一、前言

MAX30102是一款由Maxim Integrated推出的低功耗、高精度的心率和血氧饱和度检测传感器模块，适用于可穿戴设备如智能手环、智能手表等健康管理类电子产品。

该传感器主要特性如下：

（1）**光学测量**：MAX30102内置了两个LED光源（红光和红外光），以及一个光电检测器，通过光电容积脉搏波描记法（PPG）来实现心率和血氧饱和度的无创检测。

（2）**低功耗**：在典型的工作模式下，其功耗非常低，有助于延长电池供电设备的使用寿命。

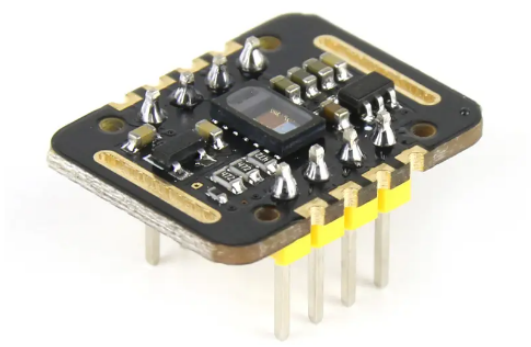
（3）**集成度高**：内部集成了AFE（模拟前端）、LED驱动器、环境光抑制功能以及I²C数字接口，方便与微控制器连接通信。

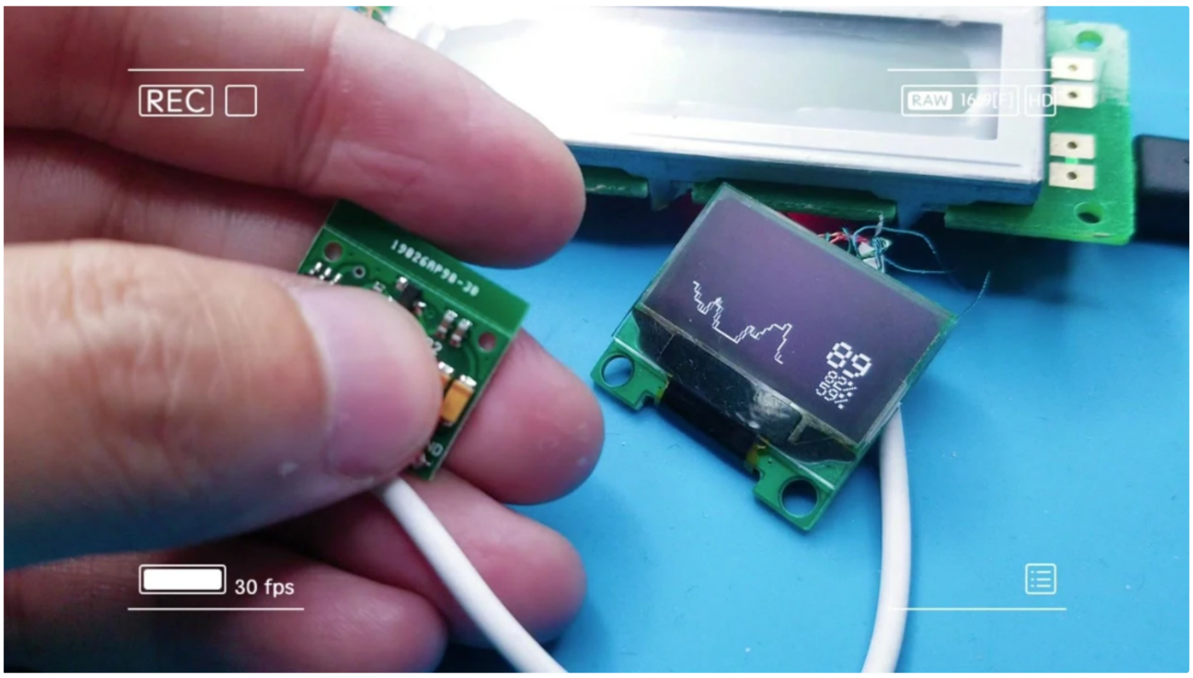
（4）**多档位配置**：支持多个LED电流输出级别和采样速率选择，可以根据实际应用需求进行灵活配置。

（5）**高精度**：通过先进的信号处理算法，可以有效降低噪声干扰，提高测量数据的准确性。

（6）**小尺寸封装**：采用紧凑型封装设计，便于在空间受限的产品中使用。

MAX30102是一款高性能的生物医学传感器，能够帮助开发者在各种便携式和穿戴式设备上实现对人体生理参数的有效监测。





二、IIC协议

MAX30102 是一款由 Maxim Integrated（现为 Analog Devices 公司的一部分）制造的生物识别传感器，它采用 I2C（Inter-Integrated Circuit）协议进行通信。I2C 协议是一种常见的串行接口标准，特别适用于在嵌入式系统中连接微控制器和其他低速周边设备，如传感器、EEPROM、RTC（实时时钟）等。

I2C 协议详解：

（1）**架构与线路：**

- **SDA（Serial Data Line）：** 串行数据线，用于传输数据。
- **SCL（Serial Clock Line）：** 串行时钟线，由主设备控制，决定数据传输速率和每个位的时间间隔。
- **多主从架构：** 支持一个主设备和多个从设备同时连接到总线上，主设备负责发起通信并控制数据传输方向。

（2）**信号特性：**

- **开始条件（Start Condition）：** 当 SDA 线在 SCL 高电平时由高电平变为低电平，表示一次传输的开始。
- **停止条件（Stop Condition）：** 反之，在 SCL 高电平时，SDA 线由低电平变为高电平，标志一次传输结束。
- **地址字节：** 每次通信开始时，主设备会通过发送包含7位从设备地址（加上一位读写位）的数据包来寻址目标从设备，例如 MAX30102。

（3）**数据传输：**

- **读/写操作：** 地址字节的最低位决定了接下来是读操作（R/W=1）还是写操作（R/W=0）。
- **应答（ACK/NACK）：** 每个被传送的数据字节后，接收方需拉低 SDA 行线以发出一个确认（ACK）信号。若不响应，则为主动非应答（NACK），可能用于指示传输结束或错误。
- **数据位传输：** 数据以高位先出（MSB-first）的方式逐位传输。

（4）**波特率：**

- I2C 协议允许不同的传输速率，称为标准模式（100kHz）、快速模式（400kHz）、快速模式+（1MHz）以及其他更高性能的模式。

对于MAX30102这样的传感器来说，通过I2C接口可以读取其内部寄存器数据，如配置寄存器、状态寄存器以及测量数据缓冲区等，从而实现对传感器的控制和数据采集。开发人员通常使用微控制器提供的硬件I2C模块或者软件模拟的I2C协议来与MAX30102进行通信。

模拟I2C协议通常涉及到对硬件时序的精确控制，以下是一个基于软件模拟的、简化版的C语言代码示例，用于演示基本原理。

```c
#include <stdio.h>
#include <unistd.h>

// 假设sda和scl是连接到GPIO的文件描述符
#define SDA 3
#define SCL 4

// 设置GPIO为输出模式
void gpio_setup_output(int pin) {
    // 这部分代码依赖于具体的GPIO库或系统调用，这里仅为示意
}

// 设置GPIO为输入模式并读取电平
int gpio_read_input(int pin) {
    // 这部分代码依赖于具体的GPIO库或系统调用，这里仅为示意
    return value; // 返回0或1
}

// 模拟SDA线上的数据传输
void sda_write(int data) {
    gpio_setup_output(SDA);
    if (data)
        // 将SDA置高
        ;
    else
        // 将SDA置低
        ;
}

// 模拟SCL线上的时钟脉冲
void scl_pulse(void) {
    gpio_setup_output(SCL);
    // 将SCL拉低
    usleep(1); // 延迟以模拟时钟周期的一部分
    // 将SCL拉高
    usleep(1); // 延迟以完成时钟周期
}

// 发送一个字节数据
void i2c_send_byte(unsigned char byte) {
    for (int i = 7; i >= 0; --i) {
        sda_write(byte & (1 << i));
        scl_pulse();
    }
    // 等待ACK
    gpio_setup_output(SDA);
    gpio_write(SDA, 1); // 主机释放SDA，从机应答
    scl_pulse();
    if (gpio_read_input(SDA)) {
        printf("No ACK received\n");
        // 处理无应答的情况...
    }
}

// 接收一个字节数据
unsigned char i2c_receive_byte(int ack) {
    unsigned char byte = 0;
    gpio_setup_input(SDA);
    for (int i = 7; i >= 0; --i) {
        byte <<= 1;
        scl_pulse();
        byte |= gpio_read_input(SDA);
    }
    gpio_setup_output(SDA);
    // 发送ACK/NAK
    sda_write(!ack);
    scl_pulse();
    return byte;
}
```

```
// I2C开始条件
void i2c_start_condition(void) {
    sda_write(1);
    scl_write(1);
    sda_write(0);
}

// I2C停止条件
void i2c_stop_condition(void) {
    sda_write(0);
    scl_write(1);
    sda_write(1);
}

// 向设备发送地址和数据
void i2c_send_address_and_data(unsigned char address, unsigned char data, int
is_write) {
    i2c_start_condition();
    i2c_send_byte((address << 1) | (is_write ? 0 : 1)); // 地址 + R/W位
    i2c_send_byte(data); // 数据
    i2c_stop_condition();
}

// 从设备接收数据
unsigned char i2c_receive_data(unsigned char address) {
    i2c_start_condition();
    i2c_send_byte((address << 1) | 1); // 地址 + R/W=1 (读操作)
    unsigned char data = i2c_receive_byte(0); // 接收数据并发送ACK
    i2c_stop_condition();
    return data;
}
```

