SNS COLLEGE OF TECHNOLOGY

Coimbatore-35
An Autonomous Institution

Accredited by NBA – AICTE and Accredited by NAAC – UGC with 'A++' Grade
Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai

DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING

19ECT312 – EMBEDDED SYSTEM DESIGN

III YEAR/ VI SEMESTER

1

UNIT 4: EMBEDDED OS & MODELING

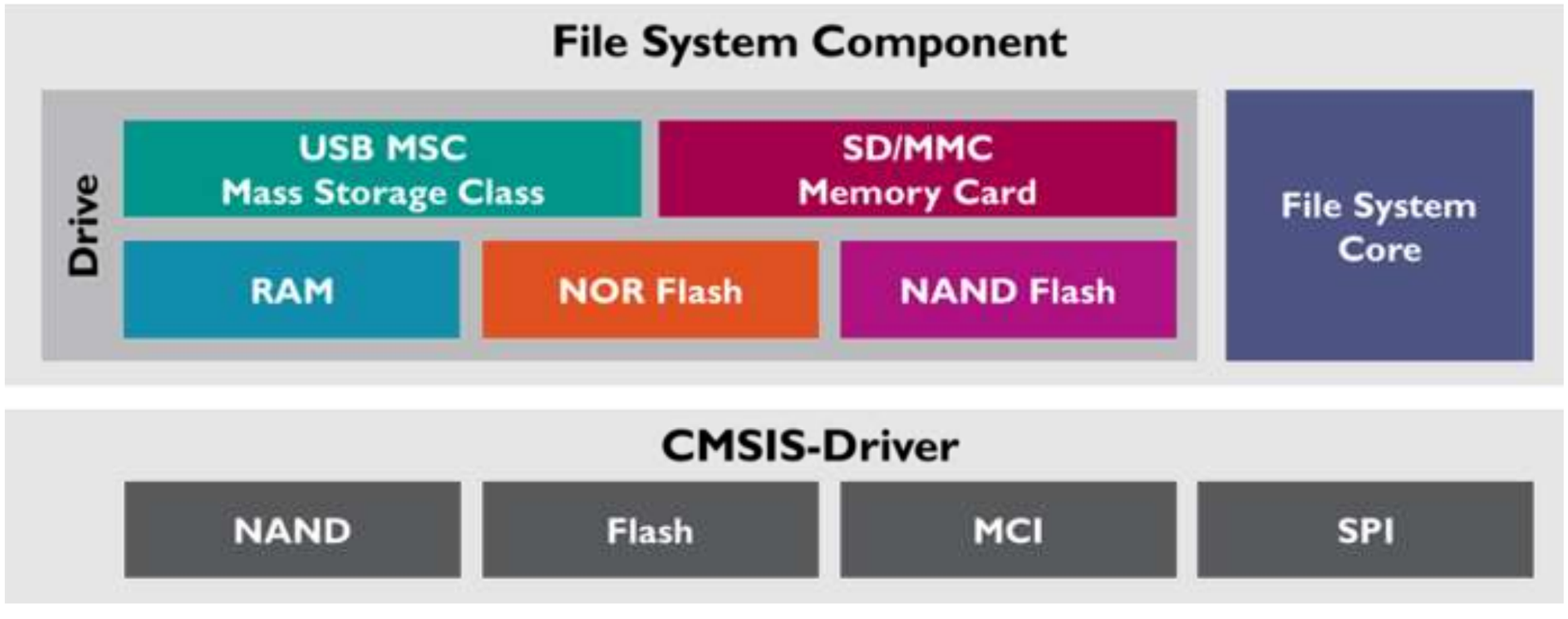
TOPIC 4.6 EMBEDDED FILE SYSTEMS

EMBEDDED FILE SYSTEMS

https://youtu.be/tizaYT3JLUs

Overview

The **File System Component** allows your embedded applications to create, save, read, and modify files in storage devices such as **RAM**, **Flash**, **memory cards**, or **USB memory devices.** It is part of **MDK-Professional** and **MDK-Plus**.



EMBEDDED FILE SYSTEMS

The File System Component is structured as follows:

Storage devices are referenced as **drives** which can be accessed by the user.

Multiple instances of the same storage device can be implemented (for example you might want to have two SD cards attached to your system).

