## 3.13 TF card read write files

**! Note: Yahboom K210 kit didn't include TF card. Please prepare TF card(with FAT32 format) by your self.**

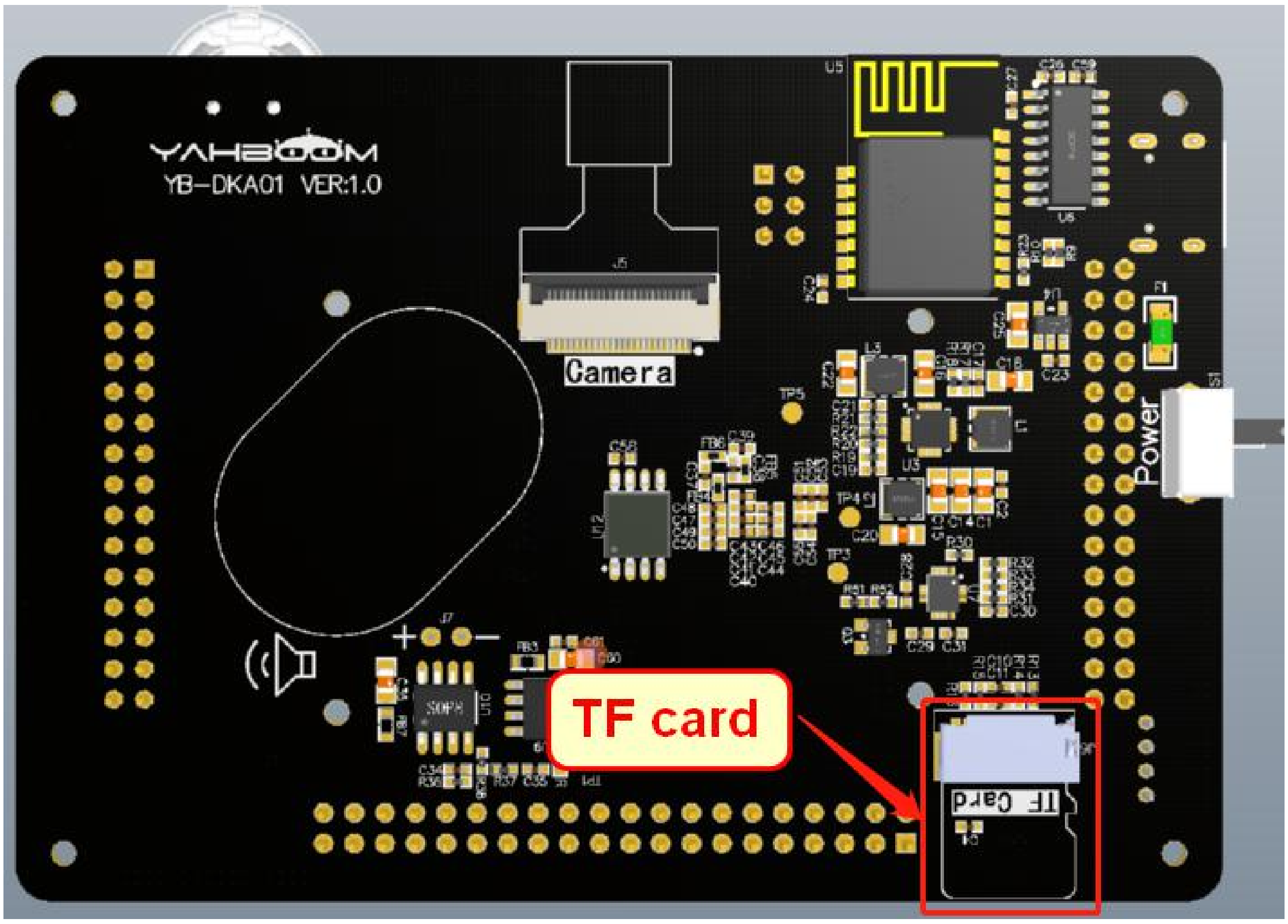
### 1. Experiment purpose

In this lesson, we will focus on how to make K210 read and write memory card files via SPI.

### 2.Experiment preparation

2.1 components

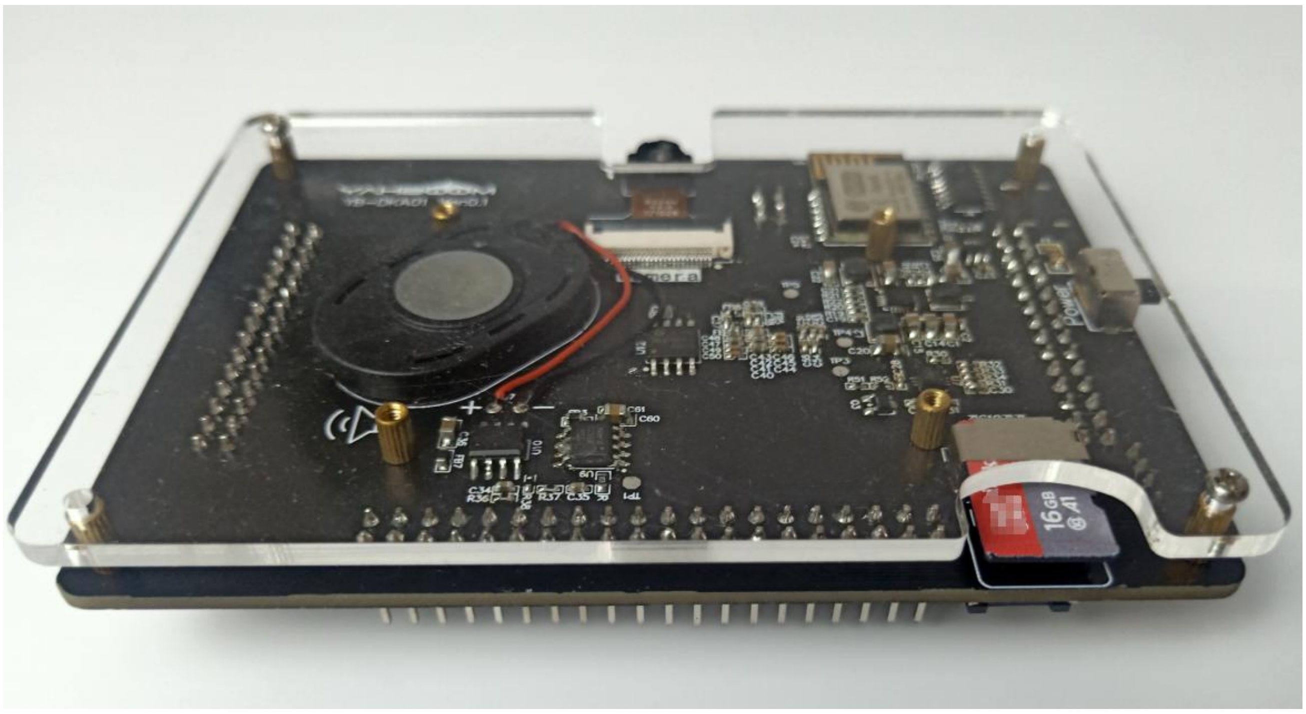
TF card, LCD screen



2.2 Component characteristics

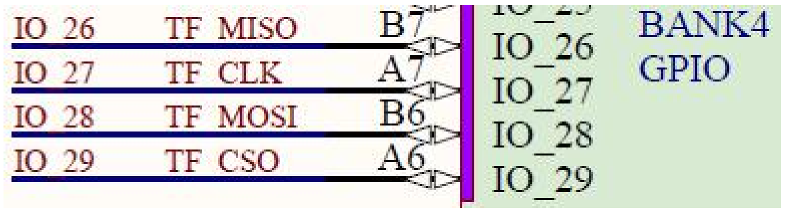
**Yahboom K210 kit didn't include TF card. Please prepare TF card(with FAT32 format) by your self.**

