零死角玩转STM32

与野火同行 乐意惬无边

1 原创教程，完全开源。

2 通俗易懂，详尽解读。

3 由浅入深，结合实操。

4 配套板子，全面玩转。

5 强强联合，不断更新。

野火团队 Wild Fire Team

0、友情提示

《零死角玩转 STM32》系列教程由初级篇、中级篇、高级篇、系统篇、四个部分组成，根据野火 STM32 开发板旧版教程升级而来，且经过重新深入编写，重新排版，更适合初学者，步步为营，从入门到精通，从裸奔到系统，让您零死角玩转 STM32。M3 的世界，与野火同行，乐意惬无边。

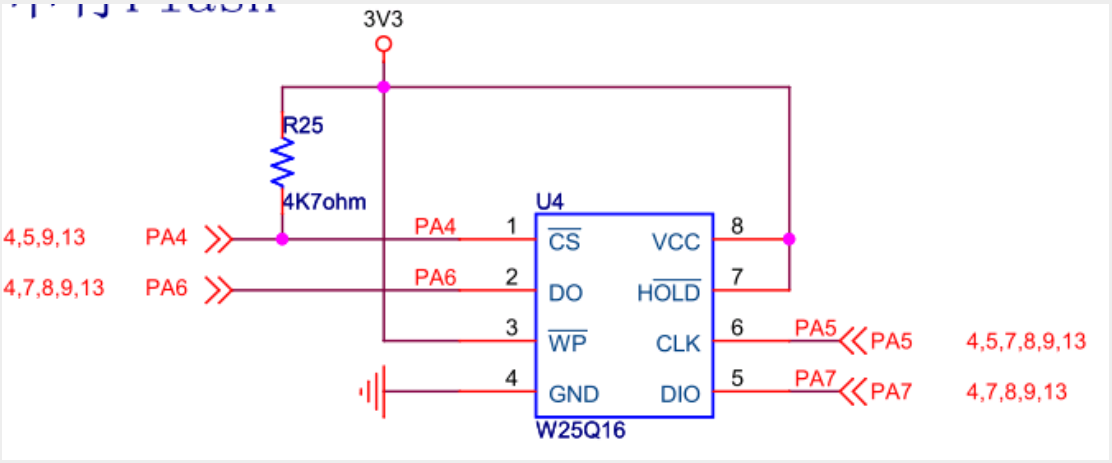
另外，野火团队历时一年精心打造的《STM32 库开发实战指南》将于今年 10 月份由机械工业出版社出版，该书的排版更适于纸质书本阅读以及更有利于查阅资料。内容上会给你带来更多的惊喜。是一本学习 STM32 必备的工具书。敬请期待！

6、SPI（2M-Flash）

6.1 实验描述及工程文件清单

实验描述	读取 FLASH 的 ID 信息，写入数据，并读取出来进行校验， 通过串口打印写入与读取出来的数据，输出测试结果。
硬件连接	PA4-SPI1-NSS ：W25X16-CS PA5-SPI1-SCK ：W25X16-CLK PA6-SPI1-MISO：W25X16-DO PA7-SPI1-MOSI：W25X16-DIO
用到的库文件	startup/start_stm32f10x_hd.c CMSIS/core_cm3.c CMSIS/system_stm32f10x.c FWlib/stm32f10x_gpio.c FWlib/stm32f10x_rcc.c FWlib/stm32f10x_usart.c FWlib/stm32f10x_spi.c
用户编写的文件	USER/main.c USER/stm32f10x_it.c USER/usart1.c USER/ spi_flash.c

野火 STM32 开发板 I2C-EEPROM 硬件原理图：



6.2 SPI 简介

SPI 是一种串行同步通讯协议，由一个主设备和一个或多个从设备组成，主设备启动一个与从设备的同步通讯，从而完成数据的交换。该总线大量用在与 EEPROM、ADC、FRAM 和显示驱动器之类的慢速外设器件通信。stm32 的 SPI 可以工作在全双工，单向发送，单向接收模式，可以使用 DMA 方式操作。

