嵌入式系统原理实验报告五

通信 1503 班 201503090323 叶启彬

实验五 RTC 时钟驱动&GPIO 驱动程序编写实验

一、实验目的

1. 了解 RTC 工作原理；
2. 掌握 RTC 时钟驱动编程。
3. 理解 Linux 驱动程序的结构、原理；
4. 掌握 Linux 驱动程序的编程；
5. 掌握 Linux 动态加载驱动程序模块的方法。

二、实验设备

1. 硬件：PC 机；AMR9 系统教学实验系统连接线。
2. 软件：PC 机操作系统（Windows XP）；Linux OS；VMware；ARM-Linux-GCC 交叉编译环境。

三、实验原理

（一） RTC 时钟驱动实验

实时时钟（RTC）器件是一种能提供日历/时钟、数据存储等功能的专用集成电路，常用作各种计算机系统的时钟信号源和参数设置存储电路。RTC 具有计时准确、耗电低和体积小等特点，特别是在各种嵌入式系统中用于记录事件发生的时间和相关信息，如通信工程、电力自动化、工业控制等自动化程度高的领域的无人值守环境。随着集成电路技术的不断发展，RTC 器件的新品也不断推出，这些新品不仅具有准确的 RTC，还有大容量的存储器、温度传感器和 A/D 数据采集通道等，已成为集 RTC、数据采集和存储于一体的综合功能器件，特别适用于以微控制器为核心的嵌入式系统。

本实验箱采用 X1205 芯片作为 RTC 时钟芯片，X1205 是一个带有时钟日历两路报警振荡器补偿和电池切换的实时时钟。

振荡器用一个外部的低价格的 32.768Khz 晶体所有补偿和调整元件集成于芯片上。这样除去了外部的离散元件和一个调整电容节约电路板空间和元器件的费用。

实时时钟用分别的时分秒寄存器跟踪时间日历有分别的日期星期月和年寄

存器日历可正确通过 2099 年具有自动闰年修正功能。

强大的双报警功能能够被设置到任何时钟日历值上与报警相匹配例如每分

钟每个星期二或三月 21 日上午 5:23 均可报警能够在状态寄存器中被查询或提供一个硬件的中断 IRQ 管脚这是一个重复模式报警容许产生一个周期性的中断。

该器件提供一个备份电源输入脚 VBACK 该脚容许器件用电池或大容量电容进行备份供电整个 X1205 器件的工作电压范围为 2.7 V 至 5.5V X1205 的时钟日历部分的工作可降到 1.8V(待机模式)。

与 RTC 核心有关的文件有下面 6 个，结构图如下图 1 所示：

1、 /drivers/rtc/class.c 这个文件向 linux 设备模型核心注册了一个类 RTC，然后向驱动程序提供了注册/注销接口；

2、/drivers/rtc/rtc-dev.c 这个文件定义了基本的设备文件操作函数，如：open,read 等；

3、 /drivers/rtc/interface.c 顾名思义，这个文件主要提供了用户程序与 RTC 驱动的接口函数，用户程序一般通过 ioctl 与 RTC 驱动交互，这里定义了每个 ioctl 命令需要调用的函数；

4、 /drivers/rtc/rtc-sysfs.c 与 sysfs 有关；

5、/drivers/rtc/rtc-proc.c 与 proc 文件系统有关；

6、/include/linux/rtc.h 定义了与 RTC 有关的数据结构。

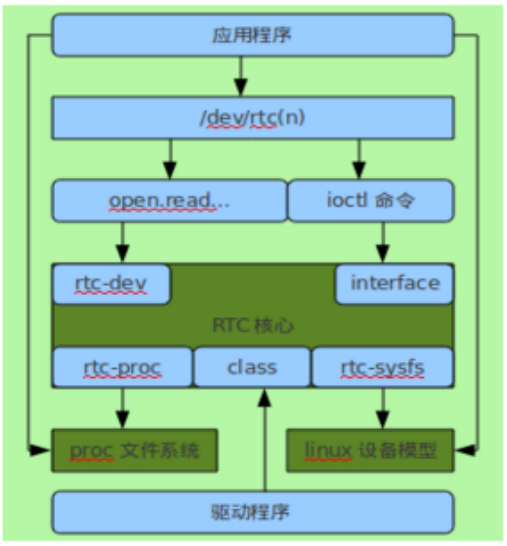

图 1. RTC 驱动结构模型

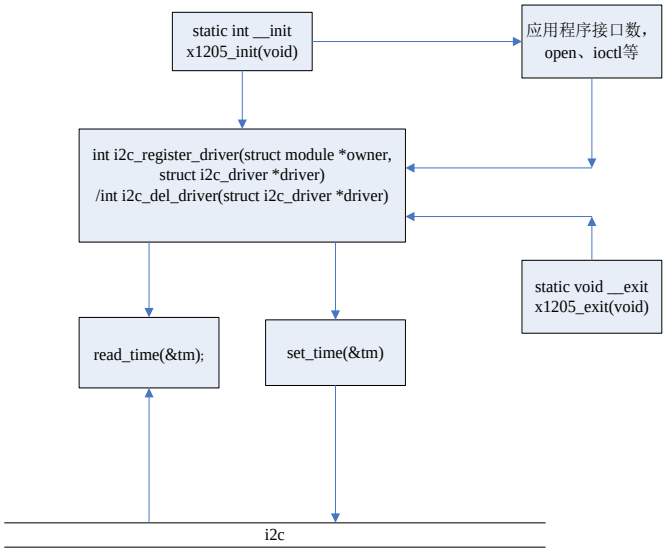

图 2. RTC 驱动流程图

（二） GPIO 驱动程序编写实验

在嵌入式系统中，常常有数量众多，但是结构却比较简单的外部设备/电路，对这些设备/电路有的需要 CPU 为之提供控制手段，有的则需要被 CPU 用作输入信号。而且，许多这样的设备/电路只要求一位，即只要有开/关两种状态就够了，例如灯的亮与灭。对这些设备/电路的控制，使用传统的串行口或并行口都不合适。所以在微控制器芯片上一般都会提供一个通用可编程 I/O 接口，即 GPIO（General Purpose Input Output）。

GPIO 的驱动主要就是读取 GPIO 口的状态，或者设置 GPIO 口的状态。就是这么简单，但是为了能够写好的这个驱动，在 LINUX 上作了一些软件上的分层。为了让其它驱动可以方便的操作到 GPIO，在 LINUX 里实现了对 GPIO 操作的统一接口，这个接口实则上就是 GPIO 驱动的框架。

