Application Note

Application Note

Document No.: AN1103

The APM32F103 software simulates SPI to read and write external Flash

Version: V1.0

1 Introduction

This application note provides how to simulate SPI to read and write external Flash through the software on the APM32F103 series. The purpose is to avoid the problem of limited serial communication ports in the development process. At the same time, it can also reduce the cost of development boards and the PCB complexity. The content of this document is developed based on the APM32F103 series.

Contents

1 Introduction 1

2 Introduction to APM32F103 SPI 3

2.1 Introduction to SPI 3

2.2 SPI transmission mode 4

2.3 Exchange of data 5

3 SPI implementation mode 7

3.1 Hardware SPI 7

3.2 GPIO simulating SPI 7

4 Introduction to Flash 8

4.1 Introduction to W25Q64 8

5 Routine of APM32F103 GPIO Simulating SPI to Read and Write External Flash 10

5.1 Hardware design 10

5.2 Software design 11

6 Revision history 25

2 Introduction to APM32F103 SPI

The full name of SPI is Serial Peripheral Interface. SPI is usually used between devices such as EEPROM, Flash, and OLED. APM32F103 SPI can be configured to support SPI protocol and I2S audio protocol. It works in SPI mode by default. SPI provides data transmitting and receiving functions based on SPI protocol, which allows chips to communicate with external devices in half duplex, full duplex, synchronous and serial modes, and can work in master or slave mode.

2.1 Introduction to SPI

SPI is usually composed of four lines: MOSI (Master Output Slave Input), MISO (Master Input Slave Output), SCLK (Serial Clock), and CS (Chip Select), as shown in Figure 1:

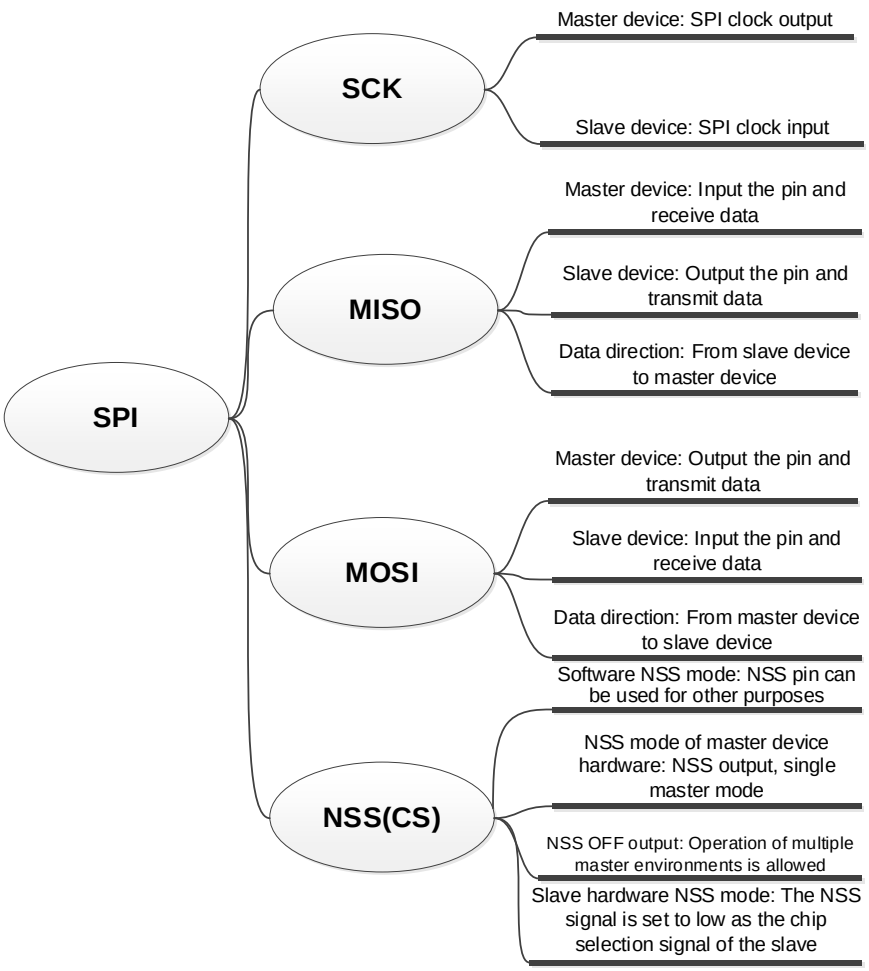

Figure 1 Introduction Diagram of Four Lines of SPI

2.2 SPI transmission mode

SPI has four transmission modes, and their main difference is CPOL (clock polarity) and CPHA (clock phase) of the CTRL1 register. CPOL means the level signal of SCK signal line when SPI is in idle state. CPHA means the sampling moment of data. SPI can be divided into four modes according to their states, as shown in Figure 2:

