides a user interface to access MMC devices through AHB-Lite interface. The following sections give functional description of CoreMMC.

3.1 System Overview

The CoreMMC provides a slave AHB-Lite interface and a master MMC interface. The AHB-Lite interface provides the slave interface for a host to access the control and status registers in the core and to initiate transactions to a MMC slave device connected on the MMC master interface. Additionally, FIFO status signals are provided to the PDMA. The following figure shows the top-level block diagram.

Figure 2 • CoreMMC Block Diagram



3.1.1 Functional Description of the MMC Controller

This section describes the components of the CoreMMC controller and its operations. The primary functional blocks are as follows:

- AHB-Lite slave interface
- Registers
- Write FIFO
- Read FIFO
- MMC master

AHB-Lite Slave Interface

The AHB-Lite slave interface provides the functionality to interface to an AHB-Lite master as defined in [R3]. This slave interface supports 8-/16-/32-bit access as defined by HSIZE. Sequential transfers are not supported.

Registers

The Register block decodes the address of AHB-Lite slave and performs the requested read or write operation. It also latches status and error flags from the MMC master as it executes and controls the reading or writing of Read or Write FIFOs by the host.

Write FIFO

The Write FIFO provides storage for block writes to the MMC slave device. The host, writes the data into the FIFO and the MMC master block, reads the data out during write transactions on MMC DAT lines. The Write FIFO can contain multiple blocks of data to write , depending on size of the core instantiated. The Write FIFO detects overruns of excess data, writes into it from the host and underrun errors of attempting to read an empty FIFO by the MMC master and sets associated bits in the register block.

Read FIFO

The Read FIFO provides storage for block reads from the MMC slave device. The MMC master block writes the data into the FIFO and the AHB-Lite slave block reads the data out. The Read FIFO can contain multiple blocks of data to read , depending on size of the core instantiated. The Read FIFO detects overruns of excess data, writes into it from the MMC master block and underrun errors of attempting to read an empty FIFO by the AHB-Lite slave block and sets associated bits in the register block.

MMC Master

The MMC master implements transactions on the MMC bus as defined by the JEDEC specification [R1] under direction of the host. It is composed of the following primary blocks.

- MMC master controller: Controls command-response transactions on the CMD pins. It drives out the command based on CR0-CR5 and checks and stores the expected response in RR0-RR15.
- MMC data controller: Controls the write block and read block transactions on DAT pins. Indicates to the MMC master controller when slave is busy (that is, when DAT0 is held Low).
- CRC7: Provides CRC7 computation for command-response on CMD.
- CRC16: Provides CRC16 for data transactions on DAT pin(s). There can be multiple CRC16 blocks in core depending on MMC_DWIDTH specified (one per pin).
- Clock Generator: Generates the CLK signal from HCLK based on configuration.

The MMC master implements the MMC bus protocol as defined in [R1].

3.1.2 MMC Protocol

Each MMC bus operation consists of a command (host to device), response (device to host), and a possible data transfer. Commands are well-defined 48-bit tokens. Responses are either 48 bits or 136 bits. Both commands and responses are protected by 7-bit CRC headers generated by CoreMMC. Data packets are fixed and can be calculated as; block length * 8 (for single data bus width), block length * 2 (for 4-bit data bus width), or block length * 1 (for 8-bit data bus width). Data packets are protected by 16-bit CRC headers generated by CoreMMC.

The MMC protocol is based on command and data bit streams that are initiated by a start bit and terminated by a stop bit. Additionally, the MMC controller provides a reference clock and is the only master interface that can initiate a transaction.

- Command: A token transmitted serially on the CMD pin that starts an operation.
- Response: A token from the card transmitted serially on the CMD pin in response to certain commands.
- Data: Transferred serially using the data pins for data movement commands.

All transactions are controlled by the host. The host configures the CoreMMC and the attached slave device such as configuring number of DAT bits to be used (1, 4, or 8) or setting the block length for read and write transactions. All communications to the attached MMC slave device are performed using commands and responses as defined in the JEDEC Specification [R1]. This communication uses the command and response registers in the core.

3.1.3 Operating Modes

There are five operation modes defined in the MMC 4.41 standard [R1].

Boot mode: The MMC device is in the state immediately following either one of these three conditions: power-cycle, hardware reset (using RST signals), or CMD0 transmission with argument 0xF0F0F0F0. Refer to [R3] for a full list of eMMC commands and their usage. This mode is not currently supported.

Card Identification mode: After booting, the MMC device will be in Card Identification mode until the SET_RELATIVE_ADDR command (CMD3) is received from the host (CoreMMC).

Interrupt mode: This mode allows the MMC device to interrupt the host (CoreMMC) and indicate that data is ready to be transmitted; this is an optional feature and is not implemented in this version of CoreMMC.

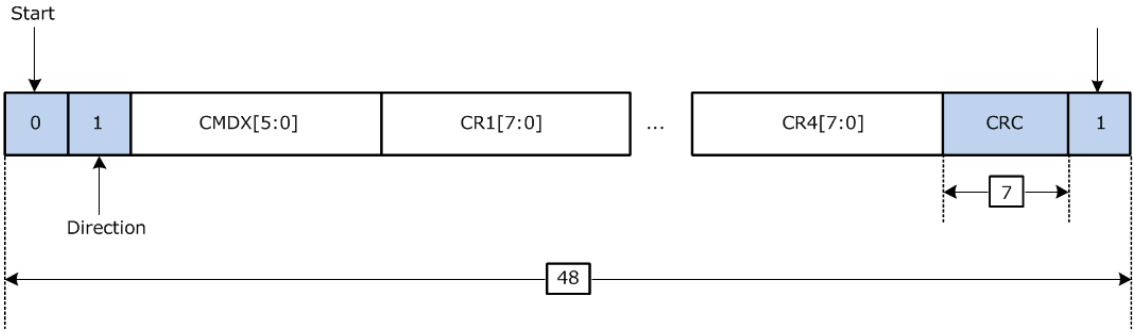
Data Transfer mode: This is the default mode for the MMC, once a Relative Card Address has been assigned to it (after Card Identification Mode). Data transfer commands can be issued from this state.

Inactive mode: The MMC device enters into Inactive mode, if the operating voltage range or access mode is not valid or CMD15 (GO_INACTIVE_STATE command) is received.

3.1.4 Command Registers

These registers store the command data to be transmitted from CoreMMC to the slave device and sent on the CMD pin. The CMD data consists of 38 bits of data (48 bits, minus 1 start bit, 1 stop bit, 1 direction bit, and 7 CRC bits). The following figure shows the full MMC command.

Figure 3 • Figure 3 Command Register use in MMC Command Token



The Command registers send the most-significant bit first on the CMDO pin. CR0 is defined as the Command Index register. This is the 6 bits that define the type of command for the slave. Registers CMD1 to CMD4 define the arguments for the command. Writing to CR4 (Command Register 4) also initiates a command from the host to the slave with whatever command data is currently present in CR0-CR4, when possible (the slave may hold the bus by pulling CMD low). As such, the command registers CR0-CR4 should be written in sequence. CR1-CR4 can be written together as a 32-bit word which causes the command transaction to be initiated. Similarly a 16-bit half-word write to CR3-CR4 will also initiate the command transaction as well as a byte write to CR4.

3.1.5 Response Registers

The CoreMMC response registers (RR0 – RR 15) are used to store and return a response to the host from the MMC slave device. Usually this will be a 38-bit value unless the response is R2, which is the CID or CSD register of the MMC device and is 128 bits long (that is, 138 minus 1 stop bit, 1 start bit, 1 transmission bit, and 7 CRC bits). Beyond the size difference for R2 responses, the formats of the Response registers are the same as the Command registers. The response data is stripped of the 7 CRC bits and stored in these read-only registers after each response.

For a 38-bit response (any response other than R2), only the first 6 registers (RR0-RR5) are updated. RR5 has the CRC and stop-bit from the received response stored to aid debugging (that is, RR5[7:0] = {CRC[6:0], Stop-Bit}). The unused response registers will be zeroed out.

After each response, the stored data is overwritten. If this data is required, it can be read back after each operation. The Response registers are cleared when a command is initiated and are set when srri is set. The response registers maintain their values until another command is initiated.

3.1.6 Multiple Block Writes

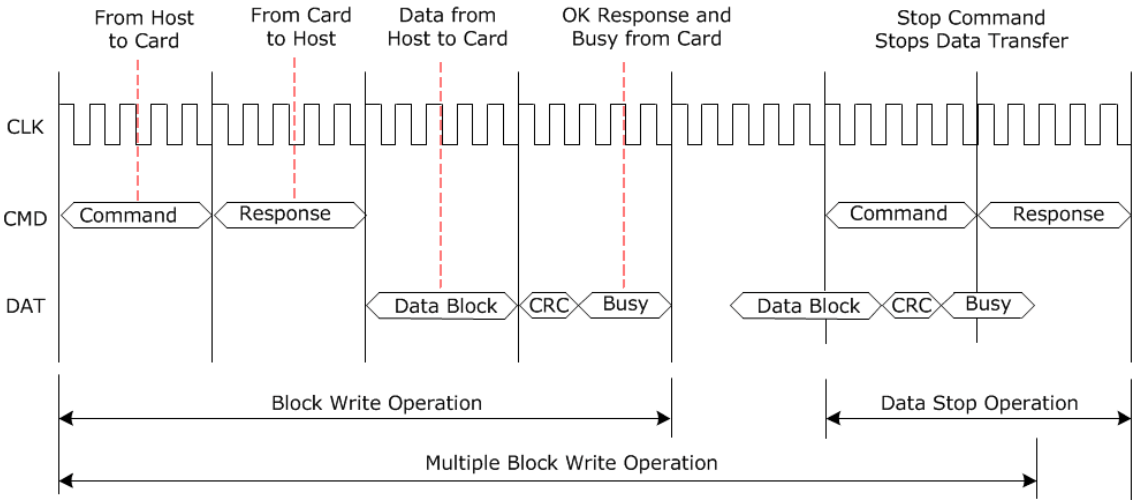
The host must specify the number of blocks in the multiple block write transfer to CoreMMC through the Block Count register and to the eMMC slave device by issuing CMD23. If the block size differs from that already defined in the Block Length register, then the register needs to be updated with the new block length and the slave, notified of the new block length by issuing CMD16.

For interrupt driven operation, the host should then set the mask accordingly in the MBIMR and IMR registers (generate interrupt off multiple block write related status and error bits) and issue CMD25 through the Command registers. After receiving successful response (without any error) from eMMC device; the host should set the mbwstrt bit in the Multiple Block Control and Status register; then the host starts filling the write FIFO with the transfer size amount of data, to avoid multiple block write FIFO timeout error (stambwdatainfifoto).

The mbwdone bit is asserted in the MBISR when the transfer is complete or any multiple block write related error occurs. If an error occurs, an interrupt will be generated off one of the multiple block write related error bits in the MBISR or the ISR (Write FIFO underrun – This should never occur due to the data in FIFO timeout logic as the start bit of the next block in a multi block transfer will only be loaded out onto DAT when there is sufficient data to complete the transfer of this block in the Write FIFO).

