

Linux3.5下I2C设备驱动程序

2016年01月15日 11:54:48

1512

- 1

知识背景：
- 1

1. I2C协议
- 2

2. 处理器I2C接口说明
- 3

3. bus-dev-drv模型(详见文章-Linux下驱动：分层、分离机制学习笔记)
- 4

4. linux内核下驱动设计基本知识

一、驱动框架

以4412+linux3.5平台为例，说明Linux下I2C设备驱动程序。

如果一条I2C总线上只连接一个I2C设备，那么只需要以字符型设备驱动框架来写驱动即可（填充file\_opreoration结构体中的各个成员），那是典型的字符型设备驱动程序。

实际上，一款处理器可能有多条I2C总线，多个I2C控制器，每条总线上可能挂接多个设备，按照典型的字符型驱动框架写驱动就要为每个这样的设备写一个驱动程序，而且要做很多重复性的工作。

所以，有很多东西可以抽象的就抽象出来，实现一种框架，这种框架模型还是bus-dev-drv模型。4412处理器有8条可用I2C总线，每条总线上挂接的每个I2C设备都明白它们跟4412通信数据的含义。4412访问I2C设备时，都需要发出Start信号，都需要发出设备地址等等，这些共性的东西就可以抽象出来用一些函数实现，这些函数就是在另外一层即核心层驱动程序。

核心层驱动程序提供统一的I2C设备操作函数，比如读写I2C设备的操作函数；还有设备驱动层(file\_opreoration结构体相关)；另外一层就是适配层，这一层是处理器的每个I2C适配器(也叫控制器)的驱动，适配层驱动程序提供统一的处理器I2C接口硬件操作函数。

应用程序需要操作I2C设备时，根据设备节点调用读写函数(系统调用)，然后就会执行设备驱动层的相应读写函数(它们是file\_opreoration结构体的成员)，这些读写函数里会调用核心层提供的统一的I2C设备读写操作函数，这些函数最终会通过适配器(I2C控制器)给连接在I2C总线上的I2C设备发送相应读写命令。适配器那一层驱动程序就是些I2C控制器的硬件操作，它是根据I2C协议来的(这里根据的I2C协议往往是硬件已经做好了，软件只需要配置CPU的I2C控制器的某个寄存器即可实现I2C协议)，是最底层，当I2C控制器的某个寄存器被控制后，I2C控制器就会自动地往I2C线上根据I2C协议发出相应信号与I2C设备通信。

由此可见，写设备驱动程序即实现设备驱动层时，只要弄清楚bus-dev-drv驱动模型，(调用了probe函数后)剩下的就是典型的字符型设备驱动那一套了(构建file\_opreoration结构体、分配主次设备号、创建设备节点)，file\_opreoration的读写函数调用的方法也由核心层提供。图一。



AGV小车



联系我们



请扫描二维码联系客服  
webmaster@csdn.net  
400-660-0108  
QQ客服 客服论坛

关于 招聘 广告服务 百度

©1999-2018 CSDN版权所有  
京ICP证09002463号

经营性网站备案信息  
网络110报警服务  
中国互联网举报中心  
北京互联网违法和不良信息举报中心

- 他的最新文章

更多文章
- 指针数组与数组指针

Linux中select函数及实例

C中的双引号与单引号

Linux常用命令及常用法记录

生命周期，作用域的定义；说明全局变量、静态变量、局部变量、const变量的生命周期、作用域

文章分类

QT	13篇
实用工具	1篇
Linux内核	7篇
数据结构	8篇
C/C++	19篇
Linux系统使用	4篇

展开

文章存档

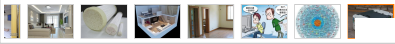
2017年12月	1篇
2017年11月	2篇
2017年10月	1篇
2017年9月	2篇
2017年3月	1篇
2016年8月	1篇

展开

- 他的热门文章



AGV小车



联系我们

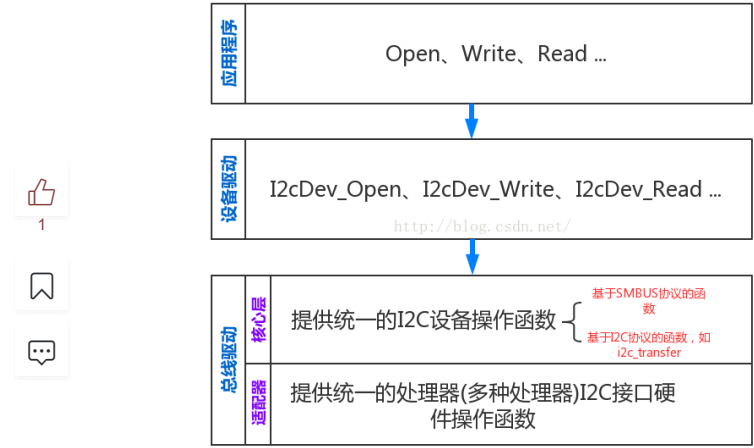


请扫描二维码联系客服  
webmaster@csdn.net  
400-660-0108  
QQ客服 客服论坛

关于 招聘 广告服务 百度

©1999-2018 CSDN版权所有  
京ICP证09002463号

经营性网站备案信息  
网络110报警服务  
中国互联网举报中心  
北京互联网违法和不良信息举报中心



上述三层是通过总线-设备-驱动模型融合到一起的。Linux内核中构建了许多总线，但并不是都是真实的，比如平台总线就是虚拟的，平台总线中也有设备链表，驱动链表。

针对I2C总线，也有一个I2C总线的结构即i2c\_bus\_type结构，此结构里面也有设备链表和也有驱动链表。设备链表里存放i2c\_client的结构体，这些结构体是注册i2c\_client时加入的，不但要加入这些结构体，还会在总线的驱动链表中一个一个地比较drv即i2c\_driver来判断是否有匹配的，如果有将调用drv里面的probe函数，匹配函数由总线提供；驱动链表里存放i2c\_driver结构体，这些结构体是注册i2c\_driver时加入的，不但要加入这些结构体，还会在总线的设备链表中一个一个地比较dev即i2c\_client来判断是否有匹配的，如果匹配将调用drv里面的probe函数。

上述的匹配函数就是i2c\_bus\_type结构里面的i2c\_device\_match函数。i2c\_device\_match函数通过id\_table(i2c\_driver结构的成员，它表示能支持哪些设备)来比较dev与drv是否匹配，具体方法是用id\_table的name去比较，name分别是i2c\_client结构和i2c\_driver结构的成员。如果名字相同，就表示此驱动i2c\_driver能支持这个设备i2c\_client。

总的说来，I2C基于bus-dev-drv模型的构建过程如下：

- (1) 左边注册一个设备，i2c\_client
- (2) 右边注册一个驱动，i2c\_driver
- (3) 比较它们的名字，如果相同，则调用driver的probe函数。
- (4) probe函数里面做的事情由用户决定（比如注册、构建设备节点等，这些属于设备驱动）。

二、设置和注册i2c\_client结构体(EEPROM为例)

在Linux内核文档（/Documentation/i2c/instantiating-devices）中，介绍了注册即构造一个i2c\_client的4种方法，并且建议使用如下前三种方法，后一种较复杂，万不得已才用。

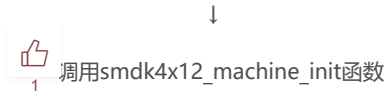
1. 通过总线号声明一个I2C设备

通过总线号声明一个I2C设备即构造一个i2c\_board\_info结构体，它里面有名字和地址，名字可以用来找到drv结构，地址可以用来访问哪个I2C设备，还可以放其他信息，这些信息会被将来的probe函数用到，也就是说在probe函数中要使用到哪些信息，这里面就可以增加哪些信息。然后使用i2c\_register\_board\_info函数注册这个结构体，此函数的第一个参数即总线号，表示此设备属于哪条总线(这就是标题通过总线号声明一个i2c设备的含义)，在此函数中会把此结构体放入链表\_i2c\_board\_list。

然后在i2c\_scan\_static\_board\_info函数中使用到\_i2c\_board\_list链表，即调用i2c\_new\_device函数把链表中的每个成员构造成一个i2c\_client，并放入bus-dev-drv模型中总线中设备链表中去。在i2c\_new\_device函数中，分配一个i2c\_client结构体后，设置它，并调用device\_register函数注册，此函数最终会调用前面文章中提到device\_add函数。i2c\_scan\_static\_board\_info函数是被i2c\_register\_adapter函数调用的，所以这里总的过程为i2c\_register\_adapter > i2c\_scan\_static\_board\_info > i2c\_new\_device。

ft5x06\_ts类型的I2C触摸屏驱动中就是使用的这种方法。在内核文件arch/arm/mach-exynos/mach-tiny4412.c中，对应触摸屏的i2c\_register\_board\_info函数调用过程如下：

```
MACHINE_START(TINY4412, "TINY4412")
    .xxx
    .init_machine = smdk4x12_machine_init,
    .xxx
MACHINE_END
```



```
i2c_register_board_info(1, smdk4x12_i2c_devs1, ARRAY_SIZE(smdk4x12_i2c_devs1));
```

