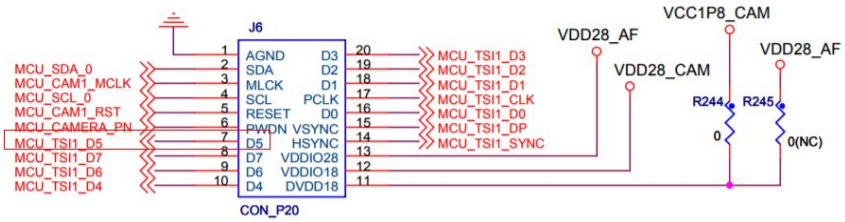
iTOP-6818-GPIO 读取配置文档

本文档主要介绍了 iTOP-6818 开发板的 gpio 引脚读取配置。有关平台文件的设备注册可以参考“平台文件设备注册”文档或者 4412 的相关视频。注册的设备名称为“readgpio7”。用户在进行本文档的操作应当提前烧写在内核中去掉“camera”驱动的内核文件。

1 配置 IO 口

我们将使用 camera 接口的第七个接口来进行 gpio 读取的操作，在底板原理图中找到该 gpio 引脚。如下图所示。



可以看到该引脚名称为“MCU_TSI1_D5”，在核心板原理图上搜索该名称可得出其对应 MCU 的 gpio 引脚编号，如下图所示。

这样便得到了我们编写驱动时所对应的 gpio 号，即“GPIOE1”。

如果不清楚 gpio 扩展口的编号方式，可以查看使用手册 1.5 章节，如下图所示。

2 设备的注册

下文介绍如何注册一个名为“readgpio7”的字符设备。对于设备注册起作用的 6818 平台文件是源码目录中的“kernel/arch/armplat-s5p6818/topeet/device.c”注意此处的 kernel 是一个软链接。用户使用喜欢的文档编辑器打开该文件，搜索“buzzer”如下图所示。

```
    .name           = DEV_NAME_PPM,
    .dev            = {
        .platform_data  = &ppm_plat_data,
        }
};
#endif
/* end add */

/* add by cym 20150921 */
#if defined(CONFIG_BUZZER_CTL)
struct platform_device buzzer_plat_device = {
        .name   = "buzzer_ctl",
        .id             = -1,
};
#endif
```

我们仿照 buzzer 的写法，用结构体的方式在平台文件中注册“readgpio7”设备，如下图所示。

```
/* add by cym 20150921 */
#if defined(CONFIG_BUZZER_CTL)
struct platform_device buzzer_plat_device = {
        .name   = "buzzer_ctl",
        .id             = -1,
};
#endif

#if defined(CONFIG_READGPIO7_CTL)
struct platform_device readgpio7_plat_device = {
        .name   = "readgpio7",
        .id             =-1,
};
#endif

#if defined(CONFIG_LEDS_CTL)
struct platform_device leds_plat_device = {
        .name   = "leds_ctl",
```

该内容仿照上下文写出即可，此时我们便在平台文件中注册了一个名称为"readgpio7"的设备。

接下来继续搜索“buzzer”，在文档的尾部再次得到搜索结果，如下图所示

```
/* add by cym 20150911 */
#if defined(CONFIG_PPM_NXP)
printk("plat: add device ppm\n");
    platform_device_register(&ppm_device);
#endif
/* end add */

/* add by cym 20150921 */
#if defined(CONFIG_BUZZER_CTL)
printk("plat: add device buzzer\n");
    platform_device_register(&buzzer_plat_device);
#endif
/* end add */
```

同样，我们仿照“buzzer”的写法，注册设备“readgpio7”，如下图所示。

```
#if defined(CONFIG_BUZZER_CTL)
printk("plat: add device buzzer\n");
    platform_device_register(&buzzer_plat_device);
#endif
/* end add */

#if defined(CONFIG_READGPIO7_CTL)
printk("plat: add device readgpio7\n");
    platform_device_register(&readgpio7_plat_device);
#endif

#if defined(CONFIG_LEDS_CTL)
printk("plat: add device leds\n");
    platform_device_register(&leds_plat_device);
#endif
```

这样，便成功的在平台文件中注册了自己的设备。

接下来要修改源码目录下的“/kernel/drivers/char”文件夹中的 Kconfig 文件，打开该文件，同样搜索“buzzer”，搜索到如下图所示的内容。

```
#add by cym 20151012
config BUZZER_CTL
        bool "Enable BUZZER config"
        default y
        help
          Enable BUZZER config
```

