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 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。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

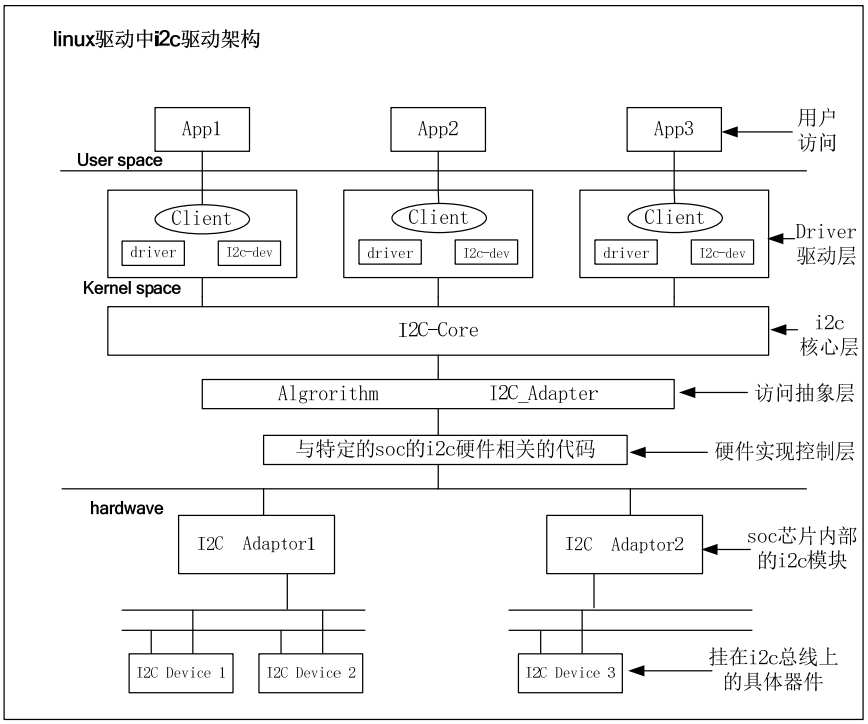

图 1　I2C 驱动框架

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-sibyte.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）。

