版本：V1.0

Nanochap

杭州暖芯迦电子科技有限公司

EPC1EVK-EEG

生命体征检测脑电开发板_使用手册

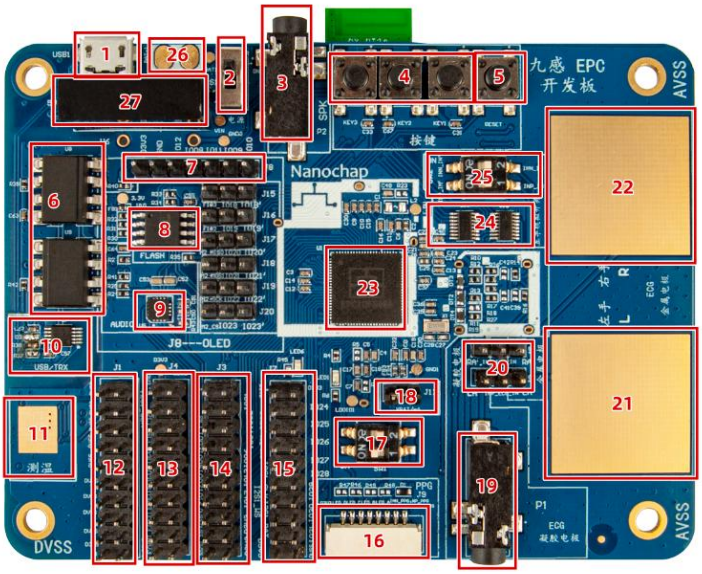
文档修订记录

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	2023-07-14	创建文档				

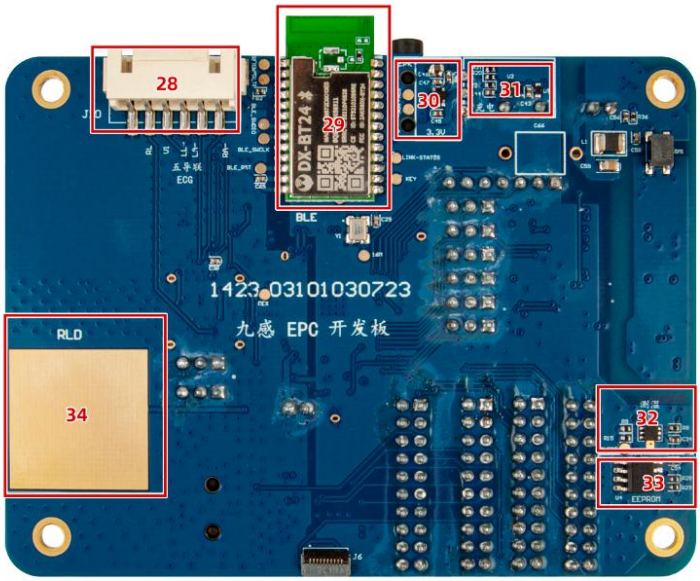
目录

1. 简介

1.1 生命体征检测脑电开发板



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		复位向里（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；

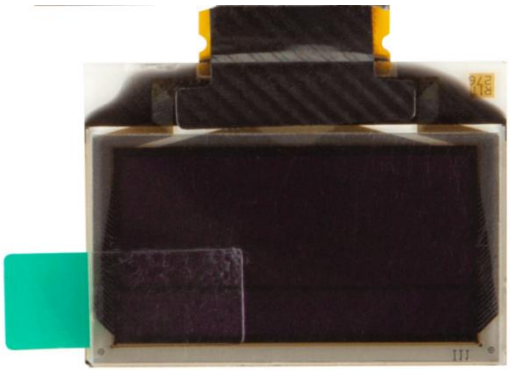
31) BATT 充电IC；

32) 温度IC TMP117；

33) AT24C128, 128K位串行EEPROM；

34) ECG RLD右腿驱动电极；

1.2 OLED 小板

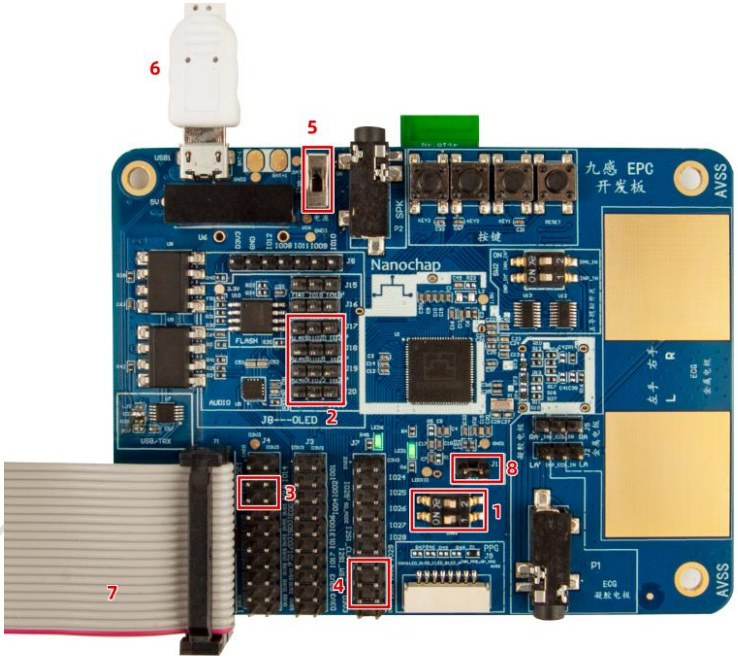


OLED 小板图

1.3 EEG导联线

EEG导联线图

1.4 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，赋给重装载寄存器
54     EPG_TIMER->VALUE = aCycl;                    //设置计时器
55
56     EPG_TIMER->INTCLEAR |= (0x1<<0);             //清中断标志
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
61
62     //==================================
63
64
65
66     ClicInstall(Timer0_Handler,TIMER0_IRQn,3);  //启动中断
67
68 }
```

1

2

3

4

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
32
33     SoftTimerScan();                           //软件定时组自减扫描
34     return;
35 }
36
```

1

2

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) 软件定器

```
enum
{
    TIMER_UART0=0,                  //串口0 接收空闲倒计时
    TIMER_UART1,                    //串口1 接收空闲倒计时
    TIMER_OLED,                     //OLED
    TIMER_IIC,                      //IIC
    TIMER_IIS,                      //IIS

                                    //添加定时器编号

    TIMER_CNT_MAX,                  //软定时器个数
};

extern uint32_t   Timing[];

#define TIMER_IS_ZERO(inx)      (Timing[inx]==0)    //定时器到
#define TIMER_GET_MS(inx)       Timing[inx]         //获取定时器值
#define TIMER_SET(inx,ms)       Timing[inx]=ms      //设置定时器值
#define TIMER_CHEEK(inx)        Timing[inx]         //检查定时器值
```

1

2

3

4

① 已用软定时器编著号；

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

```
typedef struct
{
    bool m1ms;              //1ms 标志
    bool m10ms;             //10ms 标志
    bool m100ms;            //100ms 标志
    bool m500ms;            //500ms 标志
    bool m1s;               //1S   标志
    bool m5s;               //5S   标志
    bool m10s;              //10s 标志
}sSysTimeFalgType;
```

```
void UserMain(void)
{
    static u32 cnt=0;
    printf("main start...\n");
    while(true)
    {
        SysTickPro();          //系统节拍处理          1
        if(gSysTime.m100ms) //100ms 节拍处理
        {
            //添加你的功能
        }

        if(gSysTime.m500ms) //500ms 节拍处理
        {
                //添加你的功能
        }
        if(gSysTime.m1s)     //周期打印 hello world
        {
            cnt++;
            printf("hello world cnt=%d...\n",cnt);
        }

        Uart1Pro();          //串口1接收指令解析

    }
}
```

2

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)

```
void Uart0MacInit(void)
{
    EPG_gpio_SetAltFunc_GPIOL16_Pin(EPG_GPIO0,UART0_RX_PIN,GPIO4_UART0_RXD);      //ALT FUN 功能为 RXD
    EPG_gpio_SetAltFunc_GPIOL16_Pin(EPG_GPIO0,UART0_TX_PIN,GPIO7_UART0_TXD);      //ALT FUN功能为 TXD
    EPG_gpio_SetFuncMode_GPIOL16_Pin(EPG_GPIO0,UART0_RX_PIN,GPIO_MODE_ALTFUN);    //GPIO配置为 ALTFUN功能
    EPG_gpio_SetFuncMode_GPIOL16_Pin(EPG_GPIO0,UART0_TX_PIN,GPIO_MODE_ALTFUN);    //GPIO配置为 ALTFUN功能

    EPG_UART0->FCR = EPG_UART_FCR_FIFO_EN_Msk;          // 使能FIFO
    EPG_UART0->DLL = 8;                                 // 配置波特率

    //HW8_REG(REG_FCR_ADDR) = 0x81;     // rx_fifo_th 8 bytes
    EPG_UART0->FCR = 0x81;              //
    //Enable Data ready interrupt
    EPG_UART0->IER |= EPG_UART_IER_RDAI_EN_Msk;     //使能接收中断


    ClicInstall(UART0_Handler,UART0_IRQn,1);        //启动中断
}
```

