一、 如何通过 IO 读寄存器确认引脚复用等问题

作者：lml

1、kernel 阶段使用 io 命令确认寄存器方法：

许多客户经常会碰到一些 IO 引脚虽然已经在 dts 中配置了对应的 iomux，但却发现实际示波器测量出来信号不对。一般这种情况出现，可首先排查 IO 引脚是否被复用。

在 RK 的 Android 平台，默认有包含 io 工具（源码位置：external\io），linux 系统平台默认没有此源码，如果想要使用，可以将 Android 平台此源码打包过去编译即可。

IO 工具使用方法：

Io 所包含的命令参数详解可以在串口或者 adb 输入：io? 回车后便可回显罗列出。

要查询 io 寄存器首先要有主控芯片详细规格书，可向我司 FAE 索取。

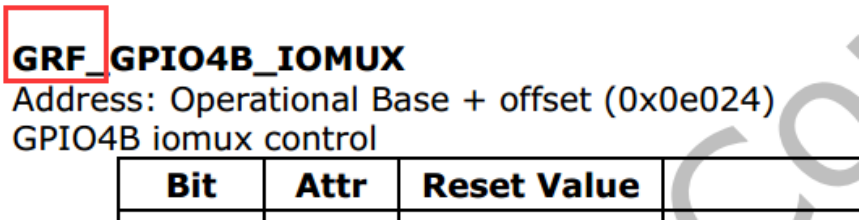
拿到规格书后，如何确认一个 IO 引脚是否被复用呢？ 以下用 RK3399 平台来举例：

例如 想查询的是 GPIO4_B0 引脚，在原理图上是：

接下来在 datasheet 上搜索 gpio4b0（不带下杠，连着输入名称），可以找到寄存器：

1:0	RW	0x0	gpio4b0_sel GPIO4B[0] iomux select 2'b00: gpio 2'b01: sdmmc_data0 2'b10: uart2dbga_sin 2'b11: reserved

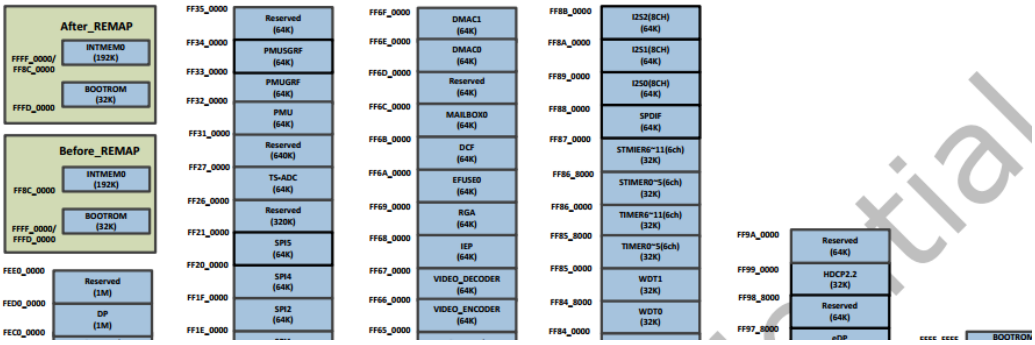
此时顺着往上翻滚可知道该寄存器的基地址名称是“GRF”：

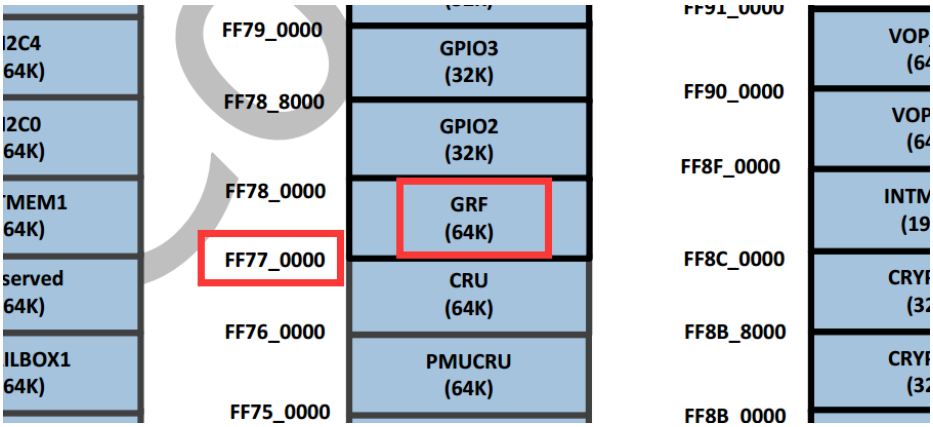


下一步就是在 datasheet 中搜索 address mapping，并在其中找到 GRF 的基地址：

2.1 Address Mapping

RK3399 supports to boot from internal bootrom, which supports remap function by softw
programming. Remap is controlled by SGRF_PMU_CON0[15]. When remap is set to 0, th
0xFFFF0000 address is mapped to bootrom. When remap is set to 1, the 0xFFFF0000 addr
is mapped to INTMEM0.