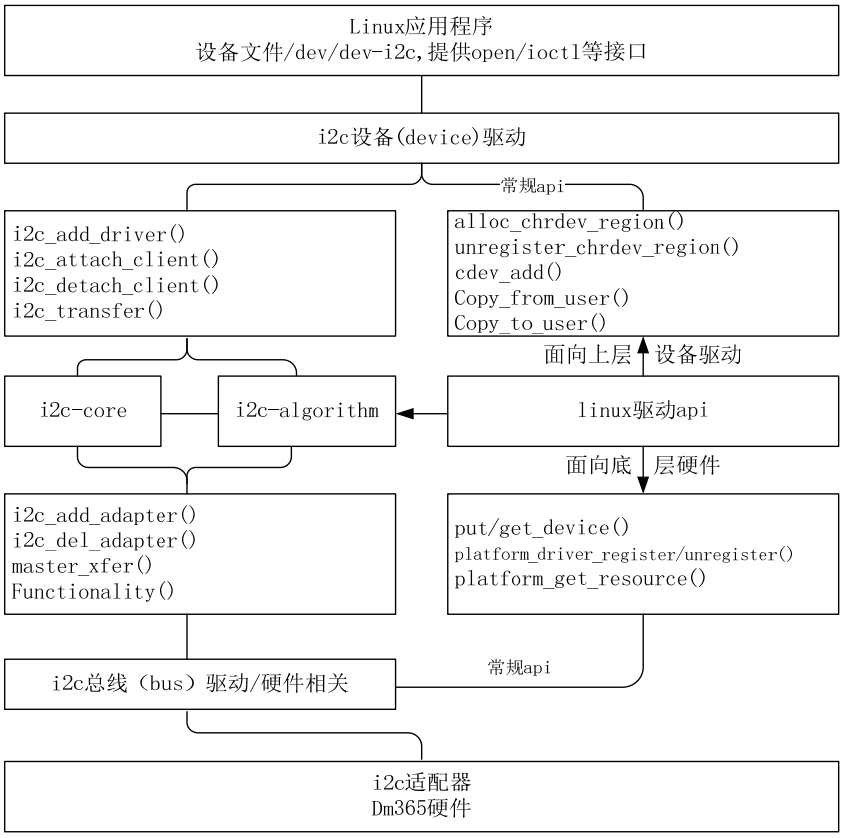

图 3　函数结构代码梳理图

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：

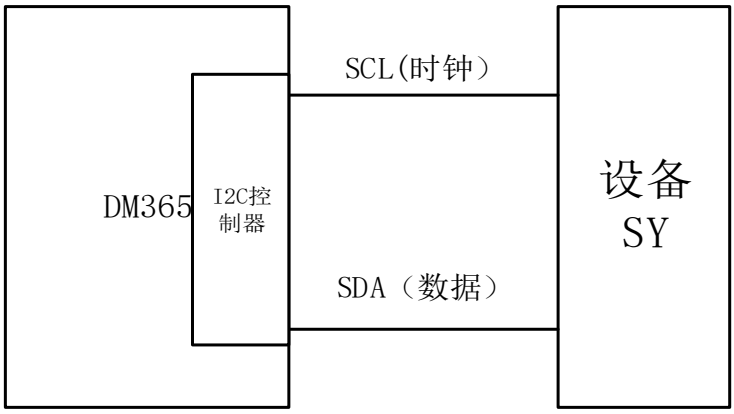

图 4　I2C 硬件电路图

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：

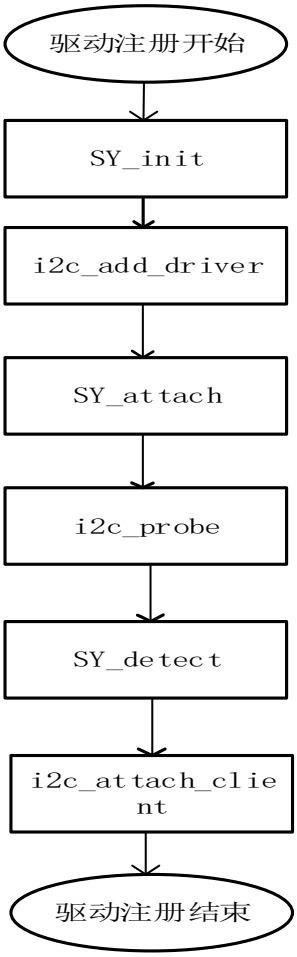

图 5　I2C 注册流程图

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

(1) 连接好实验箱的网线、串口线和电源。

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

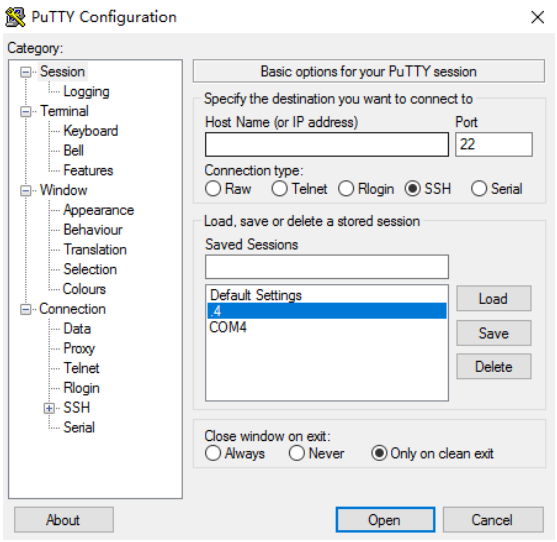

图 1 打开 putty 连接

(3) 查看串口号，右键我的电脑--->选择管理--->设备管理器--->端口，查看实验箱的串口号。如下图 2 所示：

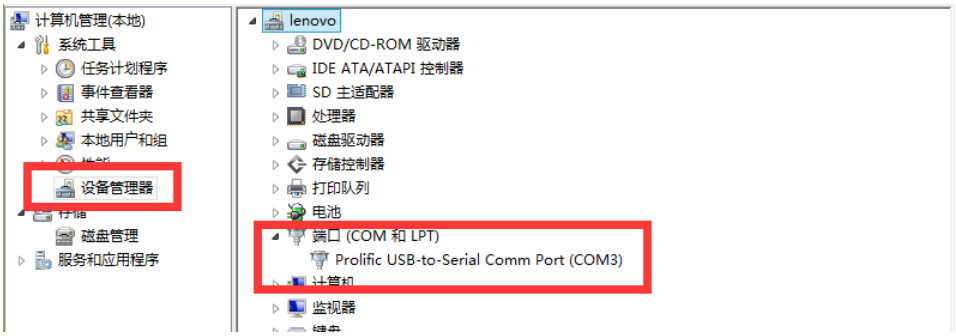

图 2 端口号查询

(4) 在 putty 软件端口栏输入(3)中查询到的串口，设置波特率为 115200，连接实验箱，如下图 3 所示：

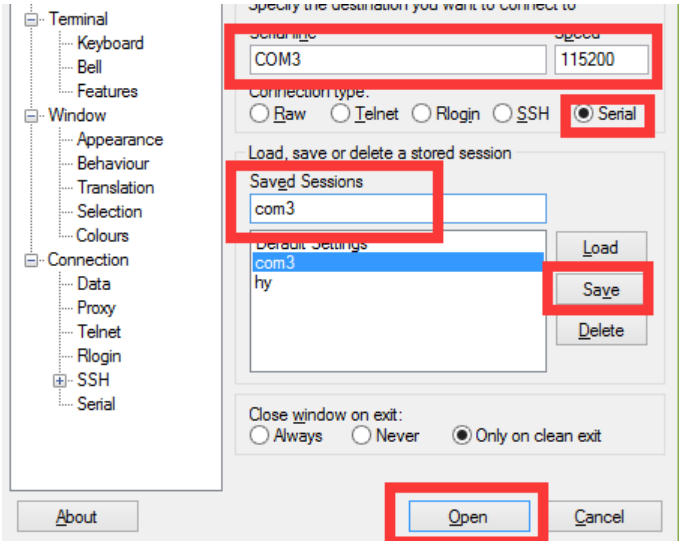

图 3 putty 串口连接配置

(5) 点击 putty 软件中的 Open 按键，进入连接页面，打开实验箱开关，在 5s 内，点击 Entet 键，然后输入挂载参数，再次点击 Enter 键，输入 boot 命令，按 Enter 键，开始进行挂载。具体信息如下所示：

```
DM365 EVM :>setenv bootargs 'mem=110M console=ttyS0,115200n8 root=/dev/nfs rw
nfsroot=192.168.1.18:/home/shiyan/filesys_clwxl
ip=192.168.1.42:192.168.1.18:192.168.1.1:255.255.255.0::eth0:off eth=00:40:01:C1:56:78
video=davincifb:vid0=OFF:vid1=OFF:osd0=640x480x16,600K:osd1=0x0x0,0K dm365_imp.oper_mode=0
davinci_capture.device_type=1 davinci_enc_mngr.ch0_output=LCD'
DM365 EVM :>boot

Loading from NAND 1GiB 3,3V 8-bit, offset 0x400000
    Image Name:   Linux-2.6.18-plc_pro500-davinci_
    Image Type:   ARM Linux Kernel Image (uncompressed)
    Data Size:    1996144 Bytes =  1.9 MB
    Load Address: 80008000
    Entry Point:  80008000
## Booting kernel from Legacy Image at 80700000 ...
    Image Name:   Linux-2.6.18-plc_pro500-davinci_
    Image Type:   ARM Linux Kernel Image (uncompressed)
    Data Size:    1996144 Bytes =  1.9 MB
    Load Address: 80008000
    Entry Point:  80008000
    Verifying Checksum ... OK
    Loading Kernel Image ... OK
OK

Starting kernel ...

Uncompressing Linux.....
done, booting the kernel.
[        0.000000] Linux version 2.6.18-plc_pro500-davinci_evm-arm_v5t_le-gfaa0b471-dirty
(zcy@punuo-Lenovo) (gcc version 4.2.0 (MontaVista 4.2.0-16.0.32.0801914 2008-08-30)) #1 PREEMPT
Mon Jun 27 15:31:35 CST 2016
[        0.000000] CPU: ARM926EJ-S [41069265] revision 5 (ARMv5TEJ), cr=00053177
[        0.000000] Machine: DaVinci DM365 EVM
[        0.000000] Memory policy: ECC disabled, Data cache writeback
[        0.000000] DaVinci DM0365 variant 0x8
[        0.000000] PLL0: fixedrate: 24000000, commonrate: 121500000, vpssrate: 243000000
[        0.000000] PLL0: vencrate_sd: 27000000, ddrrate: 243000000 mmcsdrate: 121500000
[        0.000000] PLL1: armrate: 297000000, voicerate: 20482758, vencrate_hd: 74250000
[        0.000000] CPU0: D VIVT write-back cache
[        0.000000] CPU0: I cache: 16384 bytes, associativity 4, 32 byte lines, 128 sets
[        0.000000] CPU0: D cache: 8192 bytes, associativity 4, 32 byte lines, 64 sets
[        0.000000] Built 1 zonelists.  Total pages: 28160
[        0.000000] Kernel command line: mem=110M console=ttyS0,115200n8 root=/dev/nfs rw
