《嵌入式系统荣誉》课程

作业：

题目——LCD 驱动程序开发

姓名：阮玉斌

学号：2019112043

班级：自动化 3 班软硬件方向

一、任务要求：

1、掌握挂接根文件系统；
2、编译 LCD 驱动程序与应用程序，学习驱动程序开发；
3、测试功能，在 Linux 系统中实现裸板 LCD 功能。

二、思路：

1、根文件系统挂接

（1）Ping 通虚拟机和开发板

①将 PC 机（本实验使用笔记本电脑），开发板和有线网络（服务器在远端）通过交换机连接到一起。注意，没有交换机或路由器情况下将 PC 机和开发板直连，无法实现虚拟机和开发板通信，无法使用 nfs 文件系统。

②修改三者网络设置，使其 IP 均处于统一有效网段上（以 PC 机有线网卡设置网段为基准）。

③关闭 PC 机和虚拟机防火墙。

④虚拟机和开发板互 ping，注意，需要提前在开发板上烧录 u-boot，内核和根文件系统，否则开发板无法回应虚拟机 ping 请求。

（2）在配置文件/etc/exports 中定义被挂接目录

在/etc/exports 中添加如下代码：

/work/nfs_root *(rw, sync, no_root_squash)

/work/nfs_root/first_fs　*(rw,sync,no_root_squash)

（3）重启 NFS 服务

（4）本机测试

（5）挂接单板

相应命令如下：

mount -t nfs -o intr,nolock,rsize=1024,wsize=1024 虚拟机网址:/被挂接目录 /挂接目录

（或“mount -t nfs -o nolock,vers=2　虚拟机网址:/work/nfs_root/first_fs　/mnt”）。

2、LCD 驱动程序开发

（1）编写 Linux 设备驱动程序的流程如下：

①查看原理图、芯片手册；

②在内核中找到相近的驱动程序，在此基础上进行开发；

③实现驱动程序初始化；

④设计所要实现的操作；

⑤编译该驱动程序到内核中，或用 insmod 命令加载；

⑥测试驱动程序。

（2）LCD 设备驱动概念框架如下所示

用户层	应用程序（对应提供驱动程序的读写等函数）			
	C 库	open	read	write
内核	system call interface		swi val	
VFS	sys_open	sys_read	sys_write	
驱动	drv_open	drv_read	drv_write	
硬件		4.3 寸 LCD		

（3）驱动代码思路

基于内核自带 LCD 驱动程序 fbmem.c，采用分层架构（如下图 2-1 所示），大致如下：

① 配置主设备号

② 构造驱动中应用程序操作函数“open,read,write"等

③ 注册字符设备

④ 入口函数

⑤ 出口函数

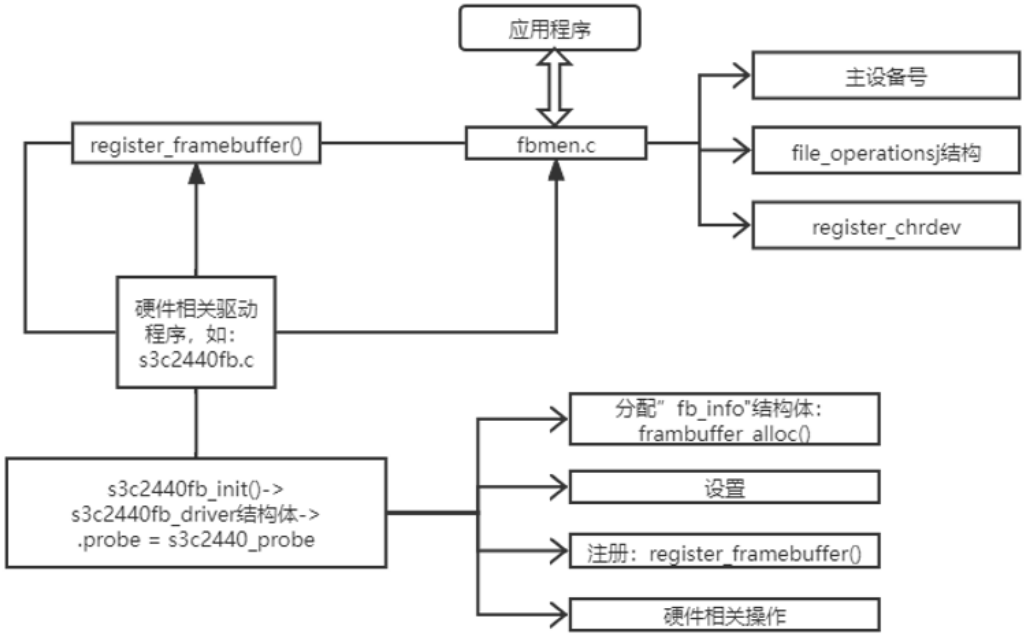

图 2-1　LCD 驱动分层架构

根据上图框架程序，依赖于底层某个设备程序给它注册一个“fb_info”结构体（由“register_framebuffer()”来注册。然后进行硬件的相关操作：①根据 LCD 手册设置 LCD 控制器（配置模式、端口、时钟参数等），②分配显存，并将显存告诉 LCD 寄存器，配置 LCD 的颜色格式和像素点表示方式，③配置 GPIO 引脚用于 LCD。