Note: A suitable value needs to be specified to the DATATO register, to ensure that host is allowed sufficient time to load data into the Write FIFO & also to allow for the slave device being slow to respond with CRC status, or whilst the slave is performing background operations indicated by DAT0 being held low for extended periods (typically writing to Flash).

Figure 4 • Multiple Block Write Operation



Note: The clock is representative only and does not show exact number of clock cycles for the full transaction.

3.1.7 Multiple Block Reads

The host must ensure that the number of bytes to be received from the slave device can be held in the Read FIFO (to ensure no overrun on writes of data into FIFO from MMC bus). The host can clear the content of the Read FIFO by setting the fiforeset bit in the Control register (clears the contents of the Write FIFO also). This resets the FIFO flags and status bits.

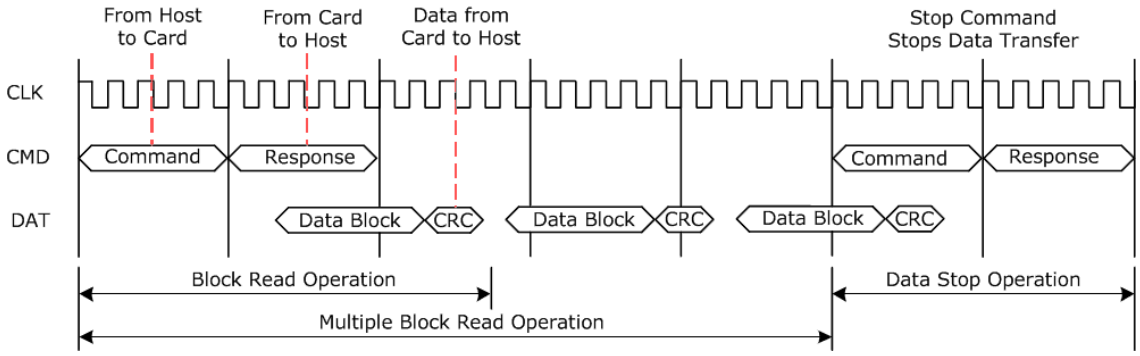
The host must specify the number of blocks in the multiple block read transfer to CoreMMC via the Block Count register and to the eMMC slave device by issuing CMD23. If the block size differs from that already defined in the Block Length register, then the register needs to be updated with the new block length and the slave notified of the new block length by issuing CMD16.

The host then needs to set the mbrstrt bit in the MBCSR, to inform CoreMMC to get ready to receive multiple blocks of data from the eMMC slave device. The host should then issue CMD18 to the slave device, which starts the multi block transfer from the slave. After receiving successful response of CMD18, host starts reading data from read FIFO.

The mbrdone bit is asserted in the MBISR, when the transfer is complete or any multiple block related error occurs. If an error occurs, an interrupt will be generated off one of the multiple block read related error bits in the MBISR or the ISR.

Note: A suitable value needs to be specified to the DATATO register to ensure that host allows for the slave to respond with the start bit of the next block within the multiple block transfer (This can be a significantly large delay – Refer the eMMC slave device manufacturer's documentation. Denoted by N_{AC} cycles in the JEDEC Specification [R1]) .

Figure 5 • Multiple Block Read Operation



3.1.8 Programming Example

An example sequence on programming an eMMC device is shown in the following steps. Specifically it is a basic sequence to program the SanDisk SDIN5C2-8G part on the Microsemi's SmartFusion®2 Development Kit to perform basic reads and writes. This is a general example of a sequence to program a device with a block and to read this block back. This is not an exhaustive example with full error checking. It gives a basic flow.

Note: By default, the reset_n is not operational and boot mode not available (by JEDEC spec). The eMMC chip powers up into *Idle* state.

- CMD0 0x0: // move to idle state.
- CMD1 0xC0_FF_80_80: // read OCR
- Check response has busy bit High (no Busy), that is, Response 0xC0_FF_80_80 (FF_80 voltage ranges) – when ready not Busy, eMMC moves into Ready state.
- CMD2 0x00_00_00_00: // read CID
  - Response R0-R15 - 45, 1, 0, 53, 45, 4d, 30, 38, 47. 90, 52, E7, 9B, 6, BF, 1
- CMD3 0x0002_0000: // set RCA – Relative Card Address
  - Response CMD3, Card Status\[31:0\] - 0x00_00_05_00; // no errors, in Indent state
- CMD9 0x0002_0000: // Send_CSD – get card specific data

Vers code in EXT_CSD, Version 4.1-4.2-4.3, 10ms,

- CMD7 0x0002_0000: // Go to transfer state
  - Response R0-R4 - 7, 00, 00, 07, 00
- CMD16 0x4: // Set_BlockLen- setting to 4 bytes
  - Response R0-R4 - 0x10, 00,00,09, 00

- CMD17 0x0: // Read_Single_Block
  - Response R0-R4 – 0x11, 20, 00, 09, 00 - block length error (just an example of an error when block length is not supported by device)
- CMD16 0x200: // Set_BlockLen – 512 bytes
  - Response R0-R4 - 0x10, 00,00,09, 00
- CMD6 0x03B7_0200: // set EXT_CSD 8-bit width data
  - Response R0-R4 - 0x6, 00,00,08, 00 - note bit\[8\] in status indicates not ready

Note: Core must have MMC_DWIDTH parameter set to 2'b11 and register Data Control Register must have dsize=2'b10 – that is, all most support 8-bit DAT bus.

- CMD13 0x0002_0000: // Send_Status from card 1
  - Response R0-R4 - 0xD, 00,00,09, 00 - now bit\[8\] in status indicates **ready**
- CMD24 0x0: // Write_Single_Block, sector 0x0
  - Response R0-R4 – 0x18, 00, 00, 09, 00 - no error – in Trans state
- CMD17 0x0: // Read_Single_Block
  - Response R0-R4 – 0x11, 00, 00, 09, 00 - no error – in Trans state

To verify correct operation, Read Data from block read in RDR can be compared to data written in block write.

4 Core Interfaces

The following sections give the details of I/O signals and Verilog or VHDL parameters of CoreMMC.

4.1 Verilog or VHDL Parameters

The following table describes the CoreMMC parameters (Verilog) or generics (VHDL) for configuring the RTL code. All parameters and generics are integer types.

Table 3 • CoreMMC Parameters or Generics Descriptions

Name	Valid Range	Default	Description
MMC_DWIDTH	0, 2, 3	0	Data bus width, allowed to be 1, 4 or 8-bit as per the JEDEC eMMC 4.41 standard 0: 1- bit 2: 4-bit 3: 8-bit
FIFO_DEPTH	0-3	0	0: 512 B 1: 4 KB 2: 16 KB 3: 32 KB

4.2 I/O Signals

The following table describes the port signals for the CoreMMC macro as shown in Figure 1 (see page 2).

Table 4 • CoreMMC I/O Signal Descriptions

Name	Type	Description
eMMC signals (to eMMC slave)		
CLK	Out	Clock output to MMC device. Derived from the HCLK at frequency determined by clock register.
RST_N	Out	Hard reset for eMMC device. Active Low signal. Only operational for slave device if enabled within the slave device.
DATO[MMC_DSIZE-1:0]	Out	Data output for bi-directional bus in Push-pull mode.
DATI[MMC_DSIZE-1:0]	In	Data input from bi-directional bus in Push-pull mode.
DAT_OE[MMC_DSIZE-1:0]	Out	Output enable for data bus.
CMDO	Out	Command bus output
CMDI	In	Command bus input
CMD_OE	Out	Output enable for command bus
AHB slave signals		
HCLK	In	AHB system clock.
HRESETN	In	AHB system reset. The signal is active Low. Asynchronous assertion and synchronous de-assertion. This is used to reset AHB registers in the block. Assertion of this signal causes RST_N to be asserted.
HSEL	In	AHB-Lite slave select. This signal indicates that the current transfer is intended for the selected slave.
HADDR[6:0]	In	AHB-Lite address. 7-bit address on the AHB-Lite interface.

Name	Type	Description
HWRITE	In	AHB-Lite write. When High, it indicates that the current transaction is a write. When Low, it indicates that the current transaction is a read.
HREADYIN	In	When High, the HREADY signal indicates to the master and all slaves, that the previous transfer is complete.
HREADYOUT	Out	When High, the HREADYOUT signal indicates that the transfer has finished on the bus. This signal can be driven Low to extend a transfer.
HTRANS[1:0]	In	AHB-Lite transfer type. Indicates the transfer type of the current transaction. b00: IDLE b01: BUSY b10: NONSEQUENTIAL b11: SEQUENTIAL (not supported) **Note:** The PDMA engine only performs single cycle accesses (no bursts), so sequential transfers are not supported.
HSIZE	In	AHB-Lite transfer size. Indicates the size of the current transfer (8-/16-/32-/64-bit transactions only): bx00: 8-bit (byte) transaction bx01: 16-bit (half word) transaction bx10 : 32-bit (word) transaction bx11: 64-bit (double word) transaction (not supported)
HWDATA[31:0]	In	AHB-Lite write data. Write data from the AHB-Lite master to the AHB-Lite slave.
HRESP	Out	AHB-Lite response status. When driven High at the end of a transaction, it indicates that the transaction has completed with errors. When driven Low at the end of a transaction, it indicates that the transaction has completed successfully.
HRDATA[31:0]	Out	AHB-Lite read data. Read data from the AHB-Lite slave to the AHB-Lite master.
Interrupt or Status signals (to DMA engine)		
WRFIFORFM	Out	Indicates that there is room to store another 32-bit word in the write FIFO.
RDFIFODAVAIL	Out	Indicates that there is a 32-bit word available to read from the read FIFO.
INTERRUPT	Out	Interrupt output, asserted if any of the masked interrupt bits in the ISR/SBISR /MBISR are asserted.

5 Timing Diagrams

AHB-Lite Master interface compliant host can access all the CoreMMC registers. The following figures shows the AHB-Lite read and write transfer timing diagrams.

AHB-Lite Interface Timing

The following figures depict typical write cycle and read cycle timing relationships relative to the system clock, HCLK.

