

实验报告

课程：嵌入式系统A

第五次实验

**姓 名 凌智城**

**学 号 201806061211**

**专业班级 通信工程1803班**

**老 师 黄国兴**

**学 院 信息工程学院**

**提交日期**  **2021年6月3日**

实验8：RTC时钟驱动实验

1. 实验目的
2. 了解RTC工作原理；
3. 掌握RTC驱动的编写；
4. 掌握RTC的加载过程及测试方法。
5. 实验内容
6. 学习RTC的工作原理；
7. 编写RTC的驱动程序；
8. 编写测试程序测试RTC。
9. 实验步骤

**步骤1：**建文件夹，编写RTC驱动程序（在服务器或虚拟机终端运行）

创建rtc文件夹，并将服务器中的RTC文件夹复制过来。

服务器：将服务器上/shiyan/2021/code文件夹下的 RTC 目录复制到自己用户目录下。

虚拟机：将虚拟机里/Desktop/shiyan/code文件下的 RTC 目录复制到自己用户目录下

# mkdir rtc

# cd rtc

虚拟机：# cp -r /home/shiyan/Desktop/shiyan/code/RTC/ ./

服务器：# cp -r /home/shiyan/2021/code/RTC ./

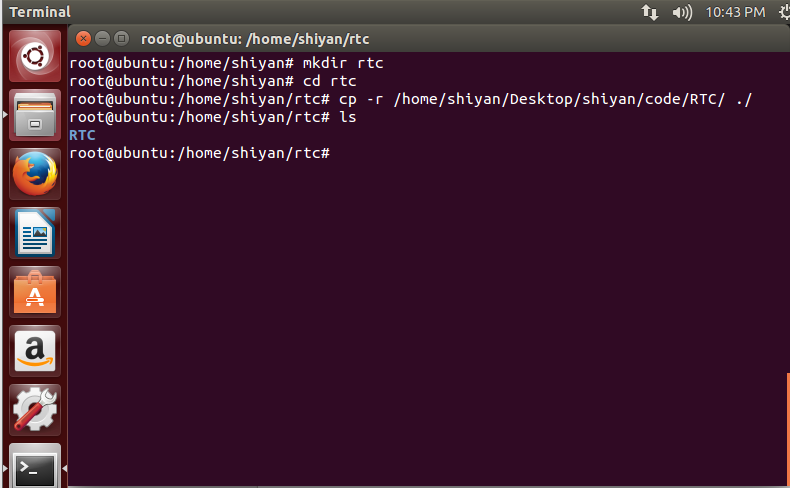


图8-1 RTC目录复制到自己用用户目录

**步骤2：**编写用于交叉编译的Makefile

|  |
| --- |
| KDIR:=/home/stx/ kernel-for-mceb //此处路径修改为虚拟机中存放内核文件的目录  CROSS\_COMPILE = arm\_v5t\_le  CC = $(CROSS\_COMPILE)gcc  .PHONY: modules clean  obj-m := rtc-x1205.o  modules:  make -C $(KDIR) M=`pwd` modules  clean:  make -C $(KDIR) M=`pwd` modules clean |