LCD 硬件操作有一部分是调色板设置，因 LCD 每像素需要 16bpp，调色板是一块内存，主要用于 LCD 像素 16bit 数据的转换。但我们实验所用 LCD 采用 RGB-565，每像素是 16 位，不许转换，代码中仍然保留一个“假的调色板”，为了兼容以前的程序，实则本实验中用不到。

出口函数主要负责使用 unregister_framebuffer（）注释去掉注册结构体 fb_info，关闭 LCD，释放掉分配的内存和 framebuffer 结构体。

3、LCD 测试程序开发

（1）编写实现对应裸板 LCD 功能的测试程序

对应测试程序的结构如下：

① 打开设备 fd=open(filename, O_RDWR);

② 实现功能：open、write、read;

③ valu 值传入驱动；

④ 增加头文件

4、测试

（1）流程思路

测设的流程整体可分为四个部分：编译、拷贝、加载、测试。

（2）编译

① 驱动程序编译

编写对应的 Makefile 文件。Makefile 链接目录为对应内核的根目录，通过编译 lcd_drv.c，可得到对应的 lcd_drv.ko 文件。

② 测试程序编译

③ 配置新内核（不含有内核自带 LCD 驱动）

到 Linux 内核根目录下，执行 make menuconfig 命令下，去掉原有内核中的 LCD 程序。

具体选项如下：

-> Device Drivers

-> Graphics support

<M> S3C2410 LCD framebuffer support

上面“ <*> S3C2410 LCD framebuffer support”是之前自带的驱动程序，这里去掉后以<M>模块方式编译进内核。

执行 config_ok .config，执行 make menuconfig（生成不包含原有 LCD 驱动的 uImage 文件）。

设置为<M>是下面三个函数：cfbcopyare、cfbimageblit、cfbfillrect，使用“make modules”可以得到对应的“.ko”文件。

上述驱动程序编译所得的“.ko”文件，在挂接到根文件系统情况下，可以加载到环境中。

（3）拷贝

实验准备前挂接好根文件系统，将编译好的新内核拷贝到网络文件系统目录下，并命名为 uImage_nolcd，对应的拷贝命令为：“cp arch/arm/boot/uImage /work/nfs_root/uImage_nolcd”。

本实验中自定义的 LCD 驱动程序中会使用部分原有内核编译后生成的模块（即上文编译过程中生成的内核的三个模块），输入命令“cp drivers/video/cfb*.ko /work/nfs_root/first_fs”（在 drivers/video/目录下生成了与 lcd 驱动相关的三个模块，cfbcopyarea.ko、cfbfillrect.ko、cfbimgblt.ko，将他们复制到之后 lcd 驱动程序在的目录下，用于与 lcd 驱动程序一起加载进内核）。

（4）挂接和加载

启动开发板进入 u-boot，输入空格+“q”退出菜单，进入命令行模式。

输入命令“nfs 30000000 虚拟机 ip 地址:/work/nfs_root/uImage_nolcd”挂载编译好的无自带 LCD 驱动的内核，挂载成功后输入命令“bootm 30000000”重新启动开发板。

使用命令“mount -t nfs -o nolock,vers=2 虚拟机 IP 地址:/work/nfs_root/first_fs /mnt”挂载驱动程序和编译好的三个模块函数的程序所在目录到开发板上。

加载驱动模块，对应命令如下：

insmod cfbcopyarea.ko

insmod cfbfillrect.ko

insmod cfbimgblt.ko

insmod lcd_drv.ko

常见的 Linux 命令行中模块命令有：①insmod（加载）、②rmmod（卸载）、③lsmod（查看已加载模块）。

lcd_drv.ko 模块装载好后，lcd 背光点亮，开始进行应用程序测试。

（5）测试程序执行

输入如下命令：

echo hello > /dev/tty1 // 可以在 LCD 上看见 hello

cat lcd_drv.ko > /dev/fb0 // 花屏.

