STM32Cube高效开发教程（高级篇）

# 第13章 直接访问SD卡

王维波
中国石油大学（华东）控制科学与工程学院

# STM32Cube高效开发教程（高级篇）

作者：王维波，鄢志丹，王钊

人民邮电出版社

2022年2月出版


如果有读者需要本书课件的PPT版本用于备课，可以给作者发邮件免费获取，并可加入专门的教学和技术交流QQ群

邮箱：wangwb@upc. edu. cn

# 第13章 直接访问SD卡

13.1 SD卡简介

13.2 SDIO接口硬件电路

13.3 SDIO的HAL驱动程序

13.4 轮询方式读写SD卡

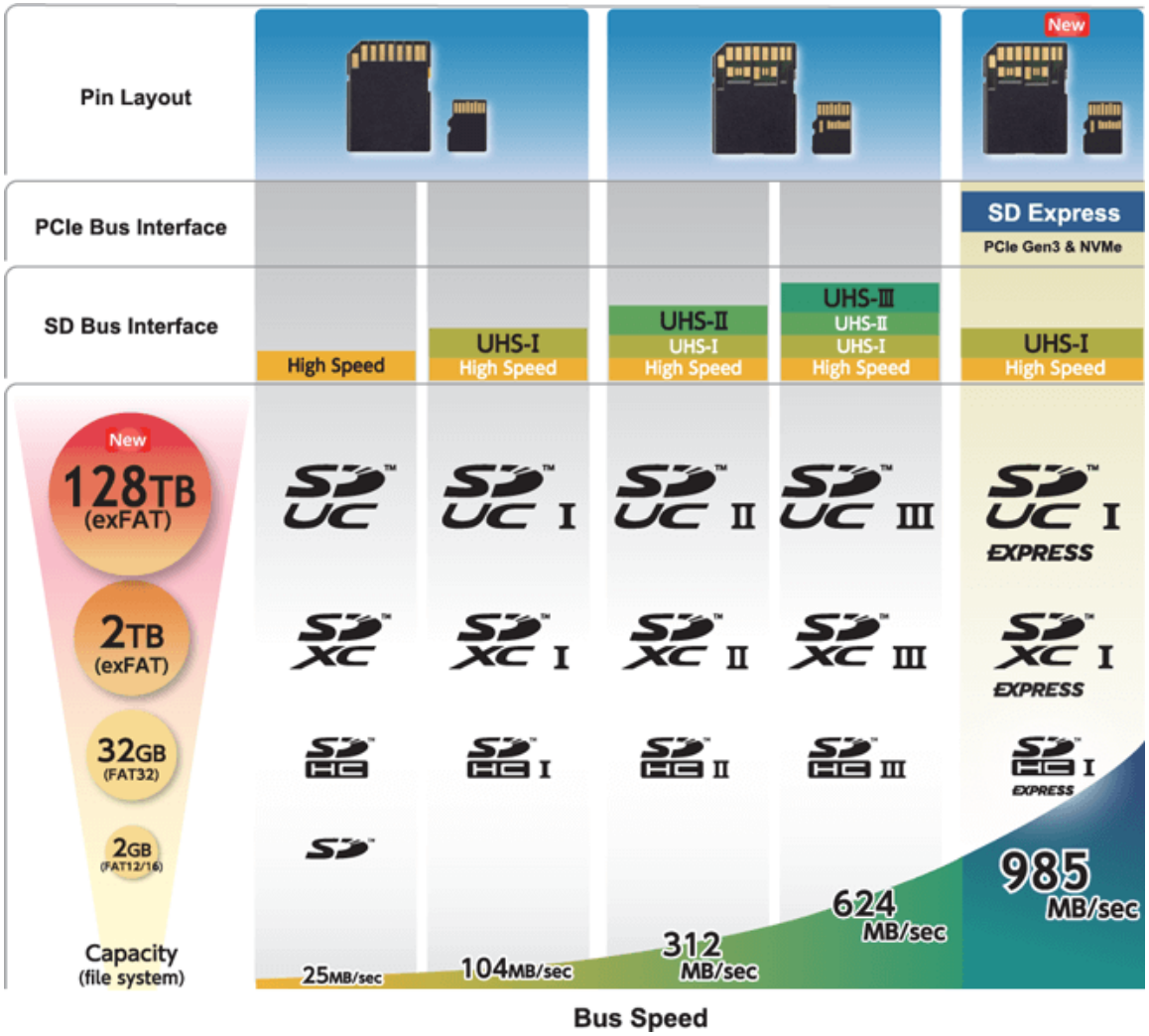
# 13.1 SD卡简介

## 13.1.1 SD卡的分类

## 13.1.2 常规SD卡的接口

# 13.1.1 SD卡的分类

SD卡(Secure Digital Memory Card)是一种Flash存储器



# 1. 外形尺寸

- 标准SD卡，大小是24mm×32mm×2.1mm
- microSD卡，通常也被称作TF（Trans-flash）卡，大小是11mm×15mm×1.0mm


标准SD卡

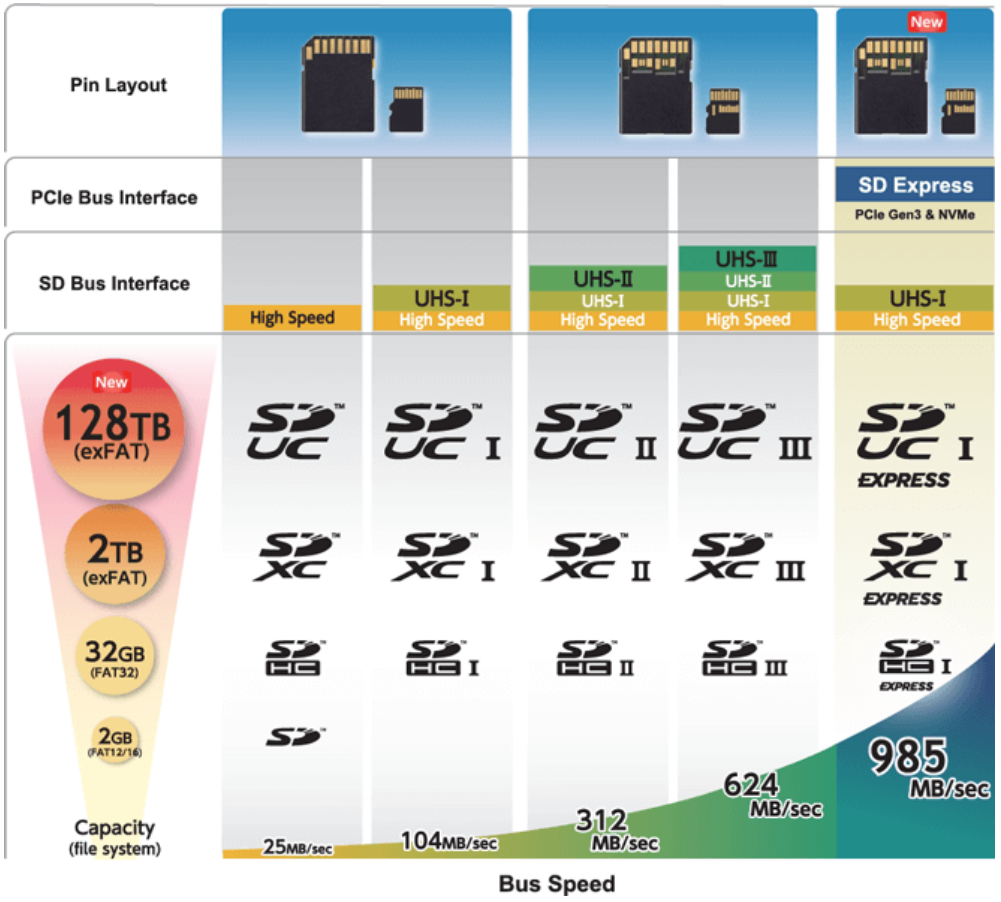

microSD卡（TF卡）


TF 32G


SD转换卡套

## 2. 存储容量

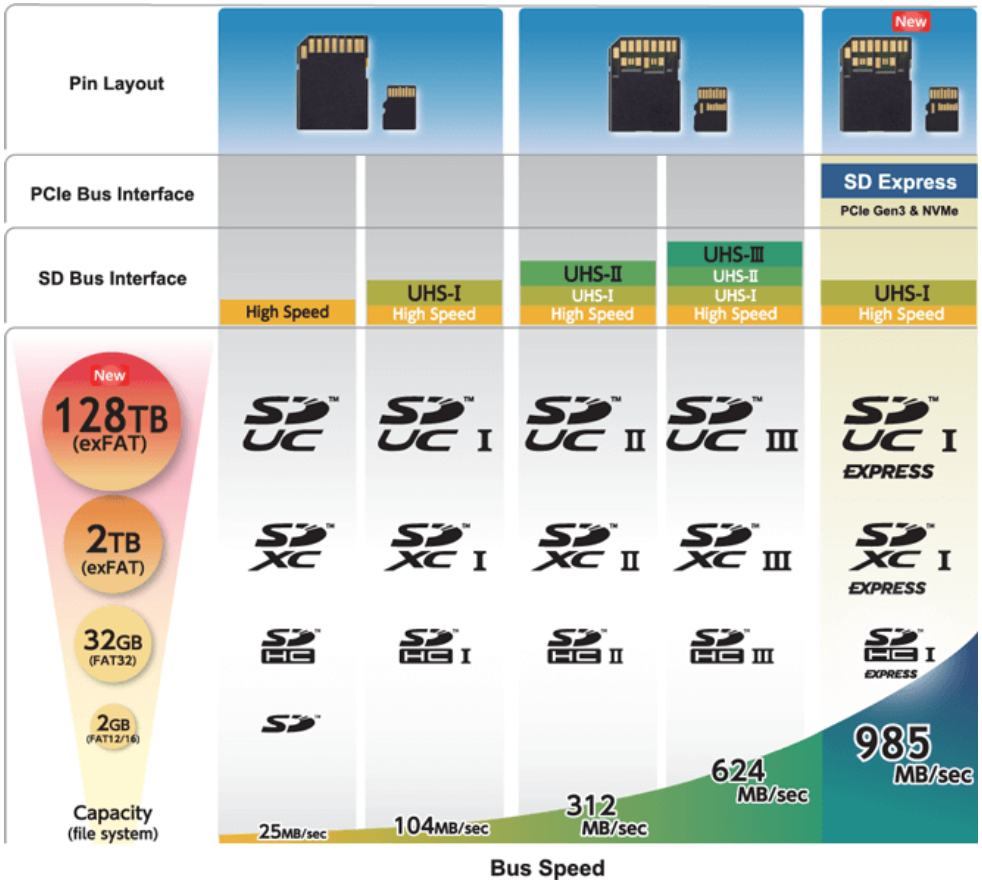
- SD，容量上限为2GB，使用FAT12和FAT16文件系统
- SDHC，容量介于2GB至32GB，使用FAT32文件系统
- SDXC，容量介于32GB至2TB，使用exFAT文件系统
- SDUC，容量介于2TB至128TB，使用exFAT文件系统