管理员权限下退到根目录执行source /etc/profile使环境变量生效

进入RTC文件夹执行make命令，生成rtc-x1205.ko文件；

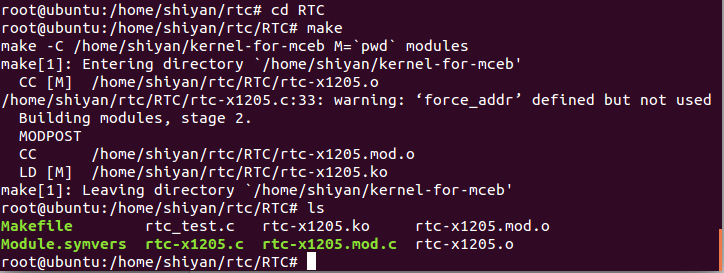


图8-2 生成rtc-x1205.ko文件

将该文件拷贝到所挂载的文件系统filesys\_test中（根据自己的目录改路径）

虚拟机：# cp rtc-x1205.ko /home/shiyan/share/filesys\_test/modules

服务器：# cp rtc-x1205.ko /home/stX/filesys\_test/modules



图8-3 将rtc-下205.ko拷贝到所挂载的文件系统

**步骤3：**编写测试程序

编写测试程序rtc\_test.c，将编写好的测试程序在服务器上输入命令

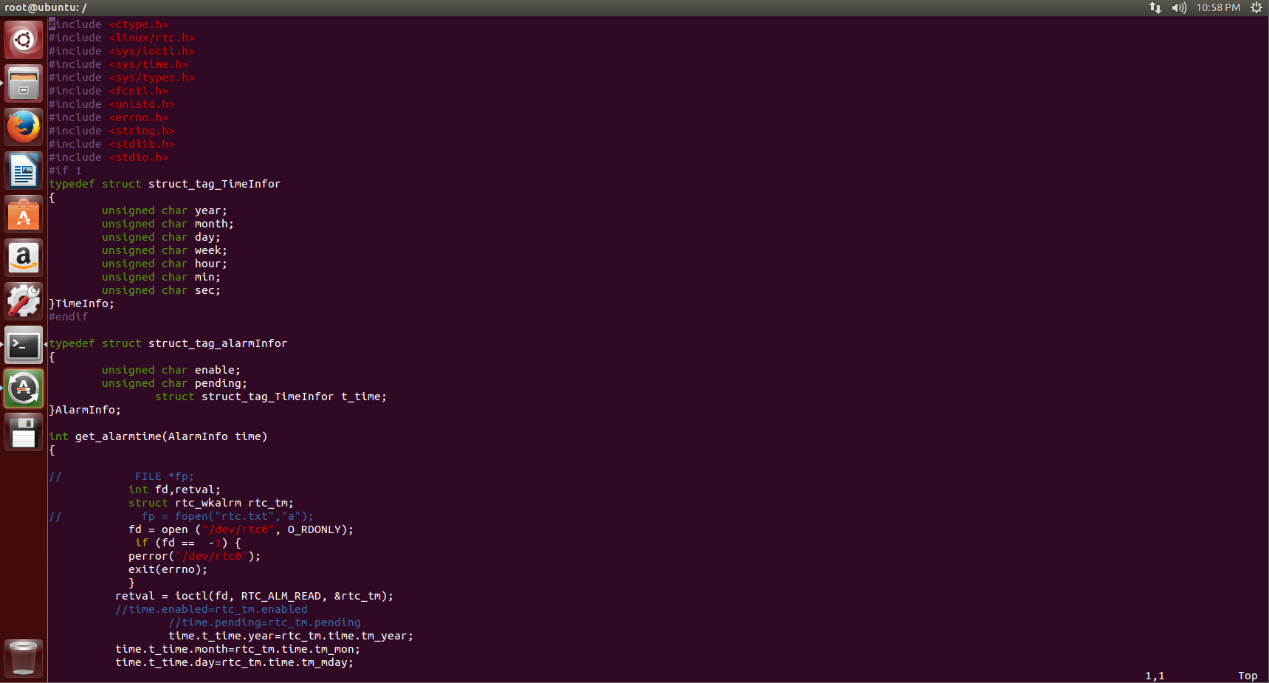


图8-4 编写测试程序rtc\_test.c

# arm\_v5t\_le-gcc -o rtc\_test rtc\_test.c

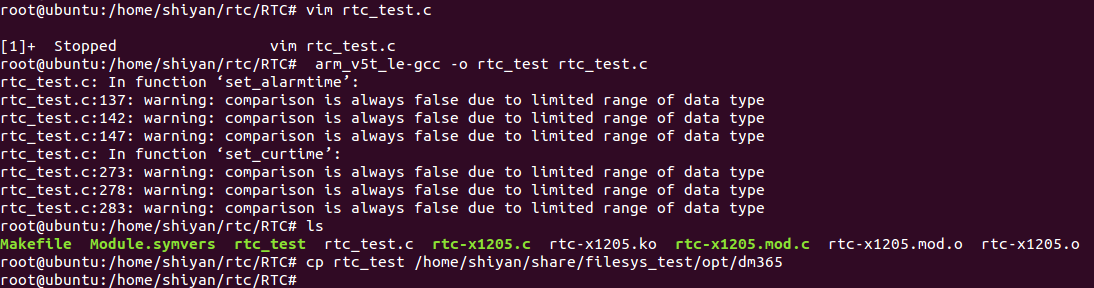


图8-5 生成可执行文件rtc\_test并复制到挂载的文件系统

将生成可执行文件rtc\_test，将其放到文件系统filesys\_test中：

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

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

**步骤4：**挂载文件系统，设置启动参数通过NFS方式挂载实验箱根文件系统。

打开putty，启动实验箱，在内核启动倒计时5s内按enter终止实验箱的启动，输入参数挂载文件系统（参考挂载实验），然后输入boot引导启动。

**步骤5：**手动加载驱动

启动完成后，输入root登录板子，进入到/modules目录，在使用insmod rtc-x1205.ko加载RTC驱动模块，如下图所示：

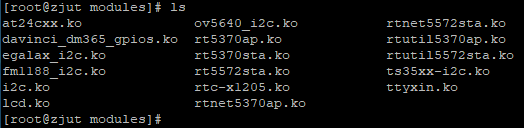


图8-6 进入modules目录

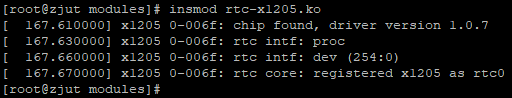


图8-7 加载RTC驱动模块

**步骤6：**查找自己的测试程序

使用命令cd/opt/dm365进入测试程序所在目录，找到自己的测试程序，如下图所示：

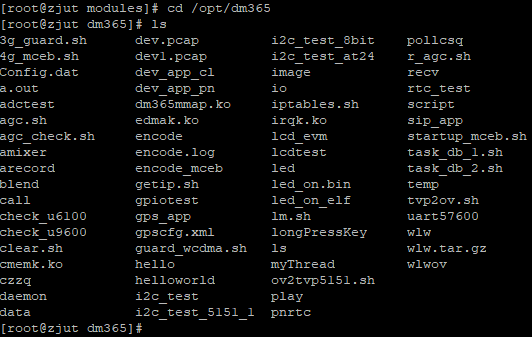


图8-8 查找自己的测试程序rtc\_test

**步骤7：**执行测试程序rtc\_test

执行测试程序./rtc\_test，根据提示输入1读取当前时间，输入2根据提示设 置时间，（先设置年，设置完后换行设置月，依次设置完，最后要多输入一个整 数以示完成输入），输入3读取闹钟时间。

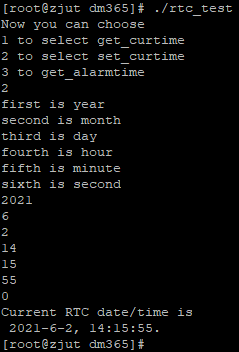


图8-9 select set\_certime

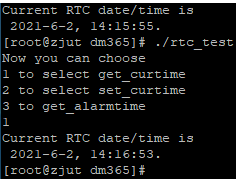


图8-10 select get\_certime

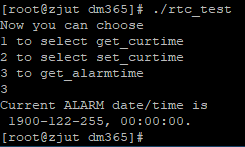


图8-11 get\_alarmtime

**步骤8：**设置系统时间并写入硬件

可以任意设置RTC时间，首先使用date xxxxxxxxxxxx（格式：月日时分年） 设置系统时间，然后使用命令hwclock -w把系统时间写入硬件RTC，最后使用命令hwclock -r读取RTC时间。

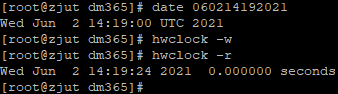


图8-12 设置系统时间并读入硬件

1. 心得与体会

通过对RTC驱动程序的编写、加载和调试，对RTC模块有了更加深入的了解，同时也对嵌入式驱动开发有了更为深入的了解，另一方面在驱动程序的编写也让我对嵌入式软件编程有了很大的锻炼，在前几次实验的基础上，熟悉了挂载文件系统的流程，更加熟悉了终端操作命令。

