3.17 flash read write

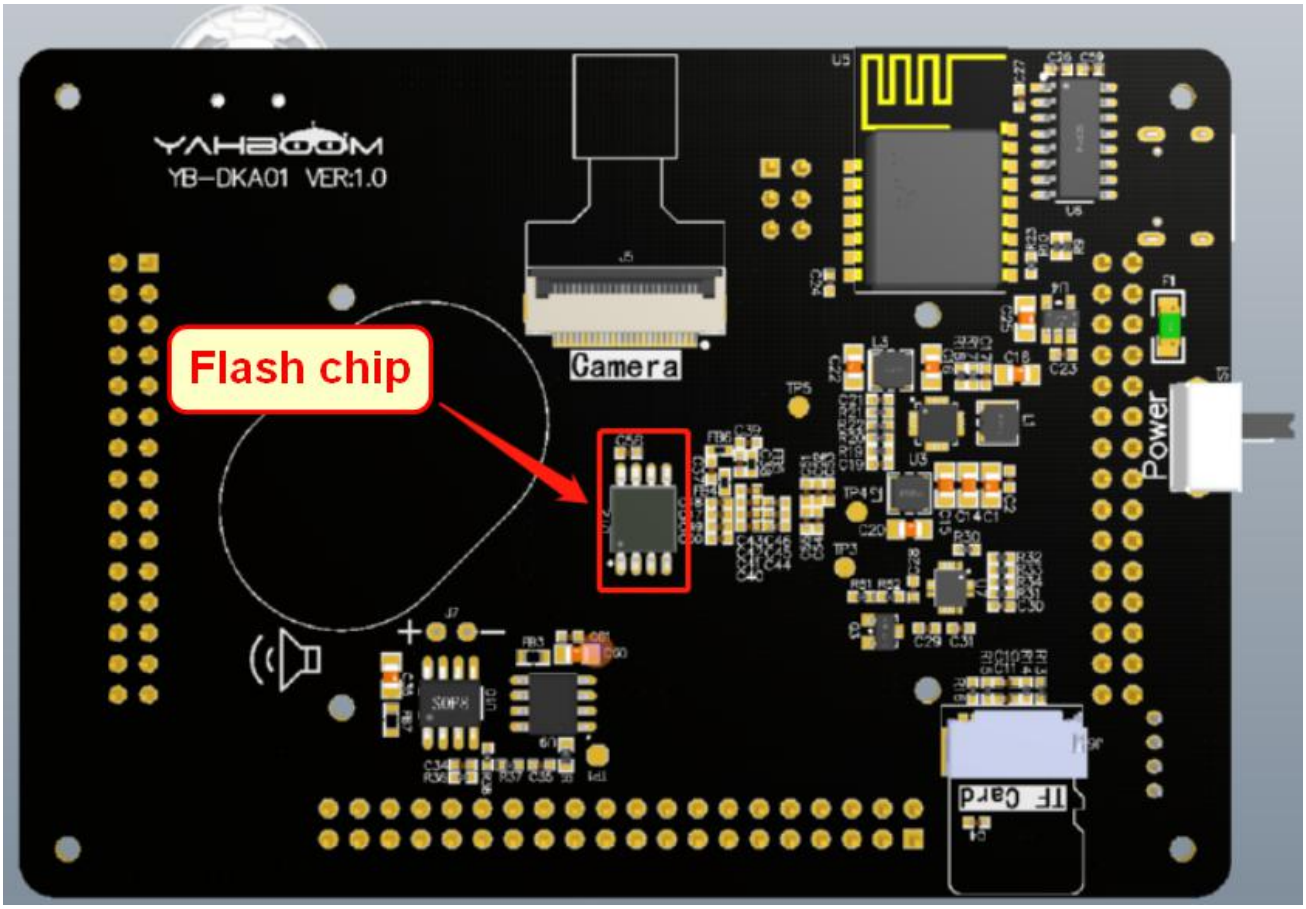
1. Experiment purpose

In this lesson, we mainly learn k210 read and write flash chip.

2.Experiment preparation

2.1 components

flash chip GD25LQ128C



2. Component characteristics

The flash chip GD25LQ128C is a chip that uses SPI serial flash memory. It has a 128M-bit (16 megabyte MByte) space and can store voice, text, data, etc.