明确地址后，在串口或者 adb 输入：

io -4 -r 0xff77eo24

后回车可得到输出结果，以下举个例子:

rk3399_mid:/ # io -4 -r 0xff77e024

ff77e024: 00000555

那么输出结果就是 0x555，怎么看呢？ 因为 0x555 是十六进制，我们把它转化为二进制就是：0000 0000 0000 0000 0000 0101 0101 0101 ，这里写出完整的转化后的 32bit 方便大家观看，转换后的结果从右至左为低位到高位，即最右边的 bit 为第 0 bit，最左边的 bit 为第 31bit。那么对应到之前查询到的寄存器 ：GRF_GPIO4B_IOMUX 便可以一一对应来查看结果了。

在 GRF_GPIO4B_IOMUX 寄存器中，gpio4b3 的功能寄存器是第 7 bit 和第 6 bit[7:6] :

7:6	RW	0x0	gpio4b3_sel GPIO4B[3] iomux select 2'b00: gpio 2'b01: sdmmc_data3 2'b10: cxcsjtag_tms 2'b11: hdcpjtag_tdo

而查看上面串口输出结果转换二进制后结果来看，[7:6] bit 就是 01，也就是对应该寄存器的
01：sdmmc_data3 这个功能。由此，我们知道了 gpio4b3 这个 IO 引脚当前的复用情况，如果这不是自己设置的状态，则去代码中查找哪里被复用即可。

那么，如何查找代码中的复用？
打个比方，比如上面的 gpio4b3 这个引脚，已经在 dts 中设置了对应的 iomux，作为一个 gpio 去拉高拉低，但上面查询结果是 sdmmc，那么这时候我们首先要去 dts 中查找 sdmmc 节点先确认节点里是否有 pinctrl 里填错了 gpio，导致 mux 错了。绝大部分情况的 IO 复用问题都可以在 dts 中找到问题点。

还有一种常见问题就是，一个 IO 引脚，客户想作为 GPIO 使用，按照上面方法确认了也确实是 GPIO 的功能没错，但是示波器测量波形感觉没有按照自己驱动代码来拉高拉低，有些“不受控”。这时候很可能就是有其它驱动也在操作这个 GPIO，导致信号紊乱。应该怎么查：
很多客户喜欢去 gpio 节点下面查询，但是很多时候节点下面并不能显示具体是被什么驱动调用。这里介绍一个个人常用的方法：

思路：驱动里如果想要去操作 GPIO，肯定会调用到 gpio_direction_output、gpio_direction_input、
gpiod_direction_output、gpiod_direction_input 这几个接口，他们的定义位置是：
vim kernel/drivers/gpio/gpiolib.c
如果我们能够在这些接口里添加判断对应查询的 IO 口条件，如果满足就打印出 dump_stack(); ，那么该 IO 口的调用就一目了然了。
这里需要说明一下的是，在 linux3.10 内核的 sdk 中用的是 gpio_direction_output 和 gpio_direction_input 接口，而在 linux4.4 内核中则是 gpiod_direction_output 和 gpiod_direction_input，功能不同，详情可查看源码了解。
下面罗列出 linux3.10 版本和 linux4.4 版本添加 dump_stack 条件的方法供参考使用：
这里我们还是使用 gpio4b3 来举例。首先，需要计算出代表 gpio4b3 的值，算法如下：
Gpio4_B3 = 4 *32 + (B-A) * 8 + 3 = 3 *32 + 1 * 8 + 3 = 139
（注：最前面和 32 相乘的数字因为是 gpio4，所以是 4*32，如果是 gpio3，那就是 3*32；；括号里面的 A B C D 分别代表数值 0 1 2 3，在计算时候分别对应去减即可；最后的 +3 是因为是 GPIO4B3，如果是 GPIO4B2，那么最后就+2。）

Linux3.10

```
int gpio_direction_output(unsigned gpio, int value)
{
+       if (gpio == 139)
+       {
+               printk("dump_stack_start\n");
+               dump_stack();
+               printk("dump_stack_end\n");
+       }
```

```
        return gpiod_direction_output(gpio_to_desc(gpio), value);
}
```

