# Linux驱动\_pinctrl子系统

## 1.pinctrl子系统初始化

### 1.1 dts树中关于pinctrl的定义

pinctrl@56000000 {

compatible = "samsung,s3c2440-pinctrl";

};

pinctrl\_0: pinctrl@56000000 {

reg = <0x56000000 0x1000>;

wakeup-interrupt-controller {

compatible = "samsung,s3c2410-wakeup-eint";

interrupts = <0 0 0 3>,

<0 0 1 3>,

<0 0 2 3>,

<0 0 3 3>,

<0 0 4 4>,

<0 0 5 4>;

};

};

&pinctrl\_0 {

/\*

\* Pin banks

\*/

gpa: gpa {

gpio-controller;

#gpio-cells = <2>;

};

gpb: gpb {

gpio-controller;

#gpio-cells = <2>;

};

gpc: gpc {

gpio-controller;

#gpio-cells = <2>;

};

gpd: gpd {

gpio-controller;

#gpio-cells = <2>;

};

gpe: gpe {

gpio-controller;

#gpio-cells = <2>;

};

gpf: gpf {

gpio-controller;

#gpio-cells = <2>;

interrupt-controller;

#interrupt-cells = <2>;

};

gpg: gpg {

gpio-controller;

#gpio-cells = <2>;

interrupt-controller;

#interrupt-cells = <2>;

};

gph: gph {

gpio-controller;

#gpio-cells = <2>;

};

gpj: gpj {

gpio-controller;

#gpio-cells = <2>;

};

/\*

\* Pin groups

\*/

uart0\_data: uart0-data {

samsung,pins = "gph-0", "gph-1";

samsung,pin-function = <2>;

};

i2c0\_bus: i2c0-bus {

samsung,pins = "gpe-14", "gpe-15";

samsung,pin-function = <EXYNOS\_PIN\_FUNC\_2>;

};

nand\_pinctrl: nand\_pinctrl {

samsung,pins = "gpa-17", "gpa-18", "gpa-19",

"gpa-20", "gpa-22";

samsung,pin-function = <1>;

};

};

### 1.2 2440 pinctrl子系统初始化

samsung,s3c2440-pinctrl的注册代码如下：

static const struct of\_device\_id samsung\_pinctrl\_dt\_match**[]** **=** **{**

**{** **.**compatible **=** "samsung,s3c2440-pinctrl"**,**

**.**data **=** s3c2440\_pin\_ctrl **},**

**};**

static struct platform\_driver samsung\_pinctrl\_driver **=** **{**

**.**probe **=** samsung\_pinctrl\_probe**,**

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

**.**name **=** "samsung-pinctrl"**,**

**.**of\_match\_table **=** samsung\_pinctrl\_dt\_match**,**

**.**suppress\_bind\_attrs **=** **true,**

**},**

**};**

static int \_\_init samsung\_pinctrl\_drv\_register**(**void**)**

**{**

register\_syscore\_ops**(&**samsung\_pinctrl\_syscore\_ops**);**

**return** platform\_driver\_register**(&**samsung\_pinctrl\_driver**);**

**}**

匹配后samsung\_pinctrl\_probe函数调用。

static int samsung\_pinctrl\_probe**(**struct platform\_device **\***pdev**)**

**{**

struct samsung\_pinctrl\_drv\_data **\***drvdata**;**

const struct samsung\_pin\_ctrl **\***ctrl**;**

struct device **\***dev **=** **&**pdev**->**dev**;**

struct resource **\***res**;**

int ret**;**

//分配samsung\_pinctrl\_drv\_data结构体空间

drvdata **=** devm\_kzalloc**(**dev**,** **sizeof(\***drvdata**),** GFP\_KERNEL**);**

//初始化samsung\_pinctrl\_drv\_data

ctrl **=** samsung\_pinctrl\_get\_soc\_data**(**drvdata**,** pdev**);**

//drvdata关联dev

drvdata**->**dev **=** dev**;**

//pinctrl操作对应的虚拟地址

res **=** platform\_get\_resource**(**pdev**,** IORESOURCE\_MEM**,** 0**);**samsung\_pinctrl\_parse\_dt

drvdata**->**virt\_base **=** devm\_ioremap\_resource**(&**pdev**->**dev**,** res**);**

