（原创）6410触摸屏驱动分析（s3c-ts.c）（Linux）（分析）

摘要：

分析内核s3c-ts.c源码，看它是如何采集坐标信息及防抖动处理的。

介绍：

直接上源码吧，完全注释：

```
/* linux/drivers/input/touchscreen/s3c-ts.c
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 * a misc driver for mini6410 touch screen
 *  by FriendlyARM 2010
 *
 * Based on following software:
 *
 ** Copyright (c) 2004 Arnaud Patard <arnaud.patard@rtp-net.org>
 ** iPAQ H1940 touchscreen support
 **
 ** ChangeLog
 **
 ** 2004-09-05: Herbert Potzl <herbert@13thfloor.at>
 ** - added clock (de-)allocation code
 **
 ** 2005-03-06: Arnaud Patard <arnaud.patard@rtp-net.org>
 **      - h1940_ -> s3c24xx (this driver is now also used on the n30
 **        machines :P)
 **      - Debug messages are now enabled with the config option
 **        TOUCHSCREEN_S3C_DEBUG
 **      - Changed the way the value are read
 **      - Input subsystem should now work
 **      - Use ioremap and readl/writel
 **
 ** 2005-03-23: Arnaud Patard <arnaud.patard@rtp-net.org>
```

```
 **      - Make use of some undocumented features of the touchscreen
 **        controller
 **
 ** 2006-09-05: Ryu Euiyoul <ryu.real@gmail.com>
 **      - added power management suspend and resume code
 *
 */

#include <linux/errno.h>
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/slab.h>
#include <linux/input.h>
#include <linux/init.h>
#include <linux/serio.h>
#include <linux/delay.h>
#include <linux/platform_device.h>
#include <linux/clk.h>
#include <linux/fs.h>
#include <linux/poll.h>
#include <linux/irq.h>
#include <linux/interrupt.h>
#include <linux/cdev.h>
#include <linux/miscdevice.h>

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

#include <mach/hardware.h>

#include <plat/regs-adc.h>
#include <mach/irqs.h>
#include <mach/map.h>
#include <mach/regs-clock.h>
#include <mach/regs-gpio.h>
#include <mach/gpio-bank-a.h>
#include <mach/ts.h>

#define CONFIG_TOUCHSCREEN_S3C_DEBUG
#undef CONFIG_TOUCHSCREEN_S3C_DEBUG
#define DEBUG_LVL       KERN_DEBUG


#ifdef CONFIG_MINI6410_ADC
DEFINE_SEMAPHORE(ADC_LOCK);  //定义并初始化了一个信号量
                             //37内核就没有DECLARE_MUTEX了吧，功能应该是一样的


/* Indicate who is using the ADC controller */
//ADC的状态，防止触摸屏转换时，ADC正在被使用
#define LOCK_FREE       0
#define LOCK_TS         1
```

```
92  #define LOCK_ADC        2
93  static int adc_lock_id = LOCK_FREE;
94
95  #define ADC_free()      (adc_lock_id == LOCK_FREE)
96  #define ADC_locked4TS() (adc_lock_id == LOCK_TS)
97
98  //==
99  static inline int s3c_ts_adc_lock(int id) {
100     int ret;
101
102     ret = down_trylock(&ADC_LOCK); //获取自旋锁
103     if (!ret) {
104         adc_lock_id = id;
105     }
106
107     return ret;  //返回状态  1:失败  0: 成功
108 }
109 //--
110
111 static inline void s3c_ts_adc_unlock(void) {
112     adc_lock_id = 0;
113     up(&ADC_LOCK);  //释放自旋锁
114 }
115 #endif
116
117
118 /* Touchscreen default configuration */
119 struct s3c_ts_mach_info s3c_ts_default_cfg __initdata = {
120     .delay              = 10000,  //转换延时
121     .presc              = 49,  //转换时钟分频
122     .oversampling_shift = 2,      //转换次数  4次
123     .resol_bit          = 12, //转换精度
124     .s3c_adc_con        = ADC_TYPE_2   //6410是type2
125 };
126 /*
127 struct s3c_ts_mach_info s3c_ts_default_cfg __initdata = {
128     .delay              = 10000,
129     .presc              = 49,
130     .oversampling_shift = 2,
131     .resol_bit          = 10
132 };
133 */
134 /*
135  * Definitions & global arrays.
