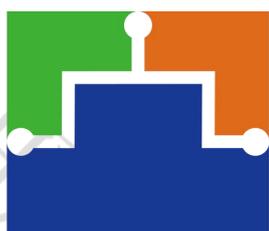


版本：V1.1



NanoChap

杭州暖芯迦电子科技有限公司

EPC1EVK-ECGPPG

生命体征检测心电图脉搏开发板_使用手册

文档修订记录

序号	版本号	修订日期	修订概述	修订人	审核人	批准人	备注
1	V1.0	2023-05-17	创建文档				
2	V1.1	2023-07-14	修改去掉 EMG/EEG				由于EEG/EMG 在同一块开发板 上使用不便，已 单独设立 EEG/EMG的开 发板型号

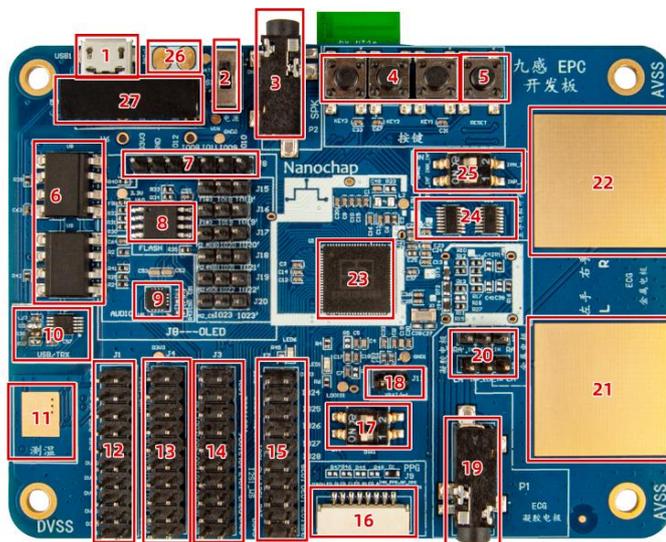
目录

文档修订记录.....	1
1. 简介.....	4
1.1 生命体征检测心电图开发板.....	4
1.2 OLED 小板.....	6
1.3 反射式灯板模块.....	6
1.4 透射式灯板模块.....	7
1.5 五导联导联线.....	7
1.6 ECG导联线.....	7
1.7 开发板通用设置.....	8
2. Hello world 实例.....	8
2.1 功能介绍.....	8
2.2 定时器功能.....	8
2.3 串口功能.....	12
3. 模拟IIC读写AT24CXX.....	14
3.1 功能介绍.....	14
3.2 测试效果.....	15
3.3 代码分析.....	15
4. 硬件IIC读写AT24CXX.....	17
4.1 功能介绍.....	17
4.2 测试效果.....	18
4.3 代码分析.....	19
5. 硬件IIC读写TMP117温度传感器.....	24
5.1 功能介绍.....	24
5.2 测试效果.....	24
5.3 代码分析.....	24
6. OLED实例.....	27
6.1 功能介绍.....	27
6.2 测试效果.....	27
6.3 代码分析.....	28
7. 硬件SPI 读取W25Q128 ID.....	30
7.1 功能介绍.....	30
7.2 测试效果.....	31
7.3 代码分析.....	31
8. PC通过板载BLE与手机APP通讯.....	34
8.1 接线方式.....	34
8.2 PC通过BLE 与手机传输数据.....	35
9. 开发板通过板载BLE与手机APP通讯.....	36
9.1 目的.....	36
9.2 准备工作.....	36
9.3 测试.....	37
9.4 代码解析.....	39

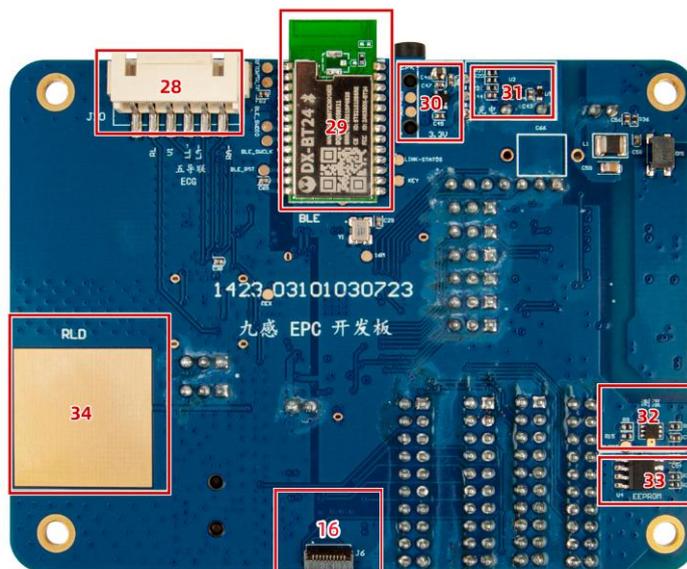
10. PPG	40
10.1 反射式硬件设置	40
10.2 反射式代码分析	43
10.3 透射式硬件设置	43
10.4 透射式代码分析	44
11. ECG 实例	45
11.1 硬件设置	45
11.2 界面及图形	46
11.3 代码分析	46
12. 综合例程使用	47
12.1 下载烧录文件	47
12.2 上位机使用	47
13. 五导联测试说明	54
13.1 目的	54
13.2 准备软硬件环境	55
13.3 下载DEMO实例	58
13.4 上位机软件	58
13.5 五导联及对应波形原理	60
13.6 硬件分析	63
13.7 导联数据上传格式	65
14. I2S 语音播放实例	67
14.1 硬件设置	67
14.2 代码分析	68
14.3 I2S 声音文件.c 生成说明	71
15. PWM实例	72
15.1 功能介绍	72
15.2 测试效果	72
15.3 代码分析	72
附录	75
1. 对应电极片位置	75
2. 正常心电图	75
16. 联系方式	76

1. 简介

1.1 生命体征检测心电图开发板



开发板正面图



开发板反面图

- 1) Micro USB接口，PCB供电和USB转串口，连接到EPC001-UART1；
- 2) 电源选择，外：接通USB，内：接通锂电池；
- 3) 音频输出接口；
- 4) 通用按键；
- 5) 复位按键；
- 6) UART1，RX，TX隔离光耦；

- 7) SPI0 接口（可接OLED或SPI转串口电路模块）；
- 8) W25Q128, 128M位串行闪存；
- 9) NAU8810YG, 音频编解码器；
- 10) CH340E, USB转串口IC；
- 11) 温度IC(TMP117)PAD；
- 12) JTAG 调试接口，连接下载调试器，进行程序的下载与调试；
- 13) IIC1_SDA, IIC1_SCL 接入短路选择（总线挂有：从设备AT24C128, TMP117, NAU8810）；
- 14) GPIO1--7, 13, 16, 17 引出排针, NAU8810YG MCLK引出管脚；
- 15) GPIO24--31; 功能选择引脚；
- 16) PPG光电传感器接入端口；
- 17) BOOTSEL: 芯片启动方式；

启动方式		复位向量 (地址)	引导模式
BOOTSEL1	BOOTSEL2		
0	1	0x20100000	选择Flash 主存作为启动区
1	0	0x80000000	选择SRAM作为启动区

- 18) Batt供电选择；
- 19) ECG 电极插孔：外接电极贴片；
- 20) 单导联输入电极选择（凝胶电极/板载金属电极）；
- 21) ECG 左手电极；
- 22) ECG 右手电极；
- 23) EPC001芯片；
- 24) 多路复用IC；
- 25) 多路复用IC 输入到EPC001 选择拨码开关；
- 26) 锂电池接入端口；
- 27) 隔离DC电源；
- 28) 5导联接入端口；
- 29) BLE透传模块；
- 30) LDO；
- 31) BATT 充电IC；
- 32) 温度IC TMP117；
- 33) AT24C128, 128K位串行EEPROM；
- 34) ECG RLD右腿驱动电极。

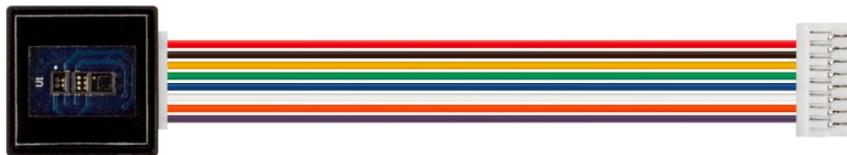
注：21，22，34为ECG 的三个电极，分别为左手电极（L）、右手电极（R）、右腿驱动电极（RLD），双手同时触摸三个电极可采集 ECG 信号。

1.2 OLED 小板



OLED 小板图

1.3 反射式灯板模块



反射式灯板模块图

1.4 透射式灯板模块



透射式灯板模块图

1.5 五导联导联线



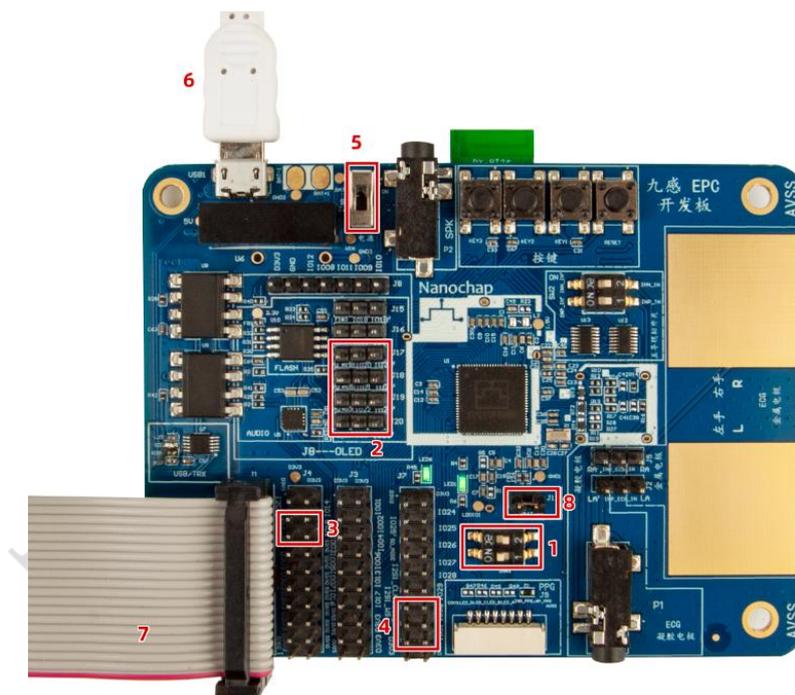
五导联导联线图

1.6 ECG导联线



ECG导联线图

1.7 开发板通用设置



使用开发板通用接线设置：位置 1 拨码如图；位置 8 短路；位置 6 接入 micro USB；位置 5 拨到如图位置，接入 USB 电源供电。

2. Hello world 实例

2.1 功能介绍

- 1) 周期打印 hello world cnt=1...
- 2) 接收串口指令, 如: ACT:sy;CMD:reset; 则产品重启。
- 3) 涉及到定时器模块, 串口模块。

定时器模块实现一个系统节拍软定时器组
串口实现日志输出和调试命令输入

2.2 定时器功能

- 1) 打开TIMER0

```
EPG_SYSCON0->FUNCEN |= (0x1<<1); //TIMER0_CLK
```

2) 初始化定时器

```
#define SYS_OSC 16000000 //系统时钟
#define TICK_1MS (SYS_OSC/1000) //1ms
```

```
Timer0MacInit(EPG_TIMER0,TICK_1MS); //定时器0 初始化
```

```

45 * 输入参数: 1: 定时器结构体
46 *           2: 定时器中断周期
47 */
48 void Timer0MacInit(EPG_TIMER_TypeDef *EPG_TIMER,uint16_t aCycl)
49 {
50     //=====
51
52
53     EPG_TIMER->RELOAD =aCycl; //把中断周期时钟数, 赋给重载寄存器 1
54     EPG_TIMER->VALUE = aCycl; //设置计时器
55
56     EPG_TIMER->INTCLEAR |= (0x1<<0); //清中断标志 2
57
58     EPG_TIMER->CTRL |= ((0x1<<3)|(0x1<<2)); //中断使能,
59     EPG_TIMER->CTRL &= ~((0x1<<2)|(0x1<<1)); //选择内部 clk
60     EPG_TIMER->CTRL |= (0x1<<0); //启动 3
61
62     //=====
63
64
65
66     ClicInstall(Timer0_Handler,TIMER0_IRQn,3); //启动中断 4
67
68 }

```

- ① 把中断周期赋给定时器寄存器;
- ② 清除中断标志, 为后面开启定时器做准备;
- ③ 开启定时器;
- ④ 启动中断 (中断处理句柄, 中断向量, 中断优先级)。

3) Timer0_Handle() //中断处理函数

```

27 static void Timer0_Handler(void) {
28     EPG_TIMER0->INTCLEAR |= (0x1<<0); //清中断标志
29     //EPG_TIMER0->INTCLEAR = 1; //清中断标志
30
31     SysTimer1msFlag = true; //1ms 标志 1
32
33     SoftTimerScan(); //软件定时组自减扫描 2
34     return;
35 }
36

```

- ① 设置 1ms 标志, 主循环调用;
- ② 扫描软定时器, 即值不为 0 的软定时器, 则自减。



4) 软件定时器

```

8 enum
9 {
10     TIMER_UART0=0,           //串口0 接收空闲倒计时
11     TIMER_UART1,           //串口1 接收空闲倒计时
12     TIMER_OLED,            //OLED
13     TIMER_IIC,             //IIC
14     TIMER_IIS,             //IIS
15
16     //添加定时器编号
17
18     TIMER_CNT_MAX,         //软定时器个数
19 };
20
21 extern uint32_t  Timing[];
22
23 #define TIMER_IS_ZERO(inx)    (Timing[inx]==0)    //定时器到
24 #define TIMER_GET_MS(inx)    Timing[inx]         //获取定时器值
25 #define TIMER_SET(inx,ms)    Timing[inx]=ms     //设置定时器值
26 #define TIMER_CHEEK(inx)    Timing[inx]         //检查定时器值

```

① 已用软定时器编号；
② 如要添加软定时器，则在这定义；
③ 软定时器个数；
④ 软定时器 API。

5) 系统节拍，会受延时程序影响

系统用节拍标志

```

13
14 typedef struct
15 {
16     bool m1ms;             //1ms 标志
17     bool m10ms;           //10ms 标志
18     bool m100ms;         //100ms 标志
19     bool m500ms;         //500ms 标志
20     bool m1s;             //1S 标志
21     bool m5s;             //5S 标志
22     bool m10s;           //10s 标志
23 }sSysTimeFalgType;

```

```

18
19= void UserMain(void)
20 {
21     static u32 cnt=0;
22     printf("main start...\n");
23     while(true)
24     {
25         SysTickPro(); //系统节拍处理 1
26         if(gSysTime.m100ms) //100ms 节拍处理
27         {
28             //添加你的功能
29         }
30
31         if(gSysTime.m500ms) //500ms 节拍处理
32         {
33             //添加你的功能
34         }
35
36         if(gSysTime.ms) //周期打印 hello world 2
37         {
38             cnt++;
39             printf("hello world cnt=%d...\n",cnt);
40         }
41
42         Uart1Pro(); //串口1接收指令解析
43     }
44 }
45

```

- ① 节拍处理;
- ② 1s 节拍到。

6) 实际效果





2.3 串口功能

1) 打开串口模块

```
EPG_SYSCON0->FUNCEN |= (0x1<<4); // UART0_CLK
```

2) 初始化串口

```
void Uart0MacInit(void)
```

```

1 void Uart0MacInit(void)
2 {
3     EPG_gpio_SetAltFunc_GPIOL16_Pin(EPG_GPIO0,UART0_RX_PIN,GPIO4_UART0_RXD); //ALT FUN 功能为 RXD
4     EPG_gpio_SetAltFunc_GPIOL16_Pin(EPG_GPIO0,UART0_TX_PIN,GPIO7_UART0_TXD); //ALT FUN功能为 TXD
5     EPG_gpio_SetFuncMode_GPIOL16_Pin(EPG_GPIO0,UART0_RX_PIN,GPIO_MODE_ALTFUN); //GPIO配置为 ALTFUN功能
6     EPG_gpio_SetFuncMode_GPIOL16_Pin(EPG_GPIO0,UART0_TX_PIN,GPIO_MODE_ALTFUN); //GPIO配置为 ALTFUN功能
7
8     EPG_UART0->FCR = EPG_UART_FCR_FIFO_EN_Msk; // 使能FIFO
9     EPG_UART0->DLL = 8; // 配置波特率
10
11     //HW8_REG(REG_FCR_ADDR) = 0x81; // rx_fifo_th 8 bytes
12     EPG_UART0->FCR = 0x81; //
13     //Enable Data ready interrupt
14     EPG_UART0->IER |= EPG_UART_IER_RDAI_EN_Msk; //使能接收中断
15
16
17
18
19     ClicInstall(UART0_Handler,UART0_IRQn,1); //启动中断
20 }

```

- ① 置配 TX, RX GPIO;
- ② 设置波特率;
- ③ 使能接收中断;
- ④ 启动中断(中断处理句柄, 中断向量, 中断优先级)。

3) 串口中断处理 UART0_Handler(void)

```

static void UART0_Handler(void)
{
    if(((EPG_UART0->IIR & EPG_UART_IIR_INT_TYPE_Msk) >> EPG_UART_IIR_INT_TYPE_Pos) == 0x3)
    {
        //Disable Line status interrupt
        EPG_UART0->IER &= ~EPG_UART_IER_RLSI_EN_Msk;
    }
    if((((EPG_UART0->IIR & EPG_UART_IIR_INT_TYPE_Msk) >> EPG_UART_IIR_INT_TYPE_Pos) == 0x2)
    || (((EPG_UART0->IIR & EPG_UART_IIR_INT_TYPE_Msk) >> EPG_UART_IIR_INT_TYPE_Pos) == 0x6))
    {
        //Disable Data Ready interrupt
        //EPG_UART0->IER &= ~EPG_UART_IER_RDAI_EN_Msk;
    }

    URxEvent = true; //中断接收事件

    //接收数据入队
    array[tInx]=EPG_UART0->RBR;
    tInx++;
    if(UART0_BUF_LEN<=tInx) tInx= UART0_BUF_LEN-1;

    TIMER_SET(TIMER_UART0,10); //重置接收计时器
}

```

- ① 清除中断标志;
- ② 设置接收事件标志;
- ③ 接收到的数据插入接收队列;
- ④ 重置接收空闲时间。

4) 接收处理, 在主循环中调用, DEMO暂不考虑实时

```

94 //接收处理程序, 在主循环中调用
95 void Uart0Pro(void)
96 {
97     if((URxEvent==true) //产生接收事件, 防止没有接收事件时重复进入
98         &&(TIMER_IS_ZERO(TIMER_UART0)) //接收倒计数, 一组数据接收结束
99     )
100     {
101         printf("uart rec over  %s  %d \n",array ,strlen(array)); //打印接收到的数据
102         CmdCheck(array); //数据处理;
103
104         URxEvent = false; //清 接收事件标志
105         Uart0BufInit(); //清 接收缓存
106     }
107 }
108
109

```

- ① 判断有接收事件并接收结束;
- ② 处理接收数据;
- ③ 清空接收缓存。

5) 测试效果

实例对应工程文件: ../project-hello_world



The screenshot shows a serial terminal window with the following content:

Received data (left pane):

```

[10:55:07.688]收←◆hello world cnt=154...
[10:55:08.689]收←◆hello world cnt=155...
[10:55:09.689]收←◆hello world cnt=156...
[10:55:10.692]收←◆hello world cnt=157...
[10:55:11.692]收←◆hello world cnt=158...
[10:55:12.693]收←◆hello world cnt=159...
[10:55:13.695]收←◆hello world cnt=160...
[10:55:14.695]收←◆hello world cnt=161...
[10:55:15.696]收←◆hello world cnt=162...
[10:55:15.927]发→◇ACT:sy;CMD:reset;
[10:55:15.944]收←◆uart rec over  ACT:sy;CMD:reset;
ACT
sy
CMD
reset
sys reset...
main start...
[10:55:16.966]收←◆hello world cnt=1...
[10:55:17.967]收←◆hello world cnt=2...

