实验 17：RTC 时钟驱动实验@

一、实验目的

1.了解 RTC 工作原理；
2.掌握 RTC 驱动的编写；
3.掌握 RTC 驱动的加载过程及测试方法。

二、实验内容

1.学习 RTC 的工作原理;
2.编写 RTC 的驱动程序;
3.编写测试程序测试 RTC。

三、实验设备

1. 硬件：PC 机；基于 ARM9 系统教学实验系统；网线；串口线。
2. 软件：PC 机操作系统（Windows XP）；Linux 服务器；putty 串口软件；内核等相关软件包；
3. 环境： ubuntu12.04 系统。文件系统版本为 filesys_test、烧写的内核版本为 uImage_wlwzbs，驱动源码见代码文件夹，驱动生成的需要加载的.ko 文件是 rtc-x1205.ko

四、预备知识

1. 了解 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。

五、预备知识

1、概述

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。

2、实现的功能

（1）设置、读取硬件时间；
（2） 读取软件时间并保存；
（3）读取闹钟的时间。

3、基本原理

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)。

4、硬件平台构架

X1205 芯片模块硬件平台比较简单，集成电路如图 1。I2C 总线结构，外接 32.768KHz 的晶体。时钟/控制寄存器的地址范围为 0000H～003FH。

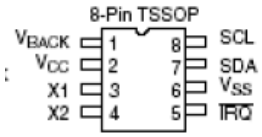


图 1 X1205 时钟芯片

X1，X2：外接石英晶体振荡器端。

IRQ：在应用报警功能时，该引脚输出中断信号，低电平有效。

SCL：由 DM365 给 X1205 提供的串行时钟的输入端。

SDA：数据输入/输出引脚。

GND：接地端。

VDD、VBAT：前者为芯片的工作电压，后者为备用电源。在实际应用中，通常可以接成如图二中所示的电路。在 VDD 与 VBAT 之间接二极管，在 VBAT 与地之间接电容。在正常供电情况下，VDD 给电容充电。掉电后，电容充当备用电源。在 VDD 掉电后，备用电源电流小于 2μA ，电容 C 用 10μF 的钽电解质电容亦可。

X1205 片内的数字微调寄存器 DTR（地址 0013H）的第 2、1、0 三位 DTR2、DTR1、DTR0 调整每秒钟的计数值和平均 ppm 误差。DTR2 是一个符号位，1 为正 ppm 补偿，0 为负补偿。DTR1 和 DTR0 是刻度位，DTR1 给出的是 10ppm 调整，DTR0 给出的是 20ppm 调整。通过这三位可以在-30ppm 至 +30ppm 范围内进行调整补偿。模拟微调寄存器 ATR（地址 0012H）的第 5 至第 0 位 ATR5、ATR4…和 ATR0 用来调整片内负载电容。ATR 值以补码形式表示，ATR（000000）=11.0pF，每步调节 0.25pF，整个调整范围从 3.25pF 至 18.75Pf。可以对额定频率提供从+116ppm 至-37ppm 的补偿。通过对 DTR 及 ATR 的调整，可以在

+146ppm 至-67ppm 范围内调整补偿。

电源控制电路认同一个 VCC 和一个 VBACK 输入电源控制电路在 VCC< VBACK–0.2V 时采用 VBACK 驱动时钟当 VCC 超过 VBACK 时切换回 VCC 给器件供电，如下图 2 所示；

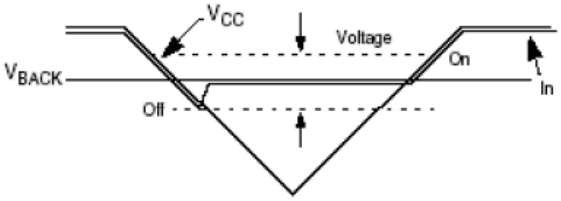


图 2 电源控制

5、软件系统架构

与 RTC 核心有关的文件有下面 6 个，结构图如下图 3 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。

