

3. 13 内存卡读写文件

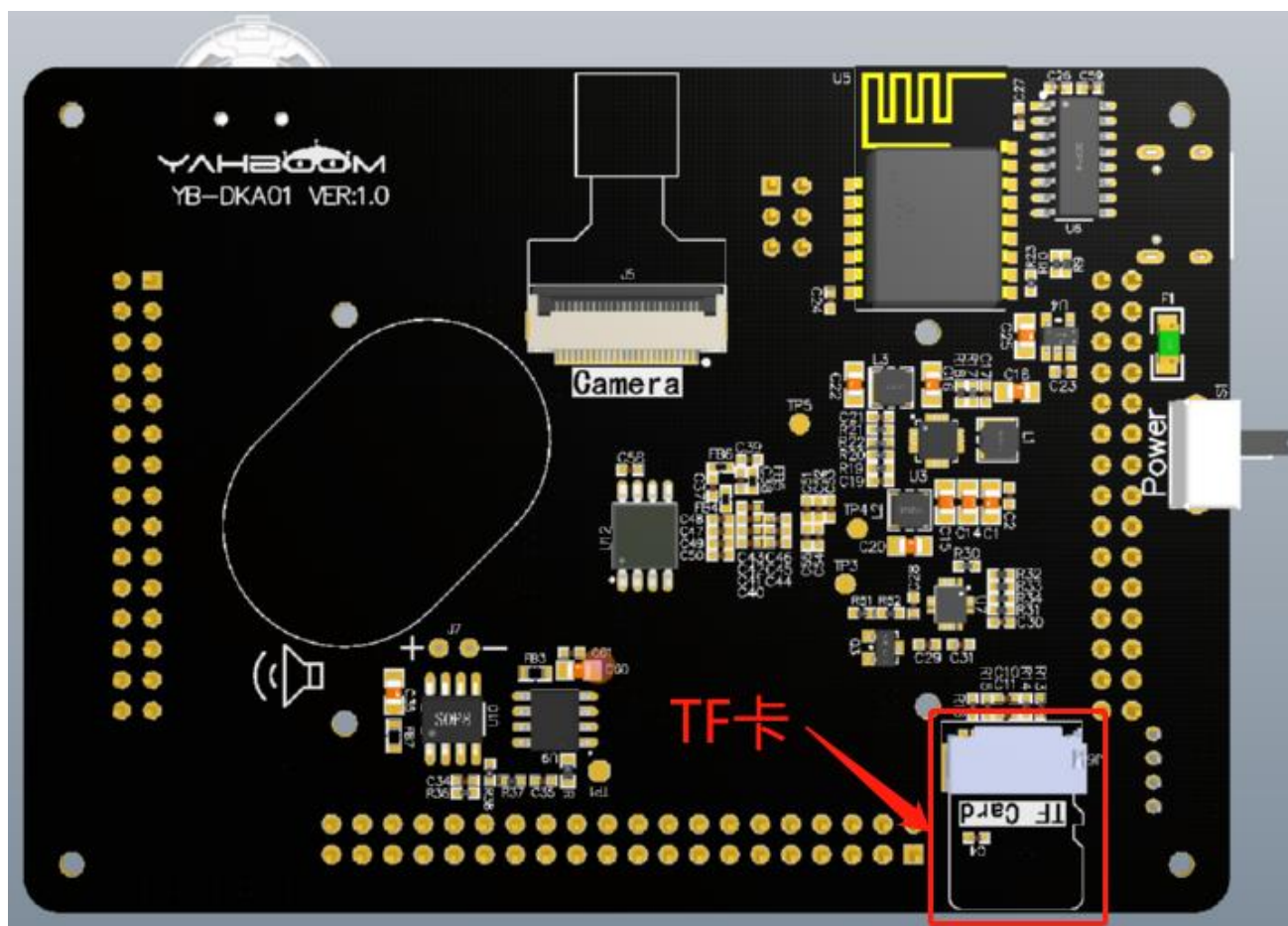
一、实验目的

本节课主要学习 K210 通过 SPI 读写内存卡文件的功能。

二、实验准备

1. 实验元件

TF 卡、LCD 显示屏



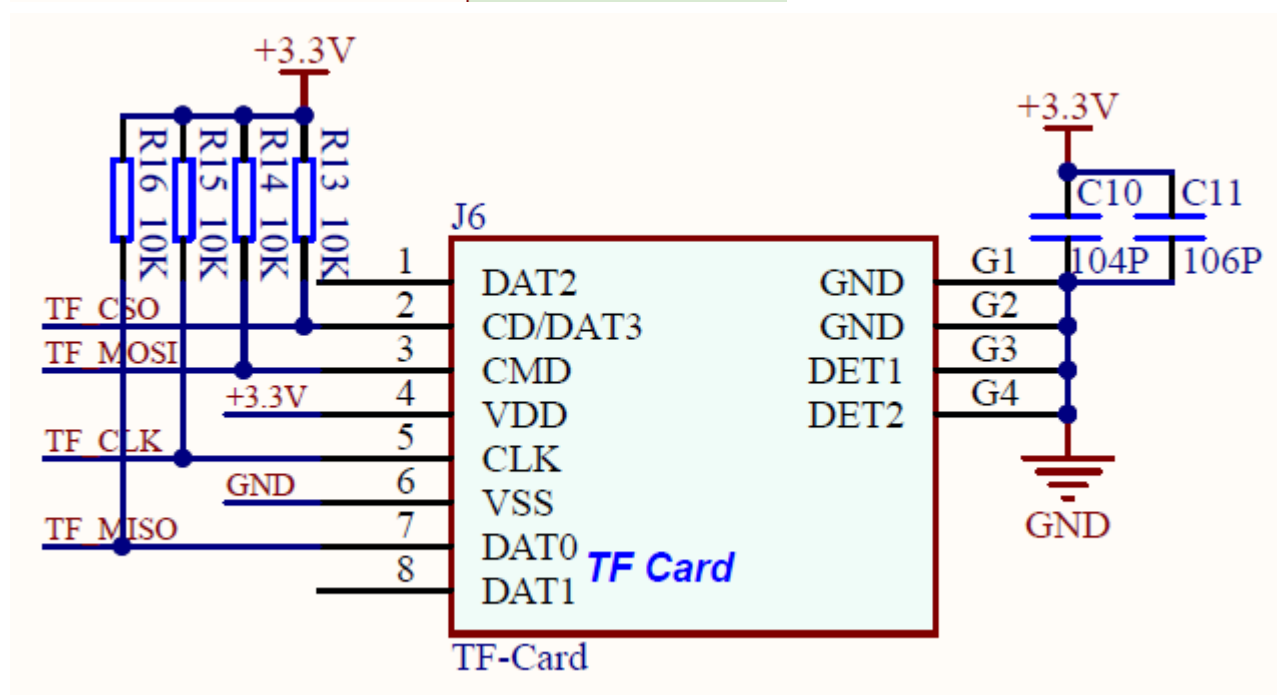
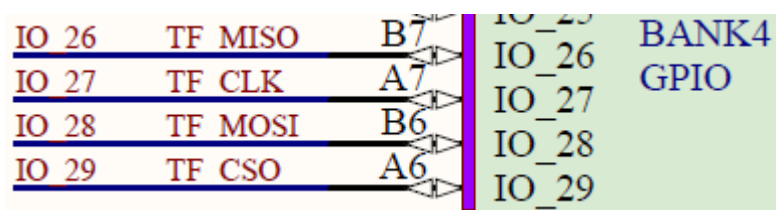
2. 元件特性

K210 开发板默认只有 TF 卡槽，不配 TF 卡，需要提前准备好 FAT32 格式的 TF 卡。TF 插入 TF 卡槽的时候注意方向，TF 卡的金手指那一面需要面向开发板，

如果方向插反了是无法读写数据的。TF 卡尺寸小，性能好，传输速度快，最初是用于支持内存扩展的手机上，比较适合储存高清摄像和高音质音频内容，因而被广泛应用于各种多媒体设备上。