上述命令直接把 lcd_drv.ko 中的内直接放到 LCD 上，无法对内容进行格式识别，则花屏。

然后执行自定义裸板 LCD 功能测试程序能否在 Linux 系统上正常运行，此外可通过运行开发

板配套的 fb_test 进行测试，获取 frame buffer 属性。

三、步骤：（含代码分析、调试过程分析及验证过程图片）

1、代码分析

下方代码仅截取项目功能主要源代码部分：

（1）/****　lcd_drv.c　****/

```
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/errno.h>
#include <linux/string.h>
#include <linux/mm.h>
#include <linux/slab.h>
#include <linux/delay.h>
#include <linux/fb.h>
#include <linux/init.h>
#include <linux/dma-mapping.h>
#include <linux/interrupt.h>
#include <linux/workqueue.h>
#include <linux/wait.h>
#include <linux/platform_device.h>
#include <linux/clk.h>

#include <asm/io.h>
#include <asm/uaccess.h>
#include <asm/div64.h>

#include <asm/mach/map.h>
#include <asm/arch/regs-lcd.h>
#include <asm/arch/regs-gpio.h>
#include <asm/arch/fb.h>

static int s3c_lcdfb_setcolreg(unsigned int regno, unsigned int red,
                               unsigned int green, unsigned int blue,
                               unsigned int transp, struct fb_info *info);


struct lcd_regs {
    unsigned long   lcdcon1;
    unsigned long   lcdcon2;
    unsigned long   lcdcon3;
    unsigned long   lcdcon4;
    unsigned long   lcdcon5;
    unsigned long   lcdsaddr1;
    unsigned long   lcdsaddr2;
    unsigned long   lcdsaddr3;
    unsigned long   redlut;
    unsigned long   greenlut;
    unsigned long   bluelut;
    unsigned long   reserved[9];
    unsigned long   dithmode;
    unsigned long   tpal;
    unsigned long   lcdintpnd;
    unsigned long   lcdsrcpnd;
    unsigned long   lcdintmsk;
    unsigned long   lpcsel;
};

static struct fb_ops s3c_lcdfb_ops = {
    .owner      = THIS_MODULE,
    .fb_setcolreg   = s3c_lcdfb_setcolreg,
    .fb_fillrect    = cfb_fillrect,
    .fb_copyarea    = cfb_copyarea,
    .fb_imageblit   = cfb_imageblit,
};
```

```c
static struct fb_info *s3c_lcd;
static volatile unsigned long *gpbcon;
static volatile unsigned long *gpbdat;
static volatile unsigned long *gpccon;
static volatile unsigned long *gpdcon;
static volatile unsigned long *gpgcon;
static volatile struct lcd_regs* lcd_regs;
static u32 pseudo_palette[16];


/* from pxafb.c */
static inline unsigned int chan_to_field(unsigned int chan, struct fb_bitfield *bf)
{
    chan &= 0xffff;
    chan >>= 16 - bf->length;
    return chan << bf->offset;
}


static int s3c_lcdfb_setcolreg(unsigned int regno, unsigned int red,
                unsigned int green, unsigned int blue,
                unsigned int transp, struct fb_info *info)
{
    unsigned int val;

    if (regno > 16)
        return 1;

    /* 用red,green,blue三原色构造出val */
    val  = chan_to_field(red,   &info->var.red);
    val |= chan_to_field(green, &info->var.green);
    val |= chan_to_field(blue,  &info->var.blue);

    //((u32 *)(info->pseudo_palette))[regno] = val;
    pseudo_palette[regno] = val;
    return 0;
}

static int lcd_init(void)
{
    /* 1. 分配一个fb_info */
    s3c_lcd = framebuffer_alloc(0, NULL);

    /* 2. 设置 */
    /* 2.1 设置固定的参数 */
    strcpy(s3c_lcd->fix.id, "mylcd");
    s3c_lcd->fix.smem_len = 480*272*16/8;
    s3c_lcd->fix.type     = FB_TYPE_PACKED_PIXELS;
    //默认值，此宏为0，表示支持大多数LCD
    s3c_lcd->fix.visual   = FB_VISUAL_TRUECOLOR; /* TFT 真彩色屏*/
    s3c_lcd->fix.line_length = 480*2;

    /* 2.2 设置可变的参数 */
    s3c_lcd->var.xres           = 480;
    s3c_lcd->var.yres           = 272;
    s3c_lcd->var.xres_virtual   = 480;
    s3c_lcd->var.yres_virtual   = 272;
    s3c_lcd->var.bits_per_pixel = 16;
    //xoffset/yoffset表示虚拟和实际分辨率的偏差值，此处不用定义

    /* RGB:565 */
    s3c_lcd->var.red.offset     = 11;
    s3c_lcd->var.red.length     = 5;

    s3c_lcd->var.green.offset   = 5;
    s3c_lcd->var.green.length   = 6;

    s3c_lcd->var.blue.offset    = 0;
    s3c_lcd->var.blue.length    = 5;
```

```
    s3c_lcd->var.activate       = FB_ACTIVATE_NOW;  //默认值

    /* 2.3 设置操作函数 */
    s3c_lcd->fbops              = &s3c_lcdfb_ops;

    /* 2.4 其他的设置 */
    /* 调色盘设置 */
    s3c_lcd->pseudo_palette = pseudo_palette;
    //s3c_lcd->screen_base  = ;  /* 显存的虚拟地址 */
    s3c_lcd->screen_size   = 480*272*16/8;

