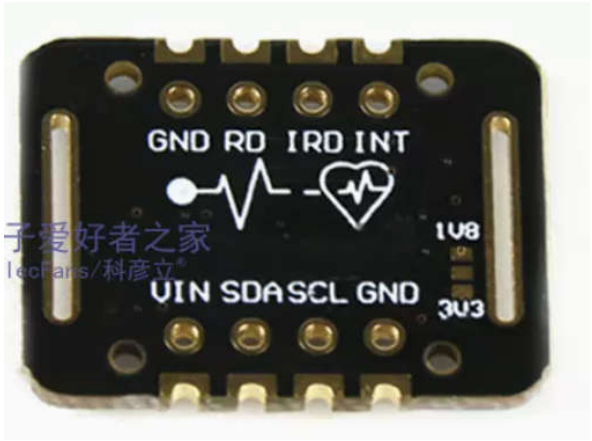
智能系统与控制

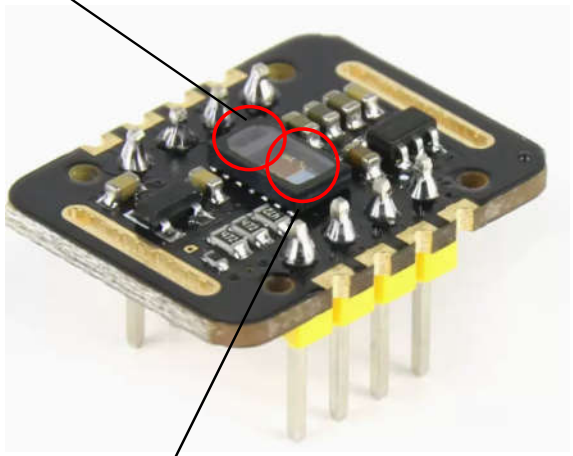
树莓派：心率血氧模块
max30102



于泓
鲁东大学
信息与电气工程学院
2021.11.8

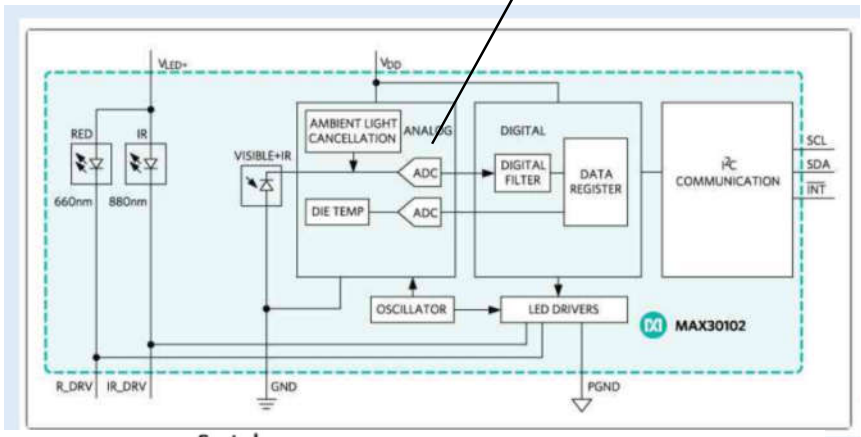
Max30102基本工作原理

LED，IR灯

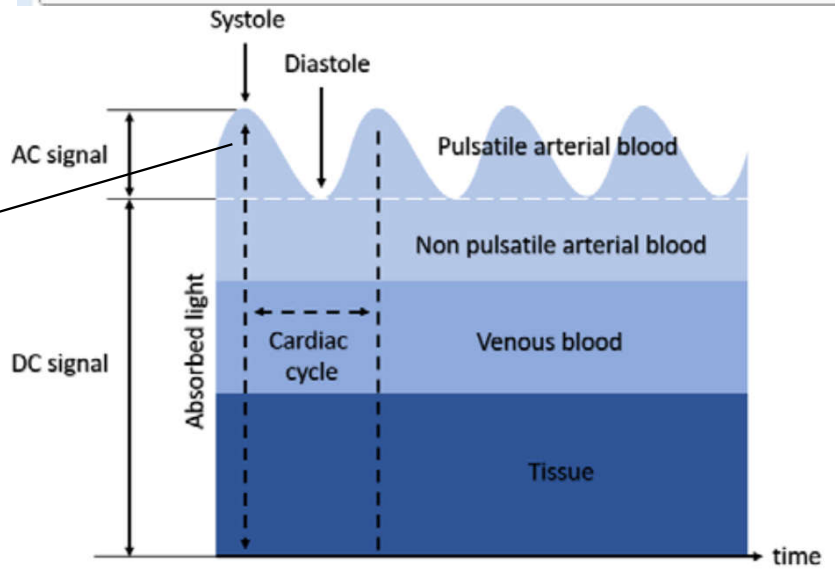


反射接受部分

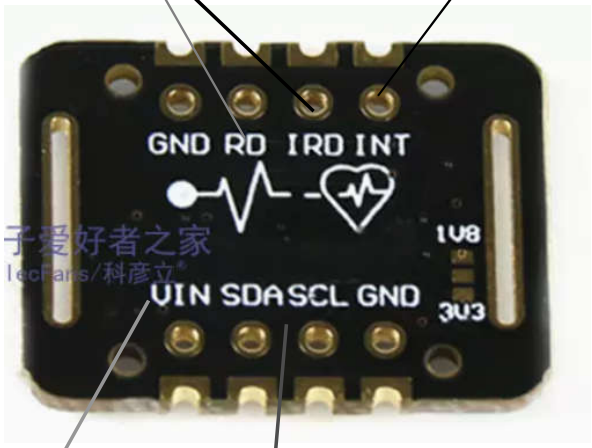
接收反射部分，并进行AD采样



接收信号
反应血液流动情况



接口及基本寄存器



与树莓派的连接

VIN: 接3.3V
GND: 接地

SDA，SCL 接树莓派的SDA与SCL

连接完成后，调用
sudo i2cdetect -y 1 检测是否连接成功

```
pi@raspberrypi:~/EXP-Raspberry/EXP_MAX30102 $ sudo i2cdetect -y 1
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- 57 -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
```

内部寄存器:

功能设置

Mode Configuration (0x09)

REGISTER	B7	B6	B5	B4	B3	B2	B1	B0	REG ADDR	POR STATE	R/W
Mode Configuration	SHDN	RESET				MODE[3:0]			0x09	0x00	R/W

复位

工作方式

关闭

MODE[2:0]	MODE	ACTIVE LED CHANNELS
000	Do not use	
001	Do not use	
010	Heart Rate mode	Red only
011	SpO2 mode	Red and IR
100–110	Do not use	
111	Multi-LED mode	Green, Red, and/or IR

通常设置为 SpO2 （血氧）模式
(可以同时测量心率和血氧饱和度)

测量量程

采样率

功率（AD采样精度）

SpO_2 Configuration (0x0A)

REGISTER	B7	B6	B5	B4	B3	B2	B1	B0	REG ADDR	POR STATE	R/W
SpO_2 Configuration		SPO2_ADC_RGE<1:0>		SPO2_SR[2:0]			LED_PW[2:0]		0x0A	0x00	R/W

SPO2_ADC_RGE[1:0]	LSB SIZE (pA)	FULL SCALE (nA)
00	7.81	2048
01	15.63	4096
02	31.25	8192
03	62.5	16384

SPO2_SR[2:0]	SAMPLES PER SECOND
000	50
001	100
010	200
011	400
100	800
101	1000
110	1600
111	3200

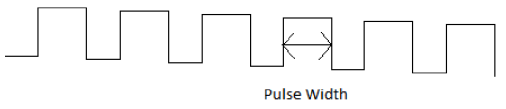
