MT793X IoT SDK for SPI Master User Guide

Version: 2.0
Release date: 2021-11-09

Version History

Version	Date	Description
2.0	2021-11-09	Official release

Table of Contents

List of Figures

List of Tables

1 SPI Master

1.1 Introduction

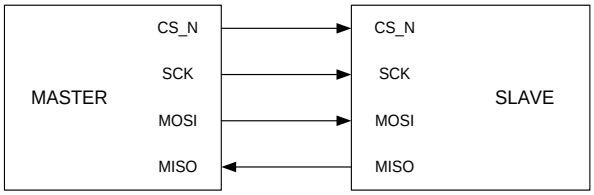

Figure 1-1. Pin connection between SPI master and SPI slave

The SPI interface is a bit-serial, four-pin transmission protocol. Figure 1-1 is an example of the connection between the SPI master and SPI slave. The SPI interface controller is a master responsible of the data transmission with the slave.

1.2 Pin Description

Table 1-1. SPI controller interface

Signal name	Type	Description
CS_N	O	Low active chip selection signal
SCK	O	The (bit) serial clock
MOSI	O	Data signal from master output to slave input
MISO	I	Data signal from slave output to master input

1.3 Features

The features of the SPI controller (master) are:

- The supported SPI_CLK is up to 50MHz.
- Configurable CS_N setup time, hold time and idle time
- Programmable SCK high time and low time
- Configurable transmitting and receiving bit order
- Two configurable modes for the source of the data to be transmitted. 1) In Tx DMA mode, the SPI controller automatically fetches the transmitted data (to be put on the MOSI line) from memory; 2) In Tx FIFO mode, the data to be transmitted on the MOSI line are written to FIFO before the start of the transaction.
- Two configurable modes for destination of the data to be received. 1) In Rx DMA mode, the SPI controller automatically stores the received data (from MISO line) to memory; 2) In Rx FIFO mode, the received data remain in Rx FIFO of the SPI controller. The processor must read back the data by itself.
- Adjustable endian order from/to memory system
- Programmable byte length for transmission
- Unlimited length for transmission. This is achieved by the operation of PAUSE mode. In PAUSE mode, the CS_N signal stays active (low) after the transmission. At this time, the SPI controller is in PAUSE_IDLE state, ready to receive the resume command. The state transition is shown in Figure 1-2.

- Configurable option to control CS_N deassert between byte transfers. The controller supports a special transmission format called CS_N deassert mode. Figure 1-3 illustrates the waveform in this transmission format.

