# Linux驱动\_GPIO子系统

## 1.向GPIO子系统注册

### 1.1 gpio\_chip结构体初始化

GPIO子系统主要结构体为gpio\_chip。主要定义了gpio相关操作的通用函数，例如设置输入输出，拉高拉低，获取gpio对应的中断号等。

struct gpio\_chip **{**

const char **\***label**;**

struct device **\***dev**;**

struct module **\***owner**;**

int **(\***request**)(**struct gpio\_chip **\***chip**,**

unsigned offset**);**

void **(\***free**)(**struct gpio\_chip **\***chip**,**

unsigned offset**);**

int **(\***direction\_input**)(**struct gpio\_chip **\***chip**,**

unsigned offset**);**

int **(\***get**)(**struct gpio\_chip **\***chip**,**

unsigned offset**);**

int **(\***direction\_output**)(**struct gpio\_chip **\***chip**,**

unsigned offset**,** int value**);**

int **(\***set\_debounce**)(**struct gpio\_chip **\***chip**,**

unsigned offset**,** unsigned debounce**);**

void **(\***set**)(**struct gpio\_chip **\***chip**,**

unsigned offset**,** int value**);**

int **(\***to\_irq**)(**struct gpio\_chip **\***chip**,**

unsigned offset**);**

void **(\***dbg\_show**)(**struct seq\_file **\***s**,**

struct gpio\_chip **\***chip**);**

int base**;**

u16 ngpio**;**

const char **\***const **\***names**;**

unsigned can\_sleep**:**1**;**

unsigned exported**:**1**;**

**};**

2440 gpio在3.4内核中的注册函数为：

core\_initcall**(**samsung\_gpiolib\_init**);**

在samsung\_gpiolib\_init中会继续调用如下函数：

s3c24xx\_gpiolib\_add\_chips**(**s3c24xx\_gpios**,**ARRAY\_SIZE**(**s3c24xx\_gpios**),** S3C24XX\_VA\_GPIO**);**

s3c24xx\_gpios的定义如下：

struct samsung\_gpio\_chip **{**

struct gpio\_chip chip**;**

struct samsung\_gpio\_cfg **\***config**;**

struct samsung\_gpio\_pm **\***pm**;**

void \_\_iomem **\***base**;**

int irq\_base**;**

int group**;**

spinlock\_t lock**;**

#ifdef CONFIG\_PM

u32 pm\_save**[**4**];**

#endif

**};**

struct samsung\_gpio\_chip s3c24xx\_gpios**[]** **=** **{**

#ifdef CONFIG\_PLAT\_S3C24XX

**{**

**.**config **=** **&**s3c24xx\_gpiocfg\_banka**,**

**.**chip **=** **{**

**.**base **=** S3C2410\_GPA**(**0**),**

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

**.**label **=** "GPIOA"**,**

**.**ngpio **=** 24**,**

**.**direction\_input **=** s3c24xx\_gpiolib\_banka\_input**,**

**.**direction\_output **=** s3c24xx\_gpiolib\_banka\_output**,**

**},**

**},** **{**

**.**chip **=** **{**

**.**base **=** S3C2410\_GPB**(**0**),**

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

**.**label **=** "GPIOB"**,**

**.**ngpio **=** 16**,**

**},**

**},** **{**

**.**chip **=** **{**

**.**base **=** S3C2410\_GPC**(**0**),**

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

**.**label **=** "GPIOC"**,**

**.**ngpio **=** 16**,**

**},**

**},** **{**

**.**chip **=** **{**

**.**base **=** S3C2410\_GPD**(**0**),**

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

**.**label **=** "GPIOD"**,**

**.**ngpio **=** 16**,**

**},**

**},** **{**

**.**chip **=** **{**

**.**base **=** S3C2410\_GPE**(**0**),**

**.**label **=** "GPIOE"**,**

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

**.**ngpio **=** 16**,**

**},**

**},** **{**

**.**chip **=** **{**

**.**base **=** S3C2410\_GPF**(**0**),**

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

**.**label **=** "GPIOF"**,**

**.**ngpio **=** 8**,**

**.**to\_irq **=** s3c24xx\_gpiolib\_fbank\_to\_irq**,**

**},**

**},** **{**

**.**irq\_base **=** IRQ\_EINT8**,**

**.**chip **=** **{**

**.**base **=** S3C2410\_GPG**(**0**),**

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

**.**label **=** "GPIOG"**,**

**.**ngpio **=** 16**,**

**.**to\_irq **=** samsung\_gpiolib\_to\_irq**,**

**},**

**},** **{**

**.**chip **=** **{**

**.**base **=** S3C2410\_GPH**(**0**),**

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

**.**label **=** "GPIOH"**,**

**.**ngpio **=** 11**,**

**},**

**},**

/\* GPIOS for the S3C2443 and later devices. \*/

**{**

**.**base **=** S3C2440\_GPJCON**,**

**.**chip **=** **{**

**.**base **=** S3C2410\_GPJ**(**0**),**

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

**.**label **=** "GPIOJ"**,**

**.**ngpio **=** 16**,**

**},**

**},** **{**

**.**base **=** S3C2443\_GPKCON**,**

**.**chip **=** **{**

**.**base **=** S3C2410\_GPK**(**0**),**

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

**.**label **=** "GPIOK"**,**

**.**ngpio **=** 16**,**

**},**

**},** **{**

**.**base **=** S3C2443\_GPLCON**,**

**.**chip **=** **{**

**.**base **=** S3C2410\_GPL**(**0**),**

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

**.**label **=** "GPIOL"**,**

**.**ngpio **=** 15**,**

**},**

**},** **{**

**.**base **=** S3C2443\_GPMCON**,**

**.**chip **=** **{**

**.**base **=** S3C2410\_GPM**(**0**),**

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

**.**label **=** "GPIOM"**,**

**.**ngpio **=** 2**,**

**},**

**},**

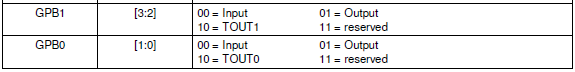
#endif

**};**

注意chip部分不一样的地方

（1）GPA的in/out函数单独写出来了，因为GPA的功能设置bit位只有一个bit，其他GPIO则是两个bit。





（2）GPF和GPG具备外部中断功能，所以有to\_irq函数。但是因为GPF的后4个中断对用的是IRQ\_EINT4t7，所以处理会有不一样。

s3c24xx\_gpiolib\_add\_chips的另外一个参数为GPIO寄存器对应的虚拟地址。

#define S3C24XX\_VA\_GPIO ((S3C24XX\_PA\_GPIO - S3C24XX\_PA\_UART) + S3C24XX\_VA\_UART)

s3c24xx\_gpiolib\_add\_chip函数代码如下：

static void \_\_init s3c24xx\_gpiolib\_add\_chips**(**struct samsung\_gpio\_chip **\***chip**,**int nr\_chips**,** void \_\_iomem **\***base**)**

