ESP8266

SPI Reference

Version 1.0

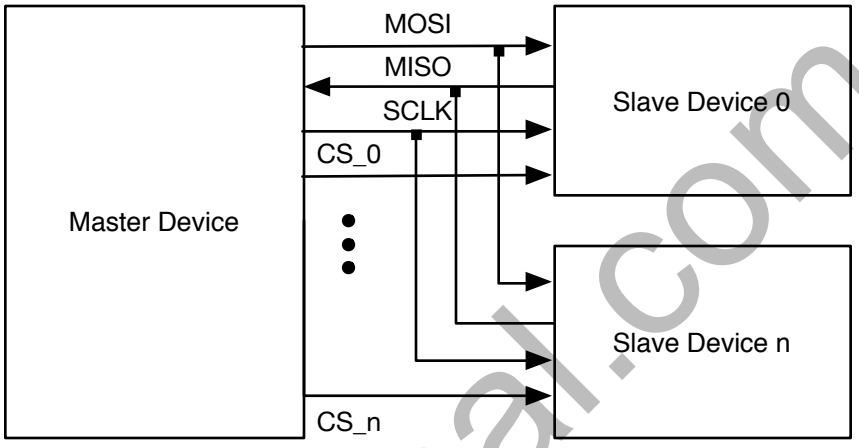
Espressif Systems IOT Team

http://bbs.espressif.com

Table of Contents

1. Overview

ESP8266 SPI (*Serial Peripheral Interface*) module is used for communication with devices supporting SPI protocols. It supports the SPI protocol standard of 4-line communication in the electrical interface as shown in *Picture 1-1*.



Picture 1-1. Four - line communication

- CS (Chip Select) or SS (slave select)
- SCLK (*Synchronous Clock*)
- MOSI (*Master Out Slave In*)
- MISO (*Master Input and Slave Output*)

ESP8266 SPI module provides special support for FLASH memory in the SPI interface. Therefore, master and slave SPI module has its corresponding hardware protocol to match with the SPI communication device.

2. Communication protocol

2.2. Communication protocol of SPI master

2.2.1. Function description

ESP8266 SPI master supports the communication protocols as shown in *Table 2-1*.

Table 2-1. Communication protocol supported by SPI master

Name	Must / Optional	Length	Interface type
Command	Optional	0 - 16bits (2 bytes)	MOSI
Address	Optional	0 - 32bits (4 bytes)	MOSI
Read/write data	Optional	0 - 512bits (64bytes)	MOSI or MISO

2.2.2. Current API function

The API function of ESP8266 SPI master has two initialization modes as shown in the following.

1. Supporting most of the general signals
2. Designed for driving a colored LCD screen

Note:

The device needs non-standard 9-bit SPI communication protocol. Refer to 4.1 for detail information.

2.3. Communication protocol of SPI slave

2.3.1. Function description

ESP8266 SPI slave supports the communication protocols as shown in *Table 2-2*.

Table 2-2. Communication protocol supported by SPI slave

Name	Must / Optional	Length	Interface type
Command	Must	3 - 16 bits (2 bytes)	MOSI
Address	Must	1 - 32 bits (4 bytes)	MOSI
Read/write data	Optional	0 - 512 bits (64 bytes)	MOSI or MISO

2.3.2. Configuration requirement

The clock polarity of the SPI master device that communicate with the ESP8266 SPI slave should be set as the following.

- idle low power
- sampling on the rising edge
- update data on the falling edge

Please make sure to keep low power for CS in a 16's reading/writing process. If the CS power is raised to high level while transmitting, the internal state of slave will be reset.

2.3.3. Command definition

The length of slave receiving command should be at least 3 bits. For low 3 bits, the reading and writing must follow the instructions as shown in *Table 2-3*.

Table 2-3. Command definition

Value	Description	Details
010	slave receiving	Write the data sent by master into the data register of slave via MOSI.
011	slave transmitting	Send the data in the data register of slave to master via MOSI.
110	slave receiving and transmitting	Send the data in the data register of slave to MISO and write the master data in MOSI into data register of slave.

Note:

The data register of slave is from SPI_FLASH_C0 to SPI_FLASH_C15.

