零线电子科技有限公司

HOLTEK 开发板使用指南

基于 BS84C12A-3 触摸单片机

零线电子科技

2017/9/29

Zero Line 零线电子科技

从合泰触摸单片机开发环境搭建，到单片机各模块的编程，带你快速入门合泰触摸单片机开发。更有空气炸锅和煮水器实战项目，后期陆续更新。

目录

1 如何使用本书

1.1 内容介绍

本书先介绍合泰触摸单片机开发环境的搭建，到开发环境介绍， 再结合开发板上的资源，实战练习 BS84C12 单片机各资源的使用编程。

1.2 本书配套开发板

开发板资源：

- 基于 BS84C12A-3 单片机、使用官方触摸库的开发。
- 触摸按键：12 个电容式触摸按键，按键灵敏度软件可调。
- 软件模拟 UART ,控制 TM1652 数码管驱动芯片，驱动 4 位数码管，亮度可调。
- 软件模拟 IIC，控制时时钟芯(RTC)BL5372，并且利用纽扣电池，系统断电，时间不丢失。
- 12Bit ADC 转换例程。
- 内部 EEPROM 用于保存程序参数。
- USB 口使用 5V 系统供电。
- ICP 下载口引出，方便程序烧写。

1.3 技术支持

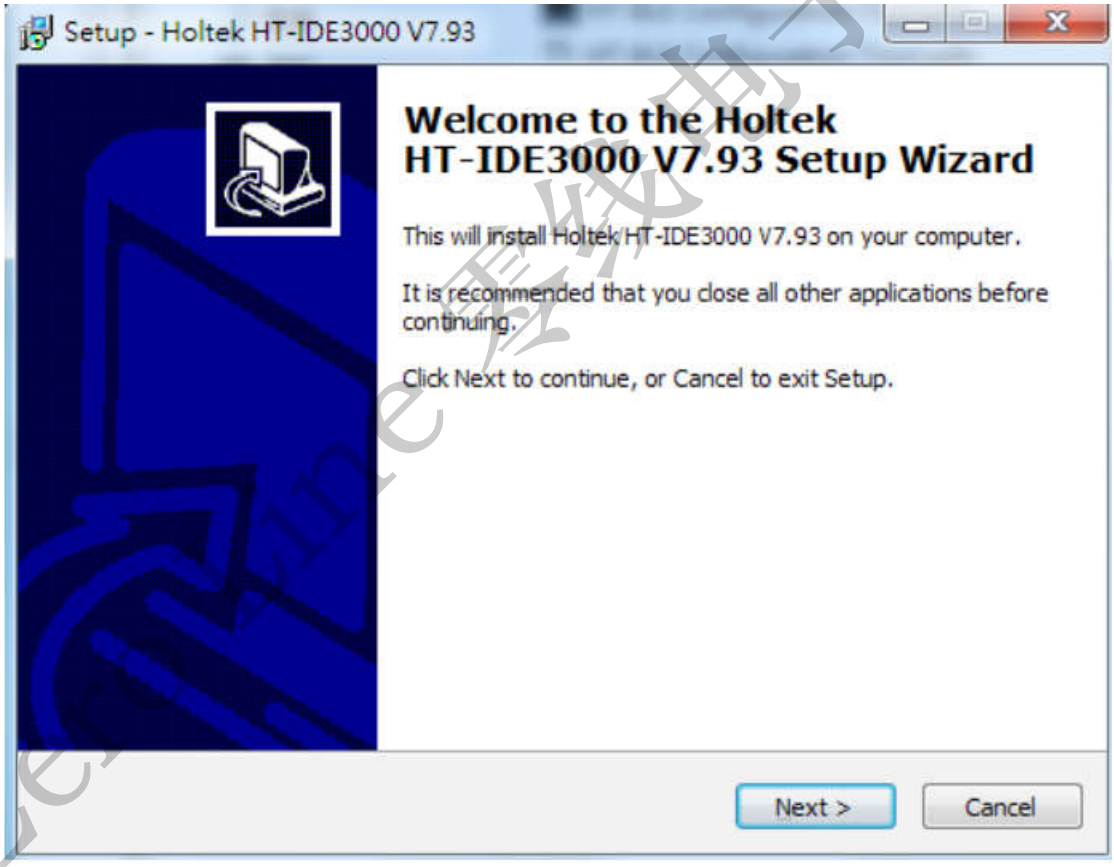
- QQ：261613460
- 微信：156 6960 8328
- 淘宝:https://shop545934258.taobao.com/shop/view_shop.htm?shop_id=545934258
- Blog：http://blog.csdn.net/qq236106303

2 .开发环境搭建

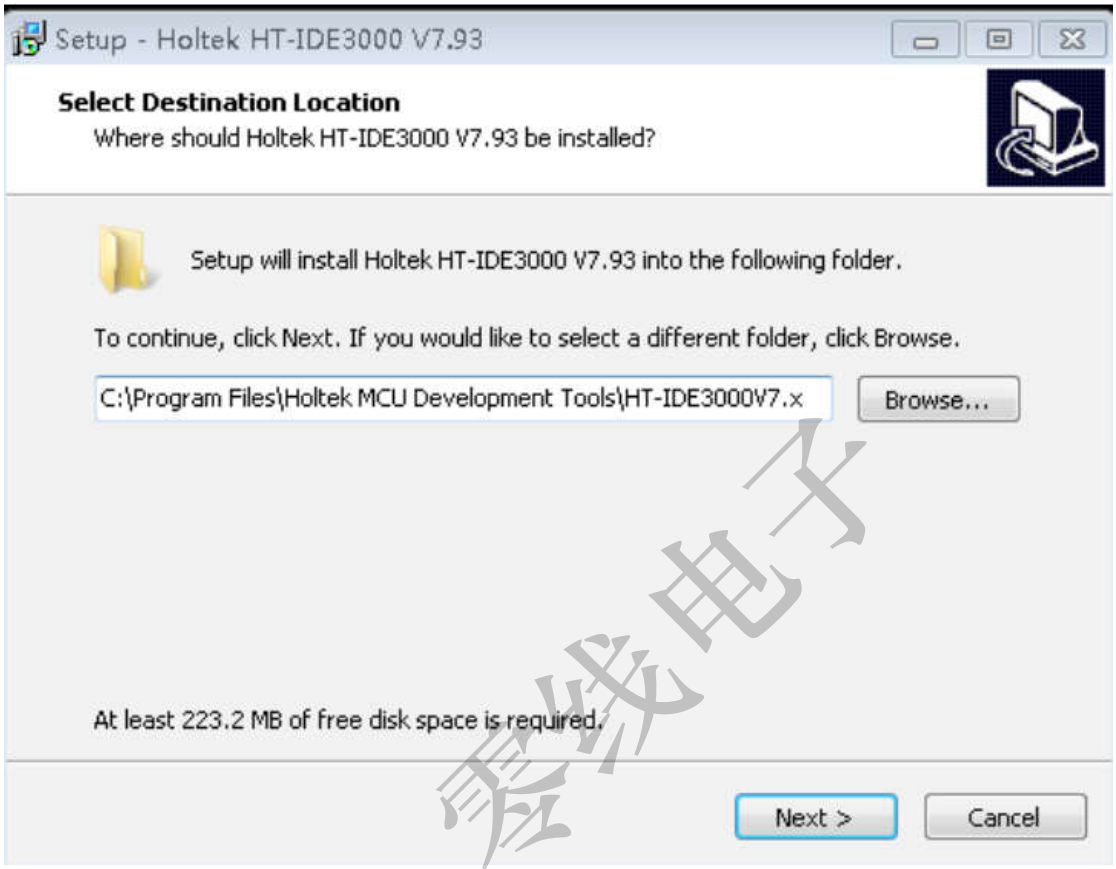
2.1 HT-IDE3000

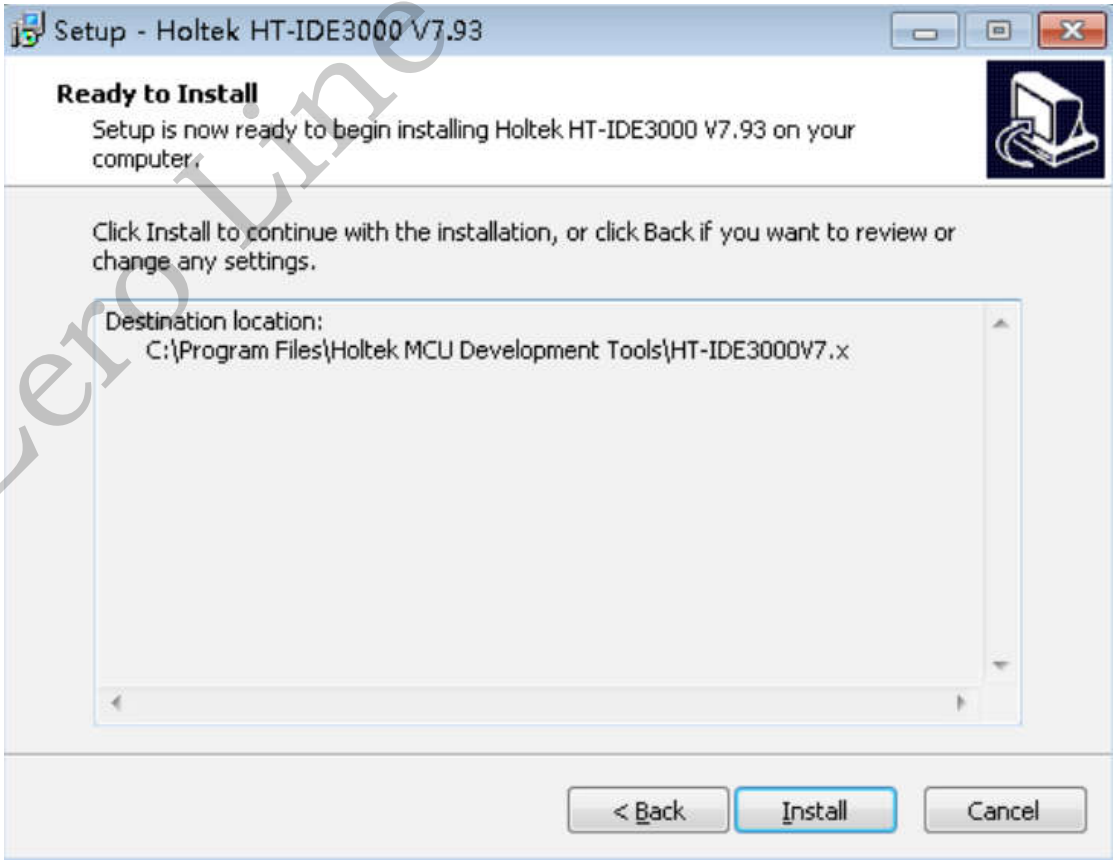
HT-IDE3000 是一款编辑，编译软件，功能同常用的 KEIL 类似。安装文件在资料包“03 开发软件”目录中。安装步骤如下。