LED_PW[1:0]	PULSE WIDTH (µs)	ADC RESOLUTION (bits)
00	69 (68.95)	15
01	118 (117.78)	16
10	215 (215.44)	17
11	411 (410.75)	18

SAMPLES PER SECOND	PULSE WIDTH (µs)			
	69	118	215	411
50	O	O	O	O
100	O	O	O	O
200	O	O	O	O
400	O	O	O	O
800	O	O	O	
1000	O	O		
1600	O			
3200				
Resolution (bits)	15	16	17	18

Figure 12. Allowed settings for SpO2 configuration.

功率 通过脉宽来调整

不同的功率对应不同的精度



小灯的驱动电流幅度控制

LED Pulse Amplitude (0x0C–0x10)

REGISTER	B7	B6	B5	B4	B3	B2	B1	B0	REG ADDR	POR STATE	R/W
LED Pulse Amplitude	LED1_PA[7:0]								0x0C	0x00	R/W
LED Pulse Amplitude	LED2_PA[7:0]								0x0D	0x00	R/W
LED Pulse Amplitude	LED3_PA[7:0]								0x0E	0x00	R/W
RESERVED									0x0F	0x00	R/W
Proximity Mode LED Pulse Amplitude	PILOT_PA[7:0]								0x10	0x00	R/W

24H

7FH

LEDx_PA [7:0], RED_PA[7:0], IR_PA[7:0], or G_PA[7:0]	TYPICAL LED CURRENT (mA)*
0x00h	0.0
0x01h	0.2
0x02h	0.4
...	...
0x0Fh	3.1
...	...
0x1Fh	6.4
...	...
0x3Fh	12.5
...	...
0x7Fh	25.4
...	...
0xFFh	50.0

中断状态 （只读）

Interrupt Status (0x00–0x01)

REGISTER	B7	B6	B5	B4	B3	B2	B1	B0	REG ADDR	POR STATE	R/W
Interrupt Status 1	A_FULL	PPG_RDY	ALC_OVF	PROX_ INT				PWR_ RDY	0x00	0X00	R
Interrupt Status 2							DIE_ TEMP_RDY		0x01	0x00	R

中断使能

Interrupt Enable (0x02-0x03)

REGISTER	B7	B6	B5	B4	B3	B2	B1	B0	REG ADDR	POR STATE	R/W
Interrupt Enable 1	A_ FULL_ EN	PPG_ RDY_EN	ALC_ OVF_EN	PROX_ INT_EN					0x02	0X00	R/W
Interrupt Enable 2							DIE_TEMP_ RDY_EN		0x03	0x00	R/W

FIFO已满　　新的数据准备好　　环境光异常

FIFO (0x04–0x07)