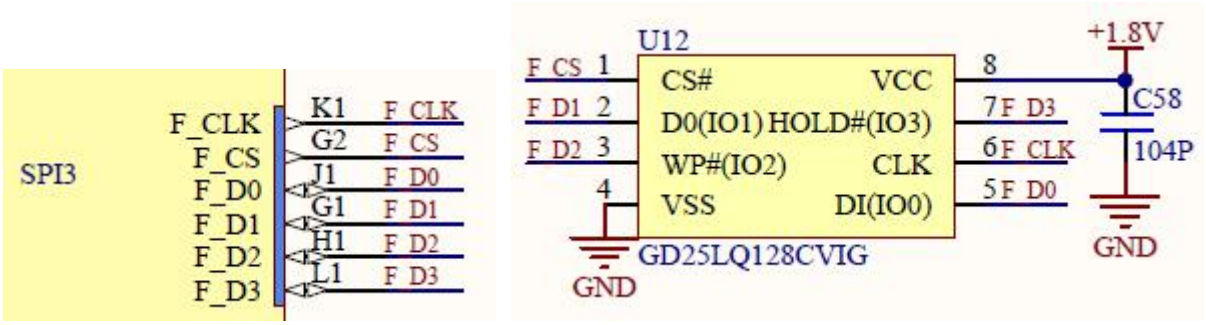
The operating power of the device is 2.7V~3.6V, and the low power consumption mode has low current. To 1uA.

Writing data: Each time you write data to GD25LQ128C, you need to do it in units of pages or sectors or clusters. One page is 256 bytes, and one sector is 4K bytes (16 pages). You can write one page at a time. That is, up to 256 bytes are written at a time, and if the data length exceeds one page, it will be completed in multiple times.

Reading data: Data can be read from any address.

Erase data: the smallest unit is a sector, or the entire flash chip can be erased directly.

2.3 Hardware connection



2.4 SDK API function

The header file is spi.h and w25qxx.h

The flash uses the SPI communication method.

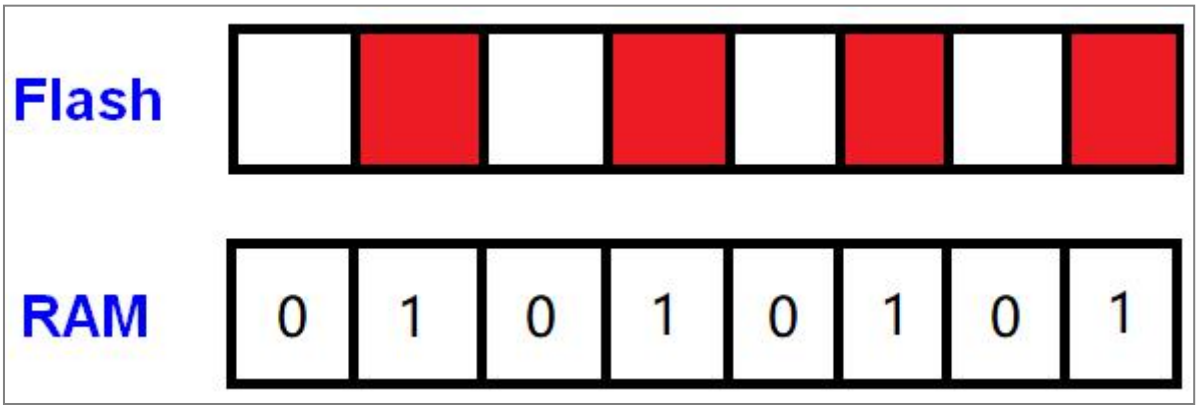
Due to the particularity of the flash chip, kendryte has officially written the flash library, and the corresponding header file is w25qxx.h.

We will provide following interfaces to users:

- w25qxx_init: Initialize the Flash chip, setting the device, channel, and rate of the SPI.
- w25qxx_read_id: Read Flash ID
- w25qxx_write_data: Write data to flash
- w25qxx_read_data: Read data from flash

3. Experimental principle

The data can be saved after the FLASH chip is powered off, but the data will not be saved after the RAM chip is powered off.



4. Experiment procedure

4.1 Set system clock PLL0 frequency.

```
/* Set new PLL0 frequency*/
sysctl_pll_set_freq(SYSCTL_PLL0, 800000000);
uarths_init();
```

4.2 CS0 channel of spi3 is used to initialize the flash. The ID of the flash is read for comparison. If it is found to be incorrect, an error message is printed, it will return 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);
    /* Read flash 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 failed to initialize */
        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;
    }
}
```

4.3 Initialize the data, write_buf is the data to be written, give the initial value, read_buf is the data to be read, clear all values to 0.

