Android系统开发之触摸屏tslib移植(内核)和原理分析

转载 2014年12月12日 09:18:07 909

0

Android系统开发之触摸屏tslib移植(内核)和原理分析

本帖最后由 haolele 于 2011-11-11 21:07 编辑

Android系统开发之触摸屏tslib移植(内核)和原理分析

首先了解一下tslib的运行原理，tslib的运行分成两部分

(1)校验

在LCD固定坐标位置依次显示出5个坐标让用户触摸，把LCD坐标和用户触摸时驱动屏驱动底层的坐标总共5组值保存起来

运行tslib库的算法对其进行运算，得出校准用7个值

(2)校准

每次触摸屏驱动读取到硬件坐标时应用校准用的7个值对该坐标进行一次运算，然后将运算后的坐标作为正常坐标即可。

按照上面的原理，

(1)我们先修改内核部分，我的平台用的触摸屏幕驱动是tsc2007，驱动文件为内核/drivers/input/touchscreen目录下的tsc2007.c和ts_linear.c

其中，ts_linear.c中定义的是校准模块，该模块在proc文件系统中建立了7个文件，用来存放校准用的7个点，7的点的默认值

为1,0,0,0,1,0,1，对应的目标平台文件系统的位置为/proc/sys/dev/ts_device目录下a0,a1,a2,a3,a4,a5,a6等7个文件

此模块中还定义了一个校准函数ts_linear_scale，此函数的主要内容是读取a0,a1,a2,a3,a4,a5,a6等7个文件中的值作为7个

校准值与传入的触摸平坐标值进行运算，返回运算结果。

ts_linear_scale函数定义如下：

```
int ts_linear_scale(int *x, int *y, int swap_xy)
{
    int xtemp, ytemp;

    xtemp = *x;
    ytemp = *y;

    if (cal.a == 0)
        return -EINVAL;

    *x = (cal.a + cal.a * xtemp + cal.a * ytemp) / cal.a;
    *y = (cal.a + cal.a * xtemp + cal.a * ytemp) / cal.a;

    if (swap_xy) {
        int tmp = *x;
        *x = *y;
        *y = tmp;
    }
    return 0;
}
```

ts2007.c为触摸屏驱，与其他驱动不同的地方是在取得硬件坐标值发送之前先调用了ts_linear_scale函数对坐标值进行了校准

```
        if (x > 0 && y > 0)
```

permike
关注

```
        {
            ts_linear_scale(&x, &y, invert);
            input_report_abs(input, ABS_X, x);
            input_report_abs(input, ABS_Y, y);
            input_report_abs(input, ABS_PRESSURE, 255);
            input_report_abs(input, ABS_TOOL_WIDTH, 1);
            input_report_key(input, BTN_TOUCH, 1);
            input_sync(input);
        }
```


、2)在android源代码/system/core/rootdir/init.rc文件中添加tslib相关的宏定义如下：

```
# touchscreen parameters
    export TSLIB_FBDEVICE /dev/graphics/fb0
    export TSLIB_CALIBFILE /data/etc/pointercal
    export TSLIB_CONFFILE  /system/etc/ts.conf
    export TSLIB_TRIGGERDEV /dev/input/event0
    export TSLIB_TSDEVICE /dev/input/event1
```

(2)移植tslib库到android系统，比较麻烦，看下一节的内容。

(3)校验程序完成后会将生成的7个校准值写入到环境变量TSLIB_CALIBFILE对应的路径/data/etc/pointercal文件中

(4)校验完后将pointercal文件中的7个值分别写入到/proc/sys/dev/ts_device目录下a0,a1,a2,a3,a4,a5,a6文件即可。

(5)开机启动的时候我们编写一个应用程序，首先判断环境变量TSLIB_CALIBFILE对应的路径/data/etc/pointercal文件是否存在，如果
文件存在而且非空，则将该文件中的7个值取出来分别写入到/proc/sys/dev/ts_device目录下a0,a1,a2,a3,a4,a5,a6文件