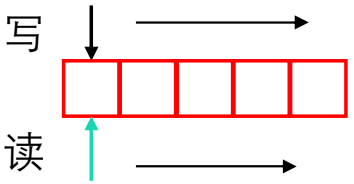
REGISTER	B7	B6	B5	B4	B3	B2	B1	B0	REG ADDR	POR STATE	R/W
FIFO Write Pointer				FIFO_WR_PTR[4:0]					0x04	0x00	R/W
Over Flow Counter				OVF_COUNTER[4:0]					0x05	0x00	R/W
FIFO Read Pointer				FIFO_RD_PTR[4:0]					0x06	0x00	R/W
FIFO Data Register	FIFO_DATA[7:0]								0x07	0x00	R/W

FIFO缓冲区，缓冲32个样本

写指针

FIFO已满
丢失的数据

读指针



每读/写**一个样本**地址+1

FIFO数据地址
每次读取1个样本
即6个字节

每个样本包括 red+IR 两个数据
每个数据3个字节
共6个字节

样本平均　缓存满了自动覆盖　缓存样本满多少触发中断

FIFO Configuration (0x08)

REGISTER	B7	B6	B5	B4	B3	B2	B1	B0	REG ADDR	POR STATE	R/W
FIFO Configuration	SMP_AVE[2:0]			FIFO_ROL LOVER_EN	FIFO_A_FULL[3:0]				0x08	0x00	R/W

SMP_AVE[2:0]	NO. OF SAMPLES AVERAGED PER FIFO SAMPLE
000	1 (no averaging)
001	2
010	4
011	8
100	16
101	32
110	32
111	32

FIFO_A_FULL[3:0]	ALMOST FULL INTERRUPT TRIGGER (NO. OF SAMPLES IN THE FIFO)
0x0h	0
0x1h	1
0x2h	2
0x3h	3
...	...
0xFh	15

在FIFO使用时，通常
不要直接对写指针和读指针进行写操作

可以通过
读/写指针之间的差距，获知
缓存区内样本的数目

基本驱动代码实现

```
from time import sleep
import smbus
# register addresses
REG_INTR_STATUS_1 = 0x00
REG_INTR_STATUS_2 = 0x01

REG_INTR_ENABLE_1 = 0x02
REG_INTR_ENABLE_2 = 0x03

REG_FIFO_WR_PTR = 0x04
REG_OVF_COUNTER = 0x05
REG_FIFO_RD_PTR = 0x06
REG_FIFO_DATA = 0x07
REG_FIFO_CONFIG = 0x08

REG_MODE_CONFIG = 0x09
REG_SPO2_CONFIG = 0x0A
REG_LED1_PA = 0x0C

REG_LED2_PA = 0x0D
REG_PILOT_PA = 0x10
REG_MULTI_LED_CTRL1 = 0x11
REG_MULTI_LED_CTRL2 = 0x12

REG_TEMP_INTR = 0x1F
REG_TEMP_FRAC = 0x20
REG_TEMP_CONFIG = 0x21
REG_PROX_INT_THRESH = 0x30
REG_REV_ID = 0xFE
REG_PART_ID = 0xFF
```

定义寄存器地址

树莓派的I2C通道编号

芯片地址

定义 I2C控制对象

```
class MAX30102():
    # by default, this assumes that the device is at 0x57 on channel 1
    def __init__(self, channel=1, address=0x57):
        # I2C初始化
        self.address = address
        self.channel = channel
        self.bus = smbus.SMBus(self.channel)

        # 芯片复位
        self.reset()
        sleep(1)  # wait 1 sec

        # 读取中断状态
        reg_data = self.bus.read_i2c_block_data(self.address, REG_INTR_STATUS_1, 1)
        print("reset state",reg_data)

        # 进行初始化
        self.setup()

    def reset(self):
        """
        芯片复位
        """
        self.bus.write_i2c_block_data(self.address, REG_MODE_CONFIG, [0x40])
```

```
def setup(self, led_mode=0x03):
    """
    芯片设置
    """
    # 中断设置
    self.bus.write_i2c_block_data(self.address, REG_INTR_ENABLE_1, [0xc0])
    self.bus.write_i2c_block_data(self.address, REG_INTR_ENABLE_2, [0x00])

    # FIFO写指针
    self.bus.write_i2c_block_data(self.address, REG_FIFO_WR_PTR, [0x00])
    # OVF_COUNTER[4:0]
    self.bus.write_i2c_block_data(self.address, REG_OVF_COUNTER, [0x00])
    # FIFO_读指针
    self.bus.write_i2c_block_data(self.address, REG_FIFO_RD_PTR, [0x00])

    # sample avg = 4, fifo rollover = false, fifo almost full = 15
    self.bus.write_i2c_block_data(self.address, REG_FIFO_CONFIG, [0x4f])

    # 0x02 for read-only, 0x03 for SpO2 mode, 0x07 multimode LED
    self.bus.write_i2c_block_data(self.address, REG_MODE_CONFIG, [led_mode])
    # 0b 0010 0111
    # SPO2_ADC range = 4096nA, SPO2 sample rate = 100Hz, LED pulse-width = 411uS
    self.bus.write_i2c_block_data(self.address, REG_SPO2_CONFIG, [0x27])

    # choose value for ~7mA for LED1
    self.bus.write_i2c_block_data(self.address, REG_LED1_PA, [0x24])
    # choose value for ~7mA for LED2
    self.bus.write_i2c_block_data(self.address, REG_LED2_PA, [0x24])
    # choose value fro ~25mA for Pilot LED
    self.bus.write_i2c_block_data(self.address, REG_PILOT_PA, [0x7f])
```

