OPL1000

ULTRA-LOW POWER 2.4GHZ WI-FI + BLUETOOTH SMART SOC

Flash User Guide

OPULINKS

OPL1000-Flash-User-Guide-R01 | Version V01

REVISION HISTORY

Date	Version	Contents Updated
TBD	0.1	• Initial Release

TABLE OF CONTENTS

TABLE OF CONTENTS

TABLE OF CONTENTS

INDEX

表目錄

圖目錄

1. 介绍

1.1. 文档应用范围

本文档介绍了在 OPL1000 DEVKIT 上，使用 Flash 的方法。

1.2. 缩略语

Abbr.	Explanation
FIM	Flash Item Management

1.3. 参考文献

- OPL1000-DS-NonNDA.pdf

2. FLASH 读写接口

头文件 (Header file)：

hal_flash.h

Flash 是以扇区 (Sector) 作为基本单位，单位大小为 4KB，起始位置为 0x00000000。

2.1. Hal_Flash_4KSectorAddrErase

功能	擦除 Flash 的某个扇区
函数定义	uint32_t Hal_Flash_4KSectorAddrErase(E_SpiIdx_t u32SpiIdx, uint32_t u32SecAddr);
参数	1. eSpiIdx: Index of SPI. refert to E_SpiIdx_t 2. u32SecAddr: Address of sector (must sector aligned, LSBs truncated)
返回值	0: setting complete 1: error

2.2. Hal_Flash_AddrRead

功能	读取 Flash 特定位置的数据
函数定义	uint32_t Hal_Flash_AddrRead(E_SpiIdx_t u32SpiIdx, uint32_t u32StartAddr, uint8_t u8UseQuadMode, uint32_t u32Size, uint8_t *pu8Data);
参数	1. eSpiIdx: Index of SPI. refert to E_SpiIdx_t 2. u32StartAddr: Start address 3. u8UseQuadMode: Qaud-mode select. 1 for enable/0 for disable 4. u32Size: Data size 5. pu8Data: Data buffer
返回值	0: setting complete 1: error

2.3. Hal_Flash_AddrProgram

功能	写入 Flash 特定位置的数据
函数定义	uint32_t Hal_Flash_AddrProgram(E_SpiIdx_t u32SpiIdx, uint32_t u32StartAddr, uint8_t u8UseQuadMode, uint32_t u32Size, uint8_t *pu8Data);
参数	1. eSpiIdx: Index of SPI. refert to E_SpiIdx_t 2. u32StartAddr: Start address 3. u8UseQuadMode: Qaud-mode select. 1 for enable/0 for disable 4. u32Size: Data size 5. pu8Data: Data buffer
返回值	0: setting complete 1: error

注意：

需要先有擦除的动作，才能进行写入。

3. FLASH LAYOUT

Flash 目前已经配置使用的区域如下：

表格 1：Flash Layout 表

程序区	0x00000000 ~ 0x00077000 476 KB	代码生成 bin 文件，刻录到此 Flash 区域，不开放使用。
系统参数区	0x00077000 ~ 0x00080000 36 KB	用于存放系统参数，有开放应用使用。(下一章节，会进一步说明)
保留区	0x00080000 ~ 0x000F8000 480 KB	保留给应用端使用的区域，可以任意配置。
BLE 资料区	0x000F8000 ~ 0x00100000 32 KB	用于存放 BLE 数据区，不开放使用。

4. FIM (FLASH ITEM MANAGEMENT)

在系统参数区，集成一个特殊功能 FIM (Flash Item Management)。将 Flash (erase / read / write) 进行抽象化，提供额外的接口，管理系统参数、应用参数。可按照后续章节说明，进行配置与使用。

4.1. FIM 配置

对应文件：

\APS_PATCH\middleware\netlink\mw_fim\mw_fim_default_patch.h

提供 4 个配置区域，每个区域最多使用 9 个扇区，应用可以根据需求，修改配置。

```
// the information of zone
#define MW_FIM_ZONE0_BASE_ADDR_PATCH    0x00077000
#define MW_FIM_ZONE0_BLOCK_SIZE_PATCH   0x1000
#define MW_FIM_ZONE0_BLOCK_NUM_PATCH    9               // swap + used

#define MW_FIM_ZONE1_BASE_ADDR_PATCH    0x00080000
#define MW_FIM_ZONE1_BLOCK_SIZE_PATCH   0x1000
#define MW_FIM_ZONE1_BLOCK_NUM_PATCH    0               // swap + used

#define MW_FIM_ZONE2_BASE_ADDR_PATCH    0x00081000
#define MW_FIM_ZONE2_BLOCK_SIZE_PATCH   0x1000
#define MW_FIM_ZONE2_BLOCK_NUM_PATCH    0               // swap + used

#define MW_FIM_ZONE3_BASE_ADDR_PATCH    0x00082000
#define MW_FIM_ZONE3_BLOCK_SIZE_PATCH   0x1000
#define MW_FIM_ZONE3_BLOCK_NUM_PATCH    0               // swap + used
```

