MT793X IoT SDK for SDIO Master User Guide

Version: 1.0
Release date: 2021-07-20

Version History

Version	Date	Description
1.0	2021-07-20	Official release

Table of Contents

1 Getting Started

This chapter introduces the MT7933 FreeRTOS project and gives you an idea of what you need to prepare to get started.

1.1 Overview

The Secure Digital Input/Output (SDIO) card is based on and compatible with the SD (Secure Digital) memory card. The controller fully supports the SD memory card bus protocol as defined in SD Memory Card Specification Part 1 Physical Layer Specification version 2.0 and SDIO Specification version 2.0.
SDIO provides high-speed data IO with low power consumption. The MT7933 SDIO Master module provides an SDIO2.0 card interface connected to the host and can support multiple speed modes including default speed and High Speed mode.

1.2 Features

Provides SDIO2.0 host interfaces

- SDIO2.0:
  - 1-bit and 4-bit SD data transfer modes
  - Default mode: Variable clock rate 0-25 MHz, up to 12.5 Mbps interface speed (using 4 parallel data lines)
  - High-Speed mode: Variable clock rate 0-50 MHz, up to 25 Mbps interface speed (using 4 data lines)
  - Data rate up to 50 Mbps in serial mode, 50 x 4 Mbps in parallel mode. The module is targeted at 50 MHz operating clock.
  - 32-bit access for control registers
  - 32-bit access for FIFO
  - Built-in 32 bytes FIFO buffers for transmit and receive. FIFO is shared for transmit and receive
  - Built-in CRC circuit
  - Interrupt capabilities
  - Does not support SPI
  - Supports DMA

1.3 Code Layout

The section provides the location of the SDIO master driver as below

1. **Common header file**
   driver/chip/inc/hal_sdio.h
2. **Internal header files**
   driver/chip/mt7933/inc/hal_sd_define.h
   driver/chip/mt7933/inc/hal_mtk_sdio.h

driver/chip/mt7933/inc/hal_msdc.h

3. **Src files**

   driver/chip/mt7933/src/hal_sdio.c

   driver/chip/mt7933/src/hal_mtk_sdio.c

   driver/chip/mt7933/src/hal_msdc.c

1.4 SDIO Master APIs

The SDIO master provides some APIs for upper layer user to communication with the SDIO slave.

✓ hal_sdio_init

```
hal_sdio_status_t hal_sdio_init ( hal_sdio_port_t      sdio_port,
                                  hal_sdio_config_t *  sdio_config
                                )
```

This function initializes the MSDC hardware and SDIO slave.

It can also be used to set the MSDC pin output clock and bus width.

Parameters

[in] **sdio_port** is the initialization configuration port. For more details about this parameter, please refer to **hal_sdio_port_t**.

[in] **sdio_config** is the initialization configuration parameter. For more details about this parameter, please refer to **hal_sdio_config_t**.

Returns

Indicates whether this function call is successful or not. If the return value is **HAL_SDIO_STATUS_OK**, the operation completed successfully. If the return value is **HAL_SDIO_STATUS_ERROR**, an initialization error occurred. If the return value is **HAL_SDIO_STATUS_BUSY**, the MSDC is busy.

See also

hal_sdio_deinit()

✓ hal_sdio_deinit

```
hal_sdio_status_t hal_sdio_deinit ( hal_sdio_port_t  sdio_port )
```

This function deinitializes the MSDC and the SDIO slave settings.

Parameters

[in] **sdio_port** is the MSDC deinitialization port.

Returns

If the return value is **HAL_SDIO_STATUS_OK**, the operation completed successfully.

See also

hal_sdio_init()

✓ hal_sdio_execute_command52

```
hal_sdio_status_t hal_sdio_execute_command52 ( hal_sdio_port_t                  sdio_port,
                                               hal_sdio_command52_config_t *    command52_config
                                             )
```

This function reads from or writes to the SDIO slave with COMMAND52.

Parameters

[in] **sdio_port** is the MSDC port to read or write.

[in] **command52_config** is the configuration parameter pointer of the COMMAND52.

Returns

If the return value is **HAL_SDIO_STATUS_OK**, the operation completed successfully. If the return value is **HAL_SDIO_STATUS_ERROR**, an error occurred, such as a wrong parameter is given. If the return value is **HAL_SDIO_STATUS_BUSY**, the MSDC is busy.

✓ Hal_sdio_execute_command53

```
hal_sdio_status_t hal_sdio_execute_command53 ( hal_sdio_port_t                  sdio_port,
                                               hal_sdio_command53_config_t *    command53_config
                                             )
```

This function reads from or writes to the SDIO slave with COMMAND53 MCU mode.

Parameters

[in] **sdio_port** is the MSDC port to read.

