青风手把手教你学 stm32f030 系列教程

---------- 库函数操作版本

出品论坛： www.qfv8.com 青风电子社区

作者： 青风
出品论坛: www.qfv8.com
淘宝店：http://qfv5.taobao.com
QQ 技术群：241364123
硬件平台：QF-STM32F030 开发板

2.13 通过 SPI 读写 SD 卡

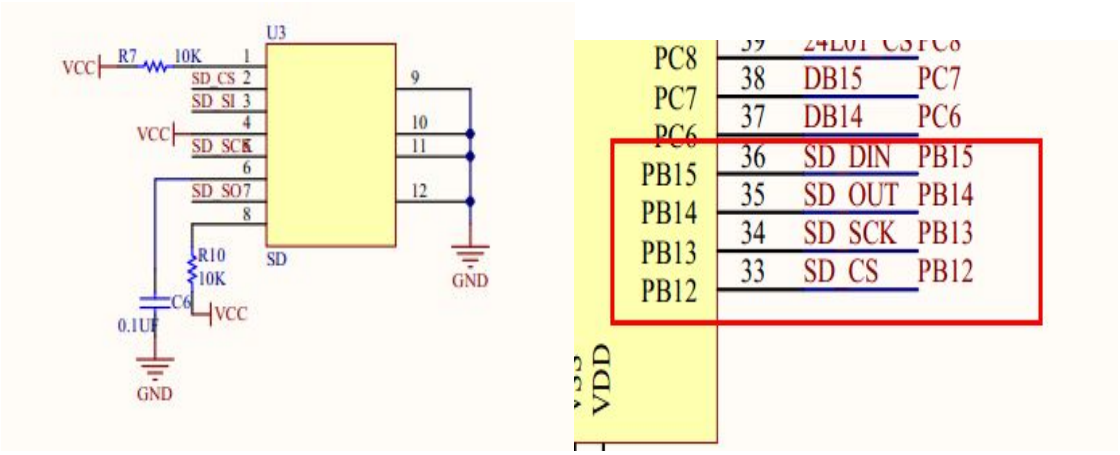
2.13.1 原理分析

很多单片机系统都需要大容量存储设备，以存储数据。目前常用的有 U 盘，FLASH 芯片，SD 卡等。他们各有优点，综合比较，最适合单片机系统的莫过于 SD 卡了，它不仅容量可以做到很大（32Gb 以上），而且支持 SPI 接口，方便移动，有几种体积的尺寸可供选择（标准的 SD 卡尺寸，以及 TF 卡尺寸），能满足不同应用的要求。只需要 4 个 IO 口，就可以外扩一个最大达 32GB 以上的外部存储器，容量选择尺度很大，更换也很方便，而且方便移动，编程也比较简单，是单片机大容量外部存储器的首选。SD 卡（Secure Digital Memory Card）中文翻译为安全数码卡，是一种基于半导体快闪记忆器的新一代记忆设备，它被广泛地于便携式装置上使用，例如数码相机、个人数码助理(PDA)和多媒体播放器等。Sd 卡的通信接口这里采用的是 SPI，SPI 接口的使用在前面读写 W25X16 时已经有了分析，这里来讨论下使用 SPI 来读写 SD 卡。

