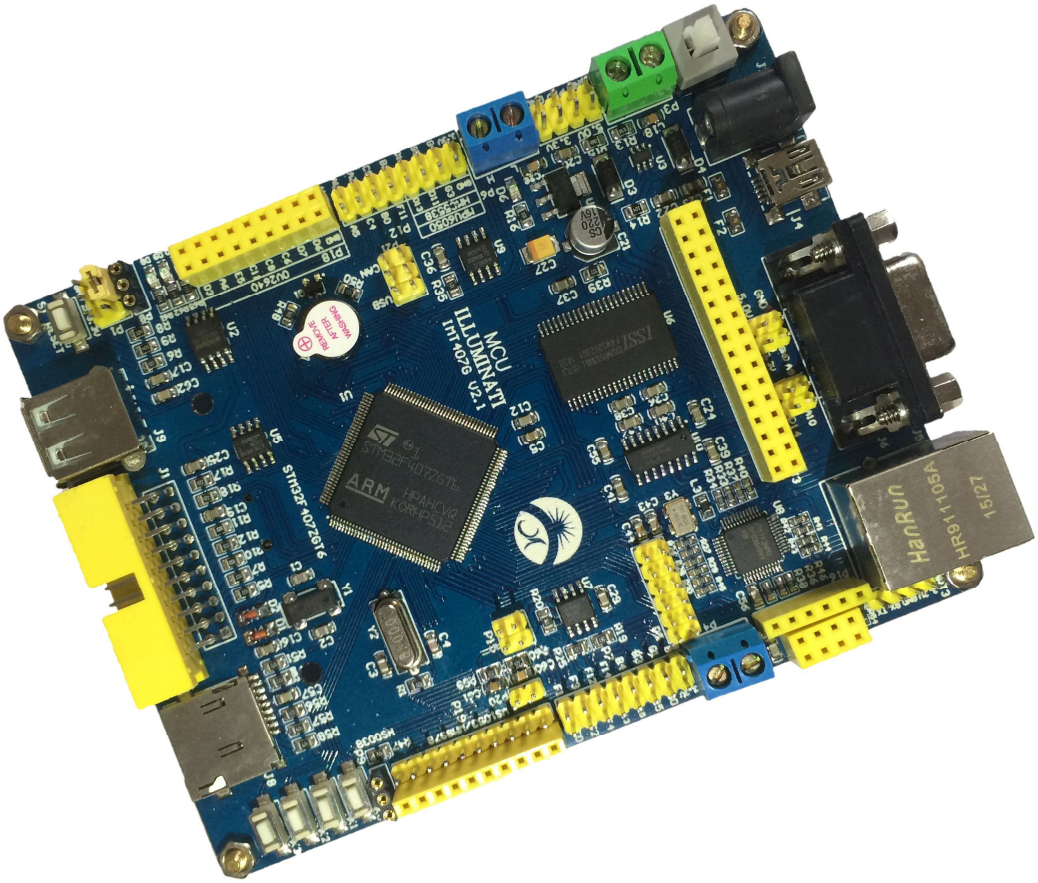
MCU 启明 STM32F407 开发板(高配版) V2.1

----使用手册



MCU 启明

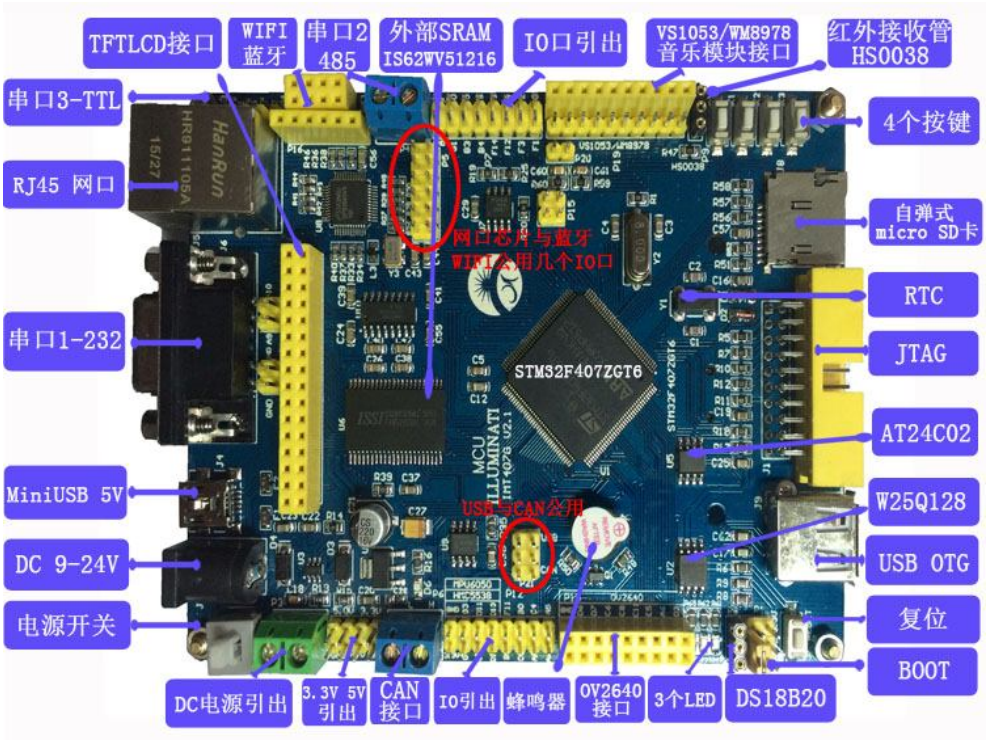
官方店铺：https://shop125046348.taobao.com

技术支持：QQ 3231824766

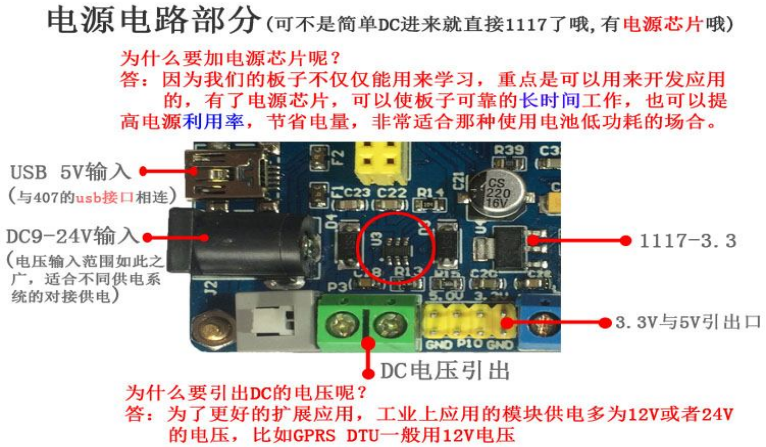
目 录

第一篇 硬件资源介绍

1.1 板载资源与接口介绍



1.2 板子供电介绍



这边注意下，DC 座供电用 6V 供电也行，但是为了供电稳定建议用 6V 以上的。**MINI USB 接口接于 STM32 的全速 usb IO 口，用于**

供电和进行 USB 通信，不能用于下载程序。

第二篇 开发工具与工程架构解析

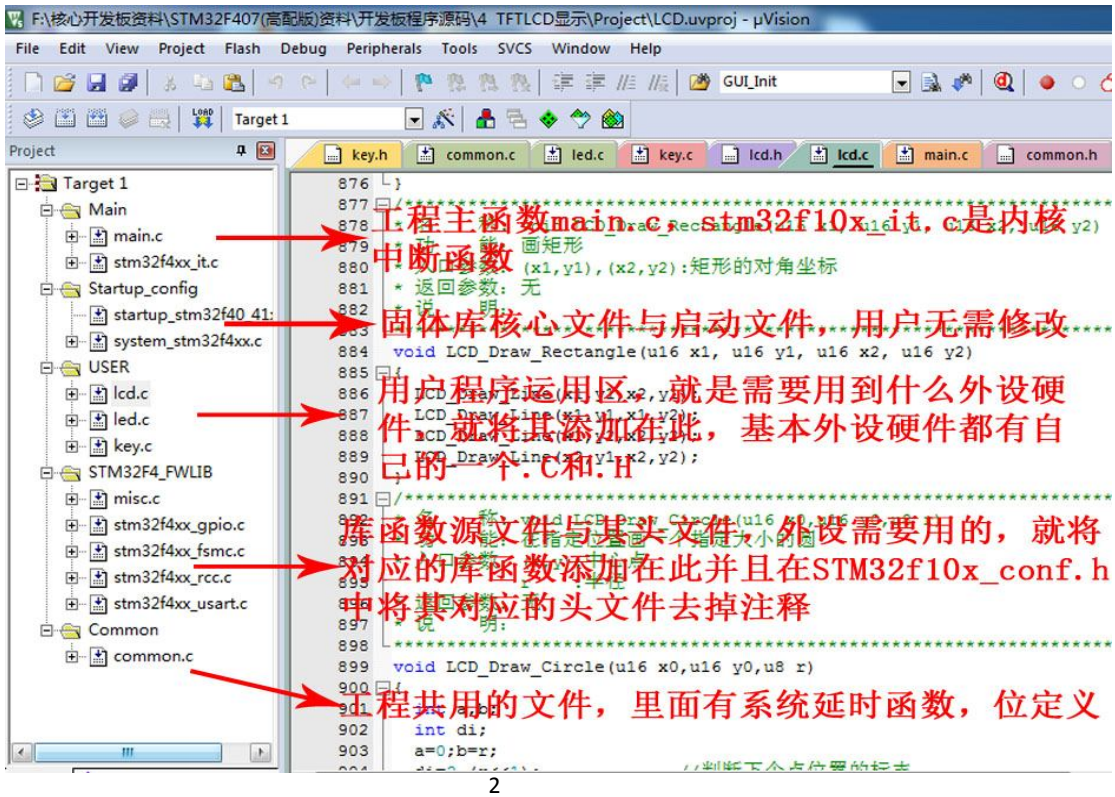
2.1 开发环境

例程使用 KEIL5(MDK5)作为开发环境，资料 MCU 启明 STM32F407(高配版)资料\常用软件\MDK5 中有安装文件和安装教程。

2.2 程序下载

MCU 启明 STM32F407开发板(高配版)可以使用串口线或者仿真器下载程序，具体怎么下载程序，请看资料 MCU 启明 STM32F407(高配版)资料\启明 F407开发板(高配版)**程序下载教程**。

