Application Note

ATK-MB020 模块使用说明

心率血氧模块

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-MB020 心率血氧模块\3，程序源码\相应板卡例程文件夹\readme.txt**”路径下查看。

心率血氧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				
心率血氧模块	VCC	GND	SDA	SCL	INT
M48Z-M3 最小系统板 STM32F103 版	3.3V/5V	GND	PA3	PA2	PB5

表 1.1.1 心率血氧模块与 M48Z-M3 最小系统板 STM32F103 版连接关系

2，实验功能

2.1 心率血氧模块测试实验

2.1.1 功能说明

在本实验中，将手指或皮肤贴近心率血氧传感器表面，串口会实时打印当前的心率值和血氧值。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。

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_MAX30102 文件夹，和 IIC 文件夹，其中 ATK_MAX30102 文件夹中就包含了心率血氧模块的驱动文件和 FIR 滤波器驱动代码，IIC 文件夹中就包含了软件模拟 IIC 的驱动文件如下图所示：

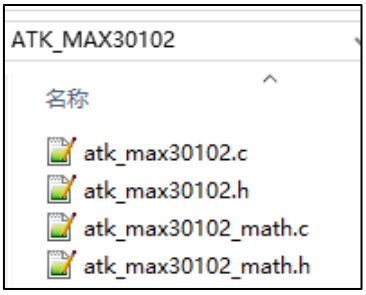

图 2.1.2.1 心率血氧模块驱动代码

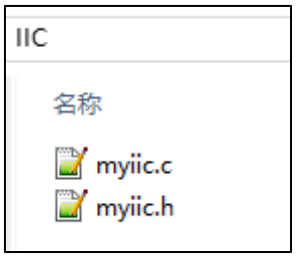

图 2.1.2.2 IIC 驱动代码

2.1.2.1 心率血氧模块驱动

下面将简要介绍 atk_max30102.c 和 atk_max30102_math.c 中几个重要的 API 函数。

1. 函数 atk_max30102_init()

该函数用于初始化心率血氧模块，具体的代码，如下所示：

```
/**
 * @brief   初始化 max30102
 * @param   无
 * @retval  无
 */
void atk_max30102_init(void)
```

```
{
    GPIO_InitTypeDef gpio_init_struct;

    INT_GPIO_CLK_ENABLE();
    /* INT 引脚初始化为外部中断线 */
    gpio_init_struct.Pin = INT_GPIO_PIN;
    gpio_init_struct.Mode = GPIO_MODE_IT_FALLING;   /* 下升沿触发 */
    gpio_init_struct.Pull = GPIO_PULLUP;            /* 上拉 */
    HAL_GPIO_Init(INT_GPIO_PORT, &gpio_init_struct);/* INT 配置为下降沿触发中断 */

    HAL_NVIC_SetPriority(INT_IRQn, 0, 2);           /* 抢占 0，子优先级 2 */
    HAL_NVIC_EnableIRQ(INT_IRQn);                   /* 使能中断线 5 */

    iic_init();                                     /* 初始化 I2C 接口 */

    atk_max30102_reset();                           /* 复位设备 */
    /* 中断使能：FIFO 满以及新 FIFO 数据就绪 */
    atk_max30102_write_byte(MAX30102_INTR_ENABLE_1, 0xC0);
    atk_max30102_write_byte(MAX30102_INTR_ENABLE_2, 0x00);
    /* 清空指针 FIFO_WR_PTR[4:0] */
    atk_max30102_write_byte(MAX30102_FIFO_WR_PTR, 0x00);
    /* 清空指针 OVF_COUNTER[4:0] */
    atk_max30102_write_byte(MAX30102_OVF_COUNTER, 0x00);
    /* 清空指针 FIFO_RD_PTR[4:0] */
    atk_max30102_write_byte(MAX30102_FIFO_RD_PTR, 0x00);
    /* 样本平均（4），FIFO 满滚（0），FIFO 几乎满值（发出中断时为 15 个空数据样本） */
    atk_max30102_write_byte(MAX30102_FIFO_CONFIG, 0x4F);
    /* 配置为 SpO2（血氧饱和）模式，会测量 RED 和 IR */
    atk_max30102_write_byte(MAX30102_MODE_CONFIG, 0x03);
    /* SpO2 配置：ADC 范围：4096nA，采样速率控制:200Hz，·LED 脉冲宽度:215us· */
    atk_max30102_write_byte(MAX30102_SPO2_CONFIG, 0x2A);
    /* IR LED 电流选择 9.4mA */
    atk_max30102_write_byte(MAX30102_LED1_PA, 0x2F);
    /* RED LED 电流选择 9.4mA */
    atk_max30102_write_byte(MAX30102_LED2_PA, 0x2F);

    /* 建议初始化时，通过读取将中断状态寄存器，将中断标志位清空 */
    atk_max30102_read_byte(MAX30102_INTR_STATUS_1);
    atk_max30102_read_byte(MAX30102_INTR_STATUS_2);
}
```