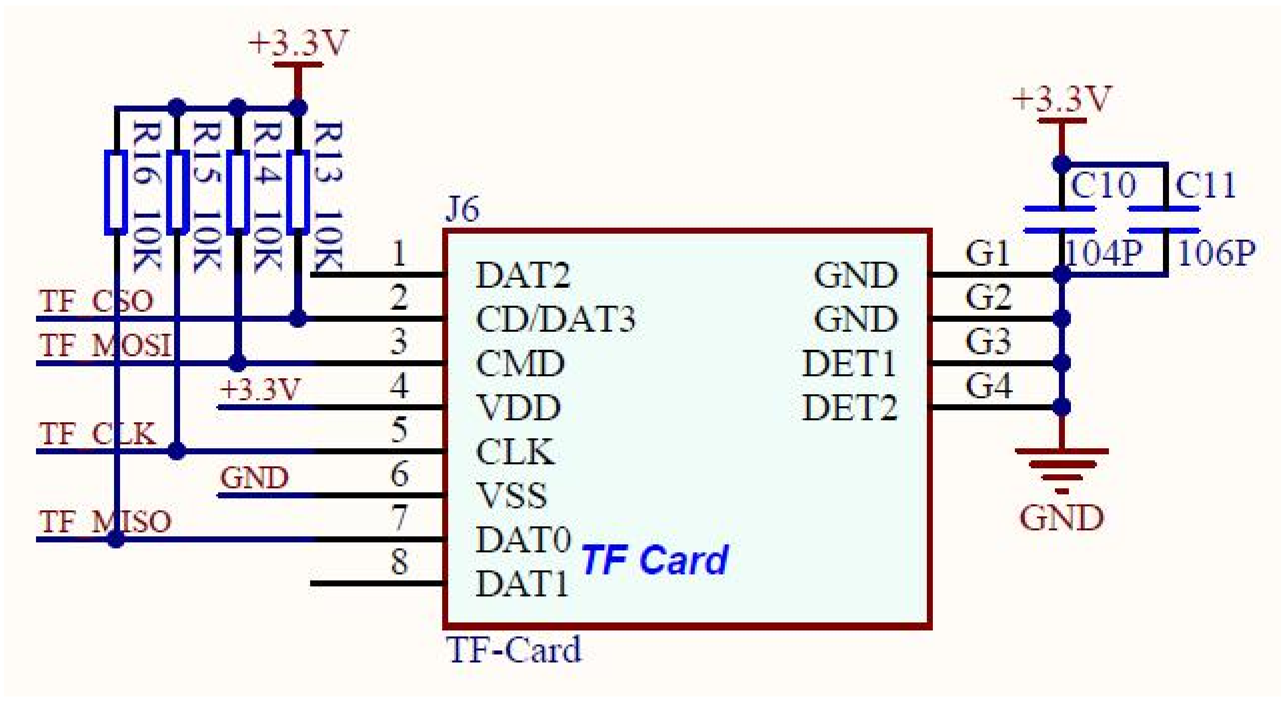
You must ensure TF card insert correctly. As show below.



## 2.3 Hardware connection

TF_MISO of the TF card slot is connected to IO26, TF_CLK is connected to IO27, TF_MOSI is connected to IO28, and TF_CSO is connected to IO29.





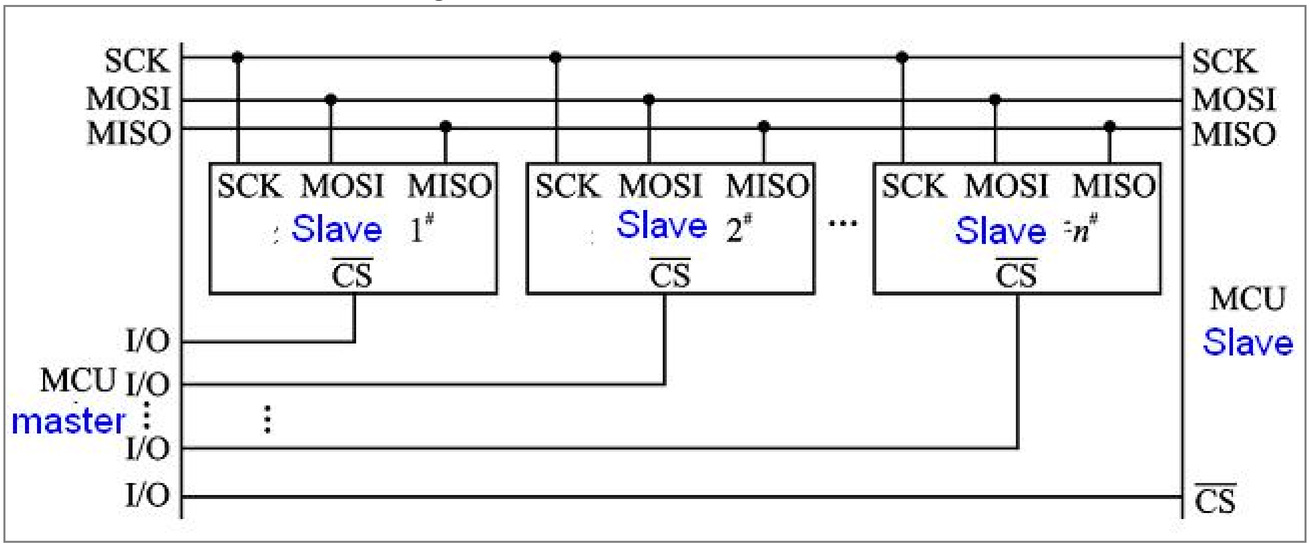
## 2.4 SDK API function

The header file is spi.h

SPI is a high-speed, high-efficiency serial interface technology. It usually consists of a master module and one or more slave modules. The master module selects a slave module for synchronous communication.

SPI need 4 wires are required for communication, MISO (master device data input), MOSI (master device data output), SCLK (clock), CS (chip select).
(1) MISO-Master Input Slave Output, master device data input, slave device data output;
(2) MOSI-Master Output Slave Input, master device data output, slave device data input;
(3) SCLK-Serial Clock, clock signal, generated by the master device;
(4) CS-Chip Select, slave device enable signal, controlled by master device.

When there are multiple slave devices, each slave device has a chip select pin connected to the master device.
When master device need to communicates with a slave device. We need to pull the chip select pin level of the slave device low or high.