2.3 程序工程架构解析

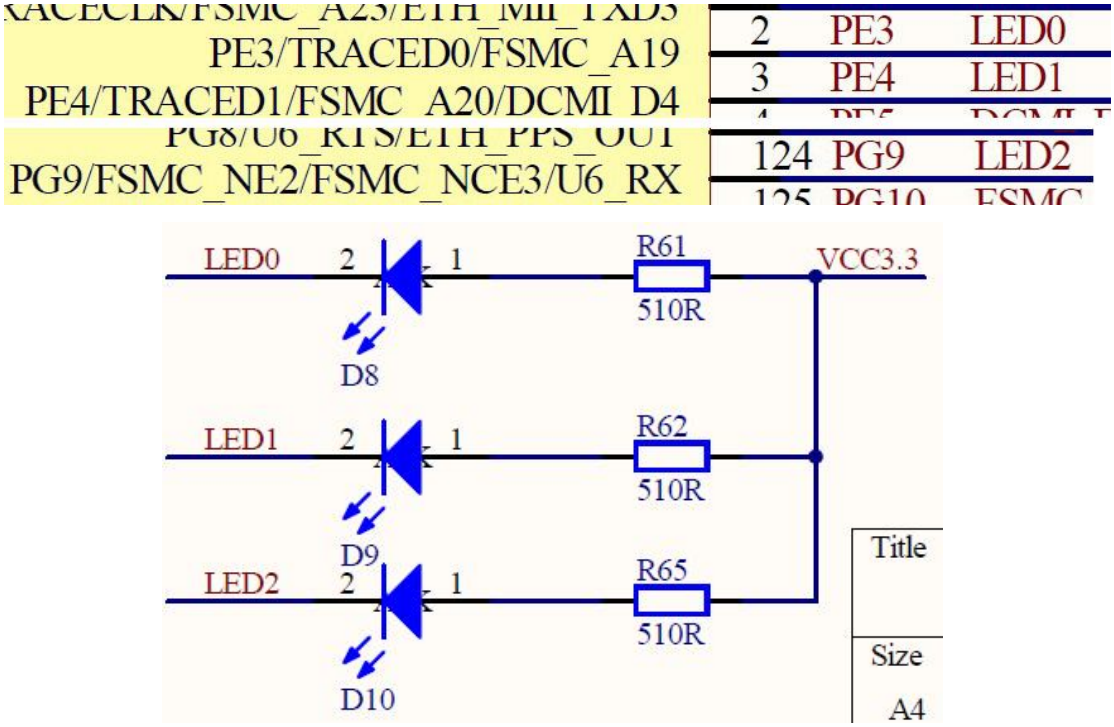


第三篇 例程使用详解

此篇会结合开发板原理图和程序源码，讲解怎么使用各个例程。

第一章 LED 跑马灯

3.1.1 硬件介绍



3.1.2 核心代码解析

代码是通过位带操作实现 IO 口控制

```
int main(void)
{
   delay_init();  //初始化延时函数
   LED_Init();    //初始化 LED 端口

  while(1)
    {
    LED0=0;        //LED0 亮
    LED1=1;        //LED1 灭
    LED2=1;         //LED2 灭
    delay_ms(500);
   LED0=1;         //LED0 灭
    LED1=0;        //LED1 亮
```

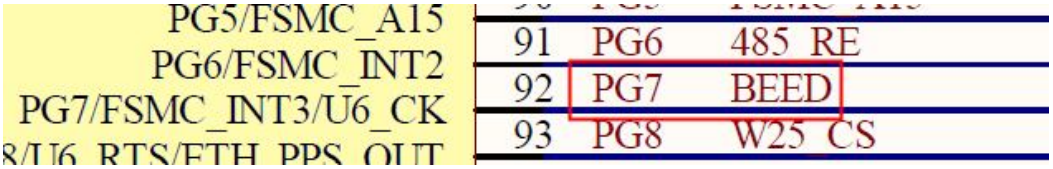
```
    LED2=1;         //LED2 灭

    delay_ms(500);
    LED0=1;         //LED0 灭
    LED1=1;         //LED1 灭
    LED2=0;         //LED2 亮
    delay_ms(500);
    }
}
```

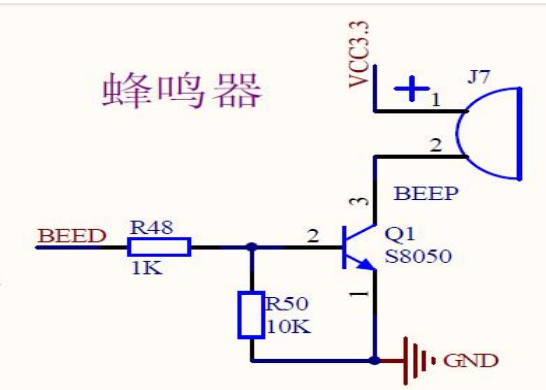
3.1.3 实验操作与现象

开发板下入该 LDE 程序，LED 会从 LED0 至 LED2 依次亮灭，形成流水灯效果。

第二章 蜂鸣器使用

3.2.1 硬件设计



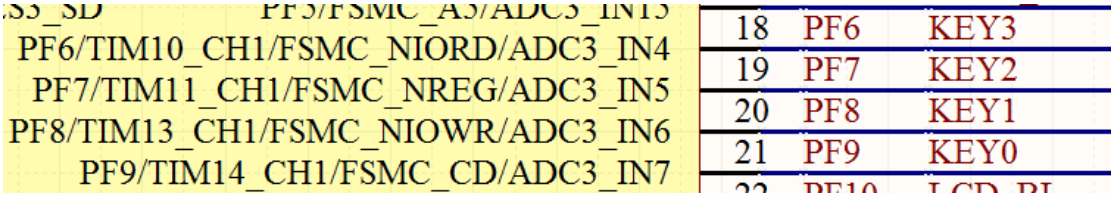


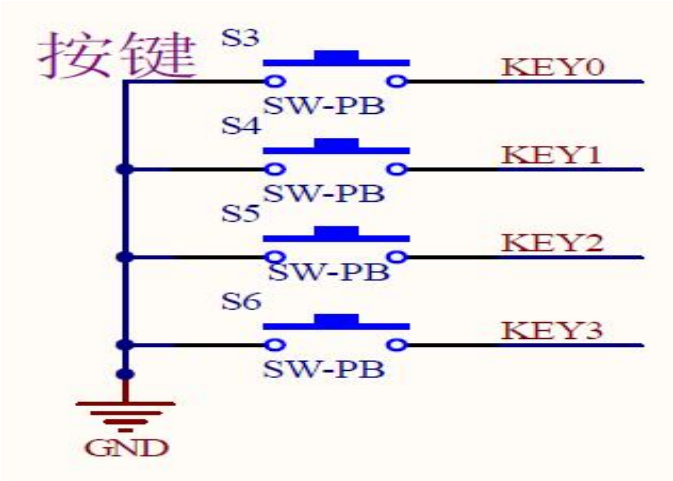
3.2.2 实验操作与现象

程序下进去后，蜂鸣器会一响一闭的。

第三章　按键使用

3.3.1 硬件设计





3.3.2 核心代码解析

```
void key_scan(u8 mode)
{
    keyup_data=0;          //键抬起后按键值一次有效
   if(KEY0==0||KEY1==0||KEY2==0||KEY3==0)    //有键正按下
   {
      if(KEY0==0)       key_tem=1;
      else if(KEY1==0) key_tem=2;
      else if(KEY2==0) key_tem=3;
      else if(KEY3==0) key_tem=4;
      if (key_tem == key_bak)       //有键按下后第一次扫描不处理，与 else 配合第二次扫描有效，这样实现了去抖动
            {
               key_time++;       //有键按下后执行一次扫描函数，该变量加 1
              keydown_data=key_tem;    //按键值赋予 keydown_data

              if( (mode==0)&&(key_time>1) )//key_time>1 按键值无效，这就是单按，如果 mode 为 1 就为连按
                 keydown_data=0;
          }
```

```
        else                                    //去抖动
          {
               key_time=0;
               key_bak=key_tem;
          }

    }
    else        //键抬起
        {
         if(key_time>2)                 //按键抬起后返回一次按键值
            {
                keyup_data=key_tem;     // 键 抬 起 后 按 键 值 赋 予
keydown_data
               }
               key_bak=0;                       //要清零，不然下次执行扫描
程序时按键的值跟上次按的值一样，就没有去抖动处理了
           key_time=0;
               keydown_data=0;
      }
  }
```

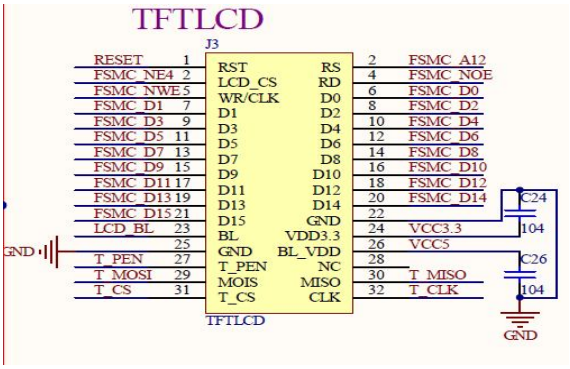
该函数实现按键多种方式的扫描，也实现的多种组合运用，具体请看代码“按键模式剖析”。

3.3.3 实验操作与现象

程序下进去之后，按 KEY0，led 全亮， 按 KEY1，蜂鸣器响， 长按 KEY2 1 秒后，关闭蜂鸣器与 led0， 长按 KEY3 2 秒后，灭掉 led1 和 led2。

第四章 TFTLCD 显示

3.4.1 硬件设计



3.4.2 核心代码解析

```
//使用 NOR/SRAM 的 Bank1.sector4,地址位 HADDR[27,26]=11
//A12 作为数据命令区分线
//注意设置时 STM32 内部会右移一位对其
#define  CMD_BASE        ((u32)(0x6C000000 | 0x00001FFE))
#define  DATA_BASE       ((u32)(0x6C000000 | 0x00002000))
#define LCD_CMD         ( * (u16 *) CMD_BASE )
#define LCD_DATA        ( * (u16 *) DATA_BASE)
```

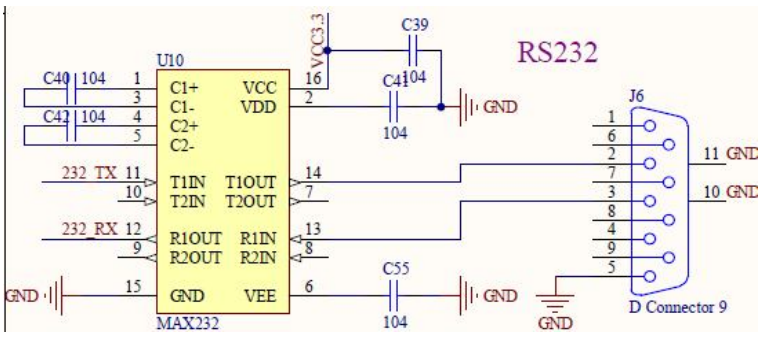
由于用 FSMC A12 作为数据命令选择线，所以数据与命令的地址线如上。

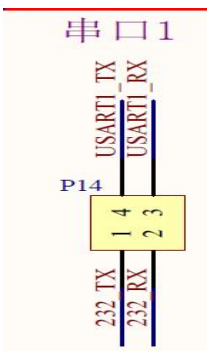
3.4.3 实验操作与现象

程序下进去之后，按 KEY0，依次进行各颜色清屏显示， 按 KEY1，屏幕进行图形显示， 长按 KEY2 进行数值显示，最高位有无填充 0 两种情况。

第五章　串口 1-RS232 实验

3.5.1 硬件设计





使用 DB9 针 232 串口时，需要将 P14 的短路帽接起来。由于使

用的串口 1，所以需要对串口 1 进行初始化和编写需要的中断函数。

3.5.2 核心代码解析

串口发送函数：

```
void uart1SendChars(u8 *str, u16 strlen)
{
      u16 k= 0 ;
   do { uart1SendChar(*(str + k)); k++; }   //循环发送,直到发送完毕
    while (k < strlen);
}
```

