


TIANXIAWUZHEI_LINUX中触摸屏驱动的实现——基于S3C6410处理器

 2012-10-31 11:45, Tags: 166人阅读

这几篇文章主要是关于linux中触摸屏驱动的，基于s3c6410处理器进行分析。这一篇主要是关于触摸屏设备作为平台设备的实现，还有对应的probe函数和remove函数的源码分析。

1、触摸屏模块的加载和卸载函数

```
static char banner[] __initdata = KERN_INFO "S3C Touchscreen driver, (c) 20

static int __init s3c_ts_init(void)
{
    printk(banner);
    return platform_driver_register(&s3c_ts_driver);
}
static void __exit s3c_ts_exit(void)
{
    platform_driver_unregister(&s3c_ts_driver);
}
module_init(s3c_ts_init);
module_exit(s3c_ts_exit);
```

万变不离其宗，还是熟悉的那个他，只不过每一次都是一番新的历程。
对应的平台设备资源：在Dev-ts.c (linux2.6.28\arch\arm\plat-s3c)文件中

```
/* Touch screen */
static struct resource s3c_ts_resource[] = {
    [0] = {
        .start = S3C_PA_ADC, I/O端口
        .end   = S3C_PA_ADC + SZ_4K - 1,
        .flags = IORESOURCE_MEM,
    },
    [1] = {
        .start = IRQ_PENDN, 中断
        .end   = IRQ_PENDN,
        .flags = IORESOURCE_IRQ,
    },
    [2] = {
        .start = IRQ_ADC,  中断
        .end   = IRQ_ADC,
        .flags = IORESOURCE_IRQ,
    }
};

struct platform_device s3c_device_ts = {
    .name = "s3c-ts",
    .id   = -1,
    .num_resources = ARRAY_SIZE(s3c_ts_resource),
    .resource  = s3c_ts_resource,
};
```

对应的platform_driver结构体的定义如下：

```
static struct platform_driver s3c_ts_driver = {
    .probe      = s3c_ts_probe,
    .remove     = s3c_ts_remove,
    .suspend    = s3c_ts_suspend,
    .resume     = s3c_ts_resume,
    .driver     = {
        .owner  = THIS_MODULE,
```

内容分类

LCD
Flash RAM
Camera
No OS
Linux 应用程序编程
Bug
MCU
Hardware
X210
u-boot
Qt
Linux设备驱动
Linux系统管理
touchscreen
Android
相关学习资源
Linux-kernel
Android_USB摄像头
Maliit
C/C++
Kconfig
gcc/arm-linux-gcc
ARM系统
Linux内核模块
ubifs/file system
usb

近期文章

单片机串口调试丢包验证过程记录_已解决
Android中间件开发---Windows下Android环境搭建（最新最方便）
u-boot环境变量配置记录
Hi-Z(高阻态)
Qt库编译出现: qconfig.cpp: 新的权限为 r-xrwxrwx, 而非 r-xr-xr-x Creating qmake. Please wait...
Android系统中SD卡各文件夹名称功能详解
x210的u-boot中的mtddpart
linux串口通信mark问题
6410中LCD配置的宏定义所在位置
error:Target dll has been cancelled.debugger aborted

