嵌入式系统设计与实例开发

——ARM与μC/OS-Ⅱ

考试方法

笔试60%，作业实验40%

其中：平时作业+2次实验报告　　20分
　　大实验（2人一组）　　20分
　　笔试卷面成绩　　60分

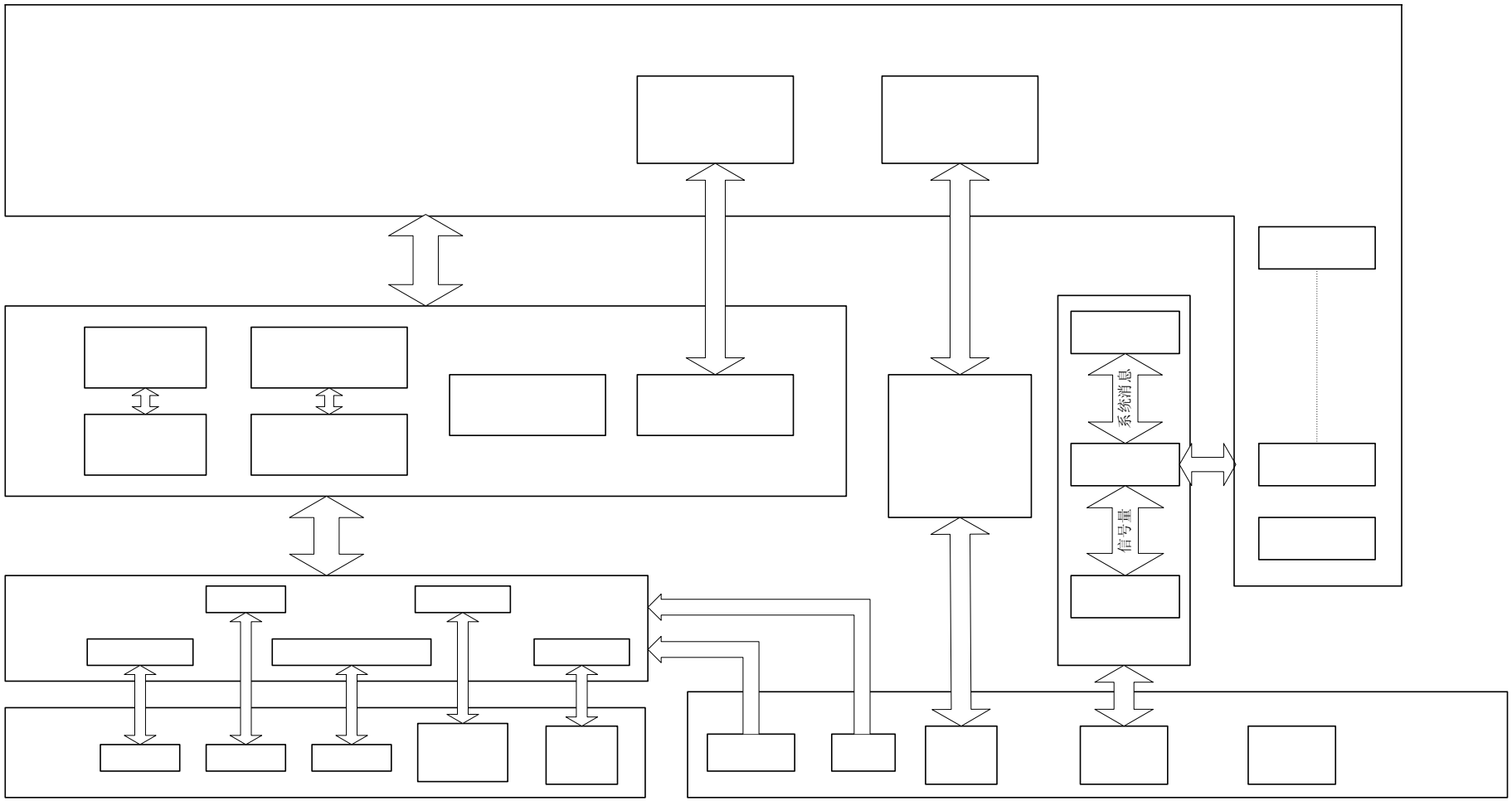
大实验题目

一、设计一个电子点菜PDA，可以直接查看菜谱，进行实时点菜，所涉及技术点包括网络、触摸屏、LCD显示等。
二、设计一个嵌入式游戏，如俄罗斯方块、贪吃蛇等
三、自拟题目

五、基于μC/OS-II的软件结构设计

- 基于μC/OS-II的软件设计
  - 文件系统
  - 图形用户接口（GUI）
- μC/OS-II for ARM BSP的实现

基于μCOS-II扩展RTOS的体系结构



1. 系统外围设备的硬件部分

- 系统外围设备的硬件部分包括：液晶显示屏（LCD）、USB通信模块、键盘、海量Flash存储器、系统的时钟和日历。外围设备的硬件部分是保证系统实现指定任务的最底层的部件。

2、驱动程序模块

- 驱动程序是连接底层的硬件和上层的API函数的纽带，有了驱动程序模块，就可以把操作系统的API函数和底层的硬件分离开来。硬件的改变、删除或者添加，只需要随之改变、删除或者添加提供给操作系统的相应的驱动程序就可以了。而不会影响到API函数的功能，更不会影响到用户的应用程序。

3. 操作系统的API函数

- 在操作系统中提供标准的应用程序接口（API）函数，可以加速用户应用程序的开发，统一应用程序的标准，同时也给操作系统版本的升级带来了方便。在API函数中，提供了大量的常用模块，可以大大简化用户应用程序的编写。

4. 实时操作系统的多任务管理

- μC/OS-II作为操作系统的内核，主要的任务就是完成多任务之间的调度和同步。

5. 系统的消息队列

- 这里所说的系统的消息队列是以μC/OS-II的消息队列派生出来的系统消息传递机制，用来实现系统的各个任务之间、用户应用程序的各个任务之间以及用户应用程序和系统的各个任务之间的通信。

6. 系统任务

- 系统任务主要包括液晶显示屏（LCD）的刷新任务、系统键盘扫描任务。这两个任务是操作系统的基本任务，随着操作系统的启动而运行。

7. 用户应用程序

- 用户的应用程序建立在系统的主任务（Main_Task）基础之上。用户应用程序主要通过调用系统的API函数对系统进行操作，完成用户的要求。在用户的应用程序中也可以创建用户自己的任务。任务之间的协调主要依赖于系统的消息队列。