设置完成后
芯片上的LED小灯会亮起

```
def shutdown(self):
    """
    设备关闭
    """
    self.bus.write_i2c_block_data(self.address, REG_MODE_CONFIG, [0x80])


def read_fifo(self):
    """
    从数据寄存器中读取一个样本.
    """
    red_led = None
    ir_led = None

    # read 1 byte from registers (values are discarded)
    reg_INTR1 = self.bus.read_i2c_block_data(self.address, REG_INTR_STATUS_1, 1)
    reg_INTR2 = self.bus.read_i2c_block_data(self.address, REG_INTR_STATUS_2, 1)

    # read 6-byte data from the device
    d = self.bus.read_i2c_block_data(self.address, REG_FIFO_DATA, 6)

    # mask MSB [23:18]
    red_led = (d[0] << 16 | d[1] << 8 | d[2]) & 0x03FFFF
    ir_led = (d[3] << 16 | d[4] << 8 | d[5]) & 0x03FFFF

    return red_led, ir_led
```

注意数据的排列顺序

测试程序

```
if __name__ == "__main__":

    import numpy as np
    import cv2
    buff_size = 100


    # 定义传感器
    sensor = MAX30102()
    flag_finger = False
    ir_data = []
    red_data = []
```

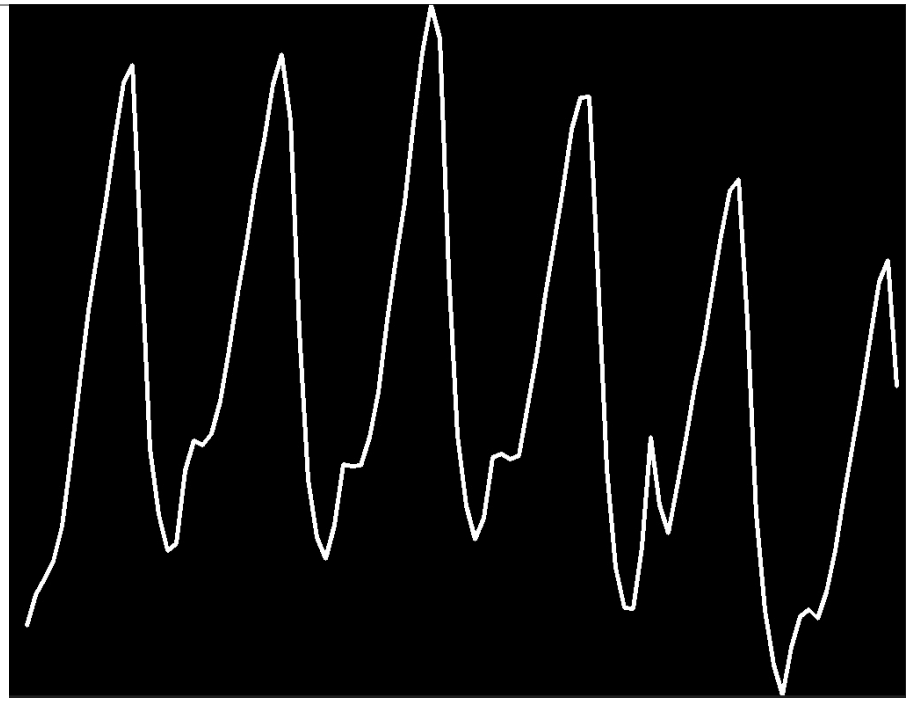
```
try:
    while True:

        # 获得FIFO中的样本数目
        num_bytes = sensor.get_data_present()
        if num_bytes > 0:
            # 依次读取数据
            while num_bytes > 0:
                red, ir = sensor.read_fifo()
                num_bytes -= 1
                ir_data.append(ir)
                red_data.append(red)
                # print("ir: %d  red :%d"%(ir,red))

                # 大于100个样本则删除开头的样本
                while len(ir_data) > buff_size:
                    ir_data.pop(0)
                    red_data.pop(0)
                if len(ir_data) == buff_size:
                    if (np.mean(ir_data) < 50000 and np.mean(red_data) < 50000):
                        flag_finger = False
                    else:
                        flag_finger = True
```

