

## 3 Memory

### 3.1 RAW NAND Flash Controller (NDFC)

The NDFC is the NAND flash controller which supports all NAND flash memory available in the market. New types of flash can be supported by software re-configuration.

The NDFC has a built-in on-the-fly error correction code (ECC) feature with BCH algorithm to enhance the reliability. It can detect and correct up to 80 bits' error per 1024 bytes' data. With the on-chip BCH code ECC circuit, the CPU is freed for other tasks. You can disable the ECC feature by software.

The NDFC supports transferring data via the DMA or CPU memory-mapped IO. It provides automatic timing control for reading or writing external Flash and maintains the proper relativity for CLE, CE#, and ALE control signal lines. There are three modes for serial read access: the mode0 is for conventional serial access, the mode1 is for EDO type, and the mode2 is for extension EDO type. The NDFC can monitor the status of the R/B# signal line.

Block management and wear leveling management are implemented in software.

The NDFC has the following features:

- Supports all SLC/MLC flash and EF-NAND memory available in the market
- Supports configuring randomize seed by software
- Software configuration method for various systems and memory types
- Up to 8-bit data bus width
- Supports 2CE/2RB
- Supports 1024, 2048, 4096, 8192, 16384, and 32768 bytes' size per page
- Conventional and EDO serial access method for serial reading Flash
- 80 bits/1 KB on-the-fly BCH code ECC check and error correction
- Output bits' number information about the corrected errors
- ECC automatic disable function for all 0xff data
- NDFC status information is reported by its registers, and interrupt is supported
- One command FIFO
- Two 256x32-bit RAM for Pipeline Procession
- Supports SDR, ONFI DDR1.0, Toggle DDR1.0, ONFI DDR2.0, and Toggle DDR2.0 RAW NAND FLASH
- Maximum IO rate of 50 MHz in SDR mode, 100 MHz in DDR1.0 and 150MHz in DDR2.0 mode
- Self-debug for NDFC debug

## 3.2 SDRAM controller (DRAMC)

The DRAMC has the following features:

- 32-bit DDR3/DDR3L/DDR4/LPDDR3/LPDDR4/LPDDR4X interface
- Memory capacity up to 4GB
- Clock frequency up to 1066 MHz for DDR3, DDR3L, and LPDDR3
- Clock frequency up to 1200 MHz for DDR4, LPDDR4, and LPDDR4x
- 17 address lines and three bank address lines per channel
- Generate initialization and refresh sequences automatically
- Runtime-configurable parameters setting for application flexibility
- Programmable priority of transferring through multiple ports
- Supports random reading or writing
- Supports Dynamic Frequency Setting(DFS) via hardware
- Supports SSC

## 3.3 SD/MMC Host Controller (SMHC)

### 3.3.1 Overview

The SMHC controls the read/write operations on the secure digital (SD) cards, multimedia cards (MMC), and various extended devices that is based on the secure digital input/output (SDIO) protocol. The processor provides three SMHC interfaces for controlling the SD cards, MMCs, and SDIO devices.

The SMHC has the following features:

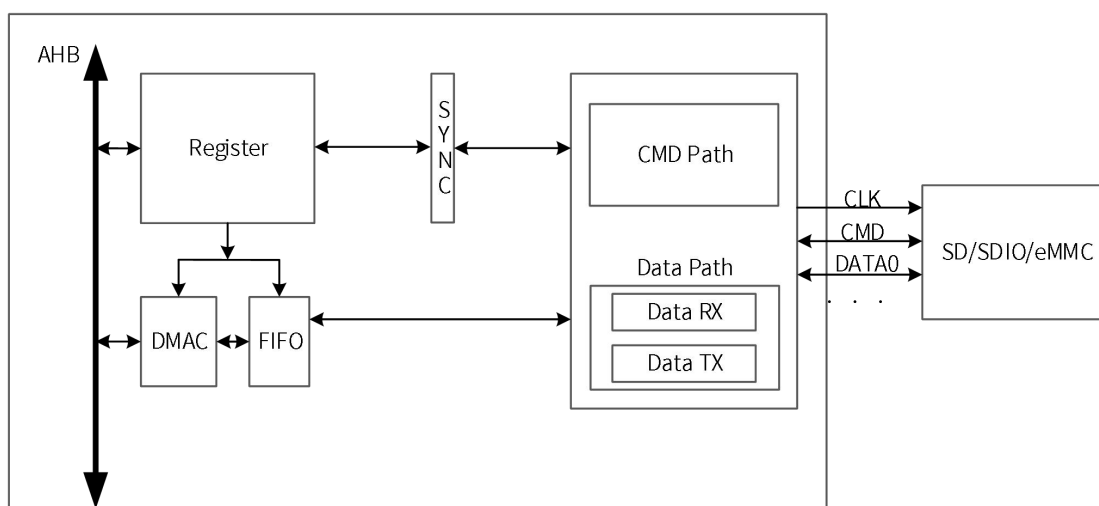
- Three SD/MMC host controller (SMHC) interfaces
  - SMHC0, compliant with the protocol Secure Digital Memory (SD3.0)
  - SMHC1, compliant with the protocol Secure Digital I/O (SDIO3.0)
  - SMHC2, compliant with the protocol Multimedia Card (eMMC5.1)
  - Supports one SD (Version 1.0 to 3.0) or MMC (Version 3.3 to 5.1)
- The SMHC0 and the SMHC1 support the following:
  - 1-bit or 4-bit data width
  - Maximum performance:
    - SDR mode 200 MHz@1.8 V IO pad
    - DDR mode 50 MHz@1.8 V IO pad
    - SDR mode 50 MHz@3.3 V IO pad
- The SMHC2 supports the following:
  - 1-bit, 4-bit, or 8-bit data width
  - Supports HS400 mode and HS200 mode
  - Maximum performance
    - SDR mode 200MHz@1.8V IO pad
    - DDR mode 200MHz@1.8V IO pad
    - SDR mode 50MHz@3.3V IO pad
    - DDR mode 50MHz@3.3V IO pad
- Support block size of 1 to 65535 bytes
- Support hardware CRC generation and error detection
- Supports eMMC boot operation and alternative boot operation
- Supports command queue for eMMC V5.1 device
- Supports serial CMDQ mode for SMHC0/2
- Supports host pull-up control

- Supports Command Completion signals and interrupts to host processor, and Command Completion signal disable feature
- Programmable baud rate
- Descriptor-based internal DMA controller
- Internal time-multiplexing 1 KB FIFO for SMHC0/2 transmitting and receiving
- Internal time-multiplexing 4 KB FIFO for SMHC1 transmitting and receiving

### 3.3.2 Block Diagram

The following figure shows a block diagram of the SMHC.

**Figure 3-1 SMHC Block Diagram**



SMHC contains the following sub-blocks:

**Table 3-1 SMHC Sub-blocks**

Sub-block	Description
Register	Used to configure the control signal for reading or writing the SD/SDIO/eMMC.
DMAC	The DMA controller that controls the data transfer between the memory and SMHC.
FIFO	A buffer for the data stream between the memory and the SMHC asynchronous clock domain.
SYNC	Synchronizes the signals from the AHB clock domain to the SMHC clock domain.
CMD Path	Sends commands to or receives commands from the SD/SDIO/eMMC.
Data Path	Consists of Data TX and Data RX sub-modules. The Data TX sends data blocks and the CRC codes to the SD/SDIO/eMMC. The Data RX receives data blocks and the CRC codes from the SD/SDIO/eMMC.