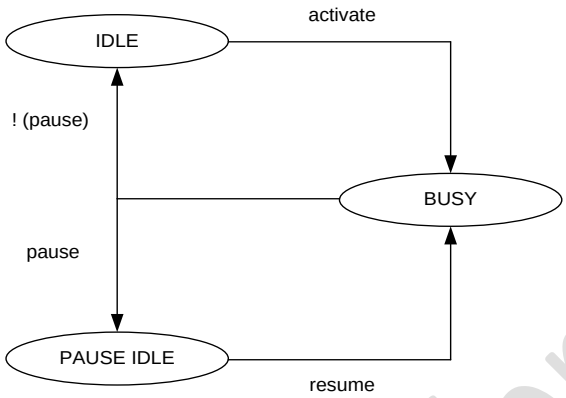

Figure 1-2. Operation flow with or without PAUSE mode

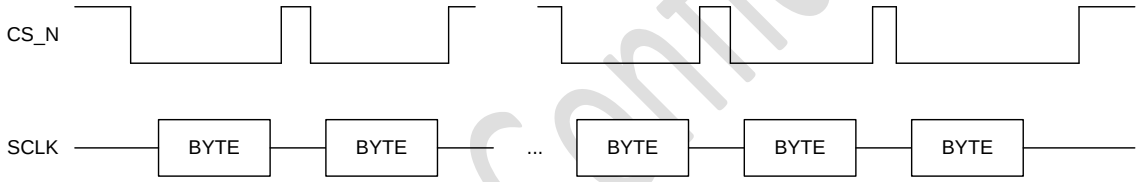

Figure 1-3. CS_N deassert mode

2 Driver Introduction

2.1 Driver API Reference

API	Description
hal_spi_master_status_t hal_spi_master_init(hal_spi_master_port_t master_port, hal_spi_master_config_t *spi_config);	This function is mainly used to initialize the SPI master and set user defined common parameters like clock frequency,
hal_spi_master_status_t hal_spi_master_deinit(hal_spi_master_port_t master_port)	This function resets the SPI master, gates its clock, disables interrupts.
hal_spi_master_status_t hal_spi_master_set_advanced_config(hal_spi_master_port_t master_port, hal_spi_master_advanced_config_t *advanced_config)	SPI master advanced configuration function.
hal_spi_master_status_t hal_spi_master_send_polling(hal_spi_master_port_t master_port, uint8_t *data, uint32_t size)	This function is used to send data synchronously with FIFO mode.
hal_spi_master_status_t hal_spi_master_send_dma(hal_spi_master_port_t master_port, uint8_t *data, uint32_t size)	This function is used to send data asynchronously with DMA mode.
hal_spi_master_status_t hal_spi_master_send_dma_blocking(hal_spi_master_port_t master_port, uint8_t *data, uint32_t size)	This function is used to send data synchronously with DMA mode.
hal_spi_master_status_t hal_spi_master_send_and_receive_polling(hal_spi_master_port_t master_port, hal_spi_master_send_and_receive_config_t *spi_send_and_receive_config)	This function simultaneously sends and receives data in the FIFO mode.
hal_spi_master_status_t hal_spi_master_send_and_receive_dma(hal_spi_master_port_t master_port, hal_spi_master_send_and_receive_config_t *spi_send_and_receive_config)	This function is used to send and receive data asynchronously with DMA mode.
hal_spi_master_status_t hal_spi_master_send_and_receive_dma_blocking(hal_spi_master_port_t master_port, hal_spi_master_send_and_receive_config_t *spi_send_and_receive_config)	This function simultaneously sends and receives data in the DMA mode.
hal_spi_master_status_t hal_spi_master_get_running_status(hal_spi_master_port_t master_port, hal_spi_master_running_status_t *running_status)	This function gets current running status of the SPI master.
hal_spi_master_status_t hal_spi_master_set_chip_select_timing(hal_spi_master_port_t master_port, hal_spi_master_chip_select_timing_t chip_select_timing)	This function is used to configure SPI master chip select timing parameter.
hal_spi_master_status_t hal_spi_master_set_deassert(hal_spi_master_port_t master_port, hal_spi_master_deassert_t deassert)	SPI master chip select de-assertion mode configuration.
hal_spi_master_status_t hal_spi_master_register_callback(hal_spi_master_port_t master_port, hal_spi_master_callback_t callback, void *user_data)	This function is used to register user's callback to SPI master driver.

2.2 Sample Code

2.2.1 Using SPI Master Polling Mode

The steps are shown below:

- Step 1. Call **hal_gpio_init()** to inite the pins, if EPT tool hasn't been used to configure the related pinmux.
- Step 2. Call **hal_pinmux_set_function()** to set GPIO pinmux, if EPT tool hasn't been used to configure the related pinmux. For more details about **hal_pinmux_set_function()**, please refer to **GPIO**.
- Step 3. Call **hal_spi_master_init()** to initialize one SPI master. If the SPI master is already initialized by another user, user will get **HAL_SPI_MASTER_STATUS_ERROR_BUSY**.
- Step 4. Call **hal_spi_master_send_polling()** to send data in the polling mode.
- Step 5. Call **hal_spi_master_send_and_receive_polling()** to send and receive data in the polling mode.
- Step 6. Call **hal_spi_master_deinit()** to deinitialize the SPI master, if it's no longer in use.

- sample code:

```
hal_spi_master_config_t spi_config;
hal_spi_master_send_and_receive_config_t spi_send_and_receive_config;
uint8_t status_cmd = SPIS_STATUS_CMD;
uint8_t status_receive[2];
uint8_t data[2] = {0x7E, 0x55};
uint32_t size = 2;
spi_config.bit_order = HAL_SPI_MASTER_LSB_FIRST;
spi_config.slave_port = HAL_SPI_MASTER_SLAVE_0;
spi_config.clock_frequency = 1000000;
spi_config.phase = HAL_SPI_MASTER_CLOCK_PHASE0;
spi_config.polarity = HAL_SPI_MASTER_CLOCK_POLARITY0;
status_receive[1] = 0;
spi_send_and_receive_config.receive_length = 2;
spi_send_and_receive_config.send_length = 1;
spi_send_and_receive_config.send_data = &status_cmd;
spi_send_and_receive_config.receive_buffer = status_receive;
```

```
// Initialize the GPIO, set GPIO pinmux (if EPT tool hasn't been used to configure the related pinmux).
hal_gpio_init(HAL_GPIO_6);
hal_gpio_init(HAL_GPIO_7);
hal_gpio_init(HAL_GPIO_8);
hal_gpio_init(HAL_GPIO_9);
// Call hal_pinmux_set_function() to set GPIO pinmux, for more information, please refer to hal_pinmux_define.h
// Not need to configure the pinmux, if EPT is used.
hal_pinmux_set_function(HAL_GPIO_6, 3);// Set the pin to be used as SCK signal of SPI Master.
hal_pinmux_set_function(HAL_GPIO_7, 3);// Set the pin to be used as CS signal of SPI Master.
hal_pinmux_set_function(HAL_GPIO_8, 3);// Set the pin to be used as MISO signal of SPI Master.
hal_pinmux_set_function(HAL_GPIO_9, 3);// Set the pin to be used as MOSI signal of SPI Mastr.
```

```
// Initialize the SPI master.
if (HAL_SPI_MASTER_STATUS_OK == hal_spi_master_init(HAL_SPI_MASTER_0,&spi_config)) {
    if (HAL_SPI_MASTER_STATUS_OK != hal_spi_master_send_polling(HAL_SPI_MASTER_0,data,size)) {
        // Error handler;
    }
    //send and receive data at the same time
    if (HAL_SPI_MASTER_STATUS_OK != hal_spi_master_send_and_receive_polling(HAL_SPI_MASTER_0,&spi_send_and_receive_config)) {
        // Error handler;
    }
    hal_spi_master_deinit(HAL_SPI_MASTER_0); // Deinitialize, if the SPI master is no longer in use.
} else {
    // Error handler;
}
```

