

嵌入式系统

An Introduction to Embedded System

第八课、嵌入式文件系统

课程大纲

 嵌入式文件系统概述

 闪速存储器硬件特性简介

 闪速存储器文件系统简介

 嵌入式Linux根文件系统简介

嵌入式文件系统介绍

- 嵌入式文件系统是在嵌入式系统中应用的文件系统，是嵌入式系统的一个重要组成部分。
- 随着嵌入式系统的广泛应用，对数据存储和管理提出了很高的要求，嵌入式文件系统的重要性不断突出。
- 嵌入式文件系统与桌面通用文件系统有较大差异：
 - 文件系统占用资源应尽可能小；
 - 满足可移动和便于携带的要求；
 - 满足断电后的数据完整性保护；
 - 满足抗辐射、单粒子翻转纠错；
 - 满足存储节能管理与设计需求。

嵌入式文件系统主要分类

□ ROM文件系统

- ROMFS是一种只读文件系统，因而可以做得很小

□ 磁盘文件系统

- FAT16、FAT32、Ext2FS、NTFS

□ Flash文件系统

- TrueFFS、MFFS
- JFFS/JFFS2、YAFFS/YAFFS2
- FAT16、FAT32、NTFS、exFAT

□ 内存文件系统

- RAMFS



课程大纲

 嵌入式文件系统概述

 闪存存储器硬件特性简介

 闪存存储器文件系统简介

 嵌入式Linux根文件系统简介

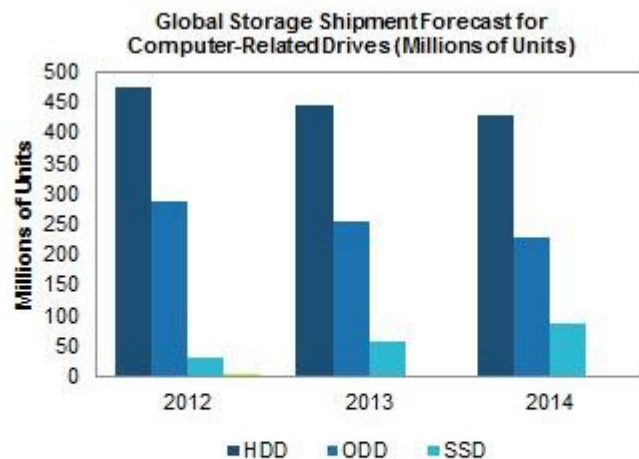
闪存存储器（固态硬盘）概述(1/2)

- 闪存（Flash）是一种固态非易失性存储器，主要依靠储存电荷保存数据，而不是移动电磁介质。
- Flash没有移动部分并且相对稳定，具有抗机械震动、轻巧、紧凑、节能等良好特性。
- 随着闪存容量的增加，价格的下降，越来越多的嵌入式系统采用闪存作为存储设备，如：电子盘（U盘）、手机、PDA、MP3、数码相机等，并且主板BIOS也已采用闪存。

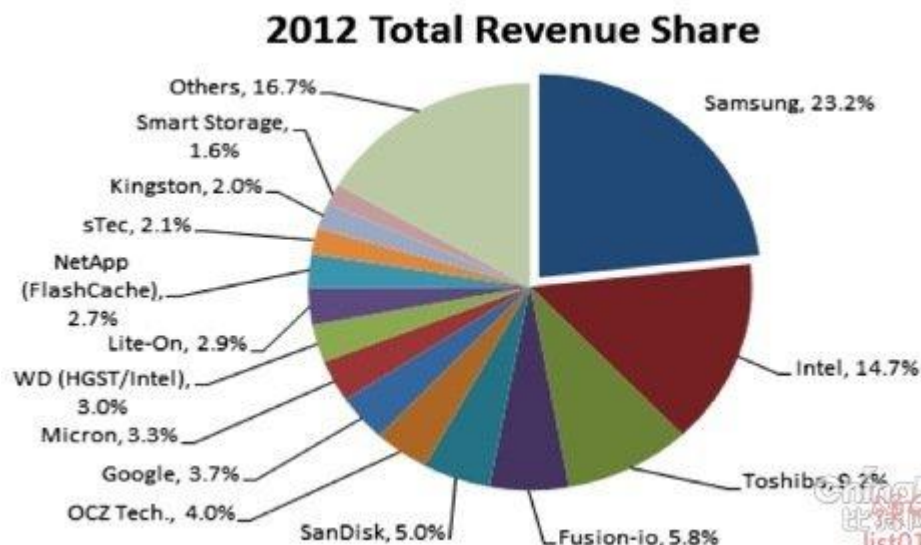


闪存存储器（固态硬盘）概述(2/2)

- 闪存产业正处于积极的扩张模式，2013年出货5700万，同比增长82%。
- HIS iSuppli公司发布的市场研究报告显示：2012年至2017年之间，闪存的出货量将增长600%。

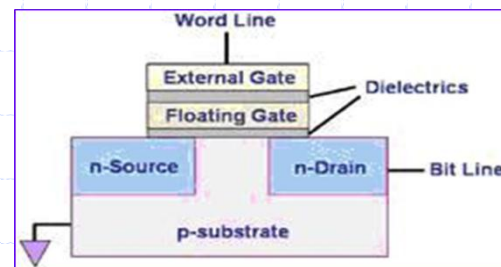
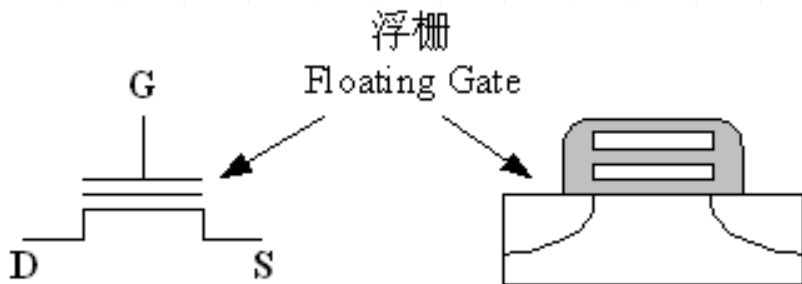


Source: IHS Technology, January 2014



闪存硬件基本原理（1/2）

- 当前主流的非易失性存储器是：EEPROM、闪速存储器，两者都是电可擦写的，基于浮栅MOS晶体管构造而成。

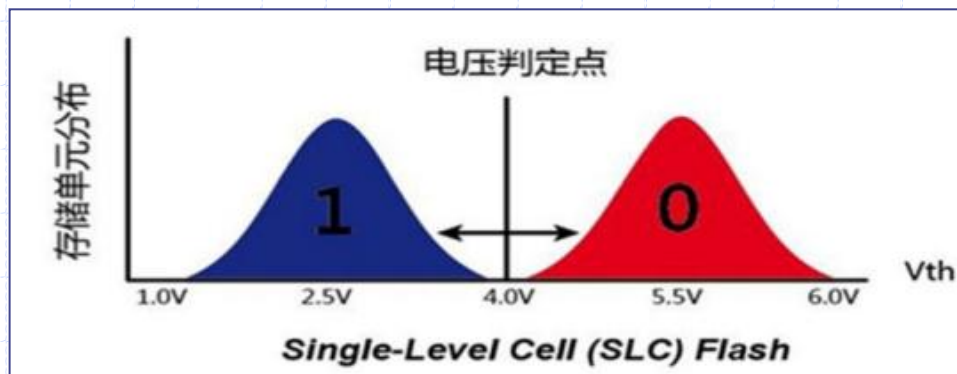
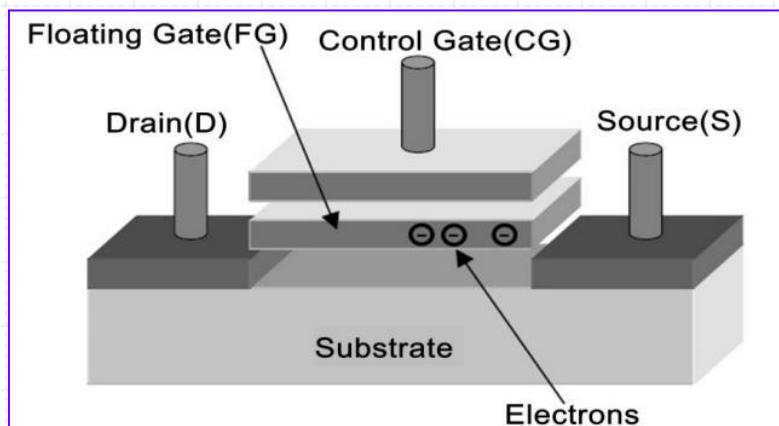


- 浮栅MOS晶体管的特点是门限值可以根据浮栅捕获的电荷来调节，电荷捕获技术是非常可靠的，基于浮栅MOS的存储器可以经受1000000次重写操作，保存时间长达10年。
- EEPROM可以一次性擦除和重写一个单元，闪存以块（block）为单位进行擦除，闪存的擦除、读、写速度更快。

闪存硬件基本原理 (2/2)

□ 闪存数据存储原理

- 数据在闪存单元中以电荷方式存储。
- 存储电荷的多少，取决于外部门（external gate）所被施加的电压。
- 数据表示，以所存储的电荷的电压是否超过一个特定的阈值 V_{th} 来表示。

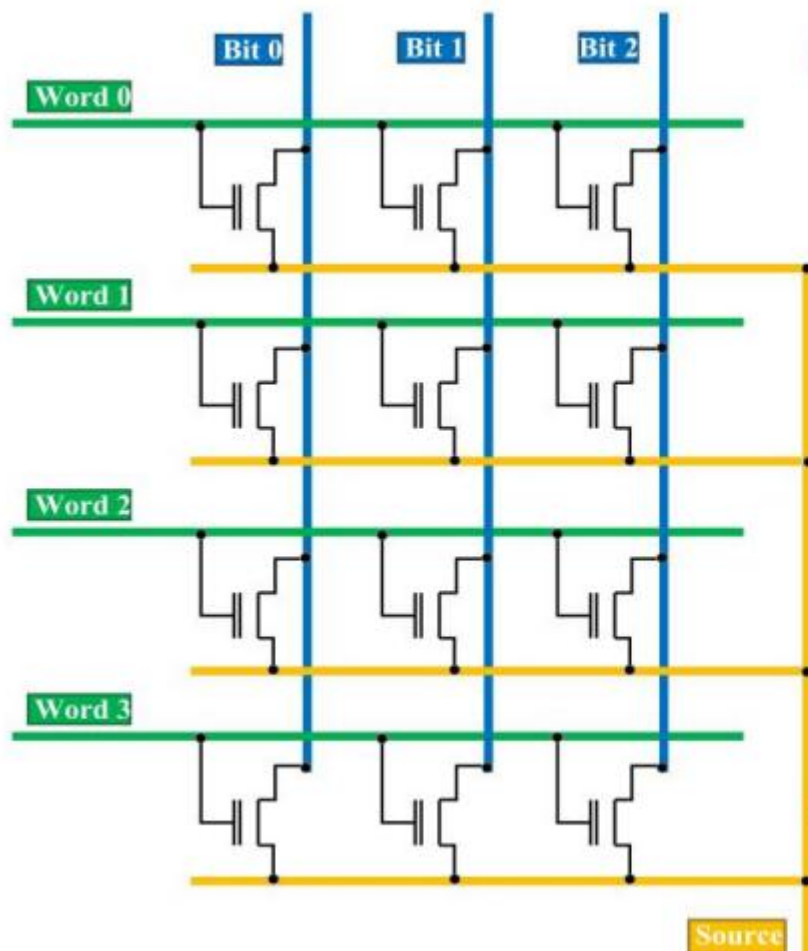


闪存发展及分类

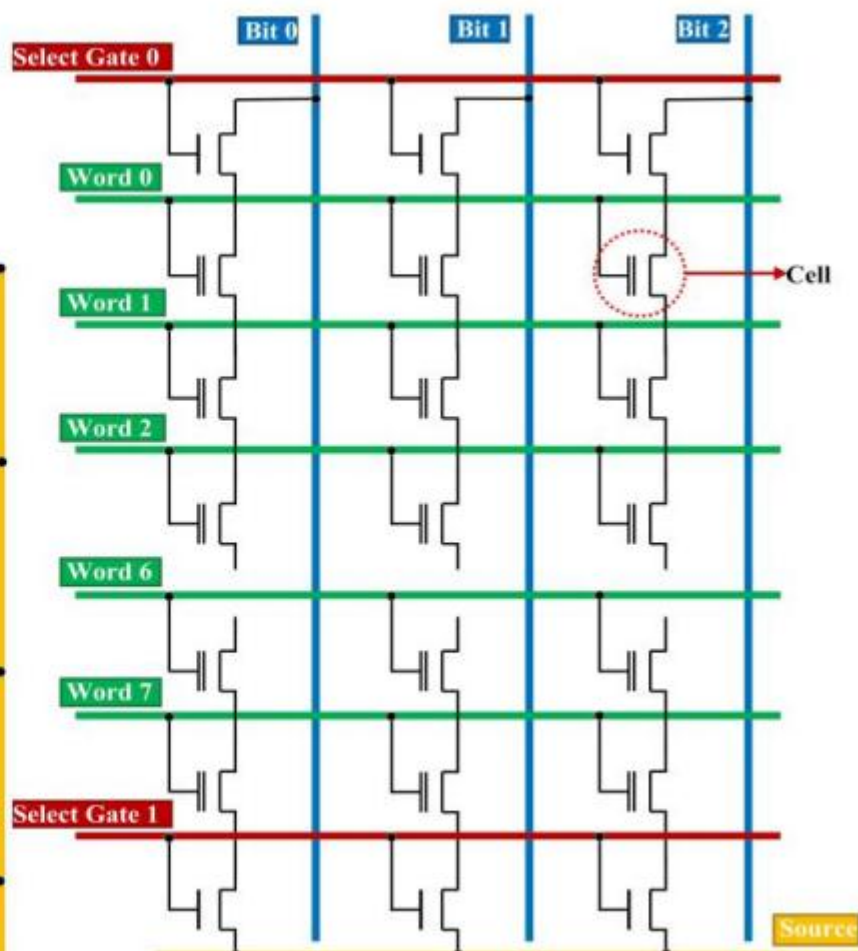
- 1984年闪存概念由东芝公司的Fujio Masuoka提出。
- Intel公司于1988年首先开发出NOR闪存产品并投放市场；1989年东芝公司开发出NAND闪存结构。
- 目前主要的闪存供应商有：Intel、三星、日立、东芝等。
- 闪存类型主要分为：
 - NAND、NOR、DINOR、AND。
 - NOR闪存作为EEPROM的替代品而设计，NAND型闪存专门为数据存储而设计。