The File System **CORE** supports thread-safe operation and uses an **Embedded File System (EFS)** (for NOR and SPI Flashes) or a **FAT File System** which is available in two variants:

The **Long File Name** variant supports up to 255 characters.

The **Short File Name** variant is limited to 8.3 file name support.

The Core allows **simultaneous access** to multiple storage devices (for example backing up data from internal flash to an external USB device).

For accessing the drives appropriate **drivers** are in place to support

Flash chips (**NAND**, **NOR, and SPI**)

Memory card interfaces (**SD/SDxC/MMC/eMMC**)

USB devices

On-chip **RAM**, Flash and external memory interfaces.

EMBEDDED FILE SYSTEMS

Documentation Structure

This user's guide contains the following chapters:

Create a File System Application explains the necessary steps to develop a project containing a file system from scratch.

File System Examples are a good starting point for implementing your own storage device.

Theory of Operation gives more detail on the basics of the File System Component.

Function Overview lists the complete API of the File System Component.

EMBEDDED FILE SYSTEMS

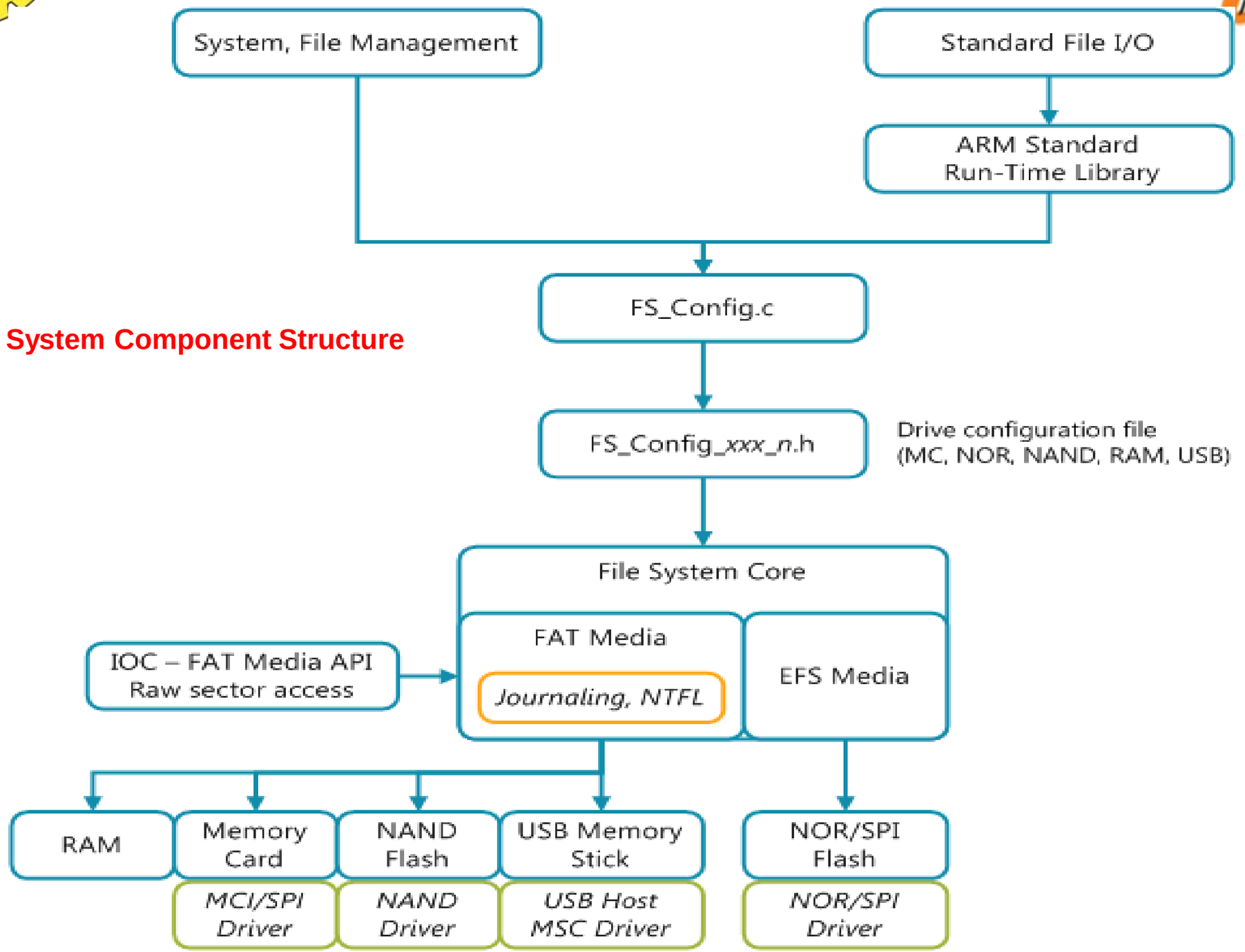
The File System Component integrates with the ARM Standard Run-Time Library and requires a CMSIS-RTOS compliant RTOS.

