華中科技大學

2021

嵌入式系统 实验报告

题　　目：	嵌入式系统实验报告
专　　业：	计算机科学与技术
班　　级：	CS1802
学　　号：	U201814531
姓　　名：	李响
电　　话：	18857802923
邮　　件：	676655103@qq.com

目 录

1 实验一 核心编译、系统烧录及简单应用程序开发

1.1 实验任务

1. 学习并完成系统镜像的编译生成：Kernel 编译、Uboot 编译以及 Android 系统编译（Uboot 与 Android 编译不要求）；

2. 学习并完成 Android+Linux 系统烧录：使用串口工具 cutecom 控制 uboot 烧写命令，并使用 usb 烧写工具 fastboot 传入镜像文件进行系统烧写；

3. 完成简单 linux 应用程序开发：使用 Android NDK 编译简单应用程序，并使用 usb 开发工具 adb 上传至平板中，进行运行与测试。

1.2 实验步骤

1.2.1 Kernel 编译

首先，使用 make menuconfig 命令，图形化配置内核；然后编译 Linux 内核，在核心源码的根目录下执行 make zImage 命令，生成内核镜像 zImage。

1.2.2 系统烧录

1. 配置 usb 设备：

在/etc/udev/rules.d 创建配置文件 51-android.rules，然后写入如下内容：

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

其中 SUBSYSTEM 表示是 USB 设备，ATTR(idVendor)为厂商的 ID 号，可以使用 lsusb 命令查看，MODE 为访问权限，GROUP 设置成“即插即用”，OWNER 指定有权限操作的用户。如果配置文件不能访问，则可以使用命令 sudo chmod a+r /etc/udev/rules.d/51-android.rules 给文件加上访问权限。

在用户根目录的.android 文件夹下创建 adb_usb.ini 文件，将厂商 ID“0x18d1”

写入文件，保存并退出。

2. 连接开发板

首先，将开发板连接上电源；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。

在终端下用 cutecom 打开串口，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；然后，在 uboot 界面输入 fastboot 命令，开发板连接完成，等待系统烧写。

3. 系统烧写：

本实验平台 uboot 已经烧录好了，不需要进行。

首先，烧写 Linux kernel，使用命令 fastboot flash kernel 路径/zImage 即可完成 Linux kernel 的烧录；然后，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；最后，烧写 Android，使用命令 fastboot flash system 路径/system.img 命令即可完成烧写。

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；
2. 然后，完成应用程序的代码，然后使用 make 命令编译生成 lab1；
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 embedded linux!\n");
    return 0;
}
```

2 实验二 Linux framebuffer 显示界面开发

2.1 实验任务

1. 理解 Linux 下的 LCD 显示驱动接口 framebuffer 的使用原理；

2. 理解双缓冲机制；

3. 理解基本图形的显示原理：点、线以及矩形区域绘制方式，并完成画矩形函数 fb_draw_rect 以及画线函数 fb_draw_line；

4. 在开发板上绘制出所要求的图形，并且记录画图时间。

2.2 函数设计与实现

2.2.1 画线函数 fb_draw_line 设计与实现

画线函数 fb_draw_line 的输入为(int x1, int y1, int x2, int y2, int color)，前 4 个参数为直线的两个端点的坐标，第 5 个参数为线的颜色。

为使得所画的直线尽可能光滑，画线函数需要考虑线的斜率，当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；当直线的斜率的绝对值大于 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。

2.2.2 画矩形函数 fb_draw_rect 设计与实现

画矩形函数可以利用画点函数实现，首先检查参数是否超出屏幕范围，防止出现段错误，然后将对应位置的像素使用画点函数填充即可。

2.3 实验步骤

1. 按照函数设计的思路，编写 common 文件夹下的 graphic.c 的两个函数，画矩形函数 fb_draw_rect 以及画线函数 fb_draw_line；

2. 使用如下命令编译文件，并将文件上传到开发板上的/data 目录

```
make
adb push lab2 /data/local
```

adb shell chmod +x /data/local/lab2

3. 使用如下命令登录到开发板上运行：

adb shell

cd /data/local

./lab2

或者直接运行程序：

adb shell /data/local/lab2

4. 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 测试结果

当函数编写不存在问题时，出现如图 2.1 所示现象，画点时间为 9ms，画矩形时间为 4ms，画线时间为 2ms：

图 2.1 Linux framebuffer 显示界面开发测试结果图

2.5 实验代码

