

**JIANGXINORMALUNIVERSITY**

**嵌入式系统开发技术**

**毕业实习设计**

专 业： 电子信息工程

班 级： 14级电子信息工程（1）班

学 号： 1408071026 1408071021

姓 名： 胡玉杰 何弘

设计题目： 按键和蜂鸣器驱动程序设计

**2018年1月**

目 录

# 一、概要

目前嵌入式系统已发展成为以高速处理器和嵌入式操作系统作为核心的软硬件综合系统，Linux作为一种开源、跨平台的操作系统，受到了越来越多开发者的青睐。随着物联网和人工智能的发展，Linux将更多地应用在嵌入式设备中，这对Linux内核中的驱动设计和实现也提出了更高的要求。

驱动程序在 Linux 内核里扮演着特殊的角色。它们完全隐藏了设备工作的细节，用户的活动通过 一套标准化的调用来进行, 这些调用与特别的驱动是独立的；设备驱动的角色就是将这些调用映射 到作用于实际硬件的和设备相关的操作上。驱动可以与内核的其他部分分开建立, 并在需要的时候 在运行时"插入"。这种模块化使得 Linux 的驱动程序非常的灵活。

Linux是一套可以免费使用和自由传播的类UNIX操作系统，其实际上只是一个操作系统的内核，主要用于Intelx86系列CPU的计算机上，谈及Linux的起源，起灵感来自于UNIX。UNIX操作系统于1969年由Bell实验室设计开发，之后Linus Torvalds设计了Linux，该系统在发展初期就得到了广大程序员的帮助，逐步发展成为现今这样一个拥有自己版权的完整的系统。Linux具有很多特点，如支持多种体系结构，支持大量的外围设备，具有完善的网络功能，开放源代码，软件资源丰富， 内核稳定等，可以总结为一下几点：

（1）强大的编程能力。由于Linux源自于世界各地成千上万的程序员和黑客，使得Linux就犹如加入到了一个感受如云的编程组织中，同时由于GPL的存在，Linux开放源代码，吸引更多专业人士的加入，在这种需求的刺激下，Linux提供的开发工具功能也越来越完善，越来越强大。

（2）完善的组网能力。Linux具有强大的组网能力，它对当前的TCP/IP协议提供了完全的支持，同时也支持下一代Internet协议Ipv6.在安全性方面，Linux内核中包括了IP防火墙代码、IP防伪及IP服务质量控制等特性。

（3）Linux是自由开放的。Linux和运行在其上的自由软件，允许成千上万的人检查软件，修改软件，最终可以按照用户自己的意愿来定义自由软件，可以定制自己的Linux。

