I2C 驱动程序框架：

内核中I2C 的处理已经做好了，我们只需要做设备驱动程序相关的内容。

总线处理好了I2C 协议，即总线知道如何收发数据，而不知道数据的含义，我们要做的只是设备相关层的代码。

一般I2C 驱动分为两层：

**总线层：**

知道设备如何读写。芯片厂家会帮我们做好。操作寄存器。drivers\i2c\busses

**设备层驱动层：**

知道数据的含义。drivers\i2c\chips

**一，分析**：\drivers\i2c\busses\I2c-s3c2410.c

1，找到probe 函数：

int \_\_init i2c\_adap\_s3c\_init(void)

platform\_driver\_register(&s3c2410\_i2c\_driver);



注册一个平台设备，当内核中有同名“s3c2440-i2c”的平台设备时，“.probe = s3c24xx\_i2c\_probe,”就被调用。

2，分析 probe 函数：

int s3c24xx\_i2c\_probe(struct platform\_device \*pdev)

-->i2c->clk = clk\_get(&pdev->dev, "i2c"); //使能I2C时钟。

-->I2C适配器结构“i2c\_adapter”：

i2c->adap.algo\_data = i2c; // /\* setup info block for the i2c core \*/

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

-->ret = s3c24xx\_i2c\_init(i2c); //硬件相关初始化。

-->request\_irq(res->start, s3c24xx\_i2c\_irq, IRQF\_DISABLED, pdev->name, i2c); //注册中断

-->i2c\_add\_adapter(&i2c->adap); //注册I2C适配器

-->i2c\_register\_adapter(adapter);

“总线设备驱动”模型：

1. I2C总线驱动程序：

如下是IIC总线相关的部分：

1. 分配i2c\_adapter结构体
2. 设置始信号、数据、响应等. 主要是algo这个核心的算法
3. 注册I 2c\_add\_adapter

插槽 （分析：linux-2.6.22.6\drivers\i2c\busses\i2c-s3c2410.c）

1，分配结构：i2c\_adapter：

int s3c24xx\_i2c\_probe(struct platform\_device \*pdev)

-->struct s3c24xx\_i2c \*i2c = &s3c24xx\_i2c;

s3c24xx\_i2c 里面有 struct i2c\_adapter adap; 成员 如下的i2c\_adapter 部分的成员：



2，设置结构i2c\_adapter：

核心是设置“i2c\_algorithm 算法结构”。

a,如何收发起始信号、数据、响应等.

b,i2c\_adapter 结构中有i2c\_algorithm 算法结构。

struct s3c24xx\_i2c \*i2c = &s3c24xx\_i2c;

-->.algo = &s3c24xx\_i2c\_algorithm, 其中的核心就是算法



1. 算法结构中“.master\_xfer”是核心。

int s3c24xx\_i2c\_xfer(struct i2c\_adapter \*adap,struct i2c\_msg \*msgs, int num)

-->s3c24xx\_i2c\_doxfer(i2c, msgs, num); “doxfer”执行传输。

②，执行传输：

int s3c24xx\_i2c\_doxfer(struct s3c24xx\_i2c \*i2c, struct i2c\_msg \*msgs, int num)

-->s3c24xx\_i2c\_enable\_irq(i2c); //使能中断

-->s3c24xx\_i2c\_message\_start(i2c, msgs); //起动传输，会产生各种中断。

-->wait\_event\_timeout(i2c->wait, i2c->msg\_num == 0, HZ \* 5); //等待事件完成。

③，起动传输：设置寄存器

寄存器：

S3C2410\_IICCON ：I2C控制寄存器

S3C2410\_IICSTAT ：I2C状态寄存器

S3C2410\_IICADD ：

S3C2410\_IICDS ：I2C DS寄存器

S3C2440\_IICLC ：

void s3c24xx\_i2c\_message\_start(struct s3c24xx\_i2c \*i2c, struct i2c\_msg \*msg)

-->iiccon = readl(i2c->regs + S3C2410\_IICCON);

-->writel(stat, i2c->regs + S3C2410\_IICSTAT);

-->writeb(addr, i2c->regs + S3C2410\_IICDS);

3，注册“i2c\_adapter”：i2c\_add\_adapter()

int s3c24xx\_i2c\_probe(struct platform\_device \*pdev)

-->i2c\_add\_adapter(&i2c->adap);

-->i2c\_register\_adapter(adapter);

int i2c\_register\_adapter(struct i2c\_adapter \*adap)

-->device\_register(&adap->dev);

1.i2c\_add\_adapter将申请的结构体放入链表。

2.调用drv的attach\_adapter函数。

3.i2c\_probe(adapter,设备地址， 函数)