nfsroot=192.168.1.18:/home/shiyan/filesys_clwxl
```

```
ip=192.168.1.42:192.168.1.18:192.168.1.1:255.255.255.0::eth0:off eth=00:40:01:C1:56:78
video=davincifb:vid0=OFF:vid1=OFF:osd0=640x480x16,600K:osd1=0x0x0,0K dm365_imp.oper_mode=0
davinci_capture.device_type=1 davinci_enc_mngr.ch0_output=LCD
[    0.000000] TI DaVinci EMAC: kernel boot params Ethernet address: 00:40:01:C1:56:78
[    0.000000] PID hash table entries: 512 (order: 9, 2048 bytes)
[    0.000000] Clock event device timer0_0 configured with caps set: 07
[    0.000000] Console: colour dummy device 80x30
[    0.000000] Dentry cache hash table entries: 16384 (order: 4, 65536 bytes)
[    0.000000] Inode-cache hash table entries: 8192 (order: 3, 32768 bytes)
[    0.010000] Memory: 110MB = 110MB total
[    0.010000] Memory: 107136KB available (3165K code, 692K data, 492K init)
[    0.220000] Security Framework v1.0.0 initialized
[    0.220000] Capability LSM initialized
[    0.220000] Mount-cache hash table entries: 512
[    0.220000] CPU: Testing write buffer coherency: ok
[    0.220000] NET: Registered protocol family 16
[    0.240000] DaVinci: 104 gpio irqs
[    0.250000] MUX: initialized GPIO20
[    2.250000] MUX: initialized I2C_SCL
[    2.250000] Pin GPIO20 already used for I2C_SCL.
[    2.250000] MUX: initialized GPIO30
[    2.250000] MUX: initialized GPIO31
[    2.250000] MUX: initialized GPIO32
[    2.250000] MUX: initialized GPIO33
[    2.250000] MUX: initialized GPIO35
[    2.250000] MUX: initialized GPIO37
[    2.250000] MUX: initialized GPIO38
[    2.250000] MUX: initialized GPIO39
[    2.250000] MUX: initialized GPIO40
[    2.250000] MUX: initialized GPIO41
[    2.250000] MUX: initialized GPIO51
[    2.250000] MUX: initialized GPIO55
[    2.250000] MUX: initialized GPIO58
[    2.250000] MUX: initialized GPIO80
[    2.250000] MUX: initialized GPIO93
[    2.250000] MUX: initialized GPIO28
[    2.250000] MUX: initialized GPIO29
[    4.450000] MUX: initialized UART1_RXD
[    4.450000] MUX: initialized UART1_TXD
[    4.450000] DM365 IPIPE initialized in Continuous mode
[    4.460000] Generic PHY: Registered new driver
[    4.460000] ch0 default output "LCD", mode "NTSC"
[    4.460000] VPBE Encoder Initialized
[    4.460000] Invalid id...
[    4.460000] Set output or mode failed, reset to encoder default...
[    4.460000] MUX: initialized VOUT_FIELD_G81
[    4.460000] LogicPD encoder initialized
```

```
[    4.460000] Avnetlcd encoder initialized
[    4.460000] dm365_afew_hw_init
[    4.460000] SCSI subsystem initialized
[    4.460000] usbcore: registered new driver usbfs
[    4.460000] usbcore: registered new driver hub
[    4.470000] NET: Registered protocol family 2
[    4.560000] IP route cache hash table entries: 1024 (order: 0, 4096 bytes)
[    4.560000] TCP established hash table entries: 4096 (order: 2, 16384 bytes)
[    4.560000] TCP bind hash table entries: 2048 (order: 1, 8192 bytes)
[    4.560000] TCP: Hash tables configured (established 4096 bind 2048)