```
void fb_draw_rect(int x, int y, int w, int h, int color)
{
    if(x < 0) { w += x; x = 0;}
    if(x+w > SCREEN_WIDTH) { w = SCREEN_WIDTH-x;}
    if(y < 0) { h += y; y = 0;}
    if(y+h >SCREEN_HEIGHT) { h = SCREEN_HEIGHT-y;}
    if(w<=0 || h<=0) return;
    int *buf = _begin_draw(x,y,w,h);
/*-------------------------------------------------*/
     int i,j;
     for(i = 0; i < h; i++)
          for(j = 0; j < w; j++)
               *(buf + (i+y)*SCREEN_WIDTH + (j+x)) = color;
/*-------------------------------------------------*/
    return;
}

void fb_draw_line(int x1, int y1, int x2, int y2, int color)
{
/*-------------------------------------------------*/
    int dx, dy, length, i;
    float xincre,yincre,x,y;
    if(x1<0 || x2<0 || y1<0 || y2<0) return;
    dx = x1-x2>0 ? x1-x2 : x2-x1;
    dy = y1-y2>0 ? y1-y2 : y2-y1;
    length = dx>dy ? dx : dy;

    xincre=(float)(x1-x2)/(float)length;
    yincre=(float)(y1-y2)/(float)length;
    x = x2;
    y = y2;
    for(i=1;i<=length;i++){
        fb_draw_pixel((int)x,(int)y,color);
        x+=xincre;
        y+=yincre;
    }
/*-------------------------------------------------*/
    return;
}
```

3 实验三 图片显示与文本显示

3.1 实验任务

1. 实现 jpg 格式不透明图片的显示；
2. 实现 png 格式半透明图片的显示；
3. 实现矢量字体显示，需要完成字模的提取以及字模的显示。

3.2 函数设计与实现

实现 jpg 格式不透明图片、png 格式半透明图片以及矢量字体的显示，需要编写函数 fb_draw_image 对不同类型的图片进行显示。

当图片的像素颜色类型为“FB_COLOR_RGB_8880”时，表示显示的图片类型为 jpg 图片。显示 jpg 文件非常简单，由于不需要处理透明度，因此只需要使用 memcpy 函数直接将图片数据复制到缓冲区即可。

当图片的像素颜色类型为“FB_COLOR_RGBA_8888”时，表示显示的图片类型为 png 图片。显示 png 图片需要考虑透明度，较为复杂，透明图片中的像素内容从高位到低位分别为 alpha、R1、G1 以及 B1，假设将缓冲区 framebuffer 的目标地址设置为 p，对于要绘制的每一个像素，使用如下方式对像素进行修正：

```
p[0] += (((B1 - p[0]) * alpha) >> 8);
p[1] += (((G1 - p[1]) * alpha) >> 8);
p[2] += (((R1 - p[2]) * alpha) >> 8);
```

当图片的像素颜色类型为“FB_COLOR_ALPHA_8”时，表示显示的图片类型为矢量字体。由于矢量字体只包含透明度，颜色根据函数的参数设置，因此与处理 png 图片的方式类似，只不过需要将目标的颜色使用函数参数进行修改即可。

3.3 实验步骤

1. 编写 common 文件夹下的 graphic.c 的画图函数 fb_draw_image；
2. 使用如下命令，编译文件 lab3 以及 test 两个测试文件，并将文件上传到开发板上的/data 目录，进行测试分析：