可见，上述i2c\_register\_board\_info函数的第一个参数为1，表示ft5x06\_ts类型的I2C触摸屏挂接在处理器的I2C1那条总线上。

**这种方法使用限制：**在注册I2C适配器之前注册i2c\_board\_info结构体，即必须在 i2c\_register\_adapter 之前 i2c\_register\_board\_info，所以不适合动态加载insmod。

2. 直接创建设备（直接调用i2c\_new\_device、i2c\_new\_probed\_device）

(1) i2c\_new\_device方法

第一种方法显得有些麻烦，这里就直接调用 i2c\_new\_device或i2c\_new\_probed\_device函数实现。

i2c\_new\_device函数总共有两个参数，第一个为要指定的适配器i2c\_adapter(一个用来标识物理I2C总线结构，即用哪个I2C控制器发出I2C信号，某些CPU有多个I2C适配器)，即要把i2c设备跟哪个适配器相连，这样以后在访问I2C设备时，就知道使用哪个适配器的操作函数了。第二个参数是用来描述I2C设备的相关信息的，即i2c\_board\_info结构体。所以这种方法需要在dev程序中定义i2c\_adapter和i2c\_board\_info结构，i2c\_board\_info结构可以直接在dev程序中定义，i2c\_adapter创建的代码如下：

```
struct i2c_adapter *i2c_adap;
i2c_adap = i2c_get_adapter(0);
at24cxx_client = i2c_new_device(i2c_adap, &at24cxx_info);
i2c_put_adapter(i2c_adap);
```

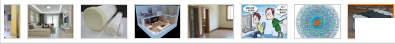
先定义一个i2c\_adapter结构指针存放i2c\_get\_adapter(0)的返回值，此函数的参数为0，表示第0条总线，i2c\_new\_device调用完后要调用 i2c\_put\_adapter函数。通过这样的方式，在第0条总线下创建了一个新设备，以后就可以使用i2c\_adap这个i2c\_adapter的操作函数发出I2C的信号了，比如起始信号、停止信号等。实验代码如下：

at24cxx\_dev.c :

```
[cpp]
1. #include <linux/kernel.h>
2. #include <linux/module.h>
3. #include <linux/platform_device.h>
4. #include <linux/i2c.h>
5. #include <linux/err.h>
6. #include <linux/regmap.h>
7. #include <linux/slab.h>
8.
9. //0x50表示I2C设备的地址，一般在 I2C设备芯片手册可以查到
10. static struct i2c_board_info at24cxx_info = {
11.     I2C_BOARD_INFO("at24c08", 0x50),//这个名字要和drv程序中的id_table中名字要一样
12. };
13.
14. static struct i2c_client *at24cxx_client;
15.
16. static int at24cxx_dev_init(void)
17. {
18.     struct i2c_adapter *i2c_adap;
19.
20.     i2c_adap = i2c_get_adapter(0);//这里要实验的EEPROM是挂接在第0条I2C总线上的，所以这里的参数是0
21.     at24cxx_client = i2c_new_device(i2c_adap, &at24cxx_info);
22.     i2c_put_adapter(i2c_adap);
23. }
```



AGV小车



联系我们



请扫描二维码联系客服  
webmaster@csdn.net  
400-660-0108  
QQ客服 客服论坛

关于 招聘 广告服务 百度  
©1999-2018 CSDN版权所有  
京ICP证09002463号

经营性网站备案信息  
网络110报警服务  
中国互联网举报中心  
北京互联网违法和不良信息举报中心

```

24.     return 0;
25. }
26.
27. static void at24cxx_dev_exit(void)
28. {
29.     i2c_unregister_device(at24cxx_client);
30. }
31.
...
module_init(at24cxx_dev_init);
module_exit(at24cxx_dev_exit);
MODULE_LICENSE("GPL");

```

α\_drv.c :

```

[cpp]
#include <linux/kernel.h>
2.  #include <linux/module.h>
3.  #include <linux/platform_device.h>
4.  #include <linux/i2c.h>
5.  #include <linux/err.h>
6.  #include <linux/regmap.h>
7.  #include <linux/slab.h>
8.
9.  static int __devinit at24cxx_probe(struct i2c_client *client,
10.                                   const struct i2c_device_id *id)
11.  {
12.      printk("%s %s %d\n", __FILE__, __FUNCTION__, __LINE__);
13.      return 0;
14.  }
15.
16.  static int __devexit at24cxx_remove(struct i2c_client *client)
17.  {
18.      printk("%s %s %d\n", __FILE__, __FUNCTION__, __LINE__);
19.      return 0;
20.  }
21.
22.  static const struct i2c_device_id at24cxx_id_table[] = {
23.      { "at24c08", 0 },//用到哪些就声明哪些内容, 比如driver_data用不到, 所以这里就写0
24.      {}
25.  };
26.
27.  /* 1. 分配/设置i2c_driver */
28.  static struct i2c_driver at24cxx_driver = {
29.      .driver = {
30.          .name = "100ask",//在这里, 这个名字并不重要, 重要的是id_table里面的名字, 所以这里可以随便起
31.          .owner = THIS_MODULE,
32.      },
33.      .probe = at24cxx_probe,
34.      .remove = __devexit_p(at24cxx_remove),
35.      .id_table = at24cxx_id_table,
36.  };
37.
38.  static int at24cxx_drv_init(void)
39.  {
40.      /* 2. 注册i2c_driver */
41.      i2c_add_driver(&at24cxx_driver);//实际使用时一定要判断返回值
42.
43.      return 0;
44.  }
45.
46.  static void at24cxx_drv_exit(void)
47.  {
48.      i2c_del_driver(&at24cxx_driver);
49.  }
50.
51.  module_init(at24cxx_drv_init);
52.  module_exit(at24cxx_drv_exit);
53.  MODULE_LICENSE("GPL");<span style="display: none; width: 0px; height: 0px;" id="transmark"></span>
an>

```

Makefile :

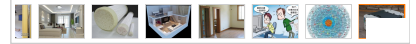
```

[cpp]
1.  KERN_DIR = /home/samba/linuxKernel_ext4Fs_src/linux-3.5-2015-8
2.
3.  all:
4.      make -C $(KERN_DIR) M=`pwd` modules
5.
6.  clean:
7.      make -C $(KERN_DIR) M=`pwd` modules clean
8.      rm -rf modules.order
9.
10. obj-m += at24cxx_dev.o

```



AGV小车



联系我们



请扫描二维码联系客服  
 ✉ webmaster@csdn.net  
 ☎ 400-660-0108  
 🗣 QQ客服 🗣 客服论坛

关于 招聘 广告服务 百度  
 ©1999-2018 CSDN版权所有  
 京ICP证09002463号

经营性网站备案信息  
 网络110报警服务  
 中国互联网举报中心  
 北京互联网违法和不良信息举报中心

```
11. | obj-m += at24cxx_drv.o
```

分别加载at24cxx\_dev.ko和at24cxx\_drv.ko, at24cxx\_probe函数会被调用, 随意修改i2c\_board\_info结构里面的地址, at24cxx\_probe函数照样被调用。所以这种方法不会去真正地验证连接的I2C设备的地址是否为i2c\_board\_info结构里面的地址。



### i2c\_new\_probed\_device方法

**i2c\_new\_device**: 认为设备肯定存在, 实验时可以用这种方法改掉I2C设备的地址, 改成任意值。  
**i2c\_new\_probed\_device**: 对于“已经识别出来的设备”(probed\_device), 才会创建(new)。i2c\_new\_probed\_device函数主要做如下三件事: probe(adap, addr\_list)  
 如果i2c\_new\_probed\_device最后个参数中没有指定probe函数, 将使用默认probe函数。  
 不管哪个, 它都要确定设备是否真实存在 \*/ info->addr = addr\_list[i]; /\* 如果在addr\_list中找到了一个地址和现实中连接在I2C总线上的设备匹配, 将这个地址放入i2c\_board\_info结构, 并传给i2c\_new\_device \*/ i2c\_new\_device(adap, info); 为了实验, 分别编译如下代码, 然后分别加载at24cxx\_dev.ko和at24cxx\_drv.ko, 如果在addr\_list数组中有一个地址是I2C总线上设备的地址, 那么在加载at24cxx\_dev.ko驱动模块时, 能加载成功, 并且加载at24cxx\_drv.ko模块后, 将调用drv的probe函数。如果没有那个地址, 那么在加载at24cxx\_dev.ko驱动模块时会失败, 提示如下信息:

```
insmod: can't insert 'at24cxx_dev.ko': No such device
```

这就是直接使用i2c\_new\_device和使用i2c\_new\_probed\_device创建i2c\_client的区别。

**at24cxx\_dev.c :**

```
[cpp]
1. #include <linux/kernel.h>
2. #include <linux/module.h>
3. #include <linux/platform_device.h>
4. #include <linux/i2c.h>
5. #include <linux/err.h>
6. #include <linux/regmap.h>
7. #include <linux/slab.h>
8.
9. static struct i2c_client *at24cxx_client;
10.
11. //如果挂在I2C总线上的i2c设备的地址在此数组里面都找不到, 将不能加载此驱动模块
12. static const unsigned short addr_list[] = { 0x60, 0x50, I2C_CLIENT_END };
13.
14. static int at24cxx_dev_init(void)
15. {
16.     struct i2c_adapter *i2c_adap;
17.     struct i2c_board_info at24cxx_info;
18.
19.     memset(&at24cxx_info, 0, sizeof(struct i2c_board_info));
20.     strncpy(at24cxx_info.type, "at24c08", I2C_NAME_SIZE);
21.
22.     i2c_adap = i2c_get_adapter(0);
23.     at24cxx_client = i2c_new_probed_device(i2c_adap, &at24cxx_info, addr_list, NULL);
24.     i2c_put_adapter(i2c_adap);
25.
26.     if (at24cxx_client)
27.         return 0;
28.     else
29.         return -ENODEV;
30. }
31.
32. static void at24cxx_dev_exit(void)
33. {
34.     i2c_unregister_device(at24cxx_client);
35. }
36.
37. module_init(at24cxx_dev_init);
38. module_exit(at24cxx_dev_exit);
39. MODULE_LICENSE("GPL");
```