（4）系统稳定。Linux提供了完全的内存保护机制，每个进程都运行在各自的虚拟地址空间中，不会损坏进程或内核使用的地址空间。

# 二、设计内容

本次设计是分别实现蜂鸣器驱动和按键驱动，以及将蜂鸣器驱动和按键驱动进行组合联调，基本熟悉Linux字符设备开发流程，达到毕业实习预期的目的。

# 三、准备工作

## 1、开发环境的搭建

### 1)、安装VMware Workstation和Red Hat Enterprise Linux 5

安装并打开VMware Workstation，如图1所示。

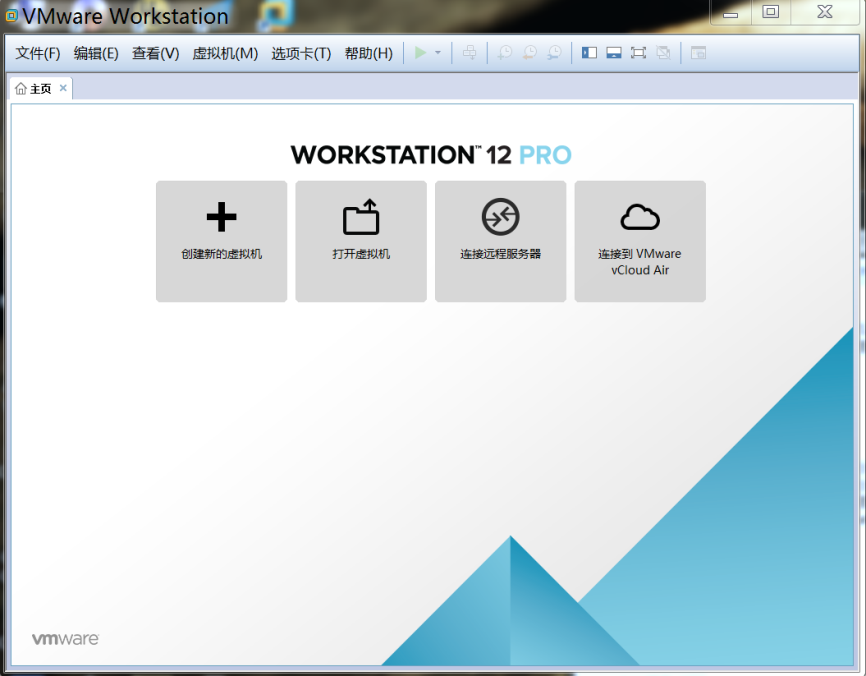


图 1 打开VMware Workstation

解压拷贝到的已安装好的Red Hat Enterprise Linux 5镜像，然后在VMware Workstation中将其打开。

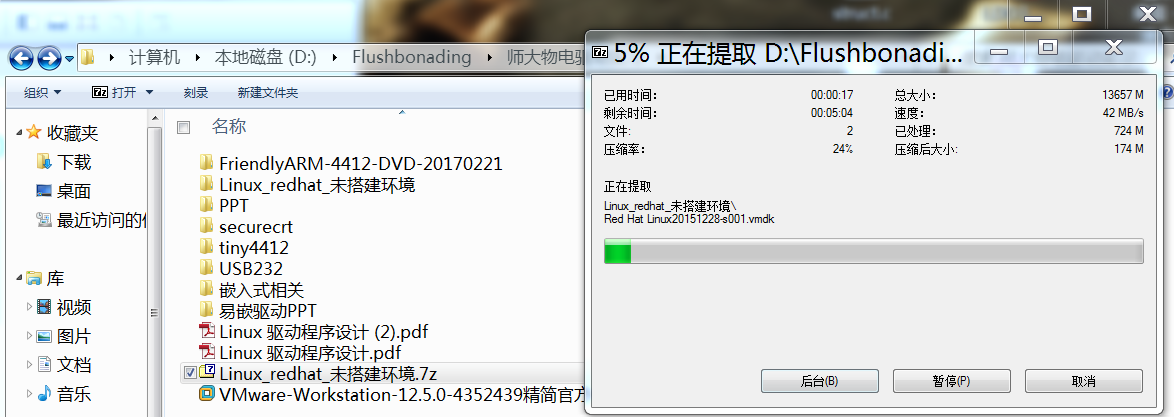


图2

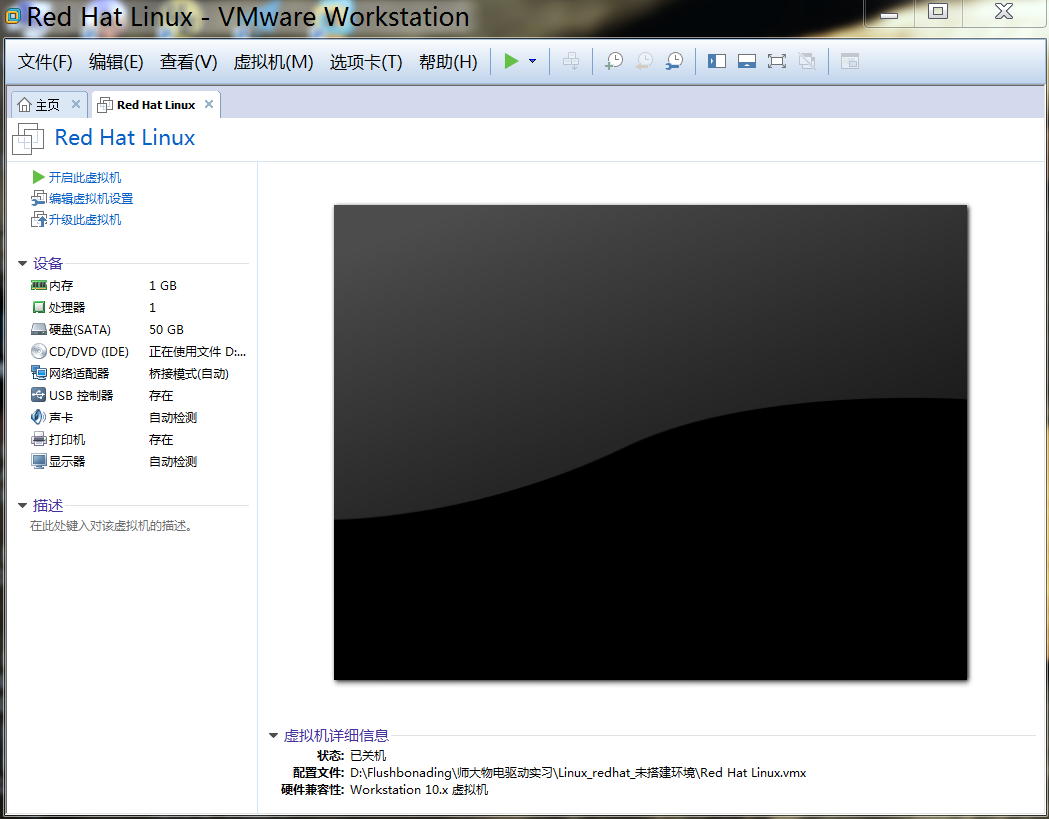


图 3

配置共享文件夹，以便于Red Hat Enterprise Linux 5虚拟机和宿主机交换文件。本次设置的共享文件夹目录为D:\Flushbonading\师大物电驱动实习\Linux\_redhat\_未搭建环境\share。