野火 STM32 开发板用的是 STM32F103VET6。它有 2 个 SPI 接口。本实验使用 SPI1，各信号线相应连接到 FLASH（型号：W25X16）的 CS、CLK、DO 和 DIO 线。实现 SPI 通讯，对 FLASH 进行读写。

本实验采用主模式，全双工通讯。

通过查询发送数据寄存器和接收数据寄存器状态确保通讯正常。

6.3 代码分析

首先要添加用的库文件，在工程文件夹下 Fwlib 下我们需添加以下库文件：

```
1. stm32f10x_gpio.c
2. stm32f10x_rcc.c
3. stm32f10x_usart.c
4. stm32f10x_spi.c
```

还要在 stm32f10x_conf.h 中把相应的头文件添加进来：

```
#include "stm32f10x_gpio.h"
#include "stm32f10x_rcc.h"
#include "stm32f10x_spi.h"
#include "stm32f10x_usart.h"
```

配置好所需的库文件之后，我们就从 main 函数开始分析：

```
/*
 * 函数名：main
 * 描述  ：主函数
 * 输入  ：无
 * 输出  ：无
 */
int main(void)
{
  /* 配置系统时钟为 72M */
    SystemInit();

  /* 配置串口 1 为：115200 8-N-1 */
    USART1_Config();
    printf("\r\n 这是一个 2M 串行 flash(W25X16)实验 \r\n");

    /* 2M 串行 flash W25X16 初始化 */
  SPI_FLASH_Init();

  /* Get SPI Flash Device ID */
  DeviceID = SPI_FLASH_ReadDeviceID();

    Delay( 200 );

  /* Get SPI Flash ID */
  FlashID = SPI_FLASH_ReadID();

  printf("\r\n FlashID is 0x%X,  Manufacturer Device ID is 0x%X\r\n",
FlashID, DeviceID);

  /* Check the SPI Flash ID */
  if (FlashID == sFLASH_ID)  /* #define  sFLASH_ID  0xEF3015 */
  {

    printf("\r\n 检测到华邦串行 flash W25X16 !\r\n");

    /* Erase SPI FLASH Sector to write on */
    SPI_FLASH_SectorErase(FLASH_SectorToErase);

    /* 将发送缓冲区的数据写到 flash 中 */
    SPI_FLASH_BufferWrite(Tx_Buffer, FLASH_WriteAddress, BufferSize);

        printf("\r\n 写入的数据为：%s \r\t", Tx_Buffer);

    /* 将刚刚写入的数据读出来放到接收缓冲区中 */
    SPI_FLASH_BufferRead(Rx_Buffer, FLASH_ReadAddress, BufferSize);
        printf("\r\n 读出的数据为：%s \r\n", Tx_Buffer);

    /* 检查写入的数据与读出的数据是否相等 */
    TransferStatus1 = Buffercmp(Tx_Buffer, Rx_Buffer, BufferSize);

        if( PASSED == TransferStatus1 )
    {
        printf("\r\n 2M 串行 flash(W25X16)测试成功!\n\r");
    }
```

```
53.     else
54.     {
55.         printf("\r\n 2M 串行 flash(W25X16)测试失败!\n\r");
56.     }
57.   }// if (FlashID == sFLASH_ID)
58.   else
59.   {
60.     printf("\r\n 获取不到 W25X16 ID!\n\r");
61.   }
62.
63.   SPI_Flash_PowerDown();
64.     while(1);
65. }
```