在本实验中，将编写简单的 GPIO 驱动程序来控制 LCD 液晶屏屏幕的亮灭，然后动态加载模块，并编写测试程序，以验证驱动程序。

GPIO 驱动是 Linux 驱动开发中最基础、但却是很常用、很重要的驱动。比如要点亮一个 LED 灯、键盘扫描、输出高低电平等等。而 Linux 内核的强大之处在于对最底层的 GPIO 硬件操作层的基础上封装了一些统一的 GPIO 操作接口，也就是所谓的 GPIO 驱动框架。这样开发人员可以调用这些接口去操作设备的 I/O 口，不需要担心硬件平台的不同导致 I/O 口的不同，方便对各个模块进行控制。

GPIO 外设提供专用的可配置为输入或输出的通用引脚。当配置为一个输出，你可以写一个内部寄存器来控制输出引脚上的状态。当配置为一个输入时，你可以通过读取内部寄存器的状态来检测输入的状态。当配置为一个高电平时，可以通过改变内部寄存器的状态来改变引脚的状态为高电平。如表 1 所示：

表 1　GPIO 寄存器

GPIO寄存器	实际执行函数	作用
DIRn	gpio_direction_output(arg,1)	设置该GPIO口为输出
DIRn	gpio_direction_input(arg)	设置该GPIO口为输入
SET_DATAn	gpio_set_value(arg,1)	设置该GPIO口为高电平
CLR_DATAn	gpio_set_value(arg,0)	设置该GPIO口为低电平
GET_DATA	gpio_get_value(arg)	获取该GPIO的状态

由于 TMS320DM365 芯片的管脚不是很多，所以大部分管脚都是复用的，需要对复用管脚进行有序的管理，保证系统正常稳定工作，而在应用层，也需要对 IO 管脚进行控制来实现一定功能。

四、实验内容（代码注释及步骤）

（一）RTC 时钟驱动实验

1. 实验内容

(1)掌握 Linux 驱动程序工作机制；

(2)掌握汇编语言和 C 语言；

(3)掌握 Linux 交叉编译和基本操作；

(4)学会驱动程序的调试方法。

2. 实验步骤：

步骤 0：编写 RTC 驱动代码

RTC 驱动代码已编写好，只需要将其复制到自己的用户目录下。

步骤 1：编写用于交叉编译的 Makefile

(1) 编写驱动程序编译成模块所需要的 Makefile 等文件在 shiyan/2018/RTC 目录下，将其复制到自己的用户目录下。

#sudo cp -r /home/shiyan/2018/RTC /home/st2

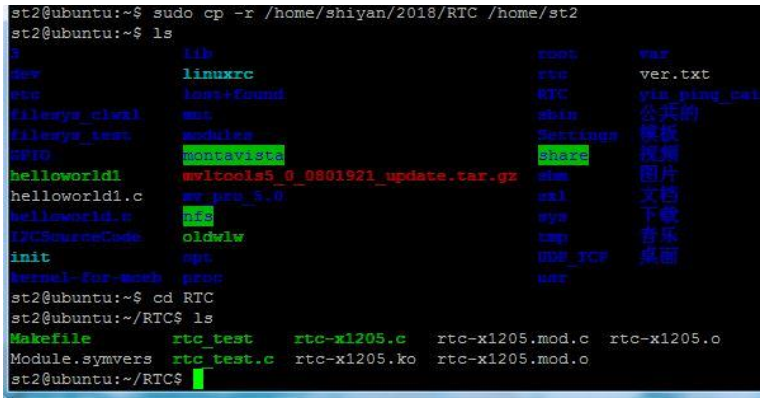

图 3.将包含 Makefile 文件的 RTC 文件夹复制到 st2 下

(2)利用 make 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。

cp rtc-x1205.ko /home/st2 filesys-test/modules

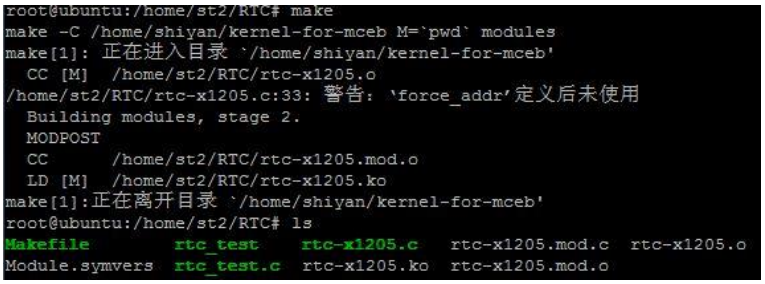

图 4.make 产生 rtc-x1205.ko

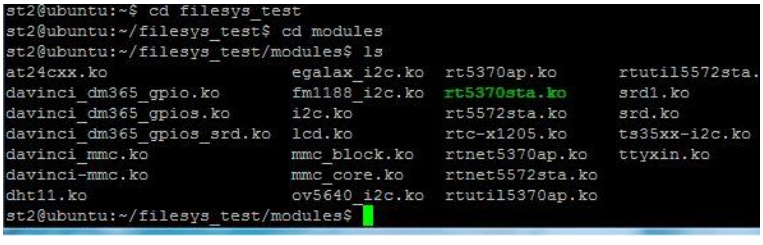

图 5. 查看 filesys_test 中的 rtc-x1205.ko

(3)编写测试程序

进入.c 文件所在目录，交叉编译测试程序 rtc_test.c，把测试程序编译成可执行文件
arm_v5t_le-gcc -o rtc_test rtc_test.c