136  */
137 #define DEVICE_NAME     "touchscreen"
138 static DECLARE_WAIT_QUEUE_HEAD(ts_waitq); //定义并初始化一个等待队列
139
140 typedef unsigned        TS_EVENT;
141 #define NR_EVENTS       64     //触摸屏fifo大小
142
143 static TS_EVENT         events[NR_EVENTS];
```

```
144 static int             evt_head, evt_tail; //fifo的头的尾
145                                                     //驱动写fifo时evt_head++，应用读fifo时
146 evt_tail++
147 

148 #define ts_evt_pending()    ((volatile u8)(evt_head != evt_tail))   //相等就表示满了
149 #define ts_evt_get()        (events + evt_tail)
150 #define ts_evt_pull()       (evt_tail = (evt_tail + 1) & (NR_EVENTS - 1))
151 #define ts_evt_clear()      (evt_head = evt_tail = 0)
152 
153 //将AD转换的值放入FIFO
154 //这里是一个先进先出的fifo
155 //只要有数据被添加进来，就会唤醒ts_waitq进程
156 static void ts_evt_add(unsigned x, unsigned y, unsigned down) {
157     unsigned ts_event;
158     int next_head;
159 
160     ts_event = ((x << 16) | (y)) | (down << 31);
161     next_head = (evt_head + 1) & (NR_EVENTS - 1);
162         //没满就装入
163     if (next_head != evt_tail) {
164         events[evt_head] = ts_event;
165         evt_head = next_head;
166         //printk("====>Add ... [ %4d,  %4d ]%s\n", x, y, down ? "":" ~~~");
167 
168         /* wake up any read call */
169         if (waitqueue_active(&ts_waitq)) { //判断等待队列是否有進程睡眠
170             wake_up_interruptible(&ts_waitq);  //唤醒ts_waitq等待队列中所有interruptible类型的进程
171         }
172     } else {
173         /* drop the event and try to wakeup readers */
174         printk(KERN_WARNING "mini6410-ts: touch event buffer full");
175         wake_up_interruptible(&ts_waitq);
176     }
177 }
178 
179 static unsigned int s3c_ts_poll( struct file *file, struct poll_table_struct *wait)
180 {
181     unsigned int mask = 0;
182 
183     //将ts_waitq等待队列添加到poll_table里去
184     poll_wait(file, &ts_waitq, wait);
185     //返回掩码
186     if (ts_evt_pending())
187         mask |= POLLIN | POLLRDNORM;  //返回设备可读

188 
189     return mask;
190 }
191 
192 //读 系统调用==
193 static int s3c_ts_read(struct file *filp, char __user *buff, size_t count, loff_t *offp)
194 {
```

```
195     DECLARE_WAITQUEUE(wait, current); //把当前进程加到定义的等待队列头wait中
196     char *ptr = buff;
197     int err = 0;
198 
199     add_wait_queue(&ts_waitq, &wait); //把wait入到等待队列头中。该队列会在进程等待的条件满足时唤醒它。
200                                       //我们必须在其他地方写相关代码，在事件发生时，对等的队列执行wake_up()操作。
201                                       //这里是在ts_evt_add里wake_up
202     while (count >= sizeof(TS_EVENT)) {
203         err = -ERESTARTSYS;
204         if (signal_pending(current)) //如果是信号唤醒     参
205 考http://www.360doc.com/content/10/1009/17/1317564_59632874.shtml
206             break;
207 
208         if (ts_evt_pending()) {
209             TS_EVENT *evt = ts_evt_get();
210 
211             err = copy_to_user(ptr, evt, sizeof(TS_EVENT));
212             ts_evt_pull();
213 
214             if (err)
215                 break;
216 
217             ptr += sizeof(TS_EVENT);
218             count -= sizeof(TS_EVENT);
219             continue;
220         }
221 
222         set_current_state(TASK_INTERRUPTIBLE); //改变进程状态为可中断的睡眠
223         err = -EAGAIN;
224         if (filp->f_flags & O_NONBLOCK) //如果上层调用是非阻塞方式，则不阻塞该进程，直接返回EAGAIN
225             break;
226         schedule();  //本进程在此处交出CPU控制权，等待被唤醒