不管SD卡容量是多大，SD卡数据读写的最小单位是块（Block），一个块的大小是512字节。

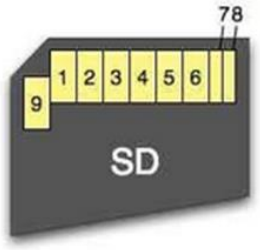
## 3. 总线速度

- SD 1.1规范定义了高速模式(25MB/s)
- SD 3.01规范定义了UHS-I模式，后续版本中又推出了UHS-II、UHS-III模式
- UHS-II和UHS-III使用了两排针脚，第二排针脚采用了低电压差分信号技术
- SD Express使用PCIe Gen.3接口和NVMe应用协议，最高可达985MB/s




STM32F4系列MCU的SDIO接口只支持到SD 2.0规范，也就是只支持到 25MB/s 的高速模式。

# 13.1.2 常规SD卡的接口




标准SD卡

SDIO接口，不提供SPI兼容模式


microSD卡（TF卡）

| 引脚编号 | 名称 | 功能 |
|---|---|---|
| 1 | CD/DATA3 | SD卡检测/数据线3 |
| 2 | CMD | 命令 |
| 3 | VSS1 | 电源地 |
| 4 | VDD | 电源 |
| 5 | CLK | 时钟信号 |
| 6 | VSS2 | 电源地 |
| 7 | DATA0 | 数据线0 |
| 8 | DATA1 | 数据线1 |
| 9 | DATA2 | 数据线2 |

| 引脚编号 | 名称 | 功能 |
|---|---|---|
| 1 | DATA2 | 数据线2 |
| 2 | CD/DATA3 | SD卡检测/数据线3 |
| 3 | CMD | 命令 |
| 4 | VDD | 电源 |
| 5 | CLK | 时钟信号 |
| 6 | VSS | 电源地 |
| 7 | DATA0 | 数据线0 |
| 8 | DATA1 | 数据线1 |

