Application Note

ATK-MB027 模块使用说明

RTC 模块

使用说明

正点原子

广州市星翼电子科技有限公司

修订历史

版本	日期	原因
V1.0	2024/11/01	第一次发布

目 录

1，硬件连接

这里以正点原子 M48Z-M3 最小系统板 STM32F103 版为例，给大家介绍一下模块和板卡的连接方法。其它板卡与模块的硬件连接方法，请大家在“**ATK-MB027 RTC 模块\3，程序源码\相应板卡例程文件夹\readme.txt**”路径下查看。

RTC 模块可通过杜邦线与正点原子 M48Z-M3 最小系统板 STM32F103 版进行连接，具体的连接关系，如下表所示：

模块对应开发板	连接关系					
RTC 模块	VCC	GND	SDA	SCL	INT	F32K
M48Z-M3 最小系统板 STM32F103 版	3.3V/5V	GND	PA3	PA2	PB5	NC

表 1.1.1 RTC 模块与 M48Z-M3 最小系统板 STM32F103 版连接关系

2，实验功能

2.1 RTC 模块测试实验

2.1.1 功能说明

在本实验中，串口会实时打印 RTC 时间。需要查看这部分实验信息的用户，可用杜邦线将最小系统板 STM32F103 的 PA9 引脚和 GND 连接至外部的 USB 转串口设备，这样就可以通过 XCOM 上位机查看串口打印的信息了。

本实验设置秒闹钟中断，每当秒钟数数到 30 秒时会触发闹钟中断，使 INT 脚输出低电平。并在串口打印提示信息。

开发板的 LED0 闪烁，提示程序运行。

2.1.2 源码解读

打开本实验的工程文件夹，能够在./Drivers/BSP 目录下看到 ATK_RTC 文件夹，和 IIC 文件夹，其中 ATK_RTC 文件夹中就包含了 RTC 模块的驱动文件，IIC 文件夹中就包含了软件模拟 IIC 的驱动文件如下图所示：

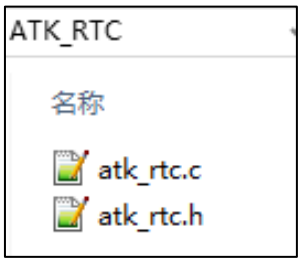

图 2.1.2.1 RTC 模块驱动代码

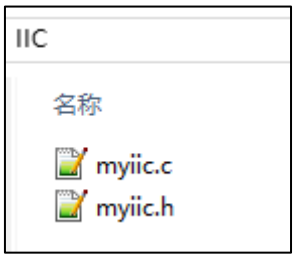

图 2.1.2.2 IIC 驱动代码

2.1.2.1 RTC 模块驱动

下面将简要介绍 atk_rtc.c 中几个重要的 API 函数。

1. 函数 atk_rtc_init 和 atk_rtc_init_time ()

这两个函数用于初始化 RTC 模块的时间以及初始时间设置功能，具体的代码，如下所示：

```
/**
 * @brief       初始化 SD3078
 * @param       无
 * @retval      检测结果
 *              0：检测成功
```

```
 *            1: 检测失败
 */
uint8_t atk_rtc_init(void)
{
    uint8_t sd3078_id[8], res = 0;
    uint8_t temp = 0;
    uint16_t vol = 0;
    /* 初始化 IIC 接口 */
    iic_init();

    /* 读取设备 ID */
    res = atk_rtc_read_device_id(sd3078_id);
    if((res == 0) && (sd3078_id[1] != 0))   /* 读取成功并且设备 ID 月份不等于 0 */
    {
        printf ("device id = %x-%x-%x-%x-%x-%x-%x-%x\r\n", sd3078_id[0],
        sd3078_id[1], sd3078_id[2], \
              sd3078_id[3], sd3078_id[4], sd3078_id[5], sd3078_id[6],
              sd3078_id[7]);
    }
    else
    {
        return 1;
    }
    atk_rtc_battery_charge_enable();        /* 使能充电 */
    atk_rtc_init_time();                    /* 设置初始时间 */

    res = atk_rtc_get_temperature(&temp);   /* 获取 RTC 芯片温度值 */
    if(res == 0)
    {
        printf("temperature :%d ℃\r\n",temp);
    }
    res = atk_rtc_get_battery_voltage(&vol);/* 读取备用电池电压值 */
    if(res == 0)
    {
        printf("battery voltage: %.2fv \r\n", (float)vol / 100.0f);
    }
    return res;
}

/**
 * @brief   设置 RTC 模块的初始时间
 * @param   无
 * @retval  结果
 * @arg     0: 成功
```

```
 * @arg     1: 失败
 */
uint8_t atk_rtc_init_time(void)
{
    uint8_t res = 0;
    uint8_t rtc_init_flag = 0;
    sd3078_time_t time_buf;

    /* 防止复位重复初始化 RTC 时间 */
    res = sd3078_read_ram(SD3078_REG_USER_RAM_START, &rtc_init_flag, 1);
    if(rtc_init_flag != 0xBB)
    {
        /* 设置初始时间为：2024 年 10 月 11 号，16 点 49 分 50 秒 星期五 */
        time_buf.second = 50;
        time_buf.minute = 49;
        time_buf.hour = 16;
        time_buf.week = 5;
        time_buf.day = 11;
        time_buf.month = 10;
        time_buf.year = 24;
        time_buf.hour_type = SD3078_HOUR_24;
        time_buf.am_pm = SD3078_AM_PM_NULL;
        res = atk_rtc_set_time(&time_buf);
        if (res == 0)
        {
            rtc_init_flag = 0xBB;
            sd3078_write_ram(SD3078_REG_USER_RAM_START, &rtc_init_flag, 1);
            printf("sd3078 set time success!\n");
            return 0;
        }
    }
    return res;
}
```

