



Hi3861V100 / Hi3861LV100 文件系统

使用指南

文档版本 01

发布日期 2020-04-30

版权所有 © 上海海思技术有限公司2020。保留一切权利。

非经本公司书面许可，任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部，并不得以任何形式传播。

商标声明



HISILICON、海思和其他海思商标均为海思技术有限公司的商标。

本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受海思公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，海思公司对本文档内容不做任何明示或默示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

上海海思技术有限公司

地址： 深圳市龙岗区坂田华为总部办公楼 邮编： 518129

网址： <https://www.hisilicon.com/cn/>

客户服务邮箱： support@hisilicon.com



前言

概述

本文档主要介绍Hi3861V100、Hi3861LV100的文件系统开发相关内容，包括接口实现机制与使用说明。

产品版本

与本文档相对应的产品版本如下。

产品名称	产品版本
Hi3861	V100
Hi3861L	V100



读者对象

本文档主要适用于以下工程师：



- 技术支持工程师
- 软件开发工程师

符号约定

在本文中可能出现下列标志，它们所代表的含义如下。

符号	说明
 危险	表示如不可避免则将会导致死亡或严重伤害的具有高等级风险的危害。
 警告	表示如不可避免则可能导致死亡或严重伤害的具有中等级风险的危害。



符号	说明
 注意	表示如不可避免则可能导致轻微或中度伤害的具有低等级风险的危害。
须知	用于传递设备或环境安全警示信息。如不可避免则可能会导致设备损坏、数据丢失、设备性能降低或其它不可预知的结果。 “须知”不涉及人身伤害。
 说明	对正文中重点信息的补充说明。 “说明”不是安全警示信息，不涉及人身、设备及环境伤害信息。

修改记录

文档版本	发布日期	修改说明
01	2020-04-30	第一次正式版本发布。
00B02	2020-02-12	<ul style="list-style-type: none">在“2.1.1 概述”中新增关于文件系统操作的注意说明。在“2.1.2 开发流程”的表2-1中新增获取错误码的接口描述。
00B01	2020-01-15	第一次临时版本发布。



目录

前言.....	i
1 概述.....	1
2 系统接口.....	2
2.1 SPIFFS 文件系统.....	2
2.1.1 概述.....	2
2.1.2 开发流程.....	2
2.1.3 注意事项.....	4
2.1.4 编程实例.....	4



1 概述

SPI 闪存文件系统（SPIFFS）是为小型嵌入式系统创建的一个文件系统，为用户提供了文件的打开、关闭、读取、写入、删除等功能，用户可以通过使用这些接口，将数据存储到NOR FLASH。



2 系统接口

2.1 SPIFFS文件系统

2.1 SPIFFS 文件系统

2.1.1 概述

SPIFFS是一个文件系统，用于嵌入式目标上的 SPI NOR FLASH闪存设备。支持磨损均衡、文件系统一致性检查等。

须知

由于文件系统操作相比针对FLASH分区的直接操作，增加了FLASH擦写次数，一定程度上缩减了FLASH寿命。因此，针对读写FLASH频率较高的场景，不建议使用文件系统，建议自定义FLASH分区，直接操作FLASH进行实现。