串口接收中断，里面定义接收命令的格式：S……E

```
//串口 1 中断服务程序
void USART1_IRQHandler(void)
{
    u8 rec_data;
    if(USART_GetITStatus(USART1, USART_IT_RXNE) != RESET)  //接收中断
        {
        rec_data =(u8)USART_ReceiveData(USART1); //(USART1->DR) 读取接收到的数据
        if(rec_data=='S')               //如果是 S，表示是命令信息的起始位
                {
                   uart_byte_count=0x01;
                }

            else if(rec_data=='E')          //如果 E，表示是命令信息传送的结束位
                {
        if(strcmp("Light_led1",(char*)receive_str)==0)   LED1=0;    //点亮 LED1
        else if(strcmp("Close_led1",(char*)receive_str)==0)  LED1=1;//关灭 LED1
        else if(strcmp("Open_beep",(char *)receive_str)==0)  BEEP=1; //蜂鸣器响
        else if(strcmp("Close_beep",(char *)receive_str)==0) BEEP=0;//蜂鸣器不响
    for(uart_byte_count=0;uart_byte_count<32;uart_byte_count++)receive_str[uart
_byte_count]=0x00;
                uart_byte_count=0;
                }
            else if((uart_byte_count>0)&&(uart_byte_count<=USART1_REC_NUM))
                {
                   receive_str[uart_byte_count-1]=rec_data;
                   uart_byte_count++;
                }
   }
}
```

3.5.3 实验操作与现象

程序下进去之后，接上 232 串口线，电脑端打开串口助手，发送控制命令。

1、串口助手接收开发板串口 1 发来的数据

按 KEY0 后，开发板会发送 UART1 TEST 给串口助手。

2、串口助手发送控制命令给开发板

发送命令格式为：S……E ……为你需要控制的内容，具体看程序代码。

如：发送 SLight_led1E 为打开 LED1，发送后，此时 LED1 就会亮

其他命令同理。

第六章 定时器中断

3.6.1 硬件设计

STM32 内部集成定时器功能。

3.6.2 核心代码解析

```
void TIM2_Init(u16 auto_data,u16 fractional)
{
……
}   定时器溢出时间计算方法:Tout=((auto_data+1)*(fractional+1))/Ft(us)
Ft 定时器时钟
TIM2_Init(4999,8399); //定时器 2 时钟 84M，分频系数 8400，84M/8400=10K 所以计数 5000 次为 500ms
```

3.6.3 实验操作与现象

程序下进去之后，每隔 500ms 蜂鸣器响闭 和 LED1 亮灭。

第七章 PWM 输出

3.7.1 硬件设计

利用 STM32 内部集成的定时器实现 PWM 输出。

3.7.2 核心代码解析

```
//初始化 TIM5 Channel1 PWM 模式
TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_PWM1; //选择定时器模式:TIM 脉冲宽度调制模式 1
 TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable; //比较输出使能
TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_Low; // 输 出 极性:TIM 输出比较极性低
TIM_OC1Init(TIM5, &TIM_OCInitStructure); //根据 T 指定的参数初始化外设 TIM1 4OC1
TIM_OC1PreloadConfig(TIM5, TIM_OCPreload_Enable); //使能 TIM5 在 CCR1 上的预装载寄存器 Ft 定时器时钟

TIM5_PWM_Init(499,83); //84M/84=1Mhz 的计数频率,重装载值 500,所以 PWM 频率为 1M/500=2Khz.

TIM_SetCompare1(TIM5,pwmzkb); 占空比为 pwmzkb/500.
```

3.7.3 实验操作与现象

程序下进去之后，按 KEY0 占空比减少 10%，按 KEY1 占空比增加 10%。

第八章　模数转换 ADC

3.8.1 硬件设计

STM32 内部集成 ADC 模块。

3.8.2 核心代码解析

ch: 通道值 0~16 取值范围为：ADC_Channel_0~ADC_Channel_16

```
u16 Get_Adc(u8 ch)
{
    ADC_RegularChannelConfig(ADC1, ch, 1, ADC_SampleTime_480Cycles );
    //ADC1,ADC 通道,480 个周期,提高采样时间可以提高精确度
    ADC_SoftwareStartConv(ADC1);//使能指定的 ADC1 的软件转换启动功能
    while(!ADC_GetFlagStatus(ADC1, ADC_FLAG_EOC ));//等待转换结束
    return ADC_GetConversionValue(ADC1);  //返回最近一次 ADC1 规则组的转换结果
}
```

该函数返回测的通道的模拟电压值。

3.8.3 实验操作与现象

程序下进去之后，PA0 接入所测量的模拟电压，屏幕会显示接入的模拟电压的电压值，会随着输入的电压的改变而实时显示电压化。

第九章　数模转换 DAC

3.9.1 硬件设计

STM32 内部集成 DAC 模块。

3.9.2 核心代码解析

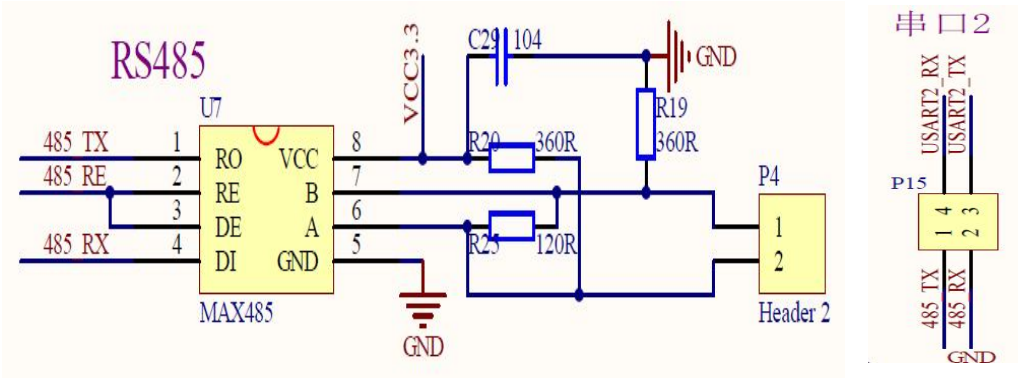
```
/**************************************************************************
* 名    称: void Dac1_Set_Vol(u16 vol)
* 功    能：设置通道 1 输出电压
* 入口参数：vol:0~3300,代表 0~3.3V
* 返回参数：无
**************************************************************************/
void Dac1_Set_Vol(u16 vol)
{
    double temp=vol;
    temp/=1000;
    temp=temp*4096/3.3;  //12 位的故为 4096
    DAC_SetChannel1Data(DAC_Align_12b_R,temp);//12 位右对齐数据格式设
置 DAC 值
}
```

3.9.3 实验操作与现象

该例程为 PA4 作为 DAC 输出电压，PA6 作为 ADC 模拟电压输入，实验时需要用一根杜邦线连接 PA4 与 PA6，此时 DAC(PA4)输出的电压值输入到 PA6，屏幕也会实时显示输出的电压值的数字值、模拟值大小和测得的 ADC 输入电压的模拟值。 按 KEY0 DAC 输出电压增加，按 KEY1 DAC 输出电压减少。

第十章　串口 2-485

3.10.1 硬件设计



由于 485 接于串口 2 并且中间用跳针隔开，所以在使用串口 2 的 485 时，请**将 P15 的短路帽接上**。

3.10.2 核心代码解析

485 发送函数

```
void RS485_Send_Data(u8 *buf,u8 len)
{
    u8 t;
    RS485_TX_EN=1;              //设置为发送模式
    for(t=0;t<len;t++)          //循环发送数据
    {
      while(USART_GetFlagStatus(USART2,USART_FLAG_TC)==RESET); //等待发送结束
     USART_SendData(USART2,buf[t]); //发送数据
    }
    while(USART_GetFlagStatus(USART2,USART_FLAG_TC)==RESET);    //等待发送结束
    uart_byte_count=0;
    RS485_TX_EN=0;                  //发送完设置为接收模式
}
```

RS485_TX_EN 一般设置为“0”，处于接收状态，当什么时候需要发送的才设置为发送，发送完又设置成接收，使 485 一直处于接收状态，不会遗漏接收信息。

RS485 查询接收到的数据

```
void RS485_Receive_Data(u8 *buf,u8 *len)
{
    u8 rxlen=uart_byte_count;
    u8 i=0;
    *len=0;             //默认为 0
    delay_ms(10);  //等待 10ms,连续超过 10ms 没有接收到一个数据,则认为
接收结束

    if(rxlen==uart_byte_count&&rxlen) //接收到了数据,且接收完成了
    {
        for(i=0;i<rxlen;i++)
        {
            buf[i]=RS485_receive_str[i];
        }
        *len=uart_byte_count;    //记录本次数据长度
        uart_byte_count=0;       //清零
    }
}
```

接收中断触发后，在中断服务函数中，就会把接收的数值存在 RS485_receive_str[ ]数组。

3.10.3 实验操作与现象

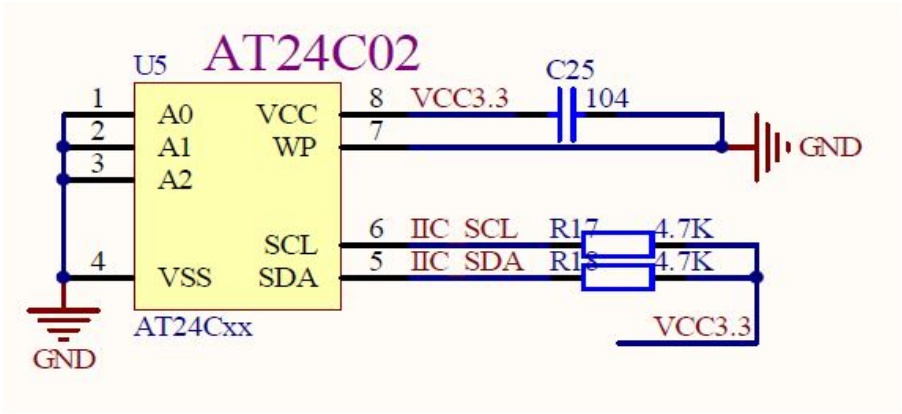
P4 接口为 485 对外的 A B 接口，请将外部 485 设备，按 A 接 A，B 接 B 的形式接起来。程序下进去就可以跟板子进行 485 通信。

