uC/OS实验：万年历设计

王跃恒PB11210191

June 2, 2014

Abstract

基于uC/OS-II操作系统在三星S3C2440开发板上进行设计，完成时间的实时显示、时间设置和闹钟提醒等功能。部分模块未测试。本实验报告叙述了实际的设计过程。

1 模块划分

首先要进行模块划分。在硬件上，使用LED数码管进行时间的显示，使用四个按钮进行时间和闹钟设置，时间基准使用开发板上已集成的RTC模块。由此，可以把编程分为几个模块：

- RTC设置与读取：用于读取和设置时间；
- 数码管显示：用于显示当前时间；
- 按钮设置：用于捕获按钮活动；
- 闹钟：用于设置闹钟并控制蜂鸣器；
- 顶层模块：连接所有其他模块。

下面是这几个模块的分析与实现。

2 RTC设置与读取

RTC是开发板上的一个硬件实时时钟模块，即使没有外部电源供电也可以持续运转。

2.1 RTC读取

直接读对应的寄存器既可以读取当前时间，可以采用如下的方式：

```
while(1){
    while(1){
            year    = 0x2000 + rBCDYEAR;
            month   = rBCDMON;
            weekday = rBCDDAY;
            date    = rBCDDATE;
            hour    = rBCDHOUR;
            min     = rBCDMIN;
            sec     = rBCDSEC;

            if(sec!=tmp)
            {
                    tmp = sec;
                    break;
            }

    }//time updated
    //print out current time
}
```

当跳出里层的while循环时，说明“秒”变化了，即可认为时间更新了，当前的时间被存在几个变量中，然后可以进行输出等下一步操作。

上面采用的是阻塞查询方式来获知时间更新的时刻，更好的是使用S3C2440的RTC的功能：TickTimeInterrupt。可以在指定的周期中产生中断请求：

TICNT	Bit	Description	Initial State
TICK INT ENABLE	[7]	Tick time interrupt enable. 0 = Disable 1 = Enable	0
TICK TIME COUNT	[6:0]	Tick time count value (1~127). This counter value decreases internally, and users cannot read this counter value in working.	000000

Period = (n+1) / 128 second
n: Tick time count value (1~127)

这里把TICNT设为0xff即可实现每秒一次中断请求：

```
pISR_TICK = (unsigned)Rtc_Tick;
rTICNT    = (1<<7) + 127;
rINTMSK   = ~(BIT_TICK);

void __irq Rtc_Tick(void){
    rSRCPND = BIT_TICK;
    rINTPND = BIT_TICK;
    rINTPND;
    Uart_Printf("%03d sec",sec_tick++);
}
```

2.2 RTC设置

开启RTC时间设置位后，直接写对应的寄存器即可以进行时间设置：

```
    rRTCCON  = rRTCCON  & ~(0xf)  | 0x1;  //RTC Control enable

    rBCDYEAR = ((syear/10)<<4)+(syear%10);
    rBCDMON  = ((smonth/10)<<4)+(smonth%10);
    rBCDDAY  = sday; //SUN:1 MON:2 TUE:3 WED:4 THU:5 FRI:6 SAT:7
    rBCDDATE = sdate;
    rBCDHOUR = shour;
    rBCDMIN  = smin;
    rBCDSEC  = ssec;

    rRTCCON = 0x0;
```

这样就可以完成RTC时间的读取和设置了。

3 数码管显示

LCD显示屏设置较复杂，这里使用了8个七段数码管来显示当前的时间。开发板上的数码管管脚并没有直接和MCU的管脚直接相连，而是使用了另外的从MCU，用来控制键盘和数码管，并且通过IIC与主机进行通信。所以使用数码管首先要初始化IIC接口。然后就可以更直接地控制数码管了。如下：

```
void xzSetSeg(U8 pos, U8 seg[8])
{
    U8 i;
    IIC_Init();
    for(i=0;i<8;i++){
        if((pos & (1<<(7-i))) != 0){
            _WrIIC(0x70,0x17-i,DTab[seg[i]]);
        }
    }
    IIC_Restore();
```

```
}
```