三、项目代码

下面贴出了STM32工程里完整的max30102的代码，因为是才有寄存器编程。 所有兼容所有的工程，不管你是STM32标准库工程还是STM32HAL库工程，只要把下面的.c文件和.h文件加载到你的STM32工程里。将max30102接好线，按照头文件里说明调用mainx30102函数完成初始化就可以。

3.1 max30102.c

```
#include "max30102.h"
#include "delay.h"

/*
MAX30102心率传感器:
SCL<——>PB6
SDA<——>PB7
IM<——>PB9
*/
```

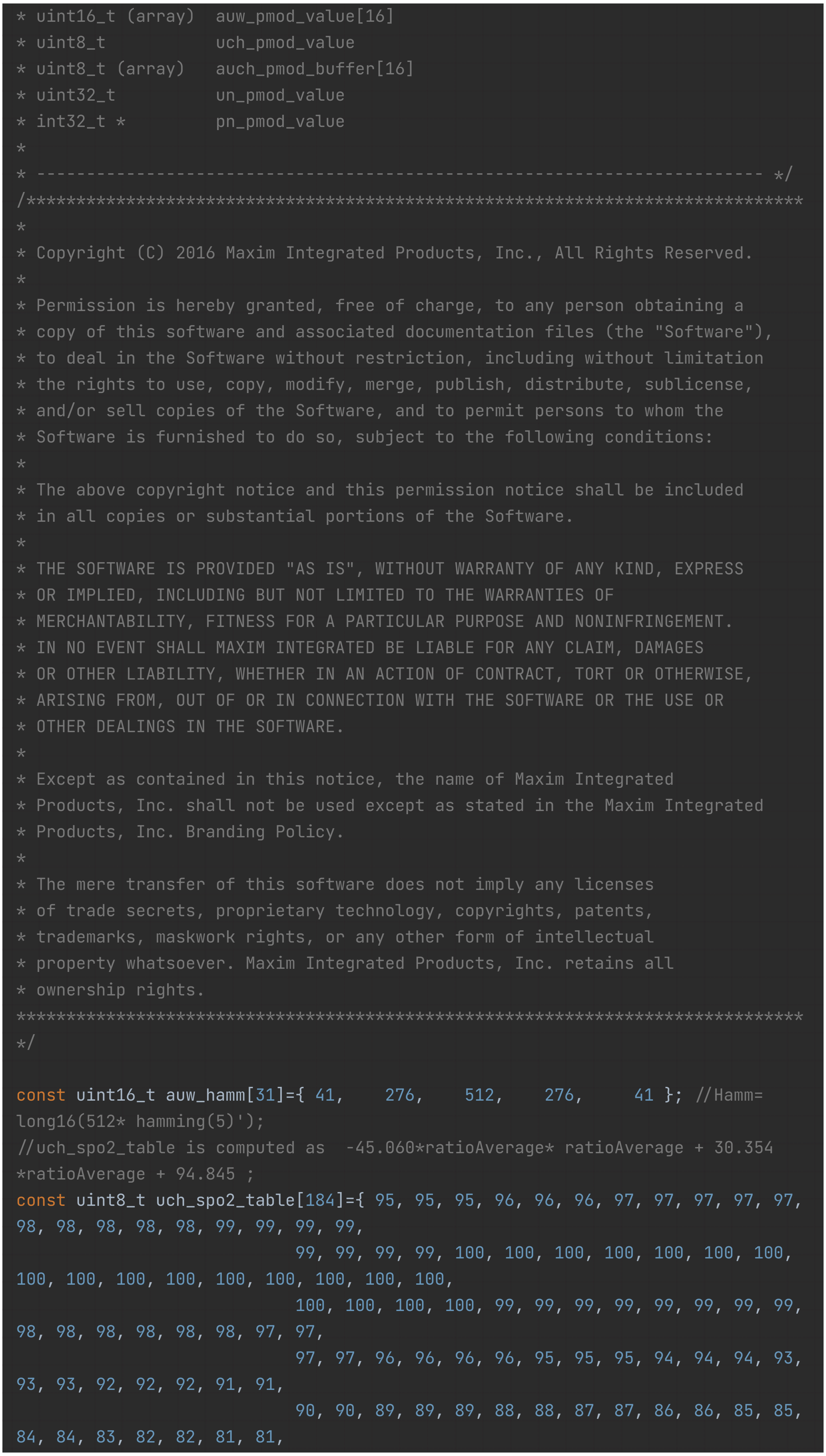
```c
//初始化IIC
void IIC_Init(void)
{
    RCC->APB2ENR|=1<<3;
    GPIOB->CRL&=0x00FFFFFF;
    GPIOB->CRL|=0x33000000;

    GPIOB->CRH&=0xFFFFFF0F;
    GPIOB->CRH|=0x00000080;

    IIC_SCL=1;
    IIC_SDA=1;

}


//产生IIC起始信号
void IIC_Start(void)
{
    SDA_OUT();     //sda线输出
    IIC_SDA=1;
    IIC_SCL=1;
    delay_us(4);
    IIC_SDA=0;//START:when CLK is high,DATA change form high to low
    delay_us(4);
    IIC_SCL=0;//钳住I2C总线，准备发送或接收数据
}
//产生IIC停止信号
void IIC_Stop(void)
{
    SDA_OUT();//sda线输出
    IIC_SCL=0;
    IIC_SDA=0;//STOP:when CLK is high DATA change form low to high
    delay_us(4);
    IIC_SCL=1;
    IIC_SDA=1;//发送I2C总线结束信号
    delay_us(4);
}
//等待应答信号到来
//返回值：1，接收应答失败
//        0，接收应答成功
u8 IIC_Wait_Ack(void)
{
    u8 ucErrTime=0;
    SDA_IN();      //SDA设置为输入
    IIC_SDA=1;delay_us(1);
    IIC_SCL=1;delay_us(1);
    while(READ_SDA)
    {
        ucErrTime++;
        if(ucErrTime>250)
        {
            IIC_Stop();
            return 1;
        }
    }
    IIC_SCL=0;//时钟输出0
    return 0;
}
//产生ACK应答
void IIC_Ack(void)
{
    IIC_SCL=0;
    SDA_OUT();
    IIC_SDA=0;
    delay_us(2);
    IIC_SCL=1;
    delay_us(2);
    IIC_SCL=0;
}
//不产生ACK应答
void IIC_NAck(void)
{
    IIC_SCL=0;
    SDA_OUT();
    IIC_SDA=1;
    delay_us(2);
    IIC_SCL=1;
    delay_us(2);
    IIC_SCL=0;
}
//IIC发送一个字节
//返回从机有无应答
//1，有应答
//0，无应答
void IIC_Send_Byte(u8 txd)
{
    u8 t;
    SDA_OUT();
    IIC_SCL=0;//拉低时钟开始数据传输
    for(t=0;t<8;t++)
    {
        IIC_SDA=(txd&0x80)>>7;
        txd<<=1;
        delay_us(2);    //对TEA5767这三个延时都是必须的
        IIC_SCL=1;
        delay_us(2);
        IIC_SCL=0;
        delay_us(2);
    }
}
//读1个字节，ack=1时，发送ACK，ack=0，发送nACK
u8 IIC_Read_Byte(unsigned char ack)
{
    unsigned char i,receive=0;
    SDA_IN();//SDA设置为输入
    for(i=0;i<8;i++ )
    {
        IIC_SCL=0;
        delay_us(2);
        IIC_SCL=1;
        receive<<=1;
        if(READ_SDA)receive++;
        delay_us(1);
    }
    if (!ack)
        IIC_NAck();//发送nACK
    else
        IIC_Ack(); //发送ACK
    return receive;
}


void IIC_WriteBytes(u8 WriteAddr,u8* data,u8 dataLength)
{
    u8 i;
    IIC_Start();

    IIC_Send_Byte(WriteAddr);       //发送写命令
    IIC_Wait_Ack();

    for(i=0;i<dataLength;i++)
    {
        IIC_Send_Byte(data[i]);
        IIC_Wait_Ack();
    }
    IIC_Stop();//产生一个停止条件
    delay_ms(10);
}

void IIC_ReadBytes(u8 deviceAddr, u8 writeAddr,u8* data,u8 dataLength)
{
    u8 i;
    IIC_Start();

    IIC_Send_Byte(deviceAddr);      //发送写命令
    IIC_Wait_Ack();
    IIC_Send_Byte(writeAddr);
    IIC_Wait_Ack();
    IIC_Send_Byte(deviceAddr|0X01);//进入接收模式
    IIC_Wait_Ack();

    for(i=0;i<dataLength-1;i++)
    {
        data[i] = IIC_Read_Byte(1);
    }
    data[dataLength-1] = IIC_Read_Byte(0);
    IIC_Stop();//产生一个停止条件
    delay_ms(10);
}

void IIC_Read_One_Byte(u8 daddr,u8 addr,u8* data)
{
    IIC_Start();
    IIC_Send_Byte(daddr);       //发送写命令
    IIC_Wait_Ack();
    IIC_Send_Byte(addr);//发送地址
    IIC_Wait_Ack();
    IIC_Start();
    IIC_Send_Byte(daddr|0X01);//进入接收模式
    IIC_Wait_Ack();
    *data = IIC_Read_Byte(0);
    IIC_Stop();//产生一个停止条件
}

void IIC_Write_One_Byte(u8 daddr,u8 addr,u8 data)
{
    IIC_Start();

    IIC_Send_Byte(daddr);        //发送写命令
    IIC_Wait_Ack();
    IIC_Send_Byte(addr);//发送地址
    IIC_Wait_Ack();
    IIC_Send_Byte(data);      //发送字节
    IIC_Wait_Ack();
    IIC_Stop();//产生一个停止条件
    delay_ms(10);
}










uint32_t aun_ir_buffer[500]; //IR LED sensor data
int32_t n_ir_buffer_length;    //data length
uint32_t aun_red_buffer[500];    //Red LED sensor data
int32_t n_sp02; //SPO2 value
int8_t ch_spo2_valid;   //indicator to show if the SP02 calculation is valid
int32_t n_heart_rate;   //heart rate value
int8_t  ch_hr_valid;    //indicator to show if the heart rate calculation is
valid
uint8_t uch_dummy;


//variables to calculate the on-board LED brightness that reflects the
heartbeats
uint32_t un_min, un_max, un_prev_data;
int i;
int32_t n_brightness;
float f_temp;
u8 temp_num=0;
u8 temp[6];
```

```c
u8 str[100];
u8 dis_hr=0,dis_spo2=0;

#define MAX_BRIGHTNESS 255



u8 max30102_Bus_Write(u8 Register_Address, u8 Word_Data)
{

    /* 采用串行EEPROM随即读取指令序列，连续读取若干字节 */

    /* 第1步：发起I2C总线启动信号 */
    IIC_Start();

    /* 第2步：发起控制字节，高7bit是地址，bit0是读写控制位，0表示写，1表示读 */
    IIC_Send_Byte(max30102_WR_address | I2C_WR);    /* 此处是写指令 */

    /* 第3步：发送ACK */
    if (IIC_Wait_Ack() != 0)
    {
        goto cmd_fail;  /* EEPROM器件无应答 */
    }

    /* 第4步：发送字节地址 */
    IIC_Send_Byte(Register_Address);
    if (IIC_Wait_Ack() != 0)
    {
        goto cmd_fail;  /* EEPROM器件无应答 */
    }

    /* 第5步：开始写入数据 */
    IIC_Send_Byte(Word_Data);

    /* 第6步：发送ACK */
    if (IIC_Wait_Ack() != 0)
    {
        goto cmd_fail;  /* EEPROM器件无应答 */
    }

    /* 发送I2C总线停止信号 */
    IIC_Stop();
    return 1;   /* 执行成功 */

cmd_fail: /* 命令执行失败后，切记发送停止信号，避免影响I2C总线上其他设备 */
    /* 发送I2C总线停止信号 */
    IIC_Stop();
    return 0;
}



u8 max30102_Bus_Read(u8 Register_Address)
{
    u8  data;
    /* 第1步：发起I2C总线启动信号 */
    IIC_Start();

    /* 第2步：发起控制字节，高7bit是地址，bit0是读写控制位，0表示写，1表示读 */
    IIC_Send_Byte(max30102_WR_address | I2C_WR);    /* 此处是写指令 */

    /* 第3步：发送ACK */
    if (IIC_Wait_Ack() != 0)
    {
        goto cmd_fail;  /* EEPROM器件无应答 */
    }

    /* 第4步：发送字节地址， */
