**宏指令使用说明**

目录

[一、 宏指令编写以及调用 2](#_Toc11196)

[二、 库函数说明 5](#_Toc4719)

[三、 宏指令案例 9](#_Toc19292)

[001：定时功能案例 9](#_Toc29045)

[002：计时功能案例 10](#_Toc4524)

[003：画面跳转案例 11](#_Toc20543)

[004：用户权限更改案例 12](#_Toc30528)

[005：从站通信屏蔽案例 13](#_Toc24901)

[006：微型打印机案例 13](#_Toc12431)

[007：按钮宏案例 15](#_Toc23383)

[008、二控一案例 16](#_Toc3834)

[附件 17](#_Toc7758)

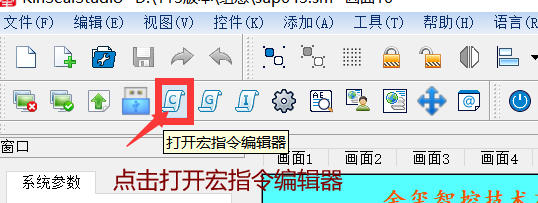
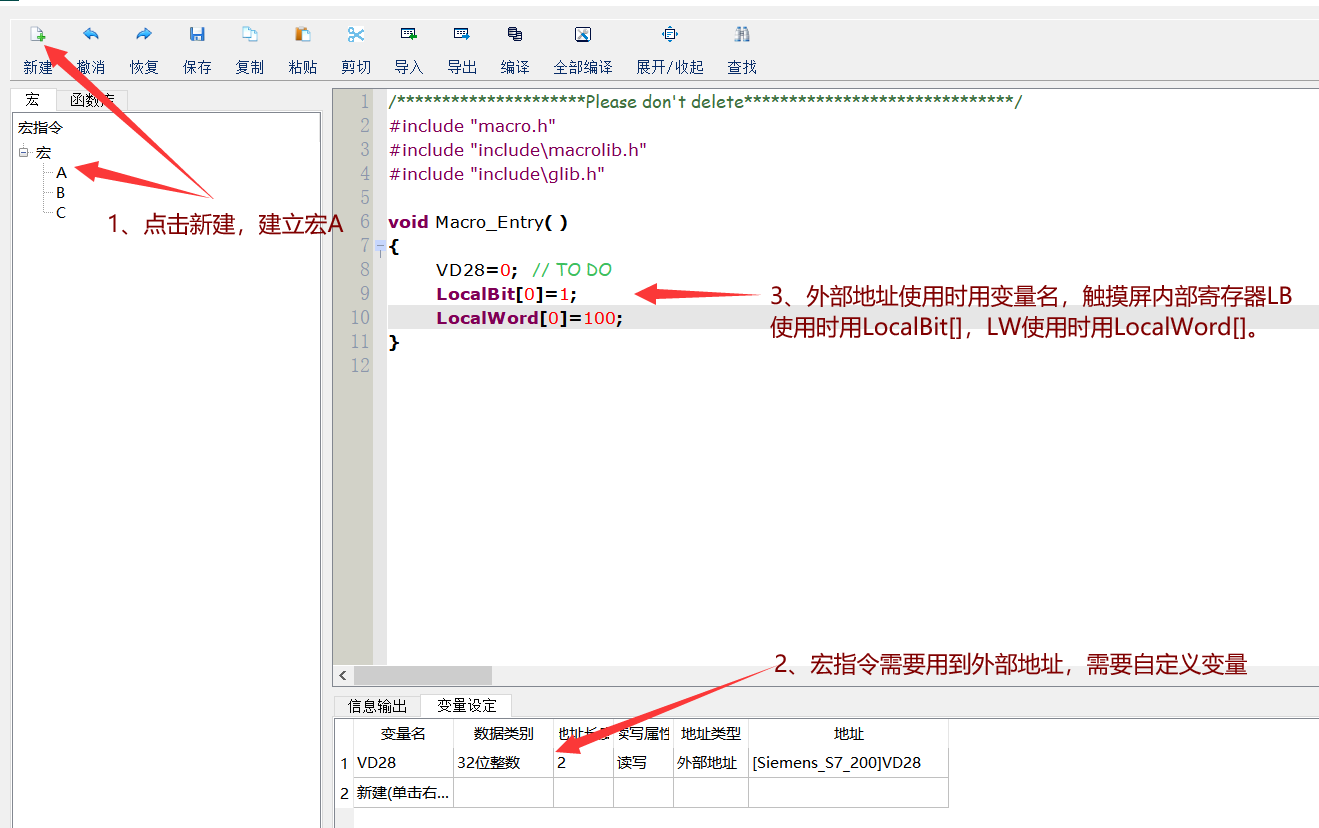
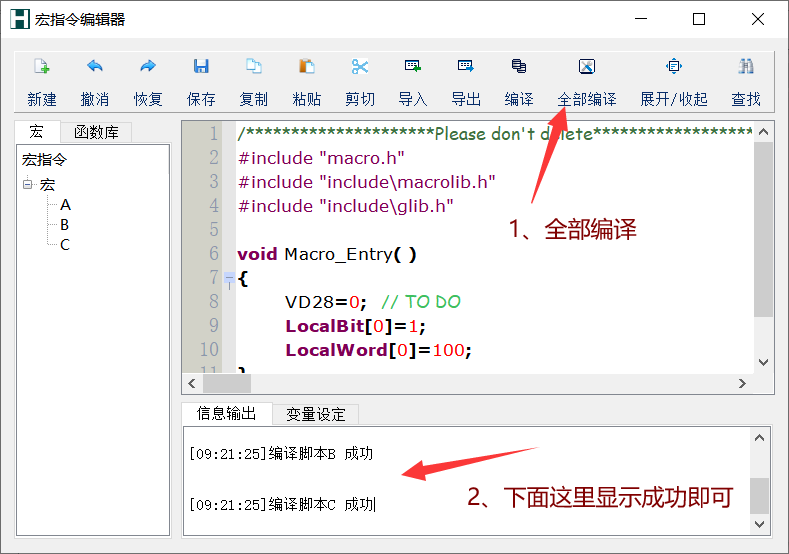
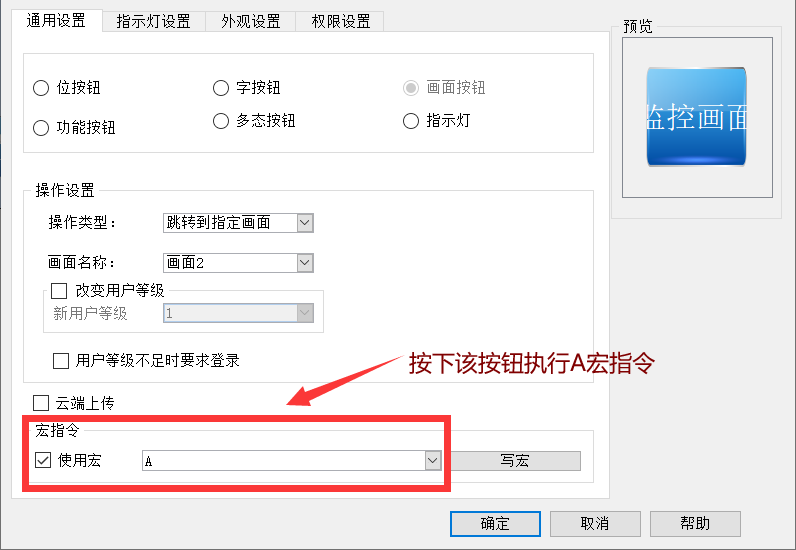
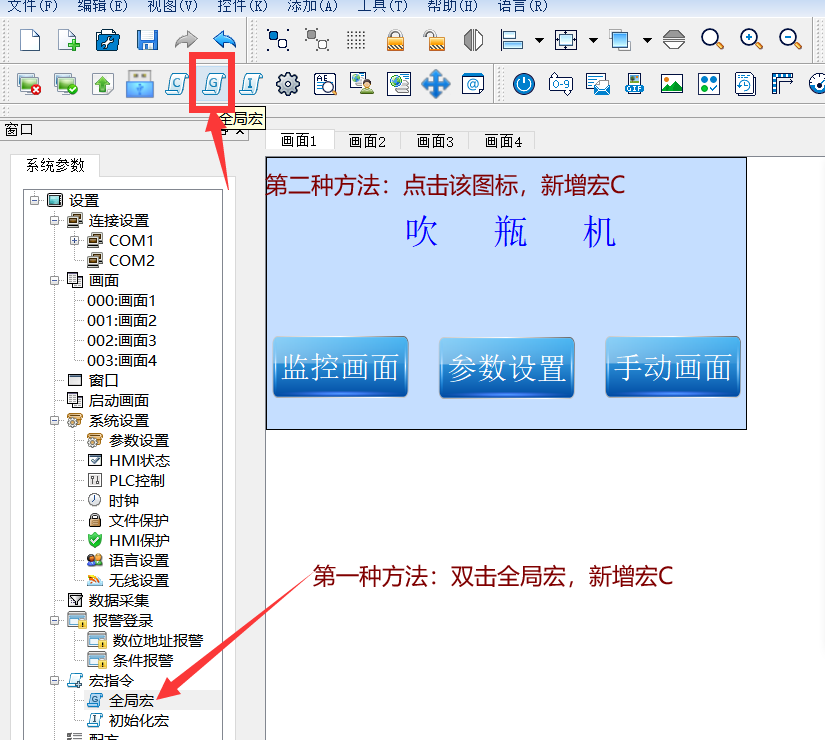
[四、自由口案例 17](#_Toc24332)