本节内容没有加入文件系统，直接采用 SPI 读写 SD 卡，从软硬件两个方面来学习如何配置：

2.13.2 硬件准备：

开发板的 SD 卡设置在液晶转接板上，其硬件电路如下图所示：



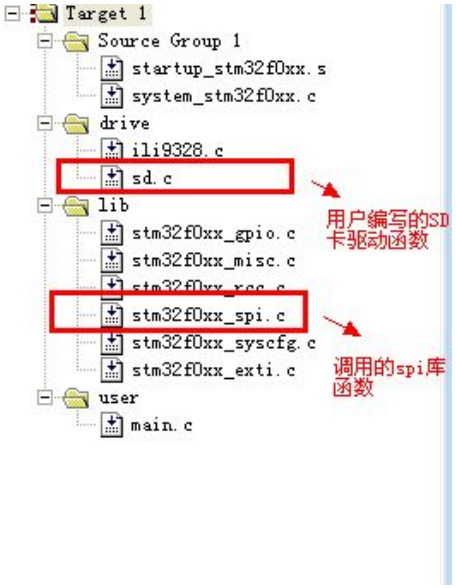
其端口配置入下所示：

硬件连接：SD_DIN---PB15
SD_OUT---PB14
SD_SCK---PB13
SD_CS---PB12

其中 SD_CS:sd 卡片选管脚，低电平有效
SD_SCK:sd 卡的时钟管脚
SD_DIN：sd 卡的 spi 输入管脚。
SD_OUT：sd 卡的 spi 输出管脚。

2.13.3 软件准备：

打开 keil 编译环境，设置系统工程树如下图所示：



如上所示，用户需要编写 SD 卡驱动函数，首先我们先来看看一个简单的 SD 卡测试主函数：

```
SD_Error Status = SD_RESPONSE_NO_ERROR ;
SD_CardInfo SDCardInfo;

int main (void)
{
    SystemInit();
    LCD_init();            // 液晶显示器初始化
    LCD_Clear(ORANGE);  // 全屏显示白色
    POINT_COLOR =BLACK; // 定义笔的颜色为黑色
    BACK_COLOR = WHITE   ;      // 定义笔的背景色为白色
      /*-------------------------- SD Init ---------------------------- */
  Status = SD_Init();

  if (Status == SD_RESPONSE_NO_ERROR )
  {
   /*----------------- Read CSD/CID MSD registers ------------------*/
   LCD_ShowString(20,20, "SD_Init is ok");
   Status = SD_GetCardInfo(&SDCardInfo);
  }
  else
    {
        LCD_ShowString(20,20,      "SD_Init is error");
        }
}
```

函数首先对 SD 卡进行了初始化，调用了 sd 卡初始化代码 SD_Init()，然后读取 sd 卡信息状态。首先我们来看看 SD 卡的初始化，代码如下：

```
SD_Error SD_Init(void)
{
  uint32_t i = 0;

  /*!< 初始化 SD_SPI */
  SD_SPI_Init();

  /*!< SD 片选写高 */
  SD_CS_HIGH();

  /*!< 发送无效字节 0xFF,  CS 至高 10 */
  /*!< CS 和 MOSI 上升为 80 个时钟周期*/
  for (i = 0; i <= 9; i++)
  {
    /*!< Send dummy byte 0xFF */
    SD_WriteByte(SD_DUMMY_BYTE);
  }

```

```
43.     /*------------Put SD in SPI mode--------------*/
44.     /*!< SD initialized and set to SPI mode properly */
45.     return (SD_GoIdleState());
46.   }
```

SD 卡的初始化，首先要对 SD 卡的硬件接口进行设置，SD 卡采用 SPI2,接口为 PB12--PB15

PB12	SPI2_NSS	EVENTOUT	TIM1_BKIN	TSC_G6_IO2
PB13	SPI2_SCK		TIM1_CH1N	TSC_G6_IO3
PB14	SPI2_MISO	TIM15_CH1	TIM1_CH2N	TSC_G6_IO4
PB15	SPI2_MOSI	TIM15_CH2	TIM1_CH3N	TIM15_CH1N