系统库函数 SystemInit(); 将系统时钟设置为 72M， USART1_Config(); 配置串口，关于这两个函数的具体讲解可以参考前面的教程，这里不再详述。

```
1.  /*****************************************************************
2.  * Function Name  : SPI_FLASH_Init
3.  * Description    : Initializes the peripherals used by the SPI FLASH d
    river.
4.  * Input          : None
5.  * Output         : None
6.  * Return         : None
7.  *****************************************************************/
8.  void SPI_FLASH_Init(void)
9.  {
10.   SPI_InitTypeDef  SPI_InitStructure;
11.   GPIO_InitTypeDef GPIO_InitStructure;
12.
13.   /* Enable SPI1 and GPIO clocks */
14.   /*!< SPI_FLASH_SPI_CS_GPIO, SPI_FLASH_SPI_MOSI_GPIO,
15.        SPI_FLASH_SPI_MISO_GPIO, SPI_FLASH_SPI_DETECT_GPIO
16.        and SPI_FLASH_SPI_SCK_GPIO Periph clock enable */
17.   RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA | RCC_APB2Periph_GPIOD,
    ENABLE);
18.
19.   /*!< SPI_FLASH_SPI Periph clock enable */
20.   RCC_APB2PeriphClockCmd(RCC_APB2Periph_SPI1, ENABLE);
21.   /*!< AFIO Periph clock enable */
22.   RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO, ENABLE);
23.
24.
25.   /*!< Configure SPI_FLASH_SPI pins: SCK */
26.   GPIO_InitStructure.GPIO_Pin = GPIO_Pin_5;
27.   GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
28.   GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
29.   GPIO_Init(GPIOA, &GPIO_InitStructure);
30.
31.   /*!< Configure SPI_FLASH_SPI pins: MISO */
32.   GPIO_InitStructure.GPIO_Pin = GPIO_Pin_6;
33.   GPIO_Init(GPIOA, &GPIO_InitStructure);
34.
35.   /*!< Configure SPI_FLASH_SPI pins: MOSI */
36.   GPIO_InitStructure.GPIO_Pin = GPIO_Pin_7;
37.   GPIO_Init(GPIOA, &GPIO_InitStructure);
38.
39.   /*!< Configure SPI_FLASH_SPI_CS_PIN pin: SPI_FLASH Card CS pin */

40.   GPIO_InitStructure.GPIO_Pin = GPIO_Pin_4;
41.   GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
42.   GPIO_Init(GPIOA, &GPIO_InitStructure);
43.
44.   /* Deselect the FLASH: Chip Select high */
```

```
45.   SPI_FLASH_CS_HIGH();
46.
47.   /* SPI1 configuration */
48.   // W25X16: data input on the DIO pin is sampled on the rising edge o
    f the CLK.
49.   // Data on the DO and DIO pins are clocked out on the falling edge o
    f CLK.
50.   SPI_InitStructure.SPI_Direction = SPI_Direction_2Lines_FullDuplex;

51.   SPI_InitStructure.SPI_Mode = SPI_Mode_Master;
52.   SPI_InitStructure.SPI_DataSize = SPI_DataSize_8b;
53.   SPI_InitStructure.SPI_CPOL = SPI_CPOL_High;
54.   SPI_InitStructure.SPI_CPHA = SPI_CPHA_2Edge;
55.   SPI_InitStructure.SPI_NSS = SPI_NSS_Soft;
56.   SPI_InitStructure.SPI_BaudRatePrescaler = SPI_BaudRatePrescaler_4;

57.   SPI_InitStructure.SPI_FirstBit = SPI_FirstBit_MSB;
58.   SPI_InitStructure.SPI_CRCPolynomial = 7;
59.   SPI_Init(SPI1, &SPI_InitStructure);
60.
61.   /* Enable SPI1  */
62.   SPI_Cmd(SPI1, ENABLE);
63. }
```

SPI_FLASH_Init()是用户编写的函数，调用 GPIO_Init()配置好 SPI 所用的 I/O 端口复用（CS 端口为普通 IO），调用 SPI_Init()来设置 SPI 的工作模式。并使能相关外设的时钟。其中 CPOL 和 CPHA 用来设置 SPI 数据采样条件，根据 FLASH 的数据手册（光盘中附带资料）。