[    4.560000] TCP reno registered
[    4.590000] yaffs Jun 27 2016 15:33:25 Installing.
[    4.600000] Initializing Cryptographic API
[    4.600000] io scheduler noop registered
[    4.600000] io scheduler anticipatory registered (default)
[    4.600000] io scheduler deadline registered
[    4.600000] io scheduler cfq registered
[    4.620000] Console: switching to colour frame buffer device 80x30
[    4.660000] davincifb davincifb.0: dm_osd0_fb: 640x480x16@0,0 with framebuffer size 600KB
[    4.670000] davincifb davincifb.0: dm_vid0_fb: 0x0x16@0,0 with framebuffer size 900KB
[    4.670000] davincifb davincifb.0: dm_vid1_fb: 0x0x16@0,0 with framebuffer size 900KB
[    4.720000] TI Davinci ADC v1.0
[    4.730000] DAVINCI-WDT: DaVinci Watchdog Timer: heartbeat 60 sec
[    4.730000] imp serializer initialized
[    4.730000] davinci_previewer initialized
[    4.730000] davinci_resizer initialized
[    4.730000] dm365_gpio initialized
[    4.730000] Serial: 8250/16550 driver $Revision: 1.90 $ 2 ports, IRQ sharing disabled
[    4.730000] serial8250.0: ttyS0 at MMIO map 0x1c20000 mem 0xfbc20000 (irq = 40) is a 16550A
[    4.750000] serial8250.0: ttyS1 at MMIO map 0x1d06000 mem 0xfbd06000 (irq = 41) is a 16550A
[    4.760000] RAMDISK driver initialized: 1 RAM disks of 32768K size 1024 blocksize
[    4.770000] PPP generic driver version 2.4.2
[    4.770000] PPP Deflate Compression module registered
[    4.780000] PPP BSD Compression module registered
[    4.790000] Davinci EMAC MII Bus: probed
[    4.790000] sjwedit --> EMAC: 00:40:01:C1:56:78.
[    4.800000] MAC address is 00:40:01:C1:56:78
[    4.800000] TI DaVinci EMAC Linux version updated 4.0
[    4.810000] Linux video capture interface: v2.00
[    4.820000] vpfe_init
[    4.820000] Pin VIN_CAM_WEN already used for GPIO93.
[    4.820000] starting ccdc_reset...<7>
[    4.830000] End of ccdc_reset...<5>vpfe_probe
[    4.830000] TVP5150 : nummber of channels = 1
[    4.840000] vpfe ccdc capture vpfe ccdc capture.1: vpif_register_decoder: decoder = TVP5150
[    4.850000] Trying to register davinci display video device.
[    4.860000] layer=c07eb600,layer->video_dev=c07eb760
```

```
[    4.860000] Trying to register davinci display video device.
[    4.870000] layer=c07eb400,layer->video_dev=c07eb560
[    4.870000] davinci_init:DaVinci V4L2 Display Driver V1.0 loaded
[    4.880000] vpfe ccdc capture vpfe ccdc capture.1: vpif_register_decoder: decoder = TVP7002
[    4.890000] af major#: 251, minor# 0
[    4.890000] AF Driver initialized
[    4.900000] aew major#: 250, minor# 0
[    4.900000] AEW Driver initialized
[    4.910000] i2c /dev entries driver
[    4.920000] nand_davinci nand_davinci.0: Using 4-bit hardware ECC
[    4.920000] NAND device: Manufacturer ID: 0xec, Chip ID: 0xd3 (Samsung NAND 1GiB 3,3V 8-bit)
[    4.940000] Creating 5 MTD partitions on "nand_davinci.0":
[    4.940000] 0x00000000-0x00780000 : "bootloader"
[    4.950000] 0x00780000-0x00800000 : "params"
