15.2 GPIO 子系统重要概念

15.2.1 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：

- 在设备树里指定 GPIO 引脚
- 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。

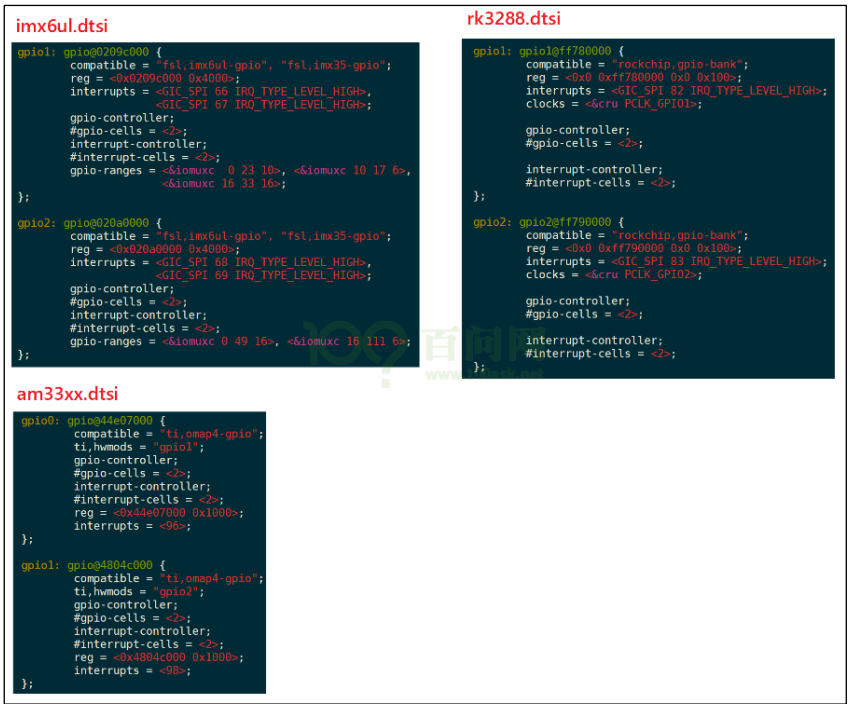
这样的驱动代码，将是单板无关的。

15.2.2 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：

100ask_imx6ull-14x14.dts

```
led0: cpu {
        label = "cpu";
        gpios = <&gpio5 3 GPIO_ACTIVE_LOW>;
        default-state = "on";
        linux,default-trigger = "heartbeat";
};
gt9xx@5d {
        compatible = "goodix,gt9xx";
        reg = <0x5d>;
        status = "okay";
        interrupt-parent = <&gpio1>;
        interrupts = <5 IRQ_TYPE_EDGE_FALLING>;
        pinctrl-names = "default";
        pinctrl-0 = <&pinctrl_tsc_reset &pinctrl_touchscreen_int>;

        reset-gpios = <&gpio5 2 GPIO_ACTIVE_LOW>;
        irq-gpios   = <&gpio1 5 IRQ_TYPE_EDGE_FALLING>;
```

上图中，可以使用 gpios 属性，也可以使用 name-gpios 属性。

15.2.3 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：

表 15-1 常用函数

descriptor-based	legacy
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。

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);
```

注意：gpiod_set_value 设置的值是“逻辑值”，不一定等于物理值。

什么意思？

Function (example)	active-low property	physical line
gpiod_set_raw_value(desc, 0);	don't care	low
gpiod_set_raw_value(desc, 1);	don't care	high
gpiod_set_value(desc, 0);	default (active-high)	low
gpiod_set_value(desc, 1);	default (active-high)	high
gpiod_set_value(desc, 0);	active-low	high
gpiod_set_value(desc, 1);	active-low	low

如果设备树里引脚指定为GPIO_ACTIVE_LOW
那么gpiod_set_value的逻辑值跟引脚的物理值相反

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。

15.2.4 sysfs 中的访问方法

在 sysfs 中访问 GPIO，实际上用的就是引脚号，老的方法。

- 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：