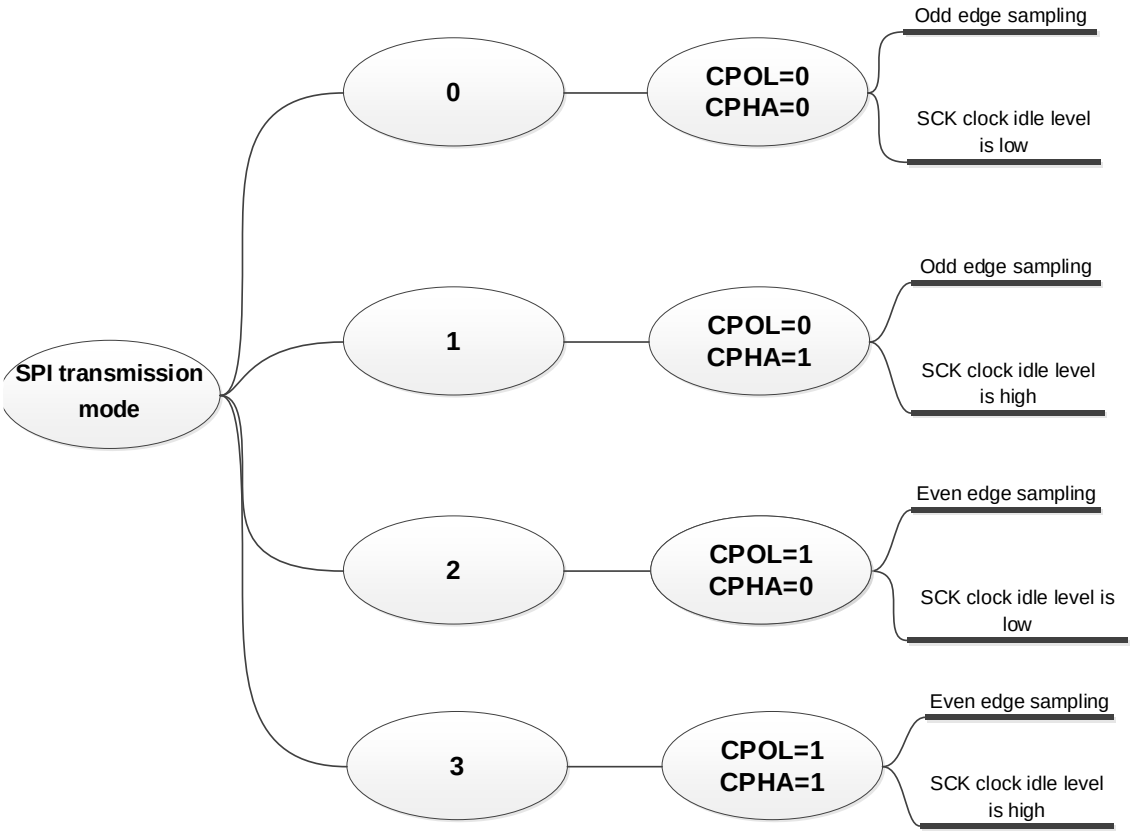

Figure 2 SPI Transmission Mode

According to the above figure, when CPHA=0, it means that data sampling will begin at the first clock edge. When CPOL=0, it means that the SCK clock is at low level when it is idle, and when it changes from low level to high level, sampling will be conducted on the rising edge. On the contrary, when CPOL=1, it means that the SCK clock is at high level when it is idle, and changes from high level to low level, and sampling will be conducted on the falling edge, as shown in Figure 3:

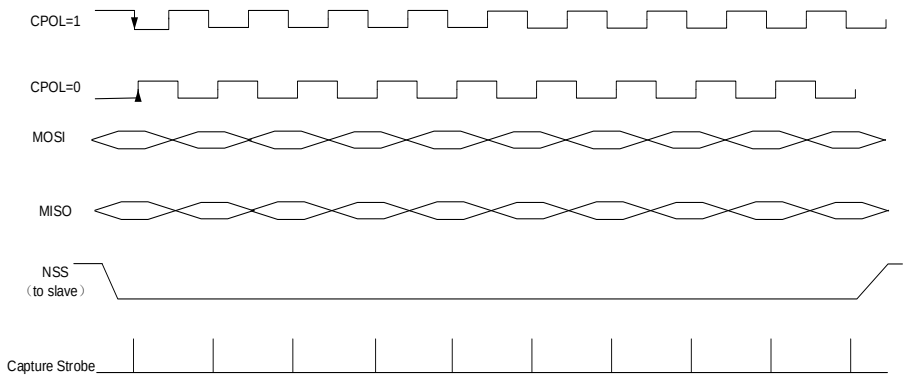

Figure 3 SPI Transmission Timing Diagram

When CPHA=1, it means that data sampling will begin at the second clock edge. When CPOL=0, it means that the SCK clock is at low level when it is idle, and when it changes from low level to high level, and then changes to low level, sampling will be conducted on the rising edge. On the contrary, when CPOL=1, it means that the SCK clock is at high level when it is idle, and changes from high level to low level, and then changes to high level, sampling will be conducted on the falling edge, as shown in Figure 4:

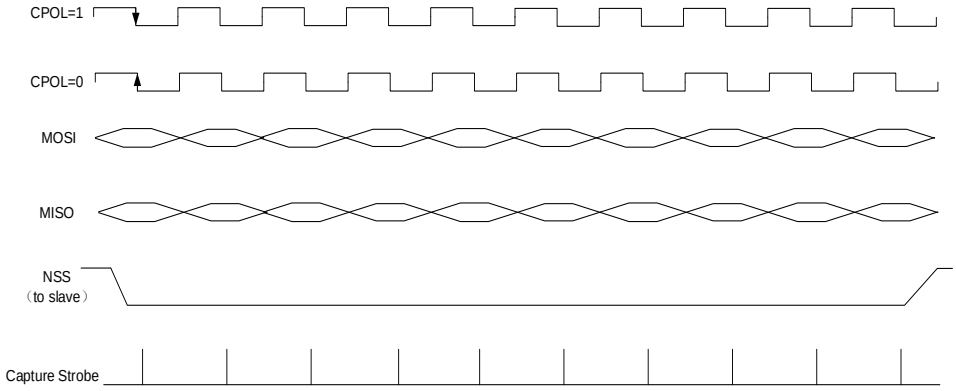

Figure 4 SPI Transmission Timing Diagram

2.3 Exchange of data

During the SCK clock cycle, both MOSI and MISO are in working state, as shown in Figure 5:

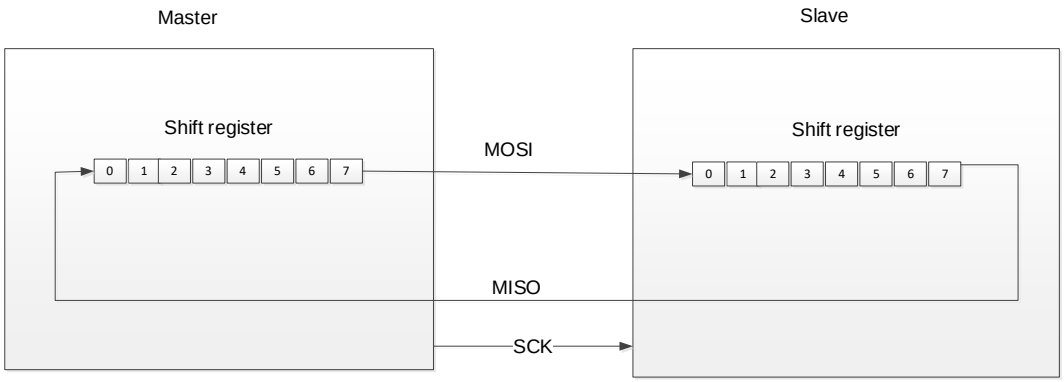

Figure 5 Data Exchange Diagram

As shown above, in SPI communication, both the master and slave have their own shift registers, which are used for data transmission. During transmission, the function of MOSI is to write data from the master shift register to the slave shift register, while the function of MISO is to transmit data from the slave shift register to the master shift register for data exchange.

3 SPI implementation mode

3.1 Hardware SPI

The APM32F103 series development board is a hardware SPI with its own hardware circuit. In the communication process, it relies on the built-in SPI peripheral, and the data transmission process is directly managed by the hardware, not needing additional programming to control each step of the SPI communication process.

3.1.1 Advantages and disadvantages of hardware SPI

Advantages:

(1) It can provide higher communication speed and consistency

(2) It can achieve higher communication modes, such as DMA transmission mode, interrupt handling mode, and master mode.

(3) Low code complexity, easy operation.

Disadvantages:

(1) The configuration and functions of hardware SPI are fixed and not easy to modify.

(2) The hardware SPI relies on specific hardware controller, and there is a compatibility problem in development of different platforms.

3.2 GPIO simulating SPI

In the development process, in order to avoid obstruction of development due to insufficient hardware SPI, the communication process of simulating SPI can be achieved through GPIO port and software control.

3.2.1 Advantages and disadvantages of GPIO simulating SPI

Advantages:

(1) The GPIO pin configuration is flexible, and the GPIO pins in the development board can be selected arbitrarily for simulation operation.

(2) With low cost, software simulating SPI does not require additional hardware design, and only GPIO pins are required.

Disadvantages:

(1) Low communication speed due to the processing capacity of the MCU.

(2) The code complexity is high and more resources are required.

4 Introduction to Flash

The full name of Flash is Flash Memory Controller. FLASH is usually used to store program codes, data and other information. Flash includes internal Flash and external Flash. The W25Q64 module used in this article is an external Flash module with SPI interface. During operation, the codes stored in Flash will not be modified, and Flash operates according to sectors.

Flash usually requires erase operation before writing data. It is because when Flash needs to write data, it cannot be directly modified on the sector that has already been programmed. Instead, it needs to erase the entire sector to the initial state and then perform the write operation.

4.1 Introduction to W25Q64

W25Q64 is a flash memory chip based on the peripheral interface (SPI), has 8Mbyte storage space and is a high-capacity SPI Flash product. W25Q64 divides the 8M byte capacity into 128 blocks, each with a size of 64K, and each block is further divided into 16 sectors, each with a size of 4K. The minimum erasing unit of W25Q64 is the sector, and a buffer of at least 4K needs to be opened up for it.

4.1.1 Common Command Instructions of W25Q64

By sending instructions to W25Q64 through SPI, W25Q64 can be operated. Some instruction lists used in the operations in this document are shown in Table 1:

Data Input Output	Byte1	Byte2	Byte3	Byte4	Byte5	Byte6	Byte7	Byte8
Number of Clock	8	8	8	8	8	8	8	8
Write Enable	06h	—						
Write Disable	04h	—						
Volatile SR Write Enable	50h	—						
Read Data	03h	A23-A16	A15-A8	A7-A0	（D7-D0）	—		
Page Program	02h	A23-A16	A15-A8	A7-A0	（D7-D0）	—		
Sector Erase （4KB）	20h	A23-A16	A15-A8	A7-A0	—			
Block Erase （32KB）	52h	A23-A16	A15-A8	A7-A0	—			
Block Erase （64KB）	D8h	A23-A16	A15-A8	A7-A0	—			