3. 硬件连接

K210 开发板出厂默认已经焊接 TF 卡槽，需要插入 TF 卡才可以使用，其中 TF 卡槽的 TF_MISO 连接到 IO26, TF_CLK 连接到 IO27, TF_MOSI 连接到 IO28, TF_CSO 连接 IO29。



4. SDK 中对应 API 功能

对应的头文件 spi.h

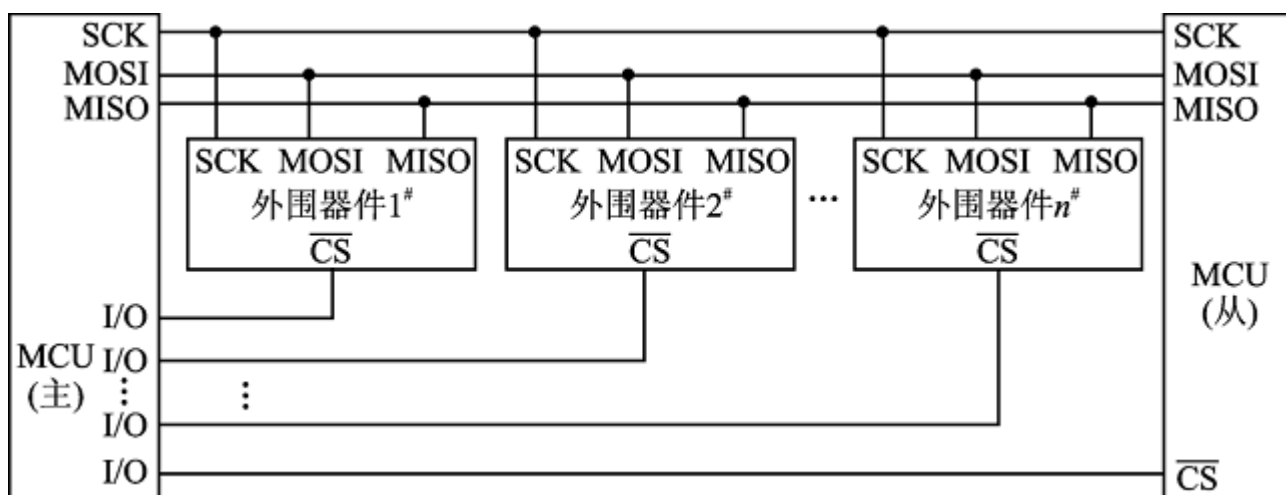
SPI 是一种高速、高效率的串行接口技术。通常由一个主模块和一个或多个从模块组成，主模块选择一个从模块进行同步通信，从而完成数据的交换。SPI 是一个环形结构，通信时需要至少 4 根线（事实上在单向传输时 3 根线也可以），它们是 MISO（主设备数据输入）、MOSI（主设备数据输出）、SCLK（时钟）、CS（片选）。

(1) MISO - Master Input Slave Output, 主设备数据输入，从设备数据输出；

(2) MOSI - Master Output Slave Input, 主设备数据输出，从设备数据输入；

(3) SCLK - Serial Clock, 时钟信号，由主设备产生；

(4) CS - Chip Select, 从设备使能信号，由主设备控制。当有多个从设备的时候，因为每个从设备上都有一个片选引脚接入到主设备机中，当我们的主设备和某个从设备通信时将需要将从设备对应的片选引脚电平拉低或者是拉高。



为用户提供以下接口：

- spi_init: 设置 SPI 工作模式、多线模式和位宽。

- spi_init_non_standard: 多线模式下设置指令长度、地址长度、等待时钟数、指令地址传输模式。
- spi_send_data_standard: SPI 标准模式传输数据。
- spi_send_data_standard_dma: SPI 标准模式下使用 DMA 传输数据。
- spi_receive_data_standard: 标准模式下接收数据。
- spi_receive_data_standard_dma: 标准模式下通过 DMA 接收数据。
- spi_send_data_multiple: 多线模式发送数据。
- spi_send_data_multiple_dma: 多线模式使用 DMA 发送数据。
- spi_receive_data_multiple: 多线模式接收数据。
- spi_receive_data_multiple_dma: 多线模式通过 DMA 接收。
- spi_fill_data_dma: 通过 DMA 始终发送同一个数据，可以用于刷新数据。
- spi_send_data_normal_dma: 通过 DMA 发送数据。不用设置指令地址。
- spi_set_clk_rate: 设置 SPI 的时钟频率。
- spi_handle_data_dma: SPI 通过 DMA 传输数据。