从上述代码可以看出，这段代码完成了 MAX30102 的初始化和配置工作，包括 GPIO 中断设置、FIFO 指针清空、传感器模式和 LED 参数的配置。它确保了传感器进入 SpO2 模式，并在新数据就绪或 FIFO 快满时触发中断，主控可以通过 I2C 接口读取数据寄存器。

2. 函数 atk_max30102_fifo_read()

这函数实现了读取 MAX30102 传感器的 FIFO 数据，将 RED 和 IR 两种光信号的数据解析并存储，具体代码，如下所示：

```
/**
 * @brief   读取 FIFO 数据
 * @param   red_data :    存储 RED 数据
 * @param   ir_data  :    存储 IR 数据
 * @retval  无
 */
void atk_max30102_fifo_read(float *red_data, float *ir_data)
{
    uint8_t receive_data[6];
    /* 通过读取将中断状态寄存器，将中断标志位清空 */
    atk_max30102_read_byte(MAX30102_INTR_STATUS_1);
    atk_max30102_read_byte(MAX30102_INTR_STATUS_2);

    atk_max30102_read_nbytes(MAX30102_FIFO_DATA, receive_data, 6);
    *red_data = ((receive_data[0] << 16 | receive_data[1] << 8 | receive_data[2])
    & 0x03ffff);  /* RED 数据 */
    *ir_data = ((receive_data[3] << 16 | receive_data[4] << 8 | receive_data[5])
    & 0x03ffff);   /* IR 数据 */
}
```

从上面的代码可以看出，在读取 RED 和 IR 数据之前，需要先通过读取中断状态寄存器来清除中断标志，确保不会发生意外的中断触发。由于模块被配置为 SpO2 模式，因此每次需要连续读取 6 个字节的数据（前 3 个字节为 RED 通道数据，后 3 个字节为 IR 通道数据），才能获取一个完整的数据样本。接着，通过拼接这 6 个字节的数据，获取 RED 和 IR 数据。

3. 函数 atk_max30102_get_heart()