其中DATA3还可作为SD卡检测线CD（Card Detection），

也就是在SD卡插入时产生一个信号让主机知道SD卡插入了

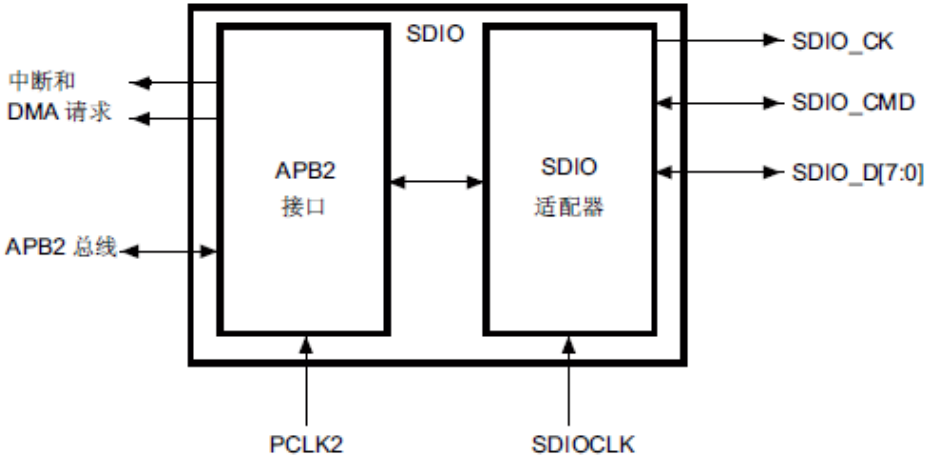
# 13.2 SDIO接口硬件电路

## 13.2.1 STM32F407的SDIO接口

## 13.2.2 开发板上的microSD卡连接电路

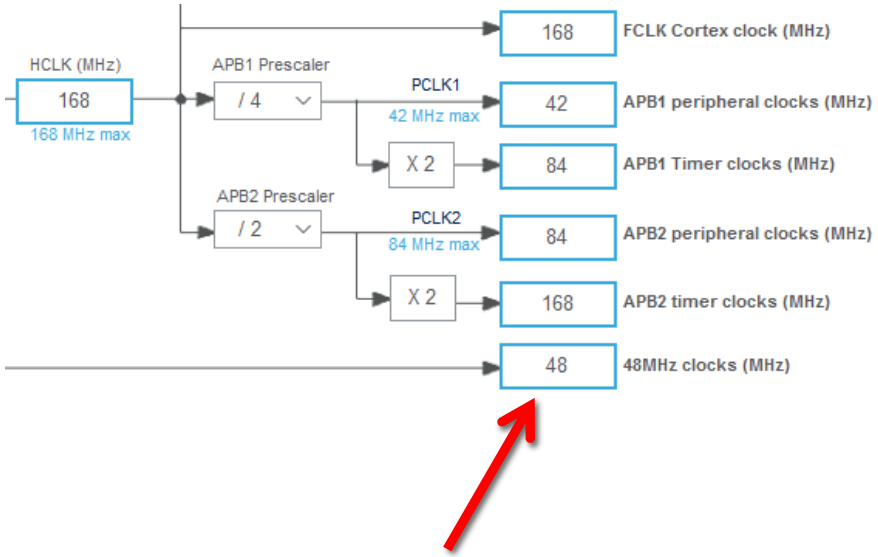
# 13.2.1 STM32F407的SDIO接口

- 完全兼容SD卡规范版本2.0
- 支持2种数据总线模式：1位（默认）或4位
- 只支持高速SD卡，速度最高25MB/s

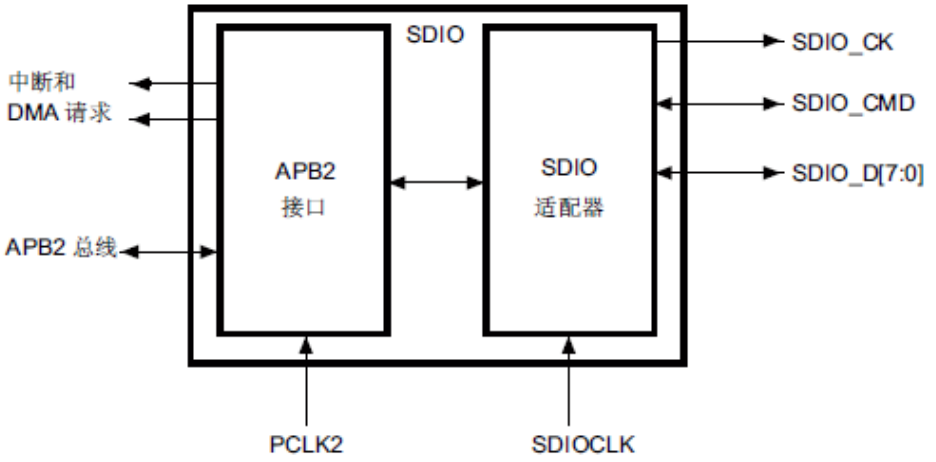



SDIO使两个时钟信号：

- SDIO适配器时钟SDIOCLK，来自于时钟树上的48MHz时钟源
- APB2总线时钟PCLK2



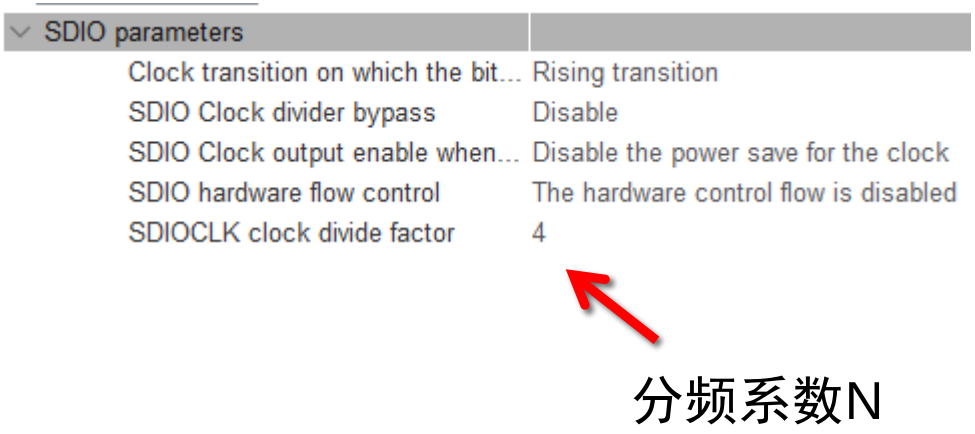
SDIO接口需要给SD卡提供一个时钟信号SDIO_CK，频率由一个分频器和SDIOCLK产生，计算公式如下：




