XR806 Flash 驱动

开发指南

版本号：1.0

发布时间：2020-11-03

版本历史

版本	日期	责任人	版本描述
1.0	2020-11-03	AWA 1680	创建文档。

目录

图片目录

表格目录

1 前言

1.1 文档简介

本文档主要介绍了 XR806 SDK 的 Flash 模块概述、驱动框架、Flash 驱动接口、使用示例、新 Flash 型号的支持等内容，方便用户对 SDK 的 Flash 模块进行二次开发和快速扩展。

1.2 目标读者

XR806 SDK 用户、Flash 开发使用人员。

1.3 适用范围

此文档适用于 XR806 SDK，支持 XR806 系列芯片产品。

1.4 文档约定

1.4.1 标志说明

本文档采用各种醒目的标志来表示在操作过程中应该特别注意的地方，这些标志的含义如下：

标识	说明
警告	该标志后的说明应给予格外关注，如果不遵守，可能会导致人员受伤或死亡。
注意	提醒操作中应注意的事项。不当的操作可能会损坏器件，影响可靠性、降低性能等。
说明	为准确理解文中指令、正确实施操作而提供的补充或强调信息。
窍门	一些容易忽视的小功能、技巧。了解这些功能或技巧能帮助解决特定问题或者节省操作时间。

1.4.2 地址与数据描述方法约定

本文档在描述地址、数据时遵循如下约定：

符号	例子	说明
0x	0x0200，0x79	地址或数据以 16 进制表示。
0b	0b010，0b00 000 111	数据采用二进制表示(寄存器描述除外)。
X	00X，XX1	数据描述中，X 代表 0 或 1。 例如，00X 代表 000 或 001；XX1 代表 001，011，101 或 111。

1.4.3 数值单位约定

本文档在描述数据容量（如 NAND 容量）时，单位词头代表的是 1024 的倍数；描述频率、数据速率等时则代表的是 1000 的倍数。具体如下：

类型	符号	对应数值
数据容量（如 NAND 容量）	1 K	1024
	1 M	1 048 576
	1 G	1 073 741 824
频率，数据速率等	1 k	1000
	1 M	1 000 000
	1 G	1 000 000 000

2 概述

2.1 背景说明

XR806 的 Flash 模块主要采用 SPI NOR Flash 芯片（后文简述为 Flash 芯片），由于它支持 XIP（eXecute In Place），即代码可直接在此 Flash 上执行，无需复制到内存中，因此具有较高的性价比。

Flash 存储器件由块（Block）组成，每个块中包含了若干个扇区（Sector），每个扇区中包含了若干页（Page），对 Flash 进行擦除时，只能按照其可擦除最小单位（Sector/Block）来擦除。由于 Flash 芯片内的数据只能由 1 写为 0，不能由 0 写为 1，只有擦除时才能将数据变为 1，因此当要写 Flash 的某一块区域时，需要确保这块区域已经被擦除。

目前，市面上的 Flash 芯片种类繁多，具有一定差异性。首先，不同厂家不同型号的 Flash 芯片在性能、命令、配置流程等方面都可能存在差异，而这种差异在通用的 Flash 驱动上无法很好地支持。其次，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，XR806 SDK 的 Flash 驱动采用通信驱动、芯片命令、Flash 控制分离的框架（可参见“3.1 驱动框架”章节），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。

2.2 规格特性

XR806 芯片中内置一个 Flash 控制器（FlashController，以下简称 Flashc），以方便 CPU 对 Flash 进行操作，因此 XR806 芯片挂接的 Flash 既可以走 SPI 通信方式，也可以走 Flashc 通信方式（可参见图 3-2），XR806 的 Flashc 模块和 SPI 硬件模块的规格特性如下表所示。

表 2-1 XR806 Flashc 硬件模块规格特性表

硬件规格	规格描述	备注
支持通信模式	SPI 1/2/4 线模式收发、SPI Mode0/1/2/3、Flash XIP	
支持 Flash 容量	Flash 容量最大 2^32 Byte（XIP 空间小于 16M）	
时钟频率	频率可配，可支持最大 120MHz	

表 2-2 XR806 SPI 硬件模块规格特性表

硬件规格	规格描述	备注
支持频率	最高 96MHz	主机模式
数据缓冲	最大 64 字节，以 word 对齐	
支持 Flash 容量	Flash 容量最大 2^32 Byte	不支持 Flash XIP

XR806 Flash 驱动模块的规格特性如下表所示。

表 2-3 XR806 Flash 驱动模块规格特性表

驱动规格	规格描述	备注
支持 SPI 接口 Flash 读模式	Normal Read、 Fast Read、 Fast Read Dual Output	
支持 Flashc 接口 Flash 读模式	Normal Read、Fast Read、Fast Read Dual Output、Fast Read Dual I/O、 Fast Read Quad Output、 Fast Read Quad I/O	
支持擦除模式	4K、32K、64K、Chip	与实际芯片匹配
支持写模式	Page Program	

2.3 文件位置

以 SDK 包为根目录，本驱动涉及到的主要文件位置如下。

表 2-4 Flash 驱动的文件位置

组件名	文件分类	文件位置
Flash	源码文件	./src/driver/chip/hal_flash.c
	头文件	./include/driver/chip/hal_flash.h
	示例工程	./project/example/flash/

说明

由于 Flash 驱动采用 Flash 控制、芯片命令、通信驱动分离的框架，除用户使用的 Flash 控制层外，其芯片命令层、通信驱动层的源文件代码也可供用户参考查阅，如下表所示。

表 2-5 FlashChip 文件位置

组件名	文件分类	文件位置
Flash	源码文件	./src/driver/chip/flashchip/
	头文件	./include/driver/chip/flashchip/

表 2-6 Flashctrl 驱动文件位置

组件名	文件分类	文件位置
Flash	源码文件	./src/driver/chip/hal_flashctrl.c
	头文件	./include/driver/chip/hal_flashctrl.h