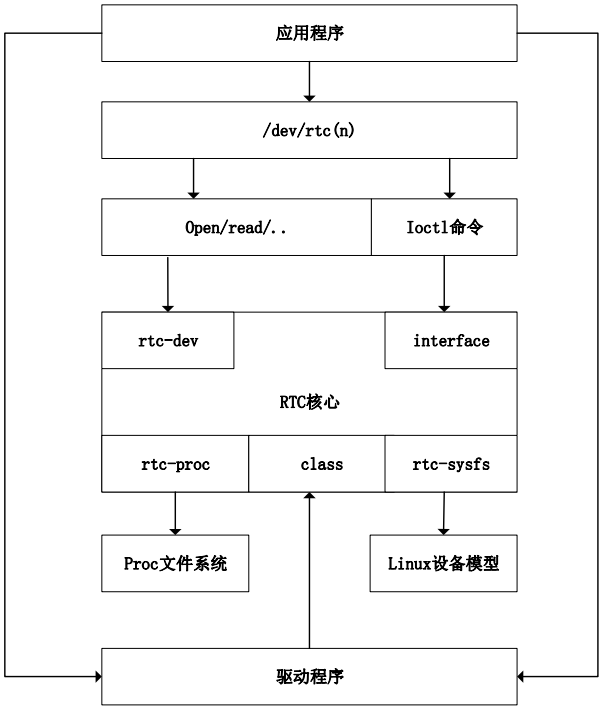


图 3 RTC 驱动结构模型

有兴趣的可以从内核中逐一找到学习，下面主要讲具体 RTC 驱动如何实现，RTC 驱动是挂载在 I2C 总线上，I2C 总线驱动相关知识参考 I2C 驱动部分：

RTC 驱动设计流程图如下图 4 所示：

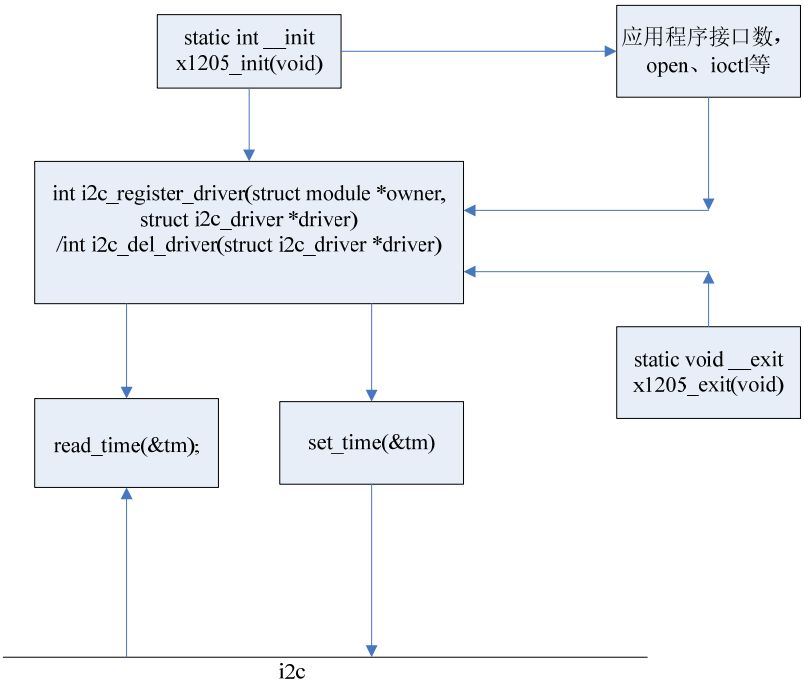


图 4 RTC 驱动流程图

主要函数及结构体说明：

1、函数：static int __init x1205_init(void)

函数功能：驱动的入口函数

函数参数：无

2、函数：static void __exit x1205_exit(void)

函数功能：驱动出口函数

函数参数：无

3、函数：static int x1205_detach(struct i2c_client *client)

函数功能：将驱动脱离 I2C 总线

函数参数：I2C 设备结构体

4、函数：static int x1205_probe(struct i2c_adapter *adapter, int address, int kind)

函数功能：在驱动和适配器匹配成功后，调用此函数进一步完成注册

函数参数：第一个为对应的 RTC 适配器，第二个设备地址，第三个参数是设备类型

5、函数：static int x1205_set_datetime(struct i2c_client *client, struct rtc_time *tm,
int datetoo, u8 reg_base)