嵌入式文件系统

什么是文件系统（File System）

文件系统的定义

——处理文件的操作系统的部分称为文件系统.是操作系统中统一管理信息资源的一种软件，管理文件的存储、检索、更新，提供安全可靠的共享和保护手段，并且方便用户使用

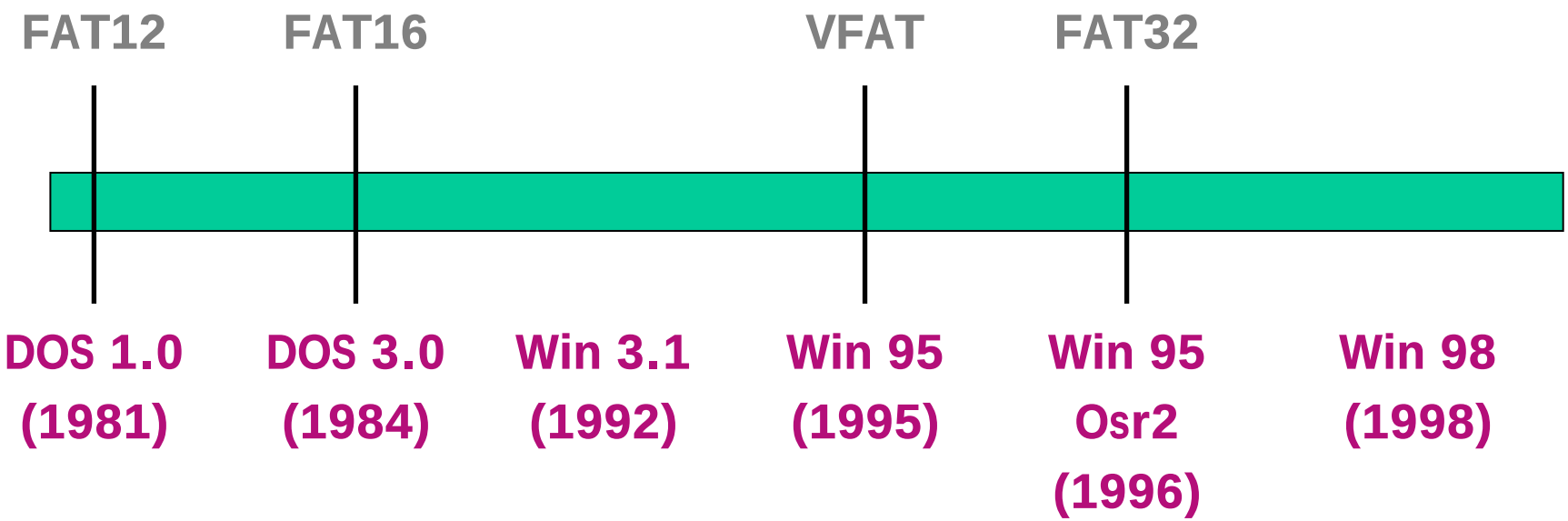
文件系统的功能

——文件的构造、命名、存取、采用、保护和实现等。

文件系统的存储媒质

——磁盘、软盘、光盘、FLASH盘等等

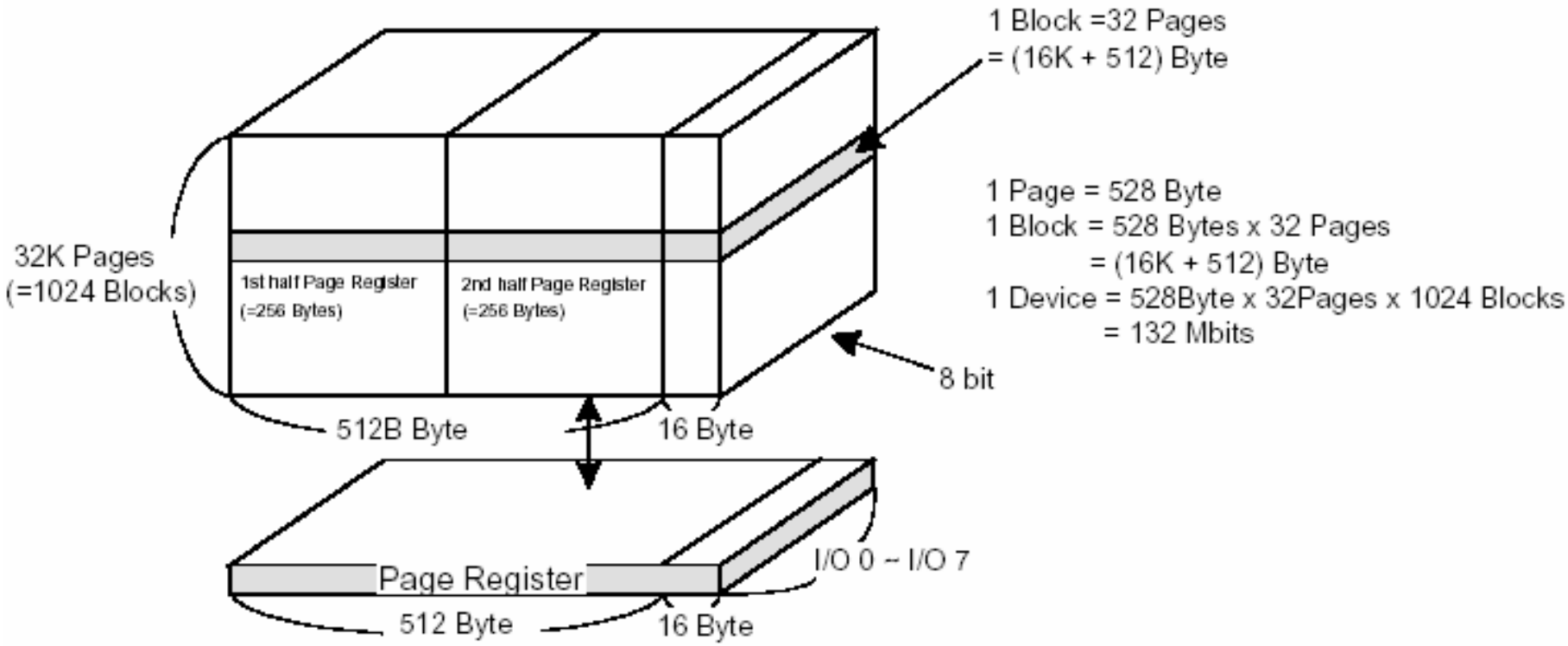
文件系统的发展



FAT12/FAT16/FAT32的比较

	FAT12	FAT16	FAT32
Size of FAT entry	12 bits	16 bits	32 bits
Max num of clusters	4,086	65,526	268,435,456
Cluster size used	0.5 KB - 4 KB	2 KB - 32 KB	4 KB - 32 KB
最大磁盘容量	16,736,256 (16M)	2,147,123,200 (2G)	about 2^41 (2T)

基于FLASH的嵌入式文件系统



FLASH读写的特点

（1）必须以Page为单位进行读写；

（2）写之前必须先擦除原有内容；

（3）擦除操作必须对Block进行，即一次至少擦除一个Block的内容

针对这种情况，将Flash的一个Page定为1个扇区，将其1个Block，32个扇区定为一个簇，这样，簇的容量刚好为512 * 32 = 16K，满足FAT16对簇大小的要求

FLASH文件系统的要求

（1）**掉电安全：**嵌入式系统的运行环境一般比较恶劣，但同时又要求有较高的可靠性。这就对FLASH文件系统进出了较高的要求，无论程序崩溃或系统掉电，都不能影响文件系统的一致性和完整性，文件系统的写入、垃圾回收等操作对系统异常中止都非常敏感，极易造成数据丢失和数据垃圾，在文件系统设计和选用时应考虑;

（2）**平均使用（wear-leveling）：**由于FLASH扇区的擦除次数有限制，要求能够均匀使用各扇区，以延长FLASH的使用寿命;

（3）**高效垃圾回收（garbage collection）：**任何存储器在分配使用一段时间后，都会出现空区和碎片数据，为保证存储空间的使用率。方法是先移动扇区数据，再擦除整个扇区;

（4）**低空间消耗（lov overhead）：**指文件系统管理结构在FLASH存储器上的空间消耗，该空间用于FS建立，而不能用于实际数据的存储，可以提高有用数据的存储空间

FLASH文件系统的分类

（1）集中管理文件系统

特点：存储器空间的使用信息集中存放在存储器的某个地方，存储器的其它区域用于存放数据，数据必须依赖关键信息区才能被索引和使用

缺点：需要大量缓存空间，当某扇区需要更新时，先将扇区数据备份到RAM中，再进行擦除操作，最后将修改后的备份写入FLASH，缺点是难以保证掉电安全，不能均匀使用存储器空间，特别是关键信息区，对文件系统的使用个性都会改写该区，导致FLASH快速损坏;

FLASH文件系统的分类

（2）线性文件系统

概述：每个文件相关的信息都连续存放在存储器中，实现简单，读写快速，文件系统的关键信息分布存放；

优点：安全性好；能保证存储器的平均使用，延长了FLASH的使用寿命；

缺点：对文件操作效率低，不易实现添加、插入、剪切等操作；

FLASH文件系统的分类

（3）日志文件系统

概述：日志结构的文件系统使用顺序的、只增的日志作为磁盘上唯一的组织文件系统数据的结构，文件的描述可以仍采用传统的索引组织方式。方法是在内存中将几次 FS的修改汇集成一个大的日志条目(被称为段映像），然后动态分配磁盘空间并通过一次写操作写到磁盘上的一个连续的、固定大小的日志段（Segment)中。并定时或当系统发出同步写请求时保证日志同步写到磁盘上;

优点：恢复快速。

几种开源的FLASH文件系统

TFS（Tiny File System）

概述：TFS是由原Lucent公司的Ed Sutter开发的嵌入式系统引导平台Umon的一部分。TFS是一种线性结构的文件系统，由多个存放的文件块组成。一个文件块包含一个文件的所有信息。

优点：TFS提供了掉电安全机制和垃圾回收机制，需要额外的辅助空间，用于垃圾回收时的文件缓存和过程状态，如果出现终止，系统根据辅助空间的信息进行文件系统的恢复；

缺点：文件的插入、剪切和个性需要较大的运行开销，即使是很小的修改，也要求将整修文件重写

几种开源的FLASH文件系统

- JFFS(Journaling Flash File System)

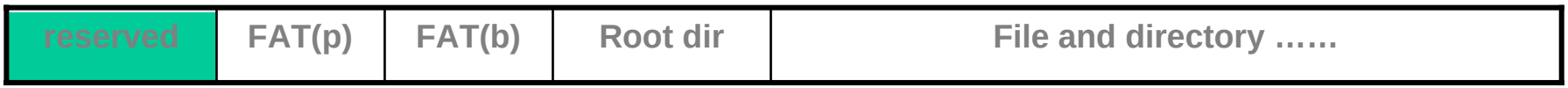
概述：由瑞典Axis通信公司开发的文件系统，主要针对NOR型Flash存储器设计，提供了掉电安全，平均使用等特性，是基于Linux，由于遵循GPL开放源代码，易实现移植;

FAT16文件系统基本结构

- FAT 文件系统由下面四部分组成
  - 保留区 **Reserved region**
    - 存放引导记录，**BIOS**信息等
  - **FAT** 区
    - **FAT**信息列表 **(12/16/32 bits)**
  - 根目录区
    - 目录信息列表 **(32 bytes)**
  - 文件和目录区
    - 存放簇信息

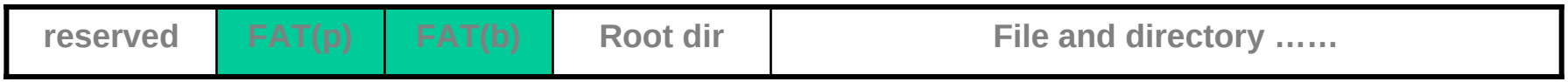
reserved	FAT(p)	FAT(b)	Root dir	File and directory ……

保留区 Reserved Region

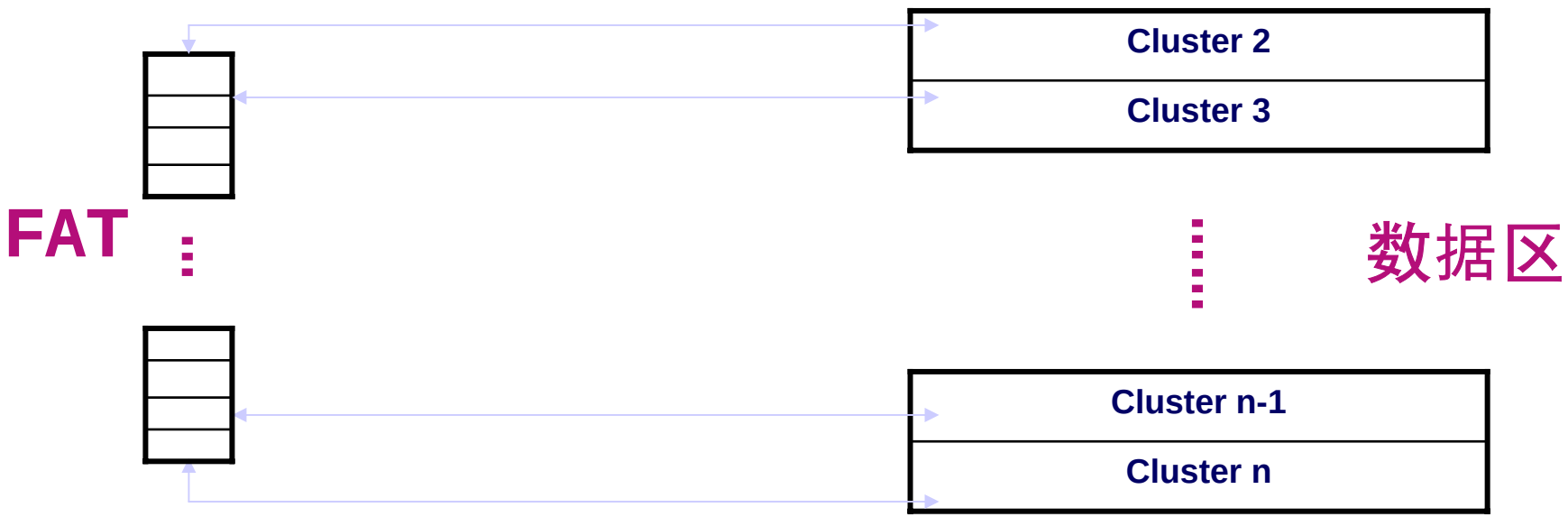


- 引导记录
  - 基本信息 ……
  - 引导代码
- BIOS参数块 (BPB)
  - 字节/扇区 (512,1024,2048,4096)
  - 扇区/簇 (1,2,4,8,16,32,64,128)
  - 根目录数
  - 总扇区值
  - 介质类型 (硬盘,软盘，FLASH）

