

AT32系列 flash application note

前言

本文档主要介绍 AT32 系列 MCU flash 具体操作说明，该文档包含了：flash 常规功能介绍；flash 特殊功能简介；断电时利用 PVM 及时写入保存资料的技巧。文档目标是让用户能够更全面的了解 Artery AT32 系列的 flash 相关操作。

该文档主要内容有：

- flash 常规功能介绍
- flash 特殊功能简介
- 断电时利用 PVM 及时写入保存资料技巧
- bsp flash 例程介绍

参考资料：

- AT32F4xx_Firmware_Library_V2.x.x\project\at_start_f4xx\examples\flash
- RM_AT32 系列文档的闪存控制器 FLASH 章节

支持型号列表：

支持型号	AT32 全系列
------	----------

目录

1	概述.....	6
2	结构介绍	7
2.1	存储器部分	7
2.2	信息块部分	7
2.2.1	启动程序代码区	7
2.2.2	用户系统数据区	8
3	特殊功能	11
3.1	安全库区（SLib）	11
3.2	启动程序代码区域作为主存扩展使用	11
4	断电时利用 PVM 及时写入保存资料的技巧	12
4.1	功能描述	12
4.2	软件配置	13
5	flash 案例	14
5.1	案例 1--访问保护开启	14
5.1.1	功能简介	14
5.1.2	资源准备	14
5.1.3	软件设计	14
5.1.4	实验效果	15
5.2	案例 2--主存储器读写	15
5.2.1	功能简介	15
5.2.2	资源准备	15
5.2.3	软件设计	15
5.2.4	实验效果	18
5.3	案例 3--外部存储器读写	18

5.3.1	功能简介	18
5.3.2	资源准备	18
5.3.3	软件设计	18
5.3.4	实验效果	20
5.4	案例 4--程序运行在外部存储器	21
5.4.1	功能简介	21
5.4.2	资源准备	21
5.4.3	软件设计	21
5.4.4	实验效果	22
5.5	案例 5--断电时利用 PVM 写入保存资料到 flash	23
5.5.1	功能简介	23
5.5.2	资源准备	23
5.5.3	软件设计	23
5.5.4	实验效果	26
6	版本历史	27

表目录

表 1. 存储结构	7
表 2. 用户系统数据区结构	8
表 3. 文档版本历史	27

图目录

图 1. 加电容前 VDD 掉电时间和 flash 写入时间对比	12
图 2. 加电容后 VDD 掉电时间和 flash 写入时间对比	13

1 概述

AT32 MCU内嵌flash根据型号的不同，主要有两种架构类型，一种串行架构flash，使用中根据地址空间范围分为零等待区（ZW）和非零等待区（NZW），另一种是并行架构flash，地址范围内效能相同，需根据系统频率设置等待周期（Wait cycle）。两种架构类型flash各有特点，用户根据应用及MCU型号选择适合的。

2 结构介绍

AT32 MCU flash主要由存储器和信息块两大部分组成。

- 存储器分为内部存储器和外部存储器（有些型号支持外部存储器，例如AT32F403A），是存储用户数据及代码的地方；
- 信息块分为启动程序代码区（boot memory）和用户系统数据区（user system data），启动程序代码区存放的是出厂烧录好的bootloader固件，用户系统数据区存放一些特殊功能的配置数据。

下表以AT32F403AxG系列为例，展示flash存储结构

表 1. 存储结构

结构	名称	地址范围
主存储器	扇区 0	0x0800 0000 – 0x0800 07FF
	扇区 1	0x0800 0800 – 0x0800 0FFF
	扇区 2	0x0800 1000 – 0x0800 17FF

	扇区 255	0x0807 F800 – 0x0807 FFFF
	扇区 256	0x0808 0000 – 0x0808 07FF
	扇区 257	0x0808 0800 – 0x0808 0FFF
	扇区 258	0x0808 1000 – 0x0808 17FF

	扇区 511	0x080F F800 – 0x080F FFFF
外部存储器	扇区 0	0x0840 0000 – 0x0840 0FFF
	扇区 1	0x0840 1000 – 0x0840 1FFF
	扇区 2	0x0840 2000 – 0x0840 2FFF

	扇区 4095	0x093F F000 – 0x093F FFFF
信息块	启动程序代码区 16KB	0x1FFF B000 – 0x1FFF EFFF
	用户系统数据区 48B	0x1FFF F800 – 0x1FFF F82F

