小雷的学习空间

用硬件包围软件 最终实现软硬通吃

目录

Linux I2C驱动完全分析（一）

分类： Linux设备驱动程序第三版学习笔记 2011-04-30 16:38 246

clinuxstructalgorithm代码分析

博主按：其实老早就想写这个I2C的了，期间有各种各样的事情给耽误了。借着五一志们参考。以后会花一些时间深入研究下内核，虽然以前对内核也有所了解，但是杂，一个适配器加几个设备而已。Linux下驱动的体系结构看着挺复杂，实际也是比实际的例子，结合硬件和软件两个方面来介绍。希望能给初学的同志们一些帮助，些指点。话不多说，开整！~

本文用到的一些资源：

1. Source Insight软件
2. mini2440原理图。 下载地址http://wenku.baidu.com/view/0521ab8da0116c17
3. S3C2440 datasheet
4. AT24C08 datasheet
5. Bq27200 datasheet
6. kernel 2.6.31中的At24.c ， Bq27x00_battery.c和i2c-s3c2410.c
7. mini2440的板文件mach-mini2440.c
8. 参考资料： 《linux设备驱动开发详解（第2版）》 by 宋宝华

本文的结构：

第一部分：At24C08驱动

1. mini2440中at24c08的电气连接

- 嵌入式系统移植(0)
- 市场营销(3)
- 心情文章(5)
- 成长中的研发经理(10)
- 源代码(1)
- 电路设计(6)
- 视频技术(1)
- 项目管理(0)
- 企业管理(1)

文章存档

2015年01月(1)
2014年07月(1)
2014年05月(1)
2014年04月(1)
2014年01月(1)
2013年12月(2)
2013年11月(1)
2013年07月(1)
2013年06月(1)
2013年01月(1)
2012年10月(1)
2011年12月(1)
2011年09月(1)
2011年05月(2)
2011年04月(3)
2011年03月(9)
2011年02月(7)
2011年01月(32)
2010年12月(5)
2010年10月(3)
2010年09月(2)
2010年07月(3)
2010年05月(4)
2010年04月(6)

阅读排行

- Linux I2C驱动完全分析（一）(24612)
- Linux DM9000网卡驱动程序完全分析(24521)
- 安装tslib中遇到的错误：./autogen.sh: 4: autoreconf: not found(19067)
- Linux I2C驱动完全分析（二）(13194)

2. Linux中I2C驱动框架分析

3. I2C总线驱动代码分析

4. at24c08驱动代码分析

第二部分：Bq27200驱动

1. Bq27200的典型应用电路

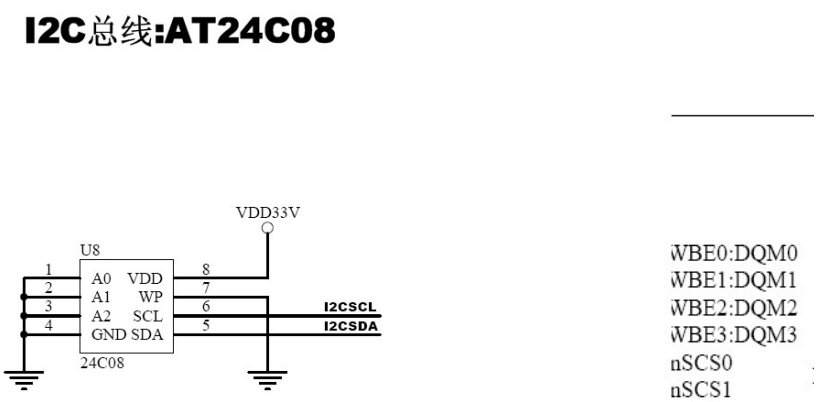
2. 主要分析一下ba27x00的代码，对比at24c08来加深理解。

----------我是分割线----------

第一部分

1. mini2440中at24c08的电气连接及其板文件

如下图。



24C08的I2C接口是与2440的IICSCL/IICSDA直接相连的。在2440内部集成了一个I
制它。先来和这四个寄存器混个脸熟吧，后面分析时还会经常用到这四个寄存器。

- Qt函数之QPainter::drawImage (6604)
- 不懂技术 如何管理好研发部门? (5182)
- 加了醋的啤酒(4402)
- 基于Qt的GPS导航系统软件源代码(4128)
- Linux驱动开发环境配置(内核源码树构造)(4078)
- v4l2驱动编写篇(3988)

评论排行

- Linux DM9000网卡驱动程序完全分析(53)
- Linux I2C驱动完全分析（二）(24)
- Linux I2C驱动完全分析（一）(10)
- 基于Qt的GPS导航系统软件源代码(10)
- 安装tslib中遇到的错误：./autogen.sh: 4: autoreconf: not found(8)
- 周立功：我的成功可以复制(5)
- v4l2驱动编写篇(4)
- 基于Qt的GPS导航系统(4)
- S3C2410看门狗驱动分析(3)
- Qt串口通信类Posix_QextserialPort中flush()函数修正(3)