(6)为了确保未校验前触摸屏可用，我们将一次校验后得出的7个坐标值作为初始值，修改到内核ts_linear.c文件中。下面是源代码：
ts_linear.c文件

```
/*
*  Touchscreen Linear Scale Adaptor
*
*  Copyright (C) 2009 Marvell Corporation
*
*  Author: Mark F. Brown <markb@marvell.com>
*  Based on tslib 1.0 plugin linear.c by Russel King
*
* This library is licensed under GPL.
*
*/

#include <linux/module.h>
#include <linux/init.h>
#include <linux/kernel.h>
#include <linux/input.h>
#include <linux/interrupt.h>
#include <linux/wait.h>
#include <linux/delay.h>
#include <linux/platform_device.h>
```

```
#include <linux/proc_fs.h>
#include <linux/sysctl.h>
#include <asm/system.h>

/*
 sysctl-tuning infrastructure.
/
static struct ts_calibration {
* Linear scaling and offset parameters for x,y (can include rotation) */
  int a;
 cal;

static ctl_table ts_proc_calibration_table[] = {
  {
   .ctl_name = CTL_UNNUMBERED,
   .procname = "a0",
   .data = &cal.a,
   .maxlen = sizeof(int),
   .mode = 0666,
   .proc_handler = &proc_dointvec,
   },
  {
   .ctl_name = CTL_UNNUMBERED,
   .procname = "a1",
   .data = &cal.a,
   .maxlen = sizeof(int),
   .mode = 0666,
   .proc_handler = &proc_dointvec,
   },
  {
   .ctl_name = CTL_UNNUMBERED,
   .procname = "a2",
   .data = &cal.a,
   .maxlen = sizeof(int),
   .mode = 0666,
   .proc_handler = &proc_dointvec,
   },
  {
   .ctl_name = CTL_UNNUMBERED,
   .procname = "a3",
   .data = &cal.a,
   .maxlen = sizeof(int),
   .mode = 0666,
   .proc_handler = &proc_dointvec,
   },
  {
   .ctl_name = CTL_UNNUMBERED,
   .procname = "a4",
   .data = &cal.a,
   .maxlen = sizeof(int),
   .mode = 0666,
   .proc_handler = &proc_dointvec,
   },
  {
```

```
        .ctl_name = CTL_UNNUMBERED,
        .procname = "a5",
        .data = &cal.a,
        .maxlen = sizeof(int),
        .mode = 0666,
        .proc_handler = &proc_dointvec,
        },
        {
        .ctl_name = CTL_UNNUMBERED,
        .procname = "a6",
        .data = &cal.a,
        .maxlen = sizeof(int),
        .mode = 0666,
        .proc_handler = &proc_dointvec,
        },

        {.ctl_name = 0}
    };

    static ctl_table ts_proc_root[] = {
        {
        .ctl_name = CTL_UNNUMBERED,
        .procname = "ts_device",
        .mode = 0555,
        .child = ts_proc_calibration_table,
        },
        {.ctl_name = 0}
    };

    static ctl_table ts_dev_root[] = {
        {
        .ctl_name = CTL_DEV,
        .procname = "dev",
        .mode = 0555,
        .child = ts_proc_root,
        },
        {.ctl_name = 0}
    };

    static struct ctl_table_header *ts_sysctl_header;

    int ts_linear_scale(int *x, int *y, int swap_xy)
    {
        int xtemp, ytemp;

        xtemp = *x;
        ytemp = *y;

        if (cal.a == 0)
            return -EINVAL;

        *x = (cal.a + cal.a * xtemp + cal.a * ytemp) / cal.a;
        *y = (cal.a + cal.a * xtemp + cal.a * ytemp) / cal.a;
```

```
    if (swap_xy) {
        int tmp = *x;
        *x = *y;
        *y = tmp;
    }
    return 0;
}

EXPORT_SYMBOL(ts_linear_scale);

static int __init ts_linear_init(void)
{
    ts_sysctl_header = register_sysctl_table(ts_dev_root);
    /* Use default values that leave ts numbers unchanged after transform */
    cal.a = 1;
    cal.a = 0;
    cal.a = 0;
    cal.a = 0;
    cal.a = 1;
    cal.a = 0;
    cal.a = 1;
    return 0;
}

static void __exit ts_linear_cleanup(void)
{
    unregister_sysctl_table(ts_sysctl_header);
}

module_init(ts_linear_init);
module_exit(ts_linear_cleanup);

MODULE_DESCRIPTION("touch screen linear scaling driver");
MODULE_LICENSE("GPL");
```

