**linux平台总线驱动设备模型之点亮LED**

上一节中，我们详细分析了平台驱动设备模型的源码，懂得了框架是如何构成的。

上一节文章链接：<http://blog.csdn.net/lwj103862095/article/details/17957637>

这一节里，我们来使用平台驱动设备这一套架构来实现我们之前使用简单的字符设备驱动点亮LED，这里并无实际意义，只是告诉大家如果编写平台总线驱动设备。

问：如何编写平台总线驱动设备这一套架构的设备驱动？

答：分为两个.c文件，一个是drv.c，另一个是dev.c；前者实现平台驱动，后者实现平台设备，平台总线不用我们自己实现。

问：编写平台驱动的核心内容有哪些？

答：分配、设置、注册一个platform\_driver

问：如何注册平台驱动？

答：使用platform\_driver\_register(struct platform\_driver \*drv)函数，该函数的参数为platform\_driver

问：如何定义platform\_driver？

答：简单示例

**[cpp]** [view plain](http://blog.csdn.net/lwj103862095/article/details/17842437) [copy](http://blog.csdn.net/lwj103862095/article/details/17842437)

1. **static** **struct** platform\_driver led\_driver = {
2. .probe      = led\_probe,
3. .remove     = led\_remove,
4. .driver     = {
5. .name   = "myled",
6. .owner  = THIS\_MODULE,
7. }
8. };

问：probe函数什么时候被调用？

答：当系统中有同名的平台设备和平台驱动时，就会调用probe函数。

问：probe函数有什么作用？

答：该函数可以做什么由你决定，你可以只打印一条语句，也可以做很复杂的事情。例如，led\_probe函数就做了获取资源，映射IO，注册字符设备。

led\_drv.c源码参考：

**[cpp]** [view plain](http://blog.csdn.net/lwj103862095/article/details/17842437) [copy](http://blog.csdn.net/lwj103862095/article/details/17842437)

1. #include <linux/module.h>
3. #include <linux/init.h>
4. #include <linux/fs.h>
5. #include <linux/interrupt.h>
6. #include <linux/irq.h>
7. #include <linux/sched.h>
8. #include <linux/pm.h>
9. #include <linux/sysctl.h>
10. #include <linux/proc\_fs.h>
11. #include <linux/delay.h>
12. #include <linux/platform\_device.h>
13. #include <linux/input.h>
14. #include <linux/gpio\_keys.h>
15. #include <asm/uaccess.h>   // copy\_from\_user
16. #include <asm/io.h>  // ioremap
18. **static** **struct** **class** \*led\_cls;
20. **static** **volatile** unsigned **long** \*gpio\_con;
21. **static** **volatile** unsigned **long** \*gpio\_dat;
22. **static** **int** pin;
23. **static** **int** major;
25. **static** **int** led\_open(**struct** inode \* inode, **struct** file \* filp)
26. {
27. \*gpio\_con &= ~(0x3<<(pin\*2));
28. \*gpio\_con |= (0x1<<(pin\*2));
29. **return** 0;
30. }
32. **static** ssize\_t
33. led\_write(**struct** file \*file, **const** **char** \_\_user \*buf, **size\_t** count, loff\_t \*ppos)
34. {
35. **int** val;
36. copy\_from\_user(&val, buf, count);
37. **if**(val == 1)
38. {
39. /\* 点灯 \*/
40. \*gpio\_dat  &= ~(1<<pin);
41. }
42. **else**
43. {
44. /\* 灭灯 \*/
45. \*gpio\_dat  |= (1<<pin);
46. }
47. **return** 0;
48. }