目前系统预先配置一个区域：

0x00077000 ~ 0x00080000

9 个扇区，使用 36 KB

MW_FIM_ZONE0_BASE_ADDR_PATCH	0x00077000	区域起始位置
MW_FIM_ZONE0_BLOCK_SIZE_PATCH	0x1000	扇区大小 (4 KB)
MW_FIM_ZONE0_BLOCK_NUM_PATCH	9	使用多少扇区 0: 不使用 2: 最小使用数量 9: 最大使用数量

4.2. 参数配置

4.2.1. 选择 Block 与版本号

对应文件：

\APS_PATCH\middleware\netlink\mw_fim\mw_fim_default_patch.h

透过批注 (Comment) 记录每个 block 使用状态，使用如下：

MW_FIM_VERmn_PATCH 0xpp // comment

m: 第几个区域

n: 第几个 block

pp: 此 block 的版本号

comment: 哪个模块使用了此 block

```
// the version of group
#define MW_FIM_VER00_PATCH      0x01    // reserve for swap
#define MW_FIM_VER01_PATCH      0x04    // system & driver
#define MW_FIM_VER02_PATCH      0x03    // for WIFI
#define MW_FIM_VER03_PATCH      0x03    // calibration data
#define MW_FIM_VER04_PATCH      0x02    // for LE Controller
```

```
#define MW_FIM_VER05_PATCH      0x01
#define MW_FIM_VER06_PATCH      0x01
#define MW_FIM_VER07_PATCH      0x02    // For BLE
#define MW_FIM_VER08_PATCH      0x01
```

版本号，更新原则：

有进行「新增参数、删除参数、初始值更新」的时候，必须更新版本号。

4.2.2. 参数

对应文件：

\APS_PATCH\middleware\netlink\mw_fim\ mw_fim_default_groupmn_patch.h

\APS_PATCH\middleware\netlink\mw_fim\ mw_fim_default_groupmn_patch.c

m: 第几个区域

n: 第几个 block

以 mw_fim_default_group08_patch.h / mw_fim_default_group08_patch.c 当范例：

■ 新增一个识别 ID

对应文件：

\APS_PATCH\middleware\netlink\mw_fim\mw_fim_default_group08_patch.h

识别 ID : MW_FIM_IDX_GP08_PATCH_TEST

```
// the file ID
typedef enum
{
    MW_FIM_IDX_GP08_PATCH_START = 0x00080000,            // the start IDX of
group 08

    MW_FIM_IDX_GP08_PATCH_TEST,

    MW_FIM_IDX_GP08_PATCH_MAX
} E_MwFimIdxGroup08_Patch;
```

起始识别 ID，规则如下：

```
// the file ID
// xxxx_xxxx_xxxx_xxxx_xxxx_xxxx_xxxx_xxxx
// ^^^^ ^^^^ Zone (0~3)
//           ^^^^ ^^^^ Group (0~8), 0 is reserved for swap
//                     ^^^^ ^^^^ ^^^^ ^^^^ File ID, start from 0
```

如果有新增其他区域、其他 block，可仿照此范例，产生对应代码。

注意：

MW_FIM_IDX_GP08_PATCH_START = 0x00080000,　　// the start IDX of group 08

是第 0 个区域、第 8 个 block 的起始识别 ID

■ 新增参数内容信息

对应文件：

\APS_PATCH\middleware\netlink\mw_fim\mw_fim_default_group08_patch.h

定义参数内容 : T_TestConfig

每个参数所占用空间 : MW_FIM_TEST_CFG_SIZE

需要的参数数量 : MW_FIM_TEST_CFG_NUM

```
// the information of Test config
typedef struct
{
    uint32_t ulTest1;
    uint8_t ubTest2;
    uint8_t ubTest3;
    uint8_t ubTest4;
    uint8_t ubTest5;
} T_TestConfig;
#define MW_FIM_TEST_CFG_SIZE    sizeof(T_TestConfig)
#define MW_FIM_TEST_CFG_NUM     16
```