1 2 3 4

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 2 3 4

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))               //接收倒计到，一组数据接收结束
99     )                                                                    1
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，          2
103
104        URxEvent = false;                                               //清 接收事件标志
105        Uart0BufInit();                                                 //清 接收缓存         3
106    }
107 }
108
109
```

① 判断有接收事件并接收结束；

② 处理接收数据；

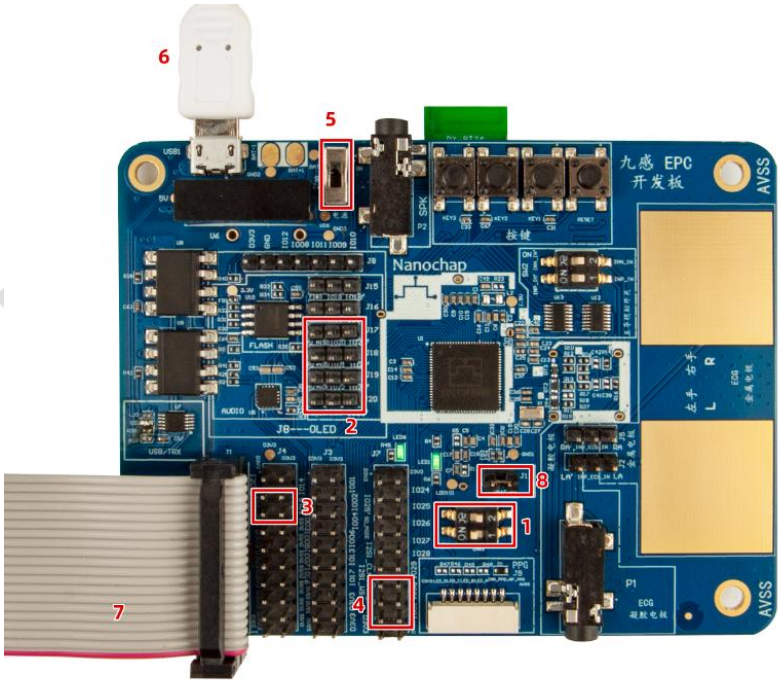
③ 清空接收缓存。

5) 测试效果

实例对应工程文件：../project-hello_world

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AT24CXX 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_GPIOL16_Pin(EPG_GPIO0, SDA_PIN, GPIO_MODE_MSK);     //清空GPIO 模式配置位
81     EPG_gpio_ClrFuncMode_GPIOL16_Pin(EPG_GPIO0, SCL_PIN, GPIO_MODE_MSK);
82
83     EPG_gpio_ClrAltFunc_GPIOH16_Pin(EPG_GPIO0, SDA_PIN, GPIO_MODE_MSK);      //清空GPIO 功能配置位
84     EPG_gpio_ClrAltFunc_GPIOH16_Pin(EPG_GPIO0, SCL_PIN, GPIO_MODE_MSK);
85
86
87     EPG_gpio_SetFuncMode_GPIOL16(EPG_GPIO0, 0x00000000);                      1
88     EPG_gpio_SetAltFunc_GPIOL16(EPG_GPIO0,  0x00000000);
89
90     EPG_gpio_SetFuncMode_GPIOL16_Pin(EPG_GPIO0, SCL_PIN, GPIO_MODE_OUTPUT);  // 配置GPOIO 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
```

① 配置 GPIO 为普通功能；

② 设置 GPIO 上拉。

2) EpcIicInit()

```
153
154 //模拟IIC实始化
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;
162     }
163     //==
164     printf("SIM IIC init\n");
165
166
167
168     Iic1GpioInit();          //GPIO 初始化
169
170     //模拟IIC SCL SDA 设置
171     memset((void*)&gSimIic,0,sizeof(sSimIicType));
172     gSimIic.SDA_IN=simSdaIn;
173     gSimIic.SDA_OUT=simSdaOut;
174     gSimIic.Clr_IIC_SCL=simClrScl;
175     gSimIic.Clr_IIC_SDA=simClrSda;
176     gSimIic.READ_SDA=simReadSda;
177     gSimIic.Set_IIC_SCL=simSetScl;
178     gSimIic.Set_IIC_SDA=simSetSda;
179
180
181
182     memset((void*)&sgI2cApi,0,sizeof(IicMacApiType));
183     //初始化模拟IIC 总线
184     simIicMacInit(&gSimIic,1,&sgI2cApi);
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()

```
148 //==========================
149 //模拟IIC 总线初始化
150 void simIicMacInit(sSimIicType *aSimIic,u16 aDelay,IicMacApiType *pApi)
151 {
152
153     sDelayTime =aDelay;          //速度参数设置，软件延时时间
154     //sDeviceAdd =DeviceAdd;
155     pSimIic = aSimIic;
156
157     //总线API 回调设置
158
159     pApi->mAdd16_RBuf =simI2C_ReadBuffer;
160     pApi->mAdd16_RByte =simI2C_ReadByte;
161     pApi->mAdd16_WBuf =simI2C_WriteBuffer;
162     pApi->mAdd16_WByte =simI2C_WriteByte;
163
164
165     pApi->mAdd8_WByte =simI2C_Add8_WriteByte;    //寄存器地址为8字节
166     pApi->mAdd8_RByte =simI2C_Add8_ReadByte;    //寄存器地址为8字节
167
168     pApi->mAdd8_WBuf =simI2C_Add8_WriteBuffer;    //寄存器地址为8字节
169     pApi->mAdd8_RBuf =simI2C_Add8_ReadBuffer;    //寄存器地址为8字节
170 }
171
```

① 模拟 IIC 速度设置，主要影响摸拟时序时，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);
tinc=At24c2xxReadU32(0x4500);                    //读写U32
inc = tinc+1;                                    4

At24cxxSaveU16(0x4700,inc1);
tinc1=At24c2xxReadU16(0x4700);                   //读写u16
inc1 = tinc1+1;                                  5

At24cxxSaveU8(0x4800,inc2);
tinc2=At24c2xxReadU8(0x4800);                    //读写u8
inc2 = tinc2+1;                                  6

//============================================================

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；

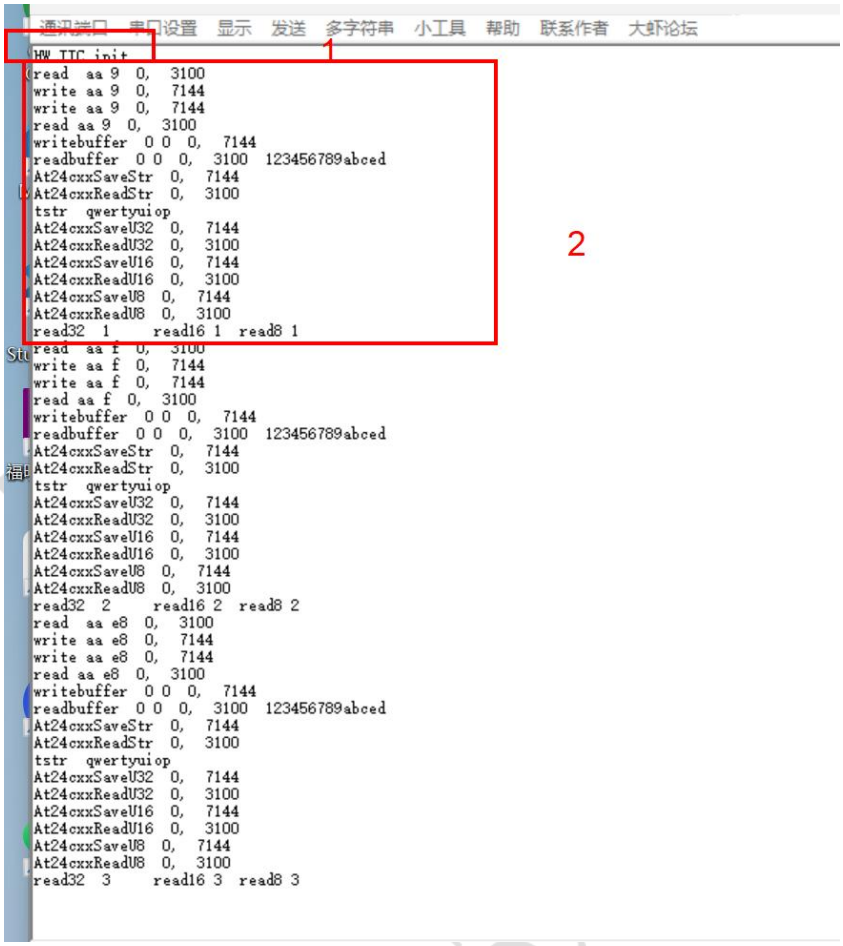
⑦ 读写字符串。

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



1) **模拟 IIC 实始化**
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
#include "../common/fir.h"
#include <stdint.h>
#include <string.h>
#include "../common/myiomap.h"

#include "UserSys.h"

static IicMacApiType sgI2cApi;

#define HW_IIC                  //如果使用硬件IIC

#define SDA_PIN     14
#define SCL_PIN     15


#ifdef HW_IIC


//=======================================
//IIC GPIO 初始化
static void Iic1GpioInit(void)
```

1

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);