**{**

int i**;**

//gpio\_chip是gpio子系统的核心结构体，对应的操作都放在该结构体中

struct gpio\_chip **\***gc **=** **&**chip**->**chip**;**

//依次将所有的GPIO组添加到GPIO子系统中

**for** **(**i **=** 0 **;** i **<** nr\_chips**;** i**++,** chip**++)** **{**

/\* skip banks not present on SoC \*/

**if** **(**chip**->**chip**.**base **>=** S3C\_GPIO\_END**)**

**continue;**

//config函数主要用于非GPIO子系统的GPIO操作。

**if** **(!**chip**->**config**)**

chip**->**config **=** **&**s3c24xx\_gpiocfg\_default**;**

**if** **(!**chip**->**pm**)**

chip**->**pm **=** \_\_gpio\_pm**(&**samsung\_gpio\_pm\_2bit**);**

//设置GPIO操作寄存器地址

**if** **((**base **!=** **NULL)** **&&** **(**chip**->**base **==** **NULL))**

chip**->**base **=** base **+** **((**i**)** **\*** 0x10**);**

//设置输入输出操作函数

**if** **(!**gc**->**direction\_input**)**

gc**->**direction\_input **=** samsung\_gpiolib\_2bit\_input**;**

**if** **(!**gc**->**direction\_output**)**

gc**->**direction\_output **=** samsung\_gpiolib\_2bit\_output**;**

//添加到GPIO子系统

samsung\_gpiolib\_add**(**chip**);**

**}**

**}**

static void \_\_init samsung\_gpiolib\_add**(**struct samsung\_gpio\_chip **\***chip**)**

**{**

struct gpio\_chip **\***gc **=** **&**chip**->**chip**;**

int ret**;**

BUG\_ON**(!**chip**->**base**);**

BUG\_ON**(!**gc**->**label**);**

BUG\_ON**(!**gc**->**ngpio**);**

spin\_lock\_init**(&**chip**->**lock**);**

**if** **(!**gc**->**direction\_input**)**

gc**->**direction\_input **=** samsung\_gpiolib\_2bit\_input**;**

**if** **(!**gc**->**direction\_output**)**

gc**->**direction\_output **=** samsung\_gpiolib\_2bit\_output**;**

//GPIO寄存器设置/读取函数

**if** **(!**gc**->**set**)**

gc**->**set **=** samsung\_gpiolib\_set**;**

**if** **(!**gc**->**get**)**

gc**->**get **=** samsung\_gpiolib\_get**;**

#ifdef CONFIG\_PM

**if** **(**chip**->**pm **!=** **NULL)** **{**

**if** **(!**chip**->**pm**->**save **||** **!**chip**->**pm**->**resume**)**

printk**(**KERN\_ERR "gpio: %s has missing PM functions\n"**,**

gc**->**label**);**

**}** **else**

printk**(**KERN\_ERR "gpio: %s has no PM function\n"**,** gc**->**label**);**

#endif

//这里将GPIO组添加到GPIO子系统

ret **=** gpiochip\_add**(**gc**);**

**if** **(**ret **>=** 0**)**

s3c\_gpiolib\_track**(**chip**);**

**}**

### 1.2 gpiochip\_add流程

gpiochip\_add将每个gpio分别添加到gpio\_desc结构体数组中，并创建了/sys/class/gpio/gpiochipN目录。

gpiochip\_add函数源代码如下：

int gpiochip\_add**(**struct gpio\_chip **\***chip**)**

**{**

unsigned long flags**;**

int status **=** 0**;**

unsigned id**;**

int base **=** chip**->**base**;**

//判断GPIO口是否超过了最大值

**if** **((!**gpio\_is\_valid**(**base**)** **||** **!**gpio\_is\_valid**(**base **+** chip**->**ngpio **-** 1**))**

**&&** base **>=** 0**)** **{**

status **=** **-**EINVAL**;**

**goto** fail**;**

**}**

spin\_lock\_irqsave**(&**gpio\_lock**,** flags**);**

**if** **(**base **<** 0**)** **{**

base **=** gpiochip\_find\_base**(**chip**->**ngpio**);**

**if** **(**base **<** 0**)** **{**

status **=** base**;**

**goto** unlock**;**

**}**

chip**->**base **=** base**;**

**}**

//确保所有的gpio\_desc对应的结构体未被占用

**for** **(**id **=** base**;** id **<** base **+** chip**->**ngpio**;** id**++)** **{**

**if** **(**gpio\_desc**[**id**].**chip **!=** **NULL)** **{**

status **=** **-**EBUSY**;**

**break;**

**}**

**}**

**if** **(**status **==** 0**)** **{**

**for** **(**id **=** base**;** id **<** base **+** chip**->**ngpio**;** id**++)** **{**

//将gpio\_chip结构体添加到gpio\_desc数组中

//每个gpio都有一个对应的gpio\_desc位置

gpio\_desc**[**id**].**chip **=** chip**;**

gpio\_desc**[**id**].**flags **=** **!**chip**->**direction\_input

**?** **(**1 **<<** FLAG\_IS\_OUT**)**

**:** 0**;**

**}**

**}**

of\_gpiochip\_add**(**chip**);**

unlock**:**

spin\_unlock\_irqrestore**(&**gpio\_lock**,** flags**);**

**if** **(**status**)**

**goto** fail**;**

//创建 /sys/class/gpio/gpiochipN

status **=** gpiochip\_export**(**chip**);**

**if** **(**status**)**

**goto** fail**;**

//打印添加信息

pr\_info**(**"gpiochip\_add: registered GPIOs %d to %d on device: %s\n"**,**

chip**->**base**,** chip**->**base **+** chip**->**ngpio **-** 1**,**

chip**->**label **?** **:** "generic"**);**

**return** 0**;**

fail**:**

/\* failures here can mean systems won't boot... \*/

pr\_err**(**"gpiochip\_add: gpios %d..%d (%s) failed to register\n"**,**

chip**->**base**,** chip**->**base **+** chip**->**ngpio **-** 1**,**

chip**->**label **?** **:** "generic"**);**

**return** status**;**

**}**

其中gpiochi\_export主要完成了/sys/class/gpio/gpiochipN目录及相关文件的添加。

static int gpiochip\_export**(**struct gpio\_chip **\***chip**)**

**{**

int status**;**

struct device **\***dev**;**

**if** **(!**gpio\_class**.**p**)**

**return** 0**;**

mutex\_lock**(&**sysfs\_lock**);**

//创建 /sys/class/gpio/gpiochipN

//gpiochipN表示的就是一个gpio\_chip,用来管理和控制一组gpio端口的控制器

//MKDEV(0,0)不在/dev下产生设备文件

dev **=** device\_create**(&**gpio\_class**,** chip**->**dev**,** MKDEV**(**0**,** 0**),** chip**,**