[in] **command53_config** is the configuration parameter pointer of the COMMAND53.

Returns

If the return value is **HAL_SDIO_STATUS_OK**, the operation completed successfully. If the return value is **HAL_SDIO_STATUS_ERROR**, an error occurred, such as a wrong parameter is given. If the return value is **HAL_SDIO_STATUS_BUSY**, the MSDC is busy.

✓ hal_sdio_execute_command53_dma

```
hal_sdio_status_t hal_sdio_execute_command53_dma ( hal_sdio_port_t                  sdio_port,
                                                   hal_sdio_command53_config_t *    command53_config
                                                 )
```

This function reads from or writes to the SDIO slave with COMMAND53 DMA interrupt mode.

This API would not block the application task.

Parameters

[in] **sdio_port** is the MSDC port to read.

[in] **command53_config** is the configuration parameter pointer of the COMMAND53.

Returns

If the return value is **HAL_SDIO_STATUS_OK**, the operation completed successfully. If the return value is **HAL_SDIO_STATUS_ERROR**, an error occurred, such as a wrong parameter is given. If the return value is **HAL_SDIO_STATUS_BUSY**, the MSDC is busy.

✓ hal_sdio_execute_command53_dma_blocking

```
hal_sdio_status_t hal_sdio_execute_command53_dma_blocking ( hal_sdio_port_t                 sdio_port,
                                                            hal_sdio_command53_config_t *  command53_config
                                                          )
```

This function reads from or writes to the SDIO slave with COMMAND53 DMA interrupt mode.

This API would block the application task.

Parameters

[in] **sdio_port** is the MSDC port to read.

[in] **command53_config** is the configuration parameter pointer of the COMMAND53.

Returns

If the return value is **HAL_SDIO_STATUS_OK**, the operation completed successfully. If the return value is **HAL_SDIO_STATUS_ERROR**, an error occurred, such as a wrong parameter is given. If the return value is **HAL_SDIO_STATUS_BUSY**, the MSDC is busy.

✓ hal_sdio_set_block_size

```
hal_sdio_status_t hal_sdio_set_block_size ( hal_sdio_port_t         sdio_port,
                                            hal_sdio_function_id_t  function,
                                            uint32_t                block_size
                                          )
```

This function sets the transaction block size of the MSDC.

Parameters

[in] **sdio_port** is the MSDC port to set.

[in] **function** is the SDIO function to set block size.

[in] **block_size** is the SDIO transaction block size.

Returns

If the return value is **HAL_SDIO_STATUS_OK**, the operation completed successfully. If the return value is **HAL_SDIO_STATUS_ERROR**, an error occurred, such as a wrong parameter is given. If the return value is **HAL_SDIO_STATUS_BUSY**, the MSDC is busy.

✓ hal_sdio_get_block_size

```
hal_sdio_status_t hal_sdio_get_block_size ( hal_sdio_port_t          sdio_port,
                                            hal_sdio_function_id_t   function,
                                            uint32_t *               block_size
                                          )
```

This function gets the transaction block size of the MSDC.

Parameters

[in] **sdio_port** is the MSDC port to get block size.

[in] **function** is the SDIO function to get block size.

[out] **block_size** is the SDIO transaction block size.

Returns

If the return value is **HAL_SDIO_STATUS_OK**, the operation completed successfully. If the return value is **HAL_SDIO_STATUS_ERROR**, an error occurred, such as a wrong parameter is given.

✓ hal_sdio_set_clock

```
hal_sdio_status_t hal_sdio_set_clock ( hal_sdio_port_t  sdio_port,
                                       uint32_t         clock
                                     )
```

This function sets the output clock of the MSDC.

Parameters

[in] **sdio_port** is the MSDC port to set clock.

[in] **clock** is the expected output clock of the MSDC.

Returns

If the return value is **HAL_SDIO_STATUS_OK**, the operation completed successfully. If the return value is **HAL_SDIO_STATUS_ERROR**, an error occurred, such as a wrong parameter is given. If the return value is **HAL_SDIO_STATUS_BUSY**, the MSDC is busy.

✓ hal_sdio_get_clock

```
hal_sdio_status_t hal_sdio_get_clock ( hal_sdio_port_t  sdio_port,
                                       uint32_t *       clock
                                     )
```

This function gets the output clock of the MSDC.

Parameters

[in] **sdio_port** is the MSDC port to get clock.

[out] **clock** is the current output clock of the MSDC.

Returns

If the return value is **HAL_SDIO_STATUS_OK**, the operation completed successfully. If the return value is **HAL_SDIO_STATUS_ERROR**, an error occurred, such as a wrong parameter is given.

✓ hal_sdio_register_callback

```
hal_sdio_status_t hal_sdio_register_callback ( hal_sdio_port_t       sdio_port,
                                               hal_sdio_callback_t  sdio_callback,
                                               void *               user_data
                                             )
```

This function registers a callback function to inform the user the transfer is complete.