说明

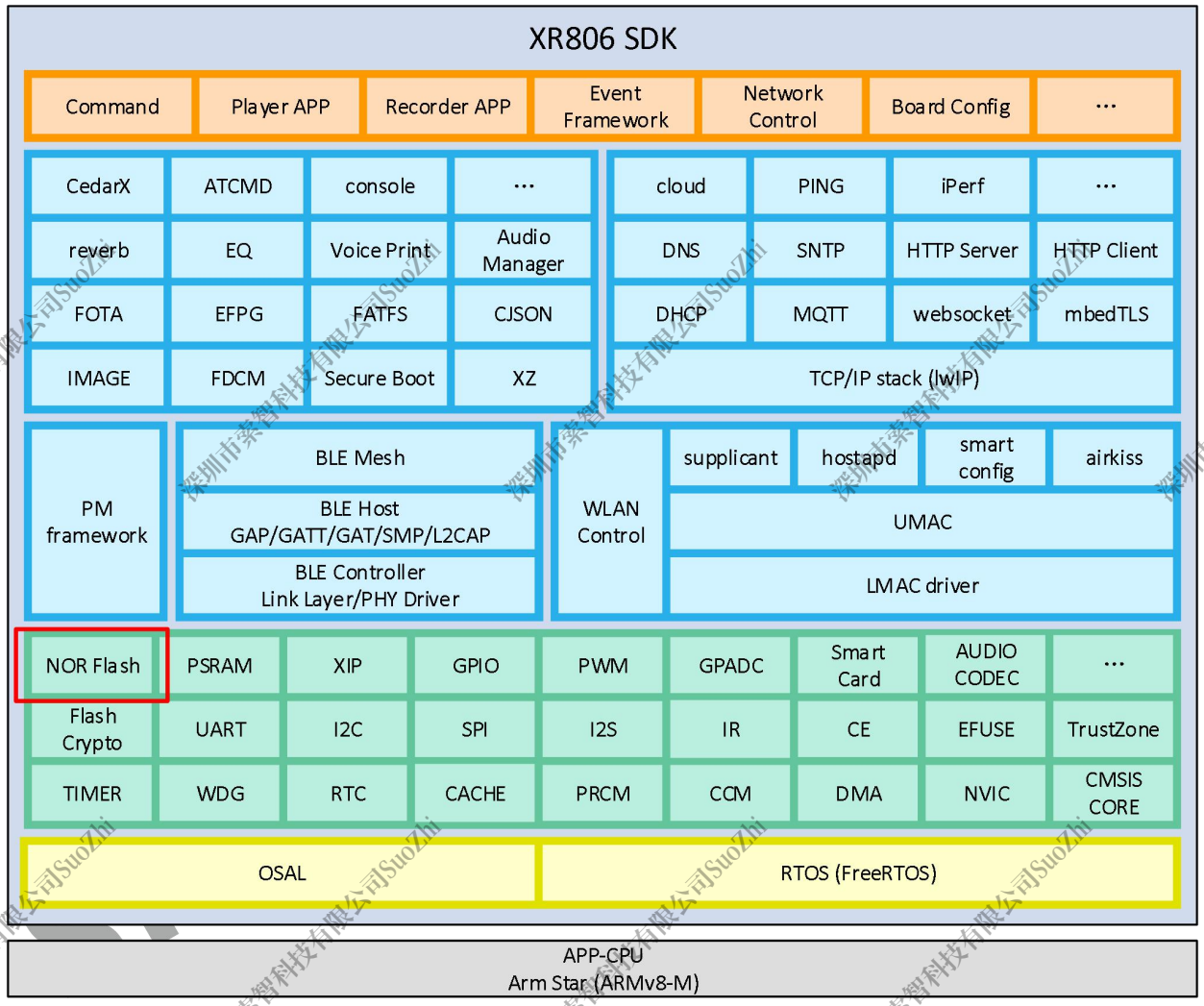
SpiDriver 通信层的驱动代码包含在./src/driver/chip/hal_flash.c 中。

XR806 SDK 可在以下 GitHub 仓库获取：https://github.com/XradioTech/xr806_sdk.git

3 技术说明

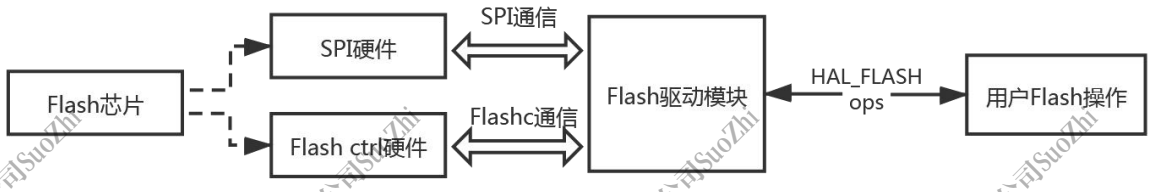
XR806 Flash 驱动模块位于 SDK 的 NOR Flash 部分，具体位置如下图所示。

图 3-1 XR806 SDK 框图



Flash 芯片在物理上既可与 SPI 硬件接口连接，也可与 Flash controller 硬件连接，在 XR806 SDK 中，可选择/配置 Flash 驱动的通信方式（参见“4.2 配置说明”章节），以供用户灵活使用，XR806 Flash 驱动模块的接口框图如下所示。

图 3-2 XR806 Flash 接口框图



3.1 驱动框架

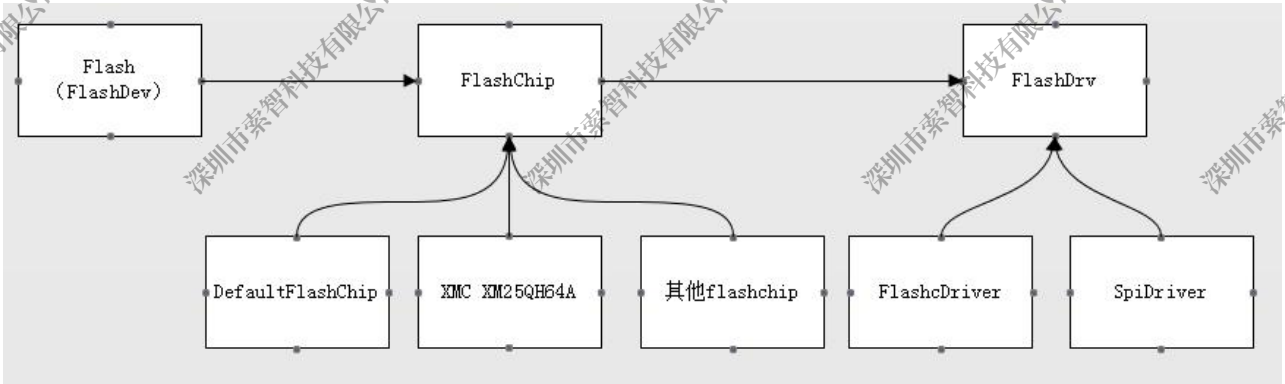
XR806 SDK 的 Flash 驱动采用通信驱动（FlashDrv）、芯片驱动（FlashChip）、Flash 控制（Flash）分离的框架。

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 3 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（见“4.3 接口说明”章节）。

图 3-3 Flash 驱动框架示意图



4 应用说明

4.1 应用简述

Flash 模块的使用流程如下：

1. 确认/检查所使用 XR806 芯片版本是否需要外部 Flash 芯片（XR806 不同的芯片版本型号对应的基本规格可参考《XR806_Product_Brief》文档），确认/检查 Flash 连接至 XR806 芯片的引脚，可参见“7 常见用户方案”章节。
2. 确认/检查所使用的 Flash 芯片是否在支持列表（参见《XR806 SPI Nor Flash Support List》文档），如无则进行 Flash Chip 对接流程，参见“6FlashChip 支持”章节。
3. 确认/检查 Flash 板级参数配置是否正确，参见“4.2 配置说明”章节。
4. 初始化 platform_init()，使用 Flash 接口，完成用户所需操作，参见“4.2 接口说明”章节。