$$f_{SDIO\_CK} = \frac{f_{SDIOCLK}}{2+N}$$

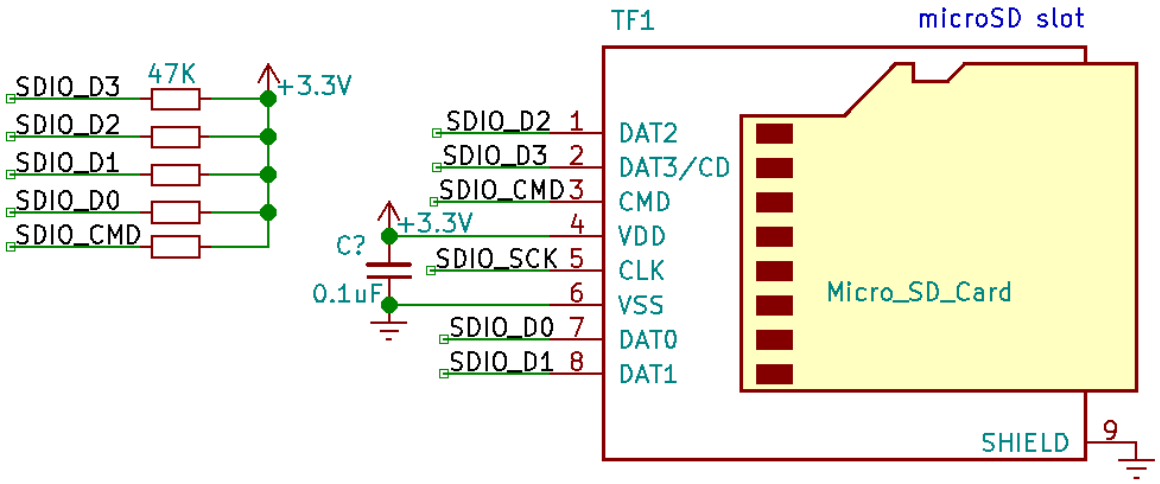
N是分频系数。一般情况下SDIOCLK为48MHz，当N=0时，SDIO_CK最高频率为24MHz

当N=4时，SDIO_CK为8MHz，通讯比较稳定




分频系数N

# 13.2.2 开发板上的microSD卡连接电路




SDIO接口使用4根数据线，4根数据线和SDIO_CMD都使用了外接上拉电阻，SDIO_SCK无外接上拉电阻。

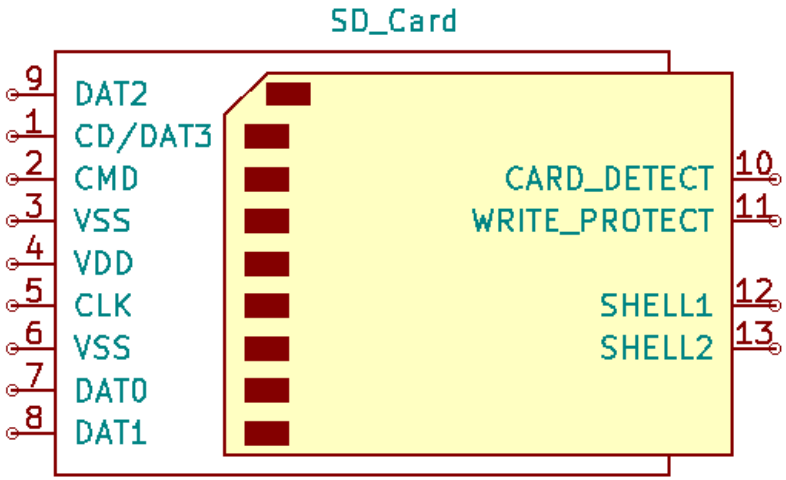
SDIO_D3引脚还可以作为SD卡检测引脚CD，但是要连接一个下拉电阻。

F407开发板上的microSD卡座没有硬件CD信号，所以不考虑microSD插入检测问题，总是假设microSD卡已经插入。

有的SD卡座有Card_Detect（CD）和Write_Protect（WP）信号引脚。

一般的，SD卡槽内对CD和WP引脚有下拉电阻，当SD卡插入时由于卡槽内簧片的机械作用，使CD输出变为高电平。如果将CD引脚接MCU的一个EXTI引脚，就可以检测SD卡的插入。

同样的，SD卡上有个写保护拨动开关，会使WP引脚输出不同的电平信号，表示SD卡是否被写保护。



# 13.3 SDIO的HAL驱动程序

## 13.3.1 SDIO驱动程序概述

## 13.3.2 初始化和配置函数

## 13.3.3 读取SD卡参数信息

## 13.3.4 获取SD卡状态

## 13.3.5 轮询方式读写SD卡

## 13.3.6 中断和DMA方式读写SD卡

# 13.3.1 SDIO驱动程序概述

SDIO驱动程序头文件stm32f4xx_hal_sd.h中有一个宏定义

```
#define  BLOCKSIZE   512U    // 块大小为 512 字节
```

SD卡读写数据的最小单位是块（Block），不管什么容量的SD卡，块的大小都是512字节

通过SDIO的HAL驱动程序，可以直接操作SD卡（SD卡无需用FatFS格式化），包括：

- SDIO和SD卡的初始化
- SD卡各种信息读取
- 轮询方式读写SD卡
- 中断方式读写SD卡
- DMA方式读写SD卡

# 13.3.2 初始化和配置函数

初始化和配置相关函数

| 函数名 | 函数功能 |
| --- | --- |
| **HAL_SD_Init()** | SDIO接口和SD卡初始化，内部会调用HAL_SD_InitCard() |
| **HAL_SD_InitCard()** | SD卡初始化，如果需要重新初始化SD卡，可以单独调用这个函数 |
| **HAL_SD_ConfigWideBusOperation()** | 设置SDIO接口数据线位数，即设置为1位、4位或8位数据线 |
| **HAL_SD_Erase()** | 擦除指定编号范围的数据块 |

