XRADIO Flash Developer Guide

Revision 1.0

Nov 1, 2019

Declaration

THIS DOCUMENTATION IS THE ORIGINAL WORK AND COPYRIGHTED PROPERTY OF XRADIO TECHNOLOGY ("XRADIO"). REPRODUCTION IN WHOLE OR IN PART MUST OBTAIN THE WRITTEN APPROVAL OF XRADIO AND GIVE CLEAR ACKNOWLEDGEMENT TO THE COPYRIGHT OWNER.

THE PURCHASED PRODUCTS, SERVICES AND FEATURES ARE STIPULATED BY THE CONTRACT MADE BETWEEN XRADIO AND THE CUSTOMER. PLEASE READ THE TERMS AND CONDITIONS OF THE CONTRACT AND RELEVANT INSTRUCTIONS CAREFULLY BEFORE USING, AND FOLLOW THE INSTRUCTIONS IN THIS DOCUMENTATION STRICTLY. XRADIO ASSUMES NO RESPONSIBILITY FOR THE CONSEQUENCES OF IMPROPER USE (INCLUDING BUT NOT LIMITED TO OVERVOLTAGE, OVERCLOCK, OR EXCESSIVE TEMPERATURE).

THE INFORMATION FURNISHED BY XRADIO IS PROVIDED JUST AS A REFERENCE OR TYPICAL APPLICATIONS, ALL STATEMENTS, INFORMATION, AND RECOMMENDATIONS IN THIS DOCUMENT DO NOT CONSTITUTE A WARRANTY OF ANY KIND, EXPRESS OR IMPLIED. XRADIO RESERVES THE RIGHT TO MAKE CHANGES IN CIRCUIT DESIGN AND/OR SPECIFICATIONS AT ANY TIME WITHOUT NOTICE.

NOR FOR ANY INFRINGEMENTS OF PATENTS OR OTHER RIGHTS OF THE THIRD PARTIES WHICH MAY RESULT FROM ITS USE. NO LICENSE IS GRANTED BY IMPLICATION OR OTHERWISE UNDER ANY PATENT OR PATENT RIGHTS OF XRADIO.THIRD PARTY LICENCES MAY BE REQUIRED TO IMPLEMENT THE SOLUTION/PRODUCT. CUSTOMERS SHALL BE SOLELY RESPONSIBLE TO OBTAIN ALL APPROPRIATELY REQUIRED THIRD PARTY LICENCES. XRADIO SHALL NOT BE LIABLE FOR ANY LICENCE FEE OR ROYALTY DUE IN RESPECT OF ANY REQUIRED THIRD PARTY LICENCE. XRADIO SHALL HAVE NO WARRANTY, INDEMNITY OR OTHER OBLIGATIONS WITH RESPECT TO MATTERS COVERED UNDER ANY REQUIRED THIRD PARTY LICENCE.

Revision History

Version	Date	Summary of Changes
1.0	2019-11-1	Initial Version

Table 1- 1 Revision History

Contents

Tables

Figures

1 Introduction

1.1 前言

市面上的 SPI Nor Flash 芯片（后文简述为 Flash 芯片）种类繁多。不同厂家不同型号的 flash 芯片在性能方面、命令方面、配置流程方面都会存在差异，而这种差异是通用的 flash 驱动无法很好地支持的。一般在嵌入式 sdk 中 flash 驱动只支持简单操作以及运行在正常的模式如 fast read，针对不同的 flash 需要修改驱动，因此无法做到真正兼容和真正的支持。考虑到上述原因，XR872 sdk 的 flash 驱动采用通信驱动、芯片命令、Flash 控制分离的框架，实现便于扩展，具备高通用性，易于使用的 flash 驱动，以方便使用者进行方案开发，减少开发者工作量。

Write Status Register	01H	(S7-S0)	(S15-S8)

Write Status Register-1	01H	S7-S0	
Write Status Register-2	31H	S15-S8	
Write Status Register-3	11H	S23-S16	

