***2019***



**嵌入式系统 ·实验报告·**

j0242087[1]

|  |  |
| --- | --- |
| 专 业： | 计算机科学与技术 |
| 班 级： | ACM1601 |
| 学 号： | U201614756 |
| 姓 名： | 江易星 |
| 电 话： | 13995887596 |
| 邮 件： | 1093824239@qq.com |
| 完成日期： | 2019-06-27 |



1.核心编译，系统烧录，简单应用程序开发 2

1.1 实验要求 2

1.2 实验步骤 2

1.2.1 Kernel编译 2

1.2.2 系统烧录 2

1.2.3 简单linux应用开发 6

1.3 实验代码 6

2.Linux framebuffer界面显示开发：点、线、矩形区域显示 8

2.1 实验要求 8

2.2 实现思路 8

2.3 实验步骤 8

2.4 实验代码 9

3.图片显示，文本显示 11

3.1 实验要求 11

3.2 实现思路 11

3.3 实验步骤 11

3.4 实验代码 11

4.Linux多点触摸开发 13

4.1 实验要求 13

4.2 实现思路 13

4.3 实验步骤 13

4.4 实验代码 13

5.Linux LED 驱动和控制界面 17

5.1 实验要求 17

5.2 实现思路 17

5.3 实验步骤 17

5.4 实验代码 18

6总结与心得 26

# 1.核心编译，系统烧录，简单应用程序开发

## 1.1 实验要求

（1）学习并练习系统镜像的编译生成：Kernel编译。

（2）学习并练习Android+Linux系统烧录的方法：窗口工具cutecom的使用，控制uboot以及usb烧写工具fastboot使用。

（3）简单linux应用程序开发：使用Android NDK编译简单应用程序，使用usb开发工具adb上传并运行程序。

## 1.2 实验步骤

### 1.2.1 Kernel编译

Linux Kernel配置：使用make menuconfig命令，即可用图形化的界面配置内核。

需要编译Linux kernel，只需要在核心源码的根目录下执行 make zImage命令即可，执行成功后会生成内核镜像zImage。

### 1.2.2 系统烧录

准备工作：

1配置usb设备

在/etc/udev/rules.d下创建一个配置文件51-android,rules。

向文件中写入如下内容：

SUBSYSTEM = "usb",ATTR(idVendor) = "18d1",MODE = "0666",GROUP= "plugdev",OWNER="admin"~

其中ATTR(idVendor) 是厂商的ID号，可以通过Isusb命令查看。

OWNER填写当前登录的用户名。

如果文件不能访问，则可以通过sudo chmod a+r /etc/udev/rules.d命令给文件加上访问权限。

2连接开发板

讲开发板连接上电源，然后将usb线一端连接开发板一端连接PC，将串口一端连接开发板上坐起第二个，一端连接PC。将拨码开关设置为1：off，2：off，3：on，4：on。

