实验16：I2C驱动程序编写实验

一、实验目的

1．熟悉I2C协议的原理。

2. 熟悉liunx下I2C的驱动构架。

3. 熟悉Linux驱动程序的编写。

4. 熟悉Linux下模块的加载等。

二、实验内容

1.编写I2C协议的驱动程序。

2.编写Makefile。

3.以模块方式加载驱动程序。

4.编写I2C测试程序。

三、实验设备

1.硬件：PC机，基于ARM9系统教学实验系统实验箱1台；网线；串口线，电压表。

2.软件：PC机操作系统；Putty；服务器Linux操作系统；arm-v5t\_le-gcc交叉编译环境。

3.环境：ubuntu12.04.4；文件系统版本为filesys\_test；烧写的内核版本为uImage\_wlw。源码及参考文档见附件I2C驱动实验文件夹。

四、预备知识

1. C语言的基础知识。

2. 软件调试的基础知识和方法。

3. Linux基本操作。

4. Linux应用程序的编写。

五、实验说明

5.1 概述

I2C（Inter－Integrated Circuit）总线是由[PHILIPS](http://baike.baidu.com/view/30962.htm" \t "_blank)公司开发的两线式串行总线，用于连接[微控制器](http://baike.baidu.com/view/368297.htm" \t "_blank)及其外围设备。是微电子通信控制领域广泛采用的一种总线标准。它是同步通信的一种特殊形式，具有接口线少，控制方式简单，器件封装形式小，通信速率较高等优点。I2C 总线支持任何IC 生产工艺(CMOS、双极型）。通过串行数据（SDA）线和串行时钟 （SCL）线在连接到总线的器件间传递信息。每个器件都有一个唯一的地址识别（无论是微控制器——MCU、LCD 驱动器、存储器或键盘接口），而且都可以作为一个发送器或接收器（由器件的功能决定）。任何被寻址的器件都被认为是从机。

I2C总线（I2C bus，Inter-IC bus）是一个双向的两线连续总线，提供集成电路（ICs）之间的通信线路。I2C总线是一种串行扩展技术，最早由Philips公司推出，广泛应用于电视，录像机和音频设备。I2C总线的意思是“完成集成电路或功能单元之间信息交换的规范或协议”。Philips公司推出的I2C总线采用一条数据线（SDA），加一条时钟线（SCL）来完成数据的传输及外围器件的扩展；对各个节点的寻址是软寻址方式，节省了片选线，标准的寻址字节SLAM为7位，可以寻址127个单元。

I2C总线有三种数据传输速度：标准，快速模式和高速模式。标准的是100Kbps，快速模式为400Kbps，高速模式支持快至3.4Mbps的速度。所有的与次之传输速度的模式都是兼容的。I2C总线支持7位和10位地址空间设备和在不同电压下运行的设备。

Linux的I2C体系结构分为3个组成部分：

I2C核心：I2C核心提供了I2C总线驱动和设备驱动的注册，注销方法，I2C通信方法(”algorithm”)上层的，与具体适配器无关的代码以及探测设备，检测设备地址的上层代码等。

I2C总线驱动：I2C总线驱动是对I2C硬件体系结构中适配器端的实现，适配器可由CPU控制，甚至可以直接集成在CPU内部。

I2C设备驱动：I2C设备驱动(也称为客户驱动)是对I2C硬件体系结构中设备端的实现，设备一般挂接在受CPU控制的I2C适配器上，通过I2C适配器与CPU交换数据。

4.2实现的功能

1> 实现了将设备挂载I2C总线。

2> 实现了设备本身驱动注册。

4.3 基本原理

4.3.1 I2C协议

以启动信号START来掌管总线，以停止信号STOP来释放总线；每次通信以START开始，以STOP结束；启动信号START后紧接着发送一个地址字节，其中7位为被控器件的地址码，一位为读/写控制位R/W，R/W位为0表示由主控向被控器件写数据，R/W为1表示由主控向被控器件读数据；当被控器件检测到收到的地址与自己的地址相同时，在第9个时钟周期间反馈应答信号；每个数据字节在传送时都是高位（MSB）在前。