Caution:

Do NOT use other values that used to read and write the status register of SPI slave, SPI_FLASH_STATUS, whose communication protocols are different from the data register, otherwise it might cause a read/write error.

2.3.4. Current API function

The API function of ESP8266 SPI master has two initialization modes as shown in the following.

1. Supporting most of the general signals
2. Designed for driving a colored LCD screen

The API function of ESP8266 SPI has a slave initialization mode which is compatible with most of the devices in bytes. The slave communication format should be set as shown in *Table 2-4*.

Table 2-4. Communication format of SPI slave

Command	Address	Read/write data
7 bits	1 bit	8 bits

Command and address combines to be high 8 bits and the address must be 0.

The other SPI master devices could read / write single - byte of slave SPI for the communication of one - time 16 bits (or two - times 8 bits with low lever CS). For detail information, refer to 4.2.

3. API description

3.1. SPI master

3.1.1. void spi_lcd_mode_init (uint8 spi_no)

Table 3-1. void spi_lcd_mode_init

Item	Description
Name	void spi_lcd_mode_init
Example	`void spi_lcd_mode_init (uint8 spi_no)`
Function	Provide SPI master initialization program for driving the chromatic LCD TM035PDZV36.
Parameter	uint8 spi_no • Description: The number of SPI module. • Range of values: SPI (0) or HSPI (1).

Note:

ESP8266 processor has two SPI modules with the same function, SPI and HSPI.

3.1.2. void spi_lcd_9bit_write

Table 3-2. void spi_lcd_9bit_write

Item	Description
Name	void spi_lcd_9bit_write
Example	`void spi_lcd_9bit_write(uint8 spi_no,uint8 high_bit,uint8 low_8bit)`
Function	Provide SPI master transmitting program for driving the chromatic LCD TM035PDZV36. The LCD module needs a 9-bit transmitting.
Parameter	uint8 spi_no • Description: The number of SPI module. • Range of values: SPI or HSPI.
	uint8 high_bit • Description: The data in the 9th bit. • Range of values: 0 in the 9th bit represents the 0 and other data in the 9th bit represents the 1.
	uint8 low_8bit : Data in the low 8 bits.

3.1.3. void spi_master_init

Table 3-3. void spi_master_init

Item	Description
Name	void spi_master_init
Example	`void spi_master_init(uint8 spi_no)`
Function	General function of SPI master initialization. Baud rate is 1/4 frequency of CPU clock. All the master functions except `spi_lcd_9bit_write` can be used after initialization.
Parameter	uint8 spi_no • Description: The number of SPI module. • Range of values: SPI or HSPI.

3.1.4. void spi_mast_byte_write

Table 3-4. void spi_mast_byte_write

Item	Description
Name	void spi_mast_byte_write
Example	`void spi_mast_byte_write(uint8 spi_no,uint8 data)`
Function	Send data in single byte from the SPI master.
Parameter	uint8 spi_no • Description: The number of SPI module. • Range of values: SPI or HSPI.
	uint8 data 8-bit data been sent.

3.1.5. void spi_byte_write_espslave

Table 3-5. void spi_byte_write_espslave

Item	Description
Name	void spi_byte_write_espslave
Example	`void spi_byte_write_espslave(uint8 spi_no,uint8 data)`
Function	Write one - byte data to the SPI slave of ESP 8266. As the slave communication format is set as 7-bit command+1bit address+8-bit data as shown in Table 4. Therefore you need to transmit the data in 16 bits, and the first byte is 0b0000010+0（refer to 2.3.3 ）, i.e. 0x04. The second byte is the data that been sent. The actual waveform is shown in *Figure 3-1*.
Parameter	uint8 spi_no • Description: The number of SPI module. • Range of values: SPI or HSPI.
	uint8 data: 8-bit data been sent.