3在终端下用cutecom打开串口，设置如图1.1所示：

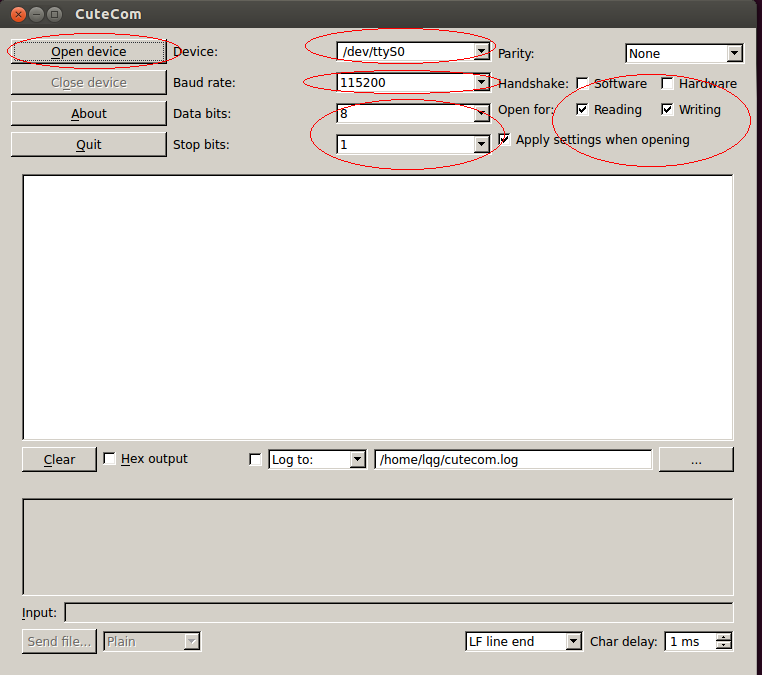


图1.1：cutecom设置

4 长按开发板上的开机键，注意PC上cutecom界面，在其进入自动引导系统之前按下任意按键进入uboot，如图1.2所示：

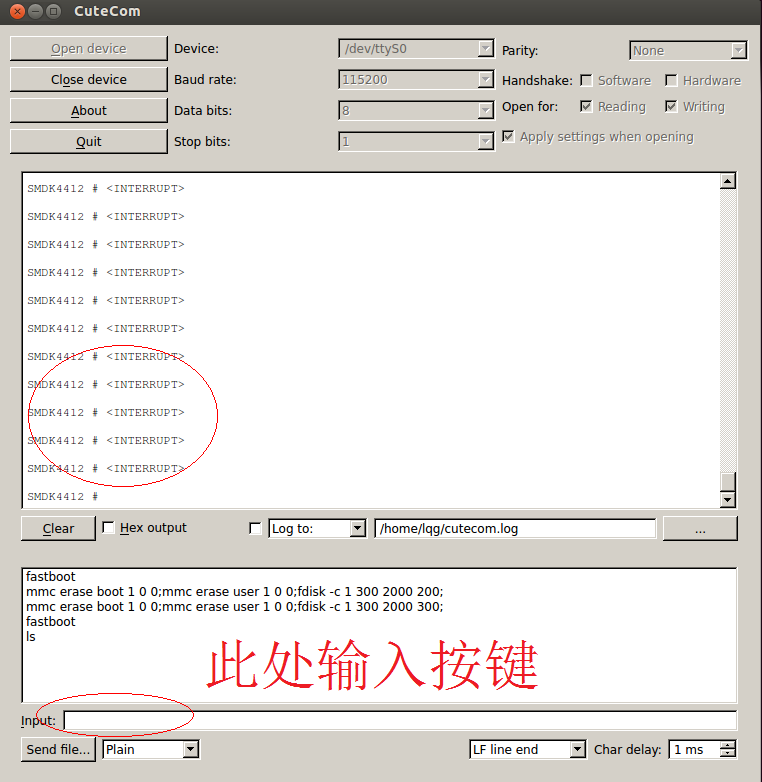


图1.2：进入自动引导之前按任意键

然后在uboot界面输入fastboot命令。如图1.3所示：

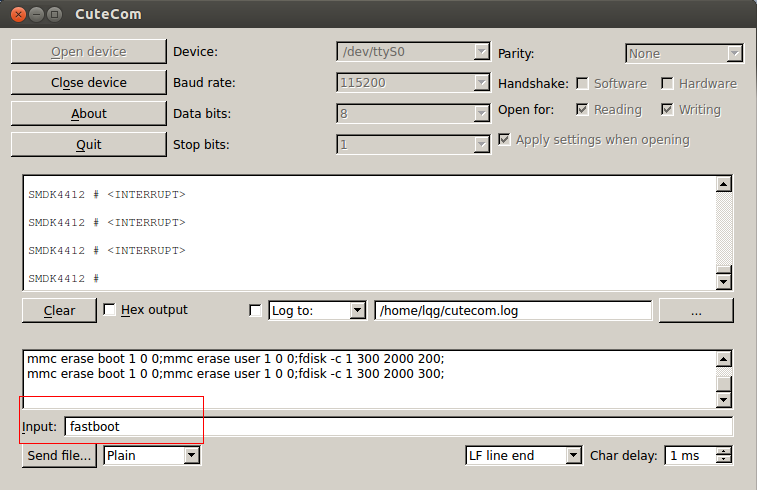


图1.3输入fastboot命令

系统烧写：

1烧写uboot：本实验平台uboot已经烧录好了，不需要学生完成。

2 烧写Linux kernel：使用命令 fastboot flash kernel 路径/zImage即可完成Linux kernel的烧录，cutecom界面显示如图1.4所示：