```
adb push lab3 /data/local        adb push test /data/local
```

adb shell chmod +x /data/local/lab3

adb shell chmod +x /data/local/test

adb shell /data/local/lab3　　adb shell /data/local/test

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 测试结果

当函数编写不存在问题时，出现如图 3.1 所示现象，lab3 测试时，屏幕显示 jpg 类型的羊驼图片、png 类型的草团图片以文本可以相互叠加；test 测试时，屏幕显示不同类别图片的显示时间，画点时间为 9ms，画矩形时间为 28ms，画线时间为 34ms，画图像时间为 146ms，画文本时间为 41ms，合计 258ms。

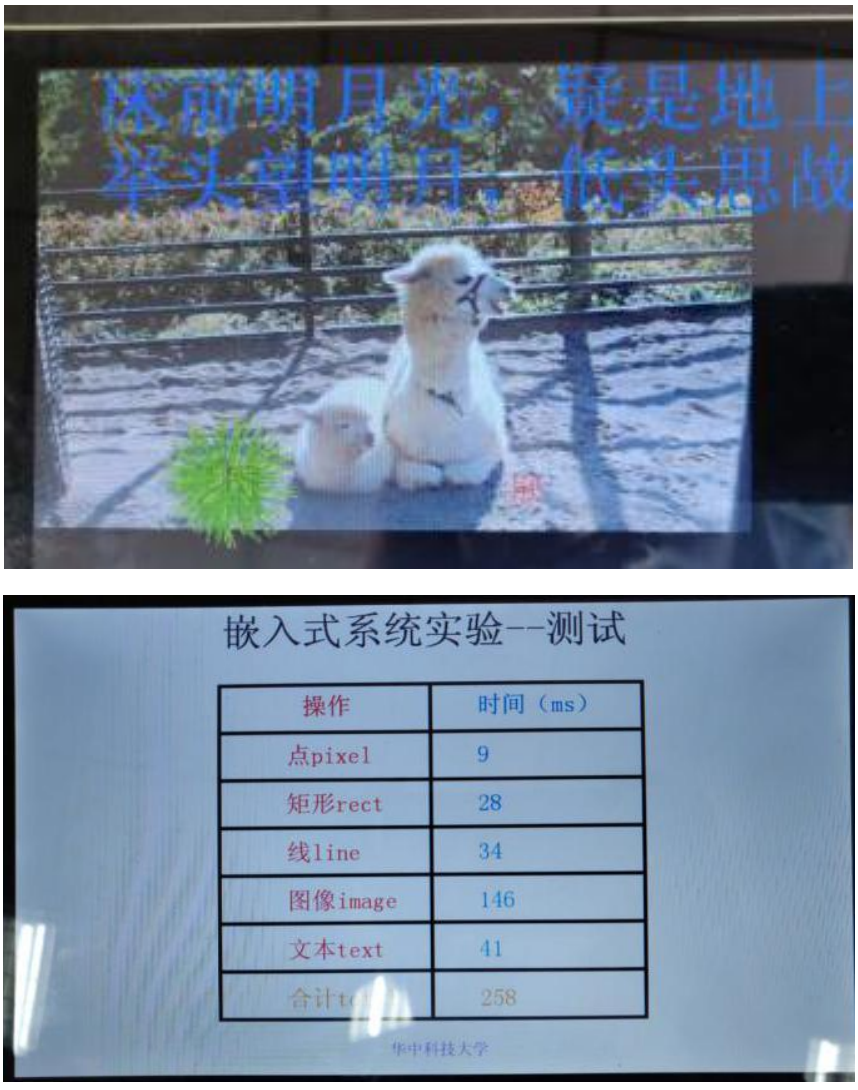

图 3.1 图片显示与文本显示测试结果图

3.5 实验代码

```
 /*lab3: jpg*/
 if(image->color_type == FB_COLOR_RGB_8880)
{
      /*  不需要处理透明度只需要将对应的点加载到缓冲区即可  */
      for(i = 0; i < h; i++)
      memcpy(dst i*SCREEN_WIDTH*4,src+i*image->line_byte,w*4);
      return;
}

 /*lab3: png*/
if(image->color_type == FB_COLOR_RGBA_8888) {
      for (i = 0 ; i < h ; ++i)
            for (j = 0 ; j < w ; ++j)
            {
                  char * pcolor = (src + i*image->line_byte + j*4);
                  char * p = (char*)((int*)dst+i*SCREEN_WIDTH+j);
                  switch (pcolor[3])
                  {
                        case 0: break;
                        case 255: p[0] = pcolor[0];
                                    p[1] = pcolor[1];
                                    p[2] = pcolor[2]; break;
                        default:
                              p[0] += (((pcolor[0] - p[0]) *pcolor[3])>>8 );
                              p[1] += (((pcolor[1] - p[1]) *pcolor[3])>>8 );
                              p[2] += (((pcolor[2] - p[2]) *pcolor[3])>>8 );
                              break;
                  }
            }
     return;
}

/*lab3: font*/
if(image->color_type == FB_COLOR_ALPHA_8)
{
      for(i = 0; i < h; i++)
      {
          char* s=src;
          char* d=dst;
          for(j = 0; j < w; j++)
```

