15.3 基于 GPIO 子系统的 LED 驱动程序

15.3.1 编写思路

GPIO 的地位跟其他模块，比如 I2C、UART 的地方是一样的，要使用某个引脚，需要先把引脚配置为 GPIO 功能，这要使用 Pinctrl 子系统，只需要在设备树里指定就可以。在驱动代码上不需要我们做任何事情。

GPIO 本身需要确定引脚，这也需要在设备树里指定。

设备树节点会被内核转换为 platform_device。

对应的，驱动代码中要注册一个 platform_driver，在 probe 函数中：获得引脚、注册 file_operations。

在 file_operations 中：设置方向、读值/写值。图 15.4 就是一个设备树的例子：

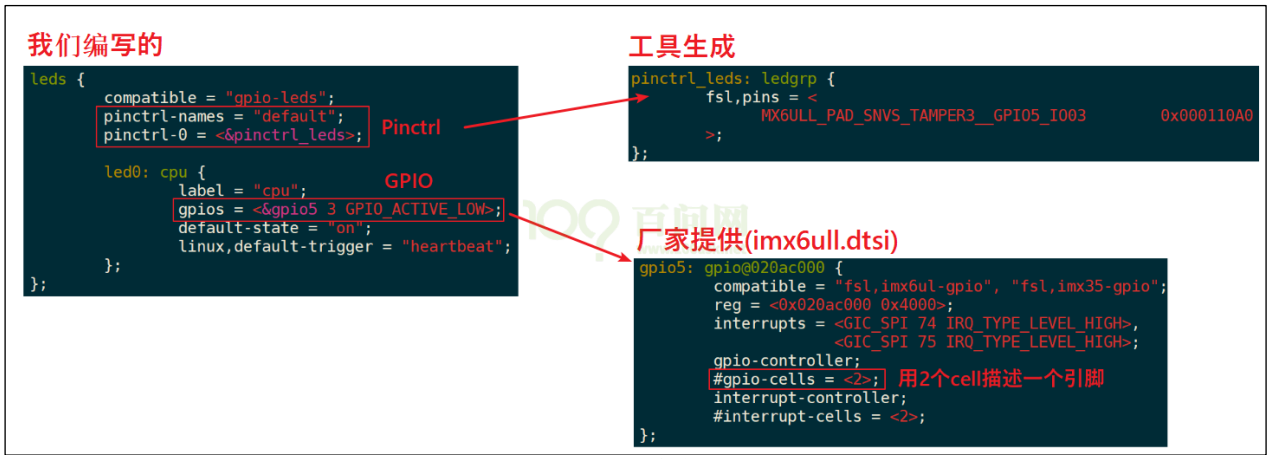

图 15.4 设备树示意

15.3.2 在设备树中添加 Pinctrl 信息

有些芯片提供了设备树生成工具，在 GUI 界面中选择引脚功能和配置信息，就可以自动生成 Pinctrl 子结点。把它复制到你的设备树文件中，再在 client device 结点中引用就可以。

有些芯片只提供文档，那就去阅读文档，一般在内核源码目录 Documentation\devicetree\bindings\pinctrl 下面，保存有该厂家的文档。

如果连文档都没有，那只能参考内核源码中的设备树文件，在内核源码目录 arch/arm/boot/dts 目录下。

最后一步，网络搜索。

Pinctrl 子节点的样式如下：

15.3.3 在设备树中添加 GPIO 信息

先查看电路原理图确定所用引脚，再在设备树中指定：添加”[name]-gpios”属性，指定使用的是哪一个 GPIO Controller 里的哪一个引脚，还有其他 Flag 信息，比如 GPIO_ACTIVE_LOW 等。具体需要多少个 cell 来描述一个引脚，需要查看设备树中这个 GPIO Controller 节点里的“#gpio-cells”属性值，也可以查看内核文档。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>;
```

15.3.4 编程示例

在实际操作过程中也许会碰到意外的问题，现场演示如何解决。

第1步 定义、注册一个 platform_driver

第2步 在它的 probe 函数里：

a) 根据 platform_device 的设备树信息确定 GPIO：gpiod_get
b) 定义、注册一个 file_operations 结构体
c) 在 file_operarions 中使用 GPIO 子系统的函数操作 GPIO：

```
gpiod_direction_output、gpiod_set_value
```

好处：这些代码对所有的板子都是完全一样的！

使用 GIT 命令载后，源码 leddrv.c 位于这个目录下：

```
01_all_series_quickstart\
05_嵌入式 Linux 驱动开发基础知识\source\05_gpio_and_pinctrl\01_led
```

摘录重点内容：

- 注册 platform_driver

注意下面第 122 行的"100ask,leddrv"，它会跟设备树中节点的 compatible 对应：