4.2 配置说明

XR806 Flash 驱动模块，在 board_config.c 中可以选择 Flash 驱动的通信方式（Flashc 或 SPI），board_config.c 位于目录 SDK/project/common/board 的子目录下。

以 xr806_dig_ver 型号开发板为例，board_config.c 位于 SDK/project/common/board/xr806_dig_ver 目录。

说明

board_config.c 中为对应型号开发板的默认板级配置，用户可根据需求对配置内容进行修改和变更。

Flash 驱动的通信方式配置步骤如下（默认的 image Flash 选用 Flashc 进行通信）：

1. 若 XR806 芯片为非 SiP Flash 型号，则在 g_pinmux_flashc[]数组中配置 Flash Controller 引脚（采用 SWD 进行 Debug 则无法跑 4 线的模式）。
2. 若为 SiP Flash 型号，则用户无需配置其 pinmux（SDK 在 board_common.c 中已经配置好）。
3. 引脚确定与配置后，再配置 g_flash_cfg[]来选择 Flash 驱动的通信方式。

表 4-1 FlashBoardCfg 结构体成员表

成员	简要说明
type	Flash 通信驱动类型，有 FLASH_DRV_FLASHC 和 FLASH_DRV_SPI
mode	Flash 读模式
clk	Flash 芯片工作频率
flashc 或 spi(选其一)	Flashc 硬件配置/SPI 硬件配置

说明

表 4-1 中 Flashc 硬件配置/SPI 硬件配置为 SDK 平台初始化中的 private 配置，一般不需修改，用户只需要在 board_config.c 中确定 type、mode、clk 即可。

示例代码如下（非 SiP Flash 型号）：

```
static const GPIO_PinMuxParam g_pinmux_flashc[] = {
    { GPIO_PORT_B, GPIO_PIN_4,  { GPIOB_P4_F5_FLASH_MOSI, GPIO_DRIVING_LEVEL_3, GPIO_PULL_UP } },
    { GPIO_PORT_B, GPIO_PIN_5,  { GPIOB_P5_F5_FLASH_MISO, GPIO_DRIVING_LEVEL_3, GPIO_PULL_UP } },
    { GPIO_PORT_B, GPIO_PIN_6,  { GPIOB_P6_F5_FLASH_CS0,  GPIO_DRIVING_LEVEL_3, GPIO_PULL_UP } },
    { GPIO_PORT_B, GPIO_PIN_7,  { GPIOB_P7_F5_FLASH_CLK,  GPIO_DRIVING_LEVEL_3, GPIO_PULL_UP } },
#if (!BOARD_SWD_EN)
    { GPIO_PORT_B, GPIO_PIN_2,  { GPIOB_P2_F5_FLASH_WP,   GPIO_DRIVING_LEVEL_3, GPIO_PULL_UP } },
    { GPIO_PORT_B, GPIO_PIN_3,  { GPIOB_P3_F5_FLASH_HOLD, GPIO_DRIVING_LEVEL_3, GPIO_PULL_UP } },
#endif
};

/* flash */
static const FlashBoardCfg g_flash_cfg[] = {
    {/* for flashc */
        .type = FLASH_DRV_FLASHC,
        .mode = FLASH_READ_DUAL_O_MODE,
        .clk = (48 * 1000 * 1000),
    },
    ...
};
```

在采用 Flashc 为通信方式时，可以在 SDK 编译前进行配置，选择开启 XIP 功能。

表 4-2 XR806 Flash 驱动模块配置列表

配置项	配置说明
Flash 的 XIP 功能使能	设置说明： 此项配置用于在 SDK 中启用 XIP 功能，即在 platform init 阶段进行 XIP 功能的初始化。 设置位置： make menuconfig 下的 Project settings 选项。 设置方式：

配置项	配置说明
	按‘Y’选中 XIP 功能，即可打开 XIP 功能，同时可选择使用 cache 的大小。
端口设置	设置说明： Flash 驱动模块支持使用多个 Flash，需要在调用接口时传入所使用 Flash 设备的次设备号（Flash 主设备号均为 64）。 次设备号用于区分不同的 Flash 设备，不同的 Flash 设备，其选用 Flash 驱动通信方式也不同（与芯片资源有关，XR806 有一个 Flashc 和一个 SPI）。若次设备号为 0，对应的通信方式可在 g_flash_cfg 数组的第 0 个元素中查看，若次设备号为 n，对应的通信方式可在 g_flash_cfg 数组的第 n 个元素中查看。 建议与默认情况： 在 SDK 中次设备号 0 表示 image Flash，对应的通信方式为 Flashc。 若用户另增一个 Flash，则创建设备时，建议 Flash 次设备号设置为 1，对应的通信方式配置为 SPI（g_flash_cfg 数组的第 1 个元素）。 涉及接口： HAL_Flash_Init、HAL_Flash_Deinit、HAL_Flash_Open、HAL_Flash_Write 等 设置方式：详细见相关接口说明。

4.3 接口说明

XR806 Flash 驱动为用户提供 1 套对 Flash 操作的通用接口，位于 SDK/include/driver/chip/hal_flash.h 文件中，各接口的简要说明如下表所示。

接口名	简要说明
配置接口	包括 2 个接口函数，用于 Flash 设备的初始化、反初始化。
基本开关接口	包括 2 个接口函数，用于用户对 Flash 设备的互斥使用。
基本读写擦接口	包括 3 个接口函数，用于 Flash 的基本读、写、擦除功能。
特殊功能接口	包括 4 个接口函数，用于 Flash 的覆盖写、IO 控制等特定功能。

上述抽象接口均于对应的 hal_flash.c 文件中实现，XR806 Flash 驱动的接口详细说明如下。

4.3.1 配置接口

4.3.1.1 HAL_Flash_Init

表 4-3 HAL_Flash_Init 接口函数说明

信息项	说明
原型	HAL_Status HAL_Flash_Init(uint32_t flash, FlashBoardCfg *param);
功能	初始化 Flash 设备

信息项	说明
参数	uint32_t flash 含义解释：Flash 次设备号，相当于 g_flash_cfg 结构体的搜引号。 使用说明：若选用 image Flash，次设备号为 0，若选用其他 Flash，次设备号为用户配置。 FlashBoardCfg *param 含义解释：对应 Flash 设备的板级配置 使用说明：从 g_flash_cfg 结构体中获取，与参数 flash 匹配使用。
返回值	HAL_OK：成功 HAL_ERROR：一般错误 HAL_BUSY：设备忙 HAL_TIMEOUT：超时 HAL_INVALID：无效参数

4.3.1.2 HAL_Flash_Deinit

表 4-4 HAL_Flash_Deinit 接口函数说明

信息项	说明
原型	HAL_Status HAL_Flash_Deinit(uint32_t flash);
功能	反初始化 Flash 设备
参数	uint32_t flash 含义解释：Flash 次设备号，相当于 g_flash_cfg 结构体的搜引号。 使用说明：若选用 image Flash，次设备号为 0，若选用其他 Flash，次设备号为用户配置。
返回值	HAL_OK：成功 HAL_ERROR：一般错误 HAL_BUSY：设备忙 HAL_TIMEOUT：超时 HAL_INVALID：无效参数

