CoreMMC v2.0

Handbook

Microsemi

Table of Contents

CoreMMC 3
Introduction 3
Key Features 3
Limitations 4
Core Version 4
Supported Interfaces 4
Utilization and Performance 5
Reference Documents 5

Design Description 7
Verilog or VHDL Parameters 7
I/O Signals 7
Register Map and Descriptions 9

Design Details 25
System Overview 25
Functional Description of the MMC Controller 25
Programming Example 29

Tool Flows 31
Licensing 31
SmartDesign 31
Simulation Flows 32
Synthesis in Libero IDE 33
Place-and-Route in Libero IDE 33

Ordering Information 35
Ordering Codes 35

Product Support 37
Customer Service 37
Customer Technical Support Center 37
Technical Support 37
Website 37
Contacting the Customer Technical Support Center 37
ITAR Technical Support 38

Introduction

CoreMMC provides a MultiMedia Card (MMC) controller to interface with external MMC flash cards and eMMC devices. The CoreMMC controller allows you to store theboot images and boot the processor from the removable flash card. You can also use the flash card to expand the on-board storage capacity for larger applications or store the user data. Other applications include interfacing with embedded MMC (eMMC) non-removable flash devices. The core is parameterized to allow user specifications of MMC clock as MMC data bus width and FIFO depth among others and is conformant to the JEDEC eMMC 4.41 Standard Specification.

CoreMMC consists of an AMBA high-performance bus lite (AHB-Lite) slave interface design 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.

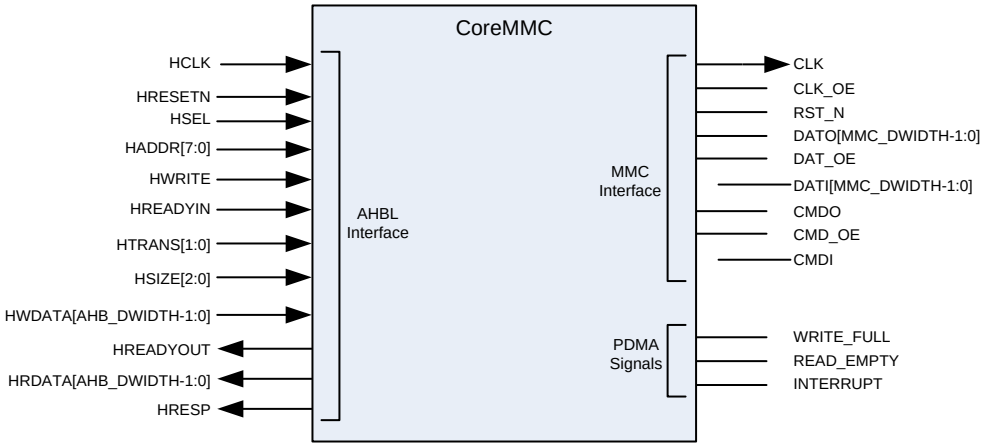


Figure 1 CoreMMC I/O Signal Diagram

Key Features

- Up to 52 MHz clock rate, making for 52 MBs throughput at 8-bit data width
- Configurable data bus widths
  - 1-/4-/8-bit
- Supports Block mode transfer:
  - WRITE_BLOCK
  - READ_BLOCK
- 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 flags interrupts
  - Write Block done and Read Block done interrupts
- Automatic Sleep mode for eMMC when clock pulled Low

Limitations

It does not support:

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

Core Version

This handbook supports CoreMMC version 2.0.

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, refer to I/O Signals section on page 10.

Utilization and Performance

All Microsemi® SmartFusion2 system-on-chip (SoC) field programmable gate array (FPGA) family and devices are supported. CoreMMC has been implemented in several devices of Microsemi using standard speed grades. Table 1 and Table 2 list a summary of various implementations data.

Table 1 Core MMC Device Utilization and Performance (Minium Configiration)

Family	Tiles			Utilization		Performance MHz
	Sequential	Combinatorial	Total	Device	Total %	
SmartFusion2	1270	807	2077	M2S050T	1.84	92
Note: Data in this table was achieved using the Verilog RTL with typical synthesis and layout settings. Top-level parameters or generics were set as: AHB_DWIDTH = 8, MMC_DWIDTH = 1, FIFO_DEPTH = 0, CoreMMC also uses 8 RAM64x18 blocks in this configuration.						

Table 2 CoreMMC Device Utilization and Performance (Maximum Configuration)

Family	Tiles			Utilization		Performance MHz
	Sequential	Combinatorial	Total	Device	Total %	
SmartFusion2	2537	1851	4388	M2S050T	3.9	92
Note: Data in this table was achieved using the Verilog RTL with typical synthesis and layout settings. Top-level parameters/generics were set as: AHB_DWIDTH = 32, MMC_DWIDTH = 3, FIFO_DEPTH = 3, CoreMMC also uses 32 RAM1Kx18 blocks in this configuration.						

Reference Documents

Table 3 shows the list of documents referred in this document.

Table 3 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
[R3]	A2F-DEV-KIT

Design Description

Verilog or VHDL Parameters

Table 4 describes the CoreMMC parameters (Verilog) or generics (VHDL) for configuring the RTL code. All parameters and generics are integer types.

Table 4 CoreMMC Parameters or Generics Descriptions

Name	Valid Range	Default	Description
FAMILY	19	19	Must be set to the required FPGA family: 19: SmartFusion2
MMC_DWIDTH	0, 2, 3	0	Serial data bus width, allowed to be 1-/4-/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 K 2: 16 K 3: 32 K

I/O Signals

Table 5 describes the port signals for the CoreMMC macro as shown in Figure 1.

Table 5 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.
CLK_OE	Out	Clock output enable. To be used by a slave MMC device to enter into Low power mode when no operation is in progress. Refer to the Jedec specifcation.
RST_N	Out	Hard reset for eMMC device. Active Low signal. Only operational for slave device if enabled within the slave device.
DATO[7:0]	Out	Data output for bi-directional bus in Push-pull mode.
DATI[7:0]	In	Data input from bi-directional bus in Push-pull mode.
DAT_OE	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 clock. All the AHB signals inside the block are clocked on the rising edge.
HRESETN	In	AHB 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

Name	Type	Description
		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[7:0]	In	AHB-Lite address. 7-bit address on the AHB-Lite interface.
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 hHigh 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)		
WRITE_FULL	Out	Indicates that the Write FIFO is full.
READ_EMPTY	Out	Indicates that the Read FIFO is empty.
INTERRUPT	Out	Interrupt output, asserted if any of the masked interrupt bits in the ISR/BISR are asserted.
Note: All signals are active High (logic 1) unless otherwise noted.		