It cannot be used with the ARM MicroLIB library since this library does not provide the hooks for I/O file handling.

The following picture shows the File System's structure from a developer's perspective.

EMBEDDED FILE SYSTEMS

File System Component Structure



EMBEDDED FILE SYSTEMS

System, File Management: functions that manage the File System and provide operation to format a drive and manage files and directories.

Standard File I/O: functions to perform input and output operations on files, such as read, write, and seek.

ARM Standard Run-Time Library: is the ARM Compiler standard C library with functions to manage file.

FS_Config.c: configuration file for general characteristics of the file system.

FS_Config_*xxx*_*n*.h: configuration file for the characteristics of each drive or media(MC, NAND, NOR, RAM, or USB memory sticks).

File System Core: handles low-level input and output file operations (some are re-targeted to use the ARM Standard Run-Time Library). Depending on configuration settings, it uses the appropriate file system (FAT or EFS) and implements the NAND Flash Translation Layer.

IOC - FAT Media API: are I/O Control Interface Routines for the FAT file system to access physical sectors.

FAT, EFS Media: the FAT file system supports Memory Cards (MC), NAND Flash, and USB memory sticks, and RAM disks. The Embedded File System (EFS) supports NOR Flash devices.

Drivers The File System Core accesses the drives via CMSIS-Drivers that are typically part of the Device Family Pack. Every drive uses dedicated driver.

EMBEDDED FILE SYSTEMS

The **Embedded File System (EFS)** is a proprietary file system used on NOR flash devices. Basic features are:

Memory Organization of the flash device is optimized for maximum performance.

Allocation Information is reduced to a minimum, allowing small data overhead.

File Names & Content are stored in fragments of variable size which provide optimal file access times.

EMBEDDED FILE SYSTEMS

Memory Organization

A NOR flash device memory array is physically divided into **sectors** or **blocks.** The File System Component designates them as **blocks**. Typically, a block's size is 64 kB which is also the smallest erasable unit. Blocks can be further divided, down to memory cells. The memory cell size depends on the device architecture and is 8- (byte), 16- (half word) or 32-bit wide (word). The memory cell architecture also defines smallest programmable unit, which must be maximum 32-bit for use with the Embedded File System.