根据均值判断是否有手指

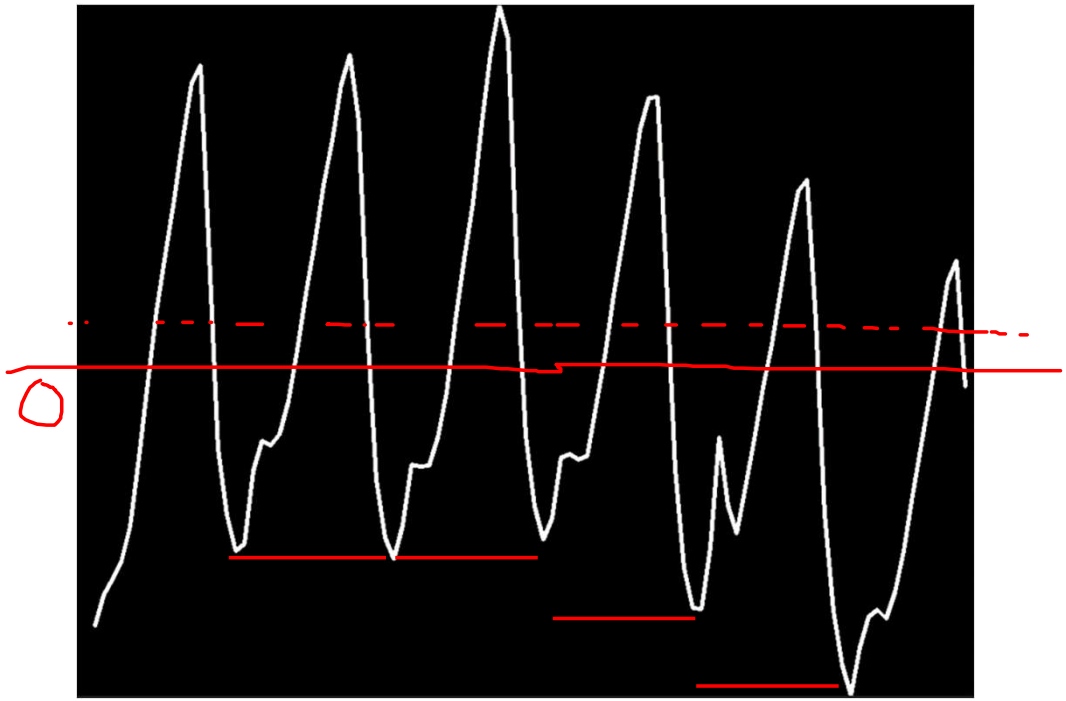
数据保存与显示

```
        sleep(0.01)
        if not flag_finger:
            print("flag_finger not find")
        else:
            img = draw_pic(ir_data)
            h,w,_ = np.shape(img)
            print(np.shape(img))
            if h<2500:
                np.save("ir.npy", np.array(ir_data), allow_pickle=True)
                cv2.imwrite("img.jpg",img)
except KeyboardInterrupt:
    print('\n Ctrl + C QUIT')
finally:
    sensor.shutdown()
```

```python
def draw_pic(data,h=2500,w=1020):

    if  max(data)-min(data)>2500:
        print(max(data),min(data))
        return np.zeros((h,w,3),np.uint8)
    else:
        h = max(data)-min(data)

    img = np.zeros((h,w,3),np.uint8)
    N  = len(data)

    for i in range(1,N):
        p1_x = (i-1)*10+20
        p2_x = i*10+20

        p1_y = -(data[i-1] - min(data)) + h
        p2_y = -(data[i] - min(data)) + h

        if p1_y<0:
            p1_y = 0
        elif p1_y>h-1:
            p1_y = h-1

        if p2_y<0:
            p2_y = 0
        elif p2_y>h-1:
            p2_y = h-1

        cv2.line(img,(int(p1_x),int(p1_y)),(int(p2_x),int(p2_y)),(255,255,255),3)
    return img
```