热评文章

```
.name = "s3c-ts",
},
};
```

2、我想应该知道要做什么了，接着来看probe函数，源码如下：

```
/*
 * The functions for inserting/removing us as a module.
 */
static int __init s3c_ts_probe(struct platform_device *pdev)
{
    struct resource *res;
    struct device *dev;
    struct input_dev *input_dev;
    struct s3c_ts_mach_info * s3c_ts_cfg;
    int ret, size;

    dev = &pdev->dev;

    res = platform_get_resource(pdev, IORESOURCE_MEM, 0);
    if (res == NULL) {
        dev_err(dev,"no memory resource specified\n");
        return -ENOENT;
    }    得到寄存器操作地址

    size = (res->end - res->start) + 1;
    ts_mem = request_mem_region(res->start, size, pdev->name);    申请这片内存区域
    注: static struct resource*ts_mem;
    if (ts_mem == NULL) {
        dev_err(dev, "failed to get memory region\n");
        ret = -ENOENT;
        goto err_req;
    }

    ts_base = ioremap(res->start, size);    进行映射
    if (ts_base == NULL) {
        dev_err(dev, "failed to ioremap() region\n");
        ret = -EINVAL;
        goto err_map;
    }

    ts_clock = clk_get(&pdev->dev, "adc");    得到时钟
    if (IS_ERR(ts_clock)) {
        dev_err(dev, "failed to find watchdog clock source\n");
        ret = PTR_ERR(ts_clock);
        goto err_clk;
    }

    clk_enable(ts_clock);    使能时钟

    s3c_ts_cfg = s3c_ts_get_platdata(&pdev->dev);
    static struct s3c_ts_mach_info *s3c_ts_get_platdata (struct device *dev)
    {
        if (dev->platform_data != NULL)
            return (struct s3c_ts_mach_info *)dev->platform_data;

        return &s3c_ts_default_cfg;
    }
```

默认值：

```
/* Touchscreen default configuration */
struct s3c_ts_mach_info s3c_ts_default_cfg __initdata = {
    .delay = 5000, //10000,
    .presc = 49,
    .oversampling_shift = 4, //2,
    .resol_bit = 10
};
```

mini2440的串口在Qt上实现

Android中间件开发---Windows下
Android环境搭建（最新最方便）

NAND Flash SLC、MLC技术解析

文件编程：创建目录mkdir()函数

时间编程：
time(),localtime(),gmtime(),asctime(),ctime(),gettime

进程控制：waitpid之status意义解析

进程控制：waitpid()

进程控制：linux中fork同时创建多个子
进程注意事项

进程通信：浅析Linux下core文件

进程通信：信号通信中的SIGABRT的验证
测试

内容归档

2013年03月

2013年02月

2013年01月

2012年12月

2012年11月

2012年10月

2012年09月

这里涉及到一个结构体s3c_ts_mach_info

```
struct s3c_ts_mach_info {
int          delay;   延时时间
int          presc;   预分频值
int          oversampling_shift; 转化次数
int resol_bit; 分频率
enum s3c_adc_types3c_adc_con;看下面:
};
```

其中有

```
enum s3c_adc_type {
ADC_TYPE_0,
ADC_TYPE_1, /* S3C2416, S3C2450 */
ADC_TYPE_2,/* S3C64XX, S5PC1XX */
};
```

```
if ((s3c_ts_cfg->presc&0xff) > 0)    设置预分频值
write1(S3C_ADCCON_PRSCEN | S3C_ADCCON_PRSCVL(s3c_ts_cfg->presc&0xFF),\
ts_base+S3C_ADCCON);
else
```

writel(0, ts_base+S3C_ADCCON);没有定义的话，写0，其实也就是禁止预分频

这里主要和ADCCON寄存器的设置有关，而且有如下定义：

```
#define S3C_ADCCON_PRSCEN(1<<14)
#define S3C_ADCCON_PRSCVL(x)((x)&0xFF)<<6)
```

看下图：

PRSCEN	[14]	A/D converter <u>prescaler enable</u> 0 = Disable 1 = Enable
PRSCVL	[13:6]	A/D converter <u>prescaler value</u> Data value: 5 ~ 255 NOTE: Note that division factor is (N+1) when the prescaler value is N. ADC frequency should be set less than PCLK by 5 times. (Ex. If PCLK=10MHz, ADC Frequency<2MHz) This A/D converter is designed to operate at maximum 5MHz clock

```
/* Initialise registers */
if ((s3c_ts_cfg->delay&0xffff) > 0)
writel(s3c_ts_cfg->delay & 0xffff, ts_base+S3C_ADCDLY);
和上面差不多，主要和ADC DLY寄存器有关。直接看图：注：在两种模式下有不同的含义
```