```
[root@imx6ull:/sys/class/gpio]# ls
export      gpio30      gpiochip128  gpiochip504  gpiochip96
gpio133     gpiochip0   gpiochip32   gpiochip64   unexport
[root@imx6ull:/sys/class/gpio]#
```

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。

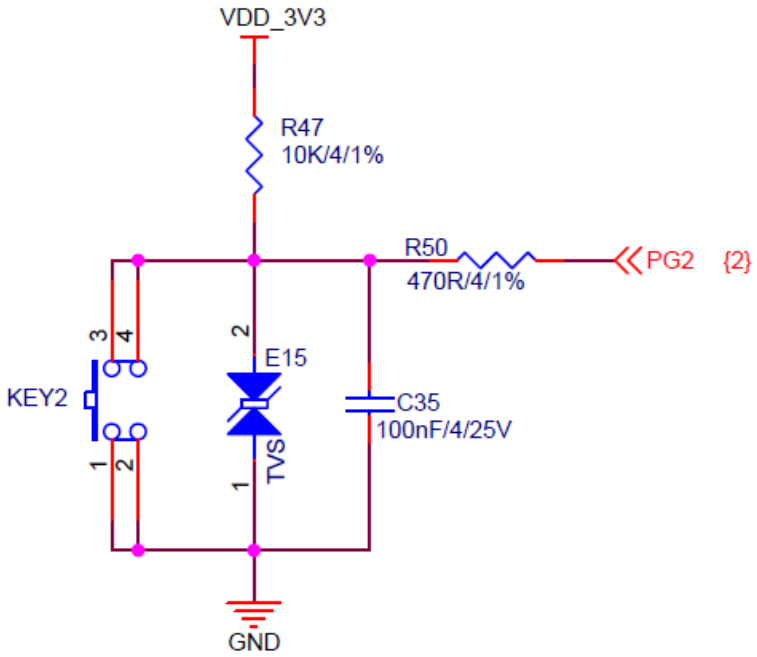
下图是在 stm32mp157 上运行的结果，通过对比设备树可知 gpiochip112 对应 gpioH：

```
[root@100ask:/sys/class/gpio]# ls
export       gpio82       gpiochip112  gpiochip16   gpiochip400  gpiochip64   gpiochip96
gpio139      gpiochip0    gpiochip128  gpiochip32   gpiochip48   gpiochip80   unexport
[root@100ask:/sys/class/gpio]# cd gpiochip112/
[root@100ask:/sys/class/gpio/gpiochip112]# ls -la
total 0
drwxr-xr-x    3 root     root             0 Jan  1  2000 .
drwxr-xr-x   11 root     root             0 Jan  1  2000 ..
-r--r--r--    1 root     root          4096 Feb  7 18:22 base
lrwxrwxrwx    1 root     root             0 Feb  7 18:22 device -> ../../../soc:pin-controller@50002000
-r--r--r--    1 root     root          4096 Feb  7 18:22 label
-r--r--r--    1 root     root          4096 Feb  7 18:22 ngpio
drwxr-xr-x    2 root     root             0 Feb  7 18:22 power
lrwxrwxrwx    1 root     root             0 Jan  1  2000 subsystem -> ../../../../../../class/gpio
-rw-r--r--    1 root     root          4096 Jan  1  2000 uevent
[root@100ask:/sys/class/gpio/gpiochip112]# ls
base       device     label      ngpio      power      subsystem  uevent
[root@100ask:/sys/class/gpio/gpiochip112]# cat label
GPIOH
[root@100ask:/sys/class/gpio/gpiochip112]#
```

所以 gpioH 这组引脚的基准引脚号就是 112，这也可以“cat base”来再次确认。

- 基于 sysfs 操作引脚：

以 STM32MP157 为例，它有一个按键，原理图如下：



那么 GPIOG_02 的号码是 96+2=98，可以如下操作读取按键值：

```
[root@100ask:~]# echo  98 > /sys/class/gpio/export
[root@100ask:~]# echo in > /sys/class/gpio/gpio98/direction
[root@100ask:~]# cat /sys/class/gpio/gpio98/value
[root@100ask:~]# echo  98 > /sys/class/gpio/unexport
```

```
[root@100ask:~]# echo  98 > /sys/class/gpio/export
[root@100ask:~]# echo in > /sys/class/gpio/gpio98/direction
[root@100ask:~]# cat /sys/class/gpio/gpio98/value
1                                    松开按键
[root@100ask:~]# cat /sys/class/gpio/gpio98/value
0                                    按下按键
[root@100ask:~]#
```

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：

```
[root@100ask:~]# echo  N > /sys/class/gpio/export
[root@100ask:~]# echo out > /sys/class/gpio/gpioN/direction
[root@100ask:~]# echo 1 > /sys/class/gpio/gpioN/value
[root@100ask:~]# echo  N > /sys/class/gpio/unexport
```