NOR vs NAND闪存比较 (1/4)

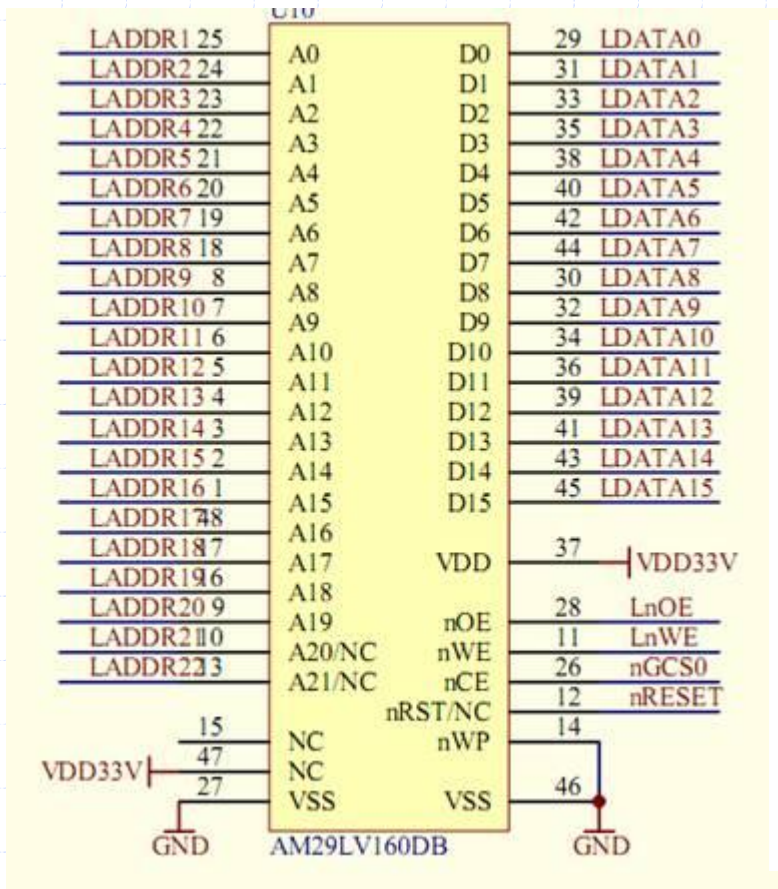


NOR闪存架构

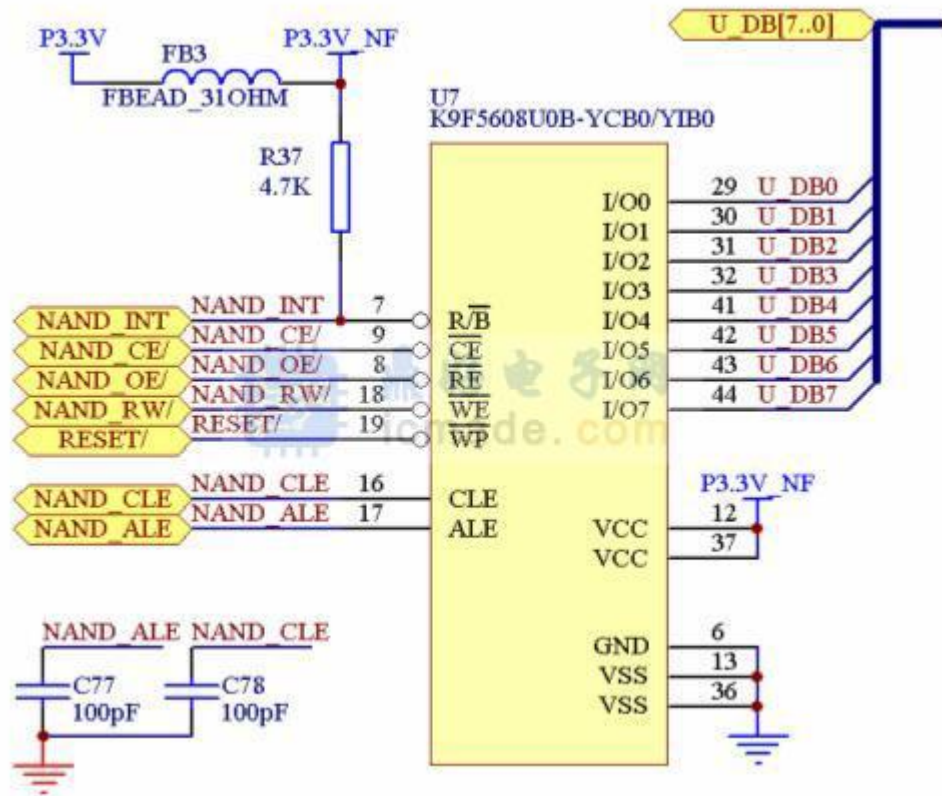


NAND闪存架构

NOR vs NAND闪存比较 (2/4)



NOR FLASH(XIP)



NAND FLASH(ΞΞXIP)

NOR vs NAND闪存比较（3/4）

□ NOR型闪存的特点：

- 具有独立的地址线、数据线，支持快速随机访问，容量较小；
- 具有芯片内执行（XIP，eXecute In Place）的功能，按照字节为单位进行随机写；
- NOR型闪存适合用来存储少量的可执行代码。

□ NAND型闪存的特点：

- 地址线、数据线共用，单元尺寸比NOR型器件小，具有更高的价格容量比，可以达到高存储密度和大容量；
- 读、写操作单位采用512字节的页面；
- NAND更适合作为高密度数据存储。

NOR vs NAND闪存比较（4/4）

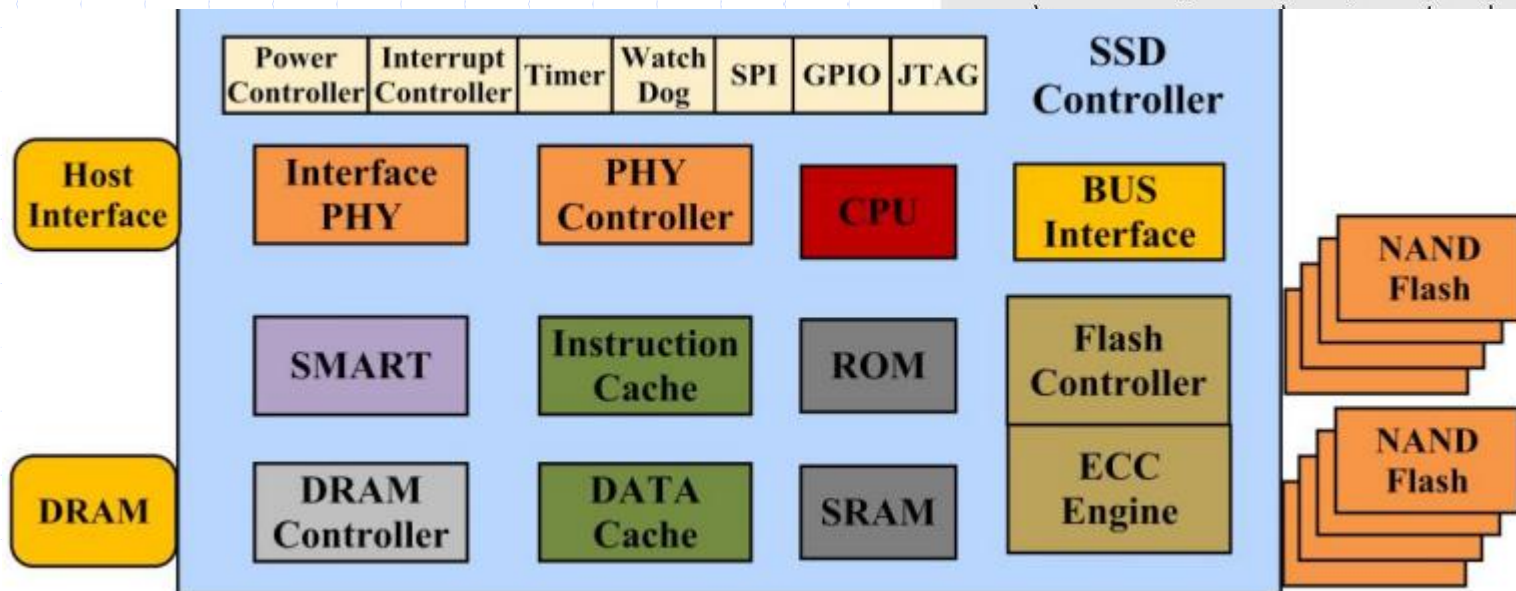
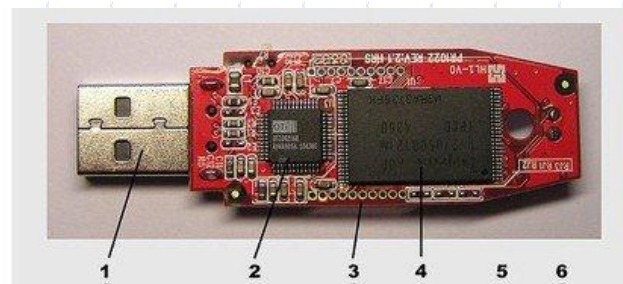
性能参数	NAND型	NOR型
读操作的时间	3.5MB/sec	15MB/sec
写操作的时间	0.65MB/sec	0.15MB/sec
擦除操作的时间	2ms	1sec
擦除大小	8-32KB	64-128KB
擦除次数限制	1,000,000 次	100,000 次

- ❑ 与NOR型器件相比，NAND型器件的写入、擦除速度较快。
- ❑ NOR闪存带有SRAM接口，可以实现随机写。
- ❑ NAND器件使用I/O口串行存取数据，操作单元为512字节，可取代硬盘或其他块设备，需要Memory Technology Devices(MTD)驱动。
- ❑ NAND型具有更高的擦除上限，对于经常大容量数据存储的应用来说，能够提供更长的使用寿命。

NAND闪存构成原理（1/2）

□ NAND闪存总体构成

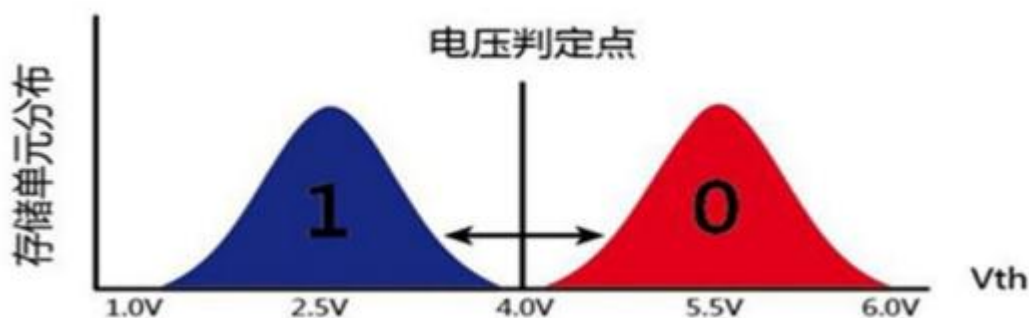
- 控制器
- 存储芯片
- 缓存：DRAM
- 物理接口



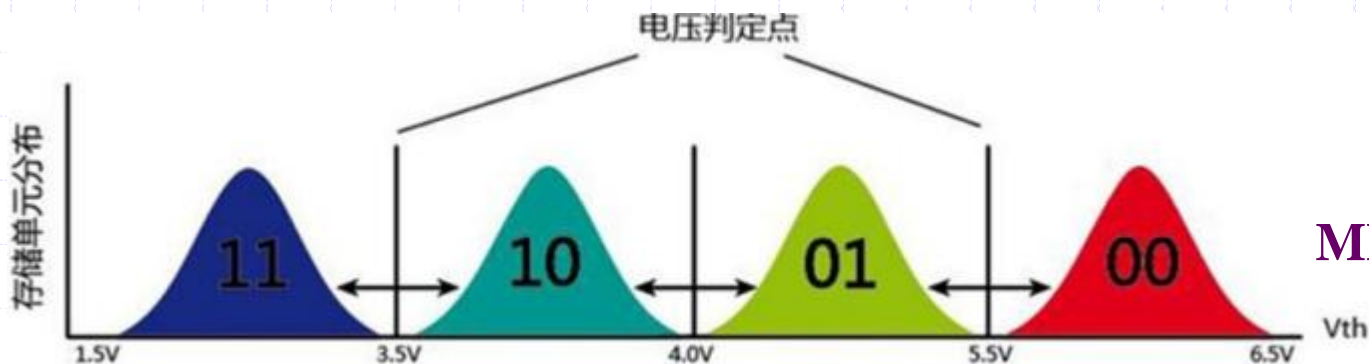
NAND闪存构成原理（2/2）

□ NAND闪存的存储结构设计

- SLC（Single Level Cell单层单元）
- MLC（Multi-Level Cell多层单元）：2003年
- TLC（Trinary-Level Cell三层单元）：2009年