**at24cxx\_drv.c :**

```
[cpp]
1. #include <linux/kernel.h>
2. #include <linux/module.h>
3. #include <linux/platform_device.h>
4. #include <linux/i2c.h>
5. #include <linux/err.h>
```



AGV小车



### 联系我们



请扫描二维码联系客服  
 webmaster@csdn.net  
 400-660-0108  
 QQ客服 客服论坛

关于 招聘 广告服务 百度  
 ©1999-2018 CSDN版权所有  
 京ICP证09002463号

经营性网站备案信息  
 网络110报警服务  
 中国互联网举报中心  
 北京互联网违法和不良信息举报中心



```

6.  #include <linux/regmap.h>
7.  #include <linux/slab.h>
8.
9.  static int __devinit at24cxx_probe(struct i2c_client *client,
10.                                   const struct i2c_device_id *id)
11.  {
12.      printk("%s %s %d\n", __FILE__, __FUNCTION__, __LINE__);
13.      return 0;
14.  }
15.
16.  static int __devexit at24cxx_remove(struct i2c_client *client)
17.  {
18.      printk("%s %s %d\n", __FILE__, __FUNCTION__, __LINE__);
19.      return 0;
20.  }
21.
22.  static const struct i2c_device_id at24cxx_id_table[] = {
23.      { "at24c08", 0 }, //用到哪些就声明哪些内容, 比如driver_data用不到, 所以这里就写0
24.      {}
25.  };
26.
27.  /* 1. 分配/设置i2c_driver */
28.  static struct i2c_driver at24cxx_driver = {
29.      .driver = {
30.          .name = "100ask", //在这里, 这个名字并不重要, 重要的是id_table里面的名字, 所以这里可以随便起
31.          .owner = THIS_MODULE,
32.      },
33.      .probe = at24cxx_probe,
34.      .remove = __devexit_p(at24cxx_remove),
35.      .id_table = at24cxx_id_table,
36.  };
37.
38.  static int at24cxx_drv_init(void)
39.  {
40.      /* 2. 注册i2c_driver */
41.      i2c_add_driver(&at24cxx_driver); //一定要判断返回值
42.
43.      return 0;
44.  }
45.
46.  static void at24cxx_drv_exit(void)
47.  {
48.      i2c_del_driver(&at24cxx_driver);
49.  }
50.
51.  module_init(at24cxx_drv_init);
52.  module_exit(at24cxx_drv_exit);
53.  MODULE_LICENSE("GPL");

```

### 3. 从用户空间创建设备(详细阅读/Documentation/i2c/instantiating-devices文档)

执行命令`cd /sys/class/i2c-adapter/`, 可以看到内容`i2c-0 i2c-1 i2c-2 i2c-3 i2c-7 i2c-8`, 说明有多款适配器, 即多个I2C控制器, 即多条I2C总线。其中EEPROM是挂接在I2C-0下面的(看板子原理图)。

< 做下面实验需要把内核中静态编译进的drv驱动给去掉, 然后加载自己的drv驱动>

#### 创建设备

`echo at24c08 0x50 > /sys/class/i2c-adapter/i2c-0/new_device`, 导致`i2c_new_device`被调用, 最后drv里的probe函数就不会被调用。如果把地址改为`0x51`, 那么也会在bus的dev链表中增加一个dev结构, 所以这种方法也是不会判断地址是否正确。

#### 删除设备

`echo 0x50 > /sys/class/i2c-adapter/i2c-0/delete_device`, 导致`i2c_unregister_device`。

### 4. 注册设置i2c\_client的第四种方法(此方法交复杂, 前三种都不行时才用)

上述三种方法都是知道I2C设备属于哪个适配器, 即知道连在了哪条I2C总线上, 如果不知道属于哪个适配器的情况下(4412有多个I2C适配器)就需要用本方法, 本方法可以参考例子`/drivers/hwmon/lm90.c`。

如果事先并不知道这个I2C设备在哪个适配器上, 怎么办? 去class表示的所有的适配器上查找。有些I2C设备的地址是一样, 怎么继续区分它是哪一款? 用detect函数。



AGV小车



#### 联系我们



请扫描二维码联系客服  
 ✉ webmaster@csdn.net  
 ☎ 400-660-0108  
 🗣 QQ客服 🗣 客服论坛





关于 招聘 广告服务 百度  
 ©1999-2018 CSDN版权所有  
 京ICP证09002463号

经营性网站备案信息  
 网络110报警服务  
 中国互联网举报中心  
 北京互联网违法和不良信息举报中心

此方法过程：

由于事先并不知道I2C设备在哪个适配器上，所以去"class表示的那一类"I2C适配器上找，用"detect函数"来确定能否找到"address\_list里的设备",如果能找到就调用i2c\_new\_device来注册i2c\_client,这会跟i2c\_driver的id\_table比较，如果匹配，调用probe。

详细代码调用过程：

- i.  id\_driver
  -  \_register\_driver
    - a. at24cxx\_driver放入i2c\_bus\_type的drv链表
    -  并且从dev链表里取出能匹配的i2c\_client并调用probe driver\_register
    -  b. 对于每一个适配器，调用\_\_process\_new\_driver（在i2c\_bus\_type的dev链表中不但要挂i2c\_client外，还会挂i2c\_adapter。当drv和dev链表比较的时候，drv不会跟i2c\_adapter比较，只会跟i2c\_client比较，因为i2c\_adapter.type成员可以用来分辨是i2c\_adapter还是i2c\_client）。

对于每一个适配器，调用它的函数确定address\_list里的设备是否存在(确定的方法是给I2C设备发一个地址，看它是否回应ACK，即SDA是否被拉低)，即是否支持这个设备。如果存在，再调用detect进一步确定、设置以确定是哪类设备，因为有些设备地址一样，单从地址是没办法分辨是哪类设备的(详细可以阅读内核文档

```

/Documentation/i2c/instantiating-devices)。然后i2c_new_device
/* Walk the adapters that are already present */
i2c_for_each_dev(driver, __process_new_driver);
__process_new_driver
i2c_do_add_adapter
/* Detect supported devices on that bus, and instantiate them */
i2c_detect(adap, driver);
for (i = 0; address_list[i] != I2C_CLIENT_END; i += 1) {
    err = i2c_detect_address(temp_client, driver);
    /* 判断这个设备是否存在：简单的发出S信号确定有ACK */
    if (!i2c_default_probe(adapter, addr))
        return 0;

    memset(&info, 0, sizeof(struct i2c_board_info));
    info.addr = addr;

    // 设置info.type，调用strncpy函数拷贝
    err = driver->detect(temp_client, &info);

    i2c_new_device(最终注册设置i2c_client)

```

at24cxx\_drv.c：

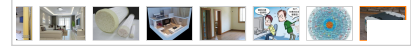
```

[cpp]
1. #include <linux/kernel.h>
2. #include <linux/module.h>
3. #include <linux/platform_device.h>
4. #include <linux/i2c.h>
5. #include <linux/err.h>
6. #include <linux/regmap.h>
7. #include <linux/slab.h>
8.
9. static int __devinit at24cxx_probe(struct i2c_client *client,
10.    const struct i2c_device_id *id)
11. {
12.     printk("%s %s %d\n", __FILE__, __FUNCTION__, __LINE__);
13.     return 0;
14. }
15.
16. static int __devexit at24cxx_remove(struct i2c_client *client)
17. {
18.     printk("%s %s %d\n", __FILE__, __FUNCTION__, __LINE__);
19.     return 0;
20. }
21.
22. static const struct i2c_device_id at24cxx_id_table[] = {
23.     { "at24c08", 0 },

```



AGV小车



联系我们



请扫描二维码联系客服  
 webmaster@csdn.net  
 400-660-0108  
 QQ客服 客服论坛

关于 招聘 广告服务 百度

©1999-2018 CSDN版权所有

京ICP证09002463号

经营性网站备案信息

网络110报警服务

中国互联网举报中心

北京互联网违法和不良信息举报中心

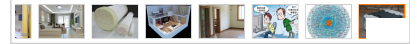
```

24.     {}
25. };
26.
27. static int at24cxx_detect(struct i2c_client *client,
28.                          struct i2c_board_info *info)
29. {
30.     /* 能运行到这里，表示该addr的设备是存在的，即dev链表中是有这个设备的
31.      * 但是有些设备单凭地址无法分辨(A芯片的地址是0x50，B芯片的地址也是0x50，当发0x50后它们都会回应，这
32.      * 时候还是不能区分到底是A还是B，A和B是不可能同时挂在一条总线上的)
33.      * 还需要进一步读写I2C设备来分辨是哪款芯片，比如读A芯片可能有一些值，读B芯片就会有另外一些值
34.      * detect就是用来进一步分辨这个芯片是哪一款，并且设置info->type
35.      */
36.
37.     printk("at24cxx_detect : addr = 0x%x\n", client->addr);
38.
39.     /* 这里应该进一步判断是哪一款，这里不用判断 */
40.
41.     strncpy(info->type, "at24c08", I2C_NAME_SIZE);
42.     return 0;
43. }
44.
45. //0x60、0x50表示I2C设备的地址
46. static const unsigned short addr_list[] = { 0x60, 0x50, I2C_CLIENT_END };
47.
48. /* 1. 分配/设置i2c_driver */
49. static struct i2c_driver at24cxx_driver = {
50.     .class = I2C_CLASS_HWMON, /* 表示去哪些适配器上找设备 */
51.     .driver = {
52.         .name = "100ask",
53.         .owner = THIS_MODULE,
54.     },
55.     .probe = at24cxx_probe,
56.     .remove = __devexit_p(at24cxx_remove),
57.     .id_table = at24cxx_id_table,
58.     .detect = at24cxx_detect, /* 用这个函数来检测设备确实存在 */
59.     .address_list = addr_list, /* 这些设备的地址 */
60. };
61.
62. static int at24cxx_drv_init(void)
63. {
64.     /* 2. 注册i2c_driver */
65.     i2c_add_driver(&at24cxx_driver);
66.
67.     return 0;
68. }
69.
70. static void at24cxx_drv_exit(void)
71. {
72.     i2c_del_driver(&at24cxx_driver);
73. }
74.
75. module_init(at24cxx_drv_init);
76. module_exit(at24cxx_drv_exit);
77. MODULE_LICENSE("GPL");