依照这个写法，我们写出下图红框所示的内容。

```
#add by cym 20151012
config BUZZER_CTL
        bool "Enable BUZZER config"
        default y
        help
          Enable BUZZER config

config READGPIO7_CTL
        bool "Enable READGPIO7 config"
        default y
        help
          Enable READGPIO7 config
```

这样在编译的时候，便会将设备名“readgpio7”编译进内核。此时回到“kernel”文件夹，使用命令“make menuconfig”便会在如下目录找到我们刚刚创建的设备。

```
Search Results
Symbol: READGPIO7_CTL [=y]
Type  : boolean
Prompt: Enable READGPIO7 config
  Defined at drivers/char/Kconfig:639
  Location:
    -> Device Drivers
      -> Character devices
```

3 驱动的编写

3.1 编写驱动

新建一个名为“6818_ReadGPIO_driver.c”的文档，并写入以下内容

```
#include <linux/module.h>
```

```
#include <linux/init.h>
#include <linux/kernel.h>
#include <linux/sched.h>
#include <linux/delay.h>
#include <linux/platform_device.h>
#include <asm/mach-types.h>
#include <linux/gpio.h>
#include <asm/gpio.h>
#include <asm/delay.h>
#include <linux/clk.h>
#include <mach/gpio.h>
#include <mach/soc.h>
#include <mach/platform.h>
#include <linux/miscdevice.h>
#include <linux/fs.h>
#include <asm/uaccess.h>


#define GPIO7      (PAD_GPIO_E+1)

MODULE_LICENSE("Dual BSD/GPL");
MODULE_AUTHOR("TOPEET");

static int readgpio_init(void){
    int ret;
    printk(KERN_EMERG "init_readgpio\n");

    ret=gpio_request(GPIO7,” gpioread” );
    if(ret){
        printk("request for gpio failed.\n");
        return ret;
    }
    else
        printk("GPIO request succeed.\n");

    ret=gpio_get_value(GPIO7);
    printk("GPIO7 is %d\n ",ret);
    return 0;
}
static void readgpio_exit(void){
```

```
    gpio_free(GPIO7);
    printk(KERN_EMERG "Unregiste readgpio7 \n");

}
module_init(readgpio_init);
module_exit(readgpio_exit);
```

由于篇幅限制，代码中略去了注册设备以及 fops 等的操作，在 driver_init 中便对 gpio 进行请求及读取的操作，结果是在使用“insmod”命令加载驱动时便可以看到 gpio 口的电平状态。

3.2 编写 Makefile

接下来进行编写 Makefile 文件。

```
export ARCH=arm
obj-m += 6818_gpio_simpleRead.o
KDIR := /home/topeet/6818/android5.1/lollipop-5.1.1_r6/kernel

PWD = $(shell pwd)

all:
        make -C $(KDIR) M=$(PWD) modules
clean:
        rm -rf *.o modules.order *.ko *mod.c Module.symvers
```

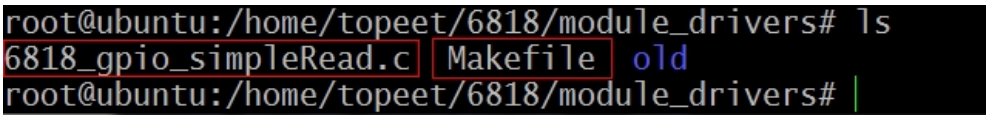
脚本中，export ARCH=arm 表示设置目标 CPU 类别为 arm，也就是编译的依赖内核和驱动模块目标 CPU 为 ARM。

obj-m += 6818_gpio_simpleRead.o 表示编译的源文件为 6818_gpio_simpleRead.c，如果源文件名有变化，则需要修改成对应的文件名。

