第9章 LED 模板驱动程序的改造：总线设备驱动模型

9.1 原来的框架

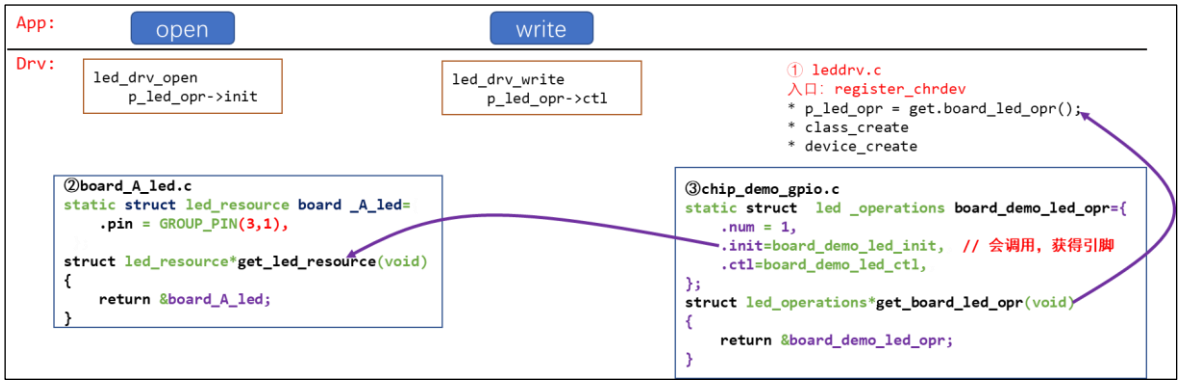

图 9.1 原本的驱动框架

9.2 要实现的框架

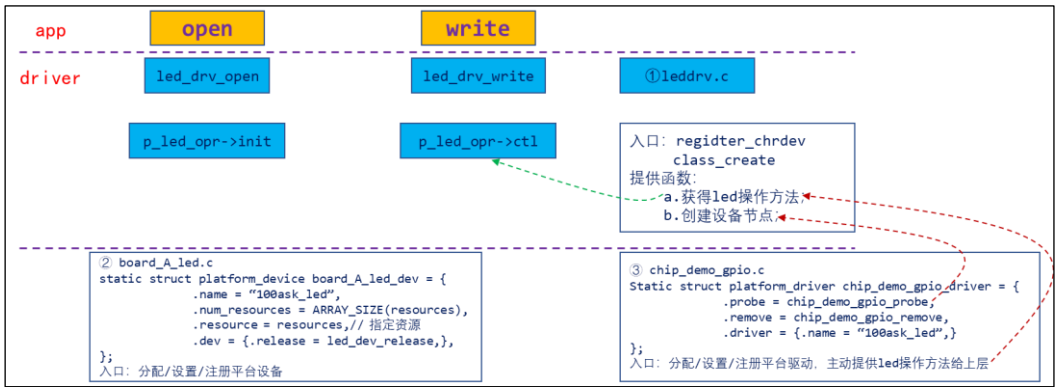

图 9.2 目标框架

9.3 写代码

使用 GIT 下载所有源码后，本节源码位于如下目录：

```
01_all_series_quickstart\
05_嵌入式 Linux 驱动开发基础知识\
source\02_led_drv\04_led_drv_template_bus_dev_drv
```

9.3.1 注意事项

① 如果 platform_device 中不提供 release 函数，如下图所示不提供红框部分的函数：

```
static struct platform_device board_A_led_dev = {
        .name = "100ask_led",
        .num_resources = ARRAY_SIZE(resources),
        .resource = resources,
        .dev = {
                .release = led_dev_release,
        },
};
```

图 9.3 release device

则在调用 platform_device_unregister 时会出现警告，如下图所示：

图 9.4 不释放的警告

你可以提供一个 release 函数，如果实在无事可做，把这函数写为空。

② EXPORT_SYMBOL

a.c 编译为 a.ko，里面定义了 func_a；如果它想让 b.ko 使用该函数，那么 a.c 里需要导出此函数(如果 a.c，b.c 都编进内核，则无需导出)：

```
EXPORT_SYMBOL(led_device_create);
```

并且，使用时要先加载 a.ko。

如果先加载 b.ko，会有类似如下“Unknown symbol”的提示：

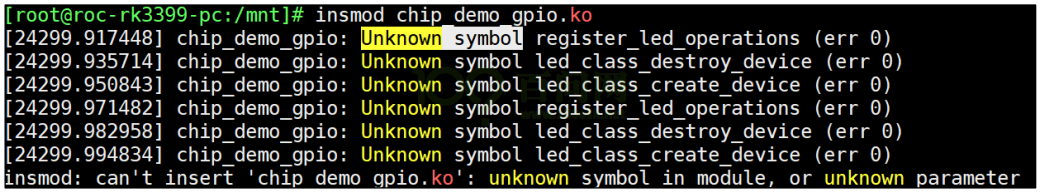

图 9.5 先加载 b.ko

9.3.2 实现 platform_device 结构体

board_A.c 作为一个可加载模块，里面也有入口函数、出口函数。在入口函数中注册 platform_device 结构体，在 platform_device 结构体中指定使用哪个 GPIO 引脚。