2.1 存储器部分

存储器读操作直接对指定地址寻址即可，擦除或者编程操作首先需要对指定地址区域解锁。

擦除操作的最小单元为一个扇区，扇区大小根据MCU型号及flash容量的不同而有差异。

详细的操作步骤及说明，可以查看各个型号MCU的RM参考手册的FLASH章节。

2.2 信息块部分

2.2.1 启动程序代码区

该区域为出厂时烧录的bootloader代码，程序通过各种外设通信对flash进行各种操作。

关于bootloader的详细介绍，可以参考bootloader相关的文档，例如

PM0005_AT32_Bootloader_UART_Protocol、PM0007_AT32_Bootloader_Program_Manual等。

2.2.2 用户系统数据区

该区域存储的是配置特殊功能的数据，例如访问保护、擦写保护、内存扩展等，只能对整个区域进行擦除，并且修改后新的配置需要系统复位才能重新加载。

详细的操作步骤及说明，可以查看各个型号MCU的RM参考手册的FLASH章节。

下表以AT32F403AxG系列为例，展示用户系统数据区结构

表 2. 用户系统数据区结构

地址	位	内容
0x1FFF_F800	[7: 0]	FAP[7: 0]: 闪存访问保护（访问保护启动/解除结果存放在用户系统数据寄存器（FLASH_USD）[1]） 0xA5: 闪存访问保护解除 其他值: 闪存访问保护启动
	[15: 8]	nFAP[7: 0]: FAP[7: 0]的反码
	[23: 16]	SSB[7: 0]: 系统配置字节（存放在用户系统数据寄存器（FLASH_USD）[9: 2]）
		位 7: 4
		保留不用
		位 3 (BTOPT)
		0: 当配置从主闪存启动时，若片 2 中没有启动程序，将从片 1 启动，否则，从片 2 启动 1: 当配置从主闪存启动时，从片 1 启动
		位 2 (nSTDBY_RST)
		0: 进入待机模式时产生复位 1: 进入待机模式时不产生复位
		位 1 (nDEPSLP_RST)
		0: 进入深度睡眠模式时产生复位 1: 进入深度睡眠模式时不产生复位
		位 0 (nWDT_ATO_EN)
		0: 看门狗自启动开启 1: 看门狗自启动关闭
	[31: 24]	nSSB[7: 0]: SSB[7: 0]的反码
0x1FFF_F804	[7: 0]	Data0[7: 0]: 用户数据 0（存放在用户系统数据寄存器（FLASH_USD）[17: 10]）
	[15: 8]	nData0[7: 0]: Data0[7: 0]的反码
	[23: 16]	Data1[7: 0]: 用户数据 1（存放在用户系统数据寄存器（FLASH_USD）[25: 18]）
	[31: 24]	nData1[7: 0]: Data1[7: 0]的反码
0x1FFF_F808	[7: 0]	EPP0[7: 0]: 闪存擦写保护字节 0（存放在擦除编程保护状态寄存器（FLASH_EPPS）[7: 0]） 用于保护主闪存存储器的扇区 0 ~ 扇区 15，每个比特位保护 2 个扇区（2K 字节/扇区） 0: 擦写保护启动 1: 擦写保护解除
	[15: 8]	nEPP0[7: 0]: EPP0[7: 0]的反码
	[23: 16]	EPP1[7: 0]: 闪存擦写保护字节 1（存放在擦除编程保护状态寄存器（FLASH_EPPS）[15: 8]） 用于保护主闪存存储器的扇区 16 ~ 扇区 31，每个比特位保护 2 个扇区（2K 字节/扇区）

		0: 擦写保护启动 1: 擦写保护解除
	[31: 24]	nEPP1[7: 0]: EPP1[7: 0]的反码
0x1FFF_F80C	[7: 0]	EPP2[7: 0]: 闪存擦写保护字节 2 (存放在擦除编程保护状态寄存器 (FLASH_EPPS) [23: 16]) 用于保护主闪存存储器的扇区 32 ~ 扇区 47, 每个比特位保护 2 个扇区 (2K 字节/扇区) 0: 擦写保护启动 1: 擦写保护解除
	[15: 8]	nEPP2[7: 0]: EPP2[7: 0]的反码
	[23: 16]	EPP3[7: 0]: 闪存擦写保护字节 3 (存放在擦除编程保护状态寄存器 (FLASH_EPPS) [31: 24]) 其中位 6: 0 用于保护主闪存存储器的扇区 48 ~ 扇区 61, 每个比特位保护 2 个扇区 (2K 字节/扇区) 位 7 用于保护主闪存存储器的扇区 62 及之后的扇区, 以及外部存储器 0: 擦写保护启动 1: 擦写保护解除
	[31: 24]	nEPP3[7: 0]: EPP3[7: 0]的反码
0x1FFF_F810	[7: 0]	EOPB0[7: 0]: 扩充的系统选项 位 7: 1: 保留不用 位 0: 0: 片上 SRAM 224K 字节 1: 片上 SRAM 96K 字节 注意: 由 1 改写成 0 只能在安全库区未启动的情况下实现
	[15: 8]	nEOPB0[7: 0]: EOPB0[7: 0]的反码
	[31: 16]	保留不用
0x1FFF_F814	[7: 0]	Data2[7: 0]: 用户数据 2
	[15: 8]	nData2[7: 0]: Data2[7: 0]的反码
	[23: 16]	Data3[7: 0]: 用户数据 3
	[31: 24]	nData3[7: 0]: Data3[7: 0]的反码
0x1FFF_F818	[7: 0]	Data4[7: 0]: 用户数据 4
	[15: 8]	nData4[7: 0]: Data4[7: 0]的反码
	[23: 16]	Data5[7: 0]: 用户数据 5
	[31: 24]	nData5[7: 0]: Data5[7: 0]的反码
0x1FFF_F81C	[7: 0]	Data6[7: 0]: 用户数据 6
	[15: 8]	nData6[7: 0]: Data6[7: 0]的反码
	[23: 16]	Data7[7: 0]: 用户数据 7
	[31: 24]	nData7[7: 0]: Data7[7: 0]的反码
0x1FFF_F820	[7: 0]	EXT_FLASH_KEY0[7: 0]: 外部存储器密文存取区加密键值字节 0 不加密的设定条件包括: EXT_FLASH_KEYx 以及 nEXT_FLASH_KEYx 均为 0xFF (即默认擦除状态) EXT_FLASH_KEYx 写入 0x00 EXT_FLASH_KEYx 写入 0xFF