■ 参数初始化与准备

对应文件：

\APS_PATCH\middleware\netlink\mw_fim\mw_fim_default_group08_patch.c

参数初始值 : g_tMwFimDefaultTestConfig

参数所以地址信息存取空间 : g_ulaMwFimAddrBufferTestConfig[MW_FIM_TEST_CFG_NUM]

填写 block 的信息 : g_taMwFimGroupTable08_patch[]

识别 ID

参数的数量

每个参数的大小

参数初始值

参数地址信息存取空间

```
// the default value of Test config
const T_TestConfig g_tMwFimDefaultTestConfig =
{
    1,      // uint32_t ulTest1;
    2,      // uint8_t ubTest2;
    3,      // uint8_t ubTest3;
    4,      // uint8_t ubTest4;
    5       // uint8_t ubTest5;
};

// the address buffer of Test config
uint32_t g_ulaMwFimAddrBufferTestConfig[MW_FIM_TEST_CFG_NUM];

// the information table of group 08
const T_MwFimFileInfo g_taMwFimGroupTable08_patch[] =
{
    {MW_FIM_IDX_GP08_PATCH_TEST,     MW_FIM_TEST_CFG_NUM,     MW_FIM_TEST_CFG_SIZE,
(uint8_t*)&g_tMwFimDefaultTestConfig, g_ulaMwFimAddrBufferTestConfig}

    // the end, don't modify and remove it
    {0xFFFFFFFF,            0x00,              0x00,               NULL,
NULL}
};
```

注意：

如果此参数，不需要初始值，填写时，可以填入 NULL

■ 更新 block 信息

对应文件：

\APS_PATCH\middleware\netlink\mw_fim\mw_fim_default_patch.c

修改对应的 block : g_ptaMwFimGroupInfoTable_patch

```
// the information table of all group
extern const T_MwFimFileInfo g_taMwFimGroupTableNull[];
T_MwFimFileInfo* g_ptaMwFimGroupInfoTable_patch[MW_FIM_ZONE_MAX][MW_FIM_GROUP_MAX]
=
{
    // zone 0
    {
        (T_MwFimFileInfo*)g_taMwFimGroupTableNull,      // reserve for swap
        (T_MwFimFileInfo*)g_taMwFimGroupTable01_patch,
        (T_MwFimFileInfo*)g_taMwFimGroupTable02_patch,
        (T_MwFimFileInfo*)g_taMwFimGroupTable03_patch,
        (T_MwFimFileInfo*)g_taMwFimGroupTable04_patch,
        (T_MwFimFileInfo*)g_taMwFimGroupTable05_patch,
        (T_MwFimFileInfo*)g_taMwFimGroupTable06_patch,
        (T_MwFimFileInfo*)g_taMwFimGroupTable07_patch,
        (T_MwFimFileInfo*)g_taMwFimGroupTable08_patch
    },
```

4.3. FIM 读写接口

頭文件 (Header file)：

mw_fim.h

4.3.1. MwFim_FileRead

功能	讀取特定識別 ID 的資料
函數定義	uint8_t MwFim_FileRead(uint32_t ulFileId, uint16_t uwRecIdx, uint16_t uwFileSize, uint8_t *pubFileData);
參數	1. ulFileId : [In] the file ID 2. uwRecIdx : [In] the index of record 3. uwFileSize : [In] the data size of file 4. pubFileData : [Out] the pointer of file data
返回值	1. MW_FIM_OK : success 2. MW_FIM_FAIL : fail

4.3.2. MwFim_FileWrite

功能	寫入特定識別 ID 的資料
函數定義	uint8_t MwFim_FileWrite(uint32_t ulFileId, uint16_t uwRecIdx, uint16_t uwFileSize, uint8_t *pubFileData);
參數	1. ulFileId : [In] the file ID 2. uwRecIdx : [In] the index of record 3. uwFileSize : [In] the data size of file 4. pubFileData : [Out] the pointer of file data
返回值	1. MW_FIM_OK : success 2. MW_FIM_FAIL : fail

4.3.3. MwFim_FileWriteDefault

功能	將特定識別 ID 的資料，回復至初始值
函數定義	uint8_t MwFim_FileWriteDefault(uint32_t ulFileId, uint16_t uwRecIdx);
參數	1. ulFileId : [In] the file ID 2. uwRecIdx : [In] the index of record
返回值	1. MW_FIM_OK : success