上面的函数可以进行数码管显示设置，pos的每一位指示刷新哪些个位置，seg里则是刷新为哪个数字。

这样就可以操作数码管进行时间显示了。

4 按钮设置

开发板上有4个触电按钮，我们用这四个按钮实现对时间和闹钟的设置。

可以采用外部中断的方式，把按钮设置为下降沿中断，在中断处理程序中进行进一步地处理。设置某一个按钮的中断如下：

```
rGPFCON = ((rGPFCON  & 0xff00) | (1<<1));//PF0 EINT Enabled
rEXTINT0 |= (0x2<<0);//PF0 Falling Edge
pISR_EINT0=(U32)Eint0Int;//ISR entry
rINTMSK &=~(1<<0);//unmask EINT0

static void __irq Eint0Int(void)
{
    OSIntEnter();
    ClearPending(BIT_EINT0);
    //do something
    OSIntExit();
}
```

也可以采用轮询的方式确定是否有哪个按钮被按下了，把几个按钮对应的管脚置为输入，当按钮没有被按下时，为高电平，被按下后则为低电平。如下：

```
U8 key0,key1,key2,key3;
while(1){
    if((rGPFDAT & 0x01)!=0){key0=0;}else{key0=1;}
    if((rGPFDAT & 0x04)!=0){key1=0;}else{key1=1;}
    if((rGPGDAT & 0x08)!=0){key2=0;}else{key2=1;}
    if((rGPGDAT & (0x01<<11))!=0){key3=0;}else{key3=1;}
    if(key0 || key1 || key2 || key3){//some key down
        if(key0==1){
            //key0 down
        }else if(key1==1){
            //key1 down
        }else if(key2==1){
            //key2 down
        }else{
            //key3 down
        }
    }else{
        OSTimeDly(15);//no key down
    }
}
```

这样可以在按钮被按下时做相应的处理了，较之于中断，设置简便。而且当按钮被一直按着时，就会在延迟15个tick后再次进入处理，这是更加符合实际的（当一直按着按钮，数字快速变化，而不是要松开后再按下才变化）。

5 闹钟

闹钟可以分为两小块，第一是判断何时该响，第二是怎么响。

5.1 判断响铃时间

最简单的，每次时间更新的时候，判断是否与设置的闹钟时间相同，如果相同则响铃。如下：

```
    //test alarm
    if(Alarm_hour%10==hour%16 && Alarm_hour/10 == hour/16){
        if(Alarm_min%10==min%16 && Alarm_min/10==min/16){
            //test ok alarm
            OSSemPost(Sem_for_Alarm);
        }
    }
```

或者，RTC本身就自带“闹钟”：AlarmFunction。

ALARM FUNCTION

The RTC generates an alarm signal at a specified time in the power-off mode or normal operation mode. In normal operation mode, the alarm interrupt (INT_RTC) is activated. In the power-off mode, the power management wakeup (PMWKUP) signal is activated as well as the INT_RTC. The RTC alarm register (RTCALM) determines the alarm enable/disable status and the condition of the alarm time setting.

而且可以设置如何触发：小时、星期、月份。当对应的被置位时，当全部匹配就会触发中断。

RTCALM	Bit	Description	Initial State
Reserved	[7]		0
ALMEN	[6]	Alarm global enable. 0 = Disable, 1 = Enable	0
YEAREN	[5]	Year alarm enable. 0 = Disable, 1 = Enable	0
MONREN	[4]	Month alarm enable. 0 = Disable, 1 = Enable	0
DATEEN	[3]	Date alarm enable. 0 = Disable, 1 = Enable	0
HOUREN	[2]	Hour alarm enable. 0 = Disable, 1 = Enable	0
MINEN	[1]	Minute alarm enable. 0 = Disable, 1 = Enable	0
SECEN	[0]	Second alarm enable. 0 = Disable, 1 = Enable	0