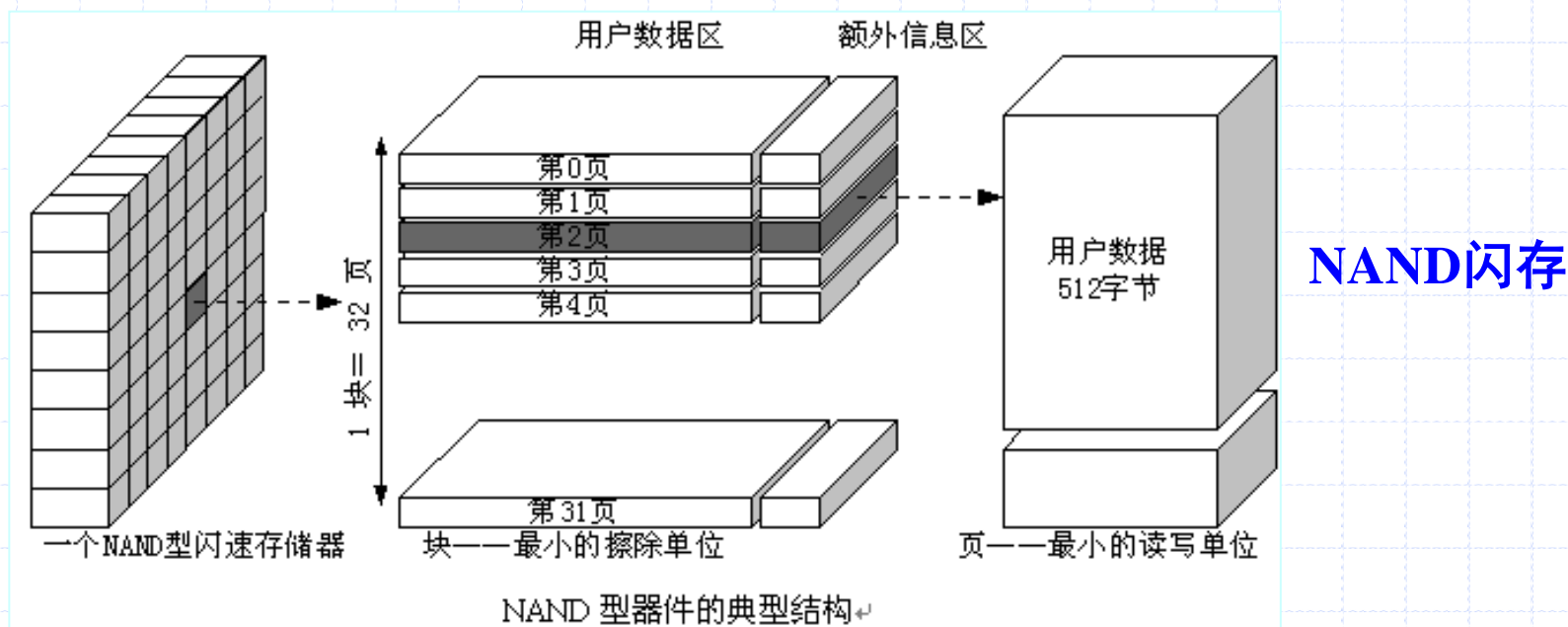
SLC数据存储



MLC数据存储

NAND闪存管理中的问题 (1/4)

□ 写前需先擦除，擦除单元（Block）> 读写单元（Page）：



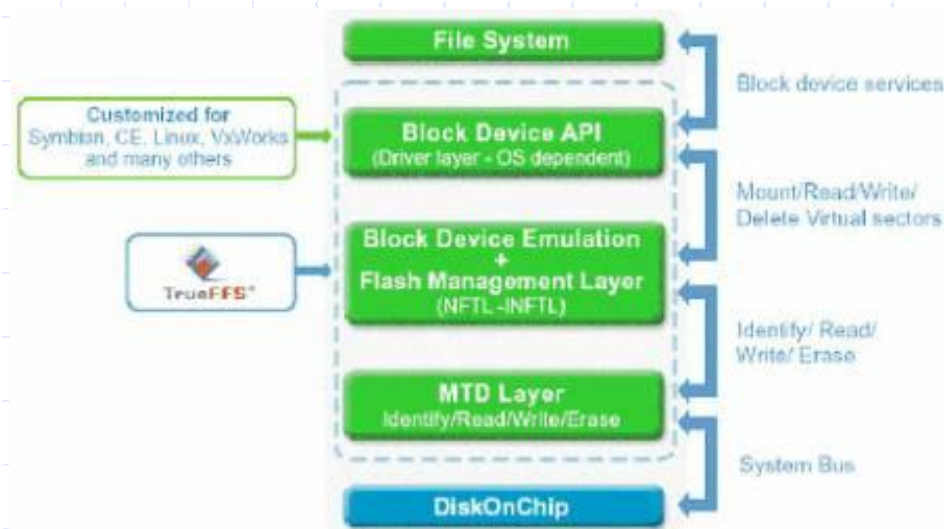
□ 数据更新策略：

- 本地更新（in-place update）
- 非本地更新（non-in-place update）

□ 垃圾回收问题

NAND闪存管理中的问题 (2/4)

- ❑ 损耗均衡(Wear Leveling): 闪存上的每个块都具有擦除次数的上限, 被称为擦除周期计数(erase cycle count)。
- ❑ NAND: 1,000,000 次 / NOR: 100,000 次
- ❑ 为了提高闪存寿命, 并减少某些块提前损坏, 闪存管理算法设计时应尽可能减少擦除次数, 并将擦除操作均布整个芯片。



NAND闪存管理中的问题 (3/4)

- 位翻转(Bit Flipping): 所有闪速存储器都受到位翻转现象的困扰, 表现为一个bit位发生翻转。
- 位翻转的产生原因
 - **Drifting Effects:** A phenomena that slowly changes a cell's voltage level from its initial value.
 - **Program-Disturb Errors:** This is sometimes referred to as "over-program" effects. A programming operation on one page induces the flip of a bit on another, unrelated page.
 - **Read-Disturb Errors:** This effect causes a page read operation to induce a permanent change of a bit value in one of the bits read.
- 当存储器用于敏感信息存储时, 需使用错误探测/更正(EDC/ECC) 算法提供可靠性支持。

NAND闪存管理中的问题 (4/4)

- ❑ 坏块处理问题：NAND器件中的坏块是随机分布的，由于消除坏块的代价太高，因而使用NAND器件的初始化阶段进行扫描以发现坏块，并将坏块标记为不可用(非0xFF)。
- ❑ 掉电保护问题：文件系统应能保证在系统突然断电时，最大限度地保护数据，使文件恢复到掉电前的一个一致性状态。

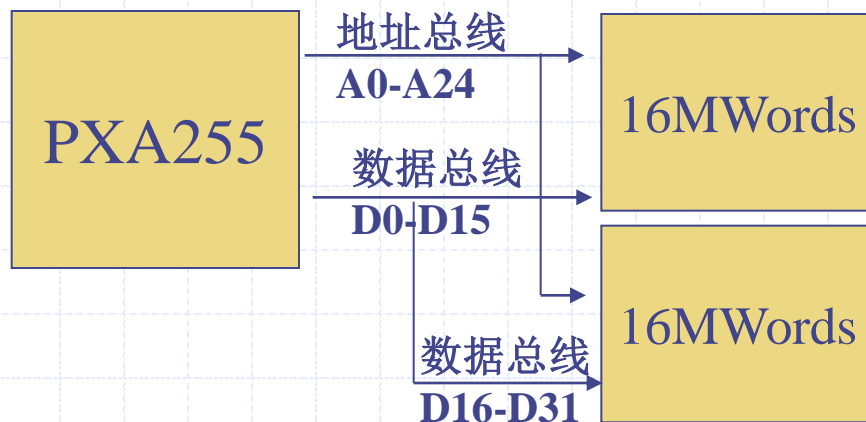


NOR闪存驱动分析 (1/5)

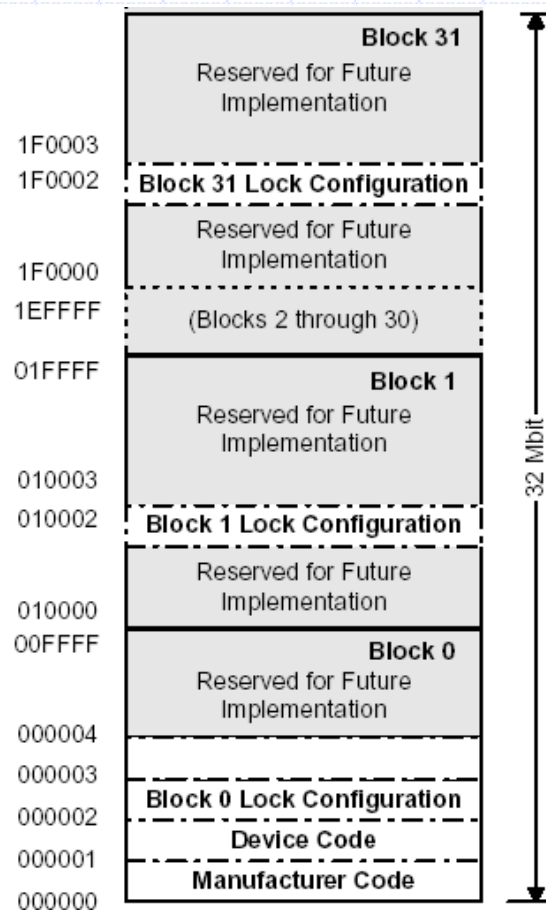
Intel的StrataFlash (TE28F128J3C) —NOR型芯片

芯片参数:

- 总容量: 2片16M×16位flash并联
- Block: 128K byte



Flash连接图

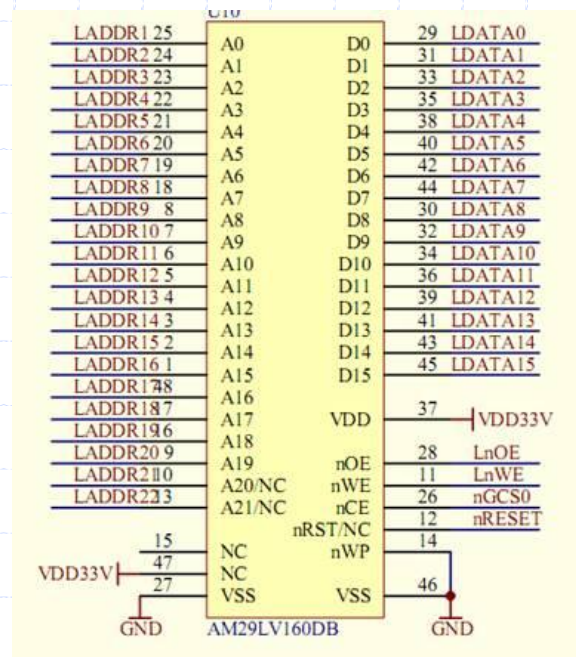


Flash分布图

NOR闪存驱动分析 (2/5)

□ Bootloader中对闪存操作的相关指令:

- flash [loader/kernel/root/ramdisk]
- erase [loader/kernel/root/ramdisk]
- lock [kernel/root/ramdisk]
- unlock



NOR闪存驱动分析 (3/5)

Intel的StrataFlash中的指令集定义:

读状态



Flash



Erase



lock

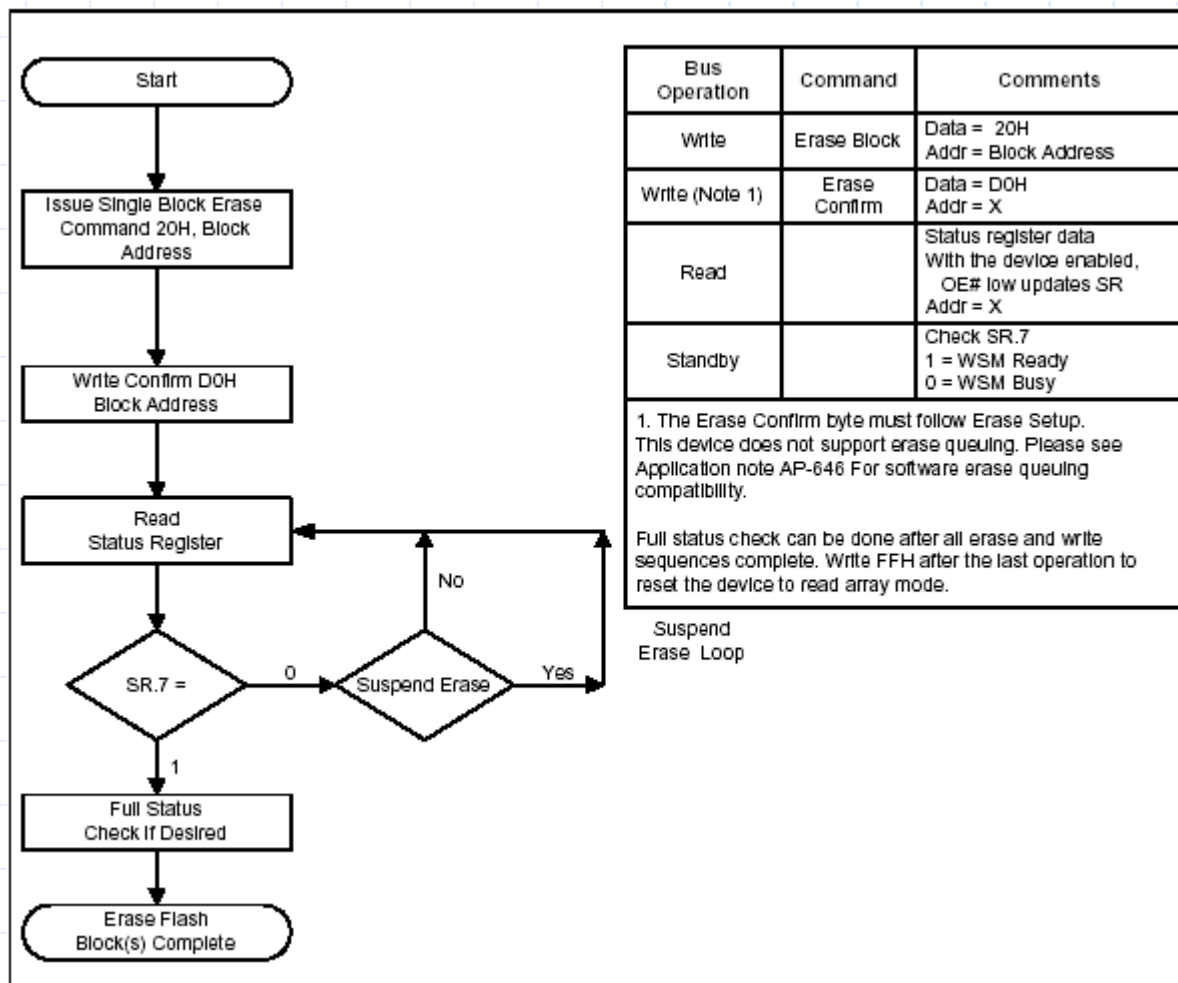
unlock



Command	Scaleable or Basic Command Set ⁽²⁾	Bus Cycles Req'd.	Notes	First Bus Cycle			Second Bus Cycle		
				Oper ⁽³⁾	Addr ⁽⁴⁾	Data ^(5,6)	Oper ⁽³⁾	Addr ⁽⁴⁾	Data ^(5,6)
Read Array	SCS/BCS	1		Write	X	FFH			
Read Identifier Codes	SCS/BCS	≥ 2	7	Write	X	90H	Read	IA	ID
Read Query	SCS	≥ 2		Write	X	98H	Read	QA	QD
<u>Read Status Register</u>	SCS/BCS	2	8	Write	X	70H	Read	X	SRD
Clear Status Register	SCS/BCS	1		Write	X	50H			
<u>Write to Buffer</u>	SCS/BCS	> 2	9, 10, 11	Write	BA	E8H	Write	BA	N
Word/Byte Program	SCS/BCS	2	12,13	Write	X	40H or 10H	Write	PA	PD
<u>Block Erase</u>	SCS/BCS	2	11,12	Write	BA	20H	Write	BA	D0H
Block Erase, Program Suspend	SCS/BCS	1	12,14	Write	X	B0H			
Block Erase, Program Resume	SCS/BCS	1	12	Write	X	D0H			
Configuration	SCS	2		Write	X	B8H	Write	X	CC
Set Read Configuration		2		Write	X	60H	Write	RCD	03H
<u>Set Block Lock-Bit</u>	SCS	2		Write	X	60H	Write	BA	01H
<u>Clear Block Lock-Bits</u>	SCS	2	15	Write	X	60H	Write	X	D0H
Protection Program		2		Write	X	C0H	Write	PA	PD

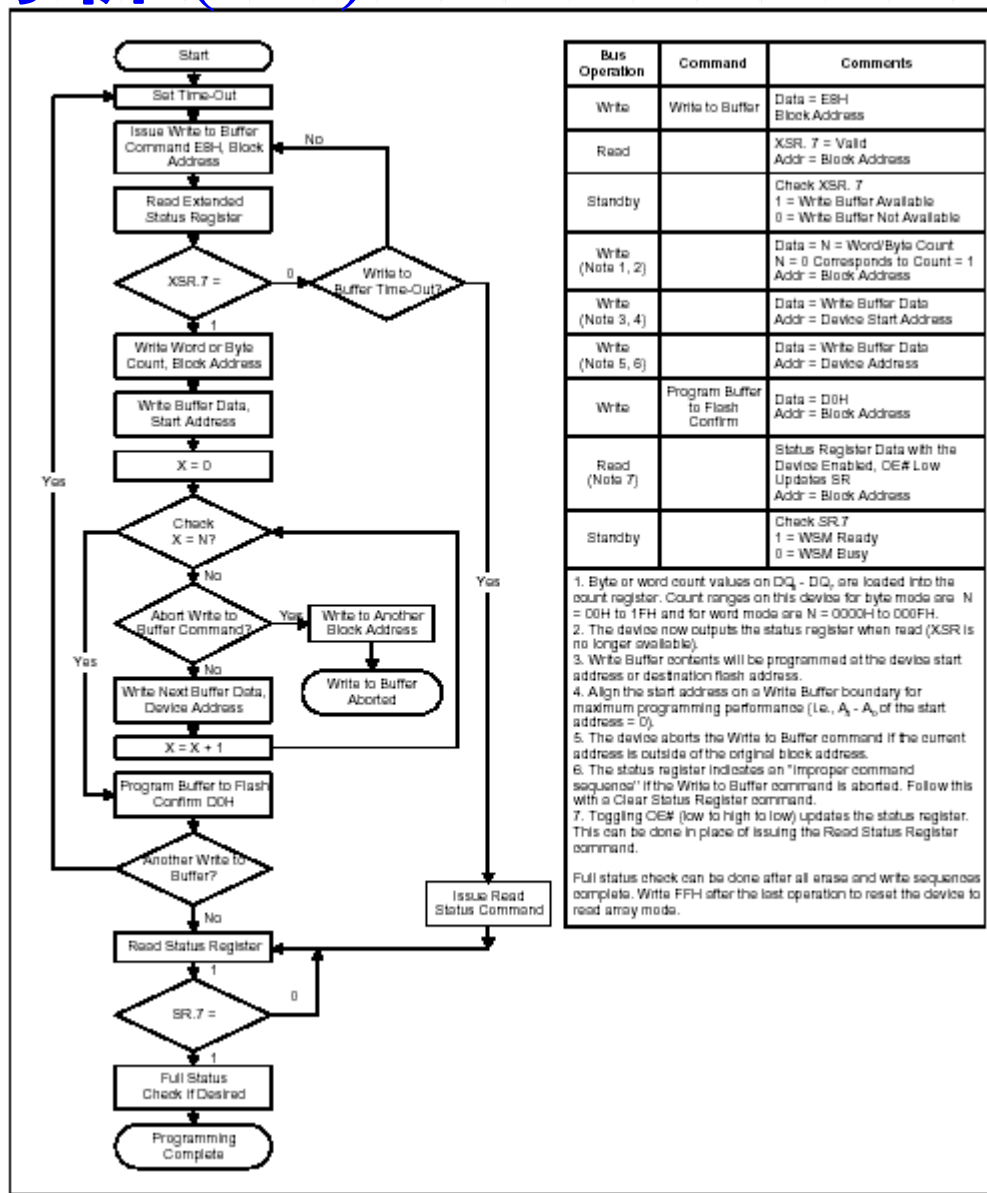
NOR闪存驱动分析 (4/5)

□ Block Erase处理流程: EraseFlashBlocks



NOR闪存驱动分析 (5/5)

Write to Buffer处理流程: DoWriteToFlashBlocks



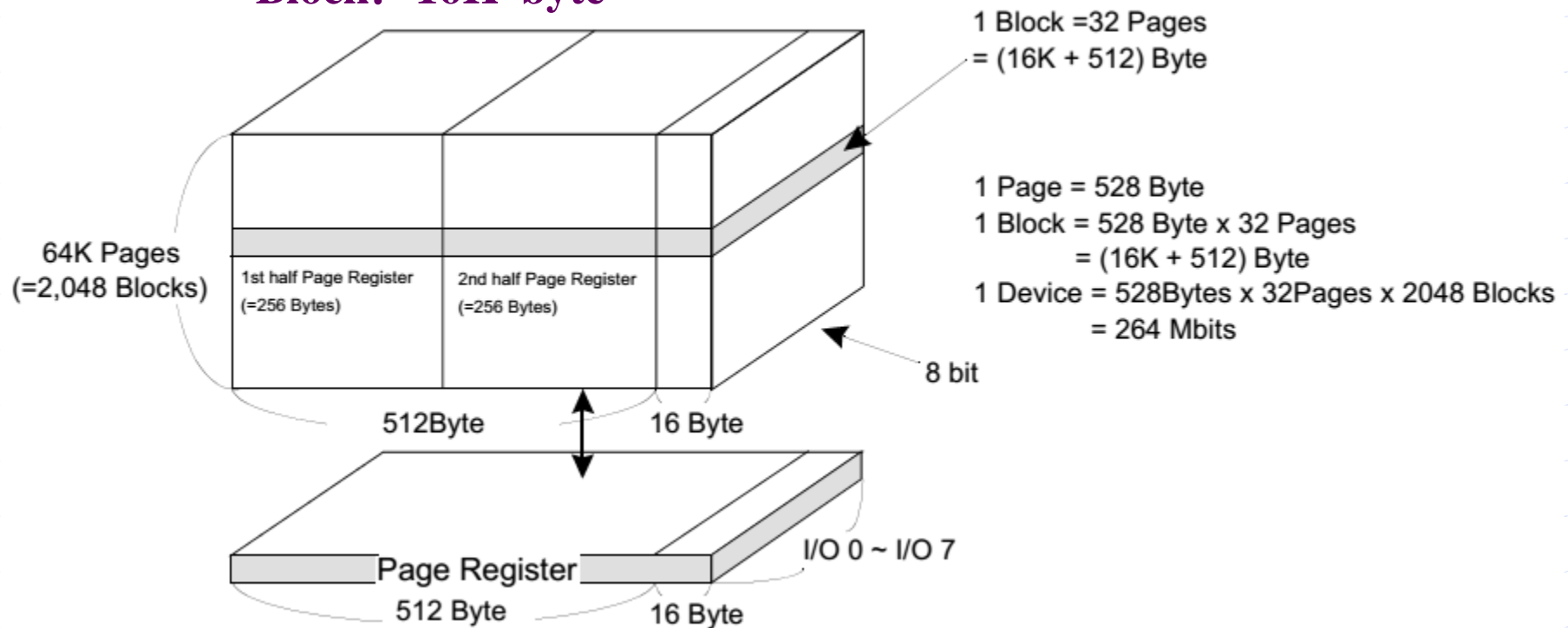
NAND闪存驱动分析 (1/10)

□ SAMSUNG的NAND Flash (K9F5608X0B (X8))