### 3.3.3 Functional Description

#### 3.3.3.1 External Signals

The following table describes the external signals of SMHC.

**Table 3-2 SMHC External Signals**

Signal Name	Description	Type
<b>SMHC0</b>		
SDC0-CMD	Command Signal for SD Card	I/O, OD
SDC0-CLK	Clock for SD Card	O
SDC0-D[3:0]	DATA INPUT AND OUTPUT FOR SD CARD	I/O
<b>SMHC1</b>		
SDC1-CMD	Command Signal for SDIO WIFI	I/O, OD
SDC1-CLK	Clock for SDIO WIFI	O
SDC1-D[3:0]	Data Input and Output for SDIO WIFI	I/O
<b>SMHC2</b>		
SDC2-CMD	Command Signal for eMMC	I/O, OD
SDC2-CLK	Clock for eMMC	O
SDC2-D[8:0]	Data Input and Output for eMMC	I/O
SDC2-RST	Reset for eMMC	O
SDC2-DS	Clock input for eMMC	I

#### 3.3.3.2 Clock Sources

The SMHC0/1/2 has 5 different clock sources. You can select one of them as the SMHC clock source. The following table describes the clock sources of the SMHC.

For clock setting, configurations, and gating information, refer to section 2.5 Clock Controller Unit (CCU).

**Table 3-3 SMHC0/1 Clock Sources**

Clock Sources	Description	Module
HOSC	24 MHz Crystal	CCU
PLL_PERI0(400M)	Peripheral Clock, the default value is 400 MHz	
PLL_PERI0(300M)	Peripheral Clock, the default value is 300 MHz	
PLL_PERI1(400M)	Peripheral Clock, the default value is 400 MHz	
PLL_PERI1(300M)	Peripheral Clock, the default value is 300 MHz	

**Table 3-4 SMHC2 Clock Sources**

Clock Sources	Description	Module
HOSC	24 MHz Crystal	CCU
PLL_PERI0(800M)	Peripheral Clock, the default value is 800 MHz	
PLL_PERI0(600M)	Peripheral Clock, the default value is 600 MHz	

Clock Sources	Description	Module
PLL_PERI1(800M)	Peripheral Clock, the default value is 800 MHz	
PLL_PERI1(600M)	Peripheral Clock, the default value is 600 MHz	

### 3.3.3.3 Timing Diagram

Refer to the following relative specifications:

- Physical Layer Specification Ver3.00 Final
- SDIO Specification Ver2.00
- Multimedia Cards (MMC – version 4.2)
- JEDEC Standard – JESD84-44, Embedded Multimedia Card (eMMC) Card Product Standard
- JEDEC Standard – JESD84-B45, Embedded Multimedia Card (eMMC) Electrical Standard (4.5 Device)
- JEDEC Standard – JESD84-B50, Embedded Multimedia Card (eMMC) Electrical Standard (5.0)

### 3.3.3.4 Data Path

The SMHC and SD/SDIO/eMMC contains the following interface buses: CLK, CMD, and DATA 1/4. During one clock cycle, the SMHC can transmit one-bit command with one or two bits' data in 1-ch DATA mode, or four or eight bits' data in 4-ch DATA mode. The CMD is a bidirection channel for initializing the SD/SDIO/eMMC and transmitting commands. It can work in both the open-drain mode and push-pull mode. The DATA is also a bidirection channel. It works in the push-pull mode.

- Reading Data from the SD/SDIO/eMMC

The register configures the signals for the read operation, and synchronize the signals to the SMHC clock domain. Then the Data RX reads data from the SD/SDIO/eMMC via the CLK/CMD/DATA interface buses and writes the data in the FIFO. After that, the DMAC transfers the data from the FIFO to the memory.

- Writing Data to the SD/SDIO/eMMC

The register configures the signals for the write operation, and synchronize the signals to the SMHC clock domain. Then the DMAC reads data from the memory and writes the data to the FIFO. After that, the Data TX reads the data from the FIFO and writes the data to the SD/SDIO/eMMC via the CLK/CMD/DATA interface buses.

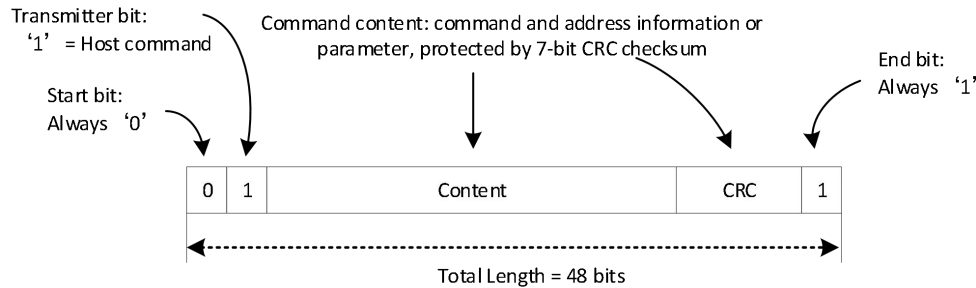
### 3.3.3.5 Package Format

Data transfer over the SD/eMMC bus is based on command and data bitstreams that are initiated by a start bit and terminated by a stop bit. There are three types of SD/eMMC packets: command token, response token, and data packet.

## Command Tokens

The command token starts an operation. A command is sent from the host to a device. It is transferred serially on the CMD line. Command tokens have the following coding scheme:

**Figure 3-2 Command Token Format**



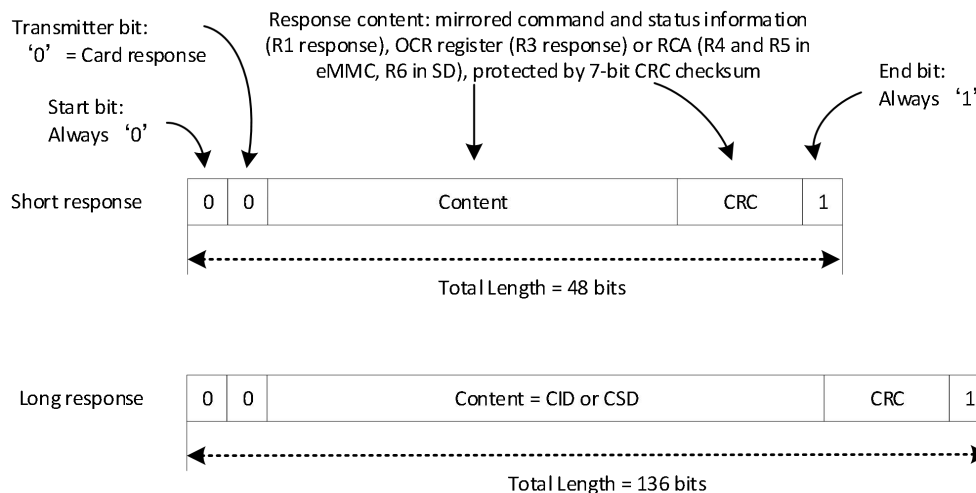
Each command token has 48 bits, preceded by a start bit ('0') and succeeded by an end bit ('1'). To detect transmission errors, each token is protected by CRC bits.

## Response Tokens

After receiving a command, the card returns a 48-bit or 136-bit response based on the command type.

A response token is sent from the device to the host as an answer to a previously received command. It is transferred serially on the CMD line.