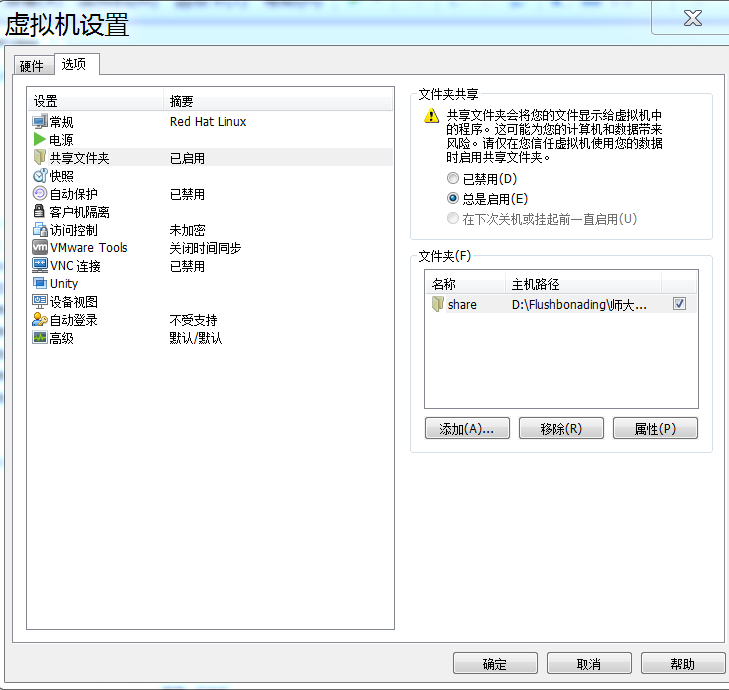


图 4

打开虚拟机，并以root账户登录，密码为123456

### 2)、安装交叉编译工具arm-linux-gcc

将arm-linux-gcc-4.5.1-v6-vfp-20120301.tgz和linux-3.5-20170221.tgz拷贝到共享文件夹。

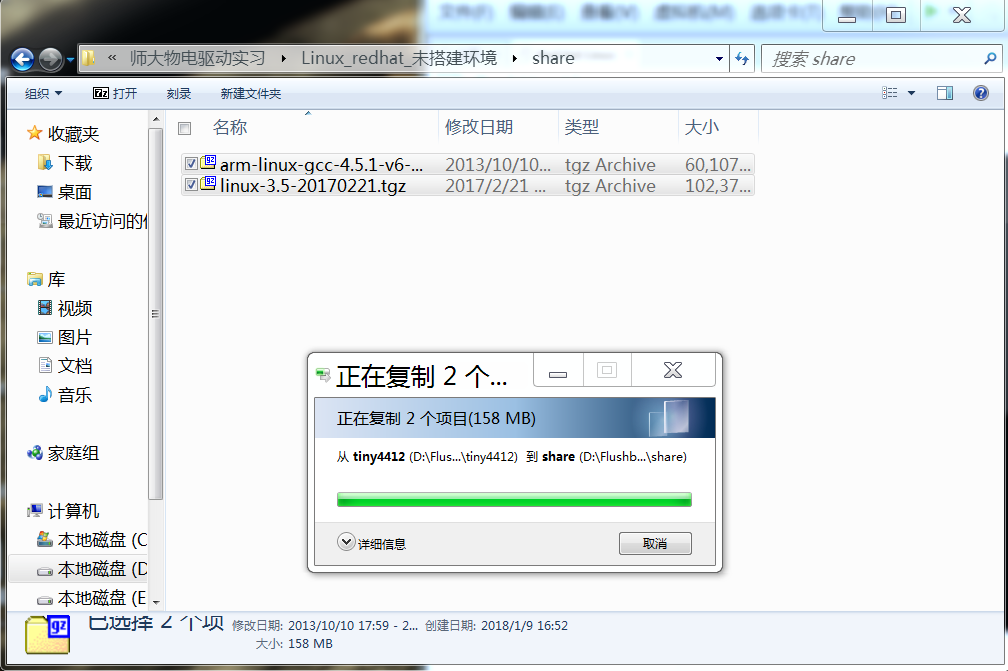


图 5

在虚拟机中打开终端，进入到共享文件夹/mnt/hgfs/share

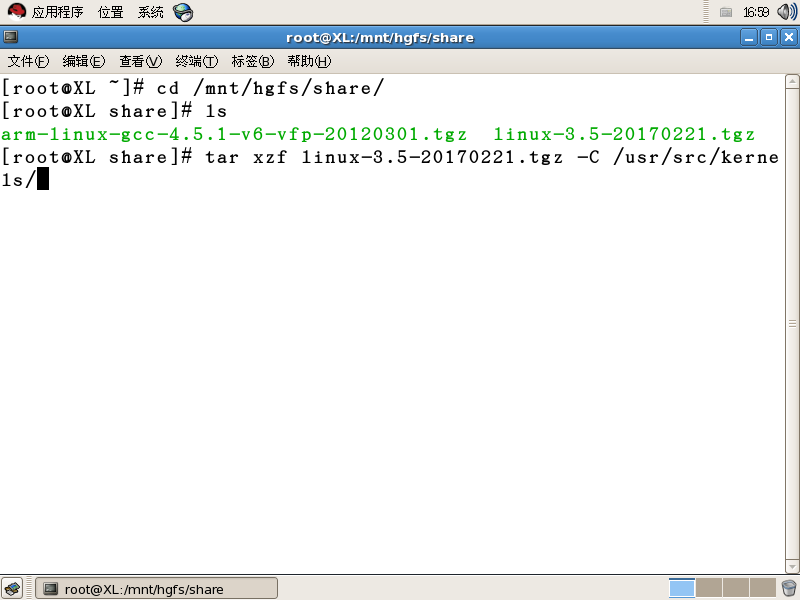


图 6

将arm-linux-gcc-4.5.1-v6-vfp-20120301.tgz tar使用tar命令解压到/opt目录下。将linux-3.5-20170221.tgz使用tar命令解压到/usr/src/kernels目录下。复制解压的交叉编译工具arm-linux-gcc的执行文件所在的目录绝对路径，并将其加入到环境变量中，且用source命令使环境变量生效。

