# Linux驱动\_触摸屏

## 1.触摸屏初始化

触摸屏驱动遵循了platform驱动注册的流程。

### 1.1 platform device注册

在smdk2440\_machine\_init函数中，会注册触摸屏设备。相关定义如下：

struct platform\_device s3c\_device\_ts **=** **{**

**.**name **=** "s3c2410-ts"**,**

**.**id **=** **-**1**,**

**};**

static struct s3c2410\_ts\_mach\_info s3c2410\_ts\_cfg **=** **{**

**.**delay **=** 10000**,**

**.**presc **=** 49**,**

**.**oversampling\_shift **=** 2**,**

**};**

void \_\_init set\_s3c2410ts\_info**(**struct s3c2410\_ts\_mach\_info **\***hard\_s3c2410ts\_info**)**

**{**

memcpy**(&**s3c2410ts\_info**,** hard\_s3c2410ts\_info**,** **sizeof(**struct s3c2410\_ts\_mach\_info**));**

s3c\_device\_ts**.**dev**.**platform\_data **=** **&**s3c2410ts\_info**;**

**}**

set\_s3c2410ts\_info**(&**s3c2410\_ts\_cfg**);**

### 1.2 platform driver注册

static int \_\_init s3c2410ts\_init**(**void**)**

**{**

**return** platform\_driver\_register**(&**s3c2410ts\_driver**);**

**}**

static struct platform\_driver s3c2410ts\_driver **=** **{**

**.**driver **=** **{**

**.**name **=** "s3c2410-ts"**,**

**.**owner **=** THIS\_MODULE**,**

**},**

**.**probe **=** s3c2410ts\_probe**,**

**.**remove **=** s3c2410ts\_remove**,**

**};**

### 1.3 probe函数

当触摸屏的platform device和platform driver匹配后，probe函数调用。

probe函数主要完成的功能如下：

(1)设置相应的gpio口为touchscreen用途并配置好相关寄存器。

(2)向输入子系统注册input device。

(3)申请ADC和TC中断。

(4)进入等待中断模式。

static int \_\_init s3c2410ts\_probe**(**struct platform\_device **\***pdev**)**

**{**

struct s3c2410\_ts\_mach\_info **\***info**;**

struct input\_dev **\***input\_dev**;**

info **=** **(** struct s3c2410\_ts\_mach\_info **\*)**pdev**->**dev**.**platform\_data**;**

adc\_clock **=** clk\_get**(NULL,** "adc"**);**

clk\_enable**(**adc\_clock**);**

//ADC寄存器映射为虚拟地址

base\_addr**=**ioremap**(**S3C2410\_PA\_ADC**,**0x20**);**

//设置gpg12~15为触摸屏接口

s3c2410\_ts\_connect**();**

//分频系数49，设置对应的寄存器，adc采样的速率

**if** **((**info**->**presc**&**0xff**)** **>** 0**)**

iowrite32**(**S3C2410\_ADCCON\_PRSCEN **|** S3C2410\_ADCCON\_PRSCVL**(**info**->**presc**&**0xFF**),\**

base\_addr**+**S3C2410\_ADCCON**);**

**else**

iowrite32**(**0**,**base\_addr**+**S3C2410\_ADCCON**);**

//设置延时，开始adc采样等待一会才稳定

**if** **((**info**->**delay**&**0xffff**)** **>** 0**)**

iowrite32**(**info**->**delay **&** 0xffff**,** base\_addr**+**S3C2410\_ADCDLY**);**

//进入等待按下中断模式

//While waiting for Touch screen Interrupt, XP\_SEN bit should be set to ‘1’(XP Output disable) and PULL\_UP bit should

//be set to ‘0’(XP Pull-up enable).

iowrite32**(**WAIT4INT**(**0**),** base\_addr**+**S3C2410\_ADCTSC**);**

//初始化s3c2410ts结构体

memset**(&**ts**,** 0**,** **sizeof(**struct s3c2410ts**));**

//分配input device

input\_dev **=** input\_allocate\_device**();**

//设置input device会产生的事件

ts**.**dev **=** input\_dev**;**

//注册事件为SYNC/KEY/ABS

ts**.**dev**->**evbit**[**0**]** **=** BIT**(**EV\_SYN**)** **|** BIT**(**EV\_KEY**)** **|** BIT**(**EV\_ABS**);**