227                       //进程调度的意思侧重于把当前任务从CPU拿掉，再从就绪队列中按照调度算法取一就绪进程占用CPU
228     }
229     current->state = TASK_RUNNING;
230     remove_wait_queue(&ts_waitq, &wait);
231 
232     return ptr == buff ? err : ptr - buff;
233 }
234 //--
235 
236 static int s3c_ts_open(struct inode *inode, struct file *filp) {
237     /* flush event queue */
238     ts_evt_clear();
239 
240     return 0;
241 }
242 
243 //当应用程序操作设备文件时调用的open read等函数，最终会调用这个结构体中对应的函数
244 static struct file_operations dev_fops = {
245     .owner              = THIS_MODULE,
246     .read               = s3c_ts_read,
```

```
247     .poll               = s3c_ts_poll,  //select系统调用
248     .open               = s3c_ts_open,
249 };
250 
251 //设备号，设备名，注册的时候用到这个数组
252 //混杂设备主设备号为10
253 static struct miscdevice misc = {
254         .minor              = MISC_DYNAMIC_MINOR, //自动分配次设置号
255     //.minor                = 180,
256     .name               = DEVICE_NAME,
257     .fops               = &dev_fops,
258 };
259 
260 //x为0时为等待按下中断，x为1是为等待抬起中断
261 #define WAIT4INT(x)     (((x) << 8) | \
262         S3C_ADCTSC_YM_SEN | S3C_ADCTSC_YP_SEN | S3C_ADCTSC_XP_SEN | \
263         S3C_ADCTSC_XY_PST(3))
264 
265 //自动连续测量X坐标和Y坐标
266 #define AUTOPST         (S3C_ADCTSC_YM_SEN | S3C_ADCTSC_YP_SEN | S3C_ADCTSC_XP_SEN | \
267         S3C_ADCTSC_AUTO_PST | S3C_ADCTSC_XY_PST(0))
268 
269 static void __iomem     *ts_base;
270 static struct resource  *ts_mem;
271 static struct resource  *ts_irq;
272 static struct clk       *ts_clock;
273 static struct s3c_ts_info   *ts;
274 
275 /**
276  * get_down - return the down state of the pen
277  * @data0: The data read from ADCDAT0 register.
278  * @data1: The data read from ADCDAT1 register.
279  *
280  * Return non-zero if both readings show that the pen is down.
281  */
282 static inline bool get_down(unsigned long data0, unsigned long data1)
283 {
284     /* returns true if both data values show stylus down */
285     return (!(data0 & S3C_ADCDAT0_UPDOWN) && !(data1 & S3C_ADCDAT1_UPDOWN)); //判断data0,data1最高位是否仍
286 为"0"，为“0”表示触摸笔状态保持为down
287 }
288 
289 
290 /*================================================================================
291     touch_timer_fire这个函数主要实现以下功能：
292 
293     1、  触摸笔开始点击的时候， 在中断函数stylus_updown里面被调用，
294          此时缓存区没有数据，ts.count为0，  并且开启AD转换,而后进入 AD 中断
295 
296     2、  ADC中断函数stylus_action把缓冲区填满的时候，作为中断后半段函数稍后被调用（由内核定时器触发中断），
297          此时ts.count为4，算出其平均值后，交给事件处理层(Event Handler)处理，
298          主要是填写缓冲，然后唤醒等待输入数据的进程。
```

```
299
300     3、  stylus抬起，等到缓冲区填满后(可能会包含一些无用的数据)被调用，
301          这时候判断出stylus up，报告stylus up事件，重新等待stylus down。
302 ============================================================================================*/
303
304 static void touch_timer_fire(unsigned long data) {
305     unsigned long data0;
306     unsigned long data1;
307     int pendown;
308
309 #ifdef CONFIG_MINI6410_ADC
310     if (!ADC_locked4TS()) {
311         /* Note: pen UP interrupt detected and handled, the lock is released,
312          * so do nothing in the timer which started by ADC ISR. */
313         return;
314     }
315 #endif
316
317     data0 = readl(ts_base + S3C_ADCDAT0);
318     data1 = readl(ts_base + S3C_ADCDAT1);//读取AD转换数据的值
319
320     pendown = get_down(data0, data1);
321
322     if (pendown) {
323         if (ts->count == (1 << ts->shift)) {  //定时器触发touch_timer_fire中断时执行这个括号里