Register Map and Descriptions

Register Summary

Table 6 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	This register indicate 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	0xC4	This register defines the version of core along with the parameters used to generate the core. Default values are shown based on default parameters.
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
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

Address	Register Name	Mnemonic	Type	Width (bits)	Reset Value	Description (8-bit)
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	8	0x0	Write Data register (8 bits)
0x28	Read Data Register	RDR	R	8	0x0	Read Data register (8 bits)
0x2C	Interrupt Mask Register	IMR	R/W	8	0x0	Indicate 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	Block Interrupt Mask register	BIMR	R/W	8	0x0	Indicate the bits in the BISR that can cause the interrupt to be asserted. When a bit is set to 1 in this register, if the corresponding bit in BISR is asserted, then the Interrupt line is asserted.
0x30	Interrupt Status register	ISR	R	8	0x0	Interrupt Status register
0x31	Block Interrupt Status register	BISR	R	8	0x0	Block Interrupt Status register
0x34	Interrupt Clear register	ICR	W	8	0x0	Interrupt Clear register
0x35	Block Interrupt Clear register	BICR	W	8	0x0	Block Interrupt Clear register
0x38	Control Register	CTRL	R/W	8	0x2C	Control register
0x39	Block Control and Status register	BCSR	R/W	8	0x0	Block Control and Status Register
0x3C	Response Time-out register	RSTOR	R/W	8	0x40	Response Timeout register
0x40	Data Time-out register	DATOR	R/W	32	0x4000	Data Timeout register
0x44	Block Length register	BLR	R/W	32	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

Note: Values shown in tables are in hexadecimal format; type designations: R = read only; W = write only; R/W = read/write

Status Register

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

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. Table 7 shows the CoreMMC Status Register.

Table 7 Status Register

HADDR[7:0]	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.

Table 8 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.
Note: All bits are active High unless otherwise indicated.			

CoreMMC Version Register

This register contains the version of the core and the parameters used to generate the instance of the core. Table 9 shows the CoreMMC Version Register.

Table 9 Version Register

HADDR[7:0]	Register Name	Type	Width	Reset Value	Description
0x01	Version	R/W	8	0xC4	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 10 Version Register Bits

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: 2-bit 10: 8-bit

Bit(s)	Type	Name	Description
			Note: This coding is different to MMC_DWIDTH codes.
1:0	R	rev	Revision of this instance of CoreMMC.
Note: All bits are active High unless otherwise indicated.			

Command Index Register

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

Table 11 Command Index Register

HADDR[7:0]	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 12 Command Index Register bits

Bit(s)	Type	Name	Description
7:6	R	-	Reserved
5:0	R/W	CMDX	Command Index (or command register 0)
Note: All bits are active High unless otherwise indicated.			

Command Argument1 Register

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

Table 13 Command Argument1 Register

HADDR[7:0]	Register Name	Type	Width	Reset Value	Description
0x08	Arg1	R/W	8	0x0	Command Argument1. Sets the first byte of the agrument for Command to be sent to the slave. Argument1, Argument2, Argument3, and Argument4 compose the full argument. All four bytes can be written together.

Command Argument2 Register

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

Table 14 Command Argument2 Register

HADDR[7:0]	Register Name	Type	Width	Reset Value	Description
0x09	Arg2	R/W	8	0x0	Command Argument2. The second byte of agrument for Command to be sent to the slave.

Command Argument3 Register

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

Table 15 Command Argument3 Register

HADDR[7:0]	Register Name	Type	Width	Reset Value	Description
0x0a	Arg3	R/W	8	0x0	Command Argument3. The third byte of agrument for Command to be sent to the slave.

Command Argument4 Register

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

Table 16 Command Arg4 Register

HADDR[7:0]	Register Name	Type	Width	Reset Value	Description
0x0b	Arg4	R/W	8	0x0	Command Argument4. The fourth byte of agrument 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 (sent as indicated by scsi bit or timed-out as indicated by ssbi in ISR). It can cause the incorrect communications with the slave MMC.

Response Register0

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

Table 17 Response Register0

HADDR[7:0]	Register Name	Type	Width	Reset Value	Description
0x10	RR0	R	8	0x0	Response Register0. First byte of response from the MMC device.

Response Register1

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

Table 18 Response Register1

HADDR[7:0]	Register Name	Type	Width	Reset Value	Description
0x14	RR1	R	8	0x0	Response Register1. Third byte of response from the MMC device.

Response Register2

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

Table 19 Response Register2

HADDR[7:0]	Register Name	Type	Width	Reset Value	Description
0x15	RR2	R	8	0x0	Response Register2. Third byte of response from the MMC device.

Response Register3

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

Table 20 Response Register3

HADDR[7:0]	Register Name	Type	Width	Reset Value	Description
0x16	RR3	R	8	0x0	Response Register3. Fourth byte of response from the MMC device.

Response Register4

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

Table 21 Response Register4

HADDR[7:0]	Register Name	Type	Width	Reset Value	Description
0x17	RR4	R	8	0x0	Response Register4. Fifth byte of response from the MMC device.

Response Register5

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

Table 22 Response Register5

HADDR[7:0]	Register Name	Type	Width	Reset Value	Description
0x18	RR5	R	8	0x0	Response Register5. Sixth byte of response from the MMC device.

Response Register6

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

Table 23 Response Register6

HADDR[7:0]	Register Name	Type	Width	Reset Value	Description
0x19	RR6	R	8	0x0	Response Register6. Seventh byte of response from the MMC device.

Response Register7

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

Table 24 Response Register7

HADDR[7:0]	Register Name	Type	Width	Reset Value	Description
0x1A	RR7	R	8	0x0	Response Register7. Eighth byte of response from the MMC device.

Response Register8

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

Table 25 Response Register8

HADDR[7:0]	Register Name	Type	Width	Reset Value	Description
0x1B	RR8	R	8	0x0	Response Register8. Ninth byte of response from the MMC device.