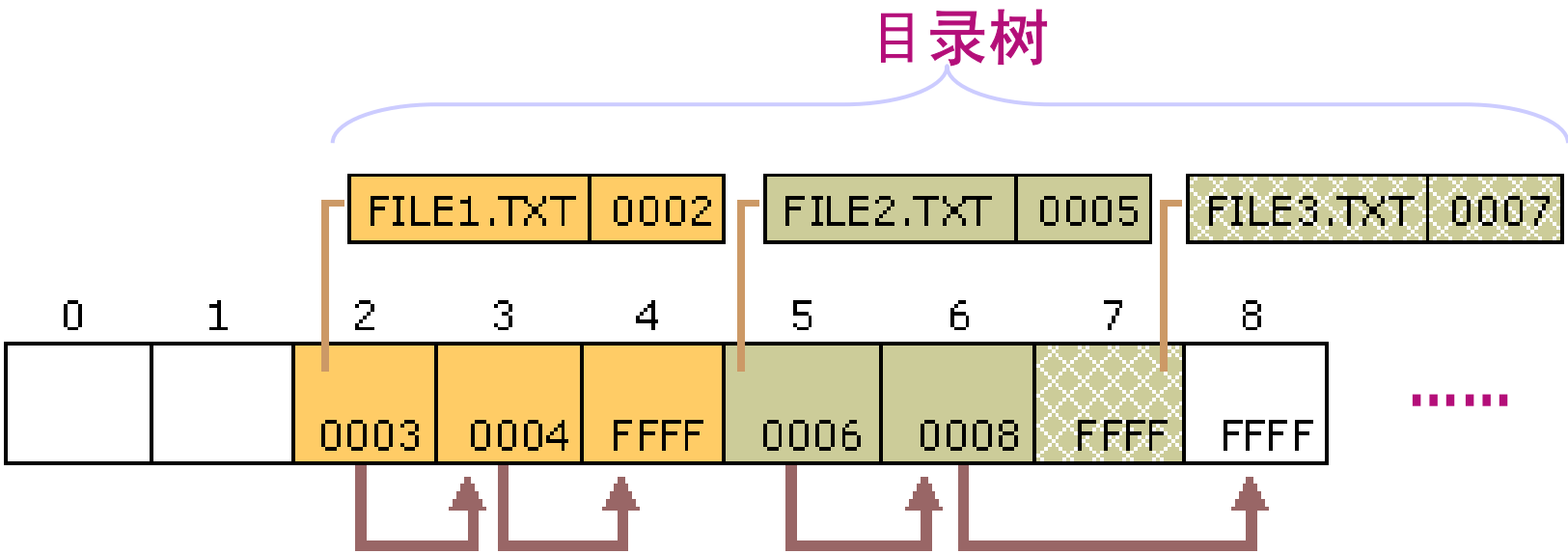
FAT 区



- 通常有 FAT(P)(primary)和FAT(B)(backup)
- 每一个簇都有一个对应的FAT目录



- FAT 链表 (FAT16)


目录树
FILE1.TXT
0002
FILE2.TXT
0005
FILE3.TXT
0007
0
1
2
3
4
5
6
7
8
0003
0004
FFFF
0006
0008
FFFF
FFFF
.....

FLASH的前两个BLOCK

LBA	Block / Page	长度	内容说明
0	0/0	512字节	MBR＝BPB＋Excutable Code+55AA（查看内容）
1~2	0/1~0/2	1024字节	FAT区（第一份FAT）
3~4	0/3~0/4	1024字节	FAT区备份（第二份FAT）
5~39H	0/5~1/31	30K字节	目录区(在BPB中调整目录项数，使其刚好占尽本簇)
40H	1/32	512字节	数据区（因目录区占尽一个簇，故数据区始于新簇首扇）

典型FLASH文件系统的结构

系统纪录 （SR，System Record)	文件分配表 (FAT,File Allocation Table)	文件登记表 (FRT,File Register Table)	数据区域 (Data Area)

（1）系统记录（SR，System Record）

存放媒质信息和最重要的文件系统信息。媒质信息诸如Flash存储器的类型、容量，划分成多少个区块，每区块包含多少个页面等。文件系统信息包括版本信息、保留区块的数目和位置、文件分配表和文件登记表所在的位置和大小、数据区域的位置和大小等。

（2）文件分配表（FAT，File Allocation Table）

存放着Flash存储器上所有区块的占用与空闲情况以及每个文件的存储连接结构。采用FAT16文件格式

FLASH文件系统的结构

（3）文件登记表（FRT，File Register Table）

存放着Flash文件系统中每一个文件的文件代号、文件长度、文件属性以及该文件的存储链在文件分配表中的入口。

（4）数据区域（Data Area）

用于存放文件的数据内容。本Flash文件系统中，数据分配的最小单位是Flash存储器的一个基本擦除单元，即一个物理区块（Block）。

FILE结构体

```
typedef struct{
    U8 Buffer[BLOCK_SIZE];          //文件缓冲区
    U32 fileblock;                  //文件当前的簇的位置
    U32 filemode;                   //打开文件的模式
    U32 filebufnum;                 //文件缓冲区中已经读取/写入的
    字节数
    U32 fileCurpos;                 //读写的当前位置
    U32 filesize;                   //文件的大小
}FILE;
```

与FLASH存储器的接口函数

块擦除unsigned char Erase_Cluster(unsigned int cluster)

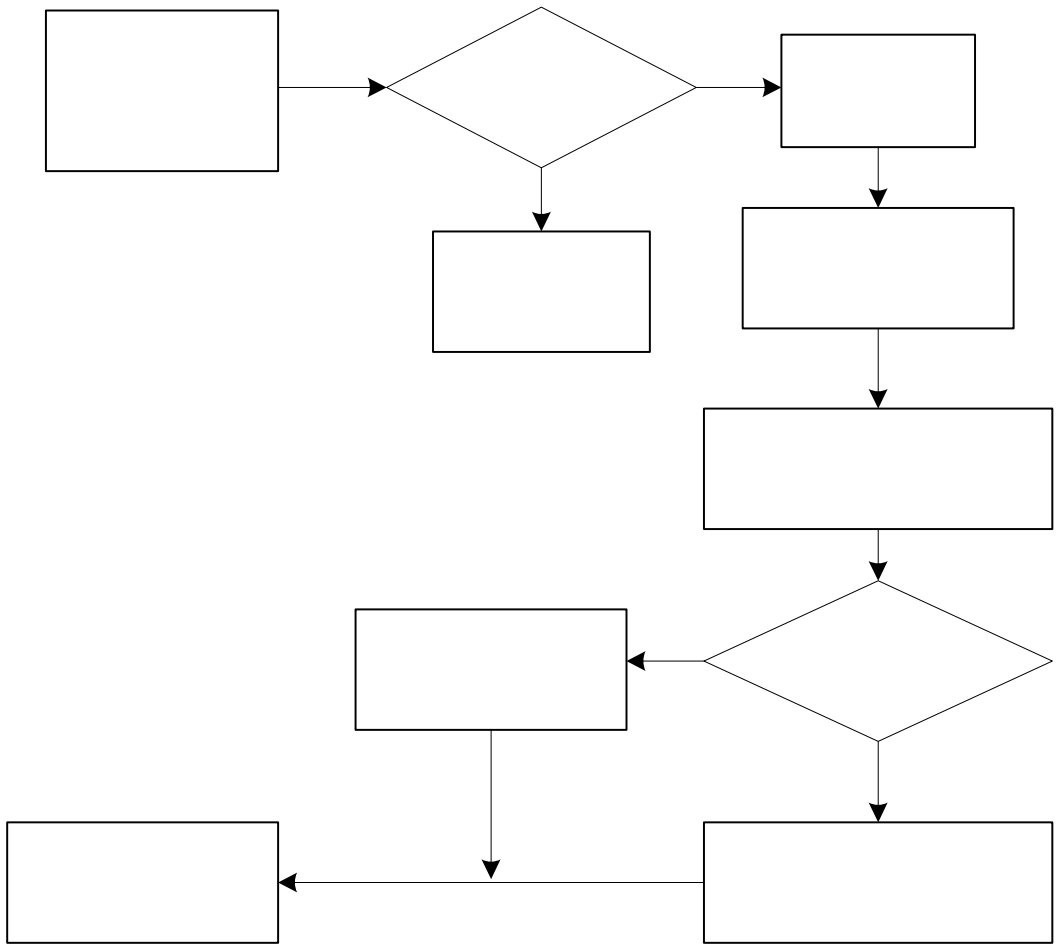
页写入int WritePage(unsigned int block,unsigned int page,unsigned char *pPage)

页读出void ReadPage(unsigned int block,unsigned int page,unsigned char *pPage)

文件系统的相关函数

- 始始化文件系统 **void initOSFile(char filename[],U32 OpenMode)**
- 读文件到缓冲区 **U32 ReadOSFile（FILE* pfile,U8* ReadBuffer,U32 nReadbyte)**
- 把缓冲区内容写入文件 **U32 WriteOSFile（FILE* pfile,U8* WriteBuffer,U32 nWritebyte)**
- 关闭文件，释放缓冲区 **void CloseOSFile(FILE* Pfile)**