```
121 static const struct of_device_id ask100_leds[] = {
122     { .compatible = "100ask,leddrv" },
123     { },
124 };
125
126 /* 1. 定义 platform_driver */
127 static struct platform_driver chip_demo_gpio_driver = {
128     .probe      = chip_demo_gpio_probe,
129     .remove     = chip_demo_gpio_remove,
130     .driver     = {
131         .name   = "100ask_led",
132         .of_match_table = ask100_leds,
133     },
134 };
135
136 /* 2. 在入口函数注册 platform_driver */
137 static int __init led_init(void)
138 {
139     int err;
140
141     printk("%s %s line %d\n", __FILE__, __FUNCTION__, __LINE__);
142
143     err = platform_driver_register(&chip_demo_gpio_driver);
144
145     return err;
146 }
```

- 在 probe 函数中获得 GPIO

核心代码是第 87 行，它从该设备(对应设备树中的设备节点)获取名为“led”的引脚。在设备树中，必定有一属性名为“led-gpios”或“led-gpio”。

```
77 /* 4. 从 platform_device 获得 GPIO
78  *    把 file_operations 结构体告诉内核：注册驱动程序
79  */
80 static int chip_demo_gpio_probe(struct platform_device *pdev)
81 {
82      //int err;
83
84      printk("%s %s line %d\n", __FILE__, __FUNCTION__, __LINE__);
85
86      /* 4.1 设备树中定义有: led-gpios=<...>; */
87     led_gpio = gpiod_get(&pdev->dev, "led", 0);
88      if (IS_ERR(led_gpio)) {
89              dev_err(&pdev->dev, "Failed to get GPIO for led\n");
90              return PTR_ERR(led_gpio);
91      }
```

- 注册 file_operations 结构体：这是老套路了：

```
93      /* 4.2 注册 file_operations      */
94      major = register_chrdev(0, "100ask_led", &led_drv);  /* /dev/led */
95
96      led_class = class_create(THIS_MODULE, "100ask_led_class");
97      if (IS_ERR(led_class)) {
98              printk("%s %s line %d\n", __FILE__, __FUNCTION__, __LINE__);
99              unregister_chrdev(major, "led");
100             gpiod_put(led_gpio);
101             return PTR_ERR(led_class);
102     }
103
104     device_create(led_class, NULL, MKDEV(major, 0), NULL, "100ask_led%d", 0); /*
 /dev/100ask_led0 */
```

- 在 open 函数中调用 GPIO 函数设置引脚方向：

```
51 static int led_drv_open (struct inode *node, struct file *file)
52 {
53      //int minor = iminor(node);
54
55      printk("%s %s line %d\n", __FILE__, __FUNCTION__, __LINE__);
56      /* 根据次设备号初始化 LED */
57      gpiod_direction_output(led_gpio, 0);
58
59      return 0;
60 }
```

- 在 write 函数中调用 GPIO 函数设置引脚值：

```
34 /* write(fd, &val, 1); */
35 static ssize_t led_drv_write (struct file *file, const char __user *buf, size_t s
ize, loff_t *offset)
36 {
37      int err;
38      char status;
39      //struct inode *inode = file_inode(file);
40      //int minor = iminor(inode);
41
42      printk("%s %s line %d\n", __FILE__, __FUNCTION__, __LINE__);
43      err = copy_from_user(&status, buf, 1);
44
45      /* 根据次设备号和 status 控制 LED */
```

```
46      gpiod_set_value(led_gpio, status);
47
48      return 1;
49 }
```

- 释放 GPIO：

```
gpiod_put(led_gpio);
```

15.4 在 100ASK_STM32MP157_Pro 上机实验

15.4.1 确定引脚并生成设备树节点

ST 公司对于 STM32MP157 系列芯片，GPIO 为默认模式 不需要再进行配置 Pinctrl 信息。

a.Pinctrl 信息：

b.设备节点信息(放在 arch/arm/boot/dts/stm32mp157c-100ask-512d-lcd-v1.dts 根节点下)：

```
        myled {
            compatible = "100ask,leddrv";
            pinctrl-names = "default";
            led-gpios = <&gpioG 2 GPIO_ACTIVE_LOW>;
        };
```

15.4.2 编译程序

编译设备树后，要更新设备树。编译驱动程序时，“leddrv_未测试的原始版本.c”是有错误信息的，“leddrv.c”是修改过的。

测试方法，在板子上执行命令：

```
[root@100ask:~]# insmod  leddrv.ko
[root@100ask:~]# ls /dev/100ask_led0
[root@100ask:~]# ./ledtest /dev/100ask_led0 on
[root@100ask:~]# ./ledtest /dev/100ask_led0 off
```