- 步骤 1：解压 HT-IDE3KV795_Setup 安装程序并开始安装 HT-IDE3000。
- 步骤 2：按下〈Next〉按钮（继续安装）或按下〈Cancel〉按钮（中止安装）。

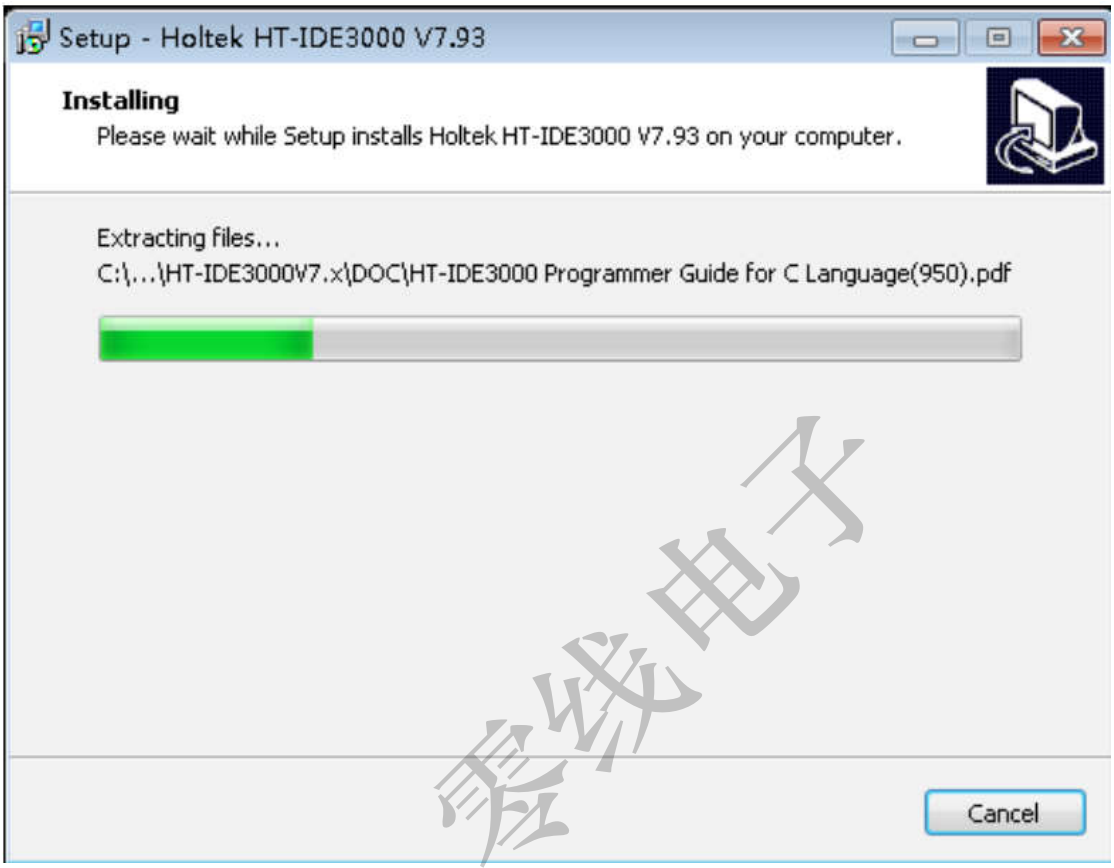


- 步骤 3：下面显示的对话框会要求使用者输入安装处的数据夹名称。

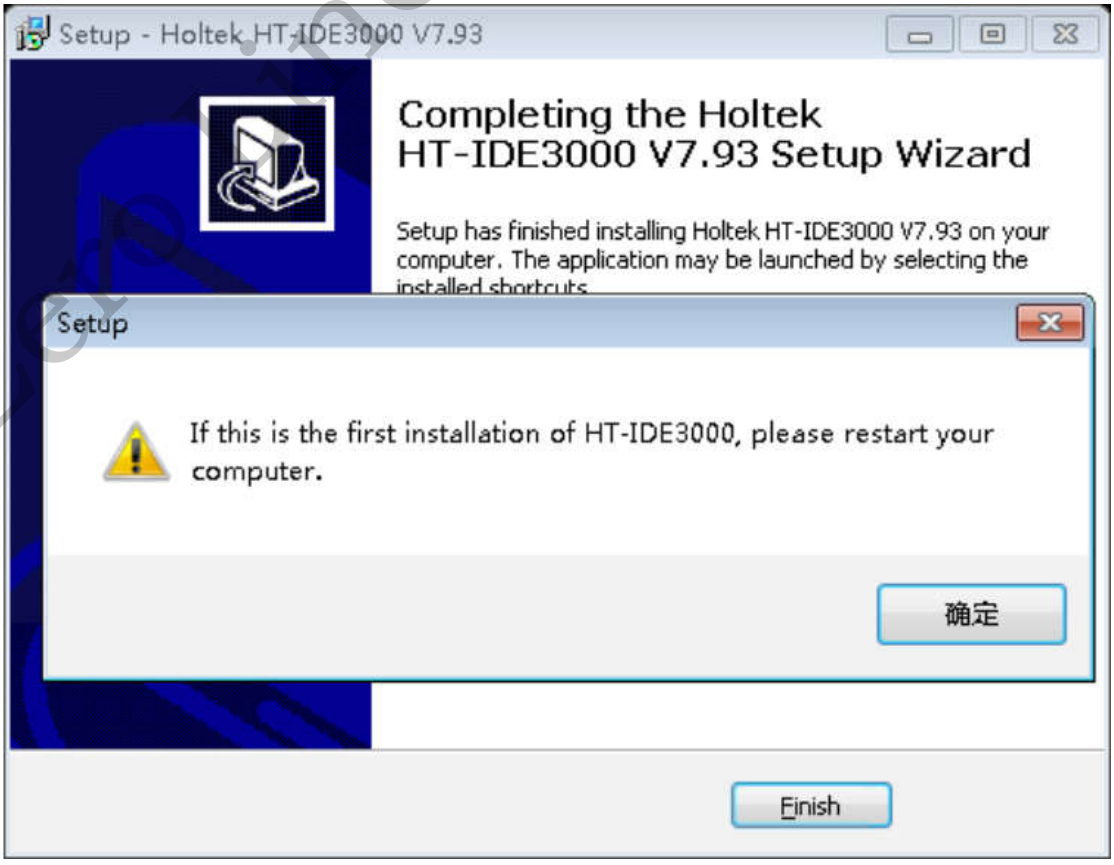




- 步骤 4：指定你希望安装 HT-IDE3000 的数据夹路径， 然后按下〈Next〉按钮。
- 步骤 5：SETUP 会将档案复制到你指定的数据夹。



- 步骤 6：安装成功的话， 则会出现下面的提示对话框。

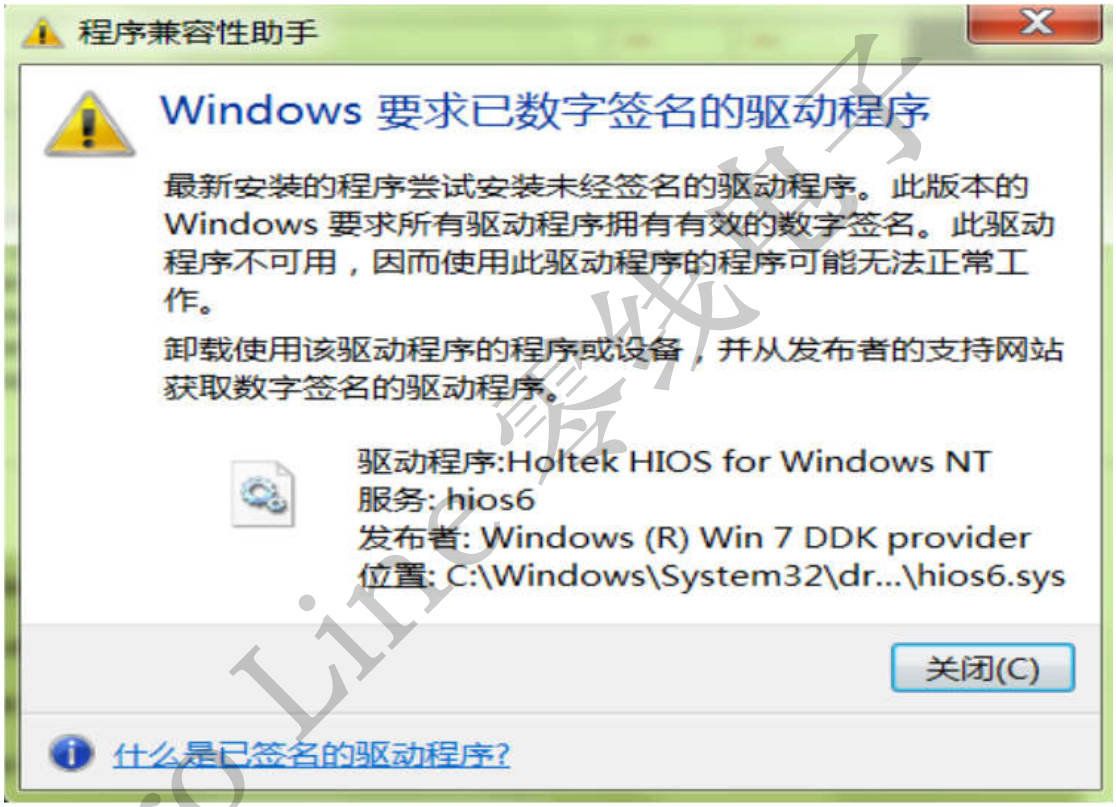


- 步骤 7：请按下确定，然后点击〈Finish〉按钮完成安装，如果是第一次安装

HTIDE3000， 需要重新启动计算机。

关于驱动程序数字签名：