Table 1 Common Operation Commands of W25Q64

In page programming operation, first pull down the CS (chip selector). In the following 8 clock

cycles, the instruction 0x02 is issued by DI, and then in the next 24 clock cycles, a 24Bit address will be sent. In the last 256*8 clock cycles, 256Byte data will be sent. The timing diagram is shown in Figure 6:

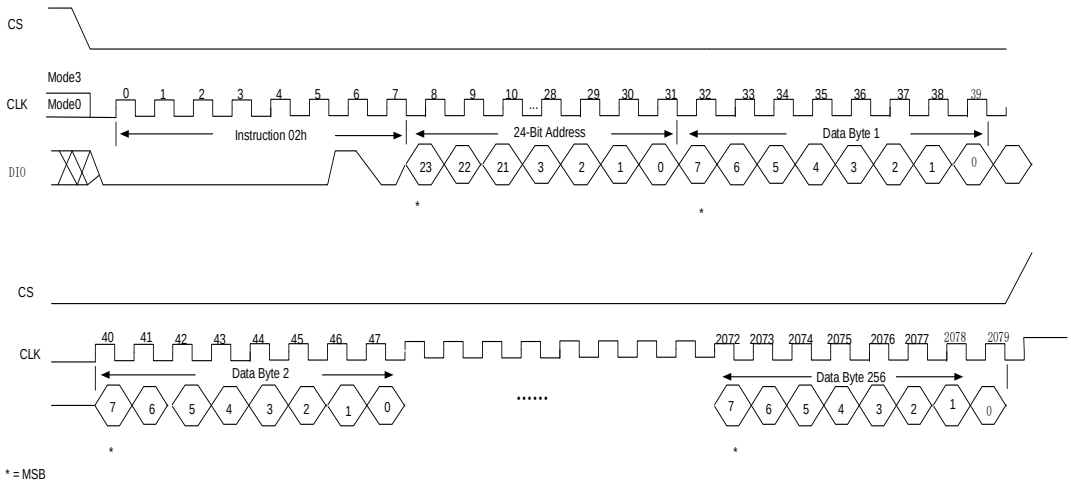

Figure 6 W25Q64 Command Operation Diagram

4.1.2 Advantages and disadvantages of W25Q64

Advantages:

(1) Large storage space

(2) It supports high-speed serial read and write operations and has a high data transmission rate

(3) Low power

(4) SPI interface is adopted for communication, making it easy to connect and control.

Disadvantages:

(1) High price

(2) Limited service life

(3) Slow erase operation

5 Routine of APM32F103 GPIO Simulating SPI to Read and Write External Flash

5.1 Hardware design

Hardware design required for this experiment:

(1) External Flash module

(2) USB-to-TTL line

The external Flash module uses the W25Q64 model, and the hardware wiring diagram of the W25Q64 module and the APM32F103 development board is shown in Figure 7:

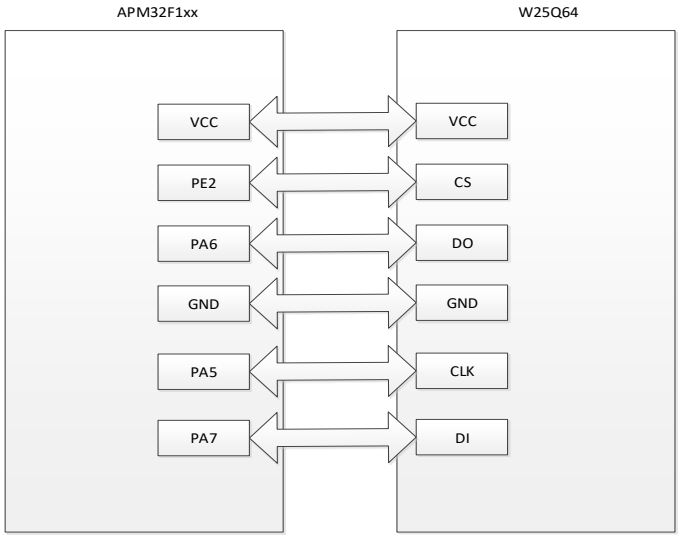

Figure 7 Hardware Wiring Diagram

The GPIO pins used for the USB-to-TTL cable are PA9 and PA10, and the USB-to-TTL line is used for data receiving and transmission. The schematic diagram is shown in Figure 8:

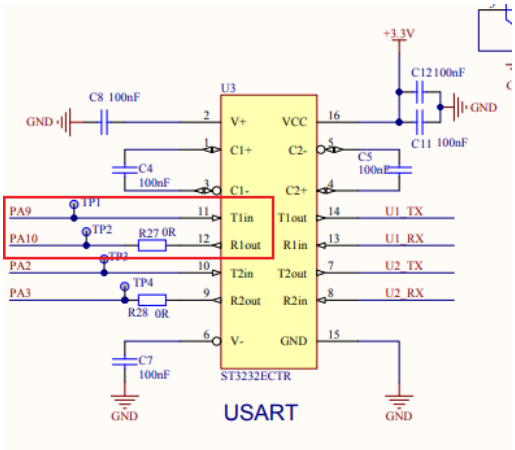

Figure 8 USART Wiring Schematic Diagram

5.2 Software design

To implement GPIO simulating PI to read and write external Flash through APM32F103, the ID information of the external Flash shall be configured according to the module model, the GPIO pin configuration design and SPI communication timing shall be completed, then the read and write operations can be performed, and the results of the read and write operations can be output. The code implementation flow chart is shown in Figure 9:

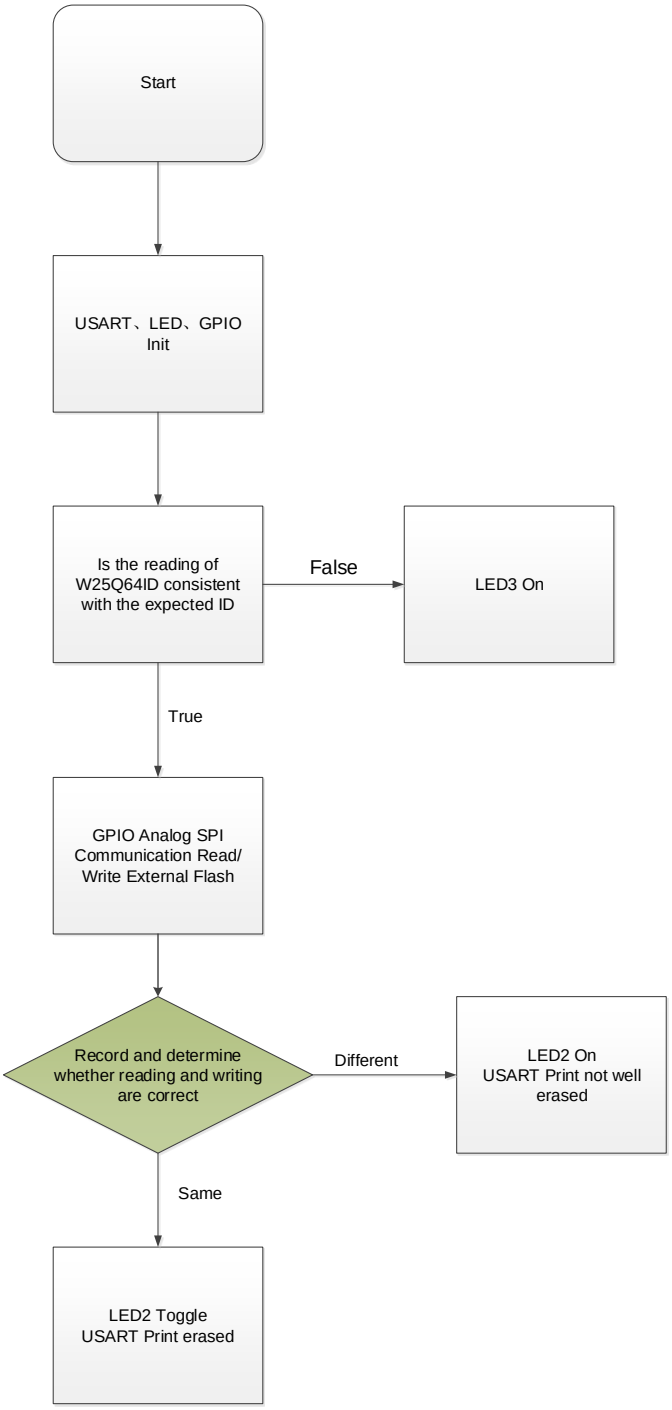

Figure 9 Main Program Flow Chart

5.2.1 SGPIO pin configuration

For the GPIO pins used to simulate SPI, configure the SCK, CS, and MOSI pins as push-pull output mode, and MISO as pull-down input mode. The reference codes are as follows:

```
void SPI_Init（void）
    {
```

```
    GPIO_Config_T   GPIO_ConfigStruct;

    RCM_EnableAPB2PeriphClock（RCM_APB2_PERIPH_GPIOA）;
    RCM_EnableAPB2PeriphClock（RCM_APB2_PERIPH_GPIOE）;

    GPIO_ConfigStruct.pin          = sFLASH_CS_PIN ;
    GPIO_ConfigStruct.mode          = GPIO_MODE_OUT_PP;
    GPIO_ConfigStruct.speed        = GPIO_SPEED_2MHz;
    GPIO_Config（GPIOE, &GPIO_ConfigStruct）; // SCK CS MOSI Output

    GPIO_ConfigStruct.pin          = sFLASH_SPI_SCK_PIN | sFLASH_SPI_MOSI_PIN ;
    GPIO_ConfigStruct.mode          = GPIO_MODE_OUT_PP;
    GPIO_ConfigStruct.speed        = GPIO_SPEED_2MHz;
    GPIO_Config（GPIOA, &GPIO_ConfigStruct）; // SCK CS MOSI Output

    GPIO_ConfigStruct.pin          = sFLASH_SPI_MISO_PIN;
    GPIO_ConfigStruct.mode          = GPIO_MODE_IN_FLOATING;
    GPIO_ConfigStruct.speed        = GPIO_SPEED_2MHz;
    GPIO_Config（GPIOA, &GPIO_ConfigStruct）; // MISO Input
    sFLASH_CS_HIGH;         // CS Init High
    sFLASH_SCK_LOW;         // SCK Init Low
}
```