[    4.950000] 0x00800000-0x00c00000 : "kernel"
[    4.960000] 0x00c00000-0x20c00000 : "filesystem"
[    4.970000] 0x20c00000-0x40000000 : "backup_filesys"
[    4.980000] nand_davinci nand_davinci.0: hardware revision: 2.3
[    4.990000] Pin SPI1_SCLK already used for GPIO28.
[    5.000000] dm_spi.0: davinci SPI Controller driver at 0xc7008000 (irq = 42) use_dma=0
[    5.000000] Initializing USB Mass Storage driver...
[    5.010000] usbcore: registered new driver usb-storage
[    5.010000] USB Mass Storage support registered.
[    5.020000] usbcore: registered new driver usbhid
[    5.030000] drivers/usb/input/hid-core.c: v2.6:USB HID core driver
[    5.030000] usbcore: registered new driver usbserial
[    5.040000] drivers/usb/serial/usb-serial.c: USB Serial support registered for generic
[    5.050000] usbcore: registered new driver usbserial_generic
[    5.050000] drivers/usb/serial/usb-serial.c: USB Serial Driver core
[    5.060000] drivers/usb/serial/usb-serial.c: USB Serial support registered for GSM modem (1-port)
[    5.070000] usbcore: registered new driver option
[    5.070000] drivers/usb/serial/option.c: USB Driver for GSM modems: v0.7.1
[    5.080000] drivers/usb/serial/usb-serial.c: USB Serial support registered for pl2303
[    5.090000] usbcore: registered new driver pl2303
[    5.100000] drivers/usb/serial/pl2303.c: Prolific PL2303 USB to serial adaptor driver
[    5.100000] musb_hdrc: version 6.0, cppi-dma, host, debug=0
[    5.130000] musb_hdrc musb_hdrc: No DMA interrupt line
[    5.130000] musb_hdrc: USB Host mode controller at c700a000 using DMA, IRQ 12
[    5.140000] musb_hdrc musb_hdrc: MUSB HDRC host driver
[    5.140000] musb_hdrc musb_hdrc: new USB bus registered, assigned bus number 1
[    5.150000] usb usb1: configuration #1 chosen from 1 choice
[    5.160000] hub 1-0:1.0: USB hub found
[    5.160000] hub 1-0:1.0: 1 port detected
[    5.280000] DaVinci DM365 Keypad Driver
[    5.280000] MUX: initialized KEYPAD
[    5.290000] input: dm365_keypad as /class/input/input0
[    5.300000] year:2000,mon:1,day:0,hour:80,min:0,sec:0
```

```
[       5.310000] davinci-mmc davinci-mmc.0: Supporting 4-bit mode
[       5.310000] davinci-mmc davinci-mmc.0: Using DMA mode
[       5.320000] Advanced Linux Sound Architecture Driver Version 1.0.12rc1 (Thu Jun 22 13:55:50 2006
UTC).
[       5.330000] ASoC version 0.13.1
[       5.330000] AIC3X Audio Codec 0.2
[       5.340000] asoc: aic3x <-> davinci-i2s mapping ok
[       5.440000] ALSA device list:
[       5.450000]      #0: DaVinci DM365 EVM (aic3x)
[       5.450000] IPv4 over IPv4 tunneling driver
[       5.460000] GRE over IPv4 tunneling driver
[       5.460000] TCP bic registered
[       5.470000] NET: Registered protocol family 1
[       5.470000] NET: Registered protocol family 17
[       5.620000] usb 1-1: new high speed USB device using musb_hdrc and address 2
[       5.760000] usb 1-1: configuration #1 chosen from 1 choice
[       5.760000] hub 1-1:1.0: USB hub found