SDIO接口的设置主要包括数据线条数、SDIOCLK时钟分频系数等。SDIO初始化的代码由CubeMX自动生成。

# 13.3.3 读取SD卡参数信息

读取SD卡参数信息相关函数

| 函数名 | 函数功能 |
| --- | --- |
| **HAL_SD_GetCardInfo()** | 读取SD卡的信息，包括SD卡类型、数据块个数、数据块大小等 |
| **HAL_SD_GetCardCID()** | 返回SD卡上CID寄存器里存储的信息，包括生产厂家ID、产品序列号等 |
| **HAL_SD_GetCardCSD()** | 返回SD卡上CSD寄存器里存储的信息，包括系统版本号、总线最高频率、读取数据块最大长度等 |
| **HAL_SD_GetCardStatus()** | 返回SD卡上SSR寄存器的内容，包括当前总线位宽、卡的类型等 |

这些SD卡用于读取SD卡上一些寄存器的信息，以获取SD卡的一些原始参数，如卡的类型、数据块个数等。详见教材。

# 13.3.4 获取SD卡状态

| 函数名 | 函数功能 |
| --- | --- |
| **HAL_SD_GetCardState()** | 获取SD卡当前数据状态，返回状态类型是HAL_SD_CardStateTypeDef |
| **HAL_SD_GetState()** | 获取SDIO接口的状态，返回状态类型是HAL_SD_StateTypeDef |
| **HAL_SD_GetError()** | 可以在回调函数HAL_SD_ErrorCallback()里调用这个函数获取错误编号 |

有些函数在执行后要求用函数HAL_SD_GetCardState()查询SD卡的状态，以确定操作是否完成。

函数HAL_SD_GetCardState()用于查询SD卡当前的状态，其函数原型定义如下：

```
HAL_SD_CardStateTypeDef HAL_SD_GetCardState(SD_HandleTypeDef *hsd)
```

函数的返回值类型是HAL_SD_CardStateTypeDef，表示SD卡的各种状态

```
#define  HAL_SD_CARD_READY                0x00000001U   //卡处于就绪状态
#define  HAL_SD_CARD_IDENTIFICATION       0x00000002U   //卡处于识别状态
#define  HAL_SD_CARD_STANDBY              0x00000003U   //卡处于休眠状态
#define  HAL_SD_CARD_TRANSFER             0x00000004U   //卡处于传输状态
#define  HAL_SD_CARD_SENDING              0x00000005U   //卡正在发送一个操作
#define  HAL_SD_CARD_RECEIVING            0x00000006U   //卡正在接收一个操作信息
#define  HAL_SD_CARD_PROGRAMMING          0x00000007U   //卡正在编程写入状态
#define  HAL_SD_CARD_DISCONNECTED         0x00000008U   //卡已断开连接
#define  HAL_SD_CARD_ERROR                0x000000FFU   //卡响应错误
```

# 13.3.5 轮询方式读写SD卡

| 函数名 | 函数功能 |
| --- | --- |
| **HAL_SD_ReadBlocks()** | 以轮询方式读取1个或多个数据块的数据 |
| **HAL_SD_WriteBlocks()** | 以轮询方式写入1个或多个数据块的数据 |

轮询方式就是阻塞式，两个函数需要在读/写操作完成后才返回。注意，SD卡读写数据的最小单位是块，一个块是512字节。

函数HAL_SD_ReadBlocks()的原型定义如下：

```
HAL_StatusTypeDef HAL_SD_ReadBlocks (SD_HandleTypeDef *hsd, uint8_t
    *pData, uint32_t BlockAdd, uint32_t NumberOfBlocks, uint32_t Timeout)
```

- hsd 是SD卡对象指针
- pData 是读出数据保存缓存区的指针
- BlockAdd 是读取数据的起始块编号
- NumberOfBlocks 是要读取的块的个数，可以大于1
- Timeout 是超时等待时间，单位是节拍数

如果在Timeout时间内成功读取了数据，函数返回值为HAL_OK

缓存区pData的大小应该是BLOCKSIZE*NumberOfBlocks个字节，BLOCKSIZE值为512，也就是一个块的字节数。

函数HAL_SD_WriteBlocks()的原型定义如下：

```
HAL_StatusTypeDef HAL_SD_WriteBlocks (SD_HandleTypeDef *hsd, uint8_t
    *pData, uint32_t BlockAdd, uint32_t NumberOfBlocks, uint32_t Timeout);
```

- hsd 是SD卡对象指针
- pData 是待写入数据缓存区的指针
- BlockAdd 是要写入位置的起始块编号
- NumberOfBlocks 是要写入的块的个数，可以大于1
- Timeout 是超时等待时间，单位是节拍数

如果在Timeout时间内成功写入了数据，函数返回值为HAL_OK

调用函数HAL_SD_WriteBlocks()写入数据块时，无需先执行块擦除操作，该函数内部会执行块擦除操作。

# 13.3.6 中断和DMA方式读写SD卡

| 分组 | 函数名 | 函数功能 |
| --- | --- | --- |
| **中断方式读写** | HAL_SD_ReadBlocks_IT() | 以中断方式读取1个或多个数据块的数据 |
| | HAL_SD_WriteBlocks_IT() | 以中断方式写入1个或多个数据块的数据 |
| | HAL_SD_IRQHandler() | SDIO中断ISR函数里调用的通用处理函数 |
| **DMA方式读写** | HAL_SD_ReadBlocks_DMA() | 以DMA方式读取1个或多个数据块的数据 |
| | HAL_SD_WriteBlocks_DMA() | 以DMA方式写入1个或多个数据块的数据 |
| **回调函数** | HAL_SD_RxCpltCallback() | 中断方式或DMA方式接收数据完成时的回调函数 |
| | HAL_SD_TxCpltCallback() | 中断方式或DMA方式发送数据完成时的回调函数 |
| | HAL_SD_AbortCallback() | 取消中断或DMA方式数据传输过程时的回调函数 |
| | HAL_SD_ErrorCallback() | 发生错误时的回调函数 |

