AN0002
入门指南

FLASH模拟EEPROM入门指南

前言

由于 AT32 单片机没有 EEPROM 功能，但是在一些应用中需要使用 EEPROM 存储数据。出于节省外置 EEPROM 芯片降低应用成本的考虑，本入门指南详细阐述了如何使用 AT32 的片上 FLASH 模拟 EEPROM 功能。

注：本应用笔记对应的代码是基于雅特力提供的V2.x.x 板级支持包（BSP）而开发，对于其他版本BSP，需要注意使用上的区别。

支持型号列表：

支持型号	AT32 全系列

目录

1 FLASH 与 EEPROM 简介 5

2 FLASH 模拟 EEPROM 原理 6

2.1 EERPOM 数据结构 6

2.2 EERPOM 物理结构 8

3 EEPROM 使用 10

3.1 初始化状态机 10

3.2 函数接口 10

4 数据直接存储模式 11

4.1 存储原理介绍 11

4.2 函数接口 12

5 两种存储模式混合使用 13

6 案例 EEPROM 读写数据 14

6.1 功能简介 14

6.2 资源准备 14

6.3 软件设计 14

6.4 实验效果 15

7 案例 数据直接存储 16

7.1 功能简介 16

7.2 资源准备 16

7.3 软件设计 16

7.4 实验效果 17

8 文档版本历史 18

表目录

表 1. FLASH 与 EEPROM 特点对比 5
表 2. 不同型号扇区大小 8
表 3. EEPROM 初始化状态机 10
表 4. 文档版本历史 18

图目录

图 1. EEPROM 数据存储结构 6
图 2. EEPROM 写入流程图 7
图 3. EEPROM 物理结构 8
图 4. EEPROM 定义示例 8
图 5. FLASH 存储结构 11
图 6. FLASH 存储定义示例 11
图 7. 混合存储定义示例 13

1 FLASH 与 EEPROM 简介

FLASH 和 EEPROM 都为非易失性存储器，在断电后数据仍然可以长期保存，这为 FLASH 模拟 EEPROM 提供了条件，FLASH 与 EEPROM 特点对比如下表所示：

表 1. FLASH 与 EEPROM 特点对比

异同	FLASH	EEPROM
相同点	非易失性存储，断电后数据仍可保存	非易失性存储，断电后数据仍可保存
不同点	写入数据前需先擦除（只能将位由 1 写 0）	写入数据前无需擦除（能将位由 1 写 0，和由 0 写 1）
	最小擦除单元为扇区	无扇区结构
	容量大，价格便宜	容量小，价格昂贵
	擦写寿命大于 10 万次	擦写寿命大于 100 万次
	数据保存时间大于 20 年	数据保存时间大于 100 年

FLASH 模拟 EEPROM 优点：

- 低成本：可节约一颗 EEPROM 芯片；
- 存储、读取速度快：通讯速度快于使用 I²C 或者 SPI 通讯的 EEPROM 元件；
- 抗干扰能力强：由于 FLASH 在单片机内部，不会存在通讯总线被外部干扰的问题；
- 容量可调：可根据实际使用，灵活调整存储空间大小。

2 FLASH 模拟 EEPROM 原理

2.1 EERPOM 数据结构

由于 FLASH 在写入数据前，需要将 FLASH 数据先擦除为 0xFF，而 FLASH 擦除时通常为扇区擦除，例如 AT32F403A 的扇区大小为 2K 字节，这个特性决定了不能简单的将旧数据擦除然后写新数据，因为这样会导致存储在这个扇区内的其他数据也被擦除，并且也会导致 FLASH 频繁擦除而降低其使用寿命。