[       5.770000] hub 1-1:1.0: 4 ports detected
[       5.930000] Bridge firewalling registered
[       5.930000] 802.1Q VLAN Support v1.8 Ben Greear <greearb@candelatech.com>
[       5.940000] All bugs added by David S. Miller <davem@redhat.com>
[       5.940000] drivers/rtc/hctosys.c: unable to open rtc device (rtc0)
[       5.950000] Time: timer0_1 clocksource has been installed.
[       5.960000] Clock event device timer0_0 configured with caps set: 08
[       5.960000] Switched to high resolution mode on CPU 0
[       6.120000] usb 1-1.4: new full speed USB device using musb_hdrc and address 3
[       6.220000] usb 1-1.4: configuration #1 chosen from 1 choice
[       6.220000] pl2303 1-1.4:1.0: pl2303 converter detected
[       6.240000] usb 1-1.4: pl2303 converter now attached to ttyUSB0
[       7.480000] IP-Config: Complete:
[       7.480000]              device=eth0, addr=192.168.1.42, mask=255.255.255.0, gw=192.168.1.1,
[       7.490000]           host=192.168.1.42, domain=, nis-domain=(none),
[       7.490000]           bootserver=192.168.1.18, rootserver=192.168.1.18, rootpath=
[       7.500000] Looking up port of RPC 100003/2 on 192.168.1.18
[       9.520000] Looking up port of RPC 100005/1 on 192.168.1.18
[       9.540000] VFS: Mounted root (nfs filesystem).
[       9.540000] Freeing init memory: 492K
[      21.060000] usb 1-1.1: new high speed USB device using musb_hdrc and address 4
[      21.160000] usb 1-1.1: configuration #1 chosen from 1 choice

INIT: Entering runlevel: 3

Starting internet superserver: inetd.
mount: special device /dev/mmcblk0p1 does not exist
open wifi ra0
/********Start RTC********/
[      25.520000] rtusb init rt2870 --->
```

```
[      25.530000] usbcore: registered new driver rt2870
[      25.610000] minor is 63
[      25.610000] ##################
[      25.610000] [egalax_i2c]:    /proc/egalax_dbg created
[      25.620000] [egalax_i2c]:    Driver init done!
[      25.630000] egalax_i2c_detect
[      25.630000] i2c_adapter->name=DaVinci I2C adapter
[      25.640000] ######################
[      25.640000] new_client->name=egalax_i2c
[      25.640000] egalax_i2c_probe with name = egalax_i2c, addr = 0x4
[      25.670000] [egalax_i2c]:    Start probe
[      25.680000] input: eGalax_Touch_Screen as /class/input/input1
[      25.690000] [egalax_i2c]:    Register input device done
[      25.700000] No IRQF_TRIGGER set_type function for IRQ 44 (AINTC)
[      25.700000] [egalax_i2c]:    INT wakeup touch controller done
[      25.720000] [egalax_i2c]:    I2C probe done
[      25.780000] Register dht11 driver
[      25.850000] Register sr04 driver
[      25.900000] ov5640_i2c: Unknown symbol scanmode
insmod: cannot insert '/modules/ov5640_i2c.ko': Unknown symbol in module (-1): No such file or directory
[      25.960000] year:2000,mon:1,day:0,hour:80,min:0,sec:0
[      26.200000] [egalax_i2c]:    INT with irq:44
[      26.210000] [egalax_i2c]:    egalax_i2c_wq run
[      26.220000] [egalax_i2c]:    I2C get vendor command packet