发什么控制命令格式跟第五章一样，请参考第五章的使用方法。

10-2 串口3-TTL实验跟第五章的使用操作一模一样，只是程序程序上把串口1改成串口3而已了。

第十一章 IIC_24C02

3.11.1 硬件设计



PB8 与 PB9 为原生的硬件 IIC。由于 STM32 原生的 IIC 不稳定，所以本例程程序使用普通 IO 口进行模拟 IIC 通信协议。

3.11.2 核心代码解析

```
//IIC_SDA 线 IO 方向配置
#define SDA_IN()  {GPIOB->MODER&=~(3<<18);GPIOB->MODER|=0<<18;}   //PB9 输入模式
#define SDA_OUT() {GPIOB->MODER&=~(3<<18);GPIOB->MODER|=1<<18;} //PB9 输出模式
```

AT24C02 读函数：

```
/**************************************************************************
* 名    称: void AT24C02_Read(u8 ReadAddr,u8 *pBuffer,u8 ReadNum)
* 功    能: 从 AT24C02 里面的指定地址开始读出指定个数的数据
* 入口参数: ReadAddr :开始读出的地址  0~255
            pBuffer  :数据数组首地址
            ReadNum:要读出数据的个数
* 返回参数: 无
**************************************************************************/
void AT24C02_Read(u8 ReadAddr,u8 *pBuffer,u8 ReadNum)
{
```

```
    while(ReadNum--)
    {
        *pBuffer++=AT24C02_ReadByte(ReadAddr++);
    }
}
```

AT24C02 写函数：

```
/*****************************************************************************
* 名    称: void AT24C02_Write(u8 WriteAddr,u8 *pBuffer,u8 WriteNum)
* 功    能: 从 AT24C02 里面的指定地址开始写入指定个数的数据
* 入口参数: WriteAddr :开始写入的地址  0~255
            pBuffer  :数据数组首地址
            WriteNum:要写入数据的个数
* 返回参数:
*****************************************************************************/
void AT24C02_Write(u8 WriteAddr,u8 *pBuffer,u8 WriteNum)
{
    while(WriteNum--)
    {
        AT24C02_WriteByte(WriteAddr,*pBuffer);
        WriteAddr++;
        pBuffer++;
    }
}
```

3.11.3 实验操作与现象

该例程需要配合串口 1 通信来完成。

1、接收串口 1 发来的数据：

程序开始运行后，屏幕会显示请用户发送数据到串口1 Please send data to usart1，此时用户就要通过串口助手向开发板发送数据，发送格式为前面串口例程那样S……E。“……”为用户所需要发送的数据，发送后数据存在 receive_str[ ]数值中并在显示屏显示。

2、将接收到的数据写入AT24C02：

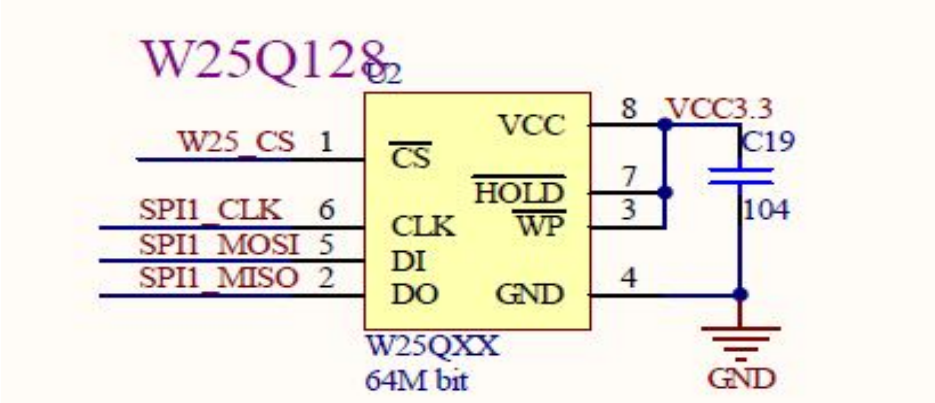
KEY0按下,将串口1接收到的数据(receive_str[ ]中数据)写入24C02

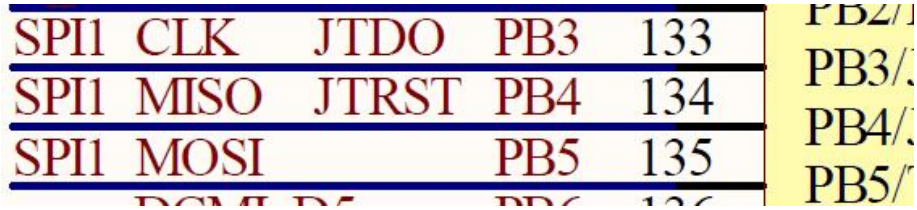
3、将写入到24C02的数据读出并显示

KEY1按下,将写入到24C02中的数据读出并显示

第十二章 SPI_W25Qxx

3.12.1 硬件设计





W25Q128 接于 STM32 原生硬件 SPI1 处。

3.12.2 核心代码解析

//读写函数

3.12.3 实验操作与现象

该例程需要配合串口 1 通信来完成。

1、接收串口 1 发来的数据：

程序开始运行后，屏幕会显示请用户发送数据到串口1

Please send data to usart1，此时用户就要通过串口助手向开发板发送数

据，发送格式为前面串口例程那样S.....E。“.....”为用户所需要发送的数据，发送后数据存在 receive_str[ ]数值中并在显示屏显示。

2、将接收到的数据写入W25Q128：

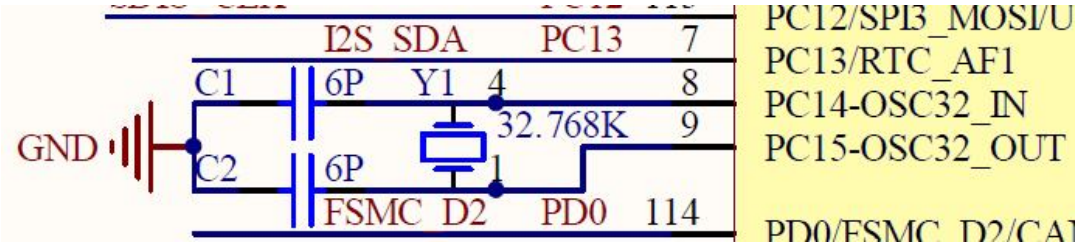
KEY0按下,将串口1接收到的数据(receive_str[ ]中数据)写入25Q128

3、将写入到W25Q128的数据读出并显示:

KEY1按下,将写入到W25Q128中的数据读出并显示

第十四章 RTC 实时时钟

3.14.1 硬件设计



STM32F4 的实时时钟（RTC）相对于 STM32F1 来说，改进了不少，带了日历功能了，STM32F4 的 RTC，是一个独立的 BCD 定时器/计数器。RTC 提供一个日历时钟（包含年月日时分秒信息）、两个可编程闹钟（ALARM A 和 ALARM B）中断，以及一个具有中断功能的周期性可编程唤醒标志。RTC 还包含用于管理低功耗模式的自动唤醒单元。

3.14.2 核心代码解析

F407 的 RTC 就是一个时钟芯片，代码操作也是操作对应的寄存器，比如年月日时分秒寄存器。所以代码看起来简单。请用户看下例程代码。

3.14.3 实验操作与现象

程序下载进去后，RTC时钟初始化后开始跑动，并在液晶屏上每秒更新显示。该时钟例程有调整设置时间功能。

1、 长按KEY3 四秒进入调整时间

2、 按KEY2右移选择你要调整的时间选项(年月日时分秒)

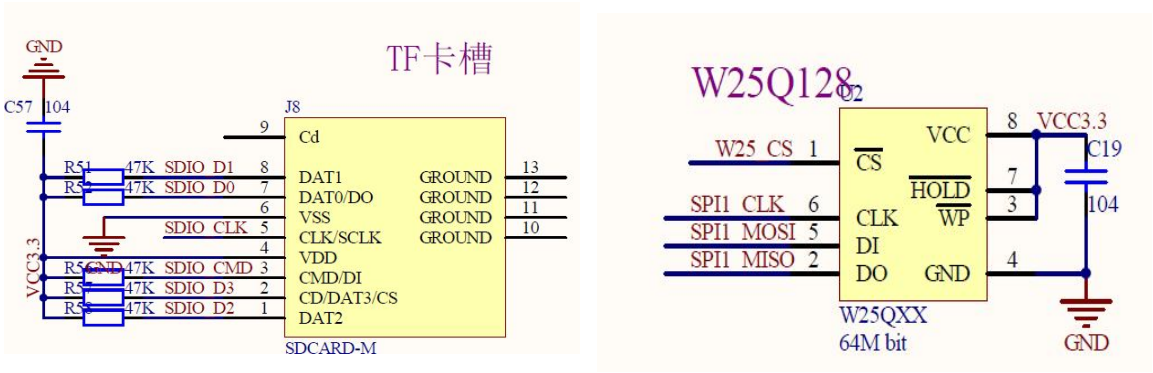
3、 按KEY0数值加1，按KEY1数值减1

4、 调整完时间，短按KEY3设置时间

第十五章 汉字显示

3.15.1 硬件设计

硬件使用到 SD 卡和 FLASH W25Q128。



将 SD 卡中存放的字库，更新到 W25Q128 中，然后液晶屏显示汉字从 W25Q128 取字库。

3.15.2 核心代码解析

由于使用到 SD 卡，所以用文件系统操作比较方便。使用到文件系统需要先为其分别内存，用内存管理来分配。

```
/*************HanZiUse.lib 调用 更新字库要用到的************/
```

```
Memory_Init(INSRAM);   //初始化内部内存池
fsTF=(FATFS*)Mem_malloc(INSRAM,sizeof(FATFS)); //为文件系统分配内存
f_mount(fsTF,"0:",1);       //挂载 TF 卡
/*********************************************************/
```

3.15.3 实验操作与现象

程序下进去后，检测W25Q128里面有没有字库，如没有字库，更新字库。 按KEY0可以更新字库(从SD卡取字库更新到W25Q128)，按KEY1进行汉字显示测试。

第十六章　RTC 农历显示

3.16.1 硬件设计

该例程为 RTC 和汉字显示结合运用。

3.16.2 核心代码解析

只要输入公历的日期就可以得到农历日期的字符串

```
//*********************************************************
*函数名称:void GetLunarCalendarStr(u16 year,u8 month,u8 day,u8 *str)
*功能描述:输入公历日期得到农历字符串
*入口参数: year              公历年
*              month          公历月
*              day             公历日
*              str             接收农历日期字符串地址
*输出参数: 无
*说       明：GetLunarCalendarStr(2015,03,15,str) 返回*str="乙未年正月廿五"
*********************************************************/
void GetLunarCalendarStr(u16 year,u8 month,u8 day,u8 *str)
{
    u8 NLyear[4];
    u8 SEyear;
```