三、实验原理

TF 有 4 个数据传输端，DAT0，DAT1，DAT2，DAT3。还有一个 CMD 脚，是用来读取卡内信息的。

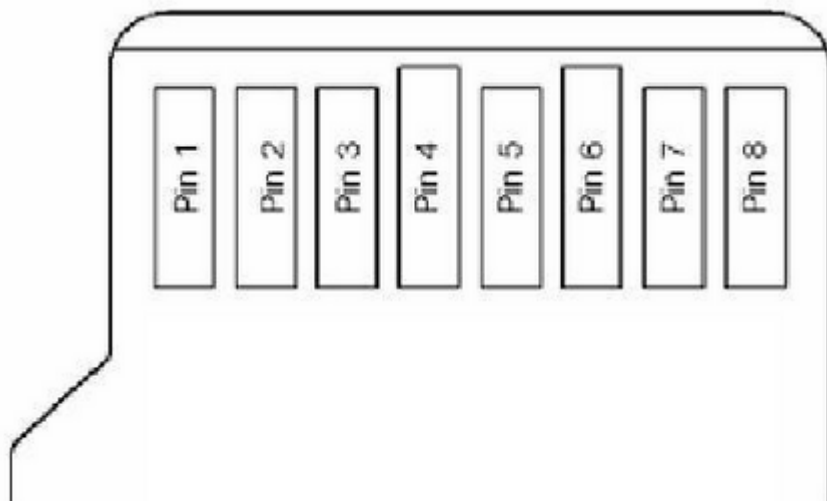
TF 卡主要管脚的功能：

CLK: 时钟信号，每个时钟周期传输一个命令或数据位，频率可在 0~25MHz 之间变化，TF 卡的总线管理器可以不受任何限制的自由产生 0~25MHz 的频率；

CMD: 双向命令和回复线，命令是主机到从卡操作的开始，命令可以是主机到单卡寻址，也可以是到所有卡；回复是对之前命令的回答，回复可以来自单卡或所有卡；

DAT0~3: 数据线，数据可以从 TF 卡传向主机也可以从主机传向 TF 卡。

TF 卡传输数据一般有两种模式，SD 模式和 SPI 模式，这里我们以 SPI 模式的方式传输数据。SPI 模式引脚如下：1: CS，2: DI，3: VSS，4: VDD，5: SCLK，6: VSS2，7: D0，8: RSV，9: RSV。



PIN	SPI Mode		
	名称	类型	备注
1	RSV		保留
2	CS/CD	输入	片选
3	DI/CMD	输入	数据输入
4	VDD	电源	电源正极
5	SCLK	输入	时钟
6	VSS	电源地	电源地
7	DO/DAT0	输出	数据输出
8	RSV		保留

四、实验过程

1. 首先初始化 K210 的硬件引脚和软件功能使用的是 FPIOA 映射关系。

```
void hardware_init(void)
{
    /*
    ** io26--miso--d1
    ** io27--clk---sclk
    ** io28--mosi--d0
    ** io29--cs----cs
    */
    fpioa_set_function(PIN_TF_MISO, FUNC_TF_SPI_MISO);
    fpioa_set_function(PIN_TF_CLK,  FUNC_TF_SPI_CLK);
    fpioa_set_function(PIN_TF_MOSI, FUNC_TF_SPI_MOSI);
    fpioa_set_function(PIN_TF_CS,   FUNC_TF_SPI_CS);
}
```

```

/*****HARDWARE-PIN*****/
// 硬件IO口，与原理图对应
#define PIN_TF_MISO      (26)
#define PIN_TF_CLK       (27)
#define PIN_TF_MOSI      (28)
#define PIN_TF_CS        (29)

/*****SOFTWARE-GPIO*****/
// 软件GPIO口，与程序对应
#define TF_CS_GPIONUM    (2)

/*****FUNC-GPIO*****/
// GPIO口的功能，绑定到硬件IO口
#define FUNC_TF_SPI_MISO  (FUNC_SPI1_D1)
#define FUNC_TF_SPI_CLK   (FUNC_SPI1_SCLK)
#define FUNC_TF_SPI_MOSI  (FUNC_SPI1_D0)
#define FUNC_TF_SPI_CS    (FUNC_GPIOHS0 + TF_CS_GPIONUM)