		即{nEXT_FLASH_KEYx, EXT_FLASH_KEYx }均设为 0xFFFF, 0xFF00, 0x00FF
	[15: 8]	nEXT_FLASH_KEY0[7: 0]: EXT_FLASH_KEY0[7: 0]的反码
	[23: 16]	EXT_FLASH_KEY1[7: 0]: 外部存储器密文存取区加密键值字节 1
	[31: 24]	nEXT_FLASH_KEY1[7: 0]: EXT_FLASH_KEY1[7: 0]的反码
0x1FFF_F824	[7: 0]	EXT_FLASH_KEY2[7: 0]: 外部存储器密文存取区加密键值字节 2
	[15: 8]	nEXT_FLASH_KEY2[7: 0]: EXT_FLASH_KEY2[7: 0]的反码
	[23: 16]	EXT_FLASH_KEY3[7: 0]: 外部存储器密文存取区加密键值字节 3
	[31: 24]	nEXT_FLASH_KEY3[7: 0]: EXT_FLASH_KEY3[7: 0]的反码
0x1FFF_F828	[7: 0]	EXT_FLASH_KEY4[7: 0]: 外部存储器密文存取区加密键值字节 4
	[15: 8]	nEXT_FLASH_KEY4[7: 0]: EXT_FLASH_KEY4[7: 0]的反码
	[23: 16]	EXT_FLASH_KEY5[7: 0]: 外部存储器密文存取区加密键值字节 5
	[31: 24]	nEXT_FLASH_KEY5[7: 0]: EXT_FLASH_KEY5[7: 0]的反码
0x1FFF_F82C	[7: 0]	EXT_FLASH_KEY6[7: 0]: 外部存储器密文存取区加密键值字节 6
	[15: 8]	nEXT_FLASH_KEY6[7: 0]: EXT_FLASH_KEY6[7: 0]的反码
	[23: 16]	EXT_FLASH_KEY7[7: 0]: 外部存储器密文存取区加密键值字节 7
	[31: 24]	nEXT_FLASH_KEY7[7: 0]: EXT_FLASH_KEY7[7: 0]的反码

3 特殊功能

AT32 MCU flash有不少特殊扩展功能，用于增加MCU的使用场景。这些特殊功能可能在各个型号上表现有差异，具体的需参考对应型号的文档介绍。下面列举几个特殊功能的简单介绍

3.1 安全库区（SLib）

设定以密码保护主存中指定范围的程式区，即安全库区，此区域仅能被执行，无法读取（I-Code, D-Code总线除外），以及写入与删除，除非输入指定密码。

设定安全库区的益处：

- 以密码保护安全库区,方案商可刻录核心算法到此区域；
- 安全库区仅能执行，无法被读取，除非输入方案商指定密码，也无法删除(包含ISP/IAP/SWD)；
- 其余空白程序区可以提供给方案商客户进行二次开发；
- 方案商可以藉由安全库功能销售核心算法，不需要每个客户都开发完整方案；
- 设定安全库区，可防止蓄意破坏或更改终端产品应用程序代码。

更多关于安全库区的介绍，可以参考各个型号关于安全库区的应用笔记，例如
AN0040_AT32F403A_407_Security_Library_Application_Note

3.2 启动程序代码区域作为主存扩展使用

对于有存储器扩充需求，并且不需要出厂时烧录的bootloader程序的用户，该功能可以提供一种方式用户只有一次机会将启动程序代码区域作为主存扩展使用。一旦设定成功，主存扩展区将具有主闪存特性。

关于该功能详细介绍可以参考BSP里utilities文件夹的boot_memory_ap_demo和应用笔记
AN0066_config_boot_memory_as_extension_of_main_memory(AP_mode)

