### 16.2 GPIO子系统重要概念

#### 16.2.1 引入

要操作GPIO引脚，先把所用引脚配置为GPIO功能，这通过Pinctrl子系统来实现。

然后就可以根据设置引脚方向(输入还是输出)、读值──获得电平状态，写值──输出高低电平。

以前我们通过寄存器来操作GPIO引脚，即使LED驱动程序，对于不同的板子它的代码也完全不同。

当BSP工程师实现了GPIO子系统后，我们就可以：

a. 在设备树里指定GPIO引脚

b. 在驱动代码中：

使用GPIO子系统的标准函数获得GPIO、设置GPIO方向、读取/设置GPIO值。

这样的驱动代码，将是单板无关的。

#### 16.2.2 在设备树中指定引脚

在几乎所有ARM芯片中，GPIO都分为几组，每组中有若干个引脚。所以在使用GPIO子系统之前，就要先确定：它是哪组的？组里的哪一个？

在设备树中，“GPIO组”就是一个GPIO Controller，这通常都由芯片厂家设置好。我们要做的是找到它名字，比如“gpio1”，然后指定要用它里面的哪个引脚，比如<&gpio1 0>。

有代码更直观，下图是一些芯片的GPIO控制器节点，它们一般都是厂家定义好，在xxx.dtsi文件中：



我们暂时只需要关心里面的这2个属性：

gpio-controller;

#gpio-cells = <2>;

“gpio-controller”表示这个节点是一个GPIO Controller，它下面有很多引脚。

“#gpio-cells = <2>”表示这个控制器下每一个引脚要用2个32位的数(cell)来描述。

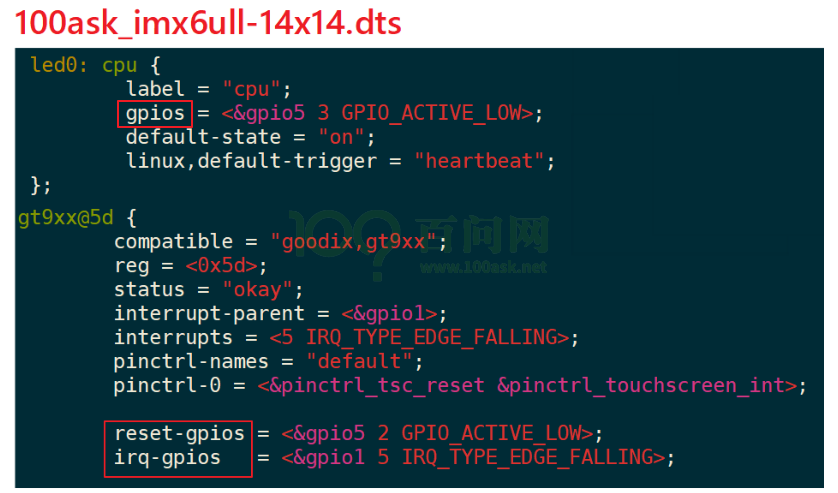
为什么要用2个数？其实使用多个cell来描述一个引脚，这是GPIO Controller自己决定的。比如可以用其中一个cell来表示那是哪一个引脚，用另一个cell来表示它是高电平有效还是低电平有效，甚至还可以用更多的cell来示其他特性。

普遍的用法是，用第1个cell来表示哪一个引脚，用第2个cell来表示有效电平：

GPIO\_ACTIVE\_HIGH ： 高电平有效

GPIO\_ACTIVE\_LOW : 低电平有效

定义GPIO Controller是芯片厂家的事，我们怎么引用某个引脚呢？在自己的设备节点中使用属性"[<name>-]gpios"，示例如下：



上图中，可以使用gpios属性，也可以使用name-gpios属性。

#### 16.2.3 在驱动代码中调用GPIO子系统

在设备树中指定了GPIO引脚，在驱动代码中如何使用？

也就是GPIO子系统的接口函数是什么？

GPIO子系统有两套接口：基于描述符的(descriptor-based)、老的(legacy)。前者的函数都有前缀“gpiod\_”，它使用gpio\_desc结构体来表示一个引脚；后者的函数都有前缀“gpio\_”，它使用一个整数来表示一个引脚。