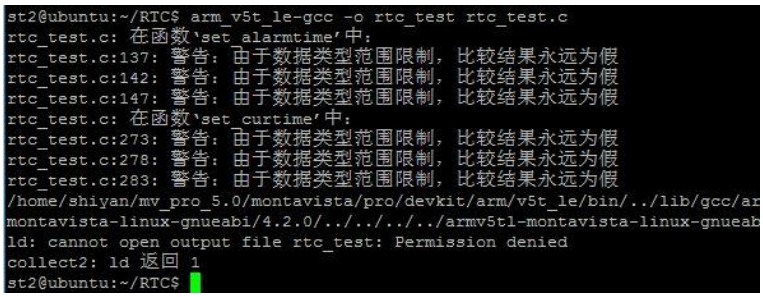

图 6.编译生成可执行文件

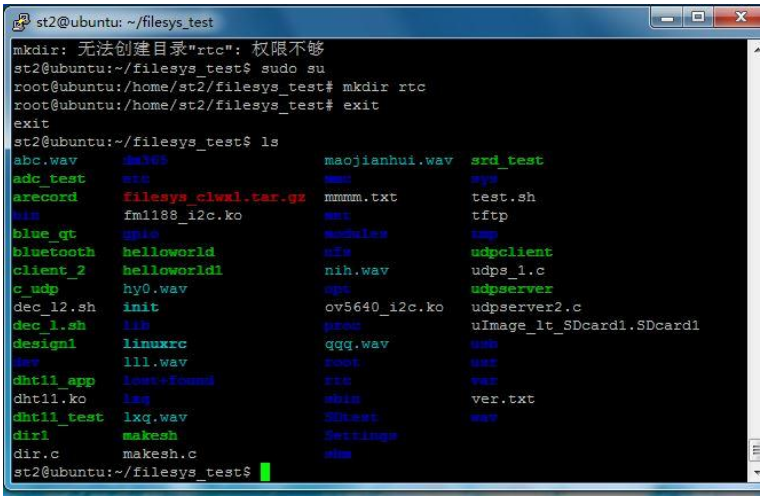

图 7.创建文件夹 rtc

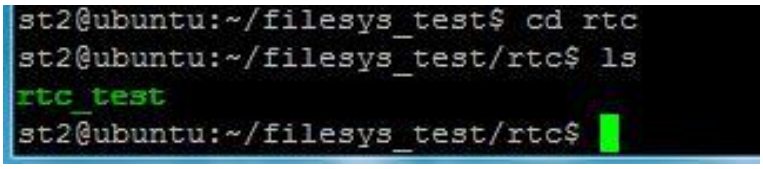

图 8.在 rtc 文件夹中有可执行文件 rtc_test

步骤 2：挂载

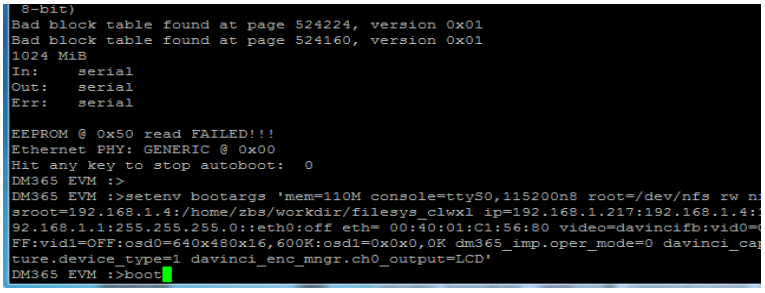

图 7.挂载文件系统

步骤 3：在开发板启动后，通过 insmod 命令加载 RTC 时钟芯片驱动程序

(1)输入 root 进入 root 权限，使用 insmod /modules/rtc-x1205.ko 加载 RTC 驱动模块。

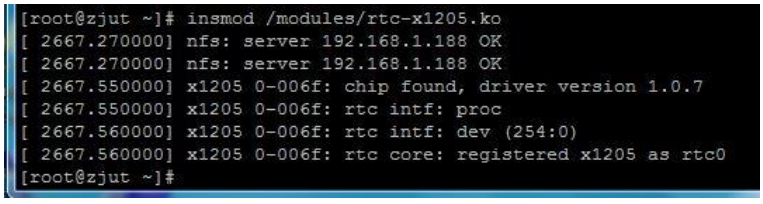

图 8. 加载 rtc-x1205.ko

(2)查找测试程序

使用命令 cd /home/st2/filesys_test/rtc 进入测试程序所在目录，找到自己的测试程序。

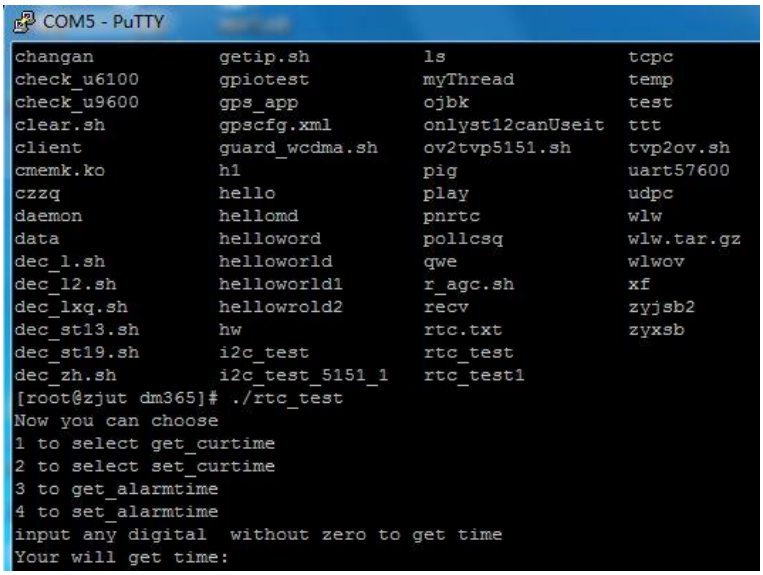

图 9.查找测试程序 rtc_test

步骤 4：执行测试程序，查看结果

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。

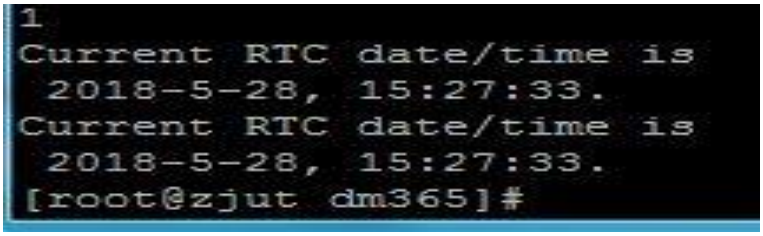

图 10.输入 1，显示现在的时间

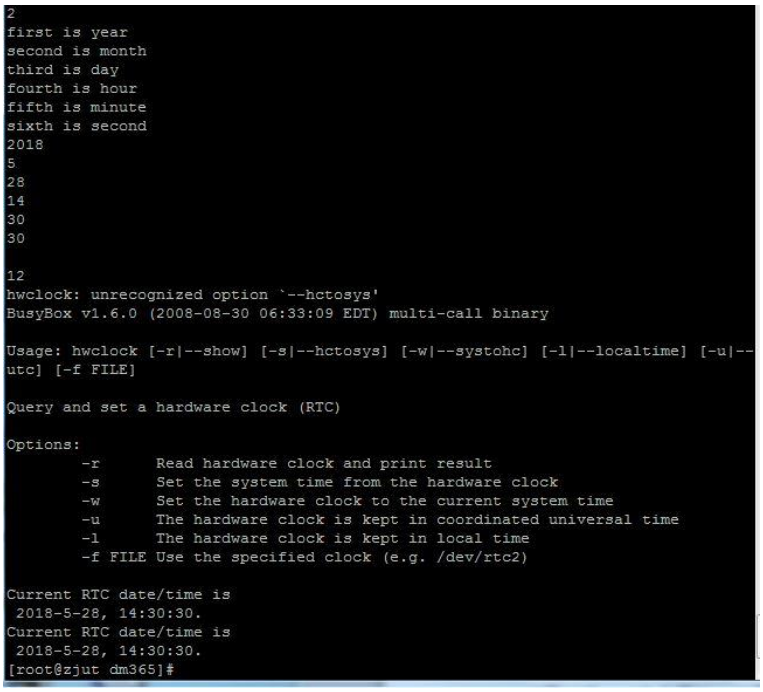

图 11.输入 2，设置时间

```
[root@zjut dm365]# ./rtc_test
Now you can choose
1 to select get_curtime
2 to select set_curtime
3 to get_alarmtime
4 to set_alarmtime
input any digital  without zero to get time
Your will get time:
3
252
252
190
0
Current RTC date/time is
 2018-5-28, 14:31:02.
[root@zjut dm365]#
```

