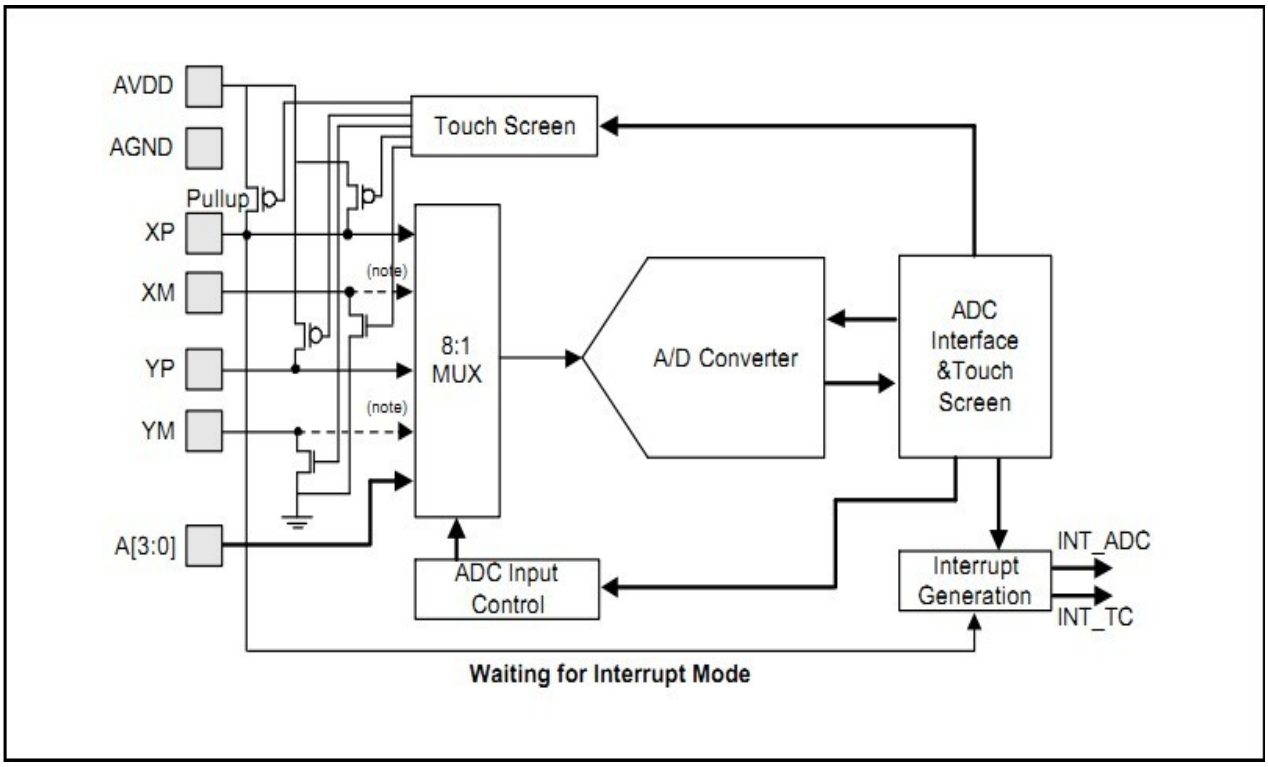
努力成为 linux kernel hacker 的人李万鹏原创作品，为梦而战。转载请标明出处

http://blog.csdn.net/woshixingaaa/archive/2011/05/18/6429002.aspx

S3C2440 内部 ADC 结构图：



对于 s3c2440 来说，实现 A/D 转换比较简单，主要应用的是 ADC 控制寄存器 ADCCON 和 ADC 转换数据寄存器 ADCDAT0。寄存器 ADCDAT0 的低 10 位用于存储 A/D 转换后的数据。寄存器 ADCCON 的第 15 位用于标识 A/D 转换是否结束。第 14 位用于使能是否进行预分频，而第 6 位到第 13 位则存储的是预分频数值，因为 A/D 转换的速度不能太快，所以要通过预分频处理才可以得到正确的 A/D 转换速度，如我们想要得到 A/D 转换频率为 1MHz，则预分频的值应为 49。第 3 位到第 5 位表示的是 A/D 转换的通道选择。第 2 位可以实现 A/D 转换的待机模式。第 1 位用于是否通过读取操作来使能 A/D 转换的开始。第 0 位则是在第 1 位被清零的情况下用于开启 A/D 转换。