Response Register9

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

Table 26 Response Register9

HADDR[7:0]	Register Name	Type	Width	Reset Value	Description
0x1C	RR9	R	8	0x0	Response Register9. Tenth byte of response from the MMC device.

Response Register10

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

Table 27 Response Register10

HADDR[7:0]	Register Name	Type	Width	Reset Value	Description
0x1D	RR10	R	8	0x0	Response Register10. Eleventh byte of response from the MMC device.

Response Register11

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

Table 28 Response Register11

HADDR[7:0]	Register Name	Type	Width	Reset Value	Description
0x1E	RR11	R	8	0x0	Response Register11. Twelfth byte of response from the MMC device.

Response Register12

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

Table 29 Response Register12

HADDR[7:0]	Register Name	Type	Width	Reset Value	Description
0x1F	RR12	R	8	0x0	Response Register12. Thirteenth byte of response from the MMC device.

Response Register13

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

Table 30 Response Register13

HADDR[7:0]	Register Name	Type	Width	Reset Value	Description
0x20	RR13	R	8	0x0	Response Register13. Fourteenth byte of response from the MMC device.

Response Register14

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

Table 31 Response Register14

HADDR[7:0]	Register Name	Type	Width	Reset Value	Description
0x21	RR14	R	8	0x0	Response Register14. Fifteenth byte of response from the MMC device.

Response Register15

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

Table 32 Response Register15

HADDR[7:0]	Register Name	Type	Width	Reset Value	Description
0x22	RR15	R	8	0x0	Response Register15. Sixteenth byte of response from the MMC device.

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 8bits. Depth of the FIFO is defined by FIFO_DEPTH parameter when the instance of the core is created.

Table 33 Write Data Register

HADDR[7:0]	Register Name	Type	Width	Reset Value	Description
0x24	WDR	W	8	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.

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 8 bits. Depth of the FIFO is defined by FIFO_DEPTH parameter when the instance of the core is created.

Table 34 Read Data Register

HADDR[7:0]	Register Name	Type	Width	Reset Value	Description
0x28	RDR	R	8	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.

Interrupt Mask Register

The Interrupt Mask register is used to determine the interrupt status bits in ISR that trigger an interrupt. Table 35 shows the Interrupt Mask register.

Table 35 Interrupt Mask Register

HADDR[7:0]	Register Name	Type	Width	Reset Value	Description
0x2C	IMR	R/W	8	0x0	This register determines the status bits that trigger an interrupt.

Table 36 Interrupt Mask Register

Bit(s)	Type	Name	Description
7	R/W	muer	When suer is set, assert an interrupt.
6	R/W	msbi	When ssbi is set, assert an interrupt.
5	R/W	mtbi	When stbi is set, assert an interrupt.
4	R/W	mtxi	When stxi is set, assert an interrupt.
3	R/W	mrri	When srri is set, assert an interrupt.
2	R/W	mcsi	When scsi is set, assert an interrupt.
1	R/W	mboi	When sboi is set, assert an interrupt.
0	R/W	mbui	When sbui is set, assert an interrupt.
Note: All bits are active High unless otherwise indicated.			

Block Interrupt Mask Register

The Block Interrupt Mask Register is used to determine the status bits in BISR that trigger an interrupt. Table 37 shows the Block Interrupt Mask register.

Table 37 Block Interrupt Mask Register

HADDR[7:0]	Register Name	Type	Width	Reset Value	Description
0x2D	BIMR	R/W	8	0x0	This register determines the block status bits that trigger an interrupt.

Table 38 Block Interrupt Mask Register

Bit(s)	Type	Name	Description
7	R	-	Reserved
6	R	-	Reserved
5	R/W	mcrcsterr	When scrcsterr is set, assert an interrupt
4	R/W	mbrsterr	When sbrsterr is set, assert an interrupt
3	R/W	mdatto	When sdatto is set, assert an interrupt
2	R/W	mbcrcerr	When sbcrcerr is set, assert an interrupt
1	R/W	mbrdone	When sbrdone is set, assert an interrupt
0	R/W	mbwdone	When sbwdone is set, assert an interrupt
Note: All bits are active High unless otherwise indicated.			

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 IMR. Table 39 describes the Interrupt Status register.

Table 39 Interrupt Status Register

HADDR[7:0]	Register Name	Type	Width	Reset Value	Description
0x30	ISR	R	8	0x0	This register provides status and error flags of transactions.

Table 40 Interrupt Status Register

Bit(s)	Type	Name	Description
7	R	suer	User error is detected and encountered. Write FIFO overrun or Read FIFO underrun error.
6	R	ssbi	Start bit error is detected or time-out error while waiting for a response.
5	R	stbi	Stop bit error is detected
4	R	stxi	Transmit bit error is detected
3	R	srri	Response received
2	R	scsi	Command sent
1	R	sboi	Buffer overflow occurred. Read FIFO was full when more data attempted to be written into it during Block Read.
0	R	sbui	Underrun occurred. Write FIFO was empty when more data attempted to be sent during Block Write.
Note: All bits are active High unless otherwise indicated.			

Block Interrupt Status Register

The Block Interrupt Status Register provides the current status and error bits for block reads or writes transactions performed. These bits are set regardless of the BIMR. Table 41 describes the Block Interrupt status register.

Table 41 Block Interrupt Status Register

HADDR[7:0]	Register Name	Type	Width	Reset Value	Description
0x31	BISR	R	8	0x0	This register provides the status and error bits for block read or write transactions.

Table 42

Table 43 Block Interrupt Status Register

Bit(s)	Type	Name	Description
7	R	-	Reserved
6	R	-	Reserved
5	R	scrcsterr	CRC Response Error. Set when no CRC Status received within period defined in DATATO register.
4	R	sbrsterr	Block Read Stop Error. Set when valid stop bit not detected on all active DATI lines.
3	R	sdatto	Block Read data time-out. Set when no incoming start-bit found on DAT for period defined in DATATO register.
2	R	sbcrcerr	Block Read or Write encountered CRC Error.
1	R	sbrdone	Block Read Done
0	R	sbwdone	Block Write Done
Note: All bits are active High unless otherwise indicated.			

Interrupt Clear Register

The Interrupt Clear Register is a write-only register used to clear (individually) the ISR bits described Table 36. This can clear an interrupt, if the associated bit is set in the IMR. Table 43 describes the Interrupt Clear register.