该函数通过峰值检测算法来计算心率。具体的代码，如下所示：

```
/**
 * @brief   获取心率值
 * @param   input_data : 经过 FIR 滤波后的红外光数据
 * @param   cache_nums : 采样数量
 * @note    经过滤波的信号，心跳会呈现周期性的峰值（每个峰值代表一次心跳）
 *          设定一个阈值（一般为采集样本的平均值），当信号超过该阈值时，识别为峰值
 *          计算两个采样点的差值，为了准确性同样多次采集取平均 得到稳定的两次采样点差值（两次心跳的间隔时间 = 两个采样点差值/采样频率）
 *          计算心率公式（bpm 即每分钟跳动次数）：Heart_Rate = 60 * 采样频率 / 两个采样点差值
 * @retval  返回心率值
 * @retval  返回 0 代表识别失败，皮肤要紧贴并保持不动
 */
uint16_t atk_max30102_get_heart(float *input_data, uint16_t cache_nums)
{
    float input_data_sum_aver = 0;
```

```
    uint16_t i, temp;
    int num_peaks = 0;
    int current_peak_time = 0;
    int last_peak_time = 0;
    int total_intervals = 0;

    for(i = 0; i < cache_nums; i++)
    {
        input_data_sum_aver += *(input_data + i);
    }
    input_data_sum_aver = input_data_sum_aver / cache_nums; /* 获得阈值 */

    for(i = 0; i < cache_nums; i++)
    {
        /* 检测峰值 */
        if((*(input_data + i) > input_data_sum_aver) && (*(input_data + i + 1) <
          input_data_sum_aver))
        {
            current_peak_time = i;                    /* 记录当前采样点 */
            if(last_peak_time != 0)
            {
                temp = current_peak_time - last_peak_time; /* 计算两个采样点间的差 */
                total_intervals += temp;                  /* 差累加 */
                num_peaks++;                              /* 次数累加，后续取平均数用 */
            }
            /* 记录上次采样点，以便下次计算两个峰值采样点的差 */
            last_peak_time = current_peak_time;
        }
    }

    /* 单位：bpm(次/min) 心率 = 60 * sampling_rate / 两个采样点间的差值   其中
    sampling_rate = 采样率/样本数：  200/4 = 50HZ */
    if(num_peaks > 0)
    {
        /* 得到两个峰值间的平均差 */
        int avg_interval = total_intervals / num_peaks;
        return (3000 / avg_interval);             /* 计算心率 */
    }
    return 0;
}
```

从上述代码可以看出，首先通过传入经过 FIR 滤波器处理后的 IR 采样数据数组及其样本数量，接着使用峰值检测算法来计算心率。具体方法是计算数据样本的平均值，并将其作为阈值。当数据超过此阈值时，即可判定为峰值。在检测到至少一个峰值对后，代码计算所有峰值之间的平均时间间隔，并利用心率计算公式得出最终的心率值。

4. 函数 atk_max30102_get_spo2()

该函数主要实现了计算血氧饱和度（SpO2），具体的代码，如下所示：

```
/**
 * @brief   获取血氧值
 * @param   ir_input_data : 经过 FIR 滤波后的红外光数据
 * @param   red_input_data : 经过 FIR 滤波后的红光数据
 * @param   cache_nums : 采样数量
 * @note    首先通过对红光和红外光信号的最大最小值，计算其 AC 和 DC 分量
 *          然后通过公式计算出 R = (ACred/DCred) / (ACir/DCir) ;
 *          最后根据 R 值使用公式 SpO2 = aR^2 + bR + c  其中 a b c 是根据大量数据拟合得出
            的系数。
 * @retval  血氧值
 */
float atk_max30102_get_spo2(float *ir_input_data, float *red_input_data,
       uint16_t cache_nums)
{
    float ir_max = *ir_input_data, ir_min = *ir_input_data;
    float red_max = *red_input_data, red_min = *red_input_data;
    float R;        /* 比率 R */
    uint16_t i;

    /* 寻找红光和红外光的最大值和最小值 */
    for(i = 1; i < cache_nums; i++)
    {
        if(ir_max < *(ir_input_data + i))
        {
            ir_max = *(ir_input_data + i);
        }
        if(ir_min > *(ir_input_data + i))
        {
            ir_min = *(ir_input_data + i);
        }
        if(red_max < *(red_input_data + i))
        {
            red_max = *(red_input_data + i);
        }
        if(red_min > *(red_input_data + i))
        {
            red_min = *(red_input_data + i);
        }
    }

    /* (ir_max - ir_min) 和 (red_max - red_min) 表示：红外和红光信号的 AC 分量（脉动成
    分）。 */
```

```
    /* (red_max + red_min) 和 (ir_max + ir_min)：表示：红光和红外信号的 DC 分量（静态
    成分） */
    R = ((ir_max + ir_min) * (red_max - red_min)) / ((red_max + red_min) * (ir_max
    - ir_min));
    return ((-45.060f) * R * R + 30.354f * R + 94.845f);    /* SpO2 = aR^2 + bR
    + c */
}
```

从上述代码可以看出，首先使用一个循环遍历输入的 IR 和 RED 数据，更 ir_max、ir_min、red_max 和 red_min。通过比较当前元素与已有最大值和最小值，找出数组中红光和红外光的最大和最小值，然后通过 AC 分量和 DC 分量计算比率 R，最后代入血氧饱和浓度公式，SpO2=−45.060 * R^2 + 30.354 * R + 94.845，获得血氧饱和度 SpO2。