```
/* Assign a value to the data written to the cache */
for (int i = 0; i < BUF_LENGTH; i++)
    write_buf[i] = (uint8_t)(i);

/* Clear read cached data */
for(int i = 0; i < BUF_LENGTH; i++)
    read_buf[i] = 0;
```

4.4 Writing the cached data write_buf into FLASH, and print the total time spent in writing.

```
void flash_write_data(uint8_t *data_buf, uint32_t length)
{
    uint64_t start = sysctl_get_time_us();
    /* flash write data  */
    w25qxx_write_data(DATA_ADDRESS, data_buf, length);
    uint64_t stop = sysctl_get_time_us();
    /* Print and write data time (us) */
    printf("write data finish:%ld us\n", (stop - start));
}
```

4.5 Reading the data from the FLASH chip, and print the read time.

```c
void flash_read_data(uint8_t *data_buf, uint32_t length)
{
    uint64_t start = sysctl_get_time_us();
    /* flash read data */
    w25qxx_read_data(DATA_ADDRESS, data_buf, length);
    uint64_t stop = sysctl_get_time_us();
    /* Print read data time(us) */
    printf("read data finish:%ld us\n", (stop - start));
}
```

4.6 Compare the difference between write_buf and read_BUf, print an error message and exit if different, and say OK if same.

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

/* flash write data */
flash_write_data(write_buf, BUF_LENGTH);

/*flash read data*/
flash_read_data(read_buf, BUF_LENGTH);

/* Compare data and print error messages if t
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;
```

4.7 Compile and debug, burn and run

Copy the flash to the src directory in the SDK.

