

iTOP-4412-irq-fasync 异步通知

异步通知 fasync 应用于系统调用 signal 和 sigaction 函数，简单的说，signal 函数就是让一个信号与一个函数对应，每当接收到这个信号就会调用相应的函数。

那么什么是异步通知？异步通知类似于中断的机制，当设备可写时，设备驱动函数发送一个信号给内核，告知内核有数据可读，在条件不满足之前，并不会造成阻塞。而不像之前学的阻塞型 IO 和 poll，它们是调用函数进去检查，条件不满足时还会造成阻塞。

1.1 平台文件注册

在内核源码目录，使用 “vi arch/arm/mach-exynos/mach-itop4412.c” 命令打开平台文件。搜索关键词 “struct platform_device s3c_device_buzzer_ctl” 找到 buzzer 配置。然后在它的下面添加如下信息。

```
struct platform_device s3c_device_irq_test = {
    .name    = "irq_test",
    .id      = -1,
};
```

如下图。

```
#ifdef CONFIG_BUZZER_CTL
struct platform_device s3c_device_buzzer_ctl = {
    .name    = "buzzer_ctl",
    .id      = -1,
};
#endif

//add by neo 20180103

struct platform_device s3c_device_irq_test = {
    .name    = "irq_test",
    .id      = -1,
};

//end add

#ifdef CONFIG_ADC_CTL
struct platform_device s3c_device_adc_ctl = {
    .name    = "adc_ctl",
    .id      = -1,
};
#endif
```

保存，退出。

再次使用 “vi arch/arm/mach-exynos/mach-itop4412.c” 命令打开平台文件。搜索关键词 “&s3c_device_buzzer_ctl”，在这一行下面添加：

```
&s3c_device_irq_test,
```

如下图。

```
#ifdef CONFIG_BUZZER_CTL
    &s3c_device_buzzer_ctl,
#endif

//add by meo 20180103
    &s3c_device_irq_test,
//end add

#ifdef CONFIG_ADC_CTL
    &s3c_device_adc_ctl,
#endif
```

保存，退出。

使用 “vi drivers/char/Kconfig ” 命令打开 Kconfig 配置文件。搜索关键词 “BUZZER_CTL” ，在该段的下面添加如下信息。

```
config IRQ_TEST
    bool "Enable IRQ_TEST config"
    default Y
    help
        Enable IRQ_TEST config
```

如下图。

```
        Enable LEDS config

config BUZZER_CTL
    bool "Enable BUZZER config"
    default n
    help
        Enable BUZZER config

#add by meo 20180103
config IRQ_TEST
    bool "Enable IRQ_TEST config"
    default Y
    help
        Enable IRQ_TEST config
#end add

config ADC_CTL
    bool "ADC driver for iTOP4412"
    select S3C_ADC
```

保存，退出。

在内核目录下使用 “make menuconfig” 命令打开内核配置界面，如下图。

```
root@ubuntu:/home/topeet/android4.0/iTop4412_Kernel_3.0#
root@ubuntu:/home/topeet/android4.0/iTop4412_Kernel_3.0# make menuconfig
```

进入到 “Device Drivers” , 如下图。

然后进入到 “Input device support” 界面，如下图。

3

[illegible]

然后保存退出内核配置界面。使用“make zImage”命令编译内核。如下图。

```
*** End of the configuration.
*** Execute 'make' to start the build or try 'make help'.

root@ubuntu:/home/topeet/android4.0/iTop4412_Kernel_3.0#
root@ubuntu:/home/topeet/android4.0/iTop4412_Kernel_3.0#
root@ubuntu:/home/topeet/android4.0/iTop4412_Kernel_3.0#
root@ubuntu:/home/topeet/android4.0/iTop4412_Kernel_3.0# make zImage
```

