

版本：V3.0



杭州暖芯迦电子科技有限公司

PWM2004 PPG+SpO2 光电多参数人 体健康检测模块用户手册

批准日期:2020 年 8 月 17 日 实施日期:2020 年 8 月 17 日

目录

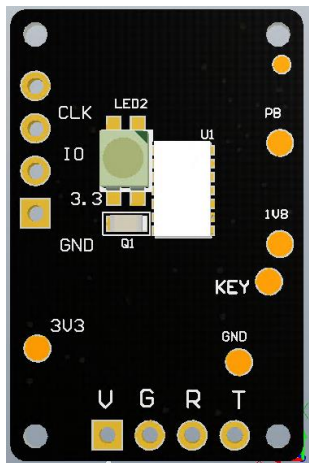
1 概述	4
2 特点	4
3 应用范围.....	5
4 电气特性.....	5
5 串口协议帧定义.....	7
5.1 帧头.....	7
5.2 数据包.....	8
5.2.1 数据头字节.....	8
5.2.2 关键数据.....	8
6 串口命令定义.....	8
6.1 主机发送指令.....	8
6.1.1 启动一次测量命令.....	8
6.1.2 停止一次测量命令.....	9
6.1.3 设置人体参数命令.....	9
6.1.4 设置密钥参数命令.....	10
6.2 从机返回数据格式.....	10
6.2.1 从机启动测试返回格式.....	10
6.2.2 从机波形数据返回格式.....	11
6.2.3 从机数据测试成功返回格式.....	11
6.2.4 从机停止测试返回格式.....	13
6.2.5 从机设置人体参数返回指令格式.....	13
6.2.6 从机设置客户密钥返回指令格式.....	13
7 蓝牙术语与定义.....	14
8 蓝牙广播.....	14
8.1 广播类型.....	14
8.2 广播规则.....	14
9 蓝牙数据通信.....	15
9.1 私有通信协议架构.....	15
9.1.1 协议架构图.....	15
9.1.2 Layer 0—Hardware Layer	15
9.1.3 Layer 1—Profile Layer	16
9.1.4 Layer 2 —Transport Layer.....	16
9.1.5 Layer 3 —Application Layer.....	17
9.1.6 举例.....	18
10 蓝牙功能命令定义.....	18
10.1 设置命令（0x03）	18
10.1.1 设置命令 Key ID 列表	18
10.1.2 设定人体参数 Key（0x03）	19
10.1.3 设定人体参数返回 Key（0x03）	19
10.1.4 设定用户密钥参数 Key（0x07）	19
10.1.5 设定用户密钥参数返回 Key（0x07）	19
10.2 数据同步（0x05）	20

10.2.1 结果数据返回 Key (0x03)	20
10.2.2 同步当前波形数据返回 Key (0x09)	20
10.3 控制命令 (0x08)	20
10.3.1 控制命令 key ID 列表.....	20
10.3.2 设备启动一次测量 key (0x01)	21
10.3.3 设备启动一次测量返回 key (0x01)	21
10.3.4 设备停止当前测量 key (0x02)	22
10.3.5 设备停止当前测量返回 key (0x02)	22
11 接口说明	23
11.1 引脚说明.....	23
12 功能框图	24
13 尺寸	25
13.1 触摸区域说明.....	26
14 典型应用与使用注意事项.....	27
15 测试 APP 和 UART 使用	28
15.1 测试 APP 使用说明	28
15.2 蓝牙 SDK 和 UART 使用说明	28
16 模块控制流程图.....	29
17 加密规则及数据解密算法.....	30
17.1 加密规则.....	30
17.2 解密算法.....	30

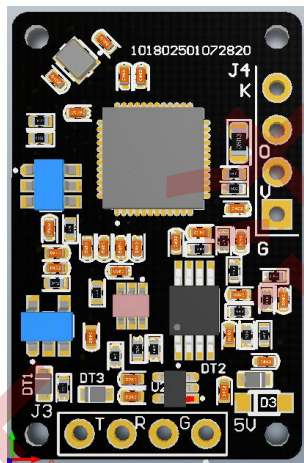
1 概述

PWM2004 光电模块是一款可以测量心率、呼吸率、血氧饱和度等多种人体参数的模块，可以通过有线（UART）或无线（BLE4.0）连接的方式从模块读取测量数据。

模块可另提供移动端 SDK 开发包，当采用 BLE4.0 连接时，方便进行移动应用端软件的开发。



模块正面



模块背面

2 特点

外形尺寸： 17.5mm X 27mm X 4.1mm

输入电压： DC5V

输入电流： 50mA

可测量参数： 心率、心律不齐、血氧饱和度、心率变异性、呼吸率、动脉硬化指数、焦虑指数、脉象。

参数测量范围、正常参考范围和测量精度：

测量参数	测量范围	正常参考范围	测量精度
HR 心率	30~200bpm(次/分)	安静状态下，成人正常心率为 60~100 次/分钟，理想心率应为 55~70 次/分钟(运动员的心率较普通成人偏慢，一般为 50 次/分钟左右)。	±3 次/分
ARR 心律不齐	有/无	无	90%以上
HRV 心率变异性	0~511ms	20~200ms	±20ms
RR 呼吸率	10~30bpm(次/分)	正常成年人每分钟呼吸大约 12~20 次。呼吸与脉搏的比是 1: 4，即每呼吸 1 次，脉搏搏动 4 次。小儿呼吸比成人快，每分钟可达 20~30 次；新生儿的呼吸频率可达每分钟 44 次。	±2 次/分
SI 动脉硬化指数	0.1~9.9	SI < 4.0	±0.5
SNA 焦虑指数	0.1~9.9	4 以下不焦虑、4~6 轻度焦虑、6~8 中度焦虑、8~9.9 重度焦虑	±0.5
SpO2 血氧饱和度	1~100%	SpO2 正常应不低于 94%，在 94%以下为供氧不足	±1%
PT 脉象	平脉、弦脉、滑脉、迟脉、缓脉、数脉、结脉、止脉、促脉	/	/

模块通讯方式： 有线 UART；无线蓝牙 BLE4.0。

3 应用范围

家庭医疗管理、健康智能硬件、健康管理平台。

4 电气特性

环境要求：

环境要求	
工作环境温度	-40℃ ~ +85℃
工作环境湿度	20% ~ 80%
存储环境温度	-40℃ ~ +85℃
存储环境湿度	10% ~ 80%

串口波特率：57600

串口设置：N 8 1

流控：无

数据格式：二进制

符号	参数	测试条件	最小	典型	最大	单位
V _{in}	工作电压	—	3.7	4.2	5.5	V
I _{DD}	工作电流	输出引脚无负载	—	—	50	mA
V _{IL}	TX 引脚低电平输入电压	—	—	—	0.8	V
V _{IH}	TX 引脚高电平输入电压	—	2.8	—	3.3	V
V _{OL}	RX 引脚低电平输出电压	IOL=TBD	—	—	0.4	V
V _{OH}	RX 引脚高电平输出电压	IOL=TBD	2.9	—	3.3	V
t _{SST}	系统启动时间	—	500	—	—	mS
RRVDD	VDD 上升速率	—	TBD	—	—	V/ms
BRPON	上电波特率	—	—	57600	—	Hz

BLE接收器特性

参数	条件	最小	典型	最大	单位
灵敏度@0.1% BER	—	—	-98	—	dBm
最大接收信号@0.1% BER	—	0	—	—	dBm
共信道C/I	—	—	10	—	dB
邻道选择性C/I	F = F ₀ + 1 MHz	—	-5	—	dB
	F = F ₀ - 1 MHz	—	-5	—	dB
	F = F ₀ + 2 MHz	—	-25	—	dB
	F = F ₀ - 2 MHz	—	-35	—	dB
	F = F ₀ + 3 MHz	—	-25	—	dB
	F = F ₀ - 3 MHz	—	-45	—	dB
抗带外阻塞性能	30 MHz - 2000 MHz	-10	—	—	dBm
	2000MHz - 2400MHz	-27	—	—	dBm
	2500MHz - 3000MHz	-27	—	—	dBm
	3000MHz - 12.5GHz	-10	—	—	dBm
互调性能	—	-36	—	—	dBm