```
                {
                    switch (s[0])
                    {
                        case 0: break;
                        case 255:
                          d[2] = (color & 0xff0000) >> 16;
                          d[1] = (color & 0xff00) >> 8;
                          d[0] = (color & 0xff);
                          break;
                        default:
                            d[2]+=(((((color&0xff0000)>>16)-d[2])*s[0])>>8);
                            d[1]+=(((((color & 0xff00) >> 8) - d[1])*s[0])>>8);
                            d[0]+=(((((color & 0xff)) - d[0])*s[0])>>8);
                            break;
                    }
                    s++; d+=4;
                }
                dst += SCREEN_WIDTH*4;
                src += image->line_byte;
            }
        return;
    }
/*-------------------------------------------------------------*/
    return;
}
```

4 Linux TouchScreen 多点触摸开发

4.1 实验任务

1. 了解 Linux 下的触摸屏驱动接口，学习 Input event 的使用，了解多点触摸协议（Multi-touch Protocol）的工作流程；

2. 使用函数获取多点触摸的坐标，并在 LCD 上显示多点触摸轨迹；

3. 完成基于多点触摸协议的应用程序，选择屏幕绘图或者触点追踪进行应用开发，具体的应用要求如下：

（1）**屏幕绘图**：使用多个手指在屏幕绘制曲线，要求每个手指轨迹的颜色不同；轨迹是连贯的，不能断断续续；轨迹要比较粗，线宽不能是 1 个点；绘制一个清除屏幕的按钮，点击后清除屏幕内容。

（2）**触点追踪**：实时跟踪触点，只显示当前位置，要求对应每个手指触点的圆的颜色不同；之前位置的圆要清除掉；屏幕不能明显闪烁。

4.2 程序设计与实现

本次实验选择**屏幕绘图**进行开发，本次程序开发主要需要编写 touch_event_cb 触摸事件处理函数，触摸事件主要分为 4 类，TOUCH_PRESS 表示首次触摸，TOUCH_MOVE 表示手指移动，TOUCH_RELEASE 表示手指离开，TOUCH_ERROR 表示触摸异常。

为实现直线的绘制，需要保存不同手指的历史位置，由于多点触摸协议最多同时处理 5 根手指的触摸检测，因此使用两个长度为 5 的数组分别保存手指上一次触摸的的 x 与 y 坐标值。

首先，在 main 函数中对缓冲区进行初始化，将屏幕的背景置为白色，然后再屏幕的左上角使用画矩形函数 fb_draw_rect 绘制一个黑色的清除按钮。

当产生 TOUCH_PRESS 事件时，表示当前手指首次接触屏幕，只需要将当前手指的坐标值保存到数组中即可；当手指接触到清除按钮所在的区域时，需要将屏幕清除，过程类似于初始化过程，将背景置为白色，并重新绘制清除按钮。

当产生 TOUCH_MOVE 事件时，表示当前手指移动，需要根据手指坐标数组

保存的数值以及当前手指所在的坐标数值绘制直线，由于要求绘制的直线直径超过1，因此需要绘制至少 4 条直线以扩展曲线的直径。

产生 TOUCH_RELEASE 事件时，不需要处理，只需要返回即可。

产生 TOUCH_ERROR 事件时，系统异常，需要关闭设备，并退出当前程序。

4.3 实验步骤

1. 按照程序设计思路编写 lab4 文件夹下的 main.c 文件；

2. 使用如下命令，编译文件 lab4 生成可执行程序文件，并将文件上传到开发板上的/data 目录，进行测试分析：

```
adb push lab4 /data/local
adb shell chmod +x /data/local/lab4
adb shell /data/local/lab4
```

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 测试结果

当函数编写不存在问题时，出现如**错误！未定义书签。**所示现象，使用多个手指在屏幕进行曲线绘制，可以发现每个手指的颜色不同，且可以同步绘制。

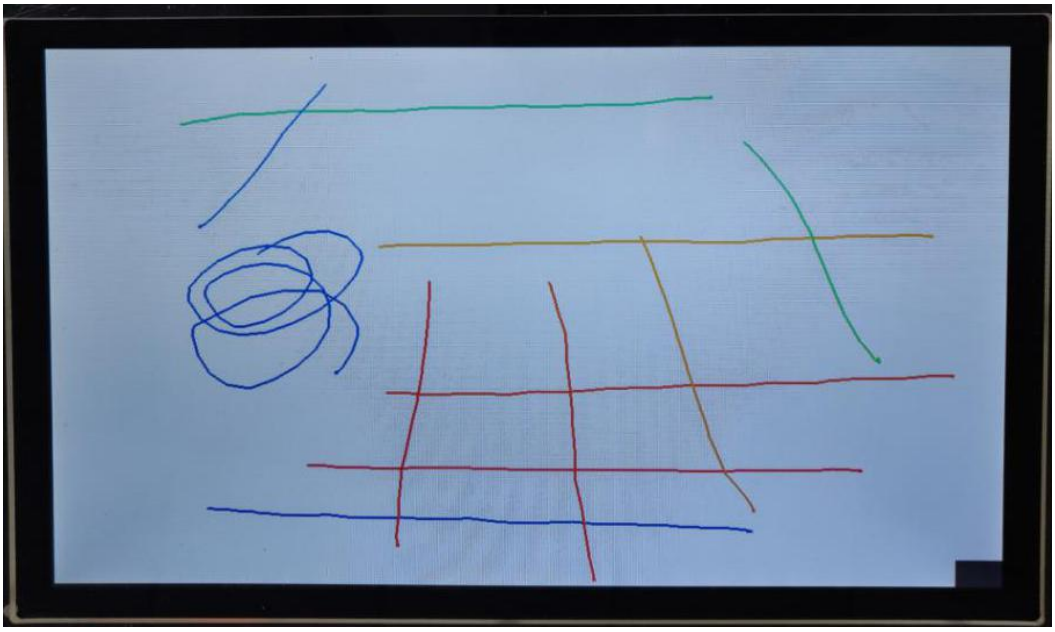

图 4.1 多点触摸开发测试结果图

4.5 实验代码