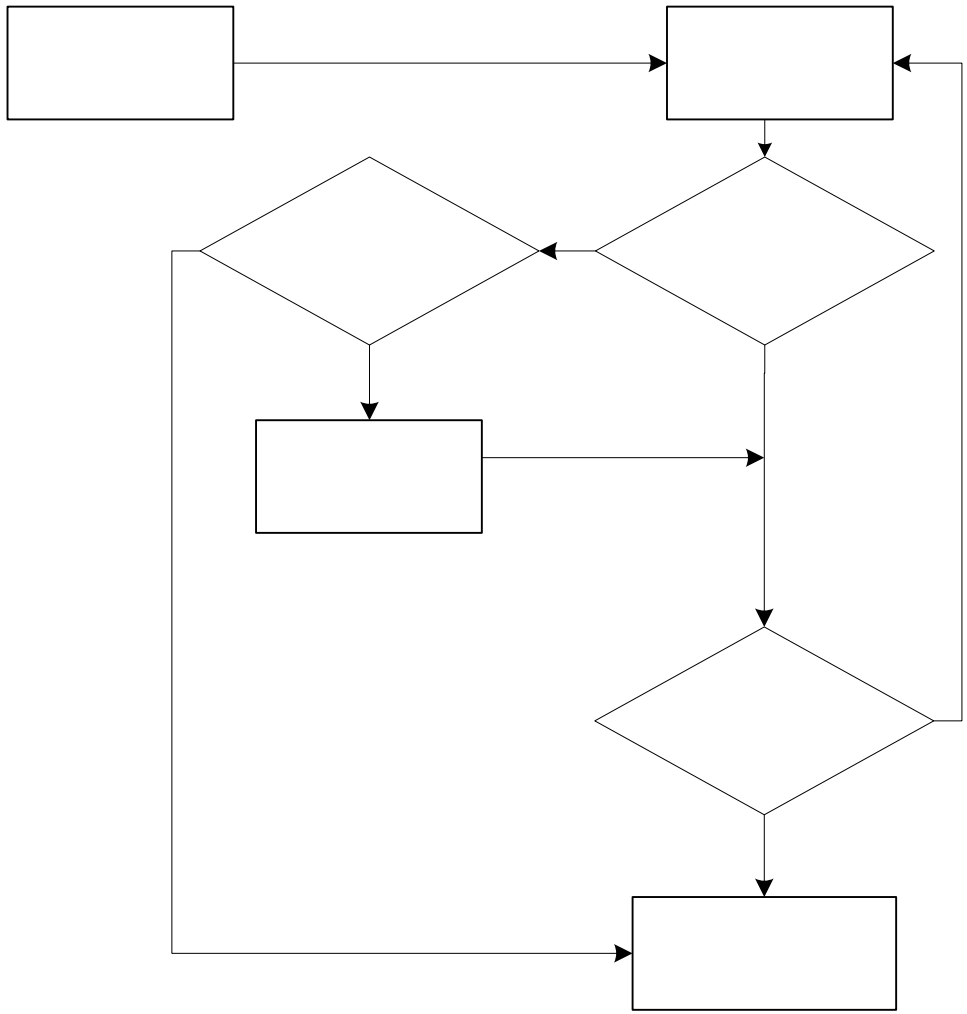
OpenOSFile的工作的流程图



●OpenOSFile的工作的流程图

Open
函数

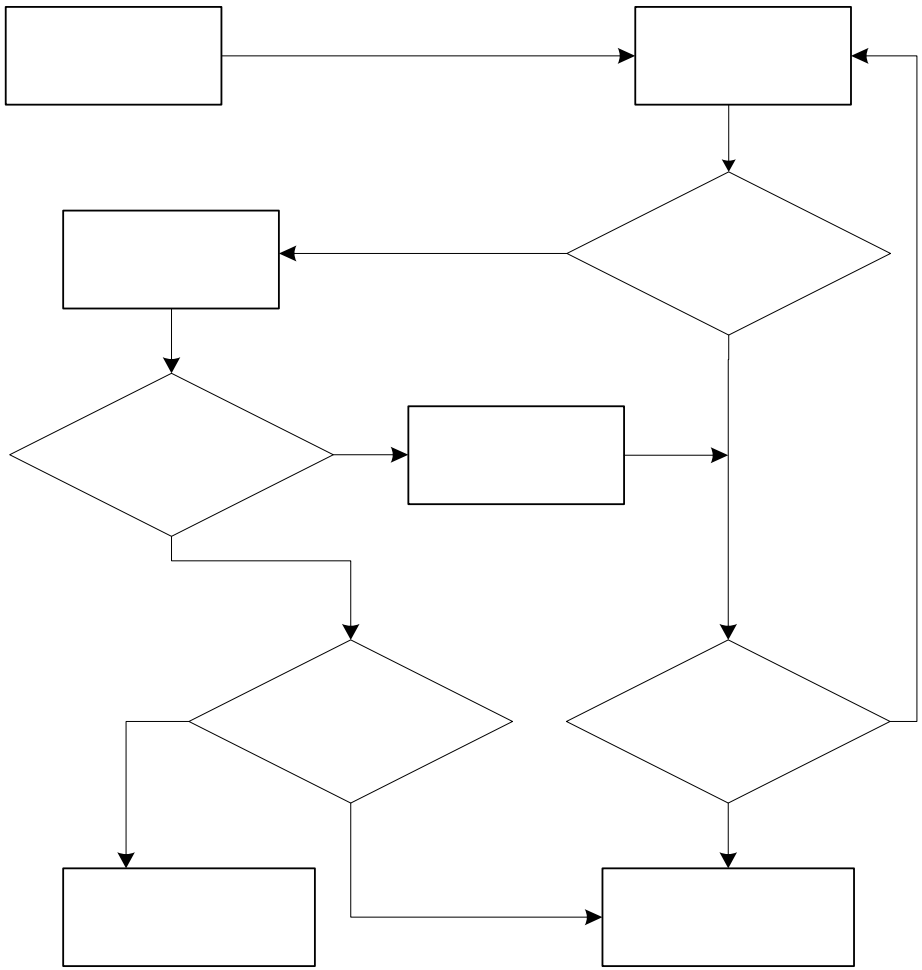
ReadOSFile函数的程序流程图



- ReadOSFile函数的程序流程图

Rea

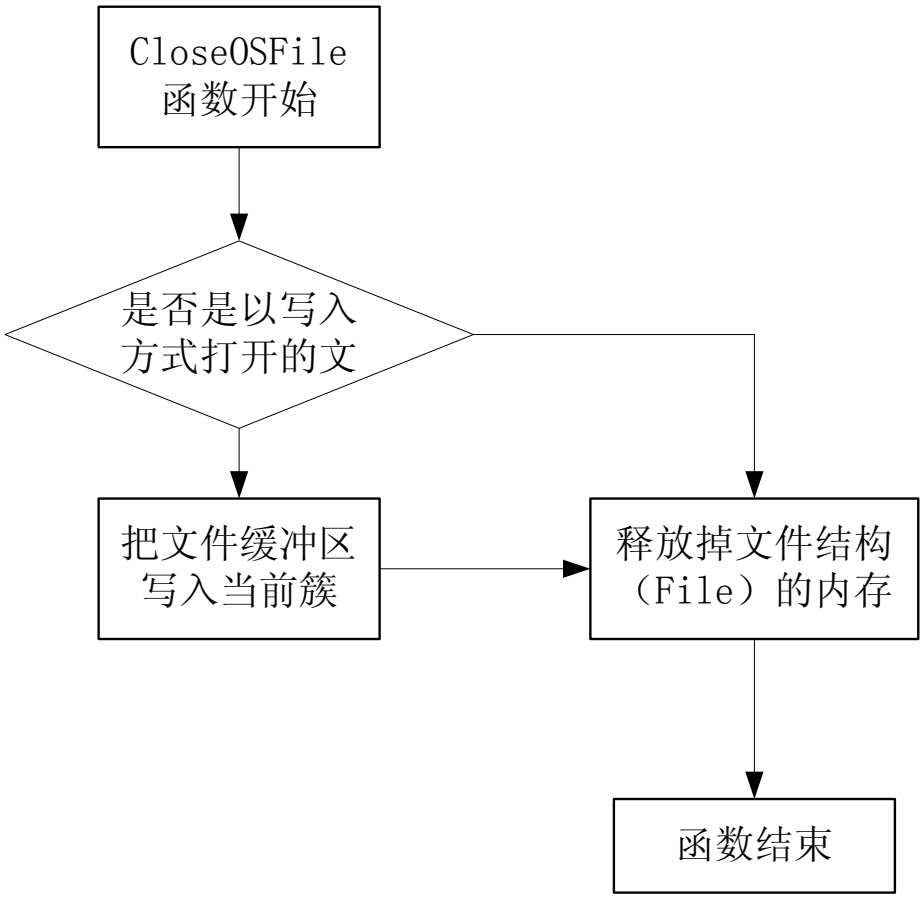
WriteOSFile函数的程序流程图



●WriteOSFile函数的程序流程图

Wri

CloseOSFile函数的程序流程图



●CloseOSFile函数的程序流程图

嵌入式GUI技术

GUI的概念

GUI的定义

Graphics User Interface，是指计算机与其使用者之间的图形化对话接口。

GUI的主要特征：

——Windows，采用窗口界面，每个窗口是用户或系统的一个工作区域。一个屏幕上可以有多个窗口。

——Icons，采用形象化的图标或图符，易于操作者理解与操作。

——Menu，采用菜单，可供用户选择的功能提示

——Pointing Devices ，指鼠标器、触摸屏等，便于用户直接对屏幕对象进行操作。

嵌入式GUI的特点

- 体积小
- 功能强;
- 图形算法简洁、快速,占用系统资源少
- 可靠性高;
- 模块结构，便于移植和定制

嵌入式GUI的实现方法

1）照需求开发满足自身特定需要的GUI系统;

2）将GUI作为一个软件层从应用程序中剥离，GUI的支持逻辑由应用程序自己负责;

3）设计一个支持大多数常见的GUI对象的应用编程接口库，使其具有与其他通用开发工具相类似的调用方法(如Win32)的GUI系统

几种典型的嵌入式GUI

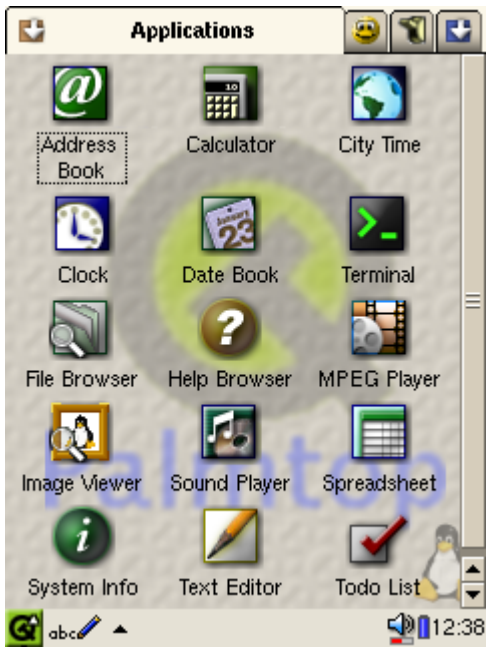
- Compact X-Window System:可扩展性好、可移植性好，代码尺寸大
- Microwindows ： 开放源码的嵌入式GUI软件，可移植性好，图形功能出色。
- OpenGUI ： 自由软件。汇编实现的内核，并利用MMX指令进行了优化，OpenGUI运行速度非常快。支持 32 位处理器，可以在MS-DOS，QNX和Linux下运行。主要用来在这些系统中开发图形应用程序和游戏
- Qt/Embedded: QT 库开发商Trolltech 推出的面向嵌入式系统的 QT 版本。可移植性好，价格较贵。
- MiniGUI: 自由软件，面向嵌入式系统或者实时系统的图形用户界面支持系统。它主要运行于 Linux ，还可以运行在任何一种具有 POSIX 线程支持的 POSIX 兼容系统上