324 #ifdef CONFIG_TOUCHSCREEN_S3C_DEBUG
325             {
326                 struct timeval tv;
327                 do_gettimeofday(&tv);
328                 printk(KERN_INFO "T: %06d, X: %03ld, Y: %03ld\n",
329                         (int)tv.tv_usec, ts->xp, ts->yp);
330             }
331 #endif
332
333             ts_evt_add((ts->xp >> ts->shift), (ts->yp >> ts->shift), 1);//求平均，并写入fifo
334
335             ts->xp = 0;
336             ts->yp = 0;
337             ts->count = 0;
338         }
339
340         /* start automatic sequencing A/D conversion */
341         //每次按下有四次AD转换，以下为在按下中断中触发的第一次AD转换，其余三次在AD转换中断处理函数中触发
342         //AUTOPST表示自动连续测量 以得到X位置，Y位置
343         writel(S3C_ADCTSC_PULL_UP_DISABLE | AUTOPST, ts_base + S3C_ADCTSC);
344         // 启动D转换，转换后会产生中断IRQ_ADC
345         writel(readl(ts_base + S3C_ADCCON) | S3C_ADCCON_ENABLE_START,
346                 ts_base + S3C_ADCCON);
347
348     } else {  //如果是松开，报告其触摸笔状态
349         ts->xp = 0;
```

```
350         ts->yp = 0;
351         ts->count = 0;
352 
353         ts_evt_add(0, 0, 0);
354 
355         /* PEN is UP, Let's wait the PEN DOWN interrupt */
356         writel(WAIT4INT(0), ts_base + S3C_ADCTSC);  // 设置INT 位，等待 DOWN 中断
357 
358 #ifdef CONFIG_MINI6410_ADC
359         if (ADC_locked4TS()) {
360             s3c_ts_adc_unlock();
361         }
362 #endif
363     }
364 }
365 
366 static DEFINE_TIMER(touch_timer, touch_timer_fire, 0, 0);
367 
368 //触摸屏按下松开中断服务==
369 static irqreturn_t stylus_updown(int irqno, void *param)
370 {
371 #ifdef CONFIG_TOUCHSCREEN_S3C_DEBUG
372     unsigned long data0;
373     unsigned long data1;
374     int is_waiting_up;
375     int pendown;
376 #endif
377 
378 #ifdef CONFIG_MINI6410_ADC
379     if (!ADC_locked4TS()) {
380         if (s3c_ts_adc_lock(LOCK_TS)) {
381             /* Locking ADC controller failed */
382             printk("Lock ADC failed, %d\n", adc_lock_id);
383             return IRQ_HANDLED;
384         }
385     }
386 #endif
387 
388 #ifdef CONFIG_TOUCHSCREEN_S3C_DEBUG
389     data0 = readl(ts_base + S3C_ADCDAT0);
390     data1 = readl(ts_base + S3C_ADCDAT1);
391 
392     is_waiting_up = readl(ts_base + S3C_ADCTSC) & (1 << 8);
393     pendown = get_down(data0, data1);
394 
395     printk("P: %d <--> %c\n", pendown, is_waiting_up ? 'u':'d');
396 #endif
397         //执行如下语句否则不断产生中断从而把系统卡死
398     if (ts->s3c_adc_con == ADC_TYPE_2) {
399         /* Clear ADC and PEN Down/UP interrupt */
400         __raw_writel(0x0, ts_base + S3C_ADCCLRWK);
401         __raw_writel(0x0, ts_base + S3C_ADCCLRINT);
```

```
402     }
403
404     /* TODO we should never get an interrupt with pendown set while
405      * the timer is running, but maybe we ought to verify that the
406      * timer isn't running anyways. */
407
408     touch_timer_fire(1);
409
410     return IRQ_HANDLED;
411 }
412
413 //ad转换结束中断服务程序==
414 static irqreturn_t stylus_action(int irqno, void *param)
415 {
416     unsigned long data0;
417     unsigned long data1;
418
419 #ifdef CONFIG_MINI6410_ADC
420     if (!ADC_locked4TS()) {
421         if (ADC_free()) {
422             printk("Unexpected\n");
423
424             /* Clear ADC interrupt */
425             __raw_writel(0x0, ts_base + S3C_ADCCLRINT);
426         }
427
428         return IRQ_HANDLED;
429     }
430 #endif
431
432     data0 = readl(ts_base + S3C_ADCDAT0);