□ 芯片参数

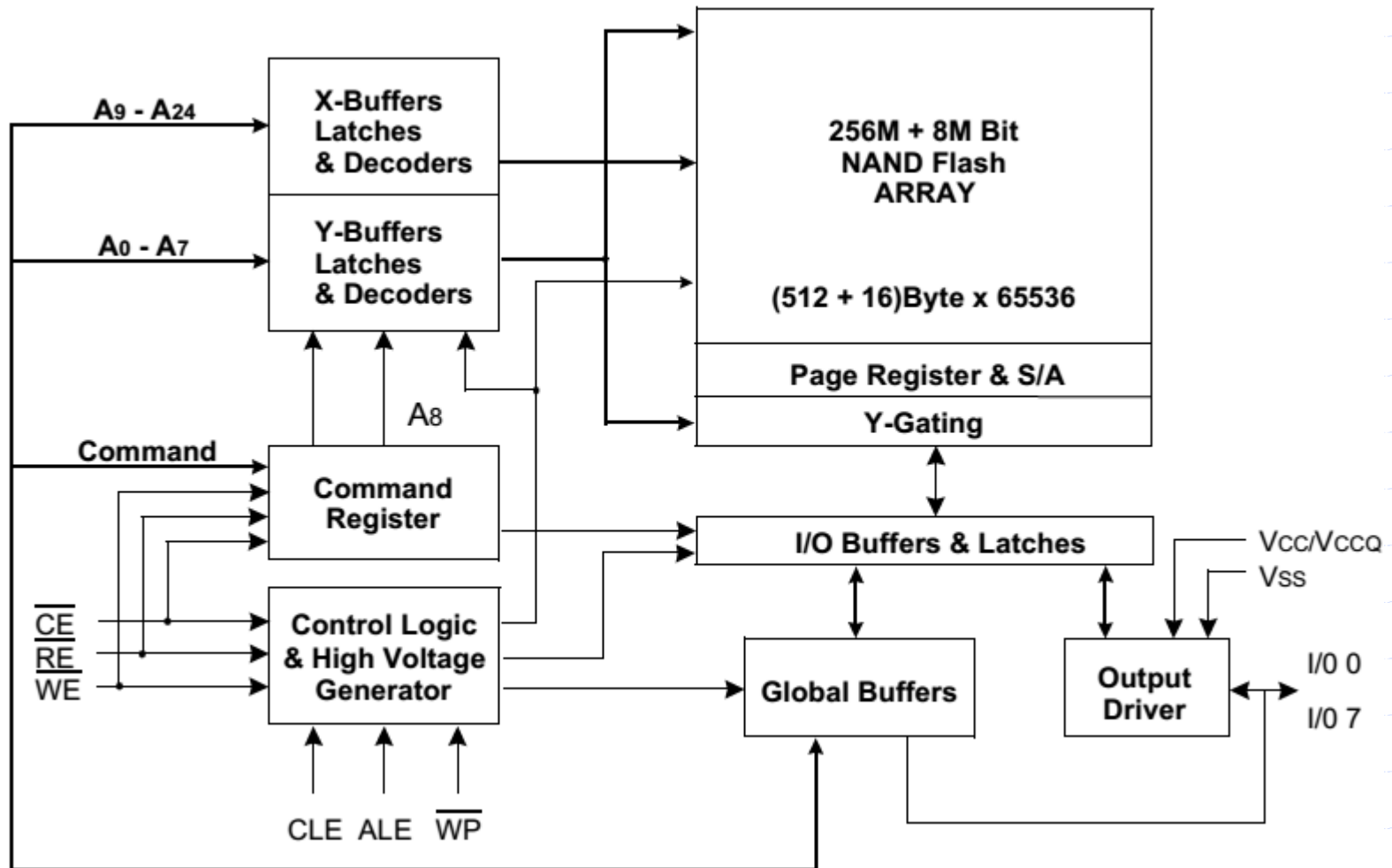
■ 总容量: $(32\text{M} \times 8)$ bit

■ Block: 16K byte



K9F5608X0B (X8) Array organization

NAND闪存驱动分析 (2/10)



Functional block diagram

NAND闪存驱动分析 (3/10)

	I/O 0	I/O 1	I/O 2	I/O 3	I/O 4	I/O 5	I/O 6	I/O 7	Column Address Row Address (Page Address)
1st Cycle	A0	A1	A2	A3	A4	A5	A6	A7	
2nd Cycle	A9	A10	A11	A12	A13	A14	A15	A16	
3rd Cycle	A17	A18	A19	A20	A21	A22	A23	A24	

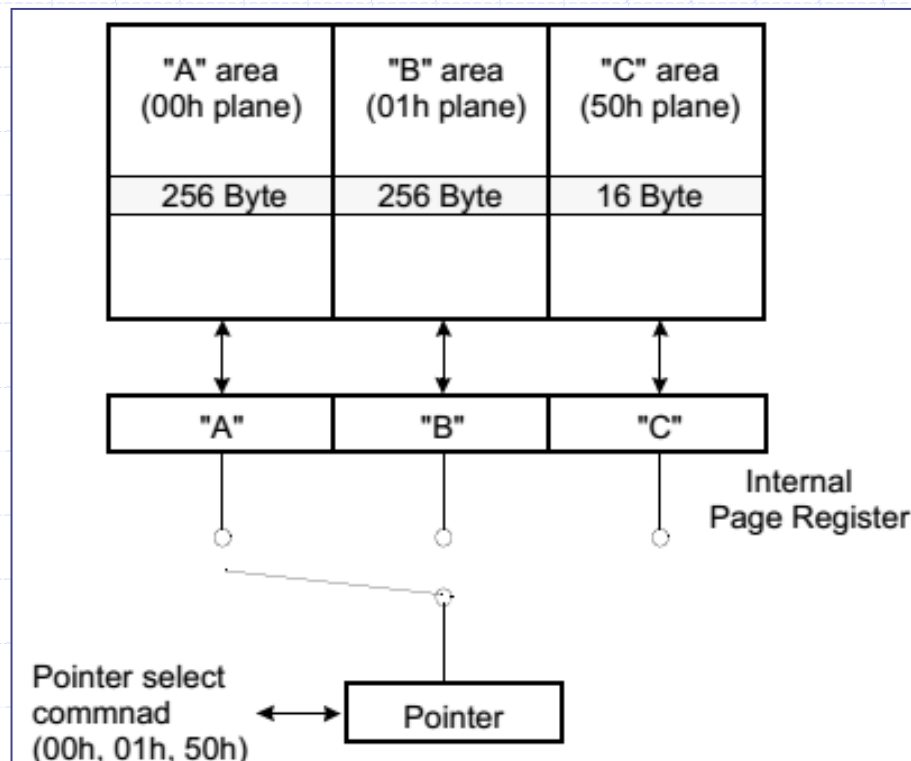
Address Cycles

- ❑ Column Address : Starting Address of the Register.
- ❑ 00h Command(Read) : Defines the starting address of the 1st half of the register.
- ❑ 01h Command(Read) : Defines the starting address of the 2nd half of the register.
- ❑ A8 is set to "Low" or "High" by the 00h or 01h Command.

NAND闪存驱动分析 (4/10)

Command	Pointer position	Area
00h	0 ~ 255 byte	1st half array(A)
01h	256 ~ 511 byte	2nd half array(B)
50h	512 ~ 527 byte	spare array(C)

Destination of the pointer

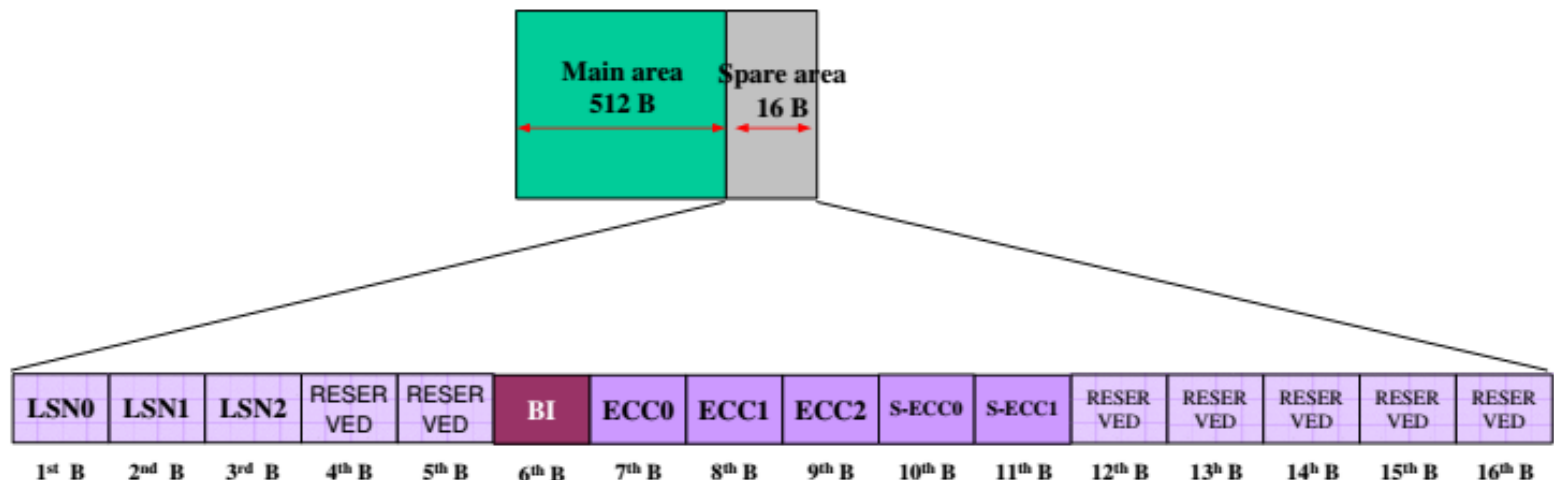


Block Diagram of Pointer Operation

NAND闪存驱动分析 (5/10)

Category 1. *Small Page(528B) & X8 org. NAND Flash*

- Device : 64Mb, 128Mb, 256Mb, 512Mb, 1Gb DDP



- > LSN : Logical Sector Number
- > ECC0,ECC1,ECC2 : ECC code for Main area data
- > S_ECC0,S_ECC1 : ECC code for LSN data
- > BI : Bad block Information

Samsung spare area assignment standard

NAND闪存驱动分析 (6/10)



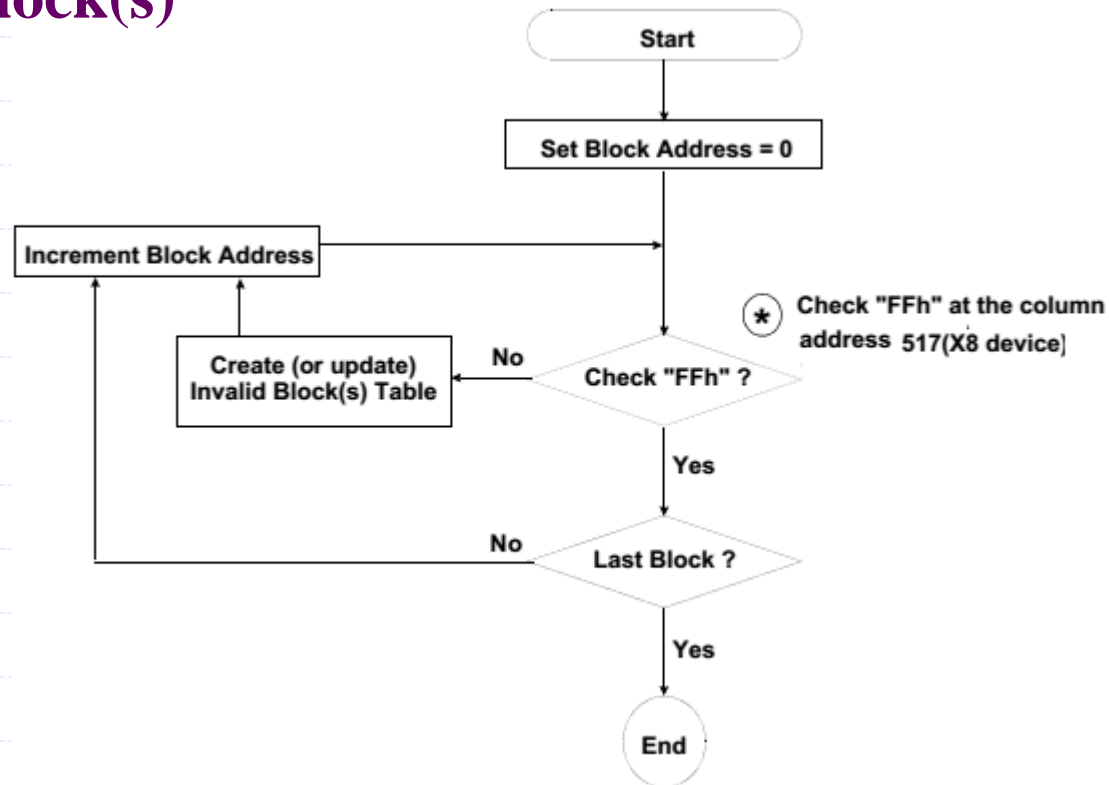
Function	1st. Cycle	2nd. Cycle	Acceptable Command during Busy
Read 1	00h/01h ⁽¹⁾	-	
Read 2	50h	-	
Read ID	90h	-	
Reset	FFh	-	O
Page Program	80h	10h	
Copy-Back Program	00h	8Ah	
Block Erase	60h	D0h	
Read Status	70h	-	O

Command sets

NAND闪存驱动分析 (7/10)

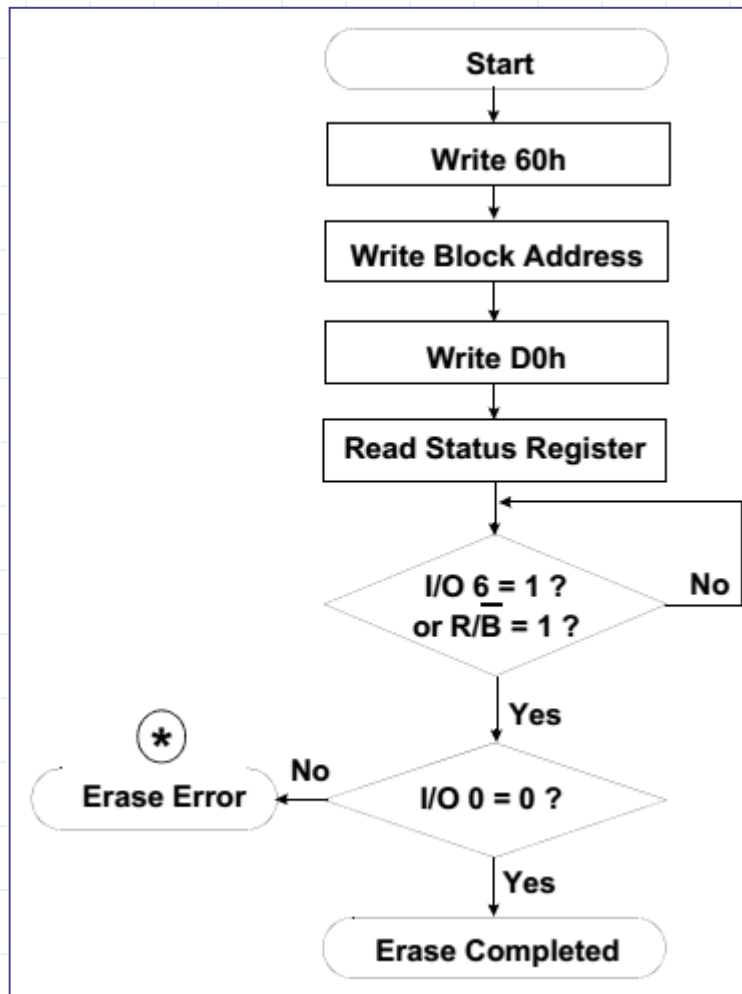
Identifying Invalid Block(s)

- The invalid block(s) status is defined by the **6th byte** in the spare area.
- The 1st block(00h block address) is fully guaranteed to be a valid block.