```

Multi-line character sending list (right pane):

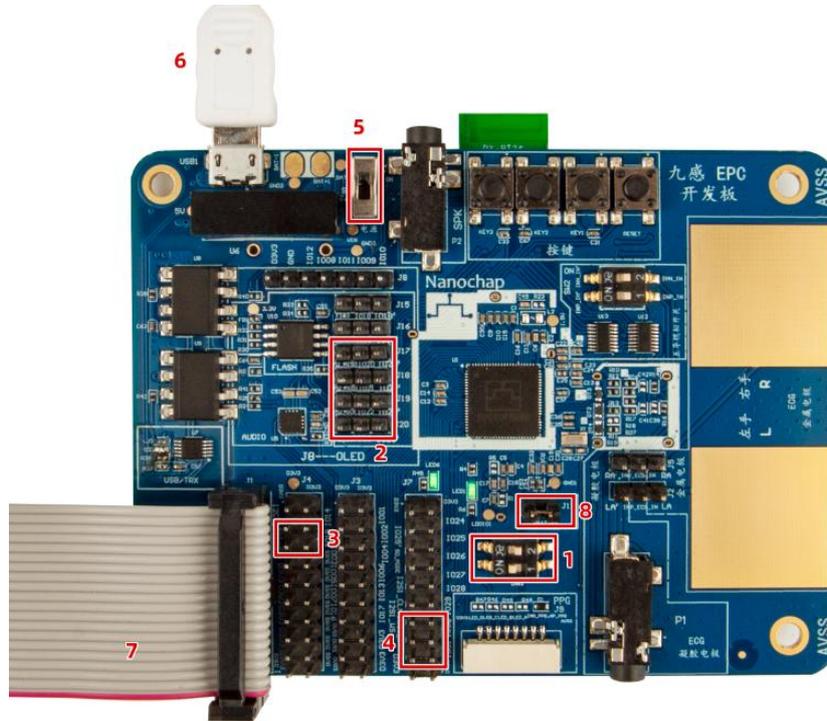
字符串	字符数	速度	次数
欢迎您使用SSCOM!	欢迎语	2	1000
1401150d	4无注释	0	1000
1400140d	5无注释	0	1000
1201130d	6无注释	0	1000
1101120d	7无注释	0	1000
1800180d	8无注释	0	1000
	9无注释	0	1000
	10无注释	0	1000
	11无注释	0	1000
	12无注释	0	1000
	13无注释	0	1000
ACT:sy;CMD:reset;	14无注释	0	1000
	15无注释	0	1000
	16无注释	0	1000
	17无注释	0	1000
	18无注释	0	1000
	19无注释	0	1000
	20无注释	0	1000
	21无注释	0	1000
	22无注释	0	1000

At the bottom, the terminal settings are visible:

- 端口号: COM4 USB-SERIAL CH340
- 波特率: 115200
- 加时间戳和分包显示: 勾选
- 超时时间: 20 ms
- 第 1 字节 至 末尾
- 加校验: None

3. 模拟 IIC 读写 AT24CXX

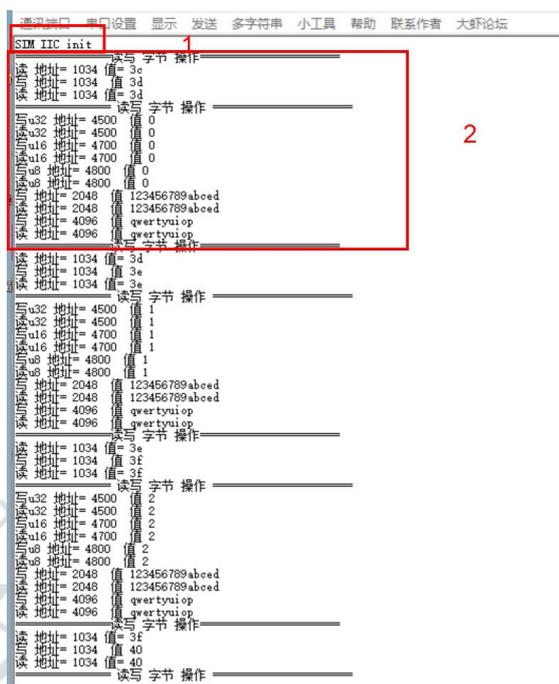
在通用设置基础上，位置 3 短路。



3.1 功能介绍

实现对 AT24CXX u8, u16, u32 字符串指定位置读写。

3.2 测试效果



- 1) 模拟IIC实始化
- 2) 读写AT24CX 测试日志输出

3.3 代码分析

实例对应工程文件: ../project-IIC_SIM_AT24C128

1) Iic1GpioInit()

```

75
76 //IIC GPIO 初始化
77 static void Iic1GpioInit(void)
78 {
79
80 EPG_gpio_ClrFuncMode_GPIO16_Pin(EPG_GPIO0, SDA_PIN, GPIO_MODE_MSK); //清空GPIO 模式配置位
81 EPG_gpio_ClrFuncMode_GPIO16_Pin(EPG_GPIO0, SCL_PIN, GPIO_MODE_MSK);
82
83 EPG_gpio_ClrAltFunc_GPIO16_Pin(EPG_GPIO0, SDA_PIN, GPIO_MODE_MSK); //清空GPIO 功能配置位
84 EPG_gpio_ClrAltFunc_GPIO16_Pin(EPG_GPIO0, SCL_PIN, GPIO_MODE_MSK);
85
86
87 EPG_gpio_SetFuncMode_GPIO16(EPG_GPIO0, 0x00000000); // 1
88 EPG_gpio_SetAltFunc_GPIO16(EPG_GPIO0, 0x00000000);
89
90 EPG_gpio_SetFuncMode_GPIO16_Pin(EPG_GPIO0, SCL_PIN, GPIO_MODE_OUTPUT); // 配置GPIO 为输出
91
92 //清下拉，置上拉
93 EPG_gpio_ClrPull_Down_Pin(EPG_GPIO0,SDA_PIN); //clear pulldown gpio14
94 EPG_gpio_SetPull_Up_Pin(EPG_GPIO0,SDA_PIN); //set pullup gpio14
95 EPG_gpio_ClrPull_Down_Pin(EPG_GPIO0,SCL_PIN); //clear pulldown gpio15
96 EPG_gpio_SetPull_Up_Pin(EPG_GPIO0,SCL_PIN); //set pullup gpio15 // 2
97
98 }
99
100

```

- ① 配置 GPIO 为普通功能;
- ② 设置 GPIO 上拉。

2) EpcIicInit()

```

153
154 //模拟IIC初始化
155 IicMacApiType * EpcIicInit(void)
156 {
157     static bool initflag=false;           //IIC总线只初始化一次，
158
159     if(initflag == true)
160     {
161         return &sgI2cApi;                1
162     }
163     //==
164     printf("SIM IIC init\n");
165
166
167     Iic1GpioInit();                       //GPIO 初始化                2
168
169     //模拟IIC SCL SDA 设置
170     memset((void*)&gSimIic,0,sizeof(sSimIicType));
171     gSimIic.SDA_IN=simSdaIn;
172     gSimIic.SDA_OUT=simSdaOut;
173     gSimIic.Clr_IIC_SCL=simClrScl;
174     gSimIic.Clr_IIC_SDA=simClrSda;
175     gSimIic.READ_SDA=simReadSda;
176     gSimIic.Set_IIC_SCL=simSetScl;
177     gSimIic.Set_IIC_SDA=simSetSda;
178
179
180
181
182     memset((void*)&sgI2cApi,0,sizeof(IicMacApiType));
183     //初始化模拟IIC 总线
184     simIicMacInit(&gSimIic,1,&sgI2cApi);  4
185
186
187     initflag = true;
188     return &sgI2cApi;
189 }
190

```

- ① 总线只初始化一次，如果在已初始化之后再次调用，则只返回总线 API；
- ② GPIO 初始化；
- ③ 模拟 IIC SCL SDA 设置，底层模拟时序时会调用；
- ④ 初始化模拟 IIC 总线。

3) simIicMacInit()

```

..
48 //=====
49 //模拟IIC 总线初始化
50 void simIicMacInit(sSimIicType *aSimIic,u16 aDelay,IicMacApiType *pApi)
51 {
52
53     sDelayTime =aDelay;                   //速度参数设置，软件延长时间        1
54     //sDeviceAdd =DeviceAdd;
55     pSimIic = aSimIic;
56
57     //总线API 回调设置
58
59     pApi->mAdd16_RBuf =simI2C_ReadBuffer;
60     pApi->mAdd16_RByte =simI2C_ReadByte;
61     pApi->mAdd16_WBuf =simI2C_WriteBuffer;
62     pApi->mAdd16_WByte =simI2C_WriteByte;
63
64
65     pApi->mAdd8_WByte =simI2C_Add8_WriteByte; //寄存器地址为8字节
66     pApi->mAdd8_RByte =simI2C_Add8_ReadByte; //寄存器地址为8字节
67
68     pApi->mAdd8_WBuf =simI2C_Add8_WriteBuffer; //寄存器地址为8字节
69     pApi->mAdd8_RBuf =simI2C_Add8_ReadBuffer; //寄存器地址为8字节
70 }
71

```

- ① 模拟 IIC 速度设置，主要影响模拟时序时，电平持续时间；
- ② 总线 API 回调设置。

4) At24cXXTest()

```

-----
//=====

tregadd=0x1034;

printf("=====读写 字节 操作===== \n");

result=At24c128_Handler->mAdd16_RByte(DEVICE_ADD_AT24C128,tregadd,&tmp);
printf("读 地址= %x 值= %x \n",tregadd,tmp); 1

data = tmp+1;

result =At24c128_Handler->mAdd16_WByte(DEVICE_ADD_AT24C128,tregadd,data);
printf("写 地址= %x 值 %x \n",tregadd,data); 2

tmp =0;
result=At24c128_Handler->mAdd16_RByte(DEVICE_ADD_AT24C128,tregadd,&tmp);
printf("读 地址= %x 值= %x \n",tregadd,tmp); 3

printf("===== 读写 字节 操作 ===== \n");

At24cxxSaveU32(0x4500,inc); //读写U32
tinc=At24cxxReadU32(0x4500); 4
inc = tinc+1;

At24cxxSaveU16(0x4700,inc1); //读写u16
tinc1=At24cxxReadU16(0x4700); 5
inc1 = tinc1+1;

At24cxxSaveU8(0x4800,inc2); //读写u8
tinc2=At24cxxReadU8(0x4800); 6
inc2 = tinc2+1;

//=====

At24CxTest1(); //读写字符串测试
At24CxxStrTest(); //读写字符串测试 7

```

- ① 读指定地址(16 位地址格式)1 字节数;
- ② 写指定地址(16 位地址格式)1 字节数;
- ③ 读指定地址(16 位地址格式)1 字节数，比较二次读出不一样，则是有变化;
- ④ 读写 u32 类型数据;
- ⑤ 读写 u16 类型数据;
- ⑥ 读写 u8 类型数据;
- ⑦ 读写 字符串。

4. 硬件 IIC 读写 AT24CXX

4.1 功能介绍

接图方式与模拟 IIC 读写 AT24CXX
 实现对 AT24CXX u8, u16, u32 字符串指定位置读写

4.2 测试效果

```
HW IIC init
read aa 9 0, 3100
write aa 9 0, 7144
write aa 9 0, 7144
read aa 9 0, 3100
writebuffer 0 0 0, 7144
readbuffer 0 0 0, 3100 123456789abcd
At24cxxSaveStr 0, 7144
At24cxxReadStr 0, 3100
tstr qwertyuiop
At24cxxSaveU32 0, 7144
At24cxxReadU32 0, 3100
At24cxxSaveU16 0, 7144
At24cxxReadU16 0, 3100
At24cxxSaveU8 0, 7144
At24cxxReadU8 0, 3100
read32 1 read16 1 read8 1
read aa f 0, 3100
write aa f 0, 7144
write aa f 0, 7144
read aa f 0, 3100
writebuffer 0 0 0, 7144
readbuffer 0 0 0, 3100 123456789abcd
At24cxxSaveStr 0, 7144
At24cxxReadStr 0, 3100
tstr qwertyuiop
At24cxxSaveU32 0, 7144
At24cxxReadU32 0, 3100
At24cxxSaveU16 0, 7144
At24cxxReadU16 0, 3100
At24cxxSaveU8 0, 7144
At24cxxReadU8 0, 3100
read32 2 read16 2 read8 2
read aa e8 0, 3100
write aa e8 0, 7144
write aa e8 0, 7144
read aa e8 0, 3100
writebuffer 0 0 0, 7144
readbuffer 0 0 0, 3100 123456789abcd
At24cxxSaveStr 0, 7144
At24cxxReadStr 0, 3100
tstr qwertyuiop
At24cxxSaveU32 0, 7144
At24cxxReadU32 0, 3100
At24cxxSaveU16 0, 7144
At24cxxReadU16 0, 3100
At24cxxSaveU8 0, 7144
At24cxxReadU8 0, 3100
read32 3 read16 3 read8 3
```

- 1) 模拟 IIC 实初始化
- 2) 读写 AT24CXX 测试日志输出

4.3 代码分析

实例对应工程文件：../project-IIC_HW_AT24C128
打开 HW_IIC 宏，使能硬件 IIC

```

simTxMac.c  hello.c  UserSys.c  EpcMac.c  At24c128test.c
L0 #include "../common/fir.h"
L1 #include <stdint.h>
L2 #include <string.h>
L3 #include "../common/myiomap.h"
L4
L5 #include "UserSys.h"
L6
L7 static IicMacApiType sgI2cApi;
L8
L9 #define HW_IIC //如果使用硬件IIC 1
L10
L11 #define SDA_PIN 14
L12 #define SCL_PIN 15
L13
L14
L15 #ifndef HW_IIC
L16
L17
L18 //=====
L19 //IIC GPIO 初始化
L20 static void Iic1GpioInit(void)
L21

```

1) Iic1GpioInit()

```

//=====
//IIC GPIO 初始化
static void Iic1GpioInit(void)
{
    EPG_gpio_ClrFuncMode_GPIOL16_Pin(EPG_GPIO0, SDA_PIN, GPIO_MODE_MSK); //清空GPIO 模式配置位
    EPG_gpio_ClrFuncMode_GPIOL16_Pin(EPG_GPIO0, SCL_PIN, GPIO_MODE_MSK);

    EPG_gpio_ClrAltFunc_GPIOH16_Pin(EPG_GPIO0, SDA_PIN, GPIO_MODE_MSK); //清空GPIO 功能配置位
    EPG_gpio_ClrAltFunc_GPIOH16_Pin(EPG_GPIO0, SCL_PIN, GPIO_MODE_MSK);

    EPG_gpio_SetFuncMode_GPIOL16(EPG_GPIO0, 0xA0000000); //配置为ALT FUN 功能
    EPG_gpio_SetAltFunc_GPIOL16(EPG_GPIO0, 0x50000000); //ALT FUN 功能为 IIC 1

    //清下位，置上拉
    EPG_gpio_ClrPull_Down_Pin(EPG_GPIO0,SDA_PIN); //clear pulldown gpio14
    EPG_gpio_SetPull_Up_Pin(EPG_GPIO0,SDA_PIN); //set pullup gpio14
    EPG_gpio_ClrPull_Down_Pin(EPG_GPIO0,SCL_PIN); //clear pulldown gpio15
    EPG_gpio_SetPull_Up_Pin(EPG_GPIO0,SCL_PIN); //set pullup gpio15 2
}

```

- ① 配置 GPIO 为普通功能；
- ② 设置 GPIO 上拉。

2) EpcIicInit()

```

// IIC 初始化
IicMacApiType * EpcIicInit(void)
{
    static bool initflag=false;          //IIC总线只初始化一次，因为同一个IIC总线会挂载多个外设，防止重复初始化
    if(initflag == true)                //已初始化，
    {
        return &sgI2cApi;                1
    }
    //==
    printf("HW IIC init\n");
    Iic1GpioInit();                      //GPIO 初始化 2
    //总线初始化
    memset((void*)&sgI2cApi,0,sizeof(IicMacApiType));
    IicMacInit(EPG_I2C1,9,&sgI2cApi);    3
}

    initflag = true;
    return &sgI2cApi;                    //返回总线API接口 4
}

```

- ① 如果总线已初始化，则直接返回总线 API 接口；
- ② GPIO 初始化；
- ③ 初始化总线(硬件 IIC 结构体，IIC 速度，返回的总线 API)；
- ④ 返回总线 API 接口。

3) simIicMacInit()

```

627 */
628 void IicMacInit(EPG_I2C_TypeDef *EPG_I2C,u16 ClockDiv,IicMacApiType *pApi)
629 {
630
631     u8 tDiv =(u8)(ClockDiv&0x3f);
632
633     //复位IIC 总线
634     EPG_I2C->I2C_CR1 |=EPG_I2C_CR1_SWRST_Msk;
635     //TIMER_SET(TIMER_IIC,100); //delay_ms(200);
636     //while(TIMER_GET_MS(TIMER_IIC)!=0);
637     EPG_I2C->I2C_CR1 &=~EPG_I2C_CR1_SWRST_Msk;
638
639
640
641     //设置硬件IIC 速度
642     EPG_I2C->I2C_CR2 = (tDiv << EPG_I2C_CR2_FREQDIV_Pos);
643     //EPG_I2C->I2C_CR1 = (EPG_I2C_CR1_PE_Msk | EPG_I2C_CR1_DBYPASS_Msk);
644     EPG_I2C->I2C_CR1 = EPG_I2C_CR1_PE_Msk; //使能外设
645
646
647     sEPG_I2C = EPG_I2C; //
648
649
650     //IIC API 回调设置
651
652     pApi->mAdd16_RBuf =I2CRBuf; //16字节地址寄存器 读数组
653     pApi->mAdd16_RByte =I2CRByte; //16字节地址寄存器 读字节
654     pApi->mAdd16_WBuf =I2CWBuf; //16字节地址寄存器 写数组
655     pApi->mAdd16_WByte =I2CWByte; //16字节地址寄存器 写字节
656
657
658
659     pApi->mAdd8_RBuf =I2CAdd8RBuf; //8字节地址寄存器 读数组
660     pApi->mAdd8_RByte =I2CAdd8RByte; //8字节地址寄存器 读字节
661     pApi->mAdd8_WBuf =I2CAdd8WBuf; //8字节地址寄存器 写数组
662     pApi->mAdd8_WByte =I2CAdd8WByte; //8字节地址寄存器 写字节
663 }
664
665

```

- ① 硬件 IIC 波特率参数合法处理;
- ② 硬件 IIC 复位;
- ③ 硬件 IIC 波特率设置, 并使能外设;
- ④ IIC 总线 API 设置, 上层使用 IIC 时, 调用。

4) At24cXXTest()

```

23
24 //AT24CXX注册
25 /*
26 设备注册到总线, 取的总线API
27 */
28 void At24cxxRegister(void)
29 {
30     At24c128_Handler=NULL; //
31     At24c128_Handler=EpcIicInit(); //取的总线API
32
33 }
34

```

- ① 把 AT24CXX 注册到总线 IIC 总线 API;

```

...
45 //=====
46 //向AT24CXX 指定位置写入一个32位的值
47 //输入参数: addr:写入的位置
48 //      val:要写入的值
49 void At24cxxSaveU32(u16 addr,u32 val)
...
7 //向AT24CXX 指定位置读出一个32位的值
8 //输入参数: addr:写入的位置
9 //输出参数: 读出的值
0 u32 At24cxxReadU32(u16 addr)
...
1 //=====
12 //向AT24CXX 指定位置写入一个16位的值
13 //输入参数: addr:写入的位置
14 //      val:要写入的值
15 void At24cxxSaveU16(u16 addr,u16 val)
...
83 //向AT24CXX 指定位置读出一个16位的值
84 //输入参数: addr:写入的位置
85 //输出参数: 读出的值
86 u16 At24cxxReadU16(u16 addr)
...
98 //=====
99 //向AT24CXX 指定位置写入一个8位的值
100 //输入参数: addr:写入的位置
101 //      val:要写入的值
102 void At24cxxSaveU8(u16 addr,u8 val)
...
109 //向AT24CXX 指定位置读出一个8位的值
110 //输入参数: addr:写入的位置
111 //输出参数: 读出的值
112 u8 At24cxxReadU8(u16 addr)
L23 //=====
L24 //向AT24CXX 指定位置写入一个字符串
L25 //输入参数: addr:写入的位置
L26 //      str:要写入的字符串
L27 void At24cxxSaveStr(u16 addr,char *str)
4 //从AT24CXX 指定位置读了一个字符串,
5 //输入参数: addr:写入的位置, str:读出的字符串, alen:要读字符串长度
6 void At24cxxReadStr(u16 addr,char *str,u16 alen)