首先看入口函数，它调用 platform_device_register 函数，向内核注册 board_A_led_dev 结构体：

```
50 static int __init led_dev_init(void)
51 {
52     int err;
53
54     err = platform_device_register(&board_A_led_dev);
55
56     return 0;
57 }
```

board_A_led_dev 结构体定义如下。

```
23 static void led_dev_release(struct device *dev)
24 {
25 }
26
27 static struct resource resources[] = {
28     {
29         .start = GROUP_PIN(3,1),
30         .flags = IORESOURCE_IRQ,
31         .name = "100ask_led_pin",
32     },
33     {
34         .start = GROUP_PIN(5,8),
35         .flags = IORESOURCE_IRQ,
```

```
36         .name = "100ask_led_pin",
37     },
38 };
39
40
41 static struct platform_device board_A_led_dev = {
42          .name = "100ask_led",
43          .num_resources = ARRAY_SIZE(resources),
44          .resource = resources,
45          .dev = {
46                  .release = led_dev_release,
47           },
48 };
```

在 resouces 数组中指定了 2 个引脚(第 27～38 行)；

我们还提供了一个空函数 led_dev_release(第 23～25 行)，它被赋给 board_A_led_dev 结构体(第 46 行)，这个函数在卸载 platform_device 时会被调用，如果不提供的话内核会打印警告信息。

9.3.3 实现 platform_driver 结构体

chip_demo_gpio.c 中注册 platform_driver 结构体，它使用 Bus/Dev/Drv 模型，当有匹配的 platform_device 时，它的 probe 函数就会被调用。

在 probe 函数中所做的事情跟之前的代码没有差别。

先看入口函数：

```
138 static struct platform_driver chip_demo_gpio_driver = {
139     .probe      = chip_demo_gpio_probe,
140     .remove     = chip_demo_gpio_remove,
141     .driver     = {
142         .name   = "100ask_led",
143     },
144 };
145
146 static int __init chip_demo_gpio_drv_init(void)
147 {
148     int err;
149
150     err = platform_driver_register(&chip_demo_gpio_driver);
151     register_led_operations(&board_demo_led_opr);
152
153     return 0;
154 }
```

第 150 行向内核注册一个 platform_driver 结构体，这个结构体的核心在于第 140 行的 chip_demo_gpio_probe 函数。

chip_demo_gpio_probe 函数代码如下：。

```
100 static int chip_demo_gpio_probe(struct platform_device *pdev)
101 {
102     struct resource *res;
103     int i = 0;
104
```

```
105     while (1)
106     {
107         res = platform_get_resource(pdev, IORESOURCE_IRQ, i++);
108         if (!res)
109             break;
110
111         g_ledpins[g_ledcnt] = res->start;
112         led_class_create_device(g_ledcnt);
113         g_ledcnt++;
114     }
115     return 0;
116
117 }
```

- 第 107 行：从匹配的 platform_device 中获取资源，确定 GPIO 引脚。
- 第 111 行：把引脚记录下来，在操作硬件时要用。
- 第 112 行：新发现了一个 GPIO 引脚，就调用上层驱动的代码创建设备节点

操作硬件的代码如下，第 31、63 行的代码里用到了数组 g_ledpins，里面的值来自 platform_device，在 probe 函数中根据 platform_device 的资源确定了引脚：

```
23 static int g_ledpins[100];
24 static int g_ledcnt = 0;
25
26 static int board_demo_led_init (int which) /* 初始化 LED, which-哪个 LED */
27 {
28     //printk("%s %s line %d, led %d\n", __FILE__, __FUNCTION__, __LINE__, which);
29
30   printk("init gpio: group %d, pin %d\n", GROUP(g_ledpins[which]), PIN(g_ledpins
[which]));
31     switch(GROUP(g_ledpins[which]))
32     {
33         case 0:
34         {
35             printk("init pin of group 0 ...\n");
36             break;
37         }
38         case 1:
39         {
40             printk("init pin of group 1 ...\n");
41             break;
42         }
43         case 2:
44         {
45             printk("init pin of group 2 ...\n");
46             break;
47         }
48         case 3:
49         {
50             printk("init pin of group 3 ...\n");
51             break;
52         }
53     }
54
55     return 0;
```

```
56 }
57
58 static int board_demo_led_ctl (int which, char status) /* 控制 LED, which-哪个 LED,
 status:1-亮,0-灭 */
59 {
60     //printk("%s %s line %d, led %d, %s\n", __FILE__, __FUNCTION__, __LINE__, whi
ch, status ? "on" : "off");
61     printk("set led %s: group %d, pin %d\n", status ? "on" : "off", GROUP(g_ledpin
s[which]), PIN(g_ledpins[which]));
62
63     switch(GROUP(g_ledpins[which]))
64     {
65         case 0:
66         {
67             printk("set pin of group 0 ...\n");
68             break;
69         }
70         case 1:
71         {
72             printk("set pin of group 1 ...\n");
73             break;
74         }
75         case 2:
76         {
77             printk("set pin of group 2 ...\n");
78             break;
79         }
80         case 3:
81         {
82             printk("set pin of group 3 ...\n");
83             break;
84         }
85     }
86
87     return 0;
88 }
89
90 static struct led_operations board_demo_led_opr = {
91     .init = board_demo_led_init,
92     .ctl  = board_demo_led_ctl,
93 };
94
95 struct led_operations *get_board_led_opr(void)
96 {
97     return &board_demo_led_opr;
98 }
```

9.4 课后作业

完善半成品程序：04_led_drv_template_bus_dev_drv_unfinished。

请仿照本节提供的程序(位于 04_led_drv_template_bus_dev_drv 目录)，改造你所用的单板的 LED 驱动程序。