**Figure 3-3 Response Token Format**

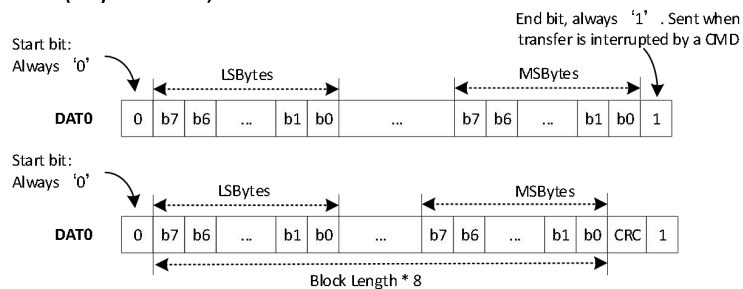


## Data Packet

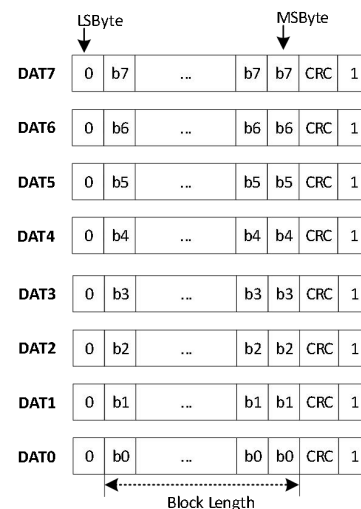
Data can be transferred from the device to the host or vice versa. Data are transferred via the data lines.