//获取pinctrl对应的irq号，这里没有

res **=** platform\_get\_resource**(**pdev**,** IORESOURCE\_IRQ**,** 0**);**

**if** **(**res**)**

drvdata**->**irq **=** res**->**start**;**

//注册到gpio子系统中

ret **=** samsung\_gpiolib\_register**(**pdev**,** drvdata**);**

//注册到pinctrl子系统中

ret **=** samsung\_pinctrl\_register**(**pdev**,** drvdata**);**

//gpio相关中断注册到中断子系统中

**if** **(**ctrl**->**eint\_gpio\_init**)**

ctrl**->**eint\_gpio\_init**(**drvdata**);**

**if** **(**ctrl**->**eint\_wkup\_init**)**

ctrl**->**eint\_wkup\_init**(**drvdata**);**

//设置device的私有数据

platform\_set\_drvdata**(**pdev**,** drvdata**);**

/\* Add to the global list \*/

list\_add\_tail**(&**drvdata**->**node**,** **&**drvdata\_list**);**

**return** 0**;**

**}**

probe函数对pinctrl、中断、gpio子系统都进行了操作。struct pinctrl\_desc和struct pinctrl\_dev 都是pin control subsystem中core driver的概念。各个具体硬件的pin controller可能会各不相同，但是可以抽取其共同的部分来形成一个HW independent的数据结构，这个数据就是pin controller描述符。整个初始化过程的核心思想就是low level的driver定义一个pinctrl\_desc，设定pin相关的定义和callback函数，注册到pin control subsystem中。我们用引脚描述符（pin descriptor）来描述一个pin。在pin control subsystem中，struct pinctrl\_pin\_desc用来描述一个可以控制的引脚，我们称之引脚描述符。

#### 1.2.1 samsung\_pinctrl\_drv\_data

samsung\_pinctrl\_drv\_data结构体定义如下：

struct samsung\_pinctrl\_drv\_data **{**

//多个pin controller的描述符可以形成链表

struct list\_head node**;**

//访问硬件寄存器的基地址

void \_\_iomem **\***virt\_base**;**

//和platform device建立联系

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

//irq number,2440不需要irq资源

int irq**;**

//指向pin control subsystem中core driver中抽象的pin controller描述符。

struct pinctrl\_desc pctl**;**

//指向core driver的pin controller class device

struct pinctrl\_dev **\***pctl\_dev**;**

//描述samsung pin controller中pin groups的信息

const struct samsung\_pin\_group **\***pin\_groups**;**

//描述samsung pin controller中pin groups的数目

unsigned int nr\_groups**;**

//描述samsung pin controller中function信息

const struct samsung\_pmx\_func **\***pmx\_functions**;**

//描述samsung pin controller中function的数目

unsigned int nr\_functions**;**

//每个bank的描述

struct samsung\_pin\_bank **\***pin\_banks**;**

//一共有多少个bank

u32 nr\_banks**;**

//pin脚初始值

unsigned int pin\_base**;**

//一共有多少个pin

unsigned int nr\_pins**;**

void **(\***suspend**)(**struct samsung\_pinctrl\_drv\_data **\*);**

void **(\***resume**)(**struct samsung\_pinctrl\_drv\_data **\*);**

**};**

devm\_kzalloc函数是为struct samsung\_pinctrl\_drv\_data数据结构分配内存。每当driver probe一个具体的device实例的时候，都需要建立一些私有的数据结构来保存该device的一些具体的硬件信息（本场景中，这个数据结构就是struct samsung\_pinctrl\_drv\_data）。在过去，驱动工程师多半使用kmalloc或者kzalloc来分配内存，但这会带来一些潜在的问题。例如：在初始化过程中，有各种各样可能的失败情况，这时候就依靠driver工程师小心的撰写代码，释放之前分配的内存。当然，初始化过程中，除了memory，driver会为probe的device分配各种资源，例如IRQ号，io memory map、DMA等等。当初始化需要管理这么多的资源分配和释放的时候，很多驱动程序都出现了资源管理的issue。而且，由于这些issue是异常路径上的issue，不是那么容易测试出来，更加重了解决这个issue的必要性。内核解决这个问题的模式（所谓解决一类问题的设计方法就叫做设计模式）是Devres，即device resource management软件模块。更细节的内容就不介绍了，其核心思想就是资源是设备的资源，那么资源的管理归于device，也就是说不需要driver过多的参与。当device和driver detach的时候，device会自动的释放其所有的资源。

#### 1.2.2 samsung\_pinctrl\_get\_soc\_data

分配了struct samsung\_pinctrl\_drv\_data数据结构的内存，当然下一步就是初始化这个数据结构了。

static const struct samsung\_pin\_ctrl **\*** samsung\_pinctrl\_get\_soc\_data**(**struct samsung\_pinctrl\_drv\_data **\***d**,**