"gpiochip%d"**,** chip**->**base**);**

**if** **(!**IS\_ERR**(**dev**))** **{**

//创建gpiochipN目录下的属性文件base label ngpio

status **=** sysfs\_create\_group**(&**dev**->**kobj**,**

**&**gpiochip\_attr\_group**);**

**}** **else**

status **=** PTR\_ERR**(**dev**);**

chip**->**exported **=** **(**status **==** 0**);**

mutex\_unlock**(&**sysfs\_lock**);**

**if** **(**status**)** **{**

unsigned long flags**;**

unsigned gpio**;**

spin\_lock\_irqsave**(&**gpio\_lock**,** flags**);**

gpio **=** chip**->**base**;**

**while** **(**gpio\_desc**[**gpio**].**chip **==** chip**)**

gpio\_desc**[**gpio**++].**chip **=** **NULL;**

spin\_unlock\_irqrestore**(&**gpio\_lock**,** flags**);**

pr\_debug**(**"%s: chip %s status %d\n"**,** \_\_func\_\_**,**

chip**->**label**,** status**);**

**}**

**return** status**;**

**}**

//访问base，显示寄存器地址

static ssize\_t chip\_base\_show**(**struct device **\***dev**,**

struct device\_attribute **\***attr**,** char **\***buf**)**

**{**

//drvdata在device\_create(&gpio\_class, chip->dev, MKDEV(0, 0), chip,"gpiochip%d", chip->base);