```

2. 设置系统时钟频率，由于 uarths 的时钟来自 PLL0，所以设置 PLL0 之后需要重新初始化以下 uarths，否则 printf 可能会打印乱码。

```

/* 设置系统时钟频率 */
sysctl_pll_set_freq(SYSCTL_PLL0, 800000000UL);
sysctl_pll_set_freq(SYSCTL_PLL1, 300000000UL);
sysctl_pll_set_freq(SYSCTL_PLL2, 45158400UL);
uarths_init();

```

3. 检测是否有 TF 卡，或者 TF 卡是否正常，如果不正常则退出。

```

if (check_sdcard())
{
    printf("SD card err\n");
    return -1;
}

```

如果初始化 TF 卡成功，则把 TF 的容量打印出来，单位为 G。

```
static int check_sdcard(void)
{
    uint8_t status;

    printf("/*****check_sdcard*****/\n");
    status = sd_init();
    printf("sd init :%d\n", status);
    if (status != 0)
    {
        return status;
    }

    printf("CardCapacity:%.1fG \n", (double)cardinfo.CardCapacity / 1024 / 1024 / 1024);
    printf("CardBlockSize:%d\n", cardinfo.CardBlockSize);
    return 0;
}
```

4. 检测 TF 卡的格式是否是 FAT32，如果不是则退出。

```
if (check_fat32())
{
    printf("FAT32 err\n");
    return -1;
}
```

如果检测符合 FAT32 格式的 TF 卡，则把 TF 卡挂在到“0:”，并且把 TF 卡根目录下的文件和文件夹的名称都打印出来。

```
static int check_fat32(void)
{
    static FATFS sdcard_fs;
    FRESULT status;
    DIR dj;
    FILINFO fno;

    printf("/*****check_fat32*****/\n");
    status = f_mount(&sdcard_fs, _T("0:"), 1);
    printf("mount sdcard:%d\n", status);
    if (status != FR_OK)
        return status;

    printf("printf filename\n");
    status = f_findfirst(&dj, &fno, _T("0:"), _T("*"));
    while (status == FR_OK && fno.fname[0])
    {
        if (fno.fattrib & AM_DIR)
            printf("dir:%s\n", fno.fname);
        else
            printf("file:%s\n", fno.fname);
        status = f_findnext(&dj, &fno);
    }
    f_closedir(&dj);
    return 0;
}
```

5. 向 TF 卡写入文件，保存在 TF 卡跟目录下的 test.txt，如果写入失败则退出。

```
sleep(1);
if (sd_write_file(_T("0:test.txt")))
{
    printf("SD write err\n");
    return -1;
}
```

在写入文件之前需要先打开文件，如果文件不存在则需要创建新文件，写入的数据都需要转化成 uint8_t 类型，写完数据后必须执行 f_close 关闭文件。

```
FRESULT sd_write_file(TCHAR *path)
{
    FIL file;
    FRESULT ret = FR_OK;
    printf("/*****sd_write_file*****/\n");
    uint32_t v_ret_len = 0;

    /* 打开文件，如果文件不存在，则新建 */
    if ((ret = f_open(&file, path, FA_CREATE_ALWAYS | FA_WRITE)) != FR_OK)
    {
        printf("open file %s err[%d]\n", path, ret);
        return ret;
    }
    else
    {
        printf("Open %s ok\n", path);
    }

    /* 要写入的数据 */
    uint8_t data[] = {'H','i',' ',' ','D','a','t','a',' ',' ','W','r','i','t','e',' ',' ','O','k','!'};

    /* 写入数据 */
    ret = f_write(&file, data, sizeof(data), &v_ret_len);
    if (ret != FR_OK)
    {
        printf("Write %s err[%d]\n", path, ret);
    }
    else
    {
        printf("Write %d bytes to %s ok\n", v_ret_len, path);
    }

    /* 关闭文件 */
    f_close(&file);
    return ret;
}
```