推荐文章

最新评论

- Linux DM9000网卡驱动程序完全分析
gotowu: 发错了，不好意思
- Linux DM9000网卡驱动程序完全分析
gotowu: 都是内核里面的。。。
- C# winform DataGridView 操作大全
youshuai168: Thanks 这个比较有用
- 关于SetupDiEnumDeviceInfo枚举设备返回false问题的解决办法
KiteRunner1992: 您好，请问cbsize是怎么计算的，为什么32位系统就是28,64位系统就是32？多谢指教。。

— Multi-master IIC-bus control register, IICCON
— Multi-master IIC-bus control/status register, IICS
— Multi-master IIC-bus Tx/Rx data shift register, IIC
— Multi-master IIC-bus address register, IICADD

在mini2440的板文件中可以找到关于at24c08的内容，如下：

```
/*
 * I2C devices
 */
static struct at24_platform_data at24c08 = {
        .byte_len       = SZ_8K / 8,
        .page_size      = 16,
};

static struct i2c_board_info mini2440_i2c_devs[] __initdata = {
        {
                I2C_BOARD_INFO("24c08", 0x50),
                .platform_data = &at24c08,
        },
};

static void __init mini2440_init(void)
{
   ... ...
   i2c_register_board_info(0, mini2440_i2c_devs,
                                ARRAY_SIZE(mini2440_i2c_devs));
   ... ...
```

可以看出，在mini2440的init函数中注册了一个i2c的设备，这个设备我们使用了一个这个结构体定义在i2c.h文件中。如下：

```
struct i2c_board_info {
        char            type[I2C_NAME_SIZE];
        unsigned short  flags;
        unsigned short  addr;
        void            *platform_data;
        struct dev_archdata     *archdata;
        int             irq;
};
```

其中的platform_data又指向一个at24_platform_data结构体。

以上只是at24c08的部分，在板文件中还可以看到关于2440内部i2c控制器的部分，

```
static struct platform_device *mini2440_devices[] __initdata =
 ... ...
 &s3c_device_i2c0,
 ... ...
};

static void __init mini2440_init(void)
{
 ... ...
 platform_add_devices(mini2440_devices, ARRAY_SIZE(mini2440_dev
 ... ...
```

其中s2c_device_i2c0定义在arch/arm/plat-s3c/Dev-i2c0.c中（在同一目录下还可以是2440内部集成的各种设备），仔细看下面的代码再对比2440的datasheet就可以

* 控制器的IO起始地址为S3C_PA_IIC =0x54000000，大小是4K，中断号是43 =

* 控制器名是"s3c2410-i2c"

```
static struct resource s3c_i2c_resource[] = {
        [0] = {
                .start = S3C_PA_IIC,
                .end   = S3C_PA_IIC + SZ_4K - 1,
                .flags = IORESOURCE_MEM,
        },
        [1] = {
                .start = IRQ_IIC,
                .end   = IRQ_IIC,
                .flags = IORESOURCE_IRQ,
        },
};

struct platform_device s3c_device_i2c0 = {
        .name             = "s3c2410-i2c",
#ifdef CONFIG_S3C_DEV_I2C1
        .id               = 0,
#else
        .id               = -1,
#endif
        .num_resources    = ARRAY_SIZE(s3c_i2c_resource),
        .resource         = s3c_i2c_resource,
```

2. Linux中I2C驱动框架分析

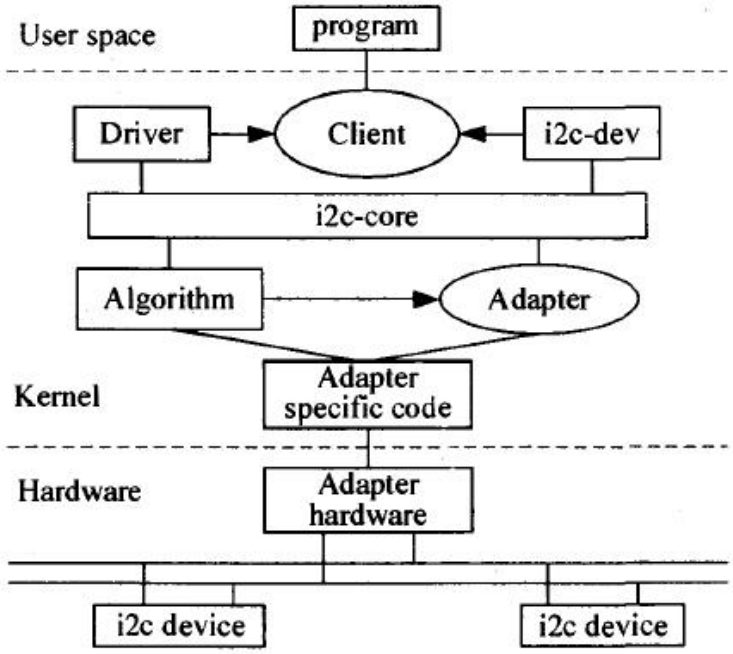
这部分是本文的重点部分。根据上面的电气连接关系我们可以看出，我们要想操动。

第一方面： 2440中I2C控制器的驱动，有了这部分驱动，我们才可以操作控制数据和接收数据。

第二方面： 24C08的驱动，有了这部分驱动，才能使用控制器正确操作芯片，

在Linux系统中，对上边第一方面的实现叫做I2C总线驱动，对第二方面的实现叫果CPU中集成了I2C控制器并且Linux内核支持这个CPU，那么总线驱动方面就不用但如果CPU中没有I2C控制器，而是外接的话，那么就要我们自己实现总线驱动了。的驱动也都包含在内核中了，如果我们用了一个内核中没有的芯片，那么就要自己