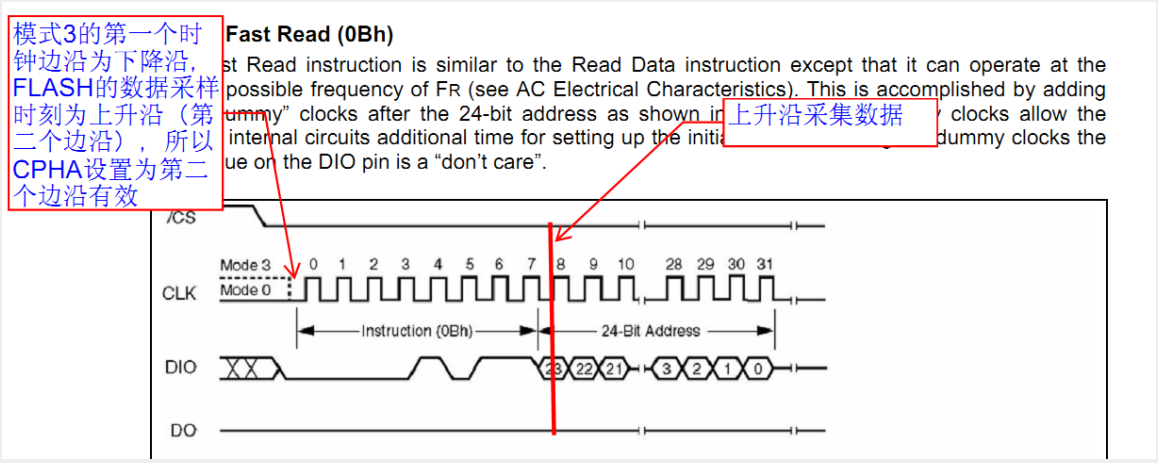
截图：

10.1 SPI OPERATIONS

10.1.1 SPI Modes

The W25X16/32/64 is accessed through an SPI compatible bus consisting of four signals: Serial Clock (CLK), Chip Select (/CS), Serial Data Input/Output (DIO) and Serial Data Output (DO). Both SPI bus operation Modes 0 (0,0) and 3 (1,1) are supported. The primary difference between Mode 0 and Mode 3 concerns the normal state of the CLK signal when the SPI bus master is in standby and data is not being transferred to the Serial Flash. For Mode 0 the CLK signal is normally low. For Mode 3 the CLK signal is normally high. In either case data input on the DIO pin is sampled on the rising edge of the CLK. Data on the DO and DIO pins are clocked out on the falling edge of CLK.

截图：



这个 FLASH 支持以模式 0 和模式 3 通讯。在通讯前 CLK 既可以一直为低电平（模式 0），也可以一直为高电平（模式 3），而数据采样时刻为上升沿；也就是说野火的 `CPOL=SPI_CPOL_High` 指令，选择了模式 3，默认情况下时钟为高。因为 FLASH 只在上升沿采集数据，在时钟默认为高的情况下，第二个边沿为上升沿。所以 `CPHA = SPI_CPHA_2Edge` 。CPHA 相应地设置为第二个时钟边沿（上升沿）为数据采样时刻。

```
/*************************************************************
* Function Name  : SPI_FLASH_ReadID
* Description    : Reads FLASH identification.
* Input          : None
* Output         : None
* Return         : FLASH identification
*************************************************************/
u32 SPI_FLASH_ReadDeviceID(void)
{
  u32 Temp = 0;
  /* Select the FLASH: Chip Select low */
  SPI_FLASH_CS_LOW();
  /* Send "RDID " instruction */
  SPI_FLASH_SendByte(W25X_DeviceID);
  SPI_FLASH_SendByte(Dummy_Byte);
  SPI_FLASH_SendByte(Dummy_Byte);
  SPI_FLASH_SendByte(Dummy_Byte);
  /* Read a byte from the FLASH */
  Temp = SPI_FLASH_SendByte(Dummy_Byte);
  /* Deselect the FLASH: Chip Select high */
  SPI_FLASH_CS_HIGH();
  return Temp;
}
```