    /* 3. 硬件相关的操作 */
    /* 3.1 配置GPIO用于LCD */

    /* IO寄存器  */
    gpccon = ioremap(0x56000020, 4);
    gpdcon = ioremap(0x56000030, 4);
    *gpccon  = 0xaaaaaaaa;   /* GPIO管脚用于VD[7:0],LCDVF[2:0],VM,VFRAME,VLINE,VCLK,LEND */
    *gpdcon  = 0xaaaaaaaa;   /* GPIO管脚用于VD[23:8] */

    /* LCD_PWENB */
    gpbcon = ioremap(0x56000010, 4);
    gpbdat = gpbcon+1;

    gpgcon = ioremap(0x56000060, 4);

    *gpbcon &= ~(3);  /* GPB0设置为输出引脚,背光 */
    *gpbcon |= 1;
    *gpbdat &= ~1;     /* 输出低电平 */

    *gpgcon |= (3<<8); /* GPG4用作LCD_PWREN */

    /* 3.2 根据LCD手册设置LCD控制器, 比如VCLK的频率等 */
    lcd_regs = ioremap(0x4D000000, sizeof(struct lcd_regs));

    /* bit[17:8]: VCLK = HCLK / [(CLKVAL+1) x 2], LCD手册P14
     *            10MHz(100ns) = 100MHz / [(CLKVAL+1) x 2]
     *            CLKVAL = 4
     * bit[6:5]: 0b11, TFT LCD
     * bit[4:1]: 0b1100, 16 bpp for TFT
     * bit[0]  : 0 = Disable the video output and the LCD control signal.
     */
    lcd_regs->lcdcon1  = (4<<8) | (3<<5) | (0x0c<<1) | (0<<0);

#if 1
    /* 垂直方向的时间参数
     * bit[31:24]: VBPD, VSYNC之后再过多长时间才能发出第1行数据
     *             LCD手册 T0-T2-T1=4
     *             VBPD=3
     * bit[23:14]: 多少行, 320, 所以LINEVAL=320-1=319
     * bit[13:6] : VFPD, 发出最后一行数据之后，再过多长时间才发出VSYNC
     *             LCD手册T2-T5=322-320=2, 所以VFPD=2-1=1
     * bit[5:0]  : VSPW, VSYNC信号的脉冲宽度, LCD手册T1=1, 所以VSPW=1-1=0
     */
    lcd_regs->lcdcon2  = (1<<24) | (271<<14) | (1<<6) | (9);

    /* 水平方向的时间参数
     * bit[25:19]: HBPD, VSYNC之后再过多长时间才能发出第1行数据
     *             LCD手册 T6-T7-T8=17
     *             HBPD=16
     * bit[18:8]: 多少列, 240, 所以HOZVAL=240-1=239
     * bit[7:0] : HFPD, 发出最后一行里最后一个象素数据之后，再过多长时间才发出HSYNC
     *             LCD手册T8-T11=251-240=11, 所以HFPD=11-1=10
     */
    lcd_regs->lcdcon3 = (1<<19) | (479<<8) | (1);

    /* 水平方向的同步信号
     * bit[7:0] : HSPW, HSYNC信号的脉冲宽度, LCD手册T7=5, 所以HSPW=5-1=4
     */
    lcd_regs->lcdcon4 = 40;
```

```
#else
lcd_regs->lcdcon2 = S3C2410_LCDCON2_VBPD(5) | \
        S3C2410_LCDCON2_LINEVAL(319) | \
        S3C2410_LCDCON2_VFPD(3) | \
        S3C2410_LCDCON2_VSPW(1);

lcd_regs->lcdcon3 = S3C2410_LCDCON3_HBPD(10) | \
        S3C2410_LCDCON3_HOZVAL(239) | \
        S3C2410_LCDCON3_HFPD(1);

lcd_regs->lcdcon4 = S3C2410_LCDCON4_MVAL(13) | \
        S3C2410_LCDCON4_HSPW(0);

#endif
    /* 信号的极性
     * bit[11]: 1=565 format
     * bit[10]: 0 = The video data is fetched at VCLK falling edge
     * bit[9] : 1 = HSYNC信号要反转,即低电平有效
     * bit[8] : 1 = VSYNC信号要反转,即低电平有效
     * bit[6] : 0 = VDEN不用反转
     * bit[3] : 0 = PWREN输出0
     ^M* bit[1] : 0 = BSWP
     * bit[0] : 1 = HWSWP 2440手册P413
     */
    lcd_regs->lcdcon5 = (1<<11) | (0<<10) | (1<<9) | (1<<8) | (1<<0);

    /* 3.3 分配显存(framebuffer), 并把地址告诉LCD控制器 */
    s3c_lcd->screen_base = dma_alloc_writecombine(NULL, s3c_lcd->fix.smem_len,\
        &s3c_lcd->fix.smem_start, GFP_KERNEL);

    lcd_regs->lcdsaddr1  = (s3c_lcd->fix.smem_start >> 1) & ~(3<<30);
    lcd_regs->lcdsaddr2  = ((s3c_lcd->fix.smem_start + s3c_lcd->fix.smem_len) >> 1) & 0x1fffff;
    lcd_regs->lcdsaddr3  = (480*16/16);  /* 一行的长度(单位: 2字节) */

    //s3c_lcd->fix.smem_start = xxx;  /* 显存的物理地址 */
    /* 启动LCD */
    lcd_regs->lcdcon1 |= (1<<0); /* 使能LCD控制器 */
    lcd_regs->lcdcon5 &= (~(1<<5)); /* 设置LCD_PWREN的极性: 正常/反转 */
    lcd_regs->lcdcon5 |= (0<<5);
    lcd_regs->lcdcon5 &= (~(1<<3));/* 使能LCD本身 */
    lcd_regs->lcdcon5 |= (1<<3);
    *gpbdat |= 1;     /* 输出高电平, 使能背光 */

    /* 4. 注册 */
    register_framebuffer(s3c_lcd);

    return 0;
}

static void lcd_exit(void)
{
    unregister_framebuffer(s3c_lcd);

    lcd_regs->lcdcon1 &= ~(1<<0); /* 关闭LCD本身 */
    *gpbdat &= ~1;     /* 关闭背光 */

    //取消端口映射
    iounmap(lcd_regs);
    iounmap(gpbcon);
    iounmap(gpccon);
    iounmap(gpdcon);
    iounmap(gpgcon);

    /* 释放帧内存  */
    dma_free_writecombine(NULL, s3c_lcd->fix.smem_len, s3c_lcd->screen_base, \
        s3c_lcd->fix.smem_start);

    /* 释放fd_info 结构体空间 */
    framebuffer_release(s3c_lcd);
}
```

```
module_init(lcd_init);
module_exit(lcd_exit);

MODULE_LICENSE("GPL");
```