功能	將特定識別 ID 的資料，回復至初始值
	2. MW_FIM_FAIL : fail

4.3.4. MwFim_FileDelete

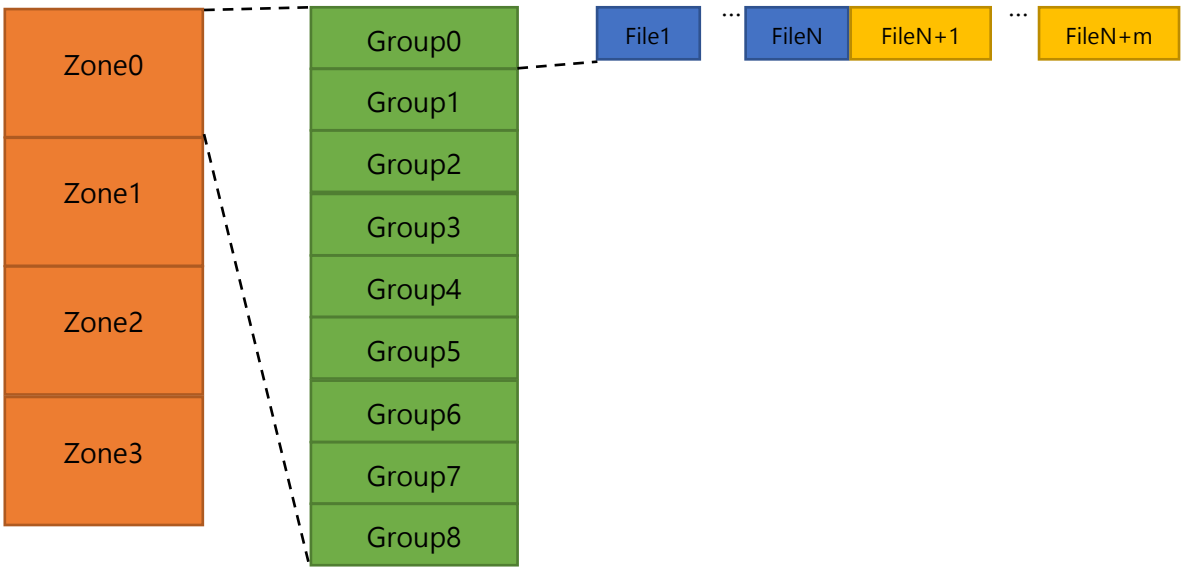
功能	將特定識別 ID 的資料，進行刪除
函數定義	uint8_t MwFim_FileDelete(uint32_t ulFileId, uint16_t uwRecIdx);
參數	1. ulFileId : [In] the file ID
	2. uwRecIdx : [In] the index of record
返回值	1. MW_FIM_OK : success
	2. MW_FIM_FAIL : fail

5. FIM 应用开发使用说明

5.1. FIM 架构

5.1.1. FIM 定义的可配置区块框架图

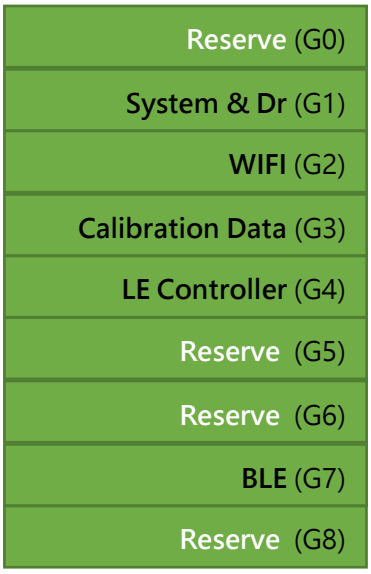
图表 1:配置区块框架图



如上图所示，在 Flash Item Management(FIM)中将 Flash 可配置空间分割为四个 Zones。其中每一个 Zone 包含了 9 个 Groups 可供配置。每个 Group 将会依据数据结构的类型与存放数量来配置连续的储存空间。以上图为例第一类数据结构的数据有 N 笔，第二类数据结构的数据有 m 笔，依次紧紧相邻储存。以此类推。

5.1.2. FIM 在默认情况下系统占用的配置区块

图表 2: 配置区块



如上图所示，在默认情况下系统已配置区块有 System & Driver (Group1)、WIFI(Group2)、Calibration Data(Group3)、LE Controller(Group4)、BLE (Group7)。目前在默认情况下系统只使用到 Zone0 的空间，上图的 Groups 指得是 Zone0 中的 Groups。在 5.2 小节中将会阐述如何这些系统已配置区块中的参数进行修改。在 5.3 小节将会阐述如何使用这些仍处于 Reserved 状态的 Group。在 5.4 小节中将会以一张列表的形式列出系统已配置区块中各个 Group 内可以配置的数据结构有哪些，这些数据结构中有哪些参数可供配置以及这些参数的默认值。