1. 附录

RTC驱动程序源码rtc-x1205.c：

|  |
| --- |
| /\*  \* An i2c driver for the Xicor/Intersil X1205 RTC  \* Copyright 2004 Karen Spearel \* Copyright 2005 Alessandro Zummo  \*  \* please send all reports to:  \* Karen Spearel <kas111@gmail.com>  \* Alessandro Zummo <a.zummo@towertech.it>  \*  \* based on a lot of other RTC drivers.  \*  \* This program is free software; you can redistribute it and/or modify  \* it under the terms of the GNU General Public License version 2 as  \* published by the Free Software Foundation.  \*/  #include <linux/module.h>  #include <linux/i2c.h>  #include <linux/bcd.h>  #include <linux/rtc.h>  #include <linux/delay.h>  #define DRV\_VERSION "1.0.7"  /\* Addresses to scan: none. This chip is located at  \* 0x6f and uses a two bytes register addressing.  \* Two bytes need to be written to read a single register,  \* while most other chips just require one and take the second  \* one as the data to be written. To prevent corrupting  \* unknown chips, the user must explicitely set the probe parameter.  \*/  static struct i2c\_driver x1205\_driver;  static unsigned short normal\_i2c[] = { 0x6f, I2C\_CLIENT\_END}; //zbs tianjia 0x6f;  static unsigned short force\_addr[] = {ANY\_I2C\_BUS, 0x6f, I2C\_CLIENT\_END};//zbs  //static unsigned short \*force[] = {force\_addr, NULL};//zbs  //static unsigned short ignore[] = {I2C\_CLIENT\_END};//zbs  /\*static struct i2c\_client\_address\_data addr\_data =  {  .normal\_i2c = normal\_i2c,  .probe = ignore,  .ignore = ignore,  //.forces = forces,  }; /zbs \*/  /\* Insmod parameters \*/  I2C\_CLIENT\_INSMOD; |
| /\* offsets into CCR area \*/  #define CCR\_SEC 0  #define CCR\_MIN 1  #define CCR\_HOUR 2  #define CCR\_MDAY 3  #define CCR\_MONTH 4  #define CCR\_YEAR 5  #define CCR\_WDAY 6  #define CCR\_Y2K 7  #define X1205\_REG\_SR 0x3F /\* status register \*/  #define X1205\_REG\_Y2K 0x37  #define X1205\_REG\_DW 0x36  #define X1205\_REG\_YR 0x35  #define X1205\_REG\_MO 0x34  #define X1205\_REG\_DT 0x33  #define X1205\_REG\_HR 0x32  #define X1205\_REG\_MN 0x31  #define X1205\_REG\_SC 0x30  #define X1205\_REG\_DTR 0x13  #define X1205\_REG\_ATR 0x12  #define X1205\_REG\_INT 0x11  #define X1205\_REG\_0 0x10  #define X1205\_REG\_Y2K1 0x0F  #define X1205\_REG\_DWA1 0x0E  #define X1205\_REG\_YRA1 0x0D  #define X1205\_REG\_MOA1 0x0C  #define X1205\_REG\_DTA1 0x0B  #define X1205\_REG\_HRA1 0x0A  #define X1205\_REG\_MNA1 0x09  #define X1205\_REG\_SCA1 0x08  #define X1205\_REG\_Y2K0 0x07  #define X1205\_REG\_DWA0 0x06  #define X1205\_REG\_YRA0 0x05  #define X1205\_REG\_MOA0 0x04  #define X1205\_REG\_DTA0 0x03  #define X1205\_REG\_HRA0 0x02  #define X1205\_REG\_MNA0 0x01  #define X1205\_REG\_SCA0 0x00  #define X1205\_CCR\_BASE 0x30 /\* Base address ofCCR \*/  #define X1205\_ALM0\_BASE 0x00 /\* Base address of ALARM0 \*/  #define X1205\_SR\_RTCF 0x01 /\* Clock failure \*/  #define X1205\_SR\_WEL 0x02 /\* Write Enable Latch \*/  #define X1205\_SR\_RWEL 0x04 /\* Register Write Enable \*/  #define X1205\_DTR\_DTR0 0x01  #define X1205\_DTR\_DTR1 0x02  #define X1205\_DTR\_DTR2 0x04  #define X1205\_HR\_MIL 0x80 /\* Set in ccr.hour for 24 hr mode \*/  /\* Prototypes \*/  static int x1205\_attach(struct i2c\_adapter \*adapter);  static int x1205\_detach(struct i2c\_client \*client);  static int x1205\_probe(struct i2c\_adapter \*adapter, int address, int kind);  static struct i2c\_driver x1205\_driver = {  .driver= {  .name = "x1205",  },  .id=I2C\_DRIVERID\_X1205,  .attach\_adapter = &x1205\_attach,  .detach\_client = &x1205\_detach,  };  /\*  \* In the routines that deal directly with the x1205 hardware, we use  \* rtc\_time -- month 0-11, hour 0-23, yr = calendar year-epoch  \* Epoch is initialized as 2000. Time is set to UTC.  \*/  static int x1205\_get\_datetime(struct i2c\_client \*client, struct rtc\_time \*tm,  unsigned char reg\_base)  {  unsigned char dt\_addr[2] = { 0, reg\_base };  unsigned char buf[8];  struct i2c\_msg msgs[] = {  { client->addr, 0, 2, dt\_addr }, /\* setup read ptr \*/  { client->addr, I2C\_M\_RD, 8, buf }, /\* read date \*/  };  /\* read date registers \*/  if ((i2c\_transfer(client->adapter, &msgs[0], 2)) != 2) {  dev\_err(&client->dev, "%s: read error\n", \_\_FUNCTION\_\_);  return -EIO;  }  dev\_dbg(&client->dev,  "%s: raw read data - sec=%02x, min=%02x, hr=%02x, "  "mday=%02x, mon=%02x, year=%02x, wday=%02x, y2k=%02x\n",  \_\_FUNCTION\_\_,  buf[0], buf[1], buf[2], buf[3],  buf[4], buf[5], buf[6], buf[7]);  tm->tm\_sec = BCD2BIN(buf[CCR\_SEC]);  tm->tm\_min = BCD2BIN(buf[CCR\_MIN]);  tm->tm\_hour = BCD2BIN(buf[CCR\_HOUR] & 0x3F);/\* hr is 0-23 \*/  tm->tm\_mday = BCD2BIN(buf[CCR\_MDAY]);  tm->tm\_mon = BCD2BIN(buf[CCR\_MONTH]) - 1; /\* mon is 0-11 \*/  tm->tm\_year = BCD2BIN(buf[CCR\_YEAR])  + (BCD2BIN(buf[CCR\_Y2K]) \* 100) - 1900;  tm->tm\_wday = buf[CCR\_WDAY];  dev\_dbg(&client->dev, "%s: tm is secs=%d, mins=%d, hours=%d, "  "mday=%d, mon=%d, year=%d, wday=%d\n",  \_\_FUNCTION\_\_,  tm->tm\_sec, tm->tm\_min, tm->tm\_hour,  tm->tm\_mday, tm->tm\_mon, tm->tm\_year, tm->tm\_wday);  return 0;  }  static int x1205\_get\_status(struct i2c\_client \*client, unsigned char \*sr)  {  static unsigned char sr\_addr[2] = { 0, X1205\_REG\_SR };  struct i2c\_msg msgs[] = {  { client->addr, 0, 2, sr\_addr }, /\* setup read ptr \*/  { client->addr, I2C\_M\_RD, 1, sr },  };  /\* read status register \*/  if ((i2c\_transfer(client->adapter, &msgs[0], 2)) != 2) {  dev\_err(&client->dev, "%s: read error\n", \_\_FUNCTION\_\_);  return -EIO;  }  return 0;  }  static int x1205\_set\_datetime(struct i2c\_client \*client, struct rtc\_time \*tm,  int datetoo, u8 reg\_base)  {  int i, xfer;  unsigned char buf[8];  static const unsigned char wel[3] = { 0, X1205\_REG\_SR,  X1205\_SR\_WEL };  static const unsigned char rwel[3] = { 0, X1205\_REG\_SR,  X1205\_SR\_WEL | X1205\_SR\_RWEL };  static const unsigned char diswe[3] = { 0, X1205\_REG\_SR, 0 };  dev\_dbg(&client->dev,  "%s: secs=%d, mins=%d, hours=%d\n",  \_\_FUNCTION\_\_,  tm->tm\_sec, tm->tm\_min, tm->tm\_hour);  buf[CCR\_SEC] = BIN2BCD(tm->tm\_sec);  buf[CCR\_MIN] = BIN2BCD(tm->tm\_min);  /\* set hour and 24hr bit \*/  buf[CCR\_HOUR] = BIN2BCD(tm->tm\_hour) | X1205\_HR\_MIL;  /\* should we also set the date? \*/  if (datetoo) {  dev\_dbg(&client->dev,  "%s: mday=%d, mon=%d, year=%d, wday=%d\n",  \_\_FUNCTION\_\_,  tm->tm\_mday, tm->tm\_mon, tm->tm\_year, tm->tm\_wday);  buf[CCR\_MDAY] = BIN2BCD(tm->tm\_mday);  /\* month, 1 - 12 \*/  buf[CCR\_MONTH] = BIN2BCD(tm->tm\_mon + 1);  /\* year, since the rtc epoch\*/  buf[CCR\_YEAR] = BIN2BCD(tm->tm\_year % 100);  buf[CCR\_WDAY] = tm->tm\_wday & 0x07;  buf[CCR\_Y2K] = BIN2BCD(tm->tm\_year / 100);  }  /\* this sequence is required to unlock the chip \*/  if ((xfer = i2c\_master\_send(client, wel, 3)) != 3) {  dev\_err(&client->dev, "%s: wel - %d\n", \_\_FUNCTION\_\_, xfer);  return -EIO;  }  if ((xfer = i2c\_master\_send(client, rwel, 3)) != 3) {  dev\_err(&client->dev, "%s: rwel - %d\n", \_\_FUNCTION\_\_, xfer);  return -EIO;  }  /\* write register's data \*/  for (i = 0; i < (datetoo ? 8 : 3); i++) {  unsigned char rdata[3] = { 0, reg\_base + i, buf[i] };  xfer = i2c\_master\_send(client, rdata, 3);  if (xfer != 3) {  dev\_err(&client->dev,  "%s: xfer=%d addr=%02x, data=%02x\n",  \_\_FUNCTION\_\_,  xfer, rdata[1], rdata[2]);  return -EIO;  }  };  /\* disable further writes \*/  if ((xfer = i2c\_master\_send(client, diswe, 3)) != 3) {  dev\_err(&client->dev, "%s: diswe - %d\n", \_\_FUNCTION\_\_, xfer);  return -EIO  }  return 0;  }  static int x1205\_fix\_osc(struct i2c\_client \*client)  {  int err;  struct rtc\_time tm;  tm.tm\_hour = tm.tm\_min = tm.tm\_sec = 0;  if ((err = x1205\_set\_datetime(client, &tm, 0, X1205\_CCR\_BASE)) < 0)  dev\_err(&client->dev,  "unable to restart the oscillator\n");  return err;  }  static int x1205\_get\_dtrim(struct i2c\_client \*client, int \*trim)  {  unsigned char dtr; static unsigned char dtr\_addr[2] = { 0, X1205\_REG\_DTR };  struct i2c\_msg msgs[] = {  { client->addr, 0, 2, dtr\_addr }, /\* setup read ptr \*/  { client->addr, I2C\_M\_RD, 1, &dtr }, /\* read dtr \*/  };  /\* read dtr register \*/  if ((i2c\_transfer(client->adapter, &msgs[0], 2)) != 2) {  dev\_err(&client->dev, "%s: read error\n", \_\_FUNCTION\_\_);  return -EIO;  }  dev\_dbg(&client->dev, "%s: raw dtr=%x\n", \_\_FUNCTION\_\_, dtr);  \*trim = 0;  if (dtr & X1205\_DTR\_DTR0)  \*trim += 20;  if (dtr & X1205\_DTR\_DTR1)  \*trim += 10;  if (dtr & X1205\_DTR\_DTR2)  \*trim = -\*trim;  return 0;  }  static int x1205\_get\_atrim(struct i2c\_client \*client, int \*trim)  {  s8 atr;  static unsigned char atr\_addr[2] = { 0, X1205\_REG\_ATR };  struct i2c\_msg msgs[] = {  { client->addr, 0, 2, atr\_addr }, /\* setup read ptr \*/  { client->addr, I2C\_M\_RD, 1, &atr }, /\* read atr \*/  };  /\* read atr register \*/  if ((i2c\_transfer(client->adapter, &msgs[0], 2)) != 2) {  dev\_err(&client->dev, "%s: read error\n", \_\_FUNCTION\_\_);  return -EIO;  }  dev\_dbg(&client->dev, "%s: raw atr=%x\n", \_\_FUNCTION\_\_, atr);  /\* atr is a two's complement value on 6 bits,  \* perform sign extension. The formula is  \* Catr = (atr \* 0.25pF) + 11.00pF.  \*/  if (atr & 0x20)  atr |= 0xC0;  dev\_dbg(&client->dev, "%s: raw atr=%x (%d)\n", \_\_FUNCTION\_\_, atr, atr);  \*trim = (atr \* 250) + 11000;  dev\_dbg(&client->dev, "%s: real=%d\n", \_\_FUNCTION\_\_, \*trim);  return 0;  }  struct x1205\_limit  {  unsigned char reg, mask, min, max;  };  static int x1205\_validate\_client(struct i2c\_client \*client)  {  int i, xfer;  /\* Probe array. We will read the register at the specified  \* address and check if the given bits are zero.  \*/  static const unsigned char probe\_zero\_pattern[] = {  /\* register, mask \*/  X1205\_REG\_SR, 0x18,  X1205\_REG\_DTR, 0xF8,  X1205\_REG\_ATR, 0xC0,  X1205\_REG\_INT, 0x18,  X1205\_REG\_0, 0xFF,  };  static const struct x1205\_limit probe\_limits\_pattern[] = {  /\* register, mask, min, max \*/  { X1205\_REG\_Y2K, 0xFF, 19, 20 },  { X1205\_REG\_DW, 0xFF, 0, 6},  { X1205\_REG\_YR, 0xFF, 0, 99},  { X1205\_REG\_MO, 0xFF, 0, 12},  { X1205\_REG\_DT, 0xFF, 0, 31},  { X1205\_REG\_HR, 0x7F, 0, 23},  { X1205\_REG\_MN, 0xFF, 0, 59},  { X1205\_REG\_SC, 0xFF, 0, 59},  { X1205\_REG\_Y2K1, 0xFF, 19, 20 },  { X1205\_REG\_Y2K0, 0xFF, 19, 20 },  };  /\* check that registers have bits a 0 where expected \*/  for (i = 0; i < ARRAY\_SIZE(probe\_zero\_pattern); i += 2) {  unsigned char buf;  unsigned char addr[2] = { 0, probe\_zero\_pattern[i] };  struct i2c\_msg msgs[2] = {  {client->addr,0,2,addr },  { client->addr, I2C\_M\_RD, 1, &buf },  };  if ((xfer = i2c\_transfer(client->adapter, msgs, 2)) != 2)  { dev\_err(&client->adapter->dev,  "%s: could not read register %x\n",  \_\_FUNCTION\_\_, probe\_zero\_pattern[i]);  return -EIO;  }  if ((buf& probe\_zero\_pattern[i+1]) != 0) {  dev\_err(&client->adapter->dev,  "%s: register=%02x, zero pattern=%d, value=%x\n",  \_\_FUNCTION\_\_, probe\_zero\_pattern[i], i, buf);  return -ENODEV;  }  }  /\* check limits (only registers with bcd values) \*/  for (i = 0; i < ARRAY\_SIZE(probe\_limits\_pattern); i++) {  unsigned char reg, value;  unsigned char addr[2] = { 0, probe\_limits\_pattern[i].reg };  struct i2c\_msg msgs[2] = {  {client->addr,0,2,addr },  { client->addr, I2C\_M\_RD, 1, &reg },  };  if ((xfer = i2c\_transfer(client->adapter, msgs, 2)) != 2) {  dev\_err(&client->adapter->dev,  "%s: could not read register %x\n",  \_\_FUNCTION\_\_, probe\_limits\_pattern[i].reg);  return -EIO;  }  value = BCD2BIN(reg & probe\_limits\_pattern[i].mask);  if (value > probe\_limits\_pattern[i].max ||  value < probe\_limits\_pattern[i].min) {  dev\_dbg(&client->adapter->dev,  "%s: register=%x, lim pattern=%d, value=%d\n",  \_\_FUNCTION\_\_, probe\_limits\_pattern[i].reg,  i, value);  return -ENODEV;  }  }  return 0;  }  static int x1205\_rtc\_read\_alarm(struct device \*dev, struct rtc\_wkalrm \*alrm)  {  return x1205\_get\_datetime(to\_i2c\_client(dev),  &alrm->time, X1205\_ALM0\_BASE);  }  static int x1205\_rtc\_set\_alarm(struct device \*dev, struct rtc\_wkalrm \*alrm)  {  return x1205\_set\_datetime(to\_i2c\_client(dev),  &alrm->time, 1, X1205\_ALM0\_BASE);  }  static int x1205\_rtc\_read\_time(struct device \*dev, struct rtc\_time \*tm)  {  return x1205\_get\_datetime(to\_i2c\_client(dev),  tm, X1205\_CCR\_BASE);  }  static int x1205\_rtc\_set\_time(struct device \*dev, struct rtc\_time \*tm)  {  return x1205\_set\_datetime(to\_i2c\_client(dev),  tm, 1, X1205\_CCR\_BASE);  }  static int x1205\_rtc\_proc(struct device \*dev, struct seq\_file \*seq)  {  int err, dtrim, atrim;  if ((err = x1205\_get\_dtrim(to\_i2c\_client(dev), &dtrim)) == 0)  seq\_printf(seq, "digital\_trim\t: %d ppm\n", dtrim);  if ((err = x1205\_get\_atrim(to\_i2c\_client(dev), &atrim)) == 0)  seq\_printf(seq, "analog\_trim\t: %d.%02d pF\n",  atrim / 1000, atrim % 1000);  return 0;  }  static struct rtc\_class\_ops x1205\_rtc\_ops = {  .proc = x1205\_rtc\_proc,  .read\_time = x1205\_rtc\_read\_time,  .set\_time = x1205\_rtc\_set\_time,  .read\_alarm = x1205\_rtc\_read\_alarm,  .set\_alarm = x1205\_rtc\_set\_alarm,  };  static ssize\_t x1205\_sysfs\_show\_atrim(struct device \*dev,  struct device\_attribute \*attr, char \*buf)  {  int err, atrim;  err = x1205\_get\_atrim(to\_i2c\_client(dev), &atrim);  if (err)  return err;  return sprintf(buf, "%d.%02d pF\n", atrim / 1000, atrim % 1000);  }  static DEVICE\_ATTR(atrim, S\_IRUGO, x1205\_sysfs\_show\_atrim, NULL);  static ssize\_t x1205\_sysfs\_show\_dtrim(struct device \*dev,  struct device\_attribute \*attr, char \*buf)  {  int err, dtrim;  err = x1205\_get\_dtrim(to\_i2c\_client(dev), &dtrim);  if (err)  return err;  return sprintf(buf, "%d ppm\n", dtrim);  }  static DEVICE\_ATTR(dtrim, S\_IRUGO, x1205\_sysfs\_show\_dtrim, NULL);  static int x1205\_attach(struct i2c\_adapter \*adapter)  {  return i2c\_probe(adapter, &addr\_data, x1205\_probe);  }  static int x1205\_probe(struct i2c\_adapter \*adapter, int address, int kind)  {  int err = 0;  unsigned char sr;  struct i2c\_client \*client;  struct rtc\_device \*rtc;  dev\_dbg(&adapter->dev, "%s\n", \_\_FUNCTION\_\_);  if (!i2c\_check\_functionality(adapter, I2C\_FUNC\_I2C)) {  err = -ENODEV;  goto exit;  }  if (!(client = kzalloc(sizeof(struct i2c\_client), GFP\_KERNEL))) {  err = -ENOMEM;  goto exit;  }  /\* I2C client \*/  client->addr = address;  client->driver = &x1205\_driver;  client->adapter = adapter;  strlcpy(client->name, x1205\_driver.driver.name, I2C\_NAME\_SIZE);  /\* Verify the chip is really an X1205 \*/  if (kind < 0) {  if (x1205\_validate\_client(client) < 0) {  err = -ENODEV;  goto exit\_kfree;  }  }  /\* Inform the i2c layer \*/  if ((err = i2c\_attach\_client(client)))  goto exit\_kfree;  dev\_info(&client->dev, "chip found, driver version " DRV\_VERSION "\n");  rtc = rtc\_device\_register(x1205\_driver.driver.name, &client->dev,  &x1205\_rtc\_ops, THIS\_MODULE);  if (IS\_ERR(rtc)) {  err = PTR\_ERR(rtc);  goto exit\_detach;  }  i2c\_set\_clientdata(client, rtc);  /\* Check for power failures and eventualy enable the osc \*/  if ((err = x1205\_get\_status(client, &sr)) == 0) {  if (sr & X1205\_SR\_RTCF) {  dev\_err(&client->dev,  "power failure detected, "  "please set the clock\n");  udelay(50);  x1205\_fix\_osc(client);  }  }  else  dev\_err(&client->dev, "couldn't read status\n");  device\_create\_file(&client->dev, &dev\_attr\_atrim);  device\_create\_file(&client->dev, &dev\_attr\_dtrim);  return 0;  exit\_detach:  i2c\_detach\_client(client);  exit\_kfree:  kfree(client);  exit:  return err;  }  static int x1205\_detach(struct i2c\_client \*client)  {  int err;  struct rtc\_device \*rtc = i2c\_get\_clientdata(client);  if (rtc)  rtc\_device\_unregister(rtc);  if ((err = i2c\_detach\_client(client)))  return err;  kfree(client);  return 0;  }  static int \_\_init x1205\_init(void)  {  return i2c\_add\_driver(&x1205\_driver);  }  static void \_\_exit x1205\_exit(void)  {  i2c\_del\_driver(&x1205\_driver);  }  MODULE\_LICENSE("GPL");  module\_init(x1205\_init);  module\_exit(x1205\_exit); |

