Note

This documentation is not for the latest stable release version. The latest stable version is v5.3.1

SD/SDIO/MMC Driver

[中文]

Overview

The SD/SDIO/MMC driver currently supports SD memory, SDIO cards, and eMMC chips. This is a protocol level driver built on top of SDMMC and SD SPI host drivers.

SDMMC and SD SPI host drivers (driver/sdmmc/include/driver/sdmmc_host.h and driver/spi/include/driver/sdspi_host.h) provide API functions for:

- Sending commands to slave devices
- Sending and receiving data
- Handling error conditions within the bus

For functions used to initialize and configure:

- SDMMC host, see SDMMC Host API
- SD SPI host, see SD SPI Host API

The SDMMC protocol layer described in this document handles the specifics of the SD protocol, such as the card initialization and data transfer commands.

The protocol layer works with the host via the `sdmmc_host_t` structure. This structure contains pointers to various functions of the host.

Pin Configurations

SDMMC pins are dedicated, you don't have to configure the pins.

Application Example

An example which combines the SDMMC driver with the FATFS library is provided in the storage/sd_card directory of ESP-IDF examples. This example initializes the card, then writes and reads data from it using POSIX and C library APIs. See README.md file in the example directory for more information.

Combo (Memory + IO) Cards

The driver does not support SD combo cards. Combo cards are treated as IO cards.

Thread Safety

Most applications need to use the protocol layer only in one task. For this reason, the protocol layer does not implement any kind of locking on the `sdmmc_card_t` structure, or when accessing SDMMC or SD SPI host drivers. Such locking is usually implemented on a higher layer, e.g., in the filesystem driver.

Protocol Layer API

The protocol layer is given the `sdmmc_host_t` structure. This structure describes the SD/MMC host driver, lists its capabilities, and provides pointers to functions of the driver. The protocol layer stores card-specific information in the `sdmmc_card_t` structure. When sending commands to the SD/MMC host driver, the protocol layer uses the `sdmmc_command_t` structure to describe the command, arguments, expected return values, and data to transfer if there is any.

Using API with SD Memory Cards

1. To initialize the host, call the host driver functions, e.g., `sdmmc_host_init()`, `sdmmc_host_init_slot()`.
2. To initialize the card, call `sdmmc_card_init()` and pass to it the parameters `host` - the host driver information, and `card` - a pointer to the structure `sdmmc_card_t` which will be filled with information about the card when the function completes.
3. To read and write sectors of the card, use `sdmmc_read_sectors()` and `sdmmc_write_sectors()` respectively and pass to it the parameter `card` - a pointer to the card information structure.
4. If the card is not used anymore, call the host driver function - e.g., `sdmmc_host_deinit()` - to disable the host peripheral and free the resources allocated by the driver.

Using API with eMMC Chips

From the protocol layer's perspective, eMMC memory chips behave exactly like SD memory cards. Even though eMMCs are chips and do not have a card form factor, the terminology for SD cards can still be applied to eMMC due to the similarity of the protocol (*sdmmc_card_t*, *sdmmc_card_init*). Note that eMMC chips cannot be used over SPI, which makes them incompatible with the SD SPI host driver.

To initialize eMMC memory and perform read/write operations, follow the steps listed for SD cards in the previous section.

Using API with SDIO Cards

Initialization and the probing process are the same as with SD memory cards. The only difference is in data transfer commands in SDIO mode.

During the card initialization and probing, performed with `sdmmc_card_init()`, the driver only configures the following registers of the IO card:

1. The IO portion of the card is reset by setting RES bit in the I/O Abort (0x06) register.
2. If 4-line mode is enabled in host and slot configuration, the driver attempts to set the Bus width field in the Bus Interface Control (0x07) register. If setting the filed is successful, which means that the slave supports 4-line mode, the host is also switched to 4-line mode.
3. If high-speed mode is enabled in the host configuration, the SHS bit is set in the High Speed (0x13) register.

