SafeFLASH File System RAM Drive User Guide

Version 1.40

For use with SafeFLASH File System RAM Drive Versions 2.02 and above

Date: 10-Oct-2017 13:06

Table of Contents

System Overview 3
Introduction 3
Feature Check 5
Packages and Documents 6
Documents 6
Change History 7
Source File List 8
API Header File 8
Configuration File 8
System Files 8
Version File 8
Configuration Options 9
Application Programming Interface 10
fs_mount_ramdrive 10
Using f_mountdrive() with RAM 11
Implementing a RAM Driver 13
File System Test 15

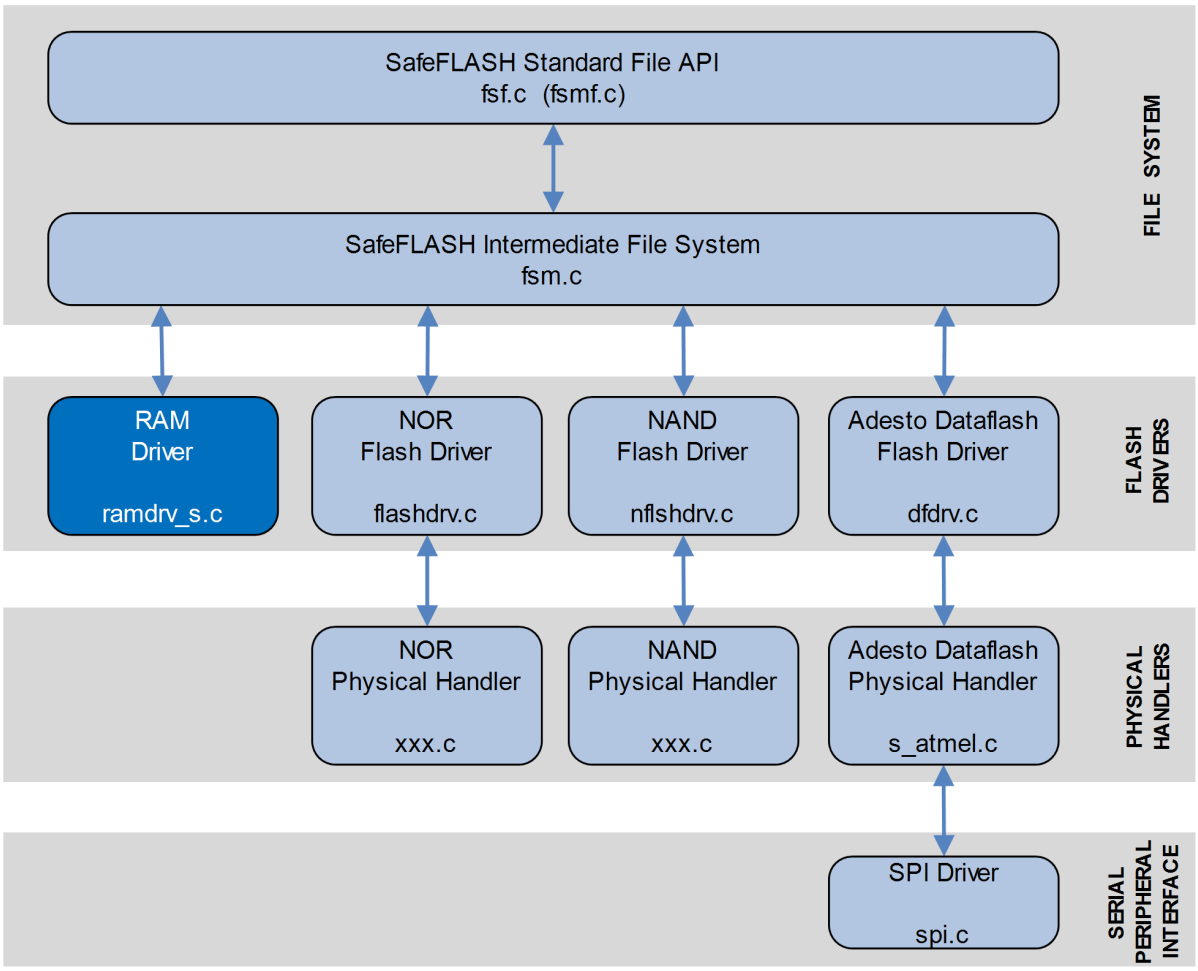
1 System Overview

1.1 Introduction

This guide is for those who wish to implement a RAM driver for HCC's SafeFLASH file system.