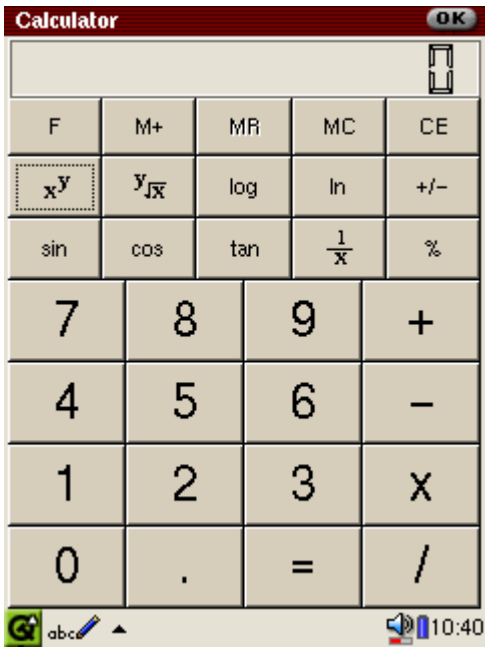
嵌入式GUI

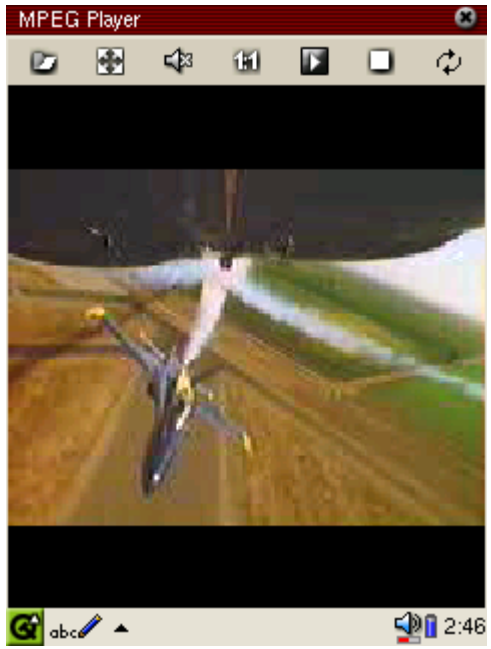
- Compact X-Window System:可扩展性好、可移植性好，代码尺寸大
- Microwindows ： 开放源码的嵌入式GUI软件，可移植性好，图形功能出色。
- OpenGUI ： 自由软件。汇编实现的内核，并利用MMX指令进行了优化，OpenGUI运行速度非常快。支持 32 位处理器，可以在MS-DOS，QNX和Linux下运行。主要用来在这些系统中开发图形应用程序和游戏
- Qt/Embedded: QT 库开发商Trolltech 推出的面向嵌入式系统的 QT 版本。可移植性好，价格较贵。
- MiniGUI: 自由软件，面向嵌入式系统或者实时系统的图形用户界面支持系统。它主要运行于 Linux ，还可以运行在任何一种具有 POSIX 线程支持的 POSIX 兼容系统上

Qt/Embedded

- 可移植性好
- 模块化设计
- 开放源代码
- 图形界面漂亮

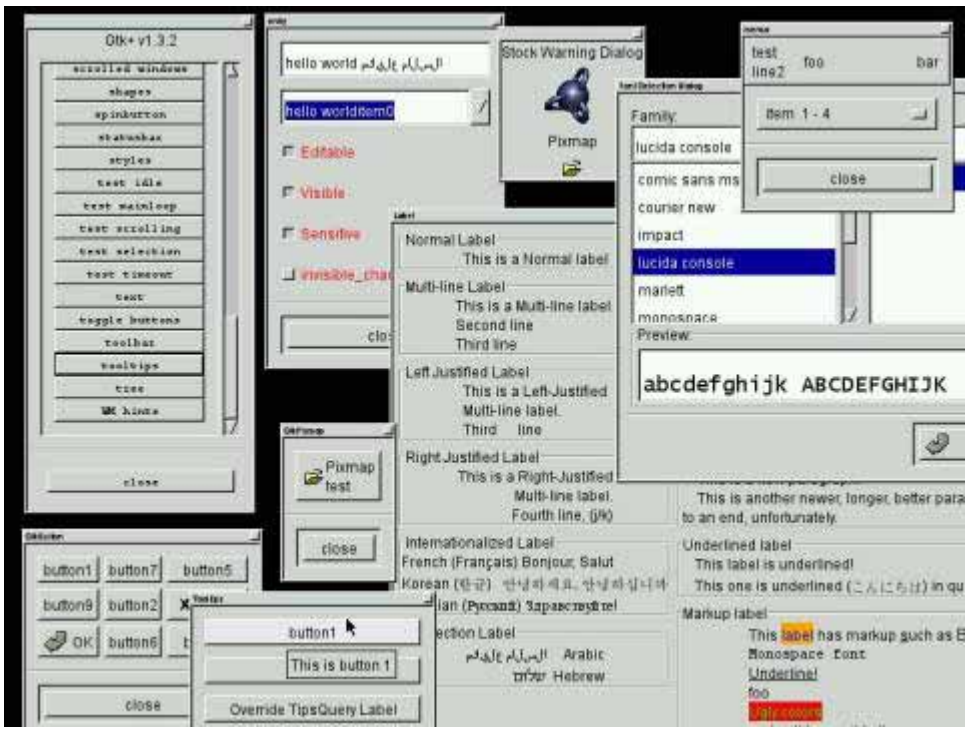






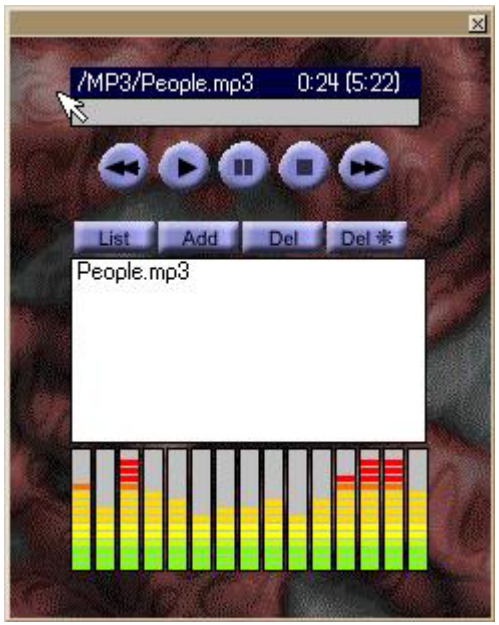
GTKFB

- 不需 X Server 直接与 FrameBuffer 连接
- API 与 Desktop 版相容。
- 采用 LGPL 授权方式
- 体积小



Microwindows

- 跨平台
- 不需 X Server
- 采用 FLTK Toolkits
- 支持 TrueType 字型

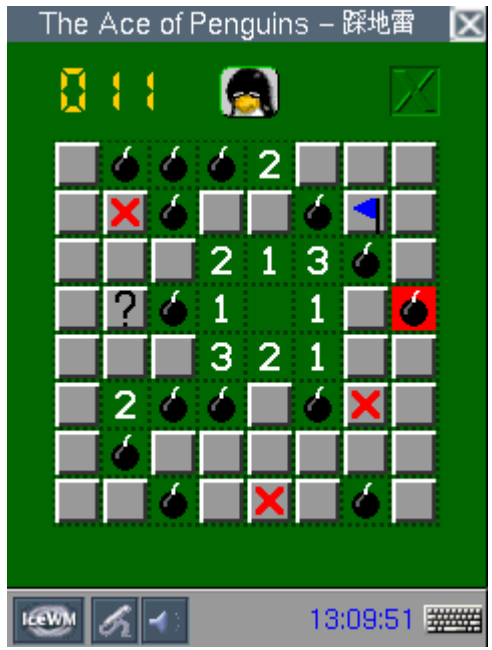


MiniGUI

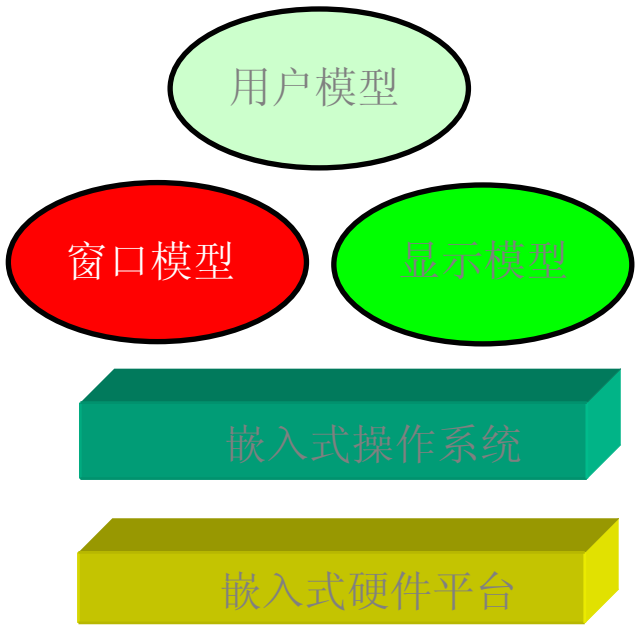
- 支持 GB2312 与 BIG5 字集
- 支持多种格式字体，例如：TrueType、Adobe Type1等。
- GU 函数（MoveTo、LineTo、FillBox、Rectangle、Circle、TextOut、DrawText ..等）