测试程序rtc\_test.c

|  |
| --- |
| #include <ctype.h>  #include <linux/rtc.h>  #include <sys/ioctl.h>  #include <sys/time.h>  #include <sys/types.h>  #include <fcntl.h>  #include <unistd.h>  #include <errno.h>  #include <string.h>  #include <stdlib.h>  #include <stdio.h>  #if 1  typedef struct struct\_tag\_TimeInfor  {  unsigned char year;  unsigned char month;  unsigned char day;  unsigned char week;  unsigned char hour;  unsigned char min;  unsigned char sec;  }TimeInfo;  #endif  typedef struct struct\_tag\_alarmInfor  {  unsigned char enable;  unsigned char pending;  struct struct\_tag\_TimeInfor t\_time;  }AlarmInfo;  int get\_alarmtime(AlarmInfo time)  {  // FILE \*fp;  int fd,retval;  struct rtc\_wkalrm rtc\_tm;  //  fp = fopen("rtc.txt","a");  fd = open ("/dev/rtc0", O\_RDONLY);  if (fd == -1) {  perror("/dev/rtc0");  exit(errno);  }  retval = ioctl(fd, RTC\_ALM\_READ, &rtc\_tm);  //time.enabled=rtc\_tm.enabled  //time.pending=rtc\_tm.pending  time.t\_time.year=rtc\_tm.time.tm\_year;  time.t\_time.month=rtc\_tm.time.tm\_mon;  time.t\_time.day=rtc\_tm.time.tm\_mday;  time.t\_time.week=rtc\_tm.time.tm\_wday;  time.t\_time.hour=rtc\_tm.time.tm\_hour;  time.t\_time.min=rtc\_tm.time.tm\_min;  time.t\_time.sec=rtc\_tm.time.tm\_sec;  fprintf(stdout,"CurrentALARMdate/timeis\n %d-%d-%d, %02d:%02d:%02d.\n",time.t\_time.year + 1900,  time.t\_time.day,time.t\_time.hour, time.t\_time.min, time.t\_time.sec);  close(fd);  return 0;  }  int set\_alarmtime(AlarmInfo time)  {  int i,fd,retval;  int b[6];  char \*rtime[6];  char \*message[]={"first is year\n"  "second is month\n"  "third is day\n"  "fourth is hour\n"  "fifth is minute\n"  "sixth is second\n"};  struct rtc\_wkalrm alarm;  fd = open ("/dev/rtc0", O\_RDWR);  if (fd == -1)  {  perror("/dev/rtc0");  exit(errno);  }  fprintf(stdout,\*message);  for(i=0;i<6;i++)  {  rtime[i]=(char \*)malloc(sizeof(char)\*10);  scanf("%s\n \n",rtime[i]);  }  for(i=0;i<6;i++)  b[i]=atoi(rtime[i]);  time.t\_time.year=b[0]-1900; // year  time.t\_time.month=b[1]-1; // month  time.t\_time.day=b[2]; // day  time.t\_time.hour=b[3]; // hour  time.t\_time.min=b[4]; // minute  time.t\_time.sec=b[5]; // second  /\* if(time.t\_time.year<2000||time.t\_time.year>2099)  {  printf("Please input the correct year, the year should be in the scope of 2000~2099!\n");  exit(1);  } \*/  if(time.t\_time.month<1||time.t\_time.month>12)  {  printf("Please input the correct mouth, the mouth should be in the scope of 1~12!\n");  exit(1);  }  if(time.t\_time.day<1||time.t\_time.day>31)  {  printf("Please input the correct days, the days should be in the scope of 1~31!\n");  exit(1);  }  else  if(time.t\_time.month==4||time.t\_time.month==6||time.t\_time.month==9||time.t\_time.month==11)  if(time.t\_time.day<1||time.t\_time.day>30)  {  printf("Please input the correct days, the days should be in the scope of 1~30!\n");  exit(1);  }  }  else if(time.t\_time.month==2)  {  if((time.t\_time.year%4==0)&&(time.t\_time.year%100!=0)||(time.t\_time.year%400==0))  { if(time.t\_time.day<1||time.t\_time.day>29) {  printf("Please input the correct days, the days should be in the scope of 1~29!\n"); exit(1); } } else {  if(time.t\_time.day<1||time.t\_time.day>28) {  printf("Please input the correct days, the days should be in the scope of 1~28!\n"); exit(1); } } }  if(time.t\_time.hour<0||time.t\_time.hour>23) {  printf("Please input the correct hour, the hour should be in the scope of 0~23!\n"); exit(1); }  if(time.t\_time.min<0||time.t\_time.min>59) {  printf("Please input the correct minute, the minute should be in the scope of 0~60!\n"); exit(1); }  if(time.t\_time.sec<0||time.t\_time.sec>59) {  printf("Please input the correct second, the second should be in the scope of 0~60!\n");  exit(1);  }  alarm.time.tm\_year=time.t\_time.year;  alarm.time.tm\_mon=time.t\_time.month;  alarm.time.tm\_mday=time.t\_time.day;  alarm.time.tm\_hour=time.t\_time.hour;  alarm.time.tm\_min=time.t\_time.min;  alarm.time.tm\_sec=time.t\_time.sec;  retval = ioctl(fd, RTC\_ALM\_SET, &alarm);  if (retval == -1)  {  perror("ioctl");  exit(errno);  }  close(fd);  // system("/bin/busybox hwclock --hctosys");  return 0;  }  int get\_curtime(TimeInfo time)  {  // FILE \*fp;  int fd,retval;  struct rtc\_time rtc\_tm;  // fp = fopen("rtc.txt","a");  fd = open ("/dev/rtc0", O\_RDONLY);  if (fd == -1) {  perror("/dev/rtc0");  exit(errno);  }  retval = ioctl(fd, RTC\_RD\_TIME, &rtc\_tm);  time.year=rtc\_tm.tm\_year;  time.month=rtc\_tm.tm\_mon;  time.day=rtc\_tm.tm\_mday;  time.week=rtc\_tm.tm\_wday;  time.hour=rtc\_tm.tm\_hour;  time.min=rtc\_tm.tm\_min;  time.sec=rtc\_tm.tm\_sec;  fprintf(stdout, "Current RTC date/time is \n %d-%d-%d, %02d:%02d:%02d.\n",  time.year + 1900, time.month + 1, time.day,time.hour, time.min,  time.sec);  close(fd);  return 0;  }  int set\_curtime(TimeInfo time)  {  int i,fd,retval;  int b[6];  char \*rtime[6];  char \*message[]={"first is year\n"  "second is month\n"  "third is day\n"  "fourth is hour\n"  "fifth is minute\n"  "sixth is second\n"};  struct rtc\_time rtc\_tm;  fd = open ("/dev/rtc0", O\_RDWR);  if (fd == -1)  {  perror("/dev/rtc0");  exit(errno);  }  fprintf(stdout,\*message);  for(i=0;i<6;i++)  {  rtime[i]=(char \*)malloc(sizeof(char)\*10);  scanf("%s\n \n",rtime[i]);  }  for(i=0;i<6;i++)  b[i]=atoi(rtime[i]);  time.year=b[0]-1900; // year  time.month=b[1]-1; // month  time.day=b[2]; // day  time.hour=b[3]; // hour  time.min=b[4]; // minute  time.sec=b[5]; // seconde  /\* if(time.year<2000||time.year>2099)  {  printf("Please input the correct year, the year should be in the scope of 2000~2099!\n");  exit(1);  }  \*/  if(time.month<1||time.month>12)  {  printf("Please input the correct mouth, the mouth should be in the scope of 1~12!\n");  exit(1);  }  if(time.day<1||time.day>31)  {  printf("Please input the correct days, the days should be in the scope of 1~31!\n");  exit(1);  }  else if(time.month==4||time.month==6||time.month==9||time.month==11)  { if(time.day<1||time.day>30) {  printf("Please input the correct days, the days should be in the scope of 1~30!\n"); exit(1);  } }  else if(time.month==2) {  if((time.year%4==0)&&(time.year%100!=0)||(time.year%400==0)) {  if(time.day<1||time.day>29) {  printf("Please input the correct days, the days should be in the scope of 1~29!\n"); exit(1); } } else {  if(time.day<1||time.day>28) {  printf("Please input the correct days, the days should be in the scope of 1~28!\n"); exit(1); } } }  if(time.hour<0||time.hour>23) {  printf("Please input the correct hour, the hour should be in the scope of 0~23!\n"); exit(1); }  if(time.min<0||time.min>59) {  printf("Please input the correct minute, the minute should be in the scope of 0~60!\n");  exit(1);  } if(time.sec<0||time.sec>59) {  printf("Please input the correct second, the second should be in the scope of 0~60!\n");  exit(1); }  rtc\_tm.tm\_year=time.year;  rtc\_tm.tm\_mon=time.month;  rtc\_tm.tm\_mday=time.day;  rtc\_tm.tm\_hour=time.hour;  rtc\_tm.tm\_min=time.min;  rtc\_tm.tm\_sec=time.sec; retval = ioctl(fd, RTC\_SET\_TIME, &rtc\_tm);  if (retval == -1)  {  perror("ioctl");  exit(errno);  }  close(fd);  // system(“/bin/busybox hwclock --hctosys”);  return 0;  }  int main()  {  TimeInfo p;  AlarmInfo a;  int choice;  fprintf(stdout,"Now you can choose\n1 to select get\_curtime\n2 to select set\_curtime\n3 to  get\_alarmtime\n");  //fprintf(stdout,"input any digital without zero to get time\n");  //fprintf(stdout,"Your will get time:\n");  scanf("%d",&choice);  switch(choice)  {  case 1 :  // set\_curtime(p);  get\_curtime(p);  break;  case 2 :  set\_curtime(p);  get\_curtime(p);  break;  case 3 :  get\_alarmtime(a);  break;  /\* case 4:  set\_curtime(p);  get\_alarmtime(a); \*/  }  // if (choice)  // get\_curtime(p);  return 0;  } |

实验9：Linux下GPIO驱动程序编写

1. 实验目的
2. 理解Linux GPIO驱动程序的结构、原理；
3. 掌握Linux GPIO驱动程序的编程；
4. 掌握Linux GPIO动态加载驱动程序模块的方法。
5. 实验内容
6. 编写GPIO字符设备驱动程序；
7. 编写Makefile文件；
8. 编写测试程序；
9. 调试GPIO驱动程序和测试程序。
10. 实验步骤