In particular, the driver does not set any bits in (1) I/O Enable and Int Enable registers, (2) I/O block sizes, etc. Applications can set them by calling `sdmmc_io_write_byte()`.

For card configuration and data transfer, choose the pair of functions relevant to your case from the table below.

Action	Read Function
Read and write a single byte using IO_RW_DIRECT (CMD52)	`sdmmc_io_read_byte(`
Read and write multiple bytes using IO_RW_EXTENDED (CMD53) in byte mode	`sdmmc_io_read_bytes`
Read and write blocks of data using IO_RW_EXTENDED (CMD53) in block mode	`sdmmc_io_read_block`

SDIO interrupts can be enabled by the application using the function `sdmmc_io_enable_int()`. When using SDIO in 1-line mode, the D1 line also needs to be connected to use SDIO interrupts.

If you want the application to wait until the SDIO interrupt occurs, use `sdmmc_io_wait_int()`.

There is a component ESSL (ESP Serial Slave Link) to use if you are communicating with an ESP32 SDIO slave. See ESP Serial Slave Link and example peripherals/sdio/host.

API Reference

Header File

- components/sdmmc/include/sdmmc_cmd.h
- This header file can be included with:

```
#include "sdmmc_cmd.h"
```

- This header file is a part of the API provided by the `sdmmc` component. To declare that your component depends on `sdmmc`, add the following to your CMakeLists.txt:

```
REQUIRES sdmmc
```

or

```
PRIV_REQUIRES sdmmc
```

Functions

esp_err_t **sdmmc_card_init**(*const* sdmmc_host_t *host, sdmmc_card_t *out_card)

Probe and initialize SD/MMC card using given host

Note

Only SD cards (SDSC and SDHC/SDXC) are supported now. Support for MMC/eMMC cards will be added later.

Parameters:
- **host** -- pointer to structure defining host controller

- **out_card** -- pointer to structure which will receive information about the card when the function completes

Returns:

- ESP_OK on success
- One of the error codes from SDMMC host controller

---

void **`sdmmc_card_print_info`**(FILE *stream, *const* sdmmc_card_t *card)

Print information about the card to a stream.

Parameters:

- **stream** -- stream obtained using fopen or fdopen
- **card** -- card information structure initialized using sdmmc_card_init

---

esp_err_t **`sdmmc_get_status`**(sdmmc_card_t *card)

Get status of SD/MMC card

Parameters: **card** -- pointer to card information structure previously initialized using sdmmc_card_init

Returns:

- ESP_OK on success
- One of the error codes from SDMMC host controller

---

esp_err_t **`sdmmc_write_sectors`**(sdmmc_card_t *card, *const* void *src, size_t start_sector, size_t sector_count)

Write given number of sectors to SD/MMC card

Parameters:

- **card** -- pointer to card information structure previously initialized using sdmmc_card_init
- **src** -- pointer to data buffer to read data from; data size must be equal to sector_count * card->csd.sector_size
- **start_sector** -- sector where to start writing
- **sector_count** -- number of sectors to write

Returns:

- ESP_OK on success or sector_count equal to 0
- One of the error codes from SDMMC host controller

---

esp_err_t **`sdmmc_read_sectors`**(sdmmc_card_t *card, void *dst, size_t start_sector, size_t sector_count)

Read given number of sectors from the SD/MMC card

Parameters:

- **card** -- pointer to card information structure previously initialized using sdmmc_card_init
- **dst** -- pointer to data buffer to write into; buffer size must be at least sector_count * card->csd.sector_size
- **start_sector** -- sector where to start reading
- **sector_count** -- number of sectors to read

Returns:

- ESP_OK on success or sector_count equal to 0
- One of the error codes from SDMMC host controller

esp_err_t `sdmmc_erase_sectors`(sdmmc_card_t *card, size_t start_sector, size_t sector_count, sdmmc_erase_arg_t arg)

Erase given number of sectors from the SD/MMC card

Note

When sdmmc_erase_sectors used with cards in SDSPI mode, it was observed that card requires re-init after erase operation.