Tiny X Server

- Tiny X Server 为 XFree86 Project 的一部分，体积比较大



嵌入式GUI的结构模型



显示模型：图形在窗口上的基本显示模式

窗口模型：窗口如何显示及改变

用户模型：构造用户界面的工具及如何在屏幕上组织各种图形对象，以及这些对象之间如何交互的说明。

图形用户系统是由显示模型接口程序、窗口模型接口程序和用户模型接口程序共同组成的。

嵌入式GUI的体系结构

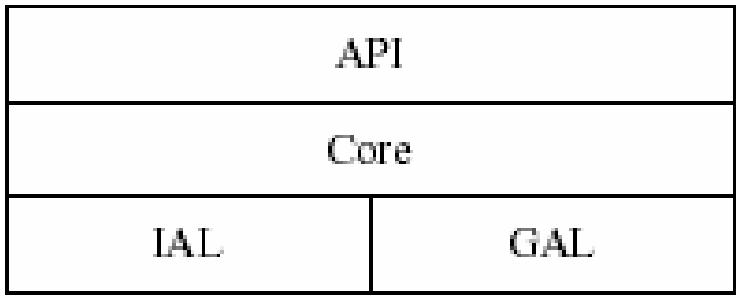


图1　嵌入式GUI分层结构图

（1）API提供操作各种GUI对象(如窗口、菜单等)的应用编程接口函数;
（2）Core提供核心的图形操作功能，如消息机制、图形设备接口、字体、窗口与桌面等的管理功能。
（3）IAL和GAL指硬件设备输入抽象层和图形输出抽象层，与底层输入输出设备接口，便于GUI挂接不同的输入输出设备，实现GUI系统良好的可移植性和通用性

核心图形操作层的结构

核心机制模块

核心机制模块是嵌入式GUI中最重要的组成部分，包括消息机制和事件驱动机制、桌面操作、初始化操作、定时器操作等几个子模块，下面介绍几个主要子模块的功能:

1) 消息机制和事件驱动机制子模块：这个子模块主要负责消息的接受、分发等操作;

2) 桌面子模块：桌面是GUI系统中非常重要的一个部分，相当于一个最底层窗口，其他主窗口都覆盖其上，维护了GUI中很多总体全局的操作;

3) 初始化子模块：它是GUI系统中最基本的一个模块，每次GUI运行之初都要通过这个模块的API函数对整个GUI环境参数进行设定和初始化操作;

4) 定时器子模块：定时器是GUI中非常重要的组成部分，当用户需要定时的处理某项操作将会使用该模块，在系统中也有使用，如编辑框控件中光标的闪烁，进度条控件进度块的前进等。

核心机制模块

核心机制模块是嵌入式GUI中最重要的组成部分，包括消息机制和事件驱动机制、桌面操作、初始化操作、定时器操作等几个子模块，下面介绍几个主要子模块的功能:

1) 消息机制和事件驱动机制子模块：这个子模块主要负责消息的接受、分发和路由等操作;

2) 桌面子模块：桌面是GUI系统中非常重要的一个部分，相当于一个最底层窗口，其他主窗口都覆盖其上，维护了GUI中很多总体全局的操作;

3) 初始化子模块：它是GUI系统中最基本的一个模块，每次GUI运行之初都要通过这个模块的API函数对整个GUI环境参数进行设定和初始化操作;

4) 定时器子模块：定时器是GUI中非常重要的组成部分，当用户需要定时的处理某项操作将会使用该模块，在系统中也有使用，如编辑框控件中光标的闪烁，进度条控件进度块的前进等。

字符集与字体模块

- 矢量字体　基于矢量的的字体，称之为可缩放的字体，轮廓字体，或矢量字体。由于这些字体在存储时亦只存储了其轮廓，因此，在不同的缩放大小下依然能保持美观而不会出现“锯齿”。这点是很适合其完整地在Web上传输。一般说来，Logo、线形艺术作品、图表、动画、抽象艺术作品等凡较容易定义颜色区域的图形，用矢量图都能产生很好的效果。
- 计算机是以处理数字为基础，如果要处理文字就需要规定一个编码系统用不同的数字来表示相应的字符。我们较为熟悉的有GB、GBK、BIG5、ASCII等等。由于编码不统一，这些编码系统之间经常相互冲突。事实上，两种编码可能使用相同的数字代表两个不同的字符；或者使用不同的数字代表相同的字符。
- 在Unicode的双字节版本中（UTF-16）使用的是16位编码方式，可提供65,000多个字符代码指针。其编码容量可涵盖世界上几乎所有的语言，不仅包括拉丁语，希腊语，斯拉夫语，希伯来语，阿拉伯语，亚美尼亚语，还包括中文，日文和韩文这样的象形文字，以及平假名，片假名，孟加拉语，泰米尔语，泰国语，老挝语等。目前还有大约8000个代码指针未用，可供扩展。

图形设备接口

- 图形设备接口(Graphics Device Interface，GDI)象GUI图形操作的中间件，主要功能是支持与设备无关的图形操作，GDI将上层应用和不同输出设备的特性隔离开来，使编制的上层应用能够毫无困难地在任意一种图形输出设备上运行。

- 它向上层应用(最终用户或者系统其他上层组件)提供了一些基本的服务：位图(bitmap)、文本(text)、一般绘图(Gen Drawing)等

控件

■一般地，GUI系统都会预先定义一些控件类，当利用某个控件类创建控件之后，所有属于这个控件类的控件均会具有相同的行为和显示。利用这些技术，可以确保一致的人机操作界面，而程序员可以像搭积木一样地组建图形用户界面。

■嵌入式GUI系统使用了控件类和控件的概念，可以方便地对已有控件进行重载，使其拥有一些特殊效果。如需要建立一个只允许输入数字的编辑框时，可以通过重载已有编辑框而实现，而不需要重新编写一个新的控件类。

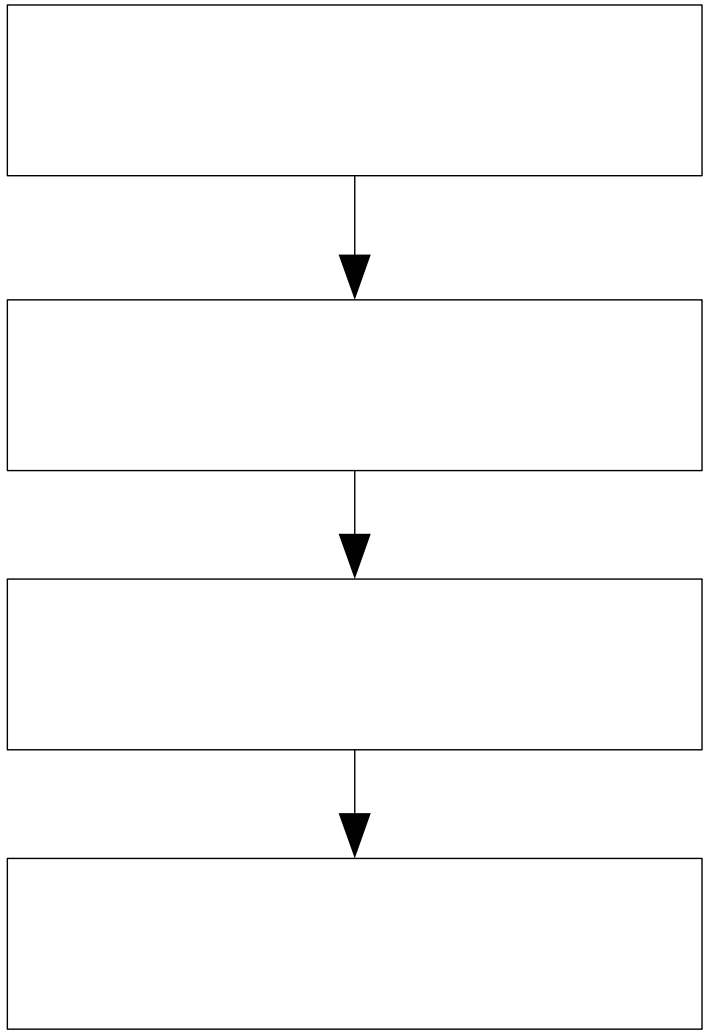
常用GUI函数简介

在多任务操作系统中，绘图设备上下文（DC）是绘图的关键。绘图设备上下文（DC）保存了每一个绘图对象的相关参数（比如：绘图画笔的宽度、绘图的原点坐标等）。在多任务操作系统中，通过绘图设备上下文（DC）来绘图，可以保证在不同的任务绘图的参数是相互独立的，不会互相影响。

DC的定义

```
typedef struct{
    int DrawPointx;
    int DrawPointy;      //绘图所使用的坐标点
    int PenWidth;        //画笔宽度
    U32 PenMode;         //画笔模式
    U32 PenColor;        //画笔的颜色
    int DrawOrgx;        //绘图的坐标原点位置
    int DrawOrgy;
    int DrawRangex;      //绘图的区域范围
    int DrawRangey;
    U8 bUpdataBuffer;    //是否更新后台缓冲区
    U32 Fontcolor;       //字符颜色
}DC,*PDC;
```

DC的使用



DC创建

```
PDC pdc;
pdc=CreateDC();
```

变量	定义	默认值
DrawPointx	绘图所使用的坐标点横坐标	0
DrawPointy	绘图所使用的坐标点纵坐标	0
PenWidth	画笔宽度	1
PenMode	画笔模式	GRAPH_MODE_NORMAL
PenColor	画笔的颜色	COLOR_BLACK
DrawOrgx	绘图的坐标原点横坐标	0
DrawOrgy	绘图的坐标原点纵坐标	0
DrawRangex	绘图的区域水平范围	LCDWIDTH
DrawRangey;	绘图的区域垂直范围	LCDHEIGHT
bUpdataBuffer	是否更新后台缓冲区及显示	TRUE
Fontcolor	字符颜色	COLOR_BLACK