                                                                                1
    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

    //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);                                          1
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;                             2
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           3
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               4
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
```

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          1
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 //============================
2  //向AT24CXX 指定位置写入一个16位的值
3  //输入参数：addr:写入的位置
4  //        val:要写入的值
5 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)
123 //============================
124  //向AT24CXX 指定位置写入一个字符串
125  //输入参数：addr:写入的位置
126  //        str:要写入的字符串
127 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");

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。

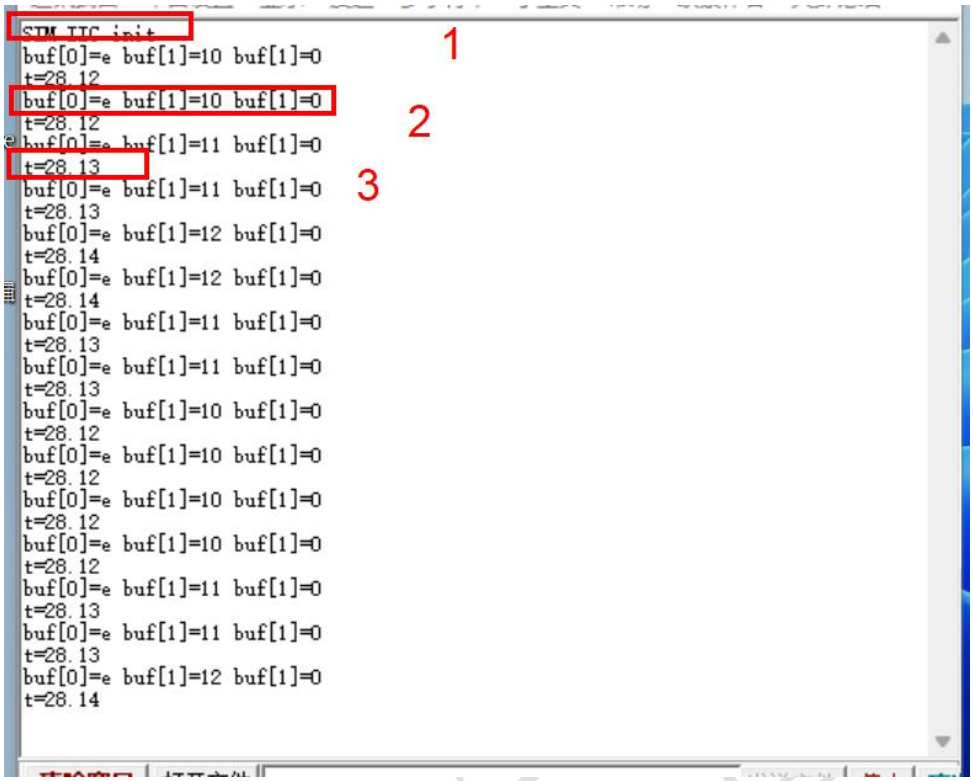
5. 硬件 IIC 读写 TMP117 温度传感器

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



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
97      Timer0MacInit(EPG_TIMER0,TICK_1MS);          //定时器0 初始化
98
99      Uart1MacInit(UART_BPS_115200);               //初始化串口1
00      Tmp117Register();                            //注册 TMP117
01  }
02
```

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
51     TMP117_Initialization_DEFAULT(TMP117_Handler);   //实始化TMP117 寄存器     2
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
        {
        }
```

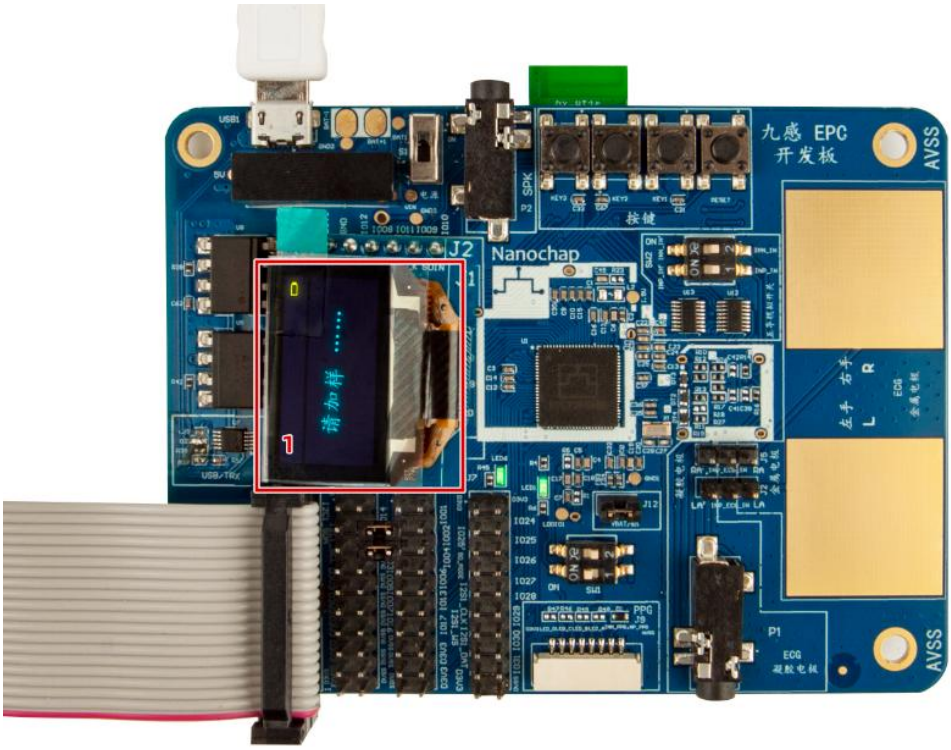
3) TMP117 底层读写操作

```
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                                                                          2
55       p->mAdd8_WBuf(TMP117_DeviceID,TMP117_ConfigurationRegister,buf,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                                              1
    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

```
*/
    #define  OLED_CS      12     //OLED片选
    #define  OLED_RST     8      //OLED复位
    #define  OLED_SDA     10     //OLED数据
    #define  OLED_SCL     9      //OLED时钟
    #define  OLED_DC      11     //OLED命令/数据选择 H=命令 L=数据

//#endif
// OLED GPIO 电平设置宏定义
#define OLED_SDA_SET        EPG_gpio_SetData_ToPad_Pin(EPG_GPIO0, OLED_SDA)   //;nrf_
#define OLED_SDA_CLEAR      EPG_gpio_ClrData_ToPad_Pin(EPG_GPIO0, OLED_SDA)   //nrf_g

#define OLED_DC_SET         EPG_gpio_SetData_ToPad_Pin(EPG_GPIO0, OLED_DC)    //nrf_gp
#define OLED_DC_CLEAR       EPG_gpio_ClrData_ToPad_Pin(EPG_GPIO0, OLED_DC)    //nrf_gp

#define OLED_SCL_SET        EPG_gpio_SetData_ToPad_Pin(EPG_GPIO0, OLED_SCL)   //nrf_g
#define OLED_SCL_CLEAR      EPG_gpio_ClrData_ToPad_Pin(EPG_GPIO0, OLED_SCL)   //nrf_g

#define OLED_CS_SET         EPG_gpio_SetData_ToPad_Pin(EPG_GPIO0, OLED_CS)    //nrf_gp
#define OLED_CS_CLEAR       EPG_gpio_ClrData_ToPad_Pin(EPG_GPIO0, OLED_CS)    //nrf_gp

#define OLED_RST_SET        EPG_gpio_SetData_ToPad_Pin(EPG_GPIO0, OLED_RST)   //nrf_g
#define OLED_RST_CLEAR      EPG_gpio_ClrData_ToPad_Pin(EPG_GPIO0, OLED_RST)   //nrf_g
```

① 定义 GPIO 端口号；

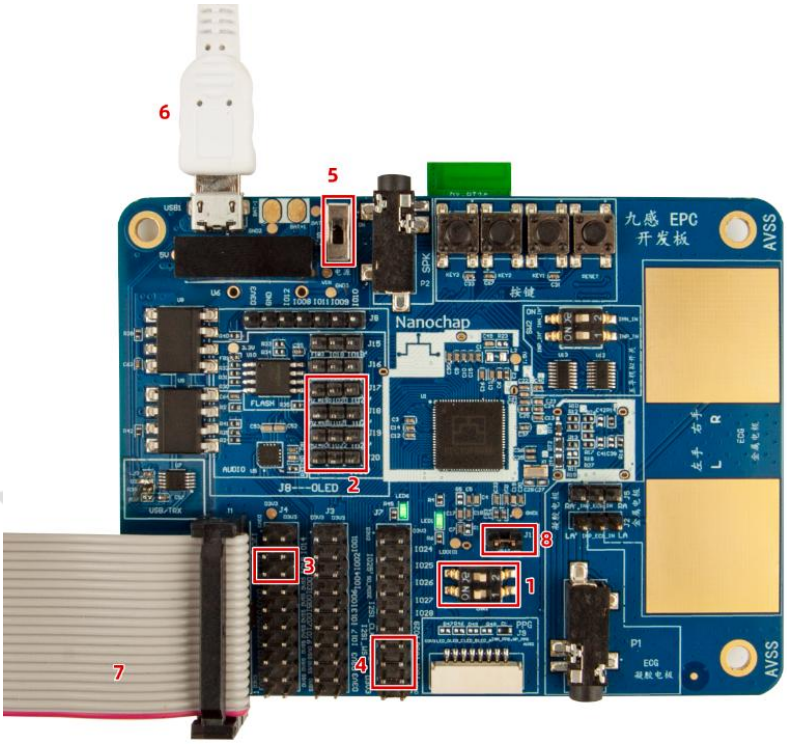
② 定义 GPIO 电平设置宏；

测试代码:

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

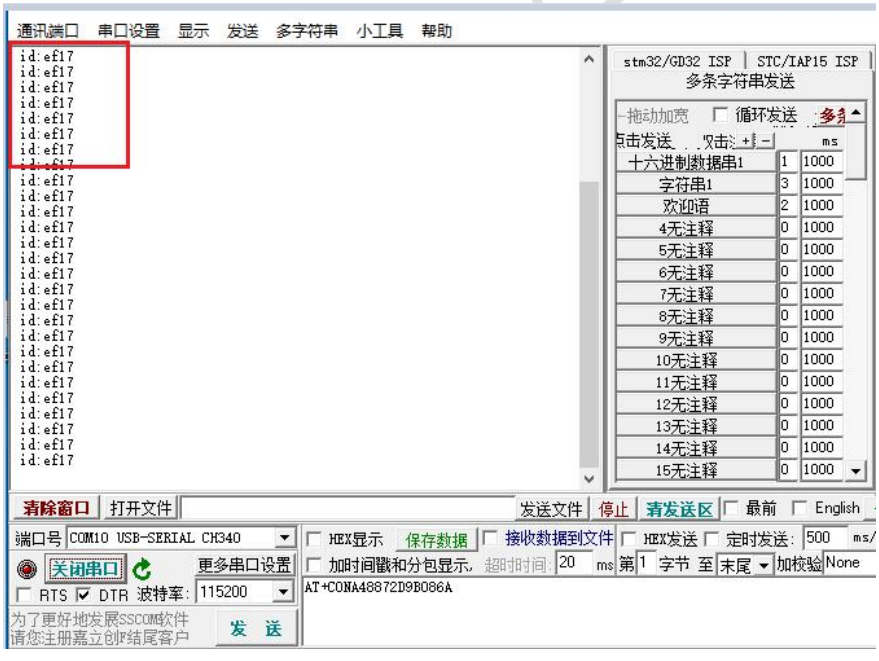
7. 硬件 SPI 读取 W25Q128 ID

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

```
//硬件SPI初始化
void SpiMacInit(EPG_SPI_TypeDef *EPG_SPI)
{

    EPG_SPI_CTRL2_config(EPG_SPI,
            0x0,                    //0x3,          //T2C
            0x0,                    //0x3,          //C2T
            0x1,                    //0x0,  //0x1,          //NSS1
            0x1,                    //0x3,//0x1,            //SAMP PHASE
            0x0,                    //tx dam 禁止
            0x0,                    //rx dma 禁止
            0x7);                   //8 bit charlength

    uint32_t new_reg_ctrl = 0;

    new_reg_ctrl |= 0x07 << EPG_SPI_BAUD_RATE_Pos;    //速度
    new_reg_ctrl &=~EPG_SPI_LSB_SEL_Msk;              //MSB

    new_reg_ctrl |=EPG_SPI_NSS_TOGGLE_Msk;  //
    new_reg_ctrl |=EPG_SPI_NSS_MST_CTRL_Msk;//硬件NSS, new_reg_ctrl &=~EPG_SPI_NSS_MST_CTRL_Msk;

    new_reg_ctrl |=EPG_SPI_CPOL_Msk;        //极性
    new_reg_ctrl |=EPG_SPI_CPHA_Msk;        //相位
    new_reg_ctrl |=EPG_SPI_MST_SLV_SEL_Msk; //mst spi
    new_reg_ctrl |=EPG_SPI_EN_Msk;          //enable spi


    EPG_SPI->CTRL1 = new_reg_ctrl;
}
```

1) **SPI 控制寄存器 2 初始化**

2) **SPI 控制寄存器 1 初始化**

```
/*
函数功能：SPI 总线传输一字符数据
输入参数：*EPG_SPI:SPI 结构体 指针，指向具体SPI
        dat:要发送的数据
输出参数： SPI接收到的数据

*/
uint8_t  SPI_ReadWriteOneByte(EPG_SPI_TypeDef *EPG_SPI,uint8_t dat)
{
    uint16_t tcnt=0;
    //等发送为空
    while(((EPG_SPI->INTSTATUS & EPG_SPI_TXE_INT_STS_Msk)>>EPG_SPI_TXE_INT_STS_Pos)==0)
    {
        tcnt++;
        if(65535<=tcnt) return 0XA5;
    };
    //发送字节写入发送寄存器
    EPG_SPI->THR = dat;

    tcnt=0;

    //while(((EPG_SPI->INTSTATUS & EPG_SPI_RXNE_INT_STS_Msk)>>EPG_SPI_RXNE_INT_STS_Pos)==0)
    //等发送完成
    while((EPG_SPI->FSR & EPG_SPI_BUSY_Msk) >> EPG_SPI_BUSY_Pos)
    {
        tcnt++;
        if(65535<=tcnt) return 0xA6;
    };
    return EPG_SPI->RBR;    //返回接收到的数据
}
```

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；

```
//W25QXX    初始化
void W25QXX_Init(void)
{
    Spi2GpioInit();                 //SPI2  GPIO初始化
    SpiMacInit(EPG_SPI2);           //实始化硬件SPI2
}
```

1

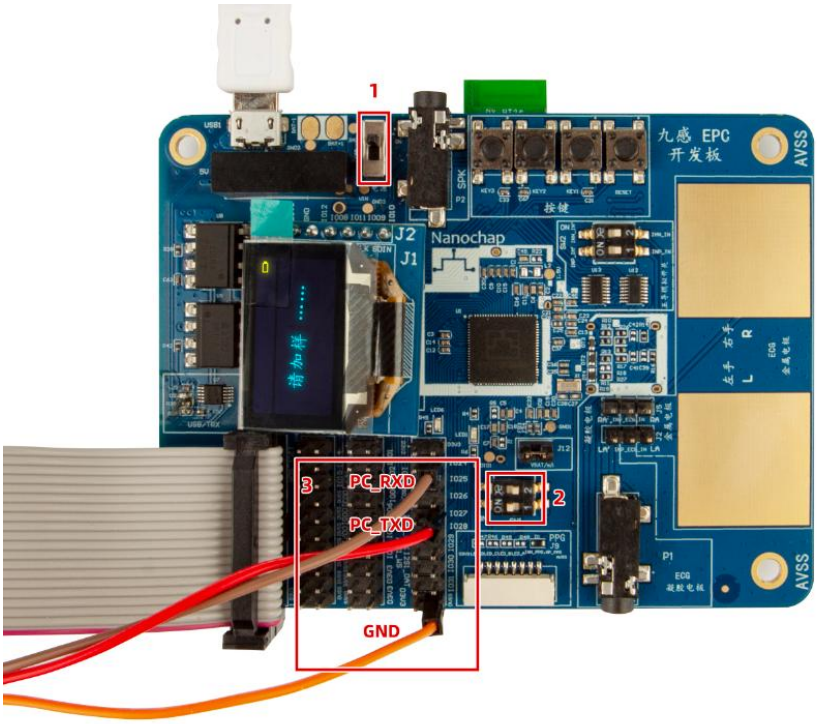
② 读 W25QXX 芯片 ID。

```
/*
  * 函数功能：读W25QXX ID
  */
u16 W25QXX_ReadID(void)
{
    u16 Temp = 0;
    //Spi2CsLow();//W25QXX_CS=0;
    SPI2_ReadWriteByte(0x90);           //发送读取ID命令
    SPI2_ReadWriteByte(0x00);
    SPI2_ReadWriteByte(0x00);
    SPI2_ReadWriteByte(0x00);
    Temp|=SPI2_ReadWriteByte(0xFF)<<8;
    Temp|=SPI2_ReadWriteByte(0xFF);
    //Spi2CsHigh();//W25QXX_CS=1;
    return Temp;
}
```