```

- ② 实现 AT24CXX 操作函数
void At24cxxTest(void)。

```

-----,
//=====

tregadd=0x1034;

printf("=====读写 字节 操作===== \n");

result=At24c128_Handler->mAdd16_RByte(DEVICE_ADD_AT24C128,tregadd,&tmp);
printf("读 地址= %x 值= %x \n",tregadd,tmp); 1

data = tmp+1;

result =At24c128_Handler->mAdd16_WByte(DEVICE_ADD_AT24C128,tregadd,data);
printf("写 地址= %x 值 %x \n",tregadd,data); 2

tmp =0;
result=At24c128_Handler->mAdd16_RByte(DEVICE_ADD_AT24C128,tregadd,&tmp);
printf("读 地址= %x 值= %x \n",tregadd,tmp); 3

printf("===== 读写 字节 操作 ===== \n");

At24c128_SaveU32(0x4500,inc); //读写U32
tinc=At24c128_ReadU32(0x4500); 4
inc = tinc+1;

At24c128_SaveU16(0x4700,inc1); //读写u16
tinc1=At24c128_ReadU16(0x4700); 5
inc1 = tinc1+1;

At24c128_SaveU8(0x4800,inc2); //读写u8
tinc2=At24c128_ReadU8(0x4800); 6
inc2 = tinc2+1;

//=====

At24CxTest1(); //读写字符串测试
At24CxStrTest(); //读写字符串测试 7

```

- ① 读指定地址(16 位地址格式)1 字节数;
- ② 写指定地址(16 位地址格式)1 字节数;
- ③ 读指定地址(16 位地址格式)1 字节数,比较二次读出不一样, 则是有变化;
- ④ 读写 u32 类型数据;
- ⑤ 读写 u16 类型数据;
- ⑥ 读写 u8 类型数据;
- ⑦ 读写字符串。

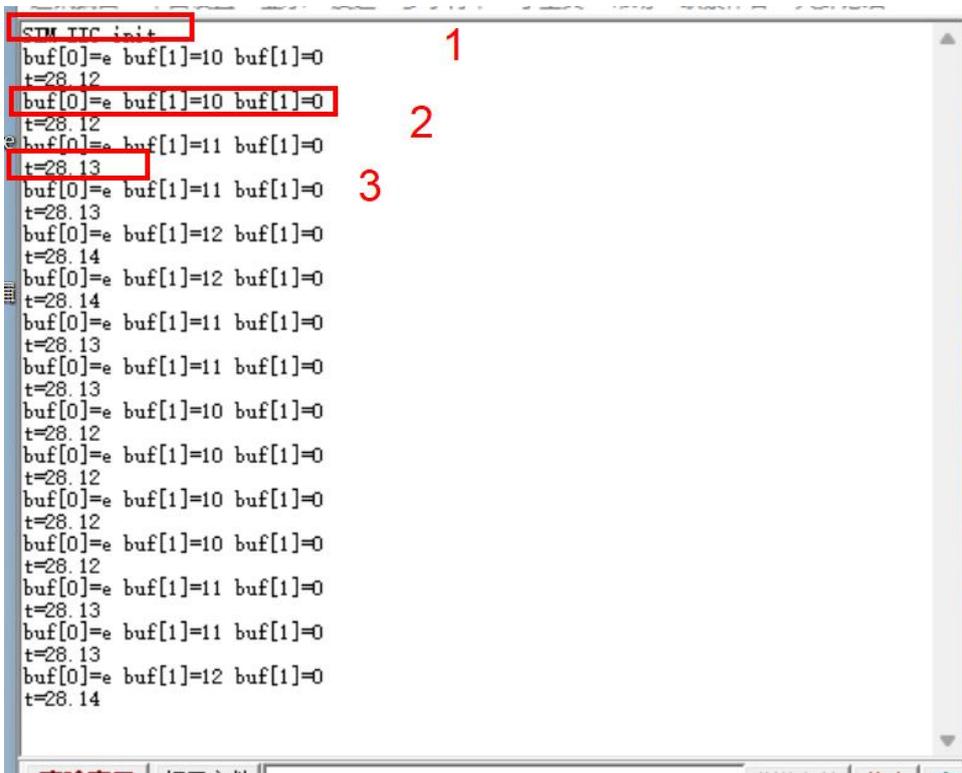
5. 硬件 IIC 读写 TMP117 温度传感器

5.1 功能介绍

实现对 AT24CXX u8, u16, u32 字符串指定位置读写。

注：接线方式如：模拟 IIC 读写 AT24CXX

5.2 测试效果



```

STM IIC init
buf[0]=e buf[1]=10 buf[1]=0
t=28.12
buf[0]=e buf[1]=10 buf[1]=0
t=28.12
buf[0]=e buf[1]=11 buf[1]=0
t=28.13
buf[0]=e buf[1]=11 buf[1]=0
t=28.13
buf[0]=e buf[1]=12 buf[1]=0
t=28.14
buf[0]=e buf[1]=12 buf[1]=0
t=28.14
buf[0]=e buf[1]=11 buf[1]=0
t=28.13
buf[0]=e buf[1]=11 buf[1]=0
t=28.13
buf[0]=e buf[1]=10 buf[1]=0
t=28.12
buf[0]=e buf[1]=10 buf[1]=0
t=28.12
buf[0]=e buf[1]=10 buf[1]=0
t=28.12
buf[0]=e buf[1]=11 buf[1]=0
t=28.13
buf[0]=e buf[1]=11 buf[1]=0
t=28.13
buf[0]=e buf[1]=12 buf[1]=0
t=28.14

```

- 1) 使用模拟串口
- 2) 从 TMP117 读出温度的 AD 值
- 3) 根据 AD 值计算出的温度

5.3 代码分析

实例对应工程文件： ../project-IIC_TMP117



1) 初始化

```
93
94 void UserInit(void)
95 {
96     Timer0MacInit(EPG_TIMER0, TICK_1MS);           //定时器0 初始化
97     Uart1MacInit(UART_BPS_115200);                 //初始化串口1
98     Tmp117Register();                               //注册 TMP117
99 }
100
101
102
```

注：注册 TMP117，因为同一个 IIC 总线上可以注册多个从设备。

```
44
45 //TMP117 注册到总线
46 void Tmp117Register(void)
47 {
48     TMP117_Handler=NULL;
49     TMP117_Handler=EpcIicInit();                   //取的IIC总线操作API 1
50     TMP117_Initialization_DEFAULT(TMP117_Handler); //实始化TMP117 寄存器 2
51 }
52
53
54
```

- ① 注册 Tmp11，取的 IIC 总线操作 API;
- ② 初始化 TMP117 寄存器。

2) 主循环，周期读取温度，并打印

```

void UserMain(void)
{
    while(true)
    {
        SysTickPro();          //系统节拍处理

        if(gSysTime.m500ms) //检测
        {
            Tmp117Test();      //TMP117 测试
        }
        if(gSysTime.m100ms) //检测
        {
        }
    }
}
    
```

3) TMP117 底层读写操作

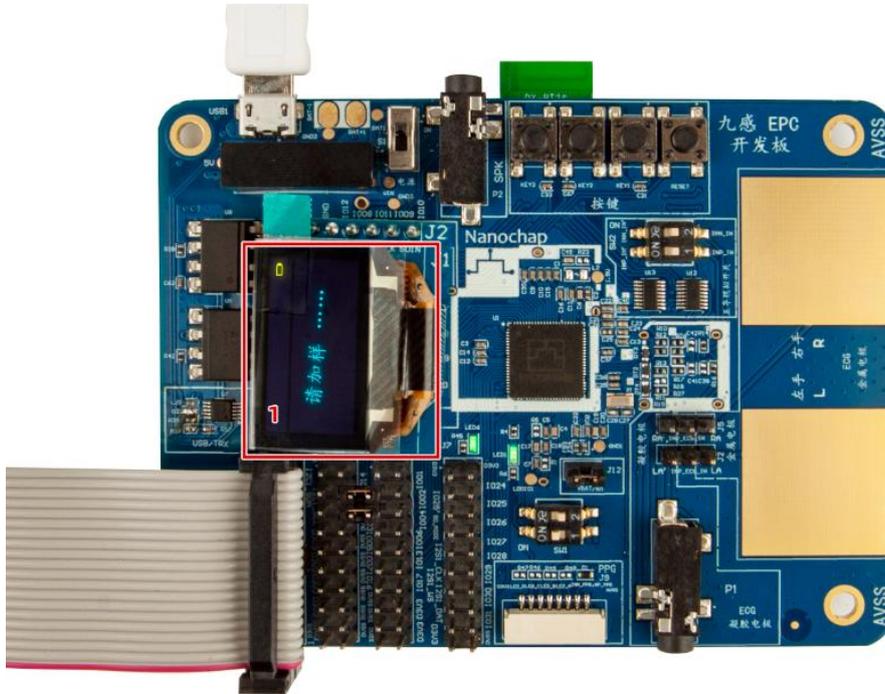
```

> / r
38
39 u16 TMP117_get_Configuration(IicMacApiType *p)
40 {
41     static uint8_t buf[3];
42     memset((void*)buf,0,3);
43
44     p->mAdd8_RBuf(TMP117_DeviceID,TMP117_ConfigurationRegister,buf,2); 1
45
46     return ((buf[0]<<8)|buf[1]);
47 }
48
49 void TMP117_set_Configuration (IicMacApiType *p,uint8_t first,uint8_t second)
50 {
51     static uint8_t buf[3];
52     buf[0]=first;
53     buf[1]=second;
54
55     p->mAdd8_WBuf(TMP117_DeviceID,TMP117_ConfigurationRegister,buf,2); 2
56
57 }
    
```

注：通过调用总线读写 API，操作 TMP117 寄存器。

6. OLED 实例

6.1 功能介绍



接入OLED模块效果图

6.2 测试效果

如上图所示（接入 OLED 模块效果图）

6.3 代码分析

实例对应工程文件：../project-OLED

- 1) 移植 nanochap_oled.c
- 2) GPIO 初始化

```

/*****
 * 描 述：LCD初始化
 * 入 参：无
 * 返回值：无
 *****/
void OLED_Init(void)
{
    //OLED 模块GPIO 初始化
    EPG_gpio_SetFuncMode_GPIOL16_Pin(EPG_GPIO0, OLED_CS, GPIO_MODE_OUTPUT);
    EPG_gpio_SetFuncMode_GPIOL16_Pin(EPG_GPIO0, OLED_SDA, GPIO_MODE_OUTPUT);
    EPG_gpio_SetFuncMode_GPIOL16_Pin(EPG_GPIO0, OLED_DC, GPIO_MODE_OUTPUT);
    EPG_gpio_SetFuncMode_GPIOL16_Pin(EPG_GPIO0, OLED_RST, GPIO_MODE_OUTPUT);
    EPG_gpio_SetFuncMode_GPIOL16_Pin(EPG_GPIO0, OLED_SCL, GPIO_MODE_OUTPUT);

    TIMER_SET(TIMER_OLED, 200); //delay_ms(200);
    while(TIMER_GET_MS(TIMER_OLED) != 0);

    OLED_Config(); //oled 初始化
    OLED_Fill(0x00); //清屏
}

```

注：根据 MCU 特性修改 1 部分代码，配置涉及的 GPIO 为输出模式。

3) GPIO 电平设置宏定义

```

*/
9  #define OLED_CS      12    //OLED片选
9  #define OLED_RST    8     //OLED复位 1
1 #define OLED_SDA    10    //OLED数据
2 #define OLED_SCL    9     //OLED时钟
3 #define OLED_DC     11    //OLED命令/数据选择 H=命令 L=数据
4
5 //endif
6 // OLED GPIO 电平设置宏定义
7 #define OLED_SDA_SET    EPG_gpio_SetData_ToPad_Pin(EPG_GPIO0, OLED_SDA) //;nrf_
8 #define OLED_SDA_CLEAR EPG_gpio_ClrData_ToPad_Pin(EPG_GPIO0, OLED_SDA) //nrf_g
9
10 #define OLED_DC_SET    EPG_gpio_SetData_ToPad_Pin(EPG_GPIO0, OLED_DC) //nrf_gp
11 #define OLED_DC_CLEAR EPG_gpio_ClrData_ToPad_Pin(EPG_GPIO0, OLED_DC) //nrf_gp
12
13 #define OLED_SCL_SET    EPG_gpio_SetData_ToPad_Pin(EPG_GPIO0, OLED_SCL) //nrf_g
14 #define OLED_SCL_CLEAR EPG_gpio_ClrData_ToPad_Pin(EPG_GPIO0, OLED_SCL) //nrf_g
15
16 #define OLED_CS_SET    EPG_gpio_SetData_ToPad_Pin(EPG_GPIO0, OLED_CS) //nrf_gp
17 #define OLED_CS_CLEAR EPG_gpio_ClrData_ToPad_Pin(EPG_GPIO0, OLED_CS) //nrf_gp
18
19 #define OLED_RST_SET    EPG_gpio_SetData_ToPad_Pin(EPG_GPIO0, OLED_RST) //nrf_g
20 #define OLED_RST_CLEAR EPG_gpio_ClrData_ToPad_Pin(EPG_GPIO0, OLED_RST) //nrf_g
21
22
23
24

```

- ① 定义 GPIO 端口号;
- ② 定义 GPIO 电平设置宏;

测试代码:

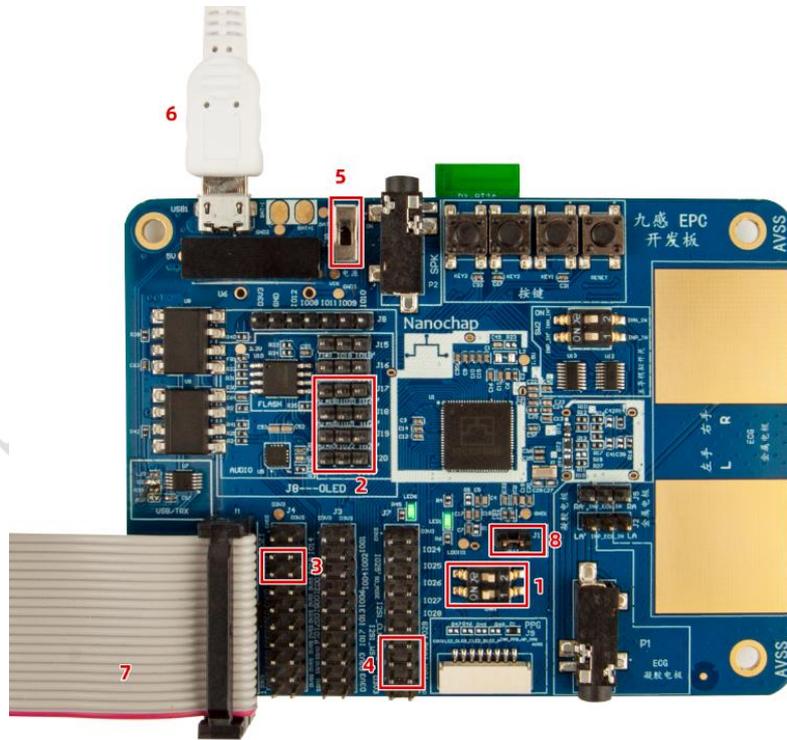
```

OLED_Init(); //OLED 初始化
please_add_liquid(); //OLED 显示初始化界面

```

7. 硬件 SPI 读取 W25Q128 ID

7.1 功能介绍



在通用接线设置的基础上，位置 2 路线设置如图,实现对读取 W25Q128 ID。

7.2 测试效果



7.3 代码分析

实例对应工程文件: ../project-SPI

```

109 //硬件SPI初始化
110 void SpiMacInit(EPG_SPI_TypeDef *EPG_SPI)
111 {
112
113
114     EPG_SPI_CTRL2_config(EPG_SPI,
115         0x0, //0x3, //T2C
116         0x0, //0x3, //C2T
117         0x1, //0x0, //0x1, //NSS1
118         0x1, //0x3, //0x1, //SAMP PHASE 1
119         0x0, //tx dma 禁止
120         0x0, //rx dma 禁止
121         0x7); //8 bit charlength
122
123
124     uint32_t new_reg_ctrl = 0;
125
126     new_reg_ctrl |= 0x07 << EPG_SPI_BAUD_RATE_Pos; //速度
127     new_reg_ctrl &=~ EPG_SPI_LSB_SEL_Msk; //MSB 2
128
129     new_reg_ctrl |= EPG_SPI_NSS_TOGGLE_Msk; //
130     new_reg_ctrl |= EPG_SPI_NSS_MST_CTRL_Msk; //硬件NSS, new_reg_ctrl &=~ EPG_SPI_NSS_MST_CTRL_Msk;
131
132     new_reg_ctrl |= EPG_SPI_CPOL_Msk; //极性
133     new_reg_ctrl |= EPG_SPI_CPHA_Msk; //相位
134     new_reg_ctrl |= EPG_SPI_MST_SLV_SEL_Msk; //mst spi
135     new_reg_ctrl |= EPG_SPI_EN_Msk; //enable spi
136
137
138     EPG_SPI->CTRL1 = new_reg_ctrl;
139 }
    
```



- 1) SPI 控制寄存器 2 初始化
- 2) SPI 控制寄存器 1 初始化

```

40= /*
41  函数功能: SPI 总线传输一字节数据
42  输入参数: *EPG_SPI:SPI 结构体 指针, 指向具体SPI
43           dat:要发送的数据
44  输出参数: SPI接收到的数据
45
46  */
47= uint8_t SPI_ReadWriteOneByte(EPG_SPI_TypeDef *EPG_SPI,uint8_t dat)
48 {
49     uint16_t tcnt=0;
50     //等发送为空
51     while(((EPG_SPI->INTSTATUS & EPG_SPI_TXE_INT_STS_Msk)>>EPG_SPI_TXE_INT_STS_Pos)==0)
52     {
53         tcnt++;
54         if(65535<=tcnt) return 0xA5;
55     };
56     //发送字节写入发送寄存器
57     EPG_SPI->THR = dat;
58     tcnt=0;
59     //while(((EPG_SPI->INTSTATUS & EPG_SPI_RXNE_INT_STS_Msk)>>EPG_SPI_RXNE_INT_STS_Pos)==0)
60     //等发送完成
61     while((EPG_SPI->FSR & EPG_SPI_BUSY_Msk) >> EPG_SPI_BUSY_Pos)
62     {
63         tcnt++;
64         if(65535<=tcnt) return 0xA6;
65     };
66     return EPG_SPI->RBR; //返回接收到的数据
67 }
68
69
70

```

SPI 传输一字节数据

- ① 等总线空闲;
- ② 发送数据写入发送寄存器;
- ③ 等发送结束;
- ④ 返回接收到的数据。

3) W25QXX 测试

- ① 初始化 SPI2 相关 GPIO，并且初始化 SPI2；

```

22 //W25QXX 初始化
23 void W25QXX_Init(void)
24 {
25     Spi2GpioInit();           //SPI2 GPIO初始化
26     SpiMacInit(EPG_SPI2);    //初始化硬件SPI2
27 }
  
```

- ② 读 W25QXX 芯片 ID。

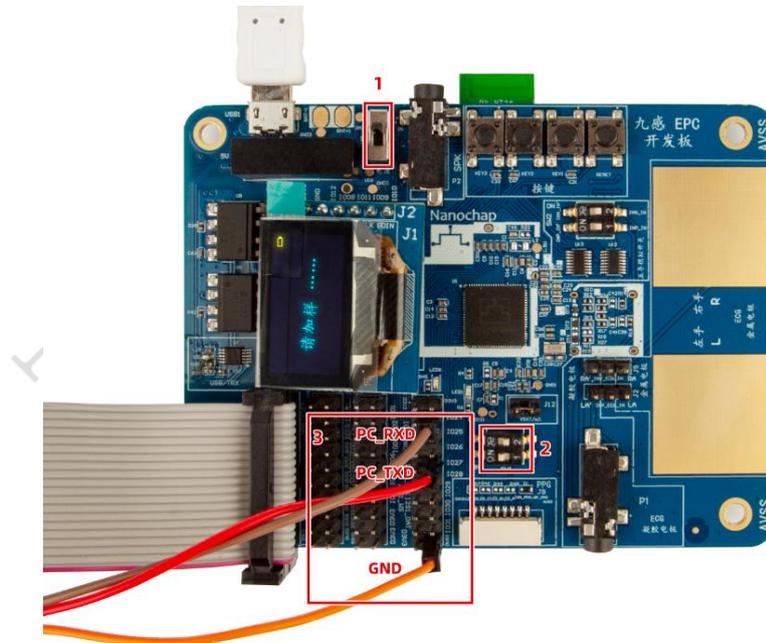
```

41 /*
42  * 函数功能：读W25QXX ID
43  */
44 u16 W25QXX_ReadID(void)
45 {
46     u16 Temp = 0;
47     //Spi2CsLow();//W25QXX_CS=0;
48     SPI2_ReadWriteByte(0x90); //发送读取ID命令
49     SPI2_ReadWriteByte(0x00);
50     SPI2_ReadWriteByte(0x00);
51     SPI2_ReadWriteByte(0x00);
52     Temp|=SPI2_ReadWriteByte(0xFF)<<8;
53     Temp|=SPI2_ReadWriteByte(0xFF);
54     //Spi2CsHigh();//W25QXX_CS=1;
55     return Temp;
56 }
  
