

专题9-Linux驱动开发前奏

一、驱动开发概述

1.1、驱动分类

1.1.1、常规分类法

1.1.1.1、字符设备

字符设备是一种按字节来访问的设备，字符驱动则负责驱动字符设备，这样的驱动通常实现 open, close, read和 write 系统调用。例：串口，LED，按键。

1.1.1.2、块设备

在大部分的Unix系统中，块设备定义为：以块(通常是512字节)为最小传输单位的设备，块设备不能按字节处理数据。而Linux则允许块设备传送任意数目的字节。因此，块和字符设备的区别仅仅是驱动的与内核的接口不同。常见的块设备包括硬盘,flash,SD卡.....

1.1.1.3、网络设备

网络接口可以是一个硬件设备,如网卡;但也可以是一个纯粹的软件设备,比如回环接口(lo).一个网络接口负责发送和接收数据报文

1.1.2、总线分类法

USB设备，PCI设备，平台总线设备

1.2、驱动学习步骤

1.2.1、驱动程序模型

1.2.2、硬件操作实现

1.2.3、驱动程序测试

驱动学习初期，请不要过多的去阅读内核代码！！！！

二、硬件访问技术

2.1、访问流程

驱动程序控制设备，主要是通过访问设备内的寄存器来达到控制目的,因此我们讨论如何访问硬件，就成了如何访问这些寄存器了。

在Linux系统中，无论是内核程序还是应用程序，都只能使用虚拟地址，而芯片手册中给出的硬件寄存器地址或者RAM地址则是物理地址，无法直接使用，因此，我们读写寄存器的第1步就是将它的物理地址映射为虚拟地址。

2.2、地址映射

2.2.1、动态映射

所谓动态映射，是指在驱动程序中采用ioremap函数将物理地址映射为虚拟地址。

原型：`void * ioremap(physaddr, size)`

参数：

Physaddr：待映射的物理地址

Size: 映射的区域长度

返回值：映射后的虚拟地址

2.2.2、静态映射

所谓静态映射，是指Linux系统根据用户事先指定的映射关系，在内核启动时，自动地将物理地址映射为虚拟地址。

1. 如何事先指定映射关系？

在静态映射中，用户是通过map_desc结构来指明物理地址与虚拟地址的映射关系。

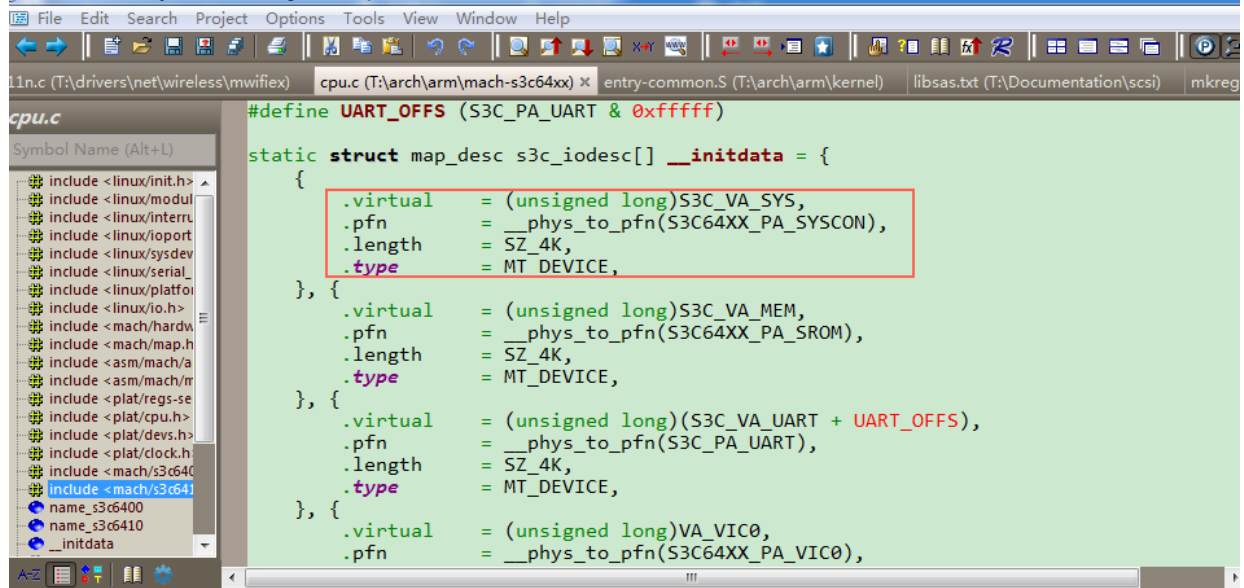
```
struct map_desc {  
    unsigned long virtual; /* 映射后的虚拟地址 */  
    unsigned long pfn; /* 物理地址所在的页帧号 */  
    unsigned long length; /* 映射长度 */  
};
```