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

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>
```

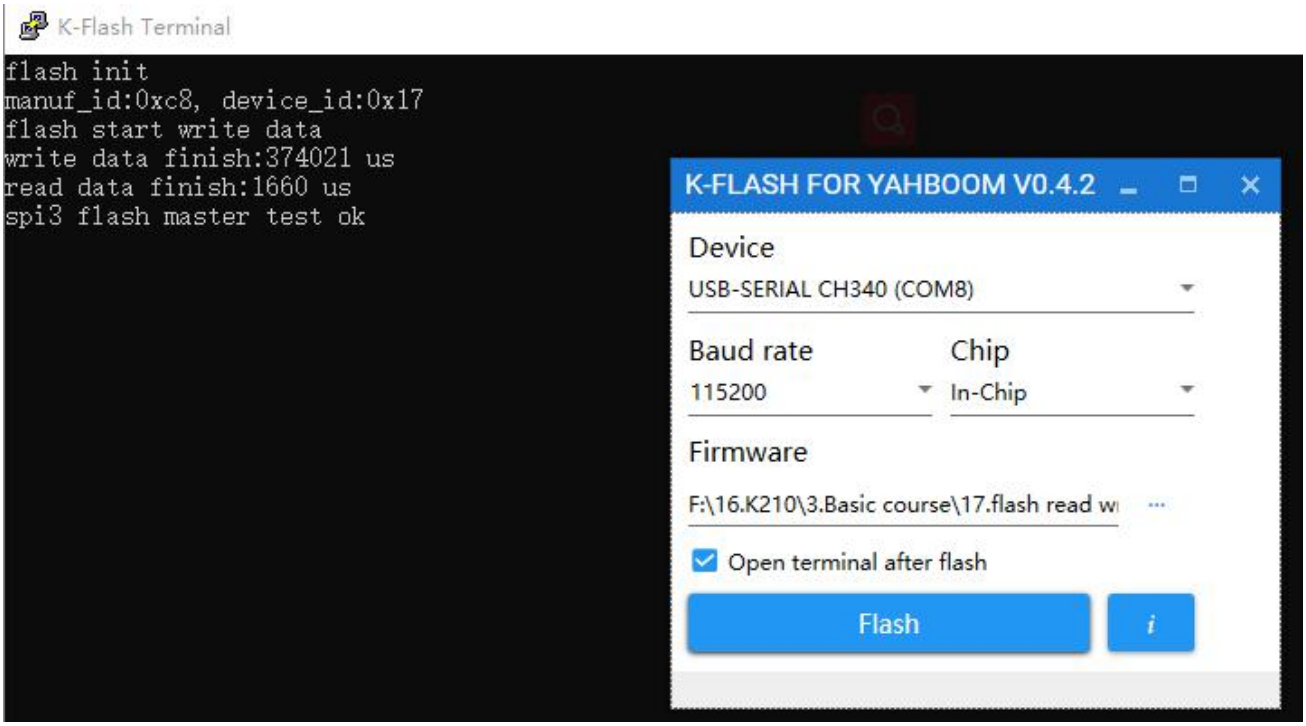
After the compilation is complete, the **flash.bin** file will be generated in the build folder.

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 **flash.bin** file to the K210 development board.

5.Experimental phenomenon

After the firmware is burned, a terminal interface will pop up. If there is no terminal interface, as shown below.

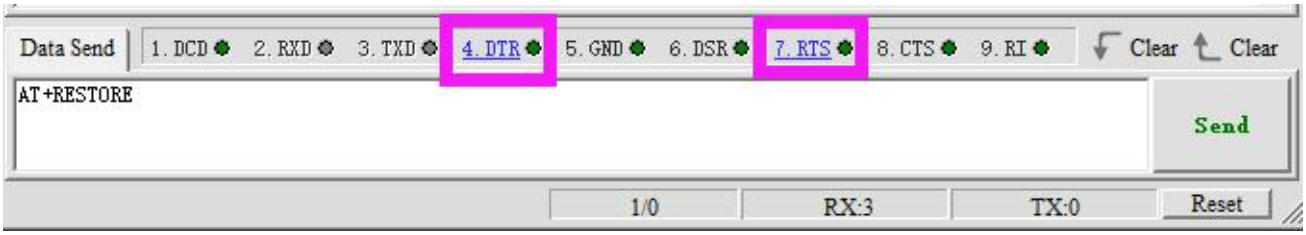


If you can't see terminal interface, please open the serial port assistant of the computer, select the corresponding serial port number of the corresponding K210 development board, set the baud rate to 115200.

Then, click to open the serial port assistant.

!Note: you also need to set the DTR and RTS of the serial port assistant.

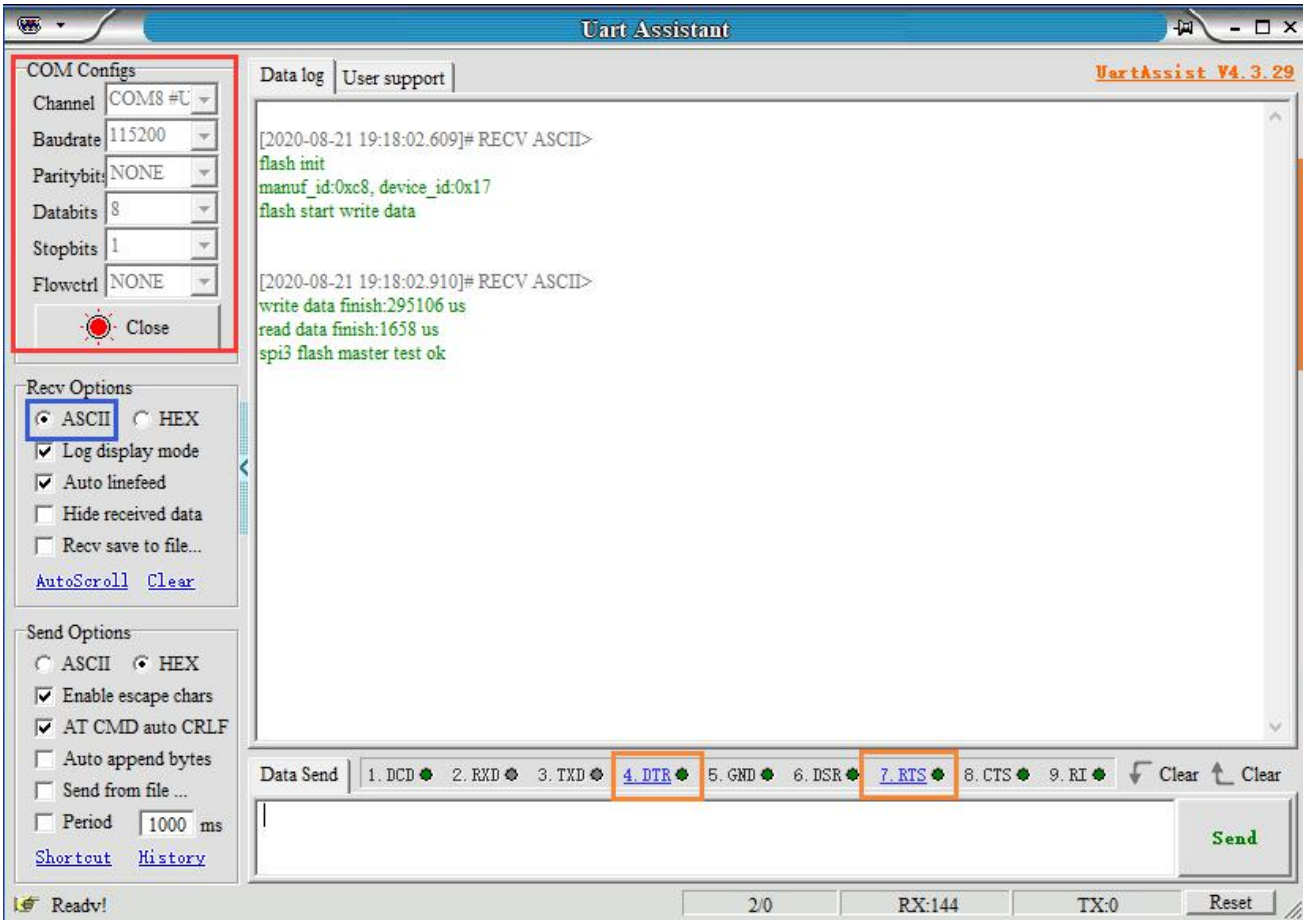
Click 4.DTR and 7.RTS to set them to green.



We can see that the flash initialization and write and read time is printed.

When you see "spi3 flash master test ok" at the end, it means the read and write is successful.

If you see "flash read error", it means Read and write failed.



6. Experiment summary

6.1 A FLASH chip of GD25LQ128C with a storage space of 16MB , it has a total of 4096 sectors, each sector has 16 pages, each page is 256 bytes.

6.2 FLASH is a storage method that can save data after power off.

6.3 The smallest unit of FLASH erasing is sector, which is 4K.

Appendix -- API

Header file is wdt.h

w25qxx_init

Description: Initialize the Flash chip and set the SPI device number, channel number, and rate.

Function prototype: **w25qxx_status_t w25qxx_init(uint8_t spi_index, uint8_t spi_ss, uint32_t rate)**

Parameter:

Parameter name	Description	Input/Output