Table 44 Interrupt Clear Register

HADDR[7:0]	Register Name	Type	Width	Reset Value	Description
0x34	ICR	W	8	0x0	This register clears (individually) the corresponding bit in the ISR bits described in Table 36.

Table 45 Interrupt Clear Register

Bit(s)	Type	Name	Description
7	W	cuer	Clear suser bit.
6	W	csbi	Clear ssbi bit
5	W	ctbi	Clear stbi bit
4	W	ctxi	Clear stxi bit
3	W	crri	Clear srri bit
2	W	ccsi	Clear scsi bit
1	W	cboi	Clear sboi bit
0	W	cbui	Clear sbui bit
Note: All bits are active High unless otherwise indicated.			

Block Interrupt Clear Register

The Interrupt Clear Register is a write-only register used to clear (individually) the BISR bits described in Table 38. This can clear an interrupt if the associated bit is set in the BIMR. Table 45 describes the Block Interrupt Clear register.

Table 46 Block Interrupt Clear Register

HADDR[7:0]	Register Name	Type	Width	Reset Value	Description
0x35	BICR	W	8	0x0	Block Interrupt Clear Register. This register clears (individually) the corresponding bit in the BISR described in Table 38.

Table 47 Block Interrupt Clear Register

Bit(s)	Type	Name	Description
7	-	-	Reserved
6	-	-	Reserved
5	W	ccrcsterr	Clear scrcsterr bit
4	W	cbrsterr	Clear sbrsterr bit
3	W	cdatto	Clear sdatto bit
2	W	cbcrcerr	Clear sbcrcerr bit
1	W	cbrdone	Clear sbrdone bit
0	W	cbwdone	Clear sbwdone bit
Note: All bits are active High unless otherwise indicated.			

Control Register

The Control Register provides the control bits to define the operation of the core and to drive hardware pins to the slave device. Table 47 describes the Control register.

Table 48 Control Register

HADDR[7:0]	Register Name	Type	Width	Reset Value	Description
0x38	CTRL	R/W	8	0x2C	This register provides the control bits for the operation of the core.

Table 49 Control Register

Bit(s)	Type	Name	Description
7	R	Busy	Slave device is indicating that it is busy by asserting DAT[0] low.
6:5	R	-	Reserved.
4	R/W	cmdFrcLow	Force CMD line to 0 (low).
3	R	midle	MMC Idle. When set to 1, it indicates that the core is in Idle state (not sending command, data, or expecting response/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 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.

Note: All bits are active High unless otherwise indicated.

Block Control and Status Register

The Block Control and Status Register provide control of block reading and writing transactions and associated status bits from these operations. Table 49 describes the Block Control and Status register.

Table 50 Block Control and Status Register

HADDR[7:0]	Register Name	Type	Width	Reset Value	Description
0x39	BCSR	R/W	8	0x0	This register provides control and status bits for block reads and writes.

Table 51 Block Control and Status Register

Bit(s)	Type	Name	Description
7:4	R	bwst	Block Write Status – CRC Status bits and End-Frame bit. Updated when bdone set for block write. (Good CRC Status results in bwst=0101)
3	R	bcrcerr	Block CRC Error. Set when bdone is asserted, if CRC error detected. It means write or read in error. Cleared when writing 1 to cbcrcerr in BICR.
2	R	bdone	Block Done. Cleared when Block Write or Block Read Start is set (bwstrt or brstrt). Set by hardware when Block Write or Read is completed and bcrcerr are updated.
1	RW	brstrt	Block Read Start. Initiates the read of a block of the length defined in Block Length register. Only held set for 1 HCLK period. Should not be written until bdone is set.
0	RW	bwstrt	Block Write Start. Initiates the write block of the length defined in Block Length register. Only held set for 1 HCLK period. Should not be written until bdone is set.
Note: All bits are active High unless otherwise indicated.			

Response Timeout Register

This register provides 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 ssbi in ISR (that is, a time-out looks like a start-bit error).

Table 52 Response Timeout Register

HADDR[7:0]	Register Name	Type	Width	Reset Value	Description
0x3C	RSPTO	R/W	8	0x40	Length of time (in HCLK ticks) that core waits for a repsonse from the slave device.

Data Timeout Register

This register provides the number of clock ticks of HCLK, that the core waits for a block read to start from the slave device after it has issued a command before timing out and setting sdatto in ISR.

Table 53 Data Timeout Register

HADDR[7:0]	Register Name	Type	Width	Reset Value	Description
0x40	DATATO	R/W	32	0x40	Length of time (in HCLK ticks) that core will waits for a block read to start.

Block Length Register

This register defines the length in bytes of block read and block write transfers. This register must be defined before initiating block reads or writes and is for CoreMMC. It does not set block size for the slave MMC device which must be done through a command and response sequence.

Table 54 Block Length Register

HADDR[7:0]	Register Name	Type	Width	Reset Value	Description
0x44	BlockLen	R/W	32	0x200	Sets the length in bytes of transfers for block reads and block writes.

Data Control Register

This register defines the width for DAT that is to be used for block read and block write transfers. This register must be defined by the user before initiating block reads or writes. This is **separate** from setting MMC_DWIDTH in core instantation. The core may have been created with 8-bit MMC DAT width but operationally may be set to work with 1-bit DAT bus.

Table 55 Data Control Register

HADDR[7:0]	Register Name	Type	Width	Reset Value	Description
0x48	DCTRL	RW	8	0x00	Data Control – defines size of active bits of DAT.

Table 56 Data Control Register

Bit(s)	Type	Name	Description
7:2	R	-	Reserved
1:0	RW	Dsize	DAT Size – number of dat bits active. User is not able to select a size greater than in created core. dsize coding is: 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
Note: All bits are active High unless otherwise indicated.			

Clock Register

This register defines a period for the CLK pin (MMC clock). The value of CLKHP defines the half clock period for the CLK. A value of 0 periods a CLK of 1/2 HCLK, a value of 1 provides a value of 1/4 HCLK, a value of 2 provides a value of 1/6 HCK.

This register has the value 0x3F by default. A 400 KHz clock from a 50 MHz HCLK is required for initial frequency for MMC device. If HCLK is different to 50 MHz this register should be modified to provide ~400 KHz clock as per JEDEC specification for initial MMC device configuration. It is need to be defined before initiating block reads or writes. This is separate from setting MMC_DWIDTH in core instantiation. The core may have been created with 8-bit MMC DAT width but operationally may be set to work with 1-bit DAT bus.