写通信过程：

主控在检测到总线空闲的状况下，首先发送一个START信号掌管总线。

发送一个地址字节（包括7位地址码和一位R/W）。

当被控器件检测到主控发送的地址与自己的地址相同时发送一个应答信号（ACK）。

主控收到ACK后开始发送第一个数据字节。

被控器收到数据字节后发送一个ACK表示继续传送数据，发送NACK表示传送数据结束。

主控发送完全部数据后，发送一个停止位STOP，结束整个通信并且释放总线。

读通信过程：

主控在检测到总线空闲的状况下，首先发送一个START信号掌管总线。

发送一个地址字节（包括7位地址码和一位R/W）。

当被控器件检测到主控发送的地址与自己的地址相同时发送一个应答信号（ACK）。

主控收到ACK后释放数据总线，开始接收第一个数据字节。

主控收到数据后发送ACK表示继续传送数据，发送NACK表示传送数据结束。

主控发送完全部数据后，发送一个停止位STOP，结束整个通信并且释放总线。

5.3.2 总线信号时序分析

总线空闲状态：SDA和SCL两条信号线都处于高电平，即总线上所有的器件都释放总线，两条信号线各自的上拉电阻把电平拉高。

启动信号START：时钟信号SCL保持高电平，数据信号SDA的电平被拉低（即负跳变）。启动信号必须是跳变信号，而且在建立该信号前必须保证总线处于空闲状态。

停止信号STOP：时钟信号SCL保持高电平，数据线被释放，使得SDA返回高电平（即正跳变），停止信号也必须是跳变信号。

数据传送：SCL线呈现高电平期间，SDA线上的电平必须保持稳定，低电平表示0（此时的线电压为低电压），高电平表示1（此时的电压由元器件的VDD决定）。只有在SCL线为低电平期间，SDA上的电平允许变化。

应答信号ACK：I2C总线的数据都是以字节（8位）的方式传送的，发送器件每发送一个字节之后，在时钟的第9个脉冲期间释放数据总线，由接收器发送一个ACK（把数据总线的电平拉低）来表示数据成功接收。

无应答信号NACK：在时钟的第9个脉冲期间发送器释放数据总线，接收器不拉低数据总线表示一个NACK，NACK有两种用途：一般表示接收器未成功接收数据字节；当接收器是主控器时，收到最后一个字节后，应发送一个NACK信号，以通知被控发送器结束数据发送，并释放总线，以便主控接收器发送一个停止信号STOP。

5.3.3 寻址约定

地址的分配方法有两种：

含CPU的智能器件，地址由软件初始化时定义，但不能与其他的器件有冲突。

不含CPU的非智能器件，由厂家在器件内部固化，不可改变。高7位为地址码，分为两部分：高4位属于固定地址不可改变，由厂家固化的统一地址；低3位为引脚设定地址，可以由外部引脚来设定（并非所有器件都可以设定）。