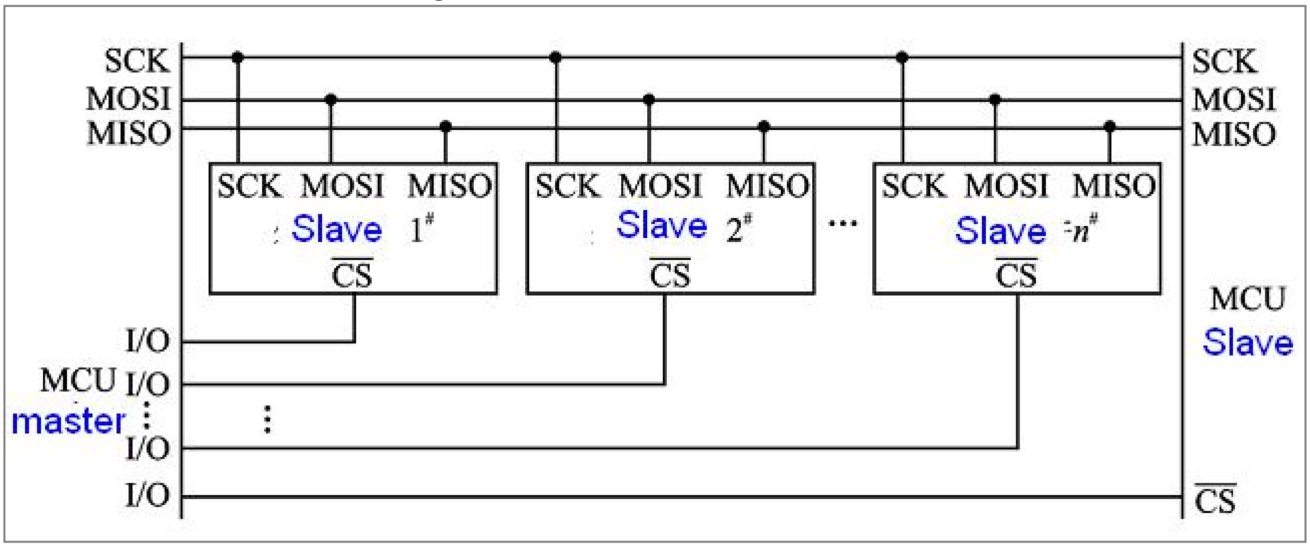


SPI is a high-speed, high-efficiency serial interface technology. It usually consists of a master module and one or more slave modules. The master module selects a slave module for synchronous communication.

SPI need 4 wires are required for communication, MISO (master device data input), MOSI (master device data output), SCLK (clock), CS (chip select).
(1) MISO-Master Input Slave Output, master device data input, slave device data output;
(2) MOSI-Master Output Slave Input, master device data output, slave device data input;
(3) SCLK-Serial Clock, clock signal, generated by the master device;
(4) CS-Chip Select, slave device enable signal, controlled by master device.

When there are multiple slave devices, each slave device has a chip select pin connected to the

master device.
When master device need to communicates with a slave device. We need to pull the chip select pin level of the slave device low or high.



**3. Experimental principle**

TF possess 4 data transmission terminals, DAT0, DAT1, DAT2, DAT3. There is also a CMD pin, which is used to read the information in the card.

The functions of the main pins of the TF card:

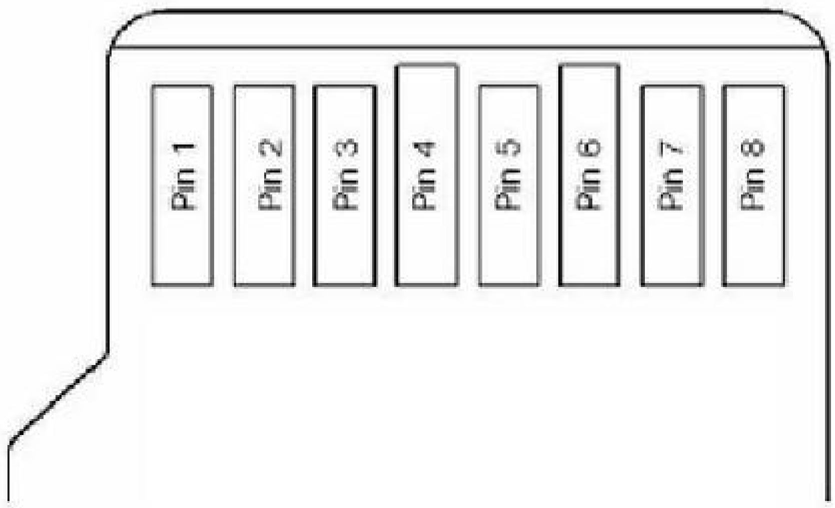
**CLK:** clock signal, each clock cycle transmits a command or data bit. (frequency0-25MHz).

**CMD**: Two-way command and reply line.

**DAT0～3**: Data line, data can be transmitted from TF card to host, or from host to TF card.

TF card transmission data possess two modes, SD mode and SPI mode.

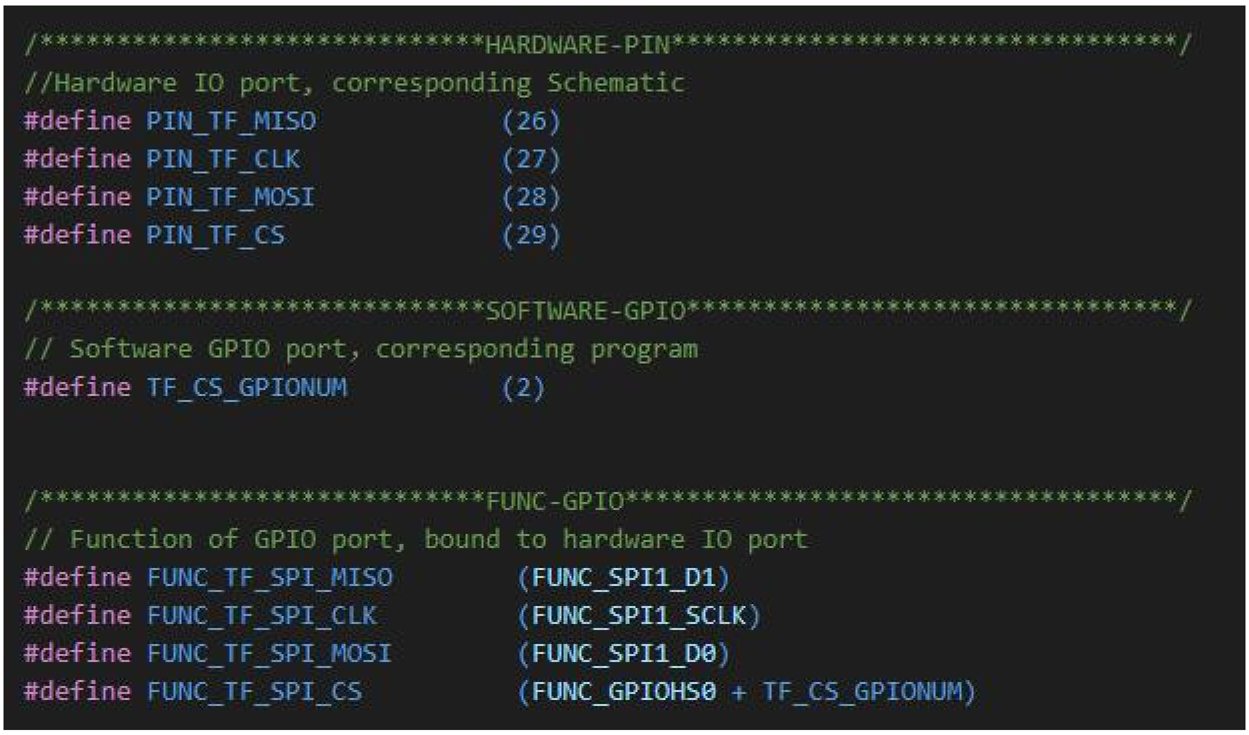
For K210 board, we transmit data in SPI mode. The SPI mode pins are as follows: 1: CS, 2: DI, 3: VSS, 4: VDD, 5: SCLK, 6: VSS2, 7: DO, 8: RSV, 9: RSV.