**步骤1：**编译GPIO驱动（在服务器或虚拟机上进行）

创建GPIO文件夹，并将服务器中的GPIO文件夹复制过来

# mkdir GPIO

# cd GPIO

虚拟机：# cp -r /home/shiyan/Desktop/shiyan/code/GPIO/ ./

服务器：# cp -r /home/shiyan/2021/code/GPIO/ ./

进入GPIO文件夹，“cd GPIO”，GPIO文件夹中包含三个文件，驱动程序davinci\_dm365\_gpios.c，Makefile和语言程序gpio.c。

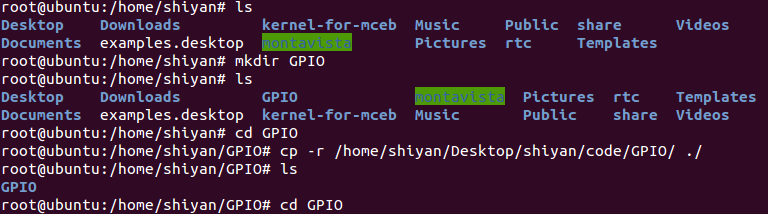


图9-1 将GPIO目录复制到自己的文件夹里

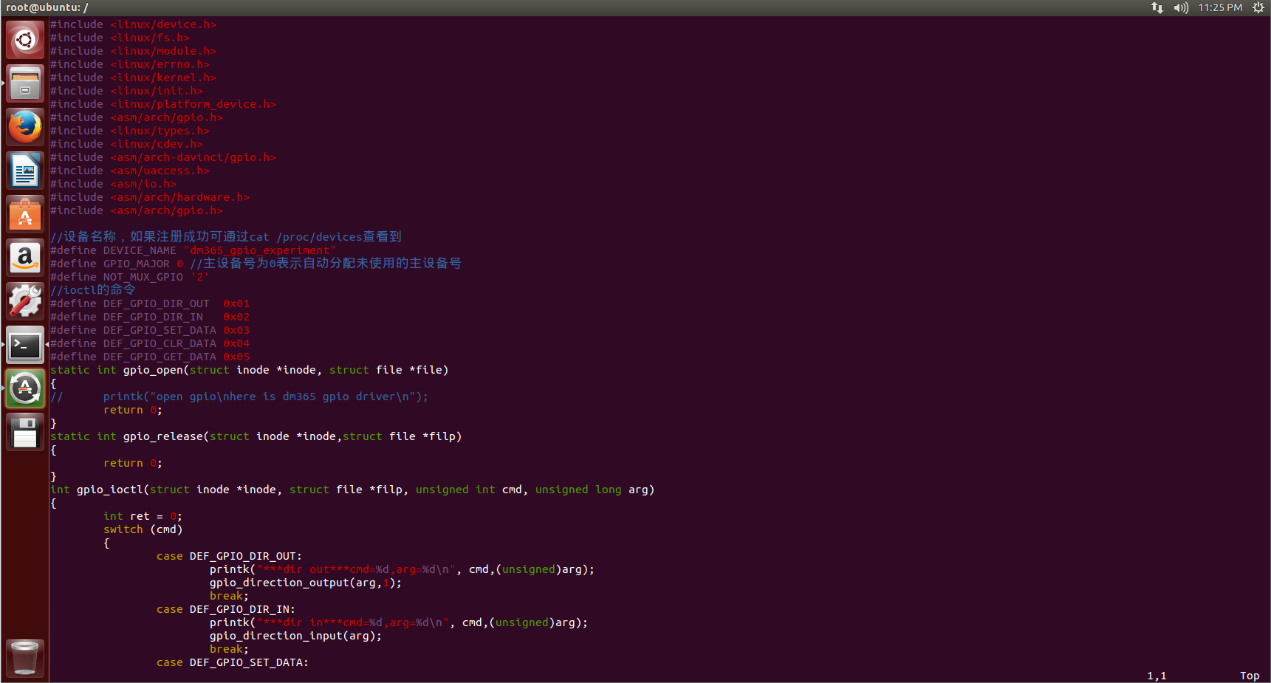


图9-2 驱动程序源码

管理员权限下退到根目录执行source /etc/profile使环境变量生效。

进入执行“make”命令，成功后会生成davinci\_dm365\_gpios.ko等文件。

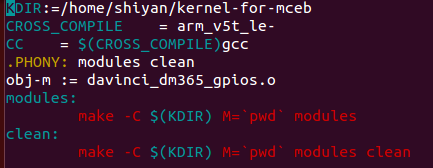


图9-3 Makefile

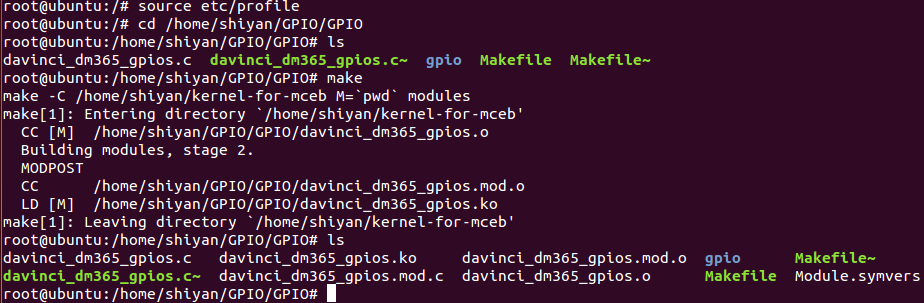


图9-4 生效环境变量后make生成.ko文件

davinci\_dm365\_gpios.ko文件生成成功后，将其复制到挂载的文件系统modules的目录下。

服务器：# cp davinci\_dm365\_gpios.ko /home/stX/filesys\_test/modules/

虚拟机：# cp davinci\_dm365\_gpios.ko /home/shiyan/share/filesys\_test/modules/



图9-5 将驱动程序复制到文件系统

**步骤2：**通过 pc 的远程登陆软件 putty 登录板子

**步骤3：**动态加载模块

（1）NFS挂载启动

正确连接好实验箱，开启电源，通过NFS作为启 动方式挂载虚拟机上filesys\_test文件系统作为根文件系统进行启动。注：启动机器前要确保实验箱设备的ip地址和 NFS 服务器（该服务器一般和实验箱设备在同一网关下）的ip地址可以相互ping通。且实验像设备的ip地址是该网关下唯一的ip地址否则将会冲突报错。登入板子后，在驱动模块加载之前查看设备节点使用命令 cat/proc/devices。

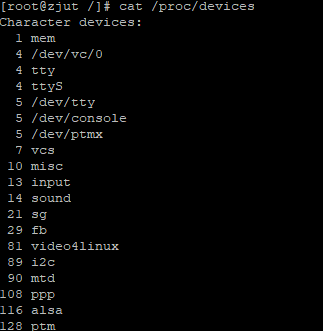


图9-6 cat查看设备节点

（2）动态加载

进入文件系统的 modules 目录下，查看已经加载的模块lsmod。运行模块加载命令insmod davinci\_dm365\_gpios.ko，模块成功加载，提示信息dm365\_gpio initialized。（如果加载失败，可以用lsmod命令查看是否已存在模块 davinci\_dm365\_gpios.ko，若已存在，可用rmmod davinci\_dm365\_gpios.ko先卸载，再依照上述方式加载查看结果）。



图9-7 dm365\_gpio模块加载成功

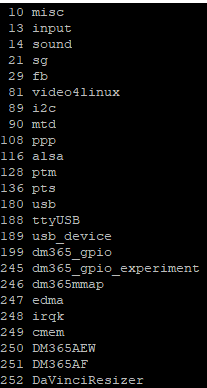


图9-8 重新查看已加载模块

**步骤4：**手动创建设备节点（以下操作在板子上进行）

当使用insmod命令成功加载davinci\_dm365\_gpios.ko驱动模块后，再次通通过命令cat /proc/devices查看已经注册的设备和设备号，可以发现dm365\_gpio\_experiment驱动已经加载完成。