Parameters:

- **card** -- pointer to card information structure previously initialized using sdmmc_card_init
- **start_sector** -- sector where to start erase
- **sector_count** -- number of sectors to erase
- **arg** -- erase command (CMD38) argument

Returns:

- ESP_OK on success or sector_count equal to 0
- One of the error codes from SDMMC host controller

esp_err_t `sdmmc_can_discard`(sdmmc_card_t *card)

Check if SD/MMC card supports discard

Parameters: **card** -- pointer to card information structure previously initialized using sdmmc_card_init

Returns:

- ESP_OK if supported by the card/device
- ESP_FAIL if not supported by the card/device

esp_err_t `sdmmc_can_trim`(sdmmc_card_t *card)

Check if SD/MMC card supports trim

Parameters: **card** -- pointer to card information structure previously initialized using sdmmc_card_init

Returns:

- ESP_OK if supported by the card/device
- ESP_FAIL if not supported by the card/device

esp_err_t `sdmmc_mmc_can_sanitize`(sdmmc_card_t *card)

Check if SD/MMC card supports sanitize

Parameters: **card** -- pointer to card information structure previously initialized using sdmmc_card_init

Returns:

- ESP_OK if supported by the card/device
- ESP_FAIL if not supported by the card/device

esp_err_t `sdmmc_mmc_sanitize`(sdmmc_card_t *card, uint32_t timeout_ms)

Sanitize the data that was unmapped by a Discard command

Note

Discard command has to precede sanitize operation. To discard, use MMC_DICARD_ARG with sdmmc_erase_sectors argument

Parameters:

- **card** -- pointer to card information structure previously initialized using sdmmc_card_init
- **timeout_ms** -- timeout value in milliseconds required to sanitize the selected range of sectors.

Returns:

- ESP_OK on success
- One of the error codes from SDMMC host controller

esp_err_t `sdmmc_full_erase`(sdmmc_card_t *card)

Erase complete SD/MMC card

Parameters: **card** -- pointer to card information structure previously initialized using sdmmc_card_init

Returns:

- ESP_OK on success
- One of the error codes from SDMMC host controller

esp_err_t **sdmmc_io_read_byte**(sdmmc_card_t *card, uint32_t function, uint32_t reg, uint8_t *out_byte)

Read one byte from an SDIO card using IO_RW_DIRECT (CMD52)

Parameters:

- **card** -- pointer to card information structure previously initialized using sdmmc_card_init
- **function** -- IO function number
- **reg** -- byte address within IO function
- **out_byte** -- **[out]** output, receives the value read from the card

Returns:

- ESP_OK on success
- One of the error codes from SDMMC host controller

esp_err_t **sdmmc_io_write_byte**(sdmmc_card_t *card, uint32_t function, uint32_t reg, uint8_t in_byte, uint8_t *out_byte)

Write one byte to an SDIO card using IO_RW_DIRECT (CMD52)

Parameters:

- **card** -- pointer to card information structure previously initialized using sdmmc_card_init
- **function** -- IO function number
- **reg** -- byte address within IO function
- **in_byte** -- value to be written
- **out_byte** -- **[out]** if not NULL, receives new byte value read from the card (read-after-write).

Returns:

- ESP_OK on success
- One of the error codes from SDMMC host controller

esp_err_t **sdmmc_io_read_bytes**(sdmmc_card_t *card, uint32_t function, uint32_t addr, void *dst, size_t size)

Read multiple bytes from an SDIO card using IO_RW_EXTENDED (CMD53)

This function performs read operation using CMD53 in byte mode. For block mode, see sdmmc_io_read_blocks.