图 12.输入 3，显示闹钟时间

步骤 6：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 121212122016（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。

```
[root@zjut dm365]# date 121212122016
Mon Dec 12 12:12:00 UTC 2016
[root@zjut dm365]# hwclock -w
[root@zjut dm365]# hwclock -r
Mon Dec 12 12:12:20 2016  0.000000 seconds
[root@zjut dm365]#
```

图 13.将系统时间写入硬件

（2）GPIO 驱动程序编写实验

1. 实验内容

(1)编写字符设备驱动程序；

(2)编写 Makefile 文件；

(3)编写测试文件；

(4)调试 GPIO 驱动程序和测试程序。

2. 实验步骤：

步骤 0：正确连实验箱及接 PC 机

将串口连接 PC 机，实验板电源线正确连接，网线正确连接 PC 机。

步骤 1：编译 GPIO 驱动（在服务器上进行）

(1)创建 GPIO 文件夹，执行 mkdir GPIO，编写 GPIO 驱动程序。

编写驱动程序编译成模块所需要的 Makefile 等文件在 shiyan/2018/GPIO 目录下，将其复制到自己的用户目录下。

#sudo cp -r /home/shiyan/2018/GPIO /home/st2/GPIO

编写驱动程序编译成模块所需要的 Makefile 文件，执行 vi Makefile 做出相应修改。

```
st2@ubuntu:~$ sudo cp -r /home/shiyan/2018/GPIO /home/st2/GPIO
st2@ubuntu:~$ cd gPIO
-bash: cd: gPIO: 没有那个文件或目录
st2@ubuntu:~$ cd GPIo
-bash: cd: GPIo: 没有那个文件或目录
st2@ubuntu:~$ cd GPIO
st2@ubuntu:~/GPIO$ ls
GPIO
st2@ubuntu:~/GPIO$
```

图 14.将文件复制到 GPIO 目录下

(2) 执行 make 命令，成功后会生成 davinci_dm365_gpios.ko 等文件。

```
st2@ubuntu:~/GPIO/GPIO$ sudo su
root@ubuntu:/home/st2/GPIO/GPIO# make
make -C /home/st2/kernel-for-mceb M=`pwd` modules
make[1]: 正在进入目录 `/home/st2/kernel-for-mceb'
  CC [M]  /home/st2/GPIO/GPIO/davinci_dm365_gpios.o
  Building modules, stage 2.
  MODPOST
  CC      /home/st2/GPIO/GPIO/davinci_dm365_gpios.mod.o
  LD [M]  /home/st2/GPIO/GPIO/davinci_dm365_gpios.ko
make[1]:正在离开目录 `/home/st2/kernel-for-mceb'
root@ubuntu:/home/st2/GPIO/GPIO# ls
davinci_dm365_gpios.c      davinci_dm365_gpios.mod.o  Makefile
davinci_dm365_gpios.ko     davinci_dm365_gpios.o      Module.symvers
davinci_dm365_gpios.mod.c  gpio.c
root@ubuntu:/home/st2/GPIO/GPIO#
```

图 15.驱动模块

步骤 2：手动加载模块（以下操作在板子上进行）

(1) 挂载

```
8-bit)
Bad block table found at page 524224, version 0x01
Bad block table found at page 524160, version 0x01
1024 MiB
In:    serial
Out:   serial
Err:   serial

EEPROM @ 0x50 read FAILED!!!
Ethernet PHY: GENERIC @ 0x00
Hit any key to stop autoboot:  0
DM365 EVM :>
DM365 EVM :>setenv bootargs 'mem=110M console=ttyS0,115200n8 root=/dev/nfs rw
sroot=192.168.1.4:/home/zbs/workdir/filesys_clwxl ip=192.168.1.217:192.168.1.
92.168.1.1:255.255.255.0::eth0:off eth= 00:40:01:C1:56:80 video=davincifb:vid
FF:vid1=OFF:osd0=640x480x16,600K:osd1=0x0x0,0K dm365_imp.oper_mode=0 davinci_
ture.device_type=1 davinci_enc_mngr.ch0_output=LCD'
DM365 EVM :>boot
```

图 16.挂载文件系统

```
st2@ubuntu:/modules$ ls
at24cxx.ko              lcd.ko          rtc-x1205.ko      ts35xx-i2c.ko
davinci_dm365_gpios.ko  ov5640_i2c.ko   rtnet5370ap.ko    ttyxin.ko
egalax_i2c.ko           rt5370ap.ko     rtnet5572sta.ko
fm1188_i2c.ko           rt5370sta.ko    rtutil5370ap.ko
i2c.ko                  rt5572sta.ko    rtutil5572sta.ko
st2@ubuntu:/modules$
```

图 17.在 modules 文件夹中含有参数 rtc-x1205.ko 文件

(2) 查看设备节点 cat /proc/devices

```
[root@zjut dm365]# 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
```

图 18.当前设备节点(1)

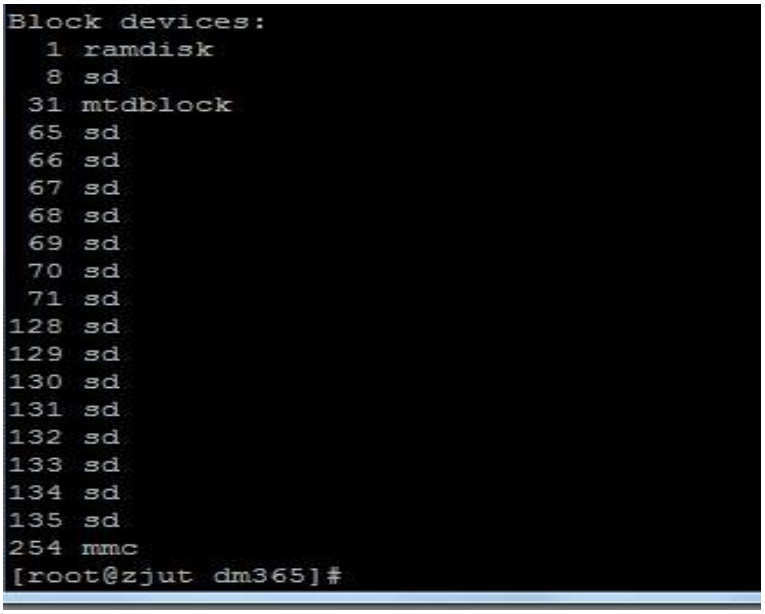

图 19.当前设备节点(2)

