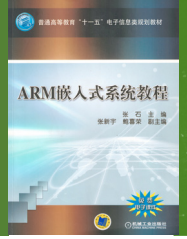
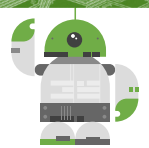
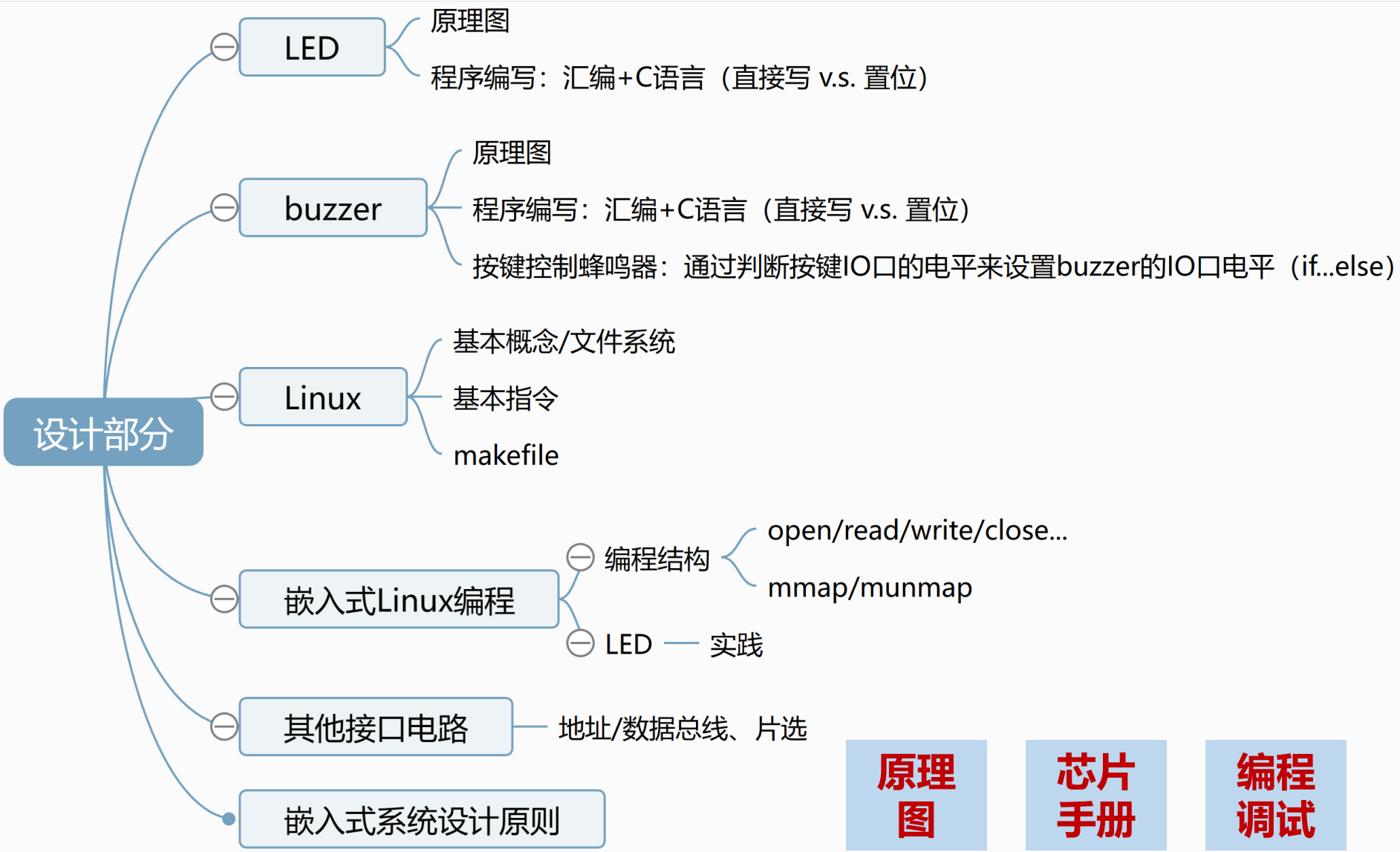
江苏师范大学 电气工程及自动化学院
JIANGSU NORMAL UNIVERSITY SCHOOL OF ELECTRICAL ENGINEERING & AUTOMATION





第6章 嵌入式系统设计实例

李灿 | 12#407A | lic@jsnu.edu.cn | https://sslic.cn/es


设计部分
LED
原理图
程序编写：汇编+C语言（直接写 v.s. 置位）
buzzer
原理图
程序编写：汇编+C语言（直接写 v.s. 置位）
按键控制蜂鸣器：通过判断按键IO口的电平来设置buzzer的IO口电平（if...else）
Linux
基本概念/文件系统
基本指令
makefile
嵌入式Linux编程
编程结构
open/read/write/close...
mmap/munmap
LED — 实践
其他接口电路
地址/数据总线、片选
嵌入式系统设计原则
原理图
芯片手册
编程调试
需求分析
芯片选型
外设选型
电路设计
驱动程序
应用程序