如果在 Win7 安装过程出现如图 提示，出现这个是源于 Win7 驱动安装的保护机制。解决方式如下：重启电脑，然后按 F8，选择 ″ 禁用驱动程序签名 ″ 方式进入操作系统， 再安装 HT-IDE3000。另外如果遇到 ICE 无法连接， 也是需要以 ″ 禁用驱动程序签名 ″ 方式进入操作系统。



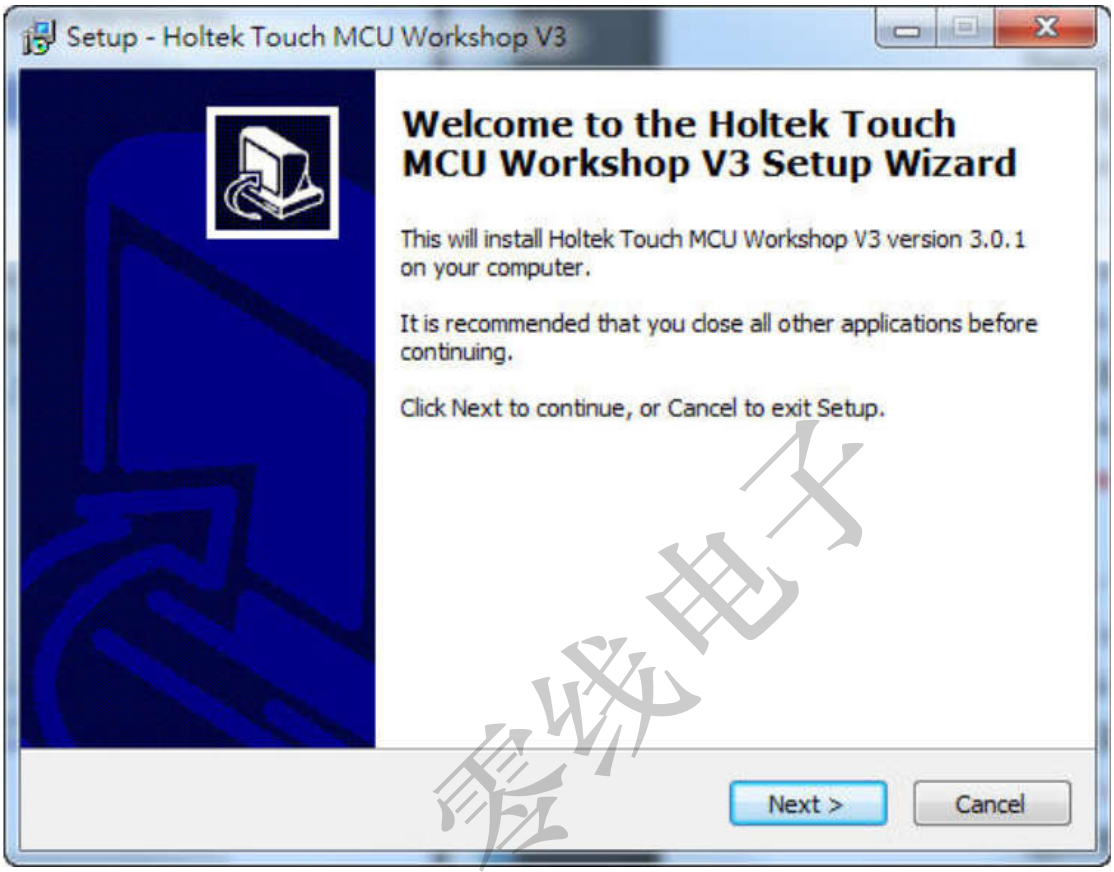
关于驱动程序安装

如果在 Win7 安装过程出现如图 1-11 的提示， 出现这个是源于 Win7 驱动安装的保护机制， 请选择始终安装此驱动程序软件。

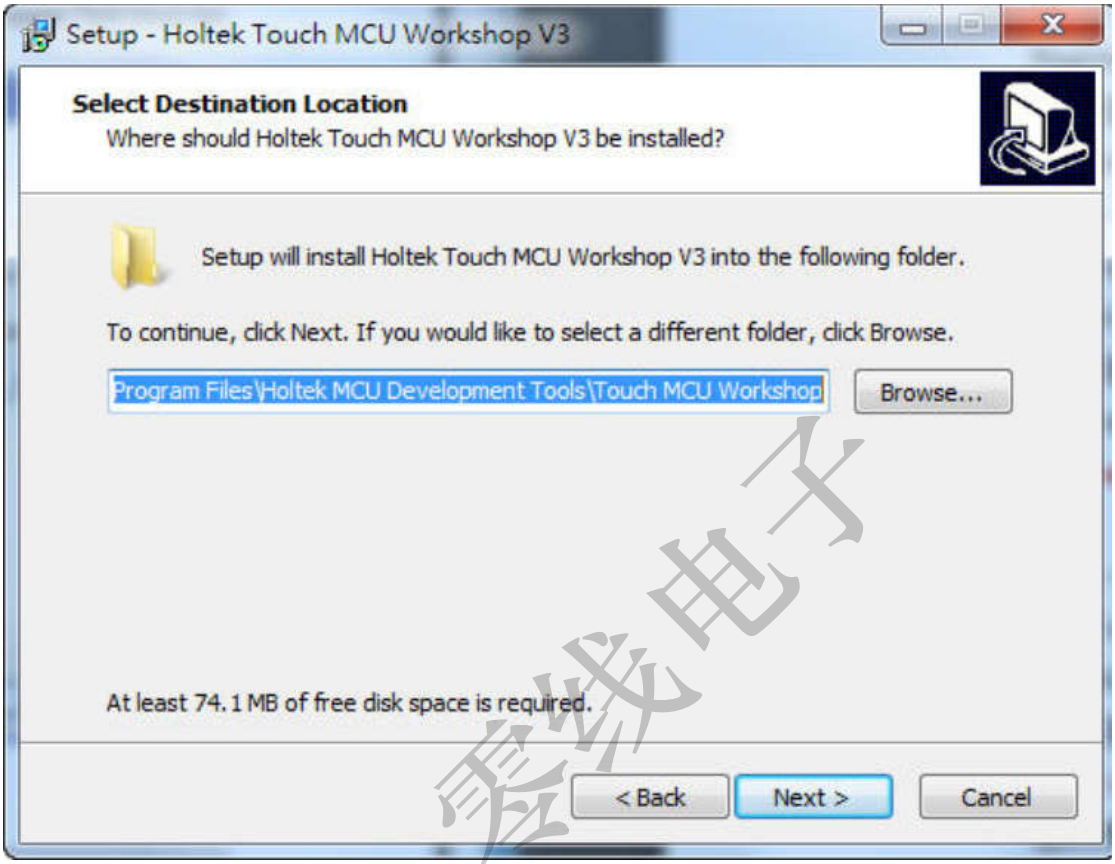
2.2 Touch MCU Workshop

Touch MCU Workshop V3 为第三代触控按键开发平台， 采用可视化的拖曳方式完成单片机 IO 功能设定。该项目也可以当做高级用户的框架 (framework) 继续开发自己的产品项目。 平台内部集成触控函数库、 编译功能与按键信号监测与按键参数调整。(使用该软件，通过图形化配置单片机，生成开发项目所用的函数库)。安装文件在资料包“03 开发软件”目录中。安装步骤如下：