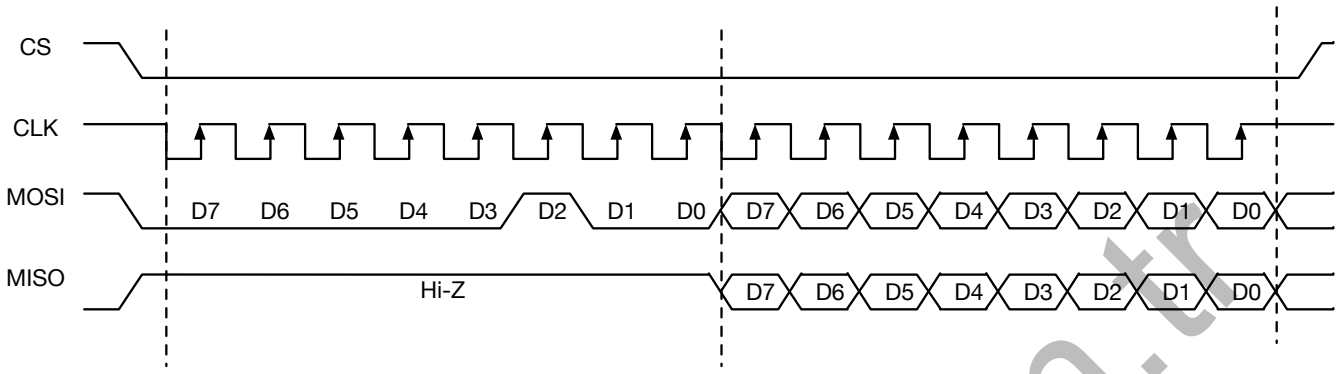


Figure 3-1. The waveform of spi_byte_write_espslave

3.1.6. void spi_byte_read_espslave

Table 3-6. void spi_byte_write_espslave

Item	Description
Name	void spi_byte_read_espslave
Example	`void spi_byte_read_espslave(uint8 spi_no,uint8* data)`

Function	• Read one - byte data from SPI slave of ESP 8266 and also the other devices.
	• The slave communication protocol should be configured to: *7-bit command + 1-bit address + 8-bit data* as shown in Table 4.
	• You need to transmit the data in 16 bits:
	  ◦ The first byte is *0b0000011 + 0*（refer to 2.3.3 ）, i.e. 0x06.
	  ◦ The second byte is the data that been received.
	• The actual waveform is shown in *Figure 3-2*.
	• For the other full duplex slave devices:
	  - You need to configure the slave before a 16-bit communication.
	  - You should placed the data which will be received by ESP8266 master in the second byte of slave's caching.

Parameter	uint8 spi_no • Description: The number of SPI module. • Range of values: SPI or HSPI.
	uint8 data: 8-bit data been received.

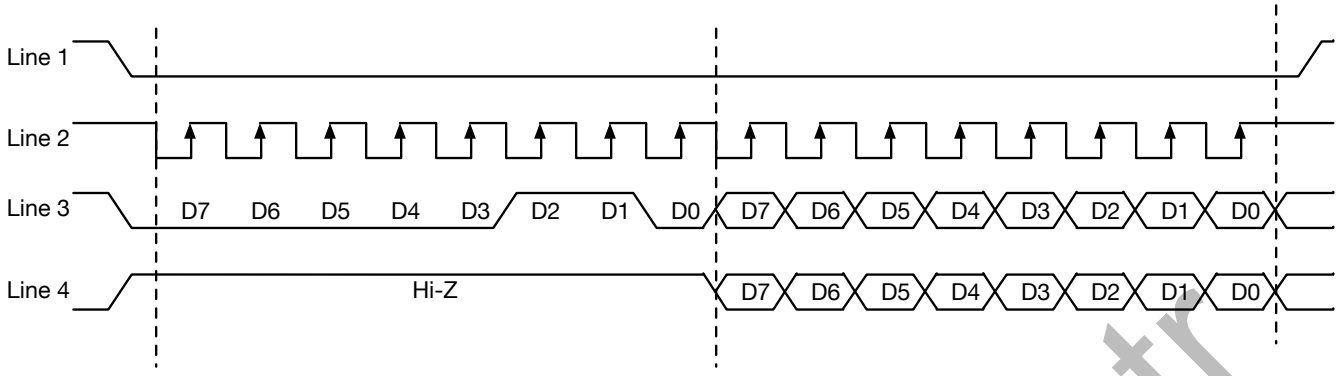


Figure 3-2. The slave waveform of spi_byte_read_espslave

3.2. SPI slave

3.2.1. void spi_slave_init

Table 3-7. void spi_slave_init

Item	Description
Name	void spi_slave_init
Example	`void spi_slave_init(uint8 spi_no)`