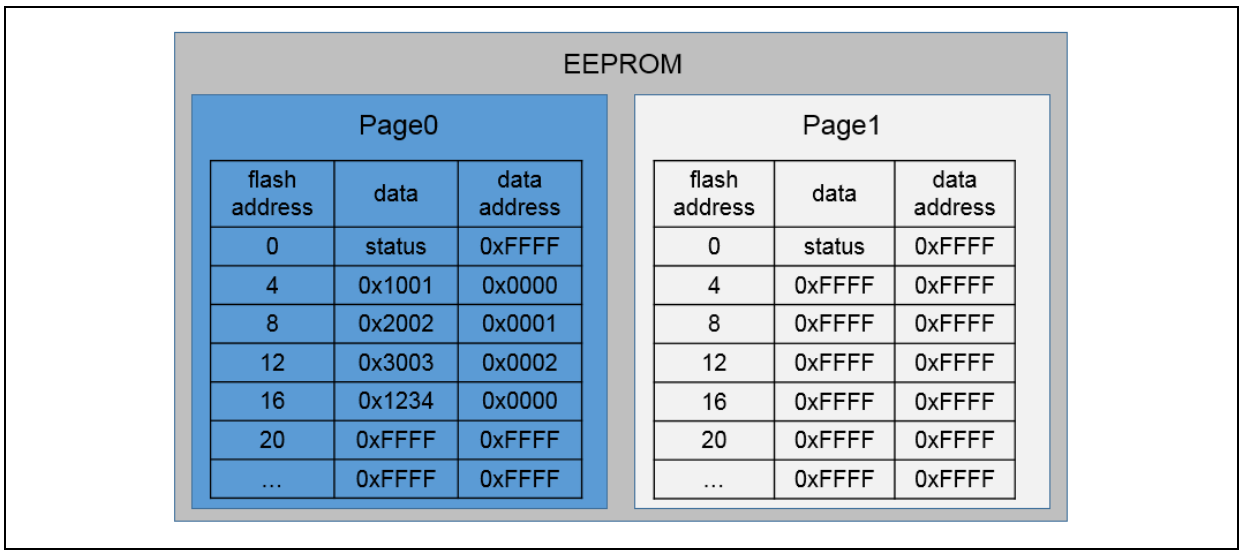
所以 FLASH 模拟 EEPROM 的思路是：

— 新数据存储不影响旧数据；

— 尽量减少 FLASH 擦除次数，延长 FLASH 使用寿命。

基于以上的考虑，我们设计了以下存储结构：

图 1. EEPROM 数据存储结构



EERPOM 结构

EEPROM 由两个页组成：页 0 和页 1，在使用的时候，1 个页处于有效状态，另外一个页处于擦除状态，读取或者写入数据都在有效状态的页进行。

数据格式

存储的数据格式为数据 + 数据地址，地址和数据都是 16 位方式存储，每一次存储占用 32 位也就是 4 个字节。图中 data 列为数据，data address 列为数据地址，flash address 列为数据存储的实际 flash 地址偏移量。例如上图中页 0 的 flash address=12 处，数据为 0x3003，数据地址为 0x0002。

页状态标志

在第一个数据存储区，存储页状态标志 status，页状态标志有 3 种：

— 有效状态：EE_PAGE_VALID，status = 0x0000，读取和写数据在此页进行；

— 数据转移状态：EE_PAGE_TRANSFER，status = 0xCCCC，另外一页满了，正在传输有效数据到本页；

— 擦除状态：EE_PAGE_ERASED，status = 0xFFFF。

数据写入

每一次写入数据前，都会从页起始地址开始寻找第一个未存储数据的区域（值为 0xFFFFFFFF），然

后将待写入的数据和数据地址写到未存储数据的区域。例如上图中页 0 的 flash address = 20 处，值为 0xFFFFFFFF，就是第一个未存储数据的区域。

当知道了页的大小后，就可以算出最大的变量存储个数：页容量/4-1。例如当页大小为 1K 时，最大可存储的变量数量为 1024/4-1=255。需要注意的是，在实际使用中，应该尽量留出较多的空闲容量，这样可以减小 FLASH 擦除次数，提高 FLASH 寿命。

另外数据地址不可以超过最大能存储的变量数量，例如当页大小为 1K 时，最大可存储的变量数量为 1024/4-1=255，那么数据地址 data address 不可以大于 255。