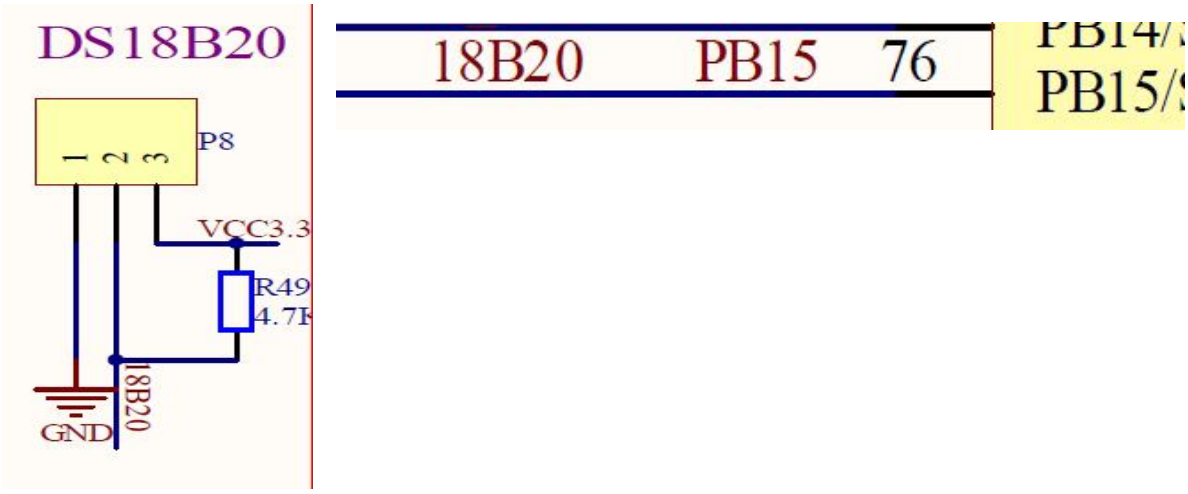
```
    StrCopyss(&str[0],(u8 *)"甲子年正月初一",15);
    if(GetChinaCalendar(year,month,day,(u8 *)NLyear)==0)  return;
    GetSkyEarth(NLyear[0]*100+NLyear[1],&SEyear);
    StrCopyss(&str[0],(u8 *)   sky[SEyear%10],2);   //   甲
    StrCopyss(&str[2],(u8 *)earth[SEyear%12],2);   //   子

    if(NLyear[2]==1)  StrCopyss(&str[6],(u8 *)"正",2);
    else              StrCopyss(&str[6],(u8 *)monthcode[NLyear[2]-1],2);

    if(NLyear[3]>10)       StrCopyss(&str[10],(u8 *)nongliday[NLyear[3]/10],2);
    else              StrCopyss(&str[10],(u8 *)"初",2);
    StrCopyss(&str[12],(u8 *)monthcode[(NLyear[3]-1)%10],2);
}
```

3.16.3 实验操作与现象

程序下进去后，检测W25Q128里面有没有字库，如没有字库，更新字库。 接着就显示时间和公历农历的日期，还有节气。如需调整时间，请参照第十四章的RTC实验，操作与其一模一样。

第十七章　温度传感器 DS18B20

3.17.1 硬件设计



3.17.2 核心代码解析

获取温度值

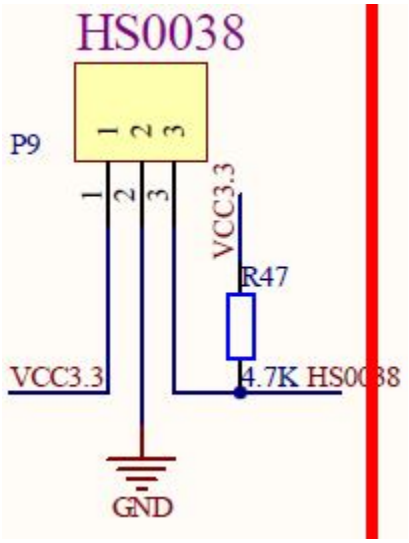
```
short DS18B20_Get_Temp(void)
{
    u8 temp;
    u8 TL,TH;
      short tem;
    DS18B20_Start ();                          // ds1820 start convert
    DS18B20_Rst();
    DS18B20_Check();
    DS18B20_Write_Byte(0xcc);// skip rom
    DS18B20_Write_Byte(0xbe);// convert
    TL=DS18B20_Read_Byte(); // LSB
    TH=DS18B20_Read_Byte(); // MSB
    if(TH>7)
    {
        TH=~TH;
        TL=~TL;
        temp=0;//温度为负
    }else temp=1;//温度为正
    tem=TH; //获得高八位
    tem<<=8;
    tem+=TL;//获得底八位
    tem=(double)tem*0.625;//转换     此时的转换已经把温度扩大 10 倍
   if(temp)return tem; //返回温度值
   else return -tem;
}
```

3.17.3 实验操作与现象

程序下进去后，每200ms读取温度值，并且在显示屏显示。LED0闪烁显示系统正在运行。

第十八章　红外传感器 HS0038

3.18.1 硬件设计



3.18.2 核心代码解析

```
说明：Ir_Record[4]; 处理后的红外码，分别是 客户码，客户码，数据码，数据码反码
*****************************************************************/
void Ir_Decode(void)
{
   u8 i, j, k;
   u8 Record,Value;

   IR_OK = 1;
   k = 1;

   for(i = 0;i < 4;i++)                    //处理 4 个字节
       {
                    delay_ms(1);
                    for(j = 1;j <= 8;j++)     //处理 1 个字节 8 位
                        {
                            Record = Ir_TimeData[k];
          if(Record  >7)        //大于某值为 1，这个和晶振有绝对关系，这里使用 12M 计算，此值可以有一定误差
                                {
                                    Value = Value|0x80;
```

```
                                    }
                                   else
                                    {
                                            Value = Value;
                                    }
                                   if(j < 8)
                                    {
                                     Value = Value >>1;
                                   }
                                   k++;
                              }
                         Ir_Record[i] = Value;
                         Value = 0;
        }
          Decode_OK = 1;//处理完毕标志位置 1
}
```

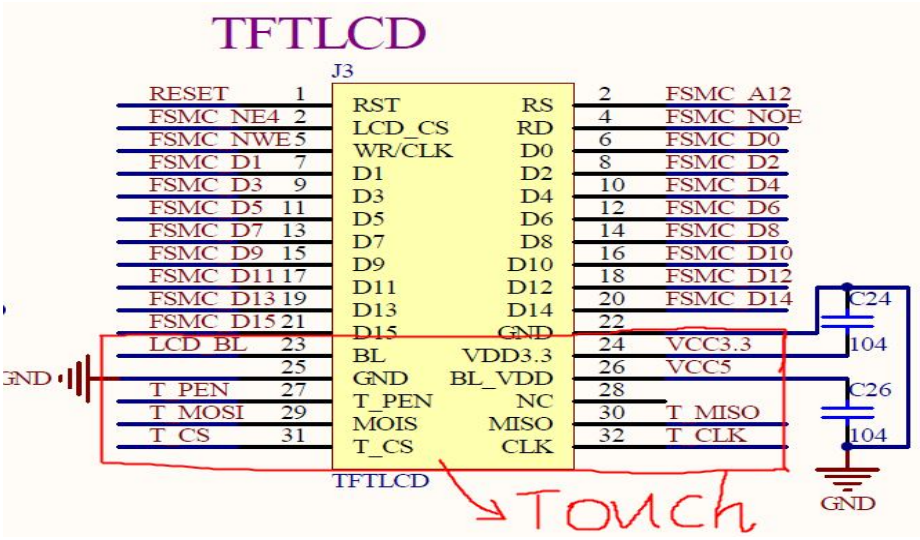
电路硬件设计红外的接口接于普通 IO 口，所以使用外部中断来检测红外数据信号。

3.18.3 实验操作与现象

程序下进去后，按下红外遥控，屏幕会显示相应的键值。

第十九章　触摸屏使用

3.19.1 硬件设计



3.19.2 核心代码解析

触摸按键扫描，type:0,屏幕坐标;1,物理坐标(校准等特殊场合用)

```
void TP_Scan(u8 type)
{
        Xup=0xffff;
     Yup=0xffff;
    if(PEN==0)//有按键按下
    {
        if(type)TP_Read_XY2(&x,&y);//读取物理坐标
        else if(TP_Read_XY2(&x,&y))//读取屏幕坐标
        {
            x=xfac*x+xoff;//将结果转换为屏幕坐标
            y=yfac*y+yoff;
        }
        Xdown=x;
        Ydown=y;
        time++;
    }else
    {
         if(time>2)
         {
                        Xup=x;
                Yup=y;
         }
         time=0;
         Xdown=0xffff;
      Ydown=0xffff;
    }
}
```

注意参数：Xup Xdown Yup Ydown ，up 为触摸按下后手抬起后返回的按键值，down 为触摸按下就返回的按键值。

3.19.3 实验操作与现象

程序下进去后，会检测触摸屏是否校准过，如没校准执行校准程序。校准后，进入写字板功能，可在屏幕书写任意笔画。

第二十章　图片显示 1-颜色点显示

3.20.1 硬件设计

使用 LCD。

3.20.2 核心代码解析

在液晶上画图（仅支持：从左到右，从上到下 or 从上到下，从左到右 的扫描方式!）

```
void picture_show(u16 xsta,u16 ysta,u16 width,u16 height,u8 scan,u8 *p)
{
    u32 i;
    u32 len=0;
    if((scan&0x03)==0)//水平扫描
    {
        LCD_AUTOScan_Dir(L2R_U2D);//从左到右,从上到下
        LCD_Set_Window(xsta,ysta,width,height);
        LCD_SetCursor(xsta,ysta);//设置光标位置

    }else   //垂直扫描
    {
        LCD_AUTOScan_Dir(U2D_L2R);//从上到下,从左到右
        LCD_Set_Window(xsta,ysta,width,height);
        LCD_SetCursor(xsta,ysta);//设置光标位置
    }
    LCD_WriteRAM_Prepare();     //开始写入 GRAM
    len=width*height;                   //写入的数据长度
    for(i=0;i<len;i++)
    {
        LCD_DATA=picture_getcolor(scan&(1<<4),p);
        p+=2;
    }
    LCD_Set_Window(0,0,lcd_width,lcd_height);
}
```