Parameters:

- **card** -- pointer to card information structure previously initialized using sdmmc_card_init
- **function** -- IO function number
- **addr** -- byte address within IO function where reading starts
- **dst** -- buffer which receives the data read from card
- **size** -- number of bytes to read

Returns:

- ESP_OK on success
- ESP_ERR_INVALID_SIZE if size exceeds 512 bytes
- One of the error codes from SDMMC host controller

---

esp_err_t **`sdmmc_io_write_bytes`**(sdmmc_card_t *card, uint32_t function, uint32_t addr, *const* void *src, size_t size)

Write multiple bytes to an SDIO card using IO_RW_EXTENDED (CMD53)

This function performs write operation using CMD53 in byte mode. For block mode, see sdmmc_io_write_blocks.

Parameters:

- **card** -- pointer to card information structure previously initialized using sdmmc_card_init
- **function** -- IO function number
- **addr** -- byte address within IO function where writing starts
- **src** -- data to be written
- **size** -- number of bytes to write

Returns:

- ESP_OK on success
- ESP_ERR_INVALID_SIZE if size exceeds 512 bytes
- One of the error codes from SDMMC host controller

---

esp_err_t **`sdmmc_io_read_blocks`**(sdmmc_card_t *card, uint32_t function, uint32_t addr, void *dst, size_t size)

Read blocks of data from an SDIO card using IO_RW_EXTENDED (CMD53)

This function performs read operation using CMD53 in block mode. For byte mode, see sdmmc_io_read_bytes.

Parameters:

- **card** -- pointer to card information structure previously initialized using sdmmc_card_init
- **function** -- IO function number
- **addr** -- byte address within IO function where writing starts
- **dst** -- buffer which receives the data read from card
- **size** -- number of bytes to read, must be divisible by the card block size.

Returns:

- ESP_OK on success
- ESP_ERR_INVALID_SIZE if size is not divisible by 512 bytes
- One of the error codes from SDMMC host controller

esp_err_t **sdmmc_io_write_blocks**(sdmmc_card_t *card, uint32_t function, uint32_t addr, *const* void *src, size_t size)

Write blocks of data to an SDIO card using IO_RW_EXTENDED (CMD53)

This function performs write operation using CMD53 in block mode. For byte mode, see sdmmc_io_write_bytes.

Parameters:
- **card** -- pointer to card information structure previously initialized using sdmmc_card_init
- **function** -- IO function number
- **addr** -- byte address within IO function where writing starts
- **src** -- data to be written
- **size** -- number of bytes to read, must be divisible by the card block size.

Returns:
- ESP_OK on success
- ESP_ERR_INVALID_SIZE if size is not divisible by 512 bytes
- One of the error codes from SDMMC host controller

esp_err_t **sdmmc_io_enable_int**(sdmmc_card_t *card)

Enable SDIO interrupt in the SDMMC host

Parameters: **card** -- pointer to card information structure previously initialized using sdmmc_card_init

Returns:
- ESP_OK on success
- ESP_ERR_NOT_SUPPORTED if the host controller does not support IO interrupts

esp_err_t **sdmmc_io_wait_int**(sdmmc_card_t *card, TickType_t timeout_ticks)

Block until an SDIO interrupt is received

Slave uses D1 line to signal interrupt condition to the host. This function can be used to wait for the interrupt.

Parameters:
- **card** -- pointer to card information structure previously initialized using sdmmc_card_init
- **timeout_ticks** -- time to wait for the interrupt, in RTOS ticks

Returns:
- ESP_OK if the interrupt is received
- ESP_ERR_NOT_SUPPORTED if the host controller does not support IO interrupts
- ESP_ERR_TIMEOUT if the interrupt does not happen in timeout_ticks

esp_err_t **sdmmc_io_get_cis_data**(sdmmc_card_t *card, uint8_t *out_buffer, size_t buffer_size, size_t *inout_cis_size)

Get the data of CIS region of an SDIO card.

You may provide a buffer not sufficient to store all the CIS data. In this case, this function stores as much data into your buffer as possible. Also, this function will try to get and return the size required for you.