51. /\* File operations struct for character device \*/
52. **static** **const** **struct** file\_operations led\_fops = {
53. .owner      = THIS\_MODULE,
54. .open       = led\_open,
55. .write      = led\_write,
56. };
58. **static** **int** \_\_devinit led\_probe(**struct** platform\_device \*pdev)
59. {
60. **struct** resource \*res;
62. printk("led\_probe, found led\n");
63. /\* 根据platform\_device的资源进行ioremap \*/
64. res = platform\_get\_resource(pdev, IORESOURCE\_MEM, 0);
65. gpio\_con =  ioremap(res->start, res->end - res->start + 1);
66. gpio\_dat = gpio\_con + 1;
68. res = platform\_get\_resource(pdev, IORESOURCE\_IRQ, 0);
69. pin =  res->start;
71. /\* 注册字符设备 \*/
72. major = register\_chrdev(0, "myled", &led\_fops);
74. led\_cls = class\_create(THIS\_MODULE,"myled");
75. device\_create(led\_cls, NULL, MKDEV(major, 0), NULL, "led"); /\* /dev/led \*/
76. **return** 0;
77. }
78. **static** **int** \_\_devexit led\_remove(**struct** platform\_device \*pdev)
79. {
80. printk("led\_remove, remove led\n");
81. device\_destroy(led\_cls, MKDEV(major, 0));
82. class\_destroy(led\_cls);
83. unregister\_chrdev(major, "myled");
84. iounmap(gpio\_con);
85. **return** 0;
86. }
88. **static** **struct** platform\_driver led\_driver = {
89. .probe      = led\_probe,
90. .remove     = led\_remove,
91. .driver     = {
92. .name   = "myled",
93. .owner  = THIS\_MODULE,
94. }
95. };
97. /\* 分配/设置/注册一个platform\_driver \*/
98. **static** **int** led\_drv\_init(**void**)
99. {
100. **return** platform\_driver\_register(&led\_driver);
101. }
103. **static** **void** led\_drv\_exit(**void**)
104. {
105. platform\_driver\_unregister(&led\_driver);
106. }
108. module\_init(led\_drv\_init);
109. module\_exit(led\_drv\_exit);
111. MODULE\_LICENSE("GPL");
112. MODULE\_AUTHOR("LWJ");
113. MODULE\_DESCRIPTION("Just for Demo");

问：编写平台设备驱动的核心内容有哪些？

答：分配、设置、注册一个platform\_device

问：如何注册平台设备？

答：使用platform\_device\_register(struct platform\_device \*pdev)函数，该函数的参数为platform\_device

问：如何定义platform\_device？

答：简单示例：led\_device

**[cpp]** [view plain](http://blog.csdn.net/lwj103862095/article/details/17842437) [copy](http://blog.csdn.net/lwj103862095/article/details/17842437)

1. **static** **struct** platform\_device led\_device = {
2. .id         = -1,
3. .name       = "myled",  /\* 与led\_driver的name一致 \*/
4. .resource       = led\_resources,
5. .num\_resources  = ARRAY\_SIZE(led\_resources),
6. .dev            ={
7. .release    = led\_release,
8. },
9. };

问：如何定义resource？

答：简单示例：

**[cpp]** [view plain](http://blog.csdn.net/lwj103862095/article/details/17842437) [copy](http://blog.csdn.net/lwj103862095/article/details/17842437)

1. **static** **struct** resource led\_resources[] = {
2. [0] = {
3. .start  = 0x56000010,      /\* TQ2440的LED是GPB5,6,7,8, GPBCON地址是0x56000010 \*/
4. .end    = 0x56000010 + 8 -1,
5. .flags  = IORESOURCE\_MEM,
6. },
7. [1] = {
8. .start  = 5,        /\* LED1 \*/
9. .end    = 5,
10. .flags  = IORESOURCE\_IRQ,
11. },
12. };

led\_dev.c源码参考：

**[cpp]** [view plain](http://blog.csdn.net/lwj103862095/article/details/17842437) [copy](http://blog.csdn.net/lwj103862095/article/details/17842437)

1. #include <linux/module.h>
2. #include <linux/version.h>
4. #include <linux/init.h>
6. #include <linux/kernel.h>
7. #include <linux/types.h>
8. #include <linux/interrupt.h>
9. #include <linux/list.h>
10. #include <linux/timer.h>
11. #include <linux/init.h>
12. #include <linux/serial\_core.h>
13. #include <linux/platform\_device.h>
15. **static** **struct** resource led\_resources[] = {
16. [0] = {
17. .start  = 0x56000010,      /\* TQ2440的LED是GPB5,6,7,8, GPBCON地址是0x56000010 \*/
18. .end    = 0x56000010 + 8 -1,
19. .flags  = IORESOURCE\_MEM,
20. },
21. [1] = {
22. .start  = 5,        /\* LED1 \*/
23. .end    = 5,
24. .flags  = IORESOURCE\_IRQ,
25. },
26. };
28. **static** **void** led\_release(**struct** device \* dev)
29. {
30. }
32. **static** **struct** platform\_device led\_device = {
33. .id         = -1,
34. .name       = "myled",  /\* 与led\_driver的name一致 \*/
35. .resource       = led\_resources,
36. .num\_resources  = ARRAY\_SIZE(led\_resources),
37. .dev            ={
38. .release    = led\_release,
39. },
40. };
42. /\* 分配/设置/注册一个platform\_device \*/
43. **static** **int** led\_dev\_init(**void**)
44. {
45. **return** platform\_device\_register(&led\_device);
46. }
48. **static** **void** led\_dev\_exit(**void**)
49. {
50. platform\_device\_unregister(&led\_device);
51. }
53. module\_init(led\_dev\_init);
54. module\_exit(led\_dev\_exit);
56. MODULE\_LICENSE("GPL");
57. MODULE\_AUTHOR("LWJ");
58. MODULE\_DESCRIPTION("Just for Demo");