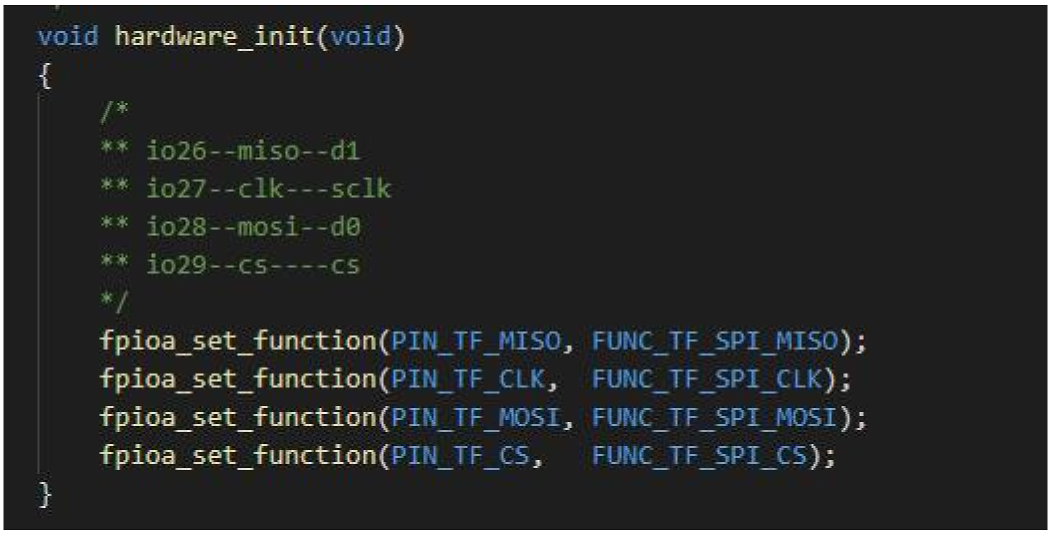
| PIN | SPI Mode | | |
|---|---|---|---|
| | name | type | remark |
| 1 | RSV | | reserve |
| 2 | CS/CD | Input | chip select |
| 3 | DI/CMD | Input | data input |
| 4 | VDD | Power | positive pole |
| 5 | SCLK | Input | clock |
| 6 | VSS | GND | GND |
| 7 | DO/DAT0 | Output | data output |
| 8 | RSV | | reserve |

**4. Experiment procedure**

4.1 According to the above hardware connection pin diagram, K210 hardware pins and software functions use FPIOA mapping relationship.

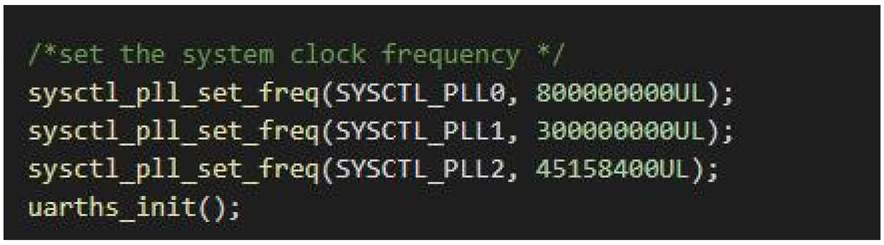


```
/*****************************HARDWARE-PIN*********************************/
//Hardware IO port, corresponding Schematic
#define PIN_TF_MISO             (26)
#define PIN_TF_CLK              (27)
#define PIN_TF_MOSI             (28)
#define PIN_TF_CS               (29)

/*****************************SOFTWARE-GPIO********************************/
// Software GPIO port, corresponding program
#define TF_CS_GPIONUM           (2)


/*****************************FUNC-GPIO************************************/
// Function of GPIO port, bound to hardware IO port
#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)
```