2.2.2 Using SPI Master DMA Mode

The steps are shown below:

- Step 1. Call **hal_gpio_init()** to inite the pins, if EPT tool hasn't been used to configure the related pinmux.
- Step 2. Call **hal_pinmux_set_function()** to set GPIO pinmux, if EPT tool hasn't been used to configure the related pinmux. For more details about **hal_pinmux_set_function()**, please refer to **GPIO**.
- Step 3. Call **hal_spi_master_init()** to init one SPI master. If the SPI master is already initialized by another user, user will get **HAL_SPI_MASTER_STATUS_ERROR_BUSY**.
- Step 4. Call **hal_spi_master_register_callback()** to register a user callback.
- Step 5. Call **hal_spi_master_send_dma()** to send data in the DMA mode.
- Step 6. Call **hal_spi_master_send_and_receive_dma()** to send and receive data in the DMA mode.
- Step 7. Call **hal_spi_master_deinit()** to deinit the SPI master, if it's no longer in use.

- sample code:

```
hal_spi_master_config_t spi_config;
hal_spi_master_send_and_receive_config_t spi_send_and_receive_config;
ATTR_ZIDATA_IN_NONCACHED_RAM_4BYTE_ALIGN uint8_t status_cmd = SPIS_STATUS_CMD;
ATTR_ZIDATA_IN_NONCACHED_RAM_4BYTE_ALIGN uint8_t status_receive[2];
ATTR_ZIDATA_IN_NONCACHED_RAM_4BYTE_ALIGN uint8_t data[2] = {0x7E, 0x55};
uint32_t size = 2;

spi_config.bit_order = HAL_SPI_MASTER_LSB_FIRST;
spi_config.slave_port = HAL_SPI_MASTER_SLAVE_0;
spi_config.clock_frequency = 1000000;
spi_config.phase = HAL_SPI_MASTER_CLOCK_PHASE0;
spi_config.polarity = HAL_SPI_MASTER_CLOCK_POLARITY0;
status_receive[1] = 0;
spi_send_and_receive_config.receive_length = 2;
spi_send_and_receive_config.send_length = 1;
spi_send_and_receive_config.send_data = &status_cmd;
spi_send_and_receive_config.receive_buffer = status_receive;
```

```
// Initialize the GPIO, set GPIO pinmux (if EPT tool hasn't been used to configure the related pinmux).
hal_gpio_init(HAL_GPIO_6);
hal_gpio_init(HAL_GPIO_7);
hal_gpio_init(HAL_GPIO_8);
hal_gpio_init(HAL_GPIO_9);
// Call hal_pinmux_set_function() to set GPIO pinmux, for more information, please refer to hal_pinmux_define.h
// Not need to configure the pinmux, if EPT is used.
hal_pinmux_set_function(HAL_GPIO_6, 3);// Set the pin to be used as SCK signal of SPI Master.
hal_pinmux_set_function(HAL_GPIO_7, 3);// Set the pin to be used as CS signal of SPI Master.
hal_pinmux_set_function(HAL_GPIO_8, 3);// Set the pin to be used as MISO signal of SPI Master.
hal_pinmux_set_function(HAL_GPIO_9, 3);// Set the pin to be used as MOSI signal of SPI Mastr.
```

```
// Initialize the SPI master.
if (HAL_SPI_MASTER_STATUS_OK == hal_spi_master_init(HAL_SPI_MASTER_0 ,&spi_config)) {
    hal_spi_master_register_callback(HAL_SPI_MASTER_0 ,user_spi_callback,NULL); // Register a user callback.
    if (HAL_SPI_MASTER_STATUS_OK != hal_spi_master_send_dma(HAL_SPI_MASTER_0 ,data,size)) {
        // Error handler;
    }
    // Send and receive data simultaneously.
    if (HAL_SPI_MASTER_STATUS_OK != hal_spi_master_send_and_receive_dma(HAL_SPI_MASTER_0, &spi_send_and_receive_config)) {
        // Error handler;
    }
} else {
    // Error handler;
}
```

```
// Callback function sample code. Pass this function to the driver while calling #hal_spi_master_register_callback().
void user_spi_callback (hal_spi_master_callback_event_t event,void *user_data)
{
    if (HAL_SPI_MASTER_EVENT_SEND_FINISHED == event) {
        // Send finish event handler;
        // User code;
        hal_spi_master_deinit(HAL_SPI_MASTER_0); // Deinitialize, if no longer in use.
    } else if (HAL_SPI_MASTER_EVENT_RECEIVE_FINISHED == event) {
        // Receive finish event handler;
        // User code;
        hal_spi_master_deinit(HAL_SPI_MASTER_0); // Deinitialize, if no longer in use.
    }
}
```