函数功能：设置 RTC 硬件时间

函数参数：第一个参数为设备结构体，第二个参数为时间结构体，第三个参数是时间类型，第四个是寄存器地址

6、函数：static int x1205_get_datetime(struct i2c_client *client, struct rtc_time *tm,
unsigned char reg_base)

函数功能：获取 RTC 时硬件间

函数参数：第一个参数为设备结构体，第二个参数为时间结构体，第三个是寄存器地址

1、结构体 struct i2c_client {

```
unsigned int flags;         /* div., see below          */
unsigned short addr;            /* chip address - NOTE: 7bit        */
                        /* addresses are stored in the  */
                        /* _LOWER_ 7 bits               */
struct i2c_adapter *adapter;   /* the adapter we sit on   */
struct i2c_driver *driver;/* and our access routines      */
int usage_count;            /* How many accesses currently    */
                        /* to the client        */
struct device dev;          /* the device structure         */
```

```
    struct list_head list;
    char name[I2C_NAME_SIZE];
    struct completion released;
};
```

2、结构体 struct rtc_time

```
{
    int sec;    /* Seconds (0-59) */
    int min;    /* Minutes (0-59) */
    int hour;   /* Hour (0-23) */
    int dow;    /* Day of the week (1-7) */
    int dom;    /* Day of the month (1-31) */
    int month;      /* Month of year (1-12) */
    int year;   /* Year (0-99) */
};
```

3、结构体 struct i2c_adapter {

```
    struct module *owner;
    unsigned int id;
    unsigned int class;
    struct i2c_algorithm *algo;/* the algorithm to access the bus       */
    void *algo_data;

    /* --- administration stuff. */
    int (*client_register)(struct i2c_client *);
    int (*client_unregister)(struct i2c_client *);

    /* data fields that are valid for all devices      */
    struct mutex bus_lock;
    struct mutex clist_lock;

    int timeout;
    int retries;
    struct device dev;          /* the adapter device */
    struct class_device class_dev;/* the class device */

    int nr;
    struct list_head clients;
    struct list_head list;
```

```
    char name[I2C_NAME_SIZE];
    struct completion dev_released;
    struct completion class_dev_released;
}
```

4、结构体 static struct i2c_driver x1205_driver = {

```
    .driver         = {
        .name       = "x1205",
    },
    .id             = I2C_DRIVERID_X1205,
    .attach_adapter = &x1205_attach,
    .detach_client  = &x1205_detach,
};
```

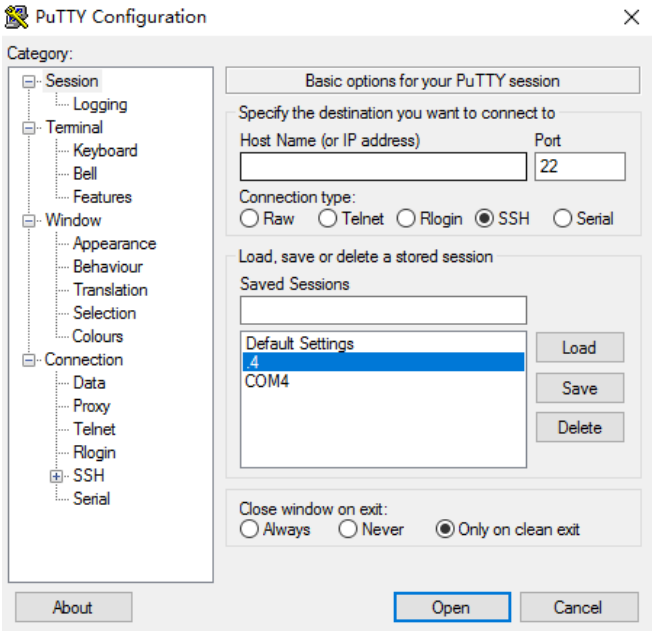
5、结构体 static struct rtc_class_ops x1205_rtc_ops = {

```
    .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,
};
```