    IIC_Send_Byte((uint8_t)Register_Address);
    if (IIC_Wait_Ack() != 0)
    {
        goto cmd_fail;  /* EEPROM器件无应答 */
    }


    /* 第6步：重新启动I2C总线。下面开始读取数据 */
    IIC_Start();

    /* 第7步：发起控制字节，高7bit是地址，bit0是读写控制位，0表示写，1表示读 */
    IIC_Send_Byte(max30102_WR_address | I2C_RD);    /* 此处是读指令 */

    /* 第8步：发送ACK */
    if (IIC_Wait_Ack() != 0)
    {
        goto cmd_fail;  /* EEPROM器件无应答 */
    }

    /* 第9步：读取数据 */
    {
        data = IIC_Read_Byte(0);    /* 读1个字节 */

        IIC_NAck(); /* 最后1个字节读完后，CPU产生NACK信号(驱动SDA = 1) */
    }
    /* 发送I2C总线停止信号 */
    IIC_Stop();
    return data;    /* 执行成功 返回data值 */

cmd_fail: /* 命令执行失败后，切记发送停止信号，避免影响I2C总线上其他设备 */
    /* 发送I2C总线停止信号 */
    IIC_Stop();
    return 0;
}


void max30102_FIFO_ReadWords(u8 Register_Address,u16 Word_Data[][2],u8 count)
{
    u8 i=0;
    u8 no = count;
    u8 data1, data2;
    /* 第1步：发起I2C总线启动信号 */
    IIC_Start();

    /* 第2步：发起控制字节，高7bit是地址，bit0是读写控制位，0表示写，1表示读 */
    IIC_Send_Byte(max30102_WR_address | I2C_WR);    /* 此处是写指令 */

    /* 第3步：发送ACK */
    if (IIC_Wait_Ack() != 0)
    {
        goto cmd_fail;  /* EEPROM器件无应答 */
    }

    /* 第4步：发送字节地址， */
    IIC_Send_Byte((uint8_t)Register_Address);
    if (IIC_Wait_Ack() != 0)
    {
        goto cmd_fail;  /* EEPROM器件无应答 */
    }


    /* 第6步：重新启动I2C总线。下面开始读取数据 */
    IIC_Start();

    /* 第7步：发起控制字节，高7bit是地址，bit0是读写控制位，0表示写，1表示读 */
    IIC_Send_Byte(max30102_WR_address | I2C_RD);    /* 此处是读指令 */

    /* 第8步：发送ACK */
    if (IIC_Wait_Ack() != 0)
    {
        goto cmd_fail;  /* EEPROM器件无应答 */
    }

    /* 第9步：读取数据 */
    while (no)
    {
        data1 = IIC_Read_Byte(0);
        IIC_Ack();
        data2 = IIC_Read_Byte(0);
        IIC_Ack();
        Word_Data[i][0] = (((u16)data1 << 8) | data2);  //


        data1 = IIC_Read_Byte(0);
        IIC_Ack();
        data2 = IIC_Read_Byte(0);
        if(1==no)
            IIC_NAck(); /* 最后1个字节读完后，CPU产生NACK信号(驱动SDA = 1) */
        else
            IIC_Ack();
        Word_Data[i][1] = (((u16)data1 << 8) | data2);

        no--;
        i++;
    }
    /* 发送I2C总线停止信号 */
    IIC_Stop();

cmd_fail: /* 命令执行失败后，切记发送停止信号，避免影响I2C总线上其他设备 */
    /* 发送I2C总线停止信号 */
    IIC_Stop();
}

void max30102_FIFO_ReadBytes(u8 Register_Address,u8* Data)
{
    max30102_Bus_Read(REG_INTR_STATUS_1);
    max30102_Bus_Read(REG_INTR_STATUS_2);

    /* 第1步：发起I2C总线启动信号 */
    IIC_Start();

    /* 第2步：发起控制字节，高7bit是地址，bit0是读写控制位，0表示写，1表示读 */
    IIC_Send_Byte(max30102_WR_address | I2C_WR);    /* 此处是写指令 */

    /* 第3步：发送ACK */
    if (IIC_Wait_Ack() != 0)
    {
        goto cmd_fail;  /* EEPROM器件无应答 */
    }

    /* 第4步：发送字节地址， */
    IIC_Send_Byte((uint8_t)Register_Address);
    if (IIC_Wait_Ack() != 0)
    {
        goto cmd_fail;  /* EEPROM器件无应答 */
    }


    /* 第6步：重新启动I2C总线。下面开始读取数据 */
    IIC_Start();

    /* 第7步：发起控制字节，高7bit是地址，bit0是读写控制位，0表示写，1表示读 */
    IIC_Send_Byte(max30102_WR_address | I2C_RD);    /* 此处是读指令 */

    /* 第8步：发送ACK */
    if (IIC_Wait_Ack() != 0)
    {
        goto cmd_fail;  /* EEPROM器件无应答 */
    }

    /* 第9步：读取数据 */
    Data[0] = IIC_Read_Byte(1);
    Data[1] = IIC_Read_Byte(1);
    Data[2] = IIC_Read_Byte(1);
    Data[3] = IIC_Read_Byte(1);
    Data[4] = IIC_Read_Byte(1);
    Data[5] = IIC_Read_Byte(0);
    /* 最后1个字节读完后，CPU产生NACK信号(驱动SDA = 1) */
    /* 发送I2C总线停止信号 */
    IIC_Stop();
```

```
cmd_fail: /* 命令执行失败后，切记发送停止信号，避免影响I2C总线上其他设备 */
    /* 发送I2C总线停止信号 */
    IIC_Stop();

//  u8 i;
//  u8 fifo_wr_ptr;
//  u8 firo_rd_ptr;
//  u8 number_tp_read;
//  //Get the FIFO_WR_PTR
//  fifo_wr_ptr = max30102_Bus_Read(REG_FIFO_WR_PTR);
//  //Get the FIFO_RD_PTR
//  firo_rd_ptr = max30102_Bus_Read(REG_FIFO_RD_PTR);
//
//  number_tp_read = fifo_wr_ptr - firo_rd_ptr;
//
//  //for(i=0;i<number_tp_read;i++){
//  if(number_tp_read>0){
//      IIC_ReadBytes(max30102_WR_address,REG_FIFO_DATA,Data,6);
//  }

    //max30102_Bus_Write(REG_FIFO_RD_PTR,fifo_wr_ptr);
}

void max30102_init(void)
{
    IIC_Init();

    max30102_reset();

//  max30102_Bus_Write(REG_MODE_CONFIG, 0x0b);  //mode configuration :
temp_en[3]      MODE[2:0]=010 HR only enabled    011 SP02 enabled
//  max30102_Bus_Write(REG_INTR_STATUS_2, 0xF0); //open all of interrupt
//  max30102_Bus_Write(REG_INTR_STATUS_1, 0x00); //all interrupt clear
//  max30102_Bus_Write(REG_INTR_ENABLE_2, 0x02); //DIE_TEMP_RDY_EN
//  max30102_Bus_Write(REG_TEMP_CONFIG, 0x01); //SET   TEMP_EN

//  max30102_Bus_Write(REG_SPO2_CONFIG, 0x47); //SPO2_SR[4:2]=001  100 per
second    LED_PW[1:0]=11  16BITS

//  max30102_Bus_Write(REG_LED1_PA, 0x47);
//  max30102_Bus_Write(REG_LED2_PA, 0x47);



    max30102_Bus_Write(REG_INTR_ENABLE_1,0xc0); // INTR setting
    max30102_Bus_Write(REG_INTR_ENABLE_2,0x00);
    max30102_Bus_Write(REG_FIFO_WR_PTR,0x00);   //FIFO_WR_PTR[4:0]
    max30102_Bus_Write(REG_OVF_COUNTER,0x00);   //OVF_COUNTER[4:0]
    max30102_Bus_Write(REG_FIFO_RD_PTR,0x00);   //FIFO_RD_PTR[4:0]
    max30102_Bus_Write(REG_FIFO_CONFIG,0x0f);   //sample avg = 1, fifo
rollover=false, fifo almost full = 17
    max30102_Bus_Write(REG_MODE_CONFIG,0x03);   //0x02 for Red only, 0x03 for
SpO2 mode 0x07 multimode LED
```

```
    max30102_Bus_Write(REG_SPO2_CONFIG,0x27);   // SPO2_ADC range = 4096nA,
SPO2 sample rate (100 Hz), LED pulseWidth (400uS)
    max30102_Bus_Write(REG_LED1_PA,0x24);       //Choose value for ~ 7mA for
LED1
    max30102_Bus_Write(REG_LED2_PA,0x24);       // Choose value for ~ 7mA for
LED2
    max30102_Bus_Write(REG_PILOT_PA,0x7f);      // Choose value for ~ 25mA for
Pilot LED



//  // Interrupt Enable 1 Register. Set PPG_RDY_EN (data available in FIFO)