心率计算



计算IR数据中波谷的位置

并通过计算平均的周期，进而计算心率（下/每分钟）

代码实现：

```
# 采样率
SAMPLE_FREQ = 25
# 均值滤波
MA_SIZE = 4
# 缓冲区大小
BUFFER_SIZE = 100
```

```
def calc_hr(ir_data):
    # 计算均值
    ir_mean = int(np.mean(ir_data))

    # 减均值，前面加- ，为了计算波谷
    x = -1 * (np.array(ir_data) - ir_mean)

    # 进行均值滤波
    for i in range(x.shape[0] - MA_SIZE):
        x[i] = np.sum(x[i:i+MA_SIZE]) / MA_SIZE

    # 计算阈值，将阈值设置在30-60之间
    n_th = int(np.mean(x))
    n_th = 30 if n_th < 30 else n_th  # min allowed
    n_th = 60 if n_th > 60 else n_th  # max allowed

    # 获取信号中波谷的个数，以及波谷的位置
    ir_valley_locs, n_peaks = find_peaks(x, BUFFER_SIZE, n_th, min_dist=4, max_num=15)

    # 计算各个波谷的间距之和
    peak_interval_sum = 0
```

```
if n_peaks >= 2:
    for i in range(1, n_peaks):
        peak_interval_sum += (ir_valley_locs[i] - ir_valley_locs[i-1])
    peak_interval_sum = int(peak_interval_sum / (n_peaks - 1))

    hr = int(SAMPLE_FREQ * 60 / peak_interval_sum)
    hr_valid = True
else:
    hr = -999  # unable to calculate because # of peaks are too small
    hr_valid = False

return hr_valid, hr
```

```python
# 寻找峰值，峰值的高度>min_height，峰值最多 max_num 个
#  峰值之间的间隔要<min_dist
# 返回峰值（波谷）的位置以及峰值的个数
def find_peaks(x, size, min_height, min_dist, max_num):

    # 寻找峰值
    ir_valley_locs, n_peaks = find_peaks_above_min_height(x, size, min_height, max_num)

    # 消除较大的峰值
    ir_valley_locs, n_peaks = remove_close_peaks(n_peaks, ir_valley_locs, x, min_dist)

    n_peaks = min([n_peaks, max_num])

    return ir_valley_locs, n_peaks
```

```python
# 寻找大于min_height的峰值
def find_peaks_above_min_height(x, size, min_height, max_num):

    i = 0
    n_peaks = 0
    ir_valley_locs = []  # [0 for i in range(max_num)]
    while i < size - 1:

        # 寻找上升的潜在峰值
        if x[i] > min_height and x[i] > x[i-1]:
            n_width = 1

            # 判断平台的情况
            while i + n_width < size - 1 and x[i] == x[i+n_width]:  # find flat peaks
                n_width += 1

            # x[i]>x[i-1] 且 x[i] > x[i+n_width] 说明发现峰值
            if x[i] > x[i+n_width] and n_peaks < max_num:  # find the right edge of peaks
                ir_valley_locs.append(i)
                n_peaks += 1
                i += n_width + 1
            else:
                i += n_width
        else:
            i += 1

    return ir_valley_locs, n_peaks
```

```python
def remove_close_peaks(n_peaks, ir_valley_locs, x, min_dist):

    # 根据幅度值进行排序
    sorted_indices = sorted(ir_valley_locs, key=lambda i: x[i])
    sorted_indices.reverse()

    i = -1
    while i < n_peaks:
        old_n_peaks = n_peaks
        n_peaks = i + 1

        # 以i为基准 j逐渐增加
        # 判断i，j 之间的距离 小于min_dist，则去掉该点，后面的点依次前移动

        # i 从-1 开始，表示距离起始位置小于 min_dist 的点都清除
        j = i + 1
        while j < old_n_peaks:
            n_dist = (sorted_indices[j] - sorted_indices[i]) if i != -1 else (sorted_indices[j] + 1)
            if n_dist > min_dist or n_dist < -1 * min_dist:
                sorted_indices[n_peaks] = sorted_indices[j]
                n_peaks += 1
            j += 1
        i += 1

    sorted_indices[:n_peaks] = sorted(sorted_indices[:n_peaks])

    return sorted_indices, n_peaks
```

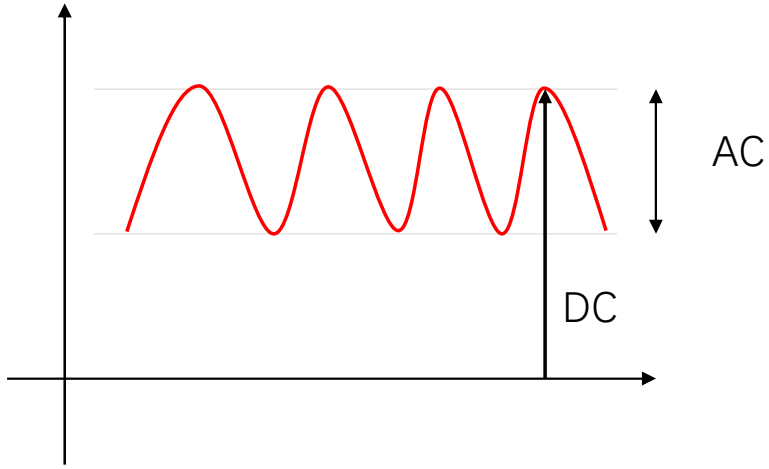
```
# 测试部分
if __name__ == "__main__":
    from max30102 import MAX30102
    import numpy as np
    import time
    buff_size = 100


    # 定义传感器
    sensor = MAX30102()
    flag_finger = False
    ir_data = []
    red_data = []
    bpms = []

    hr_mean = None
```

```
try:
    while True:

        # 获得FIFO中的样本数目
        num_bytes = sensor.get_data_present()
        if num_bytes > 0:
            # 依次读取数据
            while num_bytes > 0:
                red, ir = sensor.read_fifo()
                num_bytes -= 1
                ir_data.append(ir)
                red_data.append(red)
                # print("ir: %d  red :%d"%(ir,red))

                # 大于100个样本则删除开头的样本
                while len(ir_data) > buff_size:
                    ir_data.pop(0)
                    red_data.pop(0)

                if len(ir_data) == buff_size:
                    if (np.mean(ir_data) < 50000 and np.mean(red_data) < 50000):
                        flag_finger = False
                        hr_mean = None
                    else:
                        flag_finger = True
                        # 计算心率
                        hr_valid, hr = calc_hr(red_data)
                        if hr_valid:
                            bpms.append(hr)

                            while len(bpms)>4:
                                bpms.pop(0)

                            hr_mean = np.mean(bpms)
```