图 1-1 PN25F08 与 GD25Q64C 的写寄存器命令差异

1.2 Flash Datasheet 阅读指引

阅读 flash data sheet 文档时，建议注意包括以下关注点但不限于，

1. Flash 的存储空间大小。

2. Flash 支持的命令类型及格式。

3. Flash 的 Status 寄存器。

4. Flash 配置 WP、HOLD 与 PIN2、PIN3 切换方式。

5. Flash 进入 XIP 的方式。

6. Flash 的读命令（几种模式下）的 dummy 数以及与频率之间的关系（某些芯片存在关系）。

7. Flash 的最高工作频率以及 READ 命令的最高工作频率。

8. Flash 运行高频率的配置（有些芯片运行高频率需要执行一系列操作）。

9. 其他。

1.3 Flash 驱动框架

XR872 sdk 的 flash 驱动采用通信驱动（FlashDrv）、芯片驱动（FlashChip）、Flash 控制（Flash）分离的框架。

通信驱动（FlashDrv）作为抽象接口供其他两个模块使用。当前已实现基于 SPI 的 SpiDriver 和基于 flash controller 的 FlashcDriver。

芯片驱动（FlashChip）作为芯片抽象命令接口供 Flash 控制模块使用，提供如写使能、擦除等接口。根据芯片 data sheet 的内容实现对应接口。对于一些比较简单、命令与 default 一致的芯片，可通过第三章的简单配置实现扩展。