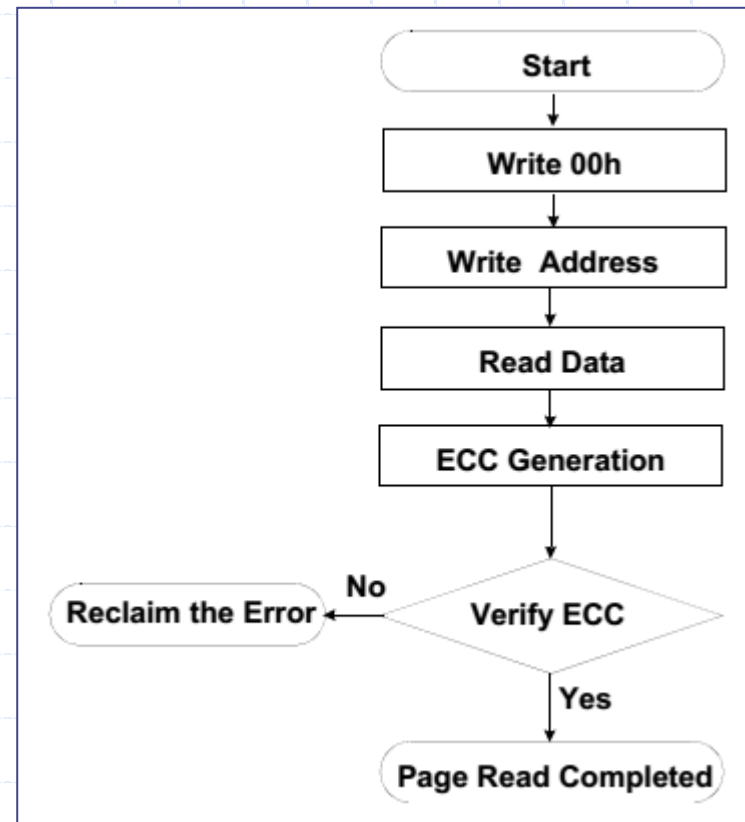


Flow chart to create invalid block table

NAND闪存驱动分析 (8/10)

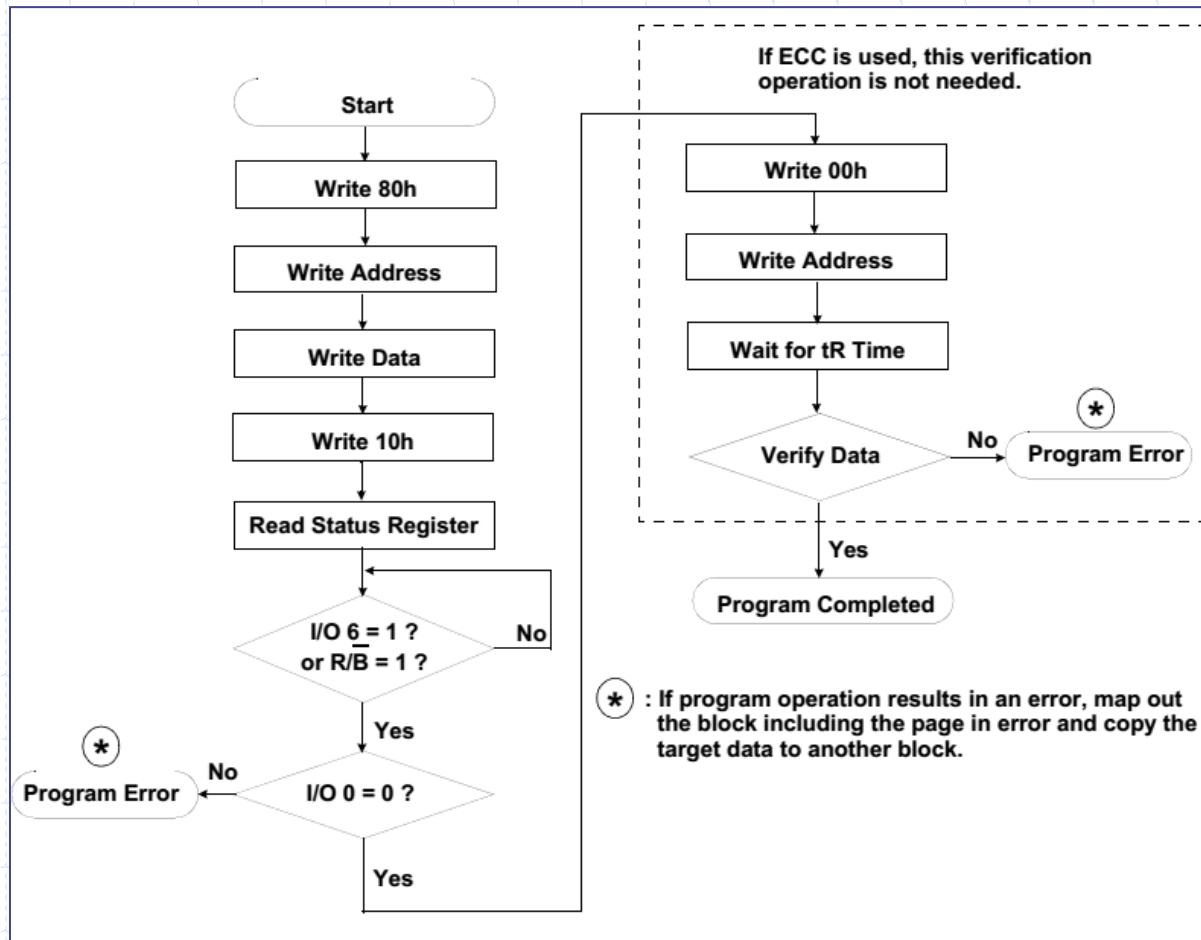


Erase Flow Chart



Read Flow Chart

NAND闪存驱动分析 (9/10)

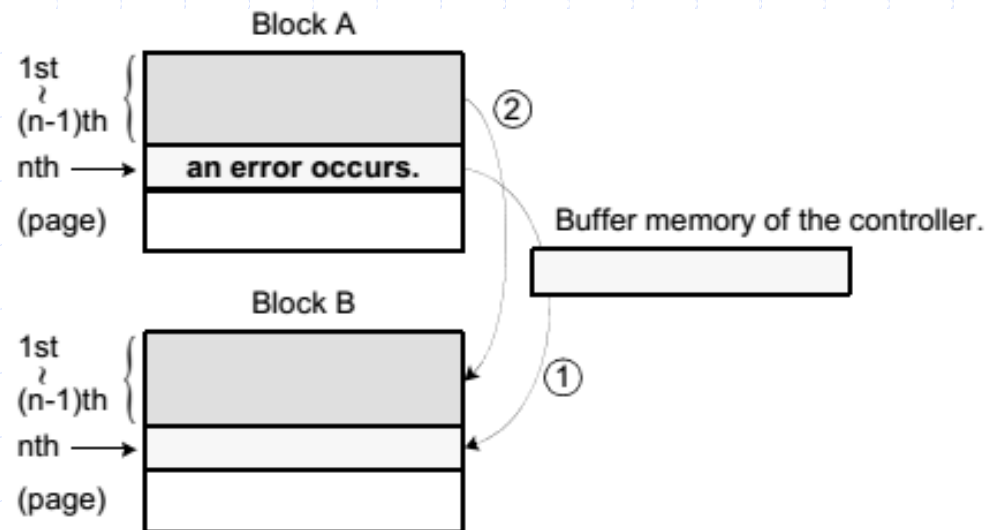


Program Flow Chart

NAND闪存驱动分析 (10/10)





❑ Error in write or read operation

Failure Mode		Detection and Countermeasure sequence
Write	Erase Failure	Status Read after Erase --> Block Replacement
	Program Failure	Status Read after Program --> Block Replacement Read back (Verify after Program) --> Block Replacement or ECC Correction
Read	Single Bit Failure	Verify ECC -> ECC Correction



Block Replacement

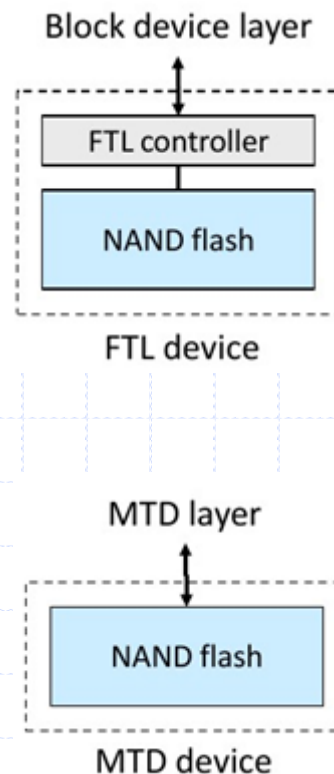
课程大纲

-  嵌入式文件系统概述
-  闪存存储器硬件特性简介
-  闪存存储器文件系统简介
-  嵌入式Linux根文件系统简介

闪存文件系统分类

□ 针对闪存设备的硬件特殊性，目前闪存文件系统主要有两种实现思路：

- 硬盘模拟法（MTD/FTL/FLASH）：将闪存设备模拟成具有每个扇区512字节的标准块设备，在此基础上使用成熟的磁盘文件系统进行管理。
- 直接实现法（MTD/FLASH）：直接对闪存设备进行操作，建立日志文件系统，避免模拟转化工作。

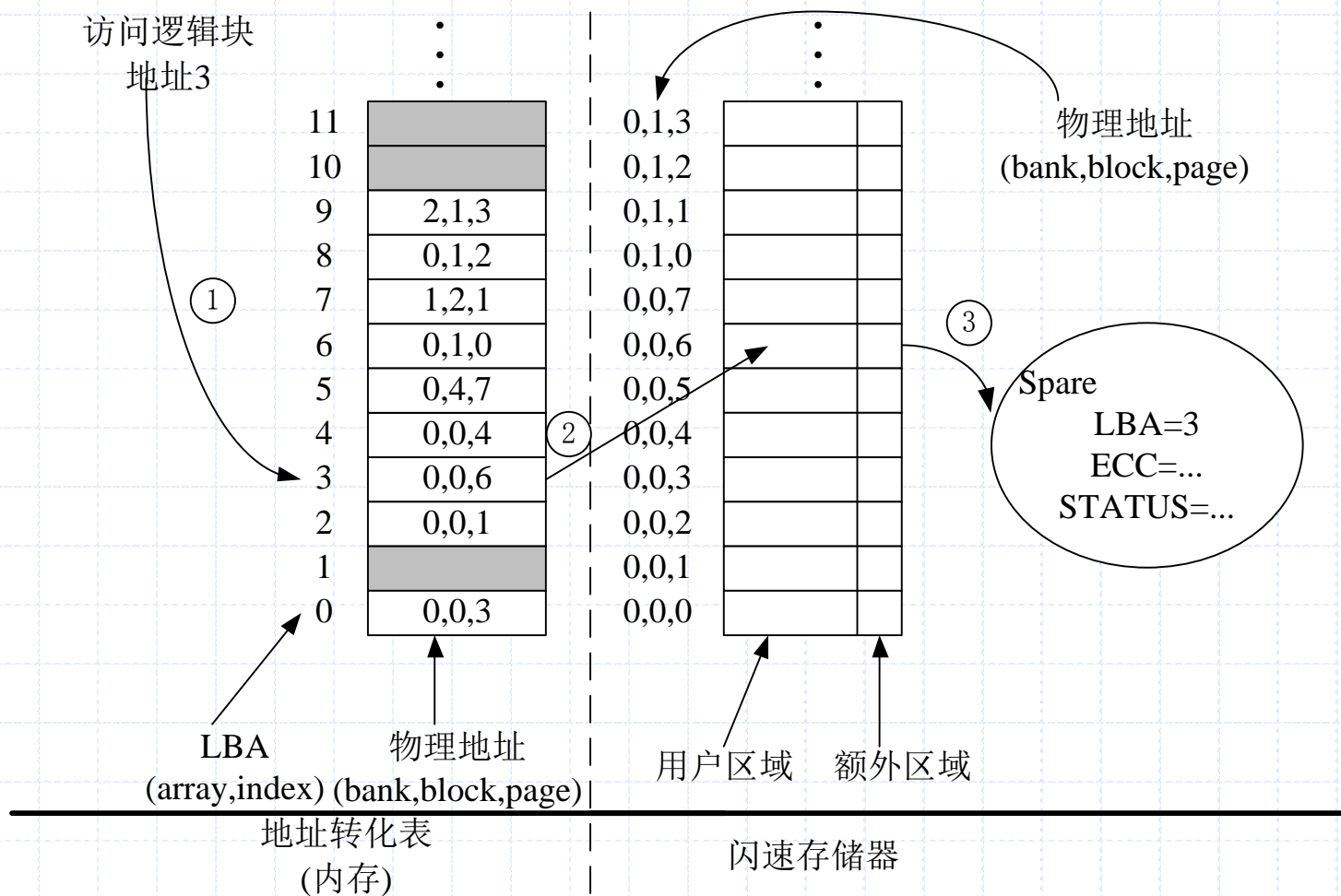


硬盘模拟法的闪存文件系统（1/4）

- ❑ 模拟方法是从模拟块设备到闪存芯片一对一的操作映射，如：模拟写请求的扇区操作时，读入整个擦除块，然后修改需要更新的部分，擦除重新写整个块。
- ❑ 硬盘模拟方法一般不考虑擦除的均衡性问题，闪存中的某些块可能会因为更新局部性迅速损坏。
- ❑ 系统一致性没有安全保证，内存随时可能断电，处于更新状态的信息会丢失，这种丢失将无法恢复。
- ❑ 为了提供均衡性和可靠的操作，模拟块设备的扇区存放在物理介质的不同位置上，地址转化层（**File Translation Layer**）用于记录当前每个扇区在模拟块设备上的位置。