最终都会调用i2c\_probe()参数有adapter 用adapter的master\_xfer发出信号 看是否有设备存在如果有的话调用第三个参数。函数

二．（分析：linux- 2.6.22.6\drivers\i2c\chips\eeprom.c）

也就是设置驱动程序我们以eeprom分析：

会注册如下的等东西：

Id //不是表示支持的设备

attach\_adapter 等……

那麽如何识别硬件设备呢？

1.i2c\_add\_driver()把i2c\_driver放入链表。

2.从adapter链表中取出适配器，调用 drv 的attach\_adapter

3.i2c\_probe(adapter,设备地址 函数)

用adapter的s3c24xx\_i2c\_xfer发信号确定是否有无设备。如有设备 调用函数

i2c\_add\_driver()//放入链表

i2c\_register\_driver()//

driver->driver.bus = &i2c\_bus\_type;

driver\_register(&driver->driver);

list\_for\_each\_entry(adapter, &adapters, list) {

driver->attach\_adapter(adapter); //对adapters链表的每一个成员 调用attach\_adapter函数

}

i2c\_probe(adapter, &addr\_data, eeprom\_detect); // attach\_adapter 下会调用i2c\_probe

i2c\_probe\_address // 发出S信号,发出设备地址(来自addr\_data)

i2c\_smbus\_xfer

i2c\_smbus\_xfer\_emulated

i2c\_transfer

adap->algo->master\_xfer // 总线的s3c24xx\_i2c\_xfer 函数

int \_\_init eeprom\_init(void)

-->i2c\_add\_driver(&eeprom\_driver);

-->i2c\_register\_driver(THIS\_MODULE, driver);



裸板程序中发现设备：

先发一个“STATT”信号，再发出7bit 的设备地址，若设备存在，则在第9 个时钟里， 此存在的设备（从机）会把“SDA”信号线拉低作为ACK 回应，这样主机就知道有相对应地址的设备存在。

int i2c\_register\_driver(struct module \*owner, struct i2c\_driver \*driver)

-->driver->driver.bus = &i2c\_bus\_type; 总线是“i2c\_bus\_type”。

-->driver\_register(&driver->driver); 注册一个“i2c\_driver”结构体。

-->list\_for\_each\_entry(adapter, &adapters, list) {driver->attach\_adapter(adapter);}

对"adapters"链表里的每一个成员，调用"i2c\_driver"结构里面的

attach\_adapter()。



此实例代码中的"i2c\_driver"结构体是“eeprom\_driver”，它其中的“attach\_adapter()”如下：



int eeprom\_attach\_adapter(struct i2c\_adapter \*adapter)

-->i2c\_probe(adapter, &addr\_data, eeprom\_detect); 其中参2就是设备地址信息。

-->i2c\_probe\_address(adapter,forces[kind][i + 1],kind, found\_proc); 发出start信号，发出设备地址。

-->i2c\_smbus\_xfer(adapter, addr, 0, 0, 0, I2C\_SMBUS\_QUICK, NULL); i2c传输

-->adapter->algo->smbus\_xfer(adapter,addr,flags,read\_write,command,size,data);

调用“i2c\_adapter” 结构里面的成员“i2c\_algorithm”结构中的“算法函数”。要是没有这个

“smbus\_xfer（）”，就使用：此代码中是用下面的代码。

-->i2c\_smbus\_xfer\_emulated(adapter,addr,flags,read\_write,command,size,data);

-->i2c\_transfer(adapter, msg, num);

-->adap->algo->master\_xfer(adap,msgs,num);

调用“i2c\_adapter” 结构里面的成员“i2c\_algorithm”结构中的“算法函数-smbus\_xfer()”。

对于“i2c\_s3c2410.c” 中，这个“smbus\_xfer()”函数即是：



USB 总线会自动识别新接入的USB 设备。但I2C 总线不能。需要：

1，发出START 信号

2，发出设备地址。

才能知道是否有此设备存在。

从“i2c\_adapter”结构的“adapter”链表取出“i2c 总线”中的一个一个驱动程序（称为“适配器”），使用里面的“.smbus\_xfer”函数发“start”信号、发设备地址（在I2C 设备驱动的i2c\_driver 结构中，成员“.id”就是表示支持哪些I2C 设备）、