struct platform\_device **\***pdev**)**

**{**

int id**;**

const struct of\_device\_id **\***match**;**

struct device\_node **\***node **=** pdev**->**dev**.**of\_node**;**

struct device\_node **\***np**;**

const struct samsung\_pin\_bank\_data **\***bdata**;**

const struct samsung\_pin\_ctrl **\***ctrl**;**

struct samsung\_pin\_bank **\***bank**;**

int i**;**

//获取device node对应的pinctrl alias id号，这里为0

id **=** of\_alias\_get\_id**(**node**,** "pinctrl"**);**

//match为如下数据

//{ .compatible = "samsung,s3c2440-pinctrl",

// .data = s3c2440\_pin\_ctrl }

match **=** of\_match\_node**(**samsung\_pinctrl\_dt\_match**,** node**);**

//获取对应的samsung\_pin\_ctrl s3c2440\_pin\_ctrl数组中的第一组数据

//struct samsung\_pin\_ctrl描述的具体samsung pin controller硬件相关的信息，

//比如说：pin bank的信息，不是所有的pin controller都是分bank的，

//因此pin bank的信息只能封装在low level的samsung pin controller driver中。

ctrl **=** **(**struct samsung\_pin\_ctrl **\*)**match**->**data **+** id**;**

//samsung\_pinctrl\_drv\_data设置对应的变量和函数

d**->**suspend **=** ctrl**->**suspend**;**

d**->**resume **=** ctrl**->**resume**;**

d**->**nr\_banks **=** ctrl**->**nr\_banks**;**

d**->**pin\_banks **=** devm\_kcalloc**(&**pdev**->**dev**,** d**->**nr\_banks**,sizeof(\***d**->**pin\_banks**),** GFP\_KERNEL**);**

//设置pin\_banks信息

bank **=** d**->**pin\_banks**;**

bdata **=** ctrl**->**pin\_banks**;**

//为每个bank设置基础数据

**for** **(**i **=** 0**;** i **<** ctrl**->**nr\_banks**;** **++**i**,** **++**bdata**,** **++**bank**)** **{**

//type按寄存器操作的bit数分类，

//例如BANK A只需一个bit完成pin功能的设置,其他的需要两个bit

bank**->**type **=** bdata**->**type**;**

//寄存器偏移

bank**->**pctl\_offset **=** bdata**->**pctl\_offset**;**

//每个BANK有多少个pin脚

bank**->**nr\_pins **=** bdata**->**nr\_pins**;**

//gpf gpg具备外部中断能力

bank**->**eint\_func **=** bdata**->**eint\_func**;**

//中断类型，这里为EINT\_TYPE\_WKUP

bank**->**eint\_type **=** bdata**->**eint\_type**;**

//那些gpio具备中断能力

bank**->**eint\_mask **=** bdata**->**eint\_mask**;**

//从第几个gpio具有中断能力

bank**->**eint\_offset **=** bdata**->**eint\_offset**;**

//gpio bank的名字,例如GPA

bank**->**name **=** bdata**->**name**;**

spin\_lock\_init**(&**bank**->**slock**);**

//根据bank能找到对应的samsung\_pinctrl\_drv\_data

bank**->**drvdata **=** d**;**

//设置bank的gpio数量偏移

bank**->**pin\_base **=** d**->**nr\_pins**;**

//更新整个2440具有的gpio数量

d**->**nr\_pins **+=** bank**->**nr\_pins**;**

**}**

//遍历device node下的gpio-controller子节点

for\_each\_child\_of\_node**(**node**,** np**)** **{**

//判断子节点是否是gpio-controller

**if** **(!**of\_find\_property**(**np**,** "gpio-controller"**,** **NULL))**

**continue;**

//轮询所有的samsung\_pin\_bank,名字相同的bank和gpio-controller做一个关联

bank **=** d**->**pin\_banks**;**

**for** **(**i **=** 0**;** i **<** d**->**nr\_banks**;** **++**i**,** **++**bank**)** **{**

**if** **(!**strcmp**(**bank**->**name**,** np**->**name**))** **{**

bank**->**of\_node **=** np**;**

**break;**

**}**

**}**

**}**

//更新全局变量pin\_base

d**->**pin\_base **=** pin\_base**;**

pin\_base **+=** d**->**nr\_pins**;**

//返回s3c2440\_pin\_ctrl

**return** ctrl**;**

**}**

struct samsung\_pinctrl\_drv\_data和硬件相关数据来自s3c2440\_pin\_ctrl：

struct samsung\_pin\_ctrl **{**

//定义具体的pin bank信息,例如操作寄存器/bank名称/有多少pin等

const struct samsung\_pin\_bank\_data **\***pin\_banks**;**

//number of pin bank

u32 nr\_banks**;**

//和中断相关的代码

int **(\***eint\_gpio\_init**)(**struct samsung\_pinctrl\_drv\_data **\*);**

int **(\***eint\_wkup\_init**)(**struct samsung\_pinctrl\_drv\_data **\*);**

void **(\***suspend**)(**struct samsung\_pinctrl\_drv\_data **\*);**

void **(\***resume**)(**struct samsung\_pinctrl\_drv\_data **\*);**

**};**

static const struct samsung\_pin\_bank\_data s3c2440\_pin\_banks**[]** \_\_initconst **=** **{**

PIN\_BANK\_A**(**25**,** 0x000**,** "gpa"**),**

PIN\_BANK\_2BIT**(**11**,** 0x010**,** "gpb"**),**

PIN\_BANK\_2BIT**(**16**,** 0x020**,** "gpc"**),**

PIN\_BANK\_2BIT**(**16**,** 0x030**,** "gpd"**),**

PIN\_BANK\_2BIT**(**16**,** 0x040**,** "gpe"**),**

PIN\_BANK\_2BIT\_EINTW**(**8**,** 0x050**,** "gpf"**,** 0**,** 0xff**),**

PIN\_BANK\_2BIT\_EINTW**(**16**,** 0x060**,** "gpg"**,** 8**,** 0xffff00**),**

PIN\_BANK\_2BIT**(**11**,** 0x070**,** "gph"**),**

PIN\_BANK\_2BIT**(**13**,** 0x0d0**,** "gpj"**),**

**};**

const struct samsung\_pin\_ctrl s3c2440\_pin\_ctrl**[]** \_\_initconst **=** **{**