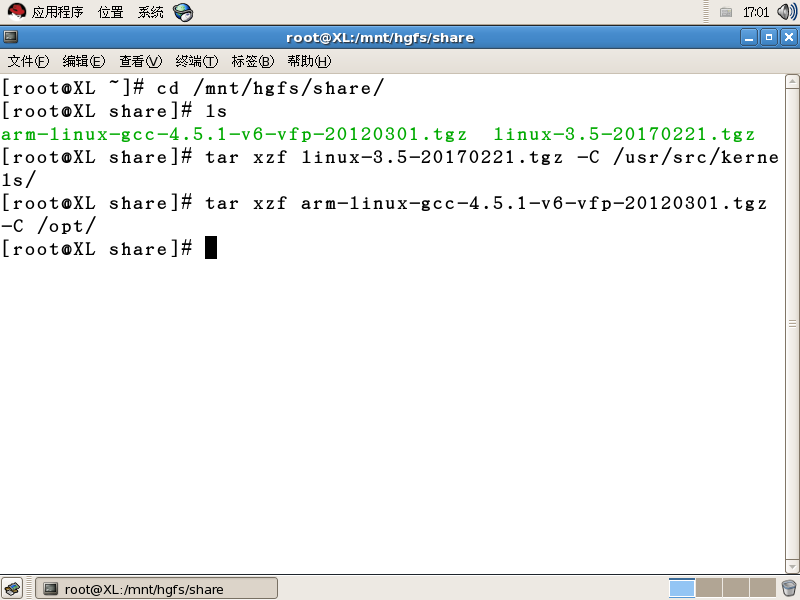


图 7

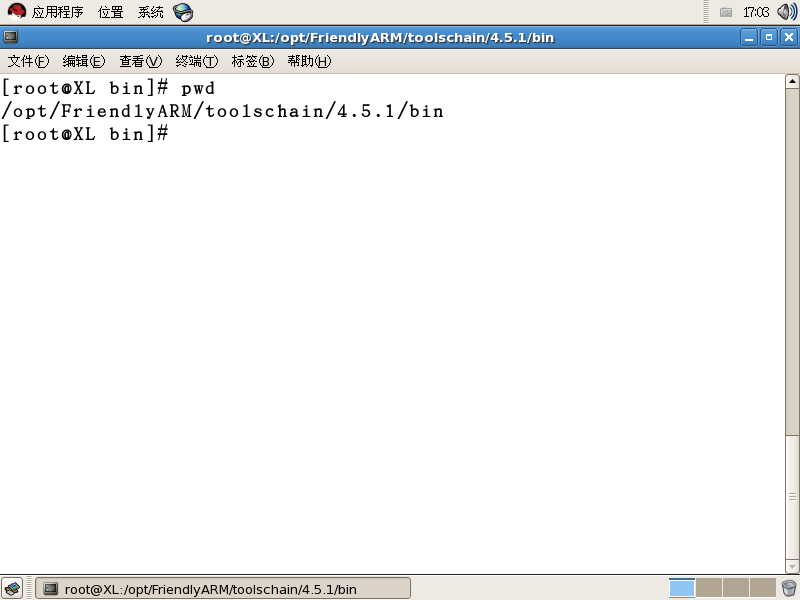


图 8

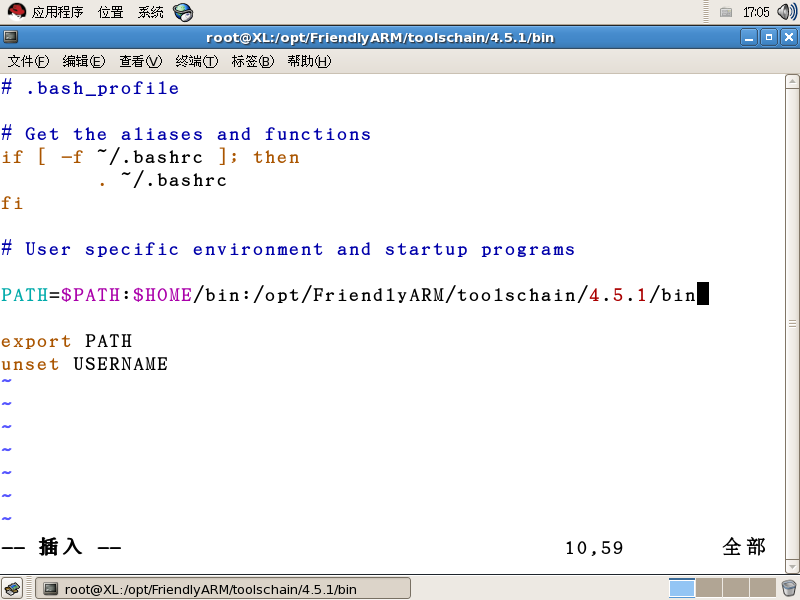


图 9

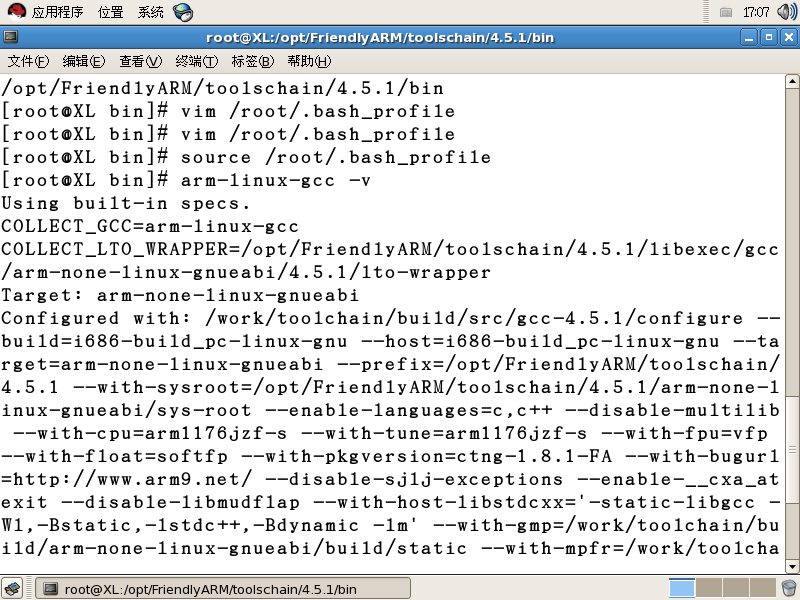


图 10

执行arm-linux-gcc –v出现如上所示的版本信息，则交叉编译工具已搭建好。

### 3)、构建开发板内核树

进入到前面解压的开发板内核源码文件夹(/usr/src/kernels/linux-3.5)。

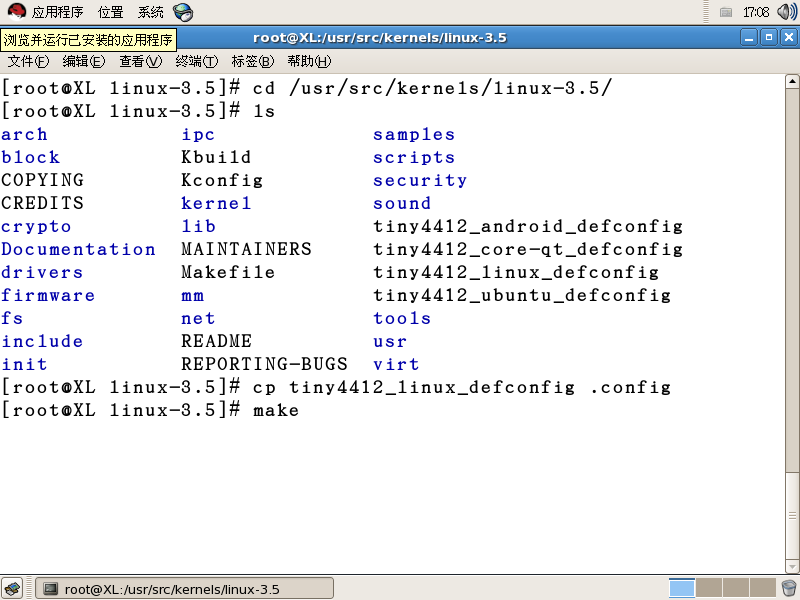


图 11

由于Linux源码支持非常多的硬件架构，编译内核的架构是仅仅由配置参数决定，所以执行cp tiny4412\_linux\_defconfig .config ，将友善之臂配置的Linux配置文件作为当前构建内核树的配置文件，然后执行make命令，等待期编译完。至此，交叉编译环境已经搭建完毕了。

## 2、C语言知识复习

### 1)、变量存储位置

栈区stack：由编译器自动分配释放，存放函数的参数值，函数的返回地址，局部变量等，#栈区容量有限，不能溢出。

堆区heap：一般由程序员分配释放。

全局区（静态区，数据区，静态数据区，数据段）static：全局变量和静态变量的存储是放在一块的，初始化的全局变量和静态变量在一块区域，未初始化的全局变量和静态变量存放在另一块相邻的区域，程序结束后由系统释放。