5.3.4 I2C驱动构架

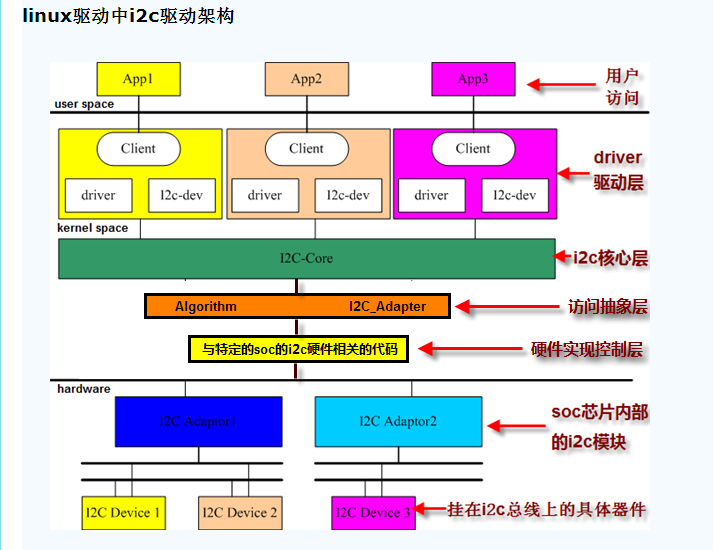
如下图所示1所示，I2C驱动框架大概可以分为3个组成部分

图1 I2C驱动框架

第一层：提供i2c adapter的硬件驱动，探测、初始化i2c adapter（如申请i2c的io地址和中断号），驱动soc控制的i2c adapter在硬件上产生信号（start、stop、ack）以及处理i2c中断。覆盖图中的硬件实现层。

第二层：提供i2c adapter的algorithm，用具体适配器的xxx\_xferf()函数来填充i2c\_algorithm的master\_xfer函数指针，并把赋值后的i2c\_algorithm再赋值给i2c\_adapter的algo指针。覆盖图中的访问抽象层、i2c核心层。

第三层：实现i2c设备驱动中的i2c\_driver接口，用具体的i2c device设备的attach\_adapter()、detach\_adapter()方法赋值给i2c\_driver的成员函数指针。实现设备device与总线（或者叫adapter）的挂接。覆盖图中的driver驱动层。

第四层：实现i2c设备所对应的具体device的驱动，i2c\_driver只是实现设备与总线的挂接，而挂接在总线上的设备则是千差万别的，例如eeprom和ov2715显然不是同一类的device，所以要实现具体设备device的write()、read()、ioctl()等方法，赋值给file\_operations，然后注册字符设备（多数是字符设备）。覆盖图中的driver驱动层。

第一层和第二层又叫i2c总线驱动(bus)，第三第四属于i2c设备驱动(device driver)。

在linux驱动架构中，几乎不需要驱动开发人员再添加bus，因为linux内核几乎集成所有总线bus，如usb、pci、i2c等等。并且总线bus中的(与特定硬件相关的代码)已由芯片提供商编写完成，例如TI davinci平台i2c总线bus与硬件相关的代码在内核目录/drivers/i2c/buses下的i2c-davinci.c源文件中；三星的s3c-2440平台i2c总线bus为/drivers/i2c/buses/i2c-s3c2410.c。

第三第四层与特定device相干的就需要驱动工程师来实现了。

5.3.5 架构层次分类

Linux I2C驱动体系结构主要由3部分组成，即I2C核心、I2C总线驱动和I2C设备驱动。I2C核心是I2C总线驱动和I2C设备驱动的中间枢纽，它以通用的、与平台无关的接口实现了I2C中设备与适配器的沟通。I2C总线驱动填充i2c\_adapter和i2c\_algorithm结构体。I2C设备驱动填充i2c\_driver和i2c\_client结构体。

（1）I2C核心框架

I2C核心框架具体实现在/drivers/i2c目录下的i2c-core.c和i2c-dev.c。 I2C核心框架提供了核心数据结构的定义和相关接口函数，用来实现I2C适配器 驱动和设备驱动的注册、注销管理，以及I2C通信方法上层的、与具体适配器无关的代码，为系统中每个I2C总线增加相应的读写方法。

（2）I2C总线驱动

I2C总线驱动具体实现在/drivers/i2c目录下busses文件夹。例如：Linux I2C GPIO总线驱动为i2c\_gpio.c. I2C总线算法在/drivers/i2c目录下algos文件夹。例如：Linux I2C GPIO总线驱动算法实现在i2c\_algo\_bit.c。 I2C总线驱动定义描述具体I2C总线适配器的i2c\_adapter数据结构、实现在具体I2C适配器上的I2C总线通信方法，并由i2c\_algorithm数据结 构进行描述。经过I2C总线驱动的的代码，可以为我们控制I2C产生开始位、停止位、读写周期以及从设备的读写、产生ACK等，此部分驱动已经编译进内核。