从上述代码可以看出，首先 atk_rtc_init_time 用于设置初始时间，这里会通过读取 RAM 数据判断是不是第一次初始化 RTC 模块，避免下次复位又重新设置时间，这里我们将初始时间设置为**2024 年 10 月 11 日，16:49:50，星期五**，并采用 24 小时制。通过 atk_rtc_set_time() 写入时间，如果成功，则将标志位写为 0xBB，防止后续重复初始化。

atk_rtc_init 函数用于初始化 RTC 模块。首先，它初始化 IIC 接口，并读取设备 ID 以确保通信正常。接着，启用电池充电功能，并调用 atk_rtc_init_time 设置初始时间。随后，函数会读取温度值和备用电池电压。需要注意的是，在 VDD 供电的情况下，温度值每 60 秒更新一次，因此即使芯片温度上升，温度读数也会在 60 秒内保持不变，这属于正常现象。

2. 函数 bcd_to_hex 和 hex_to_bcd ()

这两个函数实现了 BCD（Binary-Coded Decimal）格式和十六进制（Hexadecimal）格式

之间进行转换，具体代码，如下所示：

```
/**
 * @brief   将 BCD 数据格式转成 16 进制数据
 * @param   bcd：BCD 格式的数据
 * @retval  转换结果
 */
static uint8_t bcd_to_hex(uint8_t bcd)
{
    return (bcd >> 4) * 10 + (bcd & 0x0f);
}

/**
 * @brief   将 16 进制数据转成 BCD 数据格式
 * @param   hex：16 进制格式的数据
 * @retval  转换结果
 */
static uint8_t hex_to_bcd(uint8_t hex)
{
    return ((hex / 10) << 4) | (hex % 10);
}
```

上述代码的转换过程是比较简单的，由于 SD3078 实时时钟芯片时间设置和读取需使用到 BCD 码，所以需使用到这两个函数。转换过程这里不多做介绍，大家不熟悉的话可以在网上搜下 BCD 和 HEX 之间的转换关系。

3. 函数 atk_rtc_set_time 和 atk_rtc_get_time ()