//已经指定为chip

const struct gpio\_chip **\***chip **=** dev\_get\_drvdata**(**dev**);**

**return** sprintf**(**buf**,** "%d\n"**,** chip**->**base**);**

**}**

static DEVICE\_ATTR**(**base**,** 0444**,** chip\_base\_show**,** **NULL);**

//显示GPIOA类似信息

static ssize\_t chip\_label\_show**(**struct device **\***dev**,**

struct device\_attribute **\***attr**,** char **\***buf**)**

**{**

const struct gpio\_chip **\***chip **=** dev\_get\_drvdata**(**dev**);**

**return** sprintf**(**buf**,** "%s\n"**,** chip**->**label **?** **:** ""**);**

**}**

static DEVICE\_ATTR**(**label**,** 0444**,** chip\_label\_show**,** **NULL);**

//显示gpio个数

static ssize\_t chip\_ngpio\_show**(**struct device **\***dev**,**

struct device\_attribute **\***attr**,** char **\***buf**)**

**{**

const struct gpio\_chip **\***chip **=** dev\_get\_drvdata**(**dev**);**

**return** sprintf**(**buf**,** "%u\n"**,** chip**->**ngpio**);**

**}**

static DEVICE\_ATTR**(**ngpio**,** 0444**,** chip\_ngpio\_show**,** **NULL);**

static const struct attribute **\***gpiochip\_attrs**[]** **=** **{**

**&**dev\_attr\_base**.**attr**,**

**&**dev\_attr\_label**.**attr**,**

**&**dev\_attr\_ngpio**.**attr**,**

**NULL,**

**};**

static const struct attribute\_group gpiochip\_attr\_group **=** **{**

**.**attrs **=** **(**struct attribute **\*\*)** gpiochip\_attrs**,**

**};**

### 1.3 gpiolib\_sysfs\_init流程

gpiolib\_sysfs\_init是整个gpio子系统的初始化函数。

static struct class\_attribute gpio\_class\_attrs**[]** **=** **{**

\_\_ATTR**(**export**,** 0200**,** **NULL,** export\_store**),**

\_\_ATTR**(**unexport**,** 0200**,** **NULL,** unexport\_store**),**

\_\_ATTR\_NULL**,**

**};**

//创建/sys/class/gpio export unexport用于导出gpio方便应用层操作

static struct class gpio\_class **=** **{**

**.**name **=** "gpio"**,**

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

**.**class\_attrs **=** gpio\_class\_attrs**,**

**};**

static int \_\_init gpiolib\_sysfs\_init**(**void**)**

**{**

int status**;**

unsigned long flags**;**

unsigned gpio**;**

//创建/sys/class/gpio目录

status **=** class\_register**(&**gpio\_class**);**

**if** **(**status **<** 0**)**

**return** status**;**

spin\_lock\_irqsave**(&**gpio\_lock**,** flags**);**

**for** **(**gpio **=** 0**;** gpio **<** ARCH\_NR\_GPIOS**;** gpio**++)** **{**

struct gpio\_chip **\***chip**;**

chip **=** gpio\_desc**[**gpio**].**chip**;**

**if** **(!**chip **||** chip**->**exported**)**

**continue;**

spin\_unlock\_irqrestore**(&**gpio\_lock**,** flags**);**

status **=** gpiochip\_export**(**chip**);**

spin\_lock\_irqsave**(&**gpio\_lock**,** flags**);**

**}**

spin\_unlock\_irqrestore**(&**gpio\_lock**,** flags**);**

**return** status**;**

**}**

postcore\_initcall**(**gpiolib\_sysfs\_init**);**

#### 1.3.1 GPIO export

export/unexport主要用于gpio的导出/导入，方便在应用层操作gpio。下面来分析下export/unexport。

导出在应用层的操作为：

**/**sys**/**class**/**gpio# echo 44 **>** export

export\_store函数代码如下，当导出gpio后便会在/sys/class/gpio下创建gpioN目录，方便应用层操作。

static ssize\_t export\_store**(**struct class **\***class**,**

struct class\_attribute **\***attr**,**

const char **\***buf**,** size\_t len**)**

**{**

long gpio**;**

int status**;**

//获取gpio的number

status **=** strict\_strtol**(**buf**,** 0**,** **&**gpio**);**

**if** **(**status **<** 0**)**

**goto** done**;**

//检查该gpio是否被导出，根据需求调用request函数

status **=** gpio\_request**(**gpio**,** "sysfs"**);**

**if** **(**status **<** 0**)**

**goto** done**;**

//创建/sys/class/gpio/gpioN下面对应的操作文件

status **=** gpio\_export**(**gpio**,** **true);**

**if** **(**status **<** 0**)**

gpio\_free**(**gpio**);**

**else**

set\_bit**(**FLAG\_SYSFS**,** **&**gpio\_desc**[**gpio**].**flags**);**

done**:**

**if** **(**status**)**

pr\_debug**(**"%s: status %d\n"**,** \_\_func\_\_**,** status**);**

**return** status **?** **:** len**;**

**}**

其中gpio\_request源码如下：

int gpio\_request**(**unsigned gpio**,** const char **\***label**)**

**{**

struct gpio\_desc **\***desc**;**

struct gpio\_chip **\***chip**;**

int status **=** **-**EINVAL**;**

unsigned long flags**;**

spin\_lock\_irqsave**(&**gpio\_lock**,** flags**);**

//gpio number是否超出范围

**if** **(!**gpio\_is\_valid**(**gpio**))**

**goto** done**;**

//获取对应gpio number对应的desc

desc **=** **&**gpio\_desc**[**gpio**];**

//获取对应的gpio\_chip

chip **=** desc**->**chip**;**

**if** **(**chip **==** **NULL)**

**goto** done**;**

**if** **(!**try\_module\_get**(**chip**->**owner**))**

**goto** done**;**

//检查该gpio是否被导出过

**if** **(**test\_and\_set\_bit**(**FLAG\_REQUESTED**,** **&**desc**->**flags**)** **==** 0**)** **{**

//设置gpio\_desc的label

desc\_set\_label**(**desc**,** label **?** **:** "?"**);**

status **=** 0**;**

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

status **=** **-**EBUSY**;**

module\_put**(**chip**->**owner**);**

**goto** done**;**

**}**

//是否有request函数,如果有就调用

**if** **(**chip**->**request**)** **{**

/\* chip->request may sleep \*/

spin\_unlock\_irqrestore**(&**gpio\_lock**,** flags**);**

status **=** chip**->**request**(**chip**,** gpio **-** chip**->**base**);**

spin\_lock\_irqsave**(&**gpio\_lock**,** flags**);**

**if** **(**status **<** 0**)** **{**

desc\_set\_label**(**desc**,** **NULL);**

module\_put**(**chip**->**owner**);**

clear\_bit**(**FLAG\_REQUESTED**,** **&**desc**->**flags**);**

**}**

**}**

done**:**

**if** **(**status**)**

pr\_debug**(**"gpio\_request: gpio-%d (%s) status %d\n"**,**

gpio**,** label **?** **:** "?"**,** status**);**

spin\_unlock\_irqrestore**(&**gpio\_lock**,** flags**);**

**return** status**;**

**}**

gpio\_export函数如下：

int gpio\_export**(**unsigned gpio**,** bool direction\_may\_change**)**

**{**

unsigned long flags**;**

struct gpio\_desc **\***desc**;**

int status **=** **-**EINVAL**;**

const char **\***ioname **=** **NULL;**

/\* can't export until sysfs is available ... \*/

**if** **(!**gpio\_class**.**p**)** **{**

pr\_debug**(**"%s: called too early!\n"**,** \_\_func\_\_**);**

**return** **-**ENOENT**;**

**}**

**if** **(!**gpio\_is\_valid**(**gpio**))**

**goto** done**;**

mutex\_lock**(&**sysfs\_lock**);**

spin\_lock\_irqsave**(&**gpio\_lock**,** flags**);**

//获取gpio对应的desc

desc **=** **&**gpio\_desc**[**gpio**];**

//如果该gpio已经被request了但是未被export

**if** **(**test\_bit**(**FLAG\_REQUESTED**,** **&**desc**->**flags**)**

**&&** **!**test\_bit**(**FLAG\_EXPORT**,** **&**desc**->**flags**))** **{**

status **=** 0**;**

**if** **(!**desc**->**chip**->**direction\_input

**||** **!**desc**->**chip**->**direction\_output**)**

direction\_may\_change **=** **false;**

**}**

spin\_unlock\_irqrestore**(&**gpio\_lock**,** flags**);**

//这里names为空

**if** **(**desc**->**chip**->**names **&&** desc**->**chip**->**names**[**gpio **-** desc**->**chip**->**base**])**

ioname **=** desc**->**chip**->**names**[**gpio **-** desc**->**chip**->**base**];**

**if** **(**status **==** 0**)** **{**

struct device **\***dev**;**

//创建/sys/class/gpio/gpioN

dev **=** device\_create**(&**gpio\_class**,** desc**->**chip**->**dev**,** MKDEV**(**0**,** 0**),**

desc**,** ioname **?** ioname **:** "gpio%u"**,** gpio**);**

**if** **(!**IS\_ERR**(**dev**))** **{**

//创建value active\_low操作接口

status **=** sysfs\_create\_group**(&**dev**->**kobj**,**

**&**gpio\_attr\_group**);**

//创建direction操作接口

**if** **(!**status **&&** direction\_may\_change**)**

status **=** device\_create\_file**(**dev**,**

**&**dev\_attr\_direction**);**

//如果支持中断并且可以作为输入，创建edge接口

**if** **(!**status **&&** gpio\_to\_irq**(**gpio**)** **>=** 0

**&&** **(**direction\_may\_change

**||** **!**test\_bit**(**FLAG\_IS\_OUT**,**

**&**desc**->**flags**)))**

status **=** device\_create\_file**(**dev**,**

**&**dev\_attr\_edge**);**

**if** **(**status **!=** 0**)**

device\_unregister**(**dev**);**

**}** **else**

status **=** PTR\_ERR**(**dev**);**

**if** **(**status **==** 0**)**

set\_bit**(**FLAG\_EXPORT**,** **&**desc**->**flags**);**

**}**

mutex\_unlock**(&**sysfs\_lock**);**

done**:**

**if** **(**status**)**

pr\_debug**(**"%s: gpio%d status %d\n"**,** \_\_func\_\_**,** gpio**,** status**);**

**return** status**;**

**}**

下面对gpioN的操作函数进行分析：

##### 1.3.1.1 GPIO value

value属性和active\_low属性共同决定高低电平的设置值。

active\_low：具有读写属性，值为“0”或“1”，用于决定value中的值是否进行翻转。当值为“0”时，value中的“0”表示低电平，“1”表示高电平；当值为“1”时，value中的“1”表示低电平，“0”表示高电平。

active\_low对应的代码如下：

static const DEVICE\_ATTR**(**active\_low**,** 0644**,**

gpio\_active\_low\_show**,** gpio\_active\_low\_store**);**

static int sysfs\_set\_active\_low**(**struct gpio\_desc **\***desc**,** struct device **\***dev**,**int value**)**