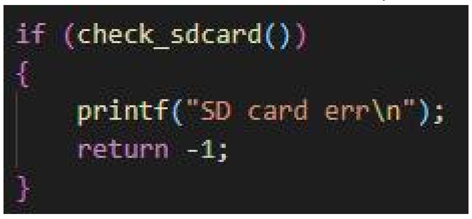
```c
void hardware_init(void)
{
    /*
    ** io26--miso--d1
    ** io27--clk---sclk
    ** io28--mosi--d0
    ** io29--cs----cs
    */
    fpioa_set_function(PIN_TF_MISO, FUNC_TF_SPI_MISO);
    fpioa_set_function(PIN_TF_CLK,  FUNC_TF_SPI_CLK);
    fpioa_set_function(PIN_TF_MOSI, FUNC_TF_SPI_MOSI);
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

4.2 Set the system clock frequency. Since the uarths clock comes from PLL0, you need to reinitialize the following uarths after setting PLL0, otherwise printf may print garbled characters.



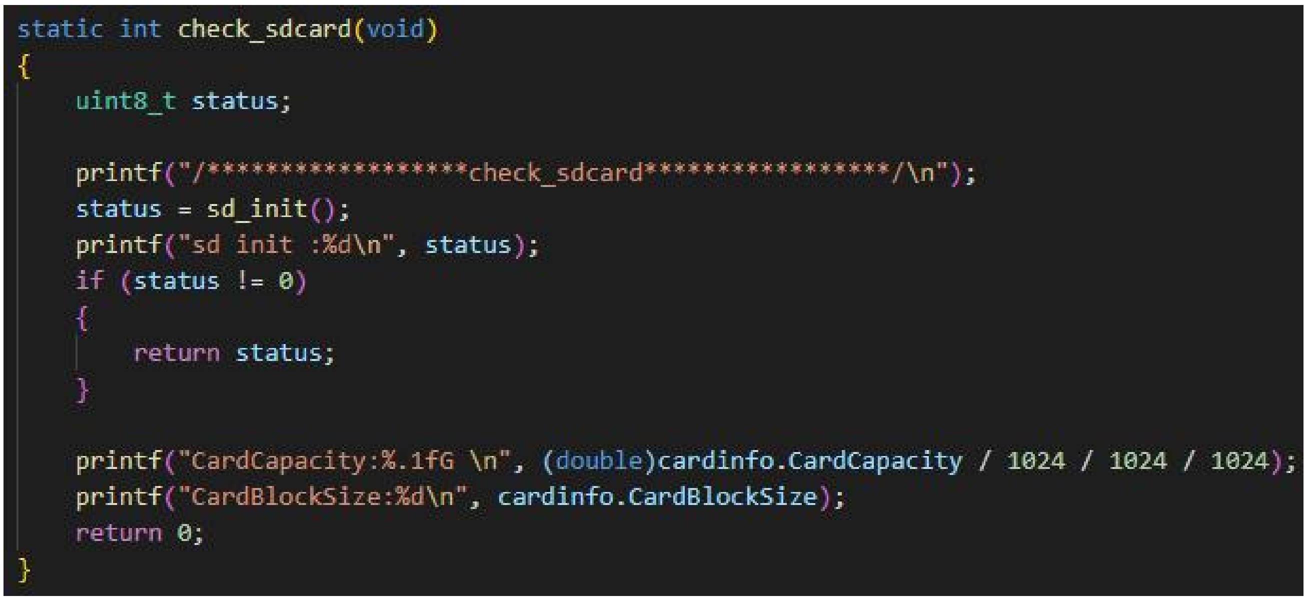
```c
/*set the system clock frequency */
sysctl_pll_set_freq(SYSCTL_PLL0, 800000000UL);
sysctl_pll_set_freq(SYSCTL_PLL1, 300000000UL);
sysctl_pll_set_freq(SYSCTL_PLL2, 45158400UL);
uarths_init();
```

4.3 Check if there is a TF card, or if the TF card is normal, if it is not normal, it will exit.



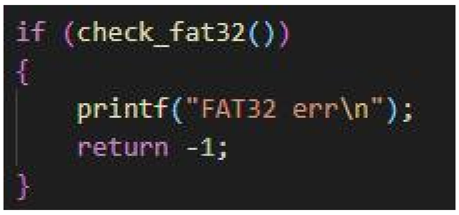
```c
if (check_sdcard())
{
    printf("SD card err\n");
    return -1;
}
```

If the TF card is initialized successfully, the capacity of the TF will be printed out in G.



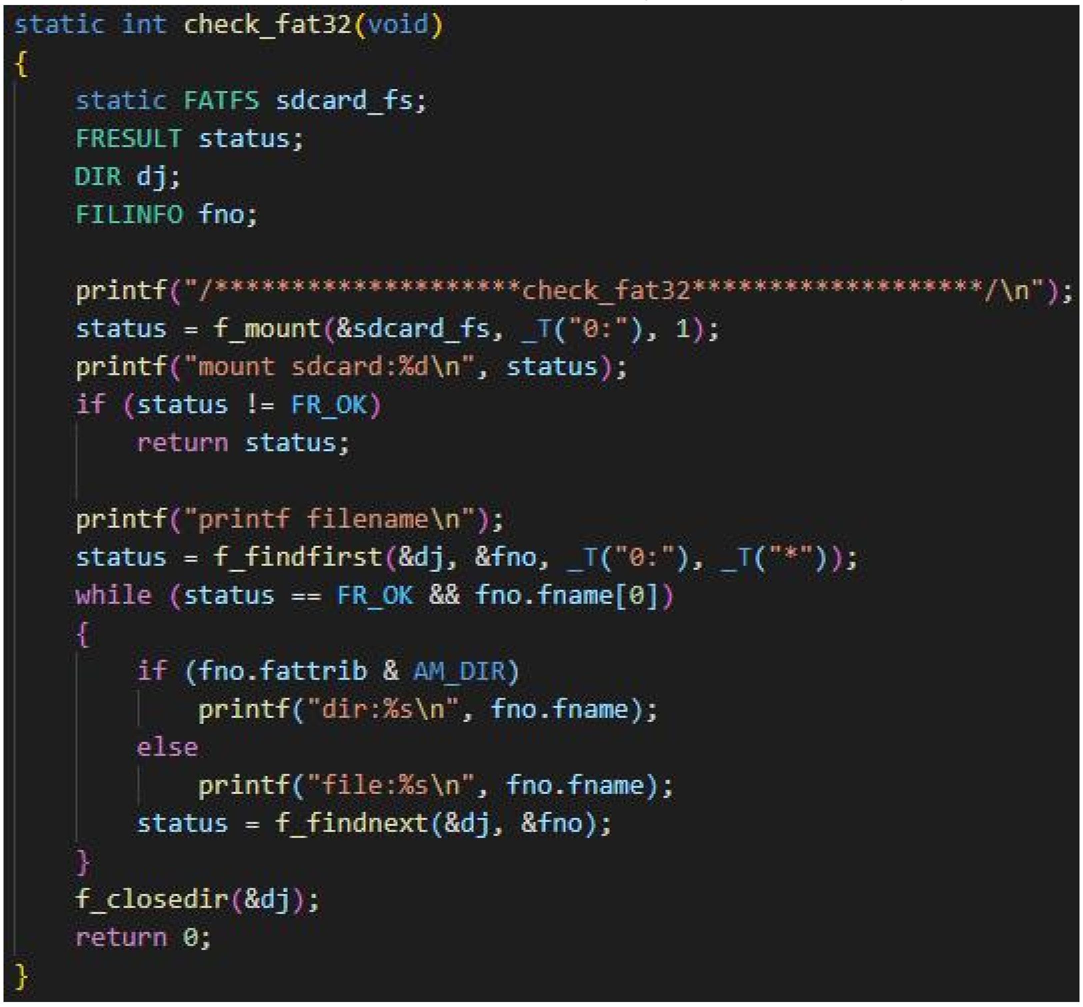
```c
static int check_sdcard(void)
{
    uint8_t status;

