Linux I2C驱动完全分析（二）

分类： Linux设备驱动程序第三版学习笔记 2011-05-01 18:17 13196人阅读 评论(24) 收藏 举报

clinuxstructalgorithmtable

博主按：大热的天，刚刚负重从五道口走到石板房，大约4公里吧。终于让我找了一个咖啡屋休息一下，继续写这篇驱动分析。单身的生活就是这样无聊啊。 不发牢骚了，活出个样儿来给自己看！千难万险脚下踩，啥也难不倒咱！继续整！~

先说一下，本文中有个疑惑，一直没有搞懂，写在这里，望高人指点一二，不胜感激！

```
#define I2C_M_NOSTART  0x4000 /* if I2C_FUNC_PROTOCOL_MANGLING */
#define I2C_M_REV_DIR_ADDR 0x2000 /* if I2C_FUNC_PROTOCOL_MANGLING */
#define I2C_M_IGNORE_NAK 0x1000 /* if I2C_FUNC_PROTOCOL_MANGLING */
#define I2C_M_NO_RD_ACK  0x0800 /* if I2C_FUNC_PROTOCOL_MANGLING */
```

这里I2C_FUNC_PROTOCOL_MANGLING 是什么意思？为什么定义这些东东？看了注释也不太理解。求解释！

3. I2C总线驱动代码分析

s3c2440的总线驱动代码在i2c-s3c2410.c中。照例先从init看起。

```
static int __init i2c_adap_s3c_init(void)
{
        return platform_driver_register(&s3c24xx_i2c_driver);
}
```

在init中只是调用了平台驱动注册函数注册了一个i2c的平台驱动s3c24xx_i2c_driver。这个驱动是一个platform_driver的结构体变量。注意这里不是i2c_driver结构体，因为i2c_driver是对设备的驱动，而这里对控制器的驱动要使用platform_driver

```
static struct platform_driver s3c24xx_i2c_driver = {
        .probe          = s3c24xx_i2c_probe,
        .remove         = s3c24xx_i2c_remove,
        .suspend_late   = s3c24xx_i2c_suspend_late,
        .resume         = s3c24xx_i2c_resume,
        .id_table       = s3c24xx_driver_ids,
        .driver         = {
                .owner  = THIS_MODULE,
                .name   = "s3c-i2c",
        },
};
```

同样的，重要的函数还是那几个：probe，remove，suspend_late，resume。再加上一个id_table和device_driver结构体变量。

下面逐个分析：

* probe函数

当调用platform_driver_register函数注册platform_driver结构体时，probe指针指向的s3c24xx_i2c_probe函数将会被调用。这部分详细解释参考本博客另一篇文章《S3C2410看门狗驱动分析》。细心的朋友可能会发现，在s3c24xx_i2c_driver中，驱动的名字是"s3c-i2c"，而在板文件中可以看到，设备的名字是"s3c2410-i2c"，这两个名字不一样，那驱动和设备是如何match的呢？答案就在于id_table。这个id_table包含了驱动所支持的设备ID表。在match的时候，判断这个表中的名字是不是和设备一致，一致则match成功。这也是为什么一个驱动可以同时match

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)
- 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)

成功多个设备的原因。如果只是靠platform_driver-->driver中的名字来匹配的话，那么驱动和设备只能是一对一的关系了。

```
static struct platform_device_id s3c24xx_driver_ids[] = {
        {
                .name           = "s3c2410-i2c",
                .driver_data    = TYPE_S3C2410,
        }, {
                .name           = "s3c2440-i2c",
                .driver_data    = TYPE_S3C2440,
        }, { },
};
```