//KEY的类型为BTN\_TOUCH

ts**.**dev**->**keybit**[**LONG**(**BTN\_TOUCH**)]** **=** BIT**(**BTN\_TOUCH**);**

//绝对坐标X,最大值0x3FF

input\_set\_abs\_params**(**ts**.**dev**,** ABS\_X**,** 0**,** 0x3FF**,** 0**,** 0**);**

//绝对位移Y,最大值0x3FF

input\_set\_abs\_params**(**ts**.**dev**,** ABS\_Y**,** 0**,** 0x3FF**,** 0**,** 0**);**

//压力值，最大值1

input\_set\_abs\_params**(**ts**.**dev**,** ABS\_PRESSURE**,** 0**,** 1**,** 0**,** 0**);**

//设置input device的额外信息

ts**.**dev**->**private **=** **&**ts**;**

ts**.**dev**->**name **=** s3c2410ts\_name**;**

ts**.**dev**->**id**.**bustype **=** BUS\_RS232**;**

ts**.**dev**->**id**.**vendor **=** 0xDEAD**;**

ts**.**dev**->**id**.**product **=** 0xBEEF**;**

ts**.**dev**->**id**.**version **=** S3C2410TSVERSION**;**

ts**.**shift **=** info**->**oversampling\_shift**;**

//申请中断ADC,adc转换成功操作

**if** **(**request\_irq**(**IRQ\_ADC**,** stylus\_action**,** IRQF\_SAMPLE\_RANDOM **|** SA\_SHIRQ**,**

"s3c2410\_action"**,** ts**.**dev**))** **{**

printk**(**KERN\_ERR "s3c2410\_ts.c: Could not allocate ts IRQ\_ADC !\n"**);**

iounmap**(**base\_addr**);**

**return** **-**EIO**;**

**}**

//申请中断TC，触摸屏按下弹起操作

**if** **(**request\_irq**(**IRQ\_TC**,** stylus\_updown**,** IRQF\_SAMPLE\_RANDOM**,**

"s3c2410\_action"**,** ts**.**dev**))** **{**

printk**(**KERN\_ERR "s3c2410\_ts.c: Could not allocate ts IRQ\_TC !\n"**);**

iounmap**(**base\_addr**);**

**return** **-**EIO**;**

**}**

printk**(**KERN\_INFO "%s successfully loaded\n"**,** s3c2410ts\_name**);**

//向输入子系统注册input device

input\_register\_device**(**ts**.**dev**);**

**return** 0**;**

**}**

## 2. 触摸屏相关的输入子系统

### 2.1 evdev

任何注册到输入子系统的device都会和evdev handler关联起来，会生成/dev/input/eventx设备。

通过读取/dev/input/eventx可以得到触摸屏产生的键值和坐标值等。

### 2.2 tsdev

触摸屏除了会和evdev handler联系起来，还会和tsdev handler联系起来。

#### 2.2.1 tsdev handler注册

static struct input\_handler tsdev\_handler **=** **{**

**.**event **=** tsdev\_event**,**

**.**connect **=** tsdev\_connect**,**

**.**disconnect **=** tsdev\_disconnect**,**

**.**fops **=** **&**tsdev\_fops**,**

**.**minor **=** TSDEV\_MINOR\_BASE**,**

**.**name **=** "tsdev"**,**

**.**id\_table **=** tsdev\_ids**,**

**};**

static const struct input\_device\_id tsdev\_ids**[]** **=** **{**

**{**

**.**flags **=** INPUT\_DEVICE\_ID\_MATCH\_EVBIT **|** INPUT\_DEVICE\_ID\_MATCH\_KEYBIT **|** INPUT\_DEVICE\_ID\_MATCH\_RELBIT**,**

**.**evbit **=** **{** BIT**(**EV\_KEY**)** **|** BIT**(**EV\_REL**)** **},**

**.**keybit **=** **{** **[**LONG**(**BTN\_LEFT**)]** **=** BIT**(**BTN\_LEFT**)** **},**

**.**relbit **=** **{** BIT**(**REL\_X**)** **|** BIT**(**REL\_Y**)** **},**

**},** /\* A mouse like device, at least one button, two relative axes \*/

**{**

**.**flags **=** INPUT\_DEVICE\_ID\_MATCH\_EVBIT **|** INPUT\_DEVICE\_ID\_MATCH\_KEYBIT **|** INPUT\_DEVICE\_ID\_MATCH\_ABSBIT**,**

