Note

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

SDMMC Host Driver

[中文]

Overview

ESP32's SDMMC host peripheral has two slots. Each slot can be used independently to connect to an SD card, SDIO device, or eMMC chip.

- Slot 0 (`SDMMC_HOST_SLOT_0`) is an 8-bit slot. It uses `HS1_*` signals in the PIN MUX.
- Slot 1 (`SDMMC_HOST_SLOT_1`) is a 4-bit slot. It uses `HS2_*` signals in the PIN MUX.

The slots are connected to ESP32 GPIOs using IO MUX. Pin mappings of these slots are given in the table below.

Signal	Slot 0	Slot 1
CMD	GPIO11	GPIO15
CLK	GPIO6	GPIO14
D0	GPIO7	GPIO2
D1	GPIO8	GPIO4
D2	GPIO9	GPIO12
D3	GPIO10	GPIO13
D4	GPIO16	
D5	GPIO17	
D6	GPIO5	
D7	GPIO18	
CD	any input via GPIO matrix	any input via GPIO matrix

Signal	Slot 0	Slot 1
WP	any input via GPIO matrix	any input via GPIO matrix

The Card Detect (CD) and Write Protect (WP) signals can be routed to arbitrary pins using the GPIO matrix. To reserve the pins, set the `cd` and `wp` members of the `sdmmc_slot_config_t` structure before calling `sdmmc_host_init_slot()`. Please note that it is not advised to specify a CD pin when working with SDIO cards, because the CD signal in ESP32 can also trigger SDIO slave interrupt.

> **Warning**
>
> Pins used by Slot 0 (`HS1_*`) are also used to connect the SPI flash chip in ESP32-WROOM and ESP32-WROVER modules. These pins cannot be concurrently shared between an SD card and an SPI flash. If you need to use Slot 0, establish an alternative connection for the SPI flash using different pins and configure the necessary eFuses accordingly.

Supported Speed Modes

SDMMC Host driver supports the following speed modes:

- Default Speed (20 MHz): 1-line or 4-line with SD cards, and 1-line, 4-line, or 8-line with 3.3 V eMMC
- High Speed (40 MHz): 1-line or 4-line with SD cards, and 1-line, 4-line, or 8-line with 3.3 V eMMC
- High Speed DDR (40 MHz): 4-line with 3.3 V eMMC

Speed modes not supported at present:

- High Speed DDR mode: 8-line eMMC
- UHS-I 1.8 V modes: 4-line SD cards

Using the SDMMC Host Driver

Of all the functions listed below, only the following ones will be used directly by most applications:

- `sdmmc_host_init()`
- `sdmmc_host_init_slot()`
- `sdmmc_host_deinit()`

Other functions, such as the ones given below, will be called by the SD/MMC protocol layer via function pointers in the `sdmmc_host_t` structure:

- `sdmmc_host_set_bus_width()`
- `sdmmc_host_set_card_clk()`
- `sdmmc_host_do_transaction()`

Configuring Bus Width and Frequency

With the default initializers for `sdmmc_host_t` and `sdmmc_slot_config_t`, i.e., `SDMMC_HOST_DEFAULT` and `SDMMC_SLOT_CONFIG_DEFAULT`, SDMMC Host driver will attempt to use the widest bus supported by the card (4 lines for SD, 8 lines for eMMC) and the frequency of 20 MHz.

In the designs where communication at 40 MHz frequency can be achieved, it is possible to increase the bus frequency by changing the `max_freq_khz` field of `sdmmc_host_t`:

```
sdmmc_host_t host = SDMMC_HOST_DEFAULT();
host.max_freq_khz = SDMMC_FREQ_HIGHSPEED;
```

If you need a specific frequency other than standard speeds, you are free to use any value from within an appropriate range of the SD interface given (SDMMC or SDSPI). However, the real clock frequency shall be calculated by the underlying driver and the value can be different from the one required.

For the SDMMC, `max_freq_khz` works as the upper limit so the final frequency value shall be always lower or equal. For the SDSPI, the nearest fitting frequency is supplied and thus the value can be greater than/equal to/lower than `max_freq_khz`.