这两个函数实现了 RTC 模的时间设置和读取功能。具体的代码，如下所示：

```
/**
 * @brief   设置 RTC 实时时钟数据
 * @param   *rtc_time ： 时间结构体指针
 * @retval  结果
 * @arg     0：成功
 * @arg     1：失败
 */
uint8_t atk_rtc_set_time(sd3078_time_t *rtc_time)
{
    /* 用于存放秒、分、小时、星期、日、月、年 */
    uint8_t data[7];
    /* 填充数组，确保每个字段对应正确的寄存器地址 */
    data[0] = hex_to_bcd(rtc_time->second);          /* second */
    data[1] = hex_to_bcd(rtc_time->minute);          /* minute */
    data[2] = hex_to_bcd(rtc_time->hour);            /* hour  */
    if (rtc_time->hour_type == SD3078_HOUR_24)
    {
        data[2] |= SD3078_12_24_BIT;     /* 如果为 24 小时制，则设置为 24 小时制 */
    }
```

```
    else if(rtc_time->am_pm == SD3078_PM)
    {
        data[2] |= SD3078_AM_PM_BIT;
    }
    data[3] = hex_to_bcd(rtc_time->week);           /* week */
    data[4] = hex_to_bcd(rtc_time->day);            /* day */
    data[5] = hex_to_bcd(rtc_time->month);          /* month */
    data[6] = hex_to_bcd(rtc_time-> year);          /* year */

    atk_rtc_write_enable();                         /* 写使能 */
     /* 设置时间数据 */
    if (atk_rtc_write_nbytes(SD3078_REG_SEC, data, sizeof(data)) != 0)
    {
        atk_rtc_write_disable();
        return 1;
    }
    atk_rtc_write_disable();                        /* 写禁止 */

    return 0;
}

/**
 * @brief   读取 RTC 实时时间
 * @param   *rtc_time :  存储时间结构体指针
 * @retval  结果
 * @arg     0: 成功
 * @arg     1: 失败
 */
uint8_t atk_rtc_get_time(sd3078_time_t *rtc_time)
{
    uint8_t data[7];
    if (atk_rtc_read_nbytes(SD3078_REG_SEC, data, sizeof(data)) != 0)
    {
        return 1;   /* 读取失败 */
    }
    /* 将读取到的数据填充到 time_t 结构体中 */
    rtc_time->second = bcd_to_hex(data[0] & 0x7f);      /* 对应秒 */
    rtc_time->minute = bcd_to_hex(data[1] & 0x7f);      /* 对应分 */
    if ((data[2] & SD3078_12_24_BIT) != 0)
    {
        rtc_time->am_pm = SD3078_AM_PM_NULL;
        rtc_time->hour_type = SD3078_HOUR_24;
        rtc_time->hour = bcd_to_hex(data[2] & 0x3f);    /* 对应小时 */
    }
```

```
    else
    {
        if ((data[2] & SD3078_AM_PM_BIT) != 0)
        {
            rtc_time->am_pm = SD3078_PM;
        }
        else
        {
            rtc_time->am_pm = SD3078_AM;
        }
        rtc_time->hour_type = SD3078_HOUR_12;
        rtc_time->hour = bcd_to_hex(data[2] & 0x1f);
    }
    rtc_time->week   = bcd_to_hex(data[3] & 0x07);      /* 对应星期 */
    rtc_time->day    = bcd_to_hex(data[4] & 0x3f);      /* 对应日 */
    rtc_time->month  = bcd_to_hex(data[5] & 0x1f);      /* 对应月 */
    rtc_time->year   = bcd_to_hex(data[6]);             /* 对应年 */
    return 0;
}
```

函数 atk_rtc_set_time 用于设置时间，首先将需要设置的时间数据转换为 BCD 格式。然后设置为 24 小时制，开启写使能，然后将时间数据写入到实时时钟数据寄存器（0x00~0x06）。最后禁止写使能。防止误操作数据。

函数 atk_rtc_get_time 用于读取时间，首先将读取出来的 BCD 格式的数据转换为十进制，判断小时制，然后将读取的数据存入 rtc_time 结构体中，方便后续调用。

4. 函数 atk_rtc_set_alarm_interrupt ()