Table 57 Clock Register

HADDR[7:0]	Register Name	Type	Width	Reset Value	Description
0x4C	CLKR	RW	8	0x3f	Clock Register – defines half period of CLK signal.

Table 58 Clock Register

Bit(s)	Type	Name	Description
7	R	-	Reserved
6:0	RW	CLKHP	CLK Half Period – defines the half period of CLK. Frequency of CLK is HCLK divided by twice (CLKHP+1). Examples of CLKHP value to HCLK: 00 – 1/2 HCLK 01 – 1/4 HCLK 02 – 1/6 HCLK 03 – 1/8 HCLK 04 – 1/10 HCLK .. 07 – 1/16 HCLK .. 3F – 1/128 HCLK
Note: All bits are active High unless otherwise indicated.			

Design Details

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. Figure 2 shows the top-level block diagram.

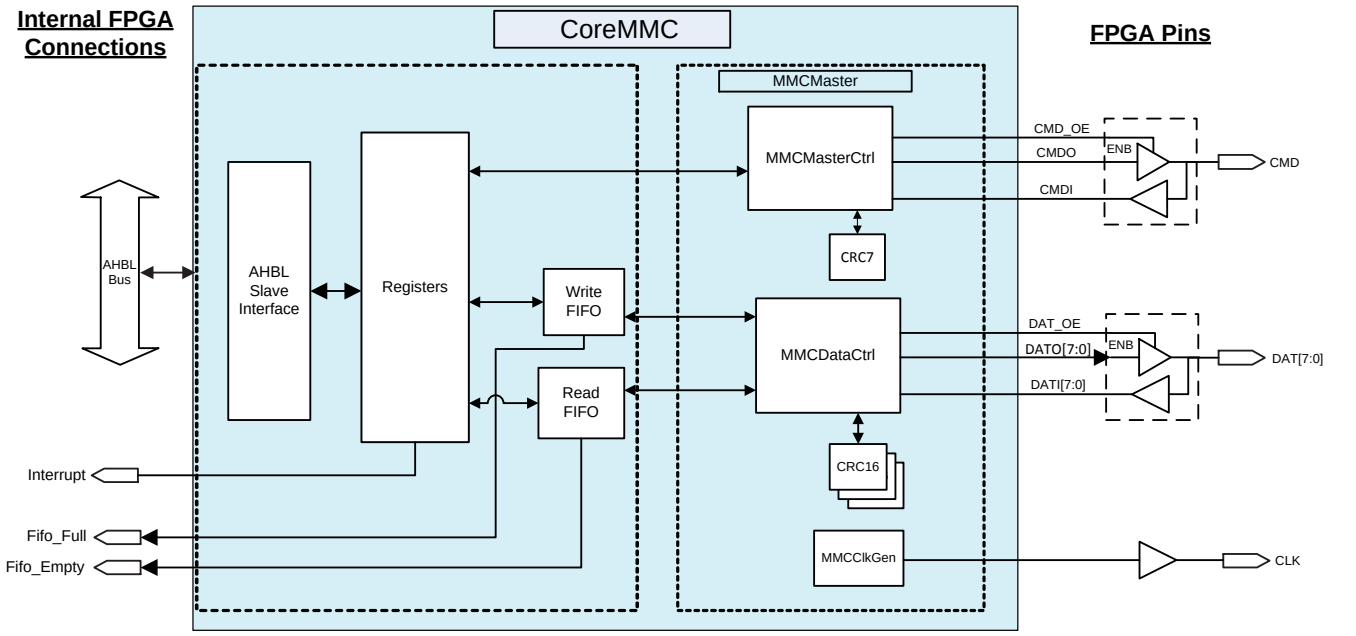


Figure 2 CoreMMC Block Diagram

Functional Description of the MMC Controller

This section describes the components of the CoreMMC controller and how the controller operates. There are the following primary functional blocks:

- 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 a 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 access from 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 (depends 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 (depends 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 specificaiton [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].

MMC Protocol

Each MMC bus operation consist 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 at the block length * 8 (for single data bus width), block length * 2 (for 4-bit data bus width), or block length (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 is performed using commands and responses as defined in the Jedec standard [R1]. This communication uses the command and response registers in the core.

Operating Modes:

There are five operation modes as 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_RCA 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 an RCA 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.

Command Registers

These registers store the command data to be transmitted from CoreMMC to the slave device and send 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). Figure 3 shows the full MMC command.

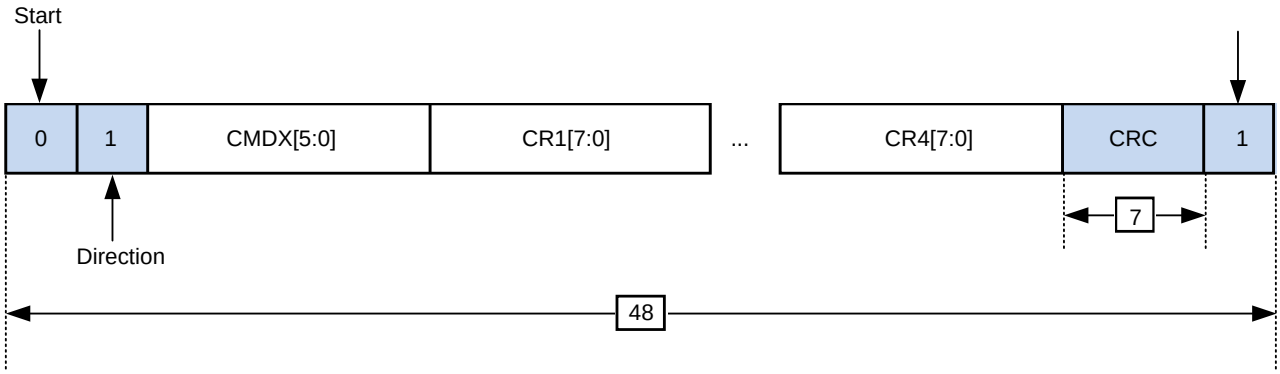


Figure 3 Command Register use in MMC Command Token

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.

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 clear when a command is initiated and are set when rri is set. The response registers maintain their values until another command is initiated.