Figure 6 • AHB-Lite Read Transfer

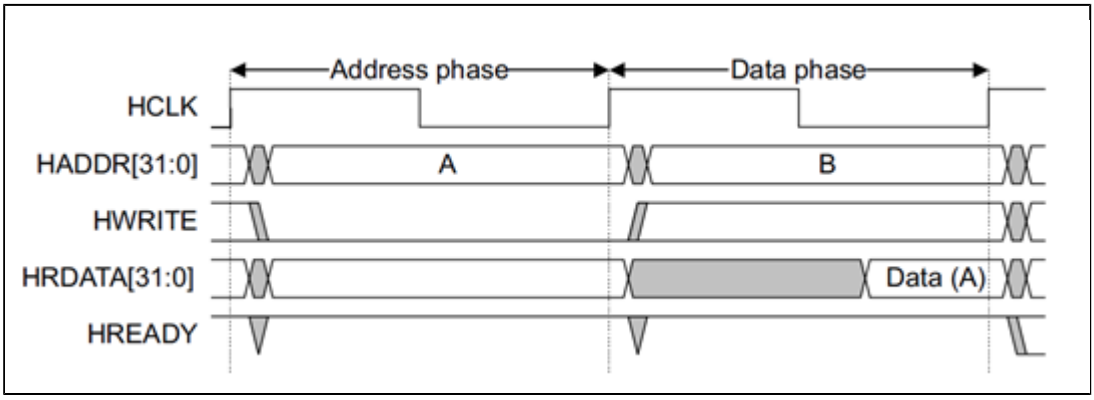
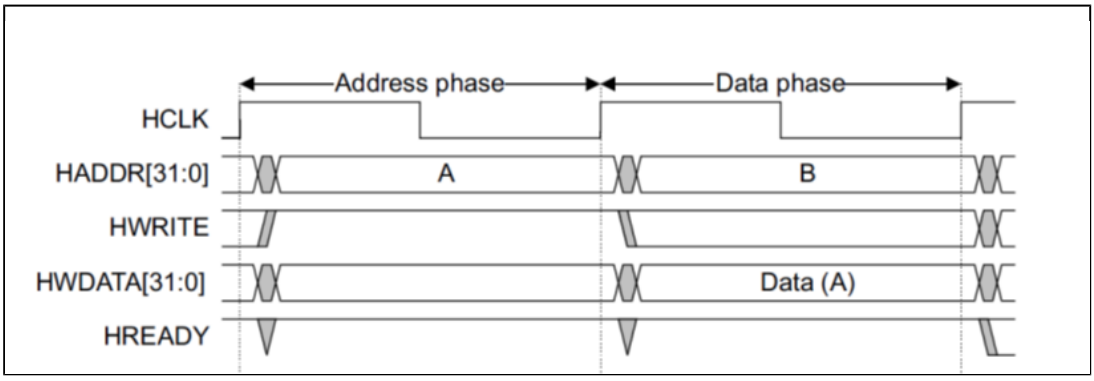


Figure 7 • AHB-Lite Write Transfer



6 Register Map and Descriptions

The external AHB master uses a 32-bit AHB slave interface for accessing eMMC registers. The following tables describe various registers and their descriptions.

6.1 Register Summary

The following table lists the register names and descriptions for CoreMMC. Values shown in tables are in hexadecimal format; type designations: R = read only; W = write only; R/W = read/write.

Table 5 • CoreMMC Internal Register Address Map

Address	Register Name	Mnemonic	Type	Width (bits)	Reset Value	Description (8-bit)
0x00	Status Register	SR	R/W	8	0x18	Indicates the status of transactions in the core such as response received and errors logged. This register does not cause interrupts to be asserted.
0x01	Version register	VR	R	8	0x3	This register defines the version of core along with the parameters used to generate the core. Default values are shown based on default parameters.
0x02	Major Version Register	MJVR	R	8	0x3	Indicates the major version of core version number.
0x03	Minor Version Register	MIVR	R	8	0x0	Indicates the minor version of core version number.
0x04	Command Index	CMDX	R/W	8	0x0	Command Index (command register 0) defines the command number to be sent to the slave device.
0x08	Command Arg1	Arg1	R/W	8	0x0	Command Argument1. Sets the first byte of the argument for the Command being issued to the slave. Argument1, Argument2, Argument3, and Argument4 compose the full argument. All four bytes can be written together.
0x09	Command Arg2	Arg2	R/W	8	0x0	Command Argument2. Sets the Second byte of the argument for the Command being issued to the slave.
0x0a	Command Arg3	Arg3	R/W	8	0x0	Command Argument3. Sets the Third byte of the argument for the Command being issued to the slave.
0x0b	Command Arg4	Arg4	R/W	8	0x0	Command Argument4. Sets the Fourth byte of the argument for the Command being issued to the slave. Writing this register causes command to be sent to slave device (whether written as Argument4 or as word write to Argument4, Argument3, Argument2, and Argument1).
0x10	Response Register0	RR0	R	8	0x0	Response Register 0
0x14	Response Register1	RR1	R	8	0x0	Response Register 1

Address	Register Name	Mnemonic	Type	Width (bits)	Reset Value	Description (8-bit)
0x15	Response Register2	RR2	R	8	0x0	Response Register 2
0x16	Response Register3	RR3	R	8	0x0	Response Register 3
0x17	Response Register4	RR4	R	8	0x0	Response Register 4
0x18	Response Register5	RR5	R	8	0x0	Response Register 5
0x19	Response Register6	RR6	R	8	0x0	Response Register 6
0x1a	Response Register7	RR7	R	8	0x0	Response Register 7
0x1b	Response Register8	RR8	R	8	0x0	Response Register 8
0x1c	Response Register9	RR9	R	8	0x0	Response Register 9
0x1d	Response Register10	RR10	R	8	0x0	Response Register 10
0x1e	Response Register11	RR11	R	8	0x0	Response Register 11
0x1f	Response Register12	RR12	R	8	0x0	Response Register 12
0x20	Response Register13	RR13	R	8	0x0	Response Register 13
0x21	Response Register14	RR14	R	8	0x0	Response Register 14
0x22	Response Register15	RR15	R	8	0x0	Response Register 15
0x24	Write Data Register	WDR	W	32	0x0	Write Data register (32 bits)
0x28	Read Data Register	RDR	R	32	0x0	Read Data register (32 bits)
0x2C	Interrupt Mask Register	IMR	R/W	8	0x0	Indicates the bits in the ISR that can cause the interrupt to be asserted. When a bit is set to 1 in this register, if the corresponding bit in ISR is asserted, then the Interrupt line is asserted.
0x2D	Singe Block Interrupt Mask register	SBIMR	R/W	8	0x0	Indicates the bits in the SBISR that can cause the interrupt to be asserted. When a bit is set to 1 in this register, if the corresponding bit in SBISR is asserted, then the Interrupt line is asserted.
0x2E	Multiple Block Interrupt Mask register	MBIMR	R/W	8	0x0	Indicates the bits in the MBISR that can cause the interrupt to be asserted. When a bit is set to 1 in this register, if the corresponding bit in MBISR is asserted, then the Interrupt line is asserted.

Address	Register Name	Mnemonic	Type	Width (bits)	Reset Value	Description (8-bit)
0x30	Interrupt Status register	ISR	R	8	0x0	Interrupt Status register
0x31	Single Block Interrupt Status register	SBISR	R	8	0x0	Single Block Interrupt Status register
0x32	Multiple Block Interrupt Status register	MBISR	R	8	0x0	Multiple Block Interrupt Status register
0x34	Interrupt Clear register	ICR	W	8	0x0	Interrupt Clear register
0x35	Single Block Interrupt Clear register	SBICR	W	8	0x0	Single Block Interrupt Clear register
0x36	Multiple Block Interrupt Clear register	MBICR	W	8	0x0	Multiple Block Interrupt Clear register
0x38	Control Register	CTRL	R/W	8	0x0C	Control register
0x39	Single Block Control and Status register	SBCSR	R/W	8	0x0	Single Block Control and Status register
0x3A	Multiple Block Control and Status register	MBCSR	R/W	8	0x0	Multiple Block Control and Status register
0x3C	Response Time-out register	RSPTO	R/W	8	0x40	Response Timeout register
0x40	Data Time-out register	DATATO	R/W	32	0x400	Data Timeout register
0x44	Block Length register	BLR	R/W	16	0x200	Block Length
0x48	Data Control Register	DCTRL	R/W	8	0x00	Data size control
0x4C	Clock Register	CLKR	R/W	8	0x3F	Clock register
0x50	Block Count Register	BCR	R/W	16	0x00	Block Count register

6.1.1 Status Register

This register contains the status and error bits pertaining to the operation of CoreMMC.

Table 6 • Status Register

HADDR[6:0]	Register Name	Type	Width	Reset Value	Description
0x00	Status	R/W	8	0x18	Status Register provides the information on the internal status of the core and error flags.

Note: This register does not actually reflect the status of the slave MMC device. Information on the MMC device can be extracted by checking the Response Registers. The following table shows the CoreMMC Status Register.

Table 7 • Status Register Bits

Bit(s)	Type	Name	Description
7	R	rdre	Response data ready: Status bit indicates the response waiting to be read from RR0-RR15. Reading any of those registers will clear this bit.
6	R	swff	Write FIFO full. Status bit indicates that the write FIFO is currently full.
5	R	srff	Read FIFO full. Status bit indicates that the read FIFO is currently full.
4	R	swfe	Write FIFO empty. Status bit indicates that the write FIFO is currently empty.
3	R	srfe	Read FIFO empty. Status bit indicates that the read FIFO is currently empty.
2	R/W	ebod	Buffer overflow detected. While receiving a block of data from the MMC device, a full read FIFO was detected. This bit stays set until it is cleared by writing a 1 to it.
1	R/W	ebud	Buffer underrun detected. While writing a block of data, the write FIFO ran out of bytes. This bit stays set until it is cleared by writing a 1 to it.
0	R/W	ecrd	CRC error detected. CRC mismatch on incoming response from MMC slave. This bit stays set until it is cleared by writing a 1 to it.

6.2 CoreMMC Version Register

This register contains the version of the core and the parameters used to generate the instance of the core. The following table shows the CoreMMC Version Register.

Table 8 • Version Register

HADDR[6:0]	Register Name	Type	Width	Reset Value	Description
0x01	Version	R/W	8	0x03	This register provides the version of core along with the parameters used to generate the core. Default values depend on configuration values. Default values are shown based on the default parameters.

Table 9 • Version Register Bit Definitions

Bit(s)	Type	Name	Description
7:6	R	-	Reserved
5:4	R	fifo_dep	FIFO Depth. The value of parameter FIFO_DEPTH used to generate core.
3:2	R	mmc_dw	MMC Data Width. Based value of parameter MMC_DWIDTH used to generate core. mmc_dw encoding is: 00: 1-bit 01: 4-bit 10: 8-bit **Note:** This coding is different to MMC_DWIDTH codes.
1:0	R	rev	Revision of this instance of CoreMMC. rev coding is: 0, 1, 2: Initial CoreMMC instance 3: Refer to the Major Version & Minor Version registers for information on this instances version.

6.2.1 Major Version Register

This register contains the major version of the core. The following table shows the CoreMMC Major Version Register.

Table 10 • Major Version Register

HADDR[6:0]	Register Name	Type	Width	Reset Value	Description
0x02	Major Version	R	8	0x03	This register contains the major version number of the core.

6.2.2 Minor Version Register

This register contains the minor version of the core. The following table shows the CoreMMC Minor Version Register.

Table 11 • Minor Version Register

HADDR[6:0]	Register Name	Type	Width	Reset Value	Description
0x03	Minor Version	R	8	0x00	This register contains the minor version number of the core.

6.2.3 Command Index Register

Command Index (or command register 0) defines the command number to be sent to the slave device.

Table 12 • Command Index Register

HADDR[6:0]	Register Name	Type	Width	Reset Value	Description
0x04	CMDX	R/W	8	0x0	Command Index (or command register 0) defines the command number to be sent to the slave device.