To configure the bus width, set the `width` field of `sdmmc_slot_config_t`. For example, to set 1-line mode:

```
sdmmc_slot_config_t slot = SDMMC_SLOT_CONFIG_DEFAULT();
slot.width = 1;
```

DDR Mode for eMMC Chips

By default, DDR mode will be used if:

- SDMMC host frequency is set to `SDMMC_FREQ_HIGHSPEED` in `sdmmc_host_t` structure, and

- eMMC chip reports DDR mode support in its CSD register

DDR mode places higher requirements for signal integrity. To disable DDR mode while keeping the `SDMMC_FREQ_HIGHSPEED` frequency, clear the `SDMMC_HOST_FLAG_DDR` bit in `sdmmc_host_t::flags` field of the `sdmmc_host_t`:

```
sdmmc_host_t host = SDMMC_HOST_DEFAULT();
host.max_freq_khz = SDMMC_FREQ_HIGHSPEED;
host.flags &= ~SDMMC_HOST_FLAG_DDR;
```

See also

- SD/SDIO/MMC Driver: introduces the higher-level driver which implements the protocol layer.
- SD SPI Host Driver: introduces a similar driver that uses the SPI controller and is limited to SD protocol's SPI mode.
- SD Pull-up Requirements: introduces pull-up support and compatibilities of modules and development kits.

API Reference

Header File

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

  ```
  #include "driver/sdmmc_host.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
  ```

Functions

esp_err_t **sdmmc_host_init**(void)

Initialize SDMMC host peripheral.

Note

This function is not thread safe

Returns:

- ESP_OK on success
- ESP_ERR_INVALID_STATE if sdmmc_host_init was already called
- ESP_ERR_NO_MEM if memory can not be allocated

esp_err_t **sdmmc_host_init_slot**(int slot, *const* sdmmc_slot_config_t *slot_config)

Initialize given slot of SDMMC peripheral.

On the ESP32, SDMMC peripheral has two slots:

- Slot 0: 8-bit wide, maps to HS1_* signals in PIN MUX
- Slot 1: 4-bit wide, maps to HS2_* signals in PIN MUX

Card detect and write protect signals can be routed to arbitrary GPIOs using GPIO matrix.

Note

This function is not thread safe

Parameters:

- **slot** -- slot number (SDMMC_HOST_SLOT_0 or SDMMC_HOST_SLOT_1)
- **slot_config** -- additional configuration for the slot

Returns:

- ESP_OK on success
- ESP_ERR_INVALID_STATE if host has not been initialized using sdmmc_host_init

esp_err_t **sdmmc_host_set_bus_width**(int slot, size_t width)

Select bus width to be used for data transfer.

SD/MMC card must be initialized prior to this command, and a command to set bus width has to be sent to the card (e.g. SD_APP_SET_BUS_WIDTH)

Note

This function is not thread safe

Parameters:
- **slot** -- slot number (SDMMC_HOST_SLOT_0 or SDMMC_HOST_SLOT_1)
- **width** -- bus width (1, 4, or 8 for slot 0; 1 or 4 for slot 1)

Returns:
- ESP_OK on success
- ESP_ERR_INVALID_ARG if slot number or width is not valid

---

size_t **sdmmc_host_get_slot_width**(int slot)

Get bus width configured in `sdmmc_host_init_slot` to be used for data transfer.

Parameters: **slot** -- slot number (SDMMC_HOST_SLOT_0 or SDMMC_HOST_SLOT_1)

Returns: configured bus width of the specified slot.

---

esp_err_t **sdmmc_host_set_card_clk**(int slot, uint32_t freq_khz)

Set card clock frequency.

Currently only integer fractions of 40MHz clock can be used. For High Speed cards, 40MHz can be used. For Default Speed cards, 20MHz can be used.

Note

This function is not thread safe

Parameters:
- **slot** -- slot number (SDMMC_HOST_SLOT_0 or SDMMC_HOST_SLOT_1)
- **freq_khz** -- card clock frequency, in kHz

Returns:
- ESP_OK on success
- other error codes may be returned in the future

---

esp_err_t **sdmmc_host_set_bus_ddr_mode**(int slot, bool ddr_enabled)

Enable or disable DDR mode of SD interface.

Parameters:
- **slot** -- slot number (SDMMC_HOST_SLOT_0 or SDMMC_HOST_SLOT_1)
- **ddr_enabled** -- enable or disable DDR mode

Returns:

- ESP_OK on success
- ESP_ERR_NOT_SUPPORTED if DDR mode is not supported on this slot

---

esp_err_t **sdmmc_host_set_cclk_always_on**(int slot, bool cclk_always_on)

Enable or disable always-on card clock When cclk_always_on is false, the host controller is allowed to shut down the card clock between the commands. When cclk_always_on is true, the clock is generated even if no command is in progress.

Parameters:

- **slot** -- slot number
- **cclk_always_on** -- enable or disable always-on clock

Returns:

- ESP_OK on success
- ESP_ERR_INVALID_ARG if the slot number is invalid

---

esp_err_t **sdmmc_host_do_transaction**(int slot, sdmmc_command_t *cmdinfo)

Send command to the card and get response.