DELAY	[15:0]	1) In case of ADC conversion mode (Normal, Separate, Auto conversion); ADC conversion is delayed by counting this value. Counting clock is PCLK. → ADC conversion delay value. 2) In case of waiting for Interrupt mode; when stylus down occurs in waiting for interrupt mode, it generates interrupt signal (INT_PNDNUP) at interval of several ms for Auto X/Y position conversion. If this interrupt occurs in STOP mode, it generates Wake-Up signal, having interval (several ms), for Exiting STOP MODE. Note: Do not use Zero value(0x0000)
-------	--------	---

```
if (s3c_ts_cfg->resol_bit==12) { 分频率
```

```
switch(s3c_ts_cfg->s3c_adc_con) {
case ADC_TYPE_2:
writel(readl(ts_base+S3C_ADCCON)|S3C_ADCCON_RESSEL_12BIT,
ts_base+S3C_ADCCON);
break;

#define S3C_ADCCON_RESSEL_12BIT(0x1<<16)
```

ADCCON	Bit	Description
RESSEL	[16]	A/D converter resolution selection 0 = 10-bit A/D conversion 1 = 12-bit A/D conversion

```
case ADC_TYPE_1:
writel(readl(ts_base+S3C_ADCCON)|S3C_ADCCON_RESSEL_12BIT_1,
ts_base+S3C_ADCCON);
break;

default:
dev_err(dev, "Touchscreen over this type of AP isn't supported !\n");
break;
}
}
```

writel(WAIT4INT(0), ts_base+S3C_ADCTSC);主要是对ADCTSC寄存器进行操作，使触摸屏处于等待中断模式

Register	Address	R/W	Description
ADCTSC	0x7E00B004	R/W	ADC Touch Screen Control Register

```
ts = kzalloc(sizeof(struct s3c_ts_info), GFP_KERNEL);

注： static struct s3c_ts_info*ts;
```

input_dev = input_allocate_device();申请并初始化一个输入设备。通过输入设备，驱动程序才能和用户交互。
注： struct input_dev *input_dev;

```
if (!input_dev) {
ret = -ENOMEM;
goto err_alloc;
}

ts->dev = input_dev;

ts->dev->evbit[0] = ts->dev->evbit[0] = BIT_MASK(EV_SYN) | BIT_MASK(EV_KEY)
ts->dev->keybit[BIT_WORD(BTN_TOUCH)] = BIT_MASK(BTN_TOUCH);

if (s3c_ts_cfg->resol_bit==12) {
input_set_abs_params(ts->dev, ABS_X, 0, 0xFFF, 0, 0);
input_set_abs_params(ts->dev, ABS_Y, 0, 0xFFF, 0, 0);
}
else {
input_set_abs_params(ts->dev, ABS_X, 0, 0x3FF, 0, 0);
input_set_abs_params(ts->dev, ABS_Y, 0, 0x3FF, 0, 0);
}

input_set_abs_params(ts->dev, ABS_PRESSURE, 0, 1, 0, 0);

sprintf(ts->phys, "input(ts)");
```

```

ts->dev->name = s3c_ts_name;
ts->dev->phys = ts->phys;
ts->dev->id.bustype = BUS_RS232;
ts->dev->id.vendor = 0xDEAD;
ts->dev->id.product = 0xBEEF;
ts->dev->id.version = S3C_TSVERSION;

ts->shift = s3c_ts_cfg->oversampling_shift;
ts->resol_bit = s3c_ts_cfg->resol_bit;
ts->s3c_adc_con = s3c_ts_cfg->s3c_adc_con;

```

上面这一段代码都是初始化触摸屏设备的全局量ts,对应的结构体原型是:

```

struct s3c_ts_info {
struct input_dev *dev;
long xp;
long yp;
int count;
int shift;
char phys[32];
int resol_bit;
enum s3c_adc_types3c_adc_con;
};

/* For IRQ_PENDUP */
ts_irq = platform_get_resource(pdev, IORESOURCE_IRQ, 0); 得到触摸屏中断IRQ_P
if (ts_irq == NULL) {
dev_err(dev, "no irq resource specified\n");
ret = -ENOENT;
goto err_irq;
}

ret = request_irq(ts_irq->start, stylus_updown, IRQF_SAMPLE_RANDOM, "s3c_up
if (ret != 0) {
dev_err(dev, "s3c_ts.c: Could not allocate ts IRQ_PENDN !\n");
ret = -EIO;
goto err_irq;
}

/* For IRQ_ADC */
ts_irq = platform_get_resource(pdev, IORESOURCE_IRQ, 1); 得到ADC中断
if (ts_irq == NULL) {
dev_err(dev, "no irq resource specified\n");
ret = -ENOENT;
goto err_irq;
}

ret = request_irq(ts_irq->start, stylus_action, IRQF_SAMPLE_RANDOM, "s3c_ac
if (ret != 0) {
dev_err(dev, "s3c_ts.c: Could not allocate ts IRQ_ADC !\n");
ret = -EIO;
goto err_irq;
}

printk(KERN_INFO "%s got loaded successfully : %d bits\n", s3c_ts_name, s3c

/* All went ok, so register to the input system */ 将触摸屏设备注册到输入子系:
ret = input_register_device(ts->dev);

if(ret) {
dev_err(dev, "s3c_ts.c: Could not register input device(touchscreen)!\n");
ret = -EIO;
goto fail;
}