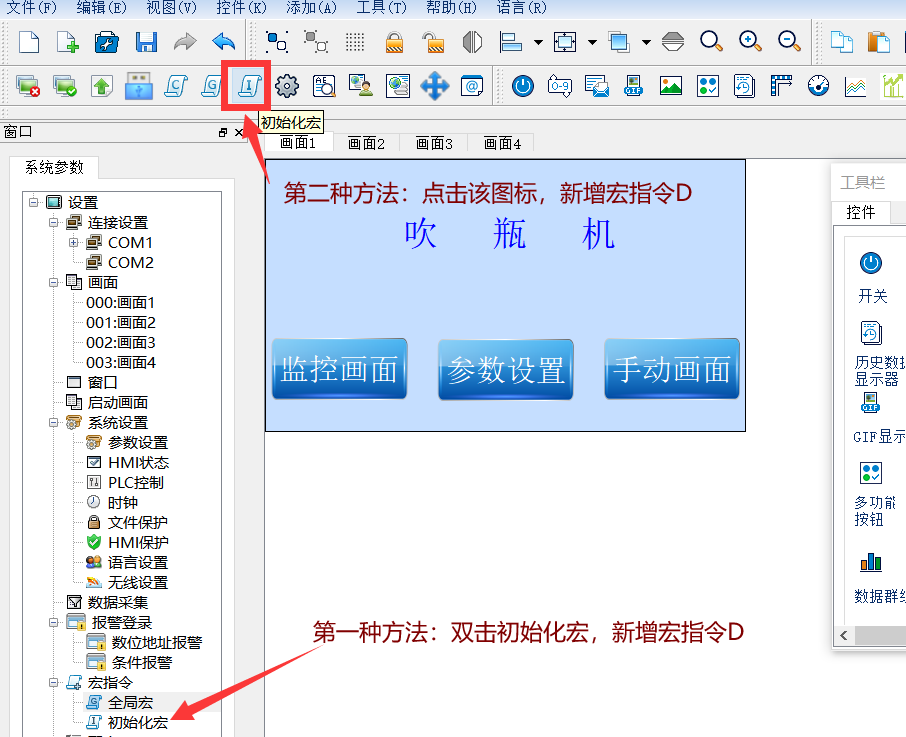
[附件 22](#_Toc27492)

[五、 宏指令注意事项 22](#_Toc31142)

[六、宏指令编译错误解决方法 22](#_Toc15296)

[附件： 27](#_Toc9341)