Figure 3-4 Data Packet Format for SDR

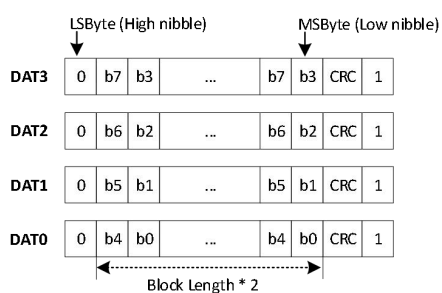
**1 Bit Bus (only DAT0 used)**



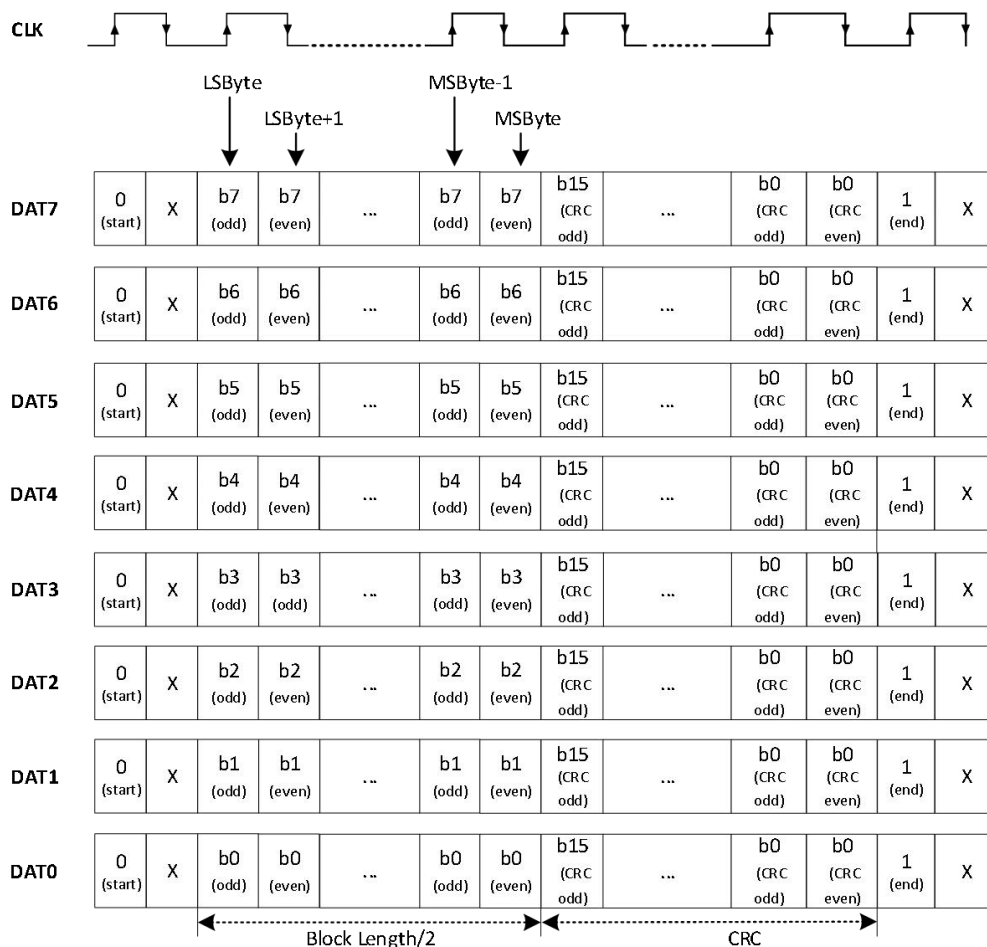
**8 Bits Bus (DAT7 – DAT0 used)**



**4 Bits Bus (DAT3 – DAT0 used)**



**4 Bits Bus DDR (DAT3 – DAT0 used)**

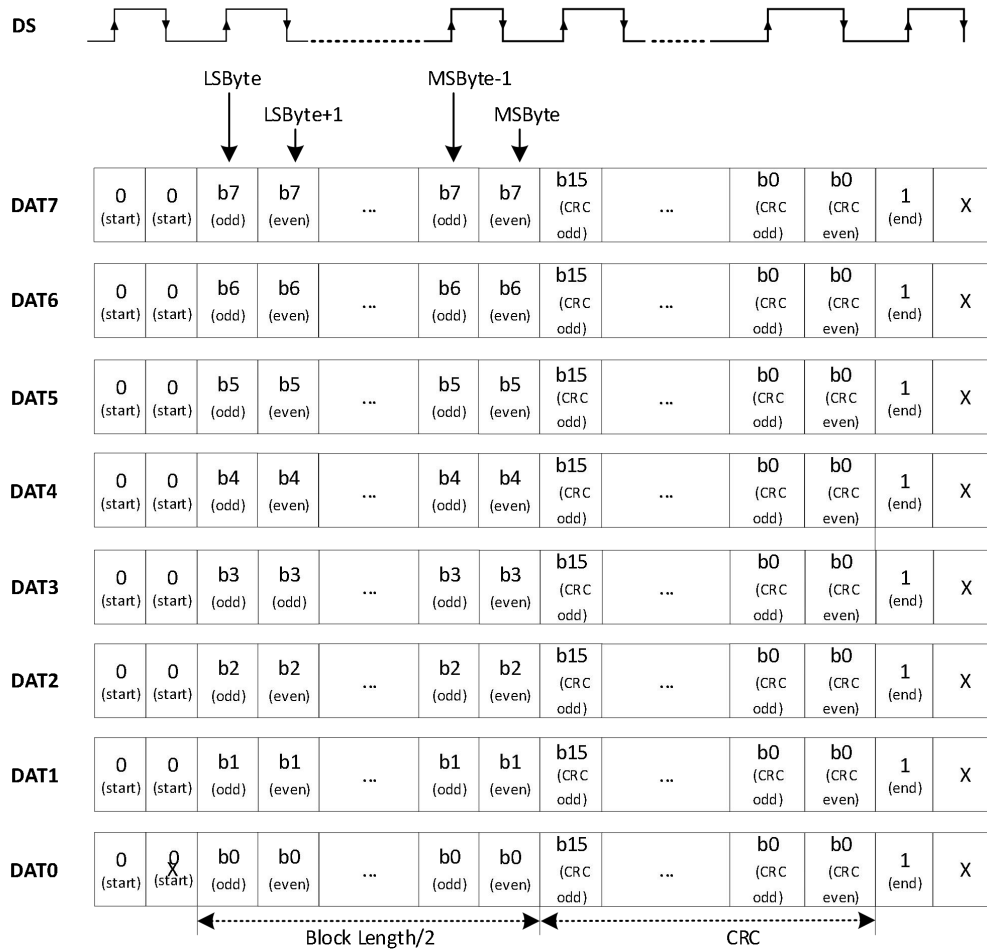


NOTE

- Bytes data are not interleaved but CRCs are interleaved.
- Start and end bits are only valid on the rising edge (“X” indicates “undefined”).

Figure 3-6 Data Packet Format for DDR in HS400 Mode

8 Bits Bus DDR for HS400 Output (DAT7 – DAT0 used)



NOTE

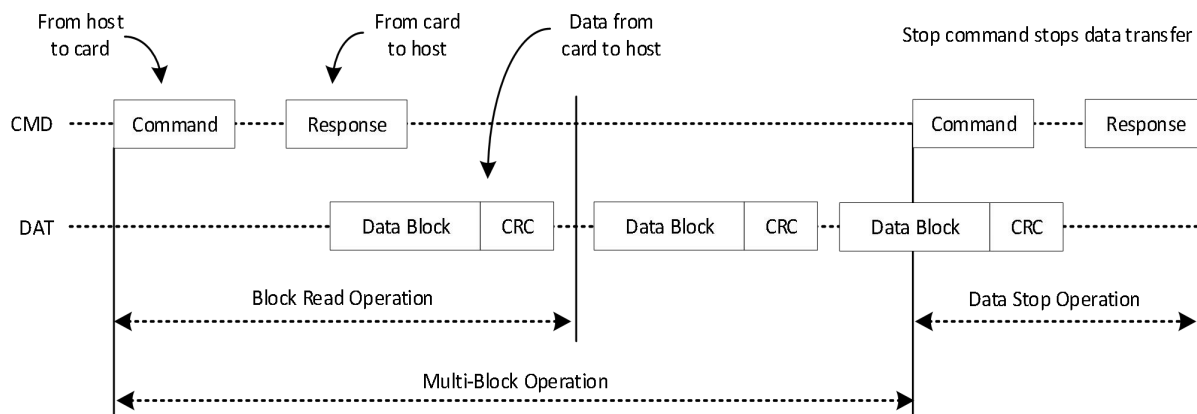
- Bytes data are not interleaved but CRCs are interleaved.
- Start bits are valid when Data Strobe is High and Low.
- End bits are only valid when Data Strobe is High (“X” indicates “undefined”).

Data Transfer

Data transfers to or from the SD/eMMC card are done in blocks. Single and multiple block operations are widely used during data transfer.

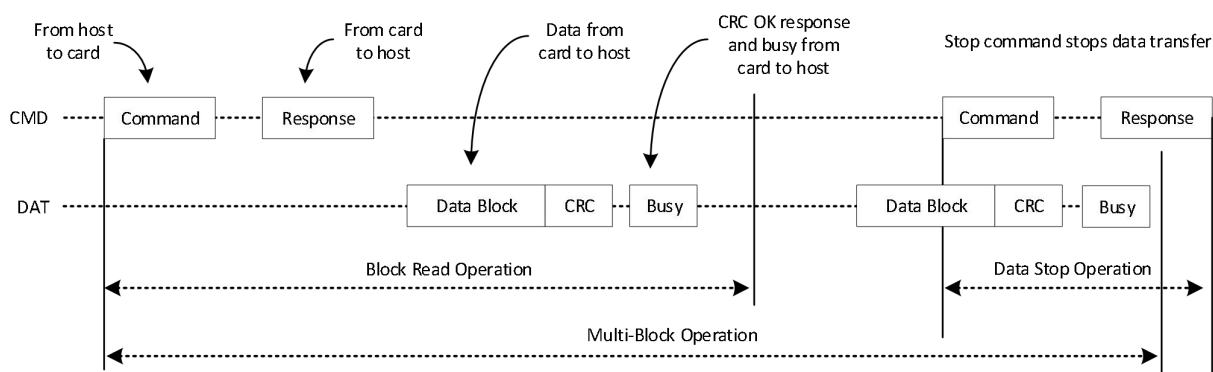
The following figure shows the single-block and multi-block read operation.

**Figure 3-7 Single-Block and Multi-Block Read Operation**



The following figure shows the single-block and multi-block write operation.

**Figure 3-8 Single-Block and Multi-Block Read Operation**



### 3.3.3.6 Phase Offset of the Command and Data

To obtain the command phase or data phase in output timing, follow the steps blow:

**Step 1** Configure the MODE\_SEL (bit [31]) and/or HS400\_NEW\_SAM\_EN (bit [0]) of [SMHC\\_NTSR \(offset: 0x005C\)](#) to choose the timing mode.

**Step 2** Configure the DAT\_DRV\_PH\_SEL (bit [17]) to select data drive phase offset and configure or CMD\_DRV\_PH\_SEL (bit [16]) to select command drive phase offset in [SMHC\\_DRV\\_DL \(offset: 0x0140\)](#) to select data or command drive phase offset.

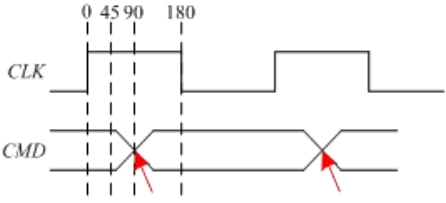
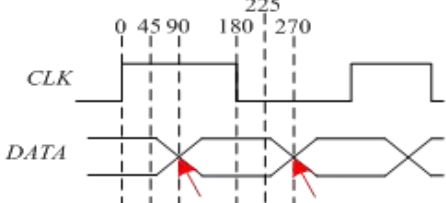
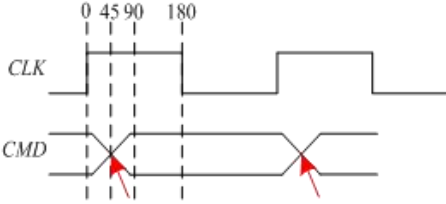
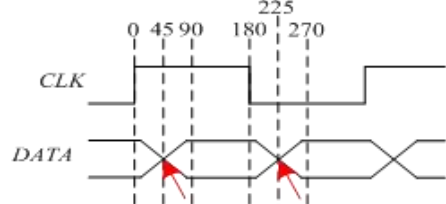
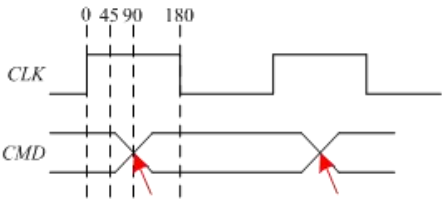
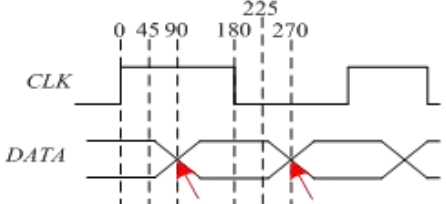
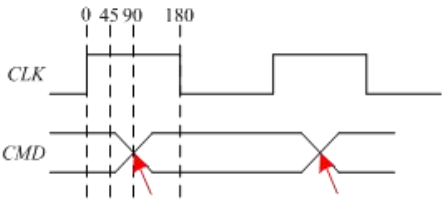
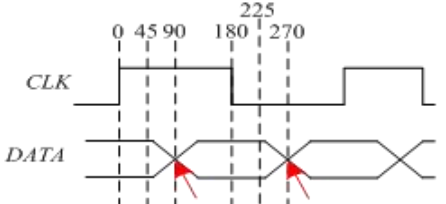
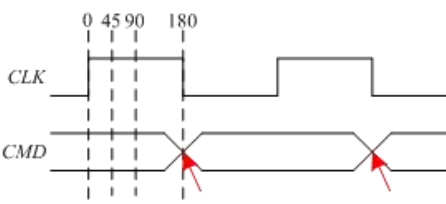
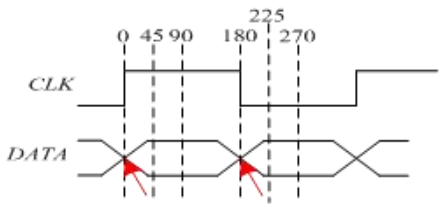
The following table shows the specific configuration of command and data location.