```

```
return 0;
```

下面这些是错误处理代码

```
fail:
free_irq(ts_irq->start, ts->dev);
free_irq(ts_irq->end, ts->dev);
```

```
err_irq:
input_free_device(input_dev);
kfree(ts);
```

```
err_alloc:
clk_disable(ts_clock);
clk_put(ts_clock);
```

```
err_clk:
iounmap(ts_base);
```

```
err_map:
release_resource(ts_mem);
kfree(ts_mem);
```

```
err_req:
return ret;
}
```

到这里，触摸屏设备驱动的probe函数就讲述完了。

3、当然，probe函数中几个重要的函数都没讲，就是关于输入子系统的，那不是我们现在关注的重点。接着看对应的remove函数,源码如下：

```
static int s3c_ts_remove(struct platform_device *dev)
{
printk(KERN_INFO "s3c_ts_remove() of TS called !\n");
```

```
disable_irq(IRQ_ADC);
disable_irq(IRQ_PENDN);
```

```
free_irq(IRQ_PENDN, ts->dev);
free_irq(IRQ_ADC, ts->dev);
```

```
if (ts_clock) {
clk_disable(ts_clock);
clk_put(ts_clock);
ts_clock = NULL;
}
```

```
input_unregister_device(ts->dev);
iounmap(ts_base);
```

```
return 0;
}
```

其实看懂了probe函数，remove函数就完全不用看了。

linux中触摸屏驱动的实现（2）——基于s3c6410处理器的链接地址

上一篇主要讲述了linux中触摸屏设备作为平台设备存在的模块加载和卸载函数，还有就是对应的probe函数和remove函数，这一篇说下在probe函数中注册的两个中断处理函数。

1、先来说第一个中断处理函数——触摸屏中断，对应的中断处理函数是stylus_updown，当触摸屏被按下时，会产生中断信号IRQ_PENDUP。函数源码如

下:

```
static irqreturn_t stylus_updown(int irqno, void *param)
{
    unsigned long data0;
    unsigned long data1;
    int updown;定义一个整型变量，用来表示触摸屏是否被按下，如果按下，这个值是1；如果没按

    data0 = readl(ts_base+S3C_ADCDAT0);
    data1 = readl(ts_base+S3C_ADCDAT1);读取寄存器ADCDAT0和寄存器ADCDAT1

    updown = (!(data0 & S3C_ADCDAT0_UPDOWN)) && (!(data1 & S3C_ADCDAT1_UPDOWN))
```

重点来分析，有如下定义：

```
#define S3C_ADCDAT0_UPDOWN(1<<15)
#define S3C_ADCDAT1_UPDOWN(1<<15)
```

从这里可知与这两个寄存器的第15位有关，而寄存器ADCDAT0和寄存器ADCDAT1分别表示X和Y方向检测到触摸屏是否被按下，也就是说只有当寄存器ADCDAT0和寄存器ADCDAT1两个寄存器的UPDOWN都等于0时，采表示触摸屏被按下。看下面这个图：

ADCDAT0	Bit	Description
UPDOWN	[15]	Up or Down state of Stylus at Waiting for Interrupt Mode. 0 = Stylus down state. 1 = Stylus up state.