1. 宏指令编写以及调用
2. 打开宏指令编辑器  
    
3. 新建一个宏，使用到外部地址时建立变量，LB使用时用LocalBit,LW使用时用LocalWord  
    
4. 宏指令编写完成后，只写了一个宏指令时点击【编译】，多个宏指令时点击【全部编译】，编译成功后将宏指令编辑器关闭  
    
5. 宏指令调用  
    Ⅰ 按钮宏：点击按钮之后执行一次，在控件【属性】下面勾选【使用宏】，选择需要执行的宏  
      
    Ⅱ 画面宏：在指定画面才执行，画面【属性】新增宏  
      
    Ⅲ 全局宏：触摸屏上电后一直在执行，在全局宏调用  
    

Ⅳ 初始化宏：触摸屏上电后只执行一次，在初始化宏调用  
 

1. 库函数说明

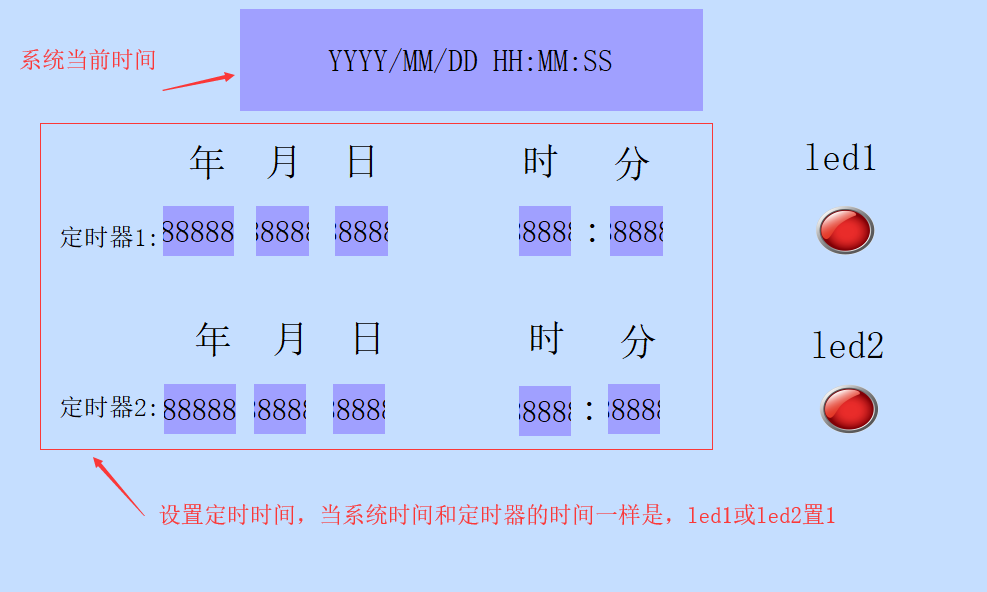
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **序号** | | **函数库** | **功能说明** | | **例子** |
| **01** | | **[ADDSUM](" \l "ADDSUM)** | 计算N个数值的和 | | char data[5];//定义五个字符  data[0]=0x1;  data[1]=0x2;  data[2]=0x3;  data[3]=0x4;  data[4]=0x20;//给字符data[0]~data[4]赋值  short checksum;//定义短整型数checksum  checksum=ADDSUM(data,5);  //checksum=data[0]~data[4]五个字符的和  // checksum=0x2a; |
| **02** | | **[ASCII2DEC](" \l "ASCII2DEC)** | 将字符串转换为十进制数 | | char source[4] ；//定义4个字符  Source[0]='2';  Source[1]='3';  Source[2]='4';  Source[3]='5';  //给字符source[2]~source[5]赋值  short result；//定义短整型数checksum  result=ASCII2DEC(source,4)；  //将source[2]~source[5]字符串转成10进制数给result  // result=2345 |
| **03** | | **[ASCII2FLOAT](" \l "ASCII2FLOAT)** | 将字符串转换为浮点数 | | char source[4] ；//定义4个字符  Source[0]=‘5’;  Source[1]=‘.’;  Source[2]=‘7’;  Source[3]=‘8’;  //给字符source[2]~source[5]赋值  float result；//定义浮点数型result  result=ASCII2FLOAT(source,4)；  //将source[0]~source[3]字符串转成浮点数给result  // result = 5.78 |
| **04** | | **[ASCII2HEX](" \l "ASCII2HEX)** | 将字符串转换为十六进制数 | | char source[5] ；//定义4个字符  Source[0] = ‘1’;  Source[1] = ‘6’;  Source[2] = ‘3’;  Source[3] = ‘4’;  //给字符source[0]~source[3]赋值  short result；//定义整数型result  result=ASCII2HEX(source,4)；  //将source[0]~source[3]字符串转成16进制数给result  // result = 0x1634 |
| **05** | | **[BCD2BIN](" \l "BCD2BIN)** | 将BCD值转换为十进制数 | | short source = 0x2648；  //将BCD值0x2648给整数型source  short result；//定义整数型result  result=BCD2BIN(source)； //将source的BCD值转成十进制值给result  // result = 2648 |
| **06** | | **[BIN2BCD](" \l "BIN2BCD)** | 将十进制数转换为BCD值 | | short source = 1234； //将十进制值1234给整数型source  short result；//定义整数型result  result=BIN2BCD(source)；  //将source的十进制值转成BCD值给result  // result = 0x1234 |
| **07** | | **[CRC](" \l "CRC)** | 计算出16位CRC校验值 | | char source[5] ；//定义5个字符  Source[0] = 0x1;  Source[1] = 0x2;  Source[2] = 0x3;  Source[3] = 0x4;  Source[4] = 0x5;  //给字符source[0]~source[4]赋值  short bit\_CRC；//定义整数型bit\_CRC  bit\_CRC=CRC(source,5)；  //计算16位CRC校验值 |
| **08** | | **[DEC2ASCII](" \l "DEC2ASCII)** | 将十进制数值转换为字符串 | | short source = 5678；  //将数值5678给整数型source  char \*result[4]；  //定义4个连续字符result[0]~result[3]  result= DEC2ASCII(result,source,4)；  //将source十进制值转换为字符串  // result[0] == '5', result[1] == '6', result[2] == '7', result[3] == '8 |
| **09** | | **[FLOAT2ASCII](" \l "FLOAT2ASCII)** | 将浮点数转换为字符串 | | float source = 56.8;  //将数值56.8给浮点数型source  unsigned char result[4];  //定义无符号字节型 result[0]~result[3]  result=FLOAT2ASCII(source,result);  //将浮点数source转换为字符串result  //result[0] = '5',result[1] = '6',  result[2] = '.',result[3] = '8 |
| **10** | | **[GETBIT](" \l "GETBIT)** | 获取指定位的值，值为0或1 | | short source = 0x5, bit\_pos = 0 ,result；  result = GETBIT(source,bit\_pos)；  // result == 1 |
| **11** | | **[GETCHARS](" \l "GETCHARS)** | 从通信端口获取数据，该函数仅适用于自由口通信 | | char Data[20]；  int PortID,Result；  PortID=0；  Result=GETCHARS(PortID,Data)； |
| **12** | | **[HEX2ASCII](" \l "HEX2ASCII)** | 将十六进制数值转换为字符串 | | short source = 0x5678；  //将数值0x5678给整数型source  char \*result；  //定义4个连续字符result[0]~result[3]  HEX2ASCII(source,result)；  //将source十六进制值转换为字符串  //result[0] = '5', result[1] = '6', result[2] = '7', result[3] = '8' |
| **13** | | **[HIBTYE](" \l "HIBYTE)** | 取一个字的高字节 | | short source = 0x1234,result；  result=HIBYTE(source)；// result = 0x12 |
| **14** | | **[HIWORD](" \l "HIWORD)** | 取一个双字的高字 | | int source = 0x12345678,result；  result=HIWORD(source)；// result = 0x1234 |
| **15** | | **[INVBIT](" \l "INVBIT)** | 将一直指定的值取反 | | short source = 0x6, bit\_pos =1 ,result；  result=INVBIT(source, bit\_pos)；  // result = 4 |
| **16** | | **[LOBTYE](" \l "LOBYTE)** | 取一个字的低字节 | | short source = 0x1234,result；  result= LOBYTE(source)；// result = 0x34 |
| **17** | | **[LOWORD](" \l "LOWORD)** | 取一个双字的低16位的字 | | int source = 0x12345678,result；  result=LOWORD(source)；// result == 0x5678 |
| **18** | | **[PUTCHARS](" \l "PUTCHARS)** | 将数据发送到通信端口，该函数仅适用于自由口通信 | | char Data[5]；  Data[0] = 0x02;  Data[1] = 0x30;  Data[2] = 0x31;  Data[3] = 0x4d;  Data[4] = 0x5e;  char PortID；  unsigned char Length;  Length=5；  PortID=0；  PUTCHARS(PortID,Data, Length)； |
| **19** | | **[SWAPB](" \l "SWAPB)** | 交换一个字的高低位字节 | | short source = 0x1234,result；  //将十六进制值0x1234给source,定义整数型result  result=SWAPB(source)；  //将source的高低位互换并赋值给result  // result = 0x3412 |
| **20** | | **[SWAPW](" \l "SWAPW)** | 交换一个双字的高低字 | | int source = 0x12345678, ；  ////将十六进制值0x12345678给双字source  int result；//定义整数型result  result=SWAPW(source)；  //将source的高低位互换并赋值给result  // result = 0x56781234 |
| **21** | | **[SETBIT](" \l "SETBIT)** | 设置指定位的值为设定值 | | short source = 0x4；  short bit\_pos = 2, result；  result = SETBIT(source,bit\_pos,1)；  // result = 4 |
| **22** | | **[XORSUM](" \l "XORSUM)** | 用于计算N个数值的异或值 | | char source[5] ;  Source[0] = 0x1;  Source[1] = 0x20;  Source[2] = 0x3;  Source[3] = 0x48;  Source[4] = 0x5;  short checksum;  checksum = XORSUM(source, 5); / ecksum =0x6f; |
| **23** | | **[DELAY](" \l "DELAY)** | 设置延迟时间，单位为毫秒 | | DELAY(1000);//延迟1000毫秒 |
| **24** | | **[GETBUFFERLENGTH](" \l "GETBUFFERLENGTH)** | 从通信端口获取缓冲区长度，该函数仅适用于自由口通信 | | int bufferlen,PortID; PortID=0; bufferlen=GETBUFFERLENGTH(PortID) |
| **25** | | **[CLEARBUFFER](" \l "CLEARBUFFER)** | 清除通讯端口的缓冲区，该函数仅适用于自由口通信 | | int Clearresult,PortID; PortID=0; Clearresult=CLEARBUFFER(PortID); |
| **26** | | **[ScreenOpen](" \l "ScreenOpen)** | 跳转画面 | | ScreenOpen(N);  // 跳转到序号为N的画面，N为常数1、2、3…… |
| **27** | | **[DISABLELCD](" \l "DISABLELCD)** | 点亮液晶显示屏 | | DISABLELCD(); |
| **28** | | **[ENABLELCD](" \l "ENABLELCD)** | 关闭液晶显示屏 | | ENABLELCD(); |
| **序号** | **图形库** | | | **功能说明** |  |
| **01** | **[PutPixel](" \l "PutPixel)** | | | 绘制像素点 | int x = 2;  int y= 4;  int c = COLOR\_BLACK;  PutPixel(x,y,c); |
| **02** | **[DrawLine](" \l "DrawLine)** | | | 绘制线段 | int x1 =2,x2=400;  int y1=4,y2=200;  int color = COLOR\_NAVY;  DrawLine(x1,y1,x2,y2,color); |
| **03** | **[DrawHLine](" \l "DrawHLine)** | | | 绘制水平线 | int x1 =2,x2=400;  int y= 200;  int color = COLOR\_DCYAN;  DrawHLine(x1,x2,y,color); |
| **04** | **[DrawVLine](" \l "DrawVLine)** | | | 绘制垂直线 | int y1 =2,y2=200;  int x= 100;  int color = COLOR\_DGREEN;  DrawVLine(y1,y2,x,color); |
| **05** | **[Rectangle](" \l "Rectangle)** | | | 绘制矩形 | int x1 =2,x2=200;  int y1=4,y2=100;  int color = COLOR\_MAROON;  Rectangle(x1,y1,x2,y2,color); |
| **06** | **[FiledRectangle](" \l "FilledRectangle)** | | | 绘制填充矩形 | int x1 =2,x2=100;  int y1=4,y2=100;  int color = COLOR\_MAROON;  FilledRectangle(x1,y1,x2,y2,color); |
| **07** | **[Circle](" \l "Circle)** | | | 绘制圆 | unsigned int x0 =200;  unsigned int y0= 150;  unsigned int r=50;  int color = COLOR\_DGREEN;  Circle(x0,y0,r,color); |
| **08** | **[Fill\_Circle](" \l "Fill_Circle)** | | | 绘制填充圆 | int x0=100;  int y0=100;  int r0= 50;  int color =COLOR\_YELLOW;  Fill\_Circle(x0,y0,r0,color); |

三、宏指令案例  
**注：案例工程组态可在底部附件另存，也可在官网下载**

**官网下载地址：http://www.kinseal.com/download.asp?cd=11&cid=19&bg=xz**

**001：定时功能案例：**实现在触摸屏时间达到指定时间后，进行内部地址LB0的地址值反转。

1. 新建工程，做一个定时组态，如下图



（2）根据组态编辑定时器宏指令作定时判断，当年月日时分和当前时间一样时，指示灯显示为1，反之为0

宏指令编辑代码如下；

void Macro\_Entry( )

{

// TO DO

int year,month,day,hour,min;

//从内部特定地址中获取当前年、月、日、时、分、秒时间

year = BCD2BIN(LocalWord[3925]);

month = BCD2BIN(LocalWord[3924]);

day = BCD2BIN(LocalWord[3923]);

hour = BCD2BIN(LocalWord[3922]);

min = BCD2BIN(LocalWord[3921]);

//判断第一段定时时间是否到达

if(year == LocalWord[3000] && month == LocalWord[3001] && day == LocalWord[3002] &&\

hour == LocalWord[3003] &&min ==LocalWord[3004] )

{

//定时时间到，亮led1；

//注：led1是变量名，地址按需要设置

led1 = 1;

}

else

led1 = 0;

//判断第二段定时时间是否到达

if(year == LocalWord[3010] && month == LocalWord[3011] && day == LocalWord[3012] &&\

hour == LocalWord[3013] &&min ==LocalWord[3014] )

{

//定时时间到，亮led2；

//注：led2是变量名，地址按需要设置

led2 = 1;