This function returns when command is sent and response is received, or data is transferred, or timeout occurs.

Attention

Data buffer passed in cmdinfo->data must be in DMA capable memory

Note

This function is not thread safe w.r.t. init/deinit functions, and bus width/clock speed configuration functions. Multiple tasks can call sdmmc_host_do_transaction as long as other sdmmc_host_* functions are not called.

Parameters:

- **slot** -- slot number (SDMMC_HOST_SLOT_0 or SDMMC_HOST_SLOT_1)
- **cmdinfo** -- pointer to structure describing command and data to transfer

Returns:

- ESP_OK on success
- ESP_ERR_TIMEOUT if response or data transfer has timed out
- ESP_ERR_INVALID_CRC if response or data transfer CRC check has failed
- ESP_ERR_INVALID_RESPONSE if the card has sent an invalid response

- ESP_ERR_INVALID_SIZE if the size of data transfer is not valid in SD protocol
- ESP_ERR_INVALID_ARG if the data buffer is not in DMA capable memory

---

esp_err_t **sdmmc_host_io_int_enable**(int slot)

Enable IO interrupts.

This function configures the host to accept SDIO interrupts.

Parameters: **slot** -- slot number (SDMMC_HOST_SLOT_0 or SDMMC_HOST_SLOT_1)

Returns: returns ESP_OK, other errors possible in the future

---

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

Block until an SDIO interrupt is received, or timeout occurs.

Parameters:
- **slot** -- slot number (SDMMC_HOST_SLOT_0 or SDMMC_HOST_SLOT_1)
- **timeout_ticks** -- number of RTOS ticks to wait for the interrupt

Returns:
- ESP_OK on success (interrupt received)
- ESP_ERR_TIMEOUT if the interrupt did not occur within timeout_ticks

---

esp_err_t **sdmmc_host_deinit**(void)

Disable SDMMC host and release allocated resources.

Note

This function is not thread safe

Returns:
- ESP_OK on success
- ESP_ERR_INVALID_STATE if sdmmc_host_init function has not been called

---

esp_err_t **sdmmc_host_get_real_freq**(int slot, int *real_freq_khz)

Provides a real frequency used for an SD card installed on specific slot of SD/MMC host controller.

This function calculates real working frequency given by current SD/MMC host controller setup for required slot: it reads associated host and card dividers from corresponding SDMMC registers, calculates respective frequency and stores the value into the 'real_freq_khz' parameter

Parameters:
- **slot** -- slot number (SDMMC_HOST_SLOT_0 or SDMMC_HOST_SLOT_1)
- **real_freq_khz** -- **[out]** output parameter for the result frequency (in kHz)

Returns:
- ESP_OK on success
- ESP_ERR_INVALID_ARG on real_freq_khz == NULL or invalid slot number used

---

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

set input delay

- This API sets delay when the SDMMC Host samples the signal from the SD Slave.
- This API will check if the given `delay_phase` is valid or not.
- This API will print out the delay time, in picosecond (ps)

Note

ESP32 doesn't support this feature, you will get an `ESP_ERR_NOT_SUPPORTED`

Parameters:
- **slot** -- slot number (SDMMC_HOST_SLOT_0 or SDMMC_HOST_SLOT_1)
- **delay_phase** -- delay phase, this API will convert the phase into picoseconds and print it out

Returns:
- ESP_OK: ON success.
- ESP_ERR_INVALID_ARG: Invalid argument.
- ESP_ERR_NOT_SUPPORTED: ESP32 doesn't support this feature.

Structures

---

struct **sdmmc_slot_config_t**

Extra configuration for SDMMC peripheral slot

Public Members

> gpio_num_t **gpio_cd**

GPIO number of card detect signal.

gpio_num_t **cd**

GPIO number of card detect signal; shorter name.

gpio_num_t **gpio_wp**

GPIO number of write protect signal.

gpio_num_t **wp**

GPIO number of write protect signal; shorter name.

uint8_t **width**

Bus width used by the slot (might be less than the max width supported)

uint32_t **flags**

Features used by this slot.

Macros

SDMMC_SLOT_FLAG_INTERNAL_PULLUP

Enable internal pullups on enabled pins. The internal pullups are insufficient however, please make sure external pullups are connected on the bus. This is for debug / example purpose only.

SDMMC_SLOT_FLAG_WP_ACTIVE_HIGH

GPIO write protect polarity. 0 means "active low", i.e. card is protected when the GPIO is low; 1 means "active high", i.e. card is protected when GPIO is high.