使用FreeRTOS时，若使用FatFS管理SD卡，只能使用DMA传输方式（第17章的内容）

# 13.4 轮询方式读写SD卡示例

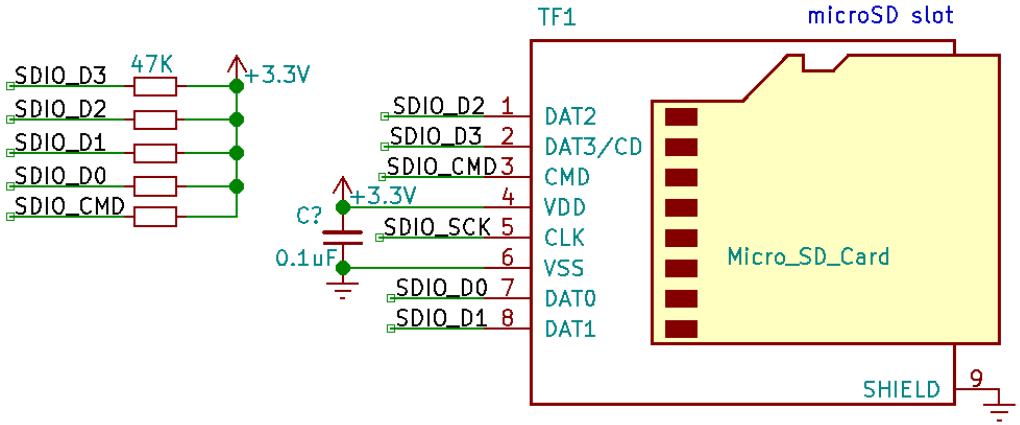
## 13.4.1 示例功能与CubeMX项目设置

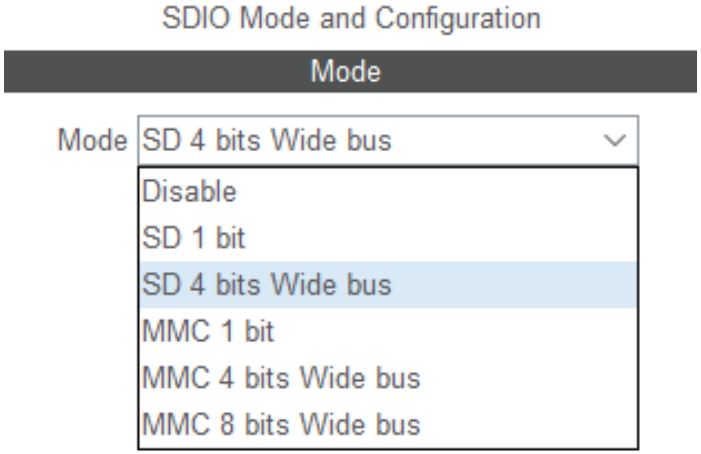
## 13.4.2 主程序与SDIO初始化

## 13.4.3 程序功能实现

# 13.4.1 示例功能与CubeMX项目设置

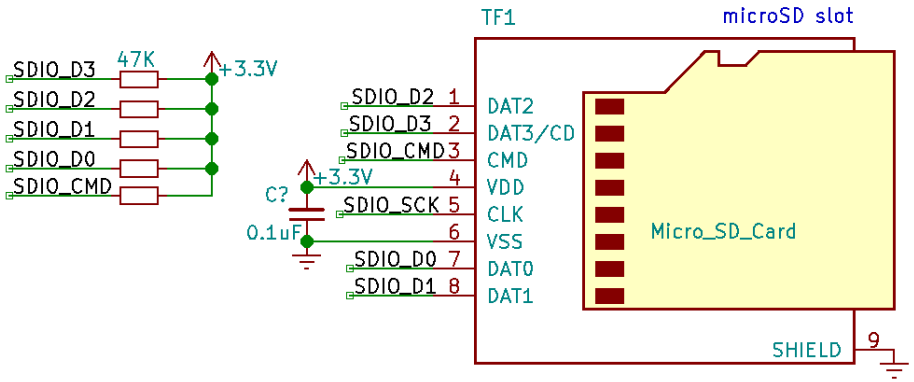
示例项目Demo13_1SDRaw，演示用轮询方式直接访问SD卡。开发板上microSD卡座的电路如图所示，使用了4根数据线。SDIO模式设置为SD 4 bits Wide bus



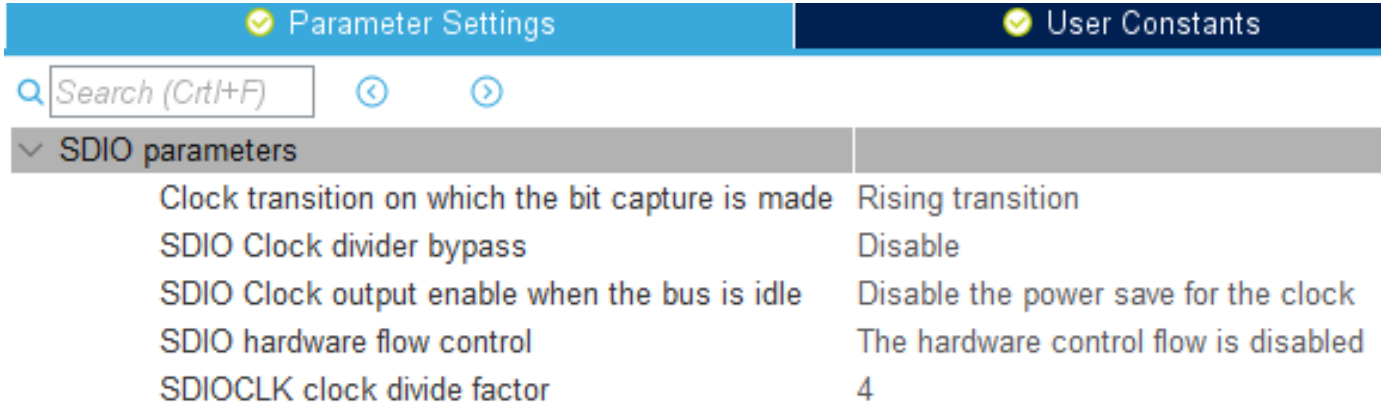



自动分配SDIO的GPIO引脚如图，自动分配的GPIO引脚与开发板上的实际电路是对应的。实际电路上除SDIO_CK外，其他几个引脚都有外部上拉电阻，所以在CubeMX中无需再为这些引脚设置内部上拉