//  max30102_Bus_Write(0x2, 1<<6);

//  // FIFO configuration register
//  // SMP_AVE: 16 samples averaged per FIFO sample
//  // FIFO_ROLLOVER_EN=1
//  //max30102_Bus_Write(0x8,  1<<4);
//  max30102_Bus_Write(0x8, (0<<5) | 1<<4);

//  // Mode Configuration Register
//  // SPO2 mode
//  max30102_Bus_Write(0x9, 3);

//  // SPO2 Configuration Register
//  max30102_Bus_Write(0xa,
//          (3<<5)  // SPO2_ADC_RGE 2 = full scale 8192 nA (LSB size 31.25pA);
3 = 16384nA
//          | (1<<2) // sample rate: 0 = 50sps; 1 = 100sps; 2 = 200sps
//          | (3<<0) // LED_PW 3 = 411µs, ADC resolution 18 bits
//  );

//  // LED1 (red) power (0 = 0mA; 255 = 50mA)
//  max30102_Bus_Write(0xc, 0xb0);

//  // LED (IR) power
//  max30102_Bus_Write(0xd, 0xa0);

}

void max30102_reset(void)
{
    max30102_Bus_Write(REG_MODE_CONFIG,0x40);
    max30102_Bus_Write(REG_MODE_CONFIG,0x40);
}





void maxim_max30102_write_reg(uint8_t uch_addr, uint8_t uch_data)
{
//  char ach_i2c_data[2];
//  ach_i2c_data[0]=uch_addr;
//  ach_i2c_data[1]=uch_data;
//
//  IIC_WriteBytes(I2C_WRITE_ADDR, ach_i2c_data, 2);
    IIC_Write_One_Byte(I2C_WRITE_ADDR,uch_addr,uch_data);
}

void maxim_max30102_read_reg(uint8_t uch_addr, uint8_t *puch_data)
{
//  char ch_i2c_data;
//  ch_i2c_data=uch_addr;
//  IIC_WriteBytes(I2C_WRITE_ADDR, &ch_i2c_data, 1);
//
//  i2c.read(I2C_READ_ADDR, &ch_i2c_data, 1);
//
//   *puch_data=(uint8_t) ch_i2c_data;
    IIC_Read_One_Byte(I2C_WRITE_ADDR,uch_addr,puch_data);
}

void maxim_max30102_read_fifo(uint32_t *pun_red_led, uint32_t *pun_ir_led)
{
    uint32_t un_temp;
    unsigned char uch_temp;
    char ach_i2c_data[6];
    *pun_red_led=0;
    *pun_ir_led=0;


  //read and clear status register
  maxim_max30102_read_reg(REG_INTR_STATUS_1, &uch_temp);
  maxim_max30102_read_reg(REG_INTR_STATUS_2, &uch_temp);

  IIC_ReadBytes(I2C_WRITE_ADDR,REG_FIFO_DATA,(u8 *)ach_i2c_data,6);

  un_temp=(unsigned char) ach_i2c_data[0];
  un_temp<<=16;
  *pun_red_led+=un_temp;
  un_temp=(unsigned char) ach_i2c_data[1];
  un_temp<<=8;
  *pun_red_led+=un_temp;
  un_temp=(unsigned char) ach_i2c_data[2];
  *pun_red_led+=un_temp;

  un_temp=(unsigned char) ach_i2c_data[3];
  un_temp<<=16;
  *pun_ir_led+=un_temp;
  un_temp=(unsigned char) ach_i2c_data[4];
  un_temp<<=8;
  *pun_ir_led+=un_temp;
  un_temp=(unsigned char) ach_i2c_data[5];
  *pun_ir_led+=un_temp;
  *pun_red_led&=0x03FFFF;  //Mask MSB [23:18]
  *pun_ir_led&=0x03FFFF;  //Mask MSB [23:18]
}
```

```c
void dis_DrawCurve(u32* data,u8 x)
{
    u16 i;
    u32 max=0,min=262144;
    u32 temp;
    u32 compress;

    for(i=0;i<128*2;i++)
    {
        if(data[i]>max)
        {
            max = data[i];
        }
        if(data[i]<min)
        {
            min = data[i];
        }
    }

    compress = (max-min)/20;

    for(i=0;i<128;i++)
    {
        temp = data[i*2] + data[i*2+1];
        temp/=2;
        temp -= min;
        temp/=compress;
        if(temp>20)temp=20;
    }
}

void MAX30102_data_set()
{
//  printf("\r\n MAX30102  init  \r\n");

    un_min=0x3FFFF;
    un_max=0;

    n_ir_buffer_length=500; //buffer length of 100 stores 5 seconds of samples
running at 100sps
     //read the first 500 samples, and determine the signal range
    for(i=0;i<n_ir_buffer_length;i++)
    {
        while(MAX30102_INT==1);   //wait until the interrupt pin asserts
//
        max30102_FIFO_ReadBytes(REG_FIFO_DATA,temp);
        aun_red_buffer[i] =  (long)((long)((long)temp[0]&0x03)<<16) |
(long)temp[1]<<8 | (long)temp[2];    // Combine values to get the actual number
        aun_ir_buffer[i] = (long)((long)((long)temp[3] & 0x03)<<16) |
(long)temp[4]<<8 | (long)temp[5];   // Combine values to get the actual number

        if(un_min>aun_red_buffer[i])
            un_min=aun_red_buffer[i];    //update signal min
        if(un_max<aun_red_buffer[i])
            un_max=aun_red_buffer[i];    //update signal max
    }
    un_prev_data=aun_red_buffer[i];
    //calculate heart rate and SpO2 after first 500 samples (first 5 seconds of
samples)
    maxim_heart_rate_and_oxygen_saturation(aun_ir_buffer, n_ir_buffer_length,
aun_red_buffer, &n_sp02, &ch_spo2_valid, &n_heart_rate, &ch_hr_valid);
}



void MAX30102_get(u8 *hr,u8 *spo2)
{
    i=0;
        un_min=0x3FFFF;
        un_max=0;

        //dumping the first 100 sets of samples in the memory and shift the
last 400 sets of samples to the top
        for(i=100;i<500;i++)
        {
            aun_red_buffer[i-100]=aun_red_buffer[i];
            aun_ir_buffer[i-100]=aun_ir_buffer[i];

            //update the signal min and max
            if(un_min>aun_red_buffer[i])
            un_min=aun_red_buffer[i];
            if(un_max<aun_red_buffer[i])
            un_max=aun_red_buffer[i];
        }
        //take 100 sets of samples before calculating the heart rate.
        for(i=400;i<500;i++)
        {
            un_prev_data=aun_red_buffer[i-1];
//            while(MAX30102_INT==1);
            max30102_FIFO_ReadBytes(REG_FIFO_DATA,temp);
            aun_red_buffer[i] =  (long)((long)((long)temp[0]&0x03)<<16) |
(long)temp[1]<<8 | (long)temp[2];    // Combine values to get the actual number
            aun_ir_buffer[i] = (long)((long)((long)temp[3] & 0x03)<<16) |
(long)temp[4]<<8 | (long)temp[5];   // Combine values to get the actual number

            if(aun_red_buffer[i]>un_prev_data)
            {
                f_temp=aun_red_buffer[i]-un_prev_data;
                f_temp/=(un_max-un_min);
                f_temp*=MAX_BRIGHTNESS;
                n_brightness-=(int)f_temp;
                if(n_brightness<0)