    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;
}
```

4.4 Check if the format of the TF card is FAT32, if not, exit.



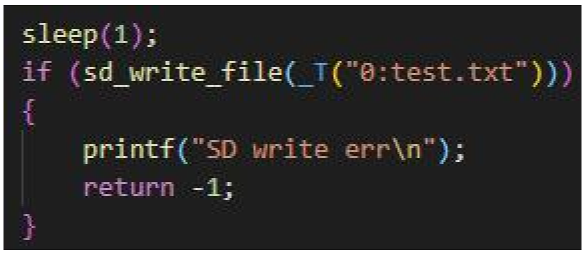
```
if (check_fat32())
{
    printf("FAT32 err\n");
    return -1;
}
```

If the TF card conforming to the FAT32 format is detected, the TF card is hung to "0:", and the names of the files and folders in the root directory of the TF card are printed out.



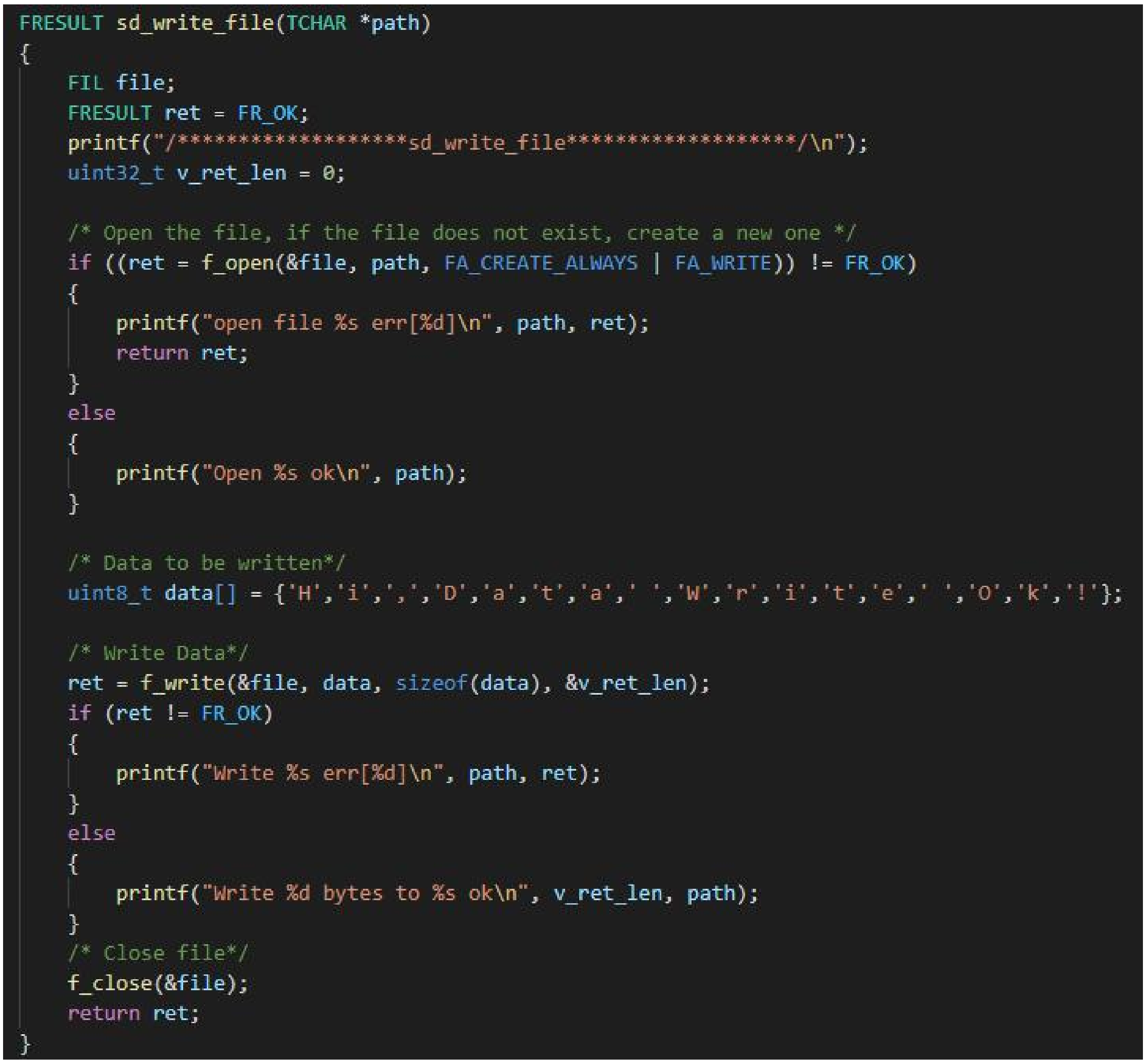
```
static int check_fat32(void)
{
    static FATFS sdcard_fs;
    FRESULT status;
    DIR dj;
    FILINFO fno;

    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;
}
```

4.5 Write a file to the TF card, save it in test.txt in the TF card and the directory. If the writing fails, it will exit.



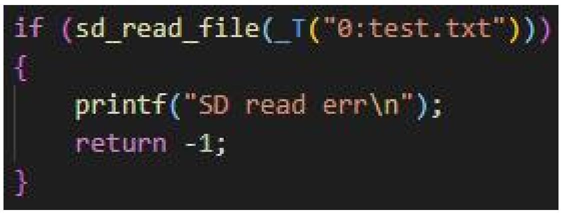
```
sleep(1);
if (sd_write_file(_T("0:test.txt")))
{
    printf("SD write err\n");
    return -1;
}
```

Before writing the file, you need to open the file. If the file does not exist, you need to create a new file. The written data needs to be converted to uint8_t type. After writing the data, you must execute f_close to close the file.



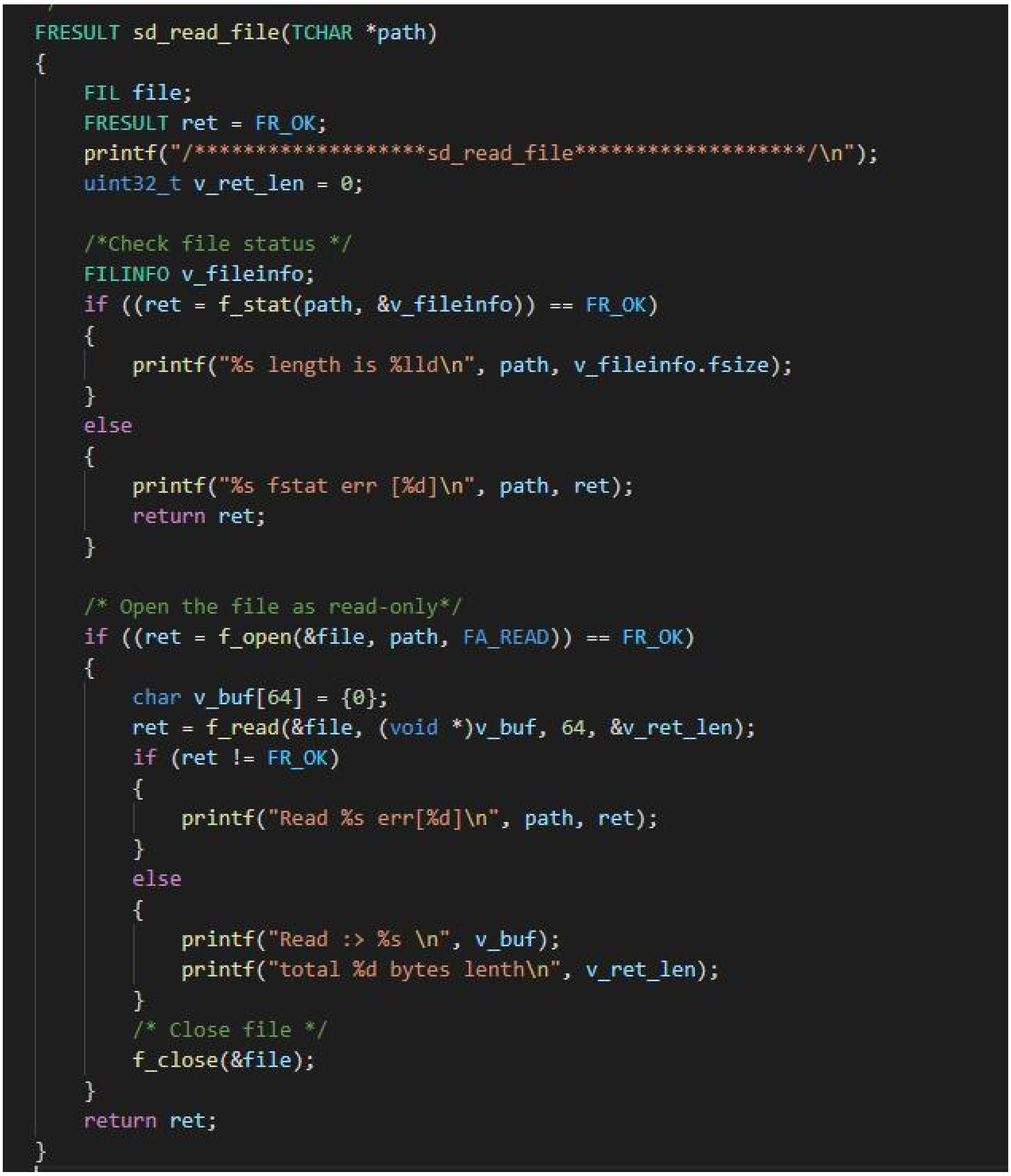
```
FRESULT sd_write_file(TCHAR *path)
{
    FIL file;
    FRESULT ret = FR_OK;
    printf("/*******************sd_write_file*******************/\n");
    uint32_t v_ret_len = 0;