| Pin Name | Signal on Pin | GPIO mode | GPIO Pull-up/Pull-down | Maximum output speed |
|---|---|---|---|---|
| PC8 | SDIO_D0 | Alternate Function Push Pull | No pull-up and no pull-down | Very High |
| PC9 | SDIO_D1 | Alternate Function Push Pull | No pull-up and no pull-down | Very High |
| PC10 | SDIO_D2 | Alternate Function Push Pull | No pull-up and no pull-down | Very High |
| PC11 | SDIO_D3 | Alternate Function Push Pull | No pull-up and no pull-down | Very High |
| PC12 | SDIO_CK | Alternate Function Push Pull | No pull-up and no pull-down | Very High |
| PD2 | SDIO_CMD | Alternate Function Push Pull | No pull-up and no pull-down | Very High |



# SDIO参数设置



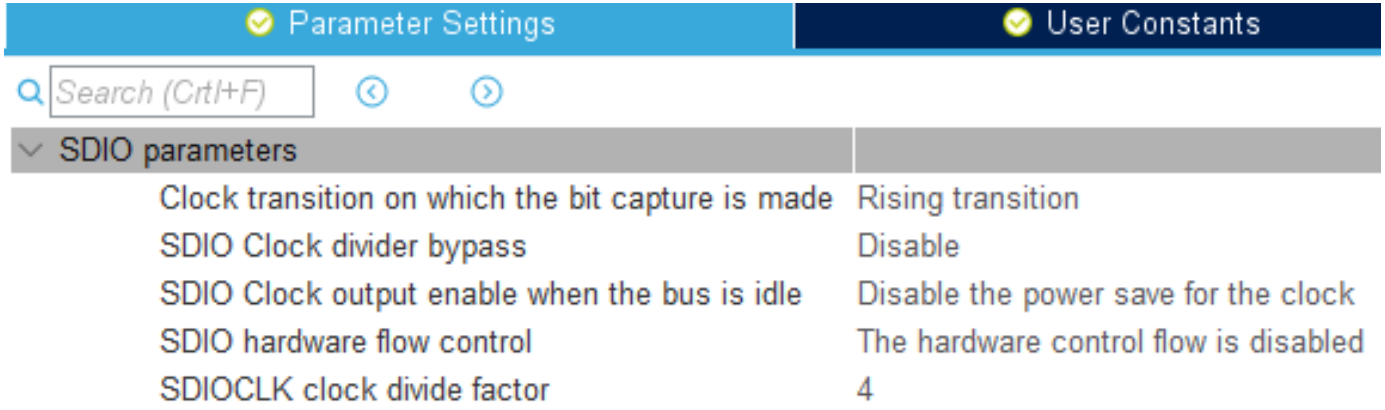

## （1）Clock transition on which the bit capture is made

设置在时钟的哪个跳变沿捕获位数据，默认是在上跳沿（Rising transition）捕获数据。

## （2）SDIO Clock divider bypass

设置SDIO时钟分频器旁路是否使能。如果设置为Disable，将根据公式（13-2）由SDIOCLK和分频系数生成SDIO_CK

# SDIO参数设置



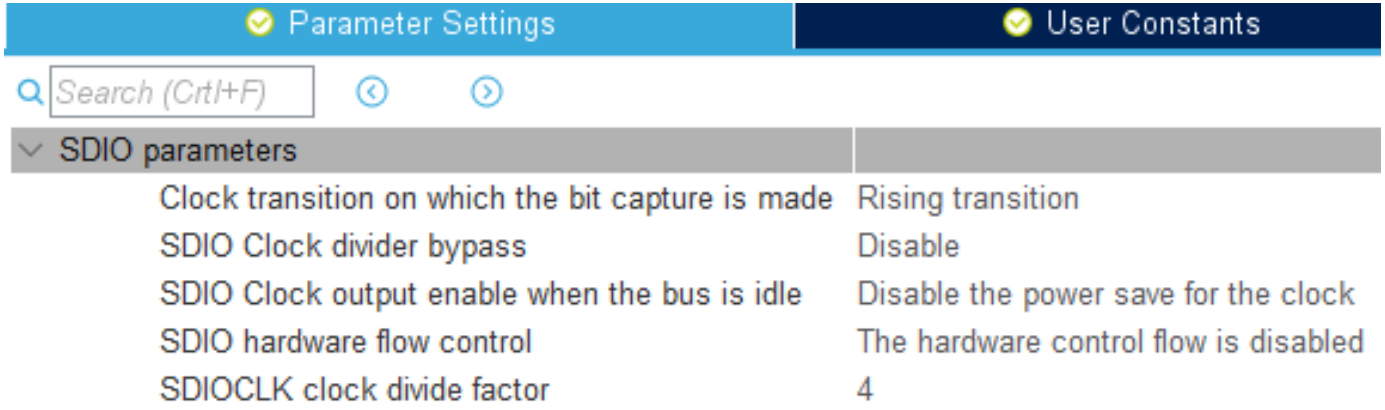

## （3）SDIO Clock output enable when the bus is idle

设置在总线空闲时是否还使能SDIO_CK时钟，也就是配置SDIO的节能模式。

## （4）SDIO hardware flow control

设置是否使用SDIO硬件流控制功能。硬件流控制功能用于避免FIFO下溢（发送模式）和上溢（接收模式）错误。

# SDIO参数设置




## （5）SDIOCLK clock divide factor

设置SDIOCLK的分频系数。SDIO_CK信号频由下式计算

$$f_{\mathrm{SDIO\_CK}} = \frac{f_{SDIOCLK}}{2 + N}$$

$f_{SDIOCLK}$一般就是48MHz。N=4时，$f_{\mathrm{SDIO\_CK}} = 8\ \mathrm{MHz}$

# 13.4.2 主程序与SDIO初始化

## SDIO初始化

CubeMX自动生成了SDIO的初始化函数MX_SDIO_SD_Init()

```
void MX_SDIO_SD_Init(void)
{
    hsd.Instance = SDIO;        //寄存器基址
    hsd.Init.ClockEdge = SDIO_CLOCK_EDGE_RISING;        //时钟上升沿
    hsd.Init.ClockBypass = SDIO_CLOCK_BYPASS_DISABLE;    //禁止旁路
    hsd.Init.ClockPowerSave = SDIO_CLOCK_POWER_SAVE_DISABLE;
    hsd.Init.BusWide = SDIO_BUS_WIDE_1B;//总线位宽，初始1位，后面再设置为4位
    hsd.Init.HardwareFlowControl = SDIO_HARDWARE_FLOW_CONTROL_DISABLE;
    hsd.Init.ClockDiv = 4;        //分频系数
    if (HAL_SD_Init(&hsd) != HAL_OK)          //SDIO初始化，会调用HAL_SD_MspInit()
        Error_Handler();

    /*  配置为4位总线宽度  */
    if (HAL_SD_ConfigWideBusOperation(&hsd, SDIO_BUS_WIDE_4B) != HAL_OK)
        Error_Handler();
}
```

详见完整源程序

# 13.4.3 程序功能实现

## 1. 主程序

主程序在LCD上显示了一个文字菜单，并作出响应

```
[1]KeyUp   =SD card info
[2]KeyDown =Erase 0-10 blocks
[3]KeyLeft =Write block
[4]KeyRight=Read block
```

详见完整源程序

- 按下KeyUP键时，调用函数SDCard_ShowInfo()显示SD卡信息
- 按下KeyLeft键时，调用函数SDCard_TestWrite()测试向Block 5写入数据
- 按下KeyRight键时，调用函数SDCard_TestRead()测试从Block 5读取数据
- 按下KeyDown键时，调用函数SDCard_EraseBlocks()擦除Block 0至10

## 2. 显示SD卡信息

自定义函数SDCard_ShowInfo()调用HAL_SD_GetCardInfo()获取SD卡信息，并在LCD上显示

在开发板上插入一个16GB的microSD卡，运行时在LCD上显示的SD卡信息如下：

函数代码详见完整源程序

```
Card Type= 1
Card Version= 1
Card Class= 1461
Relative Card Address= 7
Block Count= 30449664
Block Size(Bytes)= 512
LogiBlockCount= 30449664
