

## 3. 17flash 读写

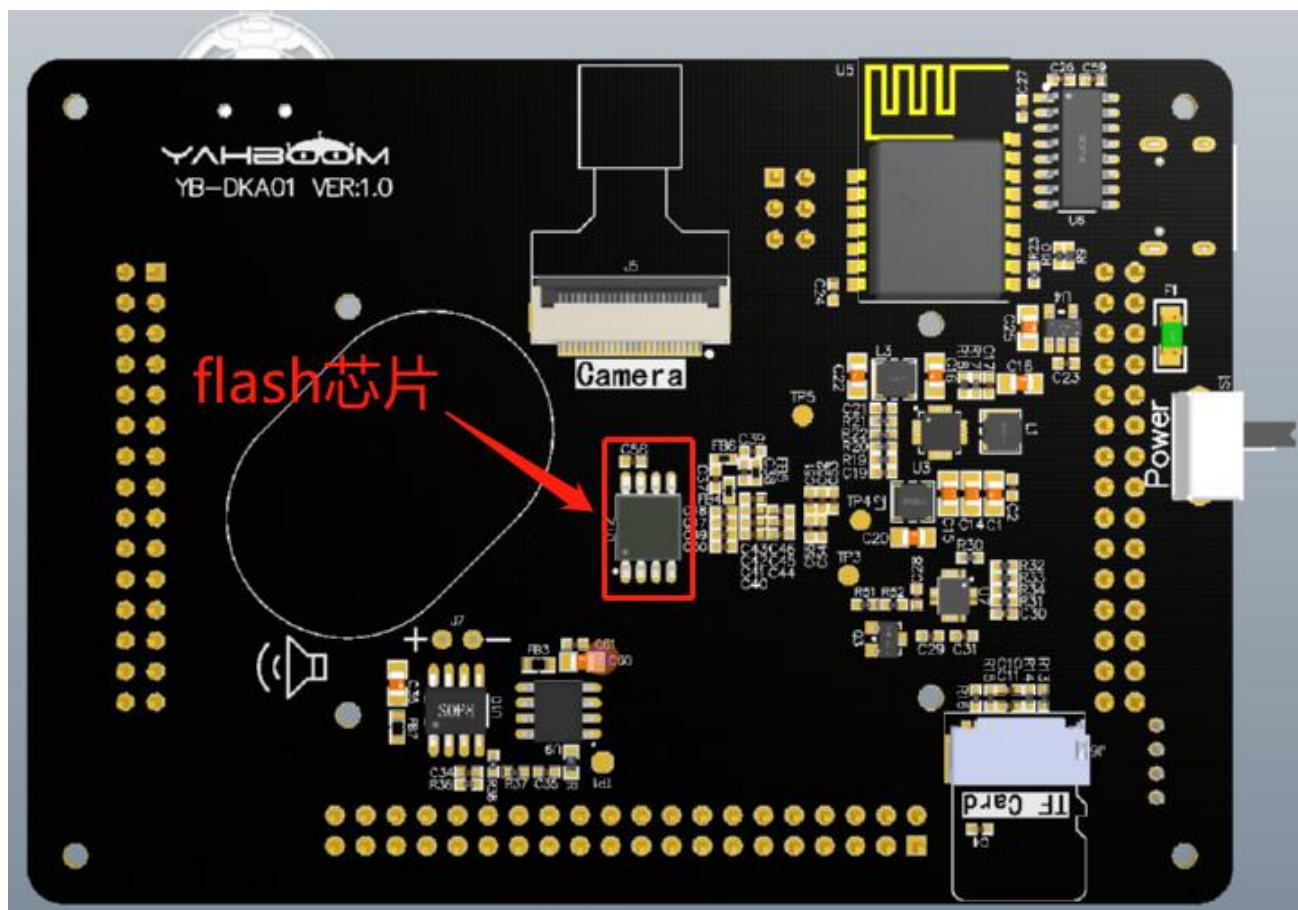
### 一、实验目的

本节课主要学习 K210 读写 flash 芯片的功能。

### 二、实验准备

#### 1. 实验元件

flash 芯片 GD25LQ128C。



#### 2. 元件特性

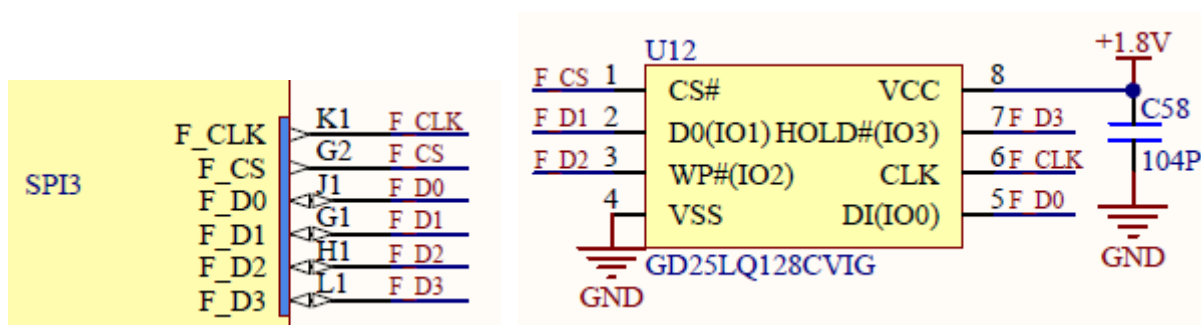
flash 芯片 GD25LQ128C 是通过 SPI 串行闪存的芯片，具有 128M-bit(16 兆字节 MByte)空间，能够储存声音、文本和数据等，设备运行电源为 2.7V~3.6V，低功耗模式电流低至 1uA。

写数据，每次向 GD25LQ128C 写入数据都需要按照页或者扇区或者簇为单位进行，一页为 256 个字节，一个扇区为 4K 个字节（16 页），一次最多写一页，也就是一次最多写 256 个字节，如果超过一页数据长度，则分多次完成。

读数据，可以从任何地址读出。

擦除数据，最小单位为一个扇区，也可以直接擦除整个 flash 芯片。

### 3. 硬件连接



### 4. SDK 中对应 API 功能

flash 使用的是 SPI 的通讯方式，所以对应的头文件是 spi.h，而由于 flash 芯片的特殊性，kendryte 官方已经写好了 flash 的库，对应的头文件是 w25qxx.h。

为用户提供以下接口：

- w25qxx\_init: 初始化 flash 芯片，主要是设置 SPI 的设备、通道和速率等。
- w25qxx\_read\_id: 读取 flash 的 ID。
- w25qxx\_write\_data: 向 flash 写入数据。
- w25qxx\_read\_data: 从 flash 读取数据。

## 三、实验原理

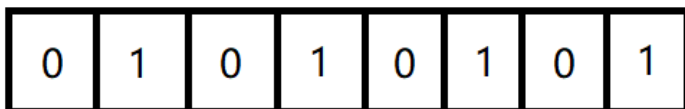
FLASH 芯片是应用非常广泛的存储材料，与之对应的是 RAM 芯片，区别在于 FLASH 芯片断电后数据可以保存，而 RAM 芯片断电后数据不会保存。那么 FLASH 是如何工作的呢？计算机的储存方式是二进制，也就是 0 和 1，在二进制中，0 和 1 可以组成任何数。FLASH 芯片对 0 和 1 的处理方式是用物质填充，1 则填充，

0 则不填充，如下图所示，这样就算断电之后，物质的性质也不会因为没有电而改变，所以再次读取数据的时候数据依然不变，这样就可以做到断电保存。