硬盘模拟法的闪存文件系统（2/4）

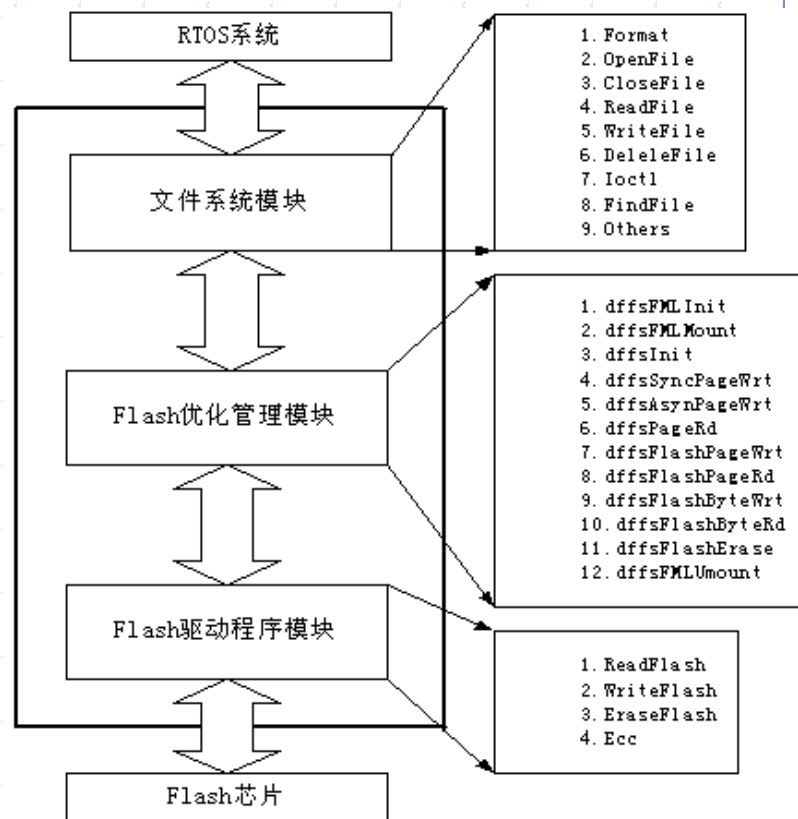
硬盘模拟法的扇区映射实例：



硬盘模拟法的闪存文件系统（3/4）

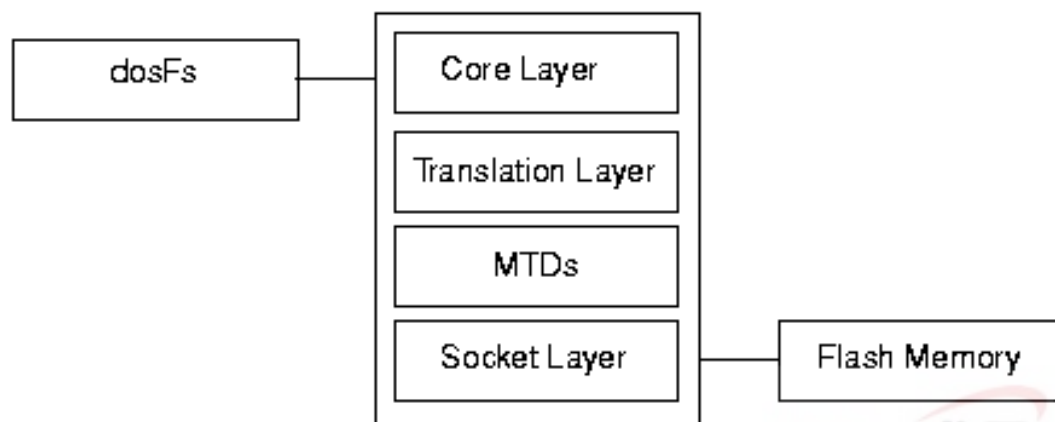
□ 硬盘模拟法的闪存文件系统实现主要分成三个层次：

- 设备驱动程序层：实现对flash设备最基本的操作。
- 地址转化层（FTL）：设备地址和逻辑地址的相互转化和数据对应关系的动态处理。
- 文件系统管理层：将文件系统操作转化成flash设备操作。



- 硬盘模拟方法通用性强，移植性好，适合商业系统，便于推广，已有多种成熟的产品被广泛应用，包括M-system公司的TrueFFS，Intel公司的MFFS等。

硬盘模拟法的闪存文件系统（4/4）



- TrueFFS是M-system公司为支持Flash存储器专门设计的一个Flash模拟硬盘程序包。
- TrueFFS采取多种提高Flash文件系统易用性的措施：
 - 动态和静态的损耗级别判定算法；
 - 安全算法保证在突然断电时数据完整性；
 - 解决位交换问题的Reed-Solomon纠错算法；
 - 自动的坏块管理；
 - 优化处理功能：减少擦除次数的算法、优化垃圾回收操作。

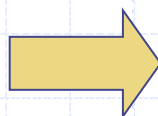
文件系统比较：TrueFFS vs FAT (1/4)

系统记录区	文件分配表	文件登记表	数据区
SR(System Record)	FAT(File Allocation Table)	FRT(File Register Table)	(Data Region)

■ FAT16文件系统简介

- 系统记录区 (SR)：存储器类型、容量、版本、数据区位置
- 文件分配表 (FAT)：存储区块占用/空闲、文件存储结构
- 文件登记表 (FRT)：文件代号、长度、属性、时间
- 数据区 (DR)：存放数据

■ FAT16文件系统簇大小



分区大小 FAT16簇大小

16MB-127MB 2KB

128MB-255MB 4KB

256MB-511MB 8KB

512MB-1023MB 16KB

1024MB-2047MB 32KB

文件系统比较：TrueFFS vs FAT (2/4)

□ 假设：

- 在256M的U盘上实现FAT16文件系统
- 簇大小为2KB
- 每个区块的擦除次数10万次

□ 向U盘中写入一个8MB文件

- 8M文件共占用： $8192\text{K}/2\text{K}=4\text{K}$ 个簇
- 每写一个簇，FAT表更新一次，共需更新4096次

□ FAT表一直位于固定扇区中，所以8MB的文件最多只能更新： $100000/4096=25$ 次

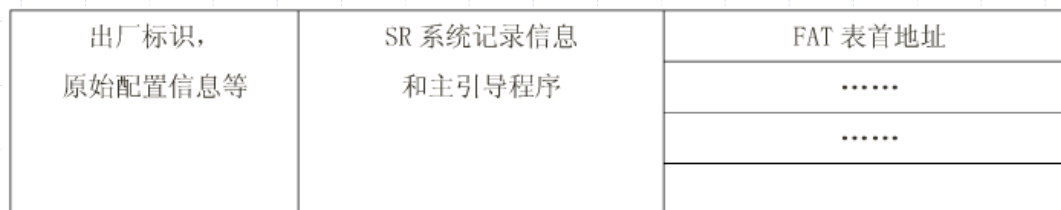
如果每天写入一个文件，那么1个U盘的寿命是25天！

文件系统比较：TrueFFS vs FAT (3/4)

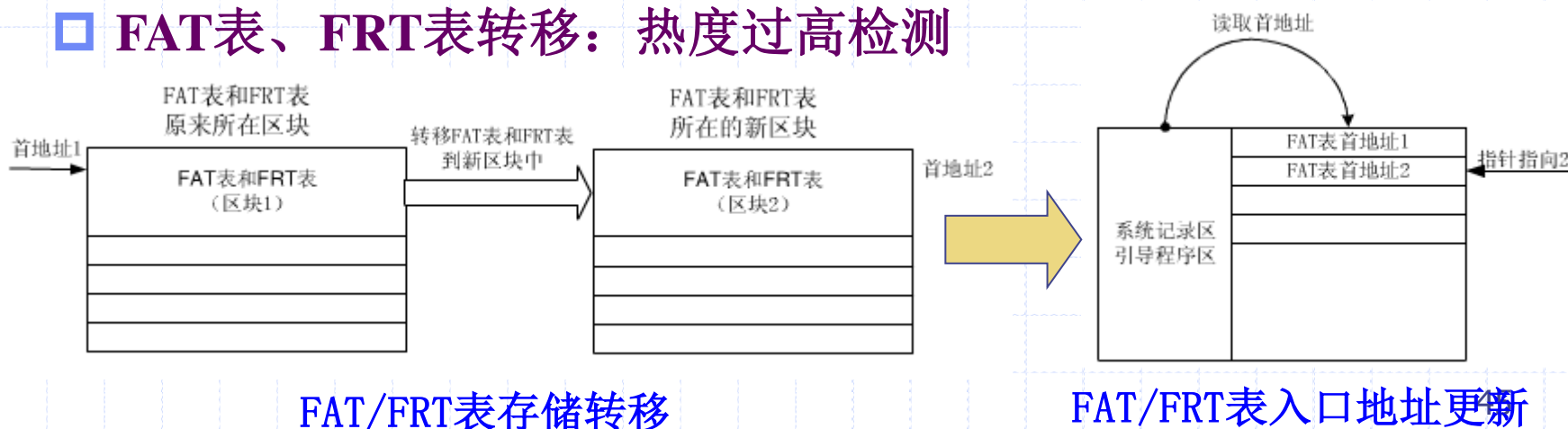
- 采用TrueFFS文件系统，损耗均衡算法不允许FAT表固定在某个扇区中，损耗平均分配给所有物理扇区。
- U盘寿命计算
 - **期望寿命** = (容量 × 总擦写次数 × 0.75) / 每天写入字节，其中：
 - 0.75：表示文件系统和TrueFFS管理结构的额外消耗系数
- 同样每天写入一个8MB文件，那么：
 - **期望寿命** = (256MB × 100000 × 0.75) / 8MB = 2400000(天) (约**6575**年)。
- TrueFFS惊人地延长了Flash器件的寿命。TrueFFS覆盖了大部分主流Flash芯片，考虑了各种芯片的不同擦写算法，效率较低。
- 微软的FAT16、FAT32、NTFS虽然实现了一定程度的损耗均衡算法，但是没有TrueFFS那么高效。

文件系统比较：TrueFFS vs FAT（4/4）

- 一种针对Flash优化的FAT16文件系统
 - FAT表、FRT表，可以在Flash的一定范围内移动
 - 在系统记录区（SR）中提供一系列FAT表的入口地址
- 系统记录区设计



□ FAT表、FRT表转移：热度过高检测



直接实现法的闪存文件系统（1/4）

- 使用硬盘模拟法的闪存文件系统，不能充分发挥Flash存储器特性，去掉FTL这一层转化，将对性能有很大提高，同时，磁盘文件系统的设计方法也不适合断电后数据的完整性保护。
- 在各种直接实现法的文件系统设计思路中，日志文件系统结构最符合Flash需要擦除的特性。

直接实现法的闪存文件系统（2/4）

- ❑ 日志文件系统结构采用了数据库系统中日志的概念，其特点是对数据的更新采用前向写入，即更新的数据部分写入空白块，而不是覆盖原先的数据，从而避免了数据块的擦除，保证了意外情况下数据存储的完整性与关联性。通过检查点、回滚技术等可以将数据恢复到某时刻的一致状态。
- ❑ 直接实现法的文件系统适用于资源少，速度要求快，以及可靠性要求高的应用场合，但缺乏通用性。

直接实现法的闪存文件系统（3/4）

□ JFFS / JFFS2/ JFFS3

- JFFS文件系统主要针对NOR Flash设计，是一种基于Flash的日志文件系统。
- JFFS2是JFFS的改进版本，改善了数据存取策略，优化了碎片整理性能，增加了数据压缩功能。
- Linux支持JFFS2文件系统，采用MTD（Memory Technology Device）驱动对flash的读、写、擦除等访问控制。

