iTOP-4412 开发板的 GPIO 是怎么操作的？

Exynos4412 所有的 GPIO 都有固定的地址，为了方便操作这些 GPIO，Linux 内核在 gpio-exynos4.h 里面定义了一些 GPIO 的宏，例如：

```
#define EXYNOS4_GPA0(_nr)   (EXYNOS4_GPIO_A0_START + (_nr))
#define EXYNOS4_GPA1(_nr)   (EXYNOS4_GPIO_A1_START + (_nr))
#define EXYNOS4_GPB(_nr) (EXYNOS4_GPIO_B_START + (_nr))
.....................................
#define EXYNOS4_GPY5(_nr)   (EXYNOS4_GPIO_Y5_START + (_nr))
#define EXYNOS4_GPY6(_nr)   (EXYNOS4_GPIO_Y6_START + (_nr))
#define EXYNOS4_GPZ(_nr) (EXYNOS4_GPIO_Z_START + (_nr))
```

这些宏就是把每个 GPIO 的地址做了一下封装，它的好处就是方便我们使用并且根据宏的名字就能直观的知道是在操作哪个 GPIO。

Linux 内核中关于 GPIO 的驱动在 driver/gpio/gpio-exynos4.c 文件里面，在这个文件中 GPIO 驱动初始化入口函数是 exynos4_gpiolib_init，因为这个文件同时也支持 4210 的 GPIO，所以开始先初始化了一些通用的 GPIO（4412 和 4210 都有的 GPIO），代码如下：

```
    chip = exynos4_gpio_common_4bit;
    nr_chips = ARRAY_SIZE(exynos4_gpio_common_4bit);

    for (i = 0; i < nr_chips; i++, chip++) {
        if (chip->config == NULL)
            chip->config = &gpio_cfg;
        if (chip->base == NULL)
            pr_err("No allocation of base address for [common gpio]");
    }

    samsung_gpiolib_add_4bit_chips(exynos4_gpio_common_4bit,
nr_chips);
```

变量 exynos4_gpio_common_4bit 是一个数组，定义了一些通用的 GPIO，nr_chips 是记录的 exynos4_gpio_common_4bit 数组里面元素个数。

首先使用 for 循环遍历 exynos4_gpio_common_4bit 所有的元素，为每个元素的 config 结构赋值：

```
    if (chip->config == NULL)
            chip->config = &gpio_cfg;
```

gpio_cfg 是类型为 s3c_gpio_cfg 的结构体，这个结构体的定义如下：

```
struct s3c_gpio_cfg {
    unsigned int  cfg_eint;

    s3c_gpio_pull_t   (*get_pull)(struct s3c_gpio_chip *chip, unsigned
offs);
    int    (*set_pull)(struct s3c_gpio_chip *chip, unsigned offs,
```

```
                    s3c_gpio_pull_t pull);

    unsigned (*get_config)(struct s3c_gpio_chip *chip, unsigned offs);
    int  (*set_config)(struct s3c_gpio_chip *chip, unsigned offs,
                    unsigned config);
};
```

通过上面的代码我们可以看到这个结构体里主要是一些函数指针，get_pull 是获取 GPIO 的上拉状态，set_pull 是设置 GPIO 上拉或下拉的，set_config 是设置 GPIO 的工作模式，例如：输出/输入/其他功能。下面我们来看看 gpio_cfg 变量的定义，如下：

```
static struct s3c_gpio_cfg gpio_cfg = {
    .set_config   = s3c_gpio_setcfg_s3c64xx_4bit,
    .set_pull  = s3c_gpio_setpull_exynos4,
    .get_pull  = s3c_gpio_getpull_exynos4,
};
```

通过上面的代码，可以看到分别对 gpio_cfg 结构的三个函数指针赋值，这三个函数的定义是在 gpio-config.c 里面实现的，这个文件在内核源码 arch/arm/plat-samsung 目录下，这三个函数的作用就是根据传进来的参数，配置 GPIO 相应的寄存器，从而实现对 GPIO 的操作。