代码分析：见上述代码注释部分。

（2）/****　lcd_drv.c 对应 Makefile　****/

```
KERN_DIR = /work/system/linux-2.6.22.6
all:
	make -C $(KERN_DIR) M=`pwd` modules
clean:
	make -C $(KERN_DIR) M=`pwd` modules clean
	rm -rf modules.order

obj-m	+= lcd_drv.o

```

（3）/****　lcd_test.c　****/

```
#include "fb.h"

#define ALLOW_OS_CODE 1
/*#include "../rua/include/rua.h"*/

#if 0
#define DEB(f) (f)
#else
#define DEB(f)
#endif

typedef unsigned char RMuint8;
typedef unsigned short RMuint16;
typedef unsigned int RMuint32;

struct fb_var_screeninfo fb_var;
struct fb_fix_screeninfo fb_fix;
char * fb_base_addr = NULL;

/*
 * 画点
 * 输入参数：
 *     x、y : 象素坐标
 *     color: 颜色值
 *         对于16BPP: color的格式为0xAARRGGBB (AA = 透明度),
 *     需要转换为5:6:5格式
 *         对于8BPP: color为调色板中的索引值,
 *     其颜色取决于调色板中的数值
 */
static void set_pixel(RMuint32 x, RMuint32 y, RMuint32 color)
{
    /*static RMuint32 i=0;*/
    /* TODO We assume for now we have contigus regions */
    RMuint8 red, green, blue;

    switch (fb_var.bits_per_pixel){
        case 16:
            {
                RMuint16 *addr = (RMuint16 *) fb_base_addr+(y*fb_var.xres+x);
                red   = (color >> 19) & 0x1f;
                green = (color >> 10) & 0x3f;
                blue  = (color >>  3) & 0x1f;
                color = (red << 11) | (green << 5) | blue; // 格式5:6:5
                *addr = (RMuint16) color;
            }
            break;
```

```
        default:
            fprintf(stderr,"Unknown bpp : %d\n",fb_var.bits_per_pixel);
            break;
    }
    /*if (i<10){
        DEB(fprintf(stderr,"(%ld,%ld) [%p] <- %lX\n",x,y,addr,*addr));
        i++;
    }*/
}

/*
 * 画线
 * 输入参数：
 *     x1、y1 : 起点坐标
 *     x2、y2 : 终点坐标
 *     color  : 颜色值
 *         对于16BPP: color的格式为0xAARRGGBB (AA = 透明度),
 *     需要转换为5:6:5格式
 *         对于8BPP: color为调色板中的索引值,
 *     其颜色取决于调色板中的数值
 */
static void DrawLine(RMuint32 x1,RMuint32 y1,RMuint32 x2,RMuint32 y2,RMuint32 color)
{
    RMuint32 dx,dy,e;
    dx=x2-x1;
    dy=y2-y1;

    if(dx>=0)
    {
        if(dy >= 0) // dy>=0
        {
            if(dx>=dy) // 1/8 octant
            {
                e=dy-dx/2;
                while(x1<=x2)
                {
                    set_pixel(x1,y1,color);
                    if(e>0){y1+=1;e-=dx;}
                    x1+=1;
                    e+=dy;
                }
            }
            else        // 2/8 octant
            {
                e=dx-dy/2;
                while(y1<=y2)
                {
                    set_pixel(x1,y1,color);
                    if(e>0){x1+=1;e-=dy;}
                    y1+=1;
                    e+=dx;
                }
            }
        }
        else           // dy<0
        {
            dy=-dy;   // dy=abs(dy)

            if(dx>=dy) // 8/8 octant
            {
                e=dy-dx/2;
                while(x1<=x2)
                {
                    set_pixel(x1,y1,color);
                    if(e>0){y1-=1;e-=dx;}
                    x1+=1;
                    e+=dy;
                }
```

```
                }
            }
            else        // 7/8 octant
            {
                e=dx-dy/2;
                while(y1>=y2)
                {
                    set_pixel(x1,y1,color);
                    if(e>0){x1+=1;e-=dy;}
                    y1-=1;
                    e+=dx;
                }
            }
        }
    }

    else //dx<0
    {
        dx=-dx;     //dx=abs(dx)
        if(dy >= 0) // dy>=0
        {
            if(dx>=dy) // 4/8 octant
            {
                e=dy-dx/2;
                while(x1>=x2)
                {
                    set_pixel(x1,y1,color);
                    if(e>0){y1+=1;e-=dx;}
                    x1-=1;
                    e+=dy;
                }
            }
            else        // 3/8 octant
            {
                e=dx-dy/2;
                while(y1<=y2)
                {
                    set_pixel(x1,y1,color);
                    if(e>0){x1-=1;e-=dy;}
                    y1+=1;
                    e+=dx;
                }
            }
        else           // dy<0
        {
            dy=-dy;   // dy=abs(dy)

            if(dx>=dy) // 5/8 octant