采用器复用功能 0，下面对其进行设置：

```
47.   void SD_SPI_Init(void)
48.   {
49.     GPIO_InitTypeDef    GPIO_InitStructure;
50.     SPI_InitTypeDef     SPI_InitStructure;
51.
52.     /*!< 初始化 SD 卡使用的 IO 端口的时钟  */
53.     RCC_AHBPeriphClockCmd(SD_CS_GPIO_CLK | SD_SPI_MOSI_GPIO_CLK |
    SD_SPI_MISO_GPIO_CLK |SD_SPI_SCK_GPIO_CLK , ENABLE);
54.
55.     /*!< SD_SPI  外设时钟使能  */
56.     RCC_APB1PeriphClockCmd(SD_SPI_CLK, ENABLE);
57.
58.     /*!<  配置 SD_SPI  管脚: SCK */
59.     GPIO_InitStructure.GPIO_Pin = SD_SPI_SCK_PIN;
60.     GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
61.     GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
62.     GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
63.     GPIO_InitStructure.GPIO_PuPd    = GPIO_PuPd_UP;
64.     GPIO_Init(SD_SPI_SCK_GPIO_PORT, &GPIO_InitStructure);
65.
66.     /*!<  配置  SD_SPI  管脚: MISO */
67.     GPIO_InitStructure.GPIO_Pin = SD_SPI_MISO_PIN;
68.     GPIO_Init(SD_SPI_MISO_GPIO_PORT, &GPIO_InitStructure);
69.
70.     /*!<  配置  SD_SPI  管脚: MOSI */
71.     GPIO_InitStructure.GPIO_Pin = SD_SPI_MOSI_PIN;
72.     GPIO_Init(SD_SPI_MOSI_GPIO_PORT, &GPIO_InitStructure);
73.
74.     /*!<配置  SD_SPI_CS_PIN  管脚: SD Card CS pin */
75.     GPIO_InitStructure.GPIO_Pin = SD_CS_PIN;
76.     GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;
77.     GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
78.     GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;
```

```
79.     GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
80.     GPIO_Init(SD_CS_GPIO_PORT, &GPIO_InitStructure);
81.
82.     /*配置 SPI 复用*/
83.     GPIO_PinAFConfig(SD_SPI_SCK_GPIO_PORT, SD_SPI_SCK_SOURCE, SD_SPI_SCK_AF);
84.     GPIO_PinAFConfig(SD_SPI_MISO_GPIO_PORT, SD_SPI_MISO_SOURCE,
    SD_SPI_MISO_AF);
85.     GPIO_PinAFConfig(SD_SPI_MOSI_GPIO_PORT, SD_SPI_MOSI_SOURCE,
    SD_SPI_MOSI_AF);
86.
87.     /*!< SD_SPI 配置参数  */
88.     SPI_InitStructure.SPI_Direction = SPI_Direction_2Lines_FullDuplex;
89.     SPI_InitStructure.SPI_Mode = SPI_Mode_Master;
90.     SPI_InitStructure.SPI_DataSize = SPI_DataSize_8b;
91.     SPI_InitStructure.SPI_CPOL = SPI_CPOL_High;
92.     SPI_InitStructure.SPI_CPHA = SPI_CPHA_2Edge;
93.     SPI_InitStructure.SPI_NSS = SPI_NSS_Soft;
94.     SPI_InitStructure.SPI_BaudRatePrescaler = SPI_BaudRatePrescaler_2;
95.
96.     SPI_InitStructure.SPI_FirstBit = SPI_FirstBit_MSB;
97.     SPI_InitStructure.SPI_CRCPolynomial = 7;
98.     SPI_Init(SD_SPI, &SPI_InitStructure);
99.
100.    SPI_RxFIFOThresholdConfig(SD_SPI, SPI_RxFIFOThreshold_QF);
101.
102.    SPI_Cmd(SD_SPI, ENABLE); /*!< SD_SPI enable */
103. }
```