433     data1 = readl(ts_base + S3C_ADCDAT1);
434
435     if (ts->resol_bit == 12) {
436 #if defined(CONFIG_TOUCHSCREEN_NEW)
437         ts->yp += S3C_ADCDAT0_XPDATA_MASK_12BIT - (data0 & S3C_ADCDAT0_XPDATA_MASK_12BIT);
438         ts->xp += S3C_ADCDAT1_YPDATA_MASK_12BIT - (data1 & S3C_ADCDAT1_YPDATA_MASK_12BIT);
439 #else
440         ts->xp += data0 & S3C_ADCDAT0_XPDATA_MASK_12BIT;
441         ts->yp += data1 & S3C_ADCDAT1_YPDATA_MASK_12BIT;
442 #endif
443     } else {
444 #if defined(CONFIG_TOUCHSCREEN_NEW)
445         ts->yp += S3C_ADCDAT0_XPDATA_MASK - (data0 & S3C_ADCDAT0_XPDATA_MASK);
446         ts->xp += S3C_ADCDAT1_YPDATA_MASK - (data1 & S3C_ADCDAT1_YPDATA_MASK);
447 #else
448         ts->xp += data0 & S3C_ADCDAT0_XPDATA_MASK;
449         ts->yp += data1 & S3C_ADCDAT1_YPDATA_MASK;
450 #endif
451     }  // 转换结果累加
452
453     ts->count++;
```

```
454
455     if (ts->count < (1 << ts->shift)) { // 采样未完成，继续下一次采样 ，通过 ENABLE_START 启动 AD 转换，一次一个
456 数据
457         writel(S3C_ADCTSC_PULL_UP_DISABLE | AUTOPST, ts_base + S3C_ADCTSC);
458         writel(readl(ts_base + S3C_ADCCON) | S3C_ADCCON_ENABLE_START, ts_base + S3C_ADCCON);
459     } else {  // 采样完毕，激活下半部处理程序touch_timer_fire，处理接收数据
460         mod_timer(&touch_timer, jiffies + 1);   //设置定时器超时的时间，目的是为了延时触发 touch_timer_fire 中
461 断，如果在这段时间有抬起中断发生，则表示是抖动
462                                                 //jiffies变量记录了系统启动以来，系统定时器已经触发的次数。内核每
463 秒钟将jiffies变量增加HZ次。
464                                                 //因此，对于HZ值为100的系统，1个jiffy等于10ms，而对于HZ为1000的
465 系统，1个jiffy仅为1ms
466
467         writel(WAIT4INT(1), ts_base + S3C_ADCTSC);   //设置为等待抬起中断
468     }
469
470     if (ts->s3c_adc_con == ADC_TYPE_2) {
471         /* Clear ADC and PEN Down/UP interrupt */
472         __raw_writel(0x0, ts_base + S3C_ADCCLRWK);
473         __raw_writel(0x0, ts_base + S3C_ADCCLRINT);
474     }
475
476     return IRQ_HANDLED;
477 }
478
479
480 #ifdef CONFIG_MINI6410_ADC
481 static unsigned int _adccon, _adctsc, _adcdly;
482
483 //其它模块要用ADC时，需要调用这个函数，来确定ADC是否可用，如果可用，则将它锁住，不让别的驱动用
484 int mini6410_adc_acquire_io(void) {
485     int ret;
486
487     ret = s3c_ts_adc_lock(LOCK_ADC); //锁住ADC，不让其它模块使用
488     if (!ret) {  //如果ADC没有被使用，则保存ADC寄存器的值
489         _adccon = readl(ts_base + S3C_ADCCON);
490         _adctsc = readl(ts_base + S3C_ADCTSC);
491         _adcdly = readl(ts_base + S3C_ADCDLY);
492     }
493
494     return ret;// 0:操作成功  1：操作失败
495 }
496 EXPORT_SYMBOL(mini6410_adc_acquire_io);  //声明为外部可用
497
498 //其它模块不要用ADC了，需要调用这个函数，来解锁ADC让别的驱动用
499 void mini6410_adc_release_io(void) {
500         //还原ADC寄存器的设置
501     writel(_adccon, ts_base + S3C_ADCCON);
502     writel(_adctsc, ts_base + S3C_ADCTSC);
503     writel(_adcdly, ts_base + S3C_ADCDLY);
504     writel(WAIT4INT(0), ts_base + S3C_ADCTSC);
505
```

```
506     s3c_ts_adc_unlock(); //释放ADC，其它模块可以使用
507 }

508
509 EXPORT_SYMBOL(mini6410_adc_release_io);
510 #endif
511
512 //获得触摸屏的配置信息==
513 static struct s3c_ts_mach_info *s3c_ts_get_platdata(struct device *dev)
514 {
515     if (dev->platform_data != NULL)
516         return (struct s3c_ts_mach_info *)dev->platform_data;  //优先使用 arch/arm/mach-s3c64xx中的定义
517
518     return &s3c_ts_default_cfg;   //如果前面没定义，则使用本函数的default定义
519 }
520 //--
521
522 /*
523  * The functions for inserting/removing us as a module.