(3) 动态加载文件

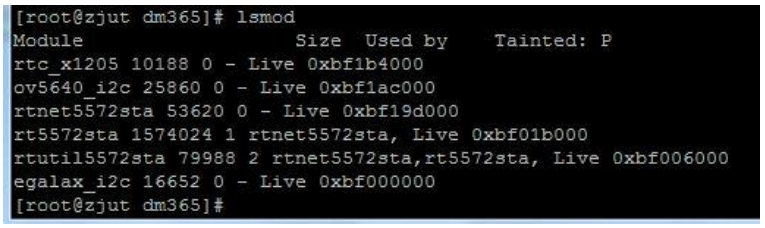

图 20.当前加载的模块

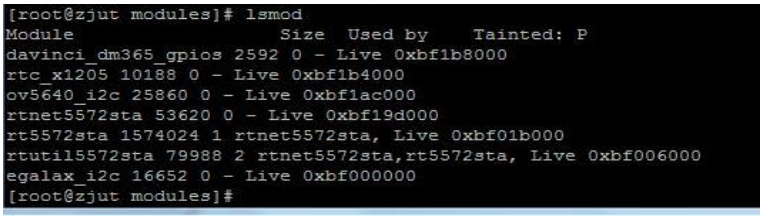

图 21.加载后的模块

步骤 3：编写测试程序，并进行测试

测试程序.c 文件已经编好，进入.c 文件所在目录，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。

#arm_v5t_le-gcc gpio.c -o gpio

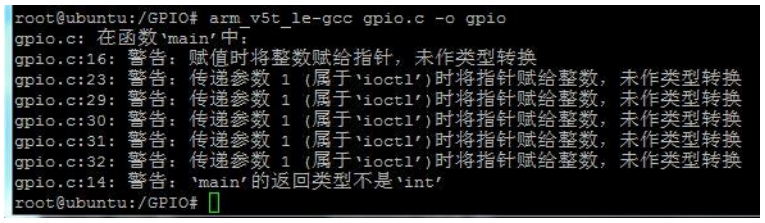

图 22. 交叉编译生成可执行文件 gpio

交叉编译生成可执行文件 gpio。编译成功后，可看见 gpio 文件，将可执行文件 gpio 复制到板子挂载的文件系统 filesys_test/gpio。

sudo cp gpio /home/st2/ filesys_test/gpio

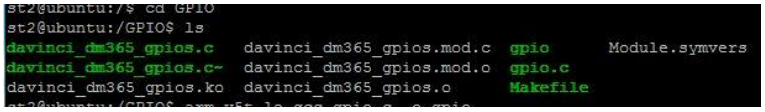

图 23. 查看文件夹 GPIO 中的 gpio 可执行文件

执行如下命令 gpio 63 0/3。观察实验箱上液晶屏暗亮有没有达到实验预期结果。

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

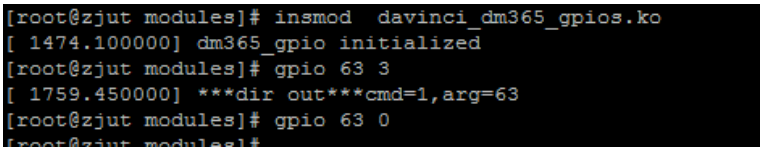

图 24. 输入控制 LCD 背光灯指令

三、实验总结及心得体会

在本次实验开始前，了解了 RTC 工作原理，以及 Linux 驱动程序的结构、原理；并掌握 RTC 时钟驱动编程、Linux 驱动程序的编程；在实验过程中，掌握了 Linux 动态加载驱动程序模块的方法。实验过程基本没有大的问题。出现的一个错误是，将 GPIO 文件间放在了新建的 GPIO 文件夹中，这样导致了文件夹目录需要进入两层。后来经过修改完成了实验。实现了对 LCD 背光灯的控制。

附录：代码注释

(1) RTC 的 Makefile 注释：

```
KDIR:=/home/shiyan/kernel-for-mceb          %KDIR 为 home 目录的实验目录下的 kernel-for-mceb
CROSS_COMPILE       = arm_v5t_le-           %编译进内核；CROSS_COMPILE 这个变量是用来告诉 Makefile 我们
                                            要用 ARM-LINUX-GCC 编译器编译，所以要告诉它你电脑上
                                            ARM-LINUX-GCC 程序被安装的目录在哪
CC      = $(CROSS_COMPILE)gcc               %以“$()”的方式来使用这个变量
.PHONY: modules clean                       %.PHONY 目标的具体意思是如果在 Makefile 的工作目录中有名如：
                                            modules,modules_install,clean 等文件时命令会出错。它是防止这出错的
                                            方式。
obj-m := rtc-x1205.o                        %Obj-m := (源文件名).o
modules:                                    %建立模块
make -C $(KDIR) M=`pwd` modules             %-C 选项的作用是指将当前工作目录转移到你所指定的位置。“M=”
                                            选项的作用是，当用户需要以某个内核为基础编译一个外部模块的话，
                                            需要在 make modules 命令中加入“M=dir”，程序会自动到你所指定
                                            的 dir 目录中查找模块源码，将其编译，生成 KO 文件。
clean:                                      %清除模块
      make -C $(KDIR) M=`pwd` modules clean %clean 的规则不要放在文件的开头，不然，这就会变成 make 的默认
                                            %目标。
```