Flash 控制（Flash）模块是基于上述两个模块，提供简便的 flash 操作接口。

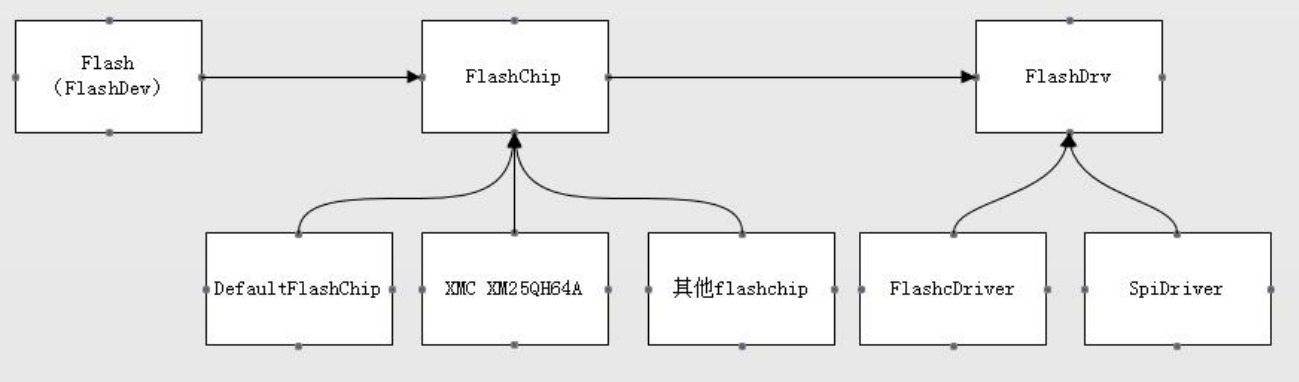

图 1-2 Flash 驱动框架示意图

2 驱动介绍

2.1 Flash 驱动

Flash 驱动主要是为了向用户提供便利的 API 接口，Flash 接口介绍如下表：

表 2-1 Flash 驱动接口

接口	功能
HAL_Status HAL_Flash_Init(uint32_t flash);	功能：初始化 flash 设备 参数 flash：flash 设备号，相当于 g_flash_cfg 结构体的搜引号。 返回值：返回驱动状态值，0 是指初始化 ok，其它表示初始化失败
HAL_Status HAL_Flash_Deinit(uint32_t flash);	功能：反初始化 flash 设备 参数 flash：flash 设备号，相当于 g_flash_cfg 结构体的搜引号 返回值：返回驱动状态值，0 是指反初始化 ok，其它表示反初始化失败
HAL_Status HAL_Flash_Open(uint32_t flash, uint32_t timeout_ms);	功能：开启 flash 设备，拿互斥锁。如果设备已经开启，则其他用户开启不了。 参数：flash：flash 设备号，相当于 g_flash_cfg 结构体的搜引号。 timeout_ms：等待获取 flash 的操作权限的时间，即获取互斥锁时间。 返回值：返回驱动状态值，0 是打开 ok，其他表示打开失败，即有其他用户在用。
HAL_Status HAL_Flash_Close(uint32_t flash)	功能：关闭 flash 设备，释放互斥锁。 参数 flash ：flash 设备号，相当于 g_flash_cfg 结构体的搜引号。 返回值：返回驱动状态值，0 是关闭正常，其他表示关闭失败。
HAL_Status HAL_Flash_Control(uint32_t flash, FlashControlCmd attr, uint32_t arg);	功能：IO 控制功能，目前包括获取最小擦除 size 功能、读写状态寄存器、IO 口状态输出（高，低、高阻）。 参数 flash：flash 设备号，相当于 g_flash_cfg 结构体

	的搜引号。 FlashControlCmd：功能操作行为类型。 arg：实现功能的参数。（64 位参数地址转换会有问题，建议定义成 void *） 返回值：返回驱动状态值，0 是正常，其他表示失败。
HAL_Status HAL_Flash_Overwrite(uint32_t flash, uint32_t addr, uint8_t *data, uint32_t size);	功能：写一段地址内存，不影响其他地址的内容，只有 flash 支持 4K 擦除，此功能才有效。 参数 flash：flash 设备号，相当于 g_flash_cfg 结构体的搜引号。 addr：内存地址。 data：写入的数据地址 size：写入的数据长度 返回值：返回驱动状态值，0 是写入成功，其他表示失败。
HAL_Status HAL_Flash_Write(uint32_t flash, uint32_t addr, const uint8_t *data, uint32_t size);	功能：写一段地址内存，写之前需要先将地址内存擦除，否则写不成功。 参数 flash：flash 设备号，相当于 g_flash_cfg 结构体的搜引号。 addr：内存地址。 data：写入的数据地址 size：写入的数据长度 返回值：返回驱动状态值，0 是写入成功，其他表示失败。
HAL_Status HAL_Flash_Read(uint32_t flash, uint32_t addr, uint8_t *data, uint32_t size);	功能：读一段地址内存。 参数 flash：flash 设备号，相当于 g_flash_cfg 结构体的搜引号。 Addr：内存地址。 data：读到的数据存放地址 size：读数据长度 返回值：返回驱动状态值，0 是读成功，其他表示失败。

HAL_Status HAL_Flash_Init(uint32_t flash);	功能：初始化 flash 设备 参数 flash：flash 设备号，相当于 g_flash_cfg 结构体的搜引号。 返回值：返回驱动状态值，0 是指初始化 ok，其它表示初始化失败

2.2 Flash 芯片驱动

FlashChip 表示该芯片支持的参数、命令和操作等。扩展新的 flash chip 需要实现 FlashChip 结构体内的接口，或复用 default flash chip 接口。

表 2-2 FlashChip 成员

Member	Description
mJedec	flash 的 jedec ID，具体参考第三章
mMaxFreq	除了 READ 命令以外，其他命令（含 FAST READ 等）允许的最高频率
mMaxReadFreq	READ 命令的最高频率
mSize	芯片存储容量
mEraseSizeSupport	flash 芯片支持哪些擦除命令，具体参考第三章
mReadStatusSupport	flash 芯片支持哪些读状态寄存器，具体参考第三章
mWriteStatusSupport	flash 芯片支持哪些写状态寄存器，具体参考第三章
mPageProgramSupport	flash 芯片支持哪些烧写命令，具体参考第三章
mReadSupport	flash 芯片支持哪些读命令，具体参考第三章
mFlashStatus	存储该芯片的状态，初始为 0
mDriver	通信驱动
mXip	XIP 驱动
writeEnable	发送写使能命令
writeDisable	发送写禁止命令
readStatus	读状态寄存器
writeStatus	写状态寄存器