[      26.220000] [egalax_i2c]:    Get Device type=1
[      26.230000] [egalax_i2c]:    I2C get vendor command packet
[      26.240000] [egalax_i2c]:    I2C get vendor command packet
osd0: xres 640 yres 480 xres_v 640 yres_v 480 line_length1280


MontaVista(R) Linux(R) Professional Edition 5.0.0 (0801921)



zjut login: [      27.200000] [egalax_i2c]:    Close egalax_i2c_wq_loopback work
[      27.210000] [egalax_i2c]:    INT with irq:44
[      27.220000] [egalax_i2c]:    egalax_i2c_wq run
[      27.230000] [egalax_i2c]:    I2C get vendor command packet
[      28.960000] CMEMK module: built on Apr    7 2014 at 10:55:46
[      28.980000]      Reference Linux version 2.6.18
[                                                  28.980000]                                                  File
/home/plc/dvsdk/linuxutils_2_24_02/packages/ti/sdo/linuxutils/cmem/src/module/cmemk.c
[      29.110000] ioremap_nocache(0x87000000, 16777216)=0xcb000000
[      29.110000] allocated heap buffer 0xcb000000 of size 0x3f7000
[      29.130000] cmem initialized 9 pools between 0x87000000 and 0x88000000
[      29.130000] CMEM Range Overlaps Kernel Physical - allowing overlap
[      29.130000] CMEM phys_start (0x1000) overlaps kernel (0x80000000 -> 0x86e00000)
```

```
[     29.150000] ioremap_nocache(0x1000, 28672)=0xc7010000
[     29.150000] no remaining memory for heap, no heap created for memory block 1
[     29.160000] cmem initialized 1 pools between 0x1000 and 0x8000
[     29.240000] IRQK module: built on Apr    7 2014 at 11:01:18
[     29.240000]      Reference Linux version 2.6.18
[                                                  29.250000]                                File
/home/plc/dvsdk/linuxutils_2_24_02/packages/ti/sdo/linuxutils/irq/src/module/irqk.c
[     29.270000] irqk initialized
[     29.340000] EDMAK module: built on Apr    7 2014 at 10:58:36
[     29.360000]      Reference Linux version 2.6.18
[                                                  29.370000]                                File
/home/plc/dvsdk/linuxutils_2_24_02/packages/ti/sdo/linuxutils/edma/src/module/edmak.c
WCDMA Autodialog
[     34.480000] Starting ccdc_config_ycbcr...<7>
[     34.480000] starting ccdc_reset...<7>
[     34.490000] End of ccdc_reset...<7>
[     34.490000] Starting ccdc_setwin...<7>ipipe_set_resizer, resizer - A enabled
[     34.610000] DavinciDisplay DavinciDisplay.1: Before finishing with S_FMT:
[     34.610000] layer.pix_fmt.bytesperline = 640,
[     34.610000]    layer.pix_fmt.width = 640,
[     34.610000]    layer.pix_fmt.height = 480,
[     34.610000]    layer.pix_fmt.sizeimage =460800
[     34.640000] DavinciDisplay DavinciDisplay.1: pixfmt->width = 640,
[     34.640000]    layer->layer_info.config.line_length= 640
KeypadDriverPlugin::create####################: optkeypad
keyboard input device ( "/dev/input/event0" ) is opened.
id= "0"
msqid= 0


MontaVista(R) Linux(R) Professional Edition 5.0.0 (0801921)
```