Function	Initialize mode of SPI slave, configure IO interface to SPI mode, enable SPI transmission interruption and register the function `spi_slave_isr_handler`.
	• Communication protocol should be configured to:
	7-bit command + 1-bit address + 8-bit read/write data
	as shown in Table 4.
	• 3 master commands are supported as the following (refer to 2.3.3).
	  • 0x04: the master write and the slave read
	  • 0x06: the master write and the slave read
	  • 0x0c: the master and the slave read and write in the same time
	The communication waveforms of read and write is shown in Figure 1 and 2.

Parameter	uint8 spi_no • Description: The number of SPI module. • Range of values: SPI or HSPI.
	uint8 data: 8-bit data been received.

3.2.2. spi_slave_isr_handler(void *para)

Table 3-8. spi_slave_isr_handler

Item	Description
Name	spi_slave_isr_handler
Example	`spi_slave_isr_handler(void *para)`
Function	SPI interrupt processing function. Interruption will be triggered if the master transmits correctly (read/write the slave).
Parameter	None

4. Example Code

```
//0x3ff00020 is isr flag register, bit4 is for spi isr
    if(READ_PERI_REG(0x3ff00020)&BIT4){
        //following 3 lines is to close spi isr enable
        regvalue=READ_PERI_REG(SPI_FLASH_SLAVE(SPI));
        regvalue&=~(0x3ff);
        WRITE_PERI_REG(SPI_FLASH_SLAVE(SPI),regvalue);
        //os_printf("SPI ISR is trigged\n"); //debug code
    }else if(READ_PERI_REG(0x3ff00020)&BIT7){ //bit7 is for hspi isr,
        //following 3 lines is to clear hspi isr signal
        regvalue=READ_PERI_REG(SPI_FLASH_SLAVE(HSPI));
        regvalue&=~(0x1f);
        WRITE_PERI_REG(SPI_FLASH_SLAVE(HSPI),regvalue);
        //when master command is write slave 0x04,
        //received data will be occur in register SPI_FLASH_C0's low 8 bit,
        //also if master command is read slave 0x06,
        //the low 8bit data in register SPI_FLASH_C0 will transmit to
master,
        //so prepare the transmit data in SPI_FLASH_C0' low 8bit,
        //if a slave transmission needs
        recv_data=(uint8)READ_PERI_REG(SPI_FLASH_C0(HSPI));
        /*put user code here*/
    //os_printf("recv data is %08x\n", recv_data);
    //debug code
    }else if(READ_PERI_REG(0x3ff00020)&BIT9){ //bit9 is for i2s isr,
    }
```

Note:

The SPI is used to read and write the flash memory chip, so HSPI is used to communicate.For ESP8266 processor, this interrupt program is sharing by multiple modules, including the SPI, HSPI and I2S the I2S module corresponding the 4's, 7's and 9's bit of the 0x3ff00020 register (whose address is 0x3ff00020).

As transmission interrupts are triggered frequently by SPI module, in which 5 interrupt sources should be disabled. The corresponding codes are shown as below.

```
regvalue=READ_PERI_REG(SPI_FLASH_SLAVE(SPI));
regvalue&=~(0x3ff);
WRITE_PERI_REG(SPI_FLASH_SLAVE(SPI),regvalue);
```

When interrupts are triggered by HSPI module, in which 5 interrupt sources should be reset by software to avoid triggering interrupt program repeatedly. The corresponding codes are shown as below.

```
regvalue=READ_PERI_REG(SPI_FLASH_SLAVE(HSPI));
regvalue&=~(0x1f);
WRITE_PERI_REG(SPI_FLASH_SLAVE(HSPI),regvalue);
```

Data receiving and transmitting shares one register(`SPI_FLASH_C0`). The corresponding codes of reading register is shown as below.

```
recv_data=(uint8)READ_PERI_REG(SPI_FLASH_C0(HSPI));
```

Note:

recv_data is a global variable. You can defined your own codes after it.

Caution:

Avoid to execute extended periods of the interrupt program, or else the watchdog timer won't able to reset correctly and the processor will restart unexpectedly.