    /* Open the file, if the file does not exist, create a new one */
    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);
    }

    /* Data to be written*/
    uint8_t data[] = {'H','i',',','D','a','t','a',' ','W','r','i','t','e',' ','O','k','!'};

    /* Write Data*/
    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);
    }
    /* Close file*/
    f_close(&file);
    return ret;
}
```

4.6 Read the file from the TF card, the file is test.txt in the root directory of the TF card.



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

Before reading the file, it is necessary to judge the status of the file and open the file. If the file does not exist or has an error, it will return. If it is normal, it will open the file in read-only mode and send the read data through the serial port.



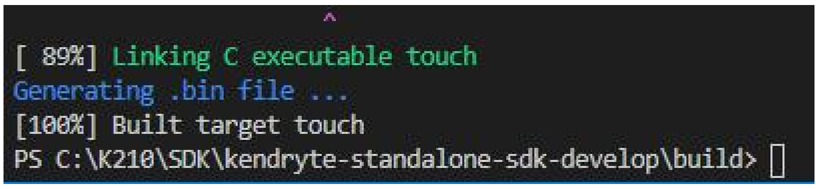
```
FRESULT sd_read_file(TCHAR *path)
{
    FIL file;
    FRESULT ret = FR_OK;
    printf("/*******************sd_read_file*******************/\n");
    uint32_t v_ret_len = 0;

    /*Check file status */
    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;
    }

    /* Open the file as read-only*/
    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);
        }
        /* Close file */
        f_close(&file);
    }
    return ret;
}
```

4.9 Compile and debug, burn and run

Copy the sdcard to the src directory in the SDK.

Then, enter the build directory and run the following command to compile.

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

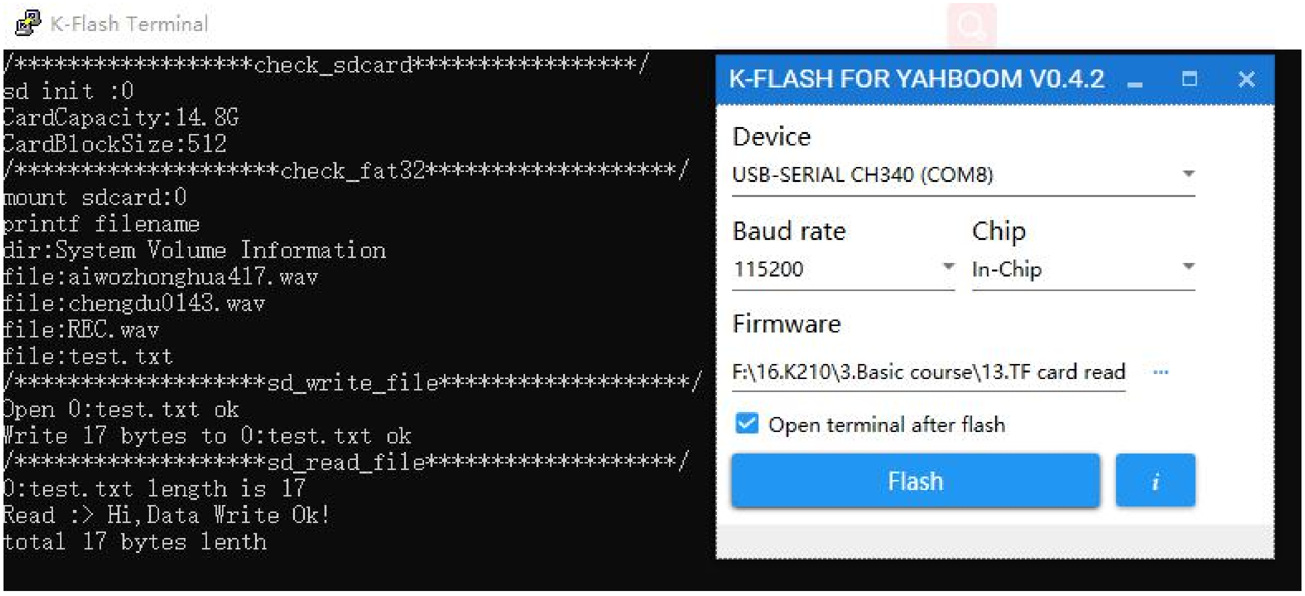