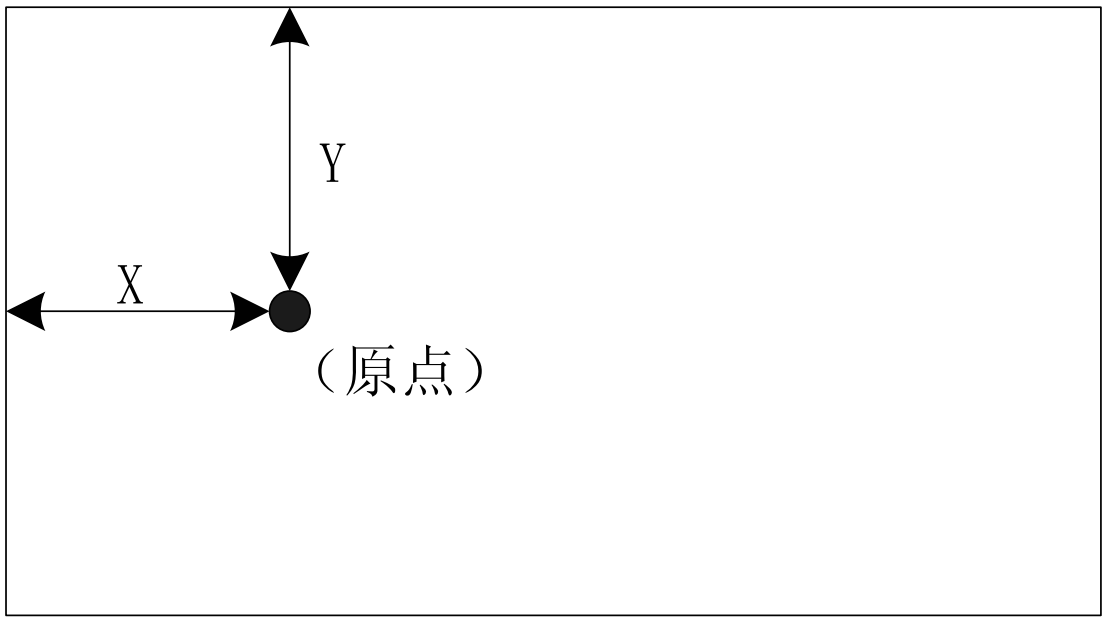
典型的绘图函数(a)

- **void** initOSDC**();**
- **PDC** CreateDC**();**
- **void** DestoryDC**(PDC pdc);**
- **void** MoveTo**(PDC pdc, int x, int y);**
- **void** LineTo**(PDC pdc, int x, int y);**
- **void** DrawRectFrame**(PDC pdc, int left,int top ,int right, int bottom);**
- **void** Circle**(PDC pdc, int x0, int y0, int r);**

典型的绘图函数(b)

void SetDrawOrg(PDC pdc, int x,int y, int* oldx, int *oldy)

设置绘图设备上下文（**DC**）的原点



典型的绘图函数(c)

void SetDrawRange(PDC pdc, int x,int y, int* oldx, int *oldy)

设置绘图设备上下文（DC）的绘图范围

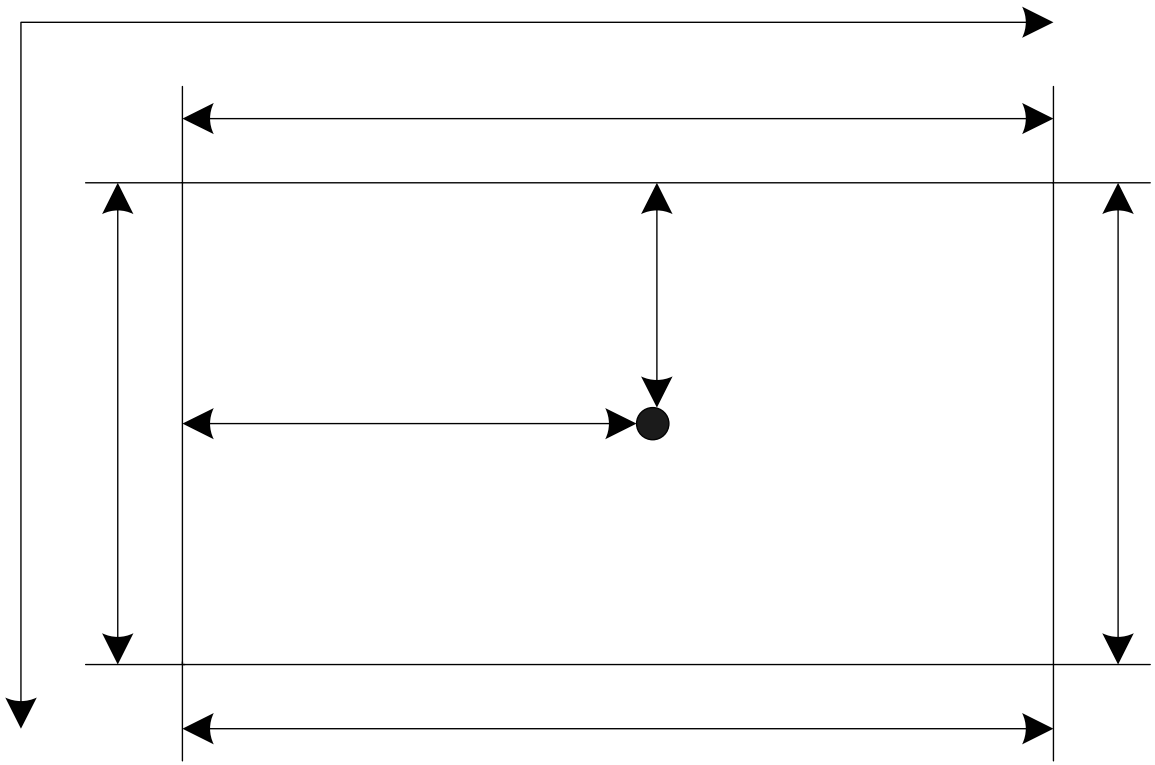
x,y：设定的横向、纵向绘图的范围，如果x（或者y）为1，则表示x（或者y）方向的比例随着y（或者x）方向的范围按比例缩放。如果参数为-1，表示方向相反

DC设置举例

```
PDC pdc;

pdc=CreateDC();

SetDrawOrg(pdc,170,50,&oldx,&oldy);

SetDrawRange(pdc,800,-1,&oldxrange, &oldyrange);
```

上面的程序代码创建一个绘图设备上下文（DC），将原点坐标设定在液晶屏设备坐标的(170,50)，并把绘图的逻辑坐标的水平值设置成800，垂直范围按照液晶屏实际的横纵比例缩放。

DC设置图例



Unicode字库分配

- 本系统中编码采用双字节版本的Unicode格式
- 收集了ASCII字符（0x0000-0x00ff）256个
- 特殊图形符号（0x2600-0x267f和0x2700-0x27bf）320个
- 中文字符（0x4e00-0x9fff）20992个。

Unicode字库相关函数

- 转换函数
  - void Int2Unicode(int number, U16 str[ ]);
  - int Unicode2Int(U16 str[ ]);
  - void strChar2Unicode(U16 ch2[ ], const char ch1[ ]);
- 输出函数
  - void TextOut(PDC pdc, int x, int y, U16 *ch, U8 bunicode, U8 fnt);
  - void TextOutRect(PDC pdc, structRECT* prect, U16* ch, U8 bunicode, U8 fnt)

典型的控件

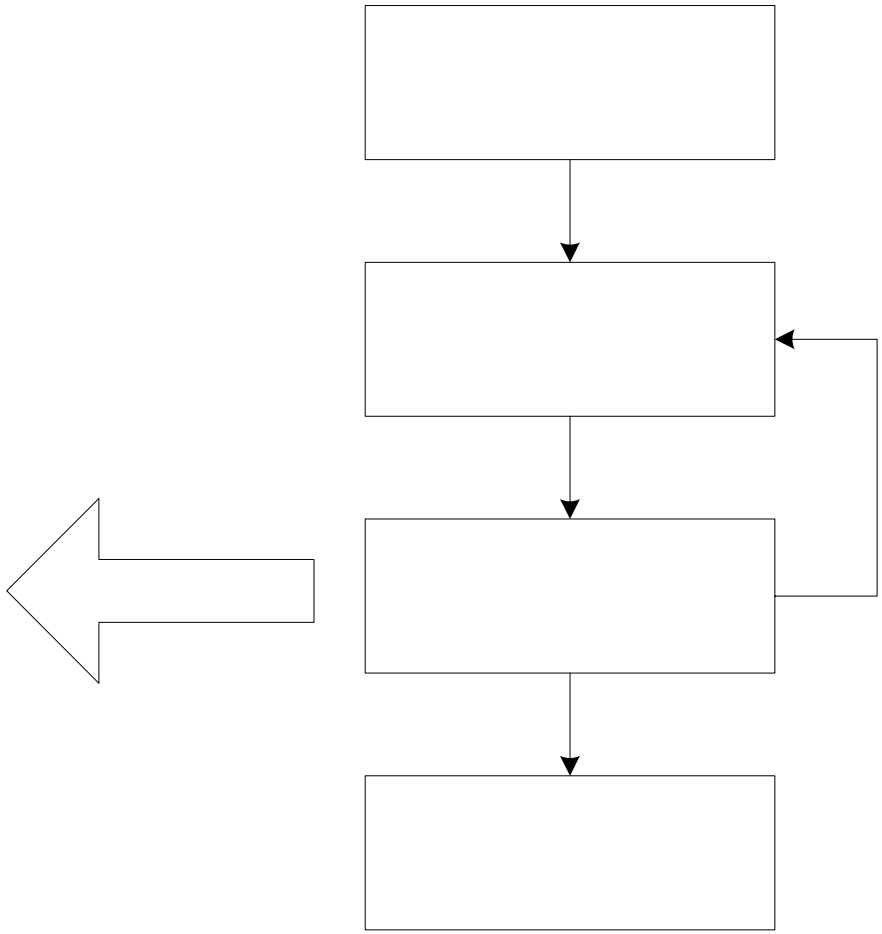
控件是可视化开发的基础。对于开发应用程序的用户来说，控件是一个独立的组件，它有着自己的显示方式，自己的动态内存管理模式，甚至有的控件还可以向系统发送自己的消息。用户不需要掌握控件的内部到底是如何工作的，用户只需要通过控件提供的API函数，改变控件相应的属性，从而改变控件的显示方式。

- 控件的引入可以方便用户的开发，加速用户应用程序界面的编写速度。
- 为运行在操作系统上的应用程序的界面提供了统一的标准，方便了使用。

系统中的控件

- 文本框控件
- 列表框控件
- 图片框控件
- 按钮控件
- 窗体

控件的使用流程



通用的系统控件

对一个通用的系统控件，包含了如下的数据结构：

```
typedef struct{
        U32 CtrlType;      //控件的类型
        U32 CtrlID;                  //控件的ID
        structRECT ListCtrlRect;//控件的位置和大小
        U32 FontSize;      //控件的字符大小
        U32 style;                   //控件的的边框风格
        U8 bVisible;       //是否可见
}OS_Ctrl;
```

- 基于μC/OS-II的软件设计
  - 文件系统
  - 图形用户接口（GUI）
- μC/OS-II for ARM BSP的实现

μC/OS-II BSP编写

BSP（板级支持包）是介于底层硬件和操作系统之间的软件层次，它完成系统上电后最初的硬件和软件初始化，并对底层硬件进行封装，使得操作系统不再面对具体的操作。

BSP的特点：

- 硬件相关性：因为嵌入式实时系统的硬件环境具有应用相关性，所以，作为高层软件与硬件之间的接口，BSP必须为操作系统提供操作和控制具体硬件的方法。
- 操作系统相关性：不同的操作系统具有各自的软件层次结构，因此，不同的操作系统具有特定的硬件接口形式。