Linux中I2C体系结构如下图所示（图片来源于网络）。图中用分割线分成了三个序），内核（也就是驱动部分）和硬件（也就是实际物理设备，这里就是2440中的清晰了吧？我们现在就是要研究中间那一层。



由上图我们还可以看出哪些信息呢？

1). 可以看到几个重要的组成部分，它们是：Driver，Client，i2c-dev，i2c-core，
分在内核中都有相应的数据结构，定义在i2c.h文件中，尽量避免粘贴打断代码来凑
下每个结构体的意义。

Driver --> struct i2c_driver

这个结构体对应了驱动方法，重要成员函数有probe，remove，suspend，re

还包括一个重要的数据结构: struct i2c_device_id *id_table; 如果驱动可以支
包含这些设备的ID

Client --> struct i2c_client

应用程序是选择性失明的，它只能看到抽象的设备文件，其他部分都是看不
有联系，所以我们可以大胆得出结论：这个Client是对应于真实的物理设备，在本文
结构体中的内容应该是描述设备的。包含了芯片地址，设备名称，设备使用的中断
所依附的驱动等内容。

Algorithm -->struct i2c_algorithm

Algorithm就是算法的意思。在这个结构体中定义了一套控制器使用的通信方
()。我们实际工作中的重要一点就是要实现这个函数。

```
int (*master_xfer)(struct i2c_adapter *adap, struct i2c_msg *ms
```

Adapter --> struct i2c_adapter

这个结构体对应一个控制器。其中包含了控制器名称，algorithm数据，控制

2). 可以看出，i2c-core起到了关键的承上启下的作用。事实上也是这样，我们将从drivers/i2c/i2c-core.c中。在这个文件中可以看到几个重要的函数。

*增加/删除i2c控制器的函数

```
int i2c_add_adapter(struct i2c_adapter *adapter)
int i2c_del_adapter(struct i2c_adapter *adap)
```

*增加/删除设备驱动的函数

```
int i2c_register_driver(struct module *owner, struct i2c_driver
void i2c_del_driver(struct i2c_driver *driver)
```

*增加/删除i2c设备的函数

```
struct i2c_client *
i2c_new_device(struct i2c_adapter *adap, struct i2c_board_info
void i2c_unregister_device(struct i2c_client *client)
```

注：在2.6.30版本之前使用的是i2c_attach_client()和i2c_detach_client()函数。之后i2c_new_device中，而detach直接被unresister取代。实际上这两个函数内部都是调device_unregister()。源码如下：

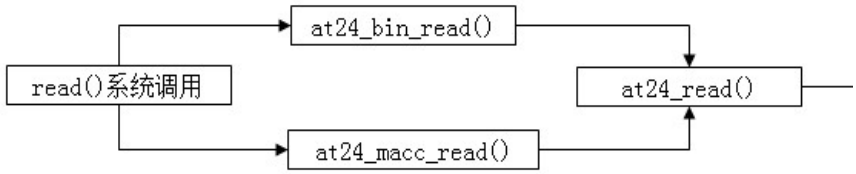
*I2C传输、发送和接收函数

```
int i2c_transfer(struct i2c_adapter *adap, struct i2c_msg *msgs

int i2c_master_send(struct i2c_client *client,const char *buf ,

int i2c_master_recv(struct i2c_client *client, char *buf ,int c
```

其中send和receive分别都调用了transfer函数，而transfer也不是直接和硬件交互，master_xfer()函数，所以我们要想进行数据传输，必须自己来实现这个master_xfer点之一。下面以read()系统调用的流程来简单梳理一下：



（待续）

上一篇 热烈庆祝博客访问量突破5000人次
下一篇 Linux I2C驱动完全分析（二）

主题推荐 linux 版权 color rgb c

猜你在找

查看评论

* 以上用户言论只代表其个人观点，不代表CSDN网站的观点或立场

核心技术类目

全部主题 Hadoop AWS 移动游戏 Java Android iOS Swift 智能硬
VPN Spark ERP IE10 Eclipse CRM JavaScript 数据库 Ubuntu
BI HTML5 Spring Apache .NET API HTML SDK IIS Fedora
Splashtop UML components Windows Mobile Rails QEMU KDE Cas
coremail OPhone CouchBase 云计算 iOS6 Rackspace Web App Spr
Compuware 大数据 aptech Perl Tornado Ruby Hibernate ThinkPHP
Angular Cloud Foundry Redis Scala Django Bootstrap