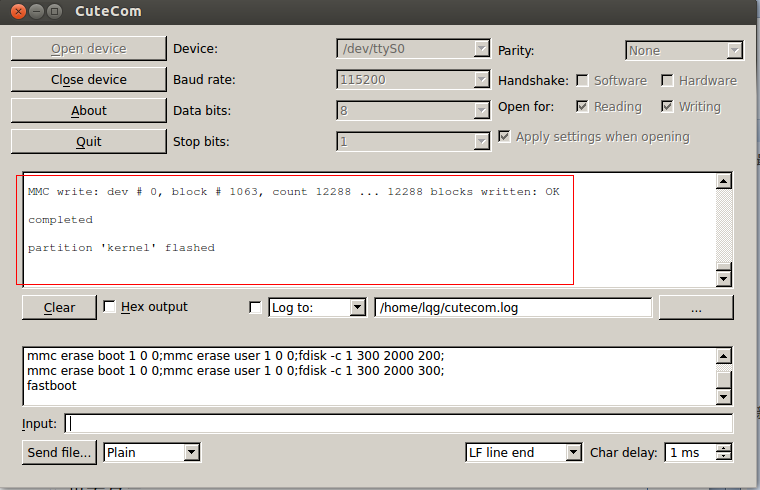


图1.4：烧写Lunix kernel

3烧写根文件系统：使用命令 fastboot flash ramdisk 路径/ramdisk-uboot.img即可完成根文件目录的烧写。对应的cutecom界面如图1.5所示：

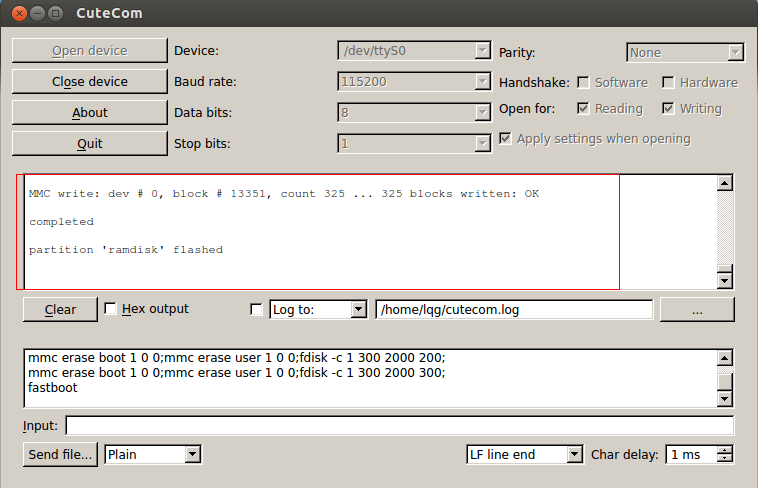


图1.5：烧写根文件目录

4烧写Android：使用命令 fastboot flash system 路径/system.img命令即可完成烧写，cutecom界面显示如图1.6所示：

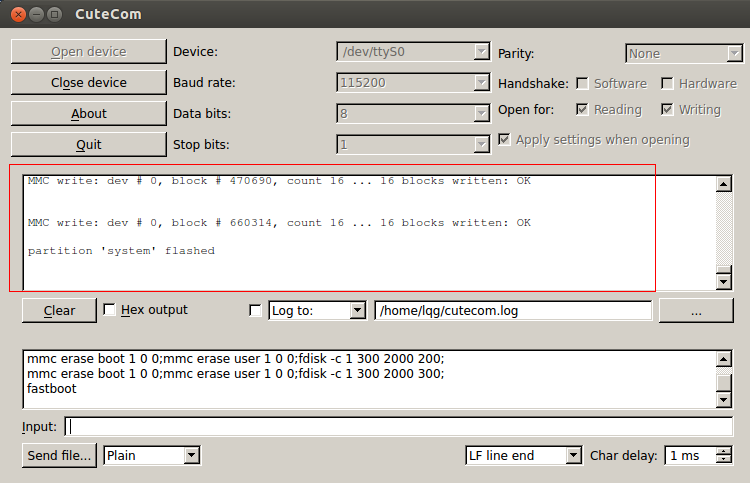


图1.6：烧写Android

烧写完毕之后重启开发板即可，不需要进行任何其他操作即可进入到Linux系统。

### 1.2.3 简单linux应用开发

1 确定common/rules.mk文件配置的编译器目录是否正确。要求DIR = "…/tools/android-ndk-r8c"。

2完成应用程序的代码，然后编译生成lab1.直接使用make命令即可。

3 将文件上传到开发板上的/data目录，使用如下命令：

adb push lab1 /data/local

adb shell chmod +x /data/local/lab1

4登录到开发板上运行,使用如下命令：  
 adb shell

cd /data/local

./lab1

或者直接运行程序：

adb shell /data/local/lab1

## 1.3 实验代码

linux应用开发的代码如下：

|  |
| --- |
| #include <stdio.h>  int main(int argc, char\* argv[])  {  printf("Hello eee linux!\n");  return 0;  } |