```
[ 89%] Linking C executable touch
Generating .bin file ...
[100%] Built target touch
PS C:\K210\SDK\kendryte-standalone-sdk-develop\build>
```

After the compilation is complete, the **sdcard.bin** file will be generated in the build folder.
We need to use the type-C data cable to connect the computer and the K210 development board.
Open kflash, select the corresponding device, and then burn the **sdcard.bin** file to the K210 development board.

## 5. Experimental phenomenon

After the firmware is write, a terminal interface will pop up. As shown below.

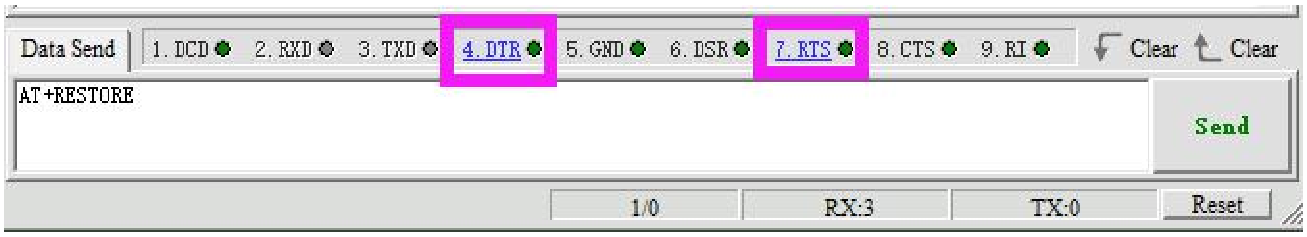


K-Flash Terminal

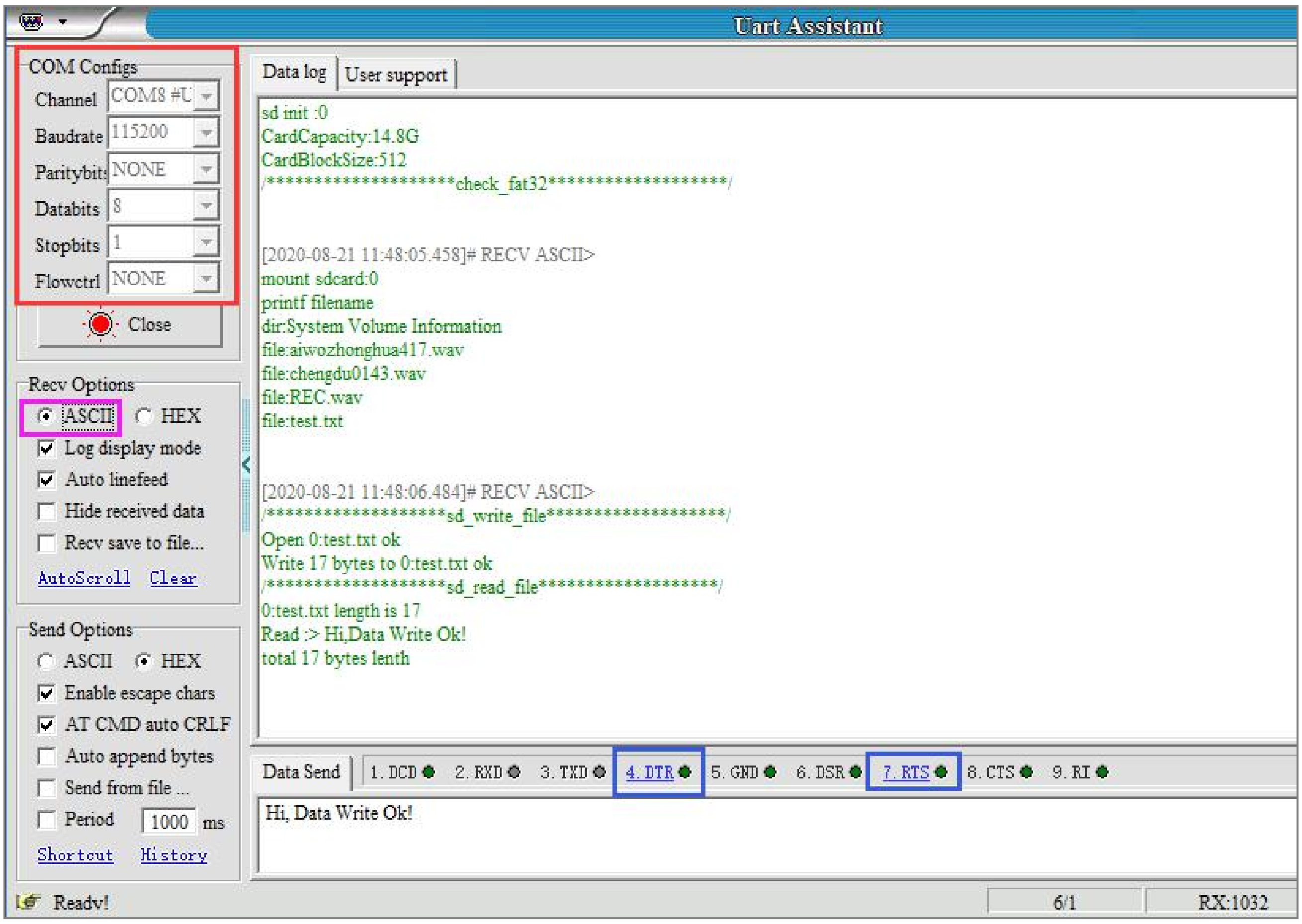
```
/*****************check_sdcard****************/
sd init :0
CardCapacity:14.8G
CardBlockSize:512
/*******************check_fat32******************/
mount sdcard:0
printf filename
dir:System Volume Information
file:aiwozhonghua417.wav
file:chengdu0143.wav
file:REC.wav
file:test.txt
/******************sd_write_file******************/
Open 0:test.txt ok
Write 17 bytes to 0:test.txt ok
/******************sd_read_file******************/
0:test.txt length is 17
Read :> Hi,Data Write Ok!
total 17 bytes lenth
```

K-FLASH FOR YAHBOOM V0.4.2
Device: USB-SERIAL CH340 (COM8)
Baud rate: 115200
Chip: In-Chip
Firmware: F:\16.K210\3.Basic course\13.TF card read
✓ Open terminal after flash

Close this terminal interface. Open the serial port assistant of the computer, select the corresponding serial port number of the corresponding K210 development board, set the baud rate to 115200.
Then, click to open the serial port assistant.
**!Note: you also need to set the DTR and RTS of the serial port assistant.**
Click 4.DTR and 7.RTS to set them to green.



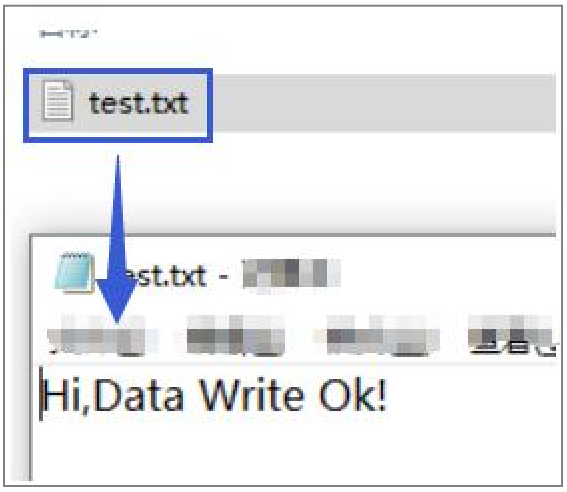
**Press reset button of K210 development board.** We can see that the capacity of the TF card and the file name in the root directory are displayed on the serial port assistant, and then write 'Hi, Data Write Ok!' in the test.txt file, and send it to the serial port by reading the test.txt file.



```
sd init :0
CardCapacity:14.8G
CardBlockSize:512
/********************check_fat32*******************/


[2020-08-21 11:48:05.458]# RECV ASCII>
mount sdcard:0
printf filename
dir:System Volume Information
file:aiwozhonghua417.wav
file:chengdu0143.wav
file:REC.wav
file:test.txt


[2020-08-21 11:48:06.484]# RECV ASCII>
/*******************sd_write_file*******************/
Open 0:test.txt ok
Write 17 bytes to 0:test.txt ok
/*******************sd_read_file*******************/
0:test.txt length is 17
Read :> Hi,Data Write Ok!
total 17 bytes lenth
```

```
Hi, Data Write Ok!
```

Using a card reader to read the test.txt file from the TF card, and the content in it is the same as that read from our K210 development board.



```
Hi,Data Write Ok!
```

**6. Experiment summary**

6.1 The file must be opened before TF card reads or writes the file, and the file must be closed after the read and write operation is over.

6.2 TF card communicates via SPI, and reads and writes data with uint8_t unit.

6.3 Every time after the firmware is burned, the K210 development board needs to be powered on again, otherwise the TF card will initialization fails and the system exits.