接下来的是这个读 FLASH 器件 ID 的函数。实质是通过 SPI 接口向 FLASH 输入命令。以下是 FLASH 的各种命令：

11.2.2 Instruction Set (1)

INSTRUCTION NAME	BYTE 1 CODE	BYTE 2	BYTE 3	BYTE 4	BYTE 5	BYTE 6	N-BYTES
Write Enable	06h						
Write Disable	04h						
Read Status Register	05h	(S7–S0)(1)					(2)
Write Status Register	01h	S7–S0					
Read Data	03h	A23–A16	A15–A8	A7–A0	(D7–D0)	(Next byte)	continuous
Fast Read	0Bh	A23–A16	A15–A8	A7–A0	dummy	(D7–D0)	(Next Byte) continuous
Fast Read Dual Output	3Bh	A23–A16	A15–A8	A7–A0	dummy	I/O = (D6,D4,D2,D0) O = (D7,D5,D3,D1)	(one byte per 4 clocks, continuous)
Page Program	02h	A23–A16	A15–A8	A7–A0	(D7–D0)	(Next byte)	Up to 256 bytes
Block Erase (64KB)	D8h	A23–A16	A15–A8	A7–A0			
Sector Erase (4KB)	20h	A23–A16	A15–A8	A7–A0			
Chip Erase	C7h						
Power-down	B9h						
Release Power-down / Device ID	ABh	dummy	dummy	dummy	(ID7-ID0)(4)		
Manufacturer/ Device ID (3)	90h	dummy	dummy	00h	(M7-M0)	(ID7-ID0)	
JEDEC ID	9Fh	(M7-M0) Manufacturer	(ID15-ID8) Memory Type	(ID7-ID0) Capacity			

在 datasheet 的后面有这些命令的详细解释。

下面以 `SPI_FLASH_ReadDeviceID()` 为例讲解指令表：

1. `SPI_FLASH_CS_LOW()`，这是一个自定义的宏拉低 CS 端口，以使能 FLASH 器件；
2. 利用 `SPI_FLASH_SendByte()`（后面再详细分析这个函数的具体实现）来向 FLASH 发送“W25X_DeviceID”（0xAB）的命令；
3. 根据指令表，发送完这个指令后，后面紧跟着三个字节的“dummy byte”意思是任意数据，野火把 Dummy_Byte 宏定义为“0xff”，实际上改成其它数据也无影响。
4. 在第 5 字节 FLASH 在 DIO 端口输出它的器件的 ID7-ID0 位，stm32 调用 `SPI_FLASH_SendByte()` 返回数据。
5. CS 端口拉高，结束通讯。

可以看到实验例程通过串口打印出来的跟表中的一样喔。

11.2.1 Manufacturer and Device Identification

MANUFACTURER ID	(M7-M0)	
Winbond Serial Flash	EFH	
Device ID	**(ID7-ID0)**	**(ID15-ID0)**
Instruction	**ABh, 90h**	**9Fh**
W25X16	14h	3015h
W25X32	15h	3016h
W25X64	16h	3017h

了解这个读器件流程后我们再来具体分析 SPI_FLASH_SendByte()。

```
1.  /*******************************************************************
2.  * Function Name  : SPI_FLASH_SendByte
3.  * Description    : Sends a byte through the SPI interface and return t
    he byte
4.  *                  received from the SPI bus.
5.  * Input          : byte : byte to send.
6.  * Output         : None
7.  * Return         : The value of the received byte.
8.  *******************************************************************/
9.  u8 SPI_FLASH_SendByte(u8 byte)
10. {
11.   /* Loop while DR register in not emplty */
12.   while (SPI_I2S_GetFlagStatus(SPI1, SPI_I2S_FLAG_TXE) == RESET);
13.
14.   /* Send byte through the SPI1 peripheral */
15.   SPI_I2S_SendData(SPI1, byte);
16.
17.   /* Wait to receive a byte */
18.   while (SPI_I2S_GetFlagStatus(SPI1, SPI_I2S_FLAG_RXNE) == RESET);
19.
20.   /* Return the byte read from the SPI bus */
21.   return SPI_I2S_ReceiveData(SPI1);
22. }
```