# 2.Linux framebuffer界面显示开发：点、线、矩形区域显示

## 2.1 实验要求

（1）理解Linux下的LCD显示驱动接口：framebuffer的使用原理。

（2）理解基本图形的显示原理：点、线以及矩形区域，并完成画矩形函数fb\_draw\_rect以及画线函数fb\_draw\_line。

（3）理解双缓冲机制。

（4）最终在开发板上绘制出所要求的图形，并且分别记录画

## 2.2 实现思路

画矩形函数fb\_draw\_rect的实现：

画矩形函数可以利用画像素点函数实现，只需要将矩形区域内所有的像素点都使用画像素点函数涂上相同的颜色即可。

画线函数fb\_draw\_line的实现：

画线函数需要考虑线的斜率，如果直线的斜率的绝对值小于1，则x每次增加1，计算对应的y的坐标，然后描出这个点，如果大于1，则按照y每次增加1，计算对应的x坐标，然后描出这个点。因为这样才能保证取点的密度更大，画出的线不会出现断开的情况。

在开发板上画出要求图形的实现：  
首先需要按照每隔两行、每隔五列，画一个像素点。然后按照每隔35行，每个25列画一个20\*20的正方形。然后以（200，100）（800，500）为起始的两个点，每次将（200，100）下移20像素点，（800，500）上移20像素点，旋转着画线，最后以（200，100）和（800，500）为起始的两个点，每次将（200，100）右移20个像素点，（800，500）左移20个像素点，旋转着画线。

## 2.3 实验步骤

1编写实验代码。

2按照实验一中方法，将代码烧录到开发板中并启动开发板查看显示图形，验证正确性。

3如果与预期实验现象不一致，则说明代码错误，修改代码，再次烧录验证，直到正确为止。

## 2.4 实验代码

|  |
| --- |
| #include <sys/mman.h>  #include <linux/fb.h>  #include <stdio.h>  #include "../common/common.h"  #define RED FB\_COLOR(255,0,0)  #define ORANGE FB\_COLOR(255,165,0)  #define YELLOW FB\_COLOR(255,255,0)  #define GREEN FB\_COLOR(0,255,0)  #define CYAN FB\_COLOR(0,127,255)  #define BLUE FB\_COLOR(0,0,255)  #define PURPLE FB\_COLOR(139,0,255)  #define WHITE FB\_COLOR(255,255,255)  #define BLACK FB\_COLOR(0,0,0)  int color[9] = {RED,ORANGE,YELLOW,GREEN,CYAN,BLUE,PURPLE,WHITE,BLACK};  int main(int argc, char\* argv[])  {  int row,column,i;  int32\_t start, end;  fb\_init("/dev/graphics/fb0");  fb\_draw\_rect(0,0,SCREEN\_WIDTH,SCREEN\_HEIGHT,BLACK);  fb\_update();  printf("\n========== Start Test ===========\n");  sleep(1);  start = fb\_get\_time();  for(row=-5; row<605; row+=2)  {  for(column=-5; column<1029; column+=2)  {  fb\_draw\_pixel(column,row,YELLOW);  fb\_update();  }  }  end = fb\_get\_time();  printf("draw pixel: %d ms\n",end - start);  sleep(1);  start = fb\_get\_time();  for(row=-35,i=0; row<635; row+=35)  {  for(column=-25; column<1050; column+=25)  {  fb\_draw\_rect(column,row,20,20,color[++i%9]);  fb\_update();  }  }  end = fb\_get\_time();  printf("draw rect: %d ms\n",end - start);  sleep(1);  fb\_draw\_rect(300,200,400,200,BLACK);  fb\_update();  start = fb\_get\_time();  for(row=0;row<=400;row+=20){  //200.100.800.500 500 300  fb\_draw\_line(500-300,300-200+row,500+300,300+200-row,color[1]);  fb\_update();  }  for(column=100;column<=500;column+=20){  //100 200 500 800  fb\_draw\_line(500-300+column,300-200,500+300-column,300+200,color[2]);  fb\_update();  }  end = fb\_get\_time();  printf("draw line: %d ms\n", end - start);  return 0;  } |

# 3.图片显示，文本显示

## 3.1 实验要求

（1）能够实现jpg格式的不透明图片的显示。

（2）能够实现png格式的半透明图片的显示，

（3）能够实现矢量字体的显示。

## 3.2 实现思路

fb\_image可以用来处理图片的显示以及矢量字体。使用fb\_read\_jpeg\_image可以读取jpg图片，使用fb\_read\_png\_image可以读取png图片，然后利用fb\_draw\_picture就可以显示图片，用font\_init可以设置字体，然后用fb\_draw\_text就可以将对应的文本用设置的字体显示。