**{**

int status **=** 0**;**

**if** **(!!**test\_bit**(**FLAG\_ACTIVE\_LOW**,** **&**desc**->**flags**)** **==** **!!**value**)**

**return** 0**;**

**if** **(**value**)**

set\_bit**(**FLAG\_ACTIVE\_LOW**,** **&**desc**->**flags**);**

**else**

clear\_bit**(**FLAG\_ACTIVE\_LOW**,** **&**desc**->**flags**);**

//如果只设置了上升沿或下降沿 需要重新设置中断

**if** **(**dev **!=** **NULL** **&&** **(!!**test\_bit**(**FLAG\_TRIG\_RISE**,** **&**desc**->**flags**)** **^**

**!!**test\_bit**(**FLAG\_TRIG\_FALL**,** **&**desc**->**flags**)))** **{**

//判断是上升沿还是下降沿

//GPIO\_TRIGGER\_MASK屏蔽TRIGGER以外的其他bit位

unsigned long trigger\_flags **=** desc**->**flags **&** GPIO\_TRIGGER\_MASK**;**

//取消上次的中断设置

gpio\_setup\_irq**(**desc**,** dev**,** 0**);**

//重新设置中断

status **=** gpio\_setup\_irq**(**desc**,** dev**,** trigger\_flags**);**

**}**

**return** status**;**

**}**

static ssize\_t gpio\_active\_low\_show**(**struct device **\***dev**,**

struct device\_attribute **\***attr**,** char **\***buf**)**

**{**

const struct gpio\_desc **\***desc **=** dev\_get\_drvdata**(**dev**);**

ssize\_t status**;**

mutex\_lock**(&**sysfs\_lock**);**

**if** **(!**test\_bit**(**FLAG\_EXPORT**,** **&**desc**->**flags**))**

status **=** **-**EIO**;**

**else**

//读出active low对应的值

status **=** sprintf**(**buf**,** "%d\n"**,**

**!!**test\_bit**(**FLAG\_ACTIVE\_LOW**,** **&**desc**->**flags**));**

mutex\_unlock**(&**sysfs\_lock**);**

**return** status**;**

**}**

static ssize\_t gpio\_active\_low\_store**(**struct device **\***dev**,**

struct device\_attribute **\***attr**,** const char **\***buf**,** size\_t size**)**

**{**

struct gpio\_desc **\***desc **=** dev\_get\_drvdata**(**dev**);**

ssize\_t status**;**

mutex\_lock**(&**sysfs\_lock**);**

**if** **(!**test\_bit**(**FLAG\_EXPORT**,** **&**desc**->**flags**))** **{**

status **=** **-**EIO**;**

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

long value**;**

//设置active low对应的值

status **=** strict\_strtol**(**buf**,** 0**,** **&**value**);**

**if** **(**status **==** 0**)**

status **=** sysfs\_set\_active\_low**(**desc**,** dev**,** value **!=** 0**);**

**}**

mutex\_unlock**(&**sysfs\_lock**);**

**return** status **?** **:** size**;**

**}**

value设置对应的代码如下：

static const DEVICE\_ATTR**(**value**,** 0644**,**

gpio\_value\_show**,** gpio\_value\_store**);**

static ssize\_t gpio\_value\_show**(**struct device **\***dev**,**

struct device\_attribute **\***attr**,** char **\***buf**)**

**{**

const struct gpio\_desc **\***desc **=** dev\_get\_drvdata**(**dev**);**

unsigned gpio **=** desc **-** gpio\_desc**;**

ssize\_t status**;**

mutex\_lock**(&**sysfs\_lock**);**

//判断是否被导出

**if** **(!**test\_bit**(**FLAG\_EXPORT**,** **&**desc**->**flags**))** **{**

status **=** **-**EIO**;**

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

int value**;**

//读出gpio对应的值

//两个!是为了把非0值转换成1,而0值还是0。

value **=** **!!**gpio\_get\_value\_cansleep**(**gpio**);**

//根据active low读出value

**if** **(**test\_bit**(**FLAG\_ACTIVE\_LOW**,** **&**desc**->**flags**))**

value **=** **!**value**;**

status **=** sprintf**(**buf**,** "%d\n"**,** value**);**

**}**

mutex\_unlock**(&**sysfs\_lock**);**

**return** status**;**

**}**

static ssize\_t gpio\_value\_store**(**struct device **\***dev**,**

struct device\_attribute **\***attr**,** const char **\***buf**,** size\_t size**)**