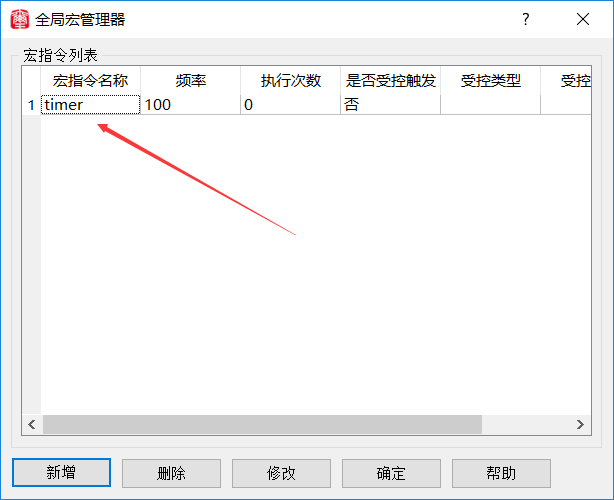
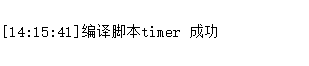
}

else

led2 = 0;

}

（3）代码编辑完成后点击右上角全部编译，下方提示编译成功时关闭窗口，然后然后在全局宏调用，如下图



1. 完成后下载组态到触摸屏即可看到定时效果

**002：计时功能案例：**实现触摸屏在达到累计时间后，进行报警提示。

1. 新建工程，做一个计时器，如下图



（2）根据组态编辑计时器宏指令作计时判断，设定总时间（单位：秒），当达到计时总时间时触发报警

宏指令编辑代码如下；

void Macro\_Entry( )

{

short NowHour,NowSec,NowWeek;

//读取RTC实时时间的秒数

NowSec = BCD2BIN(LocalWord[3920]);

//启动按钮按下

if((LocalBit[100]==1))

{

//如果RTC秒数发生变化，秒数递增记录一秒

if(NowSec != LocalWord[1])

{

//计算累计时间

LocalWord[2]++;

//给秒数标志位赋值

LocalWord[1] = NowSec;

LocalWord[1010]--;

//达到计时总时间

if((0==LocalWord[1010]))

{

//复位启动按钮

LocalBit[100]=0;

//在数位地址报警中设定了LB[10]作为报警地址

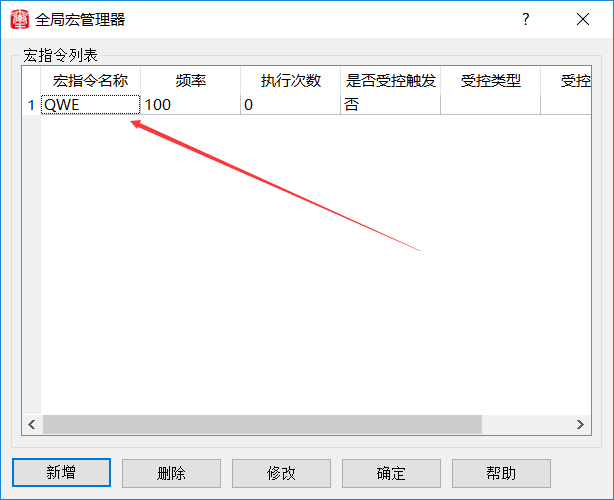
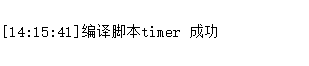
LocalBit[10]=1;

}

}

}

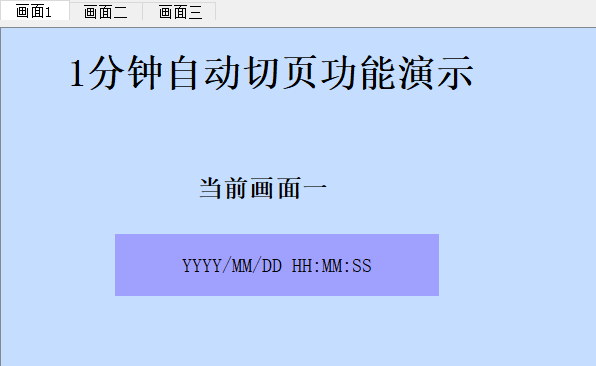
（3）代码编辑完成后点击右上角全部编译，下方提示编译成功时关闭窗口，然后在全局宏调用，如下图



（4）完成后下载组态到触摸屏即可看到定时效果

**003：画面跳转案例：**实现1分钟自动进行一次画面跳转。

1. 新建工程，建立多个画面，如下图



（2）根据组态编辑画面自动切换宏指令，实现1分钟自动进行一次画面跳转，

宏指令编辑代码如下；

void Macro\_Entry( )

{

short NowHour,NowSec,NowWeek;

//读取RTC实时时间的秒数

NowSec = BCD2BIN(LocalWord[3920]);

//如果RTC秒数发生变化，秒数递增记录一秒

if(NowSec != LocalWord[1])

{

//计算累计时间

LocalWord[2]++;

//给秒数标志位赋值

LocalWord[1] = NowSec;

//达到计时总时间,60秒

if((60==LocalWord[2]))

{

//计数标志清零

LocalWord[2] = 0;

//页面计数加1

LocalWord[3]++;

//调用函数库里面的函数，打开指定的画面

ScreenOpen(LocalWord[3]);

//当页面计数为2时，清零

if(2 == LocalWord[3])

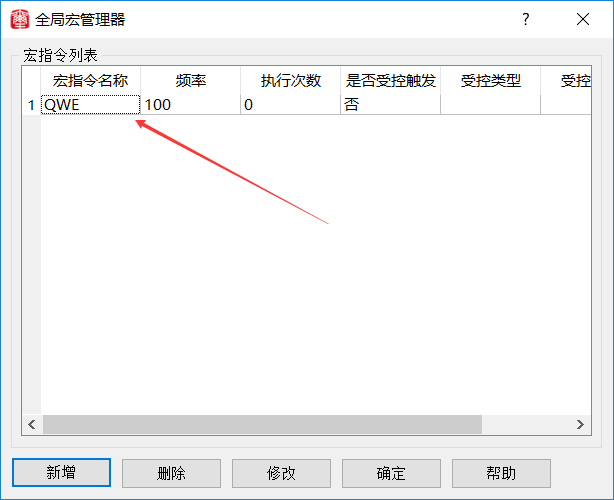
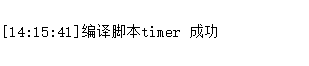
LocalWord[3] = 0;

}

}

}

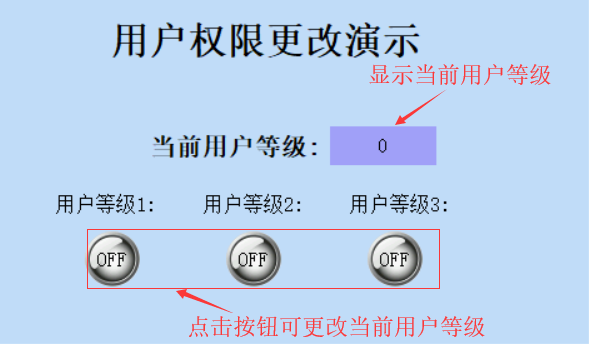
（3）代码编辑完成后点击右上角的全部编译，下方提示编译成功时关闭窗口，然后在全局宏调用



（4）完成后下载组态到触摸屏即可看到定时效果

**004：用户权限更改案例：**实现在不同条件下，触摸屏自动切换用户等级。

1. 新建工程，做一个组态，三个位按钮，用来使用宏控制用户等级，一个数值显示器，显示当前用户等级



（2）根据组态编辑宏指令实现不同条件下自动切换用户等级，

宏指令编辑代码如下；新建三个宏指令

//Change\_usr1（第一个宏指令）

void Macro\_Entry( )

{

//注意:在"系统设置->plc控制->控制当前权限等级"中，已经设定LW10用来控制用户等级

LocalWord[10] = 1;s

}

//Change\_usr2（第二个宏指令）

void Macro\_Entry( )

{

//注意:在"系统设置->plc控制->控制当前权限等级"中，已经设定LW10用来控制用户等级

LocalWord[10] = 2;

}

//Change\_usr3（第三个宏指令）

void Macro\_Entry( )

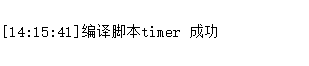
{

//注意:在"系统设置->plc控制->控制当前权限等级"中，已经设定LW10用来控制用户等级

LocalWord[10] = 3;

}

（3）代码编辑完成后点击右上角的全部编译，下方提示编译成功时关闭窗口，



1. 每个按钮都各自使用一个宏，实现点击该按钮时更改用户等级



**005：从站通信屏蔽案例：**在一主多从用法时，可以选择屏蔽某些从站的通信。

1. 新建工程，做一个一组多从的工程



（2）做好组态后进行通讯，改变LW4004的值即可控制是否屏蔽某个从站

**006：微型打印机案例：**配合荣达系列微型打印机，打印指定内容。

1. 新建工程，做一个按钮，做打印按钮用



（2）编辑宏指令实现控制打印机打印指定的打印内容，

宏指令编辑代码如下；

void Macro\_Entry( )

{

char str[50];

char \*newptr,\*oldptr;

short source;//=LocalWord[3];

char result[4];

// char \*result=[4];

// short source =LocalWord[4] ;

// short source =LocalWord[3] ;

//把系统内部时间地址定义变量

//把PLC里的值传送给触摸屏系统内部地址

//LocalWord[1]=num8;