Multiple Block Writes or Reads

The host can send multiple blocks with one Command-Response (CMD25). The host initially must fill the FIFO with the required number of blocks to write (to avoid an underrun situation) and then issue CMD25 through the Command registers. The host then set bwst bit for a block to be sent. When the sbwdone bit is asserted in BISR register, the host confirms no errors have occurred on that block write before sending the next block by setting bwst again. This way several blocks can be sent with one command-response transaction.

Similar for Multiple Block Reads, the host must ensure that the number of bytes to be received from the slave devices can be held in Read FIFO (to ensure no overrun on writes of data into FIFO from MMC bus).

Figure 4 illustrates an example of a multiple block read transaction and Figure 5 illustrates an example of a multiple block write transaction. The clock is representative only and does not show exact number of clock cycles for the full transaction.

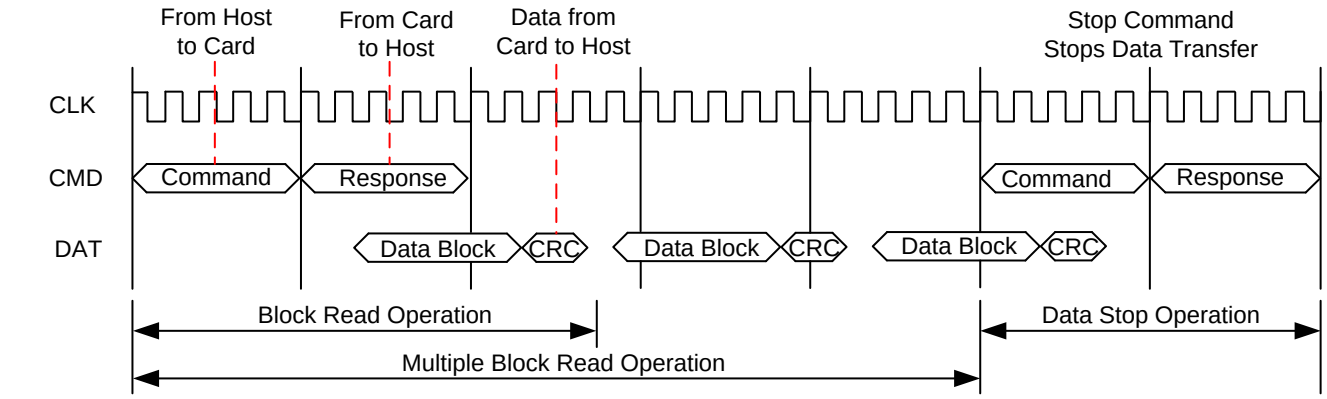


Figure 4 Multiple Block Read Operation

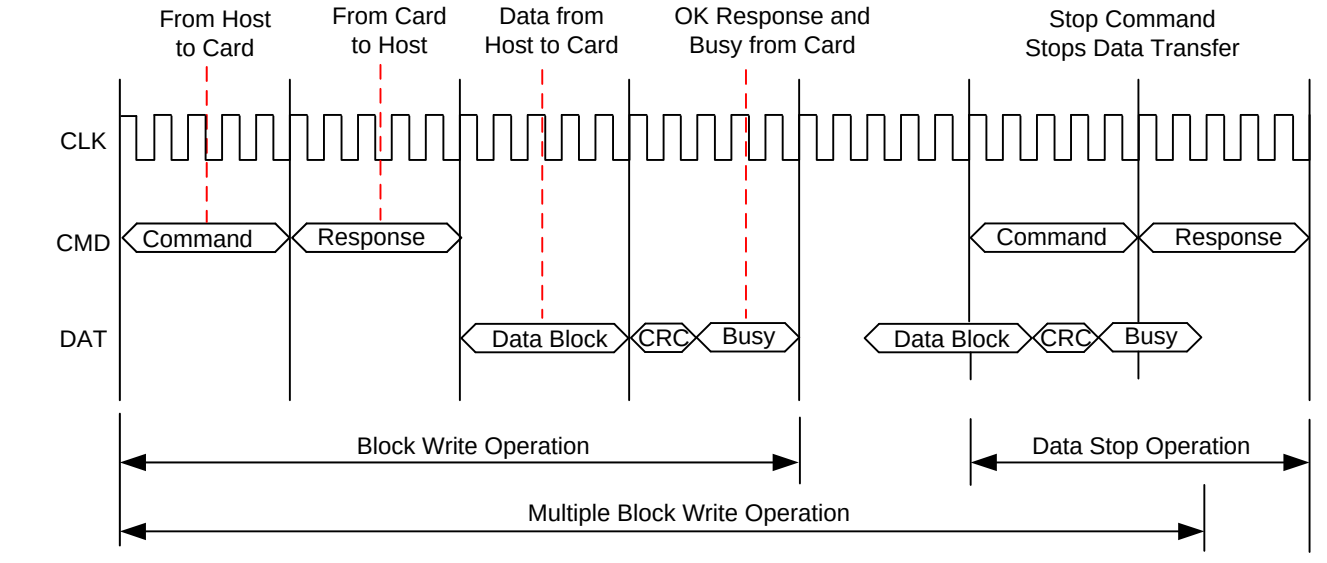


Figure 5 Multiple Block Write Operation

AHB-Lite Interface Timing

Figure 6 and Figure 7 depict typical write cycle and read cycle timing relationships relative to the system clock, HCLK.