                    n_brightness=0;
            }
            else
            {
                f_temp=un_prev_data-aun_red_buffer[i];
                f_temp/=(un_max-un_min);
                f_temp*=MAX_BRIGHTNESS;
                n_brightness+=(int)f_temp;
                if(n_brightness>MAX_BRIGHTNESS)
                    n_brightness=MAX_BRIGHTNESS;
            }
            //send samples and calculation result to terminal program through
UART
            if(ch_hr_valid == 1 && n_heart_rate<120 && ch_spo2_valid == 1 &&
n_sp02<101)//**/ ch_hr_valid == 1 && ch_spo2_valid ==1 && n_heart_rate<120 &&
n_sp02<101
            {
                dis_hr = n_heart_rate;
                dis_spo2 = n_sp02;
            }
//          else
//          {
//              dis_hr = 0;
//              dis_spo2 = 0;
//          }
//              printf("HR=%i, ", dis_hr);
//              printf("HRvalid=%i, ", ch_hr_valid);
//              printf("Sp02=%i, ", dis_spo2);
//              printf("SP02Valid=%i\r\n", ch_spo2_valid);
            *hr = dis_hr;
            *spo2 = dis_spo2;
        }
        maxim_heart_rate_and_oxygen_saturation(aun_ir_buffer,
n_ir_buffer_length, aun_red_buffer, &n_sp02, &ch_spo2_valid, &n_heart_rate,
&ch_hr_valid);


        //红光在上，红外在下
        dis_DrawCurve(aun_red_buffer,20);
        dis_DrawCurve(aun_ir_buffer,0);

}


/** \file algorithm.c ******************************************************
*
* Project: MAXREFDES117#
* Filename: algorithm.cpp
* Description: This module calculates the heart rate/Sp02 level
*
*
* --------------------------------------------------------------------
*
* This code follows the following naming conventions:
*
* char              ch_pmod_value
* char (array)      s_pmod_s_string[16]
* float             f_pmod_value
* int32_t           n_pmod_value
* int32_t (array)   an_pmod_value[16]
* int16_t           w_pmod_value
* int16_t (array)   aw_pmod_value[16]
* uint16_t          uw_pmod_value
```



```
* uint16_t (array)  auw_pmod_value[16]
* uint8_t           uch_pmod_value
* uint8_t (array)   auch_pmod_buffer[16]
* uint32_t          un_pmod_value
* int32_t *         pn_pmod_value
*
* ------------------------------------------------------------------------ */
/*******************************************************************************
*
* Copyright (C) 2016 Maxim Integrated Products, Inc., All Rights Reserved.
*
* Permission is hereby granted, free of charge, to any person obtaining a
* copy of this software and associated documentation files (the "Software"),
* to deal in the Software without restriction, including without limitation
* the rights to use, copy, modify, merge, publish, distribute, sublicense,
* and/or sell copies of the Software, and to permit persons to whom the
* Software is furnished to do so, subject to the following conditions:
*
* The above copyright notice and this permission notice shall be included
* in all copies or substantial portions of the Software.
*
* THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS
* OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
* MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.
* IN NO EVENT SHALL MAXIM INTEGRATED BE LIABLE FOR ANY CLAIM, DAMAGES
* OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE,
* ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR
* OTHER DEALINGS IN THE SOFTWARE.
*
* Except as contained in this notice, the name of Maxim Integrated
* Products, Inc. shall not be used except as stated in the Maxim Integrated
* Products, Inc. Branding Policy.
*
* The mere transfer of this software does not imply any licenses
* of trade secrets, proprietary technology, copyrights, patents,
* trademarks, maskwork rights, or any other form of intellectual
* property whatsoever. Maxim Integrated Products, Inc. retains all
* ownership rights.
*******************************************************************************
*/

const uint16_t auw_hamm[31]={ 41,    276,    512,    276,     41 }; //Hamm=
long16(512* hamming(5)');
//uch_spo2_table is computed as  -45.060*ratioAverage* ratioAverage + 30.354
*ratioAverage + 94.845 ;
const uint8_t uch_spo2_table[184]={ 95, 95, 95, 96, 96, 96, 97, 97, 97, 97, 97,
98, 98, 98, 98, 98, 99, 99, 99, 99,
                            99, 99, 99, 99, 100, 100, 100, 100, 100, 100, 100,
100, 100, 100, 100, 100, 100, 100, 100, 100,
                            100, 100, 100, 100, 99, 99, 99, 99, 99, 99, 99, 99,
98, 98, 98, 98, 98, 98, 97, 97,
                            97, 97, 96, 96, 96, 96, 95, 95, 95, 94, 94, 94, 93,
93, 93, 92, 92, 92, 91, 91,
                            90, 90, 89, 89, 89, 88, 88, 87, 87, 86, 86, 85, 85,
84, 84, 83, 82, 82, 81, 81,
                            80, 80, 79, 78, 78, 77, 76, 76, 75, 74, 74, 73, 72,
72, 71, 70, 69, 69, 68, 67,
                            66, 66, 65, 64, 63, 62, 62, 61, 60, 59, 58, 57, 56,
56, 55, 54, 53, 52, 51, 50,
                            49, 48, 47, 46, 45, 44, 43, 42, 41, 40, 39, 38, 37,
36, 35, 34, 33, 31, 30, 29,
                            28, 27, 26, 25, 23, 22, 21, 20, 19, 17, 16, 15, 14,
12, 11, 10, 9, 7, 6, 5,
                            3, 2, 1 } ;
static  int32_t an_dx[ BUFFER_SIZE-MA4_SIZE]; // delta
static  int32_t an_x[ BUFFER_SIZE]; //ir
static  int32_t an_y[ BUFFER_SIZE]; //red

void maxim_heart_rate_and_oxygen_saturation(uint32_t *pun_ir_buffer,  int32_t
n_ir_buffer_length, uint32_t *pun_red_buffer, int32_t *pn_spo2, int8_t
*pch_spo2_valid,
                              int32_t *pn_heart_rate, int8_t  *pch_hr_valid)
/**