erase	发送擦除命令
suspendErasePageprogram	暂停擦除/写操作
resumeErasePageprogram	恢复擦除/写操作
powerDown	进入掉电模式
releasePowerDown	退出掉电模式
jedecID	获取 JEDEC ID
enableQPIMode	进去 QPI 模式
disableQPIMode	退出 QPI 模式
reset	复位
uniqueID	获得 uniqueID
pageProgram	发送烧写命令（多种模式的写命令）
read	发送读命令（多种模式的读命令）
driverWrite	Chip 与 Driver 对接的驱动接口
driverRead	Chip 与 Driver 对接的驱动接口
xipDriverCfg	配 XIP 读命令格式，用于跑 XIP。
setFreq	设置频率
switchReadMode	切换读模式（Read/Fast Read/.....）
enableXIP	开启 XIP 功能
disableXIP	关闭 XIP 功能
isBusy	Flash 是否在擦除/烧写中
control	Ioctl 函数，用于扩展。
minEraseSize	返回最小的擦除大小

FlashChipCtor 用于实例化并初始化指定 Flash Chip。采用普通方式来扩展 flash chip list，需要实现该 FlashChipCtor。

表 2-3 FlashChipCtor

Member	Description
mJedecId	该 flash 芯片的 JEDEC ID

create	创建该 Flash 芯片的实体
init	初始化 Flash 芯片的实体，替换接口具体实现
destory	删除 Flash 芯片的实体

Flash Chip Creator 会放进 flashChipList[]里面。

```
FlashChipCtor *flashChipList[] = {
        &DefaultFlashChip, /*default chip must be at the last*/
        &MX25QH16B_FlashChip,
        &MX25QH32B_FlashChip,
};
```

3 Flash Chip support

3.1 基本要求

若 flash 芯片的命令与 default 实现一致，并且没有需要扩展的功能，则可通过简易配置进行扩展以稳定支持上该芯片。其中 QPI 模式下是整个指令都是 4 线通信。

表 3-1 Default Flash Chip 命令

Instruction	CMD(IO)	Address Bytes(IO)	Dummy Bytes(IO)	Data Bytes(IO)
Write Enable	0x06(1)	-	-	-
Write Disable	0x04(1)	-	-	-
Volatile SR Write Enable	0x50(1)	-	-	-
Read Status1	0x05(1)	-	-	1(1)
Read Status2	0x35(1)	-	-	1(1)
Read Status3	0x15(1)	-	-	1(1)
Write Status1	0x01(1)	-	-	1(1)
Write Status2	0x31(1)	-	-	1(1)
Write Status3	0x11(1)	-	-	1(1)
Read	0x03(1)	3(1)	-	n(1)
Fast Read	0x0B(1)	3(1)	1(1)	n(1)
Fast Read Dual Output	0x3B(1)	3(1)	1(1)	n(2)
Fast Read Dual IO	0xBB(1)	3(2)	1(2)(含 M0~M7)	n(2)
Fast Read Quad Output	0x6B(1)	3(1)	1(1)	n(4)
Fast Read Quad IO	0xEB(1)	3(4)	3(4)(含 M0~M7)	n(4)
Page Program	0x02(1)	3(1)	-	n(1)
Quad Page Program	0x32(1)	3(1)	-	n(4)
64KB Erase	0xD8(1)	3(1)	-	-
32KB Erase	0x52(1)	3(1)	-	-
4KB Erase	0x20(1)	3(1)	-	-
Chip Erase	0xC7(1)	-	-	-

JEDEC ID	0x9F(1)	-	-	3(1)
Enable Reset	0x66(1)	-	-	-
Reset Device	0x99(1)	-	-	-
Enter QPI Mode	0x38(1)	-	-	-
Set Read Parameters	0xC0(4)	-	-	-
Exit QPI Mode	0xFF(4)	-	-	1(4)