该函数用于设置 RTC 模块的报警中断功能（闹钟中断），具体的代码，如下所示：

```
/**
 * @brief   设置报警中断（闹钟功能）
 * @param   *alarm ： 报警中断结构体指针
 * @retval  结果
 * @arg     0: 成功
 * @arg     1: 失败
 */
uint8_t atk_rtc_set_alarm_interrupt(sd3078_time_alarm_t *alarm)
{
    uint8_t buf[8];

    buf[0] = hex_to_bcd(alarm->sec_a);
    buf[1] = hex_to_bcd(alarm->min_a);
    buf[2] = hex_to_bcd(alarm->hour_a);
    buf[3] = alarm->week_a & 0x7f;
    buf[4] = hex_to_bcd(alarm->day_a);
    buf[5] = hex_to_bcd(alarm->mon_a);
    buf[6] = hex_to_bcd(alarm->year_a);
```

```
    buf[7] = alarm->enable_a & 0x7f;    /* 报警中断类型，多个类型可以或起来 */

    atk_rtc_write_enable();             /* 写使能 */
    /* 设置闹钟日期时间 */
    if (atk_rtc_write_nbytes(SD3078_REG_ALARM_SEC, buf, 8) != 0)
    {
        atk_rtc_write_disable();
        return 1;
    }
    /* 读取 CTR2 的原始值 */
    if (atk_rtc_read_nbytes(SD3078_REG_CTR2, buf, 1) != 0)
    {
        atk_rtc_write_disable();
        return 1;
    }
    buf[0] = buf[0];
    buf[0] &= ~SD3078_INTS1;
    buf[0] |= SD3078_INTS0;

    if(alarm->ie_a == 1)
    {
        buf[0] |= SD3078_INTAE;
    }
    else
    {
        buf[0] &= ~SD3078_INTAE;
    }
    if(alarm->int_period == 1)
    {
        buf[0] |= SD3078_IM;
    }
    else
    {
        buf[0] &= ~SD3078_IM;
    }
    /* 设置 CTR2 寄存器 */
    if (atk_rtc_write_nbytes(SD3078_REG_CTR2, buf, 1) != 0)
    {
        atk_rtc_write_disable();
        return 1;
    }
    atk_rtc_write_disable();
    return 0;
}
```

首先将设置的闹钟中断时间，转成 BCD 格式并存入 buf，然后 enable_a 用于指定报警中断类型，支持秒中断、分中断、时中断、星期中断、日中断、月中断、年中断，具体详情可查阅 SD3078 的数据手册。然后写使能后，将 buf 的数据设置到报警寄存器。接着时钟中断模式为单事件报警模式。最后写禁止。其他函数大家可以打开源码，自行分析下。

2.1.2.2 IIC 驱动

在图 2.1.2.2 中，myiic.c 和 myiic.h 是开发板与 RTC 模块通讯而使用的模拟 IIC 驱动文件，关于模拟 IIC 的驱动介绍，请查看正点原子各个开发板对应的开发指南中模拟 IIC 对应的章节。

2.1.2.3 实验测试代码

实验的测试代码为文件 demo.c，在工程目录下的 User 子目录中。测试代码的入口函数为 demo_run()，具体的代码，如下所示：

```
/* 时间结构体缓存区 */
sd3078_time_t sd3078_time_result;

/**
 * @brief       闹钟测试函数
 * @param       无
 * @retval      无
 */
void atk_rtc_alarm_test(void)
{
    sd3078_time_alarm_t alarm;

    alarm.year_a = 24;
    alarm.mon_a = 9;
    alarm.day_a = 30;
    alarm.week_a = ALARM_WEEK_MON;  /* 星期一 */
    alarm.hour_a = 16;
    alarm.min_a = 19;
    alarm.sec_a = 30;
    /* 秒闹钟，每次计数的秒钟数为 30 秒时，会触发闹钟，拉低 INT 引脚 */
    alarm.enable_a = ALARM_SEC;
    alarm.ie_a = 1;                 /* 使能中断 */
    alarm.int_period = 0;           /* 单事件报警 */

    atk_rtc_set_alarm_interrupt(&alarm);
}


/**
 * @brief       例程演示入口函数
 * @param       无
 * @retval      无
 */
```

```
void demo_run(void)
{
    while(atk_rtc_init())                /* 检测 RTC 模块 */
    {
        printf("RTC Check Failed!\r\n");
        delay_ms(500);
        LED0_TOGGLE();                   /* 红灯闪烁 */
    }
    printf("RTC Ready!\r\n");
    atk_rtc_alarm_test();                /* 闹钟测试函数 */
    while (1)
    {
        /* 获取 RTC 数据 */
        if(atk_rtc_get_time(&sd3078_time_result) == 0)
        {
            printf("星期%d \r\n", sd3078_time_result.week);
            printf("日期: 20%d-%d-%d\r\n", sd3078_time_result.year,
            sd3078_time_result.month, sd3078_time_result.day);
            printf("时间: %d:%d:%d\r\n", sd3078_time_result.hour,
            sd3078_time_result.minute, sd3078_time_result.second);
            printf("\r\n");
        }

        /* 闹钟触发判断 */
        if(!atk_rtc_flag_get(SD3078_ALARM_TIME_FLAG))
        {
            printf("闹钟触发! \r\n");
            /* 清除标志位，INT 脚恢复高电平 */
            atk_rtc_flag_clear(SD3078_ALARM_TIME_FLAG);
        }
        delay_ms(1000);
        LED0_TOGGLE();
    }
}
```