□ YAFFS / YAFFS2

- YAFFS针对NAND FLASH设计，与JFFS在垃圾搜集、文件压缩支持上有所区别。
- 自带NAND芯片驱动，可以不采用MTD，直接对文件进行操作。
- YAFFS2是YAFFS的改进版本，在性能上做了优化，支持对大页面（如2K）NAND的优化处理。

直接实现法的闪存文件系统（4/4）

■ UBIFS（Unsorted Block Image File System）





- 由IBM、Nokia工程师联合设计，2008年10月加入Linux核心2.6.27，是 Nokia N900智能手机的默认文件系统。
- 将索引节点存储在flash上，解决了索引的out-of-place update问题。
- UBIFS在设计与性能上较YAFFS2、JFFS2更适合MLC NAND

	JFFS2	YAFFS2	LogFS	UBIFS
Mounting time	Poor	Good	Excellent	Good
I/O performance	Good	Good	Fair	Excellent
Memory consumption	Fair	Excellent	Good	Fair
Wear-leveling	Good	Fair	N/A	Excellent
Tolerance to power-failure	Good	Good	Poor	Good

■ F2F

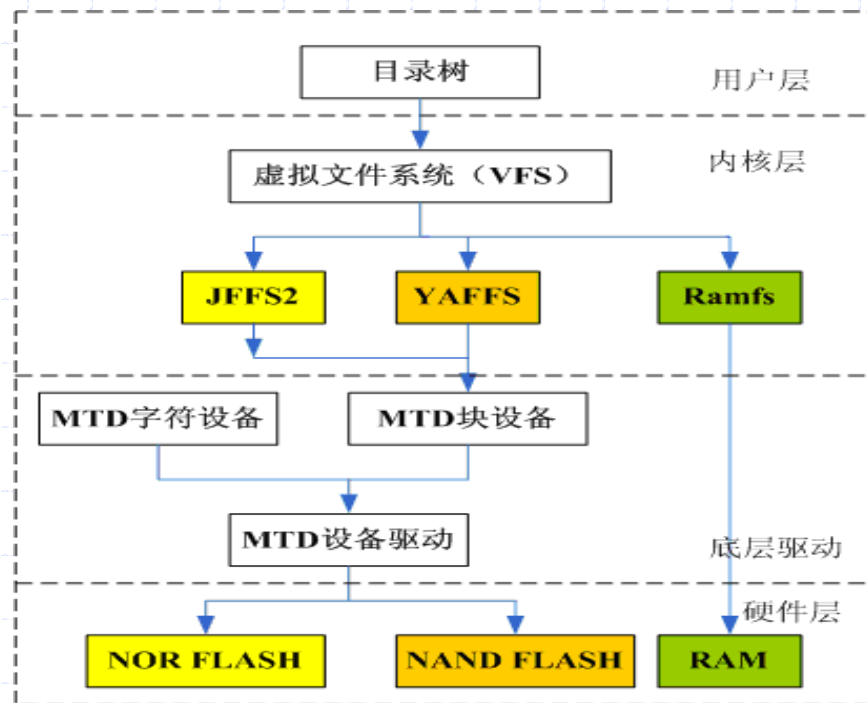
- F2F系统，于
- 2
- F2F写区域、
- 1

课程大纲

-  嵌入式文件系统概述
-  闪存存储器硬件特性简介
-  闪存存储器文件系统简介
-  嵌入式Linux根文件系统简介

Linux文件系统简介

- Linux支持多种文件系统，包括ext2、ext3、vfat、ntfs、iso9660、jffs2、YAFFS、ramfs、romfs和nfs等，为了对各类文件系统进行统一管理，Linux引入了虚拟文件系统VFS(Virtual File System)，为各类文件系统提供一个统一的操作界面和API接口。



根文件系统简介

- ❑ 根文件系统(Root File System)是一种文件系统，对应根目录“/”所对应的文件系统。
- ❑ 相对于普通的文件系统，它的特殊之处在于：它是内核启动时所mount的第一个文件系统。
- ❑ 根文件系统要包括Linux启动时所必须的目录和关键性文件，例如/sbin/init、/etc/fstab等。
- ❑ 任何包括这些Linux系统启动所必须的文件，都可以成为根文件系统。

根文件系统基本结构

- ❑ `/bin`、`/dev`、`/etc`、`/lib`、`/proc`、`/sbin`和`/usr`，都是不可或缺的目录结构。
- ❑ 为多用户提供可扩展环境的所有目录(例如`/home`、`/mnt`、`/opt`和`/root`等)可以省略。
- ❑ 调整根文件系统的时候，甚至可以进一步移除`/tmp`和`/var`，不过这么做可能会危害到某些软件的运行，因此不建议采用这种过于简化的做法。

根文件系统目录解析

目录名称	含义
/bin	存放二进制程序，如： ls ， cp
/boot	存放系统启动的一些程序
/dev	存放设备文件
/etc	存放系统配置文件，如： group ， profile
/home	用户根目录
/lib	存放库文件
/mnt	该目录用来为其他文件系统提供安装点
/opt	不随发行版本一起交付的程序
/proc	proc 文件系统
/root	超级用户根目录
/sbin	存放超级用户运行的二进制文件
/sys	sys 文件系统的目录

嵌入式根文件系统创建

□ 创建根文件系统的主要步骤

- 构建根文件系统的框架：在根文件目录下建立bin、dev、etc、lib、proc、sbin、root、tmp等一系列必备的目录
- 为根文件系统安装必要的动态库
- 为文件系统复制内核模块
- 存入设备文件，可能需要移植驱动
- 移植主要的系统应用程序
- 制作、固化根文件系统映像
 - jffs2制作工具：mkfs.jffs2
 - yaffs制作工具：mkyaffsimage
 - Cramfs制作工具：mkcramfs

BusyBox简介

- ❑ BusyBox——“繁忙的盒子”，是构建嵌入式Linux文件系统的常用工具。（“The Swiss Army Knife of Embedded Linux”）
- ❑ BusyBox集成了100+个最常用linux命令和工具，通过优化整合到一个单一的可执行文件，特别适用于存储紧凑的嵌入式系统。
- ❑ BusyBox最初是由Bruce Perens在1996年为Debian Linux安装盘编写的。其目标是在一张软盘上创建一个可引导的Linux 系统。一张软盘能保存1.4MB的内容，留给Linux内核及相关应用程序的空间受限。

BusyBox构造原理（1/3）

□ 现有Linux系统存在的一些空间浪费问题

- Linux系统中的很多独立的命令程序，如grep、find、locate等命令，均具有相似的功能模块；
- 很多命令程序，均需要用到glibc库的相关调用。

□ BusyBox构造思路

- 改造Linux系统命令程序，将所有命令合并成由一个程序（BusyBox）完成，通过符号链接的方式提供兼容，例如：`ln -s busybox ls`；`ln -s busybox cp`；
- 静态或动态链接一份glibc库，或者uclibc库。

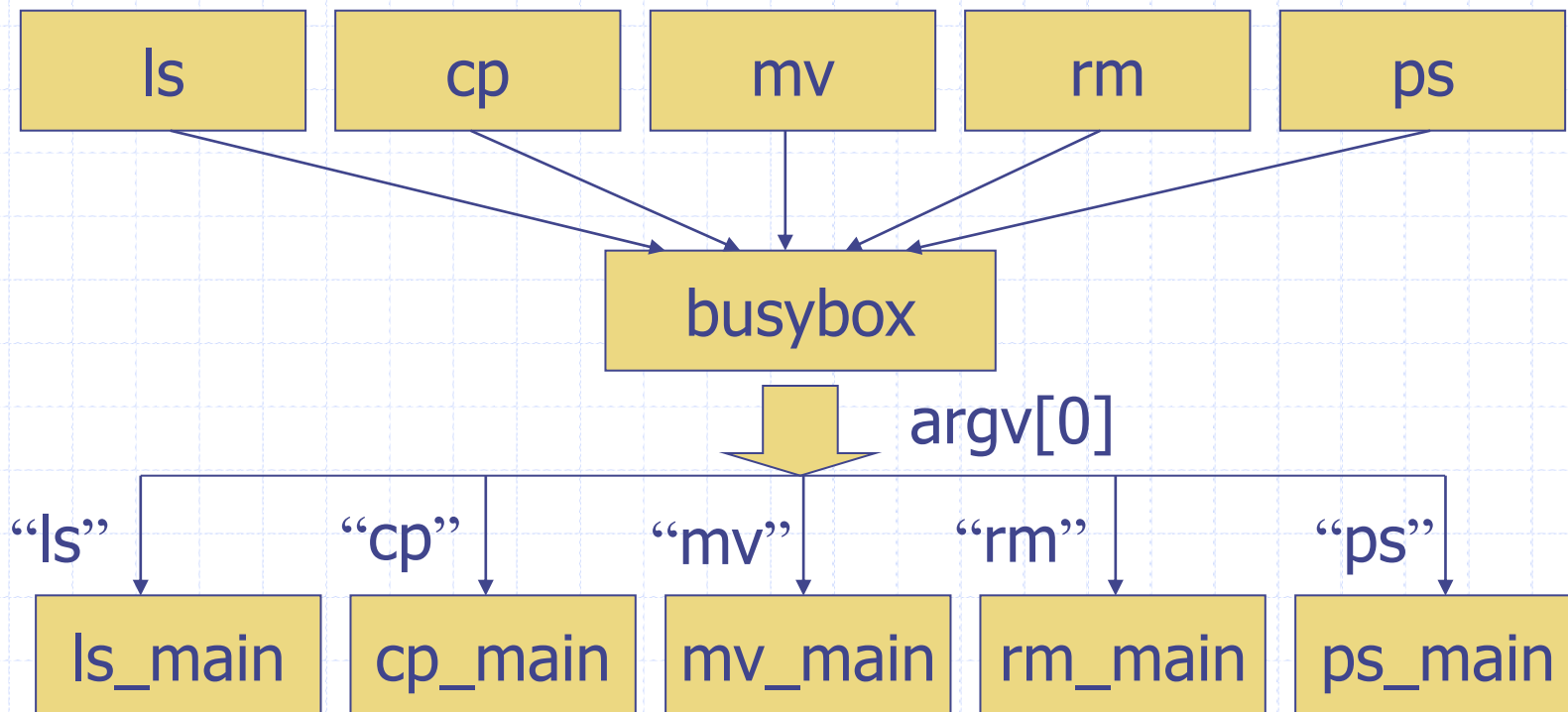
BusyBox构造原理 (2/3)

```
lrwxrwxrwx 1 root root
-rwxr-xr-x 1 root root
lrwxrwxrwx 1 root root
lrwxrwxrwx 1 root root
lrwxrwxrwx 1 root root
lrwxrwxrwx 1 root root
lrwxrwxrwx 1 root root
lrwxrwxrwx 1 root root
lrwxrwxrwx 1 root root
lrwxrwxrwx 1 root root
lrwxrwxrwx 1 root root
lrwxrwxrwx 1 root root
lrwxrwxrwx 1 root root
lrwxrwxrwx 1 root root
lrwxrwxrwx 1 root root
lrwxrwxrwx 1 root root
lrwxrwxrwx 1 root root
lrwxrwxrwx 1 root root
lrwxrwxrwx 1 root root
lrwxrwxrwx 1 root root
lrwxrwxrwx 1 root root
lrwxrwxrwx 1 root root
```

```
7 May 22 23:12 ash -> busybox
784056 May 22 23:12 busybox
7 May 22 23:12 cat -> busybox
7 May 22 23:12 chgrp -> busybox
7 May 22 23:12 chmod -> busybox
7 May 22 23:12 ciown -> busybox
7 May 22 23:12 cp -> busybox
7 May 22 23:12 date -> busybox
7 May 22 23:12 dd -> busybox
7 May 22 23:12 df -> busybox
7 May 22 23:12 dmesg -> busybox
7 May 22 23:12 echo -> busybox
7 May 22 23:12 egrep -> busybox
7 May 22 23:12 false -> busybox
7 May 22 23:12 fgrep -> busybox
7 May 22 23:12 grep -> busybox
7 May 22 23:12 gunzip -> busybox
7 May 22 23:12 gzip -> busybox
7 May 22 23:12 hostname -> busybox
7 May 22 23:12 kill -> busybox
7 May 22 23:12 ln -> busybox
7 May 22 23:12 ls -> busybox
7 May 22 23:12 mkdir -> busybox
```

所有命令，均符号链
接到一个可执行程序

BusyBox构造原理 (3/3)



BusyBox引导过程（1/2）

- 系统先启动/sbin/init，/sbin/init是/bin/busybox的符号链接，所以BusyBox是目标板系统上执行的第一应用程序。当BusyBox知道调用它的目的是要执行init，将会立即跳转到init进程。
- 如同主流的init，BusyBox可以完成系统的启动工作，并根据嵌入式系统特点进行精简处理，因此，BusyBox的init尤其适合在嵌入式系统中使用。

BusyBox引导过程（2/2）

□ BusyBox的init进程会依次进行以下工作

- 为init设置信号处理进程
- 初始化控制台
- 剖析inittab文件、/etc/inittab文件
- 执行系统初始化的命令行
- 缺省情况下使用/etc/init.d/rcS命令行
- 执行所有会导致init暂停的inittab命令
- 执行所有仅执行一次的inittab命令
- 一旦完成以上工作，init进程便会循环执行以下工作：
 - 执行所有终止时必须重新启动的inittab命令
 - 执行所有终止时必须重新启动但启动前必须先询问过用户的inittab命令



谢谢!