```
#include <stdio.h>
#include "../common/common.h"
#define BUTTON_H 30
#define BUTTON_W 50
#define COLOR_BACKGROUND      FB_COLOR(0xff,0xff,0xff)
int last_x [5];
int last_y [5];
int color[5] =
{
      FB_COLOR(65,105,225),     //Blue
      FB_COLOR(0,255,127),      //Spring Green
      FB_COLOR(255,215,0),      //Gold
      FB_COLOR(255,140,0),      //Dark Orange
      FB_COLOR(255,69,0)         //Orange Red
};
static int touch_fd;
static void touch_event_cb(int fd)
{
     int type,x,y,finger,i;
     type = touch_read(fd, &x,&y,&finger);
     switch(type)
    {
     case TOUCH_PRESS:
          /* 点击清除按钮 */
           if(x<=BUTTON_W && y<=BUTTON_H)
           {
                fb_draw_rect(0,0,SCREEN_WIDTH,SCREEN_HEIGHT
                              ,FB_COLOR(255,255,255));
                fb_draw_rect(0,0,BUTTON_W,BUTTON_H,FB_COLOR(0,0,0));
                fb_draw_text(10,10,"清除 CLEAR",25,FB_COLOR(255,255,255));
                fb_update();
                break;
           }
           /* 记录之前手指位置信息，并输出点，宽度为 4 */
           last_x[finger]=x;
           last_y[finger]=y;
           fb_draw_rect(x-2,y-2,4,4,color[finger]);
           fb_update();
           break;
```

```
        case TOUCH_MOVE:
            /* 并输出线，记录手指位置信息，线宽度为 4 */
            fb_draw_line(last_x[finger],last_y[finger],x,y,color[finger]);
            for(i = 1; i < 2; i++)
            {
                fb_draw_line(last_x[finger],last_y[finger]+i,x,y+i,color[finger]);
                fb_draw_line(last_x[finger],last_y[finger]-i,x,y-i,color[finger]);
                fb_draw_line(last_x[finger]+i,last_y[finger],x+i,y,color[finger]);
                fb_draw_line(last_x[finger]-i,last_y[finger],x-i,y,color[finger]);
            }
            last_x[finger]=x;   last_y[finger]=y;
            fb_update();
            break;
        case TOUCH_RELEASE:
            printf("TOUCH_RELEASE：x=%d,y=%d,finger=%d\n",x,y,finger);
            break;
        case TOUCH_ERROR:
            printf("close touch fd\n");
            close(fd); task_delete_file(fd); break;
        default: return;
        }
        fb_update();
        return;
}

int main(int argc, char *argv[])
{
        fb_init("/dev/graphics/fb0");
        fb_draw_rect(0,0,SCREEN_WIDTH,SCREEN_HEIGHT
                        ,COLOR_BACKGROUND);
        /* 绘制清除按钮 */
        fb_draw_rect(0,0,BUTTON_W,BUTTON_H,FB_COLOR(0,0,0));
        fb_draw_text(10,10,"清除 CLEAR",25,FB_COLOR(255,255,255));
        fb_update();

        //打开多点触摸设备文件，返回文件 fd
        touch_fd = touch_init("/dev/input/event3");
        //添加任务，当 touch_fd 文件可读时，会自动调用 touch_event_cb 函数
        task_add_file(touch_fd, touch_event_cb);
        task_loop(); //进入任务循环
        return 0;
}
```

5 实验五 蓝牙无线互联通讯

5.1 实验任务

1. 了解并学习蓝牙无线技术，使用命令正确配置并启动蓝牙服务；
2. 学习并熟练使用蓝牙的常用命令工具如 hciconfig、hcitool 以及 dptool 等；
3. 通过 RFCOMM 协议（蓝牙串行通讯）实现无线通讯，测试通过 lab5 实验代码；

5.2 实验步骤

1. 将 lab5 文件夹中的 etc 文件夹以及批处理文件 start_bt.sh 复制到开发板中对应的目录下，然后使用 source 命令执行 start_bt.sh 程序，即可启动蓝牙服务。
2. 在开发板上输入如下命令启动蓝牙服务：

```
hciconfig hci0 piscan
rfcomm -r watch  0  1
```

3. 在手机上安装蓝牙串口软件，搜索蓝牙设备进行连接；
4. 使用如下命令，将 lab5 测试程序编译，并将文件上传到开发板上的/data 目录，进行测试分析：