扯远了，还是看看probe的代码吧~

```
static int s3c24xx_i2c_probe(struct platform_device *pdev)
{
        struct s3c24xx_i2c *i2c;
        struct s3c2410_platform_i2c *pdata;
        struct resource *res;
        int ret;

        pdata = pdev->dev.platform_data;
        if (!pdata) {
                dev_err(&pdev->dev, "no platform data/n");
                return -EINVAL;
        }
        //给s3c24xx_i2c结构体申请空间
        i2c = kzalloc(sizeof(struct s3c24xx_i2c), GFP_KERNEL);
        if (!i2c) {
                dev_err(&pdev->dev, "no memory for state/n");
                return -ENOMEM;
        }
        //填充s3c24xx_i2c结构体中各项，包括名称、所有者、算法、所属class等等
        strlcpy(i2c->adap.name, "s3c2410-i2c", sizeof
(i2c->adap.name));
        i2c->adap.owner   = THIS_MODULE;
        i2c->adap.algo    = &s3c24xx_i2c_algorithm; //这个下面会重点介绍
        i2c->adap.retries = 2;
        i2c->adap.class   = I2C_CLASS_HWMON | I2C_CLASS_SPD;
        i2c->tx_setup     = 50;

        spin_lock_init(&i2c->lock);
        init_waitqueue_head(&i2c->wait);

        /* find the clock and enable it */
        // 找到i2c始终并且使能它
        i2c->dev = &pdev->dev;
        i2c->clk = clk_get(&pdev->dev, "i2c");
        if (IS_ERR(i2c->clk)) {
                dev_err(&pdev->dev, "cannot get clock/n");
                ret = -ENOENT;
                goto err_noclk;
        }

        dev_dbg(&pdev->dev, "clock source %p/n", i2c->clk);

        clk_enable(i2c->clk);

        /* map the registers */
        /*映射寄存器*/
        res = platform_get_resource(pdev, IORESOURCE_MEM, 0);
        if (res == NULL) {
                dev_err(&pdev->dev, "cannot find IO resource/n");
                ret = -ENOENT;
                goto err_clk;
        }

        i2c->ioarea = request_mem_region(res->start, resource_size
(res),
                                         pdev->name);

        if (i2c->ioarea == NULL) {
                dev_err(&pdev->dev, "cannot request IO/n");
                ret = -ENXIO;
                goto err_clk;
        }

        i2c->regs = ioremap(res->start, resource_size(res));

        if (i2c->regs == NULL) {
                dev_err(&pdev->dev, "cannot map IO/n");
                ret = -ENXIO;
                goto err_ioarea;
        }

        dev_dbg(&pdev->dev, "registers %p (%p, %p)/n",
                i2c->regs, i2c->ioarea, res);

        /* setup info block for the i2c core */

        i2c->adap.algo_data = i2c;
        i2c->adap.dev.parent = &pdev->dev;

        /* initialise the i2c controller */
        /*s3c24xx_i2c结构体变量i2c的必要的信息都填充完了以后，开始进行初始化
*/
```

*remove函数

这是和probe相反的一个函数，在i2c_adap_s3c_exit时调用。主要功能是注销适配器，释放中断，释放内存区域，禁止始终等等。看到上边代码中的err_的各个部分了吧？ remove是它们的汇总。

*suspend函数和resume函数

把这两个放一起说吧，挂起和恢复函数。挂起时保存状态并置标志位，恢复时重新初始化i2c适配器并置标志位。

Algorithm

哎呀我去，终于到这了。憋得我难受啊。这里要重点介绍一下，不仅要知其然，还要知其所以然，这样我们以后自己写驱动的时候就有把握了。

```
static const struct i2c_algorithm s3c24xx_i2c_algorithm = {
        .master_xfer            = s3c24xx_i2c_xfer,
        .functionality          = s3c24xx_i2c_func,
};
```

这里实现的就是这个s3c24xx_i2c_xfer。这个是控制器能不能动作的关键，缺了这个，控制器就是废铜烂铁。

```
static int s3c24xx_i2c_xfer(struct i2c_adapter *adap,
                            struct i2c_msg *msgs, int num)
{
        struct s3c24xx_i2c *i2c = (struct s3c24xx_i2c *)adap->algo_data;
        int retry;
        int ret;

        for (retry = 0; retry < adap->retries; retry++) {

                ret = s3c24xx_i2c_doxfer(i2c, msgs, num);

                if (ret != -EAGAIN)
                        return ret;

                dev_dbg(i2c->dev, "Retrying transmission (%d)/n", retry);

                udelay(100);
        }

        return -EREMOTEIO;
```