6 顶层模块

下面要做的就是建立几个uC/OS任务把几个分模块连接起来。这里使用了四个任务：

- Task_Display：更新数码管显示；
- Task_RTC：时间更新、闹钟测试；
- Task_Proc_Key：处理按键；
- Task_Beep：用来控制蜂鸣器。

并且创建了两个信号量：

- Sem_for_update：时间更新。
- Sem_for_Alarm：闹钟响。

用于等待把新时间显示到数码管的任务如下：

```
void Task_Display(void *pdata)
{
    U8 Reply=0;
    while(1)
    {
         OSSemPend(Sem_for_update , 0, &Reply);//get sem 1 and convert
         if(Reply==OS_NO_ERR){
```

```
            Uart_Printf(" no err ");
        }
        xzSetSeg(Pos,SegData);
        OSTimeDly(3);
    }
}
```

当时间更新时，会释放Sem_for_update，当这个任务得到更新信号时就更新显示。
用于等待响铃的如下：

```
void Task_Beep(void *pdata){
    int i=0;
    U8 Reply=0;
    while(1){
        i++;
        OSSemPend(Sem_for_Alarm,0,&Reply);
        //Alarm!!!
        for(i=0;i<10;i++){
            Beep(2000,500);OSTimeDly(200);
        }
    }
}
```

当闹钟比较成功时，会释放Sem_for_Alarm信号量，被此任务获得，即响铃。
下面等待按键：

```
void Task_Proc_Key(void *pdata){
    U8 key0,key1,key2,key3;
    U8 func=0;
    U8 p1,p2,dat;
    U8 flag=0;
    while(1){
        if((rGPFDAT & 0x01)!=0){key0=0;}else{key0=1;}
        if((rGPFDAT & 0x04)!=0){key1=0;}else{key1=1;}
        if((rGPGDAT & 0x08)!=0){key2=0;}else{key2=1;}
        if((rGPGDAT & (0x01<<11))!=0){key3=0;}else{key3=1;}

        if(key0 || key1 || key2 || key3){
        //there is some key down
            if(key0==1){
                if(flag==1){//set alarm
                    if(func==0)Alarm_hour++;
                    if(func==1)Alarm_min++;
                }else{//set time
                    xz_Rtc_TimeSet(func,1,p1,p2);
                }
            }else if(key1==1){
                if(flag==1){
                    if(func==0)Alarm_hour--;
                    if(func==1)Alarm_min--;
                }else{//set time
                    xz_Rtc_TimeSet(func,2,p1,p2);
                }
            }else if(key2==1){
                func ++;
                func=func %3;
                Uart_Printf("Set:%d\n",func);
            }else if(key3==1){
                if(flag==0){
                    flag=1;
                    Uart_Printf("Alarm Set:");
                }else{
                    flag=0;
                    Uart_Printf("Alarm Set Done:%d:%d\n",Alarm_hour,Alarm_min);
```

```
                }
            }
            OSTimeDly(15);
        }else{
        //no key down
            OSTimeDly(15);
            ;

        }
    }
}
```

上面修改了几个全局变量，比如闹钟时间。
最后就是一直运行的时间更新与闹钟测试了：

```
void Task_RTC(void *pdata){
    while(1){
        Display_Rtc();
    }
}
```

其中的函数，Display_Rtc里，每次时间更新都会刷新显示并且测试闹钟，如下：

```
void Display_Rtc(void){
    while(1){
        while(1){
            get current time
            if(time updated){
                break;
            }
        }
        //time updated
        OSSemPost(Sem_for_update);
        if(Compare Alarm time OK){
            OSSemPost(Sem_for_Alarm);
        }
    }
}
```

至此，带有闹钟的简易万年历设计完成。

7 总结

虽然部分模块没有进行测试，但是通过该实验，更加深刻理解了uC/OS的原理和工作方式，学会了其中的任务创建、信号量、中断等的使用；学会了三星S3C2440各部分模块的设置与使用，收获很大。