(3) I2C设备驱动

I2C设备驱动具体实现放在在/drivers/i2c目录下chips文件夹。I2C设备驱动是对具体I2C硬件驱动的实现。I2C设备驱动通过I2C适配器与CPU通信。其中主要包含i2c\_driver和 i2c\_client数据结构，i2c\_driver结构对应一套具体的驱动方法，例如：probe、remove、suspend等，需要自己申明。 i2c\_client数据结构由内核根据具体的设备注册信息自动生成，设备驱动根据硬件具体情况填充。

5.3.6 I2C体系文件构架

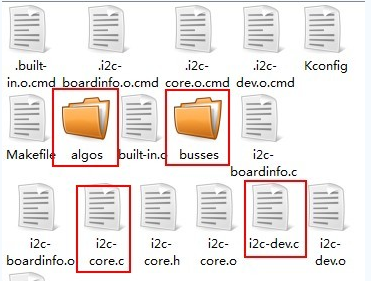
在Linux内核源代码中的driver目录下包含一个i2c目录，如下图2所示：

图2 文件夹架构

i2c-core.c这个文件实现了I2C核心的功能以及/proc/bus/i2c\*接口。

i2c-dev.c实现了I2C适配器设备文件的功能，每一个I2C适配器都被分配一个设备。通过适配器访设备时的主设备号都为89，次设备号为0-255。I2c-dev.c并没有针对特定的设备而设计，只是提供了通用的read(),write(),和ioctl()等接口，应用层可以借用这些接口访问挂接在适配器上的I2C设备的存储空间或寄存器，并控制I2C设备的工作方式。

busses文件夹这个文件中包含了一些I2C总线的驱动，如针对S3C2410，S3C2440，S3C6410等处理器的I2C控制器驱动为i2c-s3c2410.c。

algos文件夹实现了一些I2C总线适配器的algorithm。

下面张图展示是linux内核和芯片提供商为我们的的驱动程序提供了 i2c驱动的框架，以及框架底层与硬件相关的代码的实现，如图3所示。剩下的就是针对挂载在i2c两线上的i2c设备了device，如x1205,ov5640，tvp5151,触摸屏，而编写的具体设备驱动了，这里的设备就是硬件接口外挂载的设备，而非硬件接口本身（soc硬件接口本身的驱动可以理解为总线驱动）。

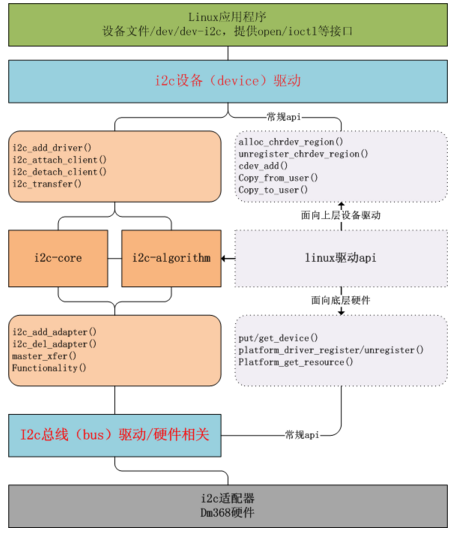


图3 函数结构代码梳理图

5.4总体硬件结构设计

外接名为SY的设备通过I2C与DM365的电路连接，硬件电路图如图4所示：

图4 I2C硬件电路图

5.5 软件框架

5.5.1 软件流程

设备名为SY的设备驱动注册流程如图5所示：



图5 I2C注册流程图

在SY\_init函数中，使用了I2C总线提供的i2c\_add\_driver函数，向I2C总线添加了SY的设备驱动，从而引起设备探测函数i2c\_probe的调用，这会使在该函数下的SY\_detect函数通过调用I2C总线驱动提供的i2c\_attach\_client函数，向内核的I2C总线注册SY设备，从而完成驱动的注册。