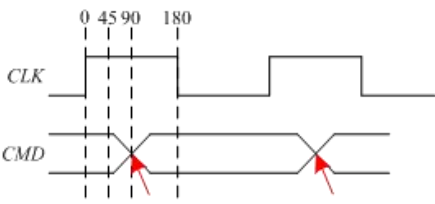
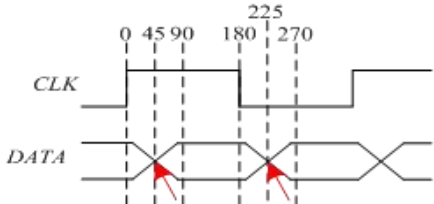
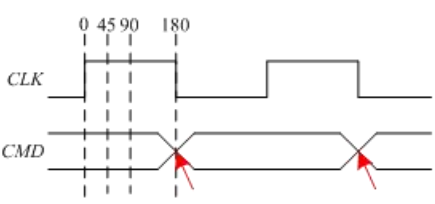
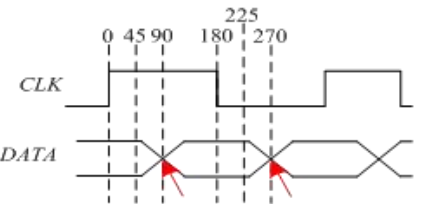
**Table 3-5 Command and Data Location**

Timing Mode	DRV	Command Drive Phase offset	Data Drive Phase offset
-------------	-----	----------------------------	-------------------------

Timing Mode	DRV	Command Drive Phase offset	Data Drive Phase offset
SDR	0		
		The Command updates at 90 degrees.	The data update at 90 degrees.
	1		
		The Command updates at 180 degrees.	The data updates at 180 degrees.
DDR4 (1x mode: 0x5c[31]=0)	0		
		The Command updates at 90 degrees.	The data update at 90 degrees.
	1		
		The Command updates at 180 degrees.	The data update at 0 or 180 degrees.
DDR4 (2x mode: 0x5c[31]=1)	0		



Timing Mode	DRV	Command Drive Phase offset	Data Drive Phase offset
	1	The Command updates at 45 degrees.	The data update at 45 degrees.
			
DDR8	0	The Command updates at 45 degrees.	The data update at 90 degrees.
			
	1	The Command updates at 45 degrees.	The data update at 45 degrees.
			
HS400 (1x mode: 0x5c[0]=0)	0	The Command updates at 90 degrees.	The data update at 90 degrees.
			
	1	The Command updates at 180 degrees.	The data update at 0 or 180 degrees.
			

Timing Mode	DRV	Command Drive Phase offset	Data Drive Phase offset
HS400 (2x mode: 0x5c[0] =1)	0		
		The Command updates at 90 degrees.	The data update at 45 degrees.
	1		
		The Command updates at 180 degrees.	The data update at 90 degrees.

### 3.3.3.7 Internal DMA Controller Description

The SMHC has an internal DMA controller (IDMAC) to transfer data between the host memory and SMHC port. With a descriptor, the IDMAC can efficiently move data from the source to destination by automatically loading the next DMA transfer arguments, which needs less CPU intervention. Before transferring data in the IDMAC, the Host CPU should construct a descriptor list, configure arguments of every DMA transfer, and then launch the descriptor and start the DMA.

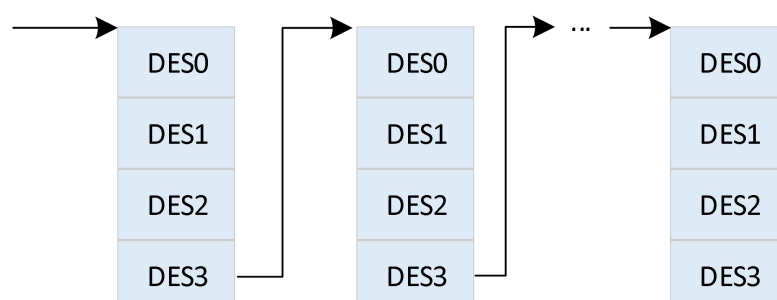
The IDMAC has an interrupt controller. When enabled, it generates an interrupt to the Host CPU in situations such as data transmission is completed or some error is happened.

#### IDMAC Descriptor Structure

The IDMAC uses a descriptor with a chain structure, and each descriptor points to a unique buffer and the next descriptor.

The following figure shows the internal formats of a descriptor.

Figure 3-9 IDMAC Descriptor Structure Diagram



This figure illustrates the internal formats of a descriptor. The descriptor address must be aligned to the bus width used for 32-bit buses. Each descriptor contains 16 bytes of control and status information.

DES0 corresponds to the [31:0] bits, DES1 corresponds to the [63:32] bits, DES2 corresponds to the [95:64] bits, and DES3 corresponds the [127:96] bits in a descriptor.

The following table shows the bit definition of DES0.

**Table 3-6 DES0 Definition**

Bits	Name	Description
31	HOLD	DES_OWN_FLAG When set, this bit indicates that the descriptor is owned by the IDMAC. When this bit is reset, it indicates that the descriptor is owned by the host. This bit is cleared when the transfer is over. <b>Note: After this bit is cleared to 0, the Host CPU is able to read the transferred data or initiate next transfer by launching a new descriptor.</b>
30	ERROR	ERR_FLAG When some errors happen in transfer, this bit will be set to 1.
29:5	/	/
4	Chain Flag	CHAIN_MOD When set to 1, this bit indicates that the second address in the descriptor is the next descriptor address. It must be set to 1.
3	First DES Flag	FIRST_FLAG When set to 1, this bit indicates that this descriptor contains the first buffer of data. It must be set to 1 in the first DES.
2	Last DES Flag	LAST_FLAG When set to 1, this bit indicates that the buffers this descriptor points to are the last data buffer.
1	Disable Interrupt on completion	CUR_TXRX_OVER_INT_DIS When set to 1, this bit will prevent the setting of the TX/RX interrupt bit of the IDMAC status register for data that ends in the buffer the descriptor points to.
0	/	/

The following table shows the bit definition of DES1.