The SafeFLASH file system driver design is highly portable while still maintaining excellent performance. The basic device architecture includes a high level driver for each general media type that shares some common properties. This driver handles issues of FAT maintenance, wear leveling, and so on. Implementing a RAM driver for the file system is simple as there is no physical driver associated with the RAM driver.

The following diagram illustrates the structure of the file system software:



In this diagram:

- The main SafeFLASH package provides the file API and intermediate file system. This is described in the *HCC SafeFLASH File System User Guide*.
- The RAM driver is the device driver. This guide shows how to add this to the build. Using the available sample drivers as a model, you can create a driver that meets your specific needs.

Note: HCC Embedded offers hardware and firmware development consultancy to assist developers with the implementation of flash file systems.

1.2 Feature Check

For a full list of SafeFlash features, see the *HCC SafeFlash File System User Guide*.

The main features of the system are the following:

- Conforms to the HCC Advanced Embedded Framework.
- Designed for integration with both RTOS and non-RTOS based systems.

1.3 Packages and Documents

The table below lists the packages that you need in order to use this module:

Package	Description
hcc_base_doc	This contains the two guides that will help you get started.
fs_safe	The SafeFLASH base package.
fs_safe_ram	The SafeFLASH RAM package described in this document.

Documents

For an overview of HCC file systems and guidance on choosing a file system, see Product Information on the main HCC website.

Readers should note the points in the HCC Documentation Guidelines on the HCC documentation website.

HCC Firmware Quick Start Guide

This document describes how to install packages provided by HCC in the target development environment. Also follow the *Quick Start Guide* when HCC provides package updates.

HCC Source Tree Guide

This document describes the HCC source tree. It gives an overview of the system to make clear the logic behind its organization.

HCC SafeFLASH File System User Guide

This document describes the base SafeFLASH System.

HCC SafeFLASH File System RAM Drive User Guide

This is this document.

Other HCC SafeFLASH User Guides

These describe other SafeFLASH components:

- *HCC SafeFLASH File System NOR Drive User Guide* – documents the SafeFLASH system for NOR flash.
- *HCC SafeFLASH File System NAND Drive User Guide* – documents the SafeFLASH system for NAND flash.
- *HCC SafeFLASH for Adesto DataFlash Drives User Guide* – documents the SafeFLASH system for Adesto® DataFlash.

1.4 Change History

This section describes past changes to this manual.

- To view or download earlier manuals, see File System PDFs.
- For the history of changes made to the package code itself, see History: fs_safe_ram.

The current version of this manual is 1.40. The full list of versions is as follows:

Manual version	Date	Software version	Reason for change
1.40	2017-10-10	2.02	Change to **fs_mount_ramdrive()**.
1.30	2017-06-26	2.01	New *Change History* format.
1.20	2015-09-11	2.01	Added *Change History* section, extra configuration option.
1.10	2014-08-20	1.01	Reorganized *System Overview*.
1.00	2014-05-30	1.01	First online version.

2 Source File List

This section describes all the source code files included in the system. These files follow the HCC Embedded standard source tree system, described in the *HCC Source Tree Guide*. All references to file pathnames refer to locations within this standard source tree, not within the package you initially receive.

> **Note:** Do not modify any files except the configuration file.

2.1 API Header File

The file **src/api/api_safe_ram.h** is the only file that should be included by an application using this module. It defines the fs_mount_ramdrive() function.

2.2 Configuration File

The file **src/config/config_safe_ram.h** contains the configurable parameters of the system. Configure these as required. For detailed explanation of these options, see Configuration Options.

2.3 System Files

These files are in the directory **src/safe-flash/ram**. **These files should only be modified by HCC**.

File	Description
ramdrv_s.c	RAM driver source code.
ramdrv_s.h	RAM driver header file.

2.4 Version File

The file **src/version/ver_safe_ram.h** contains the version number of this module. This version number is checked by all modules that use this module to ensure system consistency over upgrades.

3 Configuration Options

Set the configuration options in the file **src/config/config_safe_ram.h**. This section lists the available configuration options and their default values.

FS_RAM_SECSIZE

The sector size. The default is 4096.

FS_RAM_MAXFILE

The maximum number of files that can be open at the same time. The default is 4.

MEMCPY_LONG

Keep this value at the default of 1 if the system supports 32 bit memory access. This accelerates memory access time and results in better performance.