            {
                e=dy-dx/2;
                while(x1>=x2)
                {
                    set_pixel(x1,y1,color);
                    if(e>0){y1-=1;e-=dx;}
                    x1-=1;
                    e+=dy;
                }
            }
            else        // 6/8 octant
            {
                e=dx-dy/2;
                while(y1>=y2)
                {
                    set_pixel(x1,y1,color);
                    if(e>0){x1-=1;e-=dy;}
                    y1-=1;
                    e+=dx;
                }
            }
        }
    }
}
```

```c
/*
 * 绘制同心圆
 */
static void Mire(void)
{
    RMuint32 x,y;
    RMuint32 color;
    RMuint8 red,green,blue,alpha;

    DEB(fprintf(stderr,"begin mire\n"));
    for (y=0;y<fb_var.yres;y++)
        for (x=0;x<fb_var.xres;x++){
            color = ((x-fb_var.xres/2)*(x-fb_var.xres/2) + (y-fb_var.yres/2)*(y-fb_var.yres/2))/64;
            red   = (color/8) % 256;
            green = (color/4) % 256;
            blue  = (color/2) % 256;
            alpha = (color*2) % 256;
            /*alpha = 0xFF;*/

            color |= ((RMuint32)alpha << 24);
            color |= ((RMuint32)red   << 16);
            color |= ((RMuint32)green << 8 );
            color |= ((RMuint32)blue       );

            set_pixel(x,y,color);
        }

    DEB(fprintf(stderr,"end mire\n"));
}

/*
 * 将屏幕清成单色
 * 输入参数：
 *     color: 颜色值
 *         对于16BPP: color的格式为0xAARRGGBB (AA = 透明度),
 *     需要转换为5:6:5格式
 *         对于8BPP: color为调色板中的索引值，
 *     其颜色取决于调色板中的数值
 */
static void ClearScr(RMuint32 color)
{
    RMuint32 x,y;

    for (y = 0; y < fb_var.yres; y++)
        for (x = 0; x < fb_var.xres; x++)
            set_pixel(x, y, color);
}

/*******   测试文件命令提示    *******/
void printusage(char *name)
{
        fprintf(stderr,"Usage (example): %s /dev/fb0\n", name);
        fprintf(stderr,"                 %s /dev/fb0 <DrawLine|Mire>\n",name);
        fprintf(stderr,"                 eg.  \n");
        fprintf(stderr,"                 %s /dev/fb0 DrawLine\n",name);
        fprintf(stderr,"                 %s /dev/fb0 Mire\n",name);
}
```

```c
int main(int argc, char **argv)
{
    int fd=0;
    char* filename;
    char val;

    printf("%dx%d, %dbpp\n", fb_var.xres, fb_var.yres, fb_var.bits_per_pixel );

    if (argc != 3)
    {
        printusage(argv[0]);
        return 0;
    }

    filename = argv[1];

    fd = open(filename, O_RDWR);


    if (fd < 0)
    {
        printf("error, can't open %s\n", filename);
        return 0;
    }

    /* Get fixed screen information */      //获取fb_fix结构体参数
    if (ioctl(fd, FBIOGET_FSCREENINFO, &fb_fix)) {
        printf("Error reading fb fixed information.\n");
        exit(1);
    }

    /* Get variable screen information  */  //获取fb_var结构体参数
    if (ioctl(fd, FBIOGET_VSCREENINFO, &fb_var)) {
        printf("Error reading fb variable information.\n");
        exit(1);
    }

    fb_var.xres = fb_var.xres_virtual = 480;
    fb_var.yres = fb_var.yres_virtual = 272;
    fb_var.bits_per_pixel = 16;
    printf("%dx%d, %dbpp\n", fb_var.xres, fb_var.yres, fb_var.bits_per_pixel );

    if (!strcmp("DrawLine", argv[2]))
    {
        //画线
        ClearScr(0x0);
        DrawLine(0  , 0  , 479, 0  , 0xff0000);    // 红色
        DrawLine(0  , 0  , 0  , 271, 0x00ff00);    // 绿色
        DrawLine(479, 0  , 479, 271, 0x0000ff);    // 蓝色
        DrawLine(0  , 271, 479, 271, 0xffffff);    // 白色

        DrawLine(0  , 0  , 479, 271, 0xffff00);    // 黄色
        DrawLine(479, 0  , 0  , 271, 0x8000ff);    // 紫色
        DrawLine(240, 0  , 240, 271, 0xe6e8fa);    // 银色
        DrawLine(0  , 136, 479, 136, 0xcd7f32);    // 金色
    }
    else if (!strcmp("Mire", argv[2]))
    {
        // 画圈
        Mire();
    }
    else
    {
        ClearScr(0xff0000);             // 红色
        printusage(argv[0]);
        return 0;
    }

    close(fd);
    return 0;
}
```

2、调试过程

（1）编译不含有自带 LCD 驱动的内核

① 解压缩：“tar xjf linux-2.6.22.6.tar.bz2”

② 在该内核根目录下，打补丁：“patch -p1 <../linux-2.6.22.6_jz2440.patch”

③ 配置，使用默认配置，在上边修改，在 arch/ arm/ configs 找到配置文件 s3c2410_defconfig，输入“make s3c2410_defconfig”“make menuconfig”