```

8. PC 通过板载 BLE 与手机 APP 通讯

本文档目的在于排除EPC001的影响，单独测试BLE模块AT指令。

8.1 接线方式



- 1) 打开电源
- 2) 断开 EPC001，使 EPC001 不工作，防止 EPC001 TX，RX 影响
- 3) 连接 USB 转串口 PC_RXD PC_TXD GND (为 USB 转串口引脚)



4) 验证

PC打开串口调试助手，波特率设为9600，通过串口发送AT指令查询版本号，ADD说明接线成功。

8.2 PC通过BLE与手机传输数据

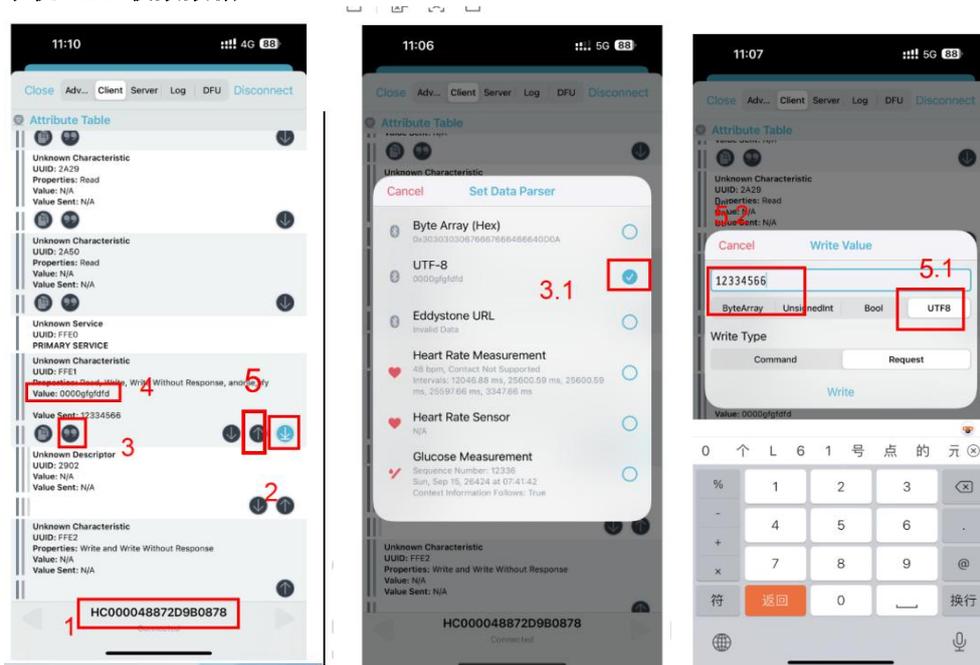
准备工作：手机端安装nRF Connect，扫描连接到BLE模块。

1) 串口调试助手收发数据



- ① 上电，并查询 BLE 处理透传开启状态；
- ② 接收到手机发送来的数据；
- ③ PC 要发送的数据；
- ④ 发送。

2) 手机 APP 收数数据



- ① 连接到的 BLE 名字;
- ② 开启自动接收;
- ③ 进入设置数据格式, 选择 3.1: 为 UTF-8;
- ④ 当 PC 发送数据时, 此为显示接收到的数据;
- ⑤ 手机发送数据 5.1, 设置发送数据格式 5.2, 发送内容 write 发送。

9. 开发板通过板载 BLE 与手机 APP 通讯

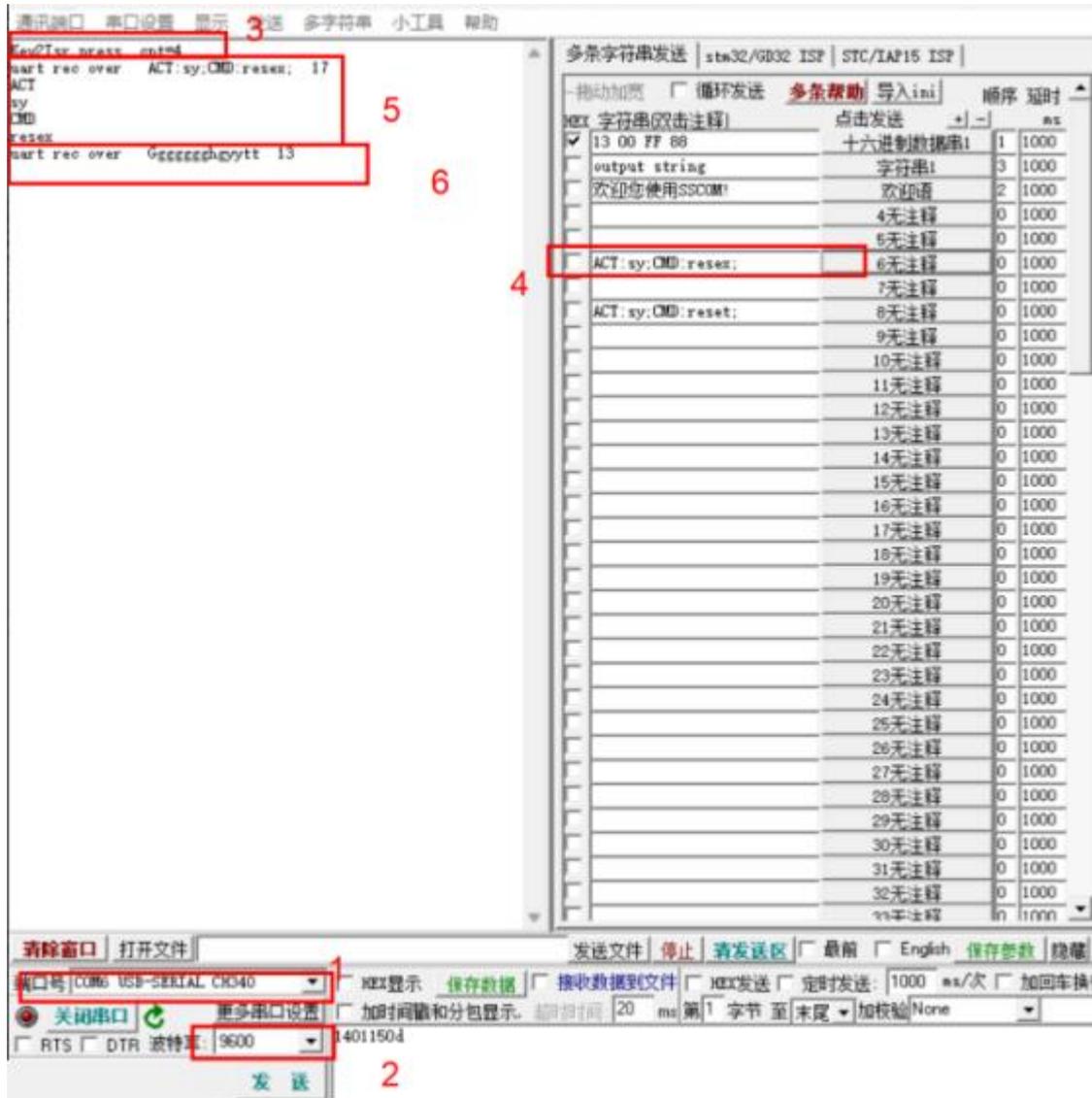
9.1 目的

- 1) 实现轻按 KEY2 手机端收到: `Key2Isr press cnt=1`
- 2) 实现 PC 串口调试助手发送字符串, 手机端同步显示
- 3) 实现手机端发送数据, EPC001 通过串口打印到 PC 机

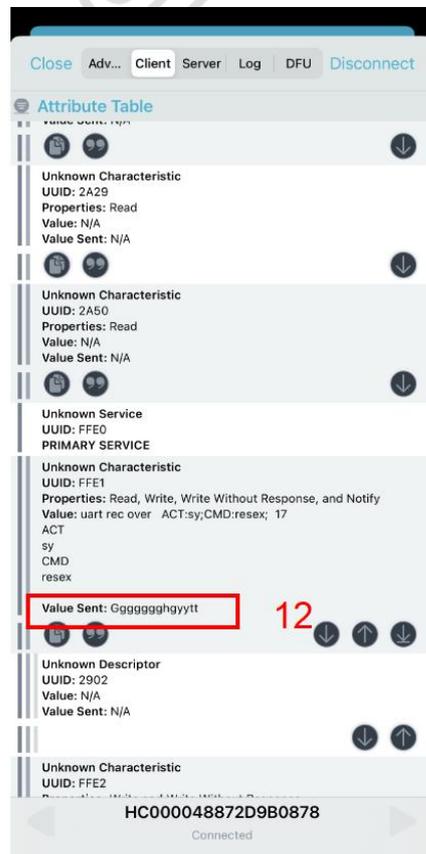
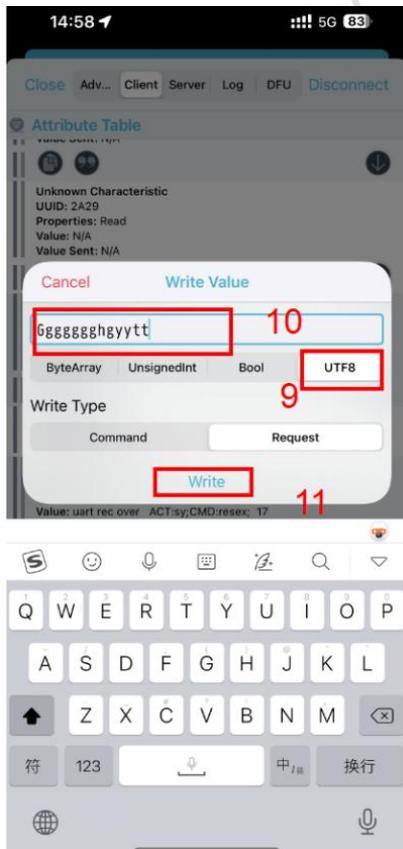
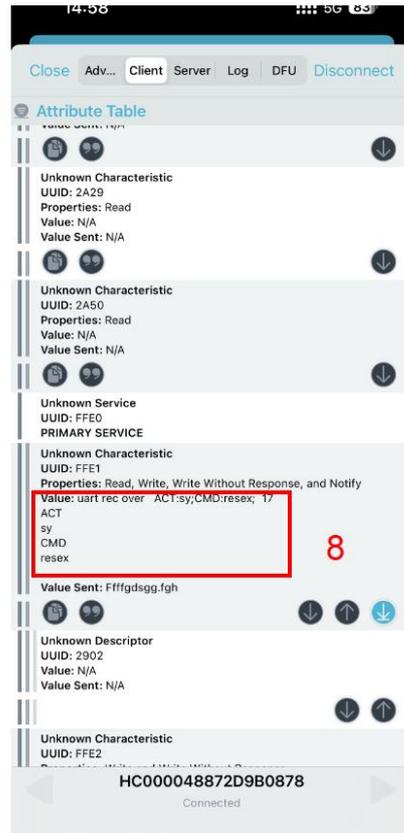
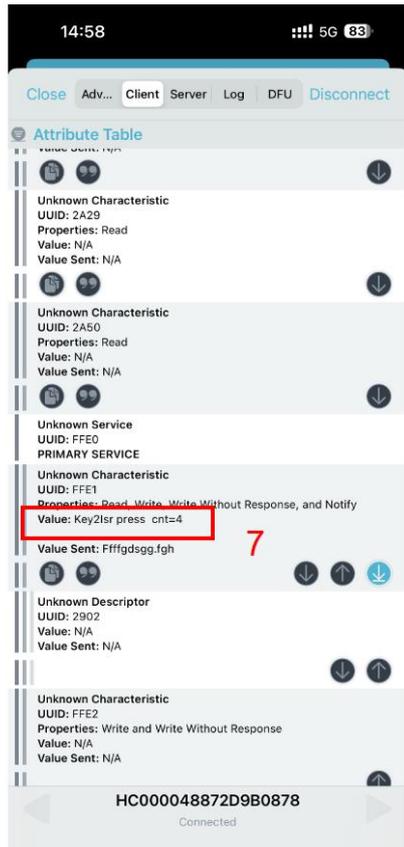
9.2 准备工作

- 1) 开发板插入 Micro USB 线, 并接入 PC, 打开电源
- 2) 下载 DEMO 程序 `../project-BLE`
- 3) PC 打开串口调试助手, 波特率设置为 9600, 手机打开 nRF Connect

9.3 测试



- 1) 选择正确的串口
- 2) 波特率设置为 9600
- 3) 当按下 PCB 上 KEY2 事产生按键事件
- 4) PC 发送指令给 EPC001
- 5) 解析出的 PC 端指令
- 6) 接收到 BLE 字符串



- 7) 手机端收到的按键事件
- 8) 手机端收到的 EPC 发送的字符串
- 9) 手机端发送数据给 EPC, 数据格式
- 10) 数据内容
- 11) 点击发送
- 12) 手机端回显

9.4 代码解析

1) 按键中断回调

```

//-----
/*
  创建一个模拟发送串口，返回发送函数
 */
void Key2Isr(void)
{
    static u32 cnt=0;
    cnt++;

    printf("Key2Isr press cnt=%d\n",cnt);
}

```

注：因串口直接与BLE连接，所以printf就是往BLE发送数据。

2) 外设实始化

```

void UserInit(void)
{
    Timer0MacInit(EPG_TIMER0,TICK_1MS);           //定时器0 初始化

    Uart1MacInit(UART_BPS_9600);                 //串口1初始化

    //gpio 中断初始化  GPIO号，中断优先级，中断回调
    GpioExitIntMacInit(KEY2_GPIO_NUM,3,Key2Isr); //按键初始化

    //LED1 初始化
    LedMacInit(LED1_GPIO_PIN,&gLed1);           //LED实始化
}

```

3) 主函数

```

void UserMain(void)
{
    while(true)
    {
        SysTickPro();           //系统节拍处理

        if(gSysTime.m1s)       //LED1 1S 周期闪烁
        {
            gLed1.mTurn(gLed1.mNum);
        }

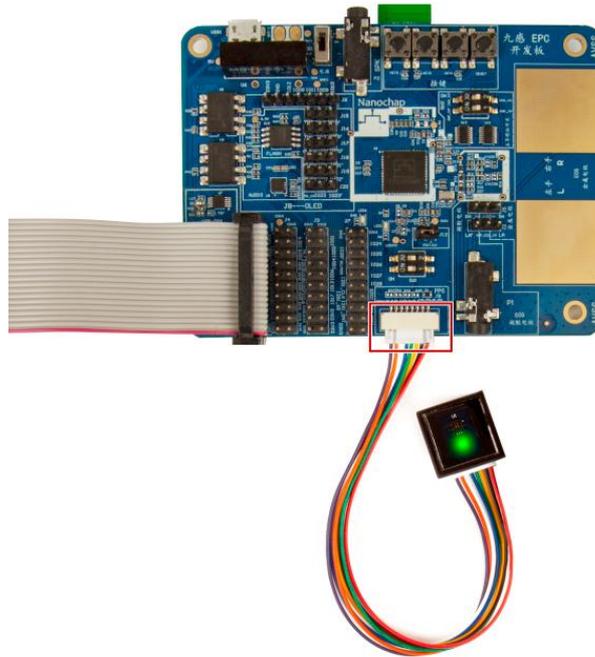
        Uart1Pro();           //串口接收处理
    }
}
    
```

Uart1Pro: 实现串口指令解析

10.PPG

10.1 反射式硬件设置

接上PPG反射式传感器，如下图所示：



开发板设置

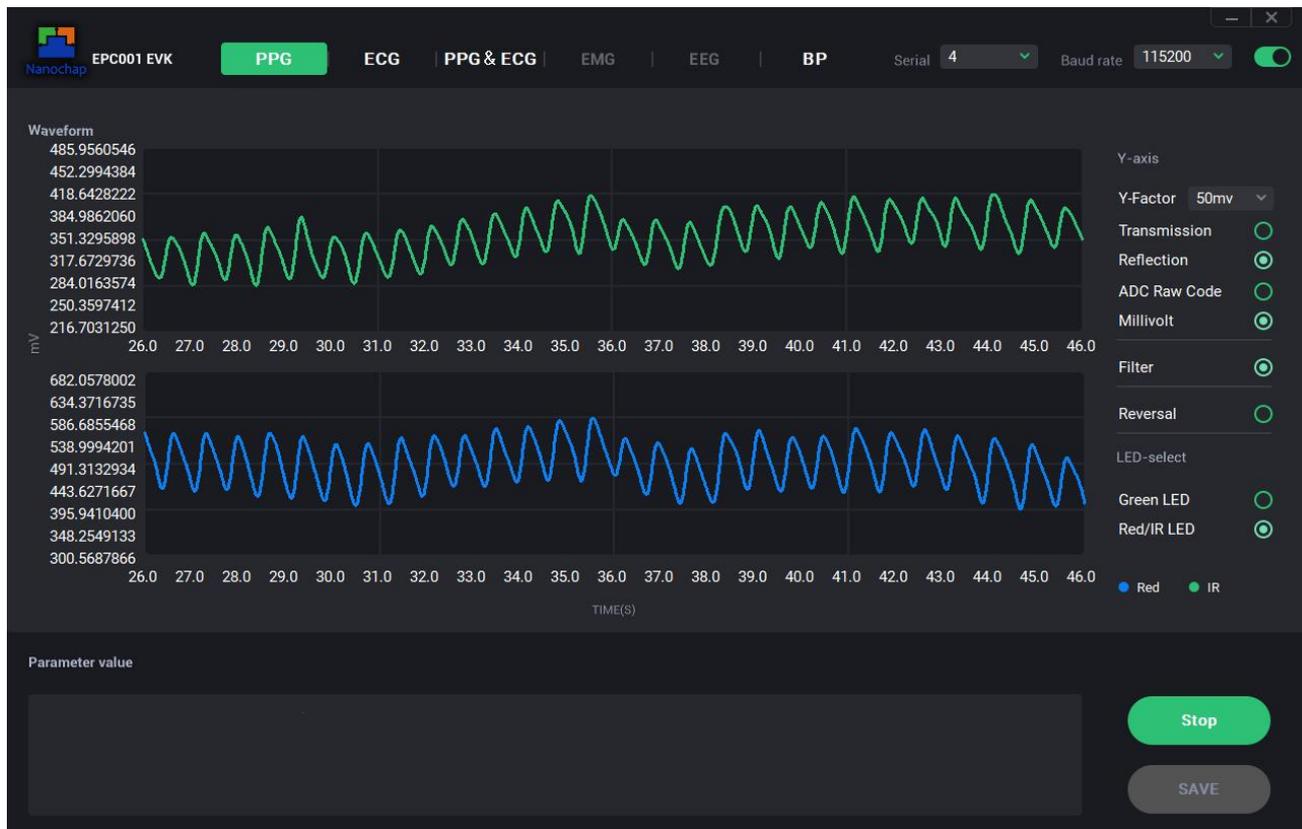
1) PPG Green LED



PPG Green

- ✧ **Y-axis:** 切换 AD 采样值与电压值，ADC RAW Code AD 原始数据；Millivolt 电压值(mV)。
- ✧ **Filter:** 滤波、不滤波。
- ✧ **Reversal:** 波形镜像。
- ✧ **Y-Factor:** 5mV，纵向每格 5mV；20mV，纵向每格 20mV；50mV，纵向每格 50mV。
- ✧ **LED-select:** Green LED，Red/IR LED。
- ✧ **Measure/Stop:** 开始/停止采集数据。
- ✧ **SAVE:** 设置波形数据保存路径。