然后我们回到 gpio-exynos4.c，接着看下面的代码，完成了 exynos4_gpio_common_4bit 中每个元素的 config 结构赋值后，接着会调用函数 samsung_gpiolib_add_4bit_chips(exynos4_gpio_common_4bit, nr_chips) 来向系统注册 GPIO 结构体。代码如下：

```
void __init samsung_gpiolib_add_4bit_chips(struct s3c_gpio_chip *chip,
                    int nr_chips)
{
    for (; nr_chips > 0; nr_chips--, chip++) {
        samsung_gpiolib_add_4bit(chip);
        s3c_gpiolib_add(chip);
    }
}
```

上面的代码主要有两个函数组成分别是 samsung_gpiolib_add_4bit(chip)和 s3c_gpiolib_add(chip)，首先我们来看下 samsung_gpiolib_add_4bit(chip)函数的实现：

```
void __init samsung_gpiolib_add_4bit(struct s3c_gpio_chip *chip)
{
    chip->chip.direction_input = samsung_gpiolib_4bit_input;
    chip->chip.direction_output = samsung_gpiolib_4bit_output;
    chip->pm = __gpio_pm(&s3c_gpio_pm_4bit);
}
```

这个函数也是为函数指针赋值，direction_input 是设置 GPIO 为输入模式，direction_output 是设置 GPIO 为输出。

s3c_gpiolib_add(chip)函数主要作用是给一些函数指针赋值，然后根据传进来的参数把对应的 GPIO 的信息保存到 gpio_desc 结构里，gpio_desc 是内核里面定义的一个全局变量，用来保存每个 GPIO 的信息。至此 GPIO 的驱动初始化就完成了，其他它主要完成的功能就是为每个 GPIO 的结构体里面的函数指针赋值，最后把每个 GPIO 结构信息保存到全局变量 gpio_desc 里面。

上面已经完成了一些通用的 GPIO 驱动的初始化，我们在回到 gpio-exynos4.c，下面是根据 CPU 的型号初始化 CPU 特定的 GPIO 了，代码如下：

```
/* Only 4210 GPIO  part */
    if (soc_is_exynos4210()) {
        chip = exynos4210_gpio_4bit;
        nr_chips = ARRAY_SIZE(exynos4210_gpio_4bit);

        for (i = 0; i < nr_chips; i++, chip++) {
            if (chip->config == NULL)
                chip->config = &gpio_cfg;
            if (chip->base == NULL)
                pr_err("No allocation of base address [4210 gpio]");
        }

        samsung_gpiolib_add_4bit_chips(exynos4210_gpio_4bit,
nr_chips);
    } else {
    /* Only 4212/4412 GPIO part */
        chip = exynos4212_gpio_4bit;
        nr_chips = ARRAY_SIZE(exynos4212_gpio_4bit);

        for (i = 0; i < nr_chips; i++, chip++) {
            if (chip->config == NULL)
                chip->config = &gpio_cfg;
            if (chip->base == NULL)
                pr_err("No allocation of base address [4212 gpio]");
        }

        samsung_gpiolib_add_4bit_chips(exynos4212_gpio_4bit,
                                                nr_chips);
    }
```

通过看上面的代码，初始化过程与前面介绍的初始化通用 GPIO 原理是一样的，这里我们不在详细介绍。对所有 GPIO 的初始化完成以后内核中的其他驱动模块就可以方便的使用我们注册到 gpio_desc 里面的 GPIO 了。内核提供了几个全局函数来操作这些 GPIO：

```
int gpio_request(unsigned gpio, const char *label)
void gpio_free(unsigned gpio)
int s3c_gpio_setpull(unsigned int pin, s3c_gpio_pull_t pull)
```

```
int s3c_gpio_cfgpin(unsigned int pin, unsigned int config)
int gpio_direction_input(unsigned gpio)
int gpio_direction_output(unsigned gpio, int value)
```