4.3.2 基本开关接口

4.3.2.1 HAL_Flash_Open

表 4-5 HAL_Flash_Open 接口函数说明

信息项	说明
原型	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，则其他用户无法开启。
参数	uint32_t flash

信息项	说明
	含义解释：Flash 次设备号，相当于 g_flash_cfg 结构体的搜引号。 使用说明：若选用 image Flash，次设备号为 0，若选用其他 Flash，次设备号为用户配置。 uint32_t timeout_ms 含义解释：等待获取 Flash 的操作权限的时间，即获取互斥锁的等待超时时间，单位是毫秒。 使用说明：N/A
返回值	HAL_OK：成功 HAL_ERROR：一般错误 HAL_BUSY：设备忙 HAL_TIMEOUT：超时 HAL_INVALID：无效参数

4.3.2.2 HAL_Flash_Close

表 4-6 HAL_Flash_Close 接口函数说明

信息项	说明
原型	HAL_Status HAL_Flash_Close(uint32_t flash);
功能	关闭 Flash 设备，释放互斥锁。
参数	uint32_t flash 含义解释：Flash 次设备号，相当于 g_flash_cfg 结构体的搜引号。 使用说明：若选用 image Flash，次设备号为 0，若选用其他 Flash，次设备号为用户配置。
返回值	HAL_OK：成功 HAL_ERROR：一般错误 HAL_BUSY：设备忙 HAL_TIMEOUT：超时 HAL_INVALID：无效参数

4.3.3 基本读写擦接口

4.3.3.1 HAL_Flash_Read

表 4-7 HAL_Flash_Read 接口函数说明

信息项	说明
原型	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

信息项	说明
参数	uint32_t flash 含义解释：Flash 次设备号，相当于 g_flash_cfg 结构体的搜引号。 使用说明：若选用 image Flash，次设备号为 0，若选用其他 Flash，次设备号为用户配置。 uint32_t addr 含义解释：需要读出的 Flash 内部物理地址 使用说明：N/A uint8_t *data 含义解释：读到的数据存放地址 使用说明：N/A uint32_t size 含义解释：读数据长度 使用说明：N/A
返回值	HAL_OK：成功 HAL_ERROR：一般错误 HAL_BUSY：设备忙 HAL_TIMEOUT：超时 HAL_INVALID：无效参数

4.3.3.2 HAL_Flash_Erase

表 4-8 HAL_Flash_Erase 接口函数说明

信息项	说明
原型	HAL_Status HAL_Flash_Erase(uint32_t flash, FlashEraseMode blk_size, uint32_t addr, uint32_t blk_cnt);
功能	擦除一段地址内存，只能按扇区或块大小来擦除，或者擦除整片 Flash。擦除地址需与擦除模式大小对齐，擦除地址可使用 HAL_Flash_MemoryOf 接口（参见“4.3.4.3HAL_Flash_MemoryOf”章节）计算获得。
参数	uint32_t flash 含义解释：Flash 次设备号，相当于 g_flash_cfg 结构体的搜引号。 使用说明：若选用 image Flash，次设备号为 0，若选用其他 Flash，次设备号为用户配置。 FlashEraseMode blk_size 含义解释：擦除大小模式 使用说明：支持情况与具体 Flash 芯片有关(4K/32K/64K/Chip) uint32_t addr 含义解释：需要擦除的 Flash 内部物理地址 使用说明：N/A

信息项	说明
	uint32_t blk_cnt 含义解释：需要擦除的扇区或块的个数 使用说明：N/A
返回值	HAL_OK：成功 HAL_ERROR：一般错误 HAL_BUSY：设备忙 HAL_TIMEOUT：超时 HAL_INVALID：无效参数

4.3.3.3 HAL_Flash_Write

表 4-9 HAL_Flash_Write 接口函数说明

信息项	说明
原型	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，写之前需要用户自己保证被写区域已被擦除，否则写入后的数据会与预期不符。（Flash 只能由 1 写成 0，不能从 0 写成 1，所以写之前需要保证被写区域为全 1，擦除操作可以将被写区域置为全 1）。 可使用 HAL_Flash_Check 接口（参见“4.3.4.1HAL_Flash_Check”章节）判断被写区域是否需要事先擦除。
参数	uint32_t flash 含义解释：Flash 次设备号，相当于 g_flash_cfg 结构体的搜引号。 使用说明：若选用 image Flash，次设备号为 0，若选用其他 Flash，次设备号为用户配置。 uint32_t addr 含义解释：需要写入的 Flash 内部物理地址 使用说明：N/A const uint8_t *data 含义解释：被写入数据的地址 使用说明：N/A uint32_t size 含义解释：被写入数据的长度 使用说明：N/A
返回值	HAL_OK：成功 HAL_ERROR：一般错误 HAL_BUSY：设备忙 HAL_TIMEOUT：超时 HAL_INVALID：无效参数

4.3.4 特殊功能接口

4.3.4.1 HAL_Flash_Check

表 4-10 HAL_Flash_Check 接口函数说明

信息项	说明
原型	int HAL_Flash_Check(uint32_t flash, uint32_t addr, uint8_t *data, uint32_t size);
功能	写 Flash 之前检查，判断被写入区域是否需要事先擦除。 （若被写入区域的当前数据与需要写入的数据相同，则不需要执行写操作； 若被写入区域的当前数据为全 1，则可以判断此区域不需要被擦除，可直接写； 若被写入区域的当前数据既不是需要写入的数据，也不是全 1，则此区域需要先擦除，再执行写操作）
参数	uint32_t flash 含义解释：Flash 次设备号，相当于 g_flash_cfg 结构体的搜引号。 使用说明：若选用 image Flash，次设备号为 0，若选用其他 Flash，次设备号为用户配置。 uint32_t addr 含义解释：需要写入的 Flash 内部物理地址 使用说明：N/A uint8_t *data 含义解释：被写入数据的地址 使用说明：N/A uint32_t size 含义解释：被写入数据的长度 使用说明：N/A
返回值	-1：检查失败 0：数据相同，不需要擦写 1：可直接写，不需要擦除 2：需要进行擦除

4.3.4.2 HAL_Flash_Overwrite

表 4-11 HAL_Flash_Overwrite 接口函数说明

信息项	说明
原型	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，用户不需要关心被写区域是否已被擦除，直接写即可，被写区域前后的地址内容也不受影响。（注：只有 Flash 支持 4K 擦除模式，此功能才有效）

信息项	说明
参数	uint32_t flash 含义解释：Flash 次设备号，相当于 g_flash_cfg 结构体的搜引号。 使用说明：若选用 image Flash，次设备号为 0，若选用其他 Flash，次设备号为用户配置。 uint32_t addr 含义解释：需要写入的 Flash 内部物理地址 使用说明：N/A uint8_t *data 含义解释：被写入数据的地址 使用说明：N/A uint32_t size 含义解释：被写入数据的长度 使用说明：N/A
返回值	HAL_OK：成功 HAL_ERROR：一般错误 HAL_BUSY：设备忙 HAL_TIMEOUT：超时 HAL_INVALID：无效参数