ts2007.c文件

```
/*
*  linux/drivers/input/touchscreen/tsc2007.c
*
*  touch screen driver for tsc2007
*
*  Copyright (C) 2006, Marvell Corporation
*
*  This program is free software; you can redistribute it and/or modify
*  it under the terms of the GNU General Public License version 2 as
*  published by the Free Software Foundation.
*/

#include <linux/module.h>
#include <linux/init.h>
#include <linux/kernel.h>
#include <linux/input.h>
```

```
#include <linux/interrupt.h>
#include <linux/wait.h>
#include <linux/delay.h>
#include <linux/platform_device.h>
#include <linux/freezer.h>
#include <linux/proc_fs.h>
#include <linux/clk.h>
#include <linux/i2c.h>
#include <mach/gpio.h>

#include <linux/sysctl.h>
#include <asm/system.h>

extern int ts_linear_scale(int *x, int *y, int swap_xy);

/* Use MAV filter */
#define TSC_CMD_SETUP 0xb0

/* Use 12-bit */
#define TSC_CMD_X 0xc0
#define TSC_CMD_PLATEX 0x80
#define TSC_CMD_Y 0xd0
#define TSC_CMD_PLATEY 0x90

#define TSC_X_MAX 4096
#define TSC_Y_MAX 4096
#define TSC_X_MIN 0
#define TSC_Y_MIN 0

/* delay time for compute x, y, computed as us */

#define DEBUG
#ifdef DEBUG
#define TS_DEBUG(fmt,args...) printk(KERN_DEBUG fmt, ##args )
#else
#define TS_DEBUG(fmt,args...)
#endif
static int x_min=TSC_X_MIN;
static int y_min=TSC_Y_MIN;
static int x_max=TSC_X_MAX;
static int y_max=TSC_Y_MAX;
static int invert = 0;
static int debounce_time  = 150;
static int init_debounce = true;
static int delay_time = 1;

enum tsc2007_status {
    PEN_UP,
    PEN_DOWN,
};

struct _tsc2007 {
    struct input_dev *dev;
    int x;        /* X sample values */
```

```
    int y;      /* Y sample values */

    int status;
    struct work_struct irq_work;
    struct i2c_client *client;
    unsigned long last_touch;
};
struct _tsc2007 *g_tsc2007;

/* update abs params when min and max coordinate values are set */
int tsc2007_proc_minmax(struct ctl_table *table, int write, struct file *filp,
                    void __user *buffer, size_t *lenp, loff_t *ppos)
{
    struct _tsc2007 *tsc2007= g_tsc2007;
    struct input_dev *input = tsc2007->dev;

    /* update value */
    int ret = proc_dointvec(table, write, filp, buffer, lenp, ppos);

    /* updated abs params */
    if (input) {
        TS_DEBUG(KERN_DEBUG "update x_min %d x_max %d"
            " y_min %d y_max %d\n", x_min, x_max,
            y_min, y_max);
        input_set_abs_params(input, ABS_X, x_min, x_max, 0, 0);
        input_set_abs_params(input, ABS_Y, y_min, y_max, 0, 0);
    }
    return ret;
}

static ctl_table tsc2007_proc_table[] = {
    {
        .ctl_name    = CTL_UNNUMBERED,
        .procname    = "x-max",
        .data        = &x_max,
        .maxlen        = sizeof(int),
        .mode        = 0666,
        .proc_handler    = &tsc2007_proc_minmax,
    },
    {
        .ctl_name    = CTL_UNNUMBERED,
        .procname    = "y-max",
        .data        = &y_max,
        .maxlen        = sizeof(int),
        .mode        = 0666,
        .proc_handler    = &tsc2007_proc_minmax,
    },
    {
        .ctl_name    = CTL_UNNUMBERED,
        .procname    = "x-min",
        .data        = &x_min,
        .maxlen        = sizeof(int),
        .mode        = 0666,
        .proc_handler    = &tsc2007_proc_minmax,
```

```
    },
    {
        .ctl_name    = CTL_UNNUMBERED,
        .procname    = "y-min",
        .data        = &y_min,
        .maxlen      = sizeof(int),
        .mode        = 0666,
        .proc_handler    = &tsc2007_proc_minmax,
    },
    {
        .ctl_name    = CTL_UNNUMBERED,