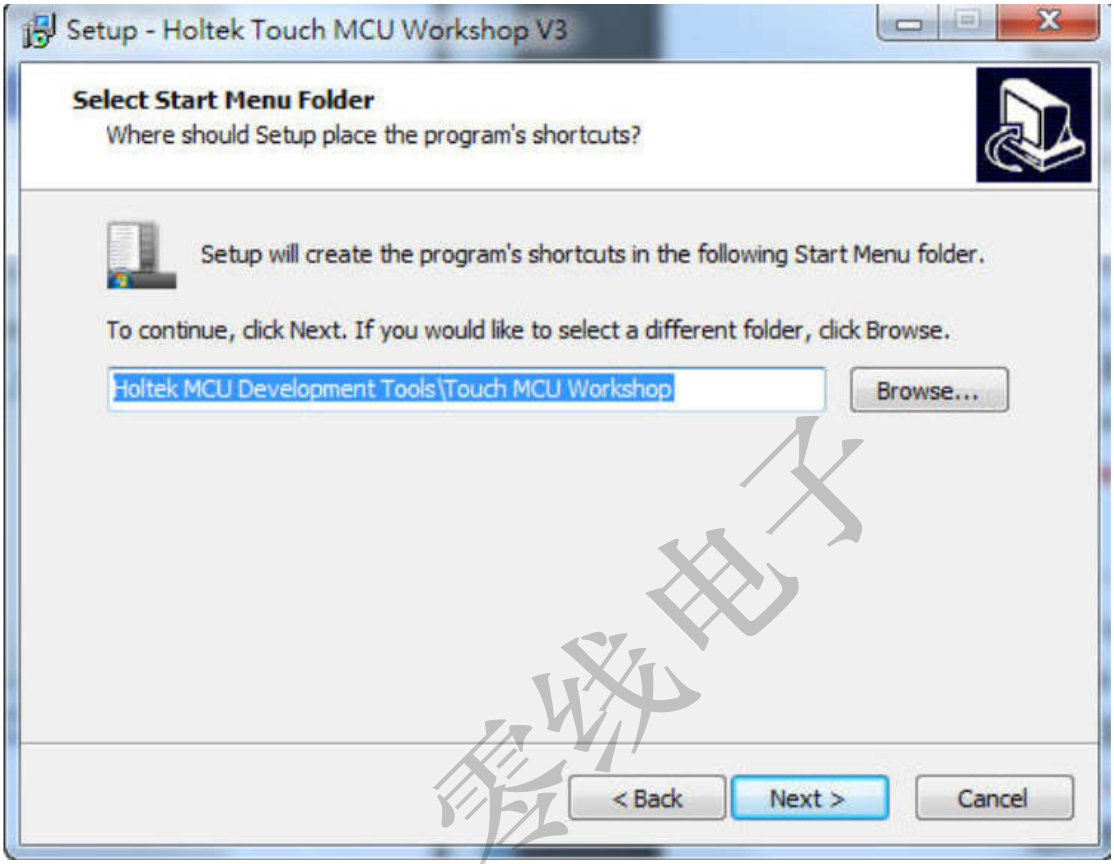
- 步骤 1：执行安装程序后欢迎的对话框将会显示如下图， 点击“Next”执行安装程序。



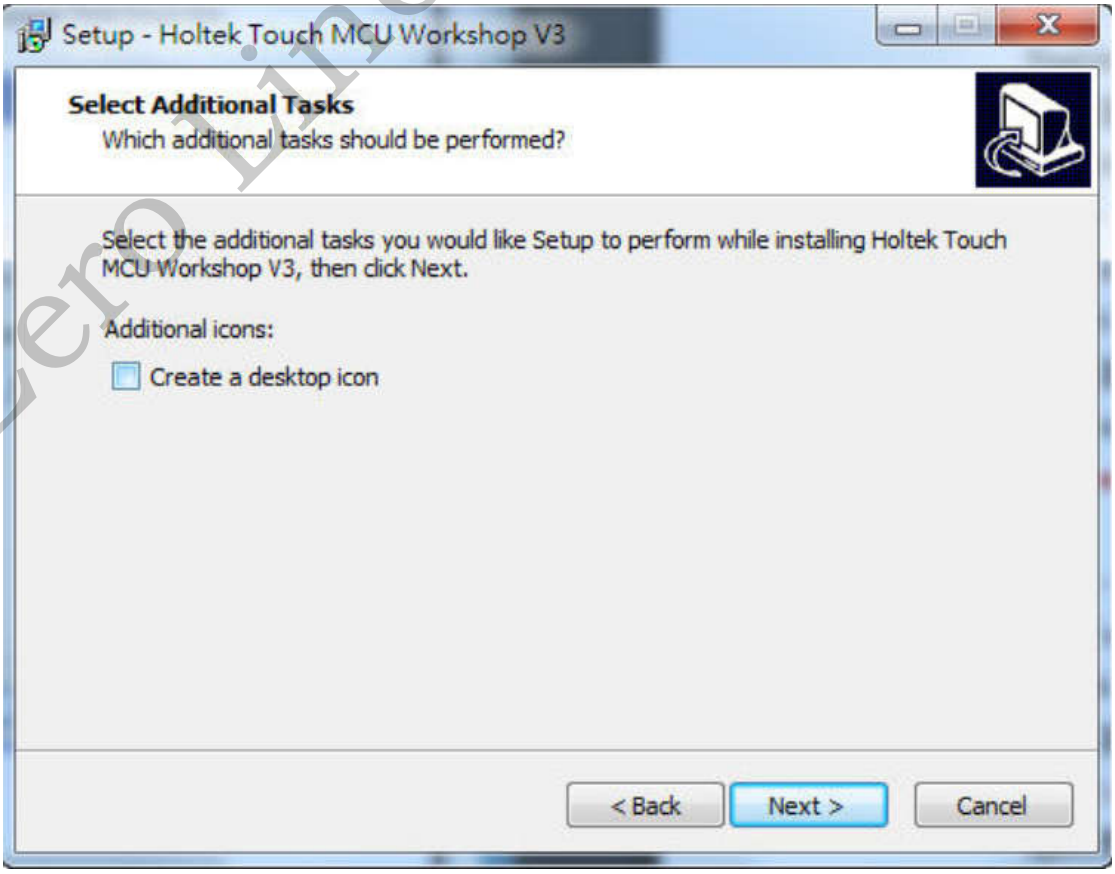
- 步骤 2：设定安装路径， 默认的路径是“C:\Program Files\Holtek MCU Development Tools\Touch MCUWorkshop”， 点击“Next”继续进行安装。



- 步骤 3：设定启动菜单程序集路径， 默认路径为“Holtek MCU Development Tools\Holtek Touch MCU Workshop”， 点击“Next”继续下一个安装步骤。