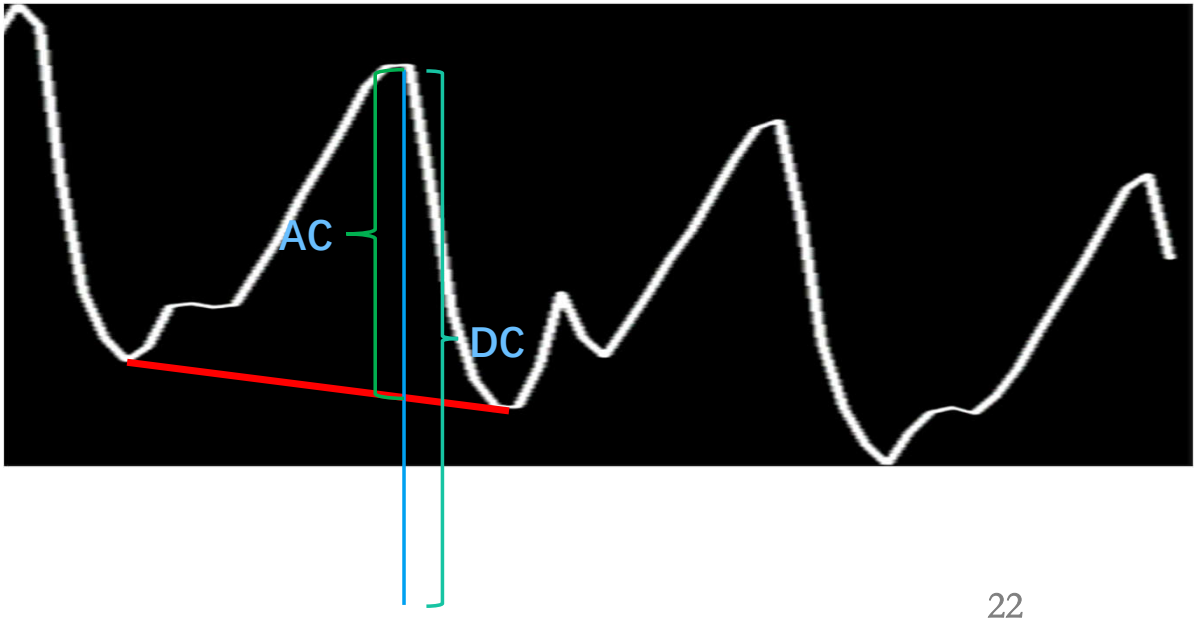
```
time.sleep(1)
if not flag_finger:
    print("flag_finger not find")
else:
    if not hr_mean is None:
        print("BPM: %.2f"%(hr_mean))
```

血氧饱和度（SPO2）

$$R=\frac{\frac{AC_{red}}{DC_{red}}}{\frac{AC_{IR}}{DC_{IR}}}$$



$$SPO2=-45.06R^2+30.054R+94.845$$



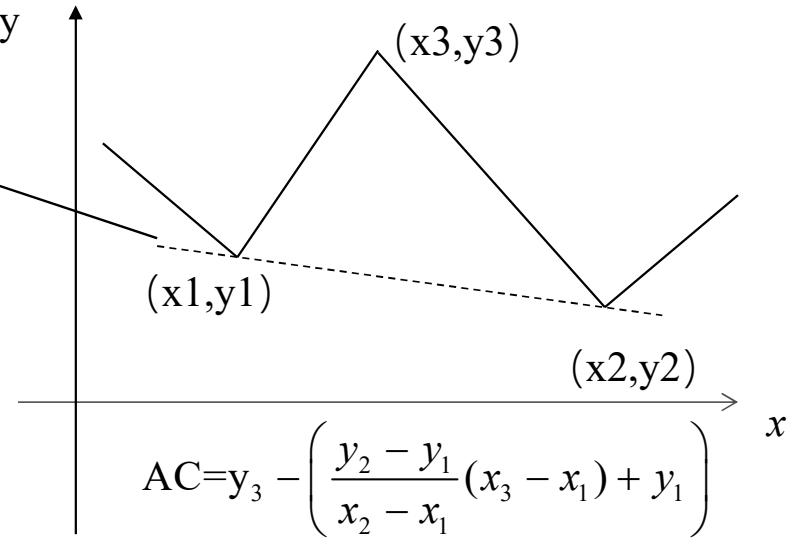
```
def calc_hr_spo2(ir_data,red_data):

    # 计算均值
    ir_mean = int(np.mean(ir_data))

    # 减均值，前面加- ，为了计算波谷
    x = -1 * (np.array(ir_data) - ir_mean)

    # 进行均值滤波
    for i in range(x.shape[0] - MA_SIZE):
        x[i] = np.sum(x[i:i+MA_SIZE]) / MA_SIZE

    # 计算阈值，将阈值设置在30-60之间
    n_th = int(np.mean(x))
    n_th = 30 if n_th < 30 else n_th  # min allowed
    n_th = 60 if n_th > 60 else n_th  # max allowed

    # 获取信号中波谷的个数，以及波谷的位置
    ir_valley_locs, n_peaks = find_peaks(x, BUFFER_SIZE, n_th, min_dist=4, max_num=15)

    # 计算各个波谷的间距之和
    peak_interval_sum = 0
    if n_peaks >= 2:
        for i in range(1, n_peaks):
            peak_interval_sum += (ir_valley_locs[i] - ir_valley_locs[i-1])
        peak_interval_sum = int(peak_interval_sum / (n_peaks - 1))

        hr = int(SAMPLE_FREQ * 60 / peak_interval_sum)
        hr_valid = True
    else:
        hr = -999  # unable to calculate because # of peaks are too small
        hr_valid = False
        spo2 = -999  # do not use SPO2 since valley loc is out of range
        spo2_valid = False

        return hr_valid, hr,spo2_valid,spo2
```

```
# 25 samples per secon
SAMPLE_FREQ = 25
# taking moving averag
# in algorithm.h, "DON
MA_SIZE = 4
# sampling frequency *
BUFFER_SIZE = 100
```