5. 函数 **max30102_fir_init/ir_max30102_fir/red_max30102_fir()**

```
/**
 * @brief   初始化 FIR 滤波器
 * @param   无
 * @retval  无
 */
void max30102_fir_init(void)
{
    arm_fir_init_f32(&S_ir, NUM_TAPS, (float32_t *)&firCoeffs32LP[0],
&firStateF32_ir[0], blockSize);
    arm_fir_init_f32(&S_red, NUM_TAPS, (float32_t *)&firCoeffs32LP[0],
&firStateF32_red[0], blockSize);
}

/**
 * @brief   红外光 FIR 滤波
 * @param   input  ：指向输入信号的数据数组
 * @param   output ：指向输出滤波结果的数据数组。
 * @retval  无
 */
void ir_max30102_fir(float *input, float *output)
{
    arm_fir_f32(&S_ir, input,output, blockSize);
}

/**
 * @brief   红光 FIR 滤波
 * @param   input  ：指向输入信号的数据数组
 * @param   output ：指向输出滤波结果的数据数组。
 * @retval  无
 */
void red_max30102_fir(float *input, float *output)
{
```

```
    arm_fir_f32(&S_red,input, output, blockSize);
}
```

这段代码利用 ARM_MATH 库实现了 FIR 滤波器的结构和基本操作。通过对红外光和红光信号应用低通 FIR 滤波器，可以有效去除高频噪声，从而提高信号质量。在初始化过程中，配置了滤波器的系数和状态，具体的信号处理则通过 ir_max30102_fir 和 red_max30102_fir 函数完成。

2.1.2.2 IIC 驱动

在图 2.1.2.2 中，myiic.c 和 myiic.h 是开发板与心率血氧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：

```
#define CACHE_NUMS          150         /* 缓存数&采集次数 */
#define PPG_DATA_THRESHOLD  100000      /* 检测阈值 */

float g_red_data[1];                    /* 红光数据缓冲区，用于计算 spo2(血氧含量) */
float g_ir_data[1];                     /* 红外光数据缓冲区，用于计算 heart(心率) */

float fir_output[2];                    /* 经过 FIR 滤波后的数据缓冲区 */
float ppg_data_cache_red[CACHE_NUMS] = {0};     /* 缓存区 */
float ppg_data_cache_ir[CACHE_NUMS] = {0};      /* 缓存区 */

/**
 * @brief       例程演示入口函数
 * @param       无
 * @retval      无
 */
void demo_run(void)
{
    uint16_t t = 0;
    uint16_t cnt = 0;               /* 缓存计数器 */
    uint16_t heart = 0;             /* 心率缓存区 */
    float spo2 = 0;                 /* 血氧缓存区 */

    atk_max30102_init();            /* 初始化 max30102 */
    max30102_fir_init();            /* FIR 滤波初始化 */
    printf("温度: %.2f℃\r\n", atk_max30102_get_temp());
    while(1)
    {
        if(g_max30102_int_flag)
        {
            g_max30102_int_flag = 0;
            atk_max30102_fifo_read(&g_red_data[0], &g_ir_data[0]);
```

```
        /* 使用 DSP 库实现 FIR 滤波 */
        red_max30102_fir(&g_red_data[0], &fir_output[0]);
        ir_max30102_fir(&g_ir_data[0], &fir_output[1]);

        /* 检测是否超过阈值，超过代表有皮肤接触传感器 */
        if((g_red_data[0] > PPG_DATA_THRESHOLD) && (g_ir_data[0] >
          PPG_DATA_THRESHOLD))    //大于阈值，说明传感器有接触
        {
            ppg_data_cache_red[cnt] = fir_output[0];
            ppg_data_cache_ir[cnt] = fir_output[1];
            cnt++;
        }
        else                        /* 小于阈值 */
        {
            cnt = 0;
        }
        if(cnt >= CACHE_NUMS)     /* 收集满了数据 */
        {
            /* 获取心率值 */
            heart = atk_max30102_get_heart(ppg_data_cache_ir, CACHE_NUMS);
            spo2 = atk_max30102_get_spo2(ppg_data_cache_ir,
            ppg_data_cache_red, CACHE_NUMS);/* 获取血氧值 */
            if(heart == 0)
            {
                printf("心率测量失败，请将皮肤紧贴传感器，并保持静止\r\n");
            }
            else
            {
                printf("心率：%d 次/min\r\n", heart);
            }
            printf("血氧：%.2f%%\r\n", spo2);
            printf("\r\n");
            cnt = 0;
        }

        t++;
        if(t == 20)
        {
            t = 0;
            LED0_TOGGLE();
        }
    }
}
```

```
}
```