然后编写 SD 卡信息检测函数，依次检测 sd 包含的信息，判断信息序列是否正确，可以按照下面方式进行编写：

```
104.    * @brief   返回有关特定卡的信息
105.    * @param    cardinfo: pointer to a SD_CardInfo structure that contains all SD
106.    *                card information.
107.    * @retval The SD Response:
108.    *                - SD_RESPONSE_FAILURE: Sequence failed
109.    *                - SD_RESPONSE_NO_ERROR: Sequence succeed
110.    */
111. SD_Error SD_GetCardInfo(SD_CardInfo *cardinfo)
112. {
113.    SD_Error status = SD_RESPONSE_FAILURE;
114.
115.    SD_GetCSDRegister(&(cardinfo->SD_csd));
116.    status = SD_GetCIDRegister(&(cardinfo->SD_cid));
117.    cardinfo->CardCapacity = (cardinfo->SD_csd.DeviceSize + 1) ;
118.    cardinfo->CardCapacity *= (1 << (cardinfo->SD_csd.DeviceSizeMul + 2));
```

```
119.    cardinfo->CardBlockSize = 1 << (cardinfo->SD_csd.RdBlockLen);
120.    cardinfo->CardCapacity *= cardinfo->CardBlockSize;
121.
122.    /*!< Returns the reponse */
123.    return status;
124. }
```

大家注意，SD 卡的整个信息，我们在 sd.h 中采用一个结构体表示 SD_CardInfo 来表示：

```
typedef struct
125. {
126.    SD_CSD SD_csd;     /*!< 卡的具体数据 */
127.    SD_CID SD_cid;          /*!< 存储卡标识数据 */
128.    uint32_t CardCapacity;   /*!< 卡片容量 */
129.    uint32_t CardBlockSize; /*!< 卡的块大小 */
130. } SD_CardInfo;
```

这个结构体中的成员 SD_CSD SD_csd，SD_CID SD_cid 我们也写成结构体的类型，这里表示了 SD 卡的几个重要信息。其详细定义可以在文件 《SD 卡协议（物理层）》中找到详细说明，这里就不再罗嗦了。

SD 卡给出一些基本操作命令，我们列出部分如下表所示：

表 1. 部分 SD 存储卡命令

Command	Mnemonic	Argument	Reply	Description
0 (0x00)	GO_IDLE_STATE	none	R1	Resets the SD card.
9 (0x09)	SEND_CSD	none	R1	Sends card-specific data.
10 (0x0a)	SEND_CID	none	R1	Sends card identification.
17 (0x11)	READ_SINGLE_BLOCK	address	R1	Reads a block at byte address.
24 (0x18)	WRITE_BLOCK	address	R1	Writes a block at byte address.
55 (0x37)	APP_CMD	none	R1	Prefix for application command.
59 (0x3b)	CRC_ON_OFF	Only Bit 0	R1	Argument sets CRC on (1) or off (0).
41 (0x29)	SEND_OP_COND	none	R1	Starts card initialization.

在 SD.H 文件中，我们需要对这些命令进行定义，这样在操作函数中可以直接使用：

```
131. #define SD_CMD_GO_IDLE_STATE                    0    /*!< CMD0 = 0x40 */
132. #define SD_CMD_SEND_OP_COND                     1    /*!< CMD1 = 0x41 */
133. #define SD_CMD_SEND_CSD                         9    /*!< CMD9 = 0x49 */
134. #define SD_CMD_SEND_CID                         10   /*!< CMD10 = 0x4A */
135. #define SD_CMD_STOP_TRANSMISSION                12   /*!< CMD12 = 0x4C */
136. #define SD_CMD_SEND_STATUS                      13   /*!< CMD13 = 0x4D */
137. #define SD_CMD_SET_BLOCKLEN                     16   /*!< CMD16 = 0x50 */
138. #define SD_CMD_READ_SINGLE_BLOCK                17   /*!< CMD17 = 0x51 */
139. #define SD_CMD_READ_MULT_BLOCK                  18   /*!< CMD18 = 0x52 */
140. #define SD_CMD_SET_BLOCK_COUNT                  23   /*!< CMD23 = 0x57 */
```

```
#define SD_CMD_WRITE_SINGLE_BLOCK       24   /*!< CMD24 = 0x58 */
#define SD_CMD_WRITE_MULT_BLOCK         25   /*!< CMD25 = 0x59 */
#define SD_CMD_PROG_CSD                 27   /*!< CMD27 = 0x5B */
#define SD_CMD_SET_WRITE_PROT           28   /*!< CMD28 = 0x5C */
#define SD_CMD_CLR_WRITE_PROT           29   /*!< CMD29 = 0x5D */
#define SD_CMD_SEND_WRITE_PROT          30   /*!< CMD30 = 0x5E */
#define SD_CMD_SD_ERASE_GRP_START       32   /*!< CMD32 = 0x60 */
#define SD_CMD_SD_ERASE_GRP_END         33   /*!< CMD33 = 0x61 */
#define SD_CMD_UNTAG_SECTOR             34   /*!< CMD34 = 0x62 */
#define SD_CMD_ERASE_GRP_START          35   /*!< CMD35 = 0x63 */
#define SD_CMD_ERASE_GRP_END            36   /*!< CMD36 = 0x64 */
#define SD_CMD_UNTAG_ERASE_GROUP        37   /*!< CMD37 = 0x65 */
#define SD_CMD_ERASE                    38   /*!< CMD38 = 0x66 */
```