BLE发射器特性

参数	条件	最小	典型	最大	单位
射频发射功率	—	—	7.5	10	dBm
射频功率控制范围	—	—	25	—	dB
邻道发射功率	$F = F_0 + 1 \text{ MHz}$	—	-14.6	—	dBm
	$F = F_0 - 1 \text{ MHz}$	—	-12.7	—	dBm
	$F = F_0 + 2 \text{ MHz}$	—	-44.3	—	dBm
	$F = F_0 - 2 \text{ MHz}$	—	-38.7	—	dBm
	$F = F_0 + 3 \text{ MHz}$	—	-29.2	—	dBm
	$F = F_0 - 3 \text{ MHz}$	—	-45	—	dBm
	$F = F_0 + > 3 \text{ MHz}$	—	-50	—	dBm
	$F = F_0 - > 3 \text{ MHz}$	—	-50	—	dBm
Δf_{1avg}	—	—	—	265	kHz
Δf_{2max}	—	247	—	—	dBm
$\Delta f_{2avg}/\Delta f_{1avg}$	—	—	-0.92	—	dBm
ICFT	—	—	-10	—	kHz
频率漂移率	—	—	0.7	—	kHz/50 μ s
频率漂移	—	—	2	—	kHz

5 串口协议帧定义

本协议中，如无特别说明，所有数值均表示十六进制格式。

协议帧的格式如下：

MSB	LSB
帧头	数据包
8 bytes	0-512 bytes

5.1 帧头

MSB				LSB
引导字	协议版本	长度 ⁽¹⁾	命令字节	序列号 ⁽²⁾
1 bytes	1 bytes	2 bytes	2 bytes	2 bytes
0xFE	0x01	长度	命令号	序列号

注(1)：长度为整个协议帧的所有数据字节长度；

注(2)：序列号为发送的数据帧序号，从 0x0000 开始，每发新的一帧，编号加 1，发送和接收的数据帧分开编号。

5.2 数据包

数据包由不同类型的数据帧构成，有三种类型的数据帧，定义如下：

类型字节	长度字节	数据字节
0A	不定	由长度位决定
12	1 bytes	由长度位决定
18	不定	由长度位决定

主机与从机的通信数据，在类型为 12 的数据包中传输，该包中的数据字节定义如下：

MSB	LSB
数据头字节	关键数据
2 bytes	0-510 bytes

5.2.1 数据头字节

数据头字节由命令字和关键字构成：

MSB	LSB
命令字	关键字
1 bytes	1 bytes

不同的命令字有不同的定义，同一命令字下不同的关键字实现不同的功能，详见下节定义。

5.2.2 关键数据

详见下节定义。

6 串口命令定义

6.1 主机发送指令

6.1.1 启动一次测量命令

帧内定义	值	说明
------	---	----

帧头	引导值 (1 Bytes)	FE	
	版本号 (1 Bytes)	01	
	长度 (2 Bytes)	00 11	
	命令号 (2 Bytes)	75 31	
	序列号 (2 Bytes)	XX XX	当前帧的序号
数据包	类型字节 (1 Bytes)	0A	
	长度字节 (1 Bytes)	00	
	类型字节 (1 Bytes)	12	
	长度字节 (1 Bytes)	03	
	数据字节 (3 Bytes)	08 01 01 (08 01 02)	触摸启动 (光感启动) 测量
	类型字节 (1 Bytes)	18	
	长度字节	00	

模块支持触摸启动和光感启动，可选择其中任一种启动模块。

6.1.2 停止一次测量命令

帧内定义		值	说明
帧头	引导值 (1 Bytes)	FE	
	版本号 (1 Bytes)	01	
	长度 (2 Bytes)	00 10	
	命令号 (2 Bytes)	75 31	
	序列号 (2 Bytes)	XX XX	当前帧的序号
数据包	类型字节 (1 Bytes)	0A	
	长度字节 (1 Bytes)	00	
	类型字节 (1 Bytes)	12	
	长度字节 (1 Bytes)	02	
	数据字节 (2 Bytes)	08 02	停止一次参数测量
	类型字节 (1 Bytes)	18	
	长度字节 (1 Bytes)	00	

6.1.3 设置人体参数命令

帧内定义		值	说明
帧头	引导值 (1 Bytes)	FE	
	版本号 (1 Bytes)	01	
	长度 (2 Bytes)	00 13	
	命令号 (2 Bytes)	75 31	

	序列号 (2 Bytes)	XX XX	当前帧的序号
数据包	类型字节 (1 Bytes)	0A	
	长度字节 (1 Bytes)	00	
	类型字节 (1 Bytes)	12	
	长度字节 (1 Bytes)	05	
	数据字节 (5 Bytes)	03 03 X1 X2 X3	
	类型字节 (1 Bytes)	18	
	长度字节 (1 Bytes)	00	

X1 X2 X3 分别代表体重 (kg), 身高 (cm), 性别&年龄。转为 HEX。

性别&年龄字节最高位为性别位, 1 代表男性, 0 代表女性; 低 7 位为年龄数据。

6.1.4 设置密钥参数命令

用户使用本模块时, 必须按照本协议命令发送密钥参数, 以确保用户数据的安全性。

帧内定义		值	说明
帧头	引导值 (1 Bytes)	FE	
	版本号 (1 Bytes)	01	
	长度 (2 Bytes)	00 1E	
	命令号 (2 Bytes)	75 31	
	序列号 (2 Bytes)	XX XX	当前帧的序号
数据包	类型字节 (1 Bytes)	0A	
	长度字节 (1 Bytes)	00	
	类型字节 (1 Bytes)	12	
	长度字节 (1 Bytes)	12	
	数据字节 (18 Bytes)	03 07 X1 ---X16	X1 ---X16 为用户密钥

6.2 从机返回数据格式

6.2.1 从机启动测试返回格式

主机发送开始测试指令后, 设备会发送该数据帧进行应答。

帧内定义		值	说明
帧头	引导值 (1 Bytes)	FE	
	版本号 (1 Bytes)	01	
	长度 (2 Bytes)	00 1E	
	命令号 (2 Bytes)	27 12	
	序列号 (2 Bytes)	XX XX	当前帧的序号
数据	类型字节 (1 Bytes)	0A	
	长度字节 (1 Bytes)	00	

包	类型字节 (1 Bytes)		12	
	长度字节 (1 Bytes)		12	
	数据字节	数据内容 1 (2 Bytes)	08 01	
		数据内容 2 (16 Bytes)	00 00 00 00	

6.2.2 从机波形数据返回格式

帧内定义			值	说明
帧头	引导值 (1 Bytes)		FE	
	版本号 (1 Bytes)		01	
	长度 (2 Bytes)		00 2E	
	命令号 (2 Bytes)		27 12	
	序列号 (2 Bytes)		XX XX	当前帧的序号
数据包	类型字节 (1 Bytes)		0A	
	长度字节 (1 Bytes)		00	
	类型字节 (1 Bytes)		12	
	长度字节 (1 Bytes)		22	
	数据字节	数据类型 (2 Bytes)	05 09	表示该数据为波形数据
		数据内容 1 (32 Bytes)	XX XX XX XX	详见举例

举例：

FE 01 00 2E 27 12 00 16 0A 00 12 22 05 09 71 00 70 40 6F 00 6E C0 6E C0 71 C0 71 00 74 00 95 00 BB 00 B6 40 97 00 82 00 78 40 70 40 6D 40

解释：

05 09 表示波形数据返回，后面为波形数据，波形数据用 16 进制表示，共 32 字节，每 2 个连续的字节（例如：71 00）为一个波形数据，波形数据共 10bit，高 8bit 在前，低 8bit 在后，在低 8bit 中高 2bit 有效。

例如 70 40 这 2 个字节的数据，可以按照如下格式转换为波形数据：(unsigned int) (0x70 << 2) + (0x40 >> 6)，数据采样点间隔时间为 40ms。

6.2.3 从机数据测试成功返回格式

帧内定义			值	说明
帧头	引导值 (1 Bytes)		FE	
	版本号 (1 Bytes)		01	
	长度 (2 Bytes)		00 2E	
	命令号 (2 Bytes)		27 12	