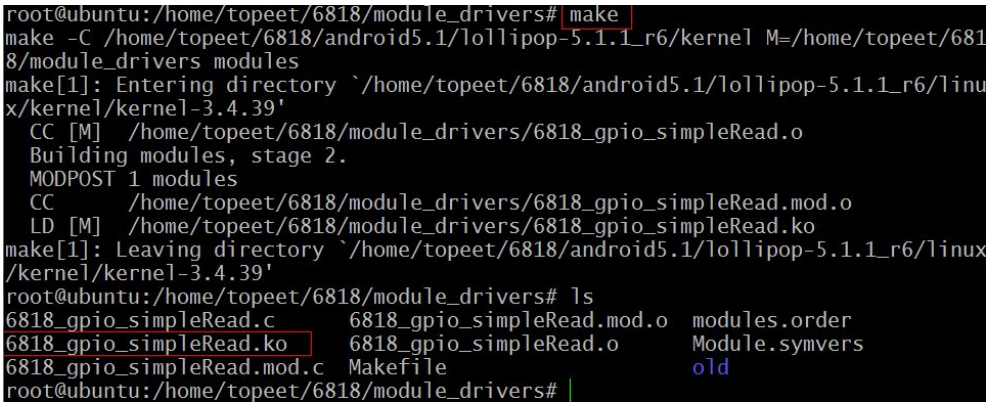
KDIR 参数指向对应的内核源码目录。作者的内核源码是在 /home/topeet/6818/android5.1/lollipop-5.1.1_r6/kernel 目录下，用户要根据自己的具体情况来修改。

3.3 编译运行

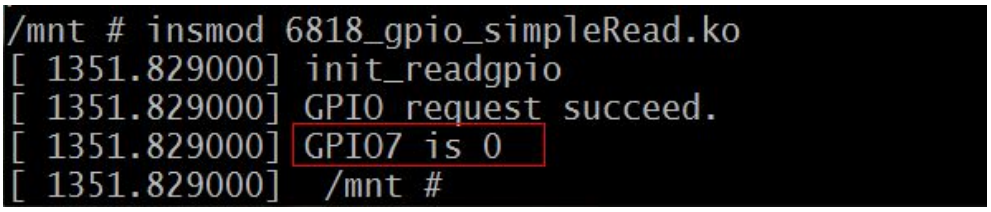
将 Makefile 与 C 程序放在 Ubuntu 系统的同一目录。如下图所示。



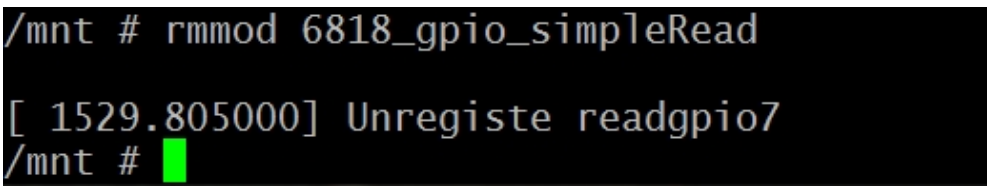
在当前目录输入“make”开始编译，生成内核模块文件“gpio_x_init.ko”，如下图所示。



将该内核模块文件拷贝到开发板，接下来在超级终端使用命令“insmod 6818_gpio_simpleRead.ko”加载该模块，如下图所示。

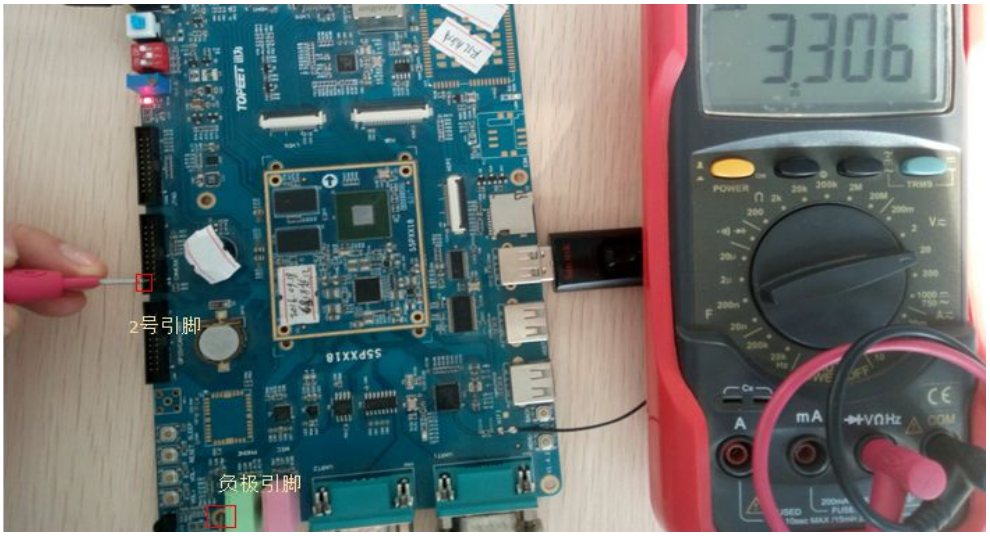


此时 camera 接口的七号引脚处于低电平状态，所以加载模块时报出该引脚电平状态为低电平状态。接下来使用命令“rmmod 6818_gpio_simpleRead”卸载该模块，如下图所示。