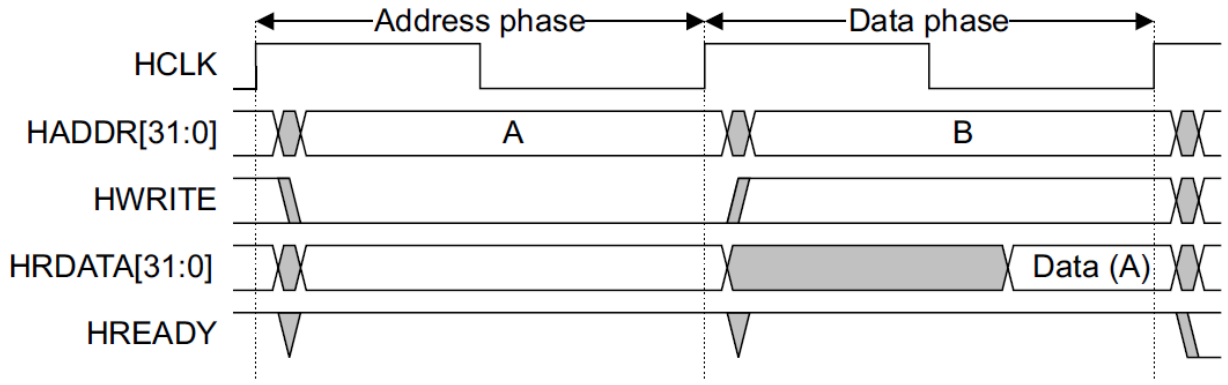


Figure 6 AHB-Lite Read Transfer

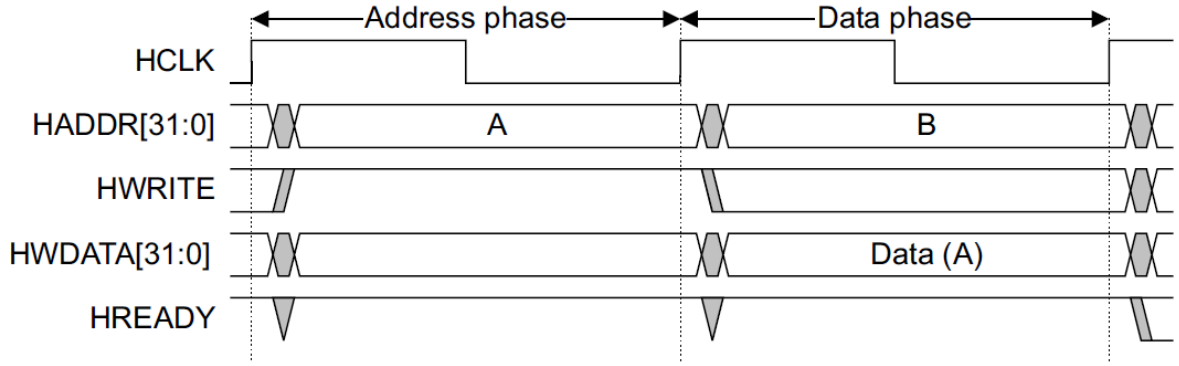


Figure 7 AHB-Lite Write Transfer

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 sdin5c2-8g device on the Microsemi development board A2F-DEV-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. The purpose is to provide 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.

1. CMD0 0x0: // move to idle state.
2. CMD1 0x40_00_00_80: // read OCR
3. 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.
4. CMD2 0xFF_FF_FF_FF: // read CID
   - Response R0-R15 - 45, 1, 0, 53, 45, 4d, 30, 38, 47. 90, 52, E7, 9B, 6, BF, 1
5. CMD3 0x0001_FFFF: // set RCA – Relative Card Address

- Response CMD3, Card Staus[31:0] - 0x00_00_05_00; // no errors, in Indent state

6. CMD9 0x0001_FFFF: // Send_CSD – get card specific data
   - Response R0-15 - D0, 0F, 00, 32, 0F, 59, 03, FF, FF, FF, FF, FF, 92, 40, 40, 11
     Vers code in EXT_CSD, Version 4.1-4.2-4.3, 10ms,
7. CMD7 0x0001_FFFF: // Go to transfer state
   - Response R0-R4 - 7, 00, 00, 07, 00
8. CMD16 0x4: // Set_BlockLen - setting to 4 bytes
   - Response R0-R4 - 0x10, 00,00,09, 00
9. 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)
10. CMD16 0x200: // Set_BlockLen – 512 bytes
    - Response R0-R4 - 0x10, 00,00,09, 00
11. 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 – ie all most support 8-bit DAT bus.
12. CMD13 0x0001_fffe: // Send_Status from card 1
    - Response R0-R4 - 0xD, 00,00,09, 00 - now bit[8] in status indicates **ready**
13. CMD24 0x0:L: // Write_Single_Block, sector 0x0
    - Response R0-R4 – 0x18, 00, 00, 09, 00 - no error – in Trans state
14. 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.

Tool Flows

Licensing

No licence is required for the use of this core.

Obfuscated

Complete unobfuscated RTL code is provided for the core, allowing the core to be instantiated with SmartDesign. Simulation, Synthesis, and Layout can be performed within Libero® Integrated Design Environment (IDE).

RTL

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

SmartDesign

CoreMMC is pre-installed in the SmartDesign IP Deployment design environment. Figure 8 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*.

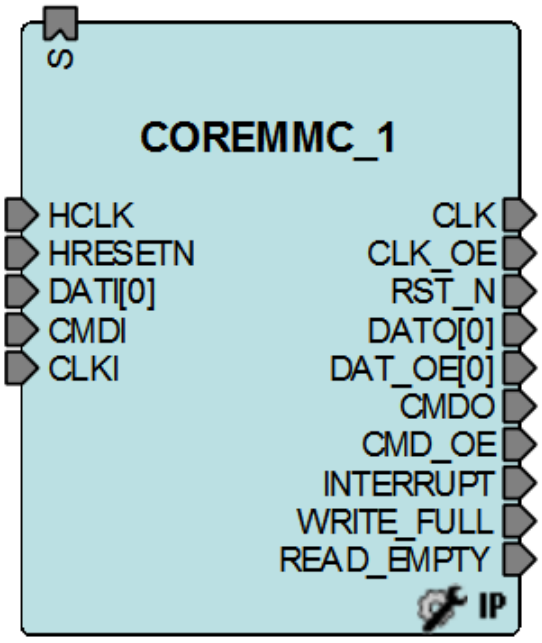


Figure 8 CoreMMC Full I/O View

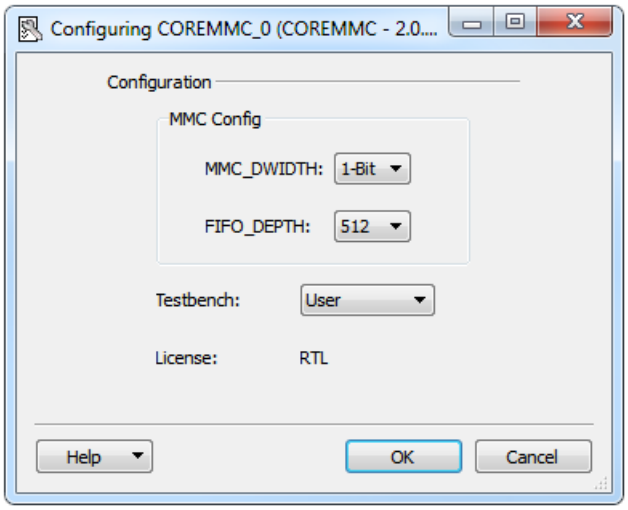


Figure 9 CoreMMC SmartDesign Configuration with Callouts to Associated Parameters

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 IDE project, it installs the user testbench files.