2) PPG Red/IR LED 反射式



PPG Red/IR LED 波形

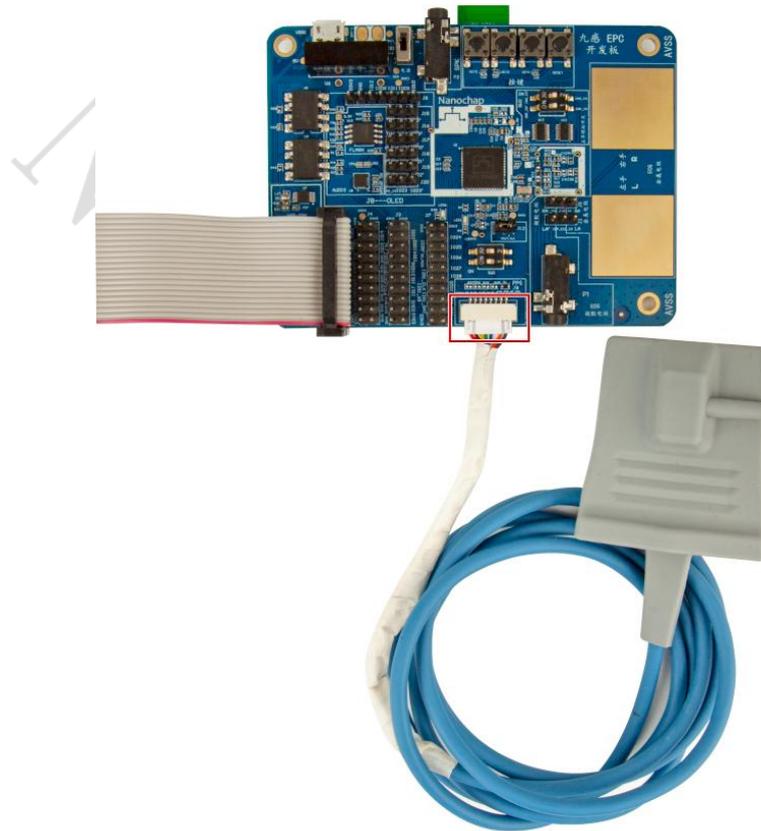
- ✧ **Y-axis:** 切换 AD 采样值与电压值，ADC RAW Code AD 原始数据；Millivolt 电压值(mV)。
- ✧ **Filter:** 滤波、不滤波。
- ✧ **Reversal:** 波形镜像。
- ✧ **LED-select:** Green LED, Red/IR LED。
- ✧ **Reflection / Transmissiion**（反射模式/透射模式选择）（默认反射式），反射模式下需选择 Reflection。
- ✧ **Measure/Stop:** 开始/停止采集数据。
- ✧ **SAVE:** 设置波形数据保存路径。

10.2 反射式代码分析

实例对应工程文件：../project_ppg

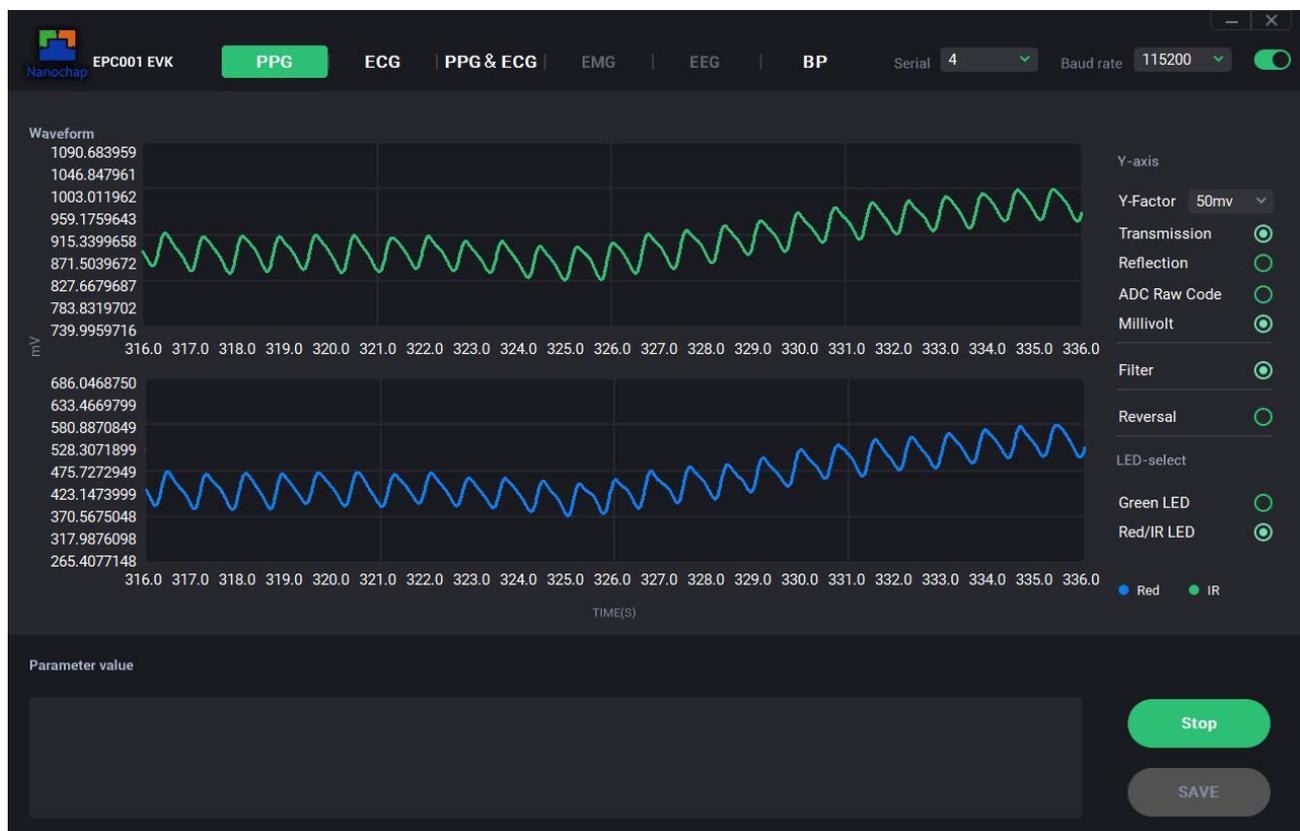
10.3 透射式硬件设置

接上PPG透射式传感器，如下图所示：



开发板设置

透射式测试效果



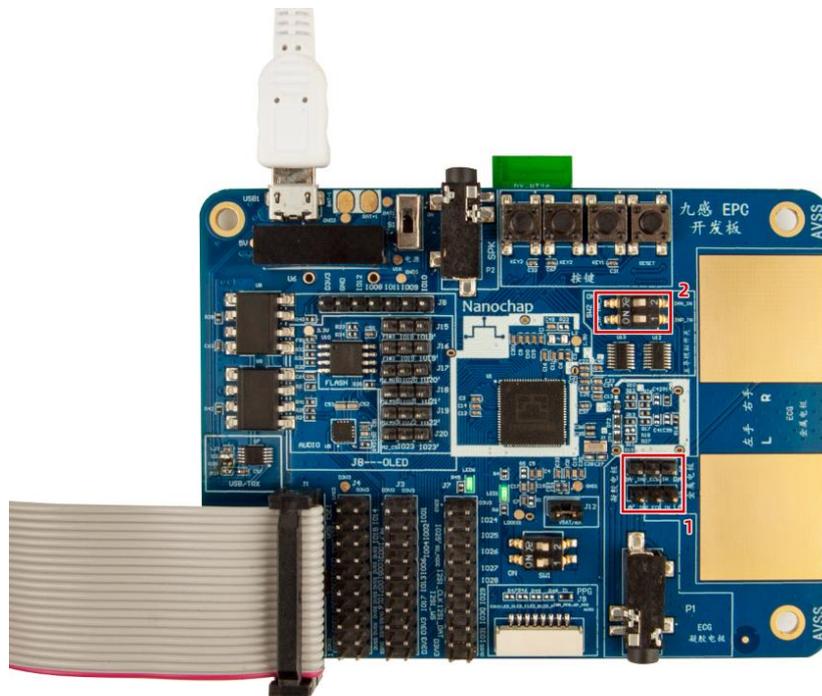
- ✧ **Y-axis:** 切换 AD 采样值与电压值，ADC RAW Code AD 原始数据；Millivolt 电压值(mV)。
- ✧ **Filter:** 滤波、不滤波。
- ✧ **Reversal:** 波形镜像。
- ✧ **LED-select:** Green LED, Red/IR LED。
- ✧ **Reflection / Transmissiion**（反射模式/透射模式选择）（默认反射式），透射模式下需选择 Transmissiion。
- ✧ **Measure/Stop:** 开始/停止采集数据。
- ✧ **SAVE:** 设置波形数据保存路径。

10.4 透射式代码分析

实例对应工程文件： ../project_ppgts

11. ECG 实例

11.1 硬件设置



开发板设置

- 1) ECG 信号连接到板载电极
- 2) ECG 信号断开外设连接，防止干扰

使用建议：

- 开发板用充电宝供电或锂电池供电，若电脑供电，会引入额外噪声；
- 采用带隔离的 USB 串口工具，尽量远离电脑，或用 USB 延长线；
- 采用手持板子金属电极，会引入肌电信号，可尝试选用 15-20Hz 低通滤波器滤除。

11.2 界面及图形



ECG界面

- ✧ **Y-axis:** 切换 AD 采样值与电压值，ADC RAW Code AD 原始数据；Millivolt 电压值(mV)。
- ✧ **Filter-set:** 滤波器设置。
- ✧ **HP- Filter:** 高通滤波器。
- ✧ **LP- Filter:** 低通滤波器。
- ✧ **GAIN:** PGA 增益，x 1、x 2、x 4、x 6、x 8、x 12、x 60、x 120。

(注：增益设置为 12 倍时，界面显示的波形幅度值已去掉了 12 倍的增益，即为原始波形大小，其它波形幅度值均为对应的增益设置。)

- ✧ **Reversal:** 波形镜像。
- ✧ **Measure/Stop:** 开始/停止采集数据。
- ✧ **SAVE:** 设置波形数据保存路径。

11.3 代码分析

实例对应工程文件：../project_ecg

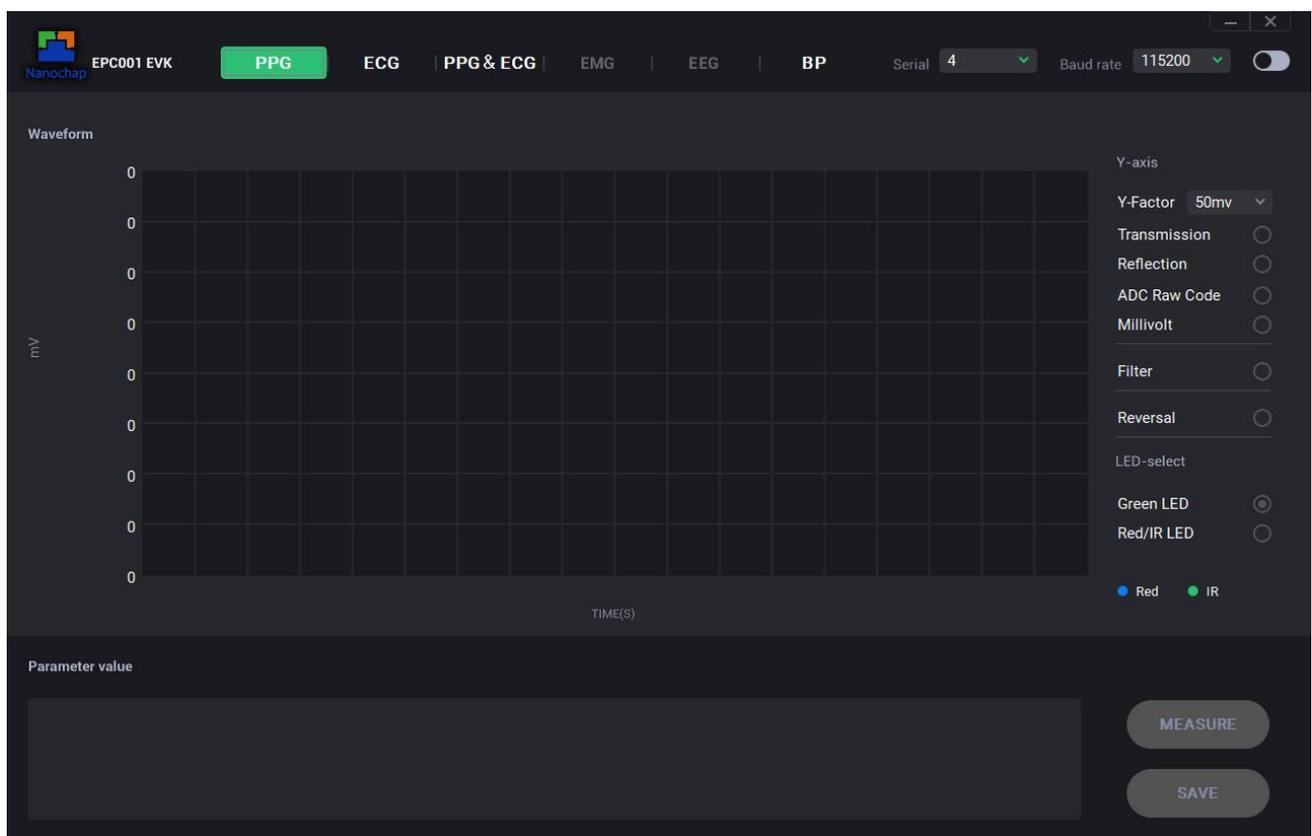
12.综合例程使用

12.1 下载烧录文件

文件所在目录：
./nine-sense-epc001.elf

12.2 上位机使用

上位机开启后默认界面：



- 1) 采用带隔离的 USB 串口工具将开发板和电脑端连接起来
- 2) 首先打开上位机软件，其次选择相应的串口以及波特率（波特率为 115200），再次点击波特率旁边的按钮即可，最后对上位机进行操作
- 3) 选择模式后点击 Measure 开始运行
- 4) 测试各模式时只需选择上位机上方的模式即可（在选择模式时需要将先运行的模式按 Stop 停止）
- 5) PPG 模式



Green LED和Red/IR LED的反射模式均需使用反射式灯板模块，将大拇指轻轻覆盖灯板上；
详见本手册 [PPG Red/IR LED 反射式](#)

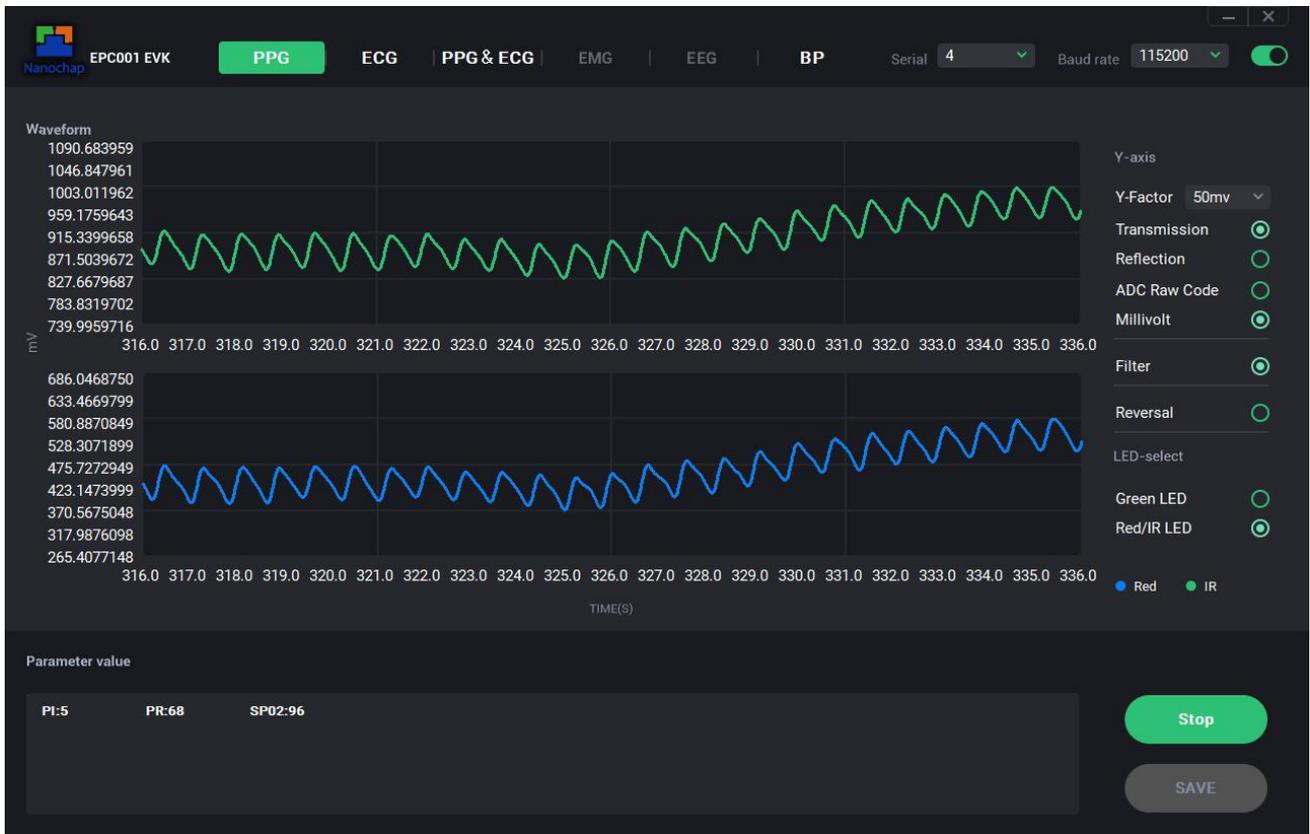


✧ **Reflection / Transmissiion**（反射模式/透射模式选择）（默认反射式），反射模式下需要手动选择 Reflection。

✧ **Parameter value:**

- i. SPO2: 血氧饱和度
- ii. PI: 灌注指数
- iii. PR: 脉率

Red/IR LED透射模式需要使用透射式灯板模块，将食指放入指套内；
 详见本手册 [PPG Red/IR LED 透射式](#)



✧ **Reflection / Transmissiion**（反射模式/透射模式选择）（默认反射式），透射模式下需要手动选择 Transmissiion。

✧ **Parameter value:**

- i. SPO2: 血氧饱和度
- ii. PI: 灌注指数
- iii. PR: 脉率

6) ECG 模式

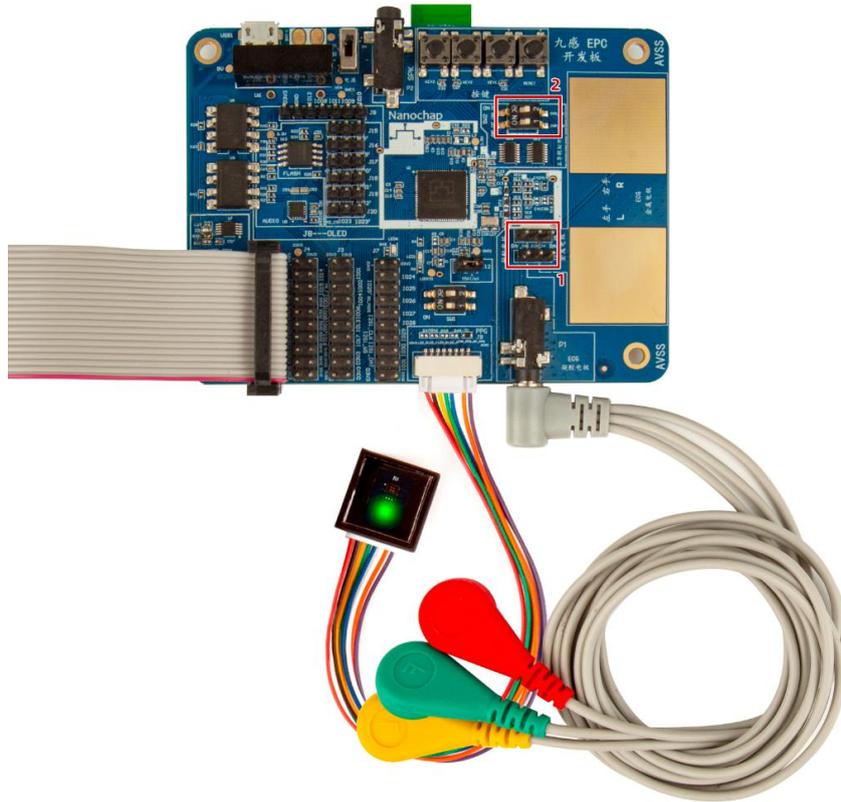
需要将电极贴到身体对应部位（红色贴右胳膊，黄色贴左胳膊，绿色贴右腿）；或者使用板载电极，双手持在金属电极上，按照丝印提示放置手指。