2.1.2 开发流程

使用场景

SPIFFS是为小型嵌入式系统创建的一个文件系统：

- 专为微控制器上的低 RAM 使用场景而设计
- 提供的接口支持：打开、关闭、读取、写入、删除、枚举等功能
- 实现静态磨损均衡
- 内置的系统一致性检查

功能

SPIFFS文件系统提供的接口如[表2-1](#)所示。



表 2-1 SPIFFS 文件系统接口描述

功能分类	接口名称	描述
文件系统初始化	hi_fs_init	初始化文件系统动态参数并挂载文件系统（系统默认已经初始化）。
文件 I/O 操作	hi_open	<p>打开/创建文件。</p> <ul style="list-style-type: none"> HI_FS_O_RDONLY: 以只读方式打开文件。 HI_FS_O_WRONLY: 以只写方式打开文件。 HI_FS_O_RDWR: 以可读写方式打开文件。 HI_FS_O_CREAT: 如果想打开的文件不存在, 则自动建立该文件。 HI_FS_O_EXCL: 如果HI_FS_O_CREAT也被设置, 此指令会去检查文件是否存在。文件如果不存在, 则建立该文件; 否则将导致打开文件错误。 HI_FS_O_TRUNC: 如果文件存在且以可写的方式打开时, 此指令会将文件长度清0, 而原来存于该文件的资料也将消失。 HI_FS_O_APPEND: 当读写文件时从文件尾开始移动, 即所写入的数据将以附加的方式加入到文件后面。
	hi_close	关闭一个文件句柄, 如果有挂起的写操作, 则在关闭之前完成这些操作。
	hi_read	读文件操作, 成功返回读取的字节数, 失败返回-1并设置错误码, 如果在调用该接口前已到达文件末尾, 则此次read返回0。
	hi_write	成功返回写入的字节数, 出错返回-1并设置错误码。
	hi_unlink	通过路径名删除一个文件。
设置读/写文件偏移	hi_lseek	<p>设置读/写文件偏移。</p> <p>HI_FS_SEEK_SET: 从文件头部开始偏移offset个字节。</p> <p>HI_FS_SEEK_CUR: 从文件当前读写的指针位置开始, 增加offset个字节的偏移量。</p> <p>HI_FS_SEEK_END: 文件偏移量设置为文件的大小加上偏移量字节, 偏移量offset只允许为负值。</p>
枚举文件	hi_enum_file	枚举根目录下的所有文件。
获取文件大小	hi_stat	通过文件名获取文件大小。
获取错误码	hi_get_fs_error	获取文件系统接口调用失败的错误码, 可在hi_errno.h中查找文件系统对应错误码。



2.1.3 注意事项

- SPIFFS文件系统不支持多级目录。
- 文件名最大长度不能超过31byte（不包含文件结束符）。
- 最大支持同时打开32个文件。
- 删除文件时，如果已经打开32个文件，则必须调用hi_close关闭其中任意一个文件，否则将无法删除文件。

2.1.4 编程实例

文件读/写

编程实例：

```
hi_void example_fs_rw(hi_void)
{
    char buf[12]={0};
    /* create a file, delete previous if it already exists, and open it for reading and writing */
    hi_s32 fd = hi_open("my_file", HI_FS_O_CREAT | HI_FS_O_TRUNC | HI_FS_O_RDWR);
    if (fd < 0) {
        printf("errno 0x%x\n", hi_get_fs_error());
        return;
    }
    // write to it
    if (hi_write(fd, "Hello world",12) < 0) {
        printf("errno 0x%x\n", hi_get_fs_error());
        return;
    }
    // close it
    if (hi_close(fd) < 0) {
        printf("errno 0x%x\n", hi_get_fs_error());
        return;
    }
    // open it
    fd = hi_open("my_file", HI_FS_O_RDWR);
    if (fd < 0) {
        printf("errno 0x%x\n", hi_get_fs_error());
        return;
    }
    // read it
    if (hi_read(fd, (hi_char *)buf, sizeof(buf)) < 0) {
        printf("errno 0x%x\n", hi_get_fs_error());
        return;
    }
    // close it
    if (hi_close(fd) < 0) {
        printf("errno 0x%x\n", hi_get_fs_error());
        return;
    }
    // check it
    printf("--> %s <--\n", buf);
}
```