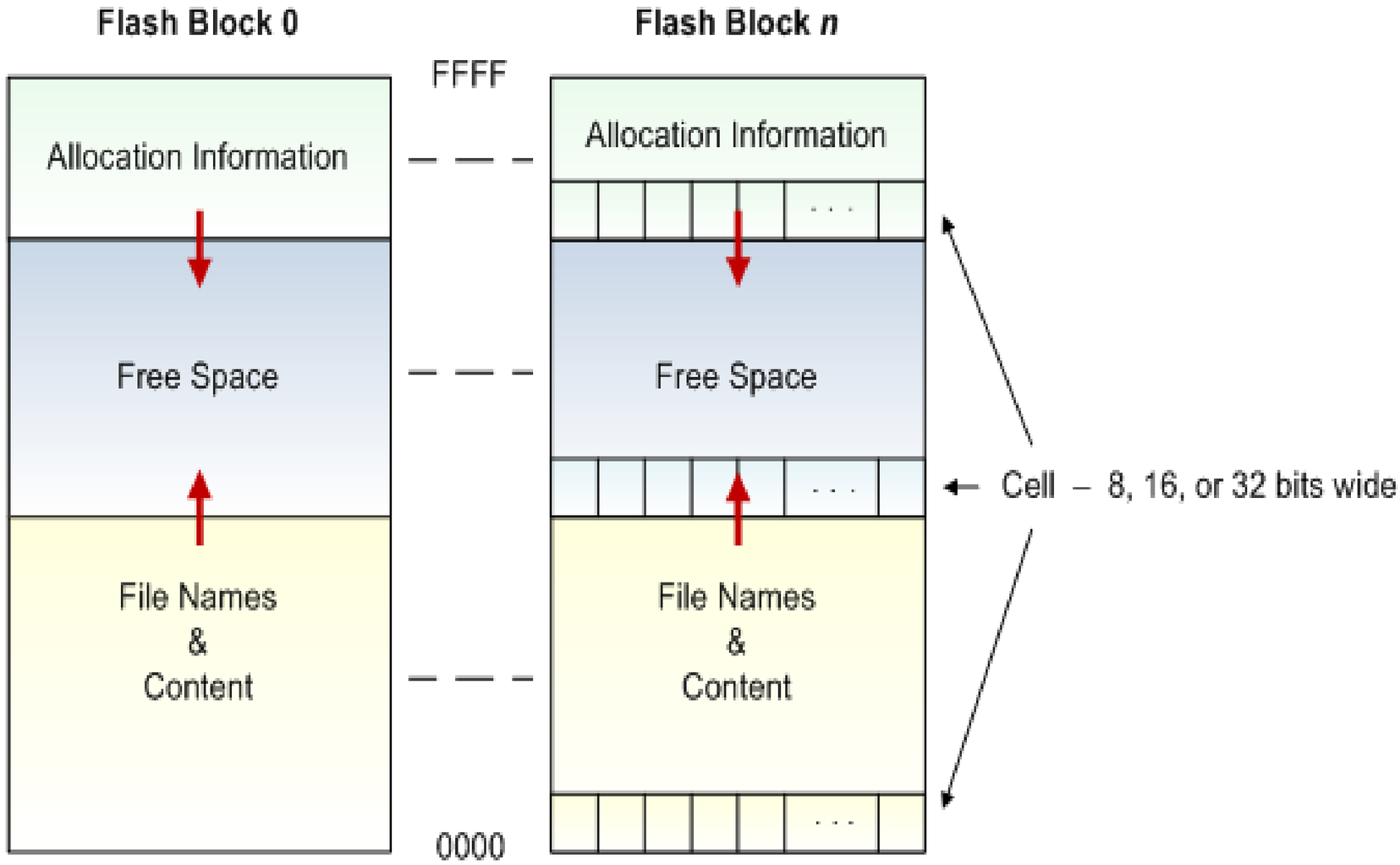
Embedded File System organizes each block into three regions:

Allocation Information, located on top of the block, grows in descending order and contains file allocation records.

Free Space

File Names & Content, located on bottom of the block, grows in ascending order and contains file names and data.

EMBEDDED FILE SYSTEMS



Allocation Information

•Allocation information region is located on top of a block and describes the block's content.