5.5.2 操作函数的接口函数和结构体

1> 驱动源码函数分析

函数：MODULE\_LICENSE("GPL");

功能：将模块的许可协议设置为GPL许可,必不可少的。

函数：module\_init(SY\_init);

功能：module\_init是内核模块一个宏。其用来声明模块的加载函数，也就是使用insmod命令加载模块时，调用的函数SY\_init()。

函数: module\_exit(SY\_exit);

功能：module\_exit是内核模块一个宏。其用来声明模块的释放函数，也就是使用rmmod命令卸载模块时，调用的函数SY\_exit（）。

函数：static int SY\_init (void)

功能：加载函数调用驱动注册函数实现驱动程序在内核的注册，同时还有可能对设备进行初始化，在驱动程序加载被调用。

函数：static void SY\_exit(void)

功能：卸载函数调用解除注册函数实现驱动程序在内核的中的解除注册，同时在驱动程序卸载时被调用。

函数：i2c\_add\_driver(&SYdriver);

功能：调用核心层函数，注册SY\_driver结构体

函数：static int SY\_attach(struct i2c\_adapter \*adapter)

功能：在attach中通过probe探测到总线上的设备并把设备和驱动建立连接以完成设备的初始化。

函数： i2c\_attach\_client(SY\_client);

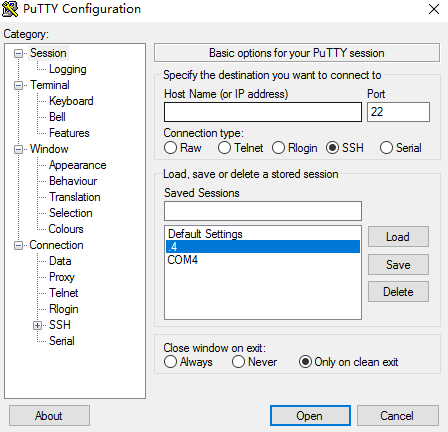
功能：等要卸载驱动时，调用SY\_detech\_adapter函数

六. 实验步骤

本次实验将执行下面几个步骤：

**步骤1：**硬件连接

首先通过putty软件使用ssh通信方式登录到服务器，如下图一所示（在Hostname栏输入服务器的ip地址）：



图一 打开putty连接

接着查看串口号，通过putty软件使用串口通信方式连接实验箱，如下图二所示：



图二 端口号查询

选择putty串口连接实验箱如三所示：

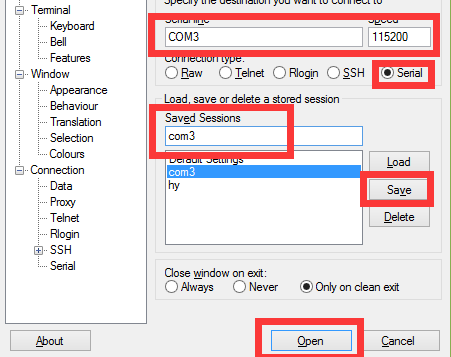
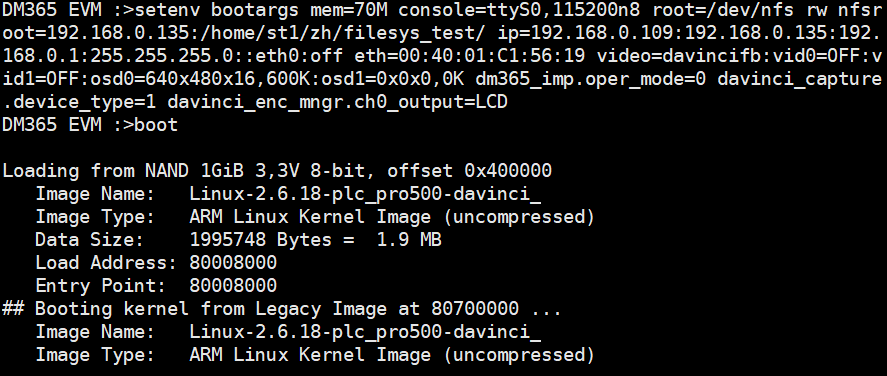