Table 13 • Command Index Register Bit Definitions

Bit(s)	Type	Name	Description
7:0	RRR	--	Reserved
5:0	R/WR	CMDXCMD	Command Index (or command register 0)

6.2.4 Command Argument1 Register

This register contains first byte (MSB) of command argument to be sent to the MMC slave.

Table 14 • Command Argument1 Register

HADDR[6:0]	Register Name	Type	Width	Reset Value	Description
0x08	Arg1	R/W	8	0x0	Command Argument1 sets the first byte (MSB) of the argument for command to be sent to the slave. Argument1, Argument2, Argument3, and Argument4 compose the full argument. All four bytes can be written together.

6.2.5 Command Argument2 Register

This register contains the second byte of command argument to be sent to the MMC slave.

Table 15 • Command Argument2 Register

HADDR[6:0]	Register Name	Type	Width	Reset Value	Description
0x09	Arg2	R/W	8	0x0	Command Argument2 is the second byte of argument for command to be sent to the slave.

6.2.6 Command Argument3 Register

This register contains the third byte of command argument to be sent to the MMC slave.

Table 16 • Command Argument3 Register

HADDR[6:0]	Register Name	Type	Width	Reset Value	Description
0x0a	Arg3	R/W	8	0x0	Command Argument3 is the third byte of argument for command to be sent to the slave.

6.2.7 Command Argument4 Register

This register contains the fourth byte (LSB) of command argument to be sent to the MMC slave.

Table 17 • Command Argument4 Register

HADDR[6:0]	Register Name	Type	Width	Reset Value	Description
0x0b	Arg4	R/W	8	0x0	Command Argument4 is the fourth byte (LSB) of argument for command to be sent to the slave. The writing of this byte initiates the sending of a command (composed of CMDX, Arg1, Arg2, Arg3, Arg4) to the slave MMC device. Once Argument4 is written, these four registers (CMDX, Arg1, Arg2, Arg3, Arg4) should not be written again until the command is completed.

6.2.8 Response Register0

This register contains the first byte (MSB) of the response received from the slave MMC device. This is updated by the core when starri is set and cleared, when Argument4 register is written.

Table 18 • Response Register0

HADDR[6:0]	Register Name	Type	Width	Reset Value	Description
0x10	RR0	R	8	0x0	Response Register0 is the first byte (MSB) of response from the MMC device.

6.2.9 Response Register1

This register contains the second byte of the response received from the slave MMC device. This is updated by the core when starri is set and cleared, when Argument4 register is written.

Table 19 • Response Register1

HADDR[6:0]	Register Name	Type	Width	Reset Value	Description
0x14	RR1	R	8	0x0	Response Register1 is the second byte of response from the MMC device.

6.2.10 Response Register2

This register contains the third byte of the response received from the slave MMC device. This is updated by the core when starri is set and cleared, when Argument4 register is written.

Table 20 • Response Register2

HADDR[6:0]	Register Name	Type	Width	Reset Value	Description
0x15	RR2	R	8	0x0	Response Register2 is the third byte of response from the MMC device.

6.2.11 Response Register3

This register contains the fourth byte of the response received from the slave MMC device. This is updated by the core when starri is set and cleared, when Argument4 register is written.

Table 21 • Response Register3

HADDR[6:0]	Register Name	Type	Width	Reset Value	Description
0x16	RR3	R	8	0x0	Response Register3 is the fourth byte of response from the MMC device.

6.2.12 Response Register4

This register contains the fifth byte of the response received from the slave MMC device. This is updated by the core when starri is set and cleared, when Argument4 register is written.

Table 22 • Response Register4

HADDR[6:0]	Register Name	Type	Width	Reset Value	Description
0x17	RR4	R	8	0x0	Response Register4 is the fifth byte of response from the MMC device.

6.2.13 Response Register5

This register contains the sixth byte of the response received from the slave MMC device. This is updated by the core when starri is set and cleared, when Argument4 register is written.

Table 23 • Response Register5

HADDR[6:0]	Register Name	Type	Width	Reset Value	Description
0x18	RR5	R	8	0x0	Response Register5 is the sixth byte of response from the MMC device.

6.2.14 Response Register6

This register contains the seventh byte of the response received from the slave MMC device. This is updated by the core when starri is set and cleared, when Argument4 register is written.

Table 24 • Response Register6

HADDR[6:0]	Register Name	Type	Width	Reset Value	Description
0x19	RR6	R	8	0x0	Response Register6 is the seventh byte of response from the MMC device.

6.2.15 Response Register7

This register contains the eighth byte of the response received from the slave MMC device. This is updated by the core when starri is set and cleared, when Argument4 register is written.

Table 25 • Response Register7

HADDR[6:0]	Register Name	Type	Width	Reset Value	Description
0x1A	RR7	R	8	0x0	Response Register7 is the eighth byte of response from the MMC device.

6.2.16 Response Register8

This register contains the ninth byte of the response received from the slave MMC device. This is updated by the core when starri is set and cleared, when Argument4 register is written.

Table 26 • Response Register8

HADDR[6:0]	Register Name	Type	Width	Reset Value	Description
0x1B	RR8	R	8	0x0	Response Register8 is the ninth byte of response from the MMC device.

6.2.17 Response Register9

This register contains the tenth byte of the response received from the slave MMC device. This is updated by the core when starri is set and cleared, when Argument4 register is written.

Table 27 • Response Register9

HADDR[6:0]	Register Name	Type	Width	Reset Value	Description
0x1C	RR9	R	8	0x0	Response Register9 is the tenth byte of response from the MMC device.

6.2.18 Response Register10

This register contains the eleventh byte of the response received from the slave MMC device. This is updated by the core when starri is set and cleared, when Argument4 register is written.

Table 28 • Response Register10

HADDR[6:0]	Register Name	Type	Width	Reset Value	Description
0x1D	RR10	R	8	0x0	Response Register10 is the eleventh byte of response from the MMC device.

6.2.19 Response Register11

This register contains the twelfth byte of the response received from the slave MMC device. This is updated by the core when starri is set and cleared, when Argument4 register is written.

Table 29 • Response Register11

HADDR[6:0]	Register Name	Type	Width	Reset Value	Description
0x1E	RR11	R	8	0x0	Response Register11 is the twelfth byte of response from the MMC device.

6.2.20 Response Register12

This register contains the thirteenth byte of the response received from the slave MMC device. This is updated by the core when starri is set and cleared, when Argument4 register is written.

Table 30 • Response Register12

HADDR[6:0]	Register Name	Type	Width	Reset Value	Description
0x1F	RR12	R	8	0x0	Response Register12 is the thirteenth byte of response from the MMC device.

6.2.21 Response Register13

This register contains the fourteenth byte of the response received from the slave MMC device. This is updated by the core when starri is set and cleared, when Argument4 register is written.

Table 31 • Response Register13

HADDR[6:0]	Register Name	Type	Width	Reset Value	Description
0x20	RR13	R	8	0x0	Response Register13 is the fourteenth byte of response from the MMC device.

6.2.22 Response Register14

This register contains the fifteenth byte of the response received from the slave MMC device. This is updated by the core when starri is set and cleared, when Argument4 register is written.

Table 32 • Response Register14

HADDR[6:0]	Register Name	Type	Width	Reset Value	Description
0x21	RR14	R	8	0x0	Response Register14 is the fifteenth byte of response from the MMC device.

6.2.23 Response Register15

This register contains the sixteenth byte of the response received from the slave MMC device. This is updated by the core when starri is set and cleared, when Argument4 register is written.

Table 33 • Response Register15

HADDR[6:0]	Register Name	Type	Width	Reset Value	Description
0x22	RR15	R	8	0x0	Response Register15 is the sixteenth byte of response from the MMC device.

6.2.24 Write Data Register

This register provides access to the Write FIFO. Writing to this register pushes the write data into the FIFO. The FIFO width is fixed at 32 bits. The FIFO depth is defined by the FIFO_DEPTH parameter at the time of instantiation.

Table 34 • Write Data Registers

HADDR[6:0]	Register Name	Type	Width	Reset Value	Description
0x24	WDR	W	32	0x0	Write Data register. Data written to this register gets pushed into Write Data FIFO and is sent to the slave MMC device when a block write is performed.

Note: Only 32-bit operations must be performed on this register. Writing an 8- or 16-bit value to this register causes malfunction.

6.2.25 Read Data Register

This register provides access to the Read FIFO. Reading from this register pops the read data from the FIFO. The FIFO width is fixed at 32 bits. The FIFO depth is defined by the FIFO_DEPTH parameter at the time of instantiation.

Table 35 • Read Data Register

HADDR[6:0]	Register Name	Type	Width	Reset Value	Description
0x28	RDR	R	32	0x0	Read Data register. Data read from this register gets popped from the Read Data FIFO. The Read Data FIFO is written with the data from the slave MMC device when a block read is performed.

Note: Only 32-bit operations must be performed on this register. Reading an 8- or 16-bit value from this register will return invalid data.

6.2.26 Interrupt Mask Register

The Interrupt Mask register enables or masks Interrupt status register interrupts from being triggered on the Interrupt port of the CoreMMC. The following table shows the Interrupt Mask register.

Table 36 • Interrupt Mask Register

HADDR[6:0]	Register Name	Type	Width	Reset Value	Description
0x2C	IMR	R/W	8	0x0	This register enables or masks Interrupt Status Register interrupts from being generated on Interrupt port of the CoreMMC.

Table 37 • Interrupt Mask Register Bit Definitions

Bit(s)	Type	Name	Description
7	R/W	mskuer	When set to 1, it enables the user error (stauer) interrupt.
6	R/W	msksbi	When set to 1, it enables the response start bit error/response time out error (stasbi) interrupt.
5	R/W	msktbi	When set to 1, it enables the stop bit error (statbi) interrupt.
4	R/W	msktxi	When set to 1, it enables the response transmit bit error (statxi) interrupt.
3	R/W	mskrri	When set to 1, it enables the command response receive (starri) interrupt.
2	R/W	mskcsi	When set to 1, it enables the command send (stacsi) interrupt.
1	R/W	mskboi	When set to 1, it enables the buffer overflow (staboi) interrupt.
0	R/W	mskbui	When set to 1, it enables the buffer under run (stabui) interrupt.

6.2.27 Single Block Interrupt Mask Register

The Single Block Interrupt Mask register enables or masks Single Block Interrupt Status Register interrupts from being triggered on the Interrupt port of the CoreMMC. The following table shows the Single Block Interrupt Mask register.

Table 38 • Single Block Interrupt Mask Register

HADDR[6:0]	Register Name	Type	Width	Reset Value	Description
0x2D	SBIMR	R/W	8	0x0	This register enables or masks Single Block Interrupt Status Register interrupts from being generated on Interrupt port of the CoreMMC.