详见本手册[ECG实例](#)

7) PPG&ECG 模式

测试需要同时采用PPG以及ECG的测试方法。

■ 硬件设置：



开发板设置

- ① ECG 信号线连接到凝胶电极；
- ② ECG 信号断开外设连接，防止干扰。

使用建议：

- 开发板用充电宝供电或锂电池供电，若电脑供电，会引入额外噪声；
- 采用带隔离的 USB 串口工具，尽量远离电脑，或用 USB 延长线。

■ 界面及图形



EEG&PPG界面

Parameter value:

- ✧ **ARRHYTHMIA:** 1=心律不齐, 0=无心律不齐
- ✧ **BREATH_FREQ:** 呼吸速率
- ✧ **ECG_HR:** 心率
- ✧ **LVET:** 左心室射血时间
- ✧ **r-MSSD:** 相邻 RR 间期差值均方平方根, 计算公式为:

$$r - MSSD = \sqrt{\frac{\sum_{i=1}^N (RR_i - RR_{i+1})^2}{N}}$$

N 为采样过程中相邻 R 波的 R-R 间期个数;

r-MSSD 为相邻 R-R 间期差值均方的均方根, 单位为毫秒 (ms);

◇ **pNN50**: 相邻 RR 间期之差>50ms 的个数占总 RR 间期个数的百分比, 计算公式为:

$$pNN_{50} = \frac{\sum_{i=1}^N NN_{50}(i)}{N}$$

其中, $NN_{50}(i) = 1, \text{ if } RR_i > 50; NN_{50}(i) = 0, \text{ if } RR_i \leq 50;$

N 为采样过程中相邻 R 波的 R-R 间期个数;

$\sum_{i=1}^N NN_{50}(i)$ 为计算出所有正常相邻心跳间差距超过 50ms 的个数;

- ◇ **PPG_PR**: 脉率
- ◇ **QT**: QT 间期
- ◇ **SDANN**: RR 间期平均值标准差, 计算公式为:

$$SDANN = \sqrt{\frac{\sum_{i=1}^N (\text{Mean } RR(i)_{32\text{intervals}} - \text{Mean}(\text{Mean } RR_{32\text{intervals}}))^2}{N}}$$

$\text{Mean}(\text{Mean } RR_{32\text{intervals}})$ 为采样过程所有的 32 个 R-R 间期平均值的平均值;

$$\text{Mean } RR = \frac{1}{N} (\sum_{i=1}^N RR_i);$$

N 为采样过程中相邻 R 波的 R-R 间期个数;

RR_i 为采样过程中每两个相邻 R 波的 R-R 间期;

N 为每包含了 32 个 R-R 间期平均值的个数, 例: 160 个 R-R 间期, 每 32 个 R-R 间期取一次平均值, 此时 $N=5$;

- ◇ **SDNN**: 全部窦性心搏 RR 间期 (瞬时心率) 的标准差, 计算公式为:

$$SDNN = \sqrt{\frac{1}{N} \sum_{i=1}^N (RR_i - \text{Mean } RR)^2}$$

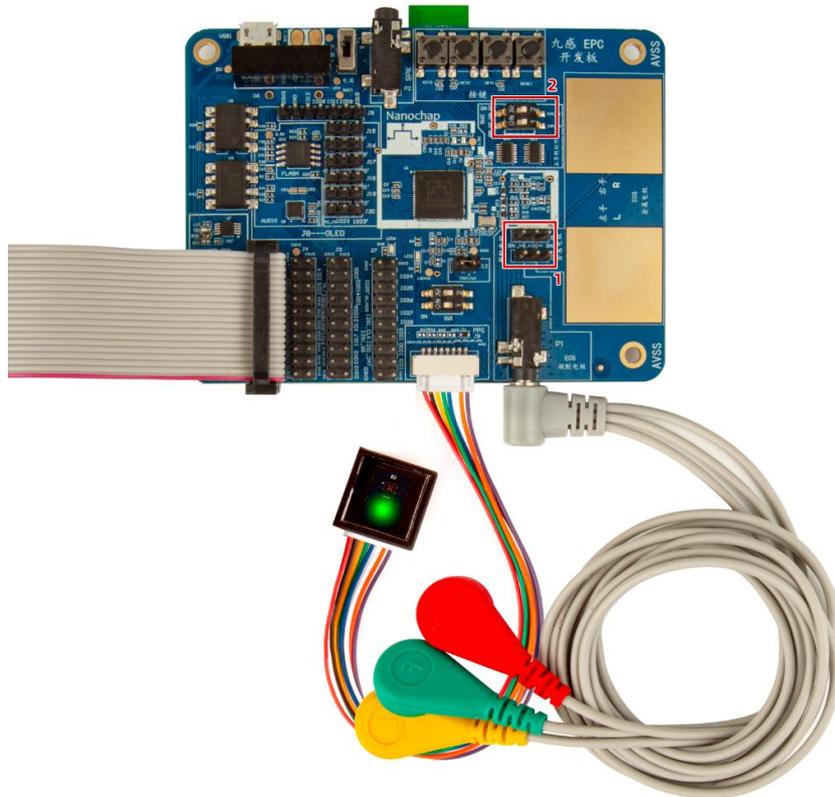
N 为采样过程中相邻 R 波的 R-R 间期个数;

- ◇ **SNA**: 焦虑指数
- ◇ **PAT**: 脉搏波到达时间

8) BP 模式

测试时的操作方法与PPG&ECG的操作方法一致。

■ 硬件设置：



开发板设置

- ① ECG 信号线连接到凝胶电极；
- ② ECG 信号断开外设连接，防止干扰。

使用建议：

- 开发板用充电宝供电或锂电池供电，若电脑供电，会引入额外噪声；
- 采用带隔离的 USB 串口工具，尽量远离电脑，或用 USB 延长线。

■ 界面及图形



BP界面

- ✧ **Height:** 身高;
- ✧ **Blood viscosity:** 血液粘稠度;
- ✧ **Measure/Stop:** 开始/停止采集;
- ✧ **SYS:** 高压;
- ✧ **DIA:** 低压。

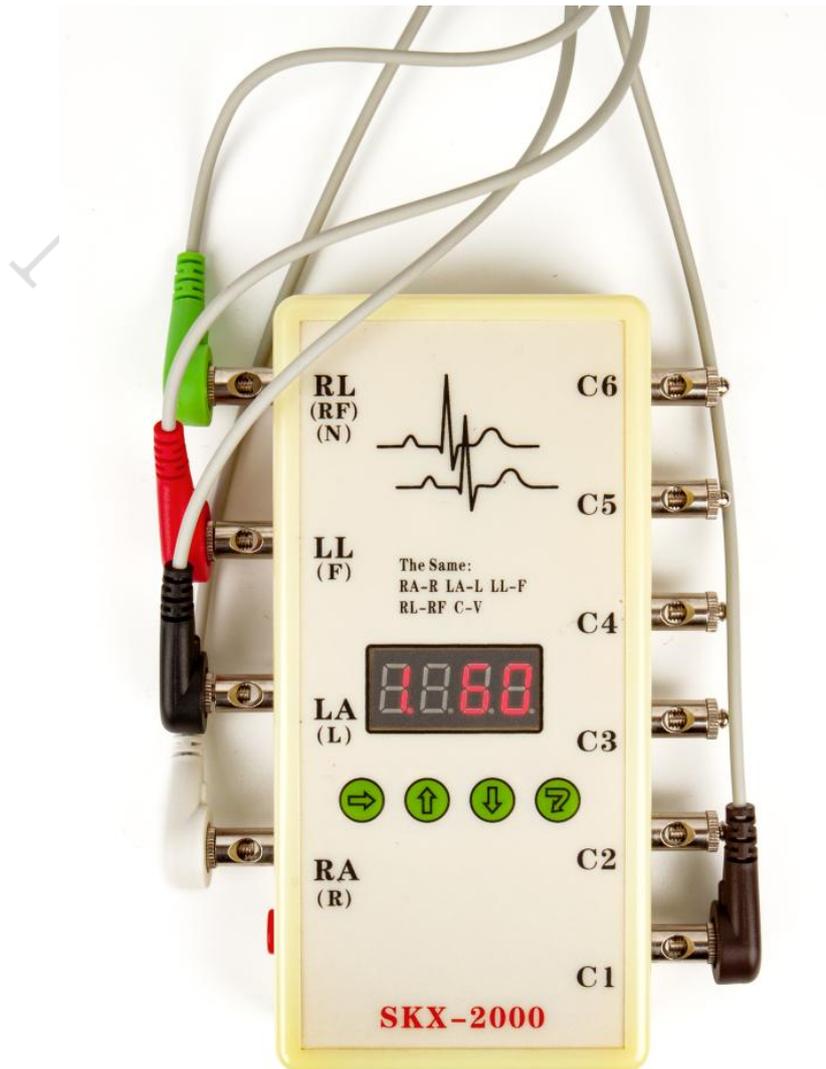
13.五导联测试说明

13.1 目的

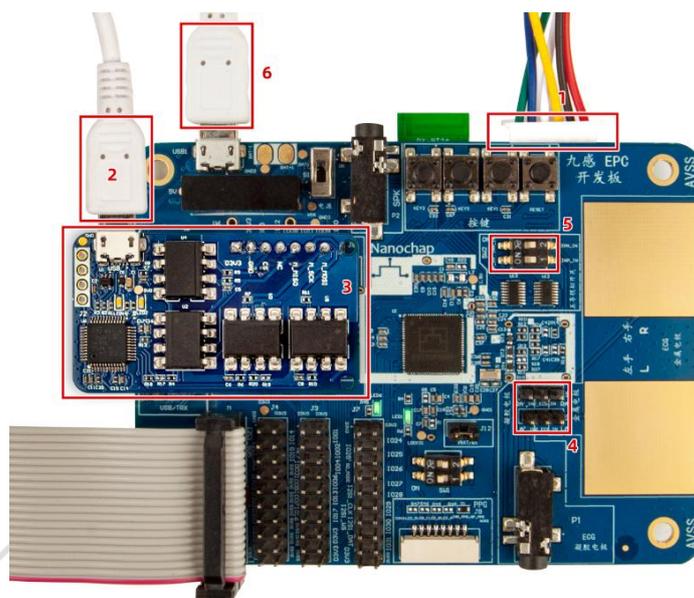
- 1) 如何搭建硬件测试环境
- 2) 五导联电极怎样接线
- 3) 五导联的波形有哪些，及它们的物理意义
- 4) 多路 ECG 测试硬件原理

13.2 准备软硬件环境

准备： 模拟器，一根5导联线、生命体征检测心电图开发板、SPI转串口电路模块、2根Micro USB线、烧录器+线（1根USB-Type A转USB-Type线、烧录器ARM-USB-TINY-H），接线后设置开发板拨码开关。

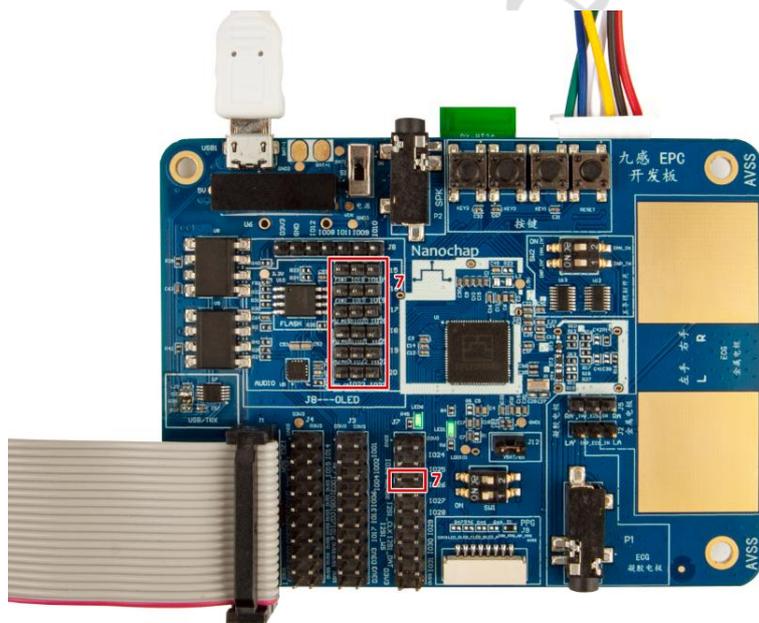


模拟器接线图



开发板接线及设置图

- 1) 模拟器信号“接”入
- 2) 插上 SPI 转串口板, 并使用 Micro USB 线连接到 PCB 本实例中为 COM5(USB 串口设备)
- 3) SPI 转串口电路模块
- 4) 板载 ECG 信息断开不接入 NG
- 5) 外部模拟信号接入 ON
- 6) 外部供电串口



跳线设置

7) 多路复用 IC 通道选择和使能控制，GPIO 跳线设置



PC显示的插入串口

① SPI 转串口电路模块对应的串口，多导联上位机读数据串口（本实例使用 SPI 转串口的原因，每次上传到 PC 数据量很大，使用 SPI 上传节省 EPC001 中断响应时间或 CPU 延时时间(避免出现波形不连续，失真;提高 EPC001 实时性)）；