数据读取

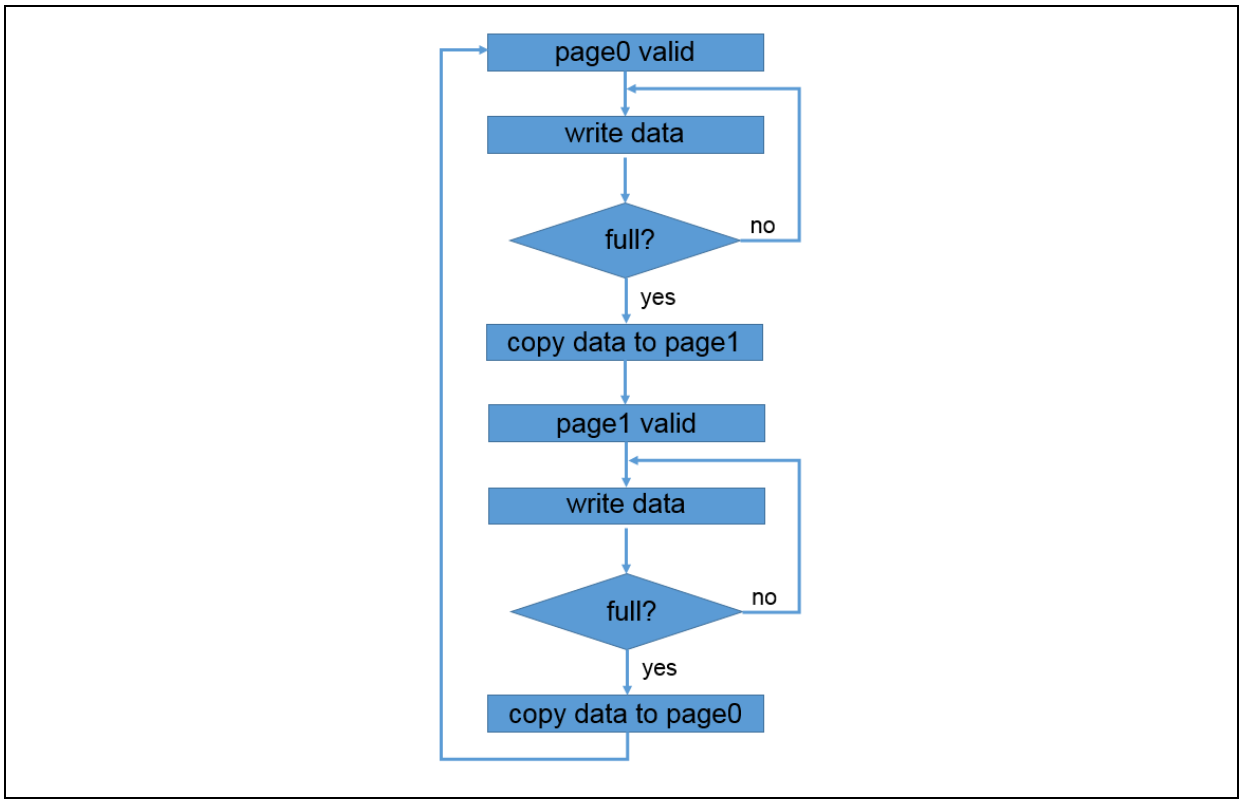
每一次读取数据都会从页结束地址开始向前寻找最后一个存储的有效数据，例如现在要读取地址为 0x0000 的数据。从上图中看到 flash address = 4 和 flash address = 16 都是地址为 0x0000 的数据，因为最后一次存储的数据为 flash address = 16 处的数据，所以此时读取地址 0x0000 的数据为 0x1234。

数据转移

当一页数据存满了之后，会将数据传输到空闲页，将会执行以下操作（以页 0 满，页 1 空为例）：

- 将页 1 状态标记为数据传输状态（EE_PAGE_TRANSFER）；
- 将所有有效数据复制到页 1；
- 擦除页 0；
- 将页 1 状态标记为有效状态（EE_PAGE_VALID）。