整个测试代码的逻辑还是比较简单的，首先定义一个全局结构体变量 sd3078_time_result，用于存放读取到的 RTC 时间数据；并且我们还实现了一个闹钟测试函数 atk_rtc_alarm_test，该函数实现了秒中断的闹钟逻辑，闹钟会在秒数每计数到 30 秒时触发。

最后是我们的测试函数 demo，首先会调用初始化 RTC 函数 atk_rtc_init()检查 RTC 模块是否正常工作。如果检查失败，则打印错误信息，延时 500 毫秒并让 LED 闪烁。如果 RTC 初始化成功，打印 "RTC Ready!"，并调用 atk_rtc_alarm_test() 设置闹钟。最后在 while 循环中调用 atk_rtc_get_time(&sd3078_time_result) 获取当前 RTC 时间，并存入 sd3078_time_result 变量。如果成功，打印当前的星期、日期和时间。当闹钟触发时，同样也会在串口打印提示信息，并且还会清除闹钟标志，以使 INT 引脚恢复高电平。

2.1.3 实验现象

将 RTC 模块按照第一节“硬件连接”中介绍的连接方式与开发板连接，并将实验代码编译烧录至开发板中，本实验使用串口输出 RTC 实时时间信息，因此需将开发板的 PA9 连接至 DAP 虚拟串口（或 USB 转 TTL 模块）的 RX 引脚。并通过串口调试助手查看实验信息输出，如下图所示：

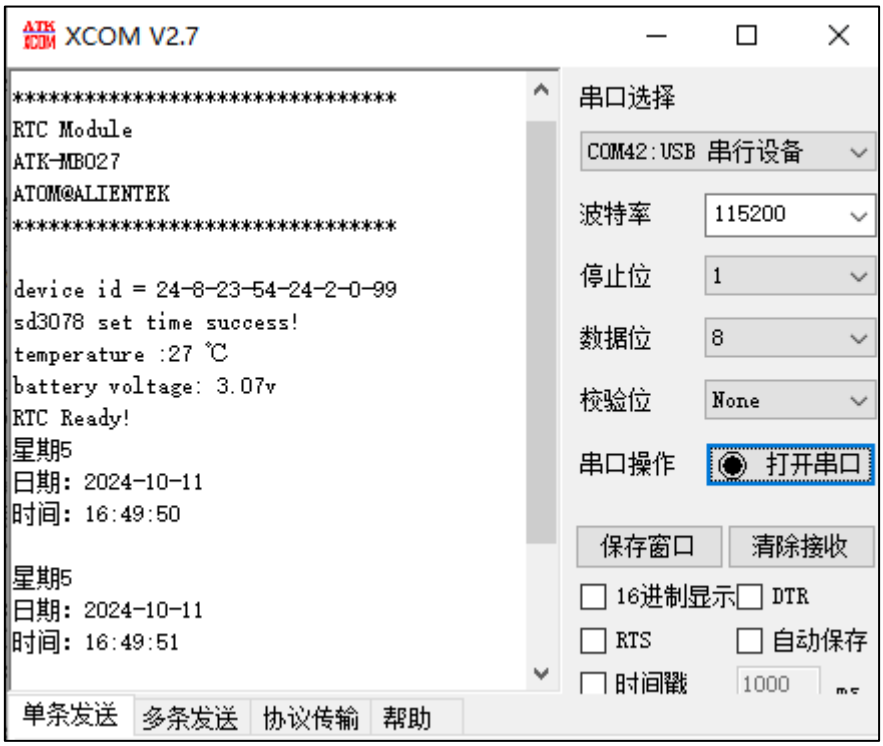

图 2.1.3.1 串口调试助手显示内容

可以看到，打印的提示信息包括：设备 ID、温度信息、备用电池电压值、然后就是 RTC 实时时间信息。那就代表实验正常！大家可以修改 atk_rtc_init_time 函数中的时间信息，更改当前的初始时间值。

3，其他

1、 购买地址：

天猫：https://zhengdianyuanzi.tmall.com

淘宝：https://openedv.taobao.com

2、 资料下载

模块资料下载地址：http://www.openedv.com/docs/index.html

3、 技术支持

公司网址：www.alientek.com

技术论坛：http://www.openedv.com/forum.php

在线教学：www.yuanzige.com

B 站视频：https://space.bilibili.com/394620890

传真：020-36773971

电话：020-38271790