代码实现，前半部分和心率计算相同，计算获取波谷的点位

```
i_ratio_count = 0
ratio = []

for k in range(n_peaks-1):
    red_dc_max = -16777216
    ir_dc_max = -16777216

    if ir_valley_locs[k+1] - ir_valley_locs[k] > 3:
        # 找到两个波谷之间的最大值
        for i in range(ir_valley_locs[k], ir_valley_locs[k+1]):
            if ir_data[i] > ir_dc_max:
                ir_dc_max = ir_data[i]
                ir_dc_max_index = i
            if red_data[i] > red_dc_max:
                red_dc_max = red_data[i]
                red_dc_max_index = i

        # 计算 AC 值
        red_ac = int((red_data[ir_valley_locs[k+1]] - red_data[ir_valley_locs[k]]) * (red_dc_max_index - ir_valley_locs[k]))
        red_ac = red_data[ir_valley_locs[k]] + int(red_ac / (ir_valley_locs[k+1] - ir_valley_locs[k]))
        red_ac = red_data[red_dc_max_index] - red_ac  # subtract linear DC components from raw

        ir_ac = int((ir_data[ir_valley_locs[k+1]] - ir_data[ir_valley_locs[k]]) * (ir_dc_max_index - ir_valley_locs[k]))
        ir_ac = ir_data[ir_valley_locs[k]] + int(ir_ac / (ir_valley_locs[k+1] - ir_valley_locs[k]))
        ir_ac = ir_data[ir_dc_max_index] - ir_ac  # subtract linear DC components from raw

        nume = red_ac * ir_dc_max
        denom = ir_ac * red_dc_max
```

$$y=\frac{y_2-y_1}{x_2-x_1}(x-x_1)+y_1$$



$$AC=y_3-\left(\frac{y_2-y_1}{x_2-x_1}(x_3-x_1)+y_1\right)$$

$$R=\frac{\frac{AC_{red}}{DC_{red}}}{\frac{AC_{IR}}{DC_{IR}}}$$

```
        if (denom > 0 and i_ratio_count < 5) and nume != 0:

            # 这里对ratio计算的结果*100，进行精度扩大
            ratio.append(int(((nume * 100) & 0xffffffff) / denom))
            i_ratio_count += 1


# 选取中值作为输出
ratio = sorted(ratio)  # sort to ascending order
mid_index = int(i_ratio_count / 2)

ratio_ave = 0
if mid_index > 1:
    ratio_ave = int((ratio[mid_index-1] + ratio[mid_index])/2)
else:
    if len(ratio) != 0:
        ratio_ave = ratio[mid_index]

# 对ratio的合理性进行判断
if ratio_ave > 2 and ratio_ave < 184:
    # -45.060 * ratioAverage * ratioAverage / 10000 + 30.354 * ratioAverage / 100 + 94.845
    spo2 = -45.060 * (ratio_ave**2) / 10000.0 + 30.054 * ratio_ave / 100.0 + 94.845
    spo2_valid = True
else:
    spo2 = -999
    spo2_valid = False

return  hr_valid, hr,spo2_valid,spo2
```

$$SPO2=-45.06R^2+30.054R+94.845$$

```
# 测试部分
if __name__ == "__main__":
    from max30102 import MAX30102
    import numpy as np
    import time
    buff_size = 100

    # 定义传感器
    sensor = MAX30102()
    flag_finger = False
    ir_data = []
    red_data = []
    bpms = []
    spo2s = []
    hr_mean = None
    spo2_mean = None
```

```
while True:
    # 获得FIFO中的样本数目
    num_bytes = sensor.get_data_present()
    if num_bytes > 0:
        # 依次读取数据
        while num_bytes > 0:
            red, ir = sensor.read_fifo()
            num_bytes -= 1
            ir_data.append(ir)
            red_data.append(red)
            # print("ir: %d  red :%d"%(ir,red))

            # 大于100个样本则删除开头的样本
            while len(ir_data) > buff_size:
                ir_data.pop(0)
                red_data.pop(0)

            if len(ir_data) == buff_size:
                if (np.mean(ir_data) < 50000 and np.mean(red_data) < 50000):
                    flag_finger = False
                    hr_mean = None
                    spo2_mean = None
                else:
                    flag_finger = True
                    # 计算心率
                    hr_valid, hr,spo2_valid,spo2 = calc_hr_spo2(ir_data,red_data)
                    if hr_valid and spo2_valid:
                        bpms.append(hr)
                        spo2s.append(spo2)
                    while len(bpms)>4:
                        bpms.pop(0)
                        spo2s.pop(0)
                        hr_mean = np.mean(bpms)
                        spo2_mean=np.mean(spo2s)
```

```
        time.sleep(1)
        if not flag_finger:
            print("flag_finger not find")
        else:
            if not hr_mean is None  and not spo2_mean is None:
                print("BPM:%.2f  spo2: %.2f%%"%(hr_mean,spo2_mean))



except KeyboardInterrupt:
    print('\n Ctrl + C QUIT')
finally:
    sensor.shutdown()
```