编译完成后如下图。

```
make CONFIG_DEBUG_SECTION_MISMATCH=y'
GEN      .version
CHK      include/generated/compile.h
UPD      include/generated/compile.h
CC      init/version.o
LD      init/built-in.o
LD      .tmp_vmlinux1
KSYM     .tmp_kallsyms1.S
AS      .tmp_kallsyms1.o
LD      .tmp_vmlinux2
KSYM     .tmp_kallsyms2.S
AS      .tmp_kallsyms2.o
LD      vmlinux
SYSMAP   System.map
SYSMAP   .tmp_System.map
OBJCOPY  arch/arm/boot/Image
Kernel:  arch/arm/boot/Image is ready
GZIP     arch/arm/boot/compressed/piggy.gzip
AS      arch/arm/boot/compressed/piggy.gzip.o
SHIPPED  arch/arm/boot/compressed/liblfuncs.S
AS      arch/arm/boot/compressed/liblfuncs.o
LD      arch/arm/boot/compressed/vmlinux
OBJCOPY  arch/arm/boot/zImage
Kernel:  arch/arm/boot/zImage is ready
root@ubuntu:/home/topeet/android4.0/iTop4412 Ke
```

然后把编译生成的 zImage （在 arch/arm/boot 目录下）烧写到 iTOP-4412 开发板上，烧写完成后启动开发板。

1.2 驱动程序

```
/*以后写驱动可以讲头文件一股脑的加载代码前面*/
#include <linux/init.h>
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/fs.h>
#include <mach/gpio.h>
#include <plat/gpio-cfg.h>
#include <linux/miscdevice.h>
#include <linux/platform_device.h>
#include <mach/regs-gpio.h>
#include <asm/io.h>
#include <linux/regulator/consumer.h>
#include <linux/delay.h>

/*中断函数头文件*/
#include <linux/interrupt.h>
#include <linux/irq.h>

#define IRQ_DEBUG
#ifdef IRQ_DEBUG
#define DPRINTK(x...) printk("IRQ_CTL DEBUG:" x)
#else
#define DPRINTK(x...)
#endif

#define DRIVER_NAME "irq_test"
//添加头文件
#include <linux/poll.h>
#include <linux/sched.h>

//添加的定义 :
static volatile int press = 0;
static unsigned char key_val;
static DECLARE_WAIT_QUEUE_HEAD(key_button);

//fasync 定义结构体
static struct fasync_struct *button_key;
```