LogiBlockSize(Bytes)= 512
SD Card Capacity(MB)= 14868
```

## 3. 擦除块

函数HAL_SD_Erase()用于擦除SD卡指定的数据块，自定义函数SDCard_EraseBlocks()演示了这个函数的使用

```
void  SDCard_EraseBlocks()                  //擦除数据块0-10
{
      uint32_t BlockAddrStart=0;                        // Block 0，地址使用块编号
      uint32_t BlockAddrEnd=10;                         // Block 10
      LCD_ShowStr(10, LCD_CurY,(uint8_t*)"*** Erasing blocks ***");
      if (HAL_SD_Erase(&hsd, BlockAddrStart, BlockAddrEnd)==HAL_OK)
            LCD_ShowStr(10,LCD_CurY+LCD_SP15,(uint8_t*)"Erasing blocks,OK");
      else
            LCD_ShowStr(10,LCD_CurY+LCD_SP15,(uint8_t*)"Erasing blocks,fail");

      HAL_SD_CardStateTypeDef cardState= HAL_SD_GetCardState(&hsd);
      LCD_ShowStr(10,LCD_CurY+LCD_SP15,(uint8_t*)"GetCardState()= ");
      LCD_ShowUint(LCD_CurX,LCD_CurY,cardState);
      while(cardState != HAL_SD_CARD_TRANSFER)    //等待返回传输状态
      {
            HAL_Delay(1);
            cardState=HAL_SD_GetCardState(&hsd);
      }
      LCD_ShowStr(10,LCD_CurY+LCD_SP15,(uint8_t*)"Blocks 0-10 is erased.");
}
```

# 4. 写入数据

函数HAL_SD_WriteBlocks()以轮询方式向SD卡写入数据。

自定义函数SDCard_TestWrite()测试向SD卡写入数据

```
void  SDCard_TestWrite()       //测试写入
{
      LCD_ShowStr(10, LCD_CurY,(uint8_t*)"*** Writing blocks ***");
      uint8_t pData[BLOCKSIZE]="Hello, welcome to UPC\0";   //BLOCKSIZE=512
      uint32_t BlockAddr=5;                    //块编号
      uint32_t BlockCount=1;                   //块的个数
      uint32_t TimeOut=1000;                   //超时等待时间,ms
      if (HAL_SD_WriteBlocks(&hsd,pData,BlockAddr,BlockCount,TimeOut)==HAL_OK)
      {
            LCD_ShowStr(10,LCD_CurY+LCD_SP15,(uint8_t*)"Write to block 5, OK");
            LCD_ShowStr(10,LCD_CurY+LCD_SP15,(uint8_t*)"The string is:");
            LCD_ShowStr(30,LCD_CurY+LCD_SP10,pData);
      }
      else
            LCD_ShowStr(10,LCD_CurY+LCD_SP15,(uint8_t*)"Write to block 5, fail ***");
}
```

## 5. 读取数据

函数HAL_SD_ReadBlocks()以轮询方式从SD卡读取数据。

自定义函数SDCard_TestRead()测试从SD卡读取数据

```
void  SDCard_TestRead()                    //测试读取
{
      LCD_ShowStr(10, LCD_CurY,(uint8_t*)"*** Reading blocks ***");
      uint8_t pData[BLOCKSIZE];            //BLOCKSIZE=512
      uint32_t BlockAddr=5;                //块编号
      uint32_t BlockCount=1;               //块的个数
      uint32_t TimeOut=1000;               //超时等待时间,ms
      if (HAL_SD_ReadBlocks(&hsd,pData,BlockAddr,BlockCount,TimeOut)== HAL_OK)
      {
            LCD_ShowStr(10,LCD_CurY+LCD_SP15,(uint8_t*)"Read block 5, OK");
            LCD_ShowStr(10,LCD_CurY+LCD_SP15,(uint8_t*)"The string is:");
            LCD_ShowStr(30,LCD_CurY+LCD_SP10,pData);
      }
      else
            LCD_ShowStr(10,LCD_CurY+LCD_SP15,(uint8_t*)"Read block 5, fail ***");
}
```

# 练习任务

1. 看教材，练习本章示例
2. 自学13.5节，练习DMA方式读写SD卡的示例