完成任务的函数是s3c24xx_i2c_doxfer()，源码清单如下，

```
static int s3c24xx_i2c_doxfer(struct s3c24xx_i2c *i2c,
                              struct i2c_msg *msgs, int num)
{
        unsigned long timeout;
        int ret;

        if (i2c->suspended)
                return -EIO;

        ret = s3c24xx_i2c_set_master(i2c);
        if (ret != 0) {
                dev_err(i2c->dev, "cannot get bus (error %d)/n", ret);
                ret = -EAGAIN;
                goto out;
        }

        spin_lock_irq(&i2c->lock);

        i2c->msg     = msgs;
        i2c->msg_num = num;
        i2c->msg_ptr = 0;
        i2c->msg_idx = 0;
        i2c->state   = STATE_START;

        s3c24xx_i2c_enable_irq(i2c);
        s3c24xx_i2c_message_start(i2c, msgs);
        spin_unlock_irq(&i2c->lock);

        timeout = wait_event_timeout(i2c->wait, i2c->msg_num == 0, HZ * 5);

        ret = i2c->msg_idx;

        /* having these next two as dev_err() makes life very
         * noisy when doing an i2cdetect */

        if (timeout == 0)
                dev_dbg(i2c->dev, "timeout/n");
        else if (ret != num)
                dev_dbg(i2c->dev, "incomplete xfer (%d)/n", ret);

        /* ensure the stop has been through the bus */

        msleep(1);
```

上面代码可以分成几个部分来看：

* s3c24xx_i2c_set_master() 这个函数每隔1ms查看一次i2c总线状态，timeout是400ms，如果在这期间总线状态不忙，则返回零。否则返回-ETIMEDOUT

* 将要发送的消息和其他信息付给i2c->msg和其他变量，并将状态设置为STATE_START

* s3c24xx_i2c_enable_irq() 使能中断

* s3c24xx_i2c_message_start() 重中之重啊。在看代码之前先来看看2440的datasheet上是怎么说的吧。

The following steps must be executed before any IIC Tx/Rx operations.

1. Write own slave address on IICADD register, if needed.
2. Set IICCON register.
   a) Enable interrupt
   b) Define SCL period
3. Set IICSTAT to enable Serial Output

代码清单如下：

```
static void s3c24xx_i2c_message_start(struct s3c24xx_i2c *i2c,
                                      struct i2c_msg *msg)
{
        unsigned int addr = (msg->addr & 0x7f) << 1;
        unsigned long stat;
        unsigned long iiccon;

        stat = 0;
        stat |=  S3C2410_IICSTAT_TXRXEN;

        if (msg->flags & I2C_M_RD) {//如果是read data, from slave to master
        stat |= S3C2410_IICSTAT_MASTER_RX;
                addr |= 1;
        } else
                stat |= S3C2410_IICSTAT_MASTER_TX;

        if (msg->flags & I2C_M_REV_DIR_ADDR)
                addr ^= 1;

        /* todo - check for wether ack wanted or not */
        s3c24xx_i2c_enable_ack(i2c);

        iiccon = readl(i2c->regs + S3C2410_IICCON);
        writel(stat, i2c->regs + S3C2410_IICSTAT);

        dev_dbg(i2c->dev, "START: %08lx to IICSTAT, %02x to DS/n", stat, addr);
        writeb(addr, i2c->regs + S3C2410_IICDS);

        /* delay here to ensure the data byte has gotten onto the bus
         * before the transaction is started */

        ndelay(i2c->tx_setup);

        dev_dbg(i2c->dev, "iiccon, %08lx/n", iiccon);
        writel(iiccon, i2c->regs + S3C2410_IICCON);
```

（今天没写完啊，明天继续~）

上一篇 Linux I2C驱动完全分析（一）

下一篇 基于Qt的GPS导航系统软件源代码