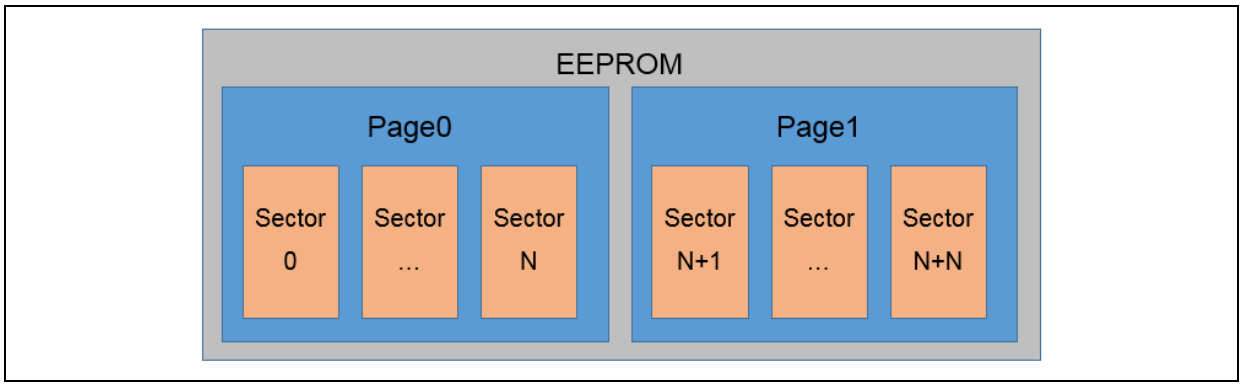
图 2. EEPROM 写入流程图



2.2 EERPOM 物理结构

AT32 所实现的 EEPROM 结构如下图所示，一个页可以由 1 个或者多个扇区组成，可以根据实际应用灵活的选择扇区数量，扇区数量越多，可以存储的数据量就越多。通常 EEPROM 存储区定义在整个 FLASH 末尾，这样程序的烧录、执行和 EEPROM 区域互不影响。

图 3. EEPROM 物理结构



扇区配置可通过 project\at_start_f403a\eeprom\inc\eeprom.h 里面的宏配置

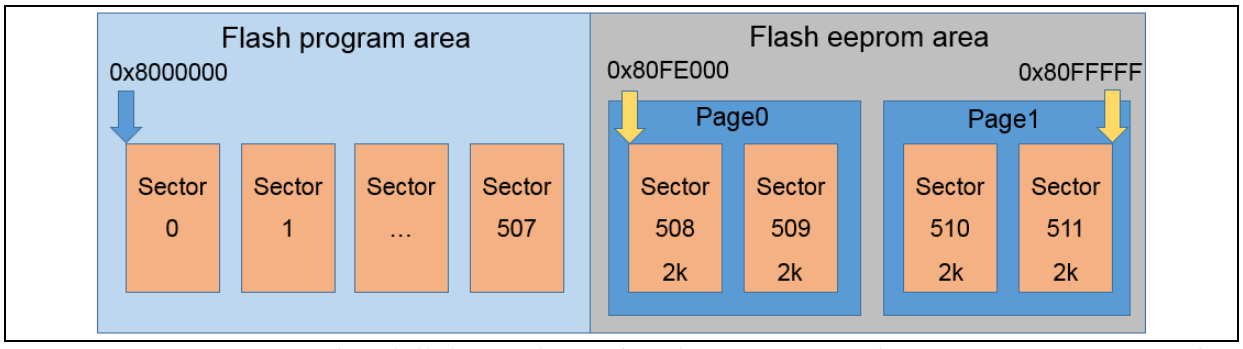
- EE_SECTOR_NUM：定义单个页的扇区个数；
- EE_SECTOR_SIZE：定义扇区大小，单位是字节，不同型号的扇区大小不一样，详情见下表；
- EE_BASE_ADDRESS：定义 EEPROM 扇区起始地址，通常将 EEPROM 放在最末尾（已经自动计算，放在末尾，无需用户配置）。