BSP的功能

- 嵌入式系统初始化
  - A、片级初始化
  - B、板级初始化
  - C、系统级初始化
- 硬件相关的设备驱动程序

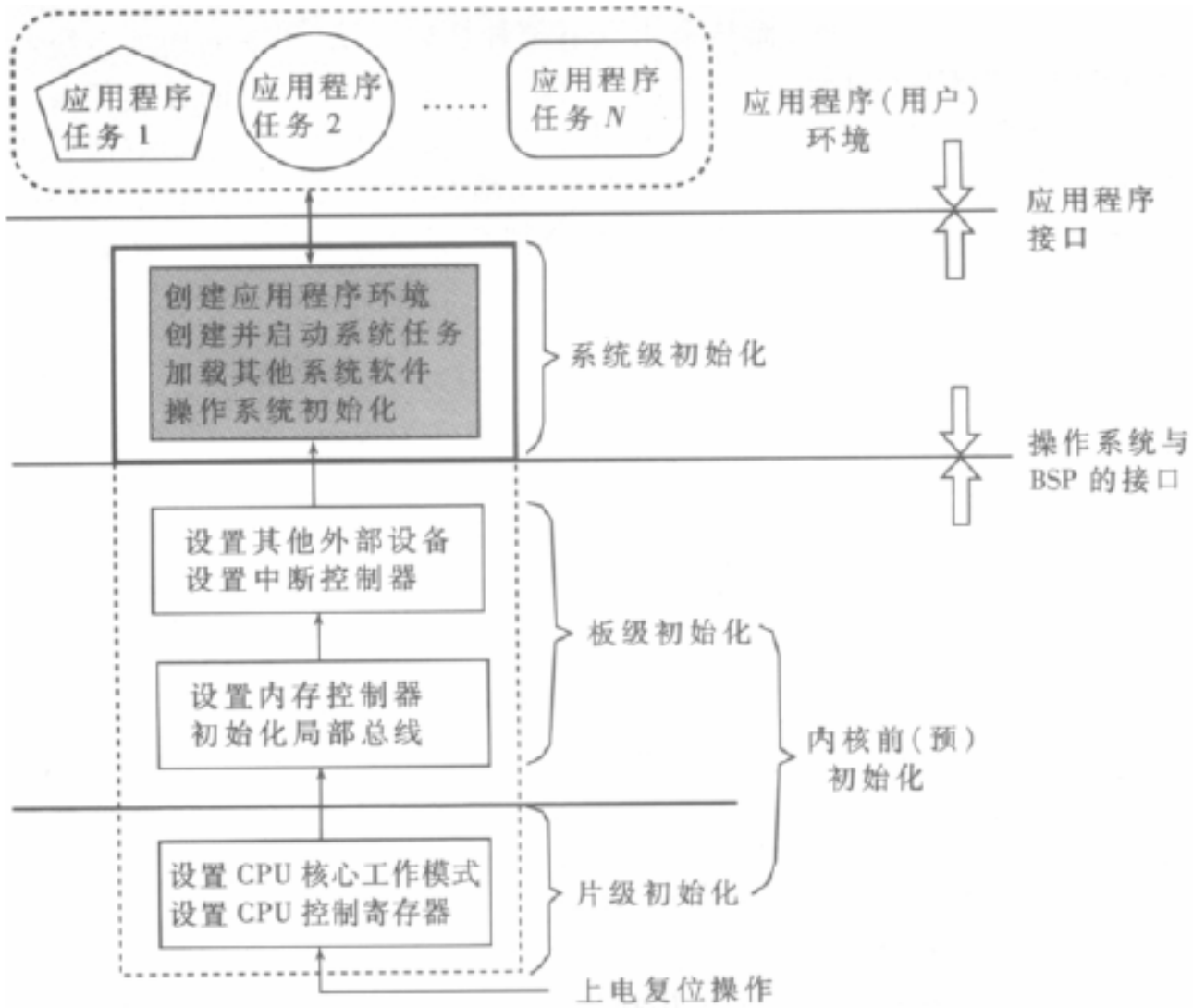
嵌入式系统始化

1）片级初始化：主要完成微处理器的初始化， 包括设置微处理器的核心寄存器和控制寄存器，微处理器核心工作模式以及其局部总线模式等。片级初始化把微处理器从上电时的缺省状态逐步设置成为系统所要求的工作状态。这是一个纯硬件的初始化过程

2）板级初始化：完成微处理器以外的其他硬件设备的初始化。除此之外，还要设置某些软件的数据结构和参数，为随后的系统级初始化和应用程序的运行建立硬件和软件环境。这是一个同时包含软硬件两部分在内的初始化过程。

3）系统级初始化：这是一个以软件初始化为主的过程， 主要进行操作系统初始化。BSP将控制转交给操作系统， 由操作系统进行余下的初始化操作。包括加载和初始化与硬件无关的设备驱动程序，建立系统内存区，加载并初始化其他系统软件模块，比如网络系统、文件系统等；最后，操作系统创建应用程序环境并将控制转交给应用程序的入口

嵌入式系统初始化过程及BSP功能

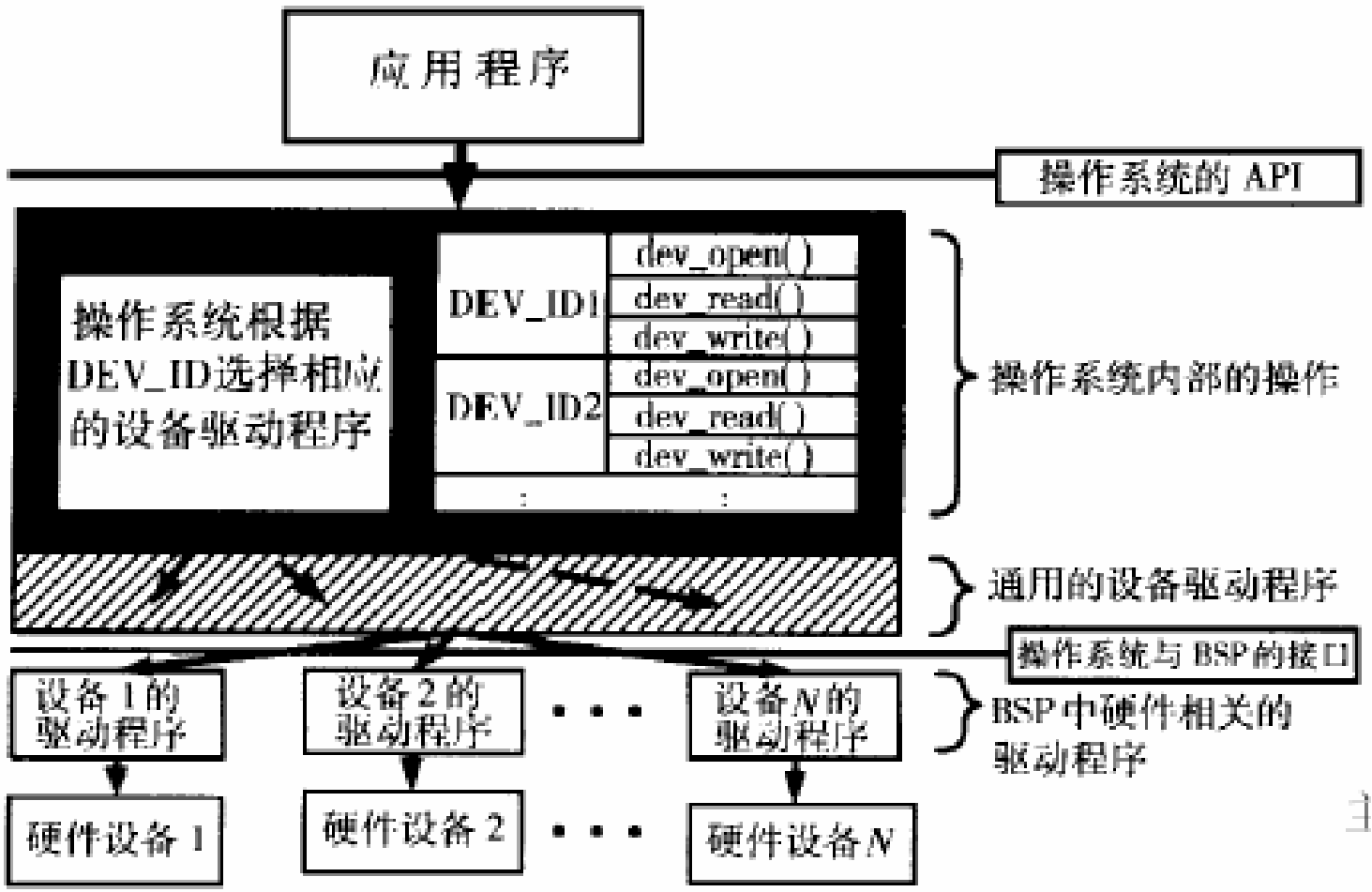


完成硬件相关的设备驱动

1）BSP另一个主要功能是硬件相关的设备驱动。与初始化过程相反，硬件相关的设备驱动程序的初始化和使用通常是一个从高层到底层的过程。尽管BSP 中包含硬件相关的设备驱动程序，但是这些设备驱动程序通常不直接由BSP使用，而是在系统初始化过程中由BSP 把它们与操作系统中通用的设备驱动程序关联起来，并在随后的应用中由通用的设备驱动程序调用，实现对硬件设备的操作。

2）设计与硬件相关的驱动程序是BSP设计中另一个关键环节

系统调用通用设备驱动程序与BSP的关系



设计BSP的方法

一、以典型的BSP做为参考

二、参照操作系统或芯片厂商提供的BSP模板

μC/OS-II BSP for ARM

- μC/OS-II编写一个简单的BSP。它首先设置CPU内部寄存器和系统堆栈，并初始化堆栈指针，建立程序的运行和调用环境;
- 然后可以方便地使用C语言设置ARM片选地址（CS0～CS7）、GPIO以及SDRAM控制器，初始化串口（UART0）作为默认打印口，并向操作系统提供一些硬件相关例程和函数如dprintf()，以方便调试;
- 在CPU、板级和程序自身初始化完成后，就可以把CPU的控制权交给操作系统了

谢谢！