gpio_request 函数是申请 GPIO 操作，根据传递进来的参数 gpio，会去全局变量 gpio_desc 里面找到对应的 GPIO 结构，判断 desc 的标志位 flag 有没有被设置 FLAG_REQUESTED，如果有设置说明其他地方在使用这个 GPIO，程序返回-EBUSY 错误，如果没有设置就设置 flags 的标记为 FLAG_REQUESTED。

gpio_free 函数是释放 GPIO 操作，根据传递进来的参数，在 gpio_desc 全局变量找到对应的 GPIO 结构，清除掉 desc 的 flag 标志变量的 FLAG_REQUESTED 位。

s3c_gpio_setpull 函数是设置 GPIO 的上拉或下拉的，变量 pull 的取值范围如下定义：

```
#define S3C_GPIO_PULL_NONE  ((__force s3c_gpio_pull_t)0x00)
#define S3C_GPIO_PULL_DOWN  ((__force s3c_gpio_pull_t)0x01)
#define S3C_GPIO_PULL_UP ((__force s3c_gpio_pull_t)0x02)
```

S3C_GPIO_PULL_NONE 是悬空
S3C_GPIO_PULL_DOWN 是下拉
S3C_GPIO_PULL_UP 是上拉

s3c_gpio_cfgpin 函数是设置 GPIO 的功能：输入/输出/其他功能，第二个参数 config 取值范围如下：

```
    #define S3C_GPIO_INPUT  (S3C_GPIO_SPECIAL(0))
    #define S3C_GPIO_OUTPUT  (S3C_GPIO_SPECIAL(1))
    #define S3C_GPIO_SFN(x)  (S3C_GPIO_SPECIAL(x))
```

S3C_GPIO_INPUT 是输入模式，S3C_GPIO_OUTPUT 是输出模式，S3C_GPIO_SFN(x)是其他模式，例如中断模式等。

gpio_direction_input 函数设置 GPIO 是输入功能。
gpio_direction_output 设置 GPIO 输出，第二个参数 value 取值 0 或 1,0 代表输出低电平，1 代表输出高电平。

下面我们来看几个 GPIO 操作的例子：

```
if (gpio_request(EXYNOS4_GPX3(3), "MPU6050 INT"))
    printk(KERN_WARNING "MPU6050 INT(GPX3.3) Port request error!!!\n");
else{
    s3c_gpio_setpull(EXYNOS4_GPX3(3), S3C_GPIO_PULL_NONE);
    s3c_gpio_cfgpin(EXYNOS4_GPX3(3), S3C_GPIO_SFN(0));
    gpio_direction_input(EXYNOS4_GPX3(3));
    gpio_free(EXYNOS4_GPX3(3));
}
```

上面的代码是设置 GPIO 引脚 GPX3_3 为输入模式，悬空。

```
err = gpio_request_one(EXYNOS4_GPX0(0), GPIOF_IN, "mcp251x_INT");
if (err) {
    printk(KERN_ERR "failed to request mcp251x_INT\n");
    return -1;
}

s3c_gpio_cfgpin(EXYNOS4_GPX0(0), S3C_GPIO_SFN(0xf));
s3c_gpio_setpull(EXYNOS4_GPX0(0), S3C_GPIO_PULL_NONE);
gpio_free(EXYNOS4_GPX0(0));
```
上面的代码设置 GPIO 引脚 GPX0_0 为中断模式。

```
if(gpio_request(EXYNOS4_GPK1(0), "GPK1_0"))
{
    printk(KERN_ERR "failed to request GPK1_0 for "
           "USI control\n");
       return err;
}
gpio_direction_output(EXYNOS4_GPK1(0), 1);

s3c_gpio_cfgpin(EXYNOS4_GPK1(0), S3C_GPIO_OUTPUT);
gpio_free(EXYNOS4_GPK1(0));
```
上面的代码设置 GPIO 引脚 GPK1_0 为输出模式，并且输出高电平。

iTOP-4412 的 GPIO 驱动就介绍到这里，大家有兴趣的话可以去内核里面详细的查看一下整个驱动的详细实现。