3.2 Flash Chip 支持示例

Flash Chip 支持是指不在默认支持列表里面，用户可以参照此小节添加新的 Flash Chip。

sdk 已经支持的 flash chip 可以通过 include/driver/chip/flashchip/flash_chip_cfg.h 文件查看：

```
#define FLASH_DEFAULTCHIP
#ifndef __CONFIG_BOOTLOADER
#define FLASH_M25P64
#define FLASH_PN25F16B
#define FLASH_W25Q16FW
.......
#define FLASH_XM25QH64A
#define FLASH_GD25Q256D
#endif /* __CONFIG_BOOTLOADER */
```

注意：如果使用的新 flash chip，又没有添加，那么默认是跑 default flash chip 接口，如果 flash chip 的寄存器读写与 default 实现的接口不同，那么四线读模式可能有异常。

3.2.1 Default Flash Chip 支持示例

简易配置通过扩展 simpleFlashChip 数组实现。在数组里增加 SimpleFlashChipCfg 结构体并配置，以支持需要扩展的 flash 芯片参数。

1. mJedec 是 flash 的 jedec ID，24bit 长度：0xX0X1X2X3X4X5，X0X1 是 ID7~ID0，X2X3 是 ID15~ID8，X4X5 是 M7～M0.

2. mSize 是芯片存储容量。

3. mEraseSizeSupport 是 flash 芯片支持哪些擦除命令，一般有 64KByte 擦除（FLASH_ERASE_64KB）、32KByte 擦除（FLASH_ERASE_32KB）、4KByte 擦除（FLASH_ERASE_4KB）和全片擦除（FLASH_ERASE_CHIP）。

4. mReadStausSupport 是 flash 芯片支持哪些读状态寄存器，FLASH_STATUS1，FLASH_STATUS2，FLASH_STATUS3，

5. mWriteStatusSupport 是 flash 芯片支持哪些写状态寄存器，FLASH_STATUS1，FLASH_STATUS2，FLASH_STATUS3，

6. mPageProgramSupport 是 flash 芯片支持哪些烧写命令，FLASH_PAGEPROGRAM ，

FLASH_QUAD_PAGEPROGRAM

7. mReadSupport 是 flash 芯片支持哪些读命令，FLASH_READ_NORMAL_MODE，FLASH_READ_FAST_MODE，FLASH_READ_DUAL_O_MODE ， FLASH_READ_DUAL_IO_MODE ， FLASH_READ_QUAD_O_MODE ，FLASH_READ_QUAD_IO_MODE，FLASH_READ_QPI_MODE，

8. mMaxFreq 是除了 READ 命令以外，其他命令（含 FAST READ 等）允许的最高频率。

9. mMaxReadFreq 是 READ 命令的最高频率。

```
typedef struct SimpleFlashChipCfg
{
    uint32_t mJedec;
    uint32_t mSize;

    uint32_t mEraseSizeSupport;
    uint16_t mReadStausSupport;
    uint8_t  mWriteStatusSupport;
    uint8_t  mPageProgramSupport;
    uint16_t mReadSupport;

    uint32_t mMaxFreq;
    uint32_t mMaxReadFreq;
} SimpleFlashChipCfg;
```

以 XTX 的 PN25F16B 为例，JEDEC ID 如图，因此 mJedec 填 0x15405E。该 flash 为 2MBytes 大小的 flash， mSize 填 32 * 16 * 0x1000（0x200000）。FR 为 100MHz，fR 为 55MHz，因此 mMaxFreq 填 100 * 1000 * 1000，mMaxReadFreq 填 55 * 1000 * 1000。以及从指令集进行擦除、读、写、读寄存器、写寄存器的支持配置。