Parameters:

- **card** -- pointer to card information structure previously initialized using sdmmc_card_init
- **out_buffer** -- Output buffer of the CIS data
- **buffer_size** -- Size of the buffer.
- **inout_cis_size** -- Mandatory, pointer to a size, input and output.
  - input: Limitation of maximum searching range, should be 0 or larger than buffer_size. The function searches for CIS_CODE_END until this range. Set to 0 to search infinitely.
  - output: The size required to store all the CIS data, if CIS_CODE_END is found.

Returns:

- ESP_OK: on success
- ESP_ERR_INVALID_RESPONSE: if the card does not (correctly) support CIS.
- ESP_ERR_INVALID_SIZE: CIS_CODE_END found, but buffer_size is less than required size, which is stored in the inout_cis_size then.
- ESP_ERR_NOT_FOUND: if the CIS_CODE_END not found. Increase input value of inout_cis_size or set it to 0, if you still want to search for the end; output value of inout_cis_size is invalid in this case.
- and other error code return from sdmmc_io_read_bytes

esp_err_t **sdmmc_io_print_cis_info**(uint8_t *buffer, size_t buffer_size, FILE *fp)

Parse and print the CIS information of an SDIO card.

Note

Not all the CIS codes and all kinds of tuples are supported. If you see some unresolved code, you can add the parsing of these code in sdmmc_io.c and contribute to the IDF through the Github repository.

```
        using sdmmc_card_init
```

Parameters:

- **buffer** -- Buffer to parse
- **buffer_size** -- Size of the buffer.
- **fp** -- File pointer to print to, set to NULL to print to stdout.

Returns:

- ESP_OK: on success
- ESP_ERR_NOT_SUPPORTED: if the value from the card is not supported to be parsed.
- ESP_ERR_INVALID_SIZE: if the CIS size fields are not correct.

Header File

- components/driver/sdmmc/include/driver/sdmmc_types.h
- This header file can be included with:

```
#include "driver/sdmmc_types.h"
```

- This header file is a part of the API provided by the `driver` component. To declare that your component depends on `driver`, add the following to your CMakeLists.txt:

```
REQUIRES driver
```

or

```
PRIV_REQUIRES driver
```

Structures

struct **sdmmc_csd_t**

Decoded values from SD card Card Specific Data register

Public Members

int **csd_ver**

CSD structure format

int **mmc_ver**

MMC version (for CID format)

int **`capacity`**

total number of sectors

int **`sector_size`**

sector size in bytes

int **`read_block_len`**

block length for reads

int **`card_command_class`**

Card Command Class for SD

int **`tr_speed`**

Max transfer speed

struct **`sdmmc_cid_t`**

Decoded values from SD card Card IDentification register

Public Members

int **`mfg_id`**

manufacturer identification number

int **`oem_id`**

OEM/product identification number

char **`name`**[8]

product name (MMC v1 has the longest)

int **`revision`**

product revision

int **`serial`**

product serial number

int **`date`**

manufacturing date

struct **`sdmmc_scr_t`**

Decoded values from SD Configuration Register Note: When new member is added, update reserved bits accordingly

Public Members

uint32_t **`sd_spec`**

SD Physical layer specification version, reported by card

uint32_t **`erase_mem_state`**

data state on card after erase whether 0 or 1 (card vendor dependent)

uint32_t **`bus_width`**

bus widths supported by card: BIT(0) — 1-bit bus, BIT(2) — 4-bit bus

uint32_t **`reserved`**

reserved for future expansion

uint32_t **`rsvd_mnf`**

reserved for manufacturer usage

struct **`sdmmc_ssr_t`**

Decoded values from SD Status Register Note: When new member is added, update reserved bits accordingly