5.2.2 Simulate read and write operations of SPI timing

As shown in the above Figure 5 错误!未找到引用源。, the transmission of SPI can be regarded as a virtual ring topology, indicating that the input and output are conducted at the same time. After the master transmits a clock signal and pulls down the chip selection signal, select the slave to transmit. According to the introduction of Figure 2 错误!未找到引用源。and Figure 3 错误!未找到引用源。, SPI mode 0 is selected here, the clock is at low level in idle state, and the data is collected at the rising edge and moved out at the falling edge. The reference codes are as follows:

```
uint8_t sFLASH_TxRxOneByte（uint8_t byte）
    {
        uint8_t temp = 0;
        uint8_t rx_data = 0xFF;
        for（uint8_t j=0;j<8;j++）
        {
            temp = （（byte&0x80）==0x80）? 1:0;
            byte = byte<<1;
            rx_data = rx_data<<1;

            sFLASH_SCK_LOW;
            if（temp == 1）
            {
                sFLASH_MOSI_HIGH;
            }
            else
            {
                sFLASH_MOSI_LOW;
            }
            Delay（10）;
            sFLASH_SCK_HIGH;
            Delay（10）;
            if（sFLASH_MISO_READ == 1）
            {
                rx_data = rx_data + 1;
            }
        }
        sFLASH_SCK_LOW;
        return rx_data;
    }
```

According to the codes, it is learned that:

(1) Each time a Byte is read and written, it will be cycled for 8 times, and a Bit is received and transmitted each time. In the circle, the most significant bit of Byte is stored in the

temp variable each time;

(2) After the most significant bit of Byte is stored, the second most significant bit of Byte will become the most significant bit. The operation performed here is to move Byte to the left by one bit.

(3) Move rx_data to the right by one bit, move the least significant bit, and after 8 cycles, rx_data will put the high bit in front.

(4) Then the idle level of the clock is set to low by pulling down the clock line.

(5) At the same time, set the relevant pin level of the MOSI according to the value stored in the temp variable at the current most significant bit of Byte.

(6) Then set the idle level of the clock to high by pulling up the clock line. Then, starting from the device, read the data transmitted by the MOSI and write the read data to the MISO.

(7) The least significant bit in the rx_data variable is used to store the data read on MISO.

(8) After reading and writing the SPI, set the idle level of the clock to low, pull down the clock line, and then the SPI enters the idle state.

5.2.3 W25Q64 device information read operation

W25Q64 mainly has three IDs: Manufacturer ID, Device ID, and Unique ID. The manufacturer ID is produced by Winbond, and its value MF [7:0] is 0xEF. The device ID consists of two parts, ID [15:0] is the chip type and 0x4017. ID [7:0] is the storage capacity. The unique ID indicates the uniqueness of the chip and is commonly used for encryption. The introduction of the W25Q64 ID content is shown in Table 2. Operate the development board to read the W25Q64 ID as shown in Table 3:

Type And Instruct	Address Content and Operation Instruction	
MANUFACTURER ID	（MF7-MF0）	-
Winbond Serial Flash	EFh	
Device ID	（ID7-ID0）	（ID15-ID0）
Instruction	ABh,90h,92h,94h	9Fh
W25Q64JV-IQ/JQ	16h	4017h

Table 2 Introduction to W25Q64 ID Content

Read Type	Instruct	Address Content				
Release Power-down/ID	ABh	Dummy	Dummy	Dummy	（ID7-ID0）	-
Manufacturer/Device ID	90h	Dummy	Dummy	00h	（MF7-MF0）	（ID7-ID0）
JEDEC ID	9Fh	（MF7-MF0）	（ID15-ID8）	（ID7-ID0）	-	-
Read Unique ID	4Bh	Dummy	Dummy	Dummy	Dummy	（UID63-0）

Table 3 W25Q64 Reading ID Operation Instructions

In the code implementation process, use the 0xAB instruction to obtain the device ID [7:0] and use the 0x9F instruction to obtain the JEDEC ID (MF [7:0] + ID [15:0]). In the implementation process, first pull down the/CS pin, and then send "ABh" and "9Fh" to the chip through DI. After "ABh" is sent, there will be a time interval of tRES1, and then the chip will resume normal operation. It is invalid to execute this instruction during the execution cycle of programming, erasing, and writing status register instructions. After "9Fh" is sent, information of the three IDs will be sent out from the DO pin on the falling edge of SCK. The reference codes are as follows:

```
uint32_t sFLASH_ReadDeviceID（void）
{
    uint32_t Temp[4];

    sFLASH_CS_LOW;
    sFLASH_TxRxOneByte（0xAB）;
    Temp[0] = sFLASH_TxRxOneByte（sFLASH_DUMMY_BYTE）;
    Temp[1] = sFLASH_TxRxOneByte（sFLASH_DUMMY_BYTE）;
    Temp[2] = sFLASH_TxRxOneByte（sFLASH_DUMMY_BYTE）;
    Temp[3] = sFLASH_TxRxOneByte（sFLASH_DUMMY_BYTE）;
    sFLASH_CS_HIGH;
```

```
    return Temp[3];
}
```

```
uint32_t sFLASH_ReadID（void）
{
    uint32_t Temp = 0, Temp0 = 0, Temp1 = 0, Temp2 = 0;

    sFLASH_CS_LOW;
    sFLASH_TxRxOneByte（0x9F）;
    Temp0 = sFLASH_TxRxOneByte（sFLASH_DUMMY_BYTE）;
    Temp1 = sFLASH_TxRxOneByte（sFLASH_DUMMY_BYTE）;
    Temp2 = sFLASH_TxRxOneByte（sFLASH_DUMMY_BYTE）;
    sFLASH_CS_HIGH;

    Temp = （Temp0 << 16） | （Temp1 << 8） | Temp2;
    return Temp;
}
```