完成这些定义之后，我们就就来编写 SD 卡的操作函数了。根据《SD 卡协议（物理层）》文件中的说明，SD 卡的操作可以分为下面三种类型：

```
/*!<SD 卡块操作 */
SD_Error SD_ReadBlock(uint8_t* pBuffer, uint32_t ReadAddr, uint16_t BlockSize);
SD_Error SD_ReadMultiBlocks(uint8_t* pBuffer, uint32_t ReadAddr, uint16_t BlockSize, uint32_t
    NumberOfBlocks);
SD_Error SD_WriteBlock(uint8_t* pBuffer, uint32_t WriteAddr, uint16_t BlockSize);
SD_Error SD_WriteMultiBlocks(uint8_t* pBuffer, uint32_t WriteAddr, uint16_t BlockSize, uint32_t
    NumberOfBlocks);

/*!<SD 寄存器相关操作 */
SD_Error SD_GetCSDRegister(SD_CSD* SD_csd);
SD_Error SD_GetCIDRegister(SD_CID* SD_cid);
void SD_SendCmd(uint8_t Cmd, uint32_t Arg, uint8_t Crc);
SD_Error SD_GetResponse(uint8_t Response);
uint8_t SD_GetDataResponse(void);
SD_Error SD_GoIdleState(void);
uint16_t SD_GetStatus(void);

/*!<SD 卡字节操作 */
uint8_t SD_WriteByte(uint8_t byte);
uint8_t SD_ReadByte(void);
```