Table 7.2(1) Manufacturer and Device Identification

OP Code	(M7-M0)	(ID15-ID0)	(ID7-ID0)
ABh	-	-	14h
90h	5E	-	14h
9Fh	5E	40h	15h

图 3-1PN25F16B JEDEC ID

Table 7.1(1) Instruction Set

INSTRUCTION NAME	BYTE1 (CODE)	BYTE2	BYTE3	BYTE4	BYTE5	BYTE6	N-BYTES
Write Enable	06h						
write Disable	04h						
Read Status Register	05h	(S7-S0)(2)					
Write Status Register	01h	(S7-S0)(2)					
Read Data	03h	A23-A16	A15-A8	A7-A0	(D7-D0)	(Next byte)	continuous
Fast Read	0Bh	A23-A16	A15-A8	A7-A0	dummy	(D7-D0)	(Next byte) continuous
Fast Read Dual Output	3Bh	A23-A16	A15-A8	A7-A0	dummy	DIO= (D6,D4,D2,D0) DO= (D7,D5,D3,D1)	(One byte per 4 clocks, continuous)
Page Program	02h	A23-A16	A15-A8	A7-A0	(D7-D0)	(Next byte)	Up to 256 bytes
Block Erase(64KB)	D8h	A23-A16	A15-A8	A7-A0			
Half Block Erase(32KB)	52h	A23-A16	A15-A8	A7-A0			
Sector Erase(4KB)	20h	A23-A16	A15-A8	A7-A0			
Chip Erase	C7h/60h						
Power-down	B9h						
Release Power-down /Device ID	ABh	dummy	dummy	dummy	(ID7-ID0)(4)		
Manufacturer /Device ID(3)	90h	dummy	dummy	00h	(M7-M0)	(ID7-ID0)	
JEDEC ID	9Fh	(M7-M0) Manufacturer	(ID15-ID8) Memory Type	(ID7-ID0) Capacity			

图 3-2 PN25F16B 指令集

具体配置：

```
static const SimpleFlashChipCfg simpleFlashChip[] =
{
/* ..... */
#ifdef FLASH_PN25F16B
    {
        /* FLASH_PN25F16B */
        .mJedec = 0x15405E,
        .mSize = 32 * 16 * 0x1000,
        .mEraseSizeSupport = FLASH_ERASE_64KB | FLASH_ERASE_32KB | FLASH_ERASE_4KB
| FLASH_ERASE_CHIP,
        .mPageProgramSupport = FLASH_PAGEPROGRAM,
        .mReadStausSupport = FLASH_STATUS1,
        .mWriteStatusSupport = FLASH_STATUS1,
        .mReadSupport   =   FLASH_READ_NORMAL_MODE   |   FLASH_READ_FAST_MODE   |
FLASH_READ_DUAL_O_MODE,
        .mMaxFreq = 100 * 1000 * 1000,
        .mMaxReadFreq = 55 * 1000 * 1000,
    },
#endif
/* ..... */
}
```

3.2.2 非 default flash chip 支持示例

这里以 P25Q16H 为例，通过 data sheet 了解到这款芯片的行为与我们的 default 接口基本一致，但是 status 寄存器的写命令与 default 接口实现有差别，于是我们重新实现了 writeStatus。

Status Register						
Read Status Register	RDSR	05h	0	0	1	read out status register
	RDSR2	35h	0	0	1	Read out status register-1
Read Configure Register	RDCR	15h	0	0	1	Read out configure register
Active Status Interrupt	ASI	25h	0	1	0	Enable the active status interrupt
Write Status Register	WRSR	01h	0	0	2	Write data to status registers
Write Configure Register	WRCR	31h	0	0	1	Write data to configuration register