数据包	序号 (2 Bytes)	XX XX	当前帧的序号
	类型字节 (1 Bytes)	0A	
	长度字节 (1 Bytes)	00	
	类型字节 (1 Bytes)	12	
	长度字节 (1 Bytes)	22	
	数据类型 (2 Bytes)	05 03	表示该数据为测试结果
	数据内容 1 (12 Bytes)	XX XX XX XX	
	数据内容 2 (1 Bytes)	XX	SI 动脉硬化指数 (需除以 10 取小数)
	数据内容 3 (1 Bytes)	XX	SNA 焦虑指数 (需除以 10 取小数)
	数据内容 4 (1 Bytes)	XX	
	数据内容 5 (1 Bytes)	XX	
	数据内容 6 (1 Bytes)	XX	
	数据内容 7 (1 Bytes)	XX	HRV 心率变异性
	数据内容 8 (1 Bytes)	XX	ARR 心律不齐
	数据内容 9 (4 Bytes)	XX XX XX XX	
	数据内容 10 (1 Bytes)	XX	RR 呼吸率
	数据内容 11 (1 Bytes)	XX	
	数据内容 12 (1 Bytes)	XX	
	数据内容 13 (1 Bytes)	XX	HR 心率
	数据内容 14 (1 Bytes)	XX	PT 脉象
	数据内容 15 (1 Bytes)	XX	SpO2 血氧饱和度
	数据内容 16 (3 Bytes)	XX XX	

举例:

FE 01 00 2E 27 12 00 80 0A 00 12 22 05 03 00 42 00 79 00 00 00 00 00 00 00 00 19 26
59 00 00 25 02 00 00 00 00 0E A6 68 46 61 63 00 00 00

解释:

关键数据长度为 34 个字节, 05 03 表示测试结果, 其后为测试结果数据, 测试结果数据用 16 进制表示, 共 32 字节, 其中 19 26 59 为动脉硬化指数、焦虑指数 (均需除以 10 取小数); 25 为 HRV (心率变异性); 其余位置数据为保留值 (具体数值可能有变化)。心律不齐字节位置 01 为无心律不齐, 02 有心律不齐。血氧饱和度协议中为 0x63, 代表血氧饱和度百分比为 99%。

关于脉象的解析如下脉象字节各位组合: 如脉象字节为 0x23, 则脉象为滑缓脉象。

Bit3-0:=0 无脉象=1 平脉=2 弦脉=3 滑脉 Bit7-4: =1 迟脉=2 缓脉=3 数脉=4 结脉 =5 止脉=6 促脉

焦虑指数范围为 0.1-9.9。 4 以下不焦虑、4-6 轻度焦虑、6-8 中度焦虑、8-9.9 重度焦虑。

6.2.4 从机停止测试返回格式

主机发送停止测试指令后，设备会发送该数据帧进行应答。

帧内定义		值	说明
帧头	引导值（1 Bytes）	FE	
	版本号（1 Bytes）	01	
	长度（2 Bytes）	00 1E	
	命令号（2 Bytes）	27 12	
	序列号（2 Bytes）	XX XX	当前帧的序号
数据包	类型字节（1 Bytes）	0A	
	长度字节（1 Bytes）	00	
	类型字节（1 Bytes）	12	
	长度字节（1 Bytes）	12	
	数据字节	数据内容 1（2 Bytes）	08 02
	数据字节	数据内容 2（16 Bytes）	00 00 00 00

6.2.5 从机设置人体参数返回指令格式

主机发送开始测试指令后，设备会发送该数据帧进行应答。

帧内定义		值	说明
帧头	引导值（1 Bytes）	FE	
	版本号（1 Bytes）	01	
	长度（2 Bytes）	00 1E	
	命令号（2 Bytes）	27 12	
	序列号（2 Bytes）	XX XX	当前帧的序号
数据包	类型字节（1 Bytes）	0A	
	长度字节（1 Bytes）	00	
	类型字节（1 Bytes）	12	
	长度字节（1 Bytes）	12	
	数据字节	数据内容 1（2 Bytes）	03 03
	数据字节	数据内容 2（16 Bytes）	00 00 00 00

6.2.6 从机设置客户密钥返回指令格式

主机发送设置密钥指令后，设备会发送该数据帧进行应答。此帧数据的数据字节内容不加密。

帧内定义	值	说明
------	---	----

帧头	引导值（1 Bytes）		FE	
	版本号（1 Bytes）		01	
	长度（2 Bytes）		00 1E	
	命令号（2 Bytes）		27 12	
	序列号（2 Bytes）		XX XX	当前帧的序号
数据包	类型字节（1 Bytes）		0A	
	长度字节（1 Bytes）		00	
	类型字节（1 Bytes）		12	
	长度字节（1 Bytes）		12	
	数据字节	数据内容 1（2 Bytes）	03 07	
		数据内容 2（16 Bytes）	00 00 00 00	

7 蓝牙术语与定义

APP: 指安装在手机上的应用程序。

设备: 指智能模块产品。

主机: 本文中指安装有APP的手机。

从机: 本文中指使用低功耗蓝牙技术的设备。

8 蓝牙广播

8.1 广播类型

按照协议，本设计选取广播类型为非定向可连接广播，即 ADV_IND。

8.2 广播规则

广播规则定义如下：

- 1、从机正常广播时间间隔设置为1.25s。
- 2、从机有充电功能时，充电状态广播时间间隔设置为 62.5ms。
- 3、蓝牙服务

在设计中使用自定义以下属性：

Attribute Handle	UUID	Attribute Type	Attribute Value
0x000C	0xFEE7	Primary Service	wechat Service

- 4、蓝牙广播名为 PWM2004。

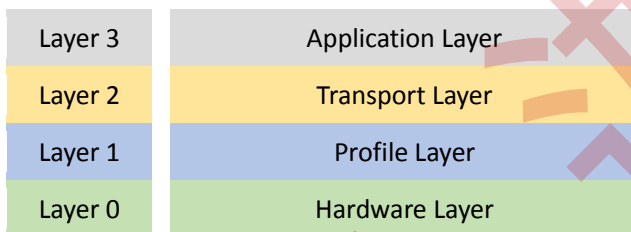
9 蓝牙数据通信

主机和从机处于连接状态时，双方进入数据通信模式。

9.1 私有通信协议架构

9.1.1 协议架构图

数据通信协议分为四层，如下所示：



9.1.2 Layer 0—Hardware Layer

9.1.2.1 Layer 0 描述

Layer 0 为蓝牙核心规范 4.0 内定义的蓝牙协议栈，主要是逻辑链路层控制和适配协议（L2CAP）和通用属性配置文件（GATT）。

根据蓝牙规范，在 L2CAP 层上传输的 Data PDU Payload 有两种：LL Control PDU 和 LL Data PDU，由 Data PDU Header 内的 LLID 字段确定：

LLID	Definition
00b	保留
01b	LL Data PDU 逻辑链路层数据 PDU，表示 L2CAP 报文延续，或者是一个空的 PDU
10b	LL Data PDU 逻辑链路层数据 PDU，表示 L2CAP 报文开始，或者是一个完整的报文
11b	LL Control PDU 逻辑链路层控制 PDU，用于管理连接

GATT 协议栈基于 L2CAP 层的 LL Data PDU。

9.1.3 Layer 1—Profile Layer

在蓝牙协议栈中，layer1 由微信定义的 UART Profile 实现，ServiceUUID 为 0xFEE7（该 uuid 经蓝牙官方授权）。蓝牙设备需要暴露 3 个特征值（Characteristics）：Write 特征值，Read 特征值，Indication 特征值。

蓝牙设备从 Write 特征值接受数据，从 Indication 特征值发送数据。Read 特征值内容为 6 字节的 MAC 地址，当 ios 上的其他 app 连接上设备时，设备不会再广播，微信会读取该特征值，以确定是否要连接该设备

write interface 和 receive interface 的 MTU 为 20 字节。

9.1.4 Layer 2 —Transport Layer

9.1.4.1 Layer 2 功能描述

Layer 2 为 Transport 层，在 Layer 1 层上实现数据的传输。

数据传输以大端（MSB）在前（高字节在前）。

9.1.4.2 Layer 2 数据包结构

Layer 2 Packet 由 Header 和 Payload 组成，定义如下：

MSB	LSB
Layer 2 Header	Layer 2 Payload
8 bytes	0-512 bytes

Layer 2 Header 部分定义：

MSB			LSB	
Preamble	Version	Length	Cmd Id	Sequence ID
1 bytes	1 bytes	2 bytes	2 bytes	2 bytes
0xFE	0x01	长度	命令号	序列号

Preamble: 数据头引导字段，固定为 0xFE。

Version: 本协议层的版本号，初始为 0x01。

Length: Layer 2 Header 及数据载荷部分的字节总长度。

Cmd Id: 命令号

Sequence ID: 数据包的序列号，从 0x0001 开始表示发送方发送的第一包数据，以后每

发送一次数据，递增一次。设备端发送的 **Sequence ID** 永不为 0，手机端发送回应的 **Sequence ID** 和设备端发送的相同。手机端主动发送的 push 命令 **Sequence ID** 永远为 0。数据包采用 Protoalbuf 打包。