Parameters

[in] **sdio_port** is the MSDC port to transfer data.

[in] **sdio_callback** is the function pointer of the callback. The callback function is called once the SDIO data transfer is complete.

[in] **user_data** is the callback parameter.

Returns

HAL_SDIO_STATUS_OK, if the operation completed successfully.

✓ hal_sdio_set_bus_width

```
hal_sdio_status_t hal_sdio_set_bus_width ( hal_sdio_port_t        sdio_port,
                                           hal_sdio_bus_width_t   bus_width
                                         )
```

This function sets the bus widths for the MSDC and SDIO slave.

Parameters

[in] **sdio_port** is the MSDC port to set.

[in] **bus_width** is the SDIO bus width.

Returns

If the return value is **HAL_SDIO_STATUS_OK**, the operation completed successfully. If the return value is **HAL_SDIO_STATUS_ERROR**, an error occurred, such as a wrong parameter is given. If the return value is **HAL_SDIO_STATUS_BUSY**, the MSDC is busy.

2 SDIO Master Sample Use Case

How to use SDIO master driver

- Read from or write to the SDIO slave with COMMAND53 in the MCU mode

- Step 1. Call **hal_sdio_init()** to initialize the MSDC and the SDIO slave to transfer states.
- Step 2. Call **hal_sdio_execute_command53()** to read data from or write data to the SDIO card.
- Read sample code:

```
hal_sdio_status_t ret;
hal_sdio_command53_config_t config;
hal_sdio_config_t sdio_config = {HAL_SDIO_BUS_WIDTH_4, 25000};
ret = hal_sdio_init(HAL_SDIO_PORT_0,&sdio_config);
if (HAL_SDIO_STATUS_OK != ret) {
    // Error handler.
}

config.direction = HAL_SDIO_DIRECTION_READ;
config.function = HAL_SDIO_FUNCTION_0;
config.is_block = true;
config.count = 2;
config.address = address;
config.buffer = buffer;
ret = hal_sdio_execute_command53(HAL_SDIO_PORT_0, &config);
if (HAL_SDIO_STATUS_OK != ret) {
    // Error handler.
}
```

- Write sample code:

```
hal_sdio_status_t ret;
hal_sdio_command53_config_t config;
hal_sdio_config_t sdio_config = {HAL_SDIO_BUS_WIDTH_4, 25000};
ret = hal_sdio_init(HAL_SDIO_PORT_0,&sdio_config);
if (HAL_SDIO_STATUS_OK != ret) {
    // Error handler.
}

config.direction = HAL_SDIO_DIRECTION_WRITE;
config.function = HAL_SDIO_FUNCTION_0;
config.is_block = true;
config.count = 2;
config.address = address;
config.buffer = buffer;
ret = hal_sdio_execute_command53(HAL_SDIO_PORT_0, &config);
if (HAL_SDIO_STATUS_OK != ret) {
    // Error handler.
}
```

- Read from or write to the SDIO slave with COMMAND53 in the DMA blocking mode

- Step 1. Call **hal_sdio_init()** to initialize the MSDC and the SDIO slave to transfer states.
- Step 2. Call **hal_sdio_execute_command53_dma_blocking()** to read data from or write data to the SDIO card.
- Read sample code:

```
hal_sdio_status_t ret;
hal_sdio_command53_config_t config;
hal_sdio_config_t sdio_config = {HAL_SDIO_BUS_WIDTH_4, 25000};
ret = hal_sdio_init(HAL_SDIO_PORT_0,&sdio_config);
if (HAL_SDIO_STATUS_OK != ret) {
   // Error handler.
}

config.direction = HAL_SDIO_DIRECTION_READ;
config.function = HAL_SDIO_FUNCTION_0;
config.is_block = true;
config.count = 2;
config.address = address;
config.buffer = buffer;
ret = hal_sdio_execute_command53_dma_blocking(HAL_SDIO_PORT_0, &config);
if (HAL_SDIO_STATUS_OK != ret) {
    // Error handler.
}
```

- Write sample code:

```
hal_sdio_status_t ret;
hal_sdio_command53_config_t config;
hal_sdio_config_t sdio_config = {HAL_SDIO_BUS_WIDTH_4, 25000};
ret = hal_sdio_init(HAL_SDIO_PORT_0,&sdio_config);
if (HAL_SDIO_STATUS_OK != ret) {
     // Error handler.
}

config.direction = HAL_SDIO_DIRECTION_WRITE;
config.function = HAL_SDIO_FUNCTION_0;
config.is_block = true;
config.count = 2;
config.address = address;
config.buffer = buffer;
ret = hal_sdio_execute_command53_blocking(HAL_SDIO_PORT_0, &config);
if (HAL_SDIO_STATUS_OK != ret) {
    // Error handler.
}
```