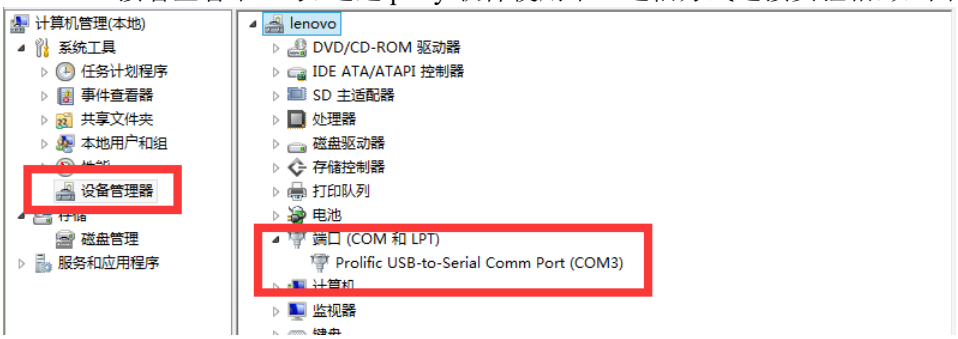
六、实验步骤

步骤 1：硬件连接

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



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



图端口号查询

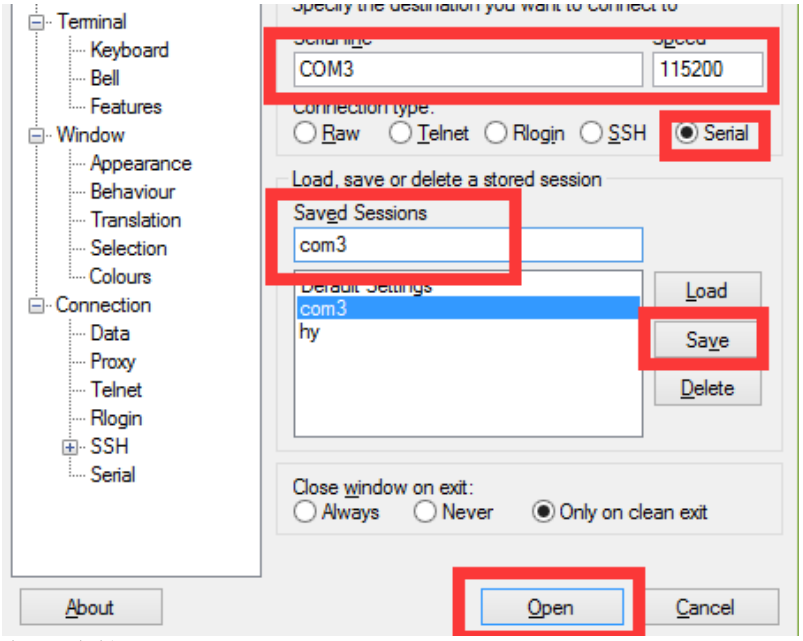


图 putty 串口连接配置

- 输入启动参数，接着启动内核，如下图所示：

```
DM365 EVM :>setenv bootargs mem=70M console=ttyS0,115200n8 root=/dev/nfs rw nfsr
oot=192.168.0.135:/home/st1/zh/filesys_test/ ip=192.168.0.109:192.168.0.135:192.
168.0.1:255.255.255.0::eth0:off eth=00:40:01:C1:56:19 video=davincifb:vid0=OFF:v
id1=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:    1995748 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)
```

- 输入用户名 root 登录实验箱如下图所示：

```
zjut login: root


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

login[754]: root login on 'console'
/******Set QT environment********/
[root@zjut ~]#
```

步骤 2：编写 RTC 驱动代码，使用 putty 工具登录服务器，在自己的目录下（/home/stx/）建一个文件夹 mkdir rtc，cd 进入 rtc 文件夹，使用 vim 编写 RTC 驱动程序 rtc-x1205.c；RTC 的驱动源码如下

```
/*
```

```
 * 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.
 */
#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 of CCR */
#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 */
        };

        /* 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);
```

步骤 3：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
```

编写 rtc-x1205.c 对应的 Makefile，编写好以后，make 一下生成 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/stx/filesys_test/modules 中（根据自己的目录改路径）；

```
$ cp rtc-x1205.ko /home/stx/filesys_test/modules
```

步骤 4：编写测试程序。

编写测试程序 rtc_test.c，具体代码如下所示：