5.2.4 Sector erase operation

According to 错误!未找到引用源。 when the 0xD8 instruction is sent, a 24-bit sector address is sent to erase all sector contents in the entire block. After erasing, all bytes become "FFh". The reference codes are as follows:

```
void sFLASH_SectorErase（uint32_t SectorAddr）
{
    sFLASH_WriteEnable（）;

    sFLASH_CS_LOW;
    sFLASH_TxRxOneByte（0xD8）;
    sFLASH_TxRxOneByte（（SectorAddr & 0xFF0000） >> 16）;
    sFLASH_TxRxOneByte（（SectorAddr & 0xFF00） >> 8）;
    sFLASH_TxRxOneByte（SectorAddr & 0xFF）;
    sFLASH_CS_HIGH;

    /* Wait FLASH completed */
    sFLASH_WaitForWriteEnd（）;
}
```

5.2.5 Page write data operation

W25Q64 can only write data by page, with 256 bytes per page, and a maximum of 256 bytes can be written at a time. According to 错误!未找到引用源。 when writing data, first send the 0x02 instruction, and then send a 24-bit sector address to perform the write operation on the sector. The reference codes are as follows:

```
void sFLASH_PageProgram（uint8_t* pBuffer, uint32_t WriteAddr, uint16_t NumByteToWrite）
    {
        sFLASH_WriteEnable（）;

        sFLASH_CS_LOW;

        sFLASH_TxRxOneByte（sFLASH_CMD_WRITE）;
        sFLASH_TxRxOneByte（（WriteAddr & 0xFF0000） >> 16）;
        sFLASH_TxRxOneByte（（WriteAddr & 0xFF00） >> 8）;
        sFLASH_TxRxOneByte（WriteAddr & 0xFF）;
        while （NumByteToWrite--）
        {
            sFLASH_TxRxOneByte（*pBuffer）;
            pBuffer++;
        }
        sFLASH_CS_HIGH;
        sFLASH_WaitForWriteEnd（）;
    }
```

5.2.6 Read data operation

According to 错误!未找到引用源。 when reading data, first send the 0x03 instruction, and then send a 24-bit sector address to perform read operation on the sector. The reference codes are as follows:

```
void sFLASH_ReadData（uint8_t* pBuffer, uint32_t ReadAddr, uint16_t NumByteToRead）
{
    sFLASH_CS_LOW;

    sFLASH_TxRxOneByte（sFLASH_CMD_READ）;
    sFLASH_TxRxOneByte（（ReadAddr & 0xFF0000） >> 16）;
    sFLASH_TxRxOneByte（（ReadAddr& 0xFF00） >> 8）;
    sFLASH_TxRxOneByte（ReadAddr & 0xFF）;
    while （NumByteToRead--）
    {
        *pBuffer = sFLASH_TxRxOneByte（sFLASH_DUMMY_BYTE）;
        pBuffer++;
    }
    sFLASH_CS_HIGH;
}
```

5.2.7 Wait for Flash to complete write operation

W25Q64 has three status registers, and bit [0] represents the erase/write status flag bit. According to 错误!未找到引用源。 when the write or read operation is over, judge the status register and send "05h" data to obtain the data of the status register. When the value of the status register bit [0] is 1, erase or write operation is still ongoing. When this bit is 0, it indicates that no operation is being performed and erase or write can be performed on W25Q64. The reference codes are as follows:

```
void sFLASH_WaitForWriteEnd（void）
    {
        uint8_t flashstatus = 0;

        sFLASH_CS_LOW;
        sFLASH_TxRxOneByte（0x05）;
        do
        {
            flashstatus = sFLASH_TxRxOneByte（sFLASH_DUMMY_BYTE）;
        }
        while （（flashstatus & sFLASH_WIP_FLAG） == SET）;
        sFLASH_CS_HIGH;
    }
```