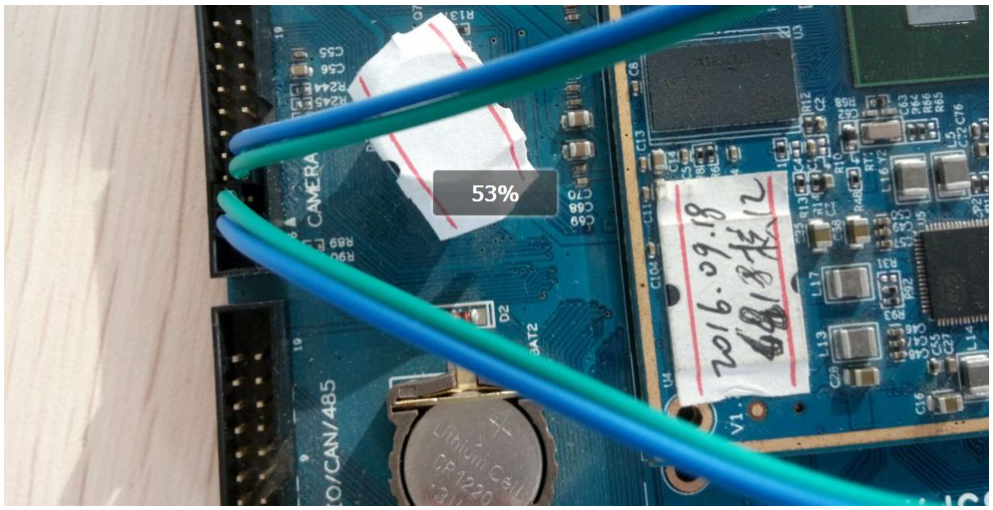


结果表示卸载成功。

接下来，笔者用万用表测量，得到 camera 接口的 2 号引脚为默认高电平约 3.3V，如下图所示。



接下来使用杜邦线连接 2 号引脚和 7 号引脚，如下图所示。



笔者只有 2pin 杜邦线，上图中起作用的只有蓝色线，它连接了位于下方的 2 号默认高电平引脚，以及上方的 7 号低电平引脚，用来拉高 7 号引脚。此时重新使用命令“insmod 6818_gpio_simpleRead.ko”，如下图所示。

```
/mnt # insmod 6818_gpio_simpleRead.ko
[  468.634000] init_readgpio
[  468.634000] GPIO request succeed.
[  468.634000] GPIO7 is 1
[  468.634000]  /mnt #
```

可以看到，camera 接口的 7 号引脚已经变成了高电平状态，说明驱动实现了预期的任务。iTOP-6818-GPIO 读取实验到此结束。

联系方式

北京迅为电子有限公司致力于嵌入式软硬件设计，是高端开发平台以及移动设备方案提供商；基于多年的技术积累，在工控、仪表、教育、医疗、车载等领域通过 OEM/ODM 方式为客户创造价值。

iTOP-6818 开发板是迅为电子基于三星最新八核处理器 Exynos6818 研制的一款实验开发平台，可以通过该产品评估 Exynos 6818 处理器相关性能，并以此为基础开发出用户需要的特定产品。

本手册主要介绍 iTOP-6818 开发板的使用方法，旨在帮助用户快速掌握该产品的应用特点，通过对开发板进行后续软硬件开发，衍生出符合特定需求的应用系统。

如需平板电脑案支持，请访问迅为平板方案网“http://www.topeet.com”，我司将有能力为您提供全方位的技术服务，保证您产品设计无忧！

本手册将持续更新，并通过多种方式发布给新老用户，希望迅为电子的努力能给您的学习和开发带来帮助。

迅为电子

2017 年 12 月