524  */
525 static int __init s3c_ts_probe(struct platform_device *pdev)
526 {
527     struct resource *res;
528     struct device *dev;
529     struct s3c_ts_mach_info * s3c_ts_cfg;
530     int ret, size;
531
532     dev = &pdev->dev;
533
534     res = platform_get_resource(pdev, IORESOURCE_MEM, 0);
535     if (res == NULL) {
536         dev_err(dev,"no memory resource specified\n");
537         return -ENOENT;
538     }
539
540     size = (res->end - res->start) + 1;
541     ts_mem = request_mem_region(res->start, size, pdev->name);
542     if (ts_mem == NULL) {
543         dev_err(dev, "failed to get memory region\n");
544         ret = -ENOENT;
545         goto err_req;
546     }
547

548     ts_base = ioremap(res->start, size);
549     if (ts_base == NULL) {
550         dev_err(dev, "failed to ioremap() region\n");
551         ret = -EINVAL;
552         goto err_map;
553     }
554
555     ts_clock = clk_get(&pdev->dev, "adc");
556     if (IS_ERR(ts_clock)) {
```

```
557         dev_err(dev, "failed to find watchdog clock source\n");
558         ret = PTR_ERR(ts_clock);
559         goto err_clk;
560     }
561 
562     clk_enable(ts_clock);
563 
564     s3c_ts_cfg = s3c_ts_get_platdata(&pdev->dev);  //获取配置参数
565 
566     //设置ADC分频
567     if ((s3c_ts_cfg->presc & 0xff) > 0)
568         writel(S3C_ADCCON_PRSCEN | S3C_ADCCON_PRSCVL(s3c_ts_cfg->presc & 0xff),
569                 ts_base + S3C_ADCCON);
570     else
571         writel(0, ts_base + S3C_ADCCON);
572 
573     /* Initialise registers */
574     //设置转换延时
575     if ((s3c_ts_cfg->delay & 0xffff) > 0)
576         writel(s3c_ts_cfg->delay & 0xffff, ts_base + S3C_ADCDLY);
577 
578     if (s3c_ts_cfg->resol_bit == 12) {
579         switch(s3c_ts_cfg->s3c_adc_con) {
580             case ADC_TYPE_2:
581                 writel(readl(ts_base + S3C_ADCCON) | S3C_ADCCON_RESSEL_12BIT,
582                         ts_base + S3C_ADCCON);
583                 break;
584 
585             case ADC_TYPE_1:
586                 writel(readl(ts_base + S3C_ADCCON) | S3C_ADCCON_RESSEL_12BIT_1,
587                         ts_base + S3C_ADCCON);
588                 break;
589 
590             default:
591                 dev_err(dev, "Touchscreen over this type of AP isn't supported !\n");
592                 break;
593         }
594     }
595 
596     writel(WAIT4INT(0), ts_base + S3C_ADCTSC);
597 
598     ts = kzalloc(sizeof(struct s3c_ts_info), GFP_KERNEL);
599 
600     ts->shift = s3c_ts_cfg->oversampling_shift;
601     ts->resol_bit = s3c_ts_cfg->resol_bit;
602     ts->s3c_adc_con = s3c_ts_cfg->s3c_adc_con;
603 
604     /* For IRQ_PENDUP */
605     ts_irq = platform_get_resource(pdev, IORESOURCE_IRQ, 0);
606     if (ts_irq == NULL) {
607         dev_err(dev, "no irq resource specified\n");
608         ret = -ENOENT;
```

```
609         goto err_irq;
610     }
611 
612     ret = request_irq(ts_irq->start, stylus_updown, IRQF_SAMPLE_RANDOM, "s3c_updown", ts);
613     if (ret != 0) {
614         dev_err(dev,"s3c_ts.c: Could not allocate ts IRQ_PENDN !\n");
615         ret = -EIO;
616         goto err_irq;
617     }
618 
619     /* For IRQ_ADC */
620     ts_irq = platform_get_resource(pdev, IORESOURCE_IRQ, 1);
621     if (ts_irq == NULL) {
622         dev_err(dev, "no irq resource specified\n");
623         ret = -ENOENT;
624         goto err_irq;