表 2. 不同型号扇区大小

型号	扇区大小
AT32F403	2K
AT32F403A	2K
AT32F413	2K
AT32F415	2K
AT32F435	4K
AT32F421	1K
AT32F425	1K
AT32L021	1K

以 AT32F403A 1024K Flash 容量为例，EEPROM 定义如下图所示：

图 4. EEPROM 定义示例



如上图所示，FLASH 总计扇区个数为 512 个，总容量为 1024K。分配扇区 0~507，共计 1016K 字

节用于程序存储；分配扇区 508~511 共计 8K 用于 EEPROM。EEPROM 存储在最后 4 个扇区，所以定义 EEPROM 扇区起始地址 EE_BASE_ADDRESS 为 0x80FE000（EE_BASE_ADDRESS 已经自动计算放在 FLASH 末尾，用户无需关心）。

每一个扇区大小为 2K，所以定义扇区大小 EE_SECTOR_SIZE 为 2048。

每一个页包含了 2 个扇区，所以定义扇区数量 EE_SECTOR_NUM 为 2。

此时单个页的容量为 4K，所以最大能存储的变量为 4096/4-1=1023 个，需要注意的是 1023 是理论上最大能存储的变量个数，在实际使用中，应该尽量留出更多的空闲容量，这样可以减小 FLASH 擦除次数，提高 FLASH 寿命。

3 EEPROM 使用

3.1 初始化状态机

由于 EEPROM 是通过页 0 和页 1 的 status 标志管理的，当一页写满了之后会进行复制有效数据到新页，有可能在复制数据过程中发生断电、MCU 复位等情况，此时当 MCU 重新启动时，需要继续完成之前的操作才能继续使用。所以在使用 EEPROM 前，需要根据 status 标志值，来执行相关的初始化操作。初始化状态机已经被封装进了函数 flash_ee_init()，用户可以直接调用。

表 3. EEPROM 初始化状态机

		页 1		
		有效 VALID	数据转移 TRANSFER	擦除 ERASE
页 0	有效 VALID	擦除页 0 标记页 0 为 VALID 擦除页 1	擦除页 1 数据从页 0 复制到页 1 擦除页 0 标记页 1 为 VALID	擦除页 1
	数据转移 TRANSFER	擦除页 0 数据从页 1 复制到页 0 擦除页 1 标记页 0 为 VALID	擦除页 0 标记页 0 为 VALID 擦除页 1	擦除页 1 标记页 0 为 VALID
	擦除 ERASE	擦除页 0	擦除页 0 标记页 1 为 VALID	擦除页 0 标记页 0 为 VALID 擦除页 1

3.2 函数接口

FLASH 模拟 EERPOM 功能提供了 3 个函数以供用户使用，分别是初始化、写数据、读数据。

1) EEPROM 初始化，每一次 MCU 复位了之后，都必须调用此函数进行初始化。

```
flash_status_type flash_ee_init(void);
```

- 返回值：Flash 操作状态。

2) 从 EEPROM 读取数据

```
uint16_t flash_ee_data_read(uint16_t address, uint16_t* pdata);
```

- address：为变量的地址；
- pdata：为读出的数据；
- 返回值：数据读取状态 0：成功读取数据，1：未找到数据。

3) 写数据到 EEPROM

```
flash_status_type flash_ee_data_write(uint16_t address, uint16_t data);
```

- address：为变量的地址；
- data：为写入的数据；
- 返回值：Flash 操作状态。

需要注意的是 address 值不可以大于变量个数，例如当页大小为 2K 时，最大能存储 2048/4-1=511 个变量，那么 address 的范围就是 0~511。