(6) 点击 Enter，输入用户名 root 登录实验箱，如下所示：

```
zjut login: root


Welcome to MontaVista(R) Linux(R) Professional Edition 5.0.0 (0801921).

login[737]: root login on 'console'

/******Set QT environment********/

[root@zjut ~]#
```

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
（注：文件夹名称以及驱动设备名称采用学生名字字母）。
测试程序源码参考文件夹 I2C 驱动实验/I2C/i2c_test.c
（注：文件夹名称以及驱动设备名称采用学生名字字母）。
编写驱动程序编译成模块所需要的 Makefile 文件，执行 vim Makefile

```
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，如下所示。

```
baozi@baozi: /mnt/hgfs /ryshare/I2C 源码$ ls
i2c.c     i2c .nod.c       i2c.o    i2c_ test.c     Module. symvers
i2c. ko    i2C nod.o      i2C_ test    Makefile
```

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。

（命令 sudo cp i2c.ko /home/XXX/nfs/ 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。如下所示：

```
[root@zjut ~]# lsmod
Module              Size          Used by             Tainted: P
dm3 65mmap 5336    0 - Live 0xbf1c1000
edmak 13192    0 - Live 0xbf1bc000
irqk 8552 0 - Live 0xbf1b8000
cmemk 28172 0 - Live 0xbf1b0000
ov5640 i2c 9572 1 - Live 0xbflac000
rtnet5572ata 53620 0 一 Live 0xbf1 9d000
rt5572sta 1574024 1 rtnet5572sta, Live 0xbf01b000
rtuti15572sta 79988 2 rtnet5572sta, rt5572sta，Live 0xbf006000
egalax_i2c 16652 0 - Live 0xbf000000
```

2> 执行 cd /sys/bus/i2c，如下所示。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。

```
[root@zjut / ]# cd    /sys/bu3/ i2c/
[root@zjut i2c]# ls
devices    drivers
[rootezjut i2c]# ls    devices/
0-0018       0-003c
[root@zjut 12c]# ls drivers/
OV5640    channe10    Video    Decoder    I2C    driver
aic3x    I2C    Codec
davinci_evm
dev_driver
egalax_i2c
i2c_adapter
[root@zjut i2c]#
```

3> 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。如下所示：

```
[ rootezjut    / ]#    insmod    /modules/i2c. ko
[ 1007.460000 ]    SY_init
[ 1007.470000 ]    SY_detect
[ 1007.480000 ]    SY_probe with name = SY,    addr = 0x6f
[ root@zjut    / ]#
```

4> 驱动加载成功之后，查看 devices 和 drivers 目录，如下所示：

```
[root@zjut    /]# cd    /sys/bus/i2c/
[root@zjut    i2c]# ls    devices/
0-0004    0-0018    0-003c      0-006f
[root@zjut 12c]# ls    drivers/
OV5640    channe10    Video    Decoder    I2C    driver
SY
aic3x    I2C    Codec
davinci_evm
dev_driver
egalax_i2c
fm1188-i2c
i2c_adapter
[root0zjut 12c]#
```

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，如下所示：

```
[root@zjut    i2c]#    cat    /proc/devices
Character devices:
1    mem
4    /dev/vc/0
4    tty
4    ttyS
5    /dev/tty
5    /dev/console
```

```
5   /dev/ptmx
7   vcs
10   misc
13   input
14   sound
21   sg
29   fb
81   video4linux
89   i2c
90   mtd
108   ppp
116   alsa
128   ptm
136   pts
180   usb
188   ttyUSB
189   usb_ device
199   dm365_ gpio
245   SY
```

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：

```
cp i2c_test    /home/shiyan/filesys_test/opt/dm365
```

在 putty 实验箱窗口执行如下命令 i2c_test，读出当前设备地址，如下所示。

```
[root@zjut dm365]# i2c_test
reading data is OK
SY    address = 0x6f
[root@zjut dm365]#
```