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。

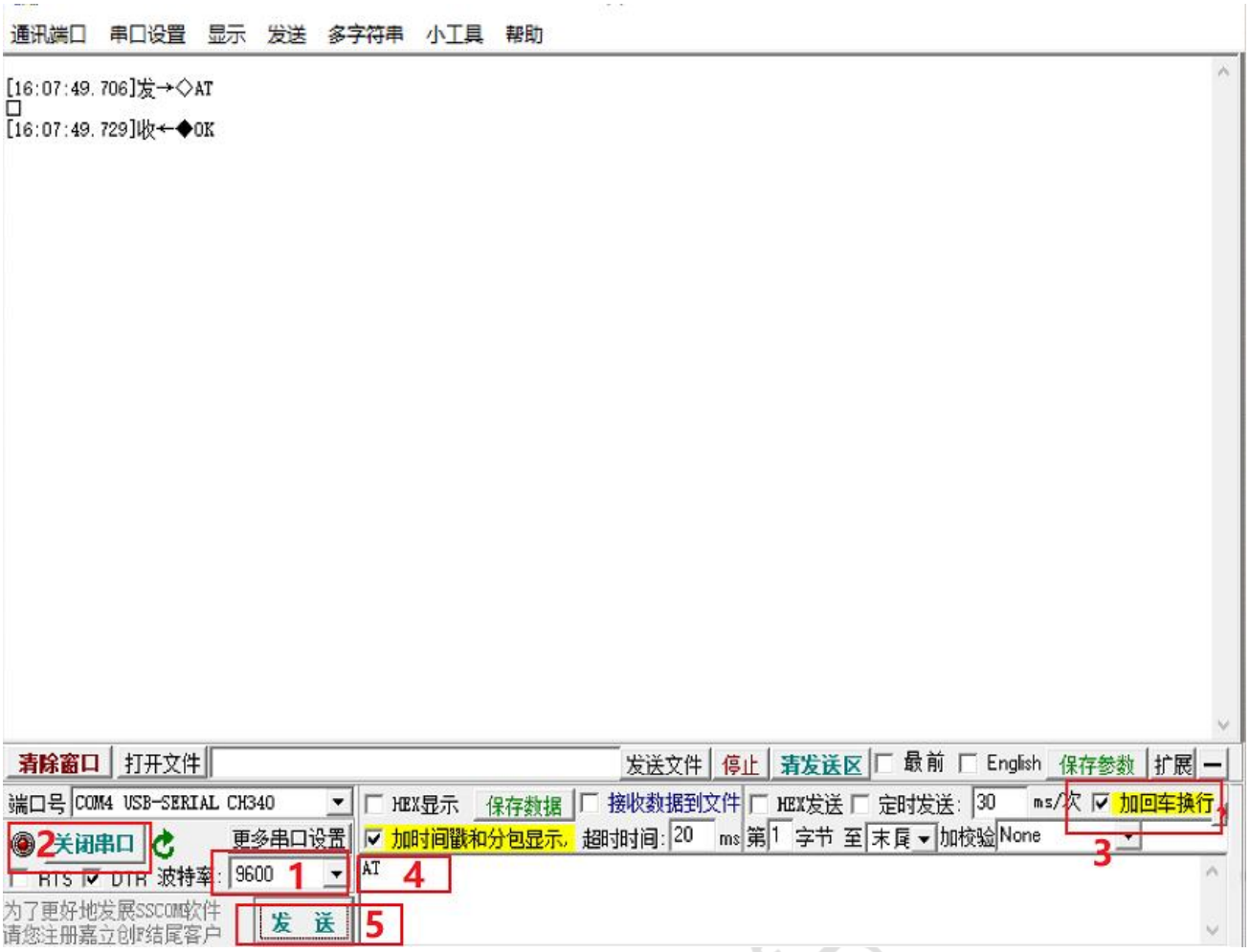
8.1 接线方式



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

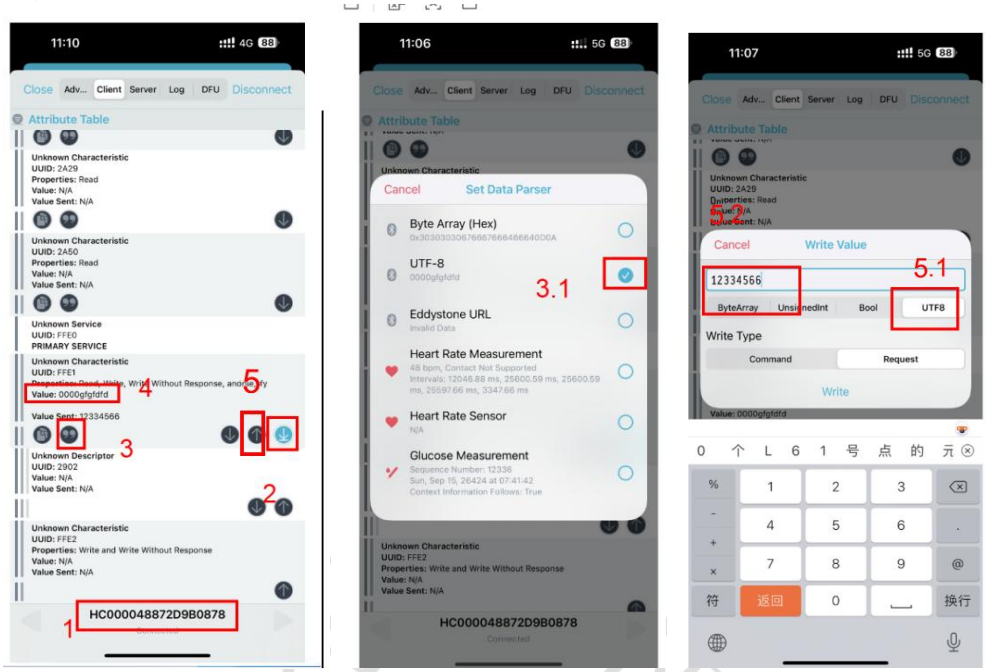
① 上电，并查询 BLE 处理透传开启状态；

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

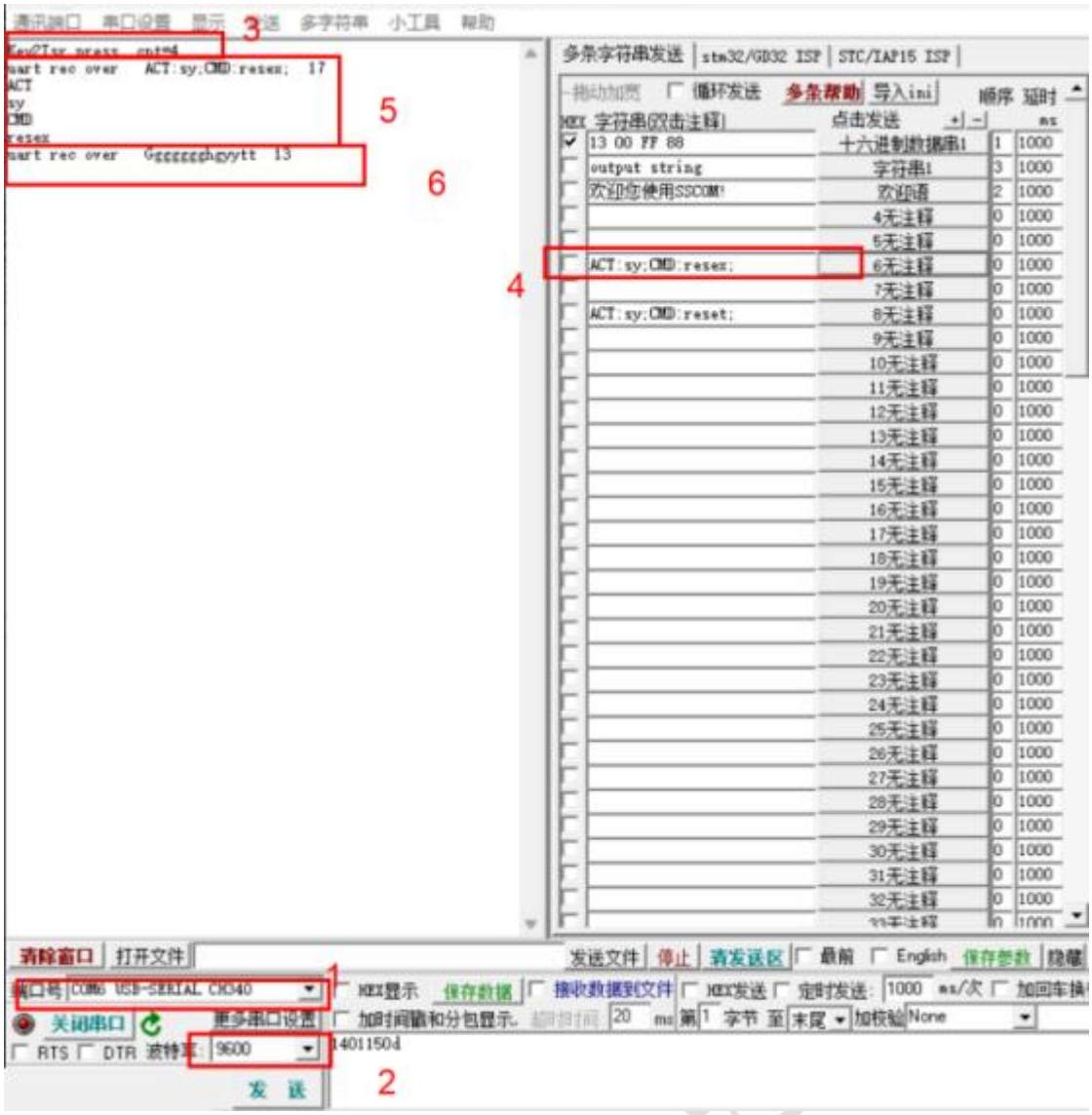
9.2 准备工作

1) 开发板插入 Micro USB 线 ,并接入 PC，打开电源

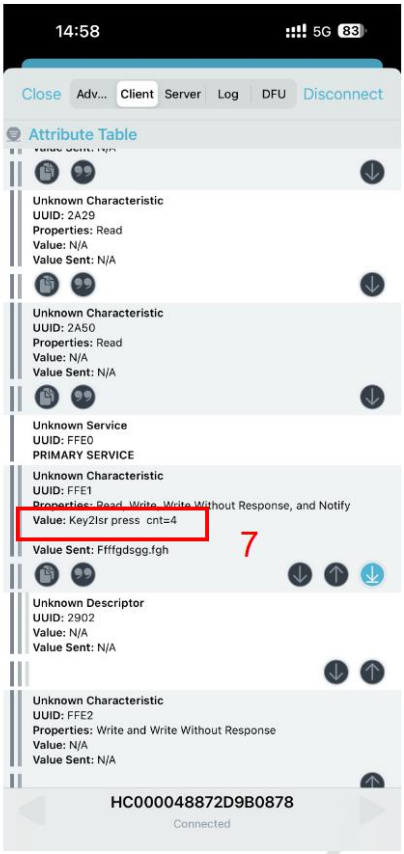
2) 下载 DEMO 程序 ../project-BLE

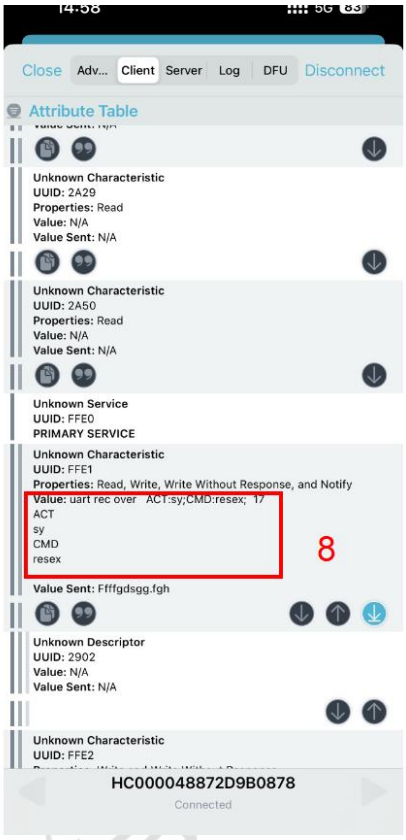
3) PC 打开串口调试助手，波特率设置为 9600，手机打开 nRF Connect

