# 一、内核自带LED驱动使能

1、内核自带的驱动，都是通过图形化界面配置，选择使能或者不使用。

输入：make menuconfig

使能驱动以后在.config里面就会存在：CONFIG\_LEDS\_GPIO=y

在linux内核源码里面一般驱动文件夹下Makefile会使用CONFIG\_XXX来决定要编译哪个文件。

obj-$(CONFIG\_LEDS\_GPIO) += leds-gpio.o

obj-y += leds-gpio.o **-> leds-gpio.c**

# 二、内核自带LED驱动分析

打开驱动文件，找到：

static struct platform\_driver gpio\_led\_driver = {

.probe = gpio\_led\_probe,

.remove = gpio\_led\_remove,

.driver = {

.name = "leds-gpio",

.of\_match\_table = of\_gpio\_leds\_match,

},

};

module\_platform\_driver(gpio\_led\_driver);

#define module\_platform\_driver(\_\_platform\_driver) \

module\_driver(\_\_platform\_driver, platform\_driver\_register, \

platform\_driver\_unregister)

#define module\_driver(\_\_driver, \_\_register, \_\_unregister, ...) \

static int \_\_init \_\_driver##\_init(void) \

{ \

return \_\_register(&(\_\_driver) , ##\_\_VA\_ARGS\_\_); \

} \

module\_init(\_\_driver##\_init); \

static void \_\_exit \_\_driver##\_exit(void) \

{ \

\_\_unregister(&(\_\_driver) , ##\_\_VA\_ARGS\_\_); \

} \

module\_exit(\_\_driver##\_exit);

展开：

module\_platform\_driver(gpio\_led\_driver);

module\_driver(gpio\_led\_driver, platform\_driver\_register, \

platform\_driver\_unregister)

#define module\_driver(\_\_driver, \_\_register, \_\_unregister, ...) \

static int \_\_init gpio\_led\_driver\_init(void)

{

return platform\_driver\_register (&gpio\_led\_driver);

}

module\_init(gpio\_led\_driver\_init);

static void \_\_exit \_\_driver##\_exit(void)

{

\_\_unregister(&(\_\_driver) , ##\_\_VA\_ARGS\_\_);

}

module\_exit(\_\_driver##\_exit);

根据gpio\_led\_driver可知，设备树的of\_match\_table为of\_gpio\_leds\_match，匹配表里面的匹配值：

static const struct of\_device\_id of\_gpio\_leds\_match[] = {

{ .compatible = "gpio-leds", },

{},

};

当驱动和设备匹配以后gpio\_led\_probe函数执行：

-> gpio\_leds\_create

-> create\_gpio\_led

# 三、内核自带LED驱动使用

1、首先将驱动编译进内核里面，

2、根据绑定文档在设备树里面添加对应的设备节点信息。

如果无设备树，那么就要使用platform\_device\_register向总线注册设备。

如果有设备树，那么就直接修改设备树，添加指定的节点。