//LocalWord[2]=num1;

//LocalWord[3]=num2;

//LocalWord[4]=num3;

//LocalWord[5]=num4;

//LocalWord[6]=num5;

// LocalWord[7]=num6;

//LocalWord[8]=num7;

short NowYear ,NowMon,NowDay, NowHour,NowMin,NowSecond;

NowYear =BCD2BIN(LocalWord[3925]);

NowMon = BCD2BIN(LocalWord[3924]);

NowDay=BCD2BIN(LocalWord[3923]);

NowHour =BCD2BIN(LocalWord[3922]);

NowMin = BCD2BIN(LocalWord[3921]);

NowSecond=BCD2BIN(LocalWord[3920]);

LocalWord[9]=NowYear ;

LocalWord[10]=NowMon;

LocalWord[11]=NowDay;

LocalWord[12]=NowHour;

LocalWord[13]=NowMin;

LocalWord[14]=NowSecond;

if(LocalBit[100]==1)

{

newptr=(char\*)&LocalWord[1090];

sprintf(str,"\*\*\*\*\*");

oldptr=(char\*)&str;

memcpy(newptr,oldptr,5);

LocalWord[1095]=10;

LocalWord[1096]=10;

newptr=(char\*)&LocalWord[1090];

PUTCHARS(0,newptr,12);

//LocalWord[1102]=10;

newptr=(char\*)&LocalWord[1070];

sprintf(str,"额定转速");

oldptr=(char\*)&str;

memcpy(newptr,oldptr,8);

LocalWord[1078]=10;

newptr=(char\*)&LocalWord[1070];

PUTCHARS(0,newptr,26);

sprintf(str,"%d%02d%02d%02d%02d%02d ",NowYear,NowMon,NowDay,NowHour,NowMin,NowSecond);

newptr=(char\*)&LocalWord[1000];

oldptr=(char\*)&str;

memcpy(newptr,oldptr,14);

LocalWord[1015]=10;

newptr=(char\*)&LocalWord[1000];

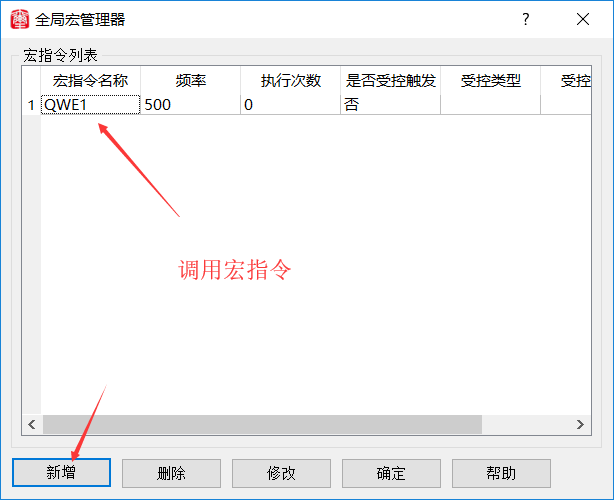
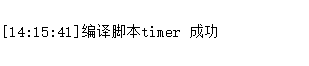
PUTCHARS(0,newptr,50);

LocalBit[100]=0;

}

}

（3）代码编辑完成后点击右上角的全部编译，下方提示编译成功，在全局宏调用



（4）下载组态到触摸屏，就可以控制打印机打印指定内容

**007：按钮宏案例：**实现一组按钮为互斥关系，即同一时刻，只能有一个按钮的值为1。

1. 新建工程，在画面上放置三个按钮



（2）编辑宏指令实现三个按钮互斥，只能有一个按钮的值能为1，

宏指令编辑代码如下；新建三个宏指令

//Change\_buttton1（第一个宏指令）

void Macro\_Entry( )

{

//第一个button按下，复位其他两个button

LocalBit[1] = 0;

LocalBit[2] = 0;

}

//Change\_buttton2（第二个宏指令）

void Macro\_Entry( )

{

//第二个button按下，复位其他两个button

LocalBit[0] = 0;

LocalBit[2] = 0;

}

//Change\_buttton3（第三个宏指令）

void Macro\_Entry( )

{

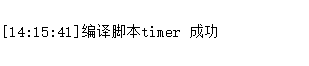
//第三个button按下，复位其他两个button

LocalBit[0] = 0;

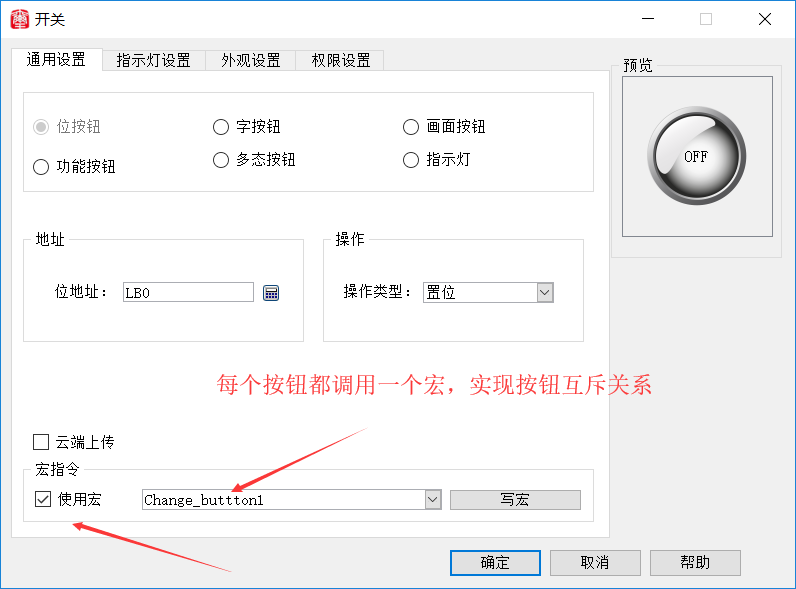
LocalBit[1] = 0;

}

（3）代码编辑完成后点击右上角的全部编译，下方提示编译成功时关闭窗口，

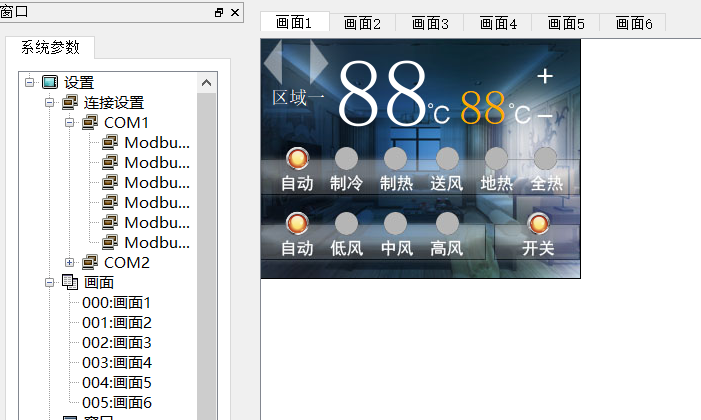


（4）给三个按钮都各自使用一个宏



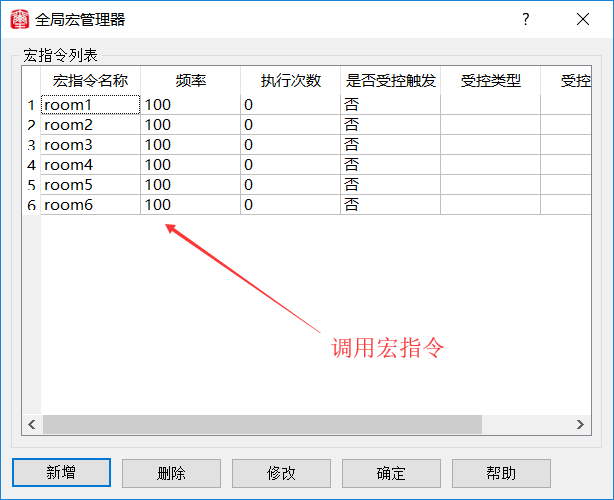
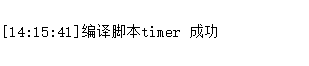
008、**二控一案例：**一个上位机、一个触摸屏与一个控制器，上位机与触摸屏同时可以控制控制器