* \brief        Calculate the heart rate and SpO2 level
* \par          Details
*               By detecting  peaks of PPG cycle and corresponding AC/DC of
red/infra-red signal, the ratio for the SPO2 is computed.
*               Since this algorithm is aiming for Arm M0/M3. formaula for SPO2
did not achieve the accuracy due to register overflow.
*               Thus, accurate SPO2 is precalculated and save longo
uch_spo2_table[] per each ratio.
*
* \param[in]    *pun_ir_buffer           - IR sensor data buffer
* \param[in]    n_ir_buffer_length      - IR sensor data buffer length
* \param[in]    *pun_red_buffer          - Red sensor data buffer
* \param[out]    *pn_spo2                - Calculated SpO2 value
* \param[out]    *pch_spo2_valid         - 1 if the calculated SpO2 value is
valid
* \param[out]    *pn_heart_rate          - Calculated heart rate value
* \param[out]    *pch_hr_valid           - 1 if the calculated heart rate value
is valid
*
* \retval       None
*/
{
    uint32_t un_ir_mean ,un_only_once ;
    int32_t k ,n_i_ratio_count;
    int32_t i, s, m, n_exact_ir_valley_locs_count ,n_middle_idx;
    int32_t n_th1, n_npks,n_c_min;
    int32_t an_ir_valley_locs[15] ;
    int32_t an_exact_ir_valley_locs[15] ;
    int32_t an_dx_peak_locs[15] ;
    int32_t n_peak_interval_sum;

    int32_t n_y_ac, n_x_ac;
    int32_t n_spo2_calc;
    int32_t n_y_dc_max, n_x_dc_max;
    int32_t n_y_dc_max_idx, n_x_dc_max_idx;
    int32_t an_ratio[5],n_ratio_average;
    int32_t n_nume,  n_denom ;
    // remove DC of ir signal
    un_ir_mean =0;
    for (k=0 ; k<n_ir_buffer_length ; k++ ) un_ir_mean += pun_ir_buffer[k] ;
    un_ir_mean =un_ir_mean/n_ir_buffer_length ;
    for (k=0 ; k<n_ir_buffer_length ; k++ )  an_x[k] =  pun_ir_buffer[k] -
un_ir_mean ;

    // 4 pt Moving Average
    for(k=0; k< BUFFER_SIZE-MA4_SIZE; k++){
        n_denom= ( an_x[k]+an_x[k+1]+ an_x[k+2]+ an_x[k+3]);
        an_x[k]=  n_denom/(int32_t)4;
    }

    // get difference of smoothed IR signal

    for( k=0; k<BUFFER_SIZE-MA4_SIZE-1;  k++)
        an_dx[k]= (an_x[k+1]- an_x[k]);

    // 2-pt Moving Average to an_dx
    for(k=0; k< BUFFER_SIZE-MA4_SIZE-2; k++){
        an_dx[k] =  ( an_dx[k]+an_dx[k+1])/2 ;
    }

    // hamming window
    // flip wave form so that we can detect valley with peak detector
    for ( i=0 ; i<BUFFER_SIZE-HAMMING_SIZE-MA4_SIZE-2 ;i++){
        s= 0;
        for( k=i; k<i+ HAMMING_SIZE ;k++){
            s -= an_dx[k] *auw_hamm[k-i] ;
                     }
        an_dx[i]= s/ (int32_t)1146; // divide by sum of auw_hamm
    }


    n_th1=0; // threshold calculation
    for ( k=0 ; k<BUFFER_SIZE-HAMMING_SIZE ;k++){
        n_th1 += ((an_dx[k]>0)? an_dx[k] : ((int32_t)0-an_dx[k])) ;
    }
    n_th1= n_th1/ ( BUFFER_SIZE-HAMMING_SIZE);
    // peak location is acutally index for sharpest location of raw signal
since we flipped the signal
    maxim_find_peaks( an_dx_peak_locs, &n_npks, an_dx, BUFFER_SIZE-
HAMMING_SIZE, n_th1, 8, 5 );//peak_height, peak_distance, max_num_peaks

    n_peak_interval_sum =0;
    if (n_npks>=2){
        for (k=1; k<n_npks; k++)
            n_peak_interval_sum += (an_dx_peak_locs[k]-an_dx_peak_locs[k -1]);
        n_peak_interval_sum=n_peak_interval_sum/(n_npks-1);
        *pn_heart_rate=(int32_t)(6000/n_peak_interval_sum);// beats per minutes
        *pch_hr_valid  = 1;
    }
    else  {
        *pn_heart_rate = -999;
        *pch_hr_valid  = 0;
    }

    for ( k=0 ; k<n_npks ;k++)
        an_ir_valley_locs[k]=an_dx_peak_locs[k]+HAMMING_SIZE/2;


    // raw value : RED(=y) and IR(=X)
    // we need to assess DC and AC value of ir and red PPG.
    for (k=0 ; k<n_ir_buffer_length ; k++ )  {
        an_x[k] =  pun_ir_buffer[k] ;
        an_y[k] =  pun_red_buffer[k] ;
    }

    // find precise min near an_ir_valley_locs
    n_exact_ir_valley_locs_count =0;
    for(k=0 ; k<n_npks ;k++){
        un_only_once =1;
        m=an_ir_valley_locs[k];
        n_c_min= 16777216;//2^24;
        if (m+5 <  BUFFER_SIZE-HAMMING_SIZE  && m-5 >0){
            for(i= m-5;i<m+5; i++)
                if (an_x[i]<n_c_min){
                    if (un_only_once >0){
                       un_only_once =0;
                   }
                   n_c_min= an_x[i] ;
                   an_exact_ir_valley_locs[k]=i;
                }
            if (un_only_once ==0)
                n_exact_ir_valley_locs_count ++ ;
        }
    }
    if (n_exact_ir_valley_locs_count <2 ){
       *pn_spo2 =  -999 ; // do not use SPO2 since signal ratio is out of range
       *pch_spo2_valid  = 0;
       return;
    }
    // 4 pt MA
    for(k=0; k< BUFFER_SIZE-MA4_SIZE; k++){
        an_x[k]=( an_x[k]+an_x[k+1]+ an_x[k+2]+ an_x[k+3])/(int32_t)4;
        an_y[k]=( an_y[k]+an_y[k+1]+ an_y[k+2]+ an_y[k+3])/(int32_t)4;
    }

    //using an_exact_ir_valley_locs , find ir-red DC andir-red AC for SPO2
calibration ratio
    //finding AC/DC maximum of raw ir * red between two valley locations
    n_ratio_average =0;
    n_i_ratio_count =0;

    for(k=0; k< 5; k++) an_ratio[k]=0;
    for (k=0; k< n_exact_ir_valley_locs_count; k++){
        if (an_exact_ir_valley_locs[k] > BUFFER_SIZE ){
            *pn_spo2 =  -999 ; // do not use SPO2 since valley loc is out of
range
            *pch_spo2_valid  = 0;
            return;
        }
    }
    // find max between two valley locations
    // and use ratio betwen AC compoent of Ir & Red and DC compoent of Ir & Red
for SPO2

    for (k=0; k< n_exact_ir_valley_locs_count-1; k++){
        n_y_dc_max= -16777216 ;
        n_x_dc_max= - 16777216;