嵌入式Linux系统交叉开发

- **基本思想：**
  - ① 在宿主机（Host）上安装开发工具，编辑、编译目标板的Linux引导程序、内核、文件系统、驱动/应用程序
  - ② 下载到目标板（Target）上执行
- **编译器：**
  - ✧ gcc, gdb(调试)
  - ✧ arm-linux-gcc
- **Linux操作系统移植3个映像文件：**
  - ✧ 启动代码（bootloader）
  - ✧ 内核（Kernel）
  - ✧ 根文件系统（Root filesystem）

嵌入式Linux系统交叉开发

- ARM Linux启动代码——Bootloader
  - Bootloader是Linux启动之前运行的一小段代码，上电后先运行

    回顾S5PV210的启动过程：

    ① iROM（片上）

    ② iRAM ← Bootloader1（Flash...）

    ③ DRAM ← Bootloader2（Flash...）

    ④ DRAM ← Linux OS（Flash...）
- Bootloader的主要功能
  - 初始化硬件：CPU时钟、内存管理、中断、GPIO、UART...
  - 启动Linux：将内核映像复制到SDRAM中，并跳转到内核入口处
  - 下载映像：从宿主机下载Bootloader、Kernel、Filesystem映像
  - 存储器管理：对目标板中固态存储的存、写、读、锁、解锁

嵌入式Linux系统交叉开发

- 不同CPU体系结构有不同的Bootloader，常用的有：
- Redboot
- U-boot
- Blob

Linux指令

命令	功能
`man`	帮助命令，如 `man ls`
`ls`	查看当前或指定路径目录，常用可选参数 `-a`，`-l`
`cd`	修改文件路径，如 `cd ~`，`cd ..`，`cd /usr/bin`
`mv`	移动文件，如 `mv test /mnt`；重命名，如 `mv source target`
`cp`	复制文件，如 `cp source target`
`mkdir`	创建文件目录，如 `mkdir -p h/h.c`，可选参数 `-p`，确保目录存在
`rm`	删除指定文件，递归删除`rm -r`，强制删除`rm -f`
`rmdir`	删除当前目录下的子目录
`pwd`	显示当前目录（路径）
`chmod`	设置文件操作权限，如，可读可写可执行 `chmod 777 filename`
`touch`	创建新文件，如 `touch test.c`

Linux指令

命令	功能
`gedit`	若已安装，可打开gedit编辑器，如 `gedit ex.c`
`subl`	若已安装，可打开subl编辑器，如 `subl ex.c`
`file`	检查二进制文件，如 `file hello`，`file hello-arm`
`cat`	显示文件的内容，如 `cat file`
`top`	查看系统信息
`sleep`	系统挂起，如，挂起9秒 `sleep 9`
`reboot`	重启系统
`ping`	网络命令，测试网络连通性，如 `ping 192.168.1.230`
`ifconfig`	查看/设置网卡
`telnet`	若已安装，可远程登陆服务器，如 `telnet 192.168.1.230`

makefile

■ 编译汇编程序时的makefile举例

```
leds: leds.s
	arm-linux-gcc -c -o leds.o leds.s
	arm-linux-gcc -Ttext 0x20000000 leds.o -o leds.elf
	arm-linux-objcopy -O binary -S leds.elf leds.bin
clean:
	rm -f *.o *.elf *.bin
```

```
leds: leds.s ledc.c
	arm-linux-gcc -c -o leds.o leds.s
	arm-linux-gcc -c -o ledc.o ledc.c
	arm-linux-gcc -Ttext 0x20000000 leds.o ledc.o -o led.elf
	arm-linux-objcopy -O binary -S led.elf led.bin
clean:
	rm -f *.o *.elf *.bin
```

makefile

■ 编译C程序的makefile举例

```
sumf: exm.o exf.o
	arm-linux-gcc exm.o exf.o -o sumf
exm.o: exm.c
	arm-linux-gcc -c exm.c
exf.o: exf.c
	arm-linux-gcc -c exf.c
clean: rm -f *.o sumf
```