//中断函数

```
static irqreturn_t eint9_interrupt(int irq, void *dev_id) {
    press = 1;                // 表示中断发生了
    key_val = 0x01; //read 返回值 ,
    wake_up_interruptible(&key_button); // 唤醒休眠的进程
    kill_fasync (&button_key, SIGIO, POLL_IN); //发送信号 SIGIO 信号给 fasync_struct 结构体所描述的
                                                //PID , 触发应用程序的 SIGIO 信号处理函数

    return IRQ_RETVAL(IRQ_HANDLED);
}
```

```
static irqreturn_t eint10_interrupt(int irq, void *dev_id) {
    press = 1;                // 表示中断发生了
    key_val = 0x02;
    wake_up_interruptible(&key_button); /* 唤醒休眠的进程 */
    kill_fasync (&button_key, SIGIO, POLL_IN); //发送信号 SIGIO 信号给 fasync_struct 结构体所描述的
                                                //PID , 触发应用程序的 SIGIO 信号处理函数

    return IRQ_RETVAL(IRQ_HANDLED);
}
```

```
static int itop_irq_open(struct inode *inode, struct file *file)
{
    return 0;
}
```

```
static int itop_irq_close(struct inode *inode, struct file *file)
{
    printk(" %s  !! !\n", __FUNCTION__);
    return 0;
}
```

```
ssize_t itop_irq_read(struct file *file, char __user *buf, size_t size, loff_t *ppos)
{
    if (size != 1)
        return -EINVAL;
    //如果没有按键动作, 休眠
    wait_event_interruptible(key_button, press);
    // 如果有按键动作, 返回键值
    copy_to_user(buf, &key_val, 1);
}
```

```
    press = 0;
    return 1;
}

//itop_irq_fasync 函数
static int itop_irq_fasync (int fd, struct file *filp, int on)
{
    printk("driver: itop_irq_fasync\n");
    //初始化 fasync_struct 结构体 (fasync_struct->fa_file->f_owner->pid)
    return fasync_helper (fd, filp, on, &button_key);
}

static struct file_operations itop_irq_fops = {
    .owner    = THIS_MODULE,    //这是一个宏，推向编译模块时自动创建的__this_module 变量
    .open     = itop_irq_open,
    .read     = itop_irq_read,
    .release  = itop_irq_close,
    .fasync   = itop_irq_fasync,
};

static struct miscdevice itop_irq_dev = {
    .minor    = MISC_DYNAMIC_MINOR,
    .fops     = &itop_irq_fops,
    .name     = "irq_test",
};

// probe 函数
static int irq_probe(struct platform_device *pdev)
{
    int ret;
    char *banner = "irq_test Initialize\n";
    printk(" %s  !!!\n",__FUNCTION__);
    printk(banner);

    ret = gpio_request(EXYNOS4_GPX1(1), "EINT9");
    if (ret) {
        printk("%s: request GPIO %d for EINT9 failed, ret = %d\n", DRIVER_NAME,
```



```
        EXYNOS4_GPX1(1), ret);
    return ret;
}
s3c_gpio_cfgpin(EXYNOS4_GPX1(1), S3C_GPIO_SFN(0xF));
s3c_gpio_setpull(EXYNOS4_GPX1(1), S3C_GPIO_PULL_UP);
gpio_free(EXYNOS4_GPX1(1));

ret = gpio_request(EXYNOS4_GPX1(2), "EINT10");
if (ret) {
    printk("%s: request GPIO %d for EINT10 failed, ret = %d\n", DRIVER_NAME,
        EXYNOS4_GPX1(2), ret);
    return ret;
}
s3c_gpio_cfgpin(EXYNOS4_GPX1(2), S3C_GPIO_SFN(0xF));
s3c_gpio_setpull(EXYNOS4_GPX1(2), S3C_GPIO_PULL_UP);
gpio_free(EXYNOS4_GPX1(2));

ret = request_irq(IRQ_EINT(9), eint9_interrupt,
    IRQ_TYPE_EDGE_FALLING /*IRQF_TRIGGER_FALLING*/, "eint9", pdev);
if (ret < 0) {
    printk("Request IRQ %d failed, %d\n", IRQ_EINT(9), ret);
    goto exit;
}

ret = request_irq(IRQ_EINT(10), eint10_interrupt,
    IRQ_TYPE_EDGE_FALLING /*IRQF_TRIGGER_FALLING*/, "eint10", pdev);
if (ret < 0) {
    printk("Request IRQ %d failed, %d\n", IRQ_EINT(10), ret);
    goto exit;
}

ret = misc_register(&itop_irq_dev);
return 0;

exit:
    return ret;
}

static int irq_remove (struct platform_device *pdev)
```

```
{
    printk(" %s  !!!\n",__FUNCTION__);
    misc_deregister(&itop_irq_dev);
    free_irq(IRQ_EINT(9),pdev);
    free_irq(IRQ_EINT(10),pdev);

    return 0;
}

static int irq_suspend (struct platform_device *pdev, pm_message_t state)
{
    printk(" %s  !!!\n",__FUNCTION__);
    DPRINTK("irq suspend:power off!\n");
    return 0;
}

static int irq_resume (struct platform_device *pdev)
{
    printk(" %s  !!!\n",__FUNCTION__);
    DPRINTK("irq resume:power on!\n");
    return 0;
}

static struct platform_driver irq_driver = {
    .probe = irq_probe,
    .remove = irq_remove,
    .suspend = irq_suspend,
    .resume = irq_resume,
    .driver = {
        .name = DRIVER_NAME,
        .owner = THIS_MODULE,
    },
};

static void __exit irq_test_exit(void)
{
    printk(" %s  !!!\n",__FUNCTION__);
    platform_driver_unregister(&irq_driver);
}
```

```
static int __init irq_test_init(void)
{
    printk(" %s  !!!\n",__FUNCTION__);
    return platform_driver_register(&irq_driver);
}

module_init(irq_test_init);
module_exit(irq_test_exit);

MODULE_LICENSE("Dual BSD/GPL");
```