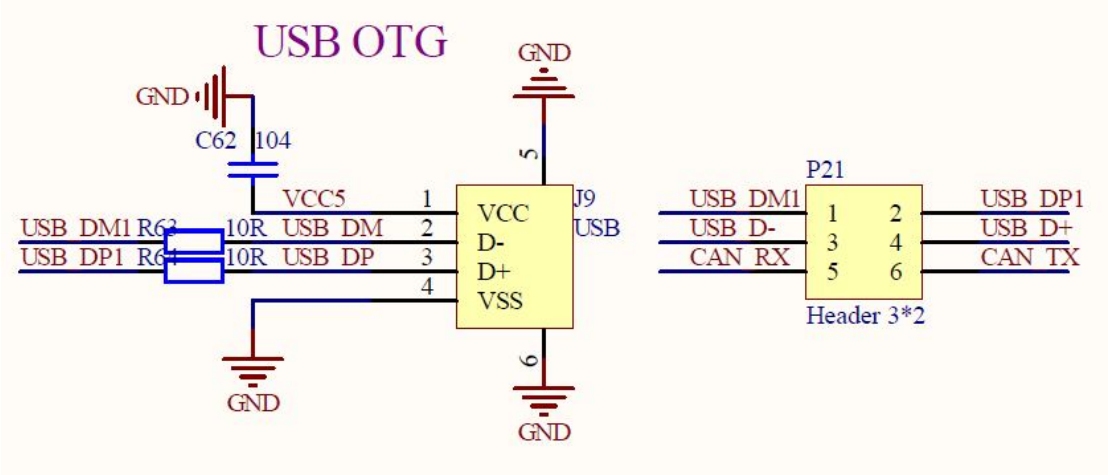
该例程使用的是像素点显示图片，设置地址自加，当写一个像素点颜色值进去之后，地址自加 1，然后接着写入下一个颜色值。由于使用的 RGB565，所以一个颜色值为两个字节。

3.20.3 实验操作与现象

程序下进去后，屏幕会显示MCU启明logo。Logo图片的颜色值存在logo[]数组里面。去图片颜色值软件在资料 常用软件\Image2Lcd 2.9(破解版) 里面。

第二十一章 USB U 盘(Host)

3.21.1 硬件设计



由于 USB 与 CAN 公用 IO 口所以在使用 USB 的时候需要将短路帽连接在 USB 端。 就是 P21 的 USB_DM1 接 USB_D-,USB_DP1 接 USB_D+。

3.21.2 核心代码解析

读 U 盘函数

```
* 名      称: u8 USBH_UDISK_Read(u8* buf,u32 sector,u32 cnt)
* 功      能：读 U 盘
* 入口参数：buf:读数据缓存区
            sector:扇区地址
            cnt:扇区个数
* 返回参数：错误状态;0,正常;其他,错误代码;
u8 USBH_UDISK_Read(u8* buf,u32 sector,u32 cnt)
{
    u8 res=1;
    if(HCD_IsDeviceConnected(&USB_OTG_Core)&&AppState==USH_USR_FS_TEST)/
```

```
/连接还存在,且是 APP 测试状态
    {
        do
        {
            res=USBH_MSC_Read10(&USB_OTG_Core,buf,sector,512*cnt);
            USBH_MSC_HandleBOTXfer(&USB_OTG_Core ,&USB_Host);
            if(!HCD_IsDeviceConnected(&USB_OTG_Core))
            {
                res=1;//读写错误
                break;
            };
        }while(res==USBH_MSC_BUSY);
    }else res=1;
    if(res==USBH_MSC_OK)res=0;
    return res;
}
```

写 U 盘函数

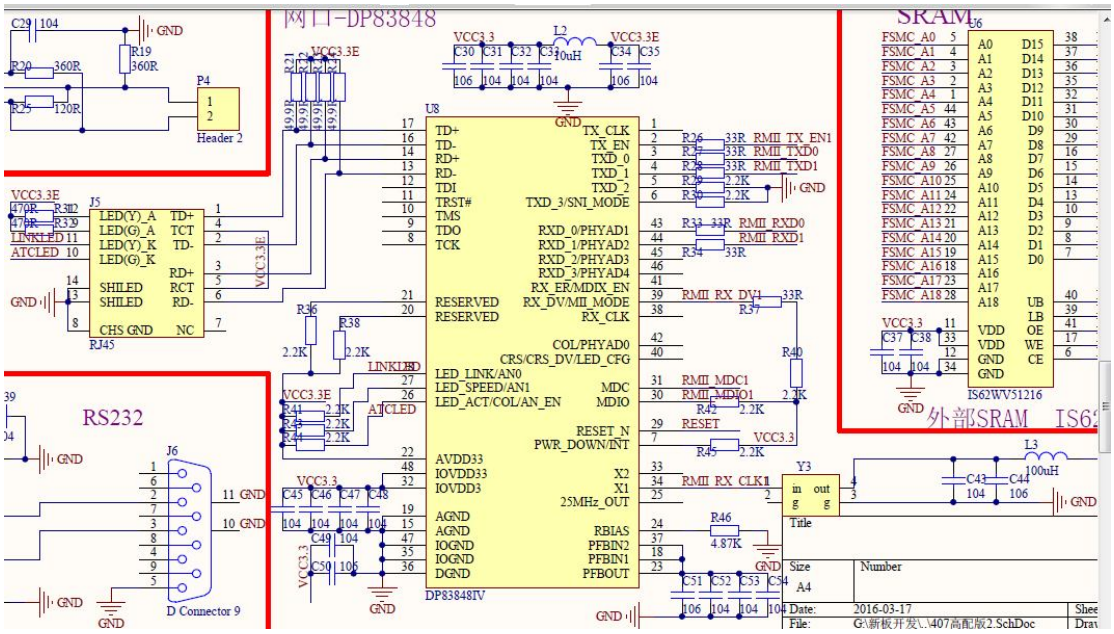
```
* 名      称: u8 USBH_UDISK_Write(u8* buf,u32 sector,u32 cnt)
* 功      能：写 U 盘
* 入口参数：buf:写数据缓存区
            sector:扇区地址
            cnt:扇区个数
* 返回参数：错误状态;0,正常;其他,错误代码;
u8 USBH_UDISK_Write(u8* buf,u32 sector,u32 cnt)
{
    u8 res=1;
    if(HCD_IsDeviceConnected(&USB_OTG_Core)&&AppState==USH_USR_FS_TEST)/
/连接还存在,且是 APP 测试状态
    {
        do
        {
            res=USBH_MSC_Write10(&USB_OTG_Core,buf,sector,512*cnt);
            USBH_MSC_HandleBOTXfer(&USB_OTG_Core ,&USB_Host);
            if(!HCD_IsDeviceConnected(&USB_OTG_Core))
            {
                res=1;//读写错误
                break;
            };
        }while(res==USBH_MSC_BUSY);
    }else res=1;
    if(res==USBH_MSC_OK)res=0;
    return res;
}
```

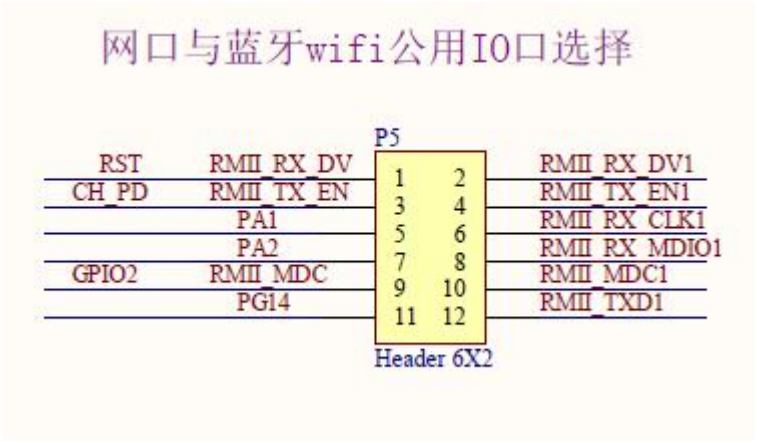
3.21.3 实验操作与现象

程序下进去后，屏幕会显示U盘的总容量。

第二十三章 TCP 服务器数据收发实验

3.23.1 硬件设计





由于网络与蓝牙和 WIFI 接口部分 IO 口公用，所以在使用网络需要把 P5 的短路帽**全部接上**。

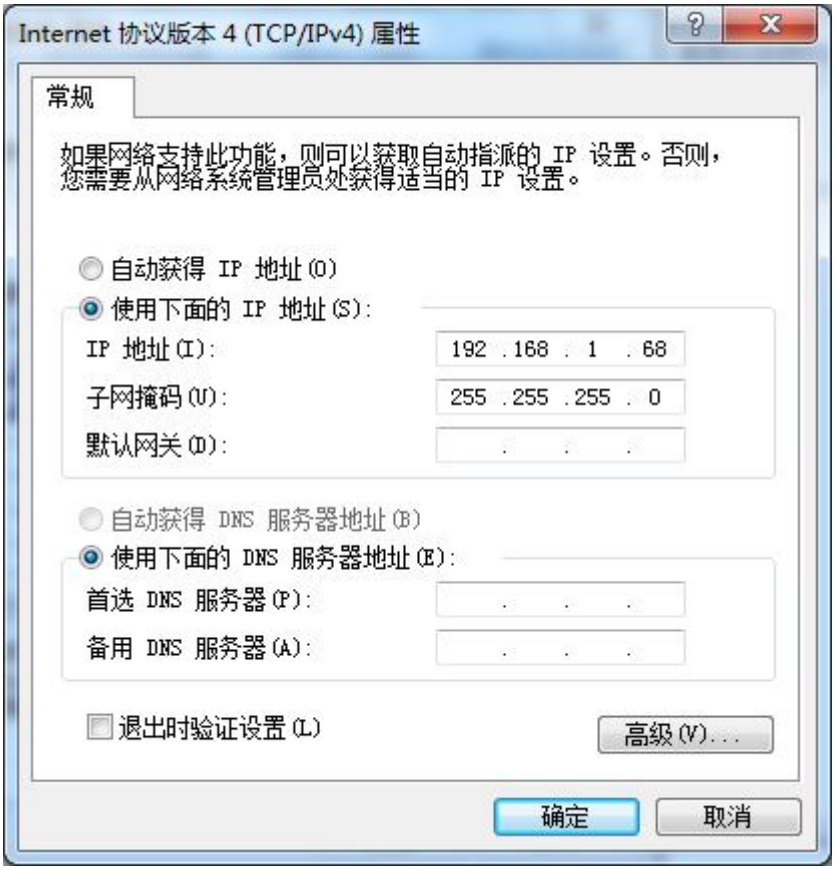
3.23.2 核心代码解析

请仔细看程序代码。

3.23.3 实验操作与现象

双排针P5的6个短路帽记得都接上去，程序下进去后，屏幕会显示开发板作为服务器的IP和端口号，此时：

1、设置电脑本地IP地址



由于是开发板作为服务器，电脑网络助手做客户端，所以客户端的设置的IP地址跟开发板作为服务器所下载的程序代码没联系，这边电脑本地IP地址后面的 68 ，用户可自行设置。这边设置 68 是要跟后续的例程统一。