FLASH存  
储方式



计算机存  
储方式



#### 四、实验过程

1. 设置系统时钟 PLL0 频率。

```
/* 设置新PLL0频率 */
sysctl_pll_set_freq(SYSCTL_PLL0, 800000000);
uarths_init();
```

2. 使用的是 spi3 的 CS0 通道初始化 flash，读取 flash 的 ID 进行对比，如果发现不对则打印错误信息，并且返回 0。

```
int flash_init(void)
{
    uint8_t manuf_id, device_id;
    uint8_t spi_index = 3, spi_ss = 0;
    printf("flash init \n");

    w25qxx_init(spi_index, spi_ss, 60000000);
    /* 读取flash的ID */
    w25qxx_read_id(&manuf_id, &device_id);
    printf("manuf_id:0x%02x, device_id:0x%02x\n", manuf_id, device_id);
    if ((manuf_id != 0xEF && manuf_id != 0xC8) || (device_id != 0x17 && device_id != 0x16))
    {
        /* flash初始化失败 */
        printf("w25qxx_read_id error\n");
        printf("manuf_id:0x%02x, device_id:0x%02x\n", manuf_id, device_id);
        return 0;
    }
    else
    {
        return 1;
    }
}
```

3. 初始化数据，write\_buf 为写入的数据，赋予初始值，read\_buf 为读取的数据，清空所有值为 0。

```
/* 给缓存写入的数据赋值 */
for (int i = 0; i < BUF_LENGTH; i++)
    write_buf[i] = (uint8_t)(i);

/* 清空读取的缓存数据 */
for(int i = 0; i < BUF_LENGTH; i++)
    read_buf[i] = 0;
```

