迅为电子Linux驱动教程
——GPIO的初始化

北京迅为电子有限公司

本节目标

- GPIO的初始化
- GPIO的调用过程
- 怎么自己看内核代码
  - 视频中查看代码的过程大家可以参考着来，如果有兴趣可以按照这个套路去看内核源码

    ls drivers/gpio/*.o

    # 看那些代码被编译了，可以看出gpio_exynos4编译了

这两期视频教程的目的

- 让大家能够接受或者理解下面的内容
  - 对宏定义EXYNOS4_GPL2(0)的操作就是对4412芯片管脚AC21寄存器的操作
- 上层程序员（包括驱动工程师）不用关心物理地址和虚拟地址具体是多少，只需要对一组宏定义操作，就是对4412内部寄存器操作

GPIO的初始化

- 在内核源码目录下使用命令“ls drivers/gpio/*.o”，可以看到“gpio-exynos4”被编译进了内核
  - 生成.o文件代表最终被编译进了内核
  - 除了menuconfig配置文件，还可以通过.o文件来判定该文件是否编译进了内核

- 在“gpio-exynos4.c”文件最下面一行
  - core_initcall(exynos4_gpiolib_init);
  - core_initcall代表在linux初始化过程中会调用
  - 初始化函数是在源码目录下“include/linux/init.h”文件中定义的，该头文件中定义了一系列的初始化函数，在linux启动的过程中会按等级

GPIO的初始化

- 初始化函数调用了“exynos4_gpiolib_init”
- 通过软件source insight查找到exynos4_gpiolib_init函数的定义
- 在该函数中引用了chip = exynos4_gpio_common_4bit结构体
- 查找到结构体exynos4_gpio_common_4bit
- 可以看到结构体中有S5P_VA_XXXX的基地址定义，VA一般用来代表虚拟地址。

GPIO的初始化
——结构体exynos4_gpio_common_4bit

- 以有带有label= "GPL2"的结构体为例

```
        },
    }, {
        .base   = (S5P_VA_GPIO2 + 0x100),
        .eint_offset = 0x20,
        .group  = 22,
        .chip   = {
            .base   = EXYNOS4_GPL2(0),
            .ngpio  = EXYNOS4_GPIO_L2_NR,
            .label  = "GPL2",
        },
    }, {
```

GPIO的初始化
——结构体exynos4_gpio_common_4bit

- .base = (S5P_VA_GPIO2 + 0x100)
  - 表示偏移地址和虚拟地址相加
- .eint_offset = 0x20
  - 表示中断部分，介绍中断的时候再讲（IO口可以配置为中断模式）
- .group = 22
  - 给GPIO分组
- chip.base = EXYNOS4_GPL2(0),
  - 宏定义EXYNOS4_GPL2(0)赋值给初始化函数
- chip.ngpio = EXYNOS4_GPIO_L2_NR
  - 表示这一小组中有几个GPIO
- chip.label = "GPL2",
  - 程序员需要关心的标志

GPIO的初始化
——结构体exynos4_gpio_common_4bit

- 宏定义EXYNOS4_GPL2(0)分析
  - EXYNOS4_GPL2(_nr) (EXYNOS4_GPIO_L2_START + (_nr))
  - 枚举GPIO
  - EXYNOS4_GPIO_L2_START= EXYNOS4_GPIO_NEXT(EXYNOS4_GPIO_L1)
  - EXYNOS4_GPIO_NEXT宏定义
  - #define EXYNOS4_GPIO_NEXT(__gpio) \ ((__gpio##_START) + (__gpio##_NR) + CONFIG_S3C_GPIO_SPACE + 1)
- GPIO的数量EXYNOS4_GPIO_L2_NR
  - 可以通过手册查到

GPIO的初始化
——结构体exynos4_gpio_common_4bit

- S5P_VA_GPIO2
  - 虚拟地址
- 查找S5P_VA_GPIO2宏定义，可以看到所有的GPIO被分为4个bank，这个和datasheet上面是一致的。
  - S5P_VA_GPIO1
  - S5P_VA_GPIO2　　S3C_ADDR(0x02240000)
  - S5P_VA_GPIO3
  - S5P_VA_GPIO4
- 查找到S3C_ADDR宏定义
  - #define S3C_ADDR(x)　　(S3C_ADDR_BASE + (x))
- 查找到S3C_ADDR_BASE宏定义，这是一个虚拟地址，可以看出，地址范围超出了1G或者2G内存的范围
  - #define S3C_ADDR_BASE　　0xF6000000