```



AGV小车



## 联系我们



请扫描二维码联系客服  
 webmaster@csdn.net  
 400-660-0108  
 QQ客服 客服论坛

关于 招聘 广告服务 百度  
 ©1999-2018 CSDN版权所有  
 京ICP证09002463号

经营性网站备案信息  
 网络110报警服务  
 中国互联网举报中心  
 北京互联网违法和不良信息举报中心

## 三、I2C设备驱动程序的编写

上面主要介绍了注册i2c\_client结构的四种方法，并伴随测试程序at24cxx\_dev.c和at24cxx\_drv.c。

然而，我们的目的是应用程序能通过系统调用读写EEPROM存储器，这就需要实现图一中的设备驱动程序。实现的地方就是在probe函数中，当i2c\_client结构和i2c\_driver结构都注册后，在i2c\_bus\_type结构的i2c\_client链表中就会有dev，i2c\_driver链表中有drv，bus的i2c\_device\_match函数中匹配dev和drv，成功将调用probe函数。在上面测试过程中probe函数基本上什么也没做，原因在于为了测试i2c\_client结构的注册，这里的目的是实现设备驱动层，所以会在里面实现注册设备驱动、创建设备节点等操作。

注意，针对i2c\_driver结构的probe成员，也就是上面说的probe函数的参数也是非常有用的。当probe函数成功调用后，它的第一个参数就记录了对应的I2C设备，也就是i2c\_client结构体，第二个参数记录对应I2C设备的i2c\_device\_id。在后面设备驱动读写函数中将调用核心层的读写函数，这些函数的第一个参数就是要知道是哪个I2C设备，即要传入i2c\_client结构体。所以，在probe函数中，可以定义结构体指针指向probe函数参数，通过这样的方式记录保存了i2c\_client和i2c\_device\_id。

这样，图一中的设备驱动层就实现了，而核心层和适配器层都是内核自带的，主要是提供接口供设备驱动使用。下面是应用层操作EEPROM将要用到的所有完整代码：



## at24cxx\_dev.c :

```
[cpp]
1. #include <linux/kernel.h>
2. #include <linux/module.h>
3. #include <linux/platform_device.h>
4. #include <linux/i2c.h>
5. #include <linux/err.h>
6. #include <linux/regmap.h>
7. #include <linux/slab.h>

//0x50表示I2C设备的地址，一般在 I2C设备芯片手册可以查到
static struct i2c_board_info at24cxx_info = {
    I2C_BOARD_INFO("at24c08", 0x50), //这个名字要和drv程序中的id_table中名字要一样
};

static struct i2c_client *at24cxx_client;

static int at24cxx_dev_init(void)
{
    struct i2c_adapter *i2c_adap;
    int busNum = 0; //把这个总线号改为1，也能成功加载此驱动，原因在于i2c_new_device而不是i2c_new_pro
    bed_device方法

    printk("at24cxx dev of bus-dev-drv module_init!\n");

    i2c_adap = i2c_get_adapter(busNum); //这里要实验的EEPROM是挂接在第0条I2C总线上的，所以这里的参数
    是0
    if (!i2c_adap) {
        pr_err("failed to get adapter i2c%d\n", busNum);
        return -ENODEV;
    }

    at24cxx_client = i2c_new_device(i2c_adap, &at24cxx_info); //设置和注册i2c_client结构体
    if (!at24cxx_client) {
        //pr_err("failed to register %s to i2c%d\n", at24cxx_info.type, busNum);
        pr_err("failed to register at24c08 to i2c%d\n", busNum);
        return -ENODEV;
    }

    i2c_put_adapter(i2c_adap);

    return 0;
}

static void at24cxx_dev_exit(void)
{
    printk("at24cxx dev of bus-dev-drv module_exit!\n");
    i2c_unregister_device(at24cxx_client);
}

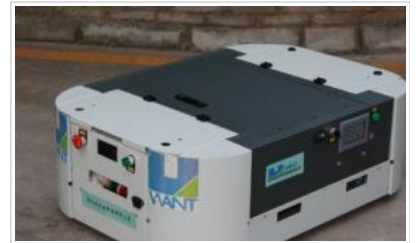
module_init(at24cxx_dev_init);
module_exit(at24cxx_dev_exit);
MODULE_LICENSE("GPL");
```

## at24cxx\_drv.c :

```
[cpp]
1. #include <linux/kernel.h>
2. #include <linux/module.h>
3. #include <linux/platform_device.h>
4. #include <linux/i2c.h>
5. #include <linux/err.h>
6. #include <linux/regmap.h>
7. #include <linux/slab.h>
8. #include <linux/fs.h>
9. #include <asm/uaccess.h>

static int major;
static struct class *class;
static struct i2c_client *at24cxx_client;

/* 传入: buf[0] : addr, 即将访问I2C设备的地址
 * 输出: buf[0] : data
 */
static ssize_t at24cxx_read(struct file * file, char __user *buf, size_t count, loff_t *of
f)
{
    unsigned char addr, data;
```



AGV小车



## 联系我们



请扫描二维码联系客服  
 ✉ webmaster@csdn.net  
 ☎ 400-660-0108  
 🗣 QQ客服 🗣 客服论坛

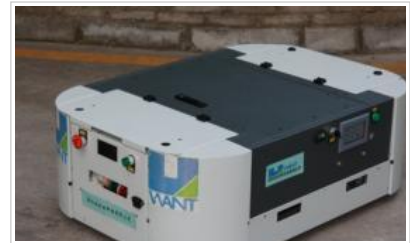
关于 招聘 广告服务 百度  
 ©1999-2018 CSDN版权所有  
 京ICP证09002463号

经营性网站备案信息  
 网络110报警服务  
 中国互联网举报中心  
 北京互联网违法和不良信息举报中心

```

22.     if (copy_from_user(&addr, buf, 1)){
23.         printk("at24cxx_read: copy_from_user error\n");
24.         return -EFAULT;
25.     }
26. /*
27.     根据EEPROM读时序, 在smbus-protocol文档中找到i2c_smbus_read_byte_data
28.     函数。
29. */
30. data = i2c_smbus_read_byte_data(at24cxx_client, addr);
31. if (data < 0) {
32.     printk("at24cxx_read: i2c_smbus_read_byte_data error\n");
33.     return data;
34. }
35.
36. if (copy_to_user(buf, &data, 1)){
37.     printk("at24cxx_read: copy_to_user error\n");
38.     return -EFAULT;
39. }
40.
41. return 1;
42. }
43.
44. /* buf[0] : addr, 即将访问I2C设备的地址
45.    * buf[1] : data
46.    */
47. static ssize_t at24cxx_write(struct file *file, const char __user *buf, size_t count, loff_t *off)
48. {
49.     unsigned char ker_buf[2];
50.     unsigned char addr, data;
51.
52.     if (copy_from_user(ker_buf, buf, 2)){
53.         printk("at24cxx_write: copy_from_user error\n");
54.         return -EFAULT;
55.     }
56.
57.     addr = ker_buf[0];
58.     data = ker_buf[1];
59.
60.     printk("addr = 0x%02x, data = 0x%02x\n", addr, data);
61.
62.     /*
63.         读写操作函数由核心层提供, 有两种方式:
64.         1. SMBUS协议, 系统管理总线, 内核文档建议用个方式, 原因详见文档<smbus-protocol>
65.         2. I2C_transfer
66.         根据smbus-protocol文档, i2c_smbus_write_byte_data函数的时序和
67.         EEPROM的写操作时序完全一样, 所以用它。第一个参数at24cxx_client在
68.         probe函数中已经做了记录。
69.     */
70.     if (!i2c_smbus_write_byte_data(at24cxx_client, addr, data)){
71.         return 2;//如果写成功, 返回写成功字节
72.     }
73.     else
74.     {
75.         printk("at24cxx_write: i2c_smbus_write_byte_data error\n");
76.         return -EIO;
77.     }
78. }
79.
80. static struct file_operations at24cxx_fops = {
81.     .owner = THIS_MODULE,
82.     .read = at24cxx_read,
83.     .write = at24cxx_write,
84. };
85.
86. static int __devinit at24cxx_probe(struct i2c_client *client,
87.     const struct i2c_device_id *id)
88. {
89.     struct device *class_dev = NULL;
90.
91.     at24cxx_client = client;//记录i2c_client结构, 调用SMBUS方法的时候方便使用
92.
93.     //printk("%s %s %d\n", __FILE__, __FUNCTION__, __LINE__);
94.     major = register_chrdev(0, "at24cxx", &at24cxx_fops);
95.     if (major < 0) {
96.         printk("at24cxx_probe: can't register at24cxx char device\n");
97.         return major;
98.     }
99.
100.     class = class_create(THIS_MODULE, "at24cxx");//创建类
101.     if (IS_ERR(class)) {
102.         printk("at24cxx_probe: class create failed\n");
103.         unregister_chrdev(major, "at24cxx");
104.         return PTR_ERR(class);
105.     }
106.
107.     /* 在类下面创建设备, 自动创建设备节点。device_create会调用那个mdev,