9.1.5 Layer 3 —Application Layer

9.1.5.1 Layer 3 数据包结构

Layer 3 Packet 基于 Layer 2 Payload，主体由 protobuf 协议构成。

每个 Protobuf 包由如下 3 个部分组成：

类型字节	长度字节	数据字节
1 bytes	不定	由长度位决定

如果 protobuf 的长度字节为 0，数据字节不存在。

Layer 3 Packet 共有 3 个 protobuf 包分别为：

类型字节	长度字节	数据字节
0A	不定	由长度位决定
12	1 bytes	由长度位决定
18	不定	由长度位决定

主机与从机的通信数据，在类型为 12 的 protobuf 包中传输，该包中的数据字节定义如下：

MSB	Header	LSB	Payload (Key Value)
	2 bytes		0-510 bytes

其中 Header 定义：

MSB	Command ID	Key ID	LSB
	1 bytes	1 bytes	

Command ID: 命令字。

Key ID: 关键字

不同的 Command ID 有不同的 Payload 定义，详见下一节描述。

相同 Command ID 下拥有不同的 Key ID 以实现不同的功能，详见下一节描述。

9.1.6 举例

一包完整数据帧的内容为：FE 01 00 11 75 31 00 00 0A 00 12 03 08 01 01 18 00。

其中：FE 01 00 11 75 31 00 00 为 Layer 2 Header，解析如下：

Layer 2 Header

Preamble	Version	Length	Cmd Id	Sequence ID
1 bytes	1 bytes	2 bytes	2 bytes	2 bytes
0xFE	0x01	00 11 即 17	75 31 即 30001	00 00 即 0

0A 00 12 03 08 01 01 18 00 为 Layer 3 Packet，由 3 个 protobuf 包组成，解析如下：

类型字节	长度字节	数据字节
0A	00	无
12	03	08 01 01
18	00	无

类型 12 的 protobuf 包中的数据字节 08 01 01 解析如下：

MSB		LSB
Header		Payload (Key Value)
2 bytes		1 bytes
08	01	01

10 蓝牙功能命令定义

10.1 设置命令 (0x03)

10.1.1 设置命令 Key ID 列表

Key ID	定义
0x03	设定人体参数
0x03	设定人体参数返回
0x07	设定用户密钥
0x07	设定用户密钥

10.1.2 设定人体参数 Key (0x03)

key	体重 (kg)	身高 (cm)	性别&年龄
0x03	转为 HEX	转为 HEX	转为 HEX

性别&年龄字节最高位为性别位，1 代表男性，0 代表女性；低 7 位为年龄数据。

10.1.3 设定人体参数返回 Key (0x03)

MSB

LSB

Key Value（8 bytes）	
Return Key	Reserve
1 byte	7 bytes

Return Key	定义
0x00	人体参数设置成功
0x01	人体参数设置失败

10.1.4 设定用户密钥参数 Key (0x07)

key	X1		X16
0x07	密钥低字节		密钥高字节

10.1.5 设定用户密钥参数返回 Key (0x07)

Return Key	定义
0x00	客户密钥参数设置成功
0x01	客户密钥参数设置失败

10.2 数据同步（0x05）

10.2.1 结果数据返回 Key （0x03）

10.2.1.1 功能描述

设备端通过这个 key 向主机发送结果数据。

10.2.1.2 Key Value 内容

结果数据。

10.2.2 同步当前波形数据返回 Key （0x09）

10.2.2.1 功能描述

设备端通过这个 key 向手机端发送当前波形数据。

10.2.2.2 Key Value 内容

波形数据数据解析见 6.2.2 章节。

10.3 控制命令（0x08）

该命令实现主机对设备的控制或设备对主机的控制功能。

10.3.1 控制命令 key ID 列表

Key ID	定义
0x01	设备启动一次测量
0x01	设备启动一次测量返回
0x02	设备停止当前测量
0x02	设备停止当前测量返回

10.3.2 设备启动一次测量 key（0x01）

10.3.2.1 功能描述

主机通过该命令控制设备启动一次测量。

10.3.2.2 Key Value 内容

Return Key	定义
0x01	触摸启动测试
0x02	光感启动测试
0x03	（预留）

10.3.3 设备启动一次测量返回 key（0x01）

10.3.3.1 功能描述

设备通过该命令返回是否成功启动测量。

10.3.3.2 Key Value 内容

MSB	LSB
Key Value（8 bytes）	
Return Key	Reserve
1 byte	0 bytes

Return Key	定义
0x00	启动测量成功
0x01	电压不足

10.3.4 设备停止当前测量 key (0x02)

10.3.4.1 功能描述

主机通过该命令控制设备停止当前测量。

10.3.4.2 Key Value 内容

Return Key	定义
0x01	测试停止
0x02	(预留)
0x03	(预留)

10.3.5 设备停止当前测量返回 key (0x02)

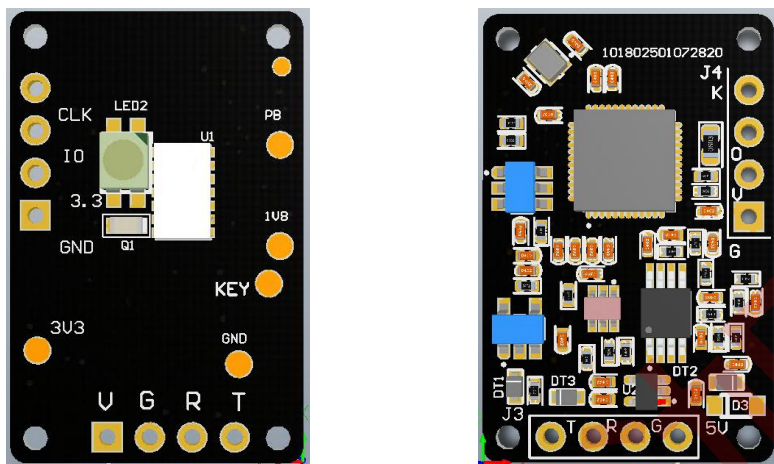
10.3.5.1 功能描述

设备通过该命令返回是否成功停止测量。

10.3.5.2 Key Value 内容

Return Key	定义
0x01	测试停止
0x02	(预留)
0x03	(预留)