1.3 应用程序

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
#include <poll.h>
#include <signal.h>
#include <sys/types.h>
#include <unistd.h>
#include <fcntl.h>

int fd;
void my_signal_fun(int signum)
{
    unsigned char key_val;
    read(fd, &key_val, 1);
    if ( key_val == 0x01 )
        printf("press home button ! \n" );
    else if ( key_val == 0x02)
        printf("press back button ! \n");
    else
        ;
}

int main(int argc, char **argv)
{

```

```
unsigned char key_val;
int ret;
int Oflags;

signal(SIGIO, my_signal_fun);

fd = open("/dev/irq_test", O_RDWR);
if (fd < 0)
{
    printf("can't open!\n");
}

fcntl(fd, F_SETOWN, getpid());

Oflags = fcntl(fd, F_GETFL);

fcntl(fd, F_SETFL, Oflags | FASYNC);

while (1)
{
    sleep(1000);
}

return 0;
}
```

在应用层启用异步通知只三个步骤：

1) signal(SIGIO, sig_handler);

调用 signal 函数，让指定的信号 SIGIO 与处理函数 sig_handler 对应。

2) fcntl(fd, F_SET_OWNER, getpid());

指定一个进程作为文件的“属主(filp->owner)”，这样内核才知道信号要发给哪个进程。

3) Oflags = fcntl(fd, F_GETFL);

fcntl(fd, F_SETFL, Oflags | FASYNC);

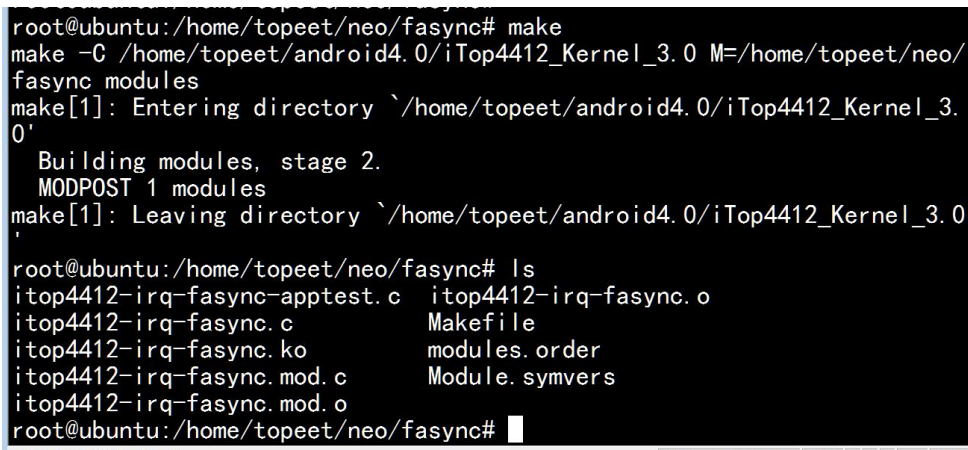
在设备文件中添加 FASYNC 标志，驱动中就会调用将要实现的 test_fasync 函数。

三个步骤执行后，一旦有信号产生，相应的进程就会收到。

1.4 编译运行测试

1.4.1 驱动编译

驱动程序编译很简单。把驱动程序 “itop4412-irq-fasync.c” 和 Makefile 文件上传到同一目录，执行 “make” 命令。如下图。



```
root@ubuntu:/home/topeet/neo/fasync# make
make -C /home/topeet/android4.0/iTop4412_Kernel_3.0 M=/home/topeet/neo/
fasync modules
make[1]: Entering directory `/home/topeet/android4.0/iTop4412_Kernel_3.
0'
Building modules, stage 2.
MODPOST 1 modules
make[1]: Leaving directory `/home/topeet/android4.0/iTop4412_Kernel_3.0'

root@ubuntu:/home/topeet/neo/fasync# ls
itop4412-irq-fasync-apptest.c  itop4412-irq-fasync.o
itop4412-irq-fasync.c          Makefile
itop4412-irq-fasync.ko         modules.order
itop4412-irq-fasync.mod.c      Module.symvers
itop4412-irq-fasync.mod.o
root@ubuntu:/home/topeet/neo/fasync#
```

通过 U 盘挂载、tftp 或者 nfs 功能把 “itop4412-irq-fasync.ko” 文件上传到开发板。

1.4.2 应用程序编译

把应用程序 “itop4412-irq-fasync-apptest.c” 上传到 ubuntu 系统。使用 “arm-none-linux-gnueabi-gcc -o itop4412-irq-fasync-apptest itop4412-irq-fasync-apptest.c -static” 命令来静态编译应用程序。如下图。


