### 16.3 基于GPIO子系统的LED驱动程序

本节视频既适用于“驱动基础”，也适用于“驱动大全”，所以它对应的源码在2个GIT仓库里都有。

驱动基础的GIT仓库：

01\_all\_series\_quickstart\

05\_嵌入式Linux驱动开发基础知识\source\

05\_gpio\_and\_pinctrl\

01\_led

驱动大全的GIT仓库：

doc\_and\_source\_for\_drivers\IMX6ULL\source\07\_GPIO\01\_led

doc\_and\_source\_for\_drivers\STM32MP157\source\A7\07\_GPIO\01\_led

#### 16.3.1 编写思路

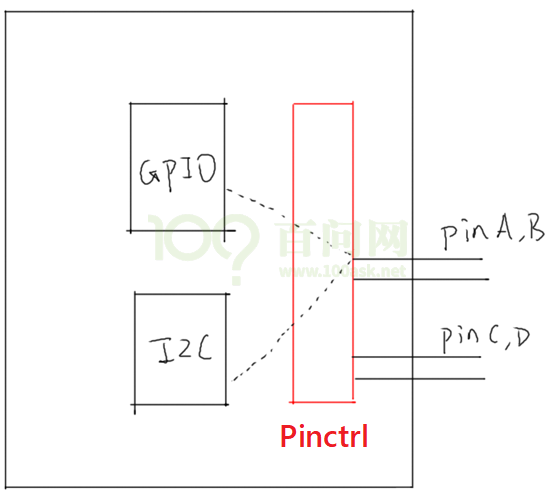
GPIO的地位跟其他模块，比如I2C、UART的地方是一样的，要使用某个引脚，需要先把引脚配置为GPIO功能，这要使用Pinctrl子系统，只需要在设备树里指定就可以。在驱动代码上不需要我们做任何事情。

GPIO本身需要确定引脚，这也需要在设备树里指定。

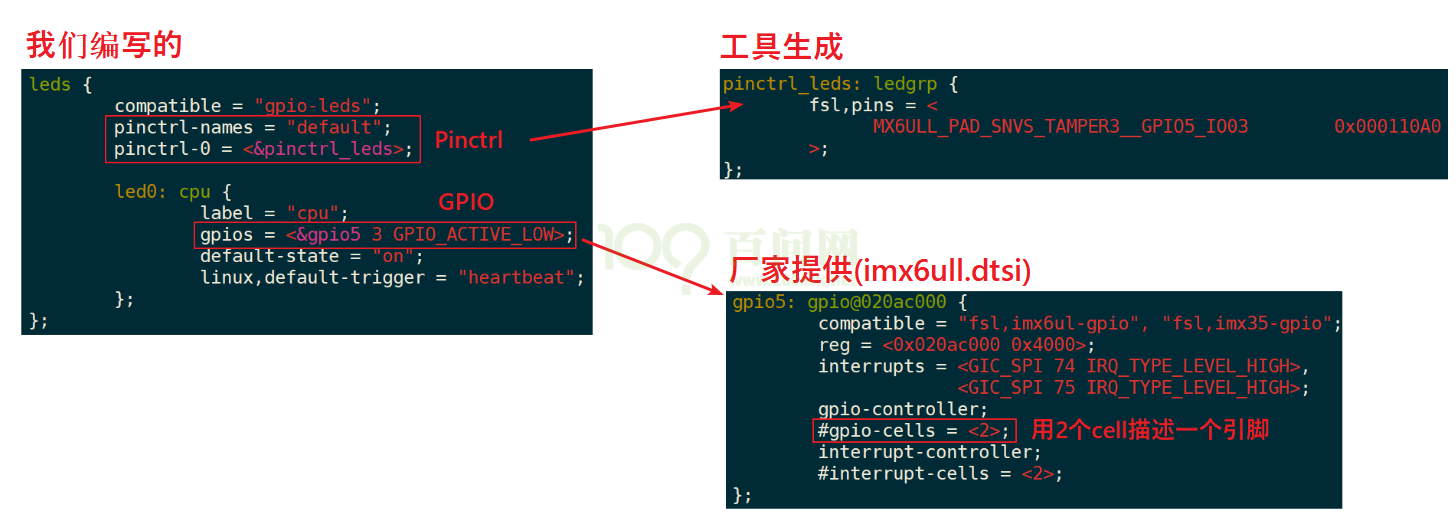
设备树节点会被内核转换为platform\_device。

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。



下图就是一个设备树的例子：



#### 16.3.2 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。

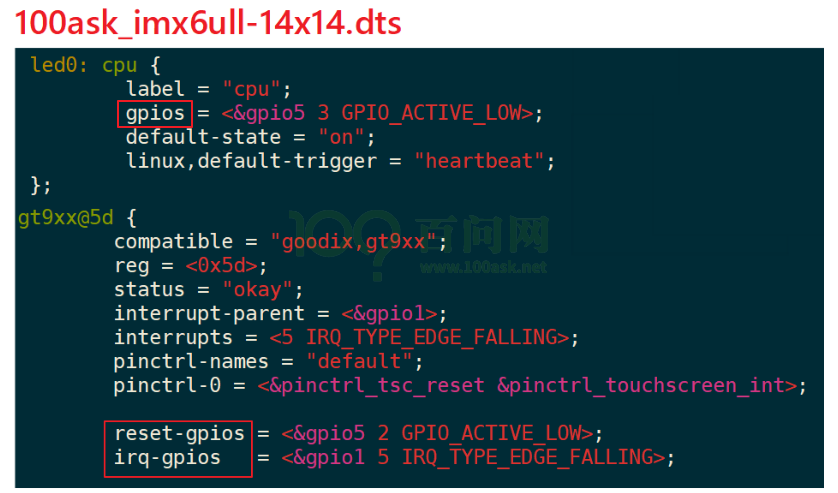
Pinctrl子节点的样式如下：



#### 16.3.3 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：



#### 16.3.4 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。

a. 定义、注册一个platform\_driver

b. 在它的probe函数里：

b.1 根据platform\_device的设备树信息确定GPIO：gpiod\_get

b.2 定义、注册一个file\_operations结构体

b.3 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2个目录下，文件是一样的：

doc\_and\_source\_for\_drivers\IMX6ULL\source\07\_GPIO\01\_led

doc\_and\_source\_for\_drivers\STM32MP157\source\A7\07\_GPIO\01\_led

摘录重点内容：

a. 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 }

b. 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 }

92

c. 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 \*/

105

d. 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 }

e. 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 size, 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 }

f. 释放GPIO：

gpiod\_put(led\_gpio);