下面我们来举其中一个例子， 从 SD 卡读取块数据，首先我们需要详细阅读《SD 卡协议（物理层）》，文件中给出了读取块数据的基本操作步骤如下图所示：

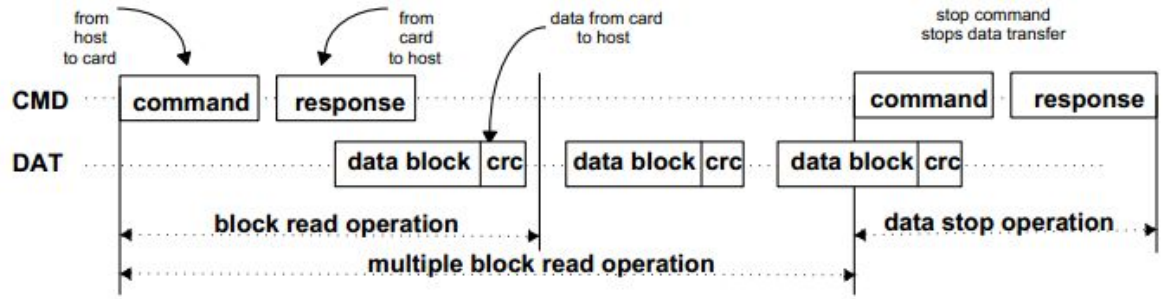


Figure 3-3: (Multiple) Block Read Operation

首先要发送读取命令，sd 卡应答无错误后开始传输数据，数据传输结束后再返回结束应答。基本就这 3 步。首先看看发送命令，文件中表示结构 48bit 如下图所示：

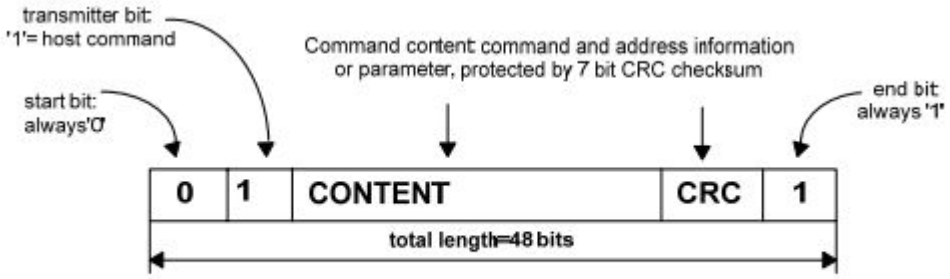


Figure 3-5: Command Token Format

根据这个方式编写发送命令函数，含 6 个字节：

```
171. void SD_SendCmd(uint8_t Cmd, uint32_t Arg, uint8_t Crc)
172. {
173.   uint32_t i = 0x00;
174.
175.   uint8_t Frame[6];
176.
177.   Frame[0] = (Cmd | 0x40); /*!< Construct byte 1 */
178.
179.   Frame[1] = (uint8_t)(Arg >> 24); /*!< Construct byte 2 */
180.
181.   Frame[2] = (uint8_t)(Arg >> 16); /*!< Construct byte 3 */
182.
183.   Frame[3] = (uint8_t)(Arg >> 8); /*!< Construct byte 4 */
184.
185.   Frame[4] = (uint8_t)(Arg); /*!< Construct byte 5 */
186.
187.   Frame[5] = (Crc); /*!< Construct CRC: byte 6 */
188.
189.   for (i = 0; i < 6; i++)
190.   {
191.     SD_WriteByte(Frame[i]); /*!< Send the Cmd bytes */
192.   }
193. }
```