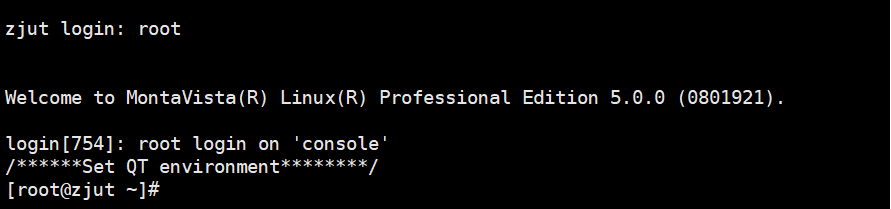
图3 putty串口连接配置

输入启动参数，接着启动内核，如下图四所示：



图四 输入启动参数启动

输入用户名root登录实验箱如下图五所示：



图五 登陆实验箱

**步骤2**：编译I2C驱动（在服务器窗口上进行）

进入学生目录，输入“mkdir I2C”命令，创建I2C驱动文件夹，编写I2C驱动程序。

设备驱动源码参考文件夹I2C驱动实验/I2C/i2c.c。

测试程序源码参考文件夹I2C驱动实验/I2C/i2c\_test.c

测试程序源码参考文件夹I2C驱动实验/I2C/i2c\_test.c

将code文件夹下的I2C程序复制到自己创建的目录下，输入命令：

虚拟机：“cp /home/shiyan/Desktop/shiyan/code/I2C/I2C\_CODE/\* I2C/”

服务器：“cp /home/shiyan/2021/code/I2C/I2C\_CODE/\* I2C/”

进入自己创建的文件夹

“cd I2C”

编写驱动程序编译成模块所需要的Makefile文件，执行vim Makefile（已编好）

KDIR:= /home/stx/kernel-for-mceb //编译驱动模块依赖的内核路径，修改为使用服务器 //上内核的路径

CROSS\_COMPILE = arm\_v5t\_le-

CC = $(CROSS\_COMPILE)gcc

.PHONY: modules clean

obj-m := i2c.o //表明有个模块要从目标文件 i2c.o 建立. 在 从目标文件建立后结果模？

块命名为i2c.ko

modules:

make -C $(KDIR) M=`pwd` modules //根据提供的内核生成i2c.o

clean:

make -C $(KDIR) M=`pwd` modules clean

i2c驱动代码如下：

#include <linux/kernel.h>

#include <linux/init.h>

#include <linux/module.h>

#include <linux/slab.h>

#include <linux/jiffies.h>

#include <linux/i2c.h>

#include <linux/mutex.h>

#include <linux/fs.h>

#include <asm/uaccess.h>

#include <linux/miscdevice.h>

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

static unsigned short normal\_addr[] = { 0x6F, I2C\_CLIENT\_END };//设备地址为：01101111（0x6f) 七位

static unsigned short force\_addr[] = {ANY\_I2C\_BUS,0x6F, I2C\_CLIENT\_END};

static unsigned short \* forces[] = {force\_addr, NULL};

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

.normal\_i2c = normal\_addr, //要发出地址信号，并且得到ACK信号，才能确定是否存在这个设备

.probe = ignore,

.ignore = ignore,

.forces = forces, //强制认为存在这个设备

};

static struct i2c\_driver SY\_driver;

static int major;

static struct class \*cls; //自动创建设备节点

struct i2c\_client \*SY\_client;

static ssize\_t SY\_read(struct file \*filp, char \_\_user \*buf, size\_t count, loff\_t \* ppos)