        .procname    = "invert_xy",
        .data        = &invert,
        .maxlen      = sizeof(int),
        .mode        = 0666,
        .proc_handler    = &proc_dointvec,
    },
    {
        .ctl_name    = CTL_UNNUMBERED,
        .procname    = "debounce_time",
        .data        = &debounce_time,
        .maxlen      = sizeof(int),
        .mode        = 0666,
        .proc_handler    = &proc_dointvec,
    },
    {
        .ctl_name    = CTL_UNNUMBERED,
        .procname    = "delay_time",
        .data        = &delay_time,
        .maxlen      = sizeof(int),
        .mode        = 0666,
        .proc_handler    = &proc_dointvec,
    },
    { .ctl_name = 0 }
};

static ctl_table tsc2007_proc_root[] = {
    {
        .ctl_name    = CTL_UNNUMBERED,
        .procname    = "ts_device",
        .mode        = 0555,
        .child       = tsc2007_proc_table,
    },
    { .ctl_name = 0 }
};

static ctl_table tsc2007_proc_dev_root[] = {
    {
        .ctl_name    = CTL_DEV,
        .procname    = "dev",
        .mode        = 0555,
        .child       = tsc2007_proc_root,
    },
    { .ctl_name = 0 }
```

```
};

static struct ctl_table_header *sysctl_header;

static int __init init_sysctl(void)

    sysctl_header = register_sysctl_table(tsc2007_proc_dev_root);
    return 0;

static void __exit cleanup_sysctl(void)
{
    unregister_sysctl_table(sysctl_header);
}

static int tsc2007_measure(struct i2c_client *client, int *x, int * y)
{
    u8 x_buf = {0, 0};
    u8 y_buf = {0, 0};

    i2c_smbus_write_byte(client, TSC_CMD_PLATEX);
    msleep_interruptible(delay_time);

    i2c_smbus_write_byte(client, TSC_CMD_X);
    i2c_master_recv(client, x_buf, 2);
    *x = (x_buf<<4) | (x_buf >>4);

    i2c_smbus_write_byte(client, TSC_CMD_PLATEY);
    msleep_interruptible(delay_time);

    i2c_smbus_write_byte(client, TSC_CMD_Y);
    i2c_master_recv(client, y_buf, 2);
    *y = (y_buf<<4) | (y_buf >>4);
    *y = 4096 - *y; //added by allen
    printk("\ntouchscreen x = 0x%x, y = 0x%x\n",*x,*y);
    return 0;
}

static void tsc2007_irq_work(struct work_struct *work)
{
    struct _tsc2007 *tsc2007= g_tsc2007;
    struct i2c_client *client = tsc2007-> client;
    struct input_dev *input = tsc2007->dev;

    int x = -1, y = -1, is_valid = 0;
    int tmp_x = 0, tmp_y = 0;

    int gpio = irq_to_gpio(client->irq);


    /* Ignore if PEN_DOWN */
    if(PEN_UP == tsc2007->status){

        if (gpio_request(gpio, "tsc2007 touch detect")) {
```

```
    printk(KERN_ERR "Request GPIO failed, gpio: %X\n", gpio);
    return;
}
gpio_direction_input(gpio);

while(0 == gpio_get_value(gpio)){

        if ((jiffies_to_msecs(
            ((long)jiffies - (long)tsc2007->last_touch)) <
     debounce_time &&
    tsc2007->status == PEN_DOWN) ||
    init_debounce)
        {
    init_debounce = false;
            tsc2007_measure(client, &tmp_x, &tmp_y);
            TS_DEBUG(KERN_DEBUG
    "dropping pen touch %lu %lu (%u)\n",
            jiffies, tsc2007->last_touch,
            jiffies_to_msecs(
    (long)jiffies - (long)tsc2007->last_touch));
            schedule();
    continue;
        }


    /* continue report x, y */
    if (x > 0 && y > 0)
    {
        ts_linear_scale(&x, &y, invert);
        input_report_abs(input, ABS_X, x);
        input_report_abs(input, ABS_Y, y);
        input_report_abs(input, ABS_PRESSURE, 255);
        input_report_abs(input, ABS_TOOL_WIDTH, 1);
        input_report_key(input, BTN_TOUCH, 1);
        input_sync(input);
    }

    tsc2007->status = PEN_DOWN;