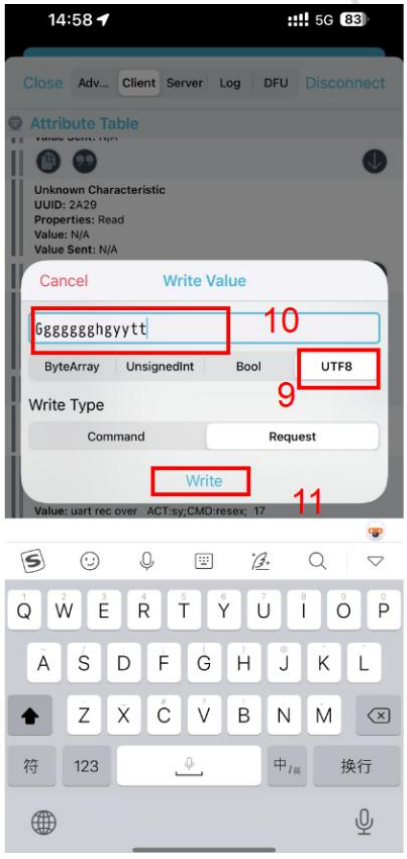
9.3 测试

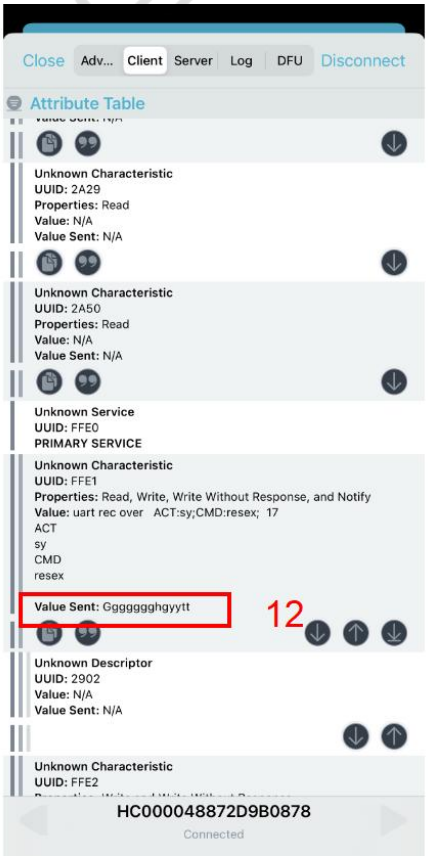


1) 选择正确的串口
2) 波特率设置为 9600
3) 当按下 PCB 上 KEY2 事产生按键事件
4) PC 发送指令给 EPC001
5) 解析出的 PC 端指令
6) 接收到 BLE 字符串


14:58
5G 83
Close
Adv...
Client
Server
Log
DFU
Disconnect
Attribute Table
Unknown Characteristic
UUID: 2A29
Properties: Read
Value: N/A
Value Sent: N/A
Unknown Characteristic
UUID: 2A50
Properties: Read
Value: N/A
Value Sent: N/A
Unknown Service
UUID: FFE0
PRIMARY SERVICE
Unknown Characteristic
UUID: FFE1
Properties: Read, Write, Write Without Response, and Notify
Value: Key2lsr press cnt=4
Value Sent: Ffffgdsgg.fgh
7
Unknown Descriptor
UUID: 2902
Value: N/A
Value Sent: N/A
Unknown Characteristic
UUID: FFE2
Properties: Write and Write Without Response
Value: N/A
Value Sent: N/A
HC000048872D9B0878
Connected


Close
Adv...
Client
Server
Log
DFU
Disconnect
Attribute Table
Unknown Characteristic
UUID: 2A29
Properties: Read
Value: N/A
Value Sent: N/A
Unknown Characteristic
UUID: 2A50
Properties: Read
Value: N/A
Value Sent: N/A
Unknown Service
UUID: FFE0
PRIMARY SERVICE
Unknown Characteristic
UUID: FFE1
Properties: Read, Write, Write Without Response, and Notify
Value: uart rec over ACT:sy;CMD:resex; 17
ACT
sy
CMD
resex
8
Value Sent: Ffffgdsgg.fgh
Unknown Descriptor
UUID: 2902
Value: N/A
Value Sent: N/A
Unknown Characteristic
UUID: FFE2
HC000048872D9B0878
Connected


14:58
5G 83
Close
Adv...
Client
Server
Log
DFU
Disconnect
Attribute Table
Unknown Characteristic
UUID: 2A29
Properties: Read
Value: N/A
Value Sent: N/A
Cancel
Write Value
Ggggggghgyytt
10
ByteArray
UnsignedInt
Bool
UTF8
9
Write Type
Command
Request
Write
11
Value: uart rec over ACT:sy;CMD:resex; 17


Close
Adv...
Client
Server
Log
DFU
Disconnect
Attribute Table
Unknown Characteristic
UUID: 2A29
Properties: Read
Value: N/A
Value Sent: N/A
Unknown Characteristic
UUID: 2A50
Properties: Read
Value: N/A
Value Sent: N/A
Unknown Service
UUID: FFE0
PRIMARY SERVICE
Unknown Characteristic
UUID: FFE1
Properties: Read, Write, Write Without Response, and Notify
Value: uart rec over ACT:sy;CMD:resex; 17
ACT
sy
CMD
resex
Value Sent: Ggggggghgyytt
12
Unknown Descriptor
UUID: 2902
Value: N/A
Value Sent: N/A
Unknown Characteristic
UUID: FFE2
HC000048872D9B0878
Connected

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.I2S 语音播放实例

10.1 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。

10.2 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);                                    1
 //I2S 复位
 EPG_SYSCON0->SWRESET |=(1<<13);
 EPG_SYSCON0->SWRESET &=~(1<<13);                                          2

 //实始化I2S GPIO
 EPG_gpio_SetAltFunc_GPIOH16(EPG_GPIO0, 0x15000000);                       3
 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                             4

 DisI2sInt();     //禁止中断                                                5
}
```

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。

```
/*
 * 函数功能：I2S发送左右声道数据
 * 输入参数：
 */
u8 I2sSendData(u32 aDatLeft,u32 aDatRight)
{
    uint32_t tmp=0;

    EPG_I2S1->TDR= aDatLeft;                                  //发送L声道数据
    EPG_I2S1->IER |= EPG_I2S_TXE_INT_Msk;                     //使能中断
    EPG_I2S1->CR = 0x1;                                       //启动发送
    while(i2s1_irq_occurred==0);                              //等发送完成
    EPG_I2S1->TDR = aDatRight;                                //发送R声道数据
    while(!(EPG_I2S1->INTSTATUS & EPG_I2S_TXE_INT_Msk));      //等发送完成
    EPG_I2S1->CR = 0x0;                                       //停止发送
    i2s1_irq_occurred = 0;                                    //清标志

    return 0;
}
```

3) I2S 声音文件数组

```
/*
 * I2S发送语音内容
 * 输入参数：    *pdata：发送的内容指针（内容为8位采样）
 *              alen:发送的内容长度
 */

 u8 I2sSendBufNew(const uint8_t *pdata ,u16 alen)
 {
   u8 tmp=0;
   for(u16 i=0;i<alen;i++)      //I2S 发送数据
   {
       //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
       tmp=I2sSendData(*(pdata+i)<<21,0);