(2) rtc.test.c 注释：

```
#include <ctype.h>                          %申明头文件
#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         %定义结构体 struct_tag_TimeInfor
{
        unsigned char year;                 %将 year、month、day、week、hour、min、sec
```

```
        unsigned char month;                    %定义为 unsigned char 类型。
        unsigned char day;
        unsigned char week;
        unsigned char hour;
        unsigned char min;
        unsigned char sec;
}TimeInfo;                                      %结构体的名称为 TimeInfo；
#endif                                          %结束

typedef struct struct_tag_alarmInfor            %定义结构体 struct_tag_ alarmInfor
{
        unsigned char enable;                   %定义使能变量 enable；
        unsigned char pending;                  %定义变量 pending 为 unsigned char 类型；
      struct struct_tag_TimeInfor t_time;       %嵌套进结构体 struct_tag_TimeInfor，并命名为 t_time;
}AlarmInfo;                                     %结构体的名称为 AlarmInfo；

int get_alarmtime(AlarmInfo time)               %定义函数 get_alarmtime，获得当前的闹钟时间；
{
//      FILE *fp;
        int fd,retval;                          %变量 fd 和 retval 定义为 int 类型；
        struct rtc_wkalrm rtc_tm;               %变量 rtc_tm 为 rtc_wkalrm 结构体变量；
//      fp = fopen("rtc.txt","a");
        fd = open ("/dev/rtc0", O_RDONLY);      %将 O_RDONLY 赋值给 fd；
        if (fd ==  -1)                          %如果 fd=-1；
          {
                perror("/dev/rtc0");            %将输入的信息和现在的 error 所对应的错误一起输出；
                exit(errno);                    %erron=0 时，exit(0),退出进程；
        }
          retval = ioctl(fd, RTC_ALM_READ, &rtc_tm);      %对 I/O 通道进行管理
          //time.enabled=rtc_tm.enabled
        //time.pending=rtc_tm.pending
        time.t_time.year=rtc_tm.time.tm_year;           %将 rtc_tm.time.tm_year 赋值给 time.t_time.year；
          time.t_time.month=rtc_tm.time.tm_mon;         %将 rtc_tm.time.tm_mon 赋值给 time.t_time.mon；
          time.t_time.day=rtc_tm.time.tm_mday;          %将 rtc_tm.time.tm_ mday 赋值给 time.t_time.mday；
          time.t_time.week=rtc_tm.time.tm_wday;         %将 rtc_tm.time.tm_ wday 赋值给 time.t_time.wday；
          time.t_time.hour=rtc_tm.time.tm_hour;         %将 rtc_tm.time.tm_ hour 赋值给 time.t_time.hour；
          time.t_time.min=rtc_tm.time.tm_min;           %将 rtc_tm.time.tm_min 赋值给 time.t_time.min；
          time.t_time.sec=rtc_tm.time.tm_sec;           %将 rtc_tm.time.tm_sec 赋值给 time.t_time.sec；
          fprintf(stdout, "Current ALARM date/time is \n %d-%d-%d, %02d:%02d:%02d.\n",
         time.t_time.year + 1900, time.t_time.month + 1, time.t_time.day,time.t_time.hour, time.t_time.min, time.t_time.sec);
                                                %按照规则输出闹钟时间；
          close(fd);                            %close(fd)；
          return 0;                             %返回 0；
}

int set_alarmtime(AlarmInfo time)               %定义函数 set_alarmtime 设置闹钟时间；
{
        int i,fd,retval;                        %变量 i，fd 和 retval 定义为 int 类型；
        int b[6];                               %定义数组 b[6]；
        char *rtime[6];                         %定义指针数组*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"};     %定义指针数组*message[]；
        struct rtc_wkalrm alarm;                %定义数组 alarm 为数组 rtc_wkalrm；
        fd = open ("/dev/rtc0", O_RDWR);        %打开/dev/rtc0，是否为可读可写；
        if (fd ==  -1)                          %如果 fd=-1；
        {
          perror("/dev/rtc0");                  %将输入的信息和现在的 error 所对应的错误一起输出；
            exit(errno);                        % erron=0 时，exit(0),退出进程；
        }
```

```
    fprintf(stdout,*message);                       %输出 message 到 stdout 中；
    for(i=0;i<6;i++)                                %for 循环
    {
        rtime[i]=(char *)malloc(sizeof(char)*10);   %给 rtime 进行动态内存分配；
        scanf("%s\n \n",rtime[i]);                  %输出 rtime；
    }
    for(i=0;i<6;i++)                                % for 循环
    b[i]=atoi(rtime[i]);                            %将字符转换成整数类型；
    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);
     }                                              %当输入的月份小于 1 大于 12 时，报错；
    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);
    }                                               %当输入的日小于 1 大于 31 时，报错；
      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);
        }                                           %判断月份为 4,6,9,11 时输入的天数应在 1~30；
    }
    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);
            }
        }                                           %判断月份为 2 且为闰年时输入的天数应在 1~29；
        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);
            }                                       %判断月份为 2 且为平年时输入的天数应在 1~28；
        }
    }
    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);
        }                                           %判断小时数是否为 1~23，否则报错；
        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);
        }                                           %判断分钟数是否为 1~59，否则报错；
        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);
        }                                           %判断秒数是否为 1~59，否则报错；
        alarm.time.tm_year=time.t_time.year;        %将 time.t_time.year 赋值给 alarm.time.tm_year；
        alarm.time.tm_mon=time.t_time.month;        %将 time.t_time.month 赋值给 alarm.time.tm_mon；
        alarm.time.tm_mday=time.t_time.day;         %将 time.t_time.day 赋值给 alarm.time.tm_day；
        alarm.time.tm_hour=time.t_time.hour;        %将 time.t_time.hour 赋值给 alarm.time.tm_hour；
        alarm.time.tm_min=time.t_time.min;          %将 time.t_time.min 赋值给 alarm.time.tm_min；
        alarm.time.tm_sec=time.t_time.sec;          %将 time.t_time.sec 赋值给 alarm.time.tm_sec；
        retval = ioctl(fd, RTC_ALM_SET, &alarm);    %对 I/O 通道进行管理
        if (retval == -1)                           %判断 retval 是否为-1；
        {
            perror("ioctl");                        %将输入的信息和现在的 error 所对应的错误一起输出；
            exit(errno);                            %erron=0 时，exit(0),退出进程；
        }
        close(fd);                                  %fd 为 1 则关闭；
        //              system("/bin/busybox hwclock --hctosys");
        return 0;                                   %返回 0；
}

int get_curtime(TimeInfo time)                      %定义函数 get_curtime；
{
//      FILE *fp;