    tsc2007_measure(client, &x, &y);
    TS_DEBUG(KERN_DEBUG "pen down x=%d y=%d!\n", x, y);
    is_valid = 1;
    schedule();
}

if (is_valid)
{
    /*consider PEN_UP */
    tsc2007->status = PEN_UP;
    input_report_abs(input, ABS_PRESSURE, 0);
    input_report_abs(input, ABS_TOOL_WIDTH, 1);
    input_report_key(input, BTN_TOUCH, 0);
    input_sync(input);
    tsc2007->last_touch = jiffies;
    TS_DEBUG(KERN_DEBUG "pen up!\n");
```

```
        }

        gpio_free(gpio);
    }
}

static irqreturn_t tsc2007_interrupt(int irq, void *dev_id)
{
    schedule_work(&g_tsc2007->irq_work);

    return IRQ_HANDLED;
}

static int __devinit tsc2007_probe(struct i2c_client *client,
                const struct i2c_device_id *id)
{
    struct _tsc2007 *tsc2007;
    struct input_dev *input_dev;
    int ret;

    tsc2007 = kzalloc(sizeof(struct _tsc2007), GFP_KERNEL);
    input_dev = input_allocate_device();

    g_tsc2007 = tsc2007;

    if (!tsc2007 || !input_dev) {
        ret = -ENOMEM;
        goto fail1;
    }

    i2c_set_clientdata(client, tsc2007);

    tsc2007->dev = input_dev;

    input_dev->name = "tsc2007";
    input_dev->phys = "tsc2007/input0";

    //input_dev->id.bustype = BUS_HOST;
    input_dev->dev.parent = &client->dev;

    __set_bit(EV_KEY, input_dev->evbit);
    __set_bit(BTN_TOUCH, input_dev->keybit);

    __set_bit(EV_ABS, input_dev->evbit);
    __set_bit(ABS_PRESSURE, input_dev->evbit);
    __set_bit(ABS_X, input_dev->evbit);
    __set_bit(ABS_Y, input_dev->evbit);

    input_set_abs_params(input_dev, ABS_X, x_min, x_max, 0, 0);
    input_set_abs_params(input_dev, ABS_Y, y_min, y_max, 0, 0);
    input_set_abs_params(input_dev, ABS_PRESSURE, 0, 255, 0, 0);

    ret = request_irq(client->irq, tsc2007_interrupt,
        IRQF_DISABLED | IRQF_TRIGGER_FALLING,
```

```
        "tsc2007 irq", NULL);
    if (ret){
        printk(KERN_ERR "tsc2007 request irq failed\n");
        goto fail2;
    }

    ret = input_register_device(tsc2007->dev);
    if (ret){
        printk(KERN_ERR "tsc2007 register device fail\n");
        goto fail2;
    }

    /*init */
    tsc2007->status = PEN_UP;
    tsc2007->client = client;
    tsc2007->last_touch = jiffies;

    INIT_WORK(&tsc2007->irq_work, tsc2007_irq_work);

    /* init tsc2007 */
    i2c_smbus_write_byte(client, TSC_CMD_SETUP);

    return 0;

fail2:
    free_irq(client->irq, client);
fail1:
    i2c_set_clientdata(client, NULL);
    input_free_device(input_dev);
    kfree(tsc2007);
    return ret;
}

static int __devexit tsc2007_remove(struct i2c_client *client)
{
    struct _tsc2007 *tsc2007 = i2c_get_clientdata(client);

    if(client->irq)
        free_irq(client->irq, client);

    i2c_set_clientdata(client, NULL);
    input_unregister_device(tsc2007->dev);
    kfree(tsc2007);

    return 0;
}

static struct i2c_device_id tsc2007_idtable[] = {
    { "tsc2007", 0 },
    { }
};

MODULE_DEVICE_TABLE(i2c, tsc2007_idtable);
```

```
static struct i2c_driver tsc2007_driver = {
    .driver = {
        .name    = "tsc2007",
    },
    .id_table    = tsc2007_idtable,
    .probe       = tsc2007_probe,
    .remove      = __devexit_p(tsc2007_remove),
};

static int __init tsc2007_ts_init(void)
{
    init_sysctl();
    return i2c_add_driver(&tsc2007_driver);
}

static void __exit tsc2007_ts_exit(void)
{
    cleanup_sysctl();
    i2c_del_driver(&tsc2007_driver);
}

module_init(tsc2007_ts_init);
module_exit(tsc2007_ts_exit);

MODULE_DESCRIPTION("tsc2007 touch screen driver");
MODULE_LICENSE("GPL");
```