5.2. 如何修改 FIM 初始参数值

某些存于 Flash 中的系统默认配置参数，可能会在系统初始化阶段被读取出来作为系统初始化参数来使用，若是希望修改储存在 Flash 中的配置参数该怎么做?在此以存在 **System & Driver** (Group1)中索引类型为 MW_FIM_IDX_GP01_UART_CFG 的数据结构为例，该索引对应的数据结构用来储存 UART 配置所需的参数，数据结构如下所示。

```
typedef struct
{
    uint32_t ulBuadrate;
    uint8_t ubDataBit;
    uint8_t ubStopBit;
    uint8_t ubParity;
    uint8_t ubFlowCtrl;
} T_HalUartConfig;
```

Step 1. 先到 APS_PATCH\middleware\netlink\mw_fim\mw_fim_default_group01.h 去查看 MW_FIM_IDX_GP01_UART_CFG 这类型数据结构总共有几份实体存放于 Flash 中。

```
#define MW_FIM_UART_CFG_SIZE     sizeof(T_HalUartConfig)
#define MW_FIM_UART_CFG_NUM      2
```

此例能够看到 UART 配置参数数据结构在 Flash 会存放两份实体，分别为编号 0 和编号 1 的实体。

Step 2. 将 APS_PATCH\driver\chip\hal_uart\hal_uart_patch.h include 到用户正在开发的 application.h 中，创建一个 T_HalUartConfig 数据结构的实体，并在各个字段填入用户想写入到 Flash 中的值。

```
#include "hal_uart_patch.h"

T_HalUartConfig modified_the_UartConfig =
{
    115200,
    DATA_BIT_8,
    STOP_BIT_1,
    PARITY_NONE,
    0
};
```

Step 3. 将 APS_PATCH\middleware\netlink\mw_fim\Mw_fim.h include 到用户正在开发的 application.h 中，接着呼叫 MwFim_FileWrite()即可将 Flash 中所存放的配置改写成 step2 所设定的值。

```
#include "mw_fim.h"

ret = MwFim_FileWrite(MW_FIM_IDX_GP01_UART_CFG, 0, MW_FIM_UART_CFG_SIZE, &modified_the_UartConfig);
```

MwFim_FileWrite 会以 MW_FIM_IDX_GP01_UART_CFG 为索引到 information table 里面到对应的 File 类型，再找到该 File 类型下编号为 0 的实体，将 modified_the_UartConfig 中设定好的值写入到该实体中。

5.3. 如何储存应用数据至 Flash 特定区块

在开发应用时可能会有需要将应用相关数据储存到 Flash 中的需求，该如何将数据储存到指定的 Group 中呢?

■ FIM 的运作机制概述

为了使读者能够理解将数据储存到 Flash 所进行的每个步骤的意义，在此先对 FIM 的运作机制进行概述

图表 3: 运作机制概述

Information Table

索引值	要創建的實體數	數據結構大小	初始化樣板實體	儲存各個編號實體的address用的數組
MW_FIM_IDX_GP01_UART_CFG	MW_FIM_UART_CFG_NUM	MW_FIM_UART_CFG_SIZE	g_tMwFimDefaultUartConfig	g_ulaMwFimAddrBufferUartConfig
MW_FIM_IDX_GP01_TRACER_CFG	MW_FIM_TRACER_CFG_NUM	MW_FIM_TRACER_CFG_SIZE	NULL	g_ulaMwFimAddrBufferTracerConfig
MW_FIM_IDX_GP01_TRACER_INT_TASK_INFO	MW_FIM_TRACER_INT_TASK_INFO_NUM	MW_FIM_TRACER_INT_TASK_INFO_SIZE	NULL	g_ulaMwFimAddrBufferTracerIntTaskInfo
MW_FIM_IDX_GP01_TRACER_EXT_TASK_INFO	MW_FIM_TRACER_EXT_TASK_INFO_NUM	MW_FIM_TRACER_EXT_TASK_INFO_SIZE	NULL	g_ulaMwFimAddrBufferTracerExtTaskInfo

在 FIM 中对于 Flash 的读写操作都是围绕着这张 Information Table 去进行的，每次的读写首先会到这张 Information Table 根据索引值去找到相应的 entry。再根据"储存 address 用的数组"去找到要读写的指定编号的实体的 address。

接着以一个体重机的应用，使用 Group8 来储存应用所需的配置讯息的例子来说明 FIM 的使用。假设在这个应用中有一个三段切换式的按钮，让体重机可以记录 3 个不同用户的使用习惯，按钮切换到不同人时，体重机初始化阶段会加载的初始化参数也不相同。纪录初始化参数的数据结构中包含 User ID, Unit of weight, Height, Age。