**{**

const struct gpio\_desc **\***desc **=** dev\_get\_drvdata**(**dev**);**

unsigned gpio **=** desc **-** gpio\_desc**;**

ssize\_t status**;**

mutex\_lock**(&**sysfs\_lock**);**

//判断是否被导出

**if** **(!**test\_bit**(**FLAG\_EXPORT**,** **&**desc**->**flags**))**

status **=** **-**EIO**;**

**else** **if** **(!**test\_bit**(**FLAG\_IS\_OUT**,** **&**desc**->**flags**))**

status **=** **-**EPERM**;**

**else** **{**

long value**;**

status **=** strict\_strtol**(**buf**,** 0**,** **&**value**);**

**if** **(**status **==** 0**)** **{**

//根据active low设置value

**if** **(**test\_bit**(**FLAG\_ACTIVE\_LOW**,** **&**desc**->**flags**))**

value **=** **!**value**;**

//写对应的寄存器

gpio\_set\_value\_cansleep**(**gpio**,** value **!=** 0**);**

status **=** size**;**

**}**

**}**

mutex\_unlock**(&**sysfs\_lock**);**

**return** status**;**

**}**

void gpio\_set\_value\_cansleep**(**unsigned gpio**,** int value**)**

**{**

struct gpio\_chip **\***chip**;**

might\_sleep\_if**(**extra\_checks**);**

chip **=** gpio\_to\_chip**(**gpio**);**

trace\_gpio\_value**(**gpio**,** 0**,** value**);**

//当设置flag为GPIOF\_OPEN\_DRAIN时，它将假设管脚是开漏极方式。

//这类管脚在输出模式将不会被驱动为1。这样的管脚需要连接上拉。

//通过使能此flag，当它在输出模式被要求设置为1时，

//gpio lib将使得方向为输入以使得管脚变高。输出模式下，管脚输出值0以驱动电平为低。

**if** **(**test\_bit**(**FLAG\_OPEN\_DRAIN**,** **&**gpio\_desc**[**gpio**].**flags**))**

\_gpio\_set\_open\_drain\_value**(**gpio**,** chip**,** value**);**

//当设置flag为GPIOF\_OPEN\_SOURCE，它假设管脚时开源极类型。

//这种管脚在输出模式不能驱动为0。此种管脚需要下拉。

//通过使能这个flag，当管脚要求输出1时，

//gpio lib将使得方向变为输入以使得管脚变低。管脚在输出模式驱动1为高

**else** **if** **(**test\_bit**(**FLAG\_OPEN\_SOURCE**,** **&**gpio\_desc**[**gpio**].**flags**))**

\_gpio\_set\_open\_source\_value**(**gpio**,** chip**,** value**);**

**else**

//正常模式下设置GPIO

chip**->**set**(**chip**,** gpio **-** chip**->**base**,** value**);**

**}**

##### 1.3.1.2 GPIO direction

static /\* const \*/ DEVICE\_ATTR**(**direction**,** 0644**,**

gpio\_direction\_show**,** gpio\_direction\_store**);**

static ssize\_t gpio\_direction\_show**(**struct device **\***dev**,**

struct device\_attribute **\***attr**,** char **\***buf**)**

**{**

const struct gpio\_desc **\***desc **=** dev\_get\_drvdata**(**dev**);**

ssize\_t status**;**

mutex\_lock**(&**sysfs\_lock**);**

**if** **(!**test\_bit**(**FLAG\_EXPORT**,** **&**desc**->**flags**))**

status **=** **-**EIO**;**

**else**

//测试FLAG\_IS\_OUT判断是否为输出

status **=** sprintf**(**buf**,** "%s\n"**,**

test\_bit**(**FLAG\_IS\_OUT**,** **&**desc**->**flags**)**

**?** "out" **:** "in"**);**

mutex\_unlock**(&**sysfs\_lock**);**

**return** status**;**

**}**

static ssize\_t gpio\_direction\_store**(**struct device **\***dev**,**

struct device\_attribute **\***attr**,** const char **\***buf**,** size\_t size**)**

**{**

const struct gpio\_desc **\***desc **=** dev\_get\_drvdata**(**dev**);**

unsigned gpio **=** desc **-** gpio\_desc**;**

ssize\_t status**;**

mutex\_lock**(&**sysfs\_lock**);**

**if** **(!**test\_bit**(**FLAG\_EXPORT**,** **&**desc**->**flags**))**

status **=** **-**EIO**;**

//如果设置为输出，会调用gpio\_chip中的direction\_output函数，并设置FLAG\_IS\_OUT

**else** **if** **(**sysfs\_streq**(**buf**,** "high"**))**

status **=** gpio\_direction\_output**(**gpio**,** 1**);**

**else** **if** **(**sysfs\_streq**(**buf**,** "out"**)** **||** sysfs\_streq**(**buf**,** "low"**))**

status **=** gpio\_direction\_output**(**gpio**,** 0**);**

**else** **if** **(**sysfs\_streq**(**buf**,** "in"**))**

status **=** gpio\_direction\_input**(**gpio**);**

**else**

status **=** **-**EINVAL**;**

mutex\_unlock**(&**sysfs\_lock**);**

**return** status **?** **:** size**;**

**}**

##### 1.3.1.3 GPIO edge

gpio edge用于设置中断。

static DEVICE\_ATTR**(**edge**,** 0644**,** gpio\_edge\_show**,** gpio\_edge\_store**);**

static const struct **{**

const char **\***name**;**

unsigned long flags**;**

**}** trigger\_types**[]** **=** **{**

**{** "none"**,** 0 **},**

**{** "falling"**,** BIT**(**FLAG\_TRIG\_FALL**)** **},**

**{** "rising"**,** BIT**(**FLAG\_TRIG\_RISE**)** **},**

**{** "both"**,** BIT**(**FLAG\_TRIG\_FALL**)** **|** BIT**(**FLAG\_TRIG\_RISE**)** **},**

**};**

static ssize\_t gpio\_edge\_show**(**struct device **\***dev**,**

struct device\_attribute **\***attr**,** char **\***buf**)**

**{**

const struct gpio\_desc **\***desc **=** dev\_get\_drvdata**(**dev**);**

ssize\_t status**;**

mutex\_lock**(&**sysfs\_lock**);**

**if** **(!**test\_bit**(**FLAG\_EXPORT**,** **&**desc**->**flags**))**

status **=** **-**EIO**;**

**else** **{**

int i**;**

status **=** 0**;**

**for** **(**i **=** 0**;** i **<** ARRAY\_SIZE**(**trigger\_types**);** i**++)**

**if** **((**desc**->**flags **&** GPIO\_TRIGGER\_MASK**)**

**==** trigger\_types**[**i**].**flags**)** **{**

status **=** sprintf**(**buf**,** "%s\n"**,**

trigger\_types**[**i**].**name**);**

**break;**

**}**

**}**

mutex\_unlock**(&**sysfs\_lock**);**

**return** status**;**

**}**

static ssize\_t gpio\_edge\_store**(**struct device **\***dev**,**

struct device\_attribute **\***attr**,** const char **\***buf**,** size\_t size**)**

**{**

struct gpio\_desc **\***desc **=** dev\_get\_drvdata**(**dev**);**

ssize\_t status**;**

int i**;**

**for** **(**i **=** 0**;** i **<** ARRAY\_SIZE**(**trigger\_types**);** i**++)**

**if** **(**sysfs\_streq**(**trigger\_types**[**i**].**name**,** buf**))**

**goto** found**;**

**return** **-**EINVAL**;**

found**:**

mutex\_lock**(&**sysfs\_lock**);**

**if** **(!**test\_bit**(**FLAG\_EXPORT**,** **&**desc**->**flags**))**

status **=** **-**EIO**;**

**else** **{**

//设置gpio中断

status **=** gpio\_setup\_irq**(**desc**,** dev**,** trigger\_types**[**i**].**flags**);**

**if** **(!**status**)**

status **=** size**;**

**}**

mutex\_unlock**(&**sysfs\_lock**);**

**return** status**;**

**}**

核心函数为gpio\_setup\_irq，在分析该函数之前，需要解释一下idr。

idr机制适用在那些需要把某个整数和特定指针关联在一起的地方。举个例子，在I2C总线中，每个设备都有自己的地址，要想在总线上找到特定的设备，就必须要先发送该设备的地址。如果我们的PC是一个I2C总线上的主节点，那么要访问总线上的其他设备，首先要知道他们的ID号，同时要在pc的驱动程序中建立一个用于描述该设备的结构体。