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

User Testbench

Figure 10 shows the simulation testbench and it includes the instantiation of the CoreMMC macro and a SlaveMMC block. The SlaveMMC block is a primitive model of a MMC slave device with basic functionality.

The user tests are executed in tasks in the testbench. These can be edited. The core tests perform checks on the default values of all registers in the memory map and then programs the CoreMMC to perform a Block Write and a Block Read.

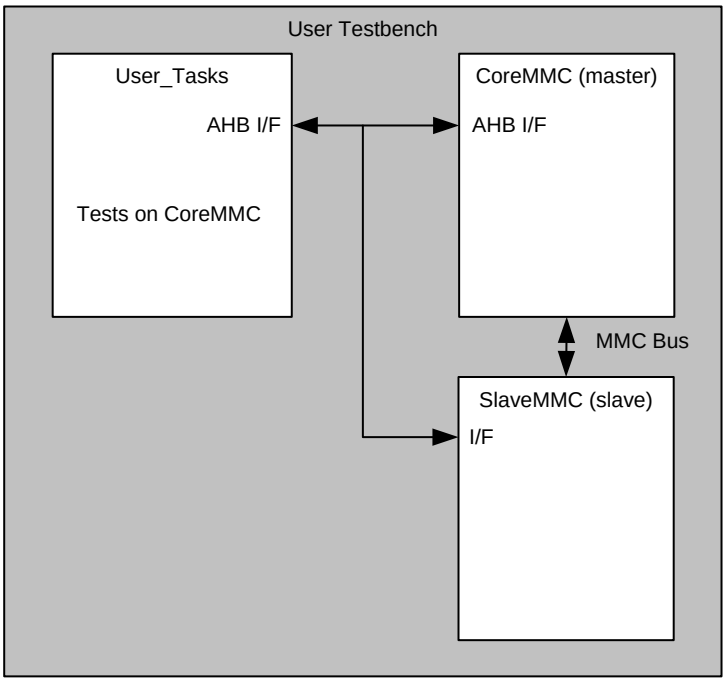


Figure 10 CoreMMC User Testbench

Synthesis in Libero IDE

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

1. Click **Synthesis** in the Libero IDE 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.

Place-and-Route in Libero IDE

After setting the design route appropriately for your design, and running Synthesis, click **Layout** in the Libero IDE software to invoke Designer. CoreMMC requires no special place-and-route settings.

Ordering Information

Ordering Codes

CoreMMC can be ordered through your local Microsemi sales representative. It should be ordered using the following number scheme: CoreMMC -XX, where XX is listed in Table 58.

Table 59 Ordering Codes

XX	Description
RM	RTL for RTL source — multiple-use license

Note: CortexM1-OM is free with a Libero IDE license.

Product Support

Microsemi SoC Products Group backs its products with various support services, including Customer Service, Customer Technical Support Center, a website, electronic mail, and worldwide sales offices. This appendix contains information about contacting Microsemi SoC Products Group and using these support services.

Customer Service

Contact Customer Service for non-technical product support, such as product pricing, product upgrades, update information, order status, and authorization.

From North America, call **800.262.1060**
From the rest of the world, call **650.318.4460**
Fax, from anywhere in the world **650. 318.8044**

Customer Technical Support Center

Microsemi SoC Products Group staffs its Customer Technical Support Center with highly skilled engineers who can help answer your hardware, software, and design questions about Microsemi SoC Products. The Customer Technical Support Center spends a great deal of time creating application notes, answers to common design cycle questions, documentation of known issues and various FAQs. So, before you contact us, please visit our online resources. It is very likely we have already answered your questions.

Technical Support

Visit the Microsemi SoC Products Group Customer Support website for more information and support (http://www.microsemi.com/soc/support/search/default.aspx). Many answers available on the searchable web resource include diagrams, illustrations, and links to other resources on website.

Website

You can browse a variety of technical and non-technical information on the Microsemi SoC Products Group home page, at http://www.microsemi.com/soc/.

Contacting the Customer Technical Support Center

Highly skilled engineers staff the Technical Support Center. The Technical Support Center can be contacted by email or through the Microsemi SoC Products Group website.

Email

You can communicate your technical questions to our email address and receive answers back by email, fax, or phone. Also, if you have design problems, you can email your design files to receive assistance. We constantly monitor the email account throughout the day. When sending your request to us, please be sure to include your full name, company name, and your contact information for efficient processing of your request.

The technical support email address is soc_tech@microsemi.com.

My Cases

Microsemi SoC Products Group customers may submit and track technical cases online by going to My Cases.

Outside the U.S.

Customers needing assistance outside the US time zones can either contact technical support via email (soc_tech@microsemi.com) or contact a local sales office. Sales office listings can be found at www.microsemi.com/soc/company/contact/default.aspx.

ITAR Technical Support

For technical support on RH and RT FPGAs that are regulated by International Traffic in Arms Regulations (ITAR), contact us via soc_tech_itar@microsemi.com. Alternatively, within My Cases, select **Yes** in the ITAR drop-down list. For a complete list of ITAR-regulated Microsemi FPGAs, visit the ITAR web page.



Microsemi Corporate Headquarters
One Enterprise, Aliso Viejo CA 92656 USA
Within the USA: +1 (949) 380-6100
Sales: +1 (949) 380-6136
Fax: +1 (949) 215-4996

Microsemi Corporation (NASDAQ: MSCC) offers a comprehensive portfolio of semiconductor solutions for: aerospace, defense and security; enterprise and communications; and industrial and alternative energy markets. Products include high-performance, high-reliability analog and RF devices, mixed signal and RF integrated circuits, customizable SoCs, FPGAs, and complete subsystems. Microsemi is headquartered in Aliso Viejo, Calif. Learn more at **www.microsemi.com**.

© 2013 Microsemi Corporation. All rights reserved. Microsemi and the Microsemi logo are trademarks of Microsemi Corporation. All other trademarks and service marks are the property of their respective owners.

50200510-0/10.13