        if (an_exact_ir_valley_locs[k+1]-an_exact_ir_valley_locs[k] >10){
            for (i=an_exact_ir_valley_locs[k]; i< an_exact_ir_valley_locs[k+1];
i++){
                if (an_x[i]> n_x_dc_max) {n_x_dc_max =an_x[i];n_x_dc_max_idx
=i; }
                if (an_y[i]> n_y_dc_max) {n_y_dc_max
=an_y[i];n_y_dc_max_idx=i;}
            }
            n_y_ac= (an_y[an_exact_ir_valley_locs[k+1]] -
an_y[an_exact_ir_valley_locs[k] ] )*(n_y_dc_max_idx -
an_exact_ir_valley_locs[k]); //red
            n_y_ac=  an_y[an_exact_ir_valley_locs[k]] + n_y_ac/
(an_exact_ir_valley_locs[k+1] - an_exact_ir_valley_locs[k])  ;


            n_y_ac=  an_y[n_y_dc_max_idx] - n_y_ac;    // subracting linear DC
compoenents from raw
            n_x_ac= (an_x[an_exact_ir_valley_locs[k+1]] -
an_x[an_exact_ir_valley_locs[k] ] )*(n_x_dc_max_idx -
an_exact_ir_valley_locs[k]); // ir
            n_x_ac=  an_x[an_exact_ir_valley_locs[k]] + n_x_ac/
(an_exact_ir_valley_locs[k+1] - an_exact_ir_valley_locs[k]);
            n_x_ac=  an_x[n_y_dc_max_idx] - n_x_ac;      // subracting linear
DC compoenents from raw
            n_nume=( n_y_ac *n_x_dc_max)>>7 ; //prepare X100 to preserve
floating value
            n_denom= ( n_x_ac *n_y_dc_max)>>7;
            if (n_denom>0  && n_i_ratio_count <5 &&  n_nume != 0)
            {
                an_ratio[n_i_ratio_count]= (n_nume*20)/n_denom ; //formular is
( n_y_ac *n_x_dc_max) / ( n_x_ac *n_y_dc_max) ;
///************************n_nume原来是*100************************//
                n_i_ratio_count++;
            }
        }
    }

    maxim_sort_ascend(an_ratio, n_i_ratio_count);
    n_middle_idx= n_i_ratio_count/2;

    if (n_middle_idx >1)
        n_ratio_average =( an_ratio[n_middle_idx-1] +an_ratio[n_middle_idx])/2;
// use median
    else
        n_ratio_average = an_ratio[n_middle_idx ];
    if( n_ratio_average>2 && n_ratio_average <184){
        n_spo2_calc= uch_spo2_table[n_ratio_average] ;
        *pn_spo2 = n_spo2_calc ;
        *pch_spo2_valid  = 1;//  float_SPO2 =  -45.060*n_ratio_average*
n_ratio_average/10000 + 30.354 *n_ratio_average/100 + 94.845 ;  // for
comparison with table
    }
    else{
        *pn_spo2 =  -999 ; // do not use SPO2 since signal ratio is out of
range
        *pch_spo2_valid  = 0;
    }
}


void maxim_find_peaks(int32_t *pn_locs, int32_t *pn_npks, int32_t *pn_x,
int32_t n_size, int32_t n_min_height, int32_t n_min_distance, int32_t
n_max_num)
/**
* \brief        Find peaks
* \par          Details
*               Find at most MAX_NUM peaks above MIN_HEIGHT separated by at
least MIN_DISTANCE
*
* \retval       None
*/
{
    maxim_peaks_above_min_height( pn_locs, pn_npks, pn_x, n_size, n_min_height
);
    maxim_remove_close_peaks( pn_locs, pn_npks, pn_x, n_min_distance );
    *pn_npks = min( *pn_npks, n_max_num );
}

void maxim_peaks_above_min_height(int32_t *pn_locs, int32_t *pn_npks, int32_t
 *pn_x, int32_t n_size, int32_t n_min_height)
/**
* \brief        Find peaks above n_min_height
* \par          Details
*               Find all peaks above MIN_HEIGHT
*
* \retval       None
*/
{
    int32_t i = 1, n_width;
    *pn_npks = 0;

    while (i < n_size-1){
        if (pn_x[i] > n_min_height && pn_x[i] > pn_x[i-1]){            // find
left edge of potential peaks
            n_width = 1;
            while (i+n_width < n_size && pn_x[i] == pn_x[i+n_width])    // find
flat peaks
                n_width++;
            if (pn_x[i] > pn_x[i+n_width] && (*pn_npks) < 15 ){
           // find right edge of peaks
                pn_locs[(*pn_npks)++] = i;
                // for flat peaks, peak location is left edge
                i += n_width+1;
            }
            else
                i += n_width;
        }
        else
            i++;
    }
}


void maxim_remove_close_peaks(int32_t *pn_locs, int32_t *pn_npks, int32_t
*pn_x, int32_t n_min_distance)
/**
* \brief        Remove peaks
* \par          Details
*               Remove peaks separated by less than MIN_DISTANCE
*
* \retval       None
*/
{

    int32_t i, j, n_old_npks, n_dist;

    /* Order peaks from large to small */
    maxim_sort_indices_descend( pn_x, pn_locs, *pn_npks );

    for ( i = -1; i < *pn_npks; i++ ){
        n_old_npks = *pn_npks;
        *pn_npks = i+1;
        for ( j = i+1; j < n_old_npks; j++ ){
            n_dist =  pn_locs[j] - ( i == -1 ? -1 : pn_locs[i] ); // lag-zero
peak of autocorr is at index -1
            if ( n_dist > n_min_distance || n_dist < -n_min_distance )
                pn_locs[(*pn_npks)++] = pn_locs[j];
        }
    }

    // Resort indices longo ascending order
    maxim_sort_ascend( pn_locs, *pn_npks );
}

void maxim_sort_ascend(int32_t *pn_x,int32_t n_size)
/**
* \brief        Sort array
* \par          Details
*               Sort array in ascending order (insertion sort algorithm)
*
* \retval       None
*/
{
```

```c
    int32_t i, j, n_temp;
    for (i = 1; i < n_size; i++) {
        n_temp = pn_x[i];
        for (j = i; j > 0 && n_temp < pn_x[j-1]; j--)
            pn_x[j] = pn_x[j-1];
        pn_x[j] = n_temp;
    }
}

void maxim_sort_indices_descend(int32_t *pn_x, int32_t *pn_indx, int32_t
n_size)
/**
* \brief        Sort indices
* \par          Details
*               Sort indices according to descending order (insertion sort
algorithm)
*
* \retval       None
*/
{
    int32_t i, j, n_temp;
    for (i = 1; i < n_size; i++) {
        n_temp = pn_indx[i];
        for (j = i; j > 0 && pn_x[n_temp] > pn_x[pn_indx[j-1]]; j--)
            pn_indx[j] = pn_indx[j-1];
        pn_indx[j] = n_temp;
    }
}
```