从上面的代码中可以看出，整个测试代码的逻辑还是比较简单的，首先通过 MAX30102 传感器采集红光和红外光数据，并通过 FIR 滤波进行处理，最终计算出心率和血氧值。程序在 while 循环中持续运行，监测传感器数据并实时输出结果。需注意的是手指或皮肤需紧贴 MAX30102 传感器保持静止，否则可能导致数据读取有误。

2.1.3 实验现象

将心率血氧模块按照第一节“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心率以及血氧浓度信息，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：

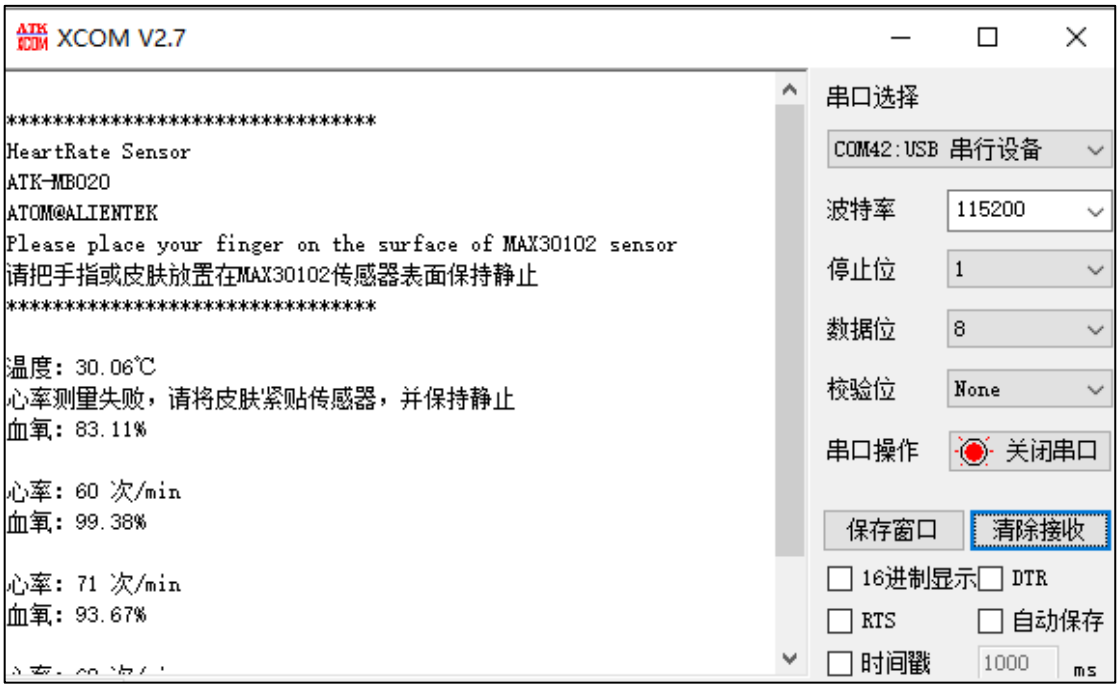

图 2.1.3.1 串口调试助手显示内容

首先一开始我们需要根据提示信息，将手指头或者皮肤紧贴心率血氧模块表面，并保持静止，之后串口就会打印相应的心率和血氧浓度值，如果未保持静止，则会提示心率测试失败等信息！大家可自行测试看看。

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

正点原子