```



AGV小车



## 联系我们



请扫描二维码联系客服

✉ webmaster@csdn.net

☎ 400-660-0108

👤 QQ客服 🗨 客服论坛

关于 招聘 广告服务 百度

©1999-2018 CSDN版权所有

京ICP证09002463号

经营性网站备案信息

网络110报警服务

中国互联网举报中心

北京互联网违法和不良信息举报中心

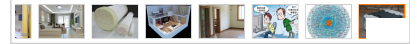
```

108.     mdev会根据环境变量创建设备节点*/
109.     /* /dev/at24cxx */
110.     class_dev = device_create(class, NULL, MKDEV(major, 0), NULL, "at24cxx");
111.     if (IS_ERR(class_dev)) {
112.         printk("at24cxx_probe: class device create failed\n");
113.         unregister_chrdev(major, "at24cxx");
114.         return PTR_ERR(class_dev);
115.     }
116.     return 0;
117. }
118.
119. static int __devexit at24cxx_remove(struct i2c_client *client)
120. {
121.     printk("at24cxx drv of bus-dev-drv at24cxx_remove!\n");
122.     device_destroy(class, MKDEV(major, 0)); //删除设备节点
123.     class_destroy(class); //摧毁类
124.     unregister_chrdev(major, "at24cxx");
125.     return 0;
126. }
127.
128. static const struct i2c_device_id at24cxx_id_table[] = {
129.     { "at24c08", 0 }, //用到哪些就声明哪些内容, 比如driver_data用不到, 所以这里就写0
130.     {}
131. };
132.
133. /* 1. 分配/设置i2c_driver */
134. static struct i2c_driver at24cxx_driver = {
135.     .driver = {
136.         .name = "100ask", //在这里, 这个名字并不重要, 重要的是id_table里面的名字, 所以这里可以随便起
137.         .owner = THIS_MODULE,
138.     },
139.     .probe = at24cxx_probe,
140.     .remove = __devexit_p(at24cxx_remove),
141.     .id_table = at24cxx_id_table,
142. };
143.
144. static int at24cxx_drv_init(void)
145. {
146.     int ret;
147.
148.     printk("at24cxx drv of bus-dev-drv module_init!\n");
149.
150.     /* 2. 注册i2c_driver */
151.     ret = i2c_add_driver(&at24cxx_driver);
152.     if (ret != 0) {
153.         pr_err("Failed to register at24cxx I2C driver: %d\n", ret);
154.     }
155.     return 0;
156. }
157.
158. static void at24cxx_drv_exit(void)
159. {
160.     printk("at24cxx drv of bus-dev-drv module_exit!\n");
161.     i2c_del_driver(&at24cxx_driver);
162. }
163.
164. module_init(at24cxx_drv_init);
165. module_exit(at24cxx_drv_exit);
166. MODULE_LICENSE("GPL");

```



AGV小车



## 联系我们



请扫描二维码联系客服

✉ webmaster@csdn.net

☎ 400-660-0108

👤 QQ客服 🗣 客服论坛

关于 招聘 广告服务 百度

©1999-2018 CSDN版权所有

京ICP证09002463号

经营性网站备案信息

网络110报警服务

中国互联网举报中心

北京互联网违法和不良信息举报中心

## 4th\_device\_driver\_i2c\_test.c :

```

[cpp]
1. #include <stdio.h>
2. #include <stdlib.h>
3. #include <string.h>
4. #include <sys/types.h>
5. #include <sys/stat.h>
6. #include <fcntl.h>
7.
8. /* i2c_test r addr
9.  * i2c_test w addr val
10.  */
11.
12. void print_usage(char *file)
13. {
14.     printf("%s r addr\n", file);
15.     printf("%s w addr val\n", file);
16. }
17.

```

```

18. int main(int argc, char **argv)
19. {
20.     int fd;
21.     unsigned char buf[2];
22.
23.     if ((argc != 3) && (argc != 4))
24.     {
25.         print_usage(argv[0]);
26.         return -1;
27.     }
28.
29.     fd = open("/dev/at24cxx", O_RDWR);
30.     if (fd < 0)
31.     {
32.         printf("can't open /dev/at24cxx\n");
33.         return -1;
34.     }
35.
36.     if (strcmp(argv[1], "r") == 0)
37.     {
38.         buf[0] = strtoul(argv[2], NULL, 0);
39.         read(fd, buf, 1);
40.         printf("data: %c, %d, 0x%2x\n", buf[0], buf[0], buf[0]);
41.     }
42.     else if ((strcmp(argv[1], "w") == 0) && (argc == 4))
43.     {
44.         buf[0] = strtoul(argv[2], NULL, 0);
45.         buf[1] = strtoul(argv[3], NULL, 0);
46.         if (write(fd, buf, 2) != 2)
47.             printf("write err, addr = 0x%02x, data = 0x%02x\n", buf[0], buf[1]);
48.     }
49.     else
50.     {
51.         print_usage(argv[0]);
52.         return -1;
53.     }
54.
55.     return 0;
56. }

```

#### 四、不自己写驱动直接访问

在图一中，核心层已经提供了统一的I2C设备操作函数：

- (1) SMBUS协议方式(SMBUS协议，是I2C协议的子集，某些设备只支持此协议，所以内核文档smbus-protocol建议优先使用此方法)
- (2) i2c\_transfer方式(I2C协议)

应用程序可以直接通过上述两种方式访问I2C设备，因为内核已经封装了一套驱动程序，应用程序只需要使用那套驱动就可以访问I2C设备了。应用程序设计方法可以参考文档dev-interface，这里给出测试代码。

Device Drivers

I2C support

<\*> I2C device interface

i2c\_usr\_test.c :

```

[cpp]
1. #include <stdio.h>
2. #include <stdlib.h>
3. #include <string.h>
4. #include <sys/types.h>
5. #include <sys/stat.h>
6. #include <fcntl.h>
7. #include "i2c-dev.h"
8.
9. /* i2c_usr_test </dev/i2c-0> <dev_addr> r addr
10.  * i2c_usr_test </dev/i2c-0> <dev_addr> w addr val
11.  */
12.
13. void print_usage(char *file)
14. {
15.     printf("%s </dev/i2c-0> <dev_addr> r addr\n", file);
16.     printf("%s </dev/i2c-0> <dev_addr> w addr val\n", file);
17. }
18.
19. int main(int argc, char **argv)
20. {
21.     int fd;
22.     unsigned char addr, data;

```



AGV小车



#### 联系我们



请扫描二维码联系客服

✉ webmaster@csdn.net

☎ 400-660-0108

🗣 QQ客服 🗣 客服论坛

关于 招聘 广告服务 百度

©1999-2018 CSDN版权所有

京ICP证09002463号

经营性网站备案信息

网络110报警服务

中国互联网举报中心

北京互联网违法和不良信息举报中心

```

23.     int dev_addr;
24.
25.     if ((argc != 5) && (argc != 6))
26.     {
27.         print_usage(argv[0]);
28.         return -1;
29.     }
30.
31.     fd = open(argv[1], O_RDWR);
32.     if (fd < 0)
33.     {
34.         printf("can't open %s\n", argv[1]);
35.         return -1;
36.     }
37.
38.     dev_addr = strtoul(argv[2], NULL, 0);
39.     if (ioctl(fd, I2C_SLAVE, dev_addr) < 0)
40.     {
41.         /* ERROR HANDLING; you can check errno to see what went wrong */
42.         printf("set addr error!\n");
43.         return -1;
44.     }
45.
46.     if (strcmp(argv[3], "r") == 0)
47.     {
48.         addr = strtoul(argv[4], NULL, 0);
49.
50.         data = i2c_smbus_read_word_data(fd, addr);
51.
52.         printf("data: %c, %d, 0x%2x\n", data, data, data);
53.     }
54.     else if ((strcmp(argv[3], "w") == 0) && (argc == 6))
55.     {
56.         addr = strtoul(argv[4], NULL, 0);
57.         data = strtoul(argv[5], NULL, 0);
58.         i2c_smbus_write_byte_data(fd, addr, data);
59.     }
60.     else
61.     {
62.         print_usage(argv[0]);
63.         return -1;
64.     }
65.
66.     return 0;
67. }

```



AGV小车



## 联系我们



请扫描二维码联系客服  
 webmaster@csdn.net  
 400-660-0108  
 QQ客服 客服论坛

关于 招聘 广告服务 百度  
 ©1999-2018 CSDN版权所有  
 京ICP证09002463号

经营性网站备案信息  
 网络110报警服务  
 中国互联网举报中心  
 北京互联网违法和不良信息举报中心

## 五、编写"总线(适配器adapter)"驱动

应用程序调用设备驱动程序，设备驱动程序会调用核心层提供的某些函数(如SMBUS相关函数)，核心层的这些函数最终会调用到适配器层的相关函数，这些函数是处理器I2C接口的相关硬件操作，它们会根据I2C协议向I2C设备发出相应信号达到控制I2C设备的目的。