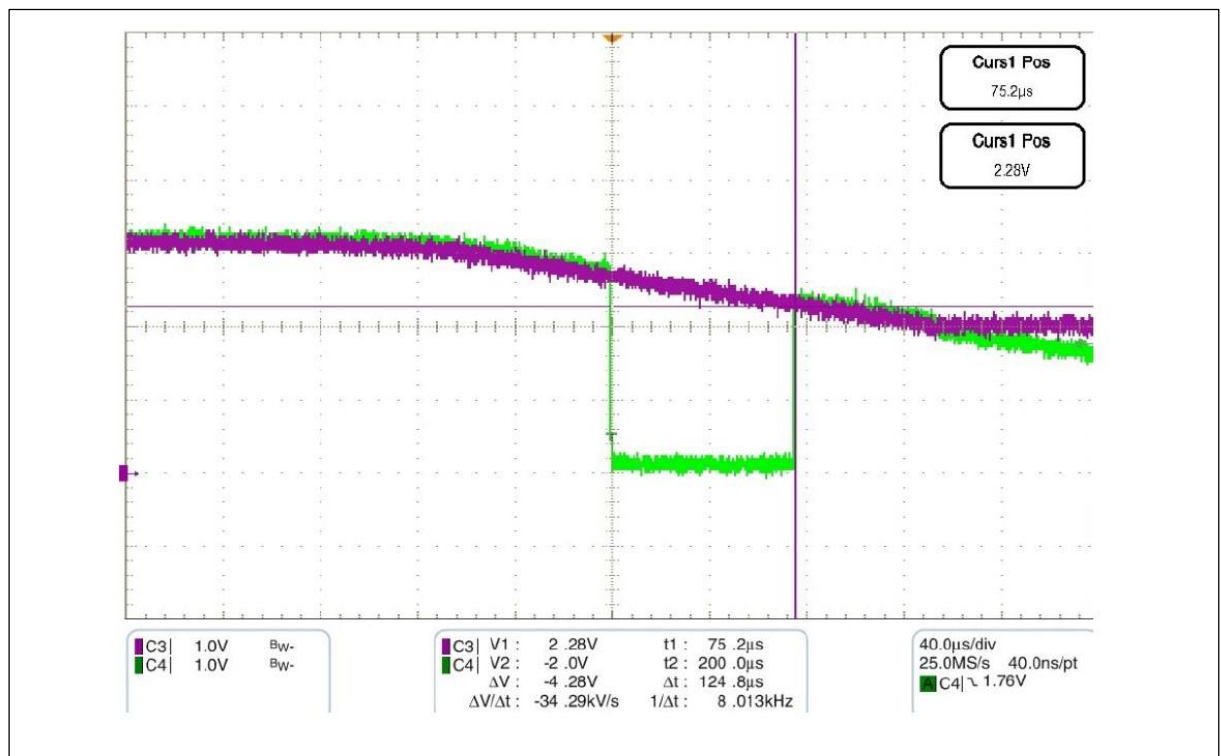
4 断电时利用 PVM 及时写入保存资料的技巧

4.1 功能描述

利用flash掉电不丢失数据特性，通过PVM检测电压变化，当掉电发生时，PVM检测到设置的电压阈值可触发中断，利用VDD掉电从阈值到LVR电压（大约2.2V）的时间，将部分数据存储在flash中。以AT-START-F403A开发板为例，设置PVM阈值为2.9V，MCU主频为240 MHz。当3.3V电源掉电触发PVM事件进入中断后，马上将32字节的保存资料写入FLASH内。

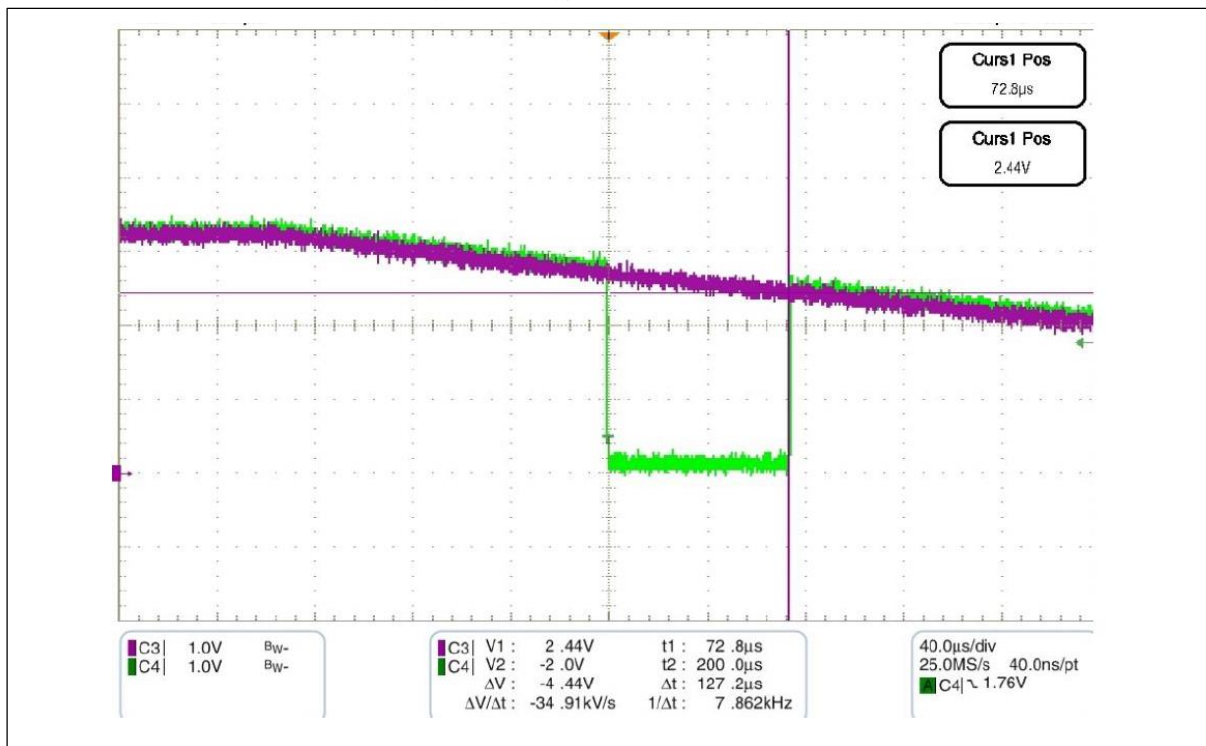
实测如下图：紫色为VDD曲线，绿色低电平区段为flash写入时间（在开始写入flash时使用GPIO输出设置低电平，完成32字节写入后设置高电平）。图中可以看到flash写完时VDD电压大约2.28V，离LVR电压（最大值2.2v）较近，考虑电源下电和LVR电压变化，可能有写入32字节未完成芯片就重置的风险。