驱动代码：

```
#include <linux/init.h>
#include <linux/module.h>
#include <linux/fs.h>
#include <mach/hardware.h>
#include <asm/io.h>
#include <linux/interrupt.h>
#include <asm/irq.h>
#include <linux/irq.h>
```

```c
#include <linux/miscdevice.h>
#include <linux/types.h>
#include <linux/clk.h>
#include <linux/errno.h>
#include <asm/uaccess.h>
#include <mach/regs-clock.h>
#include <plat/regs-adc.h>
#include <linux/kernel.h>
#define ADC_MINOR 100
#define ADC_NAME "lwp-adc"
struct clk *adc_clk;
int adc_base;
int adc_finish = 0;
static int adc_data;
DECLARE_WAIT_QUEUE_HEAD(adc_wait);

static irqreturn_t adc_interrupt(int irq, void *dev_id){
    if(!adc_finish){
        adc_data = readl(adc_base + S3C2410_ADCDAT0) & 0x3ff;                    //ad 转换结束会产生中断，此时读取 S3C2410_ADCDAT0 的 0～9 位，来获取数据
        adc_finish = 1;
        wake_up_interruptible(&adc_wait);                                        //唤醒等待其上的进程
    }
    return IRQ_HANDLED;
}

int myadc_open(struct inode *inode, struct file *file){
    int ret;
    ret = request_irq(IRQ_ADC, adc_interrupt, IRQF_SHARED, ADC_NAME, 1);         //这里注册中断，ad 转换结束通知 cpu 有两种方式，一种靠 cpu 轮询标志位，一种靠中断
    if(ret){
        printk("IRQ %d can't get/n", IRQ_ADC);
        return -1;
    }
    return 0;
}

int myadc_close(struct inode *inode, struct file *file){
    return 0;
        return -ENOENT;
}

void start_adc(){                                    //开始 ad 转换
    int tmp;
    tmp = 0xff<<6 | 1<<14;                           //设置预分频使能，值为 255，使用通道 AIN0
    writel(tmp ,adc_base + S3C2410_ADCCON);
    tmp = readl(adc_base + S3C2410_ADCCON);
    tmp |= 1<<0;                                     //使能 AD 转换
```

```
55.    writel(tmp, adc_base + S3C2410_ADCCON);
56.}
57.
58.ssize_t myadc_read(struct file *filp, char __user *buff, size_t count, loff_t *offp){
59.    start_adc();                                        //要读数据开始 adc 转换
60.    wait_event_interruptible(adc_wait, adc_finish);                    //等待转换结束
61.    adc_finish = 0;
62.    copy_to_user(buff, (char *)&adc_data, sizeof(adc_data));            //将获得的数据拷贝到用户空间，
   数据会在中断程序中获得
63.    return sizeof(adc_data);
64.}
65.
66.static struct file_operations adc_ops = {
67.    .owner = THIS_MODULE,
68.    .open = myadc_open,
69.    .release = myadc_close,
70.    .read = myadc_read,
71.};
72.
73.static struct miscdevice adc_misc = {
74.    .name = ADC_NAME,
75.    .minor = ADC_MINOR,
76.    .fops = &adc_ops,
77.};
78.
79.static int __init my_adc_init(void){
80.    unsigned int ret;
81.    adc_clk = clk_get(NULL,"adc");                    //由于 ad 转换需要时钟，所以这里获取时钟
82.    if(!adc_clk){
83.        printk(KERN_ERR "fail to find adc clk resource!/n");
84.        ret = -1;
85.        goto err_clk;
86.    }
87.    clk_enable(adc_clk);                            //使能 adc 的时钟
88.    adc_base = ioremap(S3C2410_PA_ADC, 20);            //获得 ADC 控制寄存器的虚拟地址
89.    if(adc_base == 0){
90.        printk(KERN_ERR "fail to ioremap!/n");
91.        ret = -1;
92.        goto err_nomap;
93.    }
94.    ret = misc_register(&adc_misc);                    //注册这个 adc 为混杂设备
95.    if(IS_ERR(ret)){
96.        goto err_register;
97.    }
98.    return 0;
99.
```

```
err_register:
    iounmap(adc_base);
err_nomap:
    clk_disable(adc_clk);
    clk_put(adc_clk);
err_clk:
    return ret;
}

static void __exit my_adc_exit(void){
    misc_deregister(&adc_misc);
    free_irq(IRQ_ADC, 1);
    iounmap(adc_base);
    clk_disable(adc_clk);
    clk_put(adc_clk);
}

module_init(my_adc_init);
module_exit(my_adc_exit);
MODULE_AUTHOR("liwanpeng");
MODULE_LICENSE("GPL");
```

测试代码：

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>

int main(int argc, char **argv)
{
    int fd;
    fd = open("/dev/lwp-adc", 0);
    if(fd < 0)
    {
        printf("Open ADC Device Faild!/n");
        exit(1);
    }
    while(1)
    {
        int ret;
        int data;
        ret = read(fd, &data, sizeof(data));
        if(ret != sizeof(data))
        {
            if(errno != EAGAIN)
            {
                printf("Read ADC Device Faild!/n");
            }
```

```
25.         continue;
26.     }
27.     else
28.     {
29.         printf("Read ADC value is: %d/n", data);
30.     }
31.   }
32.   close(fd);
33.   return 0;
34.}
```

实验效果：