**.**evbit **=** **{** BIT**(**EV\_KEY**)** **|** BIT**(**EV\_ABS**)** **},**

**.**keybit **=** **{** **[**LONG**(**BTN\_TOUCH**)]** **=** BIT**(**BTN\_TOUCH**)** **},**

**.**absbit **=** **{** BIT**(**ABS\_X**)** **|** BIT**(**ABS\_Y**)** **},**

**},** /\* A tablet like device, at least touch detection, two absolute axes \*/

**{**

**.**flags **=** INPUT\_DEVICE\_ID\_MATCH\_EVBIT **|** INPUT\_DEVICE\_ID\_MATCH\_ABSBIT**,**

**.**evbit **=** **{** BIT**(**EV\_ABS**)** **},**

**.**absbit **=** **{** BIT**(**ABS\_X**)** **|** BIT**(**ABS\_Y**)** **|** BIT**(**ABS\_PRESSURE**)** **},**

**},** /\* A tablet like device with several gradations of pressure \*/

**{}** /\* Terminating entry \*/

**};**

static int \_\_init tsdev\_init**(**void**)**

**{**

**return** input\_register\_handler**(&**tsdev\_handler**);**

**}**

从tsdev的ids进行分析，发现触摸屏input device和tsdev handler是匹配的，所以tsdev\_connect会调用。

#### 2.2.2 tsdev\_connect

tsdev\_connect主要完成的功能如下：  
(1)创建一个新的tsdev结构体会注册到tsdev\_table中。

(2)创建/sys/class/input/tsx目录，方便udev创建设备文件/dev/input/tsx。

static int tsdev\_connect**(**struct input\_handler **\***handler**,** struct input\_dev **\***dev**,**

const struct input\_device\_id **\***id**)**

**{**

struct tsdev **\***tsdev**;**

struct class\_device **\***cdev**;**

dev\_t devt**;**

int minor**,** delta**;**

int error**;**

**for** **(**minor **=** 0**;** minor **<** TSDEV\_MINORS **/** 2 **&&** tsdev\_table**[**minor**];** minor**++);**

**if** **(**minor **>=** TSDEV\_MINORS **/** 2**)** **{**

printk**(**KERN\_ERR

"tsdev: You have way too many touchscreens\n"**);**

**return** **-**ENFILE**;**

**}**

//分配一个tsdev结构体

tsdev **=** kzalloc**(sizeof(**struct tsdev**),** GFP\_KERNEL**);**

**if** **(!**tsdev**)**

**return** **-**ENOMEM**;**

//初始化tsdev client\_list链表

INIT\_LIST\_HEAD**(&**tsdev**->**client\_list**);**

//初始化tsdev等待队列

init\_waitqueue\_head**(&**tsdev**->**wait**);**

sprintf**(**tsdev**->**name**,** "ts%d"**,** minor**);**

//设置tsdev其他参数

tsdev**->**exist **=** 1**;**

tsdev**->**minor **=** minor**;**

//设置tsdev的handle

tsdev**->**handle**.**dev **=** dev**;**

tsdev**->**handle**.**name **=** tsdev**->**name**;**

tsdev**->**handle**.**handler **=** handler**;**

tsdev**->**handle**.**private **=** tsdev**;**

//这里absmax为0x3FF，因为2440 ADC精度为10bit，最大值为3FF

delta **=** dev**->**absmax **[**ABS\_X**]** **-** dev**->**absmin **[**ABS\_X**]** **+** 1**;**

**if** **(**delta **==** 0**)**

delta **=** 1**;**

//x轴的伸缩系数 物理电压和像素之间的关系

tsdev**->**cal**.**xscale **=** **(**xres **<<** 8**)** **/** delta**;**

tsdev**->**cal**.**xtrans **=** **-** **((**dev**->**absmin **[**ABS\_X**]** **\*** tsdev**->**cal**.**xscale**)** **>>** 8**);**