Step 1. 设定 File ID

File ID 就是 Information Table 中索引值字段的内容。因为一个 Group 里面能够储存多种不同的 Files，每一个 File ID 将用来唯一识别某一种数据结构建立的实体。以体重机为例，可以将 File ID 取名为 MW_FIM_IDX_GP08_Weight_CFG ， 在 实 作 上 可 以 将 下 列 代 码 定 义 在 APS_PATCH\middleware\netlink\mw_fim\ mw_fim_default_group08.h

```
typedef enum
{
    MW_FIM_IDX_GP08_START = 0x00080000,              // the start IDX of group 08
    MW_FIM_IDX_GP08_Weight_CFG,

    MW_FIM_IDX_GP08_MAX
} E_MwFimIdxGroup08;
```

Step 2. 在用戶的 application.h 下定义用戶要储存到 Flash 的数据的数据结构。在体重机的这个例子中，定义的数据结构如下:

```
typedef struct
{
    uint8_t userID;
    uint8_t unit_of_weight;
    uint16_t height;
    uint8_t age;
} T_WeightConfig;
```

并在 APS_PATCH\middleware\netlink\mw_fim\mw_fim_default_group08.h 中定义要存到 Flash 中的数据的数据结构长度与要存入的实体数

```
#define MW_FIM_WEIGHT_CFG_SIZE    sizeof(T_WeightConfig)
#define MW_FIM_WEIGHT_CFG_NUM     3
```

Step 3. 基于 step2 的数据结构产生一个初始化用的实体样板，每当呼叫 MwFim_FileWriteDefault()函数时将会以此实体样板对指定实体进行赋值。在这步骤中也去定义实体编号寻址所需使用的存放地址用的数组，在体重机的例子中将以下内容定义在 APS_PATCH\middleware\netlink\mw_fim\ mw_fim_default_group08.c

```
const T_WeightConfig g_tMwFimDefaultWeightConfig =
{
    Dad,
    KG,
    175,
    30,
};

uint32_t g_ulaMwFimAddrBufferWeightConfig[MW_FIM_WEIGHT_CFG_NUM];
```

Step 4. 创建 FIM 操作所需要的 information table，各字段内容的作用如"FIM 的运作机制概述"中所示

此例中将以下代码定义在 APS_PATCH\middleware\netlink\mw_fim\ mw_fim_default_group08.c 中

```
const T_MwFimFileInfo g_taMwFimGroupTable08[] =
{
    {MW_FIM_IDX_GP08_WEIGHT_CFG, MW_FIM_WEIGHT_CFG_NUM, MW_FIM_WEIGHT_CFG_SIZE, (uint8_t*)&g_tMwFimDefaultWeightConfig, g_ulaMwFimAddrBufferWeightConfig},

    // the end, don't modify and remove it
    {0xFFFFFFFF,            0x00,            0x00,            NULL,                        NULL}
};
```

Step 5. 修改 Group version number，Group version number 被 FIM 用来判定 Flash 中储存的内容是否有更改，如果 Group version number 不同才会对 Flash 内容进行更新，在此例中使用的是 Group8 因此要将对应的 MW_FIM_VER08 的值修改为原本的值+1

```
#define MW_FIM_VER00        0x01    // reserve for swap
#define MW_FIM_VER01        0x04    // system & driver
#define MW_FIM_VER02        0x04    //for WIFI
#define MW_FIM_VER03        0x03    // calibration data
#define MW_FIM_VER04        0x02    // for LE Controller
#define MW_FIM_VER05        0x01
#define MW_FIM_VER06        0x01
#define MW_FIM_VER07                // For BLE
#define MW_FIM_VER08        0x02
```

到此就成功的将三份 T_WeightConfig 数据结构的实体储存到 Flash 中 Group8，之后就能按照 5.2 所述对这三笔实体进行读写。

5.4. FIM 初始参数值

Data Structure 定义元素的 default 值会贴在列表下方 ,NULL 代表没有定义初始化用的样板，其余 Default value 在()内

5.4.1. System & Driver (Group1)

ulFileId	uwRecIdx	uwFileSize	Data Structure (Default)
MW_FIM_IDX_GP01_UART_CFG	0~1	MW_FIM_UART_CFG_SIZE	T_HalUartConfig（下方）