文字常量区：常量字符串，程序结束后由系统释放。

程序代码区：存放函数体的二进制代码。

### 2)、函数返回指针的几种方法

a、在函数内用malloc，由调用者释放

b、在函数内使用static 定义

c、由调用者分配空间，只是把指针传给函数，函数内部把数据拷贝到内存中。（推荐）

### 3)、其他杂项

函数指针

返回值类型 (\*指针变量名)([形参列表])。

## 3、Linux基本操作命令

### 1)、查看文件的几种方法

cat: 打印文件全部内容

more ： 分页显示（回车键），只能向下

less： 使用方向键向上或向下

head：head -10 xx 文件首10行

tail：tail -5 xx 文件尾部5行

### 2)、关于Linux用户管理的命令

su： 切换用户，默认的参数为root

用户组的信息构成：通过查看/etc/passwd文件可以看到如下信息，第一个500表示UID，即用户ID，第二个500表示GID，即组ID。

double:x:500:500::/home/double:/bin/bash

adduser： -g 加入组，默认建立主目录/home/xx

userdel： userdel xx [-rf] 删除用户xx[和其主目录]

usermod： usermod xx -g root 更改用户所属组

### 3)、gcc 编译过程

a、预处理 -E xx.i

b、编译 语法检查 -S xx.S

c、汇编 -c xx.o

d、链接 -l(library) xx

# 四、字符设备驱动基本结构

## 1、内核核心功能

进程管理。

内存管理。

文件管理。

网路管理。

设备管理。

## 2、驱动程序在系统中扮演的角色

在Linux系统中，Linux使用了ring0模式来运行内核态的代码，ring3来运行用户态的代码。处于ring3层面的应用程序不直接和硬件交互，也无法访问硬件，以一次简化的用户空间发起的磁盘读数据为例，过程如下：



图 12

其中，驱动程序一般工作在内核空间，应用程序工作在用户空间。在用户程序想要读取硬盘数据时，该进程首先通过系统调用read，从用户态转移到内核态执行sys\_read，在内核态中通过sys\_read对应到相应的file\_operations xx\_ops.read,再映射到xx\_read函数，然后将数据从硬盘读出，存入内核空间处的Buff，再通过copy\_to\_user，将数据传输到用户空间的Buff，执行读操作。

工作在内核空间的驱动可以访问全部的硬件资源，而用户空间的程序则不行。在操作系统中，用户空间的程序出错，一般并不会导致整个系统崩溃。操作系统运行模式的存在，有效的保护了系统的稳定性。

所以说，驱动程序在用户程序与系统硬件之间扮演者不可缺少的角色。

## 3、字符驱动程序的基本框架

如下图为字符驱动的基本架构，以及如何与用户空间的交互。



图 13

## 4、驱动程序和应用程序的区别

大部分中、小规模的应用程序从头至尾处理一个单个任务,而每个内核模块却只是事先注册自己以便服务将来的某个请求, 并且它的初始化函数立刻终止。换句话说,模块初始化函数的任务是为以后调用模块的函数做准备；好像是模块说, "我在这里,这是我能做的."模块的退出函数( 例子里是hello\_exit )就在模块被卸载时调用。它好像告诉内核, "我不再在那里了,不要要求我做任何事了."这种编程的方法类似于事件驱动的编程,但是虽然不是所有的应用程序都是事件驱动的, 但每个内核模块都是。在事件驱动的应用程序和内核代码之间的另外一个主要的不同是：应用程序退出时，可以不管资源的释放和其他的清除工作，但是模块的退出函数必须小心恢复每个由初始化函数建立的东西,否则会保留一些东西直到系统重启。

## 5、驱动类型

### 1)、字符设备

一个字符(char)设备是一种可以当作一个字节流来存取的设备(如同一个文件)；一个字符驱动负责实现这种行为。这样的驱动常常至少实现open, close, read, 和write 系统调用。文本控制台(/dev/console)和串口(/dev/ttySn)是字符设备的例子,因为它们很好地展现了流的抽象。字符设备通过文件系统结点来存取,例如/dev/tty1 和/dev/lp0。在一个字符设备和一个普通文件之间唯一的不同就是,你经常可以在普通文件中移来移去,但是大部分字符设备仅仅是数据通道,你只能顺序存取。

### 2)、块设备

如同字符设备,块设备通过位于/dev 目录的文件系统结点来存取。一个块设备(例如一个磁盘)应该是可以驻有一个文件系统的。在大部分的Unix 系统,一个块设备只能处理这样的 I/O 操作,传送一个或多个长度经常是 512 字节(或一个更大的2 的幂的数)的整块.相反, Linux 允许应用程序读写一个块设备像一个字符设备一样----它允许一次传送任意数目的字节。

结果就是,块和字符设备的区别仅仅在内核在内部管理数据的方式上,并且因此在内核和驱动的软件接口上不同。如同一个字符设备,每个块设备都通过一个文件系统结点被存取的, 它们之间的区别对用户是透明的。块驱动和字符驱动相比,与内核的接口完全不同。

### 3)、网络设备

通过单独的网络接口调用任何网络事务都通过一个接口来进行,就是说,一个能够与其他主机交换数据的设备。通常, 这个接口是一个硬件设备, 但是它也可能是一个纯粹的软件设备,比如环回接口。一个网络接口负责发送和接收数据报文,在内核网络子系统的驱动下,不必知道单个事务是如何映射到实际的被发送的报文上的。很多网络连接(特别那些使用 TCP 的)是面向流的,但是网络设备却常常设计成处理报文的发送和接收。一个网络驱动对单个连接一无所知，它只处理报文。