{

int ret=0;

static volatile unsigned char values[1]={0};

values[0]=((char)SY\_client->addr);

if ( copy\_to\_user(buf, (void\*)values,count )) { //将地址值buf内存的内容传到用户空间的values

ret = -EFAULT;

goto out;

}

out:

return ret;

}

//定义字符设备结构体

static struct file\_operations SY\_fops = {

.owner = THIS\_MODULE,

.read = SY\_read,

};

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

{

printk("SY\_detect\n");

//构建一个i2c\_client结构体；接收数据主要靠它，里面有 .address .adapter .driver

SY\_client = kzalloc(sizeof(struct i2c\_client), GFP\_KERNEL);

SY\_client->addr = address;

SY\_client->adapter = adapter;

SY\_client->driver = &SY\_driver;

strcpy(SY\_client->name, "SY");

i2c\_attach\_client(SY\_client);//等要卸载驱动时，会调用 I2C\_detach

printk("SY\_probe with name = %s, addr = 0x%x\n", SY\_client->name, SY\_client->addr);

major = register\_chrdev(0, "SY", &SY\_fops);//申请字符设备主设备号

cls = class\_create(THIS\_MODULE, "SY");//创建一个类 ，然后在类下面创建一个设备

class\_device\_create(cls, NULL, MKDEV(major, 0), NULL, "SY");

return 0;

}

static int SY\_attach(struct i2c\_adapter \*adapter)

{

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

}

static int SY\_detach(struct i2c\_client \*client)

{

printk("SY\_detach\n");

class\_device\_destroy(cls, MKDEV(major, 0));

class\_destroy(cls);

unregister\_chrdev(major, "SY");

i2c\_detach\_client(client);//client 结构体

kfree(i2c\_get\_clientdata(client));//释放client的内存

return 0;

}

//定义i2c\_driver结构体

static struct i2c\_driver SY\_driver = {

.driver = {

.name = "SY",

},

.attach\_adapter = SY\_attach,

.detach\_client = SY\_detach,

};

static int SY\_init(void)

{

printk("SY\_init\n");

i2c\_add\_driver(&SY\_driver);//注册i2c驱动

return 0;

}

static void SY\_exit(void)

{

printk("SY\_exit\n");

i2c\_del\_driver(&SY\_driver);

}

module\_init(SY\_init);

module\_exit(SY\_exit);

MODULE\_LICENSE("GPL");

执行“make”命令，成功后会生成i2c.ko等文件，如图6所示。

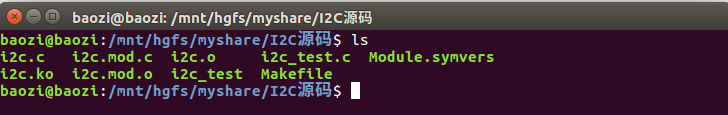


图6 编译驱动模块

i2c.ko驱动文件生成成功后，将其复制到挂载的文件系统modules的目录下。

虚拟机：“ sudo cp i2c.ko /home/shiyan/share/filesys\_test/modules“

服务器：“ sudo cp i2c.ko /home/stX/filesys\_test/modules“

root@zjut modules]# ls

adc\_driver.ko dht11.ko egalax\_i2c.ko i2c.ko ov5640\_i2c.ko sr04\_driver.ko

davinci\_dm365\_gpios.ko egalax.ko hello.ko

misc.kort5370sta.ko ts35xx-i2c.ko

**步骤3**：加载驱动(在实验箱COM窗口操作)

1. 在驱动加载之前查看当前已经加载的驱动模块，使用命令lsmod。如图8：

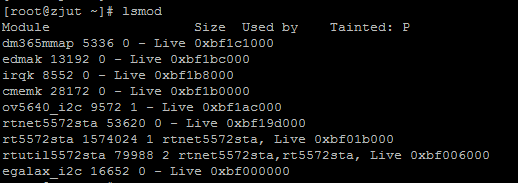


图8 当前加载的驱动模块

1. 执行cd /sys/bus/i2c，如图9所示。查看当前加载的设备地址和设备名。所有的I2C设备都在sysfs文件系统中显示。在当前目录文件夹下的devices下，是当前挂载在I2C的设备地址。在drivers目录下是挂载I2C上的设备驱文件。分别进入devices和drivers文件夹下查看一下