```
adb push lab5 /data/local
adb shell chmod +x /data/local/lab5
adb shell /data/local/lab5
```

5.3 测试结果

当蓝牙成功连接之后，执行 lab5 测试程序就可以在手机与开发板之间相互传递数据，当触摸开发板上的“发送”按钮，开发板会向手机发送“hello”字样，手机输入文本并发送后，开发板会显示该文本。

6 实验六 综合实验

6.1 实验任务

综合运用所学内容，设计并实现一个在多个开发板之间进行蓝牙互联协作的程序或一个功能比较复杂的单机程序，可参考功能如下：

1. 共享白板：两个开发板之间共享屏幕手绘内容；

2. 共享文件：显示远程开发板上的目录文件列表，选中指定文件，显示指定文件内容（文本和 png/jpg 图片）；

3. 联网游戏：如五子棋等小型游戏

4. 单机程序：

（1）视频播放器；

（2）图片浏览器：图片放大、缩小、图片尺寸超过显示区域时手指拖动显示；

（3）游戏或其他应用程序；

6.2 程序设计与实现

本次综合实验选择**共享白板功能**进行开发，使用蓝牙互联协作的方式实现多个开发板的协作，完成共享屏幕的功能。

为简化实验设计，本次实验将共享屏幕的手绘内容，其效果类似于实验 4 中的手绘效果。本次实验需要处理两方面的内容，第一方面是 touch_event_cb 多点触摸事件的处理，第二方面是 bluetooth_tty_event_cb 蓝牙通讯事件的处理。

多点触发事件的处理与实验 4 类似，主要编写 touch_event_cb 触摸事件处理函数，同样需要对 4 类主要的触摸事件进行处理，处理方式也与实验 4 基本相同。

除此之外，为了实现蓝牙通讯，除了在本机端进行手绘图像的显示，还需要使用蓝牙缓冲写入函数 myWrite_nonblock 将信息传输到远端，本次需要传输的信息比较简单，只需要将当前手指的坐标，触摸事件的类型以及手指颜色传输即可。

蓝牙通讯事件的处理过程与绘制图像过程类似，只需要将传输的数据使用蓝牙缓冲读取函数 myRead_nonblock 将数据读出，使用绘制曲线相同的方式绘制曲线。

6.3 实验步骤

1. 按照程序设计思路程序；
2. 编译文件生成可执行程序文件，并将文件上传到开发板上的/data 目录；
3. 使用实验 5 启动蓝牙服务的方式启动蓝牙服务，并启动测试程序；
4. 测试程序并记录结果。

6.4 测试结果

当程序编写不存在问题时，出现如**错误！未定义书签。**所示现象，使用多个手指在屏幕进行曲线绘制，可以发现每个手指的颜色不同，且可以在不同平板间同步显示。

```
lx@ubuntu:~/ES/lab_src_st/lab6$ adb shell
uid=0 gid=0@android:/ # cd data/local
uid=0 gid=0@android:/data/local # source start_bt.sh
already has bdaddr_mac
[1] 1383
 RDA power init sucefull-----
 RDA power init RDA_get_chipid-----
 RDA power init rdabt_read_bdaddr-----
####RDA_get_chipid(); ret=13, data_to_rec[] = {  04 0e 0a 01 09 10 00 76 58 11 22 2d ae}
##is 5876 or 5875c chip ##
####RDABT_core_Intialization(), chip_type=x02 ####!###  rp_write_mac_addr  ####
###[ae:2d:85:19:94:27]####
###[ae:2d:85:19:94:27]###
###  rp_write_mac_addr  ####
###[ae:2d:85:19:94:27]####
###[ae:2d:85:19:94:27]###
Device setup complete
[2] 1400
Serial Port service registered
uid=0 gid=0@android:/data/local # hciconfig hci0 piscan
uid=0 gid=0@android:/data/local # rfcomm -r connect  0 ae:2d:44:92:25:67  1
Connected /dev/rfcomm0 to AE:2D:44:92:25:67 on channel 1
Press CTRL-C for hangup
```

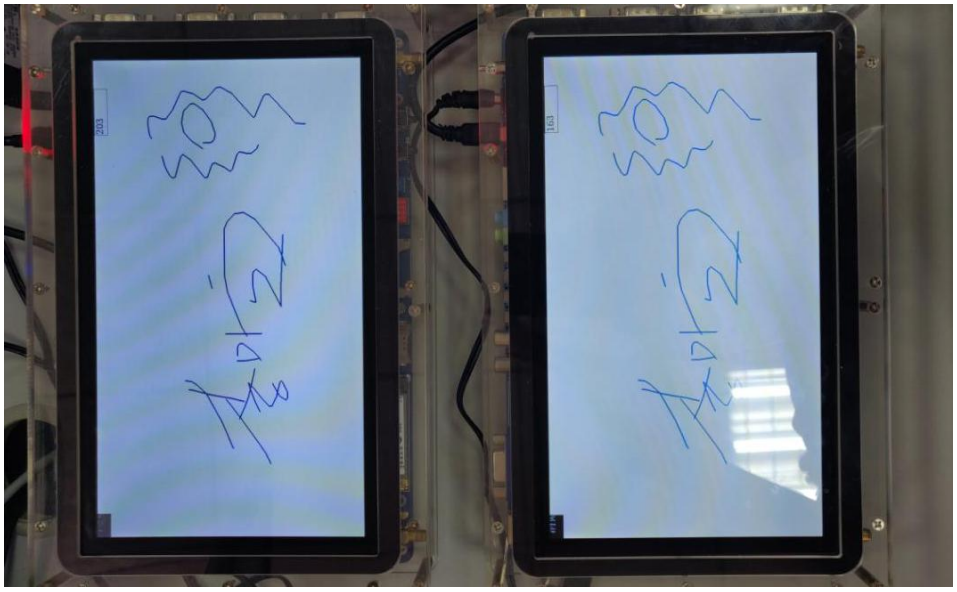

图 6.1 共享平板测试结果图

6.5 实验代码