625     }
626 
627     ret = request_irq(ts_irq->start, stylus_action, IRQF_SAMPLE_RANDOM | IRQF_SHARED,
628             "s3c_action", ts);
629     if (ret != 0) {
630         dev_err(dev, "s3c_ts.c: Could not allocate ts IRQ_ADC !\n");
631         ret = -EIO;
632         goto err_irq;
633     }
634 
635     printk(KERN_INFO "%s got loaded successfully : %d bits\n", DEVICE_NAME, s3c_ts_cfg->resol_bit);
636 
637     ret = misc_register(&misc);  //注册这混杂字符设备
638     if (ret) {
639         dev_err(dev, "s3c_ts.c: Could not register device(mini6410 touchscreen)!\n");
640         ret = -EIO;
641         goto fail;
642     }
643 
644     return 0;
645 
646 fail:
647     free_irq(ts_irq->start, ts->dev);
648     free_irq(ts_irq->end, ts->dev);
649 
650 err_irq:
651     kfree(ts);
652 
653     clk_disable(ts_clock);
654     clk_put(ts_clock);
655 
656 err_clk:
657     iounmap(ts_base);
658 
659 err_map:
660     release_resource(ts_mem);
```

```
661     kfree(ts_mem);
662 
663 err_req:
664     return ret;
665 }
666 
667 static int s3c_ts_remove(struct platform_device *dev)
668 {
669     printk(KERN_INFO "s3c_ts_remove() of TS called !\n");
670 
671     disable_irq(IRQ_ADC);
672     disable_irq(IRQ_PENDN);
673 
674     free_irq(IRQ_PENDN, ts->dev);
675     free_irq(IRQ_ADC, ts->dev);
676 
677     if (ts_clock) {
678         clk_disable(ts_clock);
679         clk_put(ts_clock);
680         ts_clock = NULL;
681     }
682 
683     misc_deregister(&misc);
684     iounmap(ts_base);
685 
686     return 0;
687 }
688 
689 #ifdef CONFIG_PM
690 static unsigned int adccon, adctsc, adcdly;
691 
692 static int s3c_ts_suspend(struct platform_device *dev, pm_message_t state)
693 {
694     adccon = readl(ts_base + S3C_ADCCON);
695     adctsc = readl(ts_base + S3C_ADCTSC);
696     adcdly = readl(ts_base + S3C_ADCDLY);
697 
698     disable_irq(IRQ_ADC);
699     disable_irq(IRQ_PENDN);
700 
701     clk_disable(ts_clock);
702 
703     return 0;
704 }
705 
706 static int s3c_ts_resume(struct platform_device *pdev)
707 {
708     clk_enable(ts_clock);
709 
710     writel(adccon, ts_base + S3C_ADCCON);
711     writel(adctsc, ts_base + S3C_ADCTSC);
```

```
712     writel(adcdly, ts_base + S3C_ADCDLY);
713     writel(WAIT4INT(0), ts_base + S3C_ADCTSC);
714 
715     enable_irq(IRQ_ADC);
716     enable_irq(IRQ_PENDN);
717     return 0;
718 }
719 #else
720 #define s3c_ts_suspend  NULL
721 #define s3c_ts_resume   NULL
722 #endif
723 
724 static struct platform_driver s3c_ts_driver = {
725     .probe          = s3c_ts_probe,
726     .remove         = s3c_ts_remove,
727     .suspend        = s3c_ts_suspend,
728     .resume         = s3c_ts_resume,
729     .driver         = {
730         .owner          = THIS_MODULE,
731         .name           = "s3c-ts",
732     },
733 };
734 
735 static char banner[] __initdata = KERN_INFO "S3C Touchscreen driver, (c) 2010 FriendlyARM,\n";
736 
737 static int __init s3c_ts_init(void)
738 {
739     printk(banner);
740     return platform_driver_register(&s3c_ts_driver);
741 }
742 
743 static void __exit s3c_ts_exit(void)
744 {
745     platform_driver_unregister(&s3c_ts_driver);
746 }
747 
748 module_init(s3c_ts_init);
749 module_exit(s3c_ts_exit);
750 
751 MODULE_AUTHOR("FriendlyARM Inc.");
752 MODULE_LICENSE("GPL");
753 
754 
755 /*
756  * 驱动分析
757  * 1、内核是如何加载驱动的?