④ 进到 make menuconfig 界面去除自带 LCD 驱动，操作过程如下图 3-1 所示：

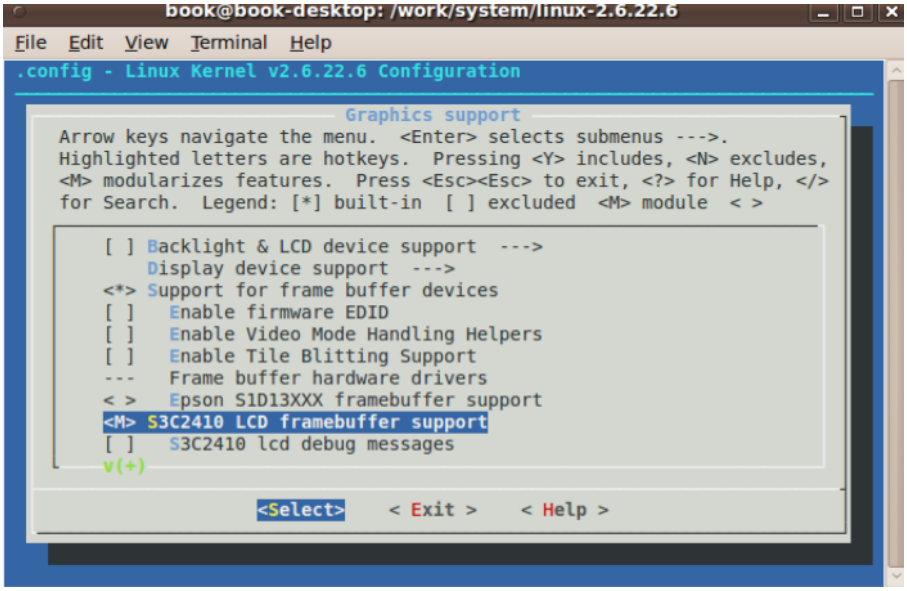

图 3-1　去除内核自带 LCD 驱动过程图

⑤ 编译新内核 “make uImage”

（2）编译自定义 LCD 驱动

编译结果如下图 3-2 所示：

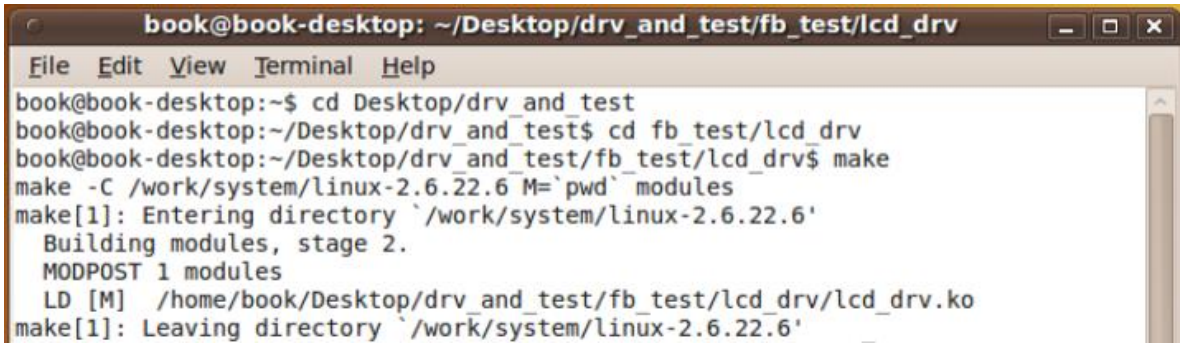

图 3-2　自定义 LCD 驱动编译结果

（3）编译 LCD 测试程序

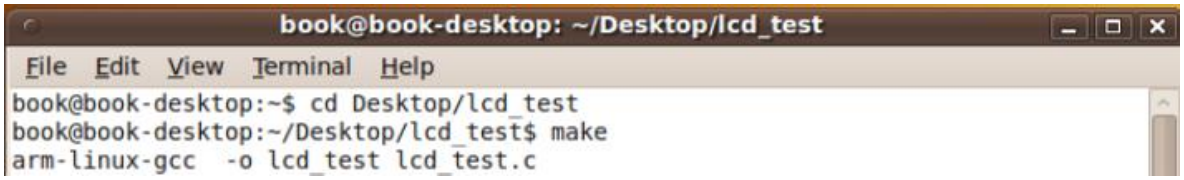

图 3-3　LCD 测试程序编译结果

3、验证过程

（1）通过 nfs 服务器挂载编译好的无自带 LCD 驱动内核

挂载过程截图如下图 3-4 所示。

图 3-4　　新内核挂载成功截图

（2）挂在成功后启动开发板，挂接根文件系统（驱动程序所在目录到开发板临时目录/mnt）

输入挂载命令，网络文件系统挂接结果如下图 3-5 所示。

图 3-5　　NFS 文件系统挂接

（3）驱动模块装载

先装在三个模块函数 cfb*，未装载 lcd_drv.ko 时，没有对应节点，装载成功后，输入 lsmod 命令查看对应模块，可以看到四个模块均成功装载，如图 3-6 所示。此时开发板 LCD 的背光灯亮起，没有输入，则无显示，如图 3-7 所示。

```
# insmod cfbcopyarea.ko
# insmod cfbfillrect.ko
# insmod cfbimgblt.ko
# ls /dev/fb*
ls: /dev/fb*: No such file or directory
# insmod lcd_drv.ko
Console: switching to colour frame buffer device 60x34
# ls /dev/fb*
/dev/fb0
#
```