Public Members

uint32_t **`alloc_unit_kb`**

Allocation unit of the card, in multiples of kB (1024 bytes)

uint32_t **`erase_size_au`**

Erase size for the purpose of timeout calculation, in multiples of allocation unit

uint32_t **`cur_bus_width`**

SD current bus width

uint32_t **discard_support**

SD discard feature support

uint32_t **fule_support**

SD FULE (Full User Area Logical Erase) feature support

uint32_t **erase_timeout**

Timeout (in seconds) for erase of a single allocation unit

uint32_t **erase_offset**

Constant timeout offset (in seconds) for any erase operation

uint32_t **reserved**

reserved for future expansion

struct **sdmmc_ext_csd_t**

Decoded values of Extended Card Specific Data

Public Members

uint8_t **rev**

Extended CSD Revision

uint8_t **power_class**

Power class used by the card

uint8_t **erase_mem_state**

data state on card after erase whether 0 or 1 (card vendor dependent)

uint8_t **sec_feature**

secure data management features supported by the card

struct **sdmmc_switch_func_rsp_t**

SD SWITCH_FUNC response buffer

Public Members

uint32_t **data**[512 / 8 / *sizeof*(uint32_t)] response data

struct **`sdmmc_command_t`**

SD/MMC command information

Public Members

uint32_t **`opcode`**

SD or MMC command index

uint32_t **`arg`**

SD/MMC command argument

sdmmc_response_t **`response`**

response buffer

void ***`data`**

buffer to send or read into

size_t **`datalen`**

length of data in the buffer

size_t **`buflen`**

length of the buffer

size_t **`blklen`**

block length

int **`flags`**

see below

esp_err_t **`error`**

error returned from transfer

uint32_t **`timeout_ms`**

response timeout, in milliseconds

struct **`sdmmc_host_t`**

SD/MMC Host description

This structure defines properties of SD/MMC host and functions of SD/MMC host which can be used by upper layers.

Public Members

uint32_t **`flags`**

flags defining host properties

int **`slot`**

slot number, to be passed to host functions

int **`max_freq_khz`**

max frequency supported by the host

float **`io_voltage`**

I/O voltage used by the controller (voltage switching is not supported)

esp_err_t (***`init`**)(void)

Host function to initialize the driver

esp_err_t (***`set_bus_width`**)(int slot, size_t width)

host function to set bus width

size_t (***`get_bus_width`**)(int slot)

host function to get bus width

esp_err_t (***`set_bus_ddr_mode`**)(int slot, bool ddr_enable)

host function to set DDR mode

esp_err_t (***`set_card_clk`**)(int slot, uint32_t freq_khz)

host function to set card clock frequency

esp_err_t (***`set_cclk_always_on`**)(int slot, bool cclk_always_on)

host function to set whether the clock is always enabled esp_err_t (*do_transaction)(int slot, sdmmc_command_t *cmdinfo)

host function to do a transaction

esp_err_t (*deinit)(void)

host function to deinitialize the driver

esp_err_t (*deinit_p)(int slot)

host function to deinitialize the driver, called with the `slot`

esp_err_t (*io_int_enable)(int slot)

Host function to enable SDIO interrupt line

esp_err_t (*io_int_wait)(int slot, TickType_t timeout_ticks)

Host function to wait for SDIO interrupt line to be active

int command_timeout_ms

timeout, in milliseconds, of a single command. Set to 0 to use the default value.

esp_err_t (*get_real_freq)(int slot, int *real_freq)

Host function to provide real working freq, based on SDMMC controller setup

sdmmc_delay_phase_t input_delay_phase

input delay phase, this will only take into effect when the host works in SDMMC_FREQ_HIGHSPEED or SDMMC_FREQ_52M. Driver will print out how long the delay is

esp_err_t (*set_input_delay)(int slot, sdmmc_delay_phase_t delay_phase)

set input delay phase

struct sdmmc_card_t

SD/MMC card information structure

Public Members

sdmmc_host_t host

Host with which the card is associated

uint32_t **`ocr`**

OCR (Operation Conditions Register) value

sdmmc_cid_t **`cid`**

decoded CID (Card IDentification) register value

sdmmc_response_t **`raw_cid`**

raw CID of MMC card to be decoded after the CSD is fetched in the data transfer mode