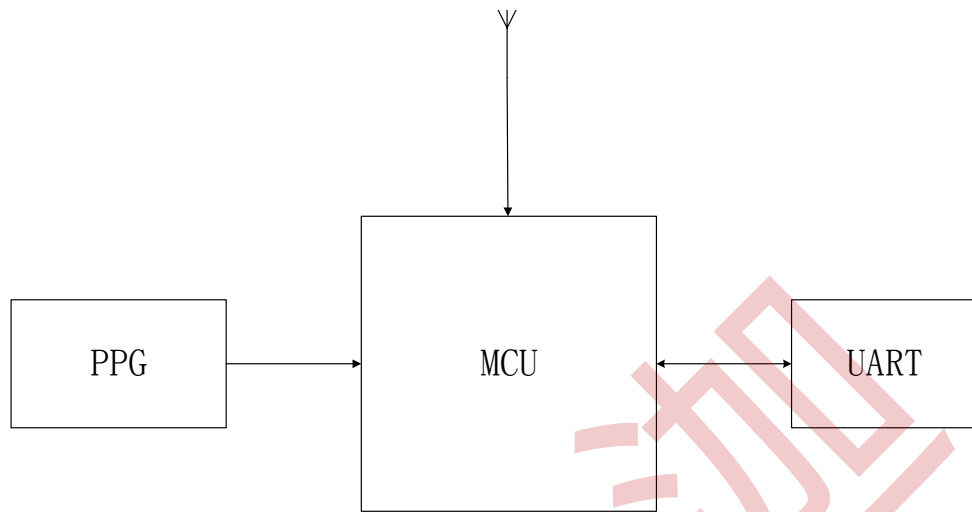
11 接口说明



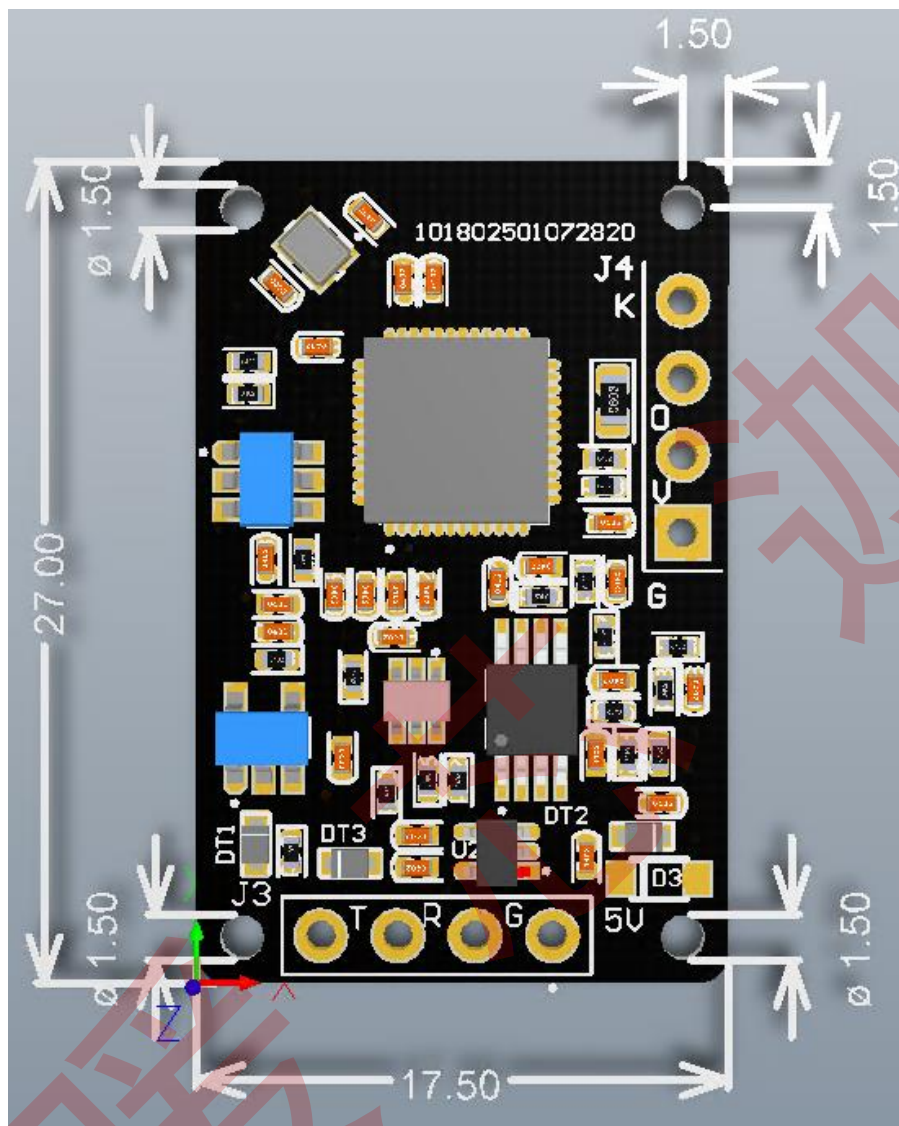
11.1 引脚说明

J3 管脚序号	信号名称	信号类型	备注
1	V	IN	模块供电管脚, 3.7V~5V 输入
2	G	IN	模块供电管脚
3	R	OUT	接外部设备的串口接收信号,3.3V 串口
4	T	IN	接外部设备的串口发送信号,3.3V 串口

12 功能框图

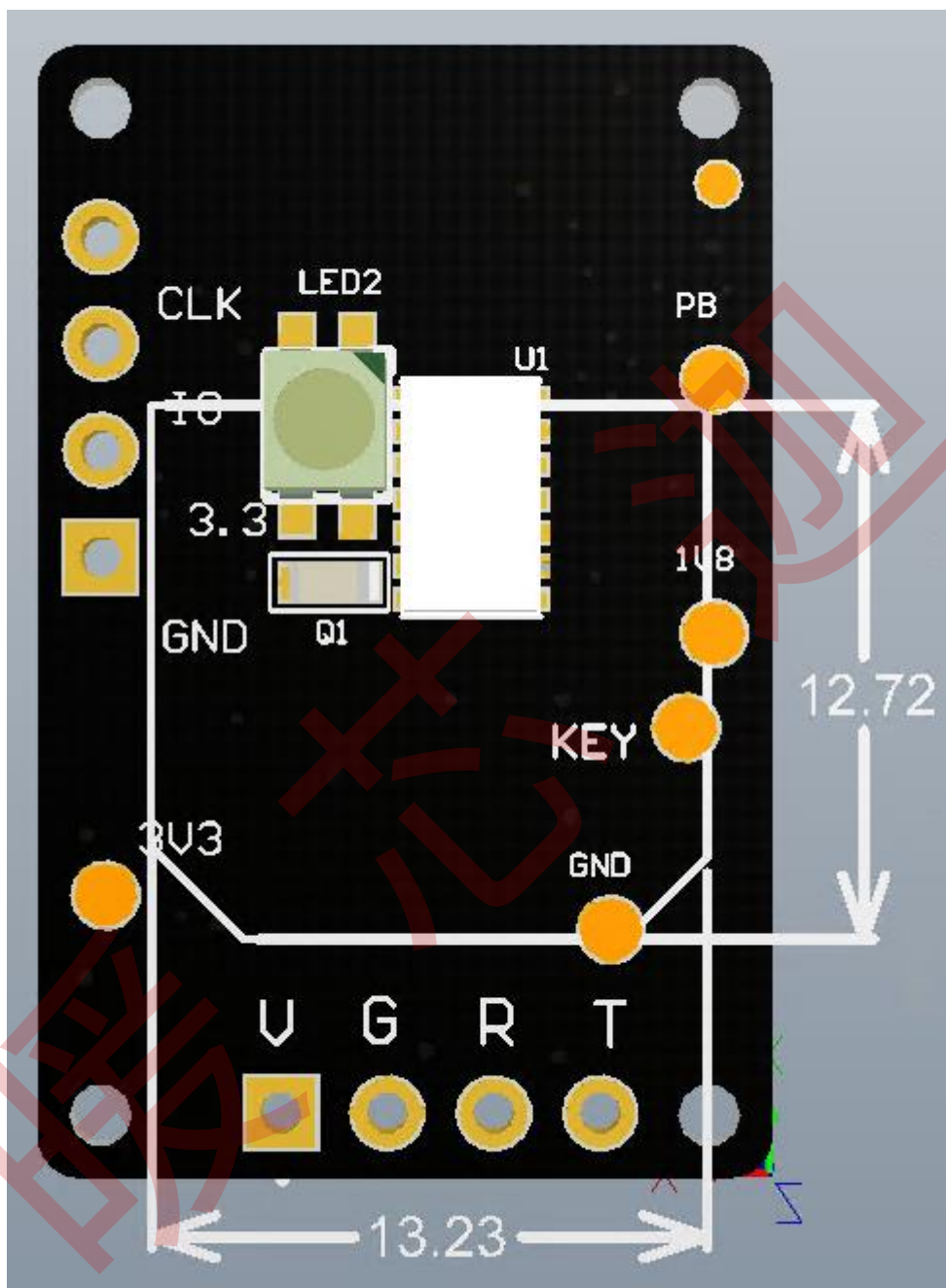


13 尺寸



具体尺寸如上图所示，单位均为毫米，模块正面器件高度 1.8mm(LED 灯)，背面器件最高 1.3mm,PCB 板厚 1mm,模块四角有直径 1.5mm 的孔方便结构设计，四个孔距板边距离为 1.5mm，其中 J4 附近为蓝牙天线，该固定孔不能使用金属类导电材质，避免影响蓝牙通讯。