758  *     首先要提到两个结构体：设备用Platform_device表示，驱动用Platform_driver进行注册
759  *     Platform机制开发发底层驱动的大致流程为：  定义 platform_device  注册 platform_device 定义 platform_driver
760 注册 platform_driver
761  *     首先要确认的就是设备的资源信息platform_device，例如设备的地址，中断号等 该结构体定义在
762 kernel\include\linux\platform_device.h
763  *     该结构一个重要的元素是resource，该元素存入了最为重要的设备资源信息，定义在kernel\include\linux\ioport.h中
```

```
764  *    下面我们以本例来进行说明：
765  *        arch/arm/mach-s3c64xx中dev-ts-mini6410.c中定义了platform_device s3c_device_ts
766  *        定义好了platform_device结构体后就可以调用函数platform_add_devices向系统中添加该设备了，之后可以调用
767 platform_driver_register()进行设备注册。
768  *        要注意的是，这里的platform_device设备的注册过程必须在相应设备驱动加载之前被调用，即执行
769 platform_driver_register之前,原因是因为驱动注册时需要
770  *        匹配内核中所以已注册的设备名。
771  *        platform_devicerr的注册是在arch/arm/mach-s3c64xx中mach-mini6410.c中的mini6410_machine_init函数实现
772 的。
773  *        mini6410_machine_init是在启动后调用，它是在module_init之前；更具体的见MACHINE_START
774  *  MACHINE_START(MINI6410, "MINI6410")
775  *
776  *         .boot_params  = S3C64XX_PA_SDRAM + 0x100,  //.boot_params是bootloader向内核传递的参数的位置，这要和
777 bootloader中参数的定义要一致。
778  *
779  *         .init_irq = s3c6410_init_irq,  //.init_irq在start_kernel() --> init_IRQ() --> init_arch_irq()中被
780 调用
781  *         .map_io       = mini6410_map_io,   //.map_io 在 setup_arch() --> paging_init() -->
782 devicemaps_init()中被调用
783  *         .init_machine = mini6410_machine_init,  //init_machine 在 arch/arm/kernel/setup.c 中被
784 customize_machine 调用，
785  *                           //放在 arch_initcall() 段里面，会自动按顺序被调用。
     *         .timer        = &s3c24xx_timer,  //.timer是定义系统时钟，定义TIMER4为系统时钟，在arch/arm/plat-
     s3c/time.c中体现。

     *                       //在start_kernel() --> time_init()中被调用。
     *        MACHINE_END
     *        再来看看platform_driver，这个定义在本文中，
     *        在驱动初始化函数中调用函数platform_driver_register()注册platform_driver，需要注意的是s3c_device_ts结构中
     name元素和s3c_ts_driver结构中driver.name
     *        必须是相同的，这样在platform_driver_register()注册时会对所有已注册的所有platform_device中的name和当前注册
     的platform_driver的driver.name进行比较，
     *        只有找到相同的名称的platfomr_device才能注册成功，当注册成功时会调用platform_driver结构元素probe函数指针，这
     里就是s3c_ts_probe
     *        参考资料：http://blogold.chinaunix.net/u2/60011/showart.php?id=1018502
     *
     * 2、timer在这里的作用
     *      timer是用来防抖的，我们知道，触摸屏处理分为两个时间段，一个是由按下中断触发的四次AD转换的时间A，一个是4次AD转换完
     成后将AD数据存入FIFO的时间B，在时间A，没有打开抬起中断，
     *      也就是说如果在这段时间有抬起事件，也不会触发中断，不会影响AD的转换。在时间B，打开抬起中断，打开定时器延时触发
     touch_timer_fire，如果在延时这段时间，有抬起事件发生
     *       则touch_timer_fire不会将前面的数据存入到FIFO中，否则写入FIFO，表示值有效。
     *
     *
     */
```