4 数据直接存储模式

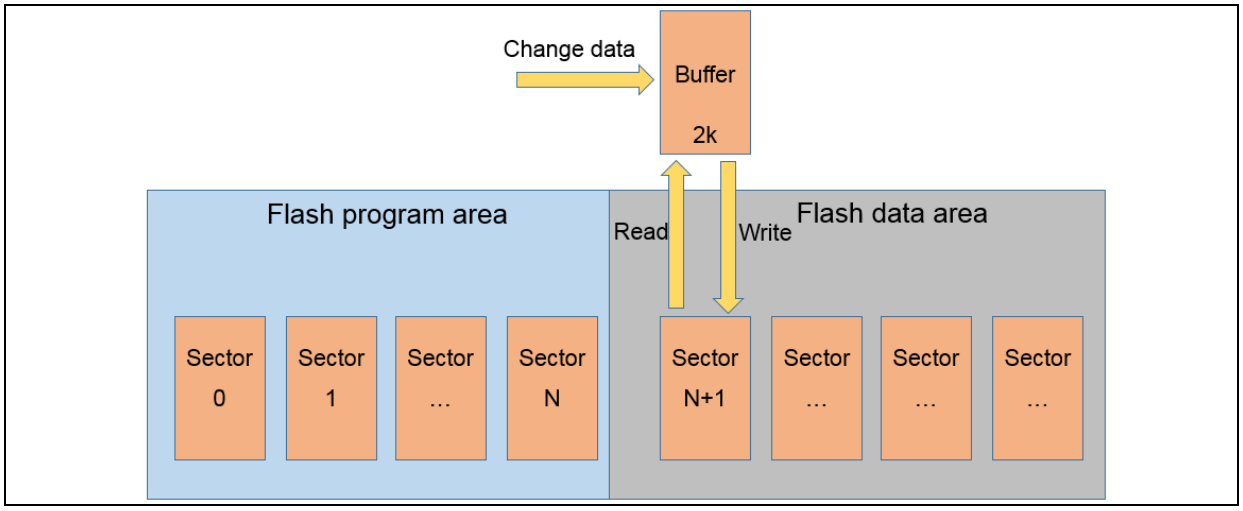
4.1 存储原理介绍

前几章节所叙述的 FLASH 模拟 EEPROM 机制，在存储少量数据时具有使用便捷、存储可靠的优点。但是在存储大量数据或者想实现对 FLASH 任意地址实现存储访问时，这种机制便不太适合。

所以我们也提供了另外一种思路，用户直接访问 FLASH 实现数据的存储。如下图所示：分配 FLASH 后面一段空间来存储数据，用户可以对这段空间的任意地址进行存储或者读取数据，由于 FLASH 擦除是按扇区擦除，而扇区大小通常为 1K、2K、4K（见表 2），所以在对扇区写数据时不能直接往里面写数据。在写数据时，应该先开辟出一个和扇区大小相同的缓存区，先将扇区数据读回来，然后在缓存中更改数据，然后擦除扇区，再将数据写进扇区。

这种数据存储方式，只适用于存储非关键数据，例如一些运行日志之类的信息，因为在将扇区数据读取到缓存，然后擦除扇区时，如果此时发生了掉电或者 MCU 复位的异常情况，将会导致这个扇区数据丢失。关键数据的存储还是要选择 FLASH 模拟 EEPROM 这种存储模式。