这里设计的驱动即设计适配器驱动程序，先找到Linux3.5中内核自带的适配器驱动。

在make menuconfig后，找到如下选项：

Device Drivers

I2C support

I2C Hardware Bus support

< > S3C2410 I2C Driver

选中S3C2410 I2C Driver，按下键盘上的h键，找到第一行出现的那个宏 CONFIG\_I2C\_S3C24

1 后在内核源码目录下执行 grep "CONFIG\_I2C\_S3C2410" -R \*后，找到如下信息：

c s/i2c/busses/Makefile:obj-\$(CONFIG\_I2C\_S3C2410) += i2c-s3c2410.o

上面信息中就可以知道I2C总线适配器对应的驱动程序文件为i2c-s3c2410.c文件。然后分析此文件，分析类似驱动程序都是先从入口函数看。下面给出测试代码(基于2440):

```

[cpp]
1. #include <linux/kernel.h>
2. #include <linux/module.h>
3. #include <linux/i2c.h>
4. #include <linux/init.h>
5. #include <linux/time.h>
6. #include <linux/interrupt.h>
7. #include <linux/delay.h>
8. #include <linux/errno.h>
9. #include <linux/err.h>
10. #include <linux/platform_device.h>
11. #include <linux/pm_runtime.h>
12. #include <linux/clock.h>
13. #include <linux/cpufreq.h>
14. #include <linux/slab.h>

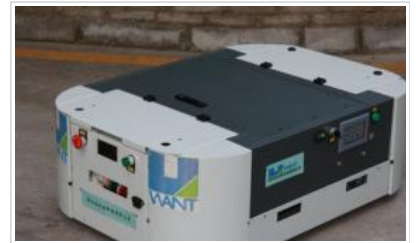
```



```

15. #include <linux/io.h>
16. #include <linux/of_i2c.h>
17. #include <linux/of_gpio.h>
18. #include <plat/gpio-cfg.h>
19. #include <mach/regs-gpio.h>
20. #include <asm/irq.h>
21. #include <plat/regs-iic.h>
22. #include <plat/iic.h>
23.
24. // #define PRINTK printk
25. #define PRINTK(...)
26.
27. enum s3c24xx_i2c_state {
28.     STATE_IDLE,
29.     STATE_START,
30.     STATE_READ,
31.     STATE_WRITE,
32.     STATE_STOP
33. };
34.
35. struct s3c2440_i2c_regs {
36.     unsigned int iiccon;
37.     unsigned int iicstat;
38.     unsigned int iicadd;
39.     unsigned int iicds;
40.     unsigned int iiclc;
41. };
42.
43. struct s3c2440_i2c_xfer_data {
44.     struct i2c_msg *msgs;
45.     int msn_num;
46.     int cur_msg;
47.     int cur_ptr;
48.     int state;
49.     int err;
50.     wait_queue_head_t wait;
51. };
52.
53. static struct s3c2440_i2c_xfer_data s3c2440_i2c_xfer_data;
54.
55. static struct s3c2440_i2c_regs *s3c2440_i2c_regs;
56.
57. static void s3c2440_i2c_start(void)
58. {
59.     s3c2440_i2c_xfer_data.state = STATE_START;
60.
61.     if (s3c2440_i2c_xfer_data.msgs->flags & I2C_M_RD) /* 读 */
62.     {
63.         s3c2440_i2c_regs->iicds = s3c2440_i2c_xfer_data.msgs->addr << 1;
64.         s3c2440_i2c_regs->iicstat = 0xb0; // 主机接收, 启动
65.     }
66.     else /* 写 */
67.     {
68.         s3c2440_i2c_regs->iicds = s3c2440_i2c_xfer_data.msgs->addr << 1;
69.         s3c2440_i2c_regs->iicstat = 0xf0; // 主机发送, 启动
70.     }
71. }
72.
73. static void s3c2440_i2c_stop(int err)
74. {
75.     s3c2440_i2c_xfer_data.state = STATE_STOP;
76.     s3c2440_i2c_xfer_data.err = err;
77.
78.     PRINTK("STATE_STOP, err = %d\n", err);
79.
80.     if (s3c2440_i2c_xfer_data.msgs->flags & I2C_M_RD) /* 读 */
81.     {
82.         // 下面两行恢复I2C操作, 发出P信号
83.         s3c2440_i2c_regs->iicstat = 0x90;
84.         s3c2440_i2c_regs->iiccon = 0xaf;
85.         ndelay(50); // 等待一段时间以便P信号已经发出
86.     }
87.     else /* 写 */
88.     {
89.         // 下面两行用来恢复I2C操作, 发出P信号
90.         s3c2440_i2c_regs->iicstat = 0xd0;
91.         s3c2440_i2c_regs->iiccon = 0xaf;
92.         ndelay(50); // 等待一段时间以便P信号已经发出
93.     }
94. }
95.
96. /* 唤醒 */
97. wake_up(&s3c2440_i2c_xfer_data.wait);
98.
99. }
100.
101. static int s3c2440_i2c_xfer(struct i2c_adapter *adap,

```



AGV小车



## 联系我们



请扫描二维码联系客服

✉ webmaster@csdn.net

☎ 400-660-0108

👤 QQ客服 🗣 客服论坛

关于 招聘 广告服务 百度

©1999-2018 CSDN版权所有

京ICP证09002463号

经营性网站备案信息

网络110报警服务

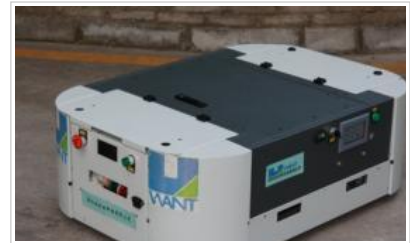
中国互联网举报中心

北京互联网违法和不良信息举报中心

```

102.         struct i2c_msg *msgs, int num)
103. {
104.     unsigned long timeout;
105.
106.     /* 把num个msg的I2C数据发送出去/读进来 */
107.     s3c2440_i2c_xfer_data.msgs = msgs;
108.     s3c2440_i2c_xfer_data.msn_num = num;
109.     s3c2440_i2c_xfer_data.cur_msg = 0;
110.     s3c2440_i2c_xfer_data.cur_ptr = 0;
111.     s3c2440_i2c_xfer_data.err = -ENODEV;
112.
113.     s3c2440_i2c_start();
114.
115.     /* 休眠 */
116.     timeout = wait_event_timeout(s3c2440_i2c_xfer_data.wait, (s3c2440_i2c_xfer_data.state == ST
ATE_STOP), HZ * 5);
117.     if (0 == timeout)
118.     {
119.         printk("s3c2440_i2c_xfer time out\n");
120.         return -ETIMEDOUT;
121.     }
122.     else
123.     {
124.         return s3c2440_i2c_xfer_data.err;
125.     }
126. }
127.
128. static u32 s3c2440_i2c_func(struct i2c_adapter *adap)
129. {
130.     return I2C_FUNC_I2C | I2C_FUNC_SMBUS_EMUL | I2C_FUNC_PROTOCOL_MANGLING;
131. }
132.
133. static const struct i2c_algorithm s3c2440_i2c_algo = {
134.     // .smbus_xfer = ,
135.     .master_xfer = s3c2440_i2c_xfer,
136.     .functionality = s3c2440_i2c_func,
137. };
138.
139. /* 1. 分配/设置i2c_adapter
140.  */
141. static struct i2c_adapter s3c2440_i2c_adapter = {
142.     .name = "s3c2440_100ask",
143.     .algo = &s3c2440_i2c_algo,
144.     .owner = THIS_MODULE,
145. };
146.
147. static int isLastMsg(void)
148. {
149.     return (s3c2440_i2c_xfer_data.cur_msg == s3c2440_i2c_xfer_data.msn_num - 1);
150. }
151.
152. static int isEndData(void)
153. {
154.     return (s3c2440_i2c_xfer_data.cur_ptr >= s3c2440_i2c_xfer_data.msgs->len);
155. }
156.
157. static int isLastData(void)
158. {
159.     return (s3c2440_i2c_xfer_data.cur_ptr == s3c2440_i2c_xfer_data.msgs->len - 1);
160. }
161.
162. static irqreturn_t s3c2440_i2c_xfer_irq(int irq, void *dev_id)
163. {
164.     unsigned int iicSt;
165.     iicSt = s3c2440_i2c_regs->iicstat;
166.
167.     if(iicSt & 0x8){ printk("Bus arbitration failed\n"); }
168.
169.     switch (s3c2440_i2c_xfer_data.state)
170.     {
171.         case STATE_START : /* 发出S和设备地址后,产生中断 */
172.         {
173.             PRINTK("Start\n");
174.             /* 如果没有ACK, 返回错误 */
175.             if (iicSt & S3C2410_IICSTAT_LASTBIT)
176.             {
177.                 s3c2440_i2c_stop(-ENODEV);
178.                 break;
179.             }
180.
181.             if (isLastMsg() && isEndData())
182.             {
183.                 s3c2440_i2c_stop(0);
184.                 break;
185.             }
186.
187.             /* 进入下一个状态 */