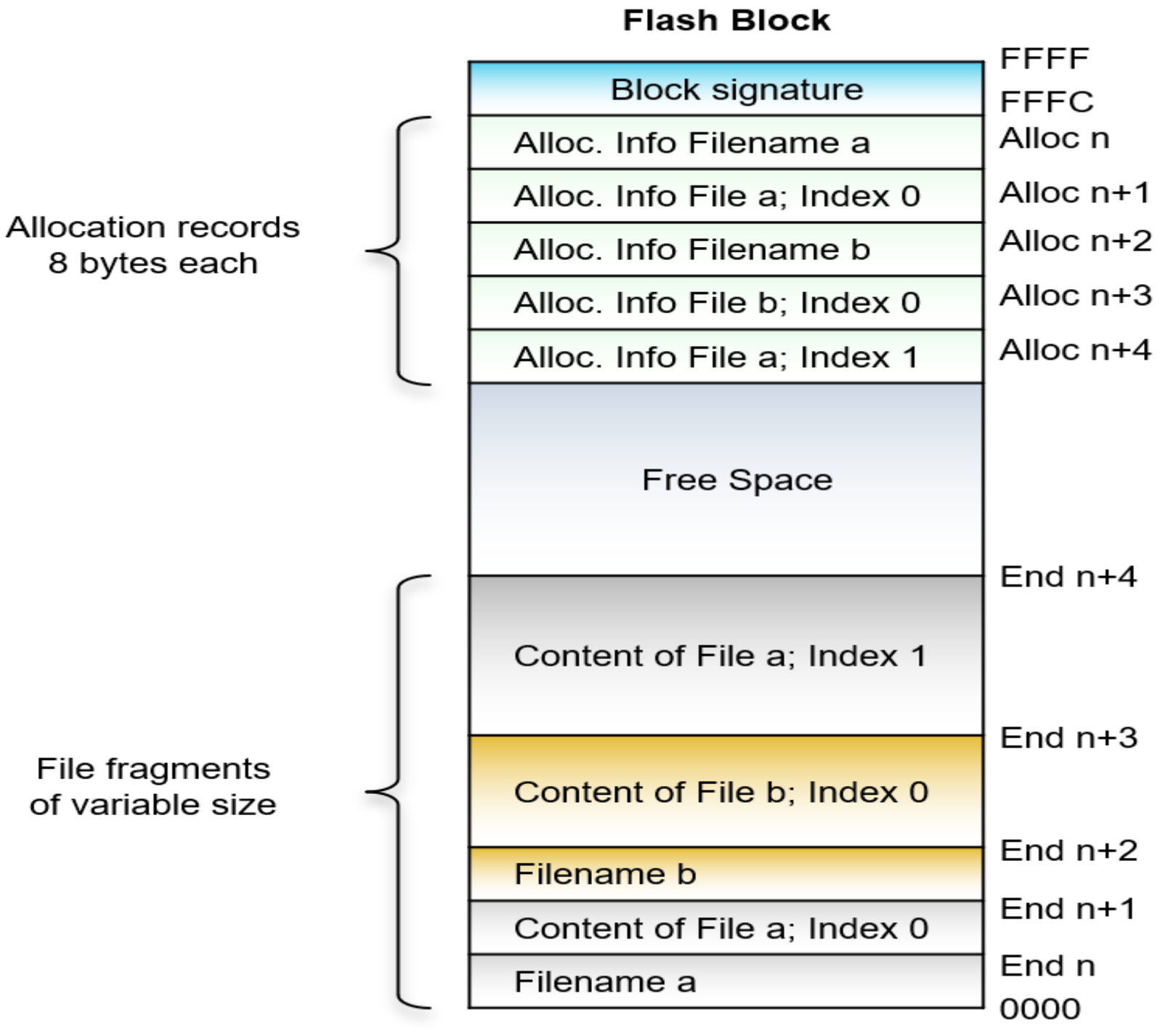
•It consists of block signature and file allocation information records, which are written in descending order.

•Each file has at least one record associated with it. Multiple records belong to files with content and to fragmented files.

•A file is fragmented when it is modified or its content size exceeds a single block size and must be stored across several blocks.

•Several small files are stored into a single block.

EMBEDDED FILE SYSTEMS



EMBEDDED FILE SYSTEMS

Block Signature

Each block contains a signature, consisting of 4 bytes and determines if block is:

empty i.e. erased

used but more data can be written into it

used temporarily during defragmentation

full and cannot be written anymore (only erased)

Allocation Information Record

The file allocation information record consists of 8 bytes and has the following components:

end is the end address of the file fragment.

fileID is the file identification number and is associated with the file name.

index is the file fragment ordering number, which starts at 0 for each file.

```
struct falloc {
uint32_t end;
uint16_t fileID;
uint16_t index;
};
```

The file allocation information is written when:

- The file is opened for writing.
- The file is closed.
- The file is flushed and file fragment is not yet defined by the allocation information record.
- The block is full and there is no more free space.

File Names & Content

The file names & content region is located at the bottom of a block and is fully defined through the file allocation information records. It consists of file names and file content, which can both be fragmented. The first file fragment always starts at the beginning of a block (at offset 0) and is written in ascending order.

EMBEDDED FILE SYSTEMS

File Names

In the Embedded File System, a file name consists of maximum 31 characters. Directories are not supported, therefore any file name which contains a directory separator character, such as slash (/) or backslash(\), is rejected as invalid. Other characters are allowed.

File Content

Since file fragments are of variable size, **create big file fragments** in order to reduce the total number of file fragments and make the best use of a block. Writing or appending small amounts of data to a file is not optimal, since such an approach creates a large number of allocation information records. They consume free space and the required processing time results in a slow file access time.

When the file content is modified, the old file content is invalidated and a new file fragment is allocated. a block is erased when all the data stored within has been invalidated.