## 3.3 实验步骤

1编写实验代码。

2按照实验一中方法，将代码烧录到开发板中并启动开发板查看显示图形，验证正确性。

3如果与预期实验现象不一致，则说明代码错误，修改代码，再次烧录验证，直到正确为止。

## 3.4 实验代码

|  |
| --- |
| #include <stdio.h>  #include "../common/common.h"  #define RED FB\_COLOR(255,0,0)  #define ORANGE FB\_COLOR(255,165,0)  #define YELLOW FB\_COLOR(255,255,0)  #define GREEN FB\_COLOR(0,255,0)  #define CYAN FB\_COLOR(0,127,255)  #define BLUE FB\_COLOR(0,0,255)  #define PURPLE FB\_COLOR(139,0,255)  #define WHITE FB\_COLOR(255,255,255)  #define BLACK FB\_COLOR(0,0,0)  int\* main(int argc, char \*argv[])  {  fb\_init("/dev/graphics/fb0");  font\_init("/data/local/font.ttc");  fb\_draw\_rect(0,0,SCREEN\_WIDTH,SCREEN\_HEIGHT,BLACK);  fb\_update();  fb\_image \*img;  img = fb\_read\_jpeg\_image("/data/local/jpg\_test.jpg");  fb\_draw\_image(0,0,img,0);  fb\_update();  fb\_free\_image(img);  img = fb\_read\_png\_image("/data/local/png\_test.png");  fb\_draw\_image(100,300,img,0);  fb\_update();  fb\_free\_image(img);  img = fb\_read\_font\_image("嵌",30,NULL);  fb\_draw\_image(400,350,img,RED);  fb\_update();  fb\_free\_image(img);  fb\_draw\_text(50,50,"床前明月光，疑是地上霜。",64,PURPLE);  fb\_draw\_text(50,120,"举头望明月，低头思故乡。",64,PURPLE);  fb\_update();  return 0;  } |

# 4.Linux多点触摸开发

## 4.1 实验要求

（1）能够在LCD屏幕上显示多点触摸轨迹，并用不同的颜色区分。

（2）绘制一个清除屏幕的按钮，点击后清除屏幕内容。

## 4.2 实现思路

将清除按钮设置在整个屏幕的左上角边长为50个像素的正方形内部，用黑色显示出来。

当手指按下时：判断按下的点是否在清除按钮内部，如果在，则将整个屏幕上所有的痕迹清除，具体做法是，先将整个屏幕全部画成一个黑色矩形，将背景图画上去，然后再把左上角的清除按钮画上去。如果不在清除按钮内部，则以按下的点为中心，画出一个边长为4的小正方形。