```
#include <stdio.h>
#include "../common/common.h"

#define COLOR_BACKGROUND      FB_COLOR(0xff,0xff,0xff)
#define COLOR_TEXT            FB_COLOR(0x0,0x0,0x0)
#define BUTTON_H 30
#define BUTTON_W 50

#define TIME_X(1024-160)
#define TIME_Y0
#define TIME_W      100
#define TIME_H30

#define SEND_X      (1024-60)
#define SEND_Y      0
#define SEND_W      60
#define SEND_H      60

/* flag, finger, x, y */
int message[4];
int num_max = 0;
int last_x [5];
int last_y [5];
int last_x1 [5];
int last_y1 [5];

int color[5] =
{
      FB_COLOR(65,105,225),     //Blue
      FB_COLOR(0,255,127),      //Spring Green
      FB_COLOR(255,215,0),      //Gold
      FB_COLOR(255,140,0),      //Dark Orange
      FB_COLOR(255,69,0)         //Orange Red
};

static int touch_fd;
static int bluetooth_fd;
```

```
static void touch_event_cb(int fd)
{
    int type,x,y,finger,i;
    type = touch_read(fd, &x,&y,&finger);
    switch(type)
    {
        case TOUCH_PRESS:
            message[0]=0;message[1]=finger;message[2]=x;message[3]=y;
            myWrite_nonblock(bluetooth_fd, message, 16);
            /* 点击清除按钮 */
            if(x<=BUTTON_W && y<=BUTTON_H)
            {
                fb_draw_rect(0,0,SCREEN_WIDTH
                        ,SCREEN_HEIGHT,FB_COLOR(255,255,255));
                fb_draw_rect(0,0,BUTTON_W
                        ,BUTTON_H,FB_COLOR(0,0,0));
                fb_draw_text(10,10
                        ,"清除 CLEAR",25,FB_COLOR(255,255,255));
                fb_update(); break;
            }
            /* 记录之前手指位置信息，并输出点，宽度为 4 */
            last_x[finger]=x; last_y[finger]=y;
            break;

        case TOUCH_MOVE:
            message[0]=1;message[1]=finger;message[2]=x;message[3]=y;
            myWrite_nonblock(bluetooth_fd, message, 16);
            /* 并输出线，记录手指位置信息，线宽度为 4 */
            fb_draw_line(last_x[finger],last_y[finger],x,y,color[finger]);
            for(i = 1; i < 2; i++)
            {
                fb_draw_line(last_x[finger],last_y[finger]+i,x,y+i,color[finger]);
                fb_draw_line(last_x[finger],last_y[finger]-i,x,y-i,color[finger]);
                fb_draw_line(last_x[finger]+i,last_y[finger],x+i,y,color[finger]);
                fb_draw_line(last_x[finger]-i,last_y[finger],x-i,y,color[finger]);
            }
            last_x[finger]=x;
            last_y[finger]=y;
            fb_update();
            break;
```

```
            case TOUCH_ERROR:
                printf("close touch fd\n");
                task_delete_file(fd);
                close(fd);
                break;
            default:
                return;
        }
        return;
    }

    static void bluetooth_tty_event_cb(int fd)
    {
        char buf[128];
        int n, *p=buf;
        n = myRead_nonblock(fd, buf, 127);
        if(n <= 0) return;
        int finger=p[1], x=p[2], y=p[3],i;
        if(p[0]==0)
        {
            /* 点击清除按钮 */
            if(x<=BUTTON_W && y<=BUTTON_H)
            {
                fb_draw_rect(0,0,SCREEN_WIDTH
                   ,SCREEN_HEIGHT,FB_COLOR(255,255,255));
                fb_draw_rect(0,0,BUTTON_W,BUTTON_H,FB_COLOR(0,0,0));
                fb_draw_text(10,10,"清除 CLEAR",25,FB_COLOR(255,255,255));
                fb_update();
            }
            /* 记录之前手指位置信息，并输出点，宽度为 4 */
            last_x1[finger]=x;
            last_y1[finger]=y;
        }
        else if(p[0]==1)
        {
            /* 并输出线，记录手指位置信息，线宽度为 4 */
            fb_draw_line(last_x1[finger],last_y1[finger],x,y,color[finger]);
            for(i = 1; i < 2; i++)
            {
                fb_draw_line(last_x1[finger],last_y1[finger]+i,x,y+i,color[finger]);
                fb_draw_line(last_x1[finger],last_y1[finger]-i,x,y-i,color[finger]);
                fb_draw_line(last_x1[finger]+i,last_y1[finger],x+i,y,color[finger]);
```

