Linux驱动编程操作GPIO的简要说明

原创 置顶 平仄散人 于 2019-02-21 00:10:55 发布 7923 收藏 59 版权
分类专栏： Linux开发 文章标签： Linux GPIO dts GPIO口

Linux开发 专栏收录该内容 18 订阅 56 篇文章 订阅专栏

gpio 简介

GPIO, 全称 General-Purpose Input/Output（通用输入输出），是一种软件运行期间能够动态配置和控制的通用引脚。Linux内核中gpio是最简单，最常用的资源。驱动程序，应用程序都能够通过相应的接口使用gpio，gpio使用0～MAX之间的整数标识，不能使用负数,gpio与硬件体系密切相关的,不过linux有一个框架处理gpio，能够使用统一的接口来操作gpio。使用gpio接口需要包含头文件#include <linux/gpio.h>和#include <linux/of_gpio.h>，Documentation/gpio.txt文件有相关说明。

RK3288有9组GPIO bank：GPIO0~GPIO8，每组又以 A0~A7, B0~B7, C0~C7, D0~D7 作为编号区分（不是所有 bank 都有全部编号，例如 GPIO0 就只有A0~C7)。 所有的GPIO在上电后的初始状态都是输入模式，可以通过软件设为上拉或下拉，也可以设置为中断脚，驱动强度都是可编程的。 每个 GPIO 口除了通用输入输出功能外，还可能有其它复用功能。

Linux常用的gpio API定义如下：

```
#include <linux/gpio.h>
#include <linux/of_gpio.h>

enum of_gpio_flags {
     OF_GPIO_ACTIVE_LOW = 0x1,
};

int of_get_named_gpio_flags(struct device_node *np, const char *propname,
int index, enum of_gpio_flags *flags);
int gpio_is_valid(int gpio);
int gpio_request(unsigned gpio, const char *label);
void gpio_free(unsigned gpio);
int gpio_direction_input(int gpio);
int gpio_direction_output(int gpio, int value);
void gpio_set_value(unsigned int gpio, int value);
int gpio_get_value(unsigned gpio);
int gpio_to_irq(unsigned int gpio);
void free_irq(unsigned int irq, void *dev_id);
```

操作普通GPIO说明

（1）、在dts添加gpio的引用描述：在dts文件上添加，通常在设备树中以 类似以下的配置来表示gpio的配置使用，这里定义了一个pin脚作为一般的输出输入口，rk3288平台的GPIO0_B5。

```
led_ctrl {

        compatible = "rk3288, led_ctrl";
    status = "okay";
        gpio-en = <&gpio1 13 GPIO_ACTIVE_HIGH>; //GPIO0_B5
};
```

（2）、解析dts并且获取gpio口：函数返回值就得到gpio号。

```
int of_get_named_gpio_flags(struct device_node *np, const char *propname, int index, enum of_gpio_flags *flags);
```

（3）、判断gpio口是否合法能用。

```
int gpio_is_valid(int number)
```

（4）、申请gpio口：gpio_request的第一个参数是需要申请的gpio号。第二个参数是我们给该gpio起个名字，申请成功后，可以通过/sys/kernel/debug/gpio文件查看到该GPIO的状态。

```
int gpio_request(unsigned gpio, const char *label)
```

（5）、设置gpio口的方向，如果是设置方向为输出的话同时设置电平拉高或拉低：

```
int gpio_direction_input(unsigned gpio); //设置gpio为输入

int gpio_direction_output(unsigned gpio, int value); //设置gpio为输出，且设置电平
```

（6）、操作gpio口（拉高或者拉低gpio的电平，value为1是拉高，0是拉低）：

```
void gpio_set_value(unsigned int gpio, int value);
```

（7）、获取gpio口的状态：get到1为高电平，得到0为低电平;

```
ret = gpio_get_value(unsigned gpio);
```

（8）、释放gpio口：在remove函数中添加对应的释放操作;

```
gpio_free(int number)
```

操作gpio API demo说明

1、dts里面的节点添加gpio的配置使用：

```
 leds {

            compatible = "rk3288, led";
            status = "okay";
            gpio_en = <&gpio4 11 GPIO_ACTIVE_HIGH>;

      };
```

2、然后再在驱动中probe函数解析，例如在这里的设备是leds，然后调用of_get_named_gpio_flags获取gpio口，然后调用gpio_direction_output接口设置gpio的电平。

```
        struct device_node *leds_node = pdev->dev.of_node;
        enum of_gpio_flags flags;

        gpio_en = of_get_named_gpio_flags(leds_node,"gpio-en", 0, &flags);
        pr_err("---[czd]--- gpio_en is %d --\n", gpio_en);
        if (!gpio_is_valid(gpio_en)) {
        pr_err("gpio_en: %d is invalid\n", gpio_en);
        return -ENODEV;
        }
        if (gpio_request(gpio_en, "gpio_en")) {
                pr_err("gpio_en: %d request failed!\n", gpio_en);