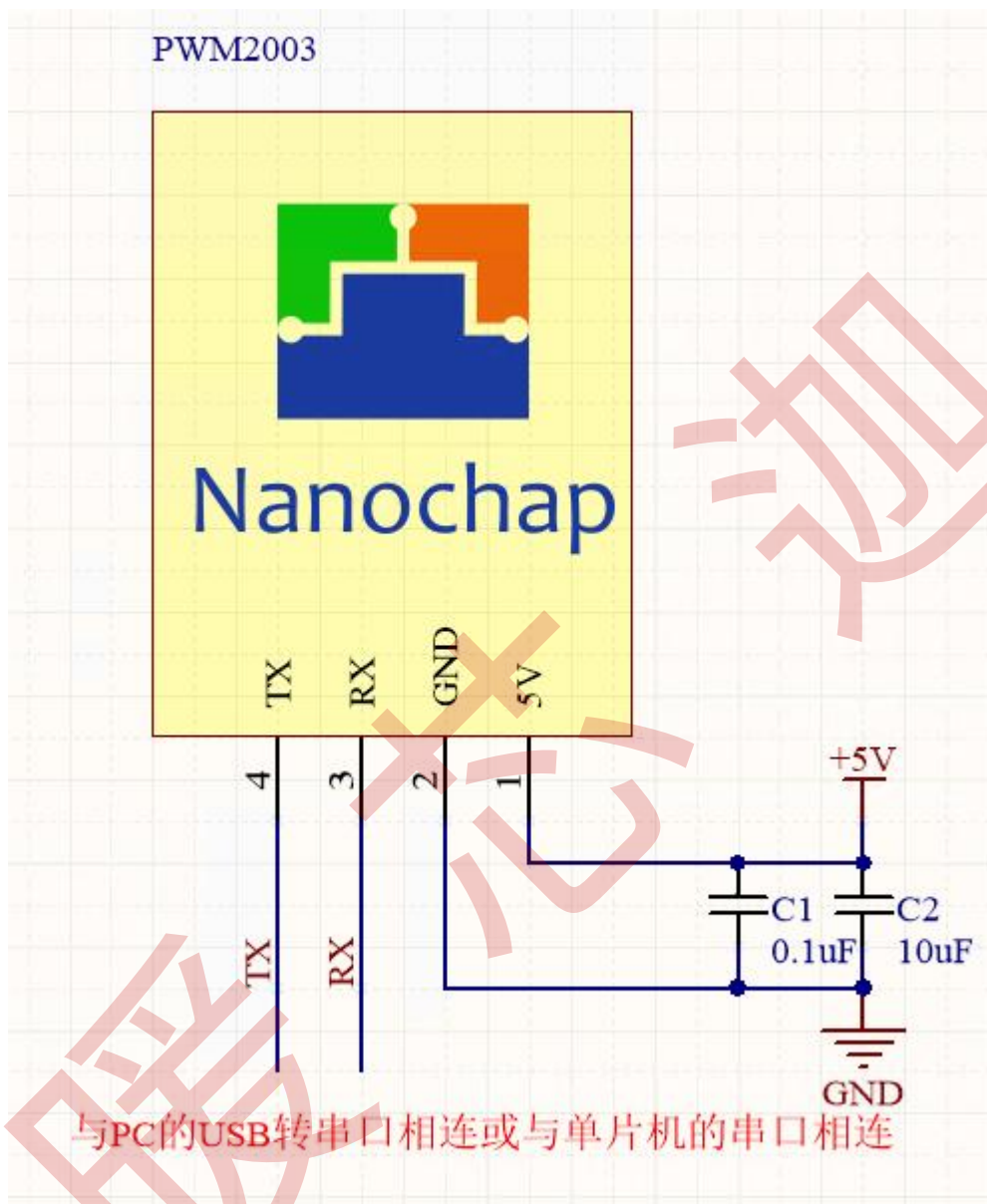
13.1 触摸区域说明



上图中白色圈为触摸区域(实物上无此白圈)，当手指与该区域感应时触摸最灵敏。

注：模块初始化上电时，手指不能接近触摸区域，初始化过程一般持续 200ms 左右，初始化结束后可正常触摸

14 典型应用与使用注意事项



典型应用原理图

由于本模块原理使用了当前最先进的生物信号测试原理，通过 PPG 信号分析人体的生理信号特征得出人体参数。需要手接触到光信号传感器才能出正确结果，如果安装了我司的 app 可见测量到的 PPG 信号。另外手指在接触光信号传感器时轻轻触碰即可，不能过紧。

15 测试 APP 和 UART 使用

测试 APP 下载请访问杭州暖芯迦电子科技有限公司官网 <http://www.nanochap.cn>。

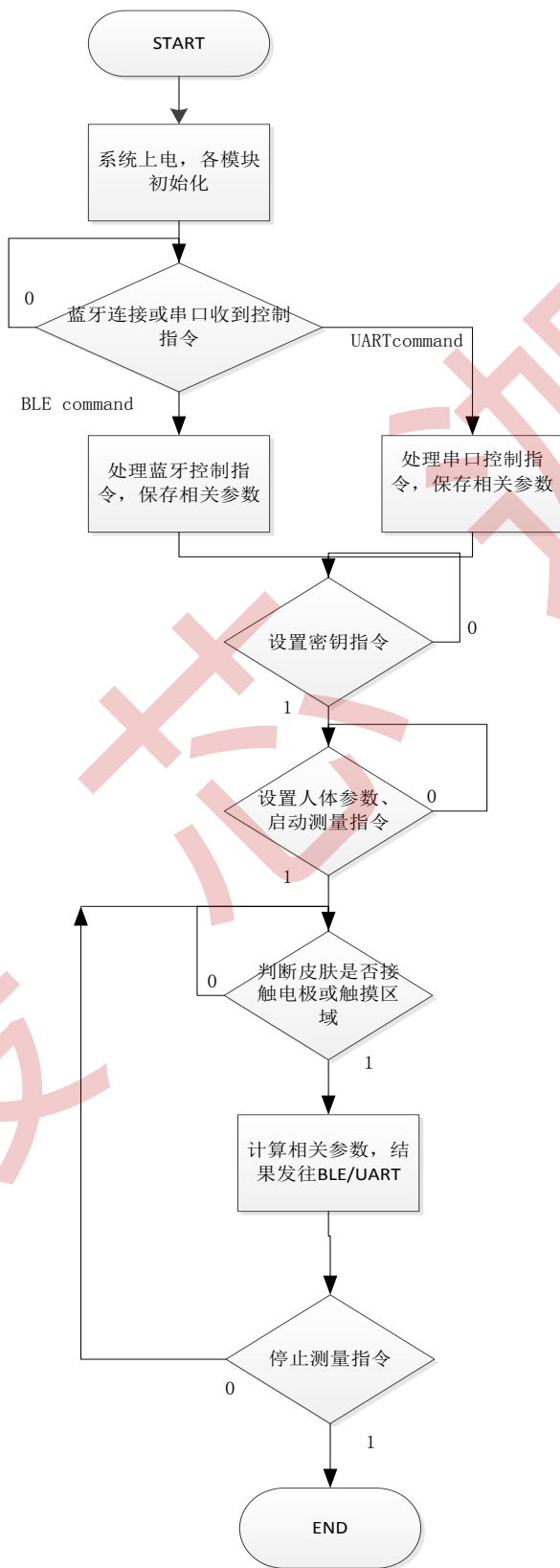
15.1 测试 APP 使用说明

客户使用原厂测试 APP 原则上仅供测试使用。如客户使用本模块配合原厂测试 APP 做产品销售于二次客户，可能存在其他客户也可连接、控制的风险。

15.2 蓝牙 SDK 和 UART 使用说明

蓝牙 SDK 和 UART 使用时，需客户根据公司或个人情况及 SDK 密钥设置规则设置加密密钥，后续数据解析必须使用此密钥解析，SDK 和 UART 数据共用一套密钥，一旦设置密钥指令生效，客户无法二次更改。模块在同一时刻只能选择使用蓝牙或串口，两者无法同时使用。如需更改密钥请联系厂家返厂重置。