此时，问题来了，我们怎么才能将这个设备的ID号和他的设备结构体联系起来呢？最简单的方法当然是通过数组进行索引，但如果ID号的范围很大(比如32位的ID号)，则用数组索引显然不可能；第二种方法是用链表，但如果网络中实际存在的设备较多，则链表的查询效率会很低。遇到这种清况，我们就可以采用idr机制，该机制内部采用radix树实现，可以很方便地将整数和指针关联起来，并且具有很高的搜索效率。

gpio\_setup\_irq函数代码如下：

static int gpio\_setup\_irq**(**struct gpio\_desc **\***desc**,** struct device **\***dev**,**

unsigned long gpio\_flags**)**

**{**

struct sysfs\_dirent **\***value\_sd**;**

unsigned long irq\_flags**;**

int ret**,** irq**,** id**;**

//判断是否重复设置了中断标志，如果这次设置和已经设置的一致，直接返回

**if** **((**desc**->**flags **&** GPIO\_TRIGGER\_MASK**)** **==** gpio\_flags**)**

**return** 0**;**

//获取中断号

irq **=** gpio\_to\_irq**(**desc **-** gpio\_desc**);**

**if** **(**irq **<** 0**)**

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

//desc的高16bit后面是ID位

id **=** desc**->**flags **>>** ID\_SHIFT**;**

//返回值是和给定id相关联的指针，如果没有，则返回NULL

value\_sd **=** idr\_find**(&**dirent\_idr**,** id**);**

//如果需要重新申请中断，则需先删除原来的中断,因为所有的gpio共用中断函数

**if** **(**value\_sd**)**

free\_irq**(**irq**,** value\_sd**);**

//清空中断相关的标志位

desc**->**flags **&=** **~**GPIO\_TRIGGER\_MASK**;**

//如果gpio\_flags为０，则直接退出

**if** **(!**gpio\_flags**)** **{**

ret **=** 0**;**

**goto** free\_id**;**

**}**

irq\_flags **=** IRQF\_SHARED**;**

**if** **(**test\_bit**(**FLAG\_TRIG\_FALL**,** **&**gpio\_flags**))**

irq\_flags **|=** test\_bit**(**FLAG\_ACTIVE\_LOW**,** **&**desc**->**flags**)** **?**

IRQF\_TRIGGER\_RISING **:** IRQF\_TRIGGER\_FALLING**;**

**if** **(**test\_bit**(**FLAG\_TRIG\_RISE**,** **&**gpio\_flags**))**

irq\_flags **|=** test\_bit**(**FLAG\_ACTIVE\_LOW**,** **&**desc**->**flags**)** **?**

IRQF\_TRIGGER\_FALLING **:** IRQF\_TRIGGER\_RISING**;**

//如果value\_sd不为空，需要重新申请

**if** **(!**value\_sd**)** **{**

//获取value对应的dirent结构体，这里用于标识中断，并唤醒对应的POLL

value\_sd **=** sysfs\_get\_dirent**(**dev**->**kobj**.**sd**,** **NULL,** "value"**);**

**if** **(!**value\_sd**)** **{**

ret **=** **-**ENODEV**;**

**goto** err\_out**;**

**}**

**do** **{**

ret **=** **-**ENOMEM**;**

**if** **(**idr\_pre\_get**(&**dirent\_idr**,** GFP\_KERNEL**))**

//从1开始找到id，用于关联value\_sd

ret **=** idr\_get\_new\_above**(&**dirent\_idr**,** value\_sd**,**

1**,** **&**id**);**

**}** **while** **(**ret **==** **-**EAGAIN**);**

**if** **(**ret**)**

**goto** free\_sd**;**

//清空中断标志位

desc**->**flags **&=** GPIO\_FLAGS\_MASK**;**

//左移16位保存ID

desc**->**flags **|=** **(**unsigned long**)**id **<<** ID\_SHIFT**;**

**if** **(**desc**->**flags **>>** ID\_SHIFT **!=** id**)** **{**

ret **=** **-**ERANGE**;**

**goto** free\_id**;**

**}**

**}**

//申请中断，中断处理函数为gpio\_sysfs\_irq

ret **=** request\_any\_context\_irq**(**irq**,** gpio\_sysfs\_irq**,** irq\_flags**,**

"gpiolib"**,** value\_sd**);**

**if** **(**ret **<** 0**)**

**goto** free\_id**;**

///重新设置中断标志位

desc**->**flags **|=** gpio\_flags**;**

**return** 0**;**

free\_id**:**

//移除id对应的idr

idr\_remove**(&**dirent\_idr**,** id**);**

desc**->**flags **&=** GPIO\_FLAGS\_MASK**;**

free\_sd**:**

**if** **(**value\_sd**)**

sysfs\_put**(**value\_sd**);**

err\_out**:**

**return** ret**;**

**}**

gpio\_sysfs\_irq中断处理函数代码如下：

static irqreturn\_t gpio\_sysfs\_irq**(**int irq**,** void **\***priv**)**

**{**

struct sysfs\_dirent **\***value\_sd **=** priv**;**

//sysfs\_notify\_dirent()用来唤醒在属性文件上调用select()或poll()而阻塞的用户进程

sysfs\_notify\_dirent**(**value\_sd**);**

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

**}**

当open /sys/class/gpio/gpioN，并在该文件上使用select或poll，这个时候会阻塞，当发生中断时，会自动唤醒阻塞的进程。

#### 1.3.2 GPIO unexport

gpio unexport用于注册对应的设备并删除中断。

void gpio\_unexport**(**unsigned gpio**)**

**{**

struct gpio\_desc **\***desc**;**

int status **=** 0**;**

struct device **\***dev **=** **NULL;**

**if** **(!**gpio\_is\_valid**(**gpio**))** **{**

status **=** **-**EINVAL**;**

**goto** done**;**

**}**

mutex\_lock**(&**sysfs\_lock**);**

desc **=** **&**gpio\_desc**[**gpio**];**

**if** **(**test\_bit**(**FLAG\_EXPORT**,** **&**desc**->**flags**))** **{**

//通过match\_export比较drvdata，找到对应的设备

dev **=** class\_find\_device**(&**gpio\_class**,** **NULL,** desc**,** match\_export**);**

**if** **(**dev**)** **{**

//删除中断

gpio\_setup\_irq**(**desc**,** dev**,** 0**);**

//清空EXPORT标志

clear\_bit**(**FLAG\_EXPORT**,** **&**desc**->**flags**);**

**}** **else**

status **=** **-**ENODEV**;**

**}**

mutex\_unlock**(&**sysfs\_lock**);**

**if** **(**dev**)** **{**

//注销设备

device\_unregister**(**dev**);**

put\_device**(**dev**);**

**}**

done**:**

**if** **(**status**)**

pr\_debug**(**"%s: gpio%d status %d\n"**,** \_\_func\_\_**,** gpio**,** status**);**