## 6、驱动程序加载到内核的方法

### 1)、直接编译进内核。

在Linux 内核中增加程序需要完成以下3 项工作：

a、将编写的源代码复制到Linux 内核源代码的相应目录。

b、在目录的Keonfig 文件中增加新源代码对应项目的编译配置选项。

c、在目录的Makefile 文件中增加对新源代码的编译条目。

### 2)、linux提供了模块Module 的机制

insmod 加载模块，将模块的代码和数据装入内核，然后使用内核的符号表解析模块中任何未解析的符号。

rmmod 卸载模块，注意，如果内核认为该模块任然在使用状态，或者内核被禁止移除该模块，则无法移除该模块。

## 7、字符驱动开发步骤

### 1)、基本步骤

a、确定主设备号和次设备号。

b、实现字符驱动程序：

实现file\_operations结构题，

实现初始化函数，注册字符设备，

实现销毁函数，释放字符设备，

c、创建设备文件节点。

### 2)、主次设备号

主设备号是内核识别一个设备的标识，是一个整数（占12bits），范围从0到4095，通常使用1-255。次设备号由内核使用，用于正确确定设备文件所指向的设备，也是一个整数（占20bits），范围从0到1048575，通常使用1-255。

设备编号的内部表达：dev\_t类型(32位)，用来保存设备编号其中高12位为主设备号，低20位为次设备号。

从dev\_t获得主次设备号：MAJOR(dev\_t)取得主设备号，MINOR(dev\_t) 取得次设备号。

分配主设备号：

a、手动分配

#include <linux/fs.h>

int register\_chrdev\_region(dev\_t first,unsigned int count,char \*name)

b、动态分配

#include <linux/fs.h>

int alloc\_chrdev\_region(dev\_t \*dev,unsigned int firstminor,unsigned int count,char \*name);

c、释放设备号

void unregister\_chrdev\_region(dev\_t first,unsigned int count);

### 3)、实现字符驱动

a、初始化cdev结构体

struct cdev

{

struct kobject kobj; //内嵌的kobject对象

struct module \*owner; //所属模块

struct file\_operations \*ops; //文件操作结构体

struct list\_head list;

dev\_t dev; //设备号

unsigned int count;

}

操作cdev的函数

初始化一个字符设备：

void cdev\_init( struct cdev \*, struc t file\_operations \*);

向操作系统添加一个字符设备

int cdev\_add(struct cdev \*, dev\_t, unsigned);

从操作系统删除一个字符设备

void cdev\_del(struct cdev \*);

b、初始化file\_operations结构体

根据硬件设备的特点，实现某些函数。应用程序调用open打开设备时，内核会将open映射到file\_operations里面的open函数指针处。主要成员有owner，

unlocked\_ioctl ，open，release，read，write 等。为了保证驱动的兼容性，按照C99的标准，以如下方式对file\_operations结构体进行初始化。

static struct file\_operations xx\_ps =

{

.owner = THIS\_MODULE,

.read = xx\_read,

.write =xx\_write,

.open =xx\_open,

.release =xx\_release,

.unlocked\_ioctl=xx\_ unlocked\_ioctl,

};

c、一般的用户程序调用驱动模块操作流程

insmod xx.ko 加载模块

mknod /dev/xx c major minor创建设备节点

chmod 777 xx\_app 使应用程序具有执行权限

./xx\_app /dev/xx 执行应用程序，调用模块

rmmod xx.kb 卸载模块

# 五、蜂鸣器驱动模块

## 1、硬件原理图

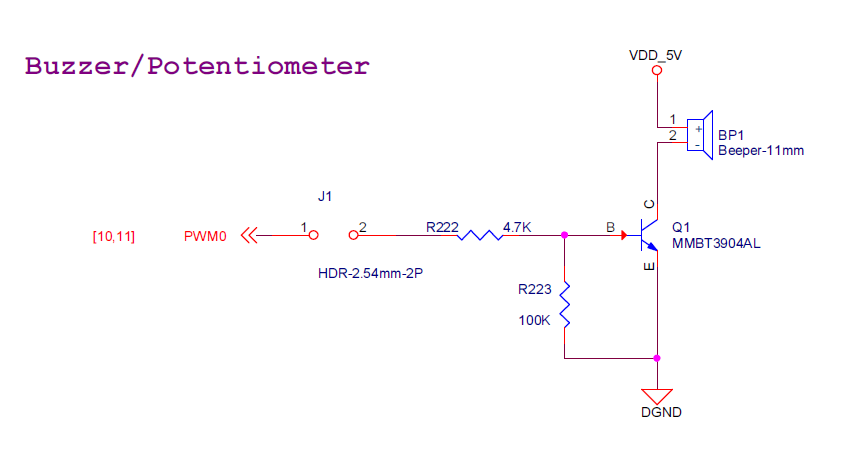


图14 底板电路图

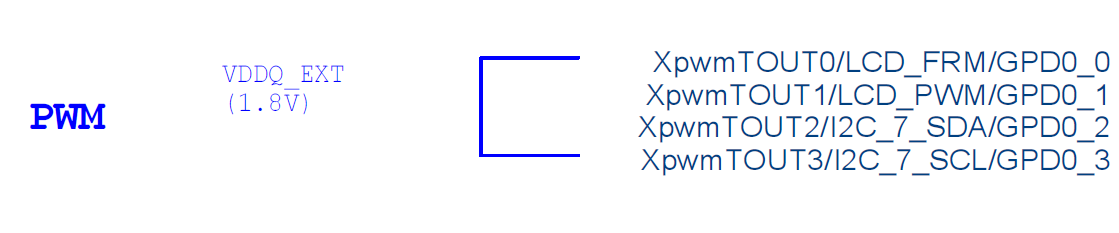
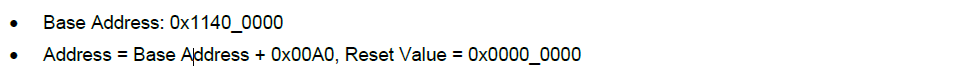