1. 调用库函数 SPI_I2S_GetFlagStatus()等待发送数据寄存器清空。
2. 发送数据寄存器准备好后，调用库函数 SPI_I2S_GetFlagStatus()发送数据。
3. 调用库函数 SPI_I2S_GetFlagStatus()等待接收数据寄存器非空。
4. 接收寄存器非空，调用 SPI_I2S_ReceiveData()返回 DIO 端口接收回来的数据。

这是最底层的发送数据和接收数据的函数，只有几句代码，很好理解。

回到 main 函数中，其它的一些命令函数，如：SPI_FLASH_ReadID()、SPI_FLASH_SectorErase()和在 SPI_FLASH_BufferWrite()中调用的 SPI_FLASH_PageWrite()。

它们的具体实现都跟读器件 ID 的类似，参考指令表了解一下流程就可以理解。

其中指令表中的 A0~A23 指地址；M0~M7 为器件的 制造商 ID（MANUFACTURER ID）；D0~D7 为数据。

根据 FLASH 的存储原理，在写入数据前，要先对存储区域进行擦除，所以执行 SPI_FLASH_SectorErase()函数对要写入的扇区进行擦除，预写。

最后是 SPI_FLASH_BufferWrite()和 SPI_FLASH_BufferRead()函数，我们可以分别调用它们来把缓冲区数据写入 FLASH 和从 FLASH 读出数据到缓冲区。具体实现跟 I2C-EEPROM 的很类似，处理好地址后就执行命令进行读或写就行了。提一下这跟 EEPROM 的区别，这个

截图：

11.2.10 Page Program (02h)

The Page Program instruction allows up to 256 bytes of data to be programmed at previously erased to all 1s (FFh) memory locations. A Write Enable instruction must be executed before the device will accept the Page Program Instruction (Status Register bit WEL must equal 1). The instruction is initiated by driving the /CS pin low then shifting the instruction code "02h" followed by a 24-bit address (A23-A0) and at least one data byte, into the DIO pin. The /CS pin must be held low for the entire length of the instruction while data is being sent to the device. The Page Program instruction sequence is shown in figure 11.

此 FLASH 的页最大字节数为 256 字节，同样地，超过页最大字节继续写入数据的话，数据会从该页的起始地址覆盖写入。

对于读数据，发出一个命令后，可以无限制地一直把整个 FLASH 的数据都读取完，不需要读取整个 FLASH 的则以 CS 拉高为命令的结束的标置。

截图：

11.2.7 Read Data (03h)

The Read Data instruction allows one more data bytes to be sequentially read from the memory. The instruction is initiated by driving the /CS pin low and then shifting the instruction code "03h" followed by a 24-bit address (A23-A0) into the DIO pin. The code and address bits are latched on the rising edge of the CLK pin. After the address is received, the data byte of the addressed memory location will be shifted out on the DO pin at the falling edge of CLK with most significant bit (MSB) first. The address is automatically incremented to the next higher address after each byte of data is shifted out allowing for a continuous stream of data. This means that the entire memory can be accessed with a single instruction as long as the clock continues. The instruction is completed by driving /CS high. The Read Data instruction sequence is shown in figure 8. If a Read Data instruction is issued while an Erase, Program or Write cycle is in process (BUSY=1) the instruction is ignored and will not have any effects on the current cycle. The Read Data instruction allows clock rates from D.C. to a maximum of f_R (see AC Electrical Characteristics).

最后，总结一下在 stm32 如何建立与 SPI-FLASH 的通讯。

1、 配置 I/O 端口,使能 GPIO；

2、 根据将要进行通讯器件的 SPI 模式，配置 stm32 的 SPI，使能 SPI 时钟；

3、 配置好后，可以发送各种 FLASH 命令。

注意：在写操作前要先进行存储扇区的擦除操作，擦除操作前要先发出“写使能”命令。

6.4 实验现象

将野火 STM32 开发板供电(DC5V)，插上 JLINK，插上串口线(两头都是母的交叉线)，打开超级终端，配置超级终端为 115200 8-N-1，将编译好的程序下载到开发板，即可看到超级终端打印出如下信息：