4.3.4.3 HAL_Flash_MemoryOf

表 4-12 HAL_Flash_MemoryOf 接口函数说明

信息项	说明
原型	HAL_Status HAL_Flash_MemoryOf(uint32_t flash, FlashEraseMode size, uint32_t addr, uint32_t *start);
功能	计算输入地址所处可擦除 block 的首地址
参数	uint32_t flash 含义解释：Flash 次设备号，相当于 g_flash_cfg 结构体的搜引号。 使用说明：若选用 image Flash，次设备号为 0，若选用其他 Flash，次设备号为用户配置。 FlashEraseMode size 含义解释：擦除模式大小 使用说明：以输入的擦除模式大小(4K/32K/64K/Chip)为 block 来划分 Flash 区域 uint32_t addr 含义解释：Flash 内部物理地址 使用说明：N/A uint32_t *start

信息项	说明
	含义解释：返回的 block 首地址 使用说明：N/A
返回值	HAL_OK：成功 HAL_ERROR：一般错误 HAL_BUSY：设备忙 HAL_TIMEOUT：超时 HAL_INVALID：无效参数

4.3.4.4 HAL_Flash_Ioctl

表 4-13 HAL_Flash_Ioctl 接口函数说明

信息项	说明
原型	HAL_Status HAL_Flash_Ioctl(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、设置读/写模式、擦/写暂停参数等。
参数	uint32_t flash 含义解释：Flash 次设备号，相当于 g_flash_cfg 结构体的搜引号。 使用说明：若选用 image Flash，次设备号为 0，若选用其他 Flash，次设备号为用户配置。 FlashControlCmd attr 含义解释：功能操作行为类型 使用说明：N/A uint32_t arg 含义解释：实现功能的参数 使用说明：N/A
返回值	HAL_OK：成功 HAL_ERROR：一般错误 HAL_BUSY：设备忙 HAL_TIMEOUT：超时 HAL_INVALID：无效参数

5 示例说明

Flash 驱动模块提供的通用接口方便了用户对 Flash 进行读写擦操作，下面以对 Flash 进行擦除、写入、读取等操作示例来简要介绍 Flash 模块接口的使用方法。

5.1 示例简述

5.1.1 示例简介

Flash 擦写读示例的演示的目的是简要介绍 Flash 接口的基本使用方法（打开、擦除、写入、读取、关闭等操作），演示的功能是通过对 Flash 指定地址写入 0-255 数值，并进行读取对比判断。

5.1.1.1 获取方式

Flash 擦写读示例有示例工程代码，位于 XR806 SDK 的/project/example/flash 目录，以下此示例工程简称为 Flash 示例工程。

说明

XR806 SDK 可在以下 GitHub 仓库获取：https://github.com/XradioTech/xr806_sdk.git

---

5.1.1.2 准备工作

Flash 示例工程的硬件准备有如下。

1. 评估板：运行示例工程代码。
2. 串口线：连接评估板的 UART0 插针，用于 console 控制台的输入输出。
3. PC 机：用于镜像烧录和 console 控制的输入输出。

Flash 示例工程的软件准备，包括烧写工具、代码编译和烧写操作，请参见《XR806_SDK_快速入门指南》。

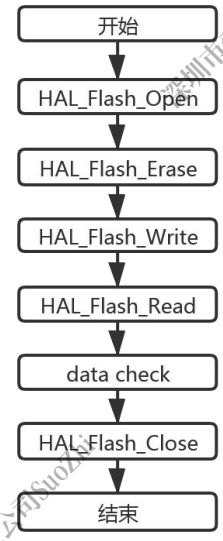
5.1.1.3 操作步骤

Flash 示例工程无需控制命令，完成烧写后，复位即可，示例代码自动运行。

5.1.2 关键实现

Flash 驱动模块的初始化、通信方式等配置可在 XR806 SDK 的 platform_init()函数中完成，所以本文重点描述 Flash 设备的打开、擦除、读取、写入、读取、关闭的操作过程。

图 5-1 Flash 示例流程图



具体操作见下面的示例代码。

```
static void flash_demo(void)
{
    uint32_t addr;
    FlashEraseMode size_type;
    uint8_t wdata[256];
    uint8_t rdata[256] = {0};
    int i;

    for (i = 0; i < 256; i++) {
        wdata[i] = i;
    }

    addr = 0x108000;
    size_type = FLASH_ERASE_4KB;

    printf("flash driver open...\n");
    if (HAL_Flash_Open(MFLASH, 5000) != HAL_OK) {
        printf("flash driver open failed\n");
        return;
    }

    printf("flash erase...\n");
    if (HAL_Flash_Erase(MFLASH, size_type, addr, 1) != HAL_OK) {
        printf("flash erase failed\n");
        goto fail;
    }
```

```
    printf("flash write...\n");
    if (HAL_Flash_Write(MFLASH, addr, wdata, sizeof(wdata)) != HAL_OK) {
        printf("flash write failed\n");
        goto fail;
    }

    printf("flash read...\n");
    if (HAL_Flash_Read(MFLASH, addr, rdata, 256) != HAL_OK) {
        printf("flash read failed\n");
        goto fail;
    }

    printf("flash read result:\n");
    for (i = 0; i < 256; i++) {
        printf("%d ", rdata[i]);
        if (i != 0 && (i % 16) == 0)
            printf("\n");
    }
    printf("\n");

    if(memcmp(wdata, rdata, sizeof(wdata)) == 0) {
        printf("flash erase write read suscess\n");
    } else
        printf("flash erase write read fail\n");

fail:
    printf("flash driver close.\n");
    if (HAL_Flash_Close(MFLASH) != HAL_OK) {
        printf("flash driver close failed\n");
        return;
    }
}
```

5.1.3 效果展示

在评估板中运行 Flash 示例程序后，在控制台中会打印输出 Flash 的擦写情况，如下所示。

```
flash demo started.
flash driver open...
flash erase...
flash write...
flash read...
flash read result:
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32
```

```
33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48
49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64
65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80
81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96
97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112
113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128
129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144
145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160
161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176
177 178 179 180 181 182 183 184 185 186 187 188 189 190 191 192
193 194 195 196 197 198 199 200 201 202 203 204 205 206 207 208
209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224
225 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240
241 242 243 244 245 246 247 248 249 250 251 252 253 254 255
flash erase write read suscess.
flash driver close.
flash demo over.
```

6 Flash Chip 支持

6.1 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 的 programmable page、sector/block 大小。
3. Flash 支持的命令类型及格式。
4. Flash 的 Status 寄存器。
5. Flash 配置 WP、HOLD 与 PIN2、PIN3 切换方式。
6. Flash 进入 XIP 的方式。
7. 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）。
8. Flash 的最高工作频率以及 READ 命令的最高工作频率。
9. 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）。

6.2 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 芯片需要实现 FlashChip 结构体内的接口(包括配置 FlashChipCfg 参数)，或复用 default flash chip 接口。