**Table 3-7 DES1 Definition**

Bits	Name	Description
31:13	/	/
12:0	Buffer size	BUFF_SIZE The bits indicate the data buffer byte size, which must be a multiple of 4 bytes. If this field is 0, the DMA ignores this buffer and proceeds to the next descriptor.

The following table shows the bit definition of DES2.

**Table 3-8 DES2 Definition**

Bits	Name	Descriptor
31:0	Buffer address pointer	BUFF_ADDR The bits indicate the physical address of the data buffer. It is a word address.

The following table shows the bit definition of DES3.

**Table 3-9 DES3 Definition**

Bits	Name	Descriptor
31:0	Next descriptor address	NEXT_DESP_ADDR The bits indicate the pointer to the physical memory where the next descriptor is present. It is a word address.

### 3.3.3.8 Calibrating the Delay Chain

There are two delay chains in SMHC: data strobe delay chain and sample delay chain.

- Data strobe delay chain: used to generate delay to make proper timing between Data Strobe and data signals.
- Sample delay chain: used to generate delay to make proper timing between the internal card clock signal and data signals.

Each delay chain is made up with 64 delay cells. The delay time of one delay cell can be estimated through delay chain calibration.

Follow the steps below to calibrate the delay chain:

- Step 1** Enable SMHC. In order to calibrate the delay chain by the operation registers in SMHC, the SMHC must be enabled through [SMHC Bus Gating Reset Register](#) and [SMHCx Clock Register](#)(x=0, 1, or 2).
- Step 2** Configure a proper clock for SMHC. The delay chain calibration is based on the clock for SMHC from Clock Control Unit (CCU). The delay chain calibration is an internal function in SMHC and needs no devices. So it is unnecessary to open the clock signal for devices. The recommended clock frequency is 200 MHz.
- Step 3** Set proper initial delay value. Writing 0xA0 to delay control register enables Delay Software Enable (bit [7]) and sets initial delay value 0x20 to Delay chain (bit [5:0]). Then write 0x0 to delay control register to clear the value.
- Step 4** Write 0x8000 to delay control register to start calibrating the delay chain.
- Step 5** Wait until the flag (bit14 in delay control register) of calibration done is set. The number of delay cells is shown at bit [13:8] in delay control register. The delay time generated by

these delay cells is equal to the cycle of the SMHC clock nearly. This value is the result of calibration.

- Step 6** Calculate the delay time of one delay cell according to the cycle of the SMHC clock and the result of calibration.

### 3.3.4 Programming Guidelines

#### 3.3.4.1 Initializing SMHC

Before data and commands are exchanged between a card and the SMHC, the SMHC needs to be initialized. Follow the steps below to initialize the SMHC:

- Step 1** Configure the corresponding GPIO register as an SMHC in section 8.5 GPIO; reset clock by writing 1 to [SMHC\\_BGR\\_REG](#)[SMHCx\_RST](x=0, 1, or 2), and open clock gating by writing 1 to [SMHC\\_BGR\\_REG](#)[SMHCx\_GATING] (x=0, 1, or 2); select clock sources and set the division factor by configuring the [SMHCx\\_CLK\\_REG](#) (x = 0, 1, or 2) register.
- Step 2** Configure [SMHC\\_CTRL](#) to reset the FIFO and controller, and enable the global interrupt; configure [SMHC\\_INTMASK](#) to 0xFFCE to enable normal interrupts and error abnormal interrupts, and then register the interrupt function.
- Step 3** Configure [SMHC\\_CLKDIV](#) to open clock for devices; configure [SMHC\\_CMD](#) as the change clock command (for example 0x80202000); send the update clock command to deliver clocks to devices.
- Step 4** Configure [SMHC\\_CMD](#) as a normal command. Configure [SMHC\\_CMDARG](#) to set command parameters. Configure [SMHC\\_CMD](#) to set parameters like whether to send the response, the response type, and the response length and then send the commands. According to the initialization process in the protocol, you can finish SMHC initialization by sending the corresponding command one by one.

#### 3.3.4.2 Writing a Single Data Block

To write a single data block, follow the steps below:

- Step 1** Write 0x1 to [SMHC\\_CTRL](#)[DMA\_RST] to reset the internal DMA controller; write 0x82 to [SMHC\\_IDMAC](#) to enable the IDMAC interrupt and configure AHB master burst transfers; configure [SMHC\\_IDIE](#) to enable the transfer interrupt, receive interrupt, and abnormal interrupt.
- Step 2** Configure [SMHC\\_FIFOTH](#) to determine the burst size and TX/RX trigger level. For example, if [SMHC\\_FIFOTH](#) is configured as 0x300F00F0, it indicates the burst size is 16, TX\_TL is 15, and RX\_TL is 240. Configure [SMHC\\_DLBA](#) to determine the start address of the DMA descriptor.

- Step 3** To write one block data to sector1, configure [SMHC\\_BYTCNT](#)[BYTE\_CNT] to 0x200 and configure the descriptor according to the data size; set the data sector address of CMD24 (Single Data Block Write) to 0x1, write 0x80002758 to [SMHC\\_CMD](#), and send CMD24 command to write data to the device.
- Step 4** Check whether [SMHC\\_RINTSTS](#)[CC] is 1. If yes, the command is sent successfully; otherwise, continue to wait until timeout, and then exit the process.
- Step 5** Check whether [SMHC\\_IDST](#)[TX\_INT] is 1. If yes, the data transfer for writing DMA is completed. Write 0x337 to [SMHC\\_IDST](#) to clear the interrupt flag. Otherwise, continue to wait until timeout, and then exit the process.
- Step 6** Check whether [SMHC\\_RINTSTS](#)[DTC] is 1. If yes, the data transfer and CMD24 writing operations are completed. Otherwise, abnormality exists. Read [SMHC\\_RINTSTS](#) and [SMHC\\_STATUS](#) to query the existing abnormality.
- Step 7** Send CMD13 command to query whether the device writing operation is completed and returns to the idle status. For example, device RCA is 0x1234, first set [SMHC\\_CMDARG](#) to 0x12340000, write 0x8000014D to [SMHC\\_CMD](#), go to step4 to ensure command transfer completed, and then check whether the highest bit of [SMHC\\_RESP0](#) (CMD13 response) is 1. If yes, the device is in the idle status, and the next command can be sent. Otherwise, the device is in the busy status. Continue to send CMD13 to wait for the device to enter the idle status until timeout.

### 3.3.4.3 Reading a Single Data Block

To read a single data block, follow the steps below:

- Step 1** Write 0x1 to [SMHC\\_CTRL](#)[DMA\_RST] to reset the internal DMA controller; write 0x82 to [SMHC\\_IDMAC](#) to enable the IDMAC interrupt and configure AHB master burst transfers; configure [SMHC\\_IDIE](#) to enable the transfer interrupt, receive interrupt, and abnormal interrupt.
- Step 2** Configure [SMHC\\_FIFOTH](#) to determine the burst size and TX/RX trigger level. For example, if [SMHC\\_FIFOTH](#) is configured as 0x300F00F0, it indicates the burst size is 16, TX\_TL is 15, and RX\_TL is 240. Configure [SMHC\\_DLBA](#) to determine the start address of the DMA descriptor.
- Step 3** To read one block data from sector1, configure [SMHC\\_BYTCNT](#)[BYTE\_CNT] to 0x200 and configure the descriptor according to the data size; set the data sector address of CMD17 command (Single Data Block Read) to 0x1, write 0x80002351 to [SMHC\\_CMD](#), and send CMD17 command to read data from the device to DRAM/SRAM.
- Step 4** Check whether [SMHC\\_RINTSTS](#)[CC] is 1. If yes, the command is sent successfully; otherwise, continue to wait until timeout, and then exit the process.