ADCDAT1	Bit	Description
UPDOWN	[15]	Up or Down state of Stylus at Waiting for Interrupt Mode. 0 = Stylus down state. 1 = No stylus down state.

```
#ifdef CONFIG_TOUCHSCREEN_S3C_DEBUG
    printk(KERN_INFO "   %c\n",updown ? 'D' : 'U');
#endif

/* TODO we should never get an interrupt with updown set while
 * the timer is running, but maybe we ought to verify that the
 * timer isn't running anyways. */
if (updown)    updown  等于1，表示触摸屏按下
{
    downflag=1;
    //printk("touch_timer_fire(0)\n");
    touch_timer_fire(0);调用此函数处理触摸屏的按下，这个函数下面再讲。
}

if(ts->s3c_adc_con==ADC_TYPE_2) {
    __raw_writel(0x0, ts_base+S3C_ADCCLRWK);
    __raw_writel(0x0, ts_base+S3C_ADCCLRINT);
}
```

其中有如下定义：

```
#define S3C_ADCCLRINTS3C_ADCREG(0x18)
#define S3C_ADCCLRWKS3C_ADCREG(0x20)
```

直接看图：

Register	Address	R/W	Description
ADCCLRINT	0x7E00B018	W	Clear ADC Interrupt

ADCCLRINT	Bit	Description
INT_ADC_CLR	[0]	INT_ADC interrupt clear

Register	Address	R/W	Description
ADCCLRINTPNDNUP	0x7E00B020	W	Clear Pen Down/Up Interrupt

ADCCLRINTPNDNUP	Bit	Description
INT_PNDNUP_CLR	[0]	INT_PNDNUP interrupt clear

```
return IRQ_HANDLED;
}
```

好了现在可以分析我们上面没有分析的那一个函数了touch_timer_fire，源码如下：

```
static void touch_timer_fire(unsigned long data)
{
    unsigned long data0;
    unsigned long data1;
    int updown;

    data0 = readl(ts_base+S3C_ADCDAT0);
    data1 = readl(ts_base+S3C_ADCDAT1);

    updown = (!(data0 & S3C_ADCDAT0_UPDOWN)) && (!(data1 & S3C_ADCDAT1_UPDOWN))
    上面这些和刚才分析的一样，都是为了判断触摸屏是否被按下

    if (updown) {  为1，触摸屏被按下
        //printk("updown=1.\n");
        if (ts->count) {

            #ifdef CONFIG_TOUCHSCREEN_S3C_DEBUG
            .....
            #endif
            if(downflag==0)
            {
                input_report_abs(ts->dev, ABS_X, ts->xp);
                input_report_abs(ts->dev, ABS_Y, ts->yp);

                input_report_key(ts->dev, BTN_TOUCH, 1);
                input_report_abs(ts->dev, ABS_PRESSURE, 1);
                input_sync(ts->dev);
            }    上面这一段用来向输入子系统报告当前触摸笔的位置
            else
            {
                // printk("downflag=1.ignore this data.\n");

                downflag=0;
            }
        }
        ts->xp = 0;
        ts->yp = 0;
        ts->count = 0;
        表示缓冲区中没有数据，也就是没有触摸屏按下时间发生

        writel(S3C_ADCTSC_PULL_UP_DISABLE | AUTOPST, ts_base+S3C_ADCTSC);主要是将ADP
        其中有如下定义：
        #define S3C_ADCTSC_PULL_UP_DISABLE(1<<3)
        #define AUTOPST    (S3C_ADCTSC_YM_SEN | S3C_ADCTSC_YP_SEN | S3C_ADCTSC_XP_S
            S3C_ADCTSC_AUTO_PST | S3C_ADCTSC_XY_PST(0))

        writel(readl(ts_base+S3C_ADCCON) | S3C_ADCCON_ENABLE_START, ts_base+S3C_ADC
        }
        else {    表示触摸屏没有被按下时的操作。

            ts->count = 0;
            input_report_key(ts->dev, BTN_TOUCH, 0);调用这个函数向输入子系统报告触摸屏被弹起
            input_report_abs(ts->dev, ABS_PRESSURE, 0); 发送触摸屏的一个绝对坐标
            input_sync(ts->dev);  该函数通知事件发送者发送一个完整的报告。

            writel(WAIT4INT(0), ts_base+S3C_ADCTSC); 把触摸屏的模式设为等待中断模式
        }
    }
}
```