//这里absmax为0x3FF，因为2440 ADC精度为10bit，最大值为3FF

delta **=** dev**->**absmax **[**ABS\_Y**]** **-** dev**->**absmin **[**ABS\_Y**]** **+** 1**;**

**if** **(**delta **==** 0**)**

delta **=** 1**;**

//y轴的伸缩系数

tsdev**->**cal**.**yscale **=** **(**yres **<<** 8**)** **/** delta**;**

tsdev**->**cal**.**ytrans **=** **-** **((**dev**->**absmin **[**ABS\_Y**]** **\*** tsdev**->**cal**.**yscale**)** **>>** 8**);**

//将tsdev加到tsdev\_table中

tsdev\_table**[**minor**]** **=** tsdev**;**

//分配子设备号

devt **=** MKDEV**(**INPUT\_MAJOR**,** TSDEV\_MINOR\_BASE **+** minor**),**

//创建目录/sys/class/input/tsx

cdev **=** class\_device\_create**(&**input\_class**,** **&**dev**->**cdev**,** devt**,**

dev**->**cdev**.**dev**,** tsdev**->**name**);**

//建立/sys链接

error **=** sysfs\_create\_link**(&**input\_class**.**subsys**.**kobj**,**

**&**cdev**->**kobj**,** tsdev**->**name**);**

//注册handle handle结构体会挂靠在device和handler的h\_list链表下

error **=** input\_register\_handle**(&**tsdev**->**handle**);**

**return** 0**;**

err\_remove\_link**:**

sysfs\_remove\_link**(&**input\_class**.**subsys**.**kobj**,** tsdev**->**name**);**

err\_cdev\_destroy**:**

class\_device\_destroy**(&**input\_class**,** devt**);**

err\_free\_tsdev**:**

tsdev\_table**[**minor**]** **=** **NULL;**

kfree**(**tsdev**);**

**return** error**;**

**}**

#### 2.2.3 tsdev\_event

static void tsdev\_event**(**struct input\_handle **\***handle**,** unsigned int type**,**unsigned int code**,** int value**)**

**{**

struct tsdev **\***tsdev **=** handle**->**private**;**

struct tsdev\_client **\***client**;**

struct timeval time**;**

**switch** **(**type**)** **{**

**case** EV\_ABS**:**

**switch** **(**code**)** **{**

**case** ABS\_X**:**

tsdev**->**x **=** value**;**

**break;**

**case** ABS\_Y**:**

tsdev**->**y **=** value**;**

**break;**

**case** ABS\_PRESSURE**:**

**if** **(**value **>** handle**->**dev**->**absmax**[**ABS\_PRESSURE**])**

value **=** handle**->**dev**->**absmax**[**ABS\_PRESSURE**];**

value **-=** handle**->**dev**->**absmin**[**ABS\_PRESSURE**];**

**if** **(**value **<** 0**)**

value **=** 0**;**

tsdev**->**pressure **=** value**;**

**break;**

**}**

**break;**

**case** EV\_REL**:**

**switch** **(**code**)** **{**

**case** REL\_X**:**

tsdev**->**x **+=** value**;**

**if** **(**tsdev**->**x **<** 0**)**

tsdev**->**x **=** 0**;**

**else** **if** **(**tsdev**->**x **>** xres**)**

tsdev**->**x **=** xres**;**

**break;**

**case** REL\_Y**:**

tsdev**->**y **+=** value**;**

**if** **(**tsdev**->**y **<** 0**)**

tsdev**->**y **=** 0**;**

**else** **if** **(**tsdev**->**y **>** yres**)**

tsdev**->**y **=** yres**;**

**break;**

**}**

**break;**

**case** EV\_KEY**:**

**if** **(**code **==** BTN\_TOUCH **||** code **==** BTN\_MOUSE**)** **{**

**switch** **(**value**)** **{**

**case** 0**:**

tsdev**->**pressure **=** 0**;**

**break;**

**case** 1**:**

**if** **(!**tsdev**->**pressure**)**

tsdev**->**pressure **=** 1**;**

**break;**

**}**

**}**

**break;**

**}**

**if** **(**type **!=** EV\_SYN **||** code **!=** SYN\_REPORT**)**

**return;**

list\_for\_each\_entry**(**client**,** **&**tsdev**->**client\_list**,** node**)** **{**