1. 新建工程，做一个一组多从的组态，



1. 根据需求编辑宏指令，宏指令见附件组态（08、二控一案例）

（3）代码编辑完成后点击右上角的全部编译，下方提示编译成功，然后在全局宏调用



（4）最后把组态下载到触摸屏即可查看功能效果

附件：  
 



四、自由口案例

**客户实际应用需求：**

**第一种宏，点动按键宏，按键每按一次，发送1条指令；**

**第二种宏，置位按键宏，按键按下，发一次指令，按键弹起需对方发指令弹起；**

**第三种宏，数据显示器宏，接收第三方指令数值。**

**2、实际应用自由协议发送帧及回传帧指令格式：共5个字节，第一个触摸屏地址(自定义），第二个按键模式（LW\LB），第三个按键ID，第四、五个是参数，global\_run宏，主要是返回给触摸屏数值，LW01-05**

**第一：str[0]=0x1; //触摸屏地址**

**第二：str[1]=0x00; //LW01 LB00**

**第三：str[2]=51; //按键ID**

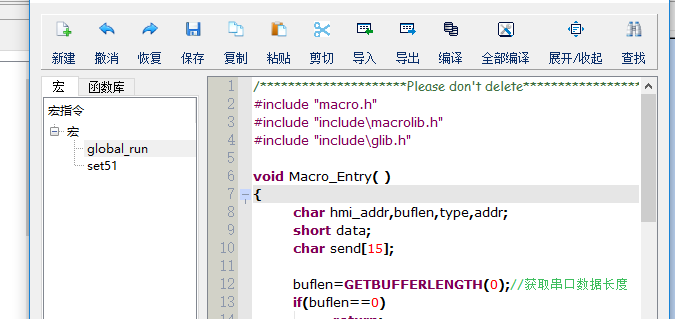
**第四：str[3]=0x01; //数据低字节**

**第五：str[4]=0x00; //数据高字节**

**根据客户需求，按以下步骤实现简单的演示功能**

**硬件环境：**一个上位机、一个触摸屏与一个串口调试助手与一个串口仿真线

1. 新建工程，并新建两条宏指令，如下图所示(global\_run和set51)：



global\_run代码：

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Please don't delete\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "macro.h"

#include "include\macrolib.h"

#include "include\glib.h"

void Macro\_Entry( )

{

char hmi\_addr,buflen,type,addr;

short data;

char send[15];

buflen=GETBUFFERLENGTH(0);//获取串口数据长度

if(buflen==0)

return;

GETCHARS(0,send);//从串口中读取数据

hmi\_addr=send[0];

type=send[1];

addr=send[2];

data=send[3]\*256+send[4]; //数据

LocalWord[0]=data;//收到串口数据后，把收到的数据传给字地址，同时数值显示器显示出来

LocalBit[1]=0;//收到串口数据后，把置位开关弹起

CLEARBUFFER(0);//清空串口缓存

}

set51代码：

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Please don't delete\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "macro.h"

#include "include\macrolib.h"

#include "include\glib.h"

void Macro\_Entry( )

{

char str[5];

str[0]=0x1; //触摸屏地址

str[1]=0x00; //LW01 LB00

str[2]=51; //按键地址

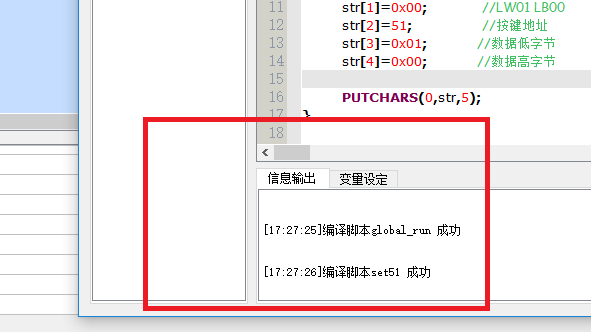
str[3]=0x01; //数据低字节

str[4]=0x00; //数据高字节

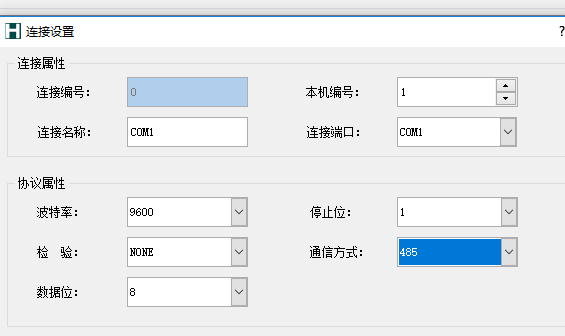
PUTCHARS(0,str,5); //把自由协议数据发送到串口

}

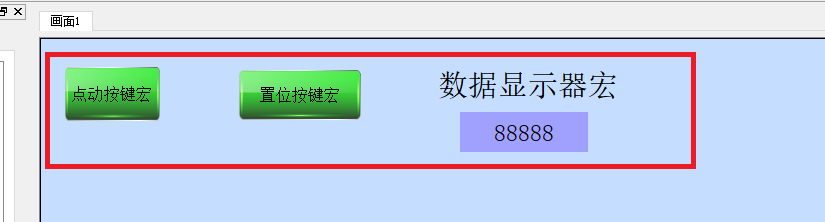
写好代码后点击编译全部，正常情况下，会提示两个宏脚本都编译通过



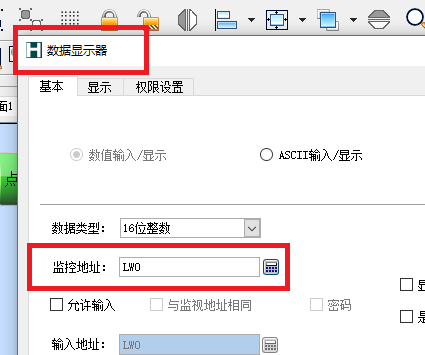
下载至触摸屏，触摸屏就会根据设定的串口1参数进行通讯



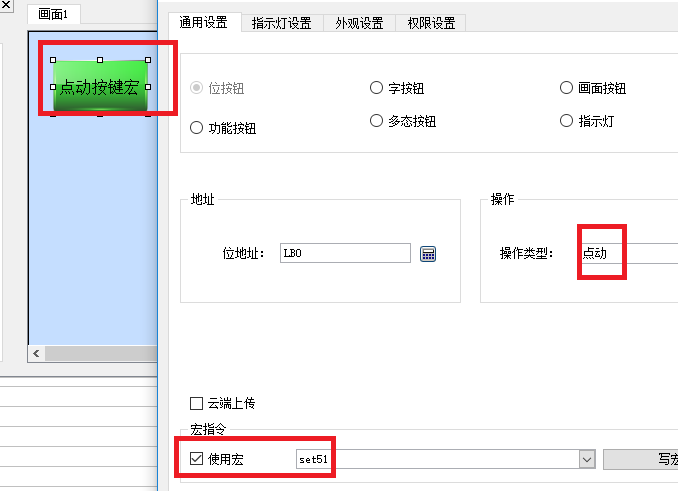
在画面上拖放如下三个控件：



数据显示器属性设置如下：



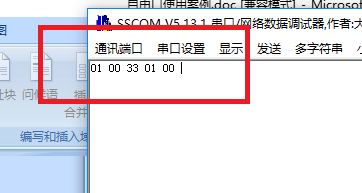
因为开关上调用了宏set51,因此在下载到触摸屏后，点击一下开关，就会向串口1发送一条数据，因为按钮属性是点动的，所以发送完后会自动弹起



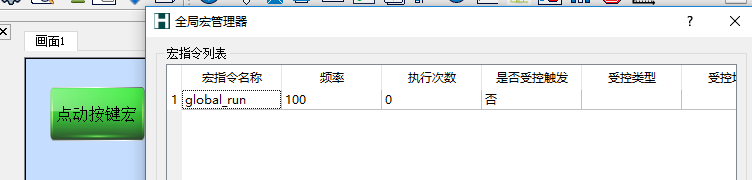
因为开关上调用了宏set51,因此在下载到触摸屏后，点击一下开关，就会向串口1发送一条数据，因为按钮属性是置位的，所以按钮不会自动弹起



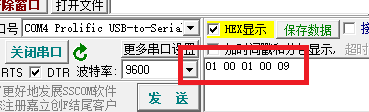
点击按钮发送送后，串口调试工具将会收到来自两个按钮的串口数据，



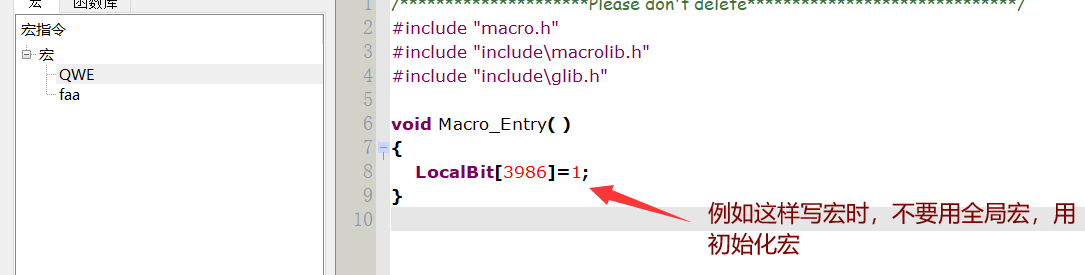
因为在全局宏中添加了global\_run这个宏指令，所以触摸屏初始化完成后就会间隔100ms执行一次宏指令，从串口中读取数据，并处理数据