haolele 发表于 2011-11-11 21:07:48

ndroid系统开发之触摸屏tslib移植(内核)和原理分析

permike 2014-12-12 09:18:07 909

0

Android系统开发之触摸屏tslib移植(内核)和原理分析 本帖最后由 haolele 于 2011-11-11 21:07 编辑 Android系统开发之触摸屏tslib移植(内核)和...

ndroid系统开发(四)-触摸屏tslib移植(内核)和原理分析

首先了解一下tslib的运行原理，tslib的运行分成两部分(1)校验在LCD固定坐标位置依次显示出5个坐标让用户触摸，把LCD坐标和用户触摸时驱动屏驱动底层的坐标总共5组值保存起来运行tslib库的...

tomew 2011-03-30 22:07:00 1163

关于友善之臂Tiny210电容屏移植tslib

Embed_coder 2016-08-08 22:20:14 1596

Tiny210（Smart210）开发板移植tslib 1、前提 内核和根文件系统移植成功 tslib制作成功 准备好将要移植的Qtopia或Qt4 2、开始移植(以移植Qtopia为例) 将Qtop...

tslibonandroid

2018年03月23日 00:00

举报人: 被举报人: hclydao 举报的资源分: 3 *类型: *详细原因: 取 消 提 交 tslibonandroid 3积分立即下载 ...

tslib配置on Android

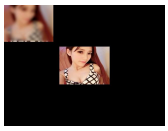
lnmcc 2012-11-20 18:01:15 316

1. export env export TSLIB_TSEVENTTYPE=INPUT export TSLIB_TSDEVICE=/dev/input/event4 export TS...

触摸屏tslib移植(内核)和原理分析

yueqian_scut 2015-04-16 15:23:17 1380

首先了解一下tslib的运行原理，tslib的运行分成两部分 (1)校验 在LCD固定坐标位置依次显示出5个坐标让用户触摸，把LCD坐标和用户触摸时驱动屏驱动底层的坐标总共5组值保存起来 运行t...



关于tslib移植usb触摸屏的一个笔记

u011003200 2013-11-25 15:44:24 916

开源upup 目前刚好有这样一个项目，搞了半天，确定了驱动没问题以后，我就找适配层tslib移植问题。 usb触摸屏是32寸，显示用的hdmi接口，板子用的A8（友善之臂的），图形界面用的是q...

android系统开发(四)-触摸屏tslib移植(内核)和原理分析

首先了解一下tslib的运行原理，tslib的运行分成两部分 (1)校验 在LCD固定坐标位置依次显示出5个坐标让用户触摸，把LCD坐标和用户触摸时驱动屏驱动底层的坐标总共5组值保存起来运行tsl...

starl1985 2012-04-16 16:09:48 2283

tslib android 参考帖

evilcode 2012-04-17 13:49:32 1562

转载1： TSLib ported to Android for touchscreen calibration. Since Android doesn't provide ...

smart210触摸屏tslib移植

2017年09月24日 19:50 3KB 下载

Android系统移植技术详解

ini2440一线触摸屏的tslib移植说明

bobbat 2016-05-23 22:52:19 1005

主机：ubuntu12.04，开发板：mini2440（TD35），交叉编译工具为4.4.3 在移植qt的过程中还算顺利，但是卡在了移植mini2440的talib，主要是它和其他的屏幕不一样，是...

tiny4412 tslib ts_test显示界面，但触摸无反应，运行qt4命令触摸有反应

今日刚买了块tiny4412+hd700开发板，想设计一个4412+12寸电容屏显示控制，想通过开发板验证自己设计的12寸电容屏好坏，以前都是用电阻屏，没有玩过电容屏，那就按照电阻屏的测试方法进行，就...

nawei87 2015-06-15 10:27:11 1206

友善之臂 tslib一线触摸完美移植

ESD1406 2015-01-30 16:29:04 386

开发板：友善之臂 mini210 博主曾耗费两三天移植tslib，都没有成功，各种失败，各种报错。 经过本人对 友善之臂 /etc/init.d/rcS (PS：该文件是文件系统初始化文件) 严格...

移植好lcd驱动和触摸屏驱动后，运行tslib库中的测试程序出现如下问题

1：问题： /bin # ./ts_calibrate ，lcd屏幕上显示软件界面，但是显示不完整 点击光标乱跑,正常的话在顶端左上角出现十字架光标，2次点击会在顶端右上角出现十字架光标，可是在...

sanmaoljh 2014-08-15 21:41:25 709

在tiny4412上移植12寸触摸屏（电容屏）/液晶屏（LCD）

其实只改了LCD的分辨率参数，我的LCD屏为1280x768，首先查看内核默认选中哪个LCD屏参数，例如选中了HD701，那么就在HD701中进行修改分辨率即可。就改了分辨率行，场。...

nawei87 2015-07-02 15:25:50 1832