int x**,** y**,** tmp**;**

do\_gettimeofday**(&**time**);**

client**->**event**[**client**->**head**].**millisecs **=** time**.**tv\_usec **/** 100**;**

client**->**event**[**client**->**head**].**pressure **=** tsdev**->**pressure**;**

x **=** tsdev**->**x**;**

y **=** tsdev**->**y**;**

/\* Calibration \*/

**if** **(!**client**->**raw**)** **{**

x **=** **((**x **\*** tsdev**->**cal**.**xscale**)** **>>** 8**)** **+** tsdev**->**cal**.**xtrans**;**

y **=** **((**y **\*** tsdev**->**cal**.**yscale**)** **>>** 8**)** **+** tsdev**->**cal**.**ytrans**;**

**if** **(**tsdev**->**cal**.**xyswap**)** **{**

tmp **=** x**;** x **=** y**;** y **=** tmp**;**

**}**

**}**

client**->**event**[**client**->**head**].**x **=** x**;**

client**->**event**[**client**->**head**].**y **=** y**;**

client**->**head **=** **(**client**->**head **+** 1**)** **&** **(**TSDEV\_BUFFER\_SIZE **-** 1**);**

kill\_fasync**(&**client**->**fasync**,** SIGIO**,** POLL\_IN**);**

**}**

wake\_up\_interruptible**(&**tsdev**->**wait**);**

**}**

tsdev\_event将上报的数据处理如下，并精简了许多信息。

struct ts\_event **{**

short pressure**;**

short x**;**

short y**;**

short millisecs**;**

**};**

## 3.触摸屏相关中断

触摸屏相关的中断分别为IRQ\_TC和IRQ\_ADC，TC中断用于检测触摸屏是否按下/弹起，ADC中断用于采集触摸屏按下的X/Y轴电压。

通过设置ADCTSC寄存器，让2440处于等待中断模式，等待触摸屏按下/抬起操作。

iowrite32**(**WAIT4INT**(**0**),** base\_addr**+**S3C2410\_ADCTSC**);**

iowrite32**(**WAIT4INT**(**1**),** base\_addr**+**S3C2410\_ADCTSC**);**

通过设置ADCTSC和ADCCON寄存器，让2440处于连续X/Y轴坐标转换模式。

iowrite32**(**S3C2410\_ADCTSC\_PULL\_UP\_DISABLE **|** AUTOPST**,** base\_addr**+**S3C2410\_ADCTSC**);**

iowrite32**(**ioread32**(**base\_addr**+**S3C2410\_ADCCON**)** **|** S3C2410\_ADCCON\_ENABLE\_START**,** base\_addr**+**S3C2410\_ADCCON**);**

### 3.1 触摸屏相关中断初始化

IRQ\_TC和IRQ\_ADC有一个父中断IRQ\_ADCPARENT。中断初始化相关代码如下：

//ADC父中断设置

set\_irq\_chip**(**IRQ\_ADCPARENT**,** **&**s3c\_irq\_level\_chip**);**

set\_irq\_chained\_handler**(**IRQ\_ADCPARENT**,** s3c\_irq\_demux\_adc**);**

//子中断IRQ\_TC IRQ\_ADC设置

**for** **(**irqno **=** IRQ\_TC**;** irqno **<=** IRQ\_ADC**;** irqno**++)** **{**

irqdbf**(**"registering irq %d (s3c adc irq)\n"**,** irqno**);**

set\_irq\_chip**(**irqno**,** **&**s3c\_irq\_adc**);**

set\_irq\_handler**(**irqno**,** handle\_edge\_irq**);**

set\_irq\_flags**(**irqno**,** IRQF\_VALID**);**

**}**

当调用set\_irq\_chained\_handler设置IRQ\_ADCPARENT时,会将IRQ\_ADCPARENT中断打开。

### 3.2 触摸屏相关中断注册

request\_irq**(**IRQ\_ADC**,** stylus\_action**,** IRQF\_SAMPLE\_RANDOM **|** SA\_SHIRQ**,**"s3c2410\_action"**,** ts**.**dev**)**

request\_irq**(**IRQ\_TC**,** stylus\_updown**,** IRQF\_SAMPLE\_RANDOM**,**"s3c2410\_action"**,** ts**.**dev**)**