图 1. 加电容前 VDD 掉电时间和 flash 写入时间对比



若在3.3V电源再增加4.7 μF电容，32字节保存资料写完时，VDD电压大约2.44V，离LVR的时间比较充裕（测试图像如下）。以上测试是以AT-START-F403A开发板为例，使用者可使用上述方法调试自己的硬件环境，配合写入数据量大小找出适当的电源电容值。

图 2. 加电容后 VDD 掉电时间和 flash 写入时间对比



4.2 软件配置

软件操作大致流程：

1. 提前执行flash擦除（若客户想在掉电时，保存相应数据，该步骤必不可少，因为flash擦除太过耗费时间，若在掉电期间再执行flash擦除动作，没有充足时间来完成flash写操作）；
2. 配置PVM中断相关；
3. 利用PVM检测到阈值电压，产生中断进入PVM_IRQHandler()函数，在中断函数内对重要数据进行写入保存。

具体软件配置可参考“案例5--断电时利用PVM写入保存资料到flash”。

5 flash 案例

因为各个型号MCU的flash可能略有差异，本文以AT32F403A的demo为例进行说明

注：所有project都是基于keil 5而建立，若用户需要在其他编译环境上使用，请参考

AT32xxx_Firmware_Library_V2.x.x\project\at_start_xxx\templates中各种编译环境（例如IAR6/7,keil 4/5）进行简单修改即可。

5.1 案例 1--访问保护开启

5.1.1 功能简介

通过程序代码判断当前访问保护状态，并配置为开启状态，开启成功后三个LED点亮。

5.1.2 资源准备

1) 硬件环境：

AT32F403A 的 AT-START BOARD

2) 软件环境：

\project\at_start_f403a\examples\flash\fap_enable\mdk_v5

5.1.3 软件设计

1) 配置流程

- 配置时钟、AT-START BOARD 板子的初始化；
- 程序判断当前芯片 FAP 功能是否开启，如果开启则点亮三个 LED，如果没开启则配置为开启状态并执行系统复位；

2) 代码介绍

■ 主函数代码描述

```
if(flash_fap_status_get() == RESET)
{
    flash_unlock();
    /* wait for operation to be completed */
    status = flash_operation_wait_for(OPERATION_TIMEOUT);

    if(status != FLASH_OPERATE_TIMEOUT)
    {
        if((status == FLASH_PROGRAM_ERROR) || (status == FLASH_EPP_ERROR))
            flash_flag_clear(FLASH_PRGMERR_FLAG | FLASH_EPPERR_FLAG);

        status = flash_fap_enable(TRUE);
        if(status == FLASH_OPERATE_DONE)
            nvic_system_reset();
    }
}
else
```

```
{  
    at32_led_on(LED2);  
    at32_led_on(LED3);  
    at32_led_on(LED4);  
}
```

5.1.4 实验效果

AT-START 的板载 LED 点亮，说明开启访问保护成功，符合程序设计预期。

5.2 案例 2--主存储器读写

5.2.1 功能简介

程序代码执行主存储器某段地址的擦除、编程，并读取回该段数据进行比较是否正确，如果正确则点亮三个LED。

5.2.2 资源准备

1) 硬件环境：

AT32F403A 的 AT-START BOARD

2) 软件环境：

\\project\\at_start_f403a\\examples\\flash\\flash_write_read\\mdk_v5

5.2.3 软件设计

1) 配置流程

- 配置时钟、AT-START BOARD 板子的初始化；
- 指定地址段写入数据，步骤包括解锁、擦除、编程
- 读取回该指定地址段的数据，如果是正确的则点亮三个 LED