5.2.8 Main function logical operation

Function introduction of this routine: After the program is downloaded, initialize and enable the pins of the light, serial port, and GPIO analog SPI. After that, perform read operation on the FlashID and DeviceID. After reading is completed, they will enter different states based on the judgment conditions. If the value read by FlashID is consistent with the defined ID, Flash will perform read and write operation. If they are inconsistent, LED3 will be normally on. In the read/write operation of Flash, before entering the write operation, perform an erase operation, and then data will be read and written respectively. After the operation is completed, compare the read and written data and put them in the variable 1. After that, perform an erase operation and then perform a read operation, compare with OXFF, and place the comparison in variable 2 to determine whether the erase operation is successful. Finally, compare the results of variables one by one, and the serial port and light will give different outputs. The reference codes are as follows:

```
int main（void）
{
    USART_Config_T usartConfig;

    APM_MINI_LEDInit（LED2）;
    APM_MINI_LEDInit（LED3）;
    APM_MINI_LEDOff（LED2）;
    APM_MINI_LEDOff（LED3）;

    usartConfig.baudRate = 115200;
    usartConfig.mode = USART_MODE_TX_RX;
    usartConfig.parity = USART_PARITY_NONE;
    usartConfig.stopBits = USART_STOP_BIT_1;
    usartConfig.wordLength = USART_WORD_LEN_8B;
    usartConfig.hardwareFlow = USART_HARDWARE_FLOW_NONE;
    APM_MINI_COMInit（COM1,&usartConfig）;

    //Init SPI GPIO
    SPI_Init（）;
    // Read SPI Device ID
    DeviceID = sFLASH_ReadDeviceID（）;
    /* Read SPI Flash ID */
    FlashID = sFLASH_ReadID（）;

    printf（"Device ID: 0x%x\r\n",DeviceID）;
    printf（"Flash ID: 0x%x\r\n",FlashID）;

    //whether is flash id
    if （FlashID == sFLASH_ID）
```

```
{
    APM_MINI_LEDOn（LED2）;
    //erase exist
    sFLASH_SectorErase（FLASH_SectorToErase）;
    //write data
    sFLASH_PageProgram（Tx_Buffer, FLASH_WriteAddress, BufferSize）;
    //read data
    sFLASH_ReadData（Rx_Buffer, FLASH_ReadAddress, BufferSize）;
    //compare write and read
    TransferStatus1 = Buffercmp（Tx_Buffer, Rx_Buffer, BufferSize）;
    //erase operate
    sFLASH_SectorErase（FLASH_SectorToErase）;
    //read again for check whether erase success
    sFLASH_ReadData（Rx_Buffer, FLASH_ReadAddress, BufferSize）;
    //compare
    for （Index = 0; Index < BufferSize; Index++）
    {
        if （Rx_Buffer[Index] != 0xFF）
        {
            TransferStatus2 = FAILED;
        }
    }
}
//if not flash id ,led3 on
else
{
    APM_MINI_LEDOn（LED3）;
}
//if erase success
if（TransferStatus2 == PASSED）
{
    printf（"the specified sector part is erased!!!\r\n"）;
}//not erase success
else{
    printf（"the specified sector part is not well erased!!!\r\n"）;
}
while（1）
{
    //compare for write and read whether same
    if（TransferStatus1 == PASSED）
    {
        APM_MINI_LEDToggle（LED2）;
    }
```

```
        Delay（0xfffff）;
    }
}
```

6 Revision history

Table 4 Document Revision History

Date	Version	Revision History
August 29, 2023	1.0	New edition

Statement

This document is formulated and published by Geehy Semiconductor Co., Ltd. (hereinafter referred to as “Geehy”). The contents in this document are protected by laws and regulations of trademark, copyright and software copyright. Geehy reserves the right to make corrections and modifications to this document at any time. Please read this document carefully before using Geehy products. Once you use the Geehy product, it means that you (hereinafter referred to as the “users”) have known and accepted all the contents of this document. Users shall use the Geehy product in accordance with relevant laws and regulations and the requirements of this document.

1. Ownership

This document can only be used in connection with the corresponding chip products or software products provided by Geehy. Without the prior permission of Geehy, no unit or individual may copy, transcribe, modify, edit or disseminate all or part of the contents of this document for any reason or in any form.

The “极海” or “Geehy” words or graphics with “®” or “™” in this document are trademarks of Geehy. Other product or service names displayed on Geehy products are the property of their respective owners.

2. No Intellectual Property License

Geehy owns all rights, ownership and intellectual property rights involved in this document.

Geehy shall not be deemed to grant the license or right of any intellectual property to users explicitly or implicitly due to the sale or distribution of Geehy products or this document.

If any third party’s products, services or intellectual property are involved in this document, it shall not be deemed that Geehy authorizes users to use the aforesaid third party’s products, services or intellectual property, unless otherwise agreed in sales order or sales contract.

3. version update

Users can obtain the latest document of the corresponding models when ordering geehy products.

If the contents in this document are inconsistent with geehy products, the agreement in thesales order or the sales contract shall prevail.

4. information reliability

The relevant data in this document are obtained from batch test by geehy laboratory or cooperative third-party testing organization. However, clerical errors in correction or errors caused by differences in testing environment may occur inevitably. Therefore, users should understand that geehy does not bear any

responsibility for such errors that may occur in this document. The relevant data in this document are only used to guide users as performance parameter reference and do not constitute geehy's guarantee for any product performance.

Users shall select appropriate geehy products according to their own needs, and effectively verify and test the applicability of geehy products to confirm that geehy products meet their own needs, corresponding standards, safety or other reliability requirements. If loses are caused to users due to the user's failure to fully verify and test geehy products, geehy will not bear any responsibility.

5. legality

Users shall abide by all applicable local laws and regulations when using this document and the matching geehy products. Users shall understand that the products may be restricted by the export, re-export or other laws of the countiries of the products suppliers, geehy, geehy distributors and users.

Users (on behalf or itself, subsidiaries and affiliated enterprises) shall agree and promise to abide by all applicable laws and regulations on the export and re-export of geehy products and/or technologies and direct products.

6. disclaimer of warranty

This document is provided by geehy "as is" and there is no warranty of any kind, either expressed or implied, including, but not limited to, the warranties of merchantability and fitness for a particular purpose, to the extent permitted by applicable law.

Geehy will bear no responsibility for any disputes arising from the subsequent design or use by users.

7. limitation of liability

In no event unless required by applicable law or agreed to in writing will geehy or any other party who provide the document "as is", be liable for damages, including any general, special, direct, incidental or consequential damages arising out of the use or inability to use the document (including but not limited to loss of data or data being rendered inaccurate or losses sustained by users or third parties).

8. scope of application

The information in this document replaces the information provided in all previous versions of the document.