申请IRQ\_ADC和IRQ\_TC中断会打开这两个子中断，并将处理函数添加到对应的action链表中。

其中打开关闭中断的相关代码如下：

static struct irq\_chip s3c\_irq\_adc **=** **{**

**.**name **=** "s3c-adc"**,**

**.**mask **=** s3c\_irq\_adc\_mask**,**

**.**unmask **=** s3c\_irq\_adc\_unmask**,**

**.**ack **=** s3c\_irq\_adc\_ack**,**

**};**

static void

s3c\_irq\_adc\_mask**(**unsigned int irqno**)**

**{**

s3c\_irqsub\_mask**(**irqno**,** INTMSK\_ADCPARENT**,** 3 **<<** 9**);**

**}**

static void

s3c\_irq\_adc\_unmask**(**unsigned int irqno**)**

**{**

s3c\_irqsub\_unmask**(**irqno**,** INTMSK\_ADCPARENT**);**

**}**

static void

s3c\_irq\_adc\_ack**(**unsigned int irqno**)**

**{**

s3c\_irqsub\_ack**(**irqno**,** INTMSK\_ADCPARENT**,** 3 **<<** 9**);**

**}**

### 3.3 触摸屏中断响应流程

当IRQ\_TC和IRQ\_ADC中断发生，处理流程如下：  
(1)cpu进入IRQ异常处理模式。  
(2)中断向量表中通过b vector\_irq + stubs\_offset跳转到vector\_irq  
(3)保存相关现场后进入\_irq\_user，并切换到svc模式  
(4) asm\_do\_IRQ  
(5) irq\_des[irq]以中断号为下标取出一项 ->handle\_irq。  
(6) handle\_irq = s3c\_irq\_demux\_adc进入s3c\_irq\_demux\_adc。

s3c\_irq\_demux\_adc代码如下：

//ADC父中断操作函数

static void s3c\_irq\_demux\_adc**(**unsigned int irq**,**

struct irq\_desc **\***desc**)**

**{**

unsigned int subsrc**,** submsk**;**

unsigned int offset **=** 9**;**

struct irq\_desc **\***mydesc**;**

subsrc **=** \_\_raw\_readl**(**S3C2410\_SUBSRCPND**);**

submsk **=** \_\_raw\_readl**(**S3C2410\_INTSUBMSK**);**

subsrc **&=** **~**submsk**;**

subsrc **>>=** offset**;**

subsrc **&=** 3**;**

//判断哪个是IRQ\_TC还是IRQ\_ADC中断发生了。

**if** **(**subsrc **!=** 0**)** **{**

**if** **(**subsrc **&** 1**)** **{**

mydesc **=** irq\_desc **+** IRQ\_TC**;**

desc\_handle\_irq**(**IRQ\_TC**,** mydesc**);**

**}**

**if** **(**subsrc **&** 2**)** **{**

mydesc **=** irq\_desc **+** IRQ\_ADC**;**

desc\_handle\_irq**(**IRQ\_ADC**,** mydesc**);**

**}**

**}**

**}**

上述代码主要判断是IRQ\_TC还是IRQ\_ADC子中断发生了。

(7)调用handle\_edge\_irq函数，该函数会清空相关中断，并从action链表中找出action并一一调用。这样stylus\_action和stylus\_updown就会被调用了。

## 4.触摸屏坐标读取处理流程

(1)当触摸屏完成初始化(probe)后，触摸屏进入等待中断模式，等待触摸屏按下中断发生。

(2)当按下触摸屏后，stylus\_updown函数被调用。

static irqreturn\_t stylus\_updown**(**int irq**,** void **\***dev\_id**)**

**{**

unsigned long data0**;**

unsigned long data1**;**

int updown**;**

//读取ADCDATA0和ADCDATA1的bit15，判断触摸屏是按下还是弹起状态

data0 **=** ioread32**(**base\_addr**+**S3C2410\_ADCDAT0**);**

data1 **=** ioread32**(**base\_addr**+**S3C2410\_ADCDAT1**);**

updown **=** **(!(**data0 **&** S3C2410\_ADCDAT0\_UPDOWN**))** **&&** **(!(**data1 **&** S3C2410\_ADCDAT0\_UPDOWN**));**