6. 从 TF 卡读取文件，文件为在 TF 卡根目录下的 test.txt。

```
if (sd_read_file(_T("0:test.txt")))
{
    printf("SD read err\n");
    return -1;
}
```

读文件前需要判断文件的状态，并且打开文件，如果文件不存在或者出错，则返回，如果正常则以只读方式打开文件，并且把读到的数据通过串口发送出来。

```
FRESULT sd_read_file(TCHAR *path)
{
    FIL file;
    FRESULT ret = FR_OK;
    printf("/******sd_read_file*****/\n");
    uint32_t v_ret_len = 0;

    /* 检测文件状态 */
    FILINFO v_fileinfo;
    if ((ret = f_stat(path, &v_fileinfo)) == FR_OK)
    {
        printf("%s length is %lld\n", path, v_fileinfo.fsize);
    }
    else
    {
        printf("%s fstat err [%d]\n", path, ret);
        return ret;
    }

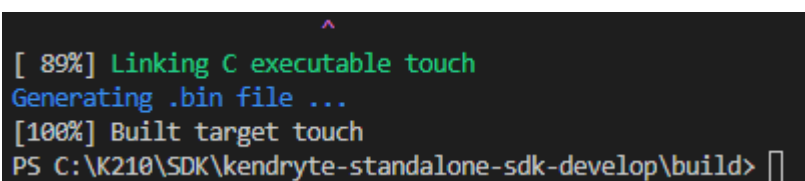
    /* 只读方式打开文件 */
    if ((ret = f_open(&file, path, FA_READ)) == FR_OK)
    {
        char v_buf[64] = {0};
        ret = f_read(&file, (void *)v_buf, 64, &v_ret_len);
        if (ret != FR_OK)
        {
            printf("Read %s err[%d]\n", path, ret);
        }
        else
        {
            printf("Read :> %s \n", v_buf);
            printf("total %d bytes lenth\n", v_ret_len);
        }
        /* 关闭文件 */
        f_close(&file);
    }
    return ret;
}
```

7. 编译调试，烧录运行

把本课程资料中的 sdcard 复制到 SDK 中的 src 目录下，然后进入 build 目录，运行以下命令编译。

```
cmake .. -DPROJ=sdcard -G "MinGW Makefiles"
```

```
make
```



```
[ 89%] Linking C executable touch
Generating .bin file ...
[100%] Built target touch
PS C:\K210\SDK\kendryte-standalone-sdk-develop\build>
```