② 板载 USB 转串口，本实例中只用于开发板供电。

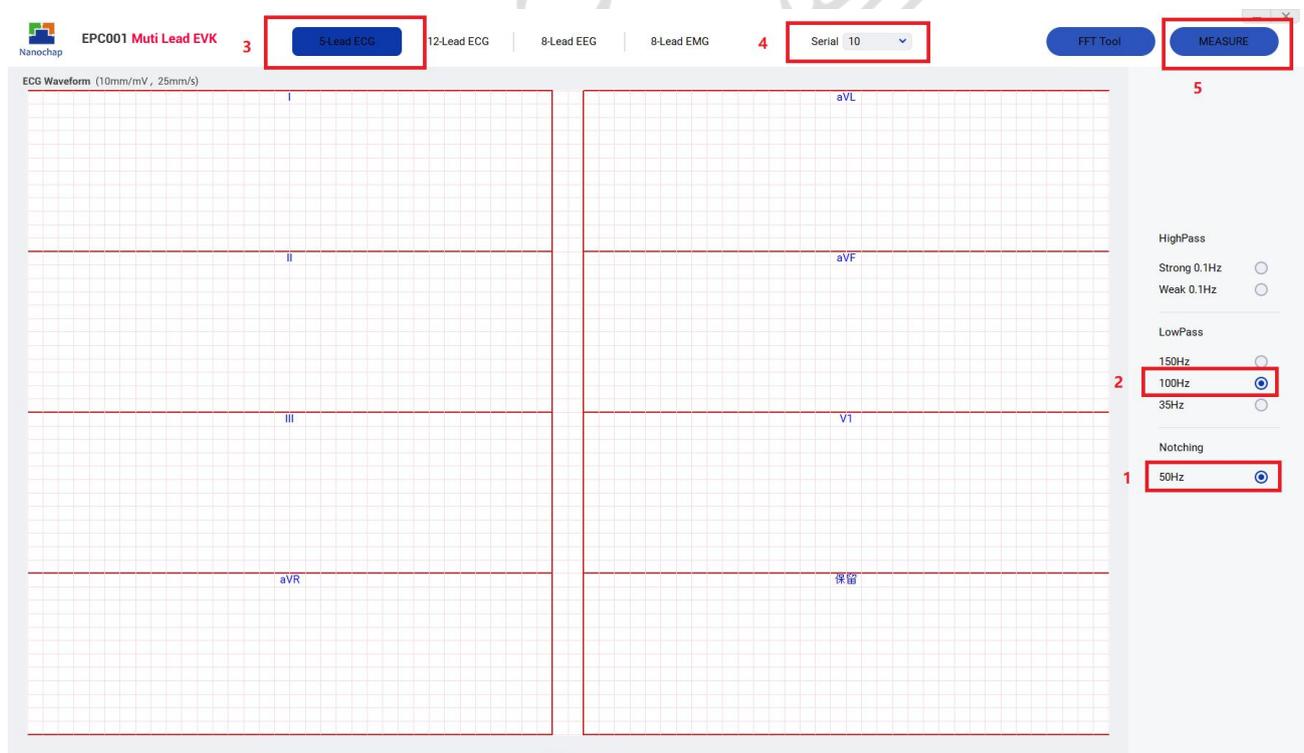
13.3 下载DEMO实例

实例对应工程文件： ../project_5_ecg

13.4 上位机软件

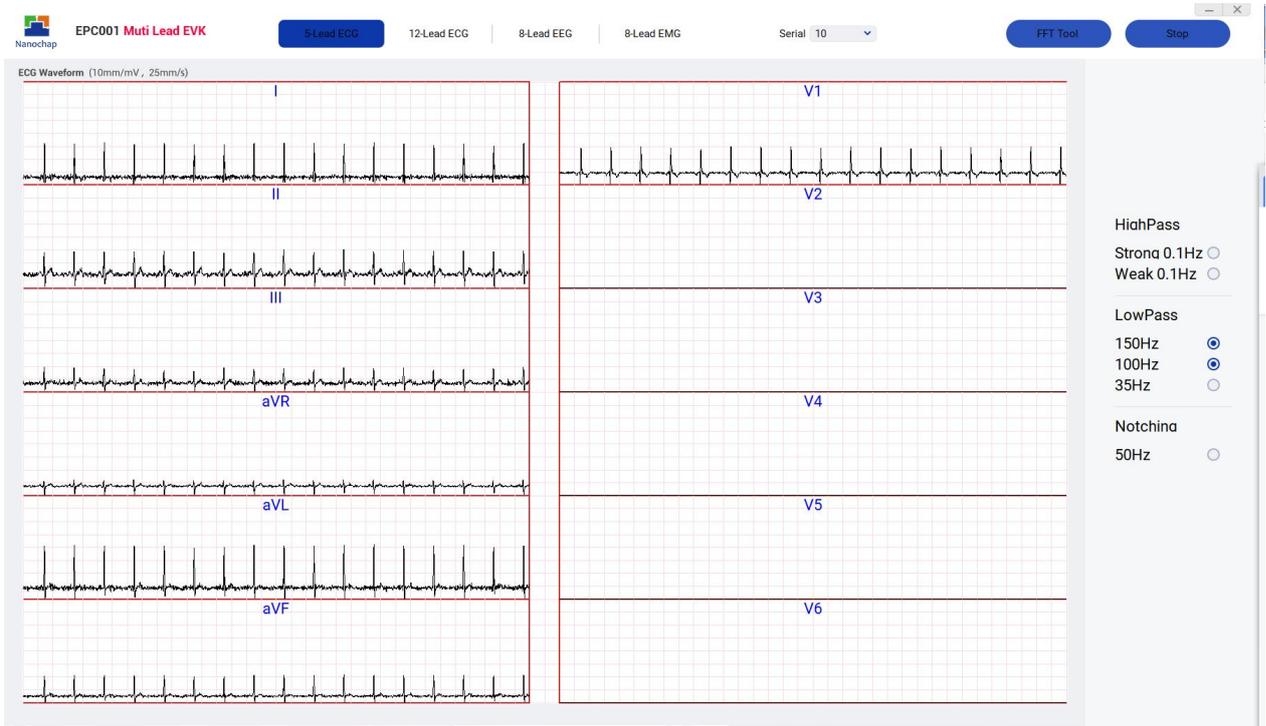


多导联上位机



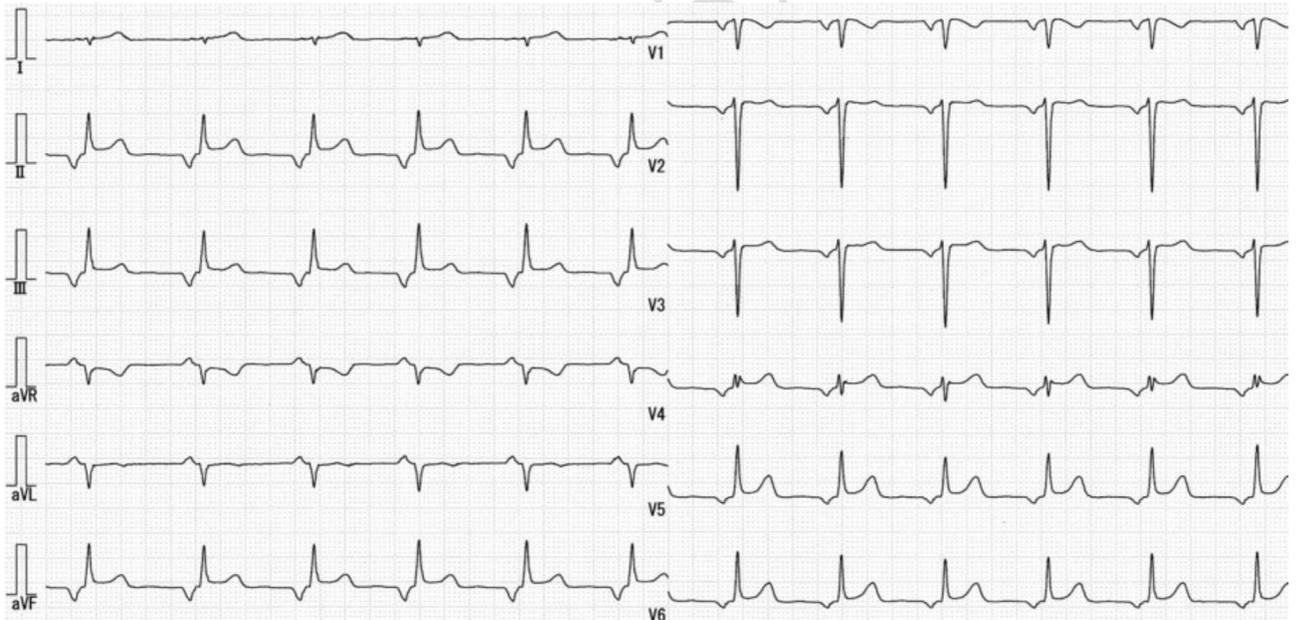
上位机设置

1) 根据上位机设置，然后得到上位机输出波形图



上位机输出波形图

2) 幅度和周期以及轮廓



13.5 五导联及对应波形原理

1) 生物特性

Lead I: $I = LA - RA$

Lead II: $II = LL - RA$

通过Lead I和Lead II可以求出以下导联：

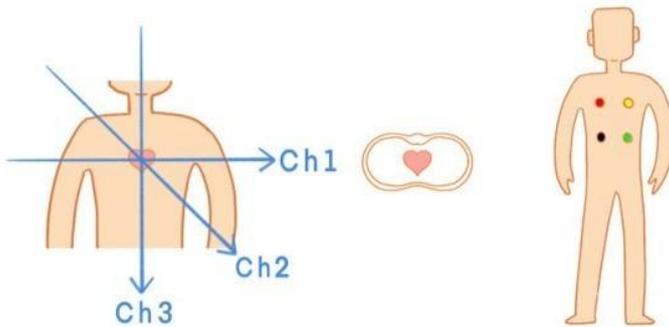
Lead III: $III = LL - LA$

$aVR = -(I + II)/2 = RA - (LA + LL)/2$

$aVL = I - II/2 = LA - (RA + LL)/2$

$aVF = II - I/2 = LL - (RA + LA)/2$

WCT(Wilson Central Terminal) = $(RA + LA + LL)/3$ ，五导联中，一般会加一个胸部导联，Lead V1威尔逊中心节点会作为胸部导联的参考电压。胸部导联Lead V1 = $V1 - WCT$ 。



① 标准导联

Lead I: $LA - RA$; Lead II: $LL - RA$; Lead III: $LL - LA$

② 加压单极肢体导联

aVR, aVL, aVF 不用再单独采集，可以用 I II III 通过软件计算

$$aVR = RA - (LA + LL) / 2 = -(I + II) / 2 ;$$

$$aVL = LA - (RA + LL) / 2 = I - II / 2 ;$$

$$aVF = LL - (RA + LA) / 2 = II - I / 2$$

$$\text{Lead V1} = V1 - WCT$$

$$= V1 - (RA + LA + LL) / 3$$

$$= (V1 - RA) / 3 + (V1 - LA) / 3 + (V1 - LL) / 3$$

$$= (V1 - RA + V1 - LA + V1 - LL) / 3$$

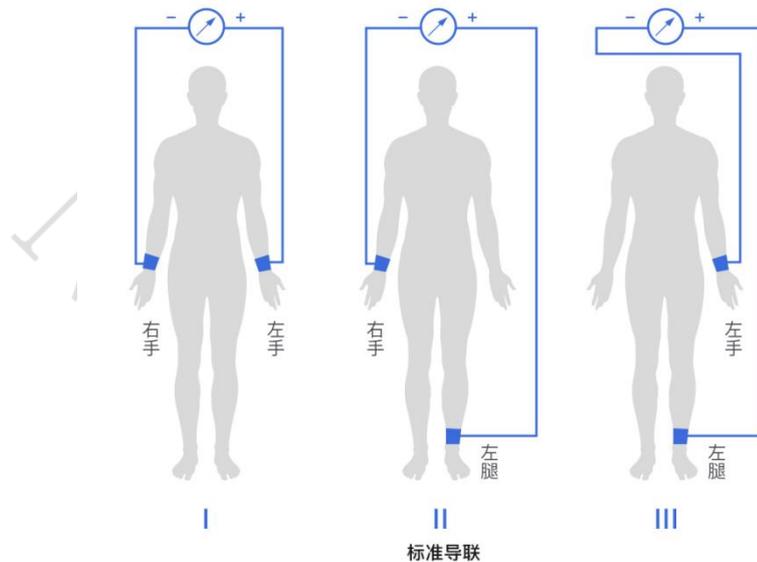
注: $WCT = (RA + LA + LL) / 3$

总结:

每个上传周期要采集 LA-RA , LL-RA, LL-LA, (V1-RA), (V1-LA), (V1-LL) 6组ECG信息号, (再计算, 封包, 上传)。

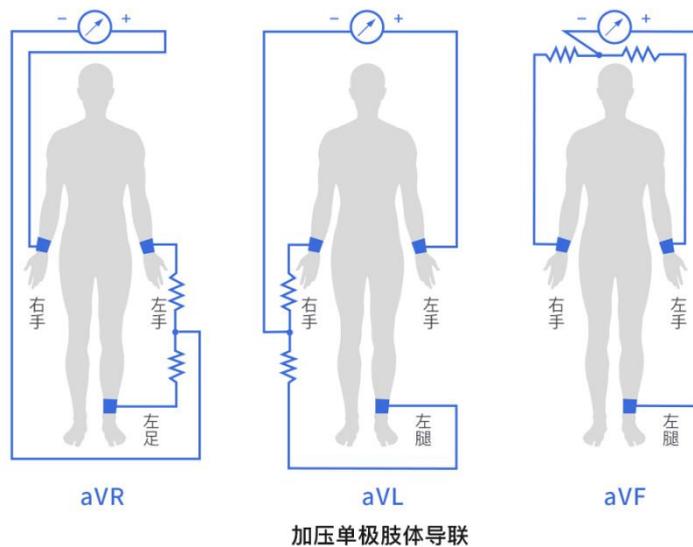
注: 辅助说明

标准导联I: 左, 右手之间电位差; 标准导联II: 左腿, 右手之间电位差; 标准导联III: 左腿和左手之间电位差; 标准导联记录的电压大小: $I+III=II$ 。



加压单极肢体导联

aVR: 右手左手, 左腿的结合电极; aVL: 左手右手, 左腿的结合电极; aVF: 左腿左手, 右手的结合电极。





2) 六组电极信号采集

分时采集ECG 数据

```

1 // 分时采集ECG数据
2 void Multi_ECG(void)
3 {
4     char data[48] = {0}; // 封装缓存区
5
6     //--- LA-RA
7     Multi_ECG_INN_LA();
8     Multi_ECG_INP_RA();
9     while((EPG_ECG0->INTSTATUS & EPG_ECG_FIFO_TH_INT_Msk) == 0);
10    for(int j = 0; j < 4; j++)
11    {
12        // 缓存 // 获取ECG数据
13        DATA[j] = EPG_ECG0->DATA & 0xffffffff;
14    }
15    result1 = DATA[3];
16
17    //--- LL-RA
18    Multi_ECG_INN_LL();
19    Multi_ECG_INP_RA();
20    while((EPG_ECG0->INTSTATUS & EPG_ECG_FIFO_TH_INT_Msk) == 0);
21    for(int j = 0; j < 4; j++)
22    {
23        // 缓存 // 获取ECG数据
24        DATA[j] = EPG_ECG0->DATA & 0xffffffff;
25    }
26    result2 = DATA[3];
27
28    //--- LL-LA
29    Multi_ECG_INN_LL();
30    Multi_ECG_INP_LA();
31    while((EPG_ECG0->INTSTATUS & EPG_ECG_FIFO_TH_INT_Msk) == 0);
32    for(int j = 0; j < 4; j++)
33    {
34        // 缓存 // 获取ECG数据
35        DATA[j] = EPG_ECG0->DATA & 0xffffffff;
36    }
37    result3 = DATA[3];
38
39    //--- V1-RA
40    Multi_ECG_INN_V1();
41    Multi_ECG_INP_RA();
42    while((EPG_ECG0->INTSTATUS & EPG_ECG_FIFO_TH_INT_Msk) == 0);
43    for(int j = 0; j < 4; j++)
44    {
45        // 缓存 // 获取ECG数据
46        DATA[j] = EPG_ECG0->DATA & 0xffffffff;
47    }
48    resultV1RA = DATA[3];
49
50    //--- V1-LA
51    Multi_ECG_INN_V1();
52    Multi_ECG_INP_LA();
53    while((EPG_ECG0->INTSTATUS & EPG_ECG_FIFO_TH_INT_Msk) == 0);
54    for(int j = 0; j < 4; j++)
55    {
56        // 缓存 // 获取ECG数据
57        DATA[j] = EPG_ECG0->DATA & 0xffffffff;
58    }
59    resultV1LA = DATA[3];
60
61    //--- V1-LL
62    Multi_ECG_INN_V1();
63    Multi_ECG_INP_LL();
64    while((EPG_ECG0->INTSTATUS & EPG_ECG_FIFO_TH_INT_Msk) == 0);
65    for(int j = 0; j < 4; j++)
66    {
67        // 缓存 // 获取ECG数据
68        DATA[j] = EPG_ECG0->DATA & 0xffffffff;
69    }
70    resultV1LL = DATA[3];
71
72    //
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100

```

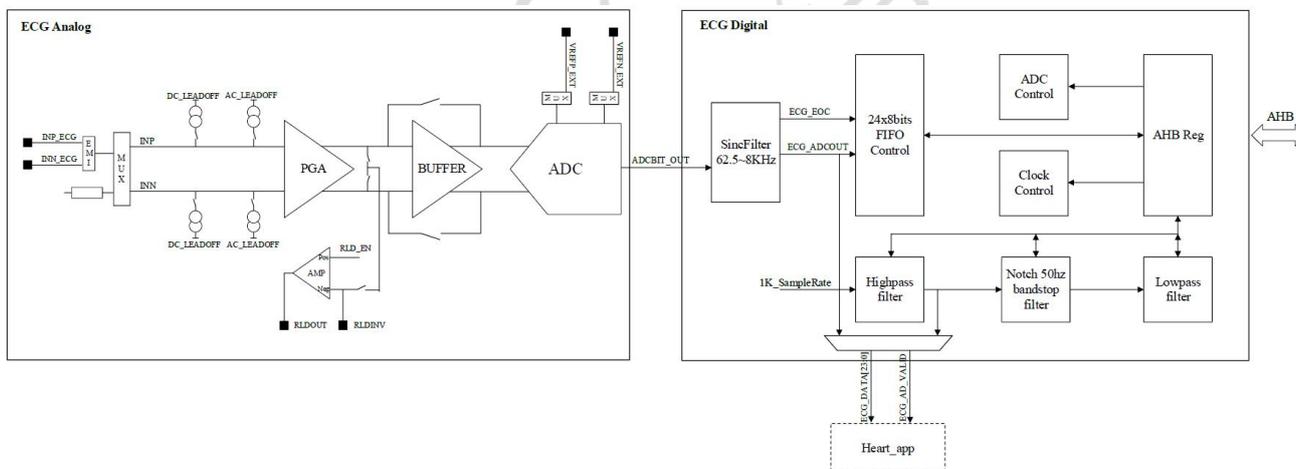
计算：根据上述生物特性，填充数组，并上传到PC。

```

}
resultV1LL = DATA[3];
//=====================================================
*(int*)(data + 0*4) = -(result1); // 用模拟器测试, 此通道大了4倍 1
*(int*)(data + 1*4) = (result2); // 2
*(int*)(data + 2*4) = result3; // 3
*(int*)(data + 3*4) = -(result1 + result2)/2; // AVR
*(int*)(data + 4*4) = (result1 - result2)/2; // AVL
*(int*)(data + 5*4) = (result2 - result1)/2; // AVF
*(int*)(data + 6*4) = (resultV1RA+resultV1LA+resultV1LL)/3; //V1-WCT=V1-(RA+LA+LL)/3
*(int*)(data + 7*4) = 0; // V2
*(int*)(data + 8*4) = 0; // V3
*(int*)(data + 9*4) = 0; // V4
*(int*)(data + 10*4) = 0; // V5
*(int*)(data + 11*4) = 0; // V6
printf_up(2,1,data,sizeof(data)); //上传数据
}
    
```

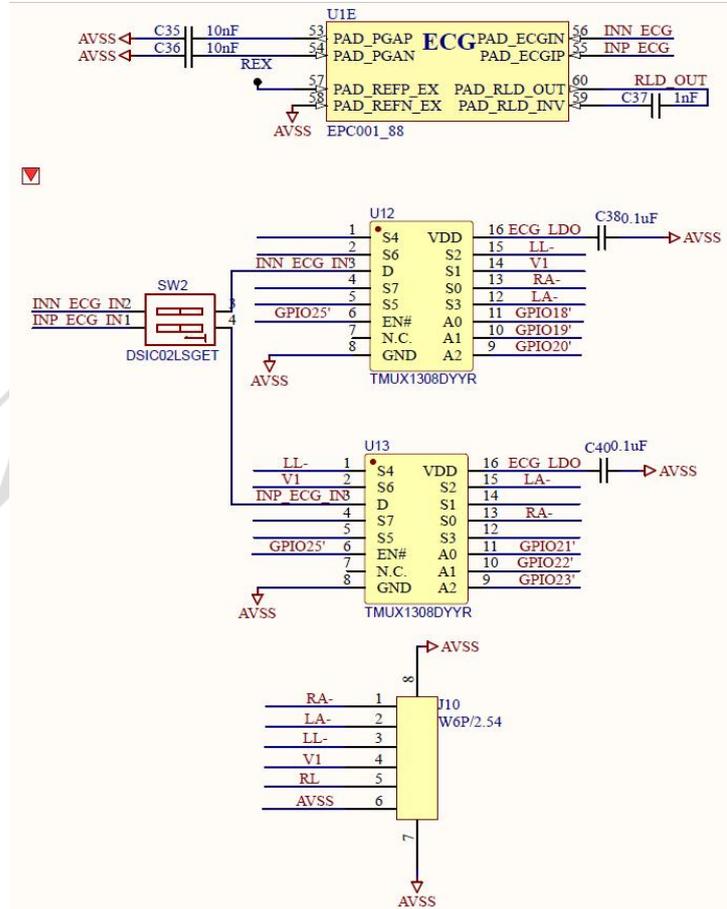
13.6 硬件分析

1) EPC001 内部 ECG 框图

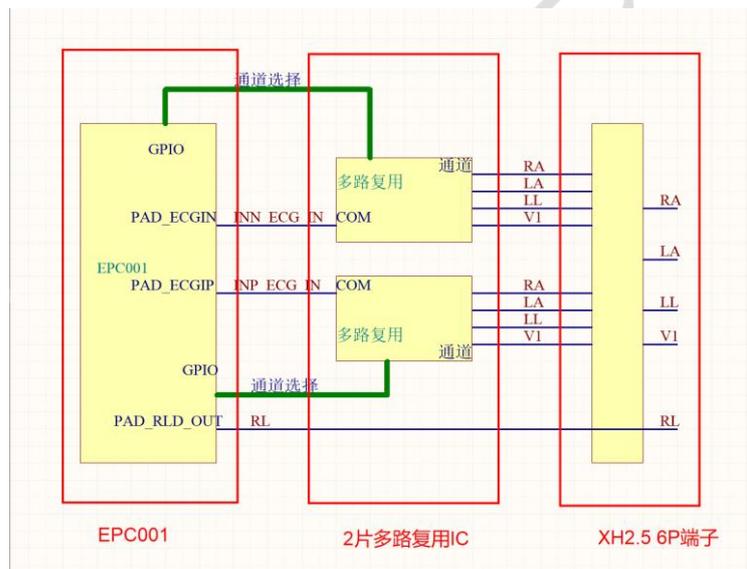


EPC001内部ECG框图

2) 多路复用及输入端口



生命体征检测心电图脉搏开发板五导联电路图



五导联方框图

13.7 导联数据上传格式

SPI转串口，串口的波特率为115200；一包数据60字节。

```

1126 *
1127 *****/
1128 void printf_up(char type1,int res1,char *info1,int size1)
1129 {
1130     //数据上传缓存区
1131     char szbuf[60] = {0};
1132     //头标识缓存区
1133     int *header = (int *)(szbuf);
1134     *header = 0x12345678;
1135     //数据长度
1136     short *size = (short *)(szbuf+4);
1137     *size = size1+3;
1138     //命令
1139     char *type = (char *)(szbuf+6);
1140     *type = type1;
1141     //命令执行状态
1142     char *res = (char *)(szbuf+7);
1143     *res = res1;
1144     //数据
1145     char *info = (char *)(szbuf+8);
1146
1147     //数据搬移至缓存区
1148     for(int i = 0 ; i<size1;i++)
1149         *(info+i) = info1[i];
1150
1151     //校验和缓存区
1152     char *check = (char *)(szbuf+(*size)-1+6);
1153     *check = 0;
1154
1155     //求校验和
1156     for(int i = 0; i<(*size+5); i++)
1157         *check = (*check+szbuf[i])&0xff;
1158
1159     //数据发送
1160     //for(int m = 0 ; m<*size+6;m++)
1161     for(int m = 0; m<60; m++)
1162     {
1163
1164         spi0_irq_occurred=0;
1165         EPG_SPI0->THR = szbuf[m];
1166         while(spi0_irq_occurred==0);
1167     }
1168 }
1169 }
1170

```

该表格是对以上代码的另一种表述

1	Header	0--3	4字节	0x12345678 小端模式（低位在前）
2	Size	4--5	2字节	
3	Type	6	1字节	
4	Res	7	1字节	
5	Into	8	12*4字节(12导联)	每导联数据占一个int(4字节)
6	Check		1字节	校验位
7	补0			补全60个字节

14.I2S 语音播放实例

14.1 硬件设置

- 1) 利用跳线帽，将 J12 位置短接，将画框的拨码开关 1 拨到左边，2 拨到右边；
- 2) 利用电源线将板子和电脑连接起来；
- 3) 在耳机接口的位置插上耳机，如下图所示；
- 4) 短接位置：将 J7 处的 IO28 和 I2S1_CLK、IO29 和 I2S1_WS、IO30 和 I2S1_DAT 用跳线帽短接，如下图所示：