表 6-1 FlashChip 部分成员

成员项	说明
FlashChipCfg cfg;	含义解释：Flash 的芯片配置结构体 使用说明：cfg 中包括 Flash 芯片的基本硬件参数配置，可参见表 6-3
uint32_t mPageSize;	含义解释：页大小 使用说明：N/A
uint16_t FlashStatus;	含义解释：存储该芯片的状态，初始为 0 使用说明：N/A
uint8_t mDummyCount;	含义解释：通信时 Dummy 数据字节数，初始为 1 使用说明：N/A
struct FlashDrv *mDriver;	含义解释：通信驱动 使用说明：N/A
struct XipDrv *mXip;	含义解释：XIP 驱动

成员项	说明
	使用说明：N/A

FlashChip 结构体中函数指针接口的功能定义如下。

表 6-2 FlashChip 内部函数接口

函数指针	功能说明
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/写操作
(*resumeErasePageprogram)()	恢复擦/写操作
(*powerDown)()	进入掉电模式
(*releasePowerDown)()	退出掉电模式
(*jedecID)()	获取 JEDEC ID
(*enableQPIMode)()	进入 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
(*isSuspend)()	Flash 是否被擦/写暂停
(*minEraseSize)()	返回最小的擦除大小

FlashChipCfg 为 Flash 芯片的配置结构体，主要用于配置 Flash 芯片的硬件参数，具体成员如下。

表 6-3 FlashChipCfg 成员

成员项	说明
uint32_t mJedec;	含义解释：Flash 的 jedec ID 使用说明：参见“6.4 支持步骤”章节
uint32_t mSize;	含义解释：芯片存储容量 使用说明：N/A
uint32_t mMaxFreq;	含义解释：除了 READ 命令以外，其他命令（含 FAST READ 等）允许的最高频率 使用说明：N/A
uint32_t mMaxReadFreq;	含义解释：READ 命令的最高频率 使用说明：N/A
uint32_t mEraseSizeSupport;	含义解释：Flash 芯片支持哪些擦除命令 使用说明：参见“6.4 支持步骤”章节
uint16_t mReadSupport;	含义解释：Flash 芯片支持哪些读命令 使用说明：参见“6.4 支持步骤”章节
uint16_t mReadStausSupport;	含义解释：Flash 芯片支持哪些读状态寄存器 使用说明：参见“6.4 支持步骤”章节
uint8_t mWriteStatusSupport;	含义解释：Flash 芯片支持哪些写状态寄存器 使用说明：参见“6.4 支持步骤”章节
uint8_t mPageProgramSupport;	含义解释：Flash 芯片支持哪些烧写命令 使用说明：参见“6.4 支持步骤”章节
uint8_t mSuspendSupport;	含义解释：Flash 芯片支持擦/写暂停情况 使用说明：N/A
uint16_t mSuspend_Latency;	含义解释：发送暂停命令后需等待的最小延时 使用说明：N/A
uint16_t mResume_Latency;	含义解释：发送恢复命令后需等待的最小延时 使用说明：N/A

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

表 6-4 FlashChipCtor 成员

成员	描述
uint32_t mJedecId;	含义解释：该 Flash 芯片的 JEDEC ID 使用说明：N/A
(*enumerate)()	含义解释：创建该 Flash 芯片的实体 使用说明：为函数指针接口
(*init)()	含义解释：初始化 Flash 芯片的实体，替换接口具体实现 使用说明：为函数指针接口
(*destory)()	含义解释：删除 Flash 芯片的实体 使用说明：为函数指针接口

Flash Chip Creator 会放进 flashChipList[]里面，如下。

```
FlashChipCtor *flashChipList[] = {
        &DefaultFlashChip, /*default chip must be at the first*/
        &XT25F16B_FlashChip,
        &XM25QH64A_FlashChip,
};
```

6.3 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(详见 6.4.1)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 6-5 Default Flash Chip 命令

指令名	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)

指令名	CMD(IO)	Address Bytes(IO)	Dummy Bytes(IO)	Data Bytes(IO)
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 I/O	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 I/O	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)

6.4 支持步骤

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。新的 Flash Chip 分为两类：

一类是其芯片命令与 default 实现一致，为 Default Flash Chip 类型，只需进行简单配置即可扩展支持该 Flash 芯片，参见“6.4.1Default Flash Chip 支持”章节；

另一类是该 Flash 芯片的某些命令操作与 default 实现不一致，该芯片为非 Default Flash Chip 类型，则需要进行对应接口的重写和挂载，参见“6.4.2 非 Default Flash Chip 支持”章节。

SDK 已经支持的 Flash 芯片可以通过 SDK/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 芯片，又没有添加，那么默认是跑 DefaultFlashChip 中的接口，如果 Flash 芯片的寄存器读写与 default 实现的接口不同，那么四线读模式可能有异常。

6.4.1 Default Flash Chip 支持

简易配置通过扩展 simpleFlashChipCfg 数组实现。在数组里增加 FlashChipCfg 结构体类型的元素并配置，以支持需要扩展的 Flash 芯片参数。(simpleFlashChipCfg 数组在 src/driver/chip/flashchip/flash_chip_cfg.c 文件中定义)

表 6-6 FlashChipCfg 参数配置

参数	配置描述
mJedec	Flash 的 jedec ID，24bit 长度：0xX0X1X2X3X4X5，X0X1 是 ID7~ID0，X2X3 是 ID15~ID8，X4X5 是 M7～M0。
mSize	芯片存储容量，XXXMbit
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
mEraseSizeSupport	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）
mReadSupport	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
mReadStatusSupport	Flash 芯片支持哪些读状态寄存器，FLASH_STATUS1，FLASH_STATUS2，FLASH_STATUS3
mWriteStatusSupport	Flash 芯片支持哪些写状态寄存器，FLASH_STATUS1，FLASH_STATUS2，FLASH_STATUS3
mPageProgramSupport	Flash 芯片支持哪些烧写命令，FLASH_PAGEPROGRAM，FLASH_QUAD_PAGEPROGRAM
mSuspendSupport	表示是否支持擦/写暂停，‘1’支持，‘0’不支持
mSuspend_Latency	发送暂停命令后需等待的最小延时
mResume_Latency	发送恢复命令后需等待的最小延时

SDK 中 FlashChipCfg 结构体声明如下：

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

```
    uint32_t mMaxFreq;
    uint32_t mMaxReadFreq;

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

    uint16_t mSuspend_Latency;
    uint16_t mResume_Latency;
} FlashChipCfg;
```

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）。F_R 为 100MHz，f_R 为 55MHz，因此 mMaxFreq 填 100 * 1000 * 1000，mMaxReadFreq 填 55 * 1000 * 1000。接着从指令集中获取信息，对擦除、读、写、读寄存器、写寄存器的支持配置进行填写。