2) 代码介绍

■ Flash 写入代码描述

```
uint32_t offset_addr;  
uint32_t sector_position;  
uint16_t sector_offset;  
uint16_t sector_remain;  
uint16_t i;  
flash_status_type status = FLASH_OPERATE_DONE;  
  
flash_unlock();  
offset_addr = write_addr - FLASH_BASE;  
sector_position = offset_addr / SECTOR_SIZE;
```

```
sector_offset = (offset_addr % SECTOR_SIZE) / 2;
sector_remain = SECTOR_SIZE / 2 - sector_offset;
if(num_write <= sector_remain)
    sector_remain = num_write;
while(1)
{
    flash_read(sector_position * SECTOR_SIZE + FLASH_BASE, flash_buf, SECTOR_SIZE / 2);
    for(i = 0; i < sector_remain; i++)
    {
        if(flash_buf[sector_offset + i] != 0xFFFF)
            break;
    }
    if(i < sector_remain)
    {
        /* wait for operation to be completed */
        status = flash_operation_wait_for(ERASE_TIMEOUT);

        if((status == FLASH_PROGRAM_ERROR) || (status == FLASH_EPP_ERROR))
            flash_flag_clear(FLASH_PRGMERR_FLAG | FLASH_EPPERR_FLAG);
        else if(status == FLASH_OPERATE_TIMEOUT)
            return ERROR;
        status = flash_sector_erase(sector_position * SECTOR_SIZE + FLASH_BASE);
        if(status != FLASH_OPERATE_DONE)
            return ERROR;
        for(i = 0; i < sector_remain; i++)
        {
            flash_buf[i + sector_offset] = p_buffer[i];
        }
        if(flash_write_noccheck(sector_position * SECTOR_SIZE + FLASH_BASE, flash_buf,
SECTOR_SIZE / 2) != SUCCESS)
            return ERROR;
    }
    else
    {
        if(flash_write_noccheck(write_addr, p_buffer, sector_remain) != SUCCESS)
            return ERROR;
    }
    if(num_write == sector_remain)
```

```
        break;
    else
    {
        sector_position++;
        sector_offset = 0;
        p_buffer += sector_remain;
        write_addr += (sector_remain * 2);
        num_write -= sector_remain;
        if(num_write > (SECTOR_SIZE / 2))
            sector_remain = SECTOR_SIZE / 2;
        else
            sector_remain = num_write;
    }
}
flash_lock();
return SUCCESS;
```

■ 主函数代码描述

```
/* fill buffer_write data to test */
for(index = 0; index < TEST_BUFEER_SIZE; index++)
{
    buffer_write[index] = index;
}

/* write data to flash */
err_status = flash_write(TEST_FLASH_ADDRESS_START, buffer_write, TEST_BUFEER_SIZE);

/* read data from flash */
flash_read(TEST_FLASH_ADDRESS_START, buffer_read, TEST_BUFEER_SIZE);

/* compare the buffer */
if((buffer_compare(buffer_write, buffer_read, TEST_BUFEER_SIZE) == SUCCESS) && (err_status == SUCCESS))
{
    at32_led_on(LED2);
    at32_led_on(LED3);
    at32_led_on(LED4);
}
```

5.2.4 实验效果

AT-START 的板载 LED 点亮，说明读写成功，符合程序设计预期。

5.3 案例 3--外部存储器读写

5.3.1 功能简介

程序代码执行外部存储器某段地址的擦除、编程，并读取回该段数据进行比较是否正确，如果正确则点亮三个LED。

5.3.2 资源准备

1) 硬件环境：

AT32F403A 的 AT-START BOARD

2) 软件环境：

\\project\\at_start_f403a\\examples\\flash\\operate_spim\\mdk_v5

5.3.3 软件设计

1) 配置流程

- 配置时钟、AT-START BOARD 板子的初始化；
- 指定地址段解锁、擦除、编程
- 读取回该指定地址段的数据，如果是正确的则点亮三个 LED

2) 代码介绍

- 外部存储器操作代码描述

```
uint16_t i = 0;

flash_status_type status = FLASH_OPERATE_DONE;

/* configures the spim flash */
spim_init();

/* fill the content to be writed to spim flash */
for(i = 0; i < SPIM_SECTOR_SIZE; i++)
{
    write_buffer[i] = i % 256;
}

/* wait for operation to be completed */
status = flash_operation_wait_for(ERASE_TIMEOUT);
```

```
if((status == FLASH_PROGRAM_ERROR) || (status == FLASH_EPP_ERROR))
    flash_flag_clear(FLASH_PRGMERR_FLAG | FLASH_EPPERR_FLAG);
else if(status == FLASH_OPERATE_TIMEOUT)
{
    /* test spim fail */
    return;
}

/* erase an spim flash sector */
if(flash_sector_erase(SPIM_TEST_ADDR) != FLASH_OPERATE_DONE)
{
    /* test spim fail */
    return;
}

/* read an spim flash sector */
memset(read_buffer, 0, SPIM_SECTOR_SIZE);
sector_read(SPIM_TEST_ADDR, SPIM_SECTOR_SIZE, read_buffer);

/* check if the desired sector are erased */
for(i = 0; i < SPIM_SECTOR_SIZE; i++)
{
    if(read_buffer[i] != 0xff)
    {
        /* test spim fail */
        return;
    }
}

/* program an spim flash sector */
i = 0;
while(i < SPIM_SECTOR_SIZE)
{
    status = flash_word_program(SPIM_TEST_ADDR + i, *(uint32_t*)(write_buffer + i));
    if(status != FLASH_OPERATE_DONE)
    {
        /* test spim fail */
        return;
    }
}
```

```
        i = i + 4;
    }

    /* read an spim flash sector */
    memset(read_buffer, 0, SPIM_SECTOR_SIZE);
    sector_read(SPIM_TEST_ADDR, SPIM_SECTOR_SIZE, read_buffer);

    /* check if reading result and writing content are the same */
    for(i = 0; i < SPIM_SECTOR_SIZE; i++)
    {
        if(read_buffer[i] != write_buffer[i])
        {
            /* test spim fail */
            return;
        }
    }

    while(1)
    {
        /* toggle led */
        at32_led_toggle(LED2);
        delay_ms(100);
        at32_led_toggle(LED3);
        delay_ms(100);
        at32_led_toggle(LED4);
        delay_ms(100);
    }
}
```