图 15 核心板电路图



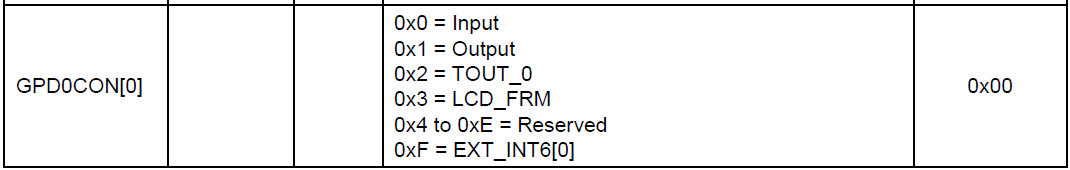


图 16 控制寄存器

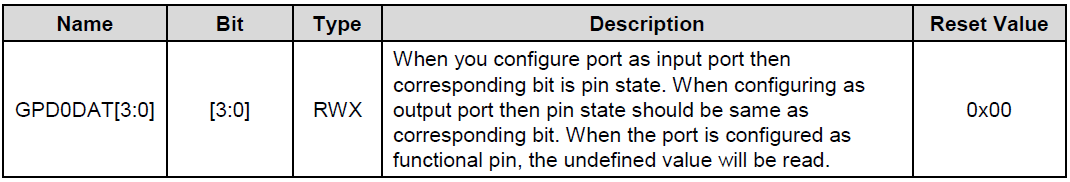


图 17 数据寄存器

经查看Tiny4412的底板原理图，可知，蜂鸣器的控制端口为PWM0，由电路连接图可知，只要向PWM0端口输出高电平，即可使蜂鸣器工作。再查找核心板可知，PWM0连接的是GPD0\_0接口。再通过查看Exynos 4412用户手册可知，GPD0\_0的控制寄存器的物理地址为0x114000A0, GPD0\_0的数据寄存器的物理地址为0x114000A04。

## 2、驱动模块流程图

## 3、应用程序流程图

## 4、驱动模块分析

### 1)、初始化函数分析

static int \_\_init hello\_init(void)

{

//alloc device number

alloc\_chrdev\_region(&beep\_no,0,1,"beep device");

//get major number from device number

beep\_major=MAJOR(beep\_no);

printk("beep major=%d\n",beep\_major);

//initial file\_operations

beep\_cdev.owner=THIS\_MODULE;

beep\_cdev.ops=&beep\_ops;

beep\_cdev.dev=beep\_no;

beep\_cdev.count=1;

//initial cdev

cdev\_init(&beep\_cdev,&beep\_ops);

//add a cdev

cdev\_add(&beep\_cdev,beep\_no,1);

printk("%s\n",\_\_FUNCTION\_\_);

return 0;

}

关键字static限定了该函数的作用域。\_\_init是内核中定义的一个宏，原型是#define \_\_init \_\_attribute\_\_ ((\_\_section\_\_(“.init.text”)))，表明该函数只是在初始化时使用。模块加载器在模块加载后会丢掉这个初始化函数, 这样可将该函数占用的内存释放出来，以作他用。注意，不要在初始化完成后还使用的函数(或者数据结构)上使用这两个标记。alloc\_chrdev\_region函数自动向内核申请一个设备号，MAJOR是内核头文件中定义的一个宏，用于从dev\_t型的设备号中取得主设备号。然后初始化cdev型结构体，并且向内核注册和添加cdev模块。

### 2)、退出函数分析

static void \_\_exit hello\_exit(void)

{

printk("%s\n",\_\_FUNCTION\_\_);

//release device number

unregister\_chrdev\_region(beep\_no,1);

//delete a cdev

cdev\_del(&beep\_cdev);

}

首先，\_\_exit类型和\_\_init类似，只能用于模块卸载函数。再通过unregister\_chrdev\_region函数和cdev\_del函数向系统注销设备号和卸载cdev字符设备。

### 3)、模块许可声明

MODULE\_LICENSE(“Dual BSD/GPL”);

在加载模块时，如果在模块中没有声明MODULE\_LICENSE协议，系统会提示：

[root@wbyq internship]#insmod key.ko

[ 7772.500000] key: **module license 'unspecified' taints kernel.**

[ 7772.500000] Disabling lock debugging due to kernel taint

### 4)、实现file\_operations结构体成员函数

static struct file\_operations beep\_ops=

{

.owner=THIS\_MODULE,

.open = beep\_open,

.release=beep\_release,

.read=beep\_read,

.write=beep\_write,

.unlocked\_ioctl=beep\_unlocked\_ioctl,

};