图 6-1 PN25F16B JEDEC ID

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 6-2 PN25F16B 频率参数

Table 8.6 AC Electrical Characteristics

SYMBOL	ALT	Parameter	SPEC			UNIT
			MIN	TYP.	MAX	
F_R	f_C	Clock frequency For all instructions, except Read Data (03h) and Dual output(3bh) 2.7V-3.6V V_{CC} & Industrial Temperature	D.C.		100	MHz
f_R		Clock freq. Read Data instruction (03h)	D.C.		55	MHz

图 6-3 PN25F16B 指令集

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			

在 simpleFlashChipCfg 数组中的具体配置如下：

```
static const FlashChipCfg simpleFlashChipCfg[] =
{
/* ..... */
#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,
        .mSuspendSupport = 0,
        .mSuspend_Latency = 0,
        .mResume_Latency = 0,
```

```
    },
#endif
/* ..... */
}
```

6.4.2 非 Default Flash Chip 支持

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 接口。

图 6-4 P25Q16H 的读写寄存器命令

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

与表 6-4 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,
};
```

P25QXXH 芯片 FlashChipCfg 配置：

```
static const FlashChipCfg _P25QXXH_FlashChipCfg = {
    .mJedec = P25Q40H_JEDEC,
    .mSize = 16 * 8 * 4096,
    .mEraseSizeSupport = FLASH_ERASE_64KB | FLASH_ERASE_32KB | FLASH_ERASE_4KB |
                         FLASH_ERASE_CHIP,
    .mPageProgramSupport = FLASH_PAGEPROGRAM | FLASH_QUAD_PAGEPROGRAM,
    .mReadStausSupport = FLASH_STATUS1 | FLASH_STATUS2 | FLASH_STATUS3,
    .mWriteStatusSupport = FLASH_STATUS1 | FLASH_STATUS2 | FLASH_STATUS3,
    .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,
```

```
    .mMaxFreq = 104 * 1000 * 1000,
    .mMaxReadFreq = 55 * 1000 * 1000,
    .mSuspendSupport = 1,
    .mSuspend_Latency = 30,
    .mResume_Latency = 200,
};
```

创建 P25QXXH 系列芯片实例：

```
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;
    }

    HAL_Memcpy(&chip->cfg, &_P25QXXH_FlashChipCfg, sizeof(FlashChipCfg));
    chip->cfg.mJedec = jedec;
    chip->cfg.mSize = size;

    if (jedec == P25Q64H_JEDEC) {
        chip->cfg.mMaxFreq = 96 * 1000 * 1000;
        chip->cfg.mReadSupport = chip->cfg.mReadSupport |
                                 FLASH_READ_QPI_MODE;
    }
    chip->mPageSize = 256;
    chip->mFlashStatus = 0;
    chip->mDummyCount = 1;
    return chip;
}
```

写状态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));

if (reg == FLASH_STATUS1) {
    if ((chip->cfg.mJedec & 0xFF0000) < 0x160000) {
        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;
} else if (reg == FLASH_STATUS2) {
    if ((chip->cfg.mJedec & 0xFF0000) < 0x160000) {
        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).len = 2;
        FCI_DATA(1).line = 1;
    } else {
        status_buf[0] = *status;

        FCI_CMD(0).data = FLASH_INSTRUCTION_WRSR2_AB15;
```

```
            FCI_DATA(1).len = 1;
            FCI_DATA(1).line = 1;
        }
        FCI_DATA(1).pdata = status_buf;
    } else if (reg == FLASH_STATUS3) {
        if ((chip->cfg.mJedec & 0xFF0000) < 0x160000) {
            FCI_CMD(0).data = FLASH_INSTRUCTION_WRCR;
        } else {
            FCI_CMD(0).data = FLASH_INSTRUCTION_WRCR_AB15;
        }
        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);

    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;

    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
};
```

7 常见用户方案

XR806 芯片中的 Flash_PSRAM Controller 可只接 Flash，也可同时接 Flash 和 PSRAM 进行使用。因此，用户对 Flash 的使用可以有多种情况。常见的使用方案有以下三种，包括：只使用 SiP Flash、SiP PSRAM 和外挂 Flash 共存、SiP Flash 和外挂 Flash 共存。

对于不同的 Flash 使用方案，用户需要先确定所用的 XR806 芯片的型号，确认 SiP 类型，再对所用 Flash 的 pinmux 进行确认和配置。

表 7-1 XR806 芯片型号

型号	主要区别
XR806AF2L	SiP 2MB Flash，QFN32 封装
XR806BF2L	SiP 2MB Flash，QFN40 封装
XR806BM2L	SiP 2MB PSRAM，QFN40 封装

7.1 Pinmux 的确认和配置

XR806 的 g_pinmux_flashc_sip 与 g_pinmux_psram_sip 引脚连接均为 PB8~PB13，见 XR806 SDK 的 project/common/board/board_common.c 文件中定义。

XR806 的 g_pinmux_flashc 与 g_pinmux_psram 引脚连接均为 PB2~PB7，见 XR806 SDK 的 project/common/board/xr806_dig_ver/board_config.c 文件中定义。

说明

为了区别 Flash_PSRAM Controller 的两组引脚，SDK 默认选用 CS1 作为 PB8~PB13(SiP)这组引脚的片选，选用 CS0 作为 PB2~PB7 这组引脚的片选。

因此，对 Flash 设备的 pinmux 的配置规则如下：

1. 与 FlashController 连接的物理引脚分别为 PB8~13(SiP)和 PB2~7，FlashController 可同时连一个 PSRAM 和一个 Flash，使用时，需要将他们的引脚和片选一一对应（CS1 为 SiP 使用，CS0 为外挂使用）。
2. 在 FlashController 的两组引脚均被占用情况下，若新增一个 Flash，则只能连接 SPI0，使用 SPI 通信。

下面以上述三种常见的 Flash 使用情况为例，分别介绍各情况下的 pinmux 配置。

7.2 方案一：仅 SiP Flash

使用的 XR806 芯片为 SiP Flash 型号（XR806AF2L/XR806BF2L），不需要外挂 Flash（可外接 PSRAM），则 pinmux 可配置为如下：

表 7-2 仅使用 SiP Flash 方案

设备	引脚连接与配置	Flash 驱动通信方式
SiP Flash	PB8、PB9、PB10、PB11、PB12、PB13	使用 FlashController
外接 PSRAM（可选）	PB2、 PB3、 PB4、 PB5、 PB6、 PB7	使用 FlashController

7.3 方案二：SiP PSRAM 和外挂 Flash

使用的 XR806 芯片为 SiP PSRAM 型号（XR806BM2L），需要外挂一个 Flash，则 pinmux 可配置为如下：

表 7-3 SiP PSRAM 与外挂 Flash 共存方式一

设备	引脚连接与配置	Flash 驱动通信方式
SiP PSRAM	PB8、PB9、PB10、PB11、PB12、PB13	使用 FlashController
外接 Flash	PB2、 PB3、 PB4、 PB5、 PB6、 PB7	使用 FlashController

表 7-4 SiP PSRAM 与外挂 Flash 共存方式二

设备	引脚连接与配置	Flash 驱动通信方式
SiP PSRAM	PB8、PB9、PB10、PB11、PB12、PB13	使用 FlashController
外接 Flash	PA（10、11、15、16）或者 PA（19、20、21、22）或者 PB（4、5、6、7）	使用 SPI0