Linux4.4

注：在 linux4.4 内核，io 引脚的值有些变化，也就是按照上文算法计算的结果要+1000，所以 GPIO4B3 如果是 linux4.4 内核里要填 1139。

```
int gpiod_direction_output(struct gpio_desc *desc, int value)
{
        if (!desc || !desc->chip) {
                pr_warn("%s: invalid GPIO\n", __func__);
                return -EINVAL;
        }
+         if ( desc_to_gpio(desc) == 1139)
+         {
                printk("dump_stack_start\n");
+               dump_stack();
                printk("dump_stack_end\n");
+         }
        if (test_bit(FLAG_ACTIVE_LOW, &desc->flags))
                value = !value;
        return _gpiod_direction_output_raw(desc, value);
}
```

2、Uboot 阶段寄存器状态确认方法：

如果是想要确认 uboot 阶段寄存器状态，可以使用 IO 命令或者 md 命令都可以，一般 SDK 里都有带，不过默认没打开。

1、 如果是想像 kernel 一样使用 IO 命令，只需打开以下宏即可：

```
diff --git a/include/configs/rk_default_config.h b/include/configs/rk_default_config.h
index 9852917d4f..305de26347 100755
--- a/include/configs/rk_default_config.h
+++ b/include/configs/rk_default_config.h
@@ -304,7 +304,7 @@


 /* rk io command tool */
-#undef CONFIG_RK_IO_TOOL
+#define CONFIG_RK_IO_TOOL
```

```
/* rockusb */
```

2、 如果是想使用 MD 命令来查看寄存器，可以添加以下宏：

```
diff --git a/include/configs/rk33plat.h b/include/configs/rk33plat.h
index 0906c0c091..3f94e9eabe 100755
--- a/include/configs/rk33plat.h
+++ b/include/configs/rk33plat.h
@@ -140,6 +140,7 @@
        #define CONFIG_RK_GPIO_EXT_FUNC
        #define CONFIG_CHARGE_LED
        #define CONFIG_POWER_FUSB302
+       #define CONFIG_CMD_MEMORY
 #endif

 #if defined(CONFIG_RKCHIP_RK322XH)
```

然后 make distclean 重新选择配置编译即可，此时发现 u-boot/common/cmd_mem.c 被编译。

IO 命令的话使用方法和上面介绍的 kernel 部分一样，这里不再做重复。下面就来介绍 md 命令使用方法：

在打开 MD 的宏后，在 uboot 代码：

```
diff --git a/common/autoboot.c b/common/autoboot.c
index c27cc2c751..fe28adb2fd 100644
--- a/common/autoboot.c
+++ b/common/autoboot.c
@@ -146,6 +146,7 @@ static int abortboot_normal(int bootdelay)
 {
        int abort = 0;
        unsigned long ts;
+       bootdelay = 3;      //此处添加时间可自由定义，单位：秒

 #ifdef CONFIG_MENUPROMPT
        printf(CONFIG_MENUPROMPT);
```

这样修改后编译烧录 uboot.img 在机器起来至打印 uboot 的：

Hit any key to stop autoboot:

这句会停下来倒数，倒数时间就是上面”bootdelay = 3;”设置的时间，设置多少就是几秒。只要在倒数时间内敲 PC 键盘任意按键便可让机器停在 uboot 阶段，这时候可使用 md 命令：

md 0x????????（这里后面的问号即是寄存器地址，想查询什么寄存器就填对应地址）来读取想要查看的寄存器值。

二、如何通过 IO 写寄存器

如果客户想要通过 IO 命令临时写一个寄存器做实验，可以通过 io –w 去写。

打个比方，已经通过命令： io -4 -r 0xff77e024 读出了寄存器的值，那么此时想对 0xff77e024 这个寄存器的第 0 个 bit 写入 1，那么可以如下操作：

Io -4 –w 0xff77e024 0x00010001

注：为什么寄存器地址后面的十六进制值的第 16 bit 要写 1？因为该寄存器的 16bit 至 31 bit 是写有效位，默认为 0，即不可写。因为要往第 0 bit 写 1，所以 0 bit 对应的写有效位 16 bit 也要对应置 1 才可写入，这个是根据寄存器描述而定的：

GRF_GPIO4B_IOMUX
Address: Operational Base + offset (0x0e024)
GPIO4B iomux control

Bit	Attr	Reset Value	Description
31:16	RW	0x0000	write_enable bit0~15 write enable When bit 16=1, bit 0 can be written by software . When bit 16=0, bit 0 cannot be written by software; When bit 17=1, bit 1 can be written by software . When bit 17=0, bit 1 cannot be written by software; ..... When bit 31=1, bit 15 can be written by software . When bit 31=0, bit 15 cannot be written by software;