   }
   DisI2sInt();                 //禁止中断
 }
```

4) 主循环周期调用播放

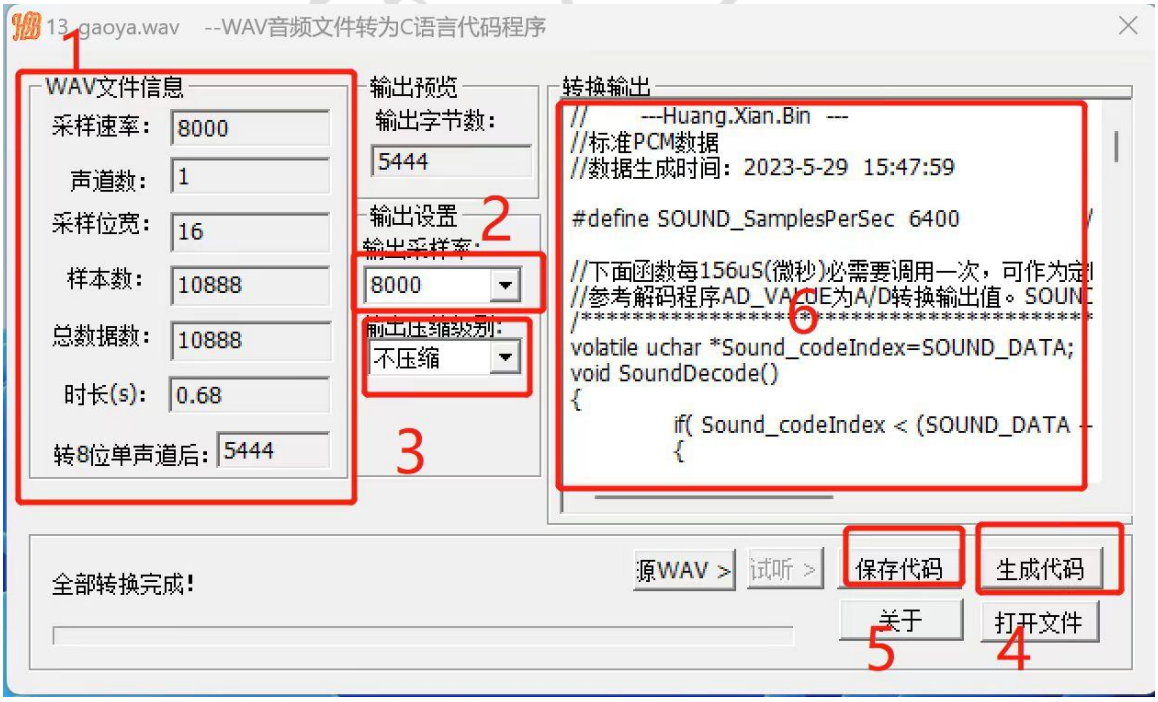
```
void IisTest(void)
{
    I2sSendBufNew(InBusy,5444);      //调用I2S发送语音内容。

 }
```

10.3 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



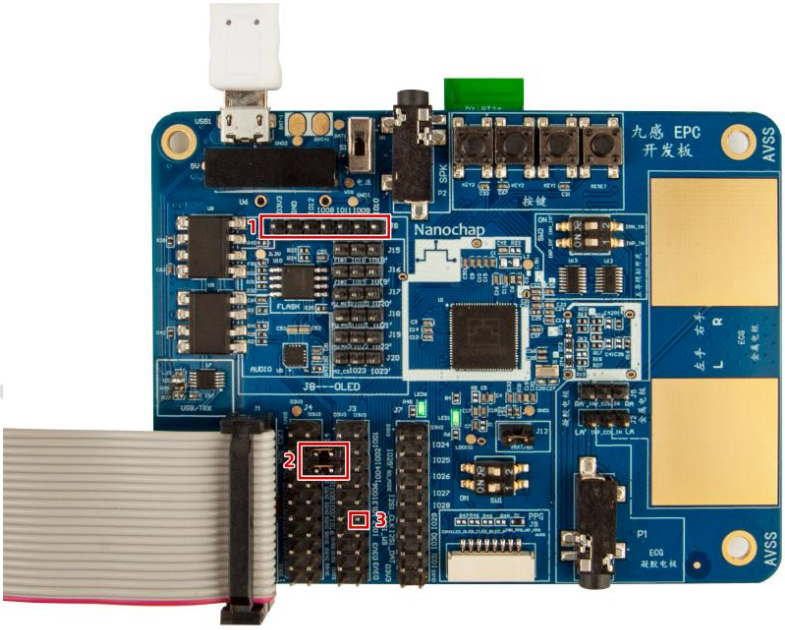
① **位置 1:**为 .wav 文件信息，采样速率为 8K，则不会失真；

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。

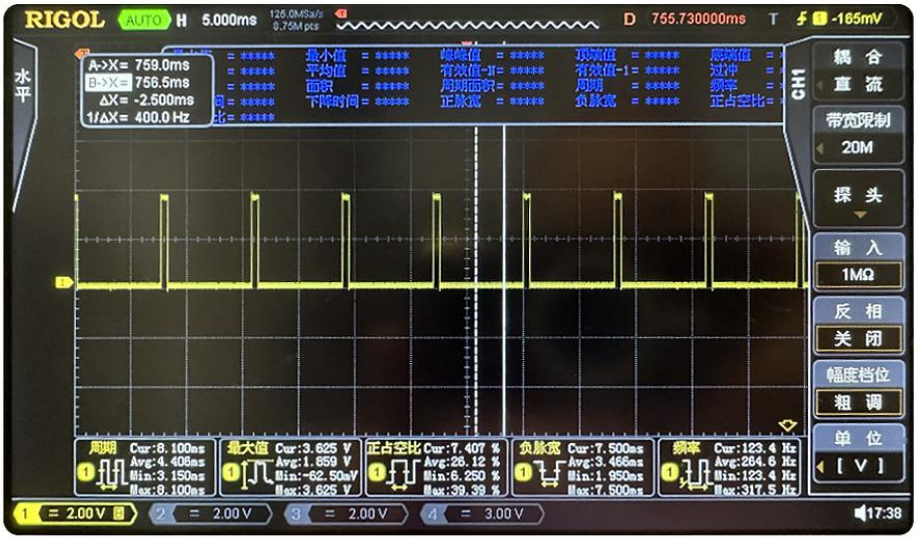
11. PWM 实例

11.1 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

11.2 测试效果

如 PWM 波形图所示

11.3 代码分析

实例对应工程文件：../project-pwm

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);
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);
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);
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);
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);
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);
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);
78     EPG_gpio_SetFuncMode_GPIOL16_Pin(EPG_GPIO0, PMW3_PIN, GPIO_MODE_ALTFUN);
79     EPG_gpio_SetFuncMode_GPIOL16_Pin(EPG_GPIO0, PMW4_PIN, GPIO_MODE_ALTFUN);
80     EPG_gpio_SetFuncMode_GPIOL16_Pin(EPG_GPIO0, PMW5_PIN, GPIO_MODE_ALTFUN);
81     EPG_gpio_SetFuncMode_GPIOL16_Pin(EPG_GPIO0, PMW6_PIN, GPIO_MODE_ALTFUN);
82
83     EPG_gpio_SetAltFunc_GPIOL16_Pin(EPG_GPIO0, PMW1_PIN, GPIO10_PWM1);          //复用模式为 PWM
84     EPG_gpio_SetAltFunc_GPIOL16_Pin(EPG_GPIO0, PMW2_PIN, GPIO11_PWM2);
85     EPG_gpio_SetAltFunc_GPIOL16_Pin(EPG_GPIO0, PMW3_PIN, GPIO12_PWM3);
86     EPG_gpio_SetAltFunc_GPIOL16_Pin(EPG_GPIO0, PMW4_PIN, GPIO13_PWM4);
87     EPG_gpio_SetAltFunc_GPIOL16_Pin(EPG_GPIO0, PMW5_PIN, GPIO14_PWM5);
88     EPG_gpio_SetAltFunc_GPIOL16_Pin(EPG_GPIO0, PMW6_PIN, GPIO15_PWM6);
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。

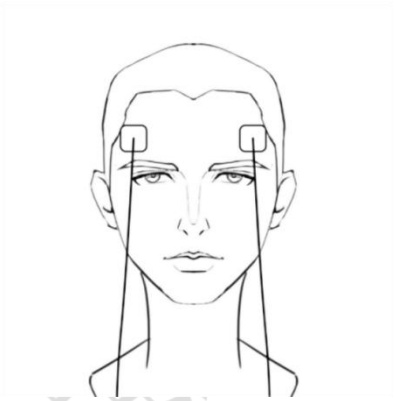
```
9  static void PWM_Handler(void) __attribute__((interrupt));
10 static void PWM_Handler(void) {
11 //MR0 status interrupt
12     if((EPG_PWM0->INTSTATUS & EPG_PWM_MR0_INT_STS_Msk)==EPG_PWM_MR0_INT_STS_Msk) {
13         EPG_PWM0->INTCLEAR = EPG_PWM_MR0_INT_STS_Msk;//Clear Interrupt
14
15     }
16 //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     }
21 //MR2 status interrupt
22     if((EPG_PWM0->INTSTATUS & EPG_PWM_MR2_INT_STS_Msk)==EPG_PWM_MR2_INT_STS_Msk) {
23         EPG_PWM0->INTCLEAR = EPG_PWM_MR2_INT_STS_Msk;//Clear Interrupt
24
25     }
26 //MR3 status interrupt
27     if((EPG_PWM0->INTSTATUS & EPG_PWM_MR3_INT_STS_Msk)==EPG_PWM_MR3_INT_STS_Msk) {
28         EPG_PWM0->INTCLEAR = EPG_PWM_MR3_INT_STS_Msk;//Clear Interrupt
29
30     }
31 //MR4 status interrupt
32     if((EPG_PWM0->INTSTATUS & EPG_PWM_MR4_INT_STS_Msk)==EPG_PWM_MR4_INT_STS_Msk) {
33         EPG_PWM0->INTCLEAR = EPG_PWM_MR4_INT_STS_Msk;//Clear Interrupt
34
35     }
36 //MR5 status interrupt
37     if((EPG_PWM0->INTSTATUS & EPG_PWM_MR5_INT_STS_Msk)==EPG_PWM_MR5_INT_STS_Msk) {
38         EPG_PWM0->INTCLEAR = EPG_PWM_MR5_INT_STS_Msk;//Clear Interrupt
39
40     }
41 //MR6 status interrupt
42     if((EPG_PWM0->INTSTATUS & EPG_PWM_MR6_INT_STS_Msk)==EPG_PWM_MR6_INT_STS_Msk) {
43         EPG_PWM0->INTCLEAR = EPG_PWM_MR6_INT_STS_Msk;//Clear Interrupt
44
45     }
46     return;
47 }
```

3) PWM 初始化

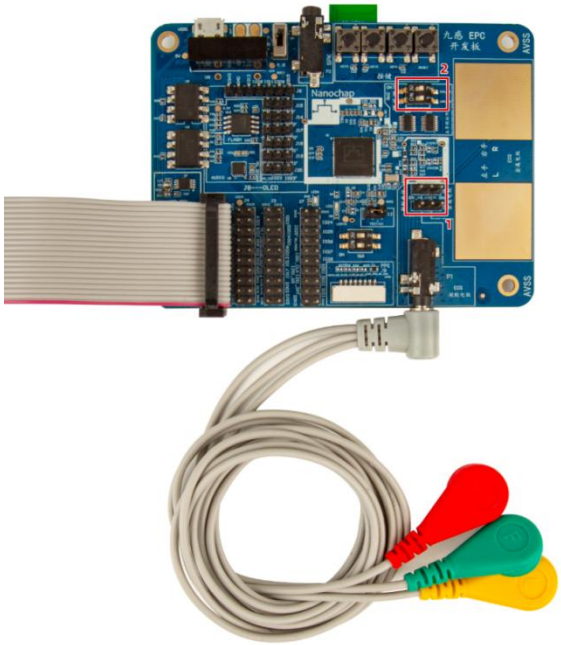
```
}
void PwmMacInit(EPG_PWM_TypeDef *EPG_PWM)
{
    PwmGpioInit();      //GPIO 初始化