图 3-3 P25Q16H 的读写寄存器命令

与表 3-1 Default Flash Chip 命令对比，P25Q16H 的写寄存器命令完全不同。

首先，我们需要创建一个文件 flash_P25QXXH.c 文件，通过 flash chip jedec 值确认使用芯片型号，然后具体配置特定参数（如 flash 的存储大小、支持的读写操作等），就可以实现代码复用。

```
#define P25Q16H_JEDEC 0x156085
.....
FlashChipCtor  P25Q16H_FlashChip = {
        .mJedecId = P25Q16H_JEDEC,
        .enumerate = P25QXXH_FlashCtor,
        .init = P25QXXH_FlashInit,
        .destory = P25QXXH_FlashDeinit,
};
```

```
static struct FlashChip *P25QXXH_FlashCtor(struct FlashChip *chip, uint32_t arg)
{
    uint32_t jedec = arg;
    uint32_t size;
    PCHECK(chip);

    if (jedec == P25Q80H_JEDEC) {
        size = 16 * 16 * 0x1000;
    } else if (jedec == P25Q40H_JEDEC) {
        size = 16 * 8 * 0x1000;
    } else if (jedec == P25Q16H_JEDEC) {
        size = 32 * 16 * 0x1000;
    }
    else {
        return NULL;
    }

    /* TODO: use HAL_Memcpy() and const to init chip->cfg to save code size */
    chip->cfg.mJedec = jedec;
    chip->cfg.mSize = size;
    chip->cfg.mMaxFreq = 104 * 1000 * 1000;
    chip->cfg.mMaxReadFreq = 55 * 1000 * 1000;
```

```
    chip->cfg.mEraseSizeSupport   =   FLASH_ERASE_64KB   |   FLASH_ERASE_32KB   |
FLASH_ERASE_4KB | FLASH_ERASE_CHIP;
    chip->cfg.mPageProgramSupport = FLASH_PAGEPROGRAM | FLASH_QUAD_PAGEPROGRAM;
    chip->cfg.mReadStausSupport = FLASH_STATUS1 | FLASH_STATUS2 | FLASH_STATUS3;
    chip->cfg.mWriteStatusSupport    =    FLASH_STATUS1    |    FLASH_STATUS2    |
FLASH_STATUS3;//
    chip->cfg.mReadSupport  =  FLASH_READ_NORMAL_MODE  |  FLASH_READ_FAST_MODE  |
FLASH_READ_DUAL_O_MODE  |  FLASH_READ_DUAL_IO_MODE  |  FLASH_READ_QUAD_O_MODE  |
FLASH_READ_QUAD_IO_MODE;
    chip->mPageSize = 256;
    chip->mFlashStatus = 0;
    chip->mDummyCount = 1;

    return chip;
}
```

写寄存器函数重新封装：

```
static int P25QXXH_WriteStatus(struct FlashChip *chip, FlashStatus reg, uint8_t
*status)
{
    int ret;
    uint8_t status_buf[2];
    InstructionField instruction[2];

    PCHECK(chip);

    if (!(reg & chip->cfg.mWriteStatusSupport)) {
        FLASH_NOTSUPPORT();
        return HAL_INVALID;
    }
/*
    HAL_Memset(&instruction, 0, sizeof(instruction));
    FCI_CMD(0).data = FLASH_INSTRUCTION_SRWREN;
    FCI_CMD(0).line = 1;
    chip->driverWrite(chip, &FCI_CMD(0), NULL, NULL, NULL);
*/

    HAL_Memset(&instruction, 0, sizeof(instruction));

    if (reg == FLASH_STATUS1)
    {
        FCI_CMD(0).data = FLASH_INSTRUCTION_RDSR2;
        FCI_CMD(0).line = 1;

        FCI_DATA(1).pdata = (uint8_t *)&status_buf[1];
        FCI_DATA(1).len = 1;
        FCI_DATA(1).line = 1;

        chip->driverRead(chip, &FCI_CMD(0), NULL, NULL, &FCI_DATA(1));

        status_buf[0] = *status;

        FCI_CMD(0).data = FLASH_INSTRUCTION_WRSR;

        FCI_DATA(1).pdata = status_buf;
```

```
        FCI_DATA(1).len = 2;
        FCI_DATA(1).line = 1;

        //printf("FLASH_STATUS1\n");
        //printf("SR1:0x%02x\n", status_buf[0]);
        //printf("SR2:0x%02x\n", status_buf[1]);
    }
    else if (reg == FLASH_STATUS2)
    {
        FCI_CMD(0).data = FLASH_INSTRUCTION_RDSR;
        FCI_CMD(0).line = 1;

        FCI_DATA(1).pdata = (uint8_t *)status_buf;
        FCI_DATA(1).len = 1;
        FCI_DATA(1).line = 1;

        chip->driverRead(chip, &FCI_CMD(0), NULL, NULL, &FCI_DATA(1));

        status_buf[1] = *status;

        FCI_CMD(0).data = FLASH_INSTRUCTION_WRSR;

        FCI_DATA(1).pdata = status_buf;
        FCI_DATA(1).len = 2;
        FCI_DATA(1).line = 1;

        //printf("FLASH_STATUS2\n");
        //printf("SR1:0x%02x\n", status_buf[0]);
        //printf("SR2:0x%02x\n", status_buf[1]);
    }
    else if (reg == FLASH_STATUS3)
    {
        FCI_CMD(0).data = FLASH_INSTRUCTION_WRCR;

        FCI_DATA(1).pdata = (uint8_t *)status;
        FCI_DATA(1).len = 1;
        FCI_DATA(1).line = 1;
    }
    else
    {
        FLASH_NOWAY();
    }

    chip->writeEnable(chip);

    ret = chip->driverWrite(chip, &FCI_CMD(0), NULL, NULL, &FCI_DATA(1));

    chip->writeDisable(chip);
/*
    while (chip->isBusy(chip)) {
        //printf("busy...\n");
    }
*/
    return ret;
}
```