当手指移动时：由移动前的点（oldx，oldy）附近向移动后的点（x,y）附近画多条线，使得线条看上去比较清晰。

当手指拿开时，不做任何操作。

## 4.3 实验步骤

1编写实验代码。

2按照实验一中方法，将代码烧录到开发板中并启动开发板，用多个手指在屏幕上进行点击和划线，观察屏幕上的显示。

3如果与预期实验现象不一致，则说明代码错误，修改代码，再次烧录验证，直到正确为止。

## 4.4 实验代码

|  |
| --- |
| #include <stdio.h>  #include "../common/common.h"  /\*  int main(int argc, char \*argv[])  {  fb\_init("/dev/graphics/fb0");  fb\_draw\_rect(0,0,SCREEN\_WIDTH,SCREEN\_HEIGHT,FB\_COLOR(255,255,255));  fb\_update();  touch\_init("/dev/input/event3");  int type,x,y,finger,i;  while(1){  type = touch\_read(&x,&y,&finger);  switch(type){  case TOUCH\_PRESS:  printf("TOUCH\_PRESS：x=%d,y=%d,finger=%d\n",type,x,y,finger);  break;  case TOUCH\_MOVE:  printf("TOUCH\_MOVE：x=%d,y=%d,finger=%d\n",type,x,y,finger);  break;  case TOUCH\_RELEASE:  printf("TOUCH\_RELEASE：x=%d,y=%d,finger=%d\n",type,x,y,finger);  break;  default:  break;  }  }  return 0;    }  \*/  #define RED FB\_COLOR(255,0,0)  #define ORANGE FB\_COLOR(255,165,0)  #define YELLOW FB\_COLOR(255,255,0)  #define GREEN FB\_COLOR(0,255,0)  #define CYAN FB\_COLOR(0,127,255)  #define BLUE FB\_COLOR(0,0,255)  #define PURPLE FB\_COLOR(139,0,255)  #define WHITE FB\_COLOR(255,255,255)  #define BLACK FB\_COLOR(0,0,0)  int main(int argc, char \*argv[])  {  fb\_init("/dev/graphics/fb0");  fb\_draw\_rect(0,0,SCREEN\_WIDTH,SCREEN\_HEIGHT,FB\_COLOR(255,255,255));  fb\_image \*img;  img = fb\_read\_jpeg\_image("/data/local/test.jpg");  fb\_draw\_image(0,0,img,0);  fb\_update();  fb\_free\_image(img);  fb\_draw\_rect(0,0,50,50,BLACK);  fb\_update();  touch\_init("/dev/input/event3");  int type,x,y,finger,i,oldx[5],oldy[5];  int color[5] = {RED,GREEN,BLUE,YELLOW,BLACK};  //int color[5] = {BLACK, 0x00ff0000, 0x0000ff00, 0x000000ff, 0x00ffff00};  while(1){  type = touch\_read(&x,&y,&finger);  switch(type){  case TOUCH\_PRESS:  if(x<=50 && y<=50){//reset  fb\_draw\_rect(0,0,SCREEN\_WIDTH,SCREEN\_HEIGHT,FB\_COLOR(255,255,255));  fb\_image \*img;  img = fb\_read\_jpeg\_image("/data/local/test.jpg");  fb\_draw\_image(0,0,img,0);  fb\_update();  fb\_free\_image(img);  fb\_draw\_rect(0,0,50,50,BLACK);  }  else{  oldx[finger]=x;  oldy[finger]=y;  fb\_draw\_rect(x-2,y-2,4,4,color[finger]);//size of line  }      fb\_update();  //fb\_draw\_line(x,y,x+100,y+100,color[finger]);  printf("TOUCH\_PRESS：x=%d,y=%d,finger=%d, color=%d\n", x,y,finger,color[finger]);  break;  case TOUCH\_MOVE:  fb\_draw\_line(oldx[finger],oldy[finger],x,y,color[finger]);  fb\_draw\_line(oldx[finger],oldy[finger]+1,x,y+1,color[finger]);  fb\_draw\_line(oldx[finger],oldy[finger]-1,x,y-1,color[finger]);  fb\_draw\_line(oldx[finger],oldy[finger]-2,x,y-2,color[finger]);  fb\_draw\_line(oldx[finger],oldy[finger]+2,x,y+2,color[finger]);  fb\_draw\_line(oldx[finger]+1,oldy[finger],x+1,y,color[finger]);  fb\_draw\_line(oldx[finger]-1,oldy[finger],x-1,y,color[finger]);  fb\_draw\_line(oldx[finger]+2,oldy[finger],x+2,y,color[finger]);  fb\_draw\_line(oldx[finger]-2,oldy[finger],x-2,y,color[finger]);  oldx[finger]=x;  oldy[finger]=y;  fb\_update();    printf("TOUCH\_MOVE：x=%d,y=%d,finger=%d,color=%d\n", x,y,finger,color[finger]);  break;  case TOUCH\_RELEASE:  printf("TOUCH\_RELEASE：x=%d,y=%d,finger=%d\n", x,y,finger);  break;  default:  break;  }  }  return 0;    } |

# 5.Linux LED 驱动和控制界面

## 5.1 实验要求

（1）编写LED驱动：初始化、LED控制函数

（2）使用模块方式编译、安装驱动。

（3）编写测试程序，绘制界面并控制LED驱动。

## 5.2 实现思路

根据当前进度条的进度，来控制LED灯的闪烁频率。当进度条在最左端时，关闭LED，在最右端时LED灯常亮，当在中间时根据进度条进度控制LED灯频率即可。

## 5.3 实验步骤

编译、安装驱动：

编译：  
1修改lab5/driver目录下的makefile文件

确定交叉编译路径为：GCC\_DIR = "…/tools/arm-2009q3/bin"

确定linux核心代码路径为：KDIR = "…/image\_src/linux-3.0.15"

2在lab5/driver目录下执行make module即可得到驱动模块s3c-leds.ko

安装：

1在主机用adb push命令将驱动模块上传至实验板：adb push 路径/s3c-leds.ko /data/local

2用adb shell命令登陆实验板，执行安装模块命令：insmod /data/local/s3c-led.ko

3 登陆实验板 ，执行命令 :cat /proc/devices 查看led 驱动安装情况 ,查看led设备节点.