4. 先把缓存的数据 write\_buf 写入到 FLASH 里面，并且打印写入总共花费的时间。

```
void flash_write_data(uint8_t *data_buf, uint32_t length)
{
    uint64_t start = sysctl_get_time_us();
    /* flash写入数据 */
    w25qxx_write_data(DATA_ADDRESS, data_buf, length);
    uint64_t stop = sysctl_get_time_us();
    /* 打印写入数据的时间(us) */
    printf("write data finish:%ld us\n", (stop - start));
}
```

5. 再将数据从 FLASH 芯片中读取出来，打印读取的时间。

```
void flash_read_data(uint8_t *data_buf, uint32_t length)
{
    uint64_t start = sysctl_get_time_us();
    /* flash读取数据 */
    w25qxx_read_data(DATA_ADDRESS, data_buf, length);
    uint64_t stop = sysctl_get_time_us();
    /* 打印读取数据的时间(us) */
    printf("read data finish:%ld us\n", (stop - start));
}
```

6. 最后是对比 write\_buf 和 read\_buf 的差异，如果有不同则打印错误信息并退出，如果相同则提示 OK。

```

printf("flash start write data\n");

/* flash写入数据 */
flash_write_data(write_buf, BUF_LENGTH);

/*flash读取数据*/
flash_read_data(read_buf, BUF_LENGTH);

/* 比较数据，如果有不同则打印错误信息 */
for (int i = 0; i < BUF_LENGTH; i++)
{
    if (read_buf[i] != write_buf[i])
    {
        printf("flash read error\n");
        return 0;
    }
}
printf("spi3 flash master test ok\n");
while (1)
;
return 0;

```

## 7. 编译调试，烧录运行

把本课程资料中的 flash 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=flash -G "MinGW Makefiles"

make

```

```

[ 95%] Building C object CMakeFiles/flash.dir/src/flash/main.c.obj
[ 97%] Linking C executable flash
Generating .bin file ...
[100%] Built target flash
PS C:\K210\SDK\kendryte-standalone-sdk-develop\build>

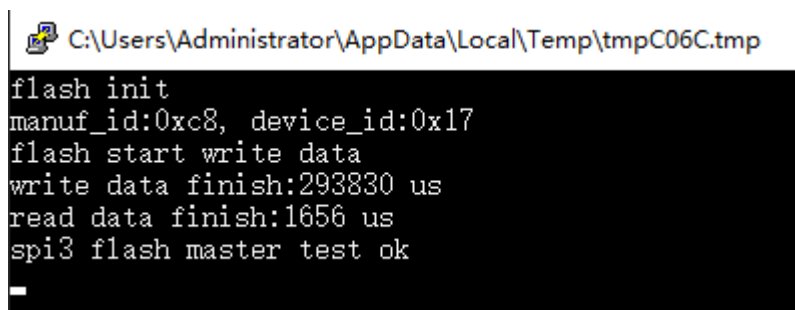
```

编译完成后，在 build 文件夹下会生成 flash.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。

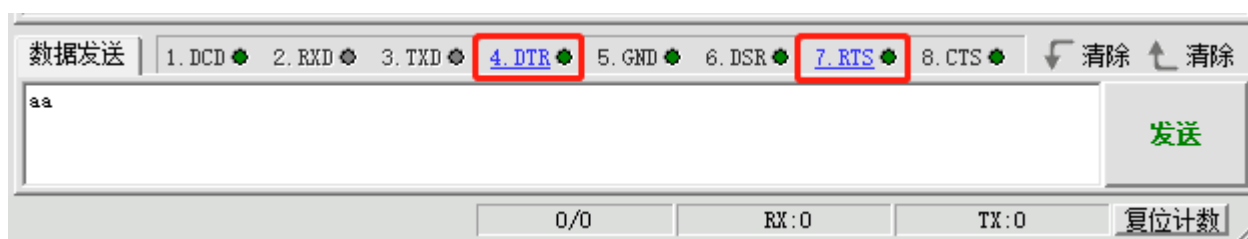
## 五、实验现象

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。

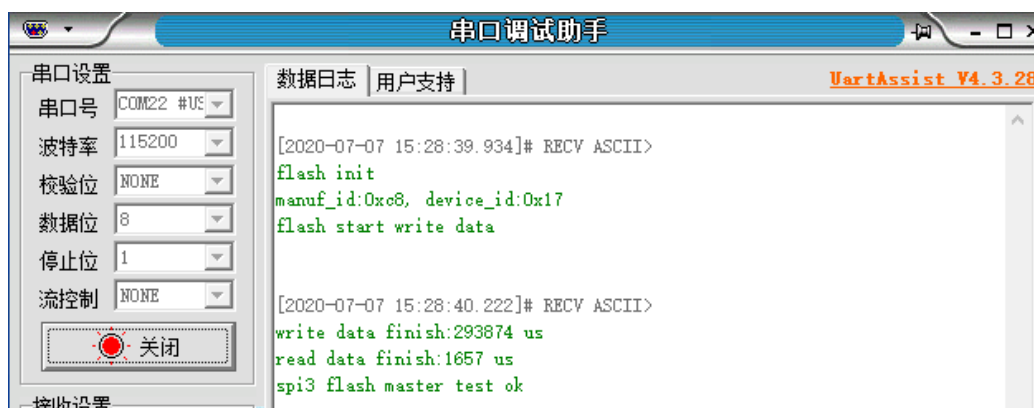