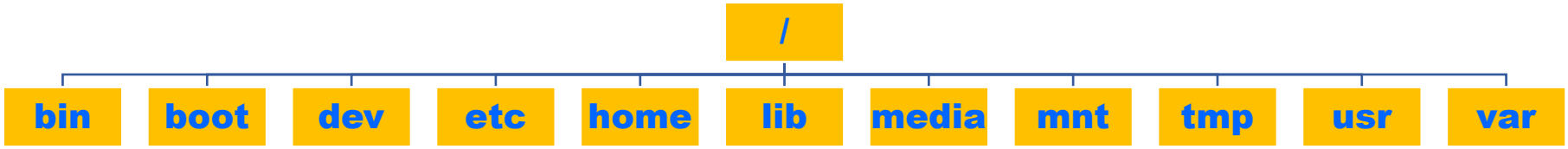
使用方法：：
终端执行：`make`
终端删除：`make clean`

```
CC = arm-linux-gcc
TARGET = sumf
all : $(TARGET)
$(TARGET) : exm.c exf.c
	$(CC) $? -o $@
clean : rm -f *.o $(TARGET)
```

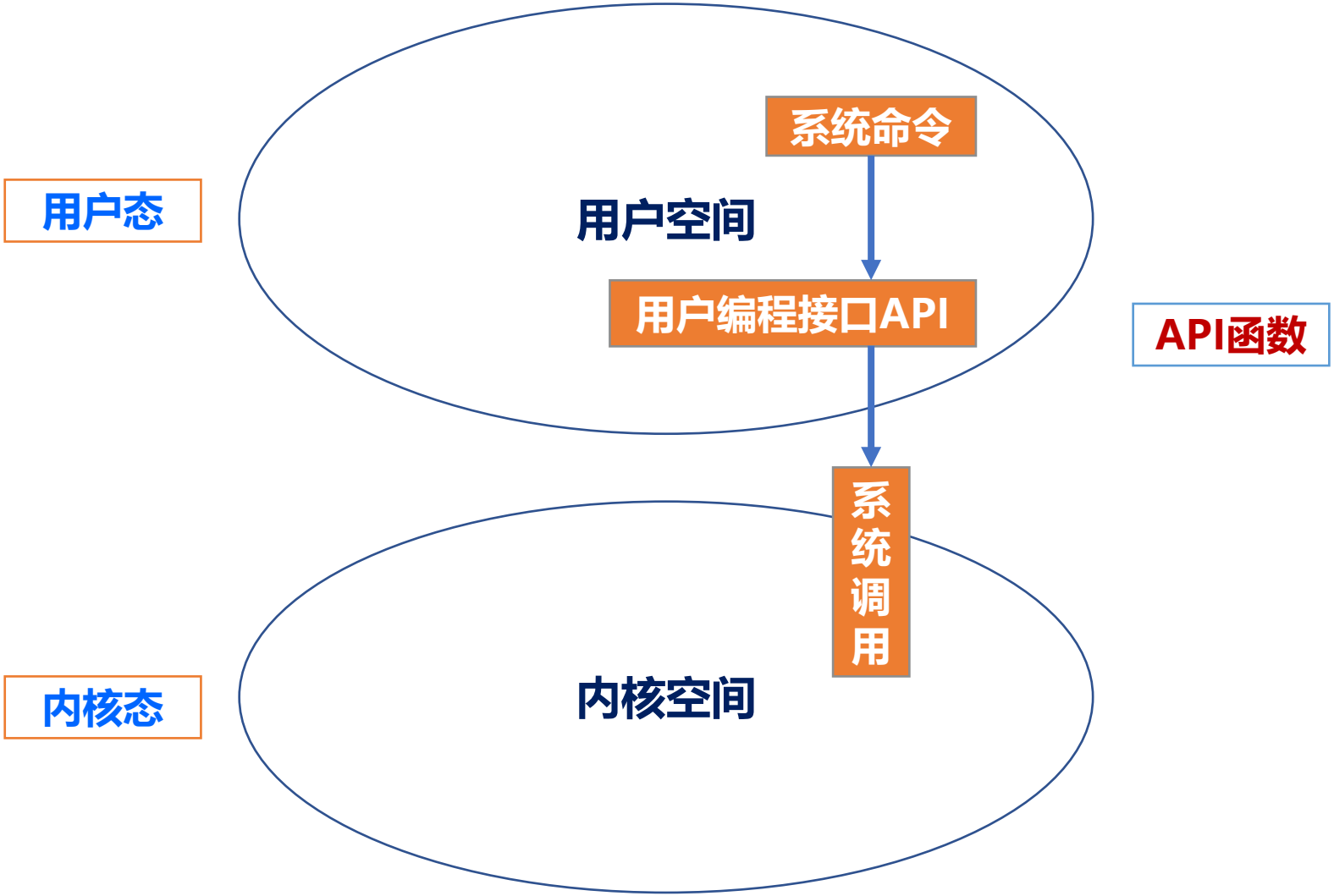
makefile中常用的自动变量

命令	含义
`$*`	不包含扩展名的目标文件名称
`$+`	所有依赖文件，以空格分开，以出现先后顺序排序
`$<`	第一个依赖文件的名称
`$?`	所有时间戳比目标文件晚的依赖文件，以空格分开
`$@`	目标文件的完整名称
`$^`	所有不重复的依赖文件，以空格分开
`%.o : %.c`	所以.c文件编译产生.o文件