编译完成后，在 build 文件夹下会生成 sdcard.bin 文件。

使用 type-C 数据线连接电脑与 K210 开发板，打开 kflash，选择对应的设备，再将程序固件烧录到 K210 开发板上。

五、实验现象

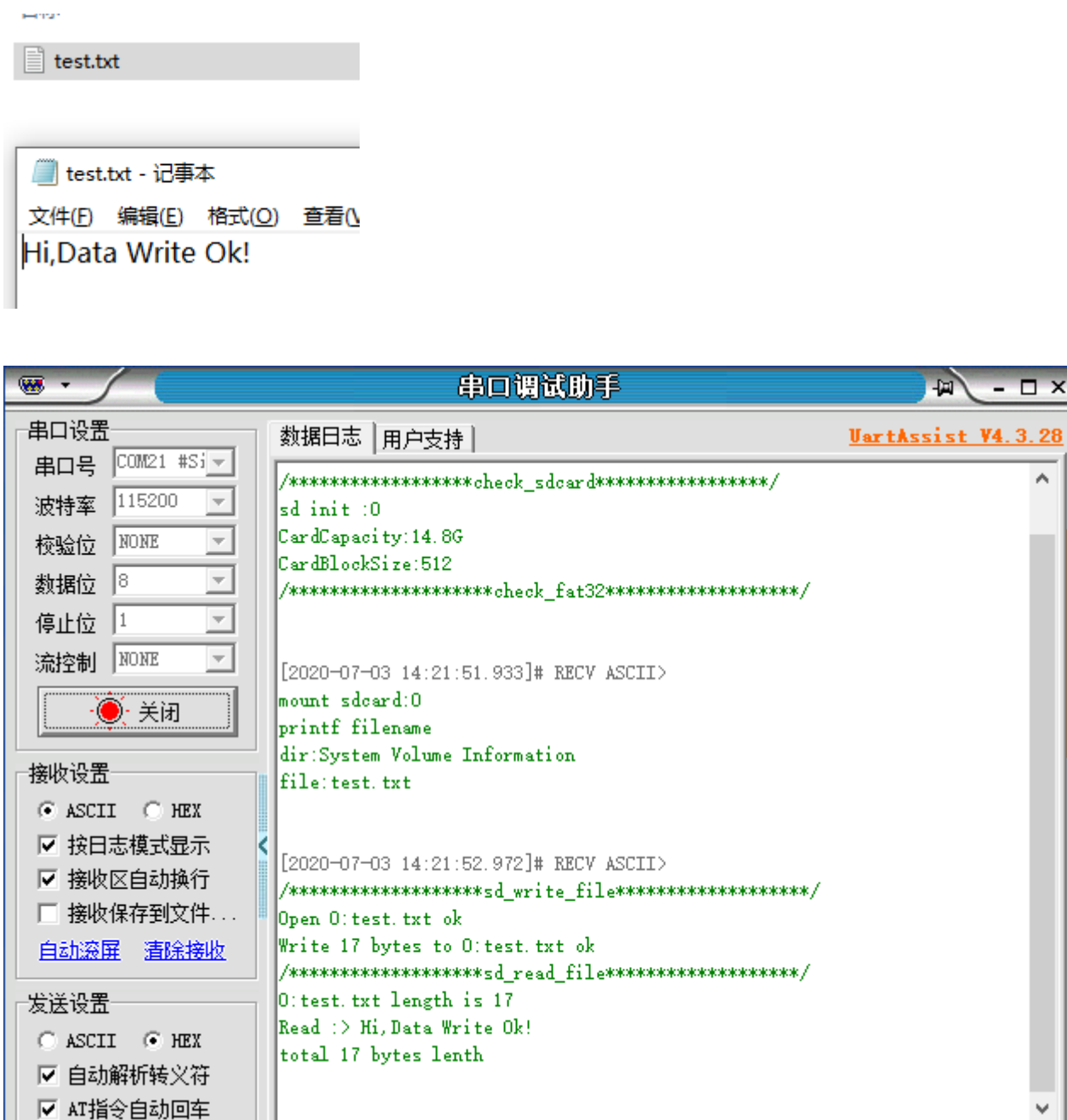
烧录完成固件后，系统会弹出一个终端界面，如果没有弹出终端界面的可以打开串口助手显示调试内容。

打开电脑的串口助手，选择对应的 K210 开发板对应的串口号，波特率设置为 115200，然后点击打开串口助手。注意还需要设置一下串口助手的 DTR 和 RTS。在串口助手底部此时的 4. DTR 和 7. RTS 默认是红色的，点击 4. DTR 和 7. RTS，都设置为绿色，然后按一下 K210 开发板的复位键。



可以看到串口助手上显示 TF 卡的容量和根目录下的文件名称，然后在 test.txt 文件写入 ‘Hi,Data Write Ok!’，并通过读取 test.txt 文件发送到串口。

用读卡器从上脑上读取 test.txt 文件，里面也内容和我们 K210 开发板读出来的一致。



六、实验总结

1. TF 读或写文件前都必须先打开文件，读写操作结束后也必须关闭文件。
2. TF 卡通过 SPI 通讯的方式，读写数据以 `uint8_t` 为基本单位。
3. 每次烧录完固件后，都需要重新给 K210 开发板上电，否则会出现 TF 卡初始化失败而退出系统的问题。