**{**

**.**pin\_banks **=** s3c2440\_pin\_banks**,**

**.**nr\_banks **=** ARRAY\_SIZE**(**s3c2440\_pin\_banks**),**

**.**eint\_wkup\_init **=** s3c24xx\_eint\_init**,**

**},**

**};**

#### 1.2.3 samsung\_gpiolib\_register

samsung\_gpiolib\_register将gpio信息注册到了gpio子系统中。

static const struct gpio\_chip samsung\_gpiolib\_chip **=** **{**

**.**request **=** gpiochip\_generic\_request**,**

**.**free **=** gpiochip\_generic\_free**,**

**.**set **=** samsung\_gpio\_set**,**

**.**get **=** samsung\_gpio\_get**,**

**.**direction\_input **=** samsung\_gpio\_direction\_input**,**

**.**direction\_output **=** samsung\_gpio\_direction\_output**,**

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

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

**};**

static int samsung\_gpiolib\_register**(**struct platform\_device **\***pdev**,** struct samsung\_pinctrl\_drv\_data **\***drvdata**)**

**{**

//得到pin\_bank描述结构体

struct samsung\_pin\_bank **\***bank **=** drvdata**->**pin\_banks**;**

//一个gpio\_chip代表一个bank

struct gpio\_chip **\***gc**;**

int ret**;**

int i**;**

**for** **(**i **=** 0**;** i **<** drvdata**->**nr\_banks**;** **++**i**,** **++**bank**)** **{**

//将三星通用的gpio chip先赋值给gpio\_chip

bank**->**gpio\_chip **=** samsung\_gpiolib\_chip**;**

//修改每个bank不同的属性

gc **=** **&**bank**->**gpio\_chip**;**

//每个bank对应的gpio起始值

gc**->**base **=** drvdata**->**pin\_base **+** bank**->**pin\_base**;**

//每个bank有多少个gpio

gc**->**ngpio **=** bank**->**nr\_pins**;**

//该bank的parent device

gc**->**parent **=** **&**pdev**->**dev**;**

//该bank对应的device node

gc**->**of\_node **=** bank**->**of\_node**;**

//该bank的名字

gc**->**label **=** bank**->**name**;**

//将该bank添加到gpio子系统中

ret **=** gpiochip\_add\_data**(**gc**,** bank**);**

**}**

**return** 0**;**

**}**

因为4.x的内核采用了设备树，所以gpio的申请采用设备树方式，下面将详细分析基于设备树的gpio子系统的使用方法。

##### 1.2.3.1 gpiochip\_add\_data

将gpio\_chip添加到gpio子系统的核心函数为：

int gpiochip\_add\_data**(**struct gpio\_chip **\***chip**,** void **\***data**)**

**{**

unsigned long flags**;**

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

unsigned i**;**

//获取该bank的gpio起始序号

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

//分配一个gpio\_device结构体

//gpio\_device是gpio子系统一个bank的标准定义

struct gpio\_device **\***gdev**;**

gdev **=** kzalloc**(sizeof(\***gdev**),** GFP\_KERNEL**);**

//gpio\_device对应gpio bus

gdev**->**dev**.**bus **=** **&**gpio\_bus\_type**;**

//设置gpio\_device对应的gpio\_chip结构体

gdev**->**chip **=** chip**;**

//设置gpio\_chip对应的gpio\_device结构体

chip**->**gpiodev **=** gdev**;**

//gpio\_chip存在parent device

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

//设置gpio\_device的parent为gpio\_chip的parent

gdev**->**dev**.**parent **=** chip**->**parent**;**

//设置gpio\_device的device node为gpio\_chip parent decvice的device node

//这里为dts samsung,s3c2440-pinctrl节点

//后面还会修改该节点

gdev**->**dev**.**of\_node **=** chip**->**parent**->**of\_node**;**

**}**

//如果支持dts树

#ifdef CONFIG\_OF\_GPIO

//如果gpio\_chip拥有device node

**if** **(**chip**->**of\_node**)**

//设置gpio\_device的device node为dts树中的gpx

gdev**->**dev**.**of\_node **=** chip**->**of\_node**;**

#endif

//得到一个id值

gdev**->**id **=** ida\_simple\_get**(&**gpio\_ida**,** 0**,** 0**,** GFP\_KERNEL**);**