Sd 卡的应答结构如下图所示：

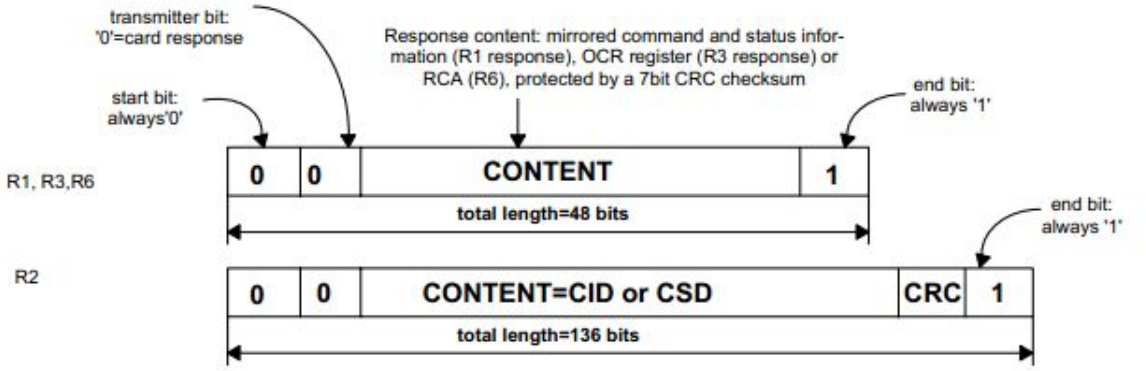


Figure 3-6: Response Token Format

因此根据上面所分析的三个步骤，读单个块数据的子函数编写代码如下图所示：

```
194. /**
195.   * @brief   从 SD 卡读取块数据.
196.   * @param   pBuffer:指向从 sd 卡读出的接收数据缓冲指针
197.   * @param   ReadAddr:sd 卡读取的内部地址.
198.   * @param   BlockSize: sd 卡块的大小.
199.   * @retval The SD Response:
200.   *              - SD_RESPONSE_FAILURE: Sequence failed
201.   *              - SD_RESPONSE_NO_ERROR: Sequence succeed
202.   */
203. SD_Error SD_ReadBlock(uint8_t* pBuffer, uint32_t ReadAddr, uint16_t BlockSize)
204. {
205.   uint32_t i = 0;
206.   SD_Error rvalue = SD_RESPONSE_FAILURE;
207.
208.   /*!< SD 片选置低*/
209.   SD_CS_LOW();
210.
211.   /*!< 发送命令 CMD17 (SD_CMD_READ_SINGLE_BLOCK) 去读取一个块 */
212.   SD_SendCmd(SD_CMD_READ_SINGLE_BLOCK, ReadAddr, 0xFF);
213.
214.   /*!< 监测 SD 卡识别读块命令: R1 response (0x00: no errors) */
215.   if (!SD_GetResponse(SD_RESPONSE_NO_ERROR))
216.   {
217.     /*!< 标示数据传送开始 */
218.     if (!SD_GetResponse(SD_START_DATA_SINGLE_BLOCK_READ))
219.     {
220.       /*!< 读取 SD 卡块数据   */
221.       for (i = 0; i < BlockSize; i++)
222.       {
223.         /*!保存接收数据值缓冲 */
224.         *pBuffer = SD_ReadByte();
```

```
225.              pBuffer++;
226.          }
227.          /*!< Get CRC bytes (not really needed by us, but required by SD) */
228.          SD_ReadByte();
229.          SD_ReadByte();
230.          /*!< 设置相应成功*/
231.          rvalue = SD_RESPONSE_NO_ERROR;
232.        }
233.    }
234.    /*!< SD 片选为高 */
235.    SD_CS_HIGH();
236.
237.    /*!< 发送空字节: 8 个时钟脉冲延迟 */
238.    SD_WriteByte(SD_DUMMY_BYTE);
239.
240.    /*!< 返回相应 */
241.    return rvalue;
242. }
```

主函数对 SD 进行测试：

```
243.  #include "stm32f0xx.h"
244. #include "sd.h"
245. #include "ili9328.h"
246.
247. SD_Error Status = SD_RESPONSE_NO_ERROR ;
248. SD_CardInfo SDCardInfo;
249.
250. int main (void)
251. {
252.       SystemInit();
253.       LCD_init();            // 液晶显示器初始化
254.       LCD_Clear(ORANGE);  // 全屏显示白色
255.       POINT_COLOR =BLACK; // 定义笔的颜色为黑色
256.       BACK_COLOR = WHITE    ;       // 定义笔的背景色为白色
257.          /*------------------------- SD Init ---------------------------- */
258.    Status = SD_Init();
259.
260.    if (Status == SD_RESPONSE_NO_ERROR )
261.    {
262.     /*----------------- Read CSD/CID MSD registers ------------------*/
263.     LCD_ShowString(20,20,  "SD_Init is ok");
264.     Status = SD_GetCardInfo(&SDCardInfo);
265.    }
266.    else
267.       {
```

```
268.            LCD_ShowString(20,20,      "SD_Init is error");
269.            }
270. }
```

这里面就简要的举了一个读取 SD 块的例子，整个 SD 卡的操作要严格按照其协议规定的时序进行书写，每个 SD 卡的操作都有相应的操作命令，大家自己编写代码的时候需要参考《SD 卡协议（物理层）》文件，在这里大家弄懂了我们怎么更加 SD 卡协议书写 SD 卡操作代码，我的任务就算完成了，当然这里面注意我们的代码只支持 2G 以内的 SD 卡，后续会更新超过 2G 的代码。谢谢大家指正。