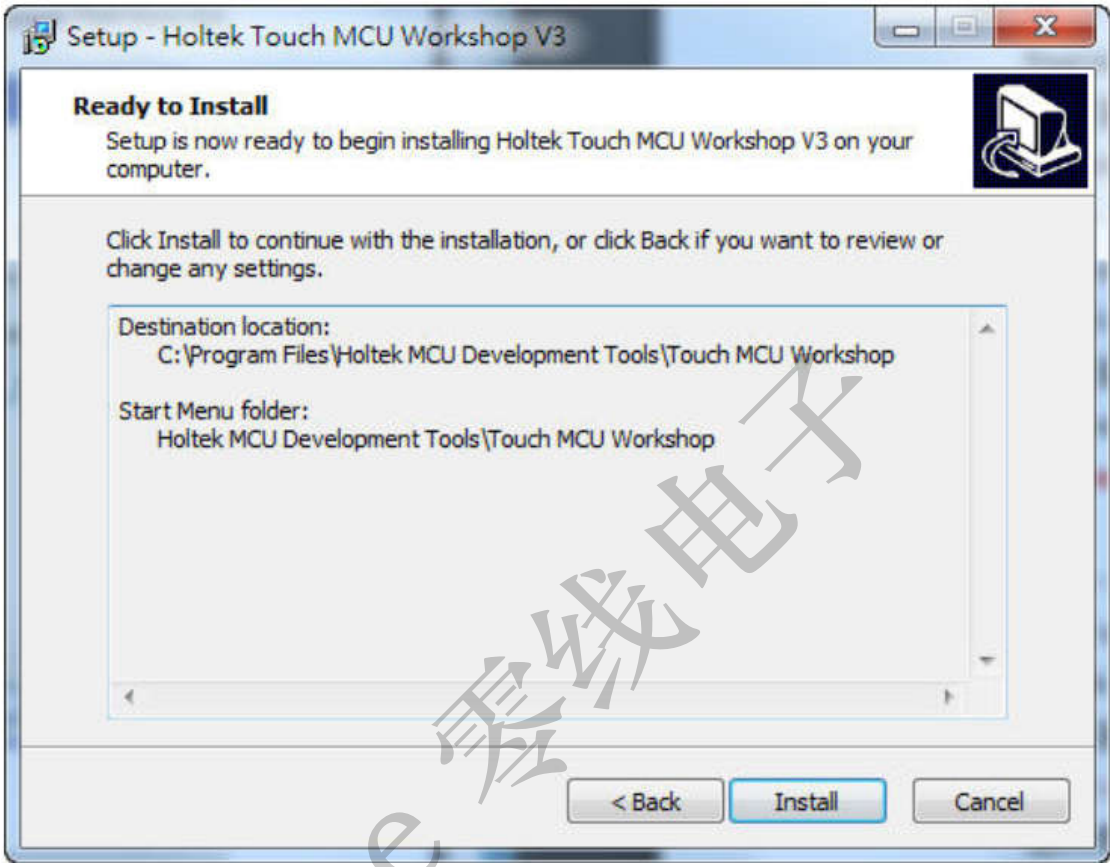


- 步骤 4：若要在桌面增加快捷方式可勾选“Create a desktop icon”。

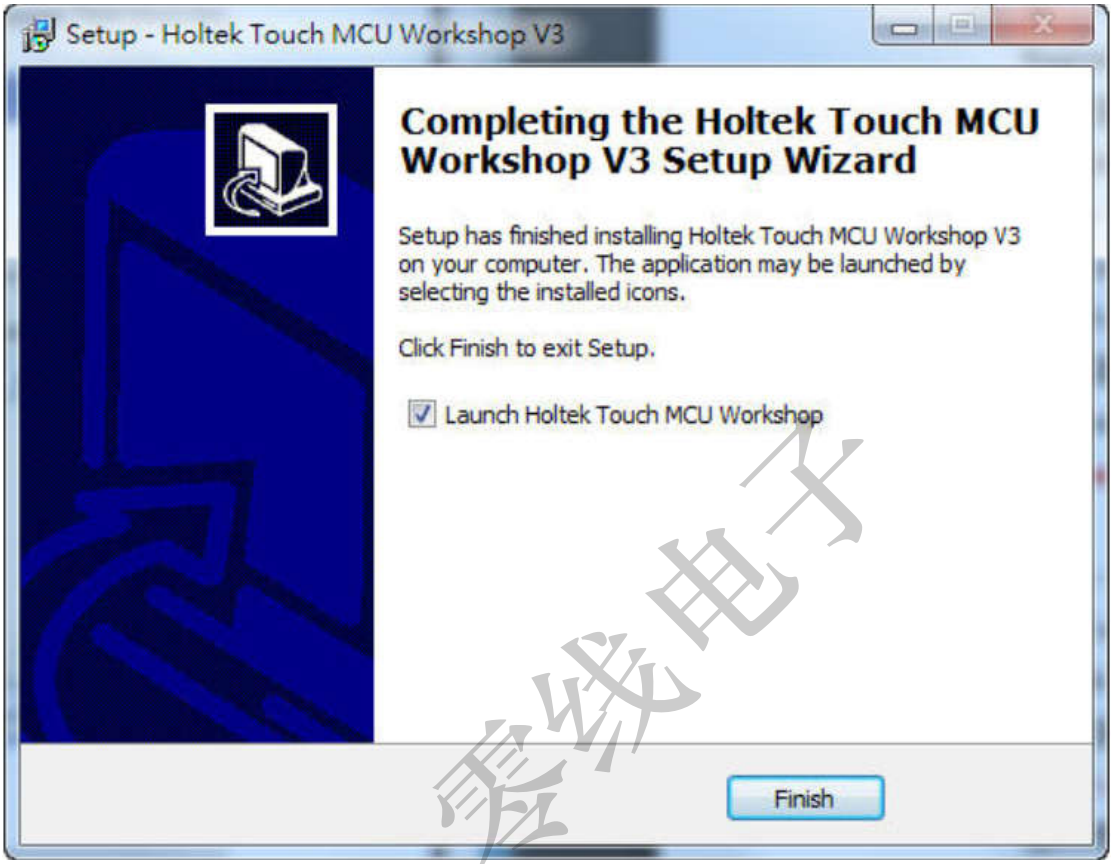


- 步骤 5：设定确认， 若要修改设定可以点击“Back”回到上一个步骤， 点击“Install”

完成安装程序。



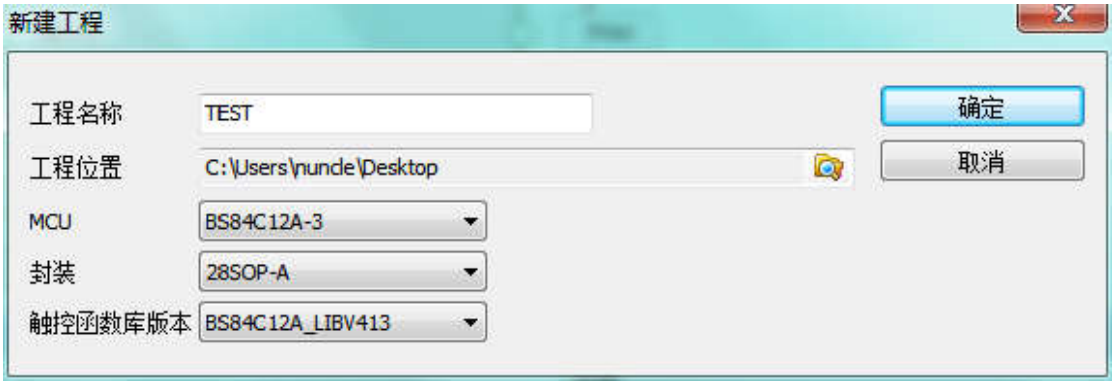
- 步骤 6：安装程序完成安装后将出现如下图的对话框，若勾选“Launch Holtek Touch MCU Workshop”， 点击“Finish”安装程序将结束并执行“Touch MCU Workshop”。