//设置device的名字，创建字符设备的时候需要使用，会生成/dev/gpiochipx

dev\_set\_name**(&**gdev**->**dev**,** "gpiochip%d"**,** gdev**->**id**);**

//初始化gpio\_device对应的device

//创建/sys/bus/gpio/devices/gpiochipx

//创建/sys/devices/platform/samsung-pinctrl/gpiochipx

device\_initialize**(&**gdev**->**dev**);**

//设置device对应的drv\_data为gpio\_device

dev\_set\_drvdata**(&**gdev**->**dev**,** gdev**);**

**if** **(**chip**->**parent **&&** chip**->**parent**->**driver**)**

gdev**->**owner **=** chip**->**parent**->**driver**->**owner**;**

**else** **if** **(**chip**->**owner**)**

gdev**->**owner **=** chip**->**owner**;**

**else**

gdev**->**owner **=** THIS\_MODULE**;**

//分配gpio\_desc用于描述该bank下所有gpio的信息，每个gpio对应一个gpio\_desc结构体

gdev**->**descs **=** kcalloc**(**chip**->**ngpio**,** **sizeof(**gdev**->**descs**[**0**]),** GFP\_KERNEL**);**

//设置gpio\_device的名称

**if** **(**chip**->**label**)**

gdev**->**label **=** kstrdup**(**chip**->**label**,** GFP\_KERNEL**);**

//设置gpio\_device的个数

gdev**->**ngpio **=** chip**->**ngpio**;**

//设置gpio\_device的data数据为samsung\_pin\_bank

gdev**->**data **=** data**;**

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

//设置gpio\_device的gpio起始序号

gdev**->**base **=** base**;**

//将gpio\_device添加到gpio\_devices链表中

//保持链表的排序方式为[base, base + ngpio - 1]

status **=** gpiodev\_add\_to\_list**(**gdev**);**

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

//遍历gpio\_desc

**for** **(**i **=** 0**;** i **<** chip**->**ngpio**;** i**++)** **{**

struct gpio\_desc **\***desc **=** **&**gdev**->**descs**[**i**];**

//设置gpio\_desc对应的gpio\_device

desc**->**gdev **=** gdev**;**

//如果get\_direction为输出或者没有direction\_input函数

//认为该gpio只有输出功能，设置标志位FLAG\_IS\_OUT

**if** **(**chip**->**get\_direction**)** **{**

int dir **=** chip**->**get\_direction**(**chip**,** i**);**

**if** **(!**dir**)**

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

**}** **else** **if** **(!**chip**->**direction\_input**)** **{**

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

**}**

**}**

//如果支持pinctrl子系统，初始化pin\_ranges链表

//pin\_ranges是干什么用的？

#ifdef CONFIG\_PINCTRL

INIT\_LIST\_HEAD**(&**gdev**->**pin\_ranges**);**

#endif

//这个函数这里没有作用，直接返回了

status **=** gpiochip\_set\_desc\_names**(**chip**);**

//这个函数这里没有作用，直接返回了

status **=** gpiochip\_irqchip\_init\_valid\_mask**(**chip**);**

//分析dts gpio相关的属性

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

//为每个bank创建字符设备，gpio\_device对应的id作为次设备号

**if** **(**gpiolib\_initialized**)** **{**

status **=** gpiochip\_setup\_dev**(**gdev**);**

**}**

**return** 0**;**

**}**

GPIO子系统为每个bank分配了一个gpio\_device结构体用于描述该bank，gpio\_device中有gpio\_desc描述每个gpio pin。