Linux文件系统



嵌入式Linux编程结构



嵌入式Linux编程

- 在Linux 中对目录和设备的操作都等同于文件的操作，极大简化了系统对不同设备的处理，提高了效率
- Linux 中的文件主要为：普通文件、目录文件、链接文件和设备文件
- 内核区分和引用特定的文件的方式——文件描述符
  - 文件描述符是一个非负整数，指向内核中每个进程打开文件的记录表
  - 当打开或创建一个文件时，内核就向进程返回一个文件描述符
  - 当需要读写文件时，要将文件描述符作为参数传递给相应函数
  - 通常，一个进程启动时，都会打开3个文件：标准输入、标准输出和标准出错处理。这3个文件分别对应文件描述符为0、1、2
    （即宏替换STDIN_FILENO、STDOUT_FILENO、STDERR_FILENO）

嵌入式Linux编程

- 常见的I/O操作函数：
  - open
  - read
  - write
  - lseek
  - close
- 虚拟地址（运行时）映射函数：
  - mmap
  - munmap

open 函数语法要点

所需头文件	#include <sys/types.h> // 提供类型 pid_t 的定义 #include <sys/stat.h> #include <fcntl.h>	
函数原型	int open(const char *pathname， flags， int perms)	
函数传入值	pathname	被打开的文件名（可包括路径名）
	flag：文件打开的方式	O_RDONLY：只读方式打开文件
		O_WRONLY：可写方式打开文件
		O_RDWR：读写方式打开文件
		O_CREAT：如果该文件不存在，就创建一个新的文件，并用第三个参数为其设置权限
		O_EXCL：如果使用 O_CREAT 时文件存在，则可返回错误消息。这一参数可测试文件是否存在
		O_NOCTTY：使用本参数时，如文件为终端，那么终端不可以作为调用 open()系统调用的那个进程的控制终端
		O_TRUNC：如文件已经存在，并且以只读或只写成功打开，那么会先全部删除文件中原有数据
		O+APPEND：以添加方式打开文件，在打开文件的同时，文件指针指向文件的末尾
	perms	被打开文件的存取权限，为 8 进制表示法
函数返回值	成功：返回文件描述符 失败：−1	

flag参数可通过“|”组合，但前3个参数不能相互组合

close 函数语法要点

所需头文件	#include <unistd.h>
函数原型	int close(int fd)
函数输入值	fd：文件描述符
函数返回值	0：成功 −1：出错

```
int main(void)
{
      int fd;
/*调用 open 函数，以可读写的方式打开，注意选项可以用“|”符号连接*/
      if((fd = open("/tmp/hello.c", O_CREAT | O_TRUNC | O_WRONLY , 0600 ))<0){
            perror("open:");
            exit(1);
      }
      else{
            printf("Open file: hello.c %d\n",fd);
      }
      if( close(fd) < 0 ){
            perror("close:");
            exit(1);
      }
      else
            printf("Close hello.c\n");
      exit(0);
}
```

返回给fd一个数字

嵌入式Linux编程

■ 虚拟地址映射与释放

```
void *mmap(void *addr, size_t length, int prot, int flags, int fd, off_t offset);
```

- **addr**：指定映射的虚拟内存地址，设为 **NULL**时，由 Linux 内核自动选择虚拟内存地址
- **length**：映射的长度
- **prot**：映射内存的保护模式（属性设置）：
  - **PROT_EXEC**：可执行
  - **PROT_READ**：可读取
  - **PROT_WRITE**：可写入
  - **PROT_NONE**：不可访问
- **flags**：指定映射的类型（属性设置），常用可选值：
  - **MAP_FIXED**：使用指定的起始虚拟内存地址进行映射
  - **MAP_SHARED**：与其它所有映射到这个文件的进程共享映射空间（可实现共享内存）
  - **MAP_PRIVATE**：建立一个写时复制（Copy on Write）的私有映射空间
  - **MAP_LOCKED**：锁定映射区的页面，从而防止页面被交换出内存
  - ……
- **fd**：进行映射的文件描述符（非负数字）
- **offset**：文件偏移量（从文件何处开始映射，操作SFR时，可以选为寄存器组的基地址）

嵌入式Linux编程

虚拟地址映射与释放

```
void *mmap(void *addr, size_t length, int prot, int flags, int fd, off_t offset);
```

```
int fd = open(filepath, O_RDWR, 600);
void *addr = mmap(NULL, 4*4096, PROT_WRITE, MAP_SHARED, fd, BaseAdrr);
```

```
int munmap(void *addr, size_t length); // 原型声明，成功时返回值为0，否则返回-1
```

```
munmap( (void *)addr, 4*4096); // 释放映射的虚拟地址addr
```