- Read from or write to the SDIO slave with COMMAND53 in the DMA interrupt mode
  - Step 1. Call **hal_sdio_init()** to initialize the MSDC and the SDIO slave to transfer states.
  - Step 2. Call **hal_sdio_register_callback()** to register the transfer result callback.
  - Step 3. Call **hal_sdio_execute_command53_dma()** to read data from or write data to the SDIO card.
  - Read sample code:

```
hal_sdio_status_t ret;
hal_sdio_command53_config_t config;
hal_sdio_config_t sdio_config = {HAL_SDIO_BUS_WIDTH_4, 25000};

void sdio_dma_transfer_callback(hal_sdio_callback_event_t sdio_event, void *user_data)
{
    if (HAL_SDIO_EVENT_SUCCESS == sdio_event) {
        // DMA transfer OK.
    } else if (HAL_SDIO_EVENT_TRANSFER_ERROR == sdio_event) {
        // DMA transfer error.
    } else if (HAL_SDIO_EVENT_CRC_ERROR == sdio_event) {
        // DMA transfer with CRC error.
    } else if (HAL_SDIO_EVENT_DATA_TIMEOUT == sdio_event) {
        // DMA transfer with timeout.
    }
}

ret = hal_sdio_init(HAL_SDIO_PORT_0,&sdio_config);

if (HAL_SDIO_STATUS_OK != ret) {
   // Error handler.
}

if (HAL_SDIO_STATUS_OK != hal_sd_register_callback(HAL_SDIO_PORT_0,
       sdio_dma_transfer_callback,NULL)) {
    // Error handler.
}

config.direction = HAL_SDIO_DIRECTION_READ;
config.function = HAL_SDIO_FUNCTION_0;
config.is_block = true;
config.count = 2;
config.address = address;
config.buffer = buffer;
ret = hal_sdio_execute_command53_dma(HAL_SDIO_PORT_0, &config);
if (HAL_SDIO_STATUS_OK != ret) {
    // Error handler.
}
```

Exhibit 1 Terms and Conditions

Your access to and use of this document and the information contained herein (collectively this "Document") is subject to your (including the corporation or other legal entity you represent, collectively "You") acceptance of the terms and conditions set forth below ("T&C"). By using, accessing or downloading this Document, You are accepting the T&C and agree to be bound by the T&C. If You don't agree to the T&C, You may not use this Document and shall immediately destroy any copy thereof.

This Document contains information that is confidential and proprietary to MediaTek Inc. and/or its affiliates (collectively "MediaTek") or its licensors and is provided solely for Your internal use with MediaTek's chipset(s) described in this Document and shall not be used for any other purposes (including but not limited to identifying or providing evidence to support any potential patent infringement claim against MediaTek or any of MediaTek's suppliers and/or direct or indirect customers). Unauthorized use or disclosure of the information contained herein is prohibited. You agree to indemnify MediaTek for any loss or damages suffered by MediaTek for Your unauthorized use or disclosure of this Document, in whole or in part.

MediaTek and its licensors retain titles and all ownership rights in and to this Document and no license (express or implied, by estoppels or otherwise) to any intellectual propriety rights is granted hereunder. This Document is subject to change without further notification. MediaTek does not assume any responsibility arising out of or in connection with any use of, or reliance on, this Document, and specifically disclaims any and all liability, including, without limitation, consequential or incidental damages.

THIS DOCUMENT AND ANY OTHER MATERIALS OR TECHNICAL SUPPORT PROVIDED BY MEDIATEK IN CONNECTION WITH THIS DOCUMENT, IF ANY, ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE. MEDIATEK SPECIFICALLY DISCLAIMS ALL WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, FITNESS FOR A PARTICULAR PURPOSE, COMPLETENESS OR ACCURACY AND ALL WARRANTIES ARISING OUT OF TRADE USAGE OR OUT OF A COURSE OF DEALING OR COURSE OF PERFORMANCE. MEDIATEK SHALL NOT BE RESPONSIBLE FOR ANY MEDIATEK DELIVERABLES MADE TO MEET YOUR SPECIFICATIONS OR TO CONFORM TO A PARTICULAR STANDARD OR OPEN FORUM.

Without limiting the generality of the foregoing, MediaTek makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does MediaTek assume any liability arising out of the application or use of any product, circuit or software. You agree that You are solely responsible for the designing, validating and testing Your product incorporating MediaTek's product and ensure such product meets applicable standards and any safety, security or other requirements.

The above T&C and all acts in connection with the T&C or this Document shall be governed, construed and interpreted in accordance with the laws of Taiwan, without giving effect to the principles of conflicts of law.