```
C:\Users\Administrator\AppData\Local\Temp\tmpC06C.tmp  
flash init  
manuf_id:0xc8, device_id:0x17  
flash start write data  
write data finish:293830 us  
read data finish:1656 us  
spi3 flash master test ok  
_
```

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。



可以看到串口助手打印打印出 flash 初始化以及写入和读取的时间，当看到最后是“spi3 flash master test ok”则表示读写成功，如果看到的是“flash read error”则表示读写失败。



## 六、实验总结

1. GD25LQ128C 的一款存储空间为 16MB 的 FLASH 芯片，总共有 4096 个扇区，每个扇区有 16 页，每页是 256 字节。
2. FLASH 是能够断电保存数据的一种储存方式。
3. FLASH 的擦除最小单位为扇区，也就是 4K。

附：API

对应的头文件 wdt.h

### w25qxx\_init

#### 描述

初始化 flash 芯片，设置 SPI 设备号、通道号和速率。

#### 函数原型

```
w25qxx_status_t w25qxx_init(uint8_t spi_index, uint8_t spi_ss, uint32_t rate)
```

#### 参数

参数名称	描述	输入输出
spi_index	SPI 设备号	输入
spi_ss	CS 片选	输入
rate	通讯速率	输入

#### 返回值

返回 flash 的状态。

### w25qxx\_read\_id

#### 描述

读取 flash 的 ID 号。

## 函数原型

```
w25qxx_status_t w25qxx_read_id(uint8_t *manuf_id, uint8_t *device_id)
```

## 参数

参数名称	描述	输入输出
manuf_id	工厂 ID	输出
device_id	设备 ID	输出

## 返回值

返回 flash 的状态。

## w25qxx\_write\_data

### 描述

向 flash 写入数据。

## 函数原型

```
w25qxx_status_t w25qxx_write_data(uint32_t addr, uint8_t *data_buf, uint32_t length)
```

## 参数

参数名称	描述	输入输出
addr	Flash 地址	输入
data_buf	写入的数据	输入
length	数据长度	输入

## 返回值

返回 flash 的状态。

## w25qxx\_read\_data

### 描述

从 flash 读取数据。



## 函数原型

```
w25qxx_status_t w25qxx_read_data(uint32_t addr, uint8_t *data_buf, uint32_t length)
```

## 参数

参数名称	描述	输入输出
addr	Flash 地址	输入
data_buf	读取的数据	输出
length	数据长度	输入

## 返回值

返回 flash 的状态。

## 数据类型

相关数据类型、数据结构定义如下：

- w25qxx\_status\_t
- w25qxx\_read\_t

### w25qxx\_status\_t

#### 描述

flash 芯片状态。

#### 定义

```
typedef enum _w25qxx_status
{
    W25QXX_OK = 0,
    W25QXX_BUSY,
    W25QXX_ERROR,
} w25qxx_status_t;
```

## 成员

成员名称	描述
W25QXX_OK	flash OK
W25QXX_BUSY	flash 忙碌
W25QXX_ERROR	flash 出错

## w25qxx\_read\_t

### 描述

spi 读取 flash 的方式。

### 定义

```
typedef enum _w25qxx_read
{
    W25QXX_STANDARD = 0,
    W25QXX_STANDARD_FAST,
    W25QXX_DUAL,
    W25QXX_DUAL_FAST,
    W25QXX_QUAD,
    W25QXX_QUAD_FAST,
} w25qxx_read_t;
```

### 成员

成员名称	描述
W25QXX_STANDARD	标准模式
W25QXX_STANDARD_FAST	标准快速模式
W25QXX_DUAL	双线模式
W25QXX_DUAL_FAST	双线快速模式
W25QXX_QUAD	四线模式
W25QXX_QUAD_FAST	四线快速模式