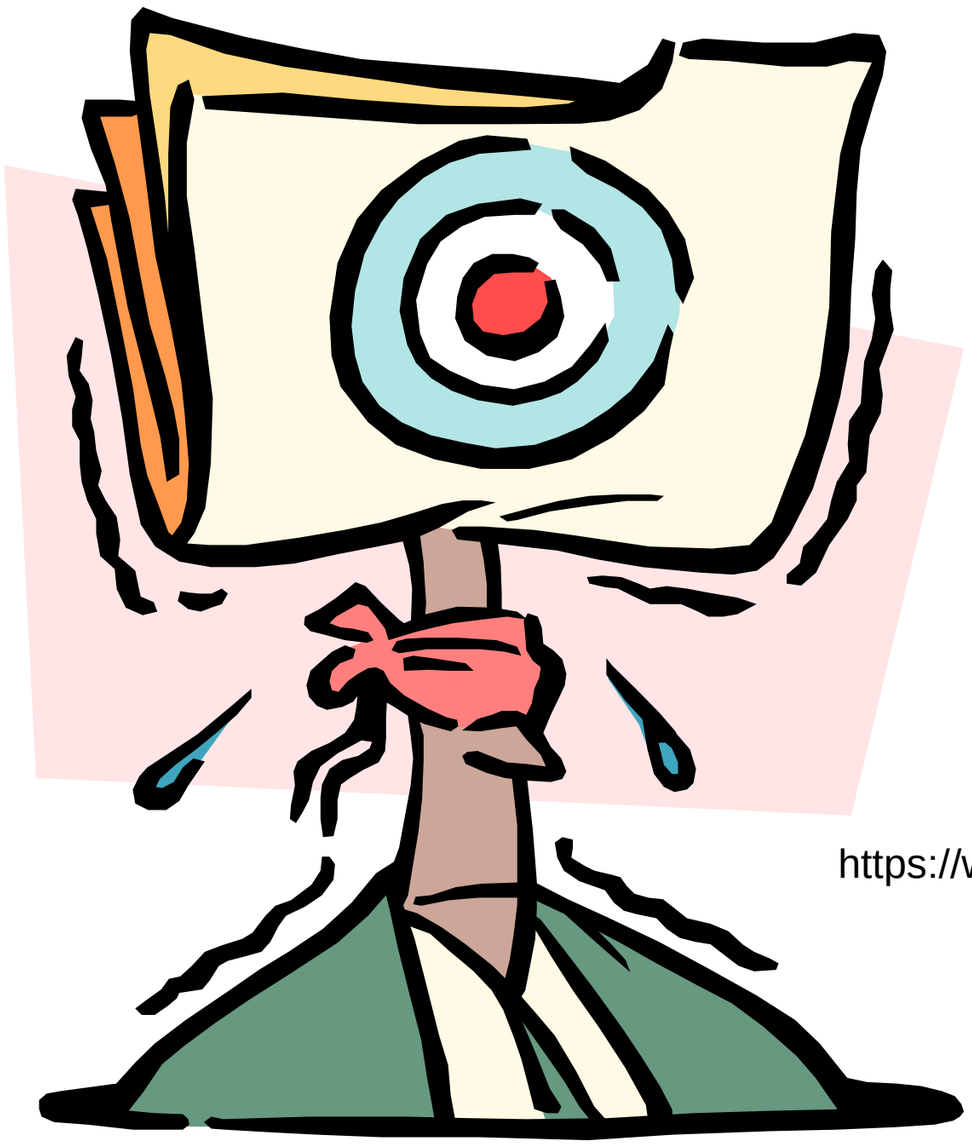
EMBEDDED FILE SYSTEMS

Limitations

- The following restrictions are applicable to the EFS:
- Maximum file name length is limited to 31 characters.
- Minimum block size should be 512 bytes or more.
- Directories or folders are not supported.
- Multiple active file handles per file are allowed only for files opened in read mode.
- Seeking (**fseek**) within files works only for files opened in read mode.
- File update modes (r+, w+, a+) (**fopen**) are not supported.
- Timestamp information is not supported for a file.
- Drive partitions are not supported.
- The EFS is not compatible with the FAT file system and cannot be used with a USB mass storage device.

EMBEDDED FILE SYSTEMS

Any Questions / Thank you



Shoot!

https://www.keil.com/pack/doc/mw/FileSystem/html/emb_fs.html