就绪 Serial: COM18 20, 3 20行, 91列 VT100 大写 数字

图 3-6 加载驱动模块

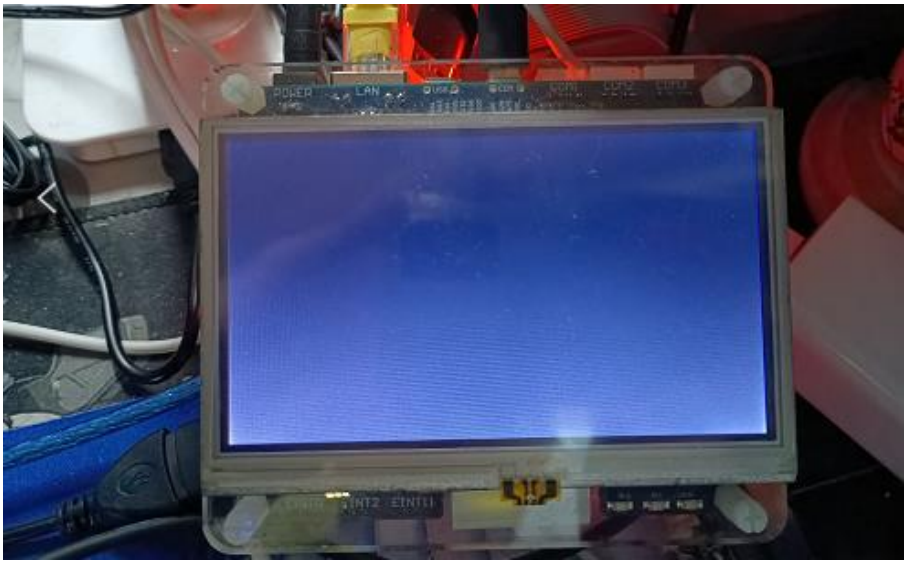

图 3-7 仅加载 LCD 驱动开发板背光开启效果

（4）测试裸板编译程序

加载好 LCD 驱动条件下，加载 LCD 测试程序，过程和测试程序执行结果如下图 3-8 和图 3-9 所示。

```
# lsmod
Module                  Size  Used by    Not tainted
lcd_drv                 3196  1
cfbimgblt               2688  1 lcd_drv
cfbfillrect             3552  1 lcd_drv
cfbcopyarea             3296  1 lcd_drv
#
# ./lcd_test /dev/fb0 set 480 272 16
Change fb as 480x272, 16bpp
480x272, 16bpp
#
```

就绪 Serial: COM18 20, 3 20行, 91列 VT100 大写 数字

图 3-8 加载执行测试过程

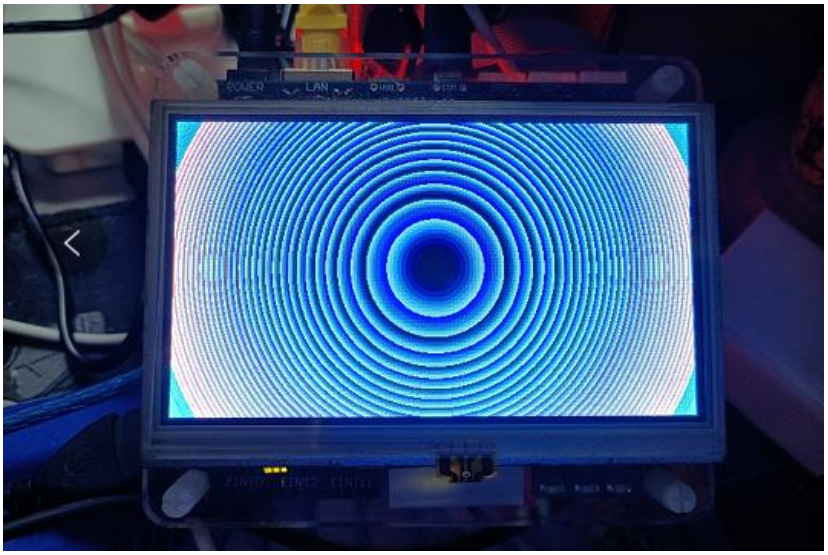

图 3-9　裸板 LCD 功能对应的测试程序执行结果

根据验证结果，编写的 LCD 驱动程序和测试程序均正常，能够执行裸板 LCD 画圆和画线的功能。

四、总结：

通过本次驱动开发，了解了 LInux 系统下驱动程序的重要性和作用，熟悉了驱动程序开发流程和掌握了间的的设备驱动程序的开发方法。本次实验的难点在于驱动程序在编写过程中涉及大量内核文件变量，仅能根据其他同类型驱动为基础进行编写，设置专门设备参数，对于调试过程中出现问题的修改有较大难度。

在本实验的基础上可以尝试多多种设备驱动进行开发，以本次 LCD 驱动程序开发为例，在开发过程中对于底层驱动程序的层次结构有了进一步的了解。