```



AGV小车



## 联系我们



请扫描二维码联系客服

✉ webmaster@csdn.net

☎ 400-660-0108

💬 QQ客服 🗨 客服论坛

关于 招聘 广告服务 百度

©1999-2018 CSDN版权所有

京ICP证09002463号

经营性网站备案信息

网络110报警服务

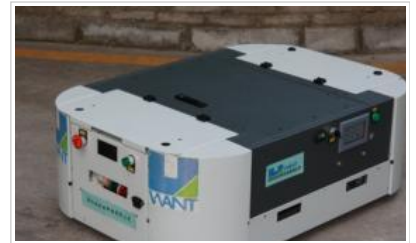
中国互联网举报中心

北京互联网违法和不良信息举报中心

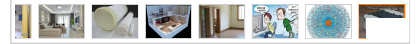
```

188.         if (s3c2440_i2c_xfer_data.msgs->flags & I2C_M_RD) /* 读 */
189.         {
190.             s3c2440_i2c_xfer_data.state = STATE_READ;
191.             goto next_read;
192.         }
193.         else
194.         {
195.             s3c2440_i2c_xfer_data.state = STATE_WRITE;
196.         }
197.     }
198.
199.     case STATE_WRITE:
200.     {
201.         PRINTK("STATE_WRITE\n");
202.         /* 如果没有ACK, 返回错误 */
203.         if (iicSt & S3C2410_IICSTAT_LASTBIT)
204.         {
205.             s3c2440_i2c_stop(-ENODEV);
206.             break;
207.         }
208.
209.         if (!isEndData()) /* 如果当前msg还有数据要发送 */
210.         {
211.             s3c2440_i2c_regs->iicds = s3c2440_i2c_xfer_data.msgs->buf[s3c2440_i2c_xfer_data.cur_ptr];
212.             s3c2440_i2c_xfer_data.cur_ptr++;
213.
214.             // 将数据写入IICDS后, 需要一段时间才能出现在SDA线上
215.             ndelay(50);
216.
217.             s3c2440_i2c_regs->iiccon = 0xaf; // 恢复I2C传输
218.             break;
219.         }
220.         else if (!isLastMsg())
221.         {
222.             /* 开始处理下一个消息 */
223.             s3c2440_i2c_xfer_data.msgs++;
224.             s3c2440_i2c_xfer_data.cur_msg++;
225.             s3c2440_i2c_xfer_data.cur_ptr = 0;
226.             s3c2440_i2c_xfer_data.state = STATE_START;
227.             /* 发出START信号和发出设备地址 */
228.             s3c2440_i2c_start();
229.             break;
230.         }
231.         else
232.         {
233.             /* 是最后一个消息的最后一个数据 */
234.             s3c2440_i2c_stop(0);
235.             break;
236.         }
237.
238.         break;
239.     }
240.
241.     case STATE_READ:
242.     {
243.         PRINTK("STATE_READ\n");
244.         /* 读出数据 */
245.         s3c2440_i2c_xfer_data.msgs->buf[s3c2440_i2c_xfer_data.cur_ptr] = s3c2440_i2c_regs->iicds;
246.         s3c2440_i2c_xfer_data.cur_ptr++;
247.         next_read:
248.         if (!isEndData()) /* 如果数据没读写, 继续发起读操作 */
249.         {
250.             if (isLastData()) /* 如果即将读的数据是最后一个, 不发ack */
251.             {
252.                 s3c2440_i2c_regs->iiccon = 0x2f; // 恢复I2C传输, 接收到下一数据时无ACK
253.             }
254.             else
255.             {
256.                 s3c2440_i2c_regs->iiccon = 0xaf; // 恢复I2C传输, 接收到下一数据时发出ACK
257.             }
258.             break;
259.         }
260.         else if (!isLastMsg())
261.         {
262.             /* 开始处理下一个消息 */
263.             s3c2440_i2c_xfer_data.msgs++;
264.             s3c2440_i2c_xfer_data.cur_msg++;
265.             s3c2440_i2c_xfer_data.cur_ptr = 0;
266.             s3c2440_i2c_xfer_data.state = STATE_START;
267.             /* 发出START信号和发出设备地址 */
268.             s3c2440_i2c_start();
269.             break;
270.         }
271.         else
272.         {

```



AGV小车



## 联系我们



请扫描二维码联系客服

✉ webmaster@csdn.net

☎ 400-660-0108

🗣 QQ客服 🗣 客服论坛

关于 招聘 广告服务 百度

©1999-2018 CSDN版权所有

京ICP证09002463号

经营性网站备案信息

网络110报警服务

中国互联网举报中心

北京互联网违法和不良信息举报中心

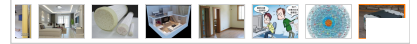
```

273.         /* 是最后一个消息的最后一个数据 */
274.         s3c2440_i2c_stop(0);
275.         break;
276.     }
277.     break;
278. }
279.
280.     default: break;
281. }
282.
283. /* 清中断 */
284. s3c2440_i2c_regs->iiccon &= ~(S3C2410_IICCON_IRQPEND);
285.
286.     return IRQ_HANDLED;
287. }
288.
289. /*
290.  * I2C初始化
291.  */
292. static void s3c2440_i2c_init(void)
293. {
294.     struct clk *clk;
295.
296.     clk = clk_get(NULL, "i2c");
297.     clk_enable(clk);
298.
299.     // 选择引脚功能: GPE15:IICSDA, GPE14:IIC_SCL
300.     s3c_gpio_cfgpin(S3C2410_GPE(14), S3C2410_GPE14_IIC_SCL);
301.     s3c_gpio_cfgpin(S3C2410_GPE(15), S3C2410_GPE15_IIC_SDA);
302.
303.     /* bit[7] = 1, 使能ACK
304.      * bit[6] = 0, IICCLK = PCLK/16
305.      * bit[5] = 1, 使能中断
306.      * bit[3:0] = 0xf, Tx clock = IICCLK/16
307.      * PCLK = 50MHz, IICCLK = 3.125MHz, Tx Clock = 0.195MHz
308.      */
309.     s3c2440_i2c_regs->iiccon = (1<<7) | (0<<6) | (1<<5) | (0xf); // 0xaf
310.
311.     s3c2440_i2c_regs->iicadd = 0x10; // S3C24xx slave address = [7:1]
312.     s3c2440_i2c_regs->iicstat = 0x10; // I2C串行输出使能(Rx/Tx)
313. }
314.
315. static int i2c_bus_s3c2440_init(void)
316. {
317.     /* 2. 硬件相关的设置 */
318.     s3c2440_i2c_regs = ioremap(0x54000000, sizeof(struct s3c2440_i2c_regs));
319.
320.     s3c2440_i2c_init();
321.
322.     request_irq(IRQ_IIC, s3c2440_i2c_xfer_irq, 0, "s3c2440-i2c", NULL);
323.
324.     init_waitqueue_head(&s3c2440_i2c_xfer_data.wait);
325.
326.     /* 3. 注册i2c_adapter */
327.     i2c_add_adapter(&s3c2440_i2c_adapter);
328.
329.     return 0;
330. }
331.
332. static void i2c_bus_s3c2440_exit(void)
333. {
334.     i2c_del_adapter(&s3c2440_i2c_adapter);
335.     free_irq(IRQ_IIC, NULL);
336.     iounmap(s3c2440_i2c_regs);
337. }
338.
339. module_init(i2c_bus_s3c2440_init);
340. module_exit(i2c_bus_s3c2440_exit);
341. MODULE_LICENSE("GPL");

```



AGV小车



## 联系我们



请扫描二维码联系客服  
 webmaster@csdn.net  
 400-660-0108  
 QQ客服 客服论坛

关于 招聘 广告服务 百度

©1999-2018 CSDN版权所有

京ICP证09002463号

经营性网站备案信息

网络110报警服务

中国互联网举报中心

北京互联网违法和不良信息举报中心

## 六、总结

通过上述分析，可以总结出一个设计驱动程序的方法，这个方法适用于Linux内核的各个版本：

1. 如果要设计I2C设备驱动，看 Documentation/i2c目录下的相关内核文档；I2C协议；处理器的I2C接口；掌握bus-dev-drv模型；掌握驱动程序设计相关知识。
2. 实现bus-dev-drv模型驱动。比如要先注册i2c\_client，而内核文档中instantiating-devices(构造设备)文件就说明了方法。然后根据方法，搜索内核中对应的例子，模仿就可以设计出需要的驱动；然后要注册i2c\_driver，也是模仿其他如何设计即可。**总之，Linux是非常庞大的系统，里面有很多例子可以参考的。**

3. 理清思路，搞清楚框架中哪些工作是需要做的，哪些是不需要做的。

《smbus-protocol》中介绍了核心层各种读写接口，这些接口被设备驱动的读写接口调用。根据此文档，下面举例说明这些接口的应用：

1

SMBus Read Byte: i2c\_smbus\_read\_byte\_data()  
: =====

1 Reads a single byte from a device, from a designated register.  
The register is specified through the Comm byte.

S Addr Wr [A] Comm [A] S Addr Rd [A] [Data] NA P

i2c\_smbus\_read\_byte\_data函数在Linux内核中原型如下：

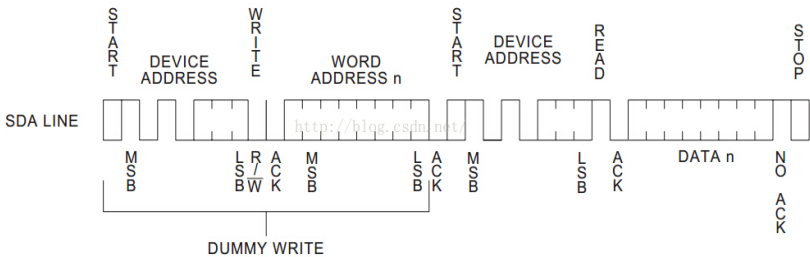
```
s32 i2c_smbus_read_byte_data(const struct i2c_client *client, u8 command)
{
    union i2c_smbus_data data;
    int status;

    status = i2c_smbus_xfer(client->adapter, client->addr, client->flags,
                            I2C_SMBUS_READ, command,
                            I2C_SMBUS_BYTE_DATA, &data);
    return (status < 0) ? status : data.byte;
}
```

从字面意思上就可以猜测i2c\_smbus\_read\_byte\_data函数是从I2C的子集smbus上读取一个字节的数 据，它的第一个参数为i2c\_client结构体，这个结构在probe函数被调用后就被记录了(记录方法就是用一个i2c\_client结构体指针指向probe函数的参数)，第二个参数的含义是明确读取寄存器的地址，比如用户要读取EEPROM地址为1地方的数据，那么这里就把1传给内核空间并传给此函数。i2c\_smbus\_read\_byte\_data函数的返回值即从它的第二个参数Comm为地址的地方读出的数据。