Table 39 • Single Block Interrupt Mask Register Bit Definitions

Bit(s)	Type	Name	Description
7	R/W	msksbwdatainfifoto	When set to 1, it enables the single block write timeout error (stasbwdatainfifoto) interrupt
6	R/W	msksbwbusyto	When set to 1, it enables the single block write busy timeout error (stasbwbusyto) interrupt
5	R/W	msksbwcrcstaerr	When set to 1, it enables the single block write crc response timeout error (stasbwcrcstaerr) interrupt
4	R/W	msksbrstperr	When set to 1, it enables single block read stop error (stasbrstperr) interrupt
3	R/W	msksbrstto	When set to 1, it enables the single block read start timeout (stasbrstto) interrupt
2	R/W	msksbcrcerr	When set to 1, it enables the single block read or write crc error (stasbcrcerr) interrupt
1	R/W	msksbrdone	When set to 1, it enables the single block read done (stasbrdone) interrupt
0	R/W	msksbwdone	When set to 1, it enables the single block write done (stasbwdone) interrupt

6.2.28 Multiple Block Interrupt Mask Register

The Multiple Block Interrupt Mask register enables or masks Multiple Block Interrupt Status Register interrupts from being triggered on the Interrupt port of the CoreMMC. The following table shows the Multiple Block Interrupt Mask register.

Table 40 • Multiple Block Interrupt Mask Register

HADDR[6:0]	Register Name	Type	Width	Reset Value	Description
0x2E	MBIMR	R/W	8	0x0	This register enables or masks Multiple Block Interrupt Status Register interrupts from being generated on Interrupt port of the CoreMMC.

Table 41 • Multiple Block Interrupt Mask Register Bit Definitions

Bit(s)	Type	Name	Description
7	R/W	mskmbwdatainfifoto	When set to 1, it enables the multiple block write FIFO timeout (stambwdatainfifoto) interrupt

Bit(s)	Type	Name	Description
6	R/W	mskmbwbusyto	When set to 1, it enables the multiple block write busy timeout (stambwbusyto) interrupt
5	R/W	mskmbwcrcstaerr	When set to 1, it enables the multiple block write crc response error (stambwcrcstaerr) interrupt
4	R/W	mskmbrstperr	When set to 1, it enables the multiple block read stop error (stambrstperr) interrupt
3	R/W	mskmbrstto	When set to 1, it enables the multiple block read start timeout (stambrstto) interrupt
2	R/W	mskmbcrcerr	When set to 1, it enables the multiple block read or write crc error (stambcrcerr) interrupt
1	R/W	mskmbrdone	When set to 1, it enables the multiple block read done (stambrdone) interrupt
0	R/W	mskmbwdone	When set to 1, it enables the multiple block write done (stambwdone) interrupt

6.2.29 Interrupt Status Register

The Interrupt Status Register stores the current status and errors bits for the transactions performed. These bits are set regardless of the Interrupt Mask register. The following table describes the Interrupt Status register.

Table 42 • Interrupt Status Register

HADDR[6:0]	Register Name	Type	Width	Reset Value	Description
0x30	ISR	R	8	0x0	This register provides status and error flags of transactions.

Table 43 • Interrupt Status Register Bit Definitions

Bit(s)	Type	Name	Description
7	R	stauer	User error is detected. Write FIFO overrun or Read FIFO underrun error.
6	R	stasbi	Response start bit error detected or time-out error while waiting for a response.
5	R	statbi	Stop bit error is detected on response to command
4	R	statxi	Transmit bit error is detected on response to command
3	R	starri	Response to command received
2	R	stacsi	Command sent
1	R	staboi	Buffer overflow occurred. Read FIFO was full when more data attempted to be written into it during Block Read.
0	R	stabui	Underrun occurred. Write FIFO was empty when more data attempted to be sent during block write.

6.2.30 Single Block Interrupt Status Register

The Single Block Interrupt Status Register provides the current status and error bits for single block read or write transactions performed. These bits are set regardless of the Single Block Interrupt Mask register. The following table describes the Single Block Interrupt status register.

Table 44 • Single Block Interrupt Status Register

HADDR[6:0]	Register Name	Type	Width	Reset Value	Description
0x31	SBISR	R	8	0x0	This register provides the status and error bits for single block read or write transactions.

Table 45 • Single Block Interrupt Status Register Bit Definitions

Bit(s)	Type	Name	Description
7	R	stasbwdatainfifoto	Single block Write FIFO timeout. Asserted when less than block length amount of data in the Write FIFO after the period defined in the DATOTO register at the start of a single block write transfer. Prevents Write FIFO underruns during single block write transfers.
6	R	stasbwbusyto	Single block write busy timeout. Set when eMMC slave device holds DAT0 low for longer than the period defined in DATATO register at the start of a single block write transfer. Indicates that the slave device is not ready to receive data.

Bit(s)	Type	Name	Description
5	R	stasbwcrcstaerr	Single block write CRC response error. Set when start bit of CRC Status frame not received within period defined in DATATO register or when no valid stop bit detected for CRC status frame.
4	R	stasbrstperr	Single Block Read Stop Error. Set when valid stop bit not detected on all active DATI lines.
3	R	stasbrstto	Single Block Read start time-out. Set when no incoming start-bit found on DATI for period defined in DATATO register.
2	R	stasbcrcerr	Single block read or write encountered CRC error.
1	R	stasbrdone	Single block read done
0	R	stasbwdone	Single block write done

6.2.31 Multiple Block Interrupt Status Register

The Multiple Block Interrupt Status Register provides the current status and error bits for multiple block read or write transactions performed. These bits are set regardless of the Multiple Block Interrupt Mask register. The following table describes the Multiple Block Interrupt status register.

Table 46 • Multiple Block Interrupt Status Register

HADDR[6:0]	Register Name	Type	Width	Reset Value	Description
0x32	MBISR	R	8	0x0	This register provides the status and error bits for multiple block read or write transactions.

Table 47 • Multiple Block Interrupt Status Register Bit Definitions

Bit(s)	Type	Name	Description
7	R	stambwdatainfifoto	Multiple block Write FIFO timeout. Asserted when less than block length amount of data in the Write FIFO after the period defined in the DATOTO register at the start of a block within a multiple block write transfer. Prevents Write FIFO underruns during Multiple block write transfers.
6	R	stambwbusyto	Multiple block write busy timeout. Set when eMMC slave device holds DAT0 low for longer than the period defined in DATATO register at the start of a block within a multiple block write transfer. Indicates that the slave device is not ready to receive data.
5	R	stambwcrcstaerr	Multiple block write CRC response error. Set when start bit of CRC Status frame not received within period defined in DATATO register or when no valid stop bit detected for CRC status frame for a block within a multiple block write transfer.
4	R	stambrstperr	Multiple Block Read Stop Error. Set when valid stop bit not detected on all active DATI lines for a block within a multiple block read transfer.
3	R	stambrstto	Multiple Block Read start time-out. Set when no incoming start-bit found on DAT for period defined in DATATO register for a block within a multiple block read transfer.
2	R	stambcrcerr	Multiple Block Read or Write encountered CRC Error.
1	R	stambrdone	Multiple Block Read Done
0	R	stambwdone	Multiple Block Write Done

6.2.32 Interrupt Clear Register

The Interrupt Clear Register is a write-only register used to clear (individually) the ISR bits described in Table 39 (see page 27). This can clear an interrupt, if the associated bit is set in the Interrupt Mask register. The following table describes the Interrupt Clear register.

Table 48 • Interrupt Clear Register

HADDR[6:0]	Register Name	Type	Width	Reset Value	Description
0x34	ICR	W	8	0x0	This register clears (individually) the corresponding bit in the interrupt status register described in Table 39 (see page 27).

Table 49 • Interrupt Clear Register Bit Definitions

Bit(s)	Type	Name	Description
7	W	clruer	Clear user error (stauer) bit.
6	W	clrsbi	Clear response start bit error/response time out error (stasbi) bit
5	W	clrtbi	Clear stop bit error (statbi) bit
4	W	clrtxi	Clear response transmit bit error (statxi) bit
3	W	clrrri	Clear command response receive (starri) bit
2	W	clrcsi	Clear command send (stacsi) bit
1	W	clrboi	Clear buffer overflow (staboi) bit
0	W	clrbui	Clear buffer under run (stabui) bit

6.2.33 Single Block Interrupt Clear Register

The Single Block Interrupt Clear Register is a write-only register used to clear (individually) the Single Block Interrupt Status register bits described in Table 41 (see page 27) . This can clear an interrupt if the associated bit is set in the Single Block Interrupt Mask register . The following table describes the Single Block Interrupt Clear register.

Table 50 • Single Block Interrupt Clear Register

HADDR[6:0]	Register Name	Type	Width	Reset Value	Description
0x35	SBICR	W	8	0x0	Single block interrupt clear register. This register clears (individually) the corresponding bit in the single block interrupt status register described in Table 41 (see page 27).

Table 51 • Single Block Interrupt Clear Register Bit Definitions

Bit(s)	Type	Name	Description
7	W	clrsbwdatainfifoto	Clear single block write timeout error (stasbwdatainfifoto) bit
6	W	clrsbwbusyto	Clear single block write busy timeout error (stasbwbusyto) bit
5	W	clrsbwcrcstaerr	Clear single block write crc response timeout error (stasbwcrcstaerr) bit
4	W	clrsbrstperr	Clear single block read stop error (stasbrstperr) bit
3	W	clrsbrstto	Clear single block read start timeout error (stasbrstto) bit
2	W	clrsbcrcerr	Clear single block read or write crc error (stasbcrcerr) bit

Bit(s)	Type	Name	Description
1	W	clrsbrdone	Clear single block read done (stasbrdone) bit
0	W	clrsbwdone	Clear single block write done (stasbwdone) bit

6.2.34 Multiple Block Interrupt Clear Register

The Multiple Block Interrupt Clear Register is a write-only register used to clear (individually) the Multiple Block Interrupt Status register bits described in Table 43 (see page 28) . This can clear an interrupt if the associated bit is set in the Multiple Block Interrupt Mask Register. The following table describes the Multiple Block Interrupt Clear register.

Table 52 • Multiple Block Interrupt Clear Register

HADDR[6:0]	Register Name	Type	Width	Reset Value	Description
0x36	MBICR	W	8	0x0	Multiple block interrupt clear register. This register clears (individually) the corresponding bit in the multiple block interrupt status register described in Table 43 (see page 28).

Table 53 • Multiple Block Interrupt Clear Register Bit Definitions

Bit(s)	Type	Name	Description
7	W	clrmbwdatainfifoto	Clear multiple block write FIFO time out (stambwdatainfifoto) bit
6	W	clrmbwbusyto	Clear multiple block write busy timeout (stambwbusyto) bit
5	W	clrmbwcrcstaerr	Clear multiple block write crc response error (stambwcrcstaerr) bit
4	W	clrmbrstperr	Clear multiple block read stop error (stambrstperr) bit
3	W	clrmbrstto	Clear multiple block read start timeout (stambrstto) bit
2	W	clrmbcrcerr	Clear multiple block read or write crc error (stambcrcerr) bit
1	W	clrmbrdone	Clear multiple block read done (stambrdone) bit
0	W	clrmbwdone	Clear multiple block write done (stambwdone) bit

6.2.35 Control Register

The Control Register provides the control bits to define the operation of the core and to drive hardware pins to the eMMC slave device. The following table describes the Control register.