按下复位按键，耳机有“高压”的声音，且LED1会闪烁则功能正常，否则不正常。

14.2 代码分析

实例对应工程文件： ../project-IIS

1) I2S 实始化

- ① 实始化 I2S 中断；
- ② 复位 I2S；
- ③ 初始化 I2S GPIO；
- ④ 初始化 I2S 外设；
- ⑤ 先禁止 I2S 中断。

```

void I2sInit(void)
{
    //I2S中断使能
    ClicInstall(I2S1_Handler,I2S1_IRQn,2);
    //I2S 复位
    EPG_SYSCON0->SWRESET |=(1<<13);
    EPG_SYSCON0->SWRESET &=~(1<<13);
    //实始化I2S GPIO
    EPG_gpio_SetAltFunc_GPIOH16(EPG_GPIO0, 0x15000000);
    EPG_gpio_SetFuncMode_GPIOH16(EPG_GPIO0, 0x2A000000);
    //I2S 设置
    EPG_I2S_CFGR_config(EPG_I2S1,
        0x0e, //波特率真
        0, //DMA
        0, //CLK pol 空闲时的电平 0: 低 1: 高
        1, //WS edge 0: WS在上升沿变化 1: WS在下降沿变化
        0, //SD edge 0: SD 在上升沿变化 1: SD在下降沿变化
        2, //位长 0:16字节 1:24 2:32
        1, //包长 0:16 1:32
        0, //DMA禁止
        1, //收发, 使能
        1); //主, 从机
    DisI2sInt(); //禁止中断
}
    
```

2) I2S 一帧数据

调用一次函数发送一次左右声道数据，数据类型为 32 位。

```

  1  /*
  2  *  函数功能: I2S发送左右声道数据
  3  *  输入参数:
  4  */
  5  u8 I2sSendData(u32 aDatLeft,u32 aDatRight)
  6  {
  7      uint32_t tmp=0;
  8
  9      EPG_I2S1->TDR= aDatLeft;                //发送L声道数据
 10      EPG_I2S1->IER |= EPG_I2S_TXE_INT_Msk;  //使能中断
 11      EPG_I2S1->CR = 0x1;                    //启动发送
 12      while(i2s1_irq_occurred==0);          //等发送完成
 13      EPG_I2S1->TDR = aDatRight;             //发送R声道数据
 14      while(!(EPG_I2S1->INTSTATUS & EPG_I2S_TXE_INT_Msk)); //等发送完成
 15      EPG_I2S1->CR = 0x0;                    //停止发送
 16      i2s1_irq_occurred = 0;                //清标志
 17
 18      return 0;
 19  }
  
```

3) I2S 声音文件数组

```

  29
  30
  31  /*
  32  *  I2S发送语音内容
  33  *  输入参数:  *pdata: 发送的内容指针 (内容为8位采样)
  34  *             alen: 发送的内容长度
  35  */
  36
  37  u8 I2sSendBufNew(const uint8_t *pdata ,u16 alen)
  38  {
  39      u8 tmp=0;
  40      for(u16 i=0;i<alen;i++)                //I2S 发送数据
  41      {
  42          //因为发送的内容为8位采样, I2S数据格式为32位, 所以左称21位
  43          tmp=I2sSendData(*(pdata+i)<<21,0);
  44      }
  45      DisI2sInt();                            //禁止中断
  46  }
  47
  48
  
```



4) 主循环周期调用播放

```
void IisTest(void)
{
    I2sSendBufNew(InBusy, 5444); //调用I2S发送语音内容。
}
}
```

NanoChap 暖芯迦

14.3 I2S 声音文件.c 生成说明

1) 解压 WavToC.rar

2) 启动软件



3) .wav 文件转成数组

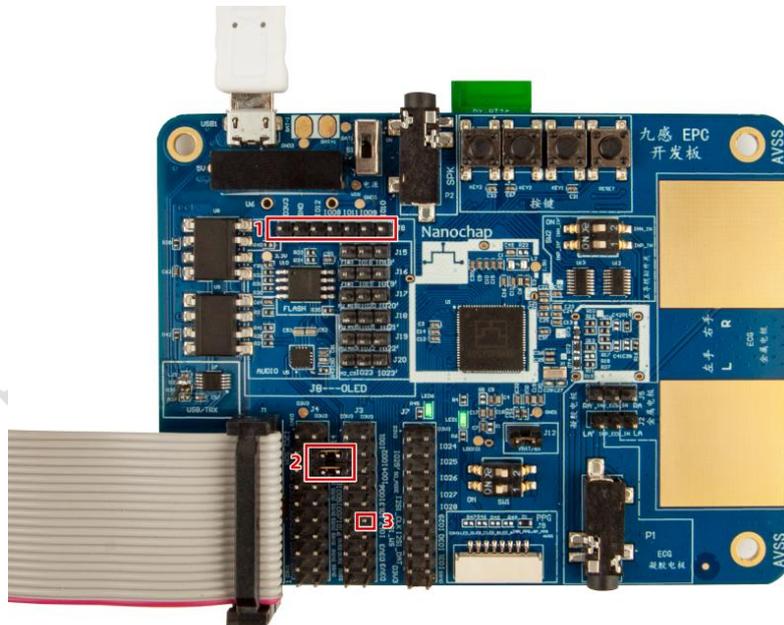


- ① 位置 1: 为 .wav 文件信息，采样速率为 8K，则不会失真；
- ② 位置 2: 要转换成的采样率，因为本实例的 IIS 播放率为 8K，然后根据 3、4、5、6 生成对应文件。

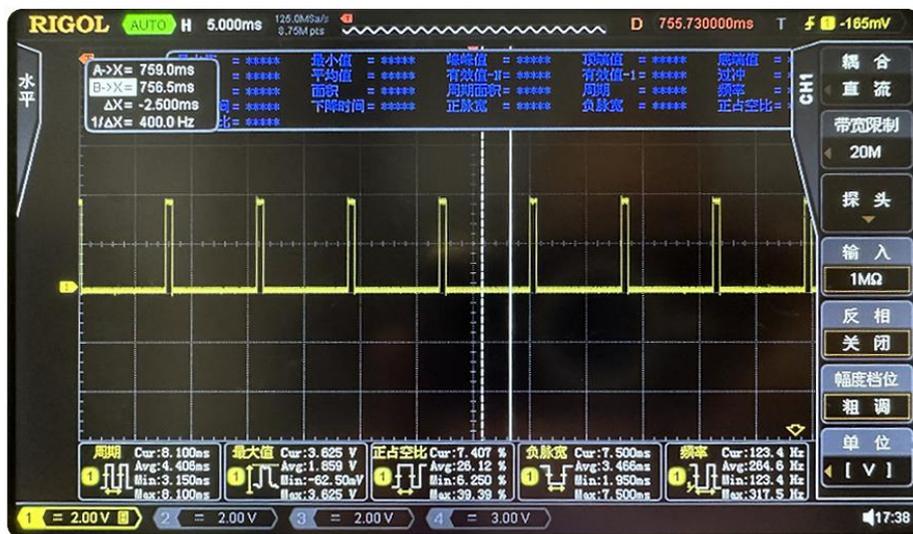
15.PWM 实例

15.1 功能介绍

1) PCB 接线，路线，端口介绍，其中位置 1，2，3 :是 PMW1---6 输出所用到的端口



2) 其中一路 PWM 波形



PWM波形图

15.2 测试效果

如 PWM 波形图所示

15.3 代码分析

实例对应工程文件: ../project-pwm

REV1.1

<https://www.nanochap.cn>

1) GPIO 初始化

```

58 //PWM gpio 初始化
59 static void PwmGpioInit(void)
60 {
61     EPG_gpio_ClrFuncMode_GPIOL16_Pin(EPG_GPIO0, PMW1_PIN, GPIO_MODE_MSK); //清空GPIO 模式配置位
62     EPG_gpio_ClrFuncMode_GPIOL16_Pin(EPG_GPIO0, PMW2_PIN, GPIO_MODE_MSK); //清空GPIO 模式配置位
63     EPG_gpio_ClrFuncMode_GPIOL16_Pin(EPG_GPIO0, PMW3_PIN, GPIO_MODE_MSK); //清空GPIO 模式配置位
64     EPG_gpio_ClrFuncMode_GPIOL16_Pin(EPG_GPIO0, PMW4_PIN, GPIO_MODE_MSK); //清空GPIO 模式配置位
65     EPG_gpio_ClrFuncMode_GPIOL16_Pin(EPG_GPIO0, PMW5_PIN, GPIO_MODE_MSK); //清空GPIO 模式配置位
66     EPG_gpio_ClrFuncMode_GPIOL16_Pin(EPG_GPIO0, PMW6_PIN, GPIO_MODE_MSK); //清空GPIO 模式配置位
67
68     EPG_gpio_ClrAltFunc_GPIOL16_Pin(EPG_GPIO0, PMW1_PIN, GPIO_MODE_MSK); //清空GPIO 功能配置位
69     EPG_gpio_ClrAltFunc_GPIOL16_Pin(EPG_GPIO0, PMW2_PIN, GPIO_MODE_MSK); //清空GPIO 功能配置位
70     EPG_gpio_ClrAltFunc_GPIOL16_Pin(EPG_GPIO0, PMW3_PIN, GPIO_MODE_MSK); //清空GPIO 功能配置位
71     EPG_gpio_ClrAltFunc_GPIOL16_Pin(EPG_GPIO0, PMW4_PIN, GPIO_MODE_MSK); //清空GPIO 功能配置位
72     EPG_gpio_ClrAltFunc_GPIOL16_Pin(EPG_GPIO0, PMW5_PIN, GPIO_MODE_MSK); //清空GPIO 功能配置位
73     EPG_gpio_ClrAltFunc_GPIOL16_Pin(EPG_GPIO0, PMW6_PIN, GPIO_MODE_MSK); //清空GPIO 功能配置位
74
75     //=====
76     EPG_gpio_SetFuncMode_GPIOL16_Pin(EPG_GPIO0, PMW1_PIN, GPIO_MODE_ALTFUN); //GPIO 配置为复用模式
77     EPG_gpio_SetFuncMode_GPIOL16_Pin(EPG_GPIO0, PMW2_PIN, GPIO_MODE_ALTFUN); //GPIO 配置为复用模式
78     EPG_gpio_SetFuncMode_GPIOL16_Pin(EPG_GPIO0, PMW3_PIN, GPIO_MODE_ALTFUN); //GPIO 配置为复用模式
79     EPG_gpio_SetFuncMode_GPIOL16_Pin(EPG_GPIO0, PMW4_PIN, GPIO_MODE_ALTFUN); //GPIO 配置为复用模式
80     EPG_gpio_SetFuncMode_GPIOL16_Pin(EPG_GPIO0, PMW5_PIN, GPIO_MODE_ALTFUN); //GPIO 配置为复用模式
81     EPG_gpio_SetFuncMode_GPIOL16_Pin(EPG_GPIO0, PMW6_PIN, GPIO_MODE_ALTFUN); //GPIO 配置为复用模式
82
83     EPG_gpio_SetAltFunc_GPIOL16_Pin(EPG_GPIO0, PMW1_PIN, GPIO10_PMW1); //复用模式为 PWM
84     EPG_gpio_SetAltFunc_GPIOL16_Pin(EPG_GPIO0, PMW2_PIN, GPIO11_PMW2); //复用模式为 PWM
85     EPG_gpio_SetAltFunc_GPIOL16_Pin(EPG_GPIO0, PMW3_PIN, GPIO12_PMW3); //复用模式为 PWM
86     EPG_gpio_SetAltFunc_GPIOL16_Pin(EPG_GPIO0, PMW4_PIN, GPIO13_PMW4); //复用模式为 PWM
87     EPG_gpio_SetAltFunc_GPIOL16_Pin(EPG_GPIO0, PMW5_PIN, GPIO14_PMW5); //复用模式为 PWM
88     EPG_gpio_SetAltFunc_GPIOL16_Pin(EPG_GPIO0, PMW6_PIN, GPIO15_PMW6); //复用模式为 PWM
89
90 }
  
```

2) PWM 中断初始化

检测到中断标志，进入中断，清除标志位。

```

9
10 static void PWM_Handler(void) __attribute__((interrupt));
11 static void PWM_Handler(void) {
12     //MR0 status interrupt
13     if((EPG_PWM0->INTSTATUS & EPG_PWM_MR0_INT_STS_Msk)==EPG_PWM_MR0_INT_STS_Msk) {
14         EPG_PWM0->INTCLEAR = EPG_PWM_MR0_INT_STS_Msk;//Clear Interrupt
15     }
16     //MR1 status interrupt
17     if((EPG_PWM0->INTSTATUS & EPG_PWM_MR1_INT_STS_Msk)==EPG_PWM_MR1_INT_STS_Msk) {
18         EPG_PWM0->INTCLEAR = EPG_PWM_MR1_INT_STS_Msk;//Clear Interrupt
19     }
20     //MR2 status interrupt
21     if((EPG_PWM0->INTSTATUS & EPG_PWM_MR2_INT_STS_Msk)==EPG_PWM_MR2_INT_STS_Msk) {
22         EPG_PWM0->INTCLEAR = EPG_PWM_MR2_INT_STS_Msk;//Clear Interrupt
23     }
24     //MR3 status interrupt
25     if((EPG_PWM0->INTSTATUS & EPG_PWM_MR3_INT_STS_Msk)==EPG_PWM_MR3_INT_STS_Msk) {
26         EPG_PWM0->INTCLEAR = EPG_PWM_MR3_INT_STS_Msk;//Clear Interrupt
27     }
28     //MR4 status interrupt
29     if((EPG_PWM0->INTSTATUS & EPG_PWM_MR4_INT_STS_Msk)==EPG_PWM_MR4_INT_STS_Msk) {
30         EPG_PWM0->INTCLEAR = EPG_PWM_MR4_INT_STS_Msk;//Clear Interrupt
31     }
32     //MR5 status interrupt
33     if((EPG_PWM0->INTSTATUS & EPG_PWM_MR5_INT_STS_Msk)==EPG_PWM_MR5_INT_STS_Msk) {
34         EPG_PWM0->INTCLEAR = EPG_PWM_MR5_INT_STS_Msk;//Clear Interrupt
35     }
36     //MR6 status interrupt
37     if((EPG_PWM0->INTSTATUS & EPG_PWM_MR6_INT_STS_Msk)==EPG_PWM_MR6_INT_STS_Msk) {
38         EPG_PWM0->INTCLEAR = EPG_PWM_MR6_INT_STS_Msk;//Clear Interrupt
39     }
40     return;
41 }
  
```

3) PWM 初始化

```

88 }
89 void PwmMacInit(EPG_PWM_TypeDef *EPG_PWM)
90 {
91     PwmGpioInit(); //GPIO 初始化
92
93     ClicInstall(Pwm_Handler,PWM_IRQn,4); //中断初始化
94     //使能通道 //通道输出使能
95     EPG_PWM->PCR = (EPG_PWM_CTRL_PWM1_EN_Msk
96     | EPG_PWM_CTRL_PWM2_EN_Msk
97     | EPG_PWM_CTRL_PWM3_EN_Msk
98     | EPG_PWM_CTRL_PWM4_EN_Msk
99     | EPG_PWM_CTRL_PWM5_EN_Msk
100    | EPG_PWM_CTRL_PWM6_EN_Msk
101    | EPG_PWM_CTRL_PWM2_SEL_Msk
102    | EPG_PWM_CTRL_PWM4_SEL_Msk
103    | EPG_PWM_CTRL_PWM6_SEL_Msk);
104
105    //设置MR值
106    EPG_PWM->MR0 = 0x00000500; //周期
107    EPG_PWM->MR1 = 0x00000000; //PWM1 占空比
108    EPG_PWM->MR2 = 0x00000040; //PWM2 占空比
109    EPG_PWM->MR3 = 0x00000020; //PWM3 占空比
110    EPG_PWM->MR4 = 0x00000080; //PWM4 占空比
111    EPG_PWM->MR5 = 0x00000040; //PWM5 占空比
112    EPG_PWM->MR6 = 0x00000100; //PWM6 占空比
113
114    //设置MCR
115    EPG_PWM->MCR |= EPG_PWM_MATCH_MR0_RET_Msk; //PWM 占空比
116    //分频设置
117    EPG_PWM->PR = 0x00000064;
118    //LOAD 使能
119    EPG_PWM->LER = (EPG_PWM_LOAD_ML6_EN_Msk
120    | EPG_PWM_LOAD_ML5_EN_Msk
121    | EPG_PWM_LOAD_ML4_EN_Msk
122    | EPG_PWM_LOAD_ML3_EN_Msk
123    | EPG_PWM_LOAD_ML2_EN_Msk
124    | EPG_PWM_LOAD_ML1_EN_Msk
125    | EPG_PWM_LOAD_ML0_EN_Msk); //0x0000007f;
126
127    //计数使能
128    EPG_PWM->TCR |= EPG_PWM_TIMER_CNT_EN_Msk;
129    //中断使能控制
130    EPG_PWM->MCR |= (EPG_PWM_MATCH_MR0_INT_Msk
131    | EPG_PWM_MATCH_MR1_INT_Msk
132    | EPG_PWM_MATCH_MR2_INT_Msk
133    | EPG_PWM_MATCH_MR3_INT_Msk
134    | EPG_PWM_MATCH_MR4_INT_Msk
135    | EPG_PWM_MATCH_MR5_INT_Msk
136    | EPG_PWM_MATCH_MR6_INT_Msk);
137
138 }

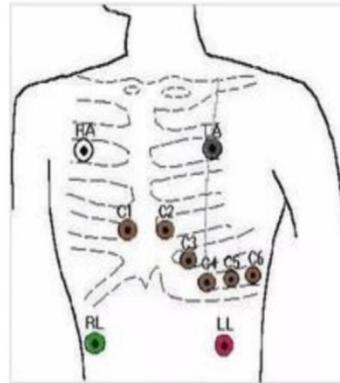
```

附录

1. 对应电极片位置

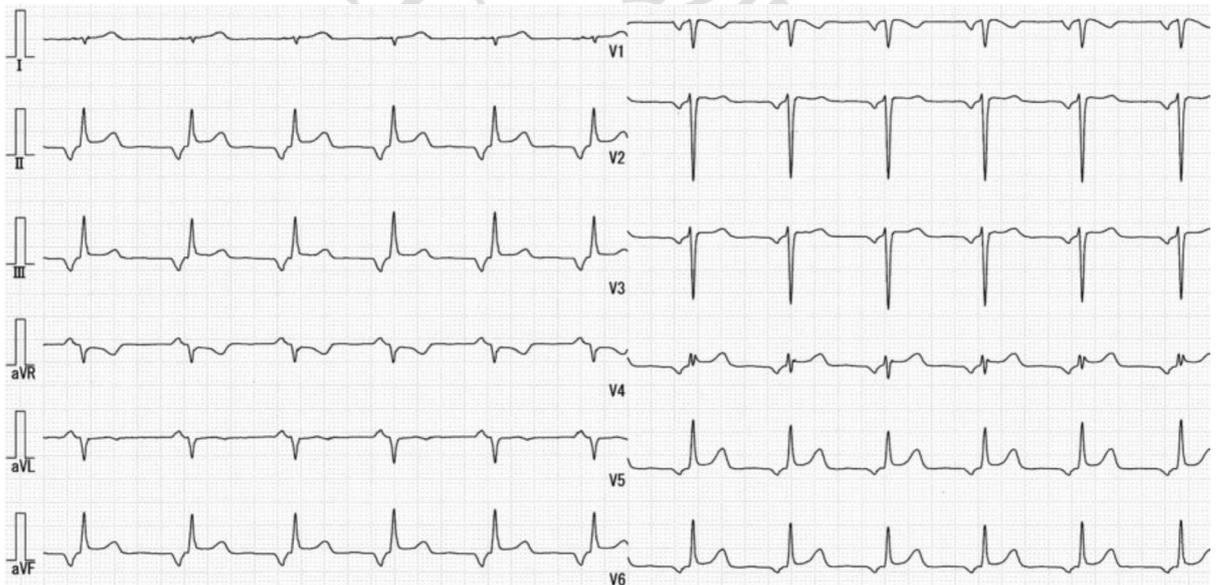
五导联的心电监护仪电极片放置位置

部位	标号	颜色	放置部位
右上	RA	白	胸骨右缘锁骨中线第一肋间
左上	LA	黑	胸骨左缘锁骨中线第一肋间
中间	C	棕	胸骨左缘第四肋间
左下	LL	红	左锁骨中线肋缘处
右下	RL	绿	右锁骨中线肋缘处



对应电极片位置图

2. 正常心电图



正常心电图

16.联系方式

可通过以下方式了解更多产品详情：

1) 公司电话：4008605922 ; 180 9470 6680

2) 技术人员QQ：1708154204



3) 公众号：暖芯迦电子



Copyright©2023 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<https://www.nanochap.cn>或与我们直接联系（4008605922）。