ulFileId	uwRecIdx	uwFileSize	Data Structure (Default)
MW_FIM_IDX_GP01_TRACER_CFG	0	MW_FIM_TRACER_CFG_SIZE	T_TracerCfg (Null)
MW_FIM_IDX_GP01_TRACER_INT_TASK_INFO	0~31	MW_FIM_TRACER_INT_TASK_INFO_SIZE	T_TracerTaskInfo (Null)
MW_FIM_IDX_GP01_TRACER_EXT_TASK_INFO	0~31	MW_FIM_TRACER_EXT_TASK_INFO_SIZE	T_TracerTaskInfo (Null)

```
typedef struct
{
    uint32_t ulBuadrate;
    uint8_t ubDataBit;
    uint8_t ubStopBit;
    uint8_t ubParity;
    uint8_t ubFlowCtrl;
} T_HalUartConfig;
```

```
const T_HalUartConfig g_tMwFimDefaultUartConfig =
{
    115200,
    DATA_BIT_8,
    STOP_BIT_1,
    PARITY_NONE,
    0                   // disable the flow control
};
```

```
typedef struct
{
    uint8_t bMode;
    uint8_t bExtTaskDefLevel;
    uint8_t bNameDisplay;
    uint8_t bPadding;
    int iPriority;
    uint32_t dwStackSize;
    uint32_t dwQueueNum;
    uint32_t dwQueueSize;
} T_TracerCfg;
```

```
typedef struct
{
    char baName[TRACER_TASK_NAME_LEN];
    uint8_t bLevel;
    uint8_t bStatus;
    uint8_t baPadding[2];
} T_TracerTaskInfo;
```

5.4.2. WIFI (Group2)

ulFileId	uwRecIdx	uwFileSize	Data Structure (Default)
MW_FIM_IDX_WIFI_AUTO_CONN_MODE	0	MW_FIM_AUTO_COMM_MODE_SIZE	uint32_t (true)

ulFileId	uwRecIdx	uwFileSize	Data Structure (Default)
MW_FIM_IDX_WIFI_AUTO_CONN_CFG	0	MW_FIM_IDX_WIFI_AUTO_CONN_CFG_SIZE	MwFimAutoConnectCFG_t (下方)
MW_FIM_IDX_WIFI_AUTO_CONN_AP_NUM	0	MW_FIM_AUTO_COMM_AP_SIZE	uint32_t (0)
MW_FIM_IDX_WIFI_AUTO_CONN_AP_INFO	0~2	MW_FIM_AUTO_CONN_INFO_SIZE	mw_wifi_auto_connect_ap_info_t (下方)
MW_FIM_IDX_STA_MAC_ADDR	0	MW_FIM_STA_MAC_ADDR_SIZE	uint8_t array[6] (0x22, 0x33, 0x44, 0x55, 0x66, 0x76)
MW_FIM_IDX_STA_SKIP_DTIM	0	MW_FIM_STA_SKIP_DTIM_SIZE	uint8_t (0)
MW_FIM_IDX_DEVICE_MANUF_NAME	0	MW_FIM_DEVICE_MANUF_NAME_SIZE	uint8_t array[32] (0x43, 0x2E, 0x42, 0x2E, 0x53, 0x20, 0xB4, 0xB4, 0xB2, 0xA9,0xCA, 0xC0)

```
typedef struct
{
    s8      front;
    s8      rear;
    bool    flag;
    u8      targetIdx;
    u8      max_save_num;
} MwFimAutoConnectCFG_t;

const MwFimAutoConnectCFG_t gMwFimDefaultAutoConnectCfg = {
    .front = -1,
    .rear = -1,
    .flag = false,
    .targetIdx = 0,
    .max_save_num = MAX_NUM_OF_AUTO_CONNECT,
};
```

```
typedef struct
{
#if 1
    bool            free_ocpy;                          //scan info buffer is free or occupied, 0:free, 1:occupied
    u8              bssid[MAC_ADDR_LEN];                /* BSS ID - 48 bit HW address */
    u8              ap_channel;                         /* Which Channel */
    u64             latest_beacon_rx_time;              /* Timestamp - Last interaction with BSS */
    s8              ssid[IEEE80211_MAX_SSID_LEN + 1];   /* SSID of the BSS - 33 bytes */
    u8              supported_rates[SUPPORTED_RATES_MAX];
    s8              rssi;                               /* Last observed Rx Power (dBm) */
    u16             beacon_interval;                    /* Beacon interval - In time units of 1024 us */
    u16             capabilities;                       /* Supported capabilities */
    u8              dtim_prod;                          //DTIM Period

    wpa_ie_data_t wpa_data;
    u8            rsn_ie[100];
    u8            wpa_ie[100];
    s8            passphrase[64];          /* maximum number of passphrase is 64 bytes */
    s8            hid_ssid[IEEE80211_MAX_SSID_LEN + 1];    /* [APS write/MSQ read] Hidden SSID of the BSS. When ssid is null, using this field. */
    u8            psk[32];
    u8            fast_connect;
#else
    u8 bssid[6];
    u8 ap_channel;
    u8 fast_connect;
    s8 ssid[IEEE80211_MAX_SSID_LEN + 1];
    u8 psk[32];
    wpa_ie_data_t wpa_data;
    u16 capabilities;
    u8 rsn[100];
#endif
} ? end {anonmw_wifi_auto_connect_ap_info_t} ? mw_wifi_auto_connect_ap_info_t;

const mw_wifi_auto_connect_ap_info_t gMwFimDefaultAutoConnectAPInfo = {
    .free_ocpy = 0,
    .bssid = {0},
    .ap_channel = 0,
    .latest_beacon_rx_time = 0,
    .ssid = {0},
    .supported_rates = {0},
    .rssi = 0,
    .beacon_interval = 0,
    .capabilities = 0,
    .dtim_prod = 0,
    .wpa_data = {0},
    .rsn_ie = {0},
    .wpa_ie = {0},
    .passphrase = {0},
    .hid_ssid = {0},
    .psk = {0},
    .fast_connect = false,
};
```