Table 54 • Control Register

HADDR[6:0]	Register Name	Type	Width	Reset Value	Description
0x38	CTRL	R/W	8	0x0C	This register provides the control bits for the operation of the core.

Table 55 • Control Register Bit Definitions

Bit(s)	Type	Name	Description
7	R	Busy	Slave device is indicating that it is busy by asserting DAT[0] low.
6	R	-	Reserved.
5	W	fiforeset	FIFO reset. Used to initialize the address pointers & flags in the Read and Write FIFOs back to default values.
4	R/W	cmdFrcLow	Force CMD line to 0 (low). Used for boot operation.

Bit(s)	Type	Name	Description
3	R	midle	MMC Idle. When set to 1, it indicates that the core is in Idle state (not sending command, or expecting response). This bit does not reflect when the core is transmitting/receiving data.
2	R/W	clkoe	CLK Output Enable. Enables CLK or disables it to allow the slave device to enter Low-power mode.
1	R/W	slrst	Slave Reset. When set, it applies a reset to the eMMC slave device through RST_N pin.
0	R/W	swrst	Software Reset. When set, it holds core in reset state and asserts RST_N pin Low.

6.2.36 Single Block Control and Status Register

The Single Block Control and Status Register provides control of single block read and write transactions and associated status bits from these operations. The following table describes the Single Block Control and Status register.

Table 56 • Single Block Control and Status Register

HADDR[6:0]	Register Name	Type	Width	Reset Value	Description
0x39	SBCSR	R/W	8	0x0	This register provides control and status bits for single block reads and writes.

Table 57 • Single Block Control and Status Register Bit Definitions

Bit(s)	Type	Name	Description
7	R	-	Reserved.
6:4	R	sbwst	Single Block Write Status – CRC Status bits. Updated when either stasbwdone, stasbcrcerr or stasbwcrcstaerr set for single block write transfers. (Good CRC Status results = 010)
3	R	sbcrcerr	Single Block CRC Error. Set when sbwdone or sbrdone asserted, if CRC error detected. Cleared by writing 1 to clrsbcrcerr in Single Block Interrupt Clear register.
2	R	sbdone	Single Block Done. Cleared when single block write or read start is set (sbwstrt or sbrstrt). Set by hardware when Single Block Write or Read is completed.
1	R/W	sbrstrt	Single Block Read Start. Initiates the read of a block of the length defined in the Block Length register from the eMMC slave device. Only held set for 1 HCLK period.
0	R/W	sbwstrt	Single Block Write Start. Initiates the write of a block of the length defined in Block Length register to the eMMC slave device. Only held set for 1 HCLK period.

6.2.37 Multiple Block Control and Status Register

The Multiple Block Control and Status Register provides control of multiple block read and write transactions and associated status bits from these operations. The following table describes the Multiple Block Control and Status register.

Table 58 • Multiple Block Control and Status Register

HADDR[6:0]	Register Name	Type	Width	Reset Value	Description
0x3A	MBCSR	R/W	8	0x0	This register provides control and status bits for multiple block reads and writes.

Table 59 • Multiple Block Control and Status Register Bit Definitions

Bit(s)	Type	Name	Description
7	R	-	Reserved.
6:4	R	mbwst	Multiple Block Write Status – CRC Status bits. Updated when either stambwdone, stambcrcerr or stambwcrcstaerr set for multiple block write transfer. (Good CRC Status results = 010)
3	R	mbcrcerr	Multiple Block CRC Error. Set when mbwdone or mbrdone asserted, if CRC error detected. Cleared by writing 1 to clrmbcrcerr in the Multiple Block Interrupt Clear register.
2	R	mbdone	Multiple Block Done. Cleared when multiple block write or read start is set (mbwstrt or mbrstrt). Set by hardware when Multiple Block Write or Read is completed.
1	R/W	mbrstrt	Multiple Block Read Start. Initiates a multiple block read of number of blocks defined in the Block Count register, with the number of bytes per block as per Block Length register. Only held set for 1 HCLK period.
0	R/W	mbwstrt	Multiple Block Write Start. Initiates a multiple block write of number of blocks defined in the Block Count register, with the number of bytes per block as per Block Length register. Only held set for 1 HCLK period.

6.2.38 Response Timeout Register

This register defines the number of clock ticks of HCLK, that the core waits for a response from the slave device after it has issued a command before timing out and setting the stasbi bit in the Interrupt Status Register.

Table 60 • Response Timeout Register

HADDR[6:0]	Register Name	Type	Width	Reset Value	Description
0x3C	RSPTO	R/W	8	0x40	Length of time (in HCLK ticks) that core waits for a response from the slave device.

6.2.39 Data Timeout Register

This register provides the number of clock ticks of HCLK, that the core waits for under the following circumstances.

- **Single/Multiple block write:**
  - Slave to indicate that it's idle by releasing DAT\[0\] (high), before timing out and setting either the stasbwbusyto bit in the SBISR or the stambwbusyto bit in the MBISR.
  - User to load at least a block length amount of data into the Write FIFO before timing out and setting either the stasbwdatainfifoto bit in the SBISR or the stambwdatainfifoto bit in the MBISR. The timeout guards against the occurrence of write underruns during single/multiple block write transfers.
- **Single/Multiple block read:**
  - Slave to load a valid start bit onto to the data bus before timing out and setting either the stasbrstto bit in the SBISR or the stambrstto bit in the MBISR.

Table 61 • Data Timeout Register

HADDR[6:0]	Register Name	Type	Width	Reset Value	Description
0x40	DATATO	R/W	32	0x400	Length of time (in HCLK ticks) that core waits for, for the cases mentioned in the Data Timeout Register in the preceding description.

6.2.40 Block Length Register

This register specifies the length in bytes of data blocks to the core. This register must be written before initiating single or multiple block write and read transfers.

Note: The block length must be specified to the eMMC slave device through a user initiated command and response sequence. Writing to this register will not specify the block length to the eMMC slave.

Table 62 • Block Length Register

HADDR[6:0]	Register Name	Type	Width	Reset Value	Description
0x44	BlockLen	R/W	16	0x200	Sets the length in bytes of data blocks used for single or multiple block write and read transfers.

6.2.41 Data Control Register

The register defines the number of DAT bus bits that the core will actively drive/sample during data transfers. This register must be defined by the user before initiating data transfers. This is a separate action from setting the MMC_DWIDTH parameter at the time of instantiation but is dependent upon the MMC_DWIDTH value set at the time of instantiation. For instance, the core may have been instantiated with a 4-bit MMC DAT width but operationally may be set to drive/sample 1-bit of the DAT bus. In this instance, the core can also be configured to drive/sample 4-bits (through dsize), but cannot be configured to drive/sample 8-bits.

Table 63 • Data Control Register

HADDR[6:0]	Register Name	Type	Width	Reset Value	Description
0x48	DCTRL	R/W	8	0x00	Data Control – defines size of active bits of DAT.

Table 64 • Data Control Register Bit Definitions

Bit(s)	Type	Name	Description
7:2	R	-	Reserved
1:0	R/W	dsize	DAT Size - Number of active DAT bits. User cannot select a size instantiated in the core, specified via MMC_DWIDTH. dsize coding is as follows: 00: 1-bit 01: 4-bit (Only available for MMC_DWIDTH = 4-bit or 8-bit) 10: 8-bit (Only available for MMC_DWIDTH = 8-bit) 11:Reserved

6.2.42 Clock Register

This register defines the period for the CLK pin (MMC interface clock). The value of CLKHP defines the half clock period for the CLK.

A suitable value needs to be written to this register initially before any commands are sent to the eMMC slave to generate a ~400 KHz MMC clock from the HCLK input (As per initialization clock frequency defined in the JEDEC Specification).

Once the initialization command-response sequences have been completed with the eMMC slave device, the user can initiate a SWITCH command to inform the eMMC slave device to change over to high-speed mode. A new value can then be written to this register to derive the desired run-time MMC interface clock frequency.

To achieve maximum overall system throughput, depending on the application, it may be beneficial to ensure that the processor controlling CoreMMC is running at its maximum operating frequency (along with HCLK) & then choose the CLKHP value based off this such that the MMC interface clock is running at as close as possible to the 52 MHz maximum defined in the JEDEC specification [R1], as overall system throughput is heavily impacted by AHB bandwidth.

Note: The default value set in this register is sufficient to derive a 400 KHz initialization MMC interface clock from a 50 MHz HCLK input. If the HCLK input differs from 50 MHz, the user must write a suitable value to this register prior to performing any command-response transactions with the eMMC slave device.

Table 65 • Clock Register

HADDR[6:0]	Register Name	Type	Width	Reset Value	Description
0x4C	CLKR	RW	8	0x3f	Clock Register – defines half period of CLK signal.

Table 66 • Clock Register Bit Definitions

Bit(s)	Type	Name	Description

Bit(s)	Type	Name	Description
7:0	R/W	CLKHP	CLK Half Period - defines the half period of the MMC interface clock (CLK), where the frequency of CLK is defined by the formula: $$f_{CLK} = \frac{f_{HCLK}}{2(CLKHP + 1)}$$ Example CLKHP values & the resultant CLK generated: CLKHP CLK 00: HCLK/2 01: HCLK/4 02: HCLK/6 03: HCLK/8 04: HCLK/10 ... 255: HCLK/512

Note: The maximum CLK operating frequency is HCLK/2 to support the 3 ns hold time of the MMC interface defined in the JEDEC specification [R1] (Data is loaded onto the bus by CoreMMC 1 HCLK cycle after the rising edge of CLK – Hold time is therefore equal to the period of HCLK).

6.2.43 Block Count Register

This register specifies the number of blocks in a multiple block transfer to the core. This register must be written to before initiating multiple block write or read transfers.

Note: The block count must be specified to the eMMC slave device through a user initiated command and response sequence. Writing to this register will not specify the block count to the eMMC slave.

Table 67 • Block Count Register

HADDR[6:0]	Register Name	Type	Width	Reset Value	Description
0x50	BlockCnt	R/W	16	0x00	Specifies the number of blocks in a multiple block transfer.

7 Tool Flows

The following sections describe installation and configuration of tools needed for CoreMMC.

7.1 Licensing

No license is required for the use of this core.

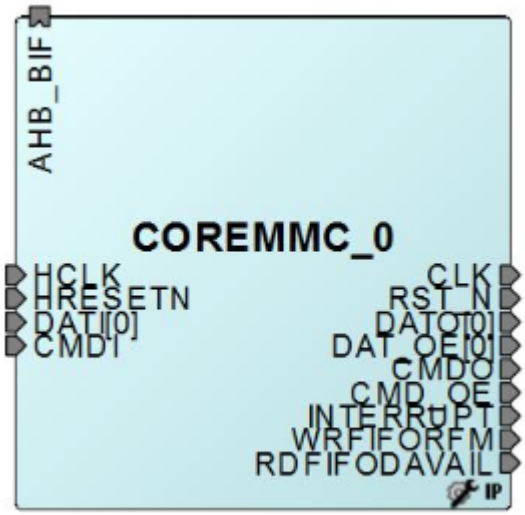
7.2 RTL

Complete RTL source code is provided for the core and testbenches.