2.2.3 Using SPI Master DMA Blocking Mode

The steps are shown below:

- Step 1. Call **hal_gpio_init()** to inite the pins, if EPT tool hasn't been used to configure the related pinmux.
- Step 2. Call **hal_pinmux_set_function()** to set GPIO pinmux, if EPT tool hasn't been used to configure the related pinmux. For more details about **hal_pinmux_set_function()**, please refer to **GPIO**.
- Step 3. Call **hal_spi_master_init()** to initialize one SPI master. If the SPI master is already initialized by another user, user will get **HAL_SPI_MASTER_STATUS_ERROR_BUSY**.
- Step 4. Call **hal_spi_master_send_dma_blocking()** to send data in the DMA blocking mode.
- Step 5. Call **hal_spi_master_send_and_receive_dma_blocking()** to send and receive data in the DMA blocking mode.
- Step 6. Call **hal_spi_master_deinit()** to deinitialize the SPI master, if it's no longer in use.

- sample code:

```
hal_spi_master_config_t spi_config;
hal_spi_master_send_and_receive_config_t spi_send_and_receive_config;
ATTR_ZIDATA_IN_NONCACHED_RAM_4BYTE_ALIGN uint8_t status_cmd = SPIS_STATUS_CMD;
ATTR_ZIDATA_IN_NONCACHED_RAM_4BYTE_ALIGN uint8_t status_receive[2];
ATTR_ZIDATA_IN_NONCACHED_RAM_4BYTE_ALIGN uint8_t data[2] = {0x7E, 0x55};
uint32_t size = 2;

spi_config.bit_order = HAL_SPI_MASTER_LSB_FIRST;
spi_config.slave_port = HAL_SPI_MASTER_SLAVE_0;
spi_config.clock_frequency = 1000000;
spi_config.phase = HAL_SPI_MASTER_CLOCK_PHASE0;
spi_config.polarity = HAL_SPI_MASTER_CLOCK_POLARITY0;
status_receive[1] = 0;
spi_send_and_receive_config.receive_length = 2;
spi_send_and_receive_config.send_length = 1;
spi_send_and_receive_config.send_data = &status_cmd;
spi_send_and_receive_config.receive_buffer = status_receive;
```

```
// Initialize the GPIO, set GPIO pinmux (if EPT tool hasn't been used to configure the related pinmux).
hal_gpio_init(HAL_GPIO_6);
hal_gpio_init(HAL_GPIO_7);
hal_gpio_init(HAL_GPIO_8);
hal_gpio_init(HAL_GPIO_9);
// Call hal_pinmux_set_function() to set GPIO pinmux, for more information, please refer to hal_pinmux_define.h
// Not need to configure the pinmux, if EPT is used.
hal_pinmux_set_function(HAL_GPIO_6, 3);// Set the pin to be used as SCK signal of SPI Master.
hal_pinmux_set_function(HAL_GPIO_7, 3);// Set the pin to be used as CS signal of SPI Master.
hal_pinmux_set_function(HAL_GPIO_8, 3);// Set the pin to be used as MISO signal of SPI Master.
hal_pinmux_set_function(HAL_GPIO_9, 3);// Set the pin to be used as MOSI signal of SPI Mastr.
```

```
// Initialize the SPI master.
if (HAL_SPI_MASTER_STATUS_OK == hal_spi_master_init(HAL_SPI_MASTER_0,&spi_config)) {
    if (HAL_SPI_MASTER_STATUS_OK != hal_spi_master_send_dma_blocking(HAL_SPI_MASTER_0,data,size)) {
        // Error handler;
    }
    //send and receive data at the same time
    if (HAL_SPI_MASTER_STATUS_OK != hal_spi_master_send_and_receive_dma_blocking(HAL_SPI_MASTER_0,&spi_send_and_receive_config)) {
        // Error handler;
    }
    hal_spi_master_deinit(HAL_SPI_MASTER_0); // Deinitialize, if the SPI master is no longer in use.
} else {
    // 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.