**}**

## 2.内核中GPIO的使用

（1）测试gpio端口是否合法。

int gpio\_is\_valid**(**int number**);**

（2）申请某个gpio端口当然在申请之前需要显示的配置该gpio端口的pinmux

int gpio\_request**(**unsigned gpio**,** const char **\***label**)**

（3）标记gpio的使用方向包括输入还是输出

int gpio\_direction\_input**(**unsigned gpio**);**

int gpio\_direction\_output**(**unsigned gpio**,** int value**);**

（4）获得gpio引脚的值和设置gpio引脚的值(对于输出)

int gpio\_get\_value**(**unsigned gpio**);**

void gpio\_set\_value**(**unsigned gpio**,** int value**);**

（5）gpio当作中断口使用

int gpio\_to\_irq**(**unsigned gpio**);**

//返回的值即中断编号可以传给request\_irq()和free\_irq()

//内核通过调用该函数将gpio端口转换为中断，在用户空间也有类似方法

（6）导出gpio端口到用户空间

int gpio\_export**(**unsigned gpio**,** bool direction\_may\_change**);**

//内核可以对已经被gpio\_request()申请的gpio端口的导出进行明确的管理，

//参数direction\_may\_change表示用户程序是否允许修改gpio的方向，假如可以

//则参数direction\_may\_change为真

        撤销GPIO的导出

void gpio\_unexport**();**

## 3.应用程序中GPIO的使用

用户空间访问gpio，即通过sysfs接口访问gpio。

首先，看看系统中有没有“/sys/class/gpio”这个文件夹。如果没有请在编译内核的时候加入   Device Drivers-> GPIO Support ->/sys/class/gpio/… (sysfs interface)。

下面是/sys/class/gpio目录下的三种文件：

**--**export**/**unexport文件

**--**gpioN指代具体的gpio引脚

**--**gpio\_chipN指代gpio控制器

### 3.1 export/unexport文件接口

/sys/class/gpio/export，该接口只能写不能读，用户程序通过写入gpio的编号来向内核申请将某个gpio的控制权导出到用户空间当然前提是没有内核代码申请这个gpio端口。

echo 19 **>** export

上述操作会为19号gpio创建一个节点gpio19，此时/sys/class/gpio目录下边生成一个gpio19的目录。

/sys/class/gpio/unexport和导出的效果相反。

echo 19 **>** unexport

上述操作将会移除gpio19这个节点。

### 3.2 /sys/class/gpio/gpioN

指代某个具体的gpio端口,里边有如下属性文件

direction 表示gpio端口的方向，读取结果是in或out。该文件也可以写，写入out 时该gpio设为输出同时电平默认为低。写入low或high则不仅可以设置为输出还可以设置输出的电平。 当然如果内核不支持或者内核代码不愿意，将不会存在这个属性,比如内核调用了gpio\_export(N,0)就表示内核不愿意修改gpio端口方向属性。

 value 表示gpio引脚的电平,0(低电平)1（高电平）,如果gpio被配置为输出，这个值是可写的，记住任何非零的值都将输出高电平, 如果某个引脚能并且已经被配置为中断，则可以调用poll(2)函数监听该中断，中断触发后poll(2)函数就会返回。

edge表示中断的触发方式，edge文件有如下四个值：none,rising,falling,both。

none表示引脚为输入，不是中断引脚

rising表示引脚为中断输入，上升沿触发

falling表示引脚为中断输入，下降沿触发

both表示引脚为中断输入，边沿触发

这个文件节点只有在引脚被配置为输入引脚的时候才存在。 当值是none时可以通过如下方法将变为中断引脚  
echo "both" > edge;对于是both,falling还是rising依赖具体硬件的中断的触发方式。此方法即用户态gpio转换为中断引脚的方式。

### 3.3 /sys/class/gpio/gpiochipN

gpiochipN表示的就是一个gpio\_chip,用来管理和控制一组gpio端口的控制器，该目录下存在一下属性文件：

base   和N相同，表示控制器管理的最小的端口编号。

lable   诊断使用的标志（并不总是唯一的）

ngpio  表示控制器管理的gpio端口数量（端口范围是：N **~** N**+**ngpio**-**1）

### 3.4 用户态使用gpio子系统监听中断

首先需要将该gpio配置为中断。

echo  "rising" **>** **/**sys**/**class**/**gpio**/**gpio12**/**edge

以下是伪代码  
poll示例：

int gpio\_id**;**

struct pollfd fds**[**1**];**

gpio\_fd **=** open**(**"/sys/class/gpio/gpio12/value"**,**O\_RDONLY**);**

**if(** gpio\_fd **==** **-**1 **)**

   err\_print**(**"gpio open"**);**

fds**[**0**].**fd **=** gpio\_fd**;**

fds**[**0**].**events  **=** POLLPRI**;**

ret **=** read**(**gpio\_fd**,**buff**,**10**);**

**if(** ret **==** **-**1 **)**

    err\_print**(**"read"**);**

**while(**1**){**

     ret **=** poll**(**fds**,**1**,-**1**);**

     if**(** ret **==** **-**1 **)**

         err\_print**(**"poll"**);**

       if**(** fds**[**0**].**revents **&** POLLPRI**){**

           ret **=** lseek**(**gpio\_fd**,**0**,**SEEK\_SET**);**

           if**(** ret **==** **-**1 **)**

               err\_print**(**"lseek"**);**

           ret **=** read**(**gpio\_fd**,**buff**,**10**);**

           if**(** ret **==** **-**1 **)**

               err\_print**(**"read"**);**

            /\*此时表示已经监听到中断触发了，该干事了\*/

**...............**

**}**

**}**

记住使用poll()函数，设置事件监听类型为POLLPRI和POLLERR在poll()返回后，使用lseek()移动到文件开头读取新的值或者关闭它再重新打开读取新值。必须这样做否则poll函数会总是返回。

select示例：

FD\_ZERO**(&**exceptfds**);**

FD\_SET**(**fd**,** **&**exceptfds**);**

ret **=** select**(**fd**+**1**,NULL,NULL,&**exceptfds**,NULL);**

**if(**ret **<** 0**)**

ERREXIT**(**"select value"**);**

//else if(ret > 0)

**if(**ret **>** 0**)**

**{**

/\* get value \*/

ret **=** lseek**(**fd**,** 0**,** SEEK\_SET**);**

**if(**ret **<** 0**)**

ERREXIT**(**"lseek value"**);**

ret **=** read**(**fd**,** buf**,** 2**);**

buf**[**1**]** **=** '\0'**;**

printf**(**"read ret = %d, value = %x\n"**,** ret**,** buf**[**0**]);**

**if(**ret **!=** 2**)**

ERREXIT**(**"read value"**);**

**}**