}

linux中触摸屏驱动的实现（3）——基于s3c6410处理器的链接地址

1、上一篇分析的是两个中断处理函数中的其中一个触摸屏中断，现在来分析另外一个ADC中断，对应的中断函数是stylus_action。当触摸屏在自动X/Y位置转换模式和独立的X/Y位置转换模式时，当坐标数据转换之后会产生IRQ_ADC中断，进而调用stylus_action函数，此函数源码如下：

```
static irqreturn_t stylus_action(int irqno, void *dev_id)
{
    unsigned long data0;
    unsigned long data1;

    //printk("stylus_action.\n");
    data0 = readl(ts_base+S3C_ADCDAT0);
    data1 = readl(ts_base+S3C_ADCDAT1);    读两个寄存器

    if(ts->resol_bit==12) {        12位分辨率
        #if defined(CONFIG_TOUCHSCREEN_NEW)
            ts->yp += S3C_ADCDAT0_XPDATA_MASK_12BIT - (data0 & S3C_ADCDAT0_XPDATA_MASK_12BIT);
            ts->xp += S3C_ADCDAT1_YPDATA_MASK_12BIT - (data1 & S3C_ADCDAT1_YPDATA_MASK_12BIT);
        #else
            ts->xp += data0 & S3C_ADCDAT0_XPDATA_MASK_12BIT;
            ts->yp += data1 & S3C_ADCDAT1_YPDATA_MASK_12BIT;
        #endif
    }
    else {        10位分辨率
        #if defined(CONFIG_TOUCHSCREEN_NEW)
            ts->yp += S3C_ADCDAT0_XPDATA_MASK - (data0 & S3C_ADCDAT0_XPDATA_MASK);
            ts->xp += S3C_ADCDAT1_YPDATA_MASK - (data1 & S3C_ADCDAT1_YPDATA_MASK);
        #else
            ts->xp += data0 & S3C_ADCDAT0_XPDATA_MASK;
            ts->yp += data1 & S3C_ADCDAT1_YPDATA_MASK;
        #endif
    }

    ts->count++;

    if (ts->count < (1<<ts->shift)) {        如果缓冲区
        writel(S3C_ADCTSC_PULL_UP_DISABLE | AUTOPST, ts_base+S3C_ADCTSC);
        writel(readl(ts_base+S3C_ADCCON) | S3C_ADCCON_1, ts_base+S3C_ADCCON);
    } else {
        mod_timer(&touch_timer, jiffies+1);    修改touch_
```

touch_timer定时器指定的函数。这个定时器的定义如下：

```
static struct timer_list touch_timer =
TIMER_INITIALIZER(touch_timer_fire, 0, 0);
writel(WAIT4INT(1), ts_base+S3C_ADCTSC);    将角
}

if(ts->s3c_adc_con==ADC_TYPE_2) {    清中断
    __raw_writel(0x0, ts_base+S3C_ADCCLRWK)
    __raw_writel(0x0, ts_base+S3C_ADCCLRIN`
}

return IRQ_HANDLED;
}
```

touch_timer定时器用来当缓冲区不为空时，不断地触发touch_timer_fire函数。此函数读取触摸屏的坐标信息，并传递给内核输入子系统。还记得吗？touch_timer_fire这个函数的源码在上一篇博客中已经分析了。

2、再说下上面的定时器定义函数。

```
static struct timer_list touch_timer =
TIMER_INITIALIZER(touch_timer_fire, 0, 0);
```

看下面，可知触摸屏设备驱动程序将touch_timer定时器函数设置为touch_timer_fire，过期时间为0，数据为0。即加载完触摸屏驱动程序后，就会执行一次定时器函数touch_timer_fire。

```
#define TIMER_INITIALIZER(_function, _expires,
    .entry = { .prev = TIMER_ENTRY_STATIC },\
    .function = (_function),\
    .expires = (_expires),\
    .data = (_data),\
    .base = &boot_tvec_bases,\
}
```