7.3 SmartDesign

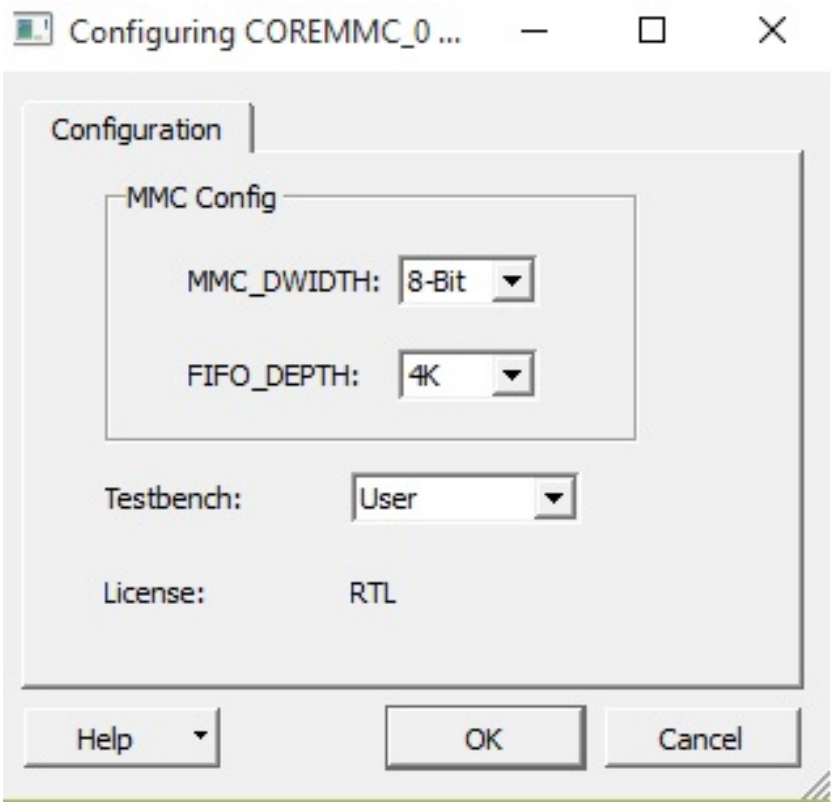
CoreMMC is pre-installed in the SmartDesign IP Deployment design environment. The following figure shows configuring the core using the configuration GUI within SmartDesign. For information on using SmartDesign to instantiate and generate cores, refer to the *Using DirectCore in Libero IDE User Guide*.

Figure 8 • CoreMMC Full I/O View



The following figure shows CoreMMC SmartDesign configuration with callouts to associated parameters.

Figure 9 • CoreMMC SmartDesign Configuration



7.4 Simulation Flows

The User Testbench for CoreMMC is included in all releases. To run simulations, select the User Testbench flow within SmartDesign and click **Save** and generate on the **Generate pane**. The User Testbench is selected through the Core Testbench Configuration GUI. When SmartDesign generates the Libero SoC project, it installs the user testbench files.

To run the user testbench, set the design route to the CoreMMC instantiation in the Libero SoC design hierarchy pane and click **Simulation** in the Libero SoC Design Flow window. This invokes ModelSim® and automatically runs the simulation.

7.4.1 User Testbench

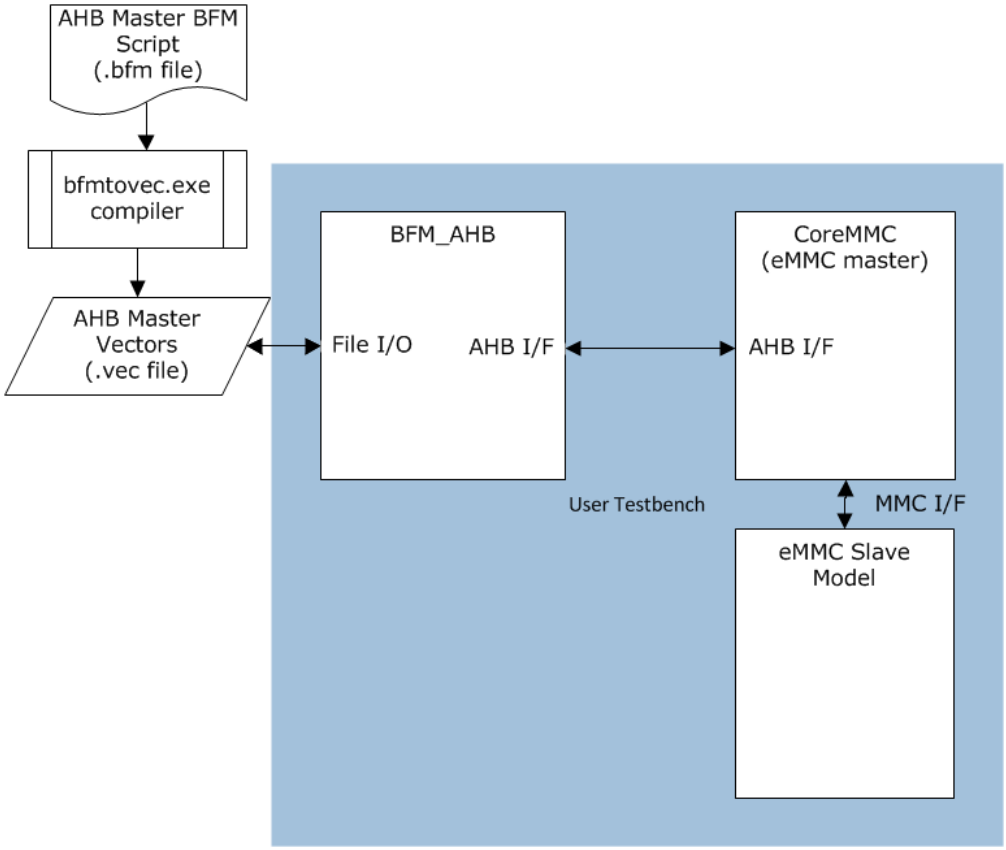
The following figure shows the hierarchal structure of the CoreMMC simulation testbench which includes an instance of CoreMMC, an eMMC slave model and a BFM AHB Master.

User tests are defined in the user_tb.bfm script which is compiled into a vector file and passed via parameter to the BFM AHB master. The BFM master emulates the operation of an AHB master controlling CoreMMC to communicate with an eMMC slave. The user_tb.bfm script packaged along with CoreMMC performs single and multiple block read and write transactions with the eMMC slave. An extensive set of procedures are defined in the user_tb.bfm script which demonstrate the intended flow for AHB master controlling CoreMMC. Users can modify the calls to these procedures in the user_tb.bfm script to generate custom simulation cases if required.

The eMMC slave model is a primitive model of an eMMC slave device with basic functionality such as support for a limited set of eMMC commands (Commands 0,1,2,3,9,7,8, 6, 16, 17, 18, 23, 24, 25), single and multiple block data transfers with 16 addressable 512-byte sectors.

Note: Support for SWITCH command (CMD6) is only implemented for writing to the DATA_WIDTH segment \[183\] of the EXT_CSD for configuring the number of DAT bits that the eMMC slave model actively drives/samples.

Figure 10 • CoreMMC User Testbench



7.5 Synthesis in Libero SoC

After setting the design root appropriately for your design, use the following steps to run the Synthesis:

1. Click **Synthesis** in the Libero SoC software. The **Synthesis** window appears, displaying the Synplicity® project.
2. Set Synplicity to use the Verilog 2001 standard if Verilog is being used.
3. Click **Run** to run the Synthesis.

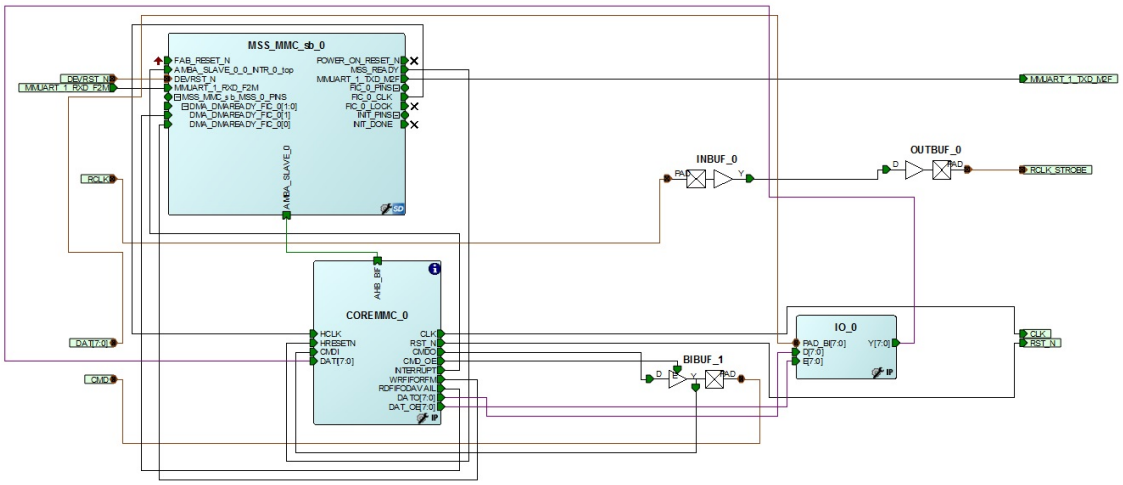
7.6 Place-and-Route in Libero SoC

After running Synthesis, run the Place and Route, by enabling the check box for **Timing Driven**, **High Effort Layout** and **Repair Minimum Delay Violations** under **Layout Options** in the Place and Route configuration window.

8 System Integration

This section provides information to ease the integration of CoreMMC.

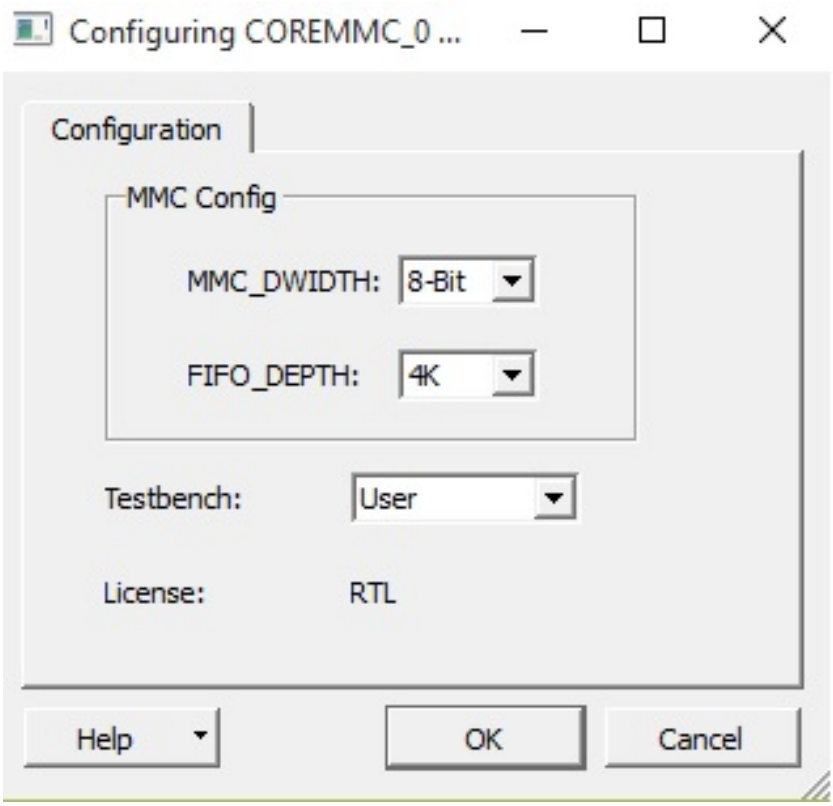
Figure 11 • CoreMMC System Integration



The example design described in this section contains CoreMMC which is connected to SmartFusion2 Microcontroller Subsystem.