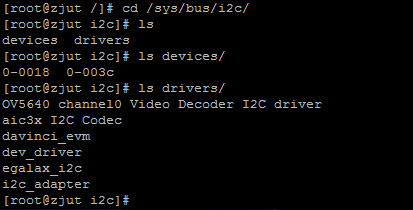


图9 设备地址和设备文件名

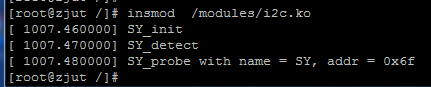
1. 执行insmod /modules/i2c.ko。在驱动加载成功后会打印出设备的I2C注册地址。具体添加设备的地址查看数据手册，每一个设备都会有一个固定地址。如下图10所示。例如添加设备地址为0x6f，转化成二进制为一个8位的数据。I2C协议地址为7位。

图10 手动加载驱动模块

1. 驱动加载成功之后，查看devices和drivers目录，会增加一个0-006f文件。如下图11所示：

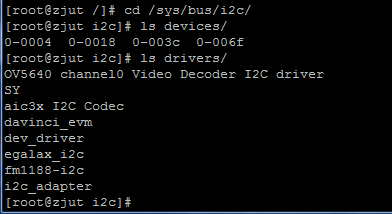
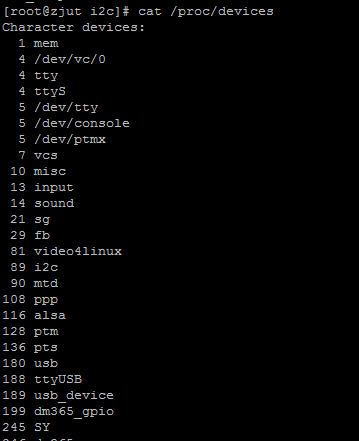


图11 设备SY注册成功图

5> 执行cat /proc/devices,显示加载的I2C设备驱动程序创建了一个主设备号为245名为SY的设备节点，如图12所示：



**步骤4**：编写测试程序，并进行调试。（在服务器上进行）

测试程序在文件夹I2C驱动实验/I2C/i2c\_test下，i2c\_test.c.就是对应的测试文件。用来查看设备的地址。

测试代码如下：

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

int main(int argc, char \*\*argv)

{

int fd,ret;

unsigned char values[0];

fd = open("/dev/SY", O\_RDWR);

if (fd < 0)

{

printf("can't open /dev/SY\n");

return -1;

}

ret= read(fd,values,sizeof(unsigned char));

if (ret >= 0){

printf("reading data is OK \n");

}

else{

printf("read data is fialed \n",ret);

}

printf("SY address = 0x%x\n",values[0]);

return 0;

}

使用交叉编译工具编译测试程序，并将编译后生成的可执行文件挂载到实验箱上运行调试。

$ arm\_v5t\_le-gcc i2c\_test.c -o i2c\_test

将交叉编译生成的i2c\_test文件拷贝到挂载的文件系统目录下：

虚拟机：cp i2c\_test /home/shiyan/share/filesys\_test/opt/dm365

服务器：cp i2c\_test /home/stX/filesys\_test/opt/dm365

在putty实验箱窗口执行如下命令i2c\_test，如下图13所示。读出当前设备地址。

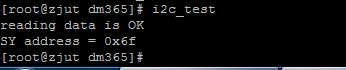


图13 执行测试程序