应用测试程序源码：

**[cpp]** [view plain](http://blog.csdn.net/lwj103862095/article/details/17842437) [copy](http://blog.csdn.net/lwj103862095/article/details/17842437)

1. #include <sys/types.h>
2. #include <sys/stat.h>
3. #include <fcntl.h>
4. #include <stdio.h>
6. /\* 9th\_led\_test on
7. \* 9th\_led\_test off
8. \*/
9. **int** main(**int** argc, **char** \*\*argv)
10. {
11. **int** fd;
12. **int** val = 1;
13. fd = open("/dev/led", O\_RDWR);
14. **if** (fd < 0)
15. {
16. printf("can't open!\n");
17. }
18. **if** (argc != 2)
19. {
20. printf("Usage :\n");
21. printf("%s <on|off>\n", argv[0]);
22. **return** 0;
23. }
25. **if** (strcmp(argv[1], "on") == 0)
26. {
27. val  = 1;
28. }
29. **else**
30. {
31. val = 0;
32. }
34. write(fd, &val, 4);
35. **return** 0;
36. }

测试步骤：

**[cpp]** [view plain](http://blog.csdn.net/lwj103862095/article/details/17842437) [copy](http://blog.csdn.net/lwj103862095/article/details/17842437)

1. 9th\_led\_test        first\_drv.ko        sddisk
2. Qt                  first\_test          second\_drv.ko
3. TQLedtest           fourth\_drv.ko       second\_test
4. app\_test            fourth\_test         sixth\_drv.ko
5. bin                 home                sixth\_test
6. busybox             led\_dev.ko          sixthdrvtest
7. buttons\_all\_drv.ko  led\_drv.ko          sys
8. buttons\_all\_test    lib                 third\_drv.ko
9. buttons\_input.ko    linuxrc             third\_test
10. dev                 mnt                 tmp
11. driver\_test         opt                 udisk
12. etc                 proc                usr
13. fifth\_drv.ko        root                var
14. fifth\_test          sbin                web
15. [WJ2440]# insmod led\_drv.ko
16. [WJ2440]# insmod led\_dev.ko
17. led\_probe, found led
18. [WJ2440]# rmmod led\_dev
19. led\_remove, remove led
20. rmmod: module 'led\_dev' not found
21. [WJ2440]# lsmod
22. led\_drv 2800 0 - Live 0xbf003000
23. [WJ2440]# insmod led\_dev.ko
24. led\_probe, found led
25. [WJ2440]# lsmod
26. led\_dev 1444 0 - Live 0xbf009000
27. led\_drv 2800 0 - Live 0xbf003000
28. [WJ2440]# ls /dev/led -l
29. crw-rw----    1 root     root      252,   0 Jan  2 07:44 /dev/led
30. [WJ2440]# ./9th\_led\_test
31. Usage :
32. ./9th\_led\_test <on|off>
33. [WJ2440]# ./9th\_led\_test off
34. [WJ2440]# ./9th\_led\_test on

当执行./9th\_led\_test off时，led1被熄灭；当执行./9th\_led\_test on时 led1被点亮。如果你需要点亮led2，那么只需要修改led\_dev的led\_resources改为：

**[cpp]** [view plain](http://blog.csdn.net/lwj103862095/article/details/17842437) [copy](http://blog.csdn.net/lwj103862095/article/details/17842437)

1. **static** **struct** resource led\_resources[] = {
2. [0] = {
3. .start  = 0x56000010,      /\* TQ2440的LED是GPB5,6,7,8, GPBCON地址是0x56000010 \*/
4. .end    = 0x56000010 + 8 -1,
5. .flags  = IORESOURCE\_MEM,
6. },
7. [1] = {
8. .start  = 6,        /\* LED2 \*/
9. .end    = 6,
10. .flags  = IORESOURCE\_IRQ,
11. },
12. };

这样，应用程序不用更改，即可点亮led2，这样一来就实现了，稳定部分不用修改，只需要修改硬件易变部分，并且应用程序不需要任何更改。