要操作一个引脚，首先要get引脚，然后设置方向，读值、写值。

驱动程序中要包含头文件，

#include <linux/gpio/consumer.h> // descriptor-based

或

#include <linux/gpio.h> // legacy

下表列出常用的函数：

|  |  |  |
| --- | --- | --- |
| descriptor-based | legacy | 说明 |
| 获得GPIO | | |
| gpiod\_get | gpio\_request |  |
| gpiod\_get\_index |  |  |
| gpiod\_get\_array | gpio\_request\_array |  |
| devm\_gpiod\_get |  |  |
| devm\_gpiod\_get\_index |  |  |
| devm\_gpiod\_get\_array |  |  |
| 设置方向 | | |
| gpiod\_direction\_input | gpio\_direction\_input |  |
| gpiod\_direction\_output | gpio\_direction\_output |  |
| 读值、写值 | | |
| gpiod\_get\_value | gpio\_get\_value |  |
| gpiod\_set\_value | gpio\_set\_value |  |
| 释放GPIO | | |
| gpio\_free | gpio\_free |  |
| gpiod\_put | gpio\_free\_array |  |
| gpiod\_put\_array |  |  |
| devm\_gpiod\_put |  |  |
| devm\_gpiod\_put\_array |  |  |

有前缀“devm\_”的含义是“设备资源管理”(Managed Device Resource)，这是一种自动释放资源的机制。它的思想是“资源是属于设备的，设备不存在时资源就可以自动释放”。

比如在Linux开发过程中，先申请了GPIO，再申请内存；如果内存申请失败，那么在返回之前就需要先释放GPIO资源。如果使用devm的相关函数，在内存申请失败时可以直接返回：设备的销毁函数会自动地释放已经申请了的GPIO资源。

建议使用“devm\_”版本的相关函数。

举例，假设备在设备树中有如下节点：

foo\_device {

compatible = "acme,foo";

...

led-gpios = <&gpio 15 GPIO\_ACTIVE\_HIGH>, /\* red \*/

<&gpio 16 GPIO\_ACTIVE\_HIGH>, /\* green \*/

<&gpio 17 GPIO\_ACTIVE\_HIGH>; /\* blue \*/

power-gpios = <&gpio 1 GPIO\_ACTIVE\_LOW>;

};

那么可以使用下面的函数获得引脚：

struct gpio\_desc \*red, \*green, \*blue, \*power;

red = gpiod\_get\_index(dev, "led", 0, GPIOD\_OUT\_HIGH);

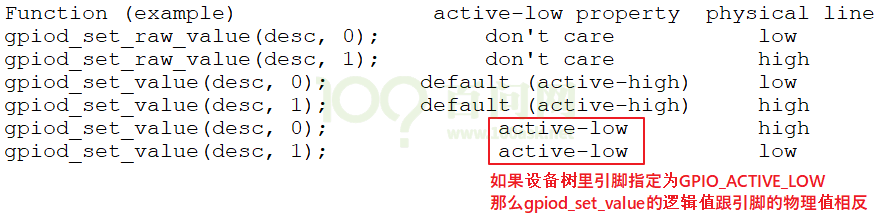
green = gpiod\_get\_index(dev, "led", 1, GPIOD\_OUT\_HIGH);

blue = gpiod\_get\_index(dev, "led", 2, GPIOD\_OUT\_HIGH);

power = gpiod\_get(dev, "power", GPIOD\_OUT\_HIGH);

要注意的是，gpiod\_set\_value设置的值是“逻辑值”，不一定等于物理值。

什么意思？



旧的“gpio\_”函数没办法根据设备树信息获得引脚，它需要先知道引脚号。

引脚号怎么确定？

在GPIO子系统中，每注册一个GPIO Controller时会确定它的“base number”，那么这个控制器里的第n号引脚的号码就是：base number + n。