```
#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, "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);
              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];                      //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);
                                                    }
                         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_alarmtime(a);
                get_alarmtime(a);           */
            }

        // if (choice)
        // get_curtime(p);

          return 0;
}
```

将编写好的测试程序在服务器上输入命令：

```
$ arm_v5t_le-gcc - o rtc_test rtc_test.c
```

把测试程序编译成可执行文件，然后将其放到文件系统 filesys_test 中 cp rtc_test /home/stx/filesys_test/opt/dm365 中（根据自己的目录改路径）；

```
$ cp rtc_test /home/stx/filesys_test/opt/dm365
```

步骤 5：挂载文件系统设置启动参数通过 NFS 方式挂载实验箱的根文件系统。

打开 putty 软件，启动实验箱，在内核启动倒计时 5 秒前按 enter 终止实验箱的启动，输入下面参数挂载文件系统，然后输入 boot 引导启动如图 5：

setenv bootargs 'mem=110M console=ttyS0,115200n8 root=/dev/nfs rw nfsroot=192.168.1.4:/home/stx/filesys_test/ ip=192.168.1.217:192.168.1.4:192.168.1.1:255.255.255.0::eth0:off eth= 00:40:01:C1:56:80 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'

注意：挂载文件系统根据自己的文件系统路径修改挂载路径（参考系统挂载部分的内容）

```
DM365 EVM :><INTERRUPT>
DM365 EVM :>setenv bootargs 'mem=110M console=ttyS0,115200n8 root=/dev/nfs rw nf
sroot=192.1655.0::eth0:off eth= 00:40:01:C1:56:80 video=davincifb:vid0=OFF:vid1=
OFF:osd0=640ngr.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:    1995748 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:    1995748 Bytes =  1.9 MB
   Load Address: 80008000
   Entry Point:  80008000
   Verifying Checksum ... OK
   Loading Kernel Image ...
```

图 5　挂载文件系统

步骤 6：手动加载驱动

启动完成后，输入 root 登录板子，进入到/modules 目录再使用 insmod rtc-x1205.ko 加载 RTC 驱动模块，如下图 6 所示；

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

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 ~]# cd /modules/
[root@zjut modules]# ls
davinci_dm365_gpios.ko  ov5640_i2c.ko           rtutil5572sta.ko
egalax_i2c.ko           rt5370sta.ko            ts35xx-i2c.ko
gpio                    rt5572sta.ko            ttyxin.ko
lcd.ko                  rtnet5572sta.ko
[root@zjut modules]#
```

图 6　进入 modules 目录

```
[root@zjut modules]# insmod rtc-x1205.ko
[  825.870000] x1205 0-006f: chip found, driver version 1.0.7
[  825.870000] x1205 0-006f: rtc intf: proc
[  825.880000] x1205 0-006f: rtc intf: dev (254:0)
[  825.880000] x1205 0-006f: rtc core: registered x1205 as rtc0
[root@zjut modules]#
```

图 7　加载 modules 模块

步骤 7：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 7 所示；

```
[root@zjut modules]# cd /opt/dm365
[root@zjut dm365]# ls
3g_guard.sh        daemon           i2c_test_5151_1    recv
ALSA               data             i2c_test_8bit      rtc_test
Config.dat         dec_zh.sh        image              script
```

图 7　查找测试程序 rtc_test

步骤 8：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。例如：设置 RTC 的时间，如下图 8 所示；（时间输入 2 分 5 秒后还有一个 0 用于结束获取字符，即在输入指定年月日时分秒后还需输入一个随意字符即可结束输入状态）

```
[root@zjut dm365]# ./rtc_test
Now you can choose
1 to select get_curtime
2 to select set_curtime
3 to get_alarmtime
2
first is year
second is month
third is day
fourth is hour
fifth is minute
sixth is second
2012
3
4
11
2
5
0
Current RTC date/time is
 2012-3-4, 11:02:05.
[root@zjut dm365]#
```

图 8　设置 RTC 时间

步骤 9：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，如下图 9 所示；

```
5
0
Current RTC date/time is
 2012-3-4, 11:02:05.
[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 12:12:50 2016  0.000000 seconds
[root@zjut dm365]#
```

图 9 设置任意 RTC 时间