16 模块控制流程图



17 加密规则及数据解密算法

17.1 加密规则

数据加密仅对数据字节内的有效结果加密，对命令字节和关键字字节均不加密。加密每次仅加密十六字节。

例：6.2.1 章节从机启动 ECG 测量返回格式中仅对数据字节 2 的内容做加密。

17.2 解密算法

用户只需将自己生成的密钥替换如下 AES128 密钥的十六字节数组值，每十六字节分别调用一次 AES 解密函数即可完成数据解密。

```
int K;                                /**< K. */
int Nb = 4;                           /**< Nb. */
int Nk;                                /**< Nk. */
int Nr;                                /**< Nr. */

uint8_t R[] = {0x02, 0x00, 0x00, 0x00}; /**< R. */

/**@brief AES128 密钥.
*/
uint8_t aes_key[] = {
    0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00,
}; //用户根据自己的实际加密密钥填入 16 字节密钥

const uint8_t s_box[256] = {           /**< 加密 S 盒. */
    // 0      1      2      3      4      5      6      7      8      9      a      b      c
    d      e      f
    0x63, 0x7c, 0x77, 0x7b, 0xf2, 0x6b, 0x6f, 0xc5, 0x30, 0x01, 0x67, 0x2b, 0xfe, 0xd7, 0xab, 0x76,
    // 0
    0xca, 0x82, 0xc9, 0x7d, 0xfa, 0x59, 0x47, 0xf0, 0xad, 0xd4, 0xa2, 0xaf, 0x9c, 0xa4, 0x72, 0xc0, //
    1
    0xb7, 0xfd, 0x93, 0x26, 0x36, 0x3f, 0xf7, 0xcc, 0x34, 0xa5, 0xe5, 0xf1, 0x71, 0xd8, 0x31, 0x15, //
    2
```

```

0x04, 0xc7, 0x23, 0xc3, 0x18, 0x96, 0x05, 0x9a, 0x07, 0x12, 0x80, 0xe2, 0xeb, 0x27, 0xb2, 0x75,
// 3
0x09, 0x83, 0x2c, 0x1a, 0x1b, 0x6e, 0x5a, 0xa0, 0x52, 0x3b, 0xd6, 0xb3, 0x29, 0xe3, 0x2f, 0x84,
// 4
0x53, 0xd1, 0x00, 0xed, 0x20, 0xfc, 0xb1, 0x5b, 0x6a, 0xcb, 0xbe, 0x39, 0x4a, 0x4c, 0x58, 0xcf,
// 5
0xd0, 0xef, 0xaa, 0xfb, 0x43, 0x4d, 0x33, 0x85, 0x45, 0xf9, 0x02, 0x7f, 0x50, 0x3c, 0x9f, 0xa8, //
6
0x51, 0xa3, 0x40, 0x8f, 0x92, 0x9d, 0x38, 0xf5, 0xbc, 0xb6, 0xda, 0x21, 0x10, 0xff, 0xf3, 0xd2, //
7
0xcd, 0x0c, 0x13, 0xec, 0x5f, 0x97, 0x44, 0x17, 0xc4, 0xa7, 0x7e, 0x3d, 0x64, 0x5d, 0x19, 0x73,
// 8
0x60, 0x81, 0x4f, 0xdc, 0x22, 0x2a, 0x90, 0x88, 0x46, 0xee, 0xb8, 0x14, 0xde, 0x5e, 0x0b, 0xdb,
// 9
0xe0, 0x32, 0x3a, 0x0a, 0x49, 0x06, 0x24, 0x5c, 0xc2, 0xd3, 0xac, 0x62, 0x91, 0x95, 0xe4, 0x79,
// a
0xe7, 0xc8, 0x37, 0x6d, 0x8d, 0xd5, 0x4e, 0xa9, 0x6c, 0x56, 0xf4, 0xea, 0x65, 0x7a, 0xae, 0x08,
// b
0xba, 0x78, 0x25, 0x2e, 0x1c, 0xa6, 0xb4, 0xc6, 0xe8, 0xdd, 0x74, 0x1f, 0x4b, 0xbd, 0x8b, 0x8a,
// c
0x70, 0x3e, 0xb5, 0x66, 0x48, 0x03, 0xf6, 0x0e, 0x61, 0x35, 0x57, 0xb9, 0x86, 0xc1, 0x1d, 0x9e,
// d
0xe1, 0xf8, 0x98, 0x11, 0x69, 0xd9, 0x8e, 0x94, 0x9b, 0x1e, 0x87, 0xe9, 0xce, 0x55, 0x28, 0xdf,
// e
0x8c, 0xa1, 0x89, 0x0d, 0xbf, 0xe6, 0x42, 0x68, 0x41, 0x99, 0x2d, 0x0f, 0xb0, 0x54, 0xbb, 0x16
// f
};

```

```

const uint8_t inv_s_box[256] = {                                     /**< 解密 S 盒.*/
// 0   1   2   3   4   5   6   7   8   9   a   b   c
d   e   f
0x52, 0x09, 0x6a, 0xd5, 0x30, 0x36, 0xa5, 0x38, 0xbf, 0x40, 0xa3, 0x9e, 0x81, 0xf3, 0xd7, 0xfb,
// 0
0x7c, 0xe3, 0x39, 0x82, 0x9b, 0x2f, 0xff, 0x87, 0x34, 0x8e, 0x43, 0x44, 0xc4, 0xde, 0xe9, 0xcb, //
1
0x54, 0x7b, 0x94, 0x32, 0xa6, 0xc2, 0x23, 0x3d, 0xee, 0x4c, 0x95, 0x0b, 0x42, 0xfa, 0xc3, 0x4e,
// 2
0x08, 0x2e, 0xa1, 0x66, 0x28, 0xd9, 0x24, 0xb2, 0x76, 0x5b, 0xa2, 0x49, 0x6d, 0x8b, 0xd1, 0x25,
// 3
0x72, 0xf8, 0xf6, 0x64, 0x86, 0x68, 0x98, 0x16, 0xd4, 0xa4, 0x5c, 0xcc, 0x5d, 0x65, 0xb6, 0x92,
// 4
0x6c, 0x70, 0x48, 0x50, 0xfd, 0xed, 0xb9, 0xda, 0x5e, 0x15, 0x46, 0x57, 0xa7, 0x8d, 0x9d, 0x84,
// 5

```

```

    0x90, 0xd8, 0xab, 0x00, 0x8c, 0xbc, 0xd3, 0x0a, 0xf7, 0xe4, 0x58, 0x05, 0xb8, 0xb3, 0x45, 0x06,
// 6
    0xd0, 0x2c, 0x1e, 0x8f, 0xca, 0x3f, 0x0f, 0x02, 0xc1, 0xaf, 0xbd, 0x03, 0x01, 0x13, 0x8a, 0x6b, //
7
    0x3a, 0x91, 0x11, 0x41, 0x4f, 0x67, 0xdc, 0xea, 0x97, 0xf2, 0xcf, 0xce, 0xf0, 0xb4, 0xe6, 0x73, //
8
    0x96, 0xac, 0x74, 0x22, 0xe7, 0xad, 0x35, 0x85, 0xe2, 0xf9, 0x37, 0xe8, 0x1c, 0x75, 0xdf, 0x6e,
// 9
    0x47, 0xf1, 0x1a, 0x71, 0x1d, 0x29, 0xc5, 0x89, 0x6f, 0xb7, 0x62, 0x0e, 0xaa, 0x18, 0xbe, 0x1b,
// a
    0xfc, 0x56, 0x3e, 0x4b, 0xc6, 0xd2, 0x79, 0x20, 0x9a, 0xdb, 0xc0, 0xfe, 0x78, 0xcd, 0x5a, 0xf4,
// b
    0x1f, 0xdd, 0xa8, 0x33, 0x88, 0x07, 0xc7, 0x31, 0xb1, 0x12, 0x10, 0x59, 0x27, 0x80, 0xec, 0x5f,
// c
    0x60, 0x51, 0x7f, 0xa9, 0x19, 0xb5, 0x4a, 0x0d, 0x2d, 0xe5, 0x7a, 0x9f, 0x93, 0xc9, 0x9c, 0xef, //
d
    0xa0, 0xe0, 0x3b, 0x4d, 0xae, 0x2a, 0xf5, 0xb0, 0xc8, 0xeb, 0xbb, 0x3c, 0x83, 0x53, 0x99, 0x61,
// e
    0x17, 0x2b, 0x04, 0x7e, 0xba, 0x77, 0xd6, 0x26, 0xe1, 0x69, 0x14, 0x63, 0x55, 0x21, 0x0c,
0x7d// f
};

//static uint8_t gadd(uint8_t a, uint8_t b)
//{
//    return a^b;
//}

//static uint8_t gsub(uint8_t a, uint8_t b)
//{
//    return a^b;
//}

/**@brief 乘法.
*/
static uint8_t gmult(uint8_t a, uint8_t b)
{
    uint8_t p = 0, i = 0, hbs = 0;

    for (i = 0; i < 8; i++) {
        if (b & 1) {
            p ^= a;
        }
    }