- Step 5** Check whether [SMHC\\_IDST](#)[TX\_INT] is 1. If yes, the data transfer for writing DMA is completed. Write 0x337 to [SMHC\\_IDST](#) to clear the interrupt flag. Otherwise, continue to wait until timeout, and then exit the process.
- Step 6** Check whether [SMHC\\_RINTSTS](#)[DTC] is 1. If yes, data transfer and CMD17 reading operation are completed. Otherwise, abnormality exists. Read [SMHC\\_RINTSTS](#) and [SMHC\\_STATUS](#) to query the existing abnormality.

#### 3.3.4.4 Writing Open-Ended Multiple Data Blocks (CMD25 + Auto CMD12)

To write open-ended multiple data blocks, follow the steps below:

- Step 1** Write 0x1 to [SMHC\\_CTRL](#)[DMA\_RST] to reset the internal DMA controller; write 0x82 to [SMHC\\_IDMAC](#) to enable the IDMAC interrupt and configure AHB master burst transfers; configure [SMHC\\_IDIE](#) to enable the transfer interrupt, receive interrupt, and abnormal interrupt.
- Step 2** Configure [SMHC\\_FIFOTH](#) to determine the burst size and TX/RX trigger level. For example, if [SMHC\\_FIFOTH](#) is configured as 0x300F00F0, it indicates the burst size is 16, TX\_TL is 15, and RX\_TL is 240. Configure [SMHC\\_DLBA](#) to determine the start address of the DMA descriptor.
- Step 3** To write three blocks of data to sectors begin with sector0, configure [SMHC\\_BYTCNT](#)[BYTE\_CNT] to 0x600 and configure the descriptor according to the data size; set the data sector address of CMD25 command (Multiple Data Blocks Write) to 0x0, write 0x80003759 to [SMHC\\_CMD](#), and send CMD25 command to read data from the device to DRAM/SRAM.
- Step 4** Check whether [SMHC\\_RINTSTS](#)[CC] is 1. If yes, the command is sent successfully; otherwise, continue to wait until timeout, and then exit the process.
- Step 5** Check whether [SMHC\\_IDST](#)[TX\_INT] is 1. If yes, the data transfer for writing DMA is completed. Write 0x337 to [SMHC\\_IDST](#) to clear the interrupt flag. Otherwise, continue to wait until timeout, and then exit the process.
- Step 6** Check whether [SMHC\\_RINTSTS](#)[ACD] and [SMHC\\_RINTSTS](#)[DTC] are both 1. If yes, the data transfer, CMD12 transfer, and CMD25 writing operations are completed. Otherwise, abnormality exists. Read [SMHC\\_RINTSTS](#) and [SMHC\\_STATUS](#) to query the existing abnormality.
- Step 7** Send CMD13 command to query whether the device writing operation is completed and returns to the idle status. For example, device RCA is 0x1234, first set [SMHC\\_CMDARG](#) to 0x12340000, write 0x8000014D to [SMHC\\_CMD](#), go to step4 to ensure command transfer completed, and then check whether the highest bit of [SMHC\\_RESP0](#) (CMD13 response) is 1. If yes, the device is in the idle status, and the next command can be sent. Otherwise,

the device is in the busy status. Continue to send CMD13 to wait for the device to enter the idle status until timeout.

#### 3.3.4.5 Reading Open-Ended Multiple Data Blocks (CMD18 + Auto CMD12)

To read open-ended multiple data blocks, follow the steps below:

- Step 1** Write 0x1 to [SMHC\\_CTRL](#)[DMA\_RST] to reset the internal DMA controller; write 0x82 to [SMHC\\_IDMAC](#) to enable the IDMAC interrupt and configure AHB master burst transfers; configure [SMHC\\_IDIE](#) to enable the transfer interrupt, receive interrupt, and abnormal interrupt.
- Step 2** Configure [SMHC\\_FIFOTH](#) to determine the burst size and TX/RX trigger level. For example, if [SMHC\\_FIFOTH](#) is configured as 0x300F00F0, it indicates the burst size is 16, TX\_TL is 15, and RX\_TL is 240. Configure [SMHC\\_DLBA](#) to determine the start address of the DMA descriptor.
- Step 3** To read three blocks of data from sectors begin with sector0, configure [SMHC\\_BYTCNT](#)[BYTE\_CNT] to 0x600 and configure the descriptor according to the data size; set the data sector address of CMD18 command (Multiple Data Blocks Read) to 0x0, write 0x80003352 to [SMHC\\_CMD](#), and send CMD18 command to read data to the device. When the data transfer is completed, CMD12 will be sent automatically.
- Step 4** Check whether [SMHC\\_RINTSTS](#)[CC] is 1. If yes, the command is sent successfully; otherwise, continue to wait until timeout, and then exit the process.
- Step 5** Check whether [SMHC\\_IDST](#)[RX\_INT] is 1. If yes, the data transfer for writing DMA is completed. Write 0x337 to [SMHC\\_IDST](#) to clear the interrupt flag. Otherwise, continue to wait until timeout, and then exit the process.
- Step 6** Check whether [SMHC\\_RINTSTS](#)[ACD] and [SMHC\\_RINTSTS](#)[DTC] are both 1. If yes, data transfer, CMD12 transfer, and CMD18 reading operation are completed. Otherwise, abnormality exists. Read [SMHC\\_RINTSTS](#) and [SMHC\\_STATUS](#) to query the existing abnormality.

#### 3.3.4.6 Writing Pre-Defined Multiple Data Blocks (CMD23 + CMD25)

To write pre-defined multiple data blocks, follow the steps below:

- Step 1** Write 0x1 to [SMHC\\_CTRL](#)[DMA\_RST] to reset the internal DMA controller; write 0x82 to [SMHC\\_IDMAC](#) to enable the IDMAC interrupt and configure AHB master burst transfers; configure [SMHC\\_IDIE](#) to enable the transfer interrupt, receive interrupt, and abnormal interrupt.