```

```
                gpio_free(gpio_en);
                return -ENODEV;
        }

        gpio_direction_output(gpio_en, 1); //这里设置GPIO的电平拉高
```

gpio用作中断脚说明

一、API函数：

1、这个函数的作用是转换gpio编号到对应irq号

```
static inline int gpio_to_irq(unsigned int gpio)
{
        return __gpio_to_irq(gpio);
}
```

2、中断注册函数

```
static inline int __must_check
request_irq(
                unsigned int irq,
                irq_handler_t handler,
                unsigned long flags,
                const char *name,
                void *dev_id)
```

参数说明：
irq中断号。（和平台架构相关，结合datasheet以及平台文件） IRQ_EINT(x)
中断处理函数： 中断发生时，系统调用这个函数，dev_id参数将被传递给它
中断标记： IRQ_TYPE_EDGE_FALLING 上升或下降中断触发
中断名字：
dev_id 一般使用设备的设备结构体或者NULL
返回值：request_irq()返回0表示成功，返回-EINVAL表示中断号无效或处理函数指针为NULL，返回-EBUSY表示中断已经被占用，且不能被共享。

3、中断卸载函数：

```
void free_irq(unsigned int irq, void *dev_id)
```

4、中断服务函数：

```
irq_handler_t irq16_handler(int irq,void *dev_id)
```

二、举例说明

1、GPIO口的中断使用与GPIO的输入输出类似，首先在DTS文件中增加gpio的配置。

```
key_gpio_irq {
    compatible = "key_gpio_irq";
    irq-gpio = <&gpio4 24 IRQ_TYPE_EDGE_RISING>;  /* GPIO4_D0,这里配置上升沿触发中断 */
};
```

IRQ_TYPE_EDGE_RISING表示中断由上升沿触发，当该引脚接收到上升沿信号时可以触发中断函数。 这里还可以配置成如下：

```
IRQ_TYPE_NONE         //默认值，无定义中断触发类型
IRQ_TYPE_EDGE_RISING  //上升沿触发
IRQ_TYPE_EDGE_FALLING //下降沿触发
IRQ_TYPE_EDGE_BOTH    //上升沿和下降沿都触发
IRQ_TYPE_LEVEL_HIGH   //高电平触发
IRQ_TYPE_LEVEL_LOW    //低电平触发
```

然后在probe函数中对dts所添加的配置进行解析，再做中断的注册申请，代码如下：

```
static irqreturn_t leds_gpio_irq(int irq, void *dev_id) //中断函数
{
    printk("Enter leds gpio irq test program!\n");
    return IRQ_HANDLED;
}

static int irq_gpio_probe(struct platform_device *pdev)
{
        struct device_node *leds_node = pdev->dev.of_node;
        enum of_gpio_flags flags;
        int ret;

        irq-gpio = of_get_named_gpio_flags(leds_node,"irq-gpio", 0, &flags);
        pr_err("---[czd]--- irq-gpio is %d --\n", irq-gpio);
        if (!gpio_is_valid(irq-gpio)) {
        pr_err("irq-gpio: %d is invalid\n", irq-gpio);
        return -ENODEV;
        }
        if (gpio_request(irq-gpio, "irq-gpio")) {
           printk("gpio %d request failed!\n", irq-gpio);
           gpio_free(irq-gpio);
           return IRQ_NONE;
     }
     ret = gpio_direction_input(irq-gpio);
        if (ret < 0)
        {
                pr_err("%s: set gpio direction input (%d) fail\n", __func__, wakeup_gpio);
                return ret;
        }
    irq-gpio_number = gpio_to_irq(irq-gpio); //转换gpio编号到对应irq号。
    if (irq-gpio) {
        ret = request_irq(irq-gpio_number, leds_gpio_irq, flag, "key_gpio_irq", key-gpio);
        if (ret != 0)
                free_irq(irq-gpio, key-gpio);
                dev_err(&pdev->dev, "Failed to request IRQ: %d\n", ret);
    }
    return 0;
}

```

调用gpio_to_irq把GPIO的PIN值转换为相应的IRQ值，调用gpio_request申请占用该GPIO口，调用request_irq申请中断，如果失败要调用free_irq释放，该函数中irq-gpio_number是要申请的硬件中断号，leds_gpio_irq是中断处理函数，flag是中断处理的属性：例如要设置上升沿触发中断则使用IRQ_TYPE_EDGE_RISING，”key_gpio_irq”是设备驱动程序名称，key-gpio是该设备的device结构，在注册共享中断时会用到，如果不用到可以设为NULL。

补充说明：

1、如果不知道自己使用的GPIO是否用对了，可以使用万用表量一下GPIO电平是否是自己控制的状态（高或者低电平）或者使用以下命令查看一下GPIO的情况：

cat /sys/kernel/debug/gpio

2、查看一下GPIO的复用情况：

cat /d/pinctrl/pinctrl/pinmux-pins //不同平台可能会有差异，一般关键字是pinmux，可以使用find命令查找一下