结果验证：

```
--> Hello world <--
```



设置读/写的偏移

编程实例：

```
hi_void example_fs_lseek(hi_void)
{
    u8_t buf[128];
    int i, res;
    u8_t test;
    for (i = 0; i < 128; i++)
    {
        buf[i] = i;
    }
    // create a file with some data in it
    hi_s32 fd = hi_open("somedata", HI_FS_O_CREAT | HI_FS_O_RDWR);
    if (fd < 0)
    {
        printf("errno 0x%x\n", hi_get_fs_error());
        return;
    }
    res = hi_write(fd, (hi_char*)buf, 128);
    if (res < 0)
    {
        printf("errno 0x%x\n", hi_get_fs_error());
        return;
    }
    res = hi_close(fd);
    if (res < 0)
    {
        printf("errno 0x%x\n", hi_get_fs_error());
        return;
    }
    // open for reading
    fd = hi_open("somedata", HI_FS_O_CREAT | HI_FS_O_RDONLY);
    if (fd < 0)
    {
        printf("errno 0x%x\n", hi_get_fs_error());
        return;
    }
    // read the last byte of the file
    res = hi_lseek(fd, -1, HI_FS_SEEK_END);
    if (res < 0)
    {
        printf("errno 0x%x\n", hi_get_fs_error());
        return;
    }
    res = hi_read(fd, (hi_char*)&test, 1);
    if (res < 0)
    {
        printf("errno 0x%x\n", hi_get_fs_error());
        return;
    }
    printf("last byte:%i\n", test); // prints "last byte:127"
    // read the middle byte of the file
    res = hi_lseek(fd, 64, HI_FS_SEEK_SET);
    if (res < 0)
    {
        printf("errno 0x%x\n", hi_get_fs_error());
        return;
    }
    res = hi_read(fd, (hi_char*)&test, 1);
    if (res < 0)
```



```
{
    printf("errno 0x%x\n", hi_get_fs_error());
    return;
}
printf("middle byte:%i\n", test); // prints "middle byte:64"
// skip 3 bytes from current offset and read next byte (NB we read one byte previously also)
res = hi_lseek(fd, 3, HI_FS_SEEK_CUR);
if (res < 0)
{
    printf("errno 0x%x\n", hi_get_fs_error());
    return;
}
res = hi_read(fd, (hi_char*)&test, 1);
if (res < 0)
{
    printf("errno 0x%x\n", hi_get_fs_error());
    return;
}
printf("middle+4 byte:%i\n", test); // prints "middle+4 byte:68"
res = hi_close(fd);
if (res < 0)
{
    printf("11errno 0x%x\n", hi_get_fs_error());
    return;
}
}
```

结果验证:

```
last byte:127
middle byte:64
middle+4 byte:68
```

删除所有文件

编程实例:

```
void example_fs_remove_file(hi_void)
{
    char* buf = NULL;
    //枚举所有文件成功后，须释放buf，否则会导致内存泄漏
    int ret = hi_enum_file("/", &buf);
    if (ret < 0)
    {
        printf("1errno 0x%x\n", hi_get_fs_error());
        return;
    }
    file_list* file = (file_list*)buf;
    do
    {
        int res = hi_unlink(file->name);
        if (res < 0) {
            printf("errno 0x%x\n", hi_get_fs_error());
            break;
        }
        else
        {
            printf("delete %s success.\n", file->name);
        }
        if (!file->next_offset)
        {
            break;
        }
    }
}
```



```
    }
    file = (file_list*)((char*)file + file->next_offset);
}while(TRUE);
hi_free(0, buf);
}
```

结果验证：

```
Delete my_file success.
Delete somedata success.
```

枚举根目录下所有文件

编程实例：

```
hi_void example_fs_enum_file(hi_void)
{
    int ret = 0;
    char* buf = NULL;
    ret = hi_enum_file("/",&buf);
    if (ret < 0) {
        printf("1errno 0x%x\n", hi_get_fs_error());
        return;
    }
    file_list* file = (file_list*)buf;
    do
    {
        printf("%s, size=0x%x\n",file->name,file->size);
        if (!file->next_offset)
        {
            break;
        }
        file = (file_list*)((char*)file + file->next_offset);
    }while(TRUE);
    hi_free(0,buf);
}
```

说明

- 如果根目录下没有文件存在，返回HI_ERR_FS_NO_MORE_FILES错误。
- 下一个文件的偏移位置为0说明此文件为最后一个文件。
- 返回的buf需要由用户释放，防止内存泄漏。

文件在缓冲区的存放格式如表2-2所示。

表 2-2 文件存放格式

文件 大小	下一个 文件偏 移位置	文件 名称	文件 大小	下一个 文件偏 移位置	文件 名称	文件大小	下一个文 件偏移位 置为0	文件 名称
----------	-------------------	----------	----------	-------------------	----------	-------	------	---------------------	----------