struct gpio\_device **{**

//gpio\_device的id，由系统分配

int id**;**

//对应的device

struct device dev**;**

//对应的字符设备结构体

struct cdev chrdev**;**

//防止重复生成/sys/class/gpio/gpiochipx

struct device **\***mockdev**;**

//模块所有者

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

//对应的gpio\_chip

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

//每个gpio pin对应的结构体

struct gpio\_desc **\***descs**;**

//起始pin number

int base**;**

//该bank有多少个gpio

u16 ngpio**;**

//bank名字

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

//这里对应为samsung\_pin\_bank

void **\***data**;**

struct list\_head list**;**

#ifdef CONFIG\_PINCTRL

struct list\_head pin\_ranges**;**

#endif

**};**

##### 1.2.3.2 of\_gpiochip\_add

of\_gpiochip\_add用于分析dts树中和gpio相关节点的信息，方便其他节点调用gpio。

int of\_gpiochip\_add**(**struct gpio\_chip **\***chip**)**

**{**

int status**;**

//每个gpio\_chip有对应的device node

//该步骤跳过

**if** **((!**chip**->**of\_node**)** **&&** **(**chip**->**parent**))**

chip**->**of\_node **=** chip**->**parent**->**of\_node**;**

//如果没有提供of\_xlate，则设置默认的of\_xlate函数

**if** **(!**chip**->**of\_xlate**)** **{**

chip**->**of\_gpio\_n\_cells **=** 2**;**

chip**->**of\_xlate **=** of\_gpio\_simple\_xlate**;**

**}**

//分析gpio-ranges-group-names gpio-ranges等dts信息，这里没有定义

status **=** of\_gpiochip\_add\_pin\_range**(**chip**);**

//分析gpio-line-names dts信息，这里没有定义

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

of\_gpiochip\_set\_names**(**chip**);**

//device node计数增加

of\_node\_get**(**chip**->**of\_node**);**

//分析gpio-hog dts信息，这里没有定义

**return** of\_gpiochip\_scan\_gpios**(**chip**);**

**}**

##### 1.2.3.3 gpiochip\_setup\_dev

gpiochip\_setup\_dev为每个bank创建一个字符设备和相关sys调试文件。

static int gpiochip\_setup\_dev**(**struct gpio\_device **\***gdev**)**

**{**

int status**;**

//创建字符设备

//设置字符设备操作函数集

cdev\_init**(&**gdev**->**chrdev**,** **&**gpio\_fileops**);**

//设置字符设备拥有者

gdev**->**chrdev**.**owner **=** THIS\_MODULE**;**

//设置字符设备对应的parent device

gdev**->**chrdev**.**kobj**.**parent **=** **&**gdev**->**dev**.**kobj**;**

//设置字符设备主次设备号

gdev**->**dev**.**devt **=** MKDEV**(**MAJOR**(**gpio\_devt**),** gdev**->**id**);**

status **=** cdev\_add**(&**gdev**->**chrdev**,** gdev**->**dev**.**devt**,** 1**);**

//添加device到内核中

// /sys/platform/samsung-pinctrl/gpiochipx/dev

// /sys/bus/gpio/devices/gpiochipx/dev

status **=** device\_add**(&**gdev**->**dev**);**

//创建/sys/class/gpio/gpiochipx及相关调试文件

status **=** gpiochip\_sysfs\_register**(**gdev**);**

//当销毁device时调用的函数

gdev**->**dev**.**release **=** gpiodevice\_release**;**

**return** 0**;**

**}**

int gpiochip\_sysfs\_register**(**struct gpio\_device **\***gdev**)**

**{**

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

struct device **\***parent**;**

struct gpio\_chip **\***chip **=** gdev**->**chip**;**

//判断gpio\_class是否存在

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

**return** 0**;**

//这里parent = chip->parent

**if** **(**chip**->**parent**)**

parent **=** chip**->**parent**;**

**else**

parent **=** **&**gdev**->**dev**;**

//创建/sys/class/gpio/gpiochipx

//该目录下有base label ngpio调试文件(gpiochip\_groups)

dev **=** device\_create\_with\_groups**(&**gpio\_class**,** parent**,**

MKDEV**(**0**,** 0**),**

chip**,** gpiochip\_groups**,**

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

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

gdev**->**mockdev **=** dev**;**

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

**return** 0**;**

**}**

##### 1.2.3.4 设备树方式使用GPIO

以模拟i2c为例子，说明dts树下使用gpio的方法。

###### 1.2.3.4.1 从设备树中获取gpio对应的pin number

dts树中关于gpio的描述：

i2c\_gpio\_1**:** i2c**-**gpio**-**1 **{**

compatible **=** "i2c-gpio"**;**

#address-cells = <1>;

#size-cells = <0>;

gpios **=** **<&**gpe 15 GPIO\_ACTIVE\_HIGH**>,** // SDA

**<&**gpe 14 GPIO\_ACTIVE\_HIGH**>;** // SCL

i2c**-**gpio**,**delay**-**us **=** **<**5**>;** // 100KHz

status **=** "okay"**;**

**};**

驱动中获取对应gpio的方法：

static int i2c\_gpio\_probe**(**struct platform\_device **\***pdev**)**

**{**

of\_i2c\_gpio\_get\_pins**(**pdev**->**dev**.**of\_node**,&**sda\_pin**,** **&**scl\_pin**);**

devm\_gpio\_request**(&**pdev**->**dev**,** sda\_pin**,** "sda"**);**

devm\_gpio\_request**(&**pdev**->**dev**,** scl\_pin**,** "scl"**);**

**}**

首先需要从dts树中获取gpio相关信息，获取函数为of\_i2c\_gpio\_get\_pins。

static int of\_i2c\_gpio\_get\_pins**(**struct device\_node **\***np**,** unsigned int **\***sda\_pin**,** unsigned int **\***scl\_pin**)**

**{**

//统计节点中gpios属性，判断有几个gpio节点

//因为是i2c，所以至少需要两个gpio

**if** **(**of\_gpio\_count**(**np**)** **<** 2**)**

**return** **-**ENODEV**;**

**\***sda\_pin **=** of\_get\_gpio**(**np**,** 0**);**

**\***scl\_pin **=** of\_get\_gpio**(**np**,** 1**);**

**return** 0**;**

**}**

从上面代码可以看出通过of\_get\_gpio从dts树中的gpio信息获取对应的gpio number。

static inline int of\_get\_gpio**(**struct device\_node **\***np**,** int index**)**

**{**

**return** of\_get\_gpio\_flags**(**np**,** index**,** **NULL);**

**}**

static inline int of\_get\_gpio\_flags**(**struct device\_node **\***np**,** int index**,** enum of\_gpio\_flags **\***flags**)**