unsigned int type; /* 映射的设备类型 */

};

pfn: 利用 __phys_to_pfn(物理地址) 可以计算出物理地址所在的物理页帧号

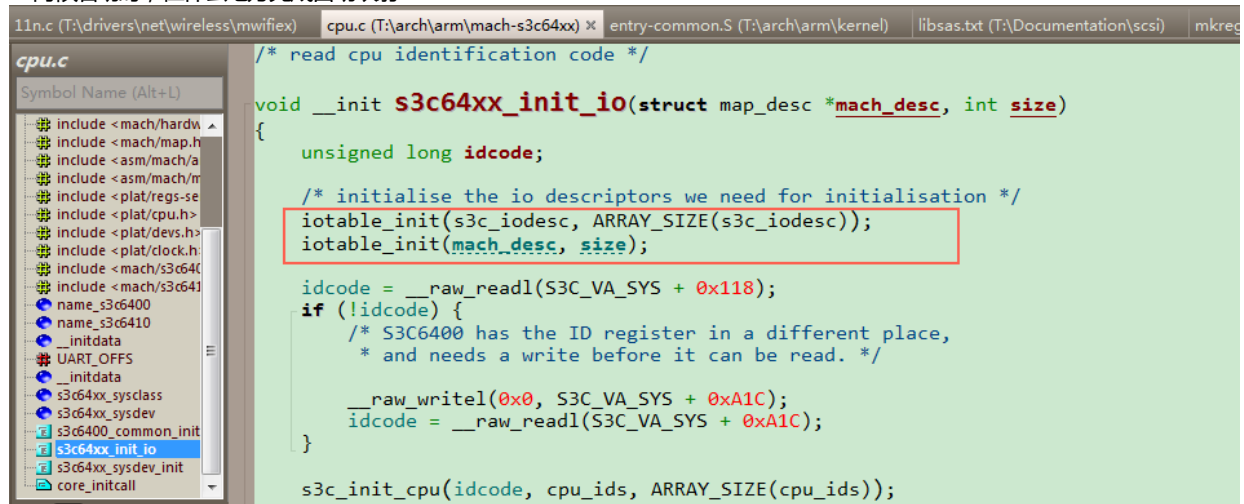
Linux-OK6410 Project - Source Insight 4.0 - [cpu.c (T:\arch\arm\mach-s3c64xx)]



```
#define UART_OFFS (S3C_PA_UART & 0xfffff)

static struct map_desc s3c_iodesc[] __initdata = {
    {
        .virtual = (unsigned long)S3C_PA_UART,
        .pfn = __phys_to_pfn(S3C64XX_PA_SYSCON),
        .length = SZ_4K,
        .type = MT_DEVICE,
    }, {
        .virtual = (unsigned long)S3C_PA_MEM,
        .pfn = __phys_to_pfn(S3C64XX_PA_SROM),
        .length = SZ_4K,
        .type = MT_DEVICE,
    }, {
        .virtual = (unsigned long)(S3C_PA_UART + UART_OFFS),
        .pfn = __phys_to_pfn(S3C_PA_UART),
        .length = SZ_4K,
        .type = MT_DEVICE,
    }, {
        .virtual = (unsigned long)VA_VIC0,
        .pfn = __phys_to_pfn(S3C64XX_PA_VIC0),
    }
```

2. 内核启动时，在什么地方完成自动映射？



```
/* read cpu identification code */

void __init s3c64xx_init_io(struct map_desc *mach_desc, int size)
{
    unsigned long idcode;

    /* initialise the io descriptors we need for initialisation */
    iotable_init(s3c_iodesc, ARRAY_SIZE(s3c_iodesc));
    iotable_init(mach_desc, size);

    idcode = __raw_readl(S3C_PA_UART + 0x118);
    if (!idcode) {
        /* S3C6400 has the ID register in a different place,
         * and needs a write before it can be read. */
        __raw_writel(0x0, S3C_PA_UART + 0xA1C);
        idcode = __raw_readl(S3C_PA_UART + 0xA1C);
    }

    s3c_init_cpu(idcode, cpu_ids, ARRAY_SIZE(cpu_ids));
}
```

2.3. 寄存器读写

在完成地址映射后，就可以读写寄存器了，Linux内核提供了一系列函数，来读写寄存器。

unsigned ioread8(void *addr)

unsigned ioread16(void *addr)

unsigned ioread32(void *addr)

unsigned readb(address)

unsigned readw(address)

unsigned readl(address)

void iowrite8(u8 value, void *addr)

void iowrite16(u16 value, void *addr)

void iowrite32(u32 value, void *addr)

void writew(unsigned value, address)

void writel(unsigned value, address)