- Step 2** Configure [SMHC\\_FIFOTH](#) to determine the burst size and TX/RX trigger level. For example, if [SMHC\\_FIFOTH](#) is configured as 0x300F00F0, it indicates the burst size is 16, TX\_TL is 15, and RX\_TL is 240. Configure [SMHC\\_DLBA](#) to determine the start address of the DMA descriptor.
- Step 3** To write three blocks of data, configure [SMHC\\_CMDARG](#) to 0x3 to specify the number of data blocks as three. Then write 0x80000157 to [SMHC\\_CMD](#) to send the CMD23 command. Check whether [SMHC\\_RINTSTS](#)[CC] is 1. If yes, the command is sent successfully; otherwise, continue to wait until timeout, and then exit the process.
- Step 4** Configure [SMHC\\_BYTCNT](#)[BYTE\_CNT] to 0x600 and configure the descriptor according to the data size; set the data sector address of CMD25 command (Multiple Data Blocks Write) to 0x0, write 0x80002759 to [SMHC\\_CMD](#), and send CMD25 command to write data to the device.
- Step 5** Check whether [SMHC\\_RINTSTS](#)[CC] is 1. If yes, the command is sent successfully; otherwise, continue to wait until timeout, and then exit the process.
- Step 6** Check whether [SMHC\\_IDST](#)[TX\_INT] is 1. If yes, the data transfer for writing DMA is completed. Write 0x337 to [SMHC\\_IDST](#) to clear the interrupt flag. Otherwise, continue to wait until timeout, and then exit the process.
- Step 7** Check whether [SMHC\\_RINTSTS](#)[DTC] is 1. If yes, the data transfer and CMD25 writing operations are completed. Otherwise, abnormality exists. Read [SMHC\\_RINTSTS](#) and [SMHC\\_STATUS](#) to query the existing abnormality.
- Step 8** Send CMD13 command to query whether the device writing operation is completed and returns to the idle status. For example, device RCA is 0x1234, first set [SMHC\\_CMDARG](#) to 0x12340000, write 0x8000014D to [SMHC\\_CMD](#), go to step5 to ensure command transfer completed, and then check whether the highest bit of [SMHC\\_RESP0](#) (CMD13 response) is 1. If yes, the device is in the idle status, and the next command can be sent. Otherwise, the device is in the busy status. Continue to send CMD13 to wait for the device to enter the idle status until timeout

#### 3.3.4.7 Reading Pre-Defined Multiple Data Blocks (CMD23 + CMD18)

To read pre-defined multiple data blocks, follow the steps below:

- Step 1** Write 0x1 to [SMHC\\_CTRL](#)[DMA\_RST] to reset the internal DMA controller; write 0x82 to [SMHC\\_IDMAC](#) to enable the IDMAC interrupt and configure AHB master burst transfers; configure [SMHC\\_IDIE](#) to enable the transfer interrupt, receive interrupt, and abnormal interrupt.
- Step 2** Configure [SMHC\\_FIFOTH](#) to determine the burst size and TX/RX trigger level. For example, if [SMHC\\_FIFOTH](#) is configured as 0x300F00F0, it indicates the burst size is 16,

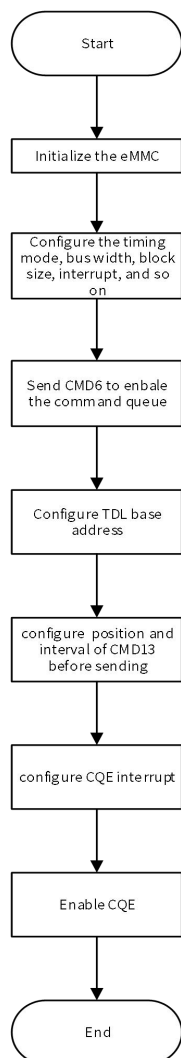
TX\_TL is 15, and RX\_TL is 240. Configure [SMHC\\_DLBA](#) to determine the start address of the DMA descriptor.

- Step 3** To read three blocks of data, configure [SMHC\\_CMDARG](#) to 0x3 to specify the number of data blocks as three. Then write 0x80000157 to [SMHC\\_CMD](#) to send the CMD23 command. Check whether [SMHC\\_RINTSTS](#)[CC] is 1. If yes, the command is sent successful; otherwise, continue to wait until timeout, and then exit the process.
- Step 4** Configure [SMHC\\_BYTCNT](#)[BYTE\_CNT] to 0x600 and configure the descriptor according to the data size; set the data sector address of CMD18 (Multiple Data Blocks Read) to 0x0, write 0x80002352 to [SMHC\\_CMD](#), and send CMD18 command to read data from device to DRAM/SRAM.
- Step 5** Check whether [SMHC\\_RINTSTS](#)[CC] is 1. If yes, the command is sent successful; otherwise, continue to wait until timeout, and then exit the process.
- Step 6** Check whether [SMHC\\_IDST](#)[RX\_INT] is 1. If yes, the data transfer for writing DMA is completed. Write 0x337 to [SMHC\\_IDST](#) to clear the interrupt flag. Otherwise, continue to wait until timeout, and then exit the process.
- Step 7** Check whether [SMHC\\_RINTSTS](#)[DTC] is 1. If yes, the data transfer and CMD18 writing operations are completed. Otherwise, abnormality exists. Read [SMHC\\_RINTSTS](#) and [SMHC\\_STATUS](#) to query the existing abnormality.

### 3.3.4.8 Initializing Command Queue for eMMC V5.1 Device

The following figure describes the initialization process of command queue.

**Figure 3-10 Initialization Process of Command Queue**



Follow the steps below to initialize the command queue:

- Step 1** Initialize the eMMC. For detailed initialization steps, refer to section 3.3.4.1 Initializing SMHC.
- Step 2** Send CMD6 to configure the timing mode and bus width. Send CMD16 to configure the block size as 512 B.



#### NOTE

The block size must be set to 512 B. After the command queue is enabled, the block size will not be able to be modified.

- Step 3** Send CMD6 to enable the command queue of the eMMC device.

**Step 4** Configure [CQDLBA](#) register to set the base address of the task descriptor list.



#### NOTE

When the base address of the task descriptor list crosses 4 GB, you need to write the bit [32] of the higher 32-bit address to the bit [0] of the CQDLBA register and write the lower 32-bit address to the CQDLBA register.

For example, assume that the base address is 0x106000000. In this case, you need to write the bit [32] of the higher 32-bit address to the bit [0] of the CQDLBA register and write 0x06000001 to the CQDLBA register.

**Step 5** Configure [CQSSC1](#) register to set how to query the status of the device's task queue.

**Step 6** Configure [CQIC](#) register to enable/disable interrupt, set interrupt count and timer protection.

**Step 7** Configure [CQRMEM](#) register to set which errors may trigger a RED interrupt.

**Step 8** Configure [CQCFG](#) register to enable CQE activity.

eMMC CMDQ supports multiple operation modes, which could be selected by configuring the CMDQ\_MODE bit (bit [5:4]) of [CQCFG](#) register.

### 3.3.5 Register List

Module Name	Base Address	Description
SMHC0	0x04020000	
SMHC1	0x04021000	SMHC1 register is the same with SMHC0
SMHC2	0x04022000	SMHC2 register is the same with SMHC0

Register Name	Offset	Description
SMHC_CTRL	0x0000	SMHC Global Control Register
SMHC_CLKDIV	0x0004	SMHC Clock Control Register
SMHC_TMOUT	0x0008	SMHC Timeout Register
SMHC_CTYPE	0x000C	SMHC Bus Width Register
SMHC_BLKSIZE	0x0010	SMHC Block Size Register
SMHC_BYTCNT	0x0014	SMHC Byte Count Register
SMHC_CMD	0x0018	SMHC Command Register
SMHC_CMDARG	0x001C	SMHC Command Argument Register
SMHC_RESP0	0x0020	SMHC Response 0 Register
SMHC_RESP1	0x0024	SMHC Response 1 Register
SMHC_RESP2	0x0028	SMHC Response 2 Register
SMHC_RESP3	0x002C	SMHC Response 3 Register
SMHC_INTMASK	0x0030	SMHC Interrupt Mask Register