三、如何确认 IO 电源域软硬件是否匹配

一般来说，电源域软件配置跟硬件实际不匹配，会导致该路的整排 IO 口不受控制，出来的信号紊乱或不达标。如果在实际开发过程中，遇到某一路 IO 异常，查看复用状态没错也没别处调用，但是测量出来的信号就是不对且软件无法控制拉高拉低等。这时候就要去查一下这路的电源域寄存器配置了。

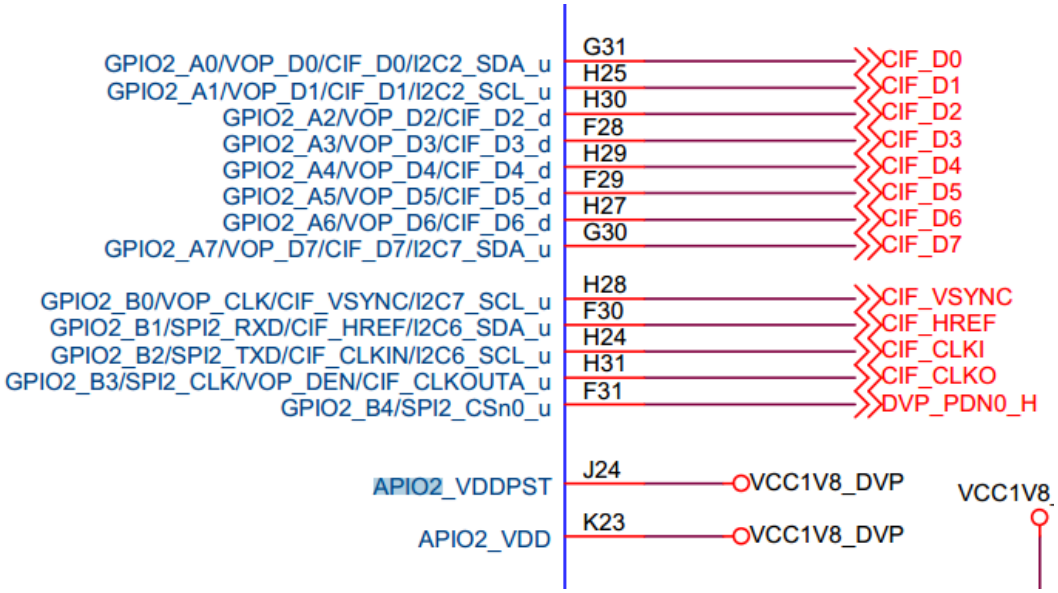
打开主控芯片对应的 TRM 芯片规格书中的 GRM/PMUGRF 章节搜索”VSEL”或者”volsel”、”vsel”这些关键字来找到 io-domain 寄存器。这里以 3399 为例：

GRF_IO_VSEL
Address: Operational Base + offset (0x0e640)

Bit	Attr	Reset Value	
			write_enable bit0~15 write

15:4	RO	0x0	reserved
3	RW	0x0	gpio1833_gpio4cd_ms
2	RW	0x0	sdmmc_gpio4b_ms
1	RW	0x0	audio_gpio3d4a_ms
0	RW	0x0	bt656_gpio2ab_ms

搜索关键字就能找到对应的寄存器。随后我们再来看原理图上引脚对应的电源，比如，我们发现 gpio2 这路的整排引脚控制不正常：



那么它们这排引脚对应的电源是 APIO2，接下来再去 sdk 代码的 kernel 目录下找到文档：kernel$ vim Documentation/devicetree/bindings/power/rockchip-io-domain.txt 这里面就有详尽的描述告知软件中是什么定义对应的 APIO2 这一路电源：

```
Possible supplies for rk3399:
- bt656-supply:  The supply connected to APIO2_VDD.
- audio-supply:  The supply connected to APIO5_VDD.
- sdmmc-supply:  The supply connected to SDMMC0_VDD.
- gpio1830       The supply connected to APIO4_VDD.

Possible supplies for rk3399 pmu-domains:
- pmu1830-supply:The supply connected to PMUIO2_VDD.
```

于是我们可以在 dts 目录下搜索“io_domain”找到：

```
&io_domains {
        status = "okay";

        bt656-supply = <&vcc1v8_dvp>;
```

```
        audio-supply = <&vcca1v8_codec>;
        sdmmc-supply = <&vcc_sd>;
        gpio1830-supply = <&vcc_3v0>;
};
```

确认软件配置里对应的项电源设置跟硬件一样即可：

bt656-supply = <&vcc1v8_dvp>;

那么此处设置的 1.8V 跟硬件原理图一致才是正确，如果不一样就改成一致，一般来说确认过电源域后可以解决绝大部分的 IO 引脚控制异常或者信号异常问题了。

注：如果以上方法还是没法解决客户的 IO 问题，请尽快联系 RK FAE 支持，谢谢！