但是如果硬件有变化、设备树有变化，这个base number并不能保证是固定的，应该查看sysfs来确定base number。

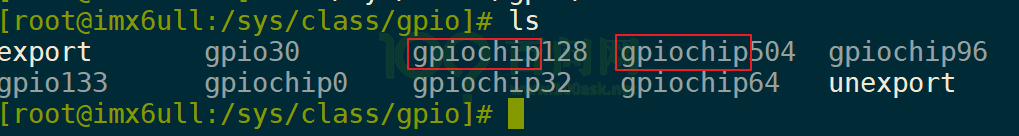
#### 16.2.4 sysfs中的访问方法\_IMX6ULL

在sysfs中访问GPIO，实际上用的就是引脚号，老的方法。

a. 先确定某个GPIO Controller的基准引脚号(base number)，再计算出某个引脚的号码。

方法如下：

① 先在开发板的/sys/class/gpio目录下，找到各个gpiochipXXX目录：



② 然后进入某个gpiochip目录，查看文件label的内容

③ 根据label的内容对比设备树

label内容来自设备树，比如它的寄存器基地址。用来跟设备树(dtsi文件)比较，就可以知道这对应哪一个GPIO Controller。

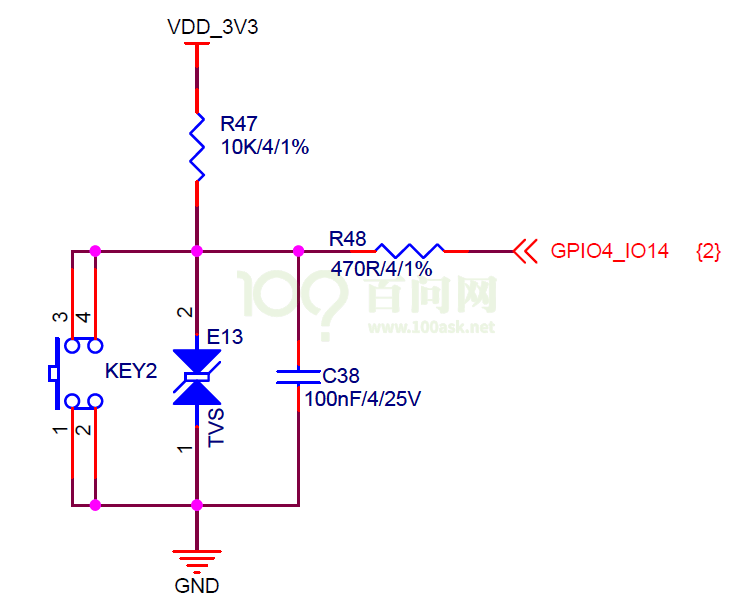
下图是在100asK\_imx6ull上运行的结果，通过对比设备树可知gpiochip96对应gpio4：