控制LED灯：

1编写实验代码。

2按照实验一中方法，将代码烧录到开发板中并启动开发板。

3操作控制界面，调整LED灯的频率，观察LED的闪烁速度是否随着调整改变，如果不改变则证明代码有错误。需要代码并重复上述步骤。

## 5.4 实验代码

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <errno.h>  #include <sys/types.h>  #include <sys/stat.h>  #include <fcntl.h>  #include <pthread.h>  #include <sys/ioctl.h>  #include "../common/common.h"  /\*=================================================================\*/  #define LED\_IOC\_MAGIC 'L'  #define LED\_ON \_IO(LED\_IOC\_MAGIC, 0)  #define LED\_OFF \_IO(LED\_IOC\_MAGIC, 1)  static int led\_fd = -1;  void led\_set(int on)  {  if(led\_fd == -1)  {  led\_fd = open("/dev/led", O\_RDWR);  if(led\_fd < 0){  printf("open /dev/led failed, errno = %d\n", errno);  return;  }  }  on? ioctl(led\_fd,LED\_ON) : ioctl(led\_fd,LED\_OFF);  return;  }  /\*=================================================================\*/  const int button\_width = 13;  const int button\_height = 42;  const int background\_width = SCREEN\_WIDTH;  const int background\_height = SCREEN\_HEIGHT;  const int progress\_shift\_x = 351;  const int progress\_shift\_y = 286;  const int progress\_width = 368;  const int progress\_height = 8;  const int tooltips\_width = 77;  const int tooltips\_height = 48;  const int tooltips\_text\_shift\_x = 12;  const int tooltips\_text\_shift\_y = 36;  const int tooltips\_text\_fontsize = 36;  const int tooltips\_text\_fontcolor = FB\_COLOR(255,255,255);  const int tooltips\_button\_margin = 3;  const int LED\_frequence\_UpperBound = 50;  const int LED\_frequence\_LowerBound = 0;  pthread\_mutex\_t freq\_lock = PTHREAD\_MUTEX\_INITIALIZER;  int freq = 0;  int button\_position\_x,button\_position\_y;  int tooltips\_position\_x,tooltips\_position\_y;  fb\_image \*bg,\*tooltips,\*progress\_content,\*button;  void reDrawPicture(int button\_center\_x) {  char text[32];  if (button\_center\_x <= progress\_shift\_x ){  button\_center\_x = progress\_shift\_x ;  }  if (button\_center\_x >= progress\_shift\_x + progress\_width) {  button\_center\_x = progress\_shift\_x + progress\_width ;  }  button\_position\_x = button\_center\_x - button\_width/2;  button\_position\_y = progress\_shift\_y + progress\_height / 2 - button\_height / 2;  tooltips\_position\_x = button\_center\_x - tooltips\_width /2;  tooltips\_position\_y = button\_position\_y - tooltips\_button\_margin - tooltips\_height;  bg = fb\_read\_jpeg\_image("/data/local/background.jpg");  tooltips = fb\_read\_png\_image("/data/local/tooltips.png");  progress\_content = fb\_read\_png\_image("/data/local/progress\_content.png");  button = fb\_read\_png\_image("/data/local/button.png");  //draw background;  fb\_draw\_image(0,0,bg,0);  //draw progress content;  progress\_content->pixel\_w = button\_center\_x - progress\_shift\_x;  fb\_draw\_image(progress\_shift\_x,progress\_shift\_y,progress\_content,0);  //draw slide button;  fb\_draw\_image(button\_position\_x,button\_position\_y,button,0);  //draw tooltips background;  fb\_draw\_image(tooltips\_position\_x,tooltips\_position\_y,tooltips,0);  //draw text;  sprintf(text,"%d%%",(button\_center\_x - progress\_shift\_x) \* 100 / (progress\_width));  fb\_draw\_text(tooltips\_position\_x + tooltips\_text\_shift\_x,tooltips\_position\_y + tooltips\_text\_shift\_y,  text,tooltips\_text\_fontsize,tooltips\_text\_fontcolor);    fb\_update();  }  int hit(int x,int y){  return (x >= progress\_shift\_x - button\_width / 2)  && (x <= progress\_shift\_x + progress\_width + button\_width/2)  && (y <= progress\_shift\_y + progress\_height / 2 + button\_height / 2)  && (y >= progress\_shift\_y + progress\_height / 2 - button\_height / 2);  }  int getPercentage(int button\_center\_x){  if (button\_center\_x <= progress\_shift\_x){  printf("Percentage = %d\n",0);  return 0;  }  else if (button\_center\_x >= progress\_shift\_x + progress\_width){  printf("Percentage = %d\n",100);  return 100;  }  else {  printf("Percentage = %d\n",(button\_center\_x - progress\_shift\_x)\*100 / progress\_width);  return (button\_center\_x - progress\_shift\_x)\*100 / progress\_width;  }    }  void \* flick(void \* ptr) {  static int freq\_local = 0;  while (1) {  if (pthread\_mutex\_trylock(&freq\_lock) == 0){  freq\_local = freq;  pthread\_mutex\_unlock(&freq\_lock);  printf("%d\n",freq\_local);  }  if (freq\_local < 0){  return NULL;  }  else if (freq\_local >= LED\_frequence\_UpperBound){  printf("setting LED on\n");  led\_set(1);  }  else if (freq\_local <= LED\_frequence\_LowerBound) {  printf("setting LED off\n");  led\_set(0);  }  else {  printf("setting LED at a period of %d us\n",1000000/ freq\_local);  led\_set(1);  usleep(1000000 / freq\_local);  led\_set(0);  usleep(1000000 / freq\_local);  }    }  }  void updateFreq(int percentage){  int freq\_new ;  if (percentage >= 0)  freq\_new = percentage \* (LED\_frequence\_UpperBound - LED\_frequence\_LowerBound) / 100 + LED\_frequence\_LowerBound;  else  freq\_new = -1;  pthread\_mutex\_lock(&freq\_lock);  freq = freq\_new;  pthread\_mutex\_unlock(&freq\_lock);  printf("setting Freq = %d\n",freq\_new);  }  int main(int argc, char\* argv[])  {  fb\_init("/dev/graphics/fb0");  font\_init("/data/local/font.ttc");  touch\_init("/dev/input/event3");  pthread\_t tid;  // pthread\_mutexattr\_t freq\_lock\_attr;  pthread\_create(&tid,NULL,flick,NULL);  pthread\_detach(tid);  pthread\_mutex\_init(&freq\_lock,NULL);  bg = fb\_read\_jpeg\_image("/data/local/background.jpg");  tooltips = fb\_read\_png\_image("/data/local/tooltips.png");  progress\_content = fb\_read\_png\_image("/data/local/progress\_content.png");  button = fb\_read\_png\_image("/data/local/button.png");  int type,x,y,finger,i;  int last\_x [5];  int last\_y [5];  int last\_finger = -1;  int k;  for (k = 0;k<5;k++){  last\_x[k] = last\_y[k] = -1;  }  reDrawPicture(progress\_shift\_x);  while(1){  type = touch\_read(&x,&y,&finger);  switch(type){  case TOUCH\_PRESS:  printf("TOUCH\_PRESSï¼šx=%d,y=%d,finger=%d\n",type,x,y,finger);  if (last\_finger != -1){  break;  }  if (!hit(x,y)){  break;  }  reDrawPicture(x);  updateFreq(getPercentage(x));  last\_finger = finger;  last\_x[finger] = x;  last\_y[finger] = y;  break;  case TOUCH\_MOVE:  printf("TOUCH\_MOVEï¼šx=%d,y=%d,finger=%d\n",type,x,y,finger);    if (last\_finger != finger){  break;  }  reDrawPicture(x);  updateFreq(getPercentage(x));  last\_x[finger] = x;  last\_y[finger] = y;      break;  case TOUCH\_RELEASE:  printf("TOUCH\_RELEASEï¼šx=%d,y=%d,finger=%d\n",type,x,y,finger);    if (last\_finger != finger){  break;  }    last\_x[finger] = -1;  last\_y[finger] = -1;  last\_finger = -1;    break;  default:  break;  }  }  fb\_free\_image (button);  fb\_free\_image (progress\_content);  fb\_free\_image (tooltips);  fb\_free\_image (bg);    // set freq to -1, exit flick thread;  updateFreq(-1);  pthread\_join(tid,NULL);  return 0 ;  } |

# 6总结与心得

六次实验总共完成了下列事情：

（1）练习了如何编译内核、如何烧录内核，以及如何开发linux应用并用adb上传并运行程序。

（2）了解了linux下LCD驱动接口framebuffer的原理，以及双缓冲机制。并动手实现了在LCD屏幕上画线、画矩形，并且消除屏幕闪烁，防止线条中断。

（3）练习了在LCD屏幕上显示jpg不透明图片、png半透明图片以及矢量字体的显示。

（4）学习了触摸屏驱动接口的使用以及多点触摸协议。并且实现了在LCD上显示多点触摸轨迹和清除的功能。

（5）实现了Linux LED驱动编程以及控制界面的实现。通过一个控制界面来控制LED灯的闪烁频率。

通过这些实验，初步的了解了嵌入式系统开发的内容。通过动手实践，加深了对理论课堂上讲解内容的理解，加强了编写代码的能力以及解决问题的能力。更加深刻的感受了linux系统在嵌入式中占有的重要地位。