//updown = 1表示按下

**if** **(**updown**)**

touch\_timer\_fire**(**0**);**

**return** IRQ\_HANDLED**;**

**}**

(3)继续执行touch\_timer\_fire函数，这个时候该函数会启动连续x/y轴坐标转换模式，该函数不仅仅完成该功能，下面再分析。

(4)等待adc中断发生，进入adc中断处理函数stylus\_action。

static irqreturn\_t stylus\_action**(**int irq**,** void **\***dev\_id**)**

**{**

unsigned long data0**;**

unsigned long data1**;**

data0 **=** ioread32**(**base\_addr**+**S3C2410\_ADCDAT0**);**

data1 **=** ioread32**(**base\_addr**+**S3C2410\_ADCDAT1**);**

//读取ADCDATA0低10位，X轴采样值，并累加

ts**.**xp **+=** data0 **&** S3C2410\_ADCDAT0\_XPDATA\_MASK**;**

//读取ADCDATA1低10位，Y轴采样值，并累加

ts**.**yp **+=** data1 **&** S3C2410\_ADCDAT1\_YPDATA\_MASK**;**

//计数器++

ts**.**count**++;**

//shift为4，表示采集4次求平均值，提高准确性。

**if** **(**ts**.**count **<** **(**1**<<**ts**.**shift**))** **{**

//如果还未到达4次，继续启动连续x/y轴坐标转换模式，继续采集

iowrite32**(**S3C2410\_ADCTSC\_PULL\_UP\_DISABLE **|** AUTOPST**,** base\_addr**+**S3C2410\_ADCTSC**);**

iowrite32**(**ioread32**(**base\_addr**+**S3C2410\_ADCCON**)** **|** S3C2410\_ADCCON\_ENABLE\_START**,** base\_addr**+**S3C2410\_ADCCON**);**

**}** **else** **{**

//如果到达4次，启动定时器，事件间隔为一个时间滴答。

mod\_timer**(&**touch\_timer**,** jiffies**+**1**);**

//停止ADC转换,防止屏幕抖动

iowrite32**(**WAIT4INT**(**1**),** base\_addr**+**S3C2410\_ADCTSC**);**

**}**

**return** IRQ\_HANDLED**;**

**}**

在stylus\_action函数里，首先读取ADC转换后的数据，判断是否连续转换够4次，如果没有则重新启动ADC转换；否则，启动1个时间滴答的定时器，这就会去执行touch\_timer\_fire定时器超时函数的上报事件和数据。

(5) 如果按键依然处于按下状态，上报键值。

static void touch\_timer\_fire**(**unsigned long data**)**

**{**

unsigned long data0**;**

unsigned long data1**;**

int updown**;**

data0 **=** ioread32**(**base\_addr**+**S3C2410\_ADCDAT0**);**

data1 **=** ioread32**(**base\_addr**+**S3C2410\_ADCDAT1**);**

updown **=** **(!(**data0 **&** S3C2410\_ADCDAT0\_UPDOWN**))** **&&** **(!(**data1 **&** S3C2410\_ADCDAT0\_UPDOWN**));**

//如果还处于按下状态

**if** **(**updown**)** **{**

**if** **(**ts**.**count **!=** 0**)** **{**

long tmp**;** /\* add by www.100ask.net \*/

tmp **=** ts**.**xp**;**

ts**.**xp **=** ts**.**yp**;**

ts**.**yp **=** tmp**;**

//将采集累加得到的数据/4

ts**.**xp **>>=** ts**.**shift**;**

ts**.**yp **>>=** ts**.**shift**;**

//上报X/Y坐标数据

input\_report\_abs**(**ts**.**dev**,** ABS\_X**,** ts**.**xp**);**

input\_report\_abs**(**ts**.**dev**,** ABS\_Y**,** ts**.**yp**);**

//报告按键事件，键值为1表示触摸点被按下

input\_report\_key**(**ts**.**dev**,** BTN\_TOUCH**,** 1**);**

//报告触摸屏压力值

input\_report\_abs**(**ts**.**dev**,** ABS\_PRESSURE**,** 1**);**

//上报同步事件

input\_sync**(**ts**.**dev**);**

**}**

ts**.**xp **=** 0**;**

ts**.**yp **=** 0**;**

ts**.**count **=** 0**;**

//重新进入连续x/y轴坐标转换模式

iowrite32**(**S3C2410\_ADCTSC\_PULL\_UP\_DISABLE **|** AUTOPST**,** base\_addr**+**S3C2410\_ADCTSC**);**