spi_index	SPI device number	Input
spi_ss	CS chip select	Input
rate	Communication rate	Input

Return value: Status of the flash.

w25qxx_read_id

Description: Read flash ID.
Function prototype: **w25qxx_status_t w25qxx_read_id(uint8_t *manuf_id, uint8_t *device_id)**
Parameter:

Parameter name	Description	Input/Output
manuf_id	The factory ID	Output
device_id	Device ID	Output

Return value: Status of the flash.

w25qxx_write_data
Description: Write data to flash
Function prototype: **w25qxx_status_t w25qxx_write_data(uint32_t addr, uint8_t *data_buf, uint32_t length)**
Parameter:

Parameter name	Description	Input/Output
addr	Flash address	Input
data_buf	Write data	Input
length	Length of data	Input

Return value: Status of the flash.

w25qxx_read_data
Description: Write data form flash
Function prototype: **w25qxx_status_t w25qxx_read_data(uint32_t addr, uint8_t *data_buf, uint32_t length)**
Parameter:

Parameter name	Description	Input/Output
addr	Flash address	Input
data_buf	Read data	Output
length	Length of data	Input

Return value: Status of the flash.

Data type
The related data types and data structure are defined as follows:

- w25qxx_status_t
- w25qxx_read_t

w25qxx_status_t
Description: Status of flash chip
Define

```
typedef enum _w25qxx_status
{
    W25QXX_OK = 0,
    W25QXX_BUSY,
```

```
    W25QXX_ERROR,
} w25qxx_status_t;
```

member

member name	Description
W25QXX_OK	flash OK
W25QXX_BUSY	Flash busy
W25QXX_ERROR	Flash error

w25qxx_read_t

Description: The way the SPI reads Flash.

Define

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

member

member name	Description
W25QXX_STANDARD	standard mode
W25QXX_STANDARD_FAST	standard fast mode
W25QXX_DUAL	dual-line mode
W25QXX_DUAL_FAST	dual-line fast mode
W25QXX_QUAD	4-lines mode
W25QXX_QUAD_FAST	4-lines fast mode