```

root@ubuntu:/home/topeet/neo/fasync# arm-none-linux-gnueabi-gcc -o itop
4412-irq-fasync-apptest itop4412-irq-fasync-apptest.c -static
root@ubuntu:/home/topeet/neo/fasync# ls
itop4412-irq-fasync-apptest  itop4412-irq-fasync.mod.o
itop4412-irq-fasync-apptest.c  itop4412-irq-fasync.o
itop4412-irq-fasync.c          Makefile
itop4412-irq-fasync.ko         modules.order
itop4412-irq-fasync.mod.c      Module.symvers
root@ubuntu:/home/topeet/neo/fasync#

```

通过 U 盘挂载、tftp 或者 nfs 功能把 “itop4412-irq-fasync-apptest ” 文件上传到开发板。如下图。

```

[root@iT0P-4412]# ls
bin                mnt
dev                proc
etc               /sbin
itop4412-irq-fasync-apptest  sys
itop4412-irq-fasync.ko      tmp
lib                  usr
linuxrc             var
[root@iT0P-4412]#

```

1.4.3 运行测试

使用 “insmod itop4412-irq-fasync.ko” 命令来加载驱动程序。如下图。

```

[root@iT0P-4412]#
[root@iT0P-4412]# insmod itop4412-irq-fasync.ko
[ 874.659251] irq_test_init !!!
[ 874.661692] irq_probe !!!
[ 874.664086] irq_test Initialize
[root@iT0P-4412]#

```

由上图可知，进入 probe 函数，并且 gpio 注册成功。

使用 “chmod 777 itop4412-irq-fasync-apptest” 命令，修改应用程序的权限。使用 “./itop4412-irq-fasync-apptest” 命令，运行应用程序。如下图。

```
[root@iTOP-4412]#  
[root@iTOP-4412]# chmod 777 itop4412-irq-fasync-apptest  
[root@iTOP-4412]# ./itop4412-irq-fasync-apptest  
[ 1677.205144] driver: itop_irq_fasync  
press back button !  
press home button !  
press back button !  
press home button !  
press home button !  
press home button !  
^C[ 1686.393782] driver: itop_irq_fasync  
[ 1686.395940] itop_irq_close !!!  
  
[root@iTOP-4412]# █
```

运行应用程序之后，按下开发板上的“back”键和“home”键。会有相应打印。长时间不按键，超级终端没有任何打印。此时再次按键，依然有打印信息。按“ctrl+c”停止运行应用程序。具体效果，如上图。

使用“rmmod itop4412-irq-fasync”命令卸载驱动。如下图。

```
[root@iTOP-4412]#  
[root@iTOP-4412]# rmmod itop4412-irq-fasync  
[ 2001.935561] irq_test_exit !!!  
[ 2001.937503] irq_remove !!!  
[root@iTOP-4412]# █
```

测试完成。

联系方式

北京迅为电子有限公司致力于嵌入式软硬件设计，是高端开发平台以及移动设备方案提供商；基于多年的技术积累，在工控、仪表、教育、医疗、车载等领域通过 OEM/ODM 方式为客户创造价值。

iTOP-4412 开发板是迅为电子基于三星最新四核处理器 Exynos4412 研制的一款实验开发平台，可以通过该产品评估 Exynos 4412 处理器相关性能，并以此为基础开发出用户需要的特定产品。

本手册主要介绍 iTOP-4412 开发板的使用方法，旨在帮助用户快速掌握该产品的应用特点，通过对开发板进行后续软硬件开发，衍生出符合特定需求的应用系统。

如需平板电脑案支持，请访问迅为平板方案网“<http://www.topeet.com>”，我司将有能力为您提供全方位的技术服务，保证您产品设计无忧！

本手册将持续更新，并通过多种方式发布给新老用户，希望迅为电子的努力能给您的学习和开发带来帮助。

迅为电子

2018 年 1 月