驱动模块的设备节点创建有俩种方式手动创建和自动创建，gpio 驱动程序 为手动创建设备节点。

mknod /dev/dm365\_gpio\_experiment c 245 0

（其中，dm365\_gpio\_experiment是设备文件名，c代表字符设备，245代表 上图中dm365\_gpio\_experiment对应的主设备号，0表示次设备号，主设备号是 0因此这个设备号是随机分配未使用的设备号，因此每次加载模块产生的设备号可能是不一样的。)

**步骤5：**编写测试程序（在虚拟机上进行），并进行调试（在实验箱上进行）

测试程序在文件下GPIO/gpio/gpio.c下，gpio.c.就是对应的测试文件查看实 验测试程序。



图9-9 测试程序gpio.c

在服务器编写测试程序的各函数后，接下来是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的板子上运行调试。

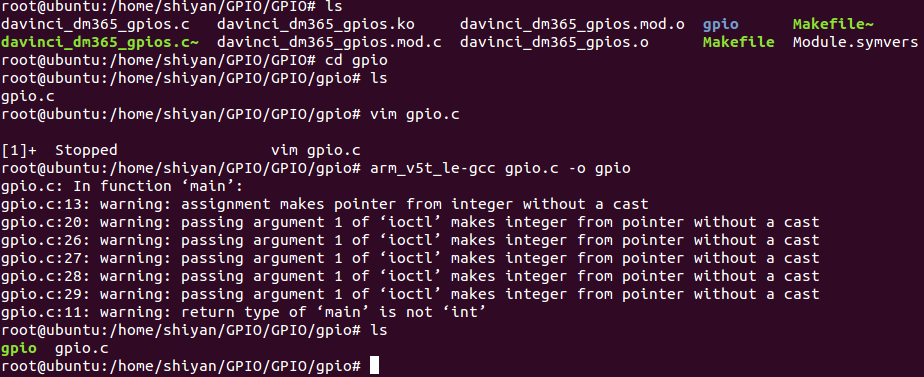


图9-10 交叉编译生成可执行文件gpio

交叉编译生成可执行文件gpio。编译成功后，可看见gpio文件, 将可执行文gpi复制到板子挂载的文件系统/home/shiyan/share/filesys\_test/opt/dm365。交叉编译生成可执行文件gpio。 编译成功后，将可执行文件gpio复制到/home/shiyan/share/filesys\_test/opt/dm365文件夹下，此时对实验箱进行操作进入/opt/dm365目录下执行测试代码。



图9-11 将gpio可执行文件复制进入opt的dm365目录下

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

服务器：# cp gpio /home/stX/file sys\_test/opt/dm365

实验箱：# cd /opt/dm365

执行如下命令 gpio 63 0/3。观察实验箱上液晶屏暗亮。

# gpio 63 0 （实验箱的板子上运行）lcd 背光打开

# gpio 63 3 （实验箱的板子上运行） lcd 背光关闭



图9-12 输入控制LCD背光指令

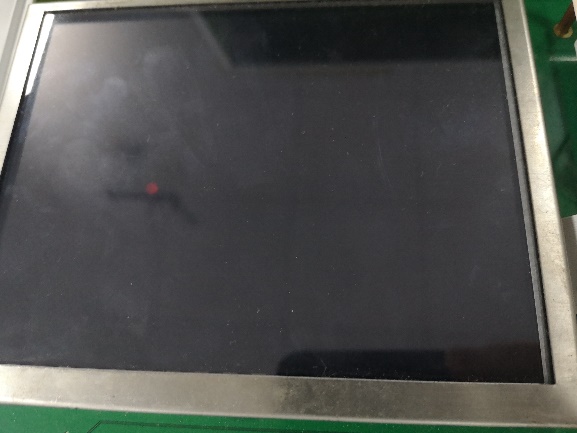


图9-13 LCD灭



图9-14 LCD亮

1. 心得与体会

GPIO口是嵌入式设备的重要模块，在嵌入式开发中的意义重大，通过此次驱动实验，对驱动程序的开发有了直观的认识，但是同样也发现，驱动程序的编写量非常大，仅仅靠这一两次实验是远远不够的，课后还是需要我们认真分析代码尝试自己写代码。

1. 附录

GPIO驱动主要代码

|  |
| --- |
| #include <linux/device.h>  #include <linux/fs.h>  #include <linux/module.h>  #include <linux/errno.h>  #include <linux/kernel.h>  #include <linux/init.h>  #include <linux/platform\_device.h>  #include <asm/arch/gpio.h>  #include <linux/types.h>  #include <linux/cdev.h>  #include <asm/arch-davinci/gpio.h>  #include <asm/uaccess.h>  include <asm/io.h>  #include <asm/arch/hardware.h>  #include <asm/arch/gpio.h>  //设备名称，如果注册成功可通过 cat /proc/devices 查看到  #define DEVICE\_NAME "dm365\_gpio\_experiment"  #define GPIO\_MAJOR 0 //主设备号为 0 表示自动分配未使用的主设备号  #define NOT\_MUX\_DATA 0x03  #define DEF\_GPIO\_CLR\_DATA 0x04  #define DEF\_GPIO\_GET\_DATA 0x05  static int gpio\_open(struct inode \*inode, struct file \*file)  {  // printk("open gpio\nhere is dm365 gpio driver\n");  return 0;  }  static int gpio\_release(struct inode \*inode,struct file \*filp)  {  return 0;  }  int gpio\_ioctl(struct inode \*inode, struct file \*filp, unsigned int cmd, unsigned long arg)  {  int ret = 0;  switch (cmd)  {  case DEF\_GPIO\_DIR\_OUT:  printk("\*\*\*dir out\*\*\*cmd=%d,arg=%d\n",cmd,(unsigned)arg);  gpio\_direction\_output(arg,1);  break;  case DEF\_GPIO\_DIR\_IN:  printk("\*\*\*dir in\*\*\*cmd=%d,arg=%d\n", cmd,(unsigned)arg);  gpio\_direction\_input(arg);  break;  case DEF\_GPIO\_SET\_DATA:  case DEF\_GPIO\_CLR\_DATA:  // printk("data=%d\n", cmd);  if (cmd == DEF\_GPIO\_SET\_DATA)  gpio\_set\_value(arg, 1);  else  gpio\_set\_value(arg, 0);  // break;  case DEF\_GPIO\_GET\_DATA:  ret = gpio\_get\_value(arg);  // printk("gpio%d = %d\n",arg,ret);  break;  }  return ret;  }  static struct file\_operations gpio\_fops = {  .owner=THIS\_MODULE,  .open=gpio\_open,  .release=gpio\_release, .ioctl=gpio\_ioctl  };  static int \_\_init davinci\_dm365\_gpio\_init (void)  {  int ret;  ret=register\_chrdev(GPIO\_MAJOR, DEVICE, &gpio\_fops);  if(ret < 0)  {  printk("dm365\_gpio register falid!\n");  return ret;  }  printk ("dm365\_gpio initialized\n");  return ret;  }  static void \_\_exit davinci\_dm365\_gpio\_exit(void)  {  unregister\_chrdev(GPIO\_MAJOR, DEVICE\_NAME);  printk("dm365\_gpio exit\n");  }  module\_init(davinci\_dm365\_gpio\_init);  module\_exit(davinci\_dm365\_gpio\_exit);  MODULE\_AUTHOR("jingwei,Song <Punuo Ltd.>");  MODULE\_DESCRIPTION("Davinci DM365 gpio driver");  MODULE\_LICENSE("GPL"); |