注意：如果使用的是笔记本电脑，使用该例程要禁用无线网络。

2、打开网络助手设置

由于是**开发板**作为服务器，网络助手作为客户端，所以协议类型选择客户端，服务器IP设置跟开发板程序代码一致。程序代码IP为：192.168.1.240　端口号 2040。

3、实验想象

前面都设置好后，网络助手点击连接，网络助手就连接上开发板的服务器。之后点击发送。 此时就是客户端向服务器发送数据，服务器接到数据后，将接收到的数据原封不动的发回给客户端。

第二十四章 TCP 客户端数据收发实验

3.24.1 硬件设计

同第二十三章。

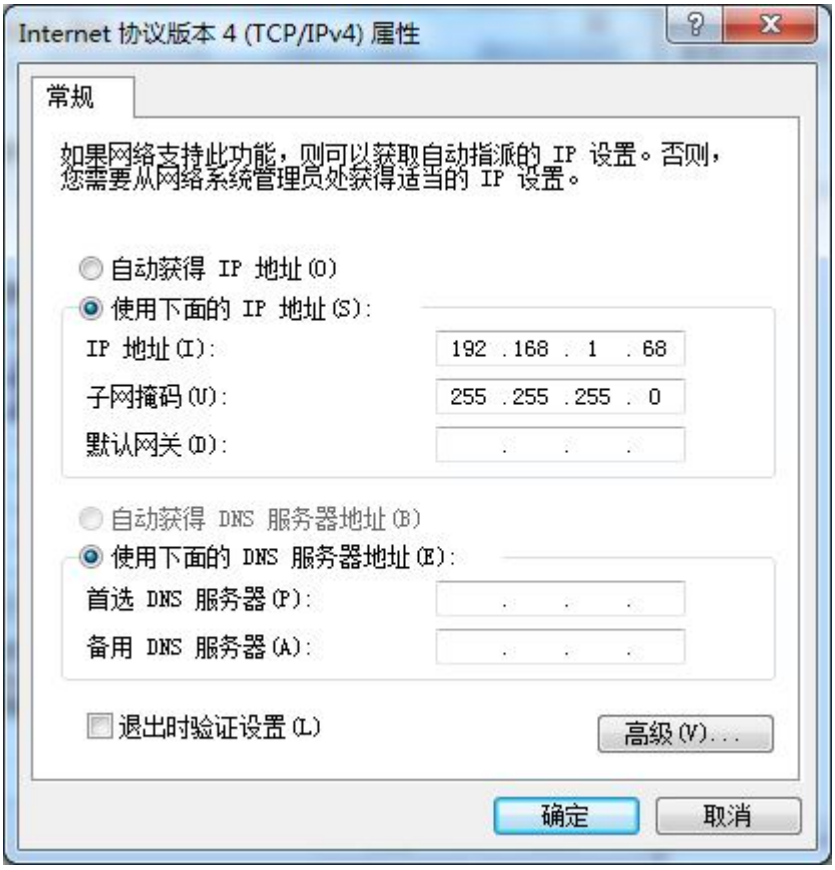
3.24.2 核心代码解析

请仔细看程序代码。

3.24.3 实验操作与现象

双排针P5的6个短路帽记得都接上去，程序下进去后，屏幕会显示开发板作为客户端的IP和端口号，网络助手作为服务器的IP(即电脑本地IP)与端口号，此时：

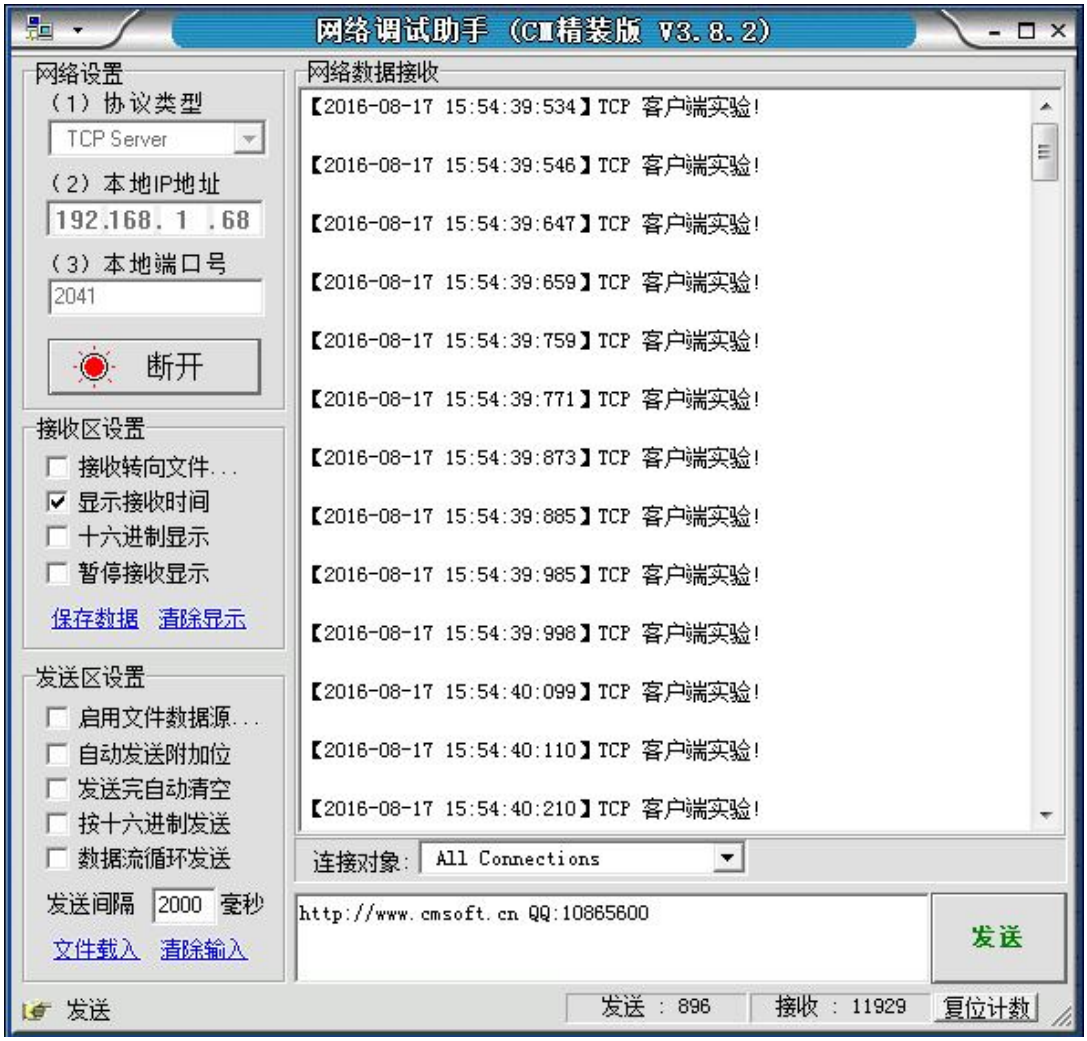
1、设置电脑本地IP地址



由于开发板的程序代码编写的服务器的IP为192.168.1.68，所以这里电脑作为服务器，IP就要设置跟程序一致。

注意：该例程下，如果使用的是笔记本电脑，可以不用禁用无线网络。

2、打开网络助手设置



由于是**网络助手**作为服务器，开发板作为客户端，所以协议类型选择服务器，服务器IP设置跟电脑本地一致。而程序代码设置需要连接到的服务器的IP为：192.168.1.68　端口号 2041，所以网络助手端口号填2041.