    ClicInstall(PWM_Handler,PWM_IRQn,4);          //中断初始化
    //使能通道                                    //通道输出使能
     EPG_PWM->PCR = (EPG_PWM_CTRL_PWM1_EN_Msk
             | EPG_PWM_CTRL_PWM2_EN_Msk
             | EPG_PWM_CTRL_PWM3_EN_Msk
             | EPG_PWM_CTRL_PWM4_EN_Msk
             | EPG_PWM_CTRL_PWM5_EN_Msk
             | EPG_PWM_CTRL_PWM6_EN_Msk
             | EPG_PWM_CTRL_PWM2_SEL_Msk
             | EPG_PWM_CTRL_PWM4_SEL_Msk
             | EPG_PWM_CTRL_PWM6_SEL_Msk);

    //设置MR值
     EPG_PWM->MR0 = 0x00000500;          //周期
     EPG_PWM->MR1 = 0x00000000;          //PWM1 占空比
     EPG_PWM->MR2 = 0x00000040;          //PWM2 占空比
     EPG_PWM->MR3 = 0x00000020;          //PWM3 占空比
     EPG_PWM->MR4 = 0x00000080;          //PWM4 占空比
     EPG_PWM->MR5 = 0x00000040;          //PWM5 占空比
     EPG_PWM->MR6 = 0x00000100;          //PWM6 占空比
     //配置MCR
     EPG_PWM->MCR |= EPG_PWM_MATCH_MR0_RET_Msk; //PWM 占空比
     //分频配置
     EPG_PWM->PR = 0x00000064;
     //LOAD  使能
     EPG_PWM->LER = (EPG_PWM_LOAD_ML6_EN_Msk
             |EPG_PWM_LOAD_ML5_EN_Msk
             |EPG_PWM_LOAD_ML4_EN_Msk
             |EPG_PWM_LOAD_ML3_EN_Msk
             |EPG_PWM_LOAD_ML2_EN_Msk
             |EPG_PWM_LOAD_ML1_EN_Msk
             |EPG_PWM_LOAD_ML0_EN_Msk);//0x0000007f;
     //计数能
     EPG_PWM->TCR |= EPG_PWM_TIMER_CNT_EN_Msk;
     //中断使能控制
     EPG_PWM->MCR |= (EPG_PWM_MATCH_MR0_INT_Msk
                     | EPG_PWM_MATCH_MR1_INT_Msk
                     | EPG_PWM_MATCH_MR2_INT_Msk
                     | EPG_PWM_MATCH_MR3_INT_Msk
                     | EPG_PWM_MATCH_MR4_INT_Msk
                     | EPG_PWM_MATCH_MR5_INT_Msk
                     | EPG_PWM_MATCH_MR6_INT_Msk);


}
```

12.EEG 脑电

测试需要将红色以及黄色电极分别贴到额头左右两侧，绿色电极不贴，但要注意静电隔离。



■ **硬件设置：**



开发板设置

■ **界面及图形：**

初始界面

■ **配置：**

Serial：串口号，选择当前串口号；

Baud rate：波特率，选择 115200；

Connect：圆形按钮，点击打开串口；

FFT Tool：可以查看原始波形数据经过 FFT 变换之后的频谱；

干电极/湿电极选择：默认湿电极；

Measure/Stop：开启、停止数据采集。

■ **波形显示：**

EEG：原始数据波形；

波段信号：Delta、Theta、Low Alpha(慢速)、High Alpha(快速)、Low Beta(慢速)、High Beta(快速)、Low Gamma(慢速)、High Gamma(快速)；

算法输出：Noise 噪声相对强度(空载最大)；Attention 专注度指数，范围 0 到 100，使用者精神“集中度”水平的强烈程度，心烦意乱、注意力不集中、焦虑等精神状态将降低专注度指数的值；Meditation 放松度指数，范围 0 到 100，使用者“平静度”水平，闭上眼睛把注意力放到呼吸上通常是提高放松度指数的有效方法，心烦意乱、焦虑、激动不安等精神状态，及感官刺激等将会降低放松度指数的值。

12.1 开始采集

1）选择电脑端对应串口，波特率选择 115200，点击右侧圆形按钮打开串口；

2）选择电极片类型：本次测试使用湿电极，对应点击湿电极选项；

3）点击“MEASURE”按钮，开启数据采集，如下图；

专注时波形

放松时波形

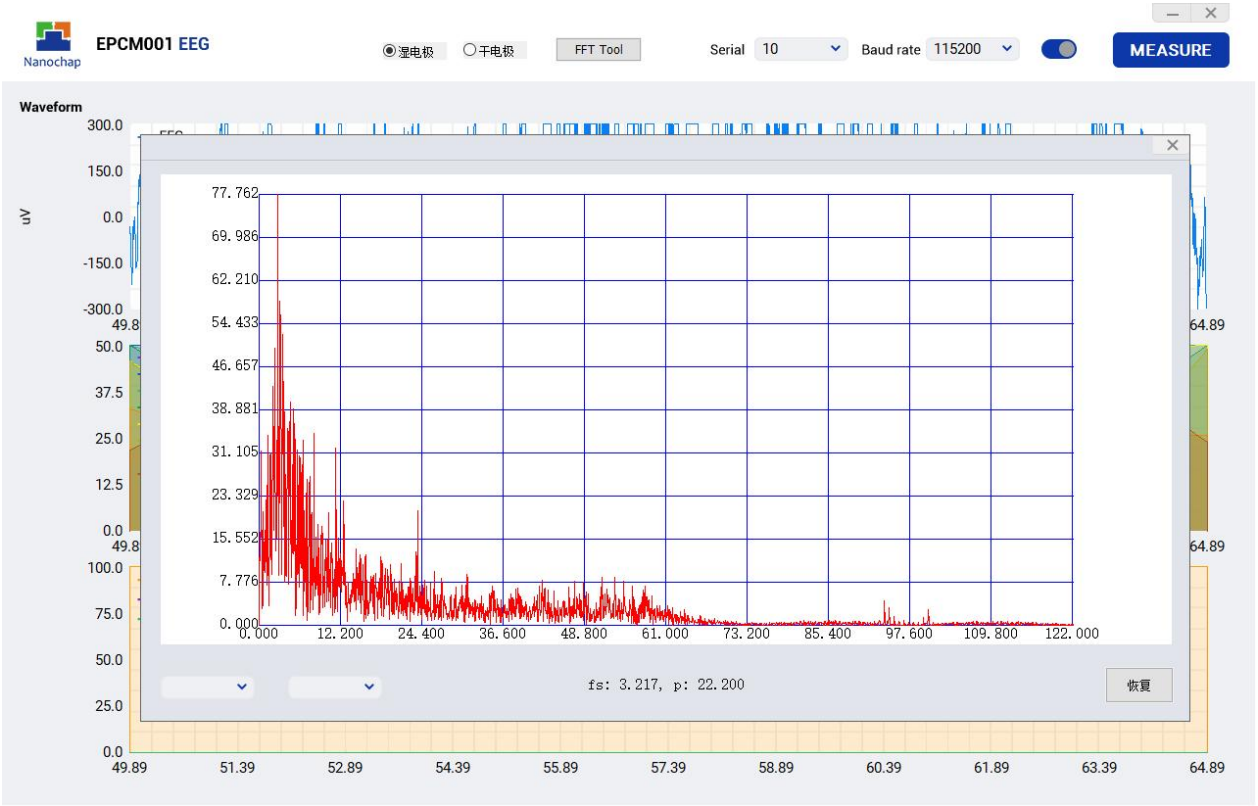
12.2 停止采集

点击“Stop”按钮，停止数据采集。

停止采集

12.3 FFT变换

点击FFT Tool 可以观察到FFT变换后频率分布，如下图：



FFT频谱图

13.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