5.4.3. Calibration Data (Group3)

ulFileId	uwRecIdx	uwFileSize	Data Structure (Default)
MW_FIM_IDX_GP03_CAL_AUXADC	0	MW_FIM_CAL_AUXADC_SIZE	T_HalAuxCalData（下方）
MW_FIM_IDX_GP03_CAL_TEMPERATURE	0	MW_FIM_CAL_TEMPERATURE_SIZE	T_HalTmprCalData (下方)

```
typedef struct
{
    float fSlopeVbat;
    float fSlopeIo;
    uint16_t uwDcOffsetVbat;            // 0V
    uint16_t uwDcOffsetIo;              // 0V
} T_HalAuxCalData;

const T_HalAuxCalData g_tMwFimDefaultCalAuxadc =
{
    0.003,      // float fSlopeVbat;
    0.003,      // float fSlopeIo;
    100,        // uint16_t uwDcOffsetVbat;      // 0V
    100         // uint16_t uwDcOffsetIo;        // 0V
};

typedef struct
{
    float fBaseTemperature;                 // 25
    float faThermistor[HAL_TMPR_STEP_MAX];  // 25 ~ 48
    float fVolDivResistor;                  // Voltage divider resistor
} T_HalTmprCalData;

const T_HalTmprCalData g_tMwFimDefaultCalTmpr =
{
    25.0,       // float fBaseTemperature;          // 25
    {           // float faThermistor[HAL_TMPR_STEP_MAX];  // 25 ~ 48
        50.0000, 47.8597, 45.8223, 43.8823,
        42.0346, 40.2400, 38.5600, 36.9500,
        35.4300, 33.9800, 32.5900, 31.2700,
        30.0100, 28.8100, 27.6700, 26.5800,
        25.5200, 24.5100, 23.5500, 22.6400,
        21.7600, 20.9473, 20.1454, 19.3781
    },
    30.0        // float fVolDivResistor;           // Voltage divider resi
};
```

5.4.4. LE Controller (Group4)

ulFileId	uwRecIdx	uwFileSize	Data Structure (Default)
MW_FIM_IDX_LE_CFG	0	MW_FIM_IDX_LE_CFG_SIZE	le_cfg_t (下方)

```
typedef struct
{
    uint8_t bd_addr[6];
} le_cfg_t;

const le_cfg_t g_tMwFimDefaultLeCfg =
{
    .bd_addr = {0x66, 0x55, 0x44, 0x33, 0x22, 0x11}
};
```

5.4.5. BLE (Group7)

ulFileId	uwRecIdx	uwFileSize	Data Structure
MW_FIM_IDX_LE_STORE0	0~3	MW_FIM_LE_STORE0_SIZE	uint32_t array[4] (NULL)
MW_FIM_IDX_LE_STORE1	0	MW_FIM_LE_STORE1_SIZE	uint32_t array[1] (NULL)
MW_FIM_IDX_LE_STORE2	0~7	MW_FIM_LE_STORE2_SIZE	uint32_t array[8] (NULL)
MW_FIM_IDX_LE_STORE3	0~15	MW_FIM_LE_STORE3_SIZE	uint32_t array[16] (NULL)

CONTACT

sales@Opulinks.com