        int fd,retval;                              %变量 fd 和 retval 定义为 int 类型；
        struct rtc_time rtc_tm;                     %变量 rtc_time 为 rtc_tm；
//      fp = fopen("rtc.txt","a");
        fd = open ("/dev/rtc0", O_RDONLY);          %打开/dev/rtc0，是否为可读可写；
        if (fd ==   -1)                             %如果 fd=-1；
          {
          perror("/dev/rtc0");                      %将输入的信息和现在的 error 所对应的错误一起输出；
            exit(errno);                            % erron=0 时，exit(0),退出进程；
        }
        }
        retval = ioctl(fd, RTC_RD_TIME, &rtc_tm);   %对 I/O 通道进行管理；
        time.year=rtc_tm.tm_year;                   %将 rtc_tm.tm_year r 赋值给 time.year；
        time.month=rtc_tm.tm_mon;                   %将 rtc_tm.tm_mon r 赋值给 time.month；
        time.day=rtc_tm.tm_mday;                    %将 rtc_tm.tm_mday r 赋值给 time.day；
        time.week=rtc_tm.tm_wday;                   %将 rtc_tm.tm_wdayr 赋值给 time.week；
        time.hour=rtc_tm.tm_hour;                   %将 rtc_tm.tm_hourr 赋值给 time.hour；
        time.min=rtc_tm.tm_min;                     %将 rtc_tm.tm_minr 赋值给 time.min；
        time.sec=rtc_tm.tm_sec;                     %将 rtc_tm.tm_sec r 赋值给 time.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);
                                                    %按照规则输出时间；
          close(fd);                                %close(fd)；
          return 0;                                 %返回 0；
}
int set_curtime(TimeInfo time)                      %定义函数 set_curtime
{
        int i,fd,retval;                            %变量 i，fd 和 retval 定义为 int 类型；
```

```
        int b[6];                                   %定义数组 b[6]；
        char *rtime[6];                             %定义指针数组*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"};                                         %定义指针数组*message[]；
        struct rtc_time rtc_tm;                     %变量 rtc_time 为 rtc_tm；
        fd = open ("/dev/rtc0", O_RDWR);            %打开/dev/rtc0，是否为可读可写；
        if (fd ==  -1)                              %如果 fd=-1；
        {
            perror("/       dev/rtc0");             %将输入的信息和现在的 error 所对应的错误一起输出；
            exit(errno);                            % erron=0 时，exit(0),退出进程；
        }
        fprintf(stdout,*message);                   %输出 message 到 stdout 中；
        for(i=0;i<6;i++)                            % for 循环
        {
            rtime[i]=(char *)malloc(sizeof(char)*10); %给 rtime 进行动态内存分配；
            scanf("%s\n \n",rtime[i]);              %输出 rtime；
        }
        for(i=0;i<6;i++)                            % for 循环
        b[i]=atoi(rtime[i]);                        %将字符转换成整数类型；
        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];                              //second
/*      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);
        }                                           %当输入的月份小于 1 大于 12 时，报错；
        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);
        }                                           %当输入的日小于 1 大于 31 时，报错；
        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);
                  }                                 %判断月份为 4,6,9,11 时输入的天数应在 1~30；
             }
        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);
                    }                               %判断月份为 2 且为闰年时输入的天数应在 1~29；
               }
        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);
            }                           %判断月份为 2 且为平年时输入的天数应在 1~28；
        }
      }
    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);
    }                                   %判断小时数是否为 1~23，否则报错；
    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);
    }                                   %判断分钟数是否为 1~59，否则报错；
    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);
    }                                   %判断秒数是否为 1~59，否则报错；
     rtc_tm.tm_year=time.year;          %将 time.year 赋值给 rtc_tm.tm_year；
    rtc_tm.tm_mon=time.month;           %将 time. month 赋值给 rtc_tm.tm_ mon；
    rtc_tm.tm_mday=time.day;            %将 time. day 赋值给 rtc_tm.tm_ mday；
    rtc_tm.tm_hour=time.hour;           %将 time. hour 赋值给 rtc_tm.tm_ hour；
    rtc_tm.tm_min=time.min;             %将 time. min 赋值给 rtc_tm.tm_ min；
    rtc_tm.tm_sec=time.sec;                 %将 time. sec 赋值给 rtc_tm.tm_ sec；
    retval = ioctl(fd, RTC_SET_TIME, &rtc_tm);  %对 I/O 通道进行管理
    if (retval == -1)                   %判断 retval 是否为-1；
    {
        perror("ioctl");                %将输入的信息和现在的 error 所对应的错误一起输出；
        exit(errno);                    % erron=0 时，exit(0),退出进程；
    }
     close(fd);                         % fd 为 1 则关闭；
     //system("/bin/busybox hwclock --hctosys");    %
     return 0;                          %返回 0；
     }
int main()                              %定义 main 函数
{
        TimeInfo p;                     %定义 p 为 TimeInfo 类型；
        AlarmInfo a;                    %定义 a 为 AlarmInfo 类型；
        int choice;                     %定义 choice 为 int 类型；
        fprintf(stdout,"Now you can choose\n1 to select get_curtime\n2 to select set_curtime\n3 to get_alarmtime\n");
                    %输入判断条件；
         //fprintf(stdout,"input any digital    without zero to get time\n");
           // fprintf(stdout,"Your will get time:\n");
        scanf("%d",&choice);            %输入判断条件 choice；
        switch(choice)                  %使用 switch 语句来判断 choice；
        {
          case 1 :                      %当 choice=1 时；
       //           set_curtime(p);   %
                  get_curtime(p);       %调用函数 get_curtime；
                  break;                %跳出
          case 2 :                      %当 choice=2 时；
                  set_curtime(p);       %设置时间
                  get_curtime(p);       %调用函数 get_curtime；
                  break;                %跳出
```

```
            case 3 :                    %当 choice=3 时；
                get_alarmtime(a);       %调用 get_alarmtime 函数；
                break;                  %跳出；
/*          case 4 :                    %当 choice=4 时；
              set_alarmtime(a);         %调用 set_alarmtime 函数；
                get_alarmtime(a); */
            }
          // if (choice)                %
          // get_curtime(p);            %
           return 0;                    %返回 0；
}
```