4 Application Programming Interface

This section shows how to implement a RAM driver.

4.1 fs_mount_ramdrive

This function is called by **f_mountdrive()** to mount and map a new drive.

Format

```
extern int fs_mount_ramdrive (
    void *        vol_dsc,
    FS_PHYGETID   phyfunc )
```

Arguments

Argument	Description	Type
vol_dsc	The volume descriptor of the volume to mount.	void *
phyfunc	The physical driver.	FS_PHYGETID

Return values

Return value	Description
0	Drive successfully mounted.
FS_VOL_NOTFORMATTED	Drive is mounted but is not formatted.
FS_VOL_NOMEMORY	Not enough memory, drive is not mounted.
FS_VOL_DRVERROR	Mount driver error, not mounted.

4.2 Using f_mountdrive() with RAM

The **f_mountdrive()** function is part of the main SafeFLASH API. It calls **fs_mount_ramdrive()**. This page shows how to use the function with RAM.

> **Note:** The main *SafeFLASH File System User Guide* describes how to use this call for all drive types.

Format

```
int f_mountdrive (
    int           drivenum,
    void *        buffer,
    long          buffsize,
    FS_DRVMOUNT   mountfunc,
    FS_PHYGETID   phyfunc )
```

Arguments

Argument	Description	Type
drivenum	The number of the drive to mount (0='A', 1='B', and so on.). The maximum value of **drivenum** is set in FS_MAXVOLUME-1 in **fsm.h**.	int
buffer	The buffer pointer the file system uses. Allocate a buffer of the size required for the whole RAM file system, as shown in the example below.	void *
buffsize	The size of the allocated buffer that is passed to the mount function.	long
mountfunc	The **fs_mount_ramdrive()** function.	FS_DRVMOUNT
phyfunc	For a RAM drive this function is NULL.	FS_PHYGETID

Return values

Return value	Description
FS_VOL_OK	Drive successfully mounted.
FS_VOL_NOTMOUNT	Drive not mounted.
FS_VOL_NOTFORMATTED	Drive is mounted but is not formatted.
FS_VOL_NOMEMORY	Not enough memory, drive is not mounted.
FS_VOL_NOMORE	No more drives available (FS_MAXVOLUME).
FS_VOL_DRVERROR	Mount driver error, not mounted.

4.3 Implementing a RAM Driver

Implementing a RAM driver for the file system is simple. There is no physical driver associated with the RAM driver.

1. Include the **ramdrv_s.c** and **ramdrv_s.h** files in your file system build. This ensures that it can be mounted.
2. Call **f_init()** as shown in the example below.
3. Call **f_mountdrive()** with a pointer to the memory area and the size of the area to be used for the driver. The example below shows this.
4. The RAM drive may now be used as a standard drive.

The following example shows the implementation.

```
static long g_tramdrive[RAM_DRIVE_SIZE / sizeof( long )];   /* Must be 32 bit aligned */
static int initted = 0;   /* Flag to signal if volume has been initialized */

void create_ram_drive()
{
    int rc;

    rc = f_init();

    if (rc == SUCCESS)
    {
        rc = f_start();
    }

    if (rc == SUCCESS)
    {
        rc = f_enterFS();
    }

    if (rc == SUCCESS)
    {
        if ( !initted )
        {
            /* Only do this once at first power on. Fill drive with random value */
            (void)memset( g_tramdrive, 0x55, sizeof( g_tramdrive ) );
            initted = 1;
        }
     }

/* Now mount the RAM drive, calling the mount function of the target driver */

    if (rc == SUCCESS)
    {
        rc = f_mountdrive( 0, g_tramdrive, sizeof( g_tramdrive ), fs_mount_ramdrive, 0 );
    }

    if (rc == SUCCESS)
    {
        rc = f_chdrive( 0 );  /* Change to the newly mounted drive */
    }

    return rc;
}
```

5 File System Test

The test suite is provided for exercising the file system and ensuring that it is working correctly. Most of the operational features of the file system are exercised by this program, including file read/write/append/seek /file content, directory and file manipulation functions.

The test program requires the functions that are defined and implemented (as samples) in the file **testport_ram_s.c**. This is part of the **fs_safe** base package. The full path of this file is **src/safe-flash/test /testport_ram_s.c**.

Port the functions to your system. Refer to the comments and simple code for reference.

To use the test program, call the following to execute the test code:

```
void f_dotest( void )
```