以下以beep\_read函数为例：

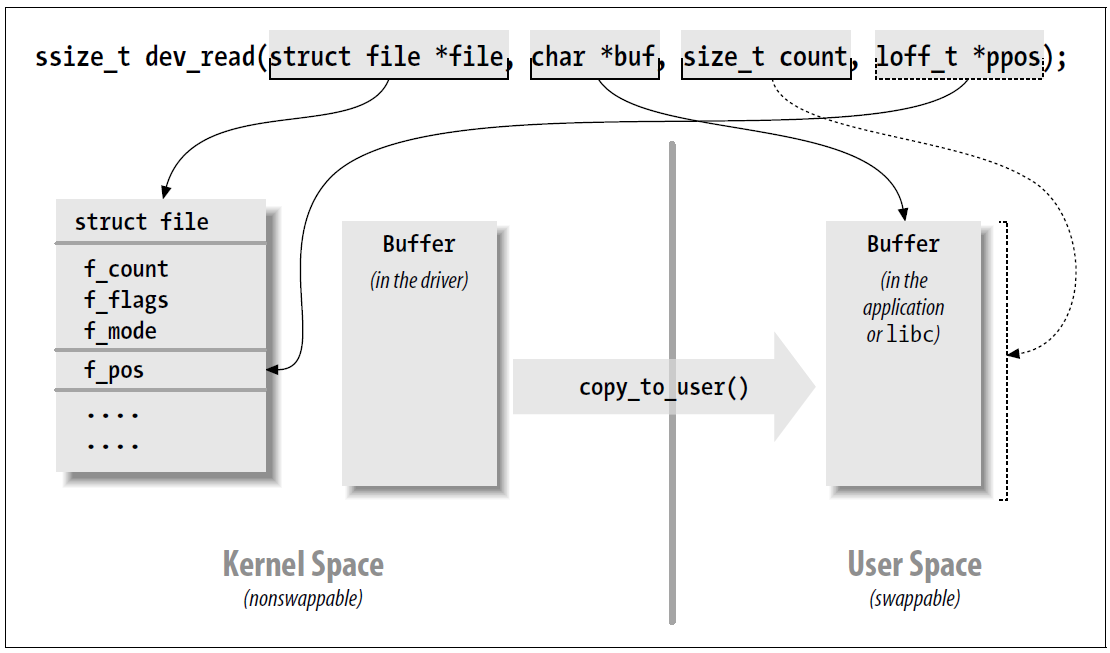


图 18

如上图所示，展示了内核空间和用户空间的数据交换过程。剩余分析见附录源码注释。

# 六、按键驱动模块

## 1、硬件原理图

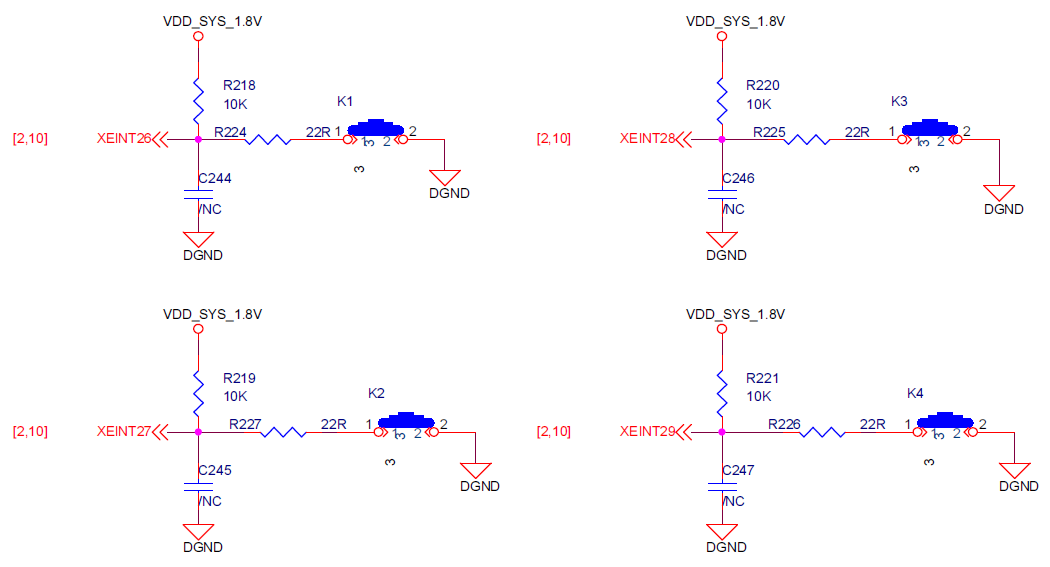


图 19

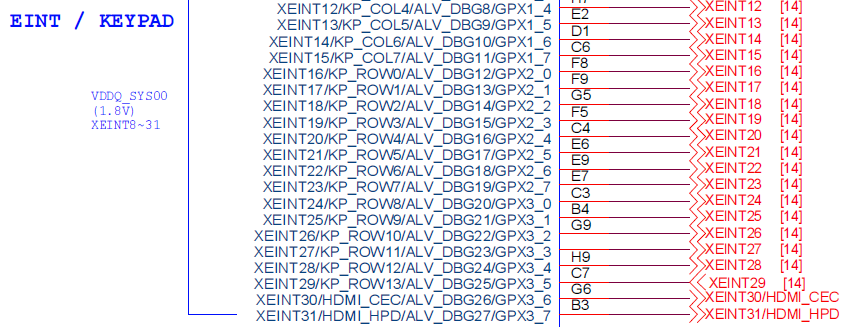
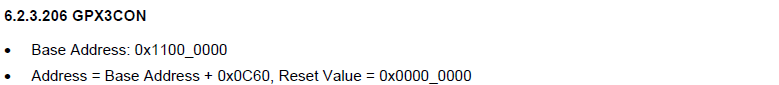
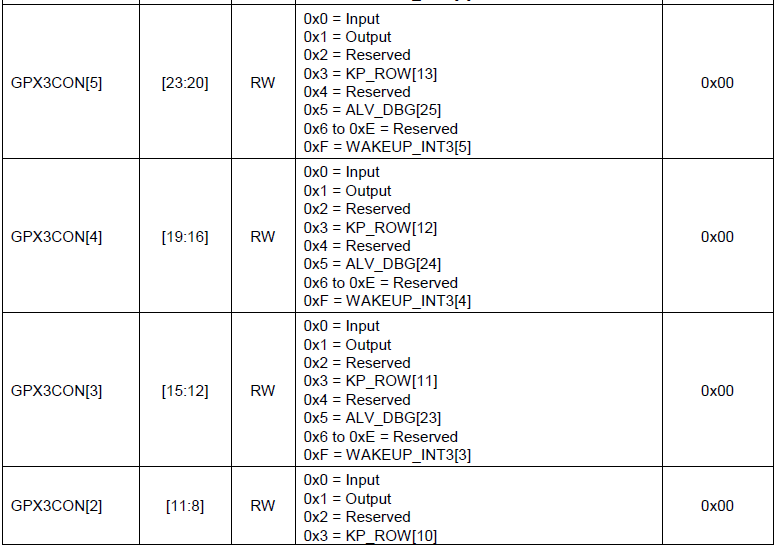


图20





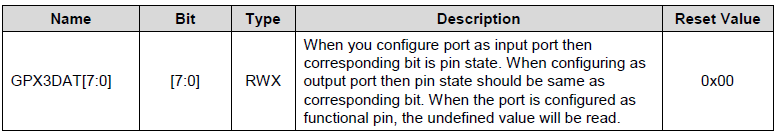


图21

## 2、驱动模块流程图

## 3、应用程序流程图

## 4、驱动模块分析

# 七、蜂鸣器模块与按键模块组合

# 八、总体设计

（1）设计思路

（2）设计步骤

# 九、小组总结

本次实习我们小组在老师的带领下学习了Linux字符驱动程序的基本框架。主要完成了对开发环境的搭建、字符设备驱动、蜂鸣器驱动模块、按键驱动模块以及蜂鸣器模块与按键模块的组合的学习，让我们认识了Linux字符驱动程序的设计，感受到了Linux的强大以及发展前景。

本次实习由于时间有限，只是对嵌入式有了一个大概的了解。还有许多问题还需我们进行进一步的了解和研究。特别是在实习中遇到过很多的问题，有些问题通过查资料或者在同学和老师的帮助下得以解决，也有些问题还是不太明白。也就是说，掌握的只是过于局限和片面，在遇到问题时不能快速准确的找到问题的根源。因此在今后的学习当中我们小组会努力学习，在Linux的世界当中遨游，争取做一名优秀的驱动工程师。

# 十、附录

程序代码