The configuration of COREMMC_0 parameters is shown in the following figure.

Figure 12 • CoreMMC Configuration

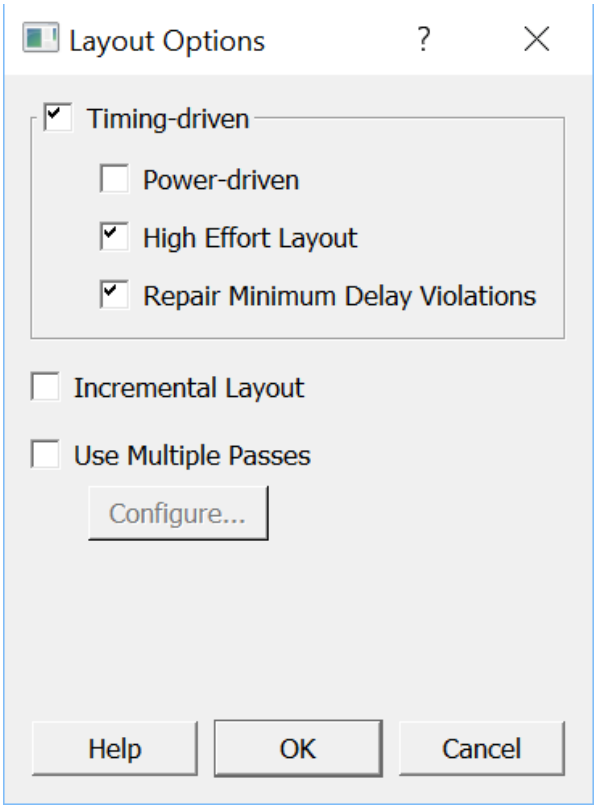


The system integration happens as per the following steps in this example.

- Output pin “MSS_HPMS_READY” of CoreResetP is used to drive COREMMC_0 reset pin “HRESETN”.
- The COREMMC_0 has HCLK and CLK clocks. Here CLK is an output clock for eMMC device clock.
- HCLK is a 104 MHz clock, driven from the output port “GL0” of CCC_0.
- CLK is a 400 KHz clock during the device initialization stage and is changed to 52 MHz during data transfer state.

- Run the Libero Place and Route, by enabling the check box for **Timing Driven**, **High Effort Layout** and **Repair Minimum Delay Violations** under **Layout Options** in the Place and Route configuration window as shown in the following figure.

Figure 13 • Place and Route Configuration



Note: The example design can be obtained from the Microsemi Technical Support.

9 Design Constraints

Designs with CoreMMC require the following constraints to be applied in the design flow to prevent the occurrence of timing violations in the design.

Note: Timing constraints explained in the following sections are for example design described in System Integration (see page 39). PLL instance name and port names are used from the example design, to provide the generated clock and output delay clock constraints. The constraints must be changed accordingly for the user who uses a different PLL instance name or port names,.

9.1 Enhanced Constraint Flow

The procedure for adding the constraints is as follows for the enhanced constraint flow.

1. Double-click **Constraints > Manage Constraints** in the **Design Flow** window and click the **timing** tab. Click on **derive constraints** to automatically create a constraints file containing the PLL constraints.
2. Select **Yes**, when prompted to automatically associate the derived constraint SDC file to the ' **Synthesis'**, '**Place and Route**' and '**Timing Verification**'.
3. In the Constraints manager, click on **new** and create a sdc file. Right-click on the newly created sdc file and set as target.
4. Select checkbox under "**Place and Route**" and "**Timing Verification**"
5. Add the generated clock constraints as shown below. There are two generated clock constraints that are used in this example. First generated clock is used to divide the source clock by 2 and second generated clock is used to route the generated clock in first step to the output CLK pad.

```
create_generated_clock -name {EMMC_CLK_DIV} -divide_by 2 -source [
get_pins { MSS_MMC_sb_0/CCC_0/CCC_INST/GL0 } ] -phase 0 [ get_pins {
*u_MMCMaster/u_MMCClkGen/CLK/Q } ]

create_generated_clock -name {EMMC_CLK} -divide_by 1 -source [
get_pins { *u_MMCMaster/u_MMCClkGen/CLK/Q } ] -phase 0 [ get_ports {
CLK } ]
```

Where,

- `MSS_MMC_sb_0/CCC_0/CCC_INST/GL0` is the source clock for the generated clock. In the example design, PLL output 0 clock is used as a source clock for CoreMMC. If user uses different source clock and different instance names, the design constraint must be changed accordingly.
- `u_MMCMaster/u_MMCClkGen/CLK/Q` is the pin name of the derived clock.
- CLK is the output pad for EMMC clock. If user uses different pad name, the design constraint must be changed accordingly.

6. Add the output delay constraint for CMD and DAT pads as shown below.

```
set_output_delay 3 -clock {EMMC_CLK} [ get_ports {CMD DAT* } ]
```

Where,

- CMD is the inout pad for EMMC command. If user uses different pad name, the design constraint must be changed accordingly.
- DAT is the inout pad for EMMC data. It can be 1-bit, 4-bits or 8-bits wide. Wildcard(*) is used, as DAT width depends on configuration of MMC_DWIDTH parameter. If user uses different pad name, the design constraint must be changed accordingly.

9.2 Classic Constraints Flow

The procedure for adding the constraints is as follows for the classic constraints flow.

1. Right-click on **Create Constraints > Timing constraints** in the **Design Flow** window and click **create new constraint**. This creates a new SDC file. The design constraints including the clock source constraints can be entered in this blank SDC file.

2. As the example design uses PLL output 0 clock as a source clock, constraints need to be applied to specify the frequency of the clock source to the PLL. The output of the PLL is determined by the PLL configuration.
3. Add the following constraints to specify the frequency of the clock input to the PLL and generated clock from PLL.
4. In the example design, on-chip oscillator of 50 MHz clock is used as a PLL input clock and 104 MHz clock is generated from PLL.

```
create_clock -name { MSS_MMC_sb_0/FABOSC_0/I_RCOSC_25_50MHZ/CLKOUT } -
period 20.000 -waveform { 0.000 10.000 } [get_pins { MSS_MMC_sb_0
/FABOSC_0/I_RCOSC_25_50MHZ:CLKOUT }]

create_generated_clock -name { MSS_MMC_sb_0/CCC_0/CCC_INST/INST_CCC_IP:
GL0 } -divide_by 25 -multiply_by 52 -source{ MSS_MMC_sb_0/CCC_0
/CCC_INST/INST_CCC_IP:RCOSC_25_50MHZ } \

{ MSS_MMC_sb_0/CCC_0/CCC_INST/INST_CCC_IP:GL0 }
```

Where,

- `MSS_MMC_sb_0/FABOSC_0/I_RCOSC_25_50MHZ`: CLKOUT is the 50 MHz on-chip oscillator clock used as an input clock of PLL. If user uses different clock source, the design constraint must be change accordingly.
- `MSS_MMC_sb_0/CCC_0/CCC_INST/INST_CCC_IP:GL0` is the 104 MHz clock generated from PLL. If user uses different source clock and different instance name, the design constraint must be changed accordingly.

5. Add the generated clock constraints as shown below. Two generated clock constraints are used in this example. First generated clock is used to divide the source clock by 2 and second generated clock is used to route the generated clock in first step to the output CLK pad.

```
create_generated_clock -name {EMMC_CLK_DIV} -divide_by 2 -source [
get_pins {MSS_MMC_sb_0/CCC_0/CCC_INST/INST_CCC_IP:GL0} ] [ get_pins {
*u_MMCMaster/u_MMCClkGen/CLK:Q } ]

create_generated_clock -name {EMMC_CLK} -divide_by 1 -source [
get_pins { *u_MMCMaster/u_MMCClkGen/CLK:Q } ] -phase 0 [ get_ports {
CLK } ]
```

Where,

- `MSS_MMC_sb_0/CCC_0/CCC_INST/INST_CCC_IP:GL0` is the source clock for the generated clock. In the example design, PLL output 0 clock is used as a source clock for CoreMMC. If user uses different source clock and different instance name, the design constraint must be changed accordingly.
- `u_MMCMaster/u_MMCClkGen/CLK:Q` is the pin name of the derived clock.
- CLK is the output pad for EMMC clock. If user uses different pad name, the design constraint must be changed accordingly.

6. Add the output delay constraint for CMD and DAT pads as shown below.

```
set_output_delay 3 -clock {EMMC_CLK} [ get_ports {CMD DAT* } ]
```

Where,

- CMD is the inout pad for EMMC command. If user uses different pad name, the design constraint must be changed accordingly.
- DAT is the inout pad for EMMC data. It can be 1-bit, 4-bits or 8-bits wide. Wildcard(*) is used, as DAT width depends on configuration of MMC_DWIDTH parameter.

10 Reference Documents

The following table gives the list of documents referred in this document.

Table 68 • Reference Documents

Document ID	Document Name
[R1]	JEDEC eMMC 4.41 Standard Specification
[R2]	SanDisk eMMC 4.41 I/F Preliminary Datasheet
[R3]	AMBA®3 AHB-Lite Protocol

Microsemi Headquarters
One Enterprise, Aliso Viejo,
CA 92656 USA
Within the USA: +1 (800) 713-4113
Outside the USA: +1 (949) 380-6100
Sales: +1 (949) 380-6136
Fax: +1 (949) 215-4996
Email: sales.support@microsemi.com
www.microsemi.com

Microsemi, a wholly owned subsidiary of Microchip Technology Inc. (Nasdaq: MCHP), offers a comprehensive portfolio of semiconductor and system solutions for aerospace & defense, communications, data center and industrial markets. Products include high-performance and radiation-hardened analog mixed-signal integrated circuits, FPGAs, SoCs and ASICs; power management products; timing and synchronization devices and precise time solutions, setting the world's standard for time; voice processing devices; RF solutions; discrete components; enterprise storage and communication solutions; security technologies and scalable anti-tamper products; Ethernet solutions; Power-over-Ethernet ICs and midspans; as well as custom design capabilities and services. Microsemi is headquartered in Aliso Viejo, California, and has approximately 4,800 employees globally. Learn more at www microsemi.com.

50200510