```
                fb_draw_line(last_x1[finger]-i,last_y1[finger],x-i,y,color[finger]);
            }
            last_x1[finger]=x;
            last_y1[finger]=y;
            fb_update();
        }
    return;
}

static int bluetooth_tty_init(const char *dev)
{
    int fd = open(dev, O_RDWR|O_NOCTTY|O_NONBLOCK); /*非阻塞模式*/
    if(fd < 0){
        printf("bluetooth_tty_init open %s error(%d):
                    %s\n", dev, errno, strerror(errno));
        return -1;
    }
    return fd;
}

static int st=0;
static void timer_cb(int period) /*该函数 0.5 秒执行一次*/
{
    char buf[100];
    sprintf(buf, "%d", st++);
    fb_draw_rect(TIME_X, TIME_Y,
                 TIME_W, TIME_H, COLOR_BACKGROUND);
    fb_draw_border(TIME_X, TIME_Y, TIME_W, TIME_H, COLOR_TEXT);
    fb_draw_text(TIME_X+2, TIME_Y+20, buf, 24, COLOR_TEXT);
    fb_update();
    return;
}

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,COLOR_BACKGROUND);
    /* 绘制清除按钮 */
    fb_draw_rect(0,0,BUTTON_W,BUTTON_H,FB_COLOR(0,0,0));
    fb_update();
```

```
    touch_fd = touch_init("/dev/input/event3");
    task_add_file(touch_fd, touch_event_cb);

    bluetooth_fd = bluetooth_tty_init("/dev/rfcomm0");
    if(bluetooth_fd == -1) return 0;
    task_add_file(bluetooth_fd, bluetooth_tty_event_cb);

    task_add_timer(500, timer_cb); /*增加 0.5 秒的定时器*/
    task_loop();
    return 0;
}
```

7 实验总结与体会

本次课程的实验的目的是帮助我们学习嵌入式系统开发的基本流程与步骤，学习嵌入式开发的基本方式和常用的工具。

实验 1 主要进行内核的编译与烧入，需要了解基本的 linux 的系统开发流程，掌握基本的系统开发工具如 adb、fastboot 等工具的使用，在实验 1 中，我们主要学习了如何使用其他设备将系统烧写进嵌入式设备中，将嵌入式设备从裸机转变为具有可用性的设备，了解了嵌入式设备的基本开发方式。

实验 2 主要进行基于 linux 下 LCD 驱动接口 framebuffer 的图像显示功能的开发，本节实验需要了解双缓冲机制的工作流程，并实现画线函数与画矩形函数的编写。本实验要求我们学习嵌入式系统交叉编译的主要流程，掌握嵌入式系统的基本开发流程，在本节中，我基本掌握了基于 framebuffer 图像显示的设计与实现。

实验 3 是实验 2 的一个延伸，需要将图片显示到屏幕上，本节是我出问题最多的一节，由于失误导致我编写的代码频繁出现“段错误”的情况，经过长时间的排查，我发现是变量在进行自增时出现问题，修改后程序正常。通过本节的实验，我发现嵌入式设备对段错误十分敏感，需要小心注意这个问题。

在实验 4 中，我学习了多点触摸协议的使用方式，了解了触摸屏驱动接口的使用，并成功实现手指轨迹绘制与清除的功能。

实验 5 和综合实验我都是基于蓝牙进行互作的应用开发，由于是首次接触蓝牙开发，导致我再启动和连接蓝牙设备的过程中频繁出现错误和问题，在进行一系列的调整后，实现了这些功能。

通过以上实验的开发，我基本掌握了基础的嵌入式系统的开发流程，了解了嵌入式系统开发的编译以及调试工具，增强了代码编写能力，加深了对理论知识的理解，感受到 Linux 在嵌入式系统中的重要地位。

一、原创性声明

本人郑重声明本报告内容，是由作者本人独立完成的。有关观点、方法、数据和文献等的引用已在文中指出。除文中已注明引用的内容外，本报告不包含任何其他个人或集体已经公开发表的作品成果，不存在剽窃、抄袭行为。

特此声明！

作者签字：李响