3.2 max30102.h

```c
#ifndef __MYIIC_H
#define __MYIIC_H
#include "sys.h"
#include "usart.h"
#include <stdio.h>

/*
MAX30102心率传感器:
SCL<-->PB6
SDA<-->PB7
IM<-->PB9
*/

#define MAX30102_INT PBin(9)

//IO方向设置
#define SDA_IN()  {GPIOB->CRL&=0x0FFFFFFF;GPIOB->CRL|=0x40000000;}
#define SDA_OUT() {GPIOB->CRL&=0x0FFFFFFF;GPIOB->CRL|=0x70000000;}
```

```c
//IO操作函数
#define IIC_SCL    PBout(6) //SCL
#define IIC_SDA    PBout(7) //SDA
#define READ_SDA   PBin(7)  //输入SDA

//IIC所有操作函数
void IIC_Init(void);                //初始化IIC的IO口
void IIC_Start(void);               //发送IIC开始信号
void IIC_Stop(void);                //发送IIC停止信号
void IIC_Send_Byte(u8 txd);         //IIC发送一个字节
u8 IIC_Read_Byte(unsigned char ack);//IIC读取一个字节
u8 IIC_Wait_Ack(void);              //IIC等待ACK信号
void IIC_Ack(void);                 //IIC发送ACK信号
void IIC_NAck(void);                //IIC不发送ACK信号

void IIC_Write_One_Byte(u8 daddr,u8 addr,u8 data);
void IIC_Read_One_Byte(u8 daddr,u8 addr,u8* data);

void IIC_WriteBytes(u8 WriteAddr,u8* data,u8 dataLength);
void IIC_ReadBytes(u8 deviceAddr, u8 writeAddr,u8* data,u8 dataLength);



#define I2C_WR  0       /* 写控制bit */
#define I2C_RD  1       /* 读控制bit */

#define max30102_WR_address 0xAE

#define I2C_WRITE_ADDR 0xAE
#define I2C_READ_ADDR 0xAF

//register addresses
#define REG_INTR_STATUS_1 0x00
#define REG_INTR_STATUS_2 0x01
#define REG_INTR_ENABLE_1 0x02
#define REG_INTR_ENABLE_2 0x03
#define REG_FIFO_WR_PTR 0x04
#define REG_OVF_COUNTER 0x05
#define REG_FIFO_RD_PTR 0x06
#define REG_FIFO_DATA 0x07
#define REG_FIFO_CONFIG 0x08
#define REG_MODE_CONFIG 0x09
#define REG_SPO2_CONFIG 0x0A
#define REG_LED1_PA 0x0C
#define REG_LED2_PA 0x0D
#define REG_PILOT_PA 0x10
#define REG_MULTI_LED_CTRL1 0x11
#define REG_MULTI_LED_CTRL2 0x12
#define REG_TEMP_INTR 0x1F
#define REG_TEMP_FRAC 0x20
#define REG_TEMP_CONFIG 0x21
#define REG_PROX_INT_THRESH 0x30
#define REG_REV_ID 0xFE
#define REG_PART_ID 0xFF
```

```c
void max30102_init(void);
void max30102_reset(void);
u8 max30102_Bus_Write(u8 Register_Address, u8 Word_Data);
u8 max30102_Bus_Read(u8 Register_Address);
void max30102_FIFO_ReadWords(u8 Register_Address,u16  Word_Data[][2],u8 count);
void max30102_FIFO_ReadBytes(u8 Register_Address,u8* Data);

void maxim_max30102_write_reg(uint8_t uch_addr, uint8_t uch_data);
void maxim_max30102_read_reg(uint8_t uch_addr, uint8_t *puch_data);
void maxim_max30102_read_fifo(uint32_t *pun_red_led, uint32_t *pun_ir_led);

void dis_DrawCurve(u32* data,u8 x);
void MAX30102_get(u8 *hr,u8 *spo2);
void MAX30102_data_set(void);





#define true 1
#define false 0
#define FS 100
#define BUFFER_SIZE  (FS* 5)
#define HR_FIFO_SIZE 7
#define MA4_SIZE  4 // DO NOT CHANGE
#define HAMMING_SIZE  5// DO NOT CHANGE
#define min(x,y) ((x) < (y) ? (x) : (y))

//const uint16_t auw_hamm[31]={ 41,    276,    512,    276,     41 }; //Hamm=
long16(512* hamming(5)');
////uch_spo2_table is computed as  -45.060*ratioAverage* ratioAverage + 30.354
*ratioAverage + 94.845 ;
//const uint8_t uch_spo2_table[184]={ 95, 95, 95, 96, 96, 96, 97, 97, 97, 97,
97, 98, 98, 98, 98, 98, 99, 99, 99, 99,
//                              99, 99, 99, 99, 100, 100, 100, 100, 100, 100,
100, 100, 100, 100, 100, 100, 100, 100, 100, 100,
//                              100, 100, 100, 100, 99, 99, 99, 99, 99, 99, 99,
99, 98, 98, 98, 98, 98, 98, 97, 97,
//                              97, 97, 96, 96, 96, 96, 95, 95, 95, 94, 94, 94,
93, 93, 93, 92, 92, 92, 91, 91,
//                              90, 90, 89, 89, 89, 88, 88, 87, 87, 86, 86, 85,
85, 84, 84, 83, 82, 82, 81, 81,
//                              80, 80, 79, 78, 78, 77, 76, 76, 75, 74, 74, 73,
72, 72, 71, 70, 69, 69, 68, 67,
//                              66, 66, 65, 64, 63, 62, 62, 61, 60, 59, 58, 57,
56, 56, 55, 54, 53, 52, 51, 50,
//                              49, 48, 47, 46, 45, 44, 43, 42, 41, 40, 39, 38,
37, 36, 35, 34, 33, 31, 30, 29,
//                              28, 27, 26, 25, 23, 22, 21, 20, 19, 17, 16, 15,
14, 12, 11, 10, 9, 7, 6, 5,
//                              3, 2, 1 } ;
//static  int32_t an_dx[ BUFFER_SIZE-MA4_SIZE]; // delta
//static  int32_t an_x[ BUFFER_SIZE]; //ir
//static  int32_t an_y[ BUFFER_SIZE]; //red
```

```
void maxim_heart_rate_and_oxygen_saturation(uint32_t *pun_ir_buffer ,  int32_t
n_ir_buffer_length, uint32_t *pun_red_buffer ,   int32_t *pn_spo2, int8_t
*pch_spo2_valid ,  int32_t *pn_heart_rate , int8_t  *pch_hr_valid);
void maxim_find_peaks( int32_t *pn_locs, int32_t *pn_npks,  int32_t *pn_x,
int32_t n_size, int32_t n_min_height, int32_t n_min_distance, int32_t n_max_num
);
void maxim_peaks_above_min_height( int32_t *pn_locs, int32_t *pn_npks,  int32_t
*pn_x, int32_t n_size, int32_t n_min_height );
void maxim_remove_close_peaks( int32_t *pn_locs, int32_t *pn_npks,   int32_t
 *pn_x, int32_t n_min_distance );
void maxim_sort_ascend( int32_t *pn_x, int32_t n_size );
void maxim_sort_indices_descend(  int32_t  *pn_x, int32_t *pn_indx, int32_t
n_size);




#endif
```