怎么写I2C 设备驱动程序？

1. 分配一个i2c\_driver 结构体。

2. 设置：

i2c\_driver 结构体中有一个“attach\_adapter”和“detach\_client”函数。

attach\_adapter // 它直接调用 i2c\_probe(adap, 设备地址, 发现这个设备后要调用的函数);

detach\_client（一个设备） // 卸载这个驱动后,如果之前发现能够支持的设备, 则调用它来清理

1. 注册：i2c\_add\_driver

代码：

1，分配i2c\_driver 结构：

/\*1,分配一个 i2c\_driver 结构体\*/

/\*2，设置 i2c\_driver 结构体 \*/

static struct i2c\_driver at24cxx\_driver = { /\*直接定义\*/

.driver = {

.name = "at24cxx",

},

/\*.id = , 可能没有用 \*/

.attach\_adapter = at24cxx\_attach, /\*添加\*/

.detach\_client = at24cxx\_detach, /\*卸载\*/

};

1. 2，有了结构后，就在入口函数中注册此结构：
2. i2c\_del\_driver(at24cxx\_driver);

3，开始写“i2c\_driver at24cxx\_driver”中的成员函数：

.attach\_adapter = at24cxx\_attach, /\*添加\*/

/\*4，写".attach\_adapter"函数\*/

static int at24cxx\_attach(struct i2c\_adapter \*adater)

{

return i2c\_probe(adapter, &addr\_data, at24cxx\_detect);

}

1. 4，实现“i2c\_probe（）”函数：

/\*5,定义i2c\_probe中的&addr\_data\*/

static unsigned short ignore[] = { I2C\_CLIENT\_END };

static unsigned short normal\_addr[] = { 0x50, I2C\_CLIENT\_END}; /\* 此时设备地址是1010000即0x50 \*/

static struct i2c\_client\_address\_data addr\_data = {/\* 结构中包含三项: \*/

1. .normal\_i2c = normal\_addr, /\* 要发出地址信号才能确定是否存在此设备 \*/

.probe = ignore, /\* ignore 是省略的意思.具体作用得跟踪i2c\_probe函数 \*/

.ignore = ignore,

/\* .forces 即forces等于某个地址，就强制认为此设备存在。 \*/

};

①，跟踪 i2c\_probe()看看 "ignore"的意思：

int i2c\_probe(struct i2c\_adapter \*adapter,struct i2c\_client\_address\_data \*address\_data,

int (\*found\_proc) (struct i2c\_adapter \*, int, int))

-->if (address\_data->forces)｛｝

1. Forces 是强制认为此设备存在。当 forces等于某地址时，就强制认为此设备存在。

②，实现“i2c\_probe()”中通过地址找到设备后的处理函数：

i2c\_probe(adapter, &addr\_data, at24cxx\_detect);其中的“at24cxx\_detect（）”。

/\*6，写i2c\_probe中的处理函数:int (\*found\_proc) (struct i2c\_adapter \*, int, int)\*/

static int at24cxx\_detect (struct i2c\_adapter \*adapter, int address, int kind)

{

printk("at24cxx\_detect\n");

return 0;

}

1. 5，开始写“i2c\_driver at24cxx\_driver”中的成员函数：
2. .detach\_client = at24cxx\_detach, /\*卸载\*/
3. 开始第一次编译测试：
4. 
5. 
6. 上面这个错误是 I2C 设备地址错误。
7. 
8. 修改I2C 设备地址后，驱动检测出来了。（I2C 设备地址要是开发板和元理图上真实的地址）。
9. 修改normal\_addr 里的0x50 为0x60，这时上面的“at24cxx\_detect”并没有打印出来。
10. 
11. 另一种情况就是，当前I2C 设备还不能使用，但在程序运行过程中可能以后才能用，好想让它调用“at24cxx\_detect”函数（即i2c\_probe()函数的参3）。这时就在“struct i2c\_client\_address\_data addr\_data”地址结构中用“.forces”（ .forces 即forces 等于某个地址，就强制认为此设备存在。）

二，实现强制使用I2C 设备地址来驱动设备：

当在“入口函数”中注册了 I2C 总线设备驱动的结构体“i2c\_driver”时，结构中有两个处理函数，其一为“attach\_adapter”，在I2C 设备注册成功后，会调用这个函数来处理此设备，此函数最终会调用“i2c\_probe()”函数，其中要求了“I2C 设备”的地址。若是强制指定了这个设备地址，通过此地址设备的回应ACK 后就可以调用“.attach\_adapter = at24cxx\_attach”函数。则实现如下：

1. 强制使用“.forces”。在 i2c\_probe()中的实现过程如下：
2. 

int i2c\_probe(struct i2c\_adapter \*adapter, struct i2c\_client\_address\_data \*address\_data,

int (\*found\_proc) (struct i2c\_adapter \*, int, int))

-->if (address\_data->forces)

-->unsigned short \*\*forces = address\_data->forces;