**{**

**return** of\_get\_named\_gpio\_flags**(**np**,** "gpios"**,** index**,** flags**);**

**}**

int of\_get\_named\_gpio\_flags**(**struct device\_node **\***np**,** const char **\***list\_name**,** int index**,** enum of\_gpio\_flags **\***flags**)**

**{**

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

//根据<&gpe 15 GPIO\_ACTIVE\_HIGH>或<&gpe 14 GPIO\_ACTIVE\_HIGH>信息

//获取对应的gpio\_desc结构体

desc **=** of\_get\_named\_gpiod\_flags**(**np**,** list\_name**,** index**,** flags**);**

//获得对应的的gpio number

**return** desc\_to\_gpio**(**desc**);**

**}**

of\_get\_named\_gpiod\_flags函数代码分析如下：

struct gpio\_desc **\***of\_get\_named\_gpiod\_flags**(**struct device\_node **\***np**,** const char **\***propname**,** int index**,** enum of\_gpio\_flags **\***flags**)**

**{**

struct of\_phandle\_args gpiospec**;**

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

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

int ret**;**

//分析gpios的选项

//gpiospec保存了解析的相关参数

//device\_node为gpe, args\_count为2, args为15 GPIO\_ACTIVE\_HIGH 或者14 GPIO\_ACTIVE\_HIGH

ret **=** of\_parse\_phandle\_with\_args**(**np**,** propname**,** "#gpio-cells"**,** index**,** **&**gpiospec**);**

//遍历gpio\_devices链表,比对gpio\_devices和gpiospec中的device node

//如果相同，则找到了需要的gpio\_devices，则返回对应的gpio\_chip结构体

chip **=** of\_find\_gpiochip\_by\_xlate**(&**gpiospec**);**

//根据gpio\_chip信息和pin number信息，得到对应的gpio\_desc结构体

desc **=** of\_xlate\_and\_get\_gpiod\_flags**(**chip**,** **&**gpiospec**,** flags**);**

**return** desc**;**

**}**

static struct gpio\_chip **\***of\_find\_gpiochip\_by\_xlate**(**struct of\_phandle\_args **\***gpiospec**)**

**{**

**return** gpiochip\_find**(**gpiospec**,** of\_gpiochip\_match\_node\_and\_xlate**);**

**}**

struct gpio\_chip **\***gpiochip\_find**(**void **\***data**,**int **(\***match**)(**struct gpio\_chip **\***chip**,** void **\***data**))**

**{**

struct gpio\_device **\***gdev**;**

struct gpio\_chip **\***chip **=** **NULL;**

unsigned long flags**;**

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

//轮询gpio\_devices链表,获取gpio\_device结构体

//并调用match函数，找到对应的gpio\_chip结构体

list\_for\_each\_entry**(**gdev**,** **&**gpio\_devices**,** list**)**

**if** **(**gdev**->**chip **&&** match**(**gdev**->**chip**,** data**))** **{**

chip **=** gdev**->**chip**;**

**break;**

**}**

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

**return** chip**;**

**}**

static int of\_gpiochip\_match\_node\_and\_xlate**(**struct gpio\_chip **\***chip**,** void **\***data**)**

**{**

struct of\_phandle\_args **\***gpiospec **=** data**;**

//两者的device node相同，并且参数有数值

**return** chip**->**gpiodev**->**dev**.**of\_node **==** gpiospec**->**np **&&**

chip**->**of\_xlate**(**chip**,** gpiospec**,** **NULL)** **>=** 0**;**

**}**

在添加到gpio子系统的时候添加了of\_xlate的对应函数：

//args[0]存储的是pin number

//args[1]存储的是GPIO\_ACTIVE\_HIGH等信息。

int of\_gpio\_simple\_xlate**(**struct gpio\_chip **\***gc**,** const struct of\_phandle\_args **\***gpiospec**,** u32 **\***flags**)**

**{**

**if** **(**gc**->**of\_gpio\_n\_cells **<** 2**)** **{**

WARN\_ON**(**1**);**

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

**}**

**if** **(**WARN\_ON**(**gpiospec**->**args\_count **<** gc**->**of\_gpio\_n\_cells**))**

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

**if** **(**gpiospec**->**args**[**0**]** **>=** gc**->**ngpio**)**

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

**if** **(**flags**)**

**\***flags **=** gpiospec**->**args**[**1**];**

**return** gpiospec**->**args**[**0**];**

**}**

###### 1.2.3.4.2 申请gpio

现在已经拿到了gpio对应的pin number了，那么下面就是通过devm\_gpio\_request申请gpio了。

devm\_gpio\_request**(&**pdev**->**dev**,** sda\_pin**,** "sda"**);**

devm\_gpio\_request**(&**pdev**->**dev**,** scl\_pin**,** "scl"**);**

那么gpio\_request完成了什么功能呢？

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

**{**

//通过gpio number获取对应的gpio\_desc结构体

struct gpio\_desc **\***desc **=** gpio\_to\_desc**(**gpio**);**

**return** gpiod\_request**(**desc**,** label**);**

**}**

内核已经将gpio\_request函数放到gpiolib-legacy中了，是否表示已经过时了呢？

继续分析\_\_gpiod\_request。

static int \_\_gpiod\_request**(**struct gpio\_desc **\***desc**,** const char **\***label**)**