所以gpio4这组引脚的基准引脚号就是96，这也可以“cat base”来再次确认。

b. 基于sysfs操作引脚：

以100ask\_imx6ull为例，它有一个按键，原理图如下：



那么GPIO4\_14的号码是96+14=110，可以如下操作读取按键值：

echo 110 > /sys/class/gpio/export

echo in > /sys/class/gpio/gpio110/direction

cat /sys/class/gpio/gpio110/value

echo 110 > /sys/class/gpio/unexport

**注意**：如果驱动程序已经使用了该引脚，那么将会export失败，会提示下面的错误：



对于输出引脚，假设引脚号为N，可以用下面的方法设置它的值为1：

echo N > /sys/class/gpio/export

echo out > /sys/class/gpio/gpioN/direction

echo 1 > /sys/class/gpio/gpioN/value

echo N > /sys/class/gpio/unexport

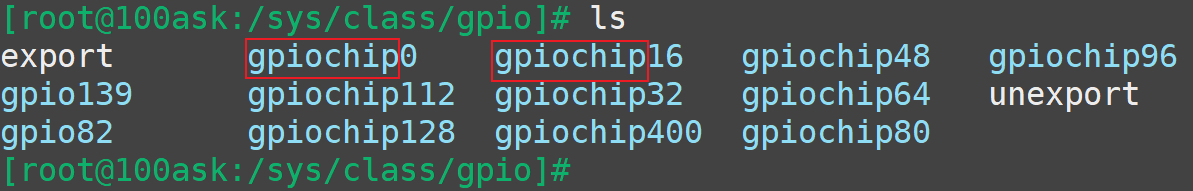
#### 16.2.5 sysfs中的访问方法\_STM32MP157

在sysfs中访问GPIO，实际上用的就是引脚号，老的方法。

a. 先确定某个GPIO Controller的基准引脚号(base number)，再计算出某个引脚的号码。

方法如下：

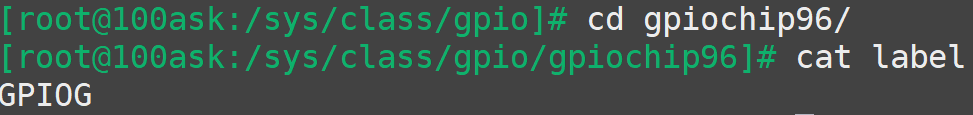
① 先在开发板的/sys/class/gpio目录下，找到各个gpiochipXXX目录：



② 然后进入某个gpiochip目录，查看文件label的内容

③ 根据label的内容就知道它是哪组引脚

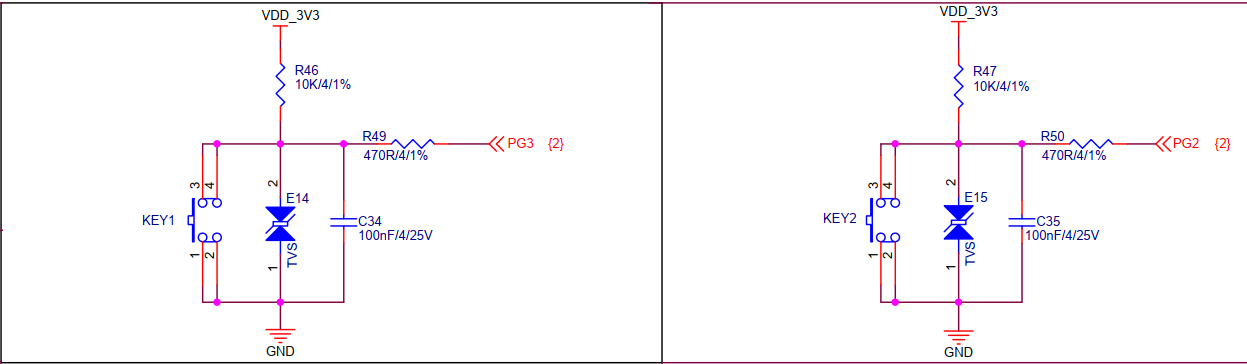
下图是在100ask\_stm32mp157上运行的结果，可知gpiochip96对应GPIOG：



所以GPIOG这组引脚的基准引脚号就是96，这也可以“cat base”来再次确认。

b. 基于sysfs操作引脚：

以100ask\_stm32mp157为例，它有一个按键，原理图如下：



那么PG2的号码是96+2=98，可以如下操作读取按键值：

echo 98 > /sys/class/gpio/export

echo in > /sys/class/gpio/gpio98/direction

cat /sys/class/gpio/gpio98/value

echo 98 > /sys/class/gpio/unexport

**注意**：如果驱动程序已经使用了该引脚，那么将会export失败，会提示下面的错误：



对于输出引脚，假设引脚号为N，可以用下面的方法设置它的值为1：

echo N > /sys/class/gpio/export

echo out > /sys/class/gpio/gpioN/direction

echo 1 > /sys/class/gpio/gpioN/value

echo N > /sys/class/gpio/unexport