3. 新建一个工程

3.1 使用 TouchMCUWorkShop 生成项目

- 步骤 1：工具栏点击“新建”。出现新建项目对话框。



- 步骤 2：执行上图配置内容。

  工程名称输入 TEST。

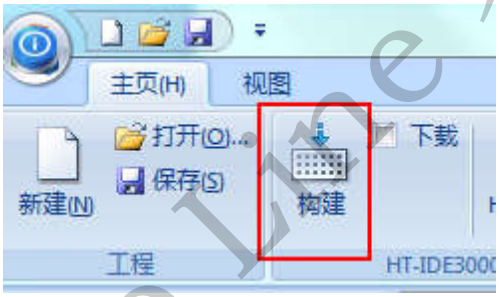
  工程位置 指定在桌面。

  MCU 选择开发板对应的 BS84C12A-3。

  选择图中对应的封装和触摸库函数版本。

  配置完成后，点击确认。

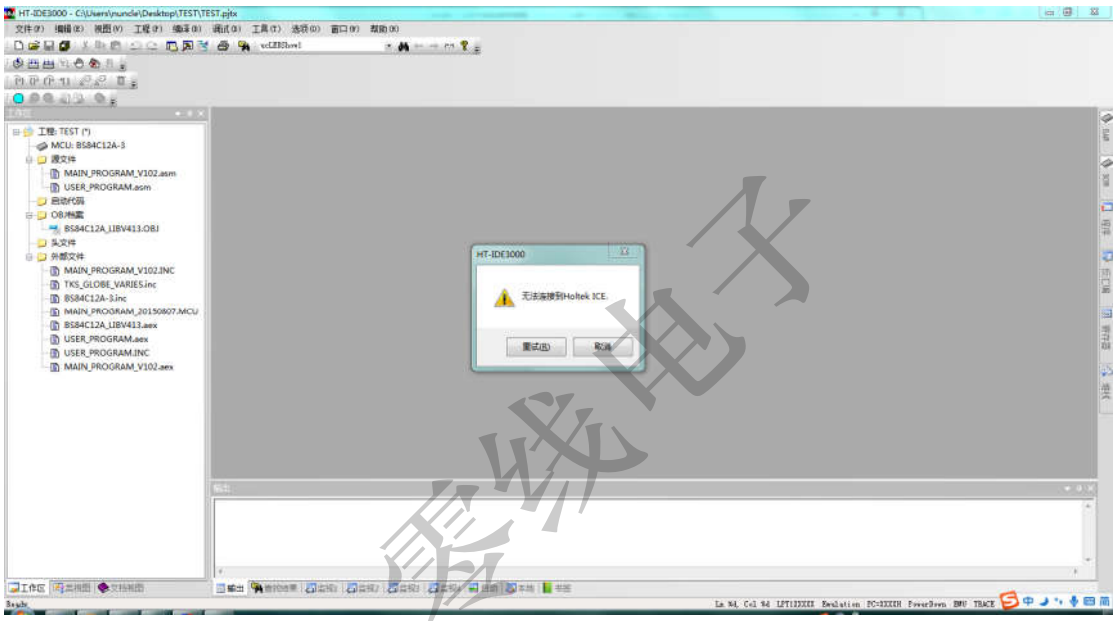
- 步骤 3：点击工具了“构建”按钮，在设定目录生成工程文件。



- 步骤 4：生成的工程目录如下

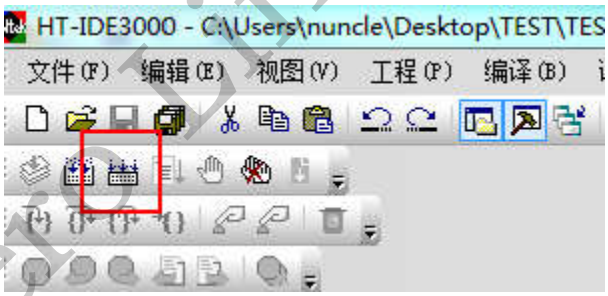
3.2 使用 HT-IDE3000 编译项目

- 步骤 1：点击上一节生成目录中的“TEST.pjtx”，将自动使用 HT-IDE3000 打开项目。



（提示宽显示没有连接到调试工具 elink ，可以点击“取消”忽略）

- 步骤 2：点击编译按钮，编译工程。

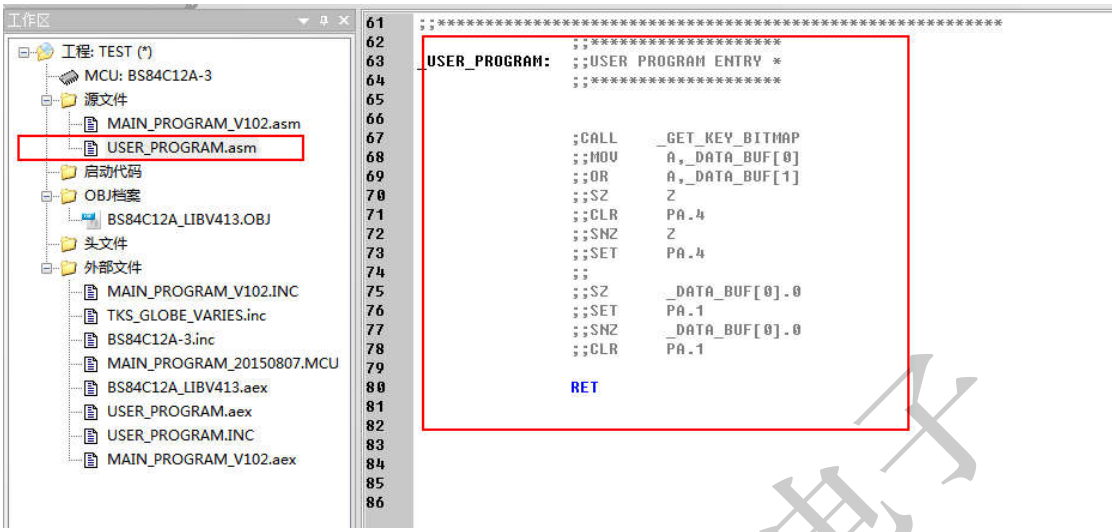


- 步骤 3：确认编译结果，编译没有出错和警告。出现一个错误是没有连接调试工具。

3.3 修改项目，使用 C 语言进行开发

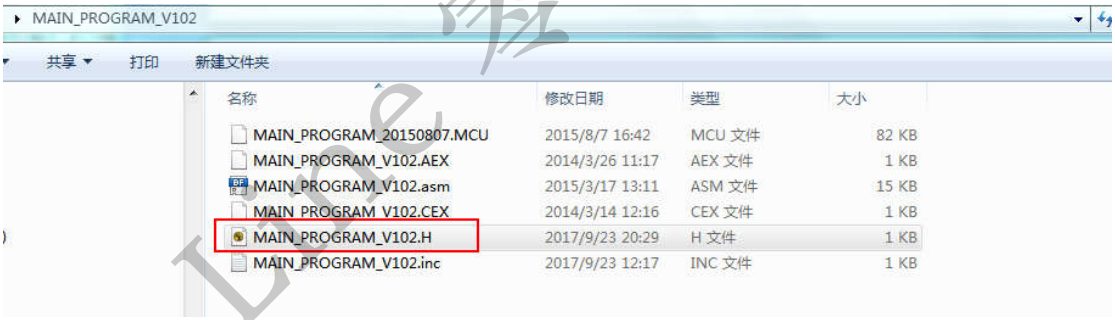
由 TouchMCUWorkShop 生成的项目，默认使用汇编进行开发，用户需要使用汇编进行

功能代码编写，如下图



为了使用 C 语言进行开发，需要进行以下步骤。

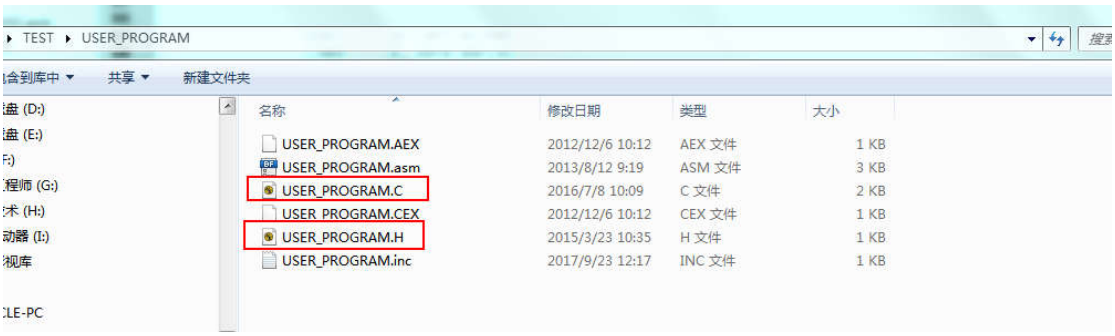
- 步骤 1：在“TEST\MAIN_PROGRAM_V102”目录，添加 MAIN_PROGRAM_V102.H 头文件。添加后如下图



文件内容为：

```
#include "..\BS84C12A_LIBV413\BS84C12A_LIBV413.CEX"
#define EXTEND_FUNCTION_1A_INITIAL      BS84C12A_LIBV413_INITIAL
#define EXTEND_FUNCTION_1A              BS84C12A_LIBV413

#include "..\USER_PROGRAM\USER_PROGRAM.CEX"
#define EXTEND_FUNCTION_1B_INITIAL      USER_PROGRAM_INITIAL
#define EXTEND_FUNCTION_1B              USER_PROGRAM
```

- 步骤 2：在“TEST\USER_PROGRAM”目录添加 USER_PROGRAM.H 和 USER_PROGRAM.C 文件。添加后如下图



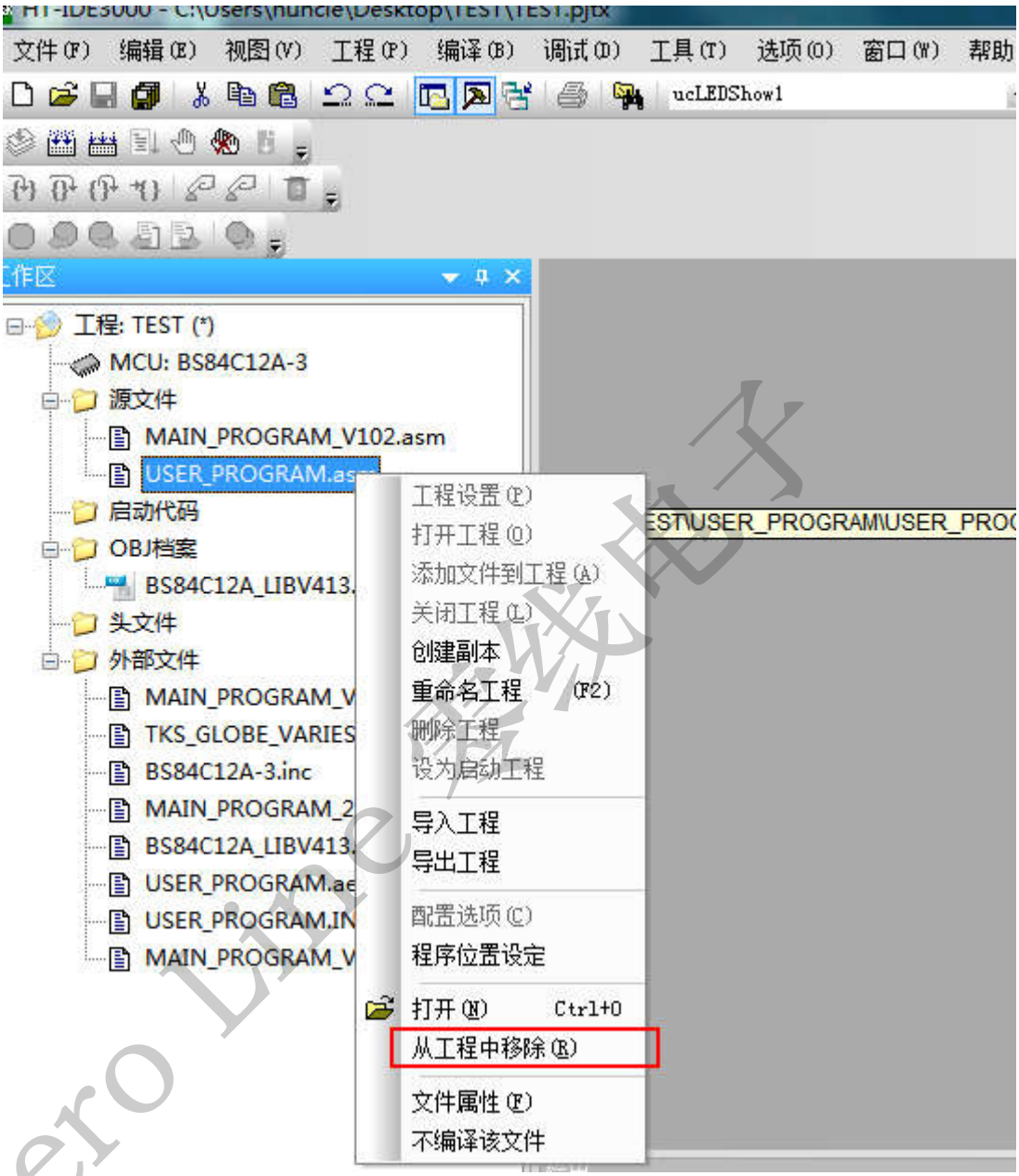
USER_PROGRAM.H 内容如下图：

```
#include    "..\TKS_GLOBE_VARIES.H"
#include    "..\MAIN_PROGRAM_V102\MAIN_PROGRAM_V102.H"
```