接着向串口回传如下图数据时，置位按钮将因收到指令而自动弹起，数据显示器宏也会显示收到的数据9



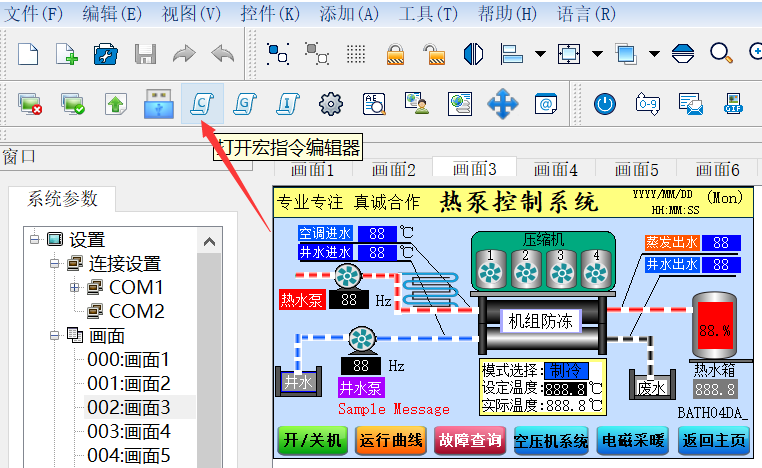
**附件：**

1. 宏指令注意事项  
   1、宏指令最多建立100条  
   2、每个宏指令自定义变量最多25个  
     
   3、将某个位置1时，不能频繁使用全局宏，只限于调用，不能一直置1，如果一直置1，会会一直写flash,影响存储寿命，导致死机  
   

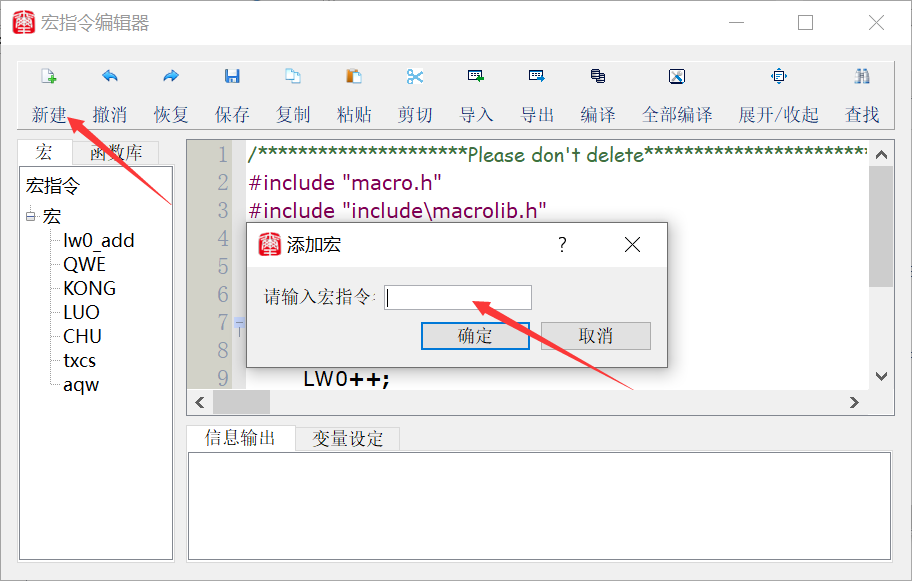
六、宏指令编译错误解决方法

1、安装好Kinseal Studio触摸屏软件后，一般我们需要重启一下电脑

2、重启电脑后打开Kinseal Studio触摸屏软件，新建一个工程，点击扩展工具栏上的打开宏指令编译器（如下图操作）

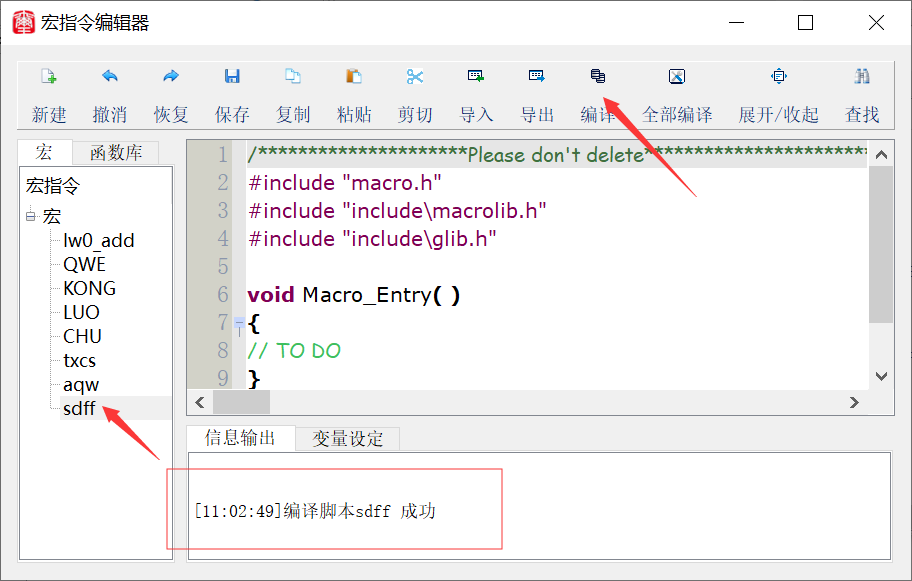


3、打开宏指令编译器后，新建宏指令，给宏指令命名后点击确定

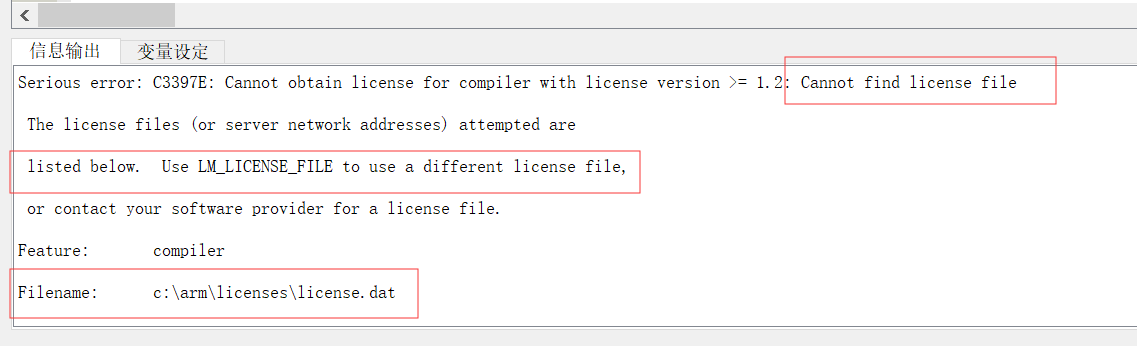


4、新建好宏指令后，先不要写宏指令语句，（如下图）操作，点击编译，如果

宏指令可以编译成功，说明我们软件宏指令编译路径在电脑系统环境变量里运行正常。