(3) rtc.x1205.c 注释：

```
/* 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 at gmail dot 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.*/                    %版本信息
#include <linux/module.h>                                          %调用头文件
#include <linux/i2c.h>
#include <linux/bcd.h>
#include <linux/rtc.h>
#include <linux/delay.h>
#define DRV_VERSION "1.0.7"                                        %定义 DRV_VERSION 为"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.*/    %版本信息
static struct i2c_driver x1205_driver;                             %定义静态变量结构体 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                       %定义 CCR_SEC 为 0；
#define CCR_MIN                 1                       %定义 CCR_MIN 为 1；
#define CCR_HOUR                2                       %定义 CCR_HOUR 为 2；
#define CCR_MDAY                3                       %定义 CCR_MDAY 为 3；
#define CCR_MONTH               4                       %定义 CCR_MONTH  为 4；
#define CCR_YEAR                5                       %定义 CCR_YEAR  为 5；
#define CCR_WDAY                6                       %定义 CCR_WDAY 为 6；
#define CCR_Y2K                 7                       %定义 CCR_Y2K 为 7；
#define X1205_REG_SR            0x3F /* status register */  %定义 X1205_REG_SR 为 0x3F；
#define X1205_REG_Y2K           0x37                    %定义 X1205_REG_Y2K 为 0x37；
#define X1205_REG_DW            0x36                    %定义 X1205_REG_DW 为 0x36；
#define X1205_REG_YR            0x35                    %定义 X1205_REG_YR 为 0x35；
#define X1205_REG_MO            0x34                    %定义 X1205_REG_MO 为 0x34；
#define X1205_REG_DT            0x33                    %定义 X1205_REG_DT 为 0x33；
#define X1205_REG_HR            0x32                    %定义 X1205_REG_HR 为 0x32；
```

```
#define X1205_REG_MN          0x31                        %定义 X1205_REG_MN 为 0x31；
#define X1205_REG_SC          0x30                        %定义 X1205_REG_SC 为 0x30；
#define X1205_REG_DTR         0x13                        %定义 X1205_REG_DTR 为 0x13；
#define X1205_REG_ATR         0x12                        %定义 X1205_REG_ATR 为 0x12；
#define X1205_REG_INT         0x11                        %定义 X1205_REG_INT 为 0x11；
#define X1205_REG_0           0x10                        %定义 X1205_REG_0 为 0x10；
#define X1205_REG_Y2K1        0x0F                        %定义 X1205_REG_Y2K1 为 0x0F；
#define X1205_REG_DWA1        0x0E                        %定义 X1205_REG_DWA1 为 0x0E；
#define X1205_REG_YRA1        0x0D                        %定义 X1205_REG_YRA1 为 0x0D；
#define X1205_REG_MOA1        0x0C                        %定义 X1205_REG_MOA1 为 0x0C；
#define X1205_REG_DTA1        0x0B                        %定义 X1205_REG_DTA1 为 0x0B；
#define X1205_REG_HRA1        0x0A                        %定义 X1205_REG_HRA1 为 0x0A；
#define X1205_REG_MNA1        0x09                        %定义 X1205_REG_MNA1 为 0x09；
#define X1205_REG_SCA1        0x08                        %定义 X1205_REG_SCA1 为 0x08；
#define X1205_REG_Y2K0        0x07                        %定义 X1205_REG_Y2K0 为 0x07；
#define X1205_REG_DWA0        0x06                        %定义 X1205_REG_DWA0为 0x06；
#define X1205_REG_YRA0        0x05                        %定义 X1205_REG_YRA0 为 0x05；
#define X1205_REG_MOA0        0x04                        %定义 X1205_REG_MOA0为 0x04；
#define X1205_REG_DTA0        0x03                        %定义 X1205_REG_DTA0 为 0x03；
#define X1205_REG_HRA0        0x02                        %定义 X1205_REG_HRA0 为 0x02；
#define X1205_REG_MNA0        0x01                        %定义 X1205_REG_MNA0为 0x01；
#define X1205_REG_SCA0        0x00                        %定义 X1205_REG_SCA0 为 0x00；
#define X1205_CCR_BASE        0x30 /* Base address of CCR */          %定义 X1205_CCR_BASE 为 0x30；
#define X1205_ALM0_BASE       0x00 /* Base address of ALARM0 */       %定义 X1205_ALM0_BASE 为 0x00；
#define X1205_SR_RTCF         0x01 /* Clock failure */                %定义 X1205_SR_RTCF 为 0x01；
#define X1205_SR_WEL          0x02 /* Write Enable Latch */           %定义 X1205_SR_WEL 为 0x02；
#define X1205_SR_RWEL         0x04 /* Register Write Enable */        %定义 X1205_SR_RWEL 为 0x04；
#define X1205_DTR_DTR0        0x01                        %定义 X1205_DTR_DTR0 为 0x01；
#define X1205_DTR_DTR1        0x02                        %定义 X1205_DTR_DTR1 为 0x02；
#define X1205_DTR_DTR2        0x04                        %定义 X1205_DTR_DTR2 为 0x04；
#define X1205_HR_MIL          0x80 /* Set in ccr.hour for 24 hr mode */  %定义 X1205_HR_MIL 为 0x80；
/* Prototypes */
static int x1205_attach(struct i2c_adapter *adapter);                   %定义静态函数 x1205_attach；
static int x1205_detach(struct i2c_client *client);                     %定义静态函数 x1205_detach；
static int x1205_probe(struct i2c_adapter *adapter, int address, int kind);     %定义静态函数 x1205_probe；
static struct i2c_driver x1205_driver = {                               %定义驱动结构体；
        .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;
}                                                       %定义静态结构体函数 x1205_get_datetime；
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 */
        };
        /* 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;
}                                                       %定义静态结构体函数 x1205_get_status；
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;
}                                                                   %定义静态结构体函数 x1205_set_datetime；
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;
}                                                                   %定义静态结构体函数 x1205_fix_osc；
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;
}                                                                   %定义静态结构体函数 x1205_get_dtrim；
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;
}                                                                   %定义静态结构体函数 x1205_get_dtrim；
struct x1205_limit
{       unsigned char reg, mask, min, max;
};                                                                  %定义结构体 x1205_limit；
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,
    };                                                          %定义静态结构体函数 x1205_validate_client；
    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     },
    };                                                  %定义静态结构体函数 x1205_limit probe_limits_pattern[]；
    /* check that registers have bits a 0 where expected */     %检查寄存器的位 a 是否为 0；
    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) */         %检查限制(仅具有 BCD 值的寄存器)；
    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);
}                                                       %定义静态结构体函数 x1205_rtc_read_alarm；
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);
}                                                       %定义静态结构体函数 x1205_rtc_set_alarm；
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);
}                                                       %定义静态结构体函数 x1205_rtc_read_time；
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);
}                                                       %定义静态结构体函数 x1205_rtc_set_time；
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;
}                                                       %定义静态结构体函数 x1205_rtc_proc；
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,
};                                                      %定义静态结构体函数 rtc_class_ops x1205_rtc_ops；
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);
}                                                       %定义静态结构体函数 ssize_t x1205_sysfs_show_atrim；
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);
}                                                       %定义静态结构体函数 ssize_t x1205_sysfs_show_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);
}                                                       %定义静态结构体函数 x1205_attach；
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;
}                                                       %定义静态结构体函数 x1205_probe；
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;
}                                                       %定义静态结构体函数 x1205_detach；
static int __init x1205_init(void)
{
        return i2c_add_driver(&x1205_driver);
}                                                       %定义静态结构体函数__init x1205_init；
static void __exit x1205_exit(void)
{
        i2c_del_driver(&x1205_driver);
}                                                       %定义静态结构体函数__exit x1205_exit；
//MODULE_AUTHOR(
//      "Karen Spearel <kas111 at gmail dot com>, "
//      "Alessandro Zummo <a.zummo@towertech.it>");
//MODULE_DESCRIPTION("Xicor/Intersil X1205 RTC driver");
MODULE_LICENSE("GPL");                                  %模块许可证声明；
//MODULE_VERSION(DRV_VERSION);
module_init(x1205_init);                                %载入模块；
        module_exit(x1205_exit);                        %退出模块；
```