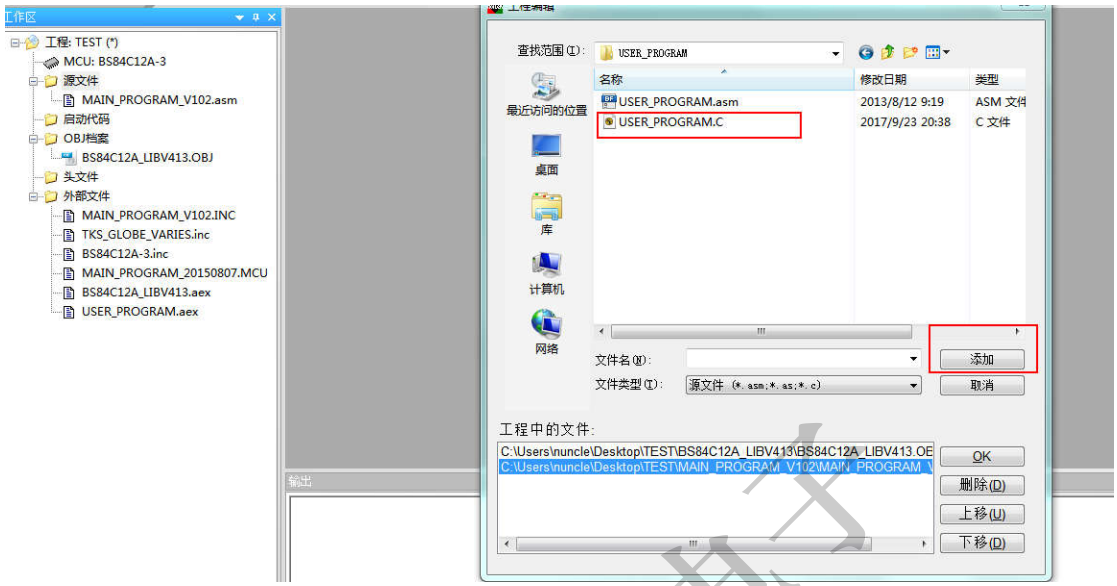
USER_PROGRAM.C 内容如下图

```
#include    "USER_PROGRAM.H"
DEFINE_ISR (Interrupt_Extemal, 0x04)

{

GCC_NOP();

}
//================================================
//************************************************
//================================================
void USER_PROGRAM_INITIAL()
{


}

//================================================
//************************************************
//================================================
void USER_PROGRAM()
{

}
```

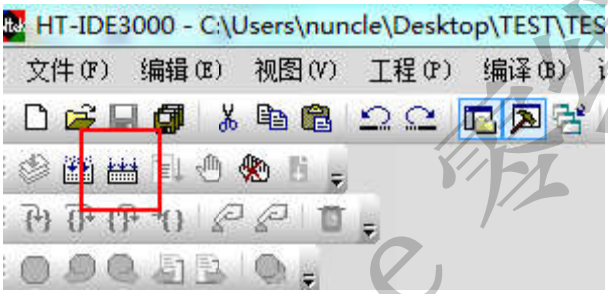
- 步骤 3：移除项目中的汇编文件 USER_PROGRAM.asm



- 步骤 4：插入 C 语言文件 USER_PROGRAM.C



- 步骤 5：重新编译工程。



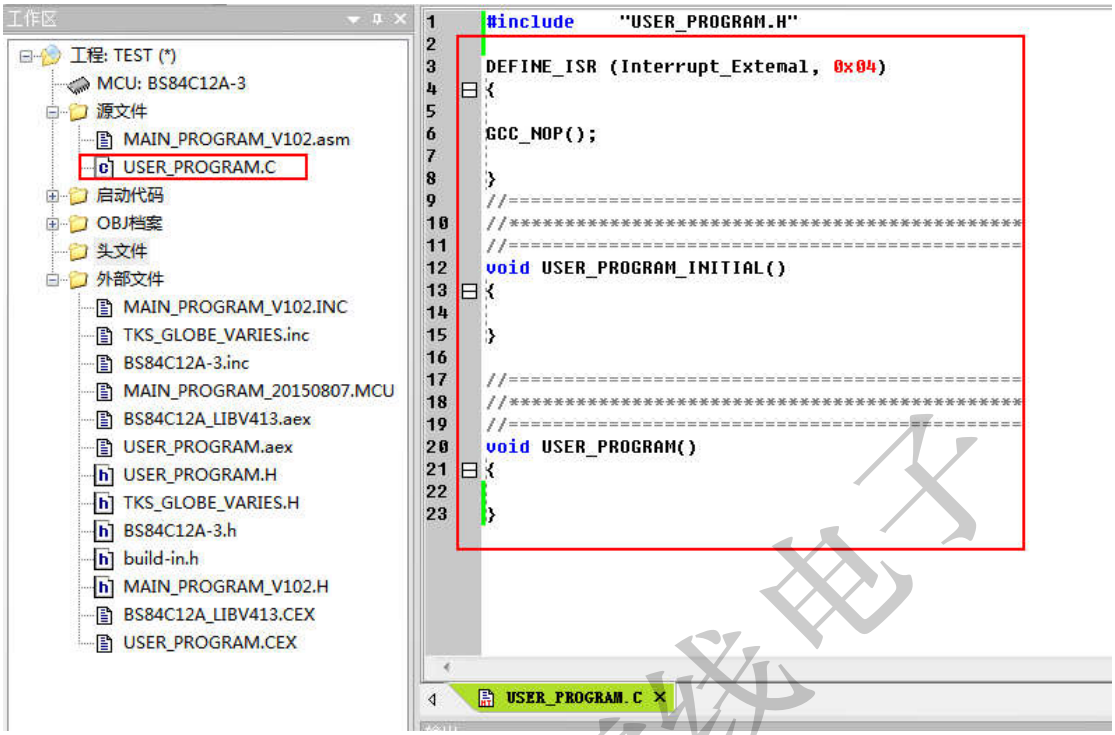
- 步骤 6：检查编译结果。程序没有错误和警告。只有提示错误没有连接 Elink。

至此，一个新的，不执行具体任务的项目工程已经建立完成。接下去介绍工程的框架，为编写用户代码做准备。

4 项目编程框架介绍

4.1 如何编程自己需要的任务

用户需要编写自己的应用程序，只需要关心一个文件 USER_PROGRAM.C。如图



初始化代码，在 USER_PROGRAM_INITIAL()函数中添加。函数中的代码只执行一次。

任务代码，在 void USER_PROGRAM()函数中添加。可以理解为该函数一直在 while（1）中循环执行。

（具体代码功能展开实例，在下一章节，点亮一个 LED 灯的工程中演示）

4.2 使用触摸库，注意事项

使用触摸库编程，将占有单片机的资源，占有情况如下：

IC	ROM	RAM	Stack	Interrupt	Other
BS83A04A	92% (total 1K)	82% (Total 96 bytes) (60H-0AEH)	3	Time Base	MP1 ; IAR1 ; MP0 ; IAR0
BS83B08A	60% (total 2K)	78% (Total 160 bytes) (60H-0DCH)	3	Time Base	MP1 ; IAR1 ; MP0 ; IAR0
BS83B12A	63% (total 2K)	61% (Total 288 bytes) Bank0 (60H-0AFH) Bank1 (80H-0DFH)	3	Time Base	MP1 ; IAR1 ; MP0 ; IAR0
BS83B16A	65% (total 2K)	77% (total 288 bytes) Bank0 (60H-0BFH) Bank1 (80H-0FFH)	3	Time Base	MP1 ; IAR1 ; MP0 ; IAR0
BS84B08A	40% (total 3K)	43% (Total 288 bytes) Bank0 (60H-084H) Bank1 (80H-0D7H)	3	Time Base	MP1 ; IAR1 ; MP0 ; IAR0
BS84C12A	31% (total 4K)	45% (Total 384 bytes) Bank0 (60H-08BH) Bank1 (80H-0A3H) Bank2 (80H-0DFH)	3	Time base	MP1 ; IAR1 ; MP0 ; IAR0
BS82C16A	32% (total 4K)	43% (Total 512 bytes) Bank0 (80H-0AFH) Bank1 (80H-0AFH) Bank2 (80H-0FFH)	3	Time base0	MP1 ; IAR1 ; MP0 ; IAR0

- 堆栈：占用了 3 级，由于 BS84C12A 是物理堆栈，只有 5 级，这就意味着，在 USER_PROGRAM()函数中，不能再进行函数嵌套。

  举例：

  ```
  void USER_PROGRAM()
  {
      fun1（）;
  }
  fun1（）; //fun1 函数中不能再调用其他函数
  ```

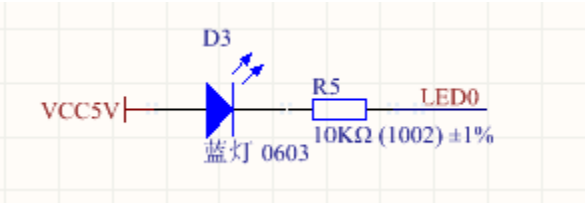
  如果 fun1（）函数再调用 fun11（）函数，程序将会跑飞。

- 由于 C Compiler 会默认把变量配置到 RAM bank0，当 bank0 满了之后，会报 RAM bank 0 overflow，对此可手动将全局变量调到其它 bank。如下图是自己实际项目中 BANK0 空间用完后，变量分配的情况。

```
//按键触发的按键编号
//0          没有触发a
//1 2 4 5    温度加减 时间加减
//7 8        菜单 功能
//3 6        切换温度  进入自检
volatile unsigned char ucKeySec=0;
static volatile unsigned int  uiKeyTimeCnt[8] __attribute__ ((at(0x1b4))); //按键去抖动延时计数器
static volatile unsigned int  uiKeyCtntyCnt[8] __attribute__ ((at(0x1c4)));  //按键连续触发的间隔延时计数器
static volatile unsigned char ucKeyLock[8] __attribute__ ((at(0x1d4))); //按键触发后自锁的变量标志
const  unsigned char uckeyBuf[8] ={0b00100000,0b00010000,0b00110000,0b00000001,
    0b00000100,0b00100001,0b00001000,0b00000010};

//默认的高低参数
```