5、新建一个宏指令，点击宏指令编译器菜单上的编译，如果编译框里提示编译失败（如下图所示信息）

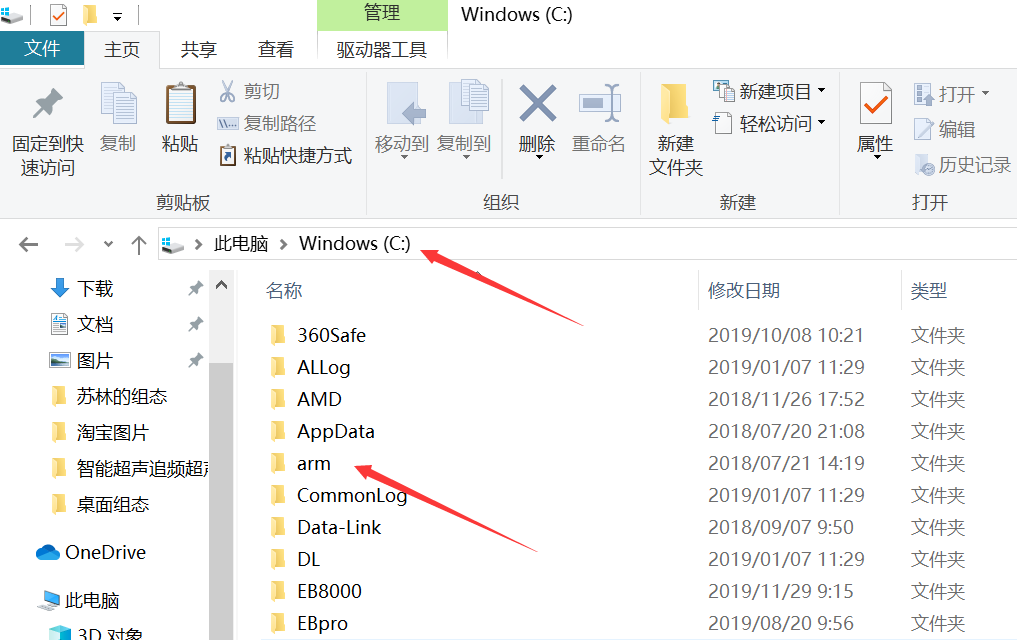


解决方法如下

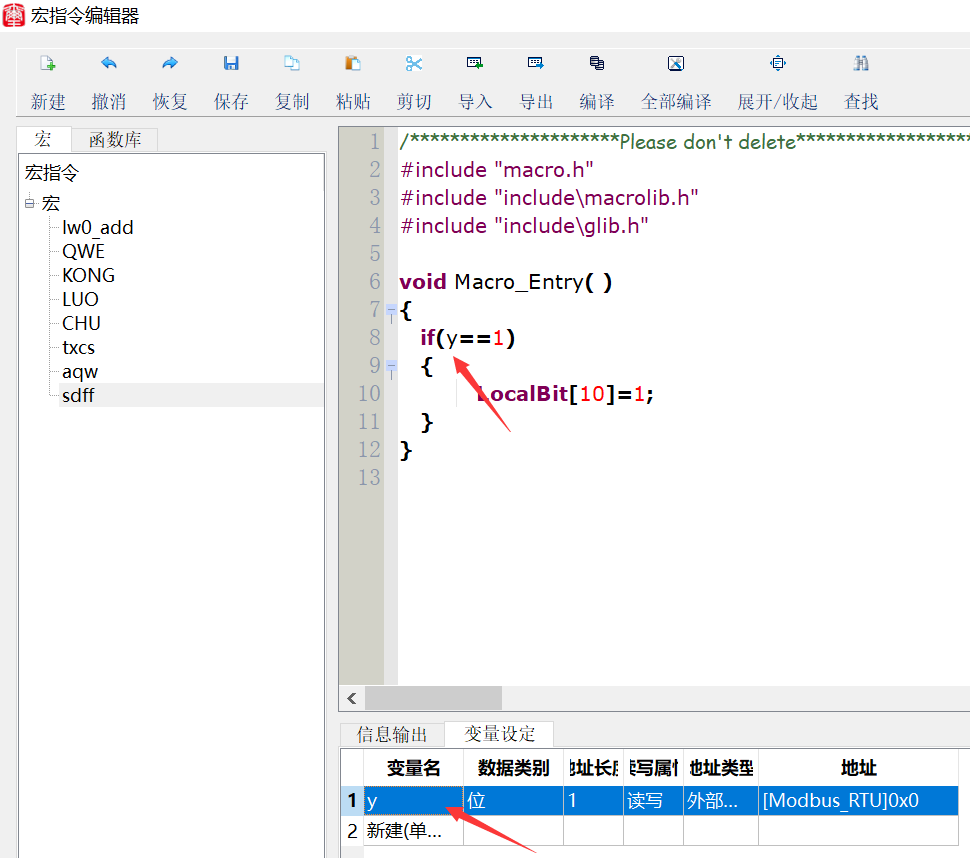
1. 右击Kinseal Studio触摸屏软件选择属性，然后点击兼容性是否勾选以管理员身份运行，如果没有勾选请勾选上(如下图操作）

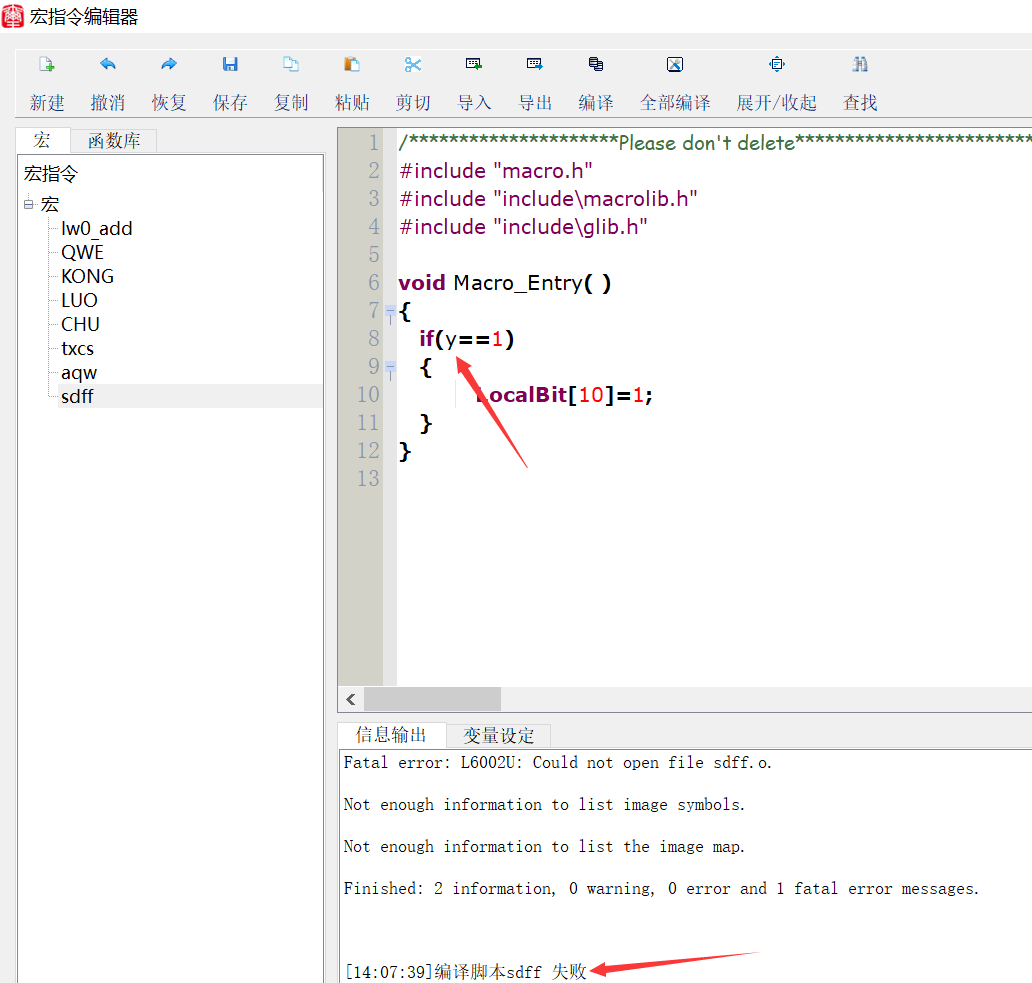


1. 需要将arm文件夹复制到C盘根目录里，arm文件夹本文档有自带。复制好之后重新打开Kinseal Studio触摸屏软件再编译宏指令



（3）在写宏指令时注意语法，变量名，标点符号、括号等是否使用正确？变量名建立变量名不要用特殊的单个字母或者数字，否则编译不成功

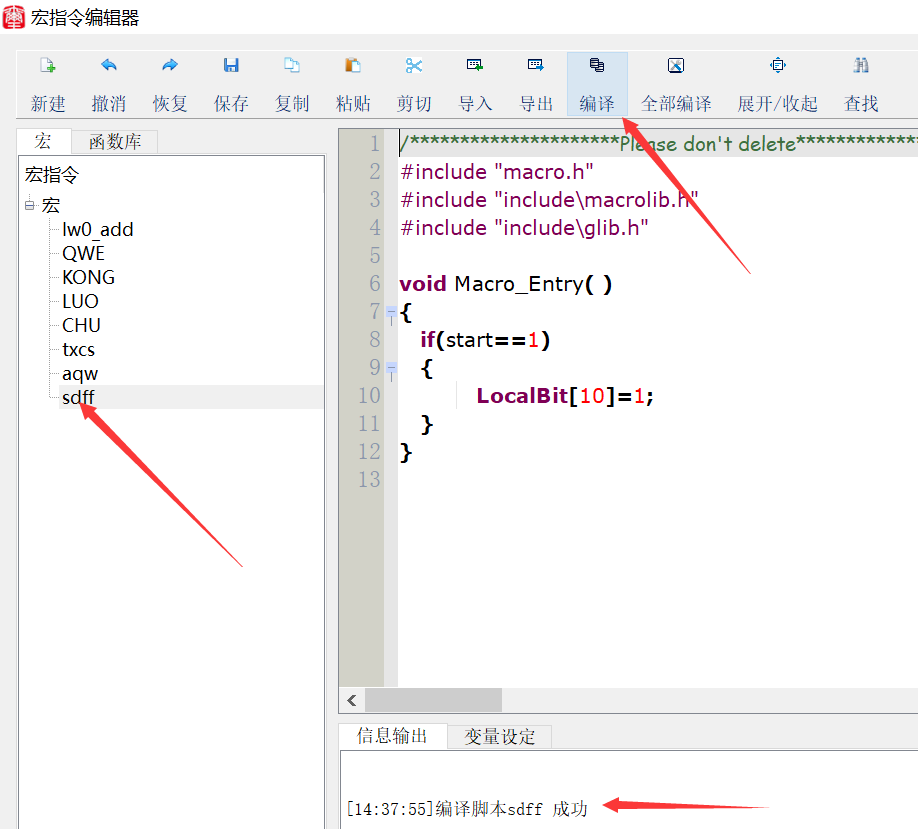




6、在下载工程时，下载编译框里提示（如下图）我们打开宏指令编译器，找到sddff这条宏指令，然后点击编译



7、宏指令编译成功后关闭宏指令编译器窗口再下载工程（如下如操作）



8、如果出现一个工程宏指令之前是可以编译成功的，在没有修改任何宏指令的情况下，偶然出现了宏指令编译失败，把软件关了重新打开工程再编译

**附件：**