3、实验想象

前面都设置好后，网络助手点击连接，开发板就连接上网络助手的服务器。接着客户端的开发板就一直向服务器网络助手发送数据。

第二十五章 UDP 服务器数据收发实验

3.25.1 硬件设计

同第二十三章。

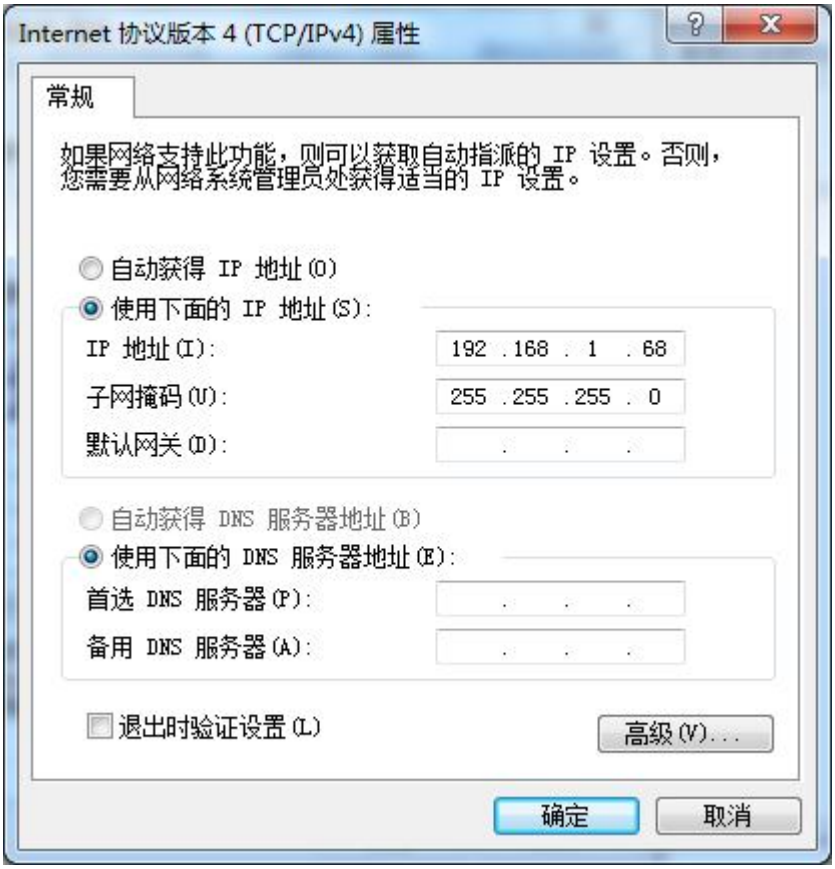
3.25.2 核心代码解析

请仔细看程序代码。

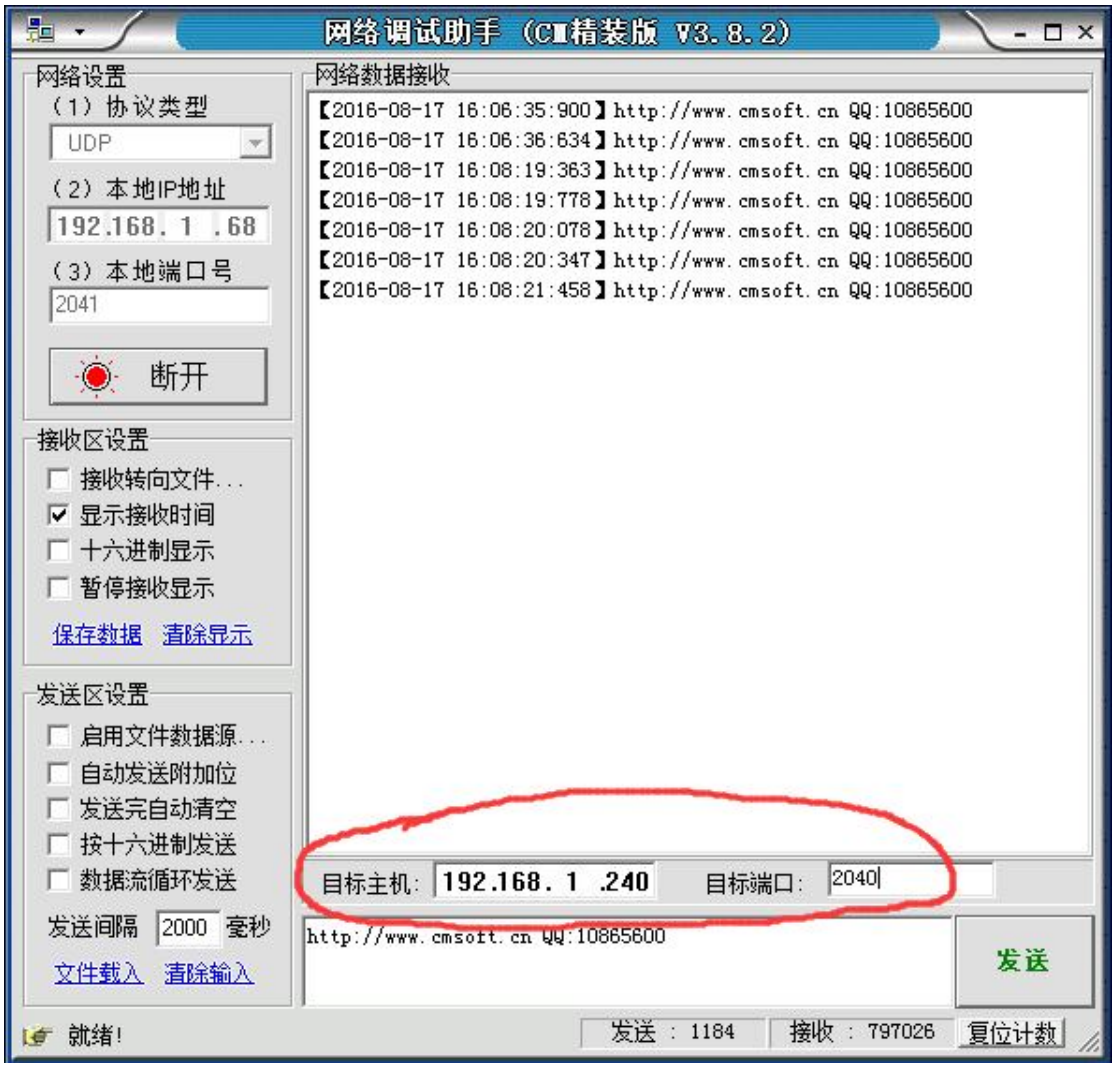
3.25.3 实验操作与现象

双排针P5的6个短路帽记得都接上去，程序下进去后，屏幕会显示开发板作为服务器的IP和端口号，此时：

1、设置电脑本地IP地址



2、打开网络助手设置



注意：红色圈圈里面的设置。目标主机为开发板。

3、实验想象

前面都设置好后，网络助手点击连接，网络助手就连接上开发板的服务器。之后点击发送。 此时就是客户端向服务器发送数据，服务器接到数据后，将接收到的数据原封不动的发回给客户端。

第二十六章 UDP 客户端数据收发实验

3.26.1 硬件设计

同第二十三章。

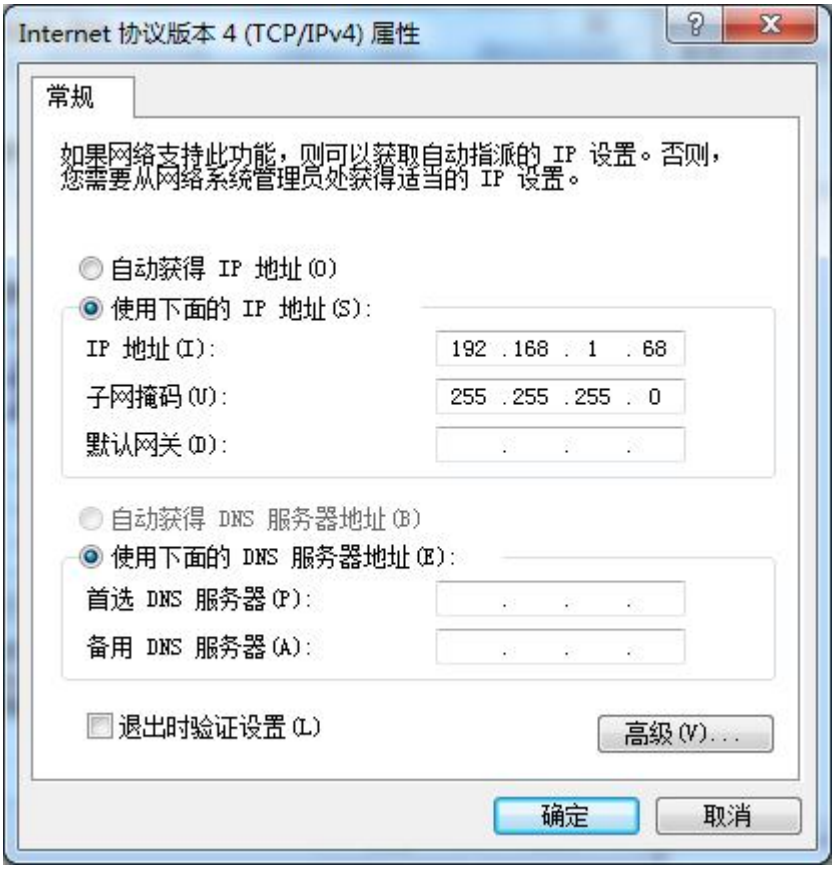
3.26.2 核心代码解析

请仔细看程序代码。

3.26.3 实验操作与现象

双排针P5的6个短路帽记得都接上去，程序下进去后，屏幕会显示开发板作为客户端的IP和端口号，网络助手作为服务器的IP(即电脑本地IP)与端口号，此时：

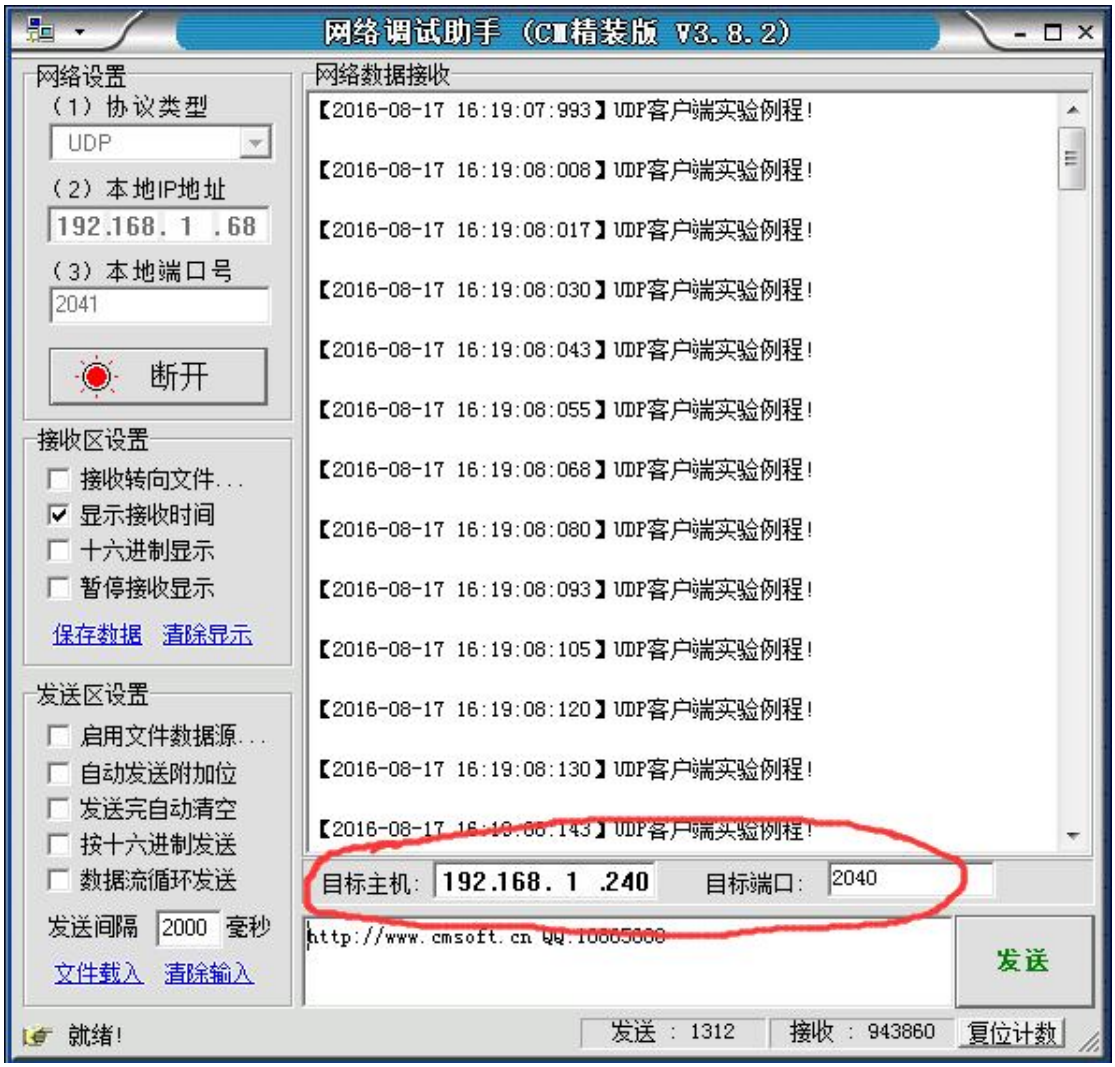
1、设置电脑本地IP地址



由于开发板的程序代码编写的服务器的IP为192.168.1.68，所以这里电脑作为服务器，IP就要设置跟程序一致。

注意：该例程下，如果使用的是笔记本电脑，可以不用禁用无线网络。

2、打开网络助手设置



注意：红色圈圈里面的设置。 目标主机为开发板。

3、实验想象

前面都设置好后，网络助手点击连接，开发板就连接上网络助手的服务器。接着客户端的开发板就一直向服务器网络助手发送数据。