```



```

        hbs = a & 0x80;
        a <<= 1;
        if (hbs) a ^= 0x1b; // 0000 0001 0001 1011
        b >>= 1;
    }

    return (uint8_t)p;
}

/**@brief 加法.
 */
static void coef_add(uint8_t a[], uint8_t b[], uint8_t d[])
{
    d[0] = a[0]^b[0];
    d[1] = a[1]^b[1];
    d[2] = a[2]^b[2];
    d[3] = a[3]^b[3];
}

/**@brief 系数加法.
 */
static void coef_mult(uint8_t *a, uint8_t *b, uint8_t *d)
{
    d[0] = gmult(a[0],b[0])^gmult(a[3],b[1])^gmult(a[2],b[2])^gmult(a[1],b[3]);
    d[1] = gmult(a[1],b[0])^gmult(a[0],b[1])^gmult(a[3],b[2])^gmult(a[2],b[3]);
    d[2] = gmult(a[2],b[0])^gmult(a[1],b[1])^gmult(a[0],b[2])^gmult(a[3],b[3]);
    d[3] = gmult(a[3],b[0])^gmult(a[2],b[1])^gmult(a[1],b[2])^gmult(a[0],b[3]);
}

static uint8_t * Rcon(uint8_t i)
{
    if (i == 1) {
        R[0] = 0x01; //  $x^{1-1} = x^0 = 1$ 
    } else if (i > 1) {
        R[0] = 0x02;
        i--;
        while (i-1 > 0) {
            R[0] = gmult(R[0], 0x02);
            i--;
        }
    }
}

```

```
}

return R;
}

/**@brief 循环加.
 */
static void add_round_key(uint8_t *state, uint8_t *w, uint8_t r)
{
    uint8_t c;

    for (c = 0; c < Nb; c++) {
        state[Nb*0+c] = state[Nb*0+c]^w[4*Nb*r+Nb*c+0];
        state[Nb*1+c] = state[Nb*1+c]^w[4*Nb*r+Nb*c+1];
        state[Nb*2+c] = state[Nb*2+c]^w[4*Nb*r+Nb*c+2];
        state[Nb*3+c] = state[Nb*3+c]^w[4*Nb*r+Nb*c+3];
    }
}

/**@brief 混合列.
 */
static void mix_columns(uint8_t *state)
{
    uint8_t a[] = {0x02, 0x01, 0x01, 0x03}; //  $a(x) = \{02\} + \{01\}x + \{01\}x^2 + \{03\}x^3$ 
    uint8_t i, j, col[4], res[4];

    for (j = 0; j < Nb; j++) {
        for (i = 0; i < 4; i++) {
            col[i] = state[Nb*i+j];
        }

        coef_mult(a, col, res);

        for (i = 0; i < 4; i++) {
            state[Nb*i+j] = res[i];
        }
    }
}

/**@brief 逆向列混合.
 */
```

```
static void inv_mix_columns(uint8_t *state)
{

    uint8_t a[] = {0x0e, 0x09, 0x0d, 0x0b}; //  $a(x) = \{0e\} + \{09\}x + \{0d\}x^2 + \{0b\}x^3$ 
    uint8_t i, j, col[4], res[4];

    for (j = 0; j < Nb; j++) {
        for (i = 0; i < 4; i++) {
            col[i] = state[Nb*i+j];
        }

        coef_mult(a, col, res);

        for (i = 0; i < 4; i++) {
            state[Nb*i+j] = res[i];
        }
    }
}

/**@brief 行移.
 */
static void shift_rows(uint8_t *state)
{
    uint8_t i, k, s, tmp;

    for (i = 1; i < 4; i++) {
        // shift(1,4)=1; shift(2,4)=2; shift(3,4)=3
        // shift(r, 4) = r;
        s = 0;
        while (s < i) {
            tmp = state[Nb*i+0];

            for (k = 1; k < Nb; k++) {
                state[Nb*i+k-1] = state[Nb*i+k];
            }

            state[Nb*i+Nb-1] = tmp;
            s++;
        }
    }
}
```

```
/**@brief 逆向行移.
 */
static void inv_shift_rows(uint8_t *state)
{
    uint8_t i, k, s, tmp;

    for (i = 1; i < 4; i++) {
        s = 0;
        while (s < i) {
            tmp = state[Nb*i+Nb-1];

            for (k = Nb-1; k > 0; k--) {
                state[Nb*i+k] = state[Nb*i+k-1];
            }

            state[Nb*i+0] = tmp;
            s++;
        }
    }
}

static void sub_bytes(uint8_t *state)
{
    uint8_t i, j;
    uint8_t row, col;

    for (i = 0; i < 4; i++) {
        for (j = 0; j < Nb; j++) {
            row = (state[Nb*i+j] & 0xf0) >> 4;
            col = state[Nb*i+j] & 0x0f;
            state[Nb*i+j] = s_box[16*row+col];
        }
    }
}

static void inv_sub_bytes(uint8_t *state)
{
    uint8_t i, j;
    uint8_t row, col;
```

```
for (i = 0; i < 4; i++) {
    for (j = 0; j < Nb; j++) {
        row = (state[Nb*i+j] & 0xf0) >> 4;
        col = state[Nb*i+j] & 0x0f;
        state[Nb*i+j] = inv_s_box[16*row+col];
    }
}

static void sub_word(uint8_t *w)
{
    uint8_t i;

    for (i = 0; i < 4; i++) {
        w[i] = s_box[16*((w[i] & 0xf0) >> 4) + (w[i] & 0x0f)];
    }
}

static void rot_word(uint8_t *w)
{
    uint8_t tmp;
    uint8_t i;

    tmp = w[0];

    for (i = 0; i < 3; i++) {
        w[i] = w[i+1];
    }

    w[3] = tmp;
}

/**@brief 密钥展开.
 */
static void key_expansion(const uint8_t *key, uint8_t *w)
{
    uint8_t tmp[4];
    uint8_t i;
    uint8_t len = Nb*(Nr+1);
```

```

for (i = 0; i < Nk; i++) {
    w[4*i+0] = key[4*i+0];
    w[4*i+1] = key[4*i+1];
    w[4*i+2] = key[4*i+2];
    w[4*i+3] = key[4*i+3];
}

for (i = Nk; i < len; i++) {
    tmp[0] = w[4*(i-1)+0];
    tmp[1] = w[4*(i-1)+1];
    tmp[2] = w[4*(i-1)+2];
    tmp[3] = w[4*(i-1)+3];

    if (i%Nk == 0) {

        rot_word(tmp);
        sub_word(tmp);
        coef_add(tmp, Rcon(i/Nk), tmp);

    } else if (Nk > 6 && i%Nk == 4) {

        sub_word(tmp);

    }

    w[4*i+0] = w[4*(i-Nk)+0]^tmp[0];
    w[4*i+1] = w[4*(i-Nk)+1]^tmp[1];
    w[4*i+2] = w[4*(i-Nk)+2]^tmp[2];
    w[4*i+3] = w[4*(i-Nk)+3]^tmp[3];
}
}

/**@brief AES 解密函数. inv_cipher
 */
static void inv_cipher(uint8_t *in, uint8_t *out, uint8_t *w)
{

    uint8_t state[4*Nb];
    uint8_t r, i, j;

    for (i = 0; i < 4; i++) {

```

```

    for (j = 0; j < Nb; j++) {
        state[Nb*i+j] = in[i+4*j];
    }
}

add_round_key(state, w, Nr);

for (r = Nr-1; r >= 1; r--) {
    inv_shift_rows(state);
    inv_sub_bytes(state);
    add_round_key(state, w, r);
    inv_mix_columns(state);
}

inv_shift_rows(state);
inv_sub_bytes(state);
add_round_key(state, w, 0);

for (i = 0; i < 4; i++) {
    for (j = 0; j < Nb; j++) {
        out[i+4*j] = state[Nb*i+j];
    }
}
}

/**@brief AES 解密函数.
 */
uint32_t nanochap_AES_inv_cipher(uint8_t *in, uint8_t *out)
{
    uint8_t *w;           // expanded key
    switch (sizeof(aes_key)) {
    default:
    case 16: Nk = 4; Nr = 10; break;
    case 24: Nk = 6; Nr = 12; break;
    case 32: Nk = 8; Nr = 14; break;
    }
    w = malloc(Nb*(Nr+1)*4);
    key_expansion(aes_key, w);
    inv_cipher(in, out, w);

    free(w);

    return 0;
}

```

}



Copyright© 2020 by Hangzhou Nanochap Electronics Co.,Ltd.

使用指南中所出现的信息在出版当时相信是正确的，然而暖芯迦对于说明书的使用不负任何责任。文中提到的应用目的仅仅是用来做说明，暖芯迦不保证或表示这些没有进一步修改的应用将是适当的，也不推荐它的产品使用在会由于故障或其它原因可能会对人身造成危害的地方。暖芯迦产品不授权使用于救生、维生从机或系统中做为关键从机。暖芯迦拥有不事先通知而修改产品的权利，对于最新的信息，请参考我们的网址<http://www.nanochap.cn>或与我们直接联系。