此种情况，一般建议采用上述方式一，外接 Flash 选择使用 FlashController 通信。

7.4 方案三：SiP Flash 和外挂 Flash

使用的 XR806 芯片为 SiP Flash 型号（XR806AF2L/XR806BF2L），若 Flash 容量不够使用，可外挂一个 Flash，pinmux 可配置为如下：

表 7-5 SiP Flash 与外挂 Flash 共存方式一（无 PSRAM）

设备	引脚连接与配置	Flash 驱动通信方式
SiP Flash	PB8、PB9、PB10、PB11、PB12、PB13	使用 FlashController
外接 Flash	PA（10、11、15、16）或者 PA（19、20、21、22）或者 PB（4、5、6、7）	使用 SPI0

若用户还需外接一个 PSRAM，则 pinmux 可配置为如下：

表 7-6 SiP Flash 与外挂 Flash 共存方式二（含 PSRAM）

设备	引脚连接与配置	Flash 驱动通信方式
SiP Flash	PB8、PB9、PB10、PB11、PB12、PB13	使用 FlashController
外接 Flash	PA（10、11、15、16）或者 PA（19、20、21、22）	使用 SPI0
外接 PSRAM（可选）	PB2、PB3、PB4、PB5、PB6、PB7	使用 FlashController

7.5 方案使用示例

使用一块 Flash 设备的方法可参见“4.2 配置说明”章节即可，若用户需要在 XR806 芯片上使用两块 Flash 设备（SiP Flash 与外挂 Flash 共存情况），则需要先对外挂 Flash 的引脚进行选择与确认，可参见表 7-7 与表 7-8。

下面以外挂 Flash 选择 SPI0（PA10、PA11、PA15、PA16）为例，进行两块 Flash 的配置使用，与“4.2 配置说明”章节思路一致，步骤如下：

1. 分别配置两个 Flash 设备的 pinmux。
2. pinmux 配置完毕后，检查/确认对应 Flash 驱动的通信方式（g_flash_cfg 数组）。
3. 在 XR806 SDK 的 platform_init_level0()接口中创建新增的 Flash 设备。

示例配置如下：

检查 SiP Flash 的 Pinmux

```
static GPIO_PinMuxParam g_pinmux_flashc_sip[] = {
    { GPIO_PORT_B, GPIO_PIN_8,  { GPIOB_P8_F2_FLASH_WP,   GPIO_DRIVING_LEVEL_3, GPIO_PULL_UP   } },
    { GPIO_PORT_B, GPIO_PIN_9,  { GPIOB_P9_F2_FLASH_HOLD, GPIO_DRIVING_LEVEL_3, GPIO_PULL_UP   } },
    { GPIO_PORT_B, GPIO_PIN_10, { GPIOB_P10_F2_FLASH_MOSI, GPIO_DRIVING_LEVEL_3, GPIO_PULL_NONE } },
    { GPIO_PORT_B, GPIO_PIN_11, { GPIOB_P11_F2_FLASH_MISO, GPIO_DRIVING_LEVEL_3, GPIO_PULL_NONE } },
    { GPIO_PORT_B, GPIO_PIN_12, { GPIOB_P12_F2_FLASH_CS1, GPIO_DRIVING_LEVEL_3, GPIO_PULL_UP   } },
    { GPIO_PORT_B, GPIO_PIN_13, { GPIOB_P13_F2_FLASH_CLK, GPIO_DRIVING_LEVEL_3, GPIO_PULL_NONE } },
};
```

配置外挂 Flash 的 Pinmux（用户需保证 PA10、PA11、PA15、PA16 为空闲 SPI 引脚，不作它用）

```
__xip_rodata
static const GPIO_PinMuxParam g_pinmux_spi0[] = {
    { GPIO_PORT_A, GPIO_PIN_10, { GPIOA_P10_F3_SPI0_MOSI, GPIO_DRIVING_LEVEL_1, GPIO_PULL_NONE } },
```

```
    { GPIO_PORT_A, GPIO_PIN_11,   { GPIOA_P11_F3_SPI0_MISO,   GPIO_DRIVING_LEVEL_1,
GPIO_PULL_NONE } },
    { GPIO_PORT_A, GPIO_PIN_16,   { GPIOA_P16_F3_SPI0_CLK,    GPIO_DRIVING_LEVEL_1,
GPIO_PULL_NONE } },
};

__xip_rodata
static const GPIO_PinMuxParam g_pinmux_spi0_cs0[] = {
    { GPIO_PORT_A, GPIO_PIN_15,   { GPIOA_P15_F3_SPI0_CS0,    GPIO_DRIVING_LEVEL_1,
GPIO_PULL_UP } },
};
```

检查 Flash 驱动的通信方式（g_flash_cfg 数组）：

```
/* flash */
static const FlashBoardCfg g_flash_cfg[] = {
    {/* for flashc */
        .type = FLASH_DRV_FLASHC,
        .mode = FLASH_READ_DUAL_O_MODE,
        .clk = (48 * 1000 * 1000),
    },
    {/* for spi */
        .type = FLASH_DRV_SPI,
        .mode = FLASH_READ_FAST_MODE,
        .clk = BOARD_SPI_MCLK,
    },
    ...
};
```

在 platform_init_level0()接口中 image Flash 初始化后面复制如下代码，以创建新增的 Flash 设备：

```
int dev_flash_add;
FlashBoardCfg *cfg_flash_add = NULL;
...
    /* The minor number is the index number of g_flash_cfg array. */
    dev_flash_add = HAL_MKDEV(HAL_DEV_MAJOR_FLASH, 1);
    HAL_BoardIoctl(HAL_BIR_GET_CFG, dev_flash_add, (uint32_t)&cfg_flash_add);
    if (cfg == NULL) {
        FWK_NX_LOG(1, "getFlashBoardCfg2 failed");
        return;
    }
    if (cfg->type == FLASH_DRV_SPI) {
        cfg->spi.port = BOARD_SPI_PORT;
        cfg->spi.cs = SPI_TCTRL_SS_SEL_SS0;
        cfg->spi.cs_level = BOARD_SPI_CS_LEVEL;
```

```
}/* The minor number 1 can be defined as a macro */
HAL_Flash_Init(1, cfg_flash_add);
```

新 Flash 设备的次设备号为 1，与 g_flash_cfg 数组中的搜引号匹配。上述步骤完成，用户调用 platform_init() 接口进行平台初始化后，即可对两个 Flash 进行使用，Flash 接口调用时，以次设备号来区分。

其他情况下 Flash 使用方案，用户可以此类推。

附录 A： 术语表

表 A-1 术语表

Q		
QPI	Quad Peripheral Interface	四路外设接口
S		
SPI	Serial Peripheral Interface	串行外设接口
SiP	System in Package	系统级封装
X		
XIP	eXecute In Place	芯片内执行

著作权声明

商标声明

免责声明