■ 主函数代码描述

```
/* operate spim flash */
spim_operate();
```

5.3.4 实验效果

AT-START 的板载 LED 点亮，说明读写成功，符合程序设计预期。

5.4 案例 4--程序运行在外部存储器

5.4.1 功能简介

程序代码一部分烧录到外部存储器，运行在外部存储器中的代码执行三个LED的翻转。

5.4.2 资源准备

1) 硬件环境：

AT32F403A 的 AT-START BOARD

2) 软件环境：

\\project\\at_start_f403a\\examples\\flash\\run_in_spim\\mdk_v5

5.4.3 软件设计

1) 配置流程

- 配置时钟、AT-START BOARD 板子的初始化；
- 初始化外部存储器，这部分代码烧录并运行在内部存储器
- 执行 LED 翻转，这部分代码烧录并运行在外部存储器

2) 代码介绍

- 外部存储器初始化代码描述

```
gpio_init_type gpio_init_struct;

/* enable the clock */
crm_periph_clock_enable(CRM_IOMUX_PERIPH_CLOCK, TRUE);
crm_periph_clock_enable(CRM_GPIOA_PERIPH_CLOCK, TRUE);
crm_periph_clock_enable(CRM_GPIOB_PERIPH_CLOCK, TRUE);

/* init spim io */
gpio_default_para_init(&gpio_init_struct);
gpio_init_struct.gpio_drive_strength = GPIO_DRIVE_STRENGTH_STRONGER;
gpio_init_struct.gpio_out_type = GPIO_OUTPUT_PUSH_PULL;
gpio_init_struct.gpio_mode = GPIO_MODE_MUX;
gpio_init_struct.gpio_pull = GPIO_PULL_NONE;
gpio_init_struct.gpio_pins = GPIO_PINS_8;
gpio_init(GPIOA, &gpio_init_struct);
gpio_init_struct.gpio_pins = GPIO_PINS_1 | GPIO_PINS_6 | GPIO_PINS_7 | GPIO_PINS_10 |
GPIO_PINS_11;
gpio_init(GPIOB, &gpio_init_struct);

/* enable spim, and select pb10, pb11 as spim io */
gpio_pin_remap_config(EXT_SPIM_GMUX_1001, TRUE);
```

```
/* in this example, use on-board en25qh128a as spim flash */
flash_spim_model_select(FLASH_SPIM_MODEL2);

/* unlock the spim flash program erase controller */
while(flash_flag_get(FLASH_SPIM_OBF_FLAG));
flash_spim_unlock();
while(FLASH->ctrl3_bit.oplk);

/* if the data written to spim flash need to be scrambled, please specify the scrambled range */
flash_spim_encryption_range_set(0);

return;
```

■ 运行在外部存储器的代码描述

```
while(1)
{
    /* toggle led */
    at32_led_toggle(LED2);
    delay_ms(100);
    at32_led_toggle(LED3);
    delay_ms(100);
    at32_led_toggle(LED4);
    delay_ms(100);
}
```

■ 主函数代码描述

```
/* configures the spim flash */
spim_init();

/* check the led toggle in spim */
spim_run();
```

5.4.4 实验效果

AT-START的板载LED翻转，说明程序成功的运行在外部存储器，符合程序设计预期。

5.5 案例 5--断电时利用 PVM 写入保存资料到 flash

5.5.1 功能简介

详见“断电时利用PVM 及时写入保存资料的技巧”章节。

5.5.2 资源准备

1) 硬件环境:

AT32F403A 的 AT-START BOARD

2) 软件环境:

AN0014_FLASH_Application_Note\SourceCode\utilities\save_data_when_power_off\mdk_v5

5.5.3 软件设计

1) 配置流程

- 配置时钟、AT-START BOARD 板子的初始化;
- 初始化 PVM;
- 提前擦除 flash 用于数据存储的区域。
- 设计 PVM 中断服务函数，实现在发生掉电情况下时将待保存数据写入 flash。

2) 代码介绍

■ PVM 中断服务函数代码

```
uint32_t save_data[8] = {0x11111111,0x22222222,0x33333333,0x44444444,0x55555555,0x66666666,
0x77777777,0x88888888};

extern uint32_t pvd_write_address;

void PVM_IRQHandler(void)
{
    int index = 0;
    if(exint_flag_get(EXINT_LINE_16) != RESET)
    {
        /* clear exint line flag */
        exint_flag_clear(EXINT_LINE_16);

        /* unlock the flash controller */
        flash_unlock();

        for(index = 0; index < 8; index++)
        {
            flash_word_program(pvd_write_address, save_data[index]);

            /* address add 4 */
            pvd_write_address+=4;
        }
    }
}
```

```
}

/* lock the flash controller */
flash_lock();
}
}
```