“unsigned short \*\*forces”指向数组的指针。

-->int i2c\_probe\_address(struct i2c\_adapter \*adapter, int addr, int kind,

int (\*found\_proc) (struct i2c\_adapter \*, int, int))

-->if (kind < 0) 若kind小于0时才调用“i2c\_smbus\_xfer”。

-->i2c\_smbus\_xfer(adapter, addr, 0, 0, 0,I2C\_SMBUS\_QUICK, NULL)

1. -->err = found\_proc(adapter, addr, kind); 若 kind不小于0则直接找到设备。

static unsigned short force\_addr[] = {ANY\_I2C\_BUS, 0x60, I2C\_CLIENT\_END};

数组第一个值“ANY\_I2C\_BUS”（任何的I2C总线）

1. static unsigned short \* forces[] = {force\_addr, NULL};
2. 最后编译出错：
3. 
4. 在“i2c\_probe”中出错。

具体的错误解决的办法查看我们的代码.比较考验我们的c功底，是二维数组的处理过程

测试完成后。

加上我们字符设备驱动。

进行测试。测试ok.

编写我我们写的字符驱动的读写函数。以及测试的应用程序。

测试读写的时候直接：

./at24cxxmain r val(地址)

./ at24cxxmain w address value

在写驱动的write函数时：

//可以参考ds1337中的代码

write函数：

参考ds1337构造i2c消息：ds1337\_get\_datetime()函数

struct i2c\_msg msg[1];

从应用层得到地址和数据：

copy\_from\_user(val,buf,2);

/\* 数据传输三要素: 源,目的,长度 \*/

将写的地址发给iic设备

msg[0].addr = at24cxx\_client->addr; /\*目的\*/

msg[0].buf = val;/\*源\*/

msg[0].len = 2;

msg[0].flags = 0; /\*表示写\*/

ret = i2c\_transfer(at24cxx\_client->adapter,msg,1);//参数1适配器。参数2 ：iic消息

if (ret == 1)

return 2;

else

return -EIO;

在写驱动的read函数时：

//读比写稍微麻烦点

copy\_from\_user(&address,buf,1);//将应用层传输的要读的地址读到address

/\* 数据传输三要素: 源,目的,长度 \*/

/\* 读AT24CXX时,要先把要读的存储空间的地址发给它 \*/

msg[0].addr = at24cxx\_client->addr; /\* 目的 \*/

msg[0].buf = &address; /\* 源 \*/

msg[0].len = 1; /\* 地址=1 byte \*/

msg[0].flags = 0; /\* 表示写 \*/

/\* 然后启动读操作 \*/

msg[1].addr = at24cxx\_client->addr; /\* 源 \*/

msg[1].buf = &data; /\* 目的 \*/

msg[1].len = 1; /\* 数据=1 byte \*/

msg[1].flags = I2C\_M\_RD; /\* 表示读 \*/

ret = i2c\_transfer(at24cxx\_client->adapter,msg,2);

写iic 驱动的核心是：构造i2c\_driver结构体：

static struct i2c\_driver at24cxx\_driver = {

.driver = {

.name = "at24cxx",

},

//.id = I2C\_DRIVERID\_EEPROM,不重要

.attach\_adapter = at24cxx\_attach,

.detach\_client = at24cxx\_detach,

};

注册了i2c\_driver结构体后，从总线的adapter遍历然后调用at24cxx\_attach,

然后调用i2c\_probe(adapter,设备，function)。用adapter的传输函数看是否有这个设备

如果有的话就调用function函数.

如果不考虑是否有normal\_addr对应的设备地址。就实现forces。

static unsigned short normal\_addr[] = { 0x50, I2C\_CLIENT\_END };//这个地址只取高七位1010000 如果改为别的地址则不存在

static struct i2c\_client\_address\_data addr\_data = {

.normal\_i2c = ignore,/\*normal\_addr\*//\*要发出s信号和设备地址并得到ack信号，地址信号才能确定是否存在这个设备才能调这个函数at24cxx\_detect\*/

.probe = ignore,

.ignore = ignore,

//.forces = forces, /\*强制认为存在这个设备\*/

};

以前收发是操作寄存器现在我们用总线提供的i2c\_transfer（）来收发。

int i2c\_transfer(struct i2c\_adapter \* adap, struct i2c\_msg \*msgs, int num)

{

ret = adap->algo->master\_xfer(adap,msgs,num);//适配器的算法的传输函数

}

对应的2440就是如下的：

static const struct i2c\_algorithm s3c24xx\_i2c\_algorithm = {

.master\_xfer = s3c24xx\_i2c\_xfer,

.functionality = s3c24xx\_i2c\_func,

};

s3c24xx\_i2c\_xfer

s3c24xx\_i2c\_message\_start//启动传输