sdmmc_csd_t **`csd`**

decoded CSD (Card-Specific Data) register value

sdmmc_scr_t **`scr`**

decoded SCR (SD card Configuration Register) value

sdmmc_ssr_t **`ssr`**

decoded SSR (SD Status Register) value

sdmmc_ext_csd_t **`ext_csd`**

decoded EXT_CSD (Extended Card Specific Data) register value

uint16_t **`rca`**

RCA (Relative Card Address)

uint16_t **`max_freq_khz`**

Maximum frequency, in kHz, supported by the card

int **`real_freq_khz`**

Real working frequency, in kHz, configured on the host controller

uint32_t **`is_mem`**

Bit indicates if the card is a memory card

uint32_t **`is_sdio`**

Bit indicates if the card is an IO card

uint32_t **`is_mmc`**

Bit indicates if the card is MMC

uint32_t **`num_io_functions`**

If is_sdio is 1, contains the number of IO functions on the card

uint32_t **`log_bus_width`**

log2(bus width supported by card)

uint32_t **`is_ddr`**

Card supports DDR mode

uint32_t **`reserved`**

Reserved for future expansion

Macros

`SDMMC_HOST_FLAG_1BIT`

host supports 1-line SD and MMC protocol

`SDMMC_HOST_FLAG_4BIT`

host supports 4-line SD and MMC protocol

`SDMMC_HOST_FLAG_8BIT`

host supports 8-line MMC protocol

`SDMMC_HOST_FLAG_SPI`

host supports SPI protocol

`SDMMC_HOST_FLAG_DDR`

host supports DDR mode for SD/MMC

`SDMMC_HOST_FLAG_DEINIT_ARG`

host `deinit` function called with the slot argument

`SDMMC_FREQ_DEFAULT`

SD/MMC Default speed (limited by clock divider)

`SDMMC_FREQ_HIGHSPEED`

SD High speed (limited by clock divider)

`SDMMC_FREQ_PROBING`

SD/MMC probing speed

`SDMMC_FREQ_52M`

MMC 52MHz speed

`SDMMC_FREQ_26M`

MMC 26MHz speed

Type Definitions

typedef uint32_t **`sdmmc_response_t`**[4]

SD/MMC command response buffer

Enumerations

enum **`sdmmc_delay_phase_t`**

SD/MMC Host clock timing delay phases

This will only take effect when the host works in SDMMC_FREQ_HIGHSPEED or SDMMC_FREQ_52M. Driver will print out how long the delay is, in picosecond (ps).

Values:

enumerator **`SDMMC_DELAY_PHASE_0`**

Delay phase 0

enumerator **`SDMMC_DELAY_PHASE_1`**

Delay phase 1

enumerator **`SDMMC_DELAY_PHASE_2`**

Delay phase 2

enumerator **`SDMMC_DELAY_PHASE_3`**

Delay phase 3

enum `sdmmc_erase_arg_t`

SD/MMC erase command(38) arguments SD: ERASE: Erase the write blocks, physical/hard erase.

DISCARD: Card may deallocate the discarded blocks partially or completely. After discard operation the previously written data may be partially or fully read by the host depending on card implementation.

MMC: ERASE: Does TRIM, applies erase operation to write blocks instead of Erase Group.

DISCARD: The Discard function allows the host to identify data that is no longer required so that the device can erase the data if necessary during background erase events. Applies to write blocks instead of Erase Group After discard operation, the original data may be remained partially or fully accessible to the host dependent on device.

Values:

enumerator **`SDMMC_ERASE_ARG`**

Erase operation on SD, Trim operation on MMC

enumerator **`SDMMC_DISCARD_ARG`**

Discard operation for SD/MMC

Provide feedback about this document