■ PVM 初始化函数代码

```
void pvm_exint_config(void)
{
    exint_init_type exint_init_struct;

    /* config the exint line of the power voltage monitor */
    exint_init_struct.line_select = EXINT_LINE_16;
    exint_init_struct.line_enable = TRUE;
    exint_init_struct.line_mode = EXINT_LINE_INTERRUPT;
    exint_init_struct.line_polarity = EXINT_TRIGGER_RISING_EDGE;
    exint_init(&exint_init_struct);
}
```

■ main 函数代码

```
uint32_t main_index1 = 0;
uint32_t main_index2 = 0;
uint32_t pvd_write_address = 0;
extern uint32_t save_data[8];
int main(void)
{
    nvic_priority_group_config(NVIC_PRIORITY_GROUP_4);
    system_clock_config();
    at32_board_init();

    /* turn on all led, means that code is running */
    at32_led_on(LED2);
    at32_led_on(LED3);
    at32_led_on(LED4);
    delay_ms(500);

    /* enable pwc clock */
```

```
crm_periph_clock_enable(CRM_PWC_PERIPH_CLOCK, TRUE);

/* set the threshold voltage to 2.9v */
pwc_pvm_level_select(PWC_PVM_VOLTAGE_2V9);

/* enable power voltage monitor */
pwc_power_voltage_monitor_enable(TRUE);

/* config the exint line of the power voltage monitor */
pvm_exint_config();

/* enable power voltage monitor interrupt */
nvic_irq_enable(PVM_IRQn, 0, 0);

if(FLASH_SIZE < 256)
{
    pvd_write_address = 0x08000000 + FLASH_SIZE * 1024 -1024;
}
else
{
    pvd_write_address = 0x08000000 + FLASH_SIZE * 1024 -2048;
}

/* the saved data check */
if(FLASH_SIZE < 256)
{
    main_index1 = *(volatile uint32_t *) (0x08000000 + FLASH_SIZE * 1024 -1024);
    main_index2 = *(volatile uint32_t *) (0x08000000 + FLASH_SIZE * 1024 -1024 + 28);
}
else
{
    main_index1 = *(volatile uint32_t *) (0x08000000 + FLASH_SIZE * 1024 -2048);
    main_index2 = *(volatile uint32_t *) (0x08000000 + FLASH_SIZE * 1024 -2048 + 28);
}
if((main_index1 == save_data[0]) && (main_index2 == save_data[7]))
{
    /* led2 on and led3 off,means that the data is successfully saved */
    at32_led_on(LED2);
}
```

```
    at32_led_off(LED3);
}
else
{
    /* led2 off and led3 on,means that their was no data being saved or data saved fail */
    at32_led_off(LED2);
    at32_led_on(LED3);
}

/* erase flash page in advance */
while(at32_button_press() != USER_BUTTON)
{
}

/* unlock the flash controller */
flash_unlock();
flash_sector_erase(FLASH_BASE + FLASH_SIZE * 1024 - 2048);

/* lock the flash controller */
flash_lock();

while(1)
{
}
}
```

5.5.4 实验效果

可通过AT-START BOARD上的LED翻转查看实现效果。

LED4亮：MCU处于运行状态；

LED2亮且LED3灭：资料有成功保存到flash；

LED2灭且LED3亮：资料未被成功保存到flash。

6 版本历史

表 3. 文档版本历史

日期	版本	变更
2021.12.13	2.0.0	最初版本
2022.6.8	2.0.1	更新结构介绍和案例说明
2023.4.25	2.0.2	添加“案例5--断电时利用PVM写入保存资料到flash”并更新文档结构

重要通知 - 请仔细阅读

买方自行负责对本文所述雅特力产品和服务的选择和使用，雅特力概不承担与选择或使用本文所述雅特力产品和服务相关的任何责任。

无论之前是否有过任何形式的表示，本文档不以任何方式对任何知识产权进行任何明示或默示的授权或许可。如果本文档任何部分涉及任何第三方产品或服务，不应被视为雅特力授权使用此类第三方产品或服务，或许可其中的任何知识产权，或者被视为涉及以任何方式使用任何此类第三方产品或服务或其中任何知识产权的保证。

除非在雅特力的销售条款中另有说明，否则，雅特力对雅特力产品的使用和/或销售不做任何明示或默示的保证，包括但不限于有关适销性、适合特定用途（及其依据任何司法管辖区的法律的对应情况），或侵犯任何专利、版权或其他知识产权的默示保证。

雅特力产品并非设计或专门用于下列用途的产品：（A）对安全性有特别要求的应用，例如：生命支持、主动植入设备或对产品功能安全有要求的系统；（B）航空应用；（C）航天应用或航天环境；（D）武器，且/或（E）其他可能导致人身伤害、死亡及财产损失的应用。如果采购商擅自将其用于前述应用，即使采购商向雅特力发出了书面通知，风险及法律责任仍将由采购商单独承担，且采购商应独立负责在前述应用中满足所有法律和法规要求。

经销的雅特力产品如有不同于本文档中提出的声明和/或技术特点的规定，将立即导致雅特力针对本文所述雅特力产品或服务授予的任何保证失效，并且不应以任何形式造成或扩大雅特力的任何责任。

© 2023 雅特力科技 保留所有权利