图 5. FLASH 存储结构

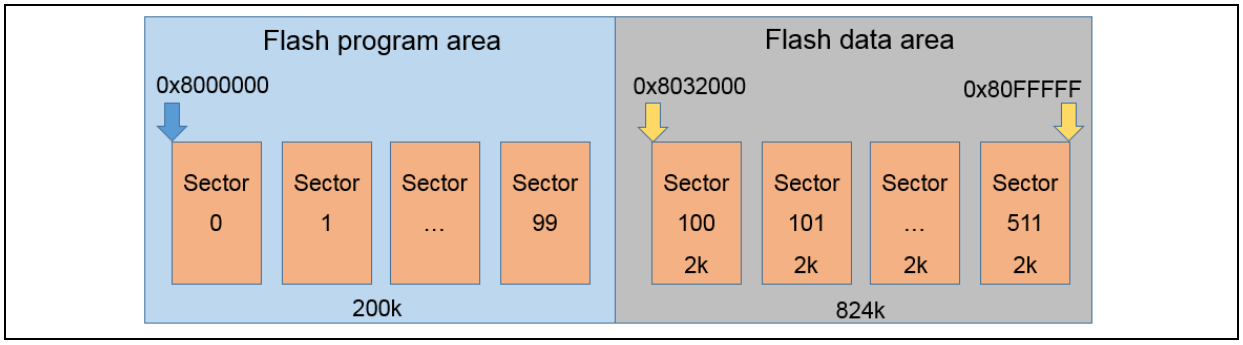


扇区配置可通过 project\at_start_f403a\flash_write_read\inc\flash.h 里面的宏配置

- FLASH_SECTOR_SIZE：定义扇区大小，单位是字节，不同型号的扇区大小不一样，详情见表 2；
- FLASH_CODE_SIZE：定义程序存储区域大小，单位是字节。

以 AT32F403A 1024K Flash 容量为例定义存储区：

图 6. FLASH 存储定义示例



如上图所示，FLASH 总计扇区个数为 512 个，总容量为 1024K。分配扇区 0~99，共计 200K 字节

用于程序存储；分配扇区 100~511 共计 824K 用于数据存储。

所以定义扇区大小 FLASH_SECTOR_SIZE 为 2048，定义程序存储区大小 FLASH_CODE_SIZE 为为 1024*200。

4.2 函数接口

FLASH 数据直接访问模式提供了 2 个函数以供用户使用，分别是写数据、读数据。

1) 从 FLASH 读取数据

```
void flash_read(uint32_t address, uint16_t *pdata, uint32_t number);
```

- address：为 FLASH 的地址，需要在数据存储区域里面，并且 2 字节对齐，例如地址可以配置为 0x8032002、0x8032004，但是不可以为 0x8032003；
- pdata：为读出的数据；
- number：数据读取个数。

2) 写数据到 FLASH

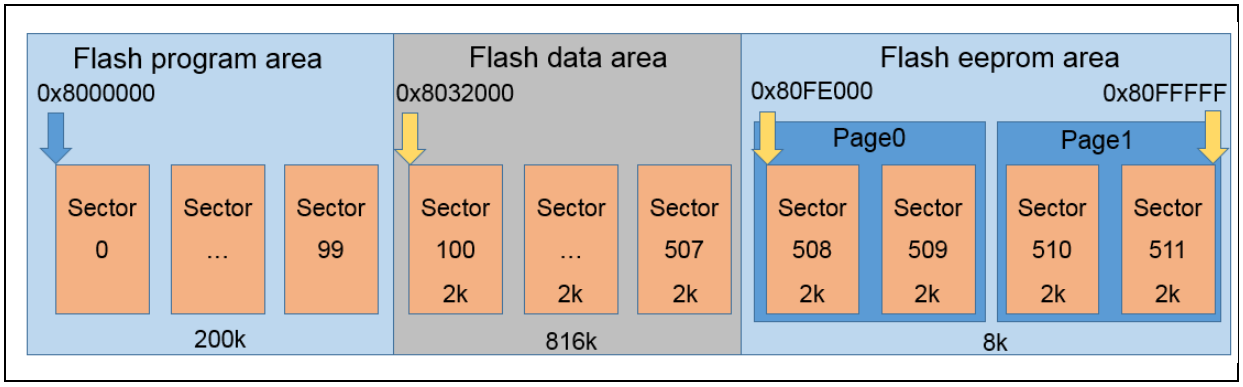
```
void flash_write(uint32_t address, uint16_t *pdata, uint32_t number);
```

- address：为 FLASH 的地址，需要在数据存储区域里面，并且 2 字节对齐，例如地址可以配置为 0x8032002、0x8032004，但是不可以为 0x8032003；
- data：为写入的数据；
- number：数据写入个数。