iowrite32**(**ioread32**(**base\_addr**+**S3C2410\_ADCCON**)** **|** S3C2410\_ADCCON\_ENABLE\_START**,** base\_addr**+**S3C2410\_ADCCON**);**

**}** **else** **{**

//如果触摸屏弹起了

ts**.**count **=** 0**;**

//报告弹起事件

input\_report\_key**(**ts**.**dev**,** BTN\_TOUCH**,** 0**);**

input\_report\_abs**(**ts**.**dev**,** ABS\_PRESSURE**,** 0**);**

input\_sync**(**ts**.**dev**);**

//重新进入等待中断模式

iowrite32**(**WAIT4INT**(**0**),** base\_addr**+**S3C2410\_ADCTSC**);·**

**}**

**}**

(6)继续进入x/y轴坐标转换模式，直到检测到触摸屏弹起，如果触摸屏弹起了，重新进入等待中断模式，等待触摸屏按下。

## 5.测试

当触摸屏驱动加载时，会生成如下文件：

# ls -l /sys/class/input/

**/**sys**/**class**/**input**/**event0**/** **/**sys**/**class**/**input**/**mice**/** **/**sys**/**class**/**input**/**ts0**/**

**/**sys**/**class**/**input**/**input0**/** **/**sys**/**class**/**input**/**mouse0**/**

# /dev/input/

**/**dev**/**input**/**event0 **/**dev**/**input**/**mice **/**dev**/**input**/**mouse0 **/**dev**/**input**/**ts0

可以读取/dev/input/event0设备的值

# cat /dev/input/event0 | hexdump

/\*按下\*/

//0003 ABS 0000 ABS\_X 0115 X坐标

0000000 0145 0000 5d91 0002 0003 0000 0115 0000

//0003 ABS 0001 ABS\_Y 0137 Y坐标

0000010 0145 0000 5da3 0002 0003 0001 0137 0000

//0001 KEY 014a BTN\_TOUCH 0001 按下

0000020 0145 0000 5da8 0002 0001 014a 0001 0000

//0003 ABS 0018 ABS\_PRESSURE 0001 压力

0000030 0145 0000 5daa 0002 0003 0018 0001 0000

//0000 SYNC

0000040 0145 0000 5dad 0002 0000 0000 0000 0000

/\*松开\*/

//0001 KEY 014a BTN\_TOUCH 0000 松开

0000050 0145 0000 7145 0002 0001 014a 0000 0000

//0003 ABS 0018 ABS\_PRESSURE 0000 压力

0000060 0145 0000 7160 0002 0003 0018 0000 0000

//0000 SYNC

0000070 0145 0000 7163 0002 0000 0000 0000 0000

当然也可以读取/dev/input/ts0设备的值。

# cat /dev/input/ts0 | hexdump

tsdev **(**compaq touchscreen emulation**)** is scheduled **for** removal**.**

See Documentation**/**feature**-**removal**-**schedule**.**txt **for** details**.**

//每一行有两组数据

//第一个数据0001(按下)0000(松开)

//第二个数据表示X轴坐标

//第三个数据表示Y轴坐标

//坐标是按320\*240进行转换的，而不是电压采样值

0000210 0001 0048 005d 079e 0001 004b 005d 07d0

0000220 0001 004c 005d 0802 0001 004f 005c 0834

0000230 0001 004c 005c 0866 0001 004f 005c 0898

0000240 0001 004d 005c 08fc 0001 0051 005d 092e

0000250 0001 0050 005d 0960 0001 0051 005d 0992

0000260 0001 0051 005d 09c4 0000 0051 005d 09f6

从打印看tsdev有可能要被抛弃，tslib也是读取/dev/input/event0的数据。