添加 writestatus 函数的挂载以及其他函数复用 default chip 的接口：

```
static int P25QXXH_FlashInit(struct FlashChip *chip)
{
    PCHECK(chip);

    chip->writeEnable = defaultWriteEnable;
    chip->writeDisable = defaultWriteDisable;
    chip->readStatus = defaultReadStatus;
    chip->erase = defaultErase;
    chip->jedecID = defaultGetJedecID;
    chip->pageProgram = defaultPageProgram;
    chip->read = defaultRead;

    chip->driverWrite = defaultDriverWrite;
    chip->driverRead = defaultDriverRead;
    chip->xipDriverCfg = defaultXipDriverCfg;
    chip->setFreq = defaultSetFreq;
    chip->switchReadMode = defaultSwitchReadMode;
    chip->enableXIP = defaultEnableXIP;
    chip->disableXIP = defaultDisableXIP;
    chip->isBusy = defaultIsBusy;
    chip->control = defaultControl;
    chip->minEraseSize = defaultGetMinEraseSize;
    //chip->writeStatus = defaultWriteStatus;
    chip->writeStatus = P25QXXH_WriteStatus;
    chip->enableQPIMode = defaultEnableQPIMode;
    chip->disableQPIMode = defaultDisableQPIMode;
//  chip->enableReset = defaultEnableReset;
    chip->reset = defaultReset;

    chip->suspendErasePageprogram = NULL;
    chip->resumeErasePageprogram = NULL;
    chip->powerDown = NULL;
    chip->releasePowerDown = NULL;
    chip->uniqueID = NULL;
    /*TODO: a NULL interface for showing invalid interface*/

    FLASH_DEBUG("P25QXXH_Flash inited");

    return 0;
}
```

最后，在 flash_chip.c 的 flashChipList 里补充 P25Q16H_FlashChip 就完成扩展。

```
FlashChipCtor *flashChipList[] = {
#ifdef FLASH_DEFAULTCHIP
    &DefaultFlashChip, /*default chip must be at the first*/
#endif
...
#ifdef FLASH_P25Q16H
    &P25Q16H_FlashChip,
#endif
#ifdef FLASH_EN25QH64A
    &EN25QH64A_FlashChip,
```

```
#endif
#ifdef FLASH_XM25QH64A
    &XM25QH64A_FlashChip,
#endif
};
```