5 两种存储模式混合使用

上述两种存储模式可以混合使用，以达到既能实现关键数据的可靠存储，也能实现大量数据的存储。以 AT32F403A 1024K Flash 容量为例定义存储区：

图 7. 混合存储定义示例



如上图所示，FLASH 总计扇区个数为 512 个，总容量为 1024K。分配扇区 0~99，共计 200K 字节用于程序存储；分配扇区 100~507 共计 816K 用于数据存储；分配扇区 508~511 共计 8K 用于 EEPROM。

6 案例 EEPROM 读写数据

6.1 功能简介

演示对 EEPROM 进行写数据和读数据。

本示例程序在 AT32 所有系列单片机上通用，不同型号间移植时只需要更改扇区个数 EE_SECTOR_NUM、扇区大小 EE_SECTOR_SIZE 即可，详细介绍见 2.2 章节。

6.2 资源准备

1) 硬件环境:

AT32F403A AT-START BOARD

2) 软件环境

EEPROM_Emulation_V2.0.0\project\at_start_f403a\eeprom

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。

6.3 软件设计

1) 配置流程

- 初始化 EEPROM
- 向 EEPROM 写数据
- 从 EEPROM 读数据
- 比较读取的数据和写入的数据是否相等

2) 代码介绍

- main 函数代码描述

```
int main(void)
{
  uint16_t i, address;

  /* 初始化系统时钟 */
  system_clock_config();

  /* 初始 START 板子资源 */
  at32_board_init();

  /* 初始化 flash eeprom */
  flash_ee_init();

  /* 写数据到 eeprom */
  for(i = 0; i < BUF_SIZE; i++)
  {
    address = i;
```

```
    flash_ee_data_write(address, buf_write[i]);
  }

  /* 从 eeprom 读取数据 */
  for(i = 0; i < BUF_SIZE; i++)
  {
    address = i;

    flash_ee_data_read(address, &buf_read[i]);
  }

  /* 比较读取的数据和写入的数据是否相等 */
  if(buffer_compare(buf_write, buf_read, BUF_SIZE) == 0)
  {
    at32_led_on(LED3);
  }
  else
  {
    at32_led_off(LED3);
  }

  while(1)
  {
  }
}
```

6.4 实验效果

- 如果读取的数据和写入的数据相等，则 LED3 点亮，否则 LED3 不亮。

7 案例 数据直接存储

7.1 功能简介

演示对 FLASH 进行直接写数据和读数据。

本示例程序在 AT32 所有系列单片机上通用，不同型号间移植时只需要更改程序存储区域大小 FLASH_CODE_SIZE、扇区大小 FLASH_SECTOR_SIZE 即可，详细介绍见 4.1 章节。

7.2 资源准备

3) 硬件环境:

AT32F403A AT-START BOARD

4) 软件环境

EEPROM_Emulation_V2.0.0\project\at_start_f403a\flash_write_read

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。

7.3 软件设计

3) 配置流程

- 向 FLASH 写数据
- 从 FLASH 读数据
- 比较读取的数据和写入的数据是否相等

4) 代码介绍

- main 函数代码描述

```
int main(void)
{
  /* 初始化系统时钟 */
  system_clock_config();

  /* 初始 START 板子资源 */
  at32_board_init();

  /* 写数据到 FLASH */
  flash_write(0x8000000 + 1024 * 256, buf_write, BUF_SIZE);

  /* 从 FLASH 读取数据 */
  flash_read(0x8000000 + 1024 * 256, buf_read, BUF_SIZE);

  /* 比较读取的数据和写入的数据是否相等 */
  if(buffer_compare(buf_write, buf_read, BUF_SIZE) == 0)
  {
    at32_led_on(LED3);
  }
  else
```

```
  {
    at32_led_off(LED3);
  }

  while(1)
  {
  }
}
```

7.4 实验效果

- 如果读取的数据和写入的数据相等，则 LED3 点亮，否则 LED3 不亮。

8 文档版本历史

表 4. 文档版本历史

日期	版本	变更
2021.12.15	2.0.0	最初版本
2022.09.07	2.0.1	修改表1 EEPROM描述错误“由1写0”更改为“由0写1”

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、死亡及财产损害的应用。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，且采购商应独力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。