**{**

struct gpio\_chip **\***chip **=** desc**->**gdev**->**chip**;**

int status**;**

unsigned long flags**;**

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

//检查FLAG\_REQUESTED标志，看是否被申请了

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

//如果没有被申请，设置gpio\_desc的label属性，这里为scl或sda

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

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

**}**

//检查gpio\_chip是否有request函数

//这里会调用gpiochip\_generic\_request

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

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

status **=** chip**->**request**(**chip**,** gpio\_chip\_hwgpio**(**desc**));**

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

**}**

//调用get\_direction,这里未定义

**if** **(**chip**->**get\_direction**)** **{**

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

gpiod\_get\_direction**(**desc**);**

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

**}**

**}**

由于gpio\_chip定义了request回调函数gpiochip\_generic\_request，进一步分析：

int gpiochip\_generic\_request**(**struct gpio\_chip **\***chip**,** unsigned offset**)**

**{**

//申请一个pin作为gpio

**return** pinctrl\_request\_gpio**(**chip**->**gpiodev**->**base **+** offset**);**

**}**

到这里是否发现pinctrl已经和gpio有关联了。

虽然pinctrl提供了pinctrl\_request\_gpio()这样的API，但在代码中不可以直接调用pinctrl\_request\_gpio，当设备驱动申请一个gpio时，仍然需要调用gpio\_request()，这里会调用pinctrl\_request\_gpio()，调用过程如下：

gpio\_request**()**

gpiod\_request**()**

chip**->**request**(**chip**,**gpio\_chip\_hwgpio**(**desc**));**

同样地，对于pinctrl\_free\_gpio()、pinctrl\_gpio\_direction\_input()和pinctrl\_gpio\_direction\_output()也有类似说明。因此在clientdevice驱动中，申请和释放gpio仍然要调gpio\_request()、gpio\_free()；设置gpio为input/output仍然要调gpio\_direction\_input()和gpio\_direction\_output()。向pinctrl请求pin用作GPIO，成功后，该Pin仅能被GPIO驱动所使用，不能被其他驱动所混用。

由于pinctrl子系统代码还未分析，这里不详细分析pinctrl\_request\_gpio代码，该函数会根据gpio的pin number找到对应的pin control device和gpio range，并标记该pin已经用作gpio了，如果后面有模块request该资源，则会失败。

###### 1.2.3.4.3 gpio的操作

(1) direction

static void i2c\_gpio\_setsda\_dir**(**void **\***data**,** int state**)**

**{**

struct i2c\_gpio\_platform\_data **\***pdata **=** data**;**

**if** **(**state**)**

gpio\_direction\_input**(**pdata**->**sda\_pin**);**

**else**

gpio\_direction\_output**(**pdata**->**sda\_pin**,** 0**);**

**}**

static void i2c\_gpio\_setscl\_dir**(**void **\***data**,** int state**)**

**{**

struct i2c\_gpio\_platform\_data **\***pdata **=** data**;**

**if** **(**state**)**

gpio\_direction\_input**(**pdata**->**scl\_pin**);**

**else**

gpio\_direction\_output**(**pdata**->**scl\_pin**,** 0**);**

**}**

gpio\_direction\_input和gpio\_direction\_output代码比较简单，调用gpio注册的回调函数samsung\_gpio\_direction\_input和samsung\_gpio\_direction\_output完成输入输出的设置。

(2)value

static void i2c\_gpio\_setsda\_val**(**void **\***data**,** int state**)**

**{**

struct i2c\_gpio\_platform\_data **\***pdata **=** data**;**

gpio\_set\_value**(**pdata**->**sda\_pin**,** state**);**

**}**

static void i2c\_gpio\_setscl\_val**(**void **\***data**,** int state**)**

**{**

struct i2c\_gpio\_platform\_data **\***pdata **=** data**;**

gpio\_set\_value**(**pdata**->**scl\_pin**,** state**);**

**}**

static int i2c\_gpio\_getsda**(**void **\***data**)**

**{**

struct i2c\_gpio\_platform\_data **\***pdata **=** data**;**

**return** gpio\_get\_value**(**pdata**->**sda\_pin**);**

**}**

static int i2c\_gpio\_getscl**(**void **\***data**)**

**{**

struct i2c\_gpio\_platform\_data **\***pdata **=** data**;**

**return** gpio\_get\_value**(**pdata**->**scl\_pin**);**

**}**

gpio\_set\_value和gpio\_get\_value也是调用gpio注册时的回调函数samsung\_gpio\_set和samsung\_gpio\_get完成相关寄存器的设置。

(4)irq

gpio\_to\_irq最终也会调用samsung\_gpio\_to\_irq得到对应的irq number，该函数和中断相关，后面会进行分析。

#### 1.2.4 samsung\_pinctrl\_register