物理地址和虚拟地址的映射关系

- 虚拟地址和物理地址映射
  - 虚拟地址一般很好查找，一般在平台相关gpio的文件中就可以找到宏定义
- 在source insight中搜索关键字“S5P_VA_GPIO2”，看看那里用到了这个宏定义。搜索时间会比较长，1-5分钟吧。

- 搜索出来之后，可以看到除了gpio-exynos4.c文件中使用，cpu-exynos中也使用了，这是一个平台文件

物理地址和虚拟地址的映射关系

```
}, {
        .virtual        = (unsigned long)S5P_VA_GPIO2,
        .pfn            = __phys_to_pfn(EXYNOS4_PA_GPIO2),
        .length         = SZ_4K,
        .type           = MT_DEVICE,
}, {
```

物理地址和虚拟地址的映射关系

- 映射数组如下图所示

```
/* Initial IO mappings */
static struct map_desc exynos4_iodesc[] __initdata = {
    {
        .virtual    = (unsigned long)S5P_VA_SYSTIMER,
        .pfn        = __phys_to_pfn(EXYNOS4_PA_SYSTIMER),
        .length     = SZ_4K,
        .type       = MT_DEVICE,
    }, { // Robin Wang, Pls disable this section when using exynos4210 chip.
        .virtual    = (unsigned long)S5P_VA_SYSRAM,
```

物理地址和虚拟地址的映射关系

- 结构体解释
  - .virtual = (unsigned long)S5P_VA_GPIO2,表示虚拟地址
  - .pfn = __phys_to_pfn(EXYNOS4_PA_GPIO2),表示物理地址
  - .length = SZ_4K,表示映射的宽度
  - .type = MT_DEVICE,
- 查找到宏定义EXYNOS4_PA_GPIO2
  - #define EXYNOS4_PA_GPIO2 0x11000000
  - 这个物理地址0x11000000就是

GPIO的初始化流程

- 初始化过程简单描述
  - 平台文件分别定义好物理地址和虚拟地址
  - 物理地址和虚拟地址之间映射

- 在初始化中，引入了程序员需要使用的GPIO宏定义，并将宏定义装入chip结构体中

GPIO的调用函数

- 例如头文件gpio-cfg.h中s3c_gpio_cfgpin函数。这个函数是给GPIO做配置，第一个参数是宏EXYNOS4_GPL2(0)，第二个是配置的状态参数
  - 配置头文件在arm/arm/plat-samsung/include/plat/gpio-cfg.h

- 查找该函数，可以看到进入函数就会调用chip结构体
  - s3c_gpiolib_getchip，这个函数通过pin调用之后，会返回s3c_gpios[chip] 的参数
  - exynos4_gpio_common_4bit[]和s3c_gpios都是结构体s3c_gpio_chip类型的数据
  - 然后计算偏移地址等等一系列操作，这一部分是linux内核以及三星平台完成的，具体细节不用管。

GPIO的调用函数

- 也就是我们控制GPIO的时候，可以通过GPIO的一些处理函数加上类似EXYNOS4_GPL2(0)的宏定义，就可以操作GPIO

- 后面再具体介绍GPIO操作中，常用函数的使用

常见问题

- 不是说好的分页大小要一样，怎么GPIO经过mmu处理的时候，又有SZ_256又有SZ_4K？
  - 实际上CPU查找地址的时候，仍旧是通过内存。mmu本身不保存具体的数据，主要是提供一个虚拟地址和物理地址的表格，表格中还有字段的长度。这个分页和mmu没什么关系，是CPU内存以及物理地址之间通信使用的概念。这个只是一个抽象的概念，理解mmu只是一个表格，CPU对GPIO的操作就很好理解了。

常见问题

- 内部寄存器不是很快么，CPU为什么不直接读取？
  - 内部寄存器是很快，但是相对于CPU还是非常慢。CPU处理数据是将内存中一大段一大段处理，如果单个的读取内部寄存器的值，对CPU是极大的浪费。把内部寄存器也看成“特殊的物理地址”即可。

- 只讲了虚拟地址和物理地址对应数组，怎么没介绍哪里调用了？
  - 大家可以看一下函数ioremap，linux会调用这个函数来实现gpio的映射关系
  - 今天讲的已经够多够深入了，大家只要能够理解这么一层意思就可以了，这个东西对我们实际写驱动的帮助其实不是那么大！

常见问题

- 如果我还是理解不了“对宏定义EXYNOS4_GPL2(0)的操作就是对4412芯片管脚AC21寄存器的操作”，怎么办？
  - 记住这个结论，能够将宏变量EXYNOS4_GPL2(0)和GPL这一组GPIO的第0位寄存器联想起来。
  - 后面跟着我依葫芦画瓢，不影响大家实际写程序，有兴趣再回过头理解

谢谢！