然而，对于4412上的I2C总线0上接的EEPROM设备来说，为什么使用这个函数呢？原因在于这个函数的时序和EEPROM芯片手册中读时序类似，下图是EEPROM读时序：

Figure 5. Random Read



根据上图，对照smbus-protocol文档中对i2c\_smbus\_read\_byte\_data()函数的介绍，即S Ad dr Wr [A] Comm [A] S Addr Rd [A] [Data] NA P，下面是详细说明：

- 第一个为S表示起始信号；
- 第二个为Addr表示I2C设备的地址，这里就是EEPOM的地址0x50；
- 第三个为Wr表示操作方向为写；
- 第四个为[A]表示EEPROM给处理器I2C控制器的回应，实际上就是把SDA拉低而已；

第五个为Comm，写地址。表示要读取EEPROM中哪个地方的数据，是个地址，也是i2c\_smbus\_read\_byte\_data函数的第二个参数，

一般由用户空间提供；



AGV小车



联系我们



请扫描二维码联系客服  
webmaster@csdn.net  
400-660-0108  
QQ客服 客服论坛

关于 招聘 广告服务 百度  
©1999-2018 CSDN版权所有  
京ICP证09002463号

经营性网站备案信息  
网络110报警服务  
中国互联网举报中心  
北京互联网违法和不良信息举报中心





AGV小车



联系我们



请扫描二维码联系客服  
✉ webmaster@csdn.net  
☎ 400-660-0108  
👤 QQ客服 🗨 客服论坛

关于 招聘 广告服务 百度  
©1999-2018 CSDN版权所有  
京ICP证09002463号

经营性网站备案信息  
网络110报警服务  
中国互联网举报中心  
北京互联网违法和不良信息举报中心



1



第六个为[A]表示EEPOM给处理器I2C控制器的回应，实际上就是把SDA拉低而已；

第七个为S表示重新发送起始信号；

第八个为Addr表示I2C设备的地址，这里就是EEPOM的地址0x50；

第九个为Rd表示操作方向为读；

第十个为[A]表示EEPOM给处理器I2C控制器的回应，实际上就是把SDA拉低而已；

第十一个为[Data]，表示从EEPROM中Comm地址处读取的数据；

第十二个为NA，表示上述读取的数据为最后一个数据，这个时候处理器不用给eeprom回应，ACK；

第十三个为P，表示由主机给出的停止信号，结束对EEPROM的操作。



严禁讨论涉及中国之军/政相关话题，违者会被禁言、封号！

linux的I2C驱动——移植篇



Smile\_Smilling 2017年08月05日 21:33 731

\*\*一、简介\*\* 1、I2C是一个一主多从的通信协议，通信都是由主设备发起的。 SCL：时钟线，由主端控制； SDA：数据线，主端和从端都可以配置； ...

详解Linux-I2C驱动



omnispace 2016年02月25日 18:06 2563

目录 一、LinuxI2C驱动--概述 1.1 写在前面 1.2 I2C 1.3 硬件 1.4 软件 1.5 参考二、LinuxI2C驱动--I2C总线 2.1 I2C总线物理结构 ...

一个数学公式教你秒懂天下英语

老司机教你一个数学公式秒懂天下英语

Linux的i2c驱动详解

ylyuanlu 2011年08月21日 13:05 11117

I<sup>2</sup>C 总线仅仅使用 SCL、SDA 两根信号线就实现了设备之间的数据交互，极大地简化对硬件资源和 PCB 板布线空间的占用。因此，I<sup>2</sup>C 总线被非常广泛地应用在 EEP...

linux I2C驱动移植

Edroid1530 2017年05月08日 16:34 555

I<sup>2</sup>C总线仅使用SCL, SDA两根信号线实现设备间的数据交互，被广泛应用于微控制领域芯片与芯片之间的通信，如EEPROM，实时时钟，小型LCD等与CPU之间的通信。I2C协议I2C利...

Linux I2C设备驱动编写（一）

airk000 2014年03月16日 23:24 34218

在Linux驱动中I2C系统中主要包含以下几个成员：I2C adapter 即I2C适配器 I2C driver 某个I2C设备的设备驱动，可以以driver理解。I2C client 某个I2...

十大猎头公司

国内知名猎头公司的排名

百度广告



linux下I2C驱动架构全面分析

wangpengqi 2013年12月31日 11:04 20582

I2C 概述 I2C是philips提出的外设总线。 I2C只有两条线,一条串行数据线:SDA,一条是时钟线SCL，使用SCL, SDA这两根信号线就实现了设备之间的数据交互，它方便了工程...

Linux I2C驱动完全分析（一）

ypoflyer 2011年04月30日 16:38 37863

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。借着五一放假的时间把这个写出来，供同志们参考。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，但是还不系统。I2C的硬件结...

Linux I2C驱动

coding\_\_madman 2016年06月10日 00:39 704

I2C总线介绍：1. I2C硬件结构 1.1 I2C电气特性 I2C总线是由PHILIPS公司开发的两线式串行总线，用于连接微控制器及其外围设备。 I2C总线只有两根...

Linux I2C驱动分析(一)----I2C架构和总线驱动

本文作为i2c驱动分析的第一部分，主要讲述：1、I2C总线原理。2、I2C架构概述。3、I2C代码在内核中的结构。4、Algorithm中的传输函数master\_xfer。5、总线驱动注册和...

apple\_guet 2014年03月17日 11:50 4730

LINUX\_I2C驱动程序

2013年12月11日 22:43 296KB 下载



Linux 设备驱动篇之I2c设备驱动

sonbai 2013年06月02日 20:24 10946

Linux 设备驱动篇之I2c设备驱动 fulinux 一、I2C驱动体系 虽然I2C硬件体系结构和协议都很容易理解，但是Linux I2C驱动体系结构却有相当的复杂度，它主要由3部分组成，即I2C设...

嘿！阿里云服务套餐，今日起可免费体验>>

阿里巴巴集团旗下云计算品牌，全球卓越的云计算技术和服务提供商。



Linux I2C驱动分析与实现--例子

bingqingsuimeng 2012年09月03日 14:39 3739

====文系本站原创,欢迎转载! 转载请注明出处:http://blog.csdn.net/yyplc==== 通过上篇《Linux I2C驱动分析与实现(二)》，我们对Linux子系统已经...



AGV小车



联系我们



请扫描二维码联系客服  
webmaster@csdn.net  
400-660-0108  
QQ客服 客服论坛

关于 招聘 广告服务 百度  
©1999-2018 CSDN版权所有  
京ICP证09002463号

经营性网站备案信息  
网络110报警服务  
中国互联网举报中心  
北京互联网违法和不良信息举报中心

Linux下I2C驱动分析（三）

ouchao0727

2015年12月17日 13:50

912

分析了两天的I2C驱动，发现每次解决一个问题的时候都会带来新的问题，当大致读完MMA7660驱动程序的时候发现，作为一个字符设备I2C驱动，并不存在有open，close等接口，而我们知道，在Linu...

lii

I2C设备驱动编写

hanmengaidudu

2013年08月12日 12:54

2662

实例解析linux内核I2C体系结构

[http://blog.csdn.net/hongtao\\_liu/archive/2009/12/08/4964244.aspx](http://blog.csdn.net/hongtao_liu/archive/2009/12/08/4964244.aspx)

<http://bk>

Li

系统I2C设备驱动编写方法

BorntoX

2016年07月11日 08:56

2228

硬件平台：飞思卡尔iMX6 内核版本：kernel3.0.35 Linux的I2C子系统分为三层，I2C核心层，I2C总线驱动层和I2C设备驱动层。I2C核心层由内核开发者提供，I2C总线...

i2c设备驱动的四种构造方法

wangweiqiang1325

2016年09月23日 16:33

641

i2c设备驱动属于字符设备驱动，其构造自然是跟字符设备的结构一样了，字符设备：1、 分配字符设备号（主次设备号），设置为0，表示自动分配设备号 2、构造file\_operatios 3、注册设备， ...

Linux设备驱动篇——[I2C设备驱动-1]

luoyouren

2015年04月24日 12:13

1170

Linux 设备驱动篇之I2c设备驱动 fulinux 一、I2C驱动体系 虽然I2C硬件体系结构和协议都很容易理解，但是Linux I2C驱动体系结构却有相当的复杂度，它主要由3部分组成， ...

Linux驱动之I2C设备驱动完全解析

cmh477660693

2017年02月11日 10:47

784

上一节介绍了I2C的相关协议，本节主要讲I2C的设备驱动的创建 在内核iTop4412\_Kernel\_3.0\Documentation\i2c\instantiating-devices这个文档介...

两种方式的i2c设备驱动的编写方法

我的理念：简单实用即可，不要搞一堆源码出来，结果让人看了以后还不知道怎么用，看我的： 1、在arch/arm/mach-xxx/ 自己的平台文件里添加i2c信息，美其名曰：i2c\_b...

y

yyyyyyyyywwwwwwwwww

2015年12月18日 10:57

781

i2c设备与驱动匹配过程

u013952558

2015年12月04日 15:47

2887

linux下i2c驱动笔记 1. 几个基本概念 1.1. 设备模型 由 总线（ bus\_type ） + 设备（ device ） + 驱动（ device\_d...



AGV小车



联系我们



请扫描二维码联系客服

webmaster@csdn.net

400-660-0108

QQ客服 客服论坛

关于 招聘 广告服务 百度

©1999-2018 CSDN版权所有

京ICP证09002463号

经营性网站备案信息

网络110报警服务

中国互联网举报中心

北京互联网违法和不良信息举报中心

[http://blog.csdn.net/fengel\\_cs/article/details/50522495](http://blog.csdn.net/fengel_cs/article/details/50522495)

21/21