5 第一个项目——点亮一个 LED 灯

5.1 硬件设计



其中 LED0 连接到单片机 BS84C12A 的 PA1 引脚。IO 输出低电平，LED 亮；IO 输出高电平，LED 灭。

5.2 GPIO 端口初始化

- 步骤 1：PA1 口配置成输出模式，需要配置 PAC 寄存器相应位为 0。

PAC 寄存器

Bit	7	6	5	4	3	2	1	0
Name	D7	—	—	D4	D3	D2	D1	D0
R/W	R/W	—	—	R/W	R/W	R/W	R/W	R/W
POR	1	—	—	1	1	1	1	1

Bit 7　PA 口 bit 7 输入 / 输出控制
　　0: 输出
　　1: 输入
Bit 6 ~ 5　未使用，读为“0”
Bit 4 ~ 0　PA 口 bit 4 ~ bit 0 输入 / 输出控制
　　0: 输出
　　1: 输入

- 步骤 2：PA1 口配置成上拉模式，需要配置 PAPU 寄存器相应位为 1。

PAPU 寄存器

Bit	7	6	5	4	3	2	1	0
Name	D7	—	—	D4	D3	D2	D1	D0
R/W	R/W	—	—	R/W	R/W	R/W	R/W	R/W
POR	0	—	—	0	0	0	0	0

Bit 7　PA 口 bit 7 上拉电阻控制
0: 除能
1: 使能

Bit 6 ~ 5　未使用，读为“0”

Bit 4 ~ 0　PA 口 bit 4 ~ bit 0 上拉电阻控制
0: 除能
1: 使能

- 步骤 3：控制 IO 口输出高低电平，需要控制寄存器 PA。

5.3 软件设计

设计程序，让开发板上的 LED0 灯闪烁。

具体代码见资料包中，程序目录。

```
//==================================
//主函数
//==================================
void USER_PROGRAM()
{
    _pa1 = 1;    // LED灭
    GCC_DELAY(100000);
    _clrwdt();
    GCC_DELAY(100000);
    _clrwdt();
    GCC_DELAY(100000);
    _clrwdt();
    _pa1 = 0;    // LED亮
    GCC_DELAY(100000);
    _clrwdt();
    GCC_DELAY(100000);
    _clrwdt();
    GCC_DELAY(100000);
    _clrwdt();
}
```

点击工具栏“全部重建”编译代码。

5.4 程序烧写与运行

连接 e-link 后，ICP 烧入按钮变成蓝色，点击进行程序烧写，重新上电后执行程序。

5. IO 口模拟串口

5.1 串口通信原理

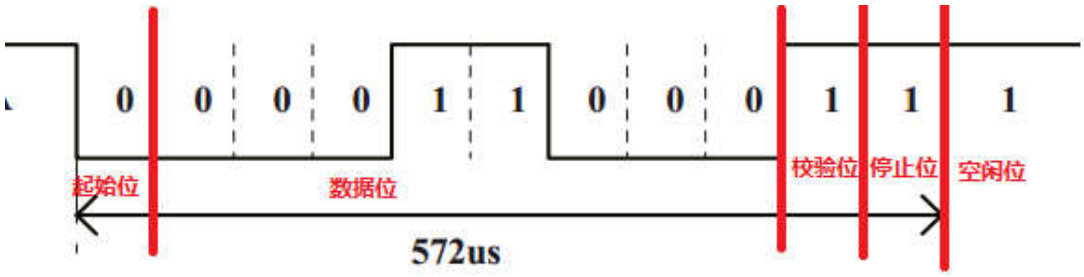
工作原理是将传输数据的每个字符以串行方式一位接一位的传输。下图给出了其工作模式：

标准串口数据格式为：**起始位（1bit）+数据位（8bit）+校验位(1bit)+停止位(1bit)**。其中起始位为低电平，停止位为高电平。

- 起始位：为由高变低，低电平时间为一位的时间，表示传输字符的开始。
- 数据位：紧跟起始位之后，D0-D7，低位先发。
- 校验位(奇校验)：为一位的时间，如果 8 位数据位中 1 的个数为奇数，该位设为 0（置低电平），否则为 1（置高电平）。
- 停止位：置高。时间为一位的时间，它是发送完一个字符数据的结束标志。
- 空闲位：置高。

波特率：传输 1 位所需要的时间。接下去以 19200bps 为例，则传输 1 位需要的时间是 1/19200≈52us（微秒）。

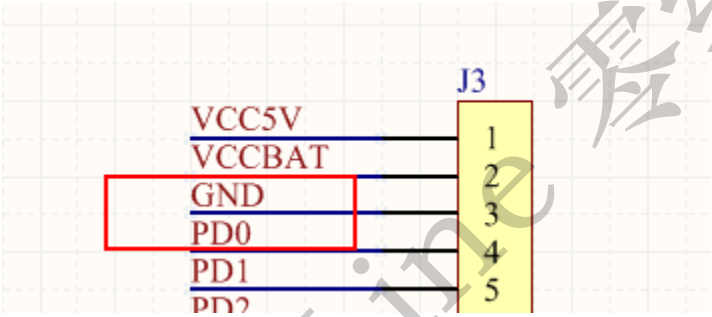
下面我们以波特率为 19200bps 为例，给 TX 脚发送显示控制命令“0X18”，时序波形图如下：



如上图，发送“0X18”总共 11 位，从左至右依次为 1 位起始位、8 位数据位、1 位校验位、1 位停止位，每位的时间约为 52us ，11 位总时间约为 572us。由上图读出二进制为“00011000”即转换为十六进制为“0X18”。

5.2 硬件设计

本实验使用 IO 口 PD0 模拟串口的 TX 发送引脚。



5.3 软件设计

IO 口模拟串口，使用 19200 波特率发送数据，PC 端使用串口调试工具接收开发板发送的数据。详细代码参考程序。这里对主要的函数进行说明。

```
//============================================

void SimUART_TxByte( unsigned char SendData )
{
    unsigned char  BitNum   = 0;          //位计数
    bit xor = 1;                          //校验位

    //STEP1 发送起始位
    UART_TX = 0;                          //由高电平变成低电平，表示传输字符开始
    GCC_DELAY(116);                       //示波器测量延时52us 19200bps传输1bit需要的时间
    //STEP2 开始发送数据字节8位，低位在前
    for(BitNum = 0; BitNum < 8; BitNum++)
    {
        if(SendData & 0x01)
        {
            UART_TX = 1;
        }
        else
        {
            UART_TX = 0;
        }
        xor = xor ^ (SendData&0x01);      //如果 8 位数据位中 1 的个数为奇数，该位设为 0（置低电平）
        SendData >>= 1;                   //右移一位
        GCC_DELAY(100);
    }
    //STEP3 发送校验位
    if(xor)
    {
        UART_TX = 1;
    }
    else
    {
        UART_TX = 0;
    }
    GCC_DELAY(125);
    //STEP4 发送停止位
    UART_TX = 1;
    GCC_DELAY(125);
    GCC_DELAY(10000);
}
```

上面是软件模拟串口发送一个字节的函数。每 1 位的时间通过示波器观察得到。这里使用的是奇校验的方式。

上图是程序实际执行效果，串口调试软件的参数请按照图中设置。

5.5 其他

如果使用过程中改成其他串口参数有疑问， 比如波特率和校验模式，欢迎找本店的技术支持。

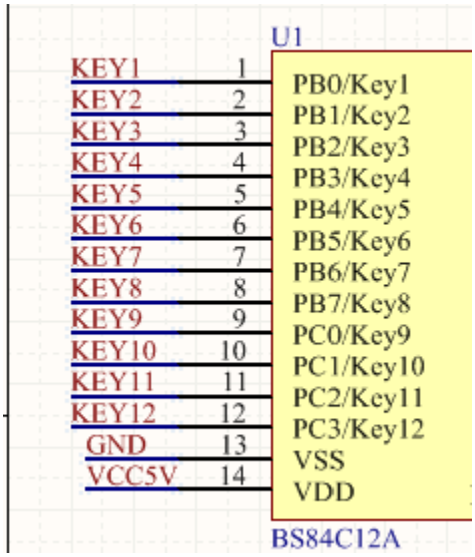
6 .触摸按键

6.1 触摸按键实现

正如第 3 章，使用 TouchMCUWorkShop 新建工程说描述，使用该软件，生成的项目只

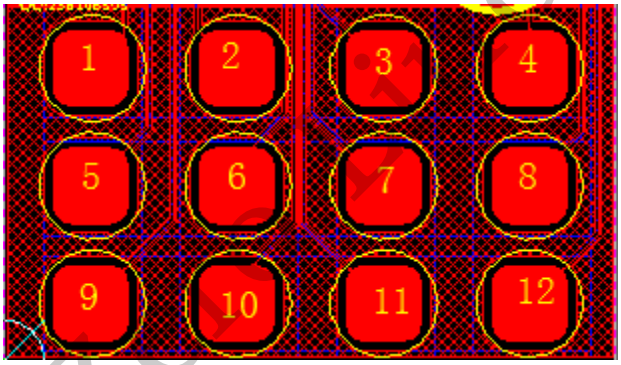
需要简单的修改配置文件就能实现触摸按键。

6.2 硬件设计



KEY1~KEY12 总共 12 个触摸按键。

PCB 按键分布



6.3 软件设计

在上一章模拟串口的基础上，增加触摸按键功能，通过串口打印按键值。配置触摸按键相关步骤如下：

- 步骤 1：打开 TKS_GLOBE_VARIES.inc 文件。
- 步骤 2：修改#define IO_TOUCH_ATTR 00000000000000000000111111111111b

  比如开发板用到 KEY1~KEY12，所以宏定义最低 12 为设定为 1，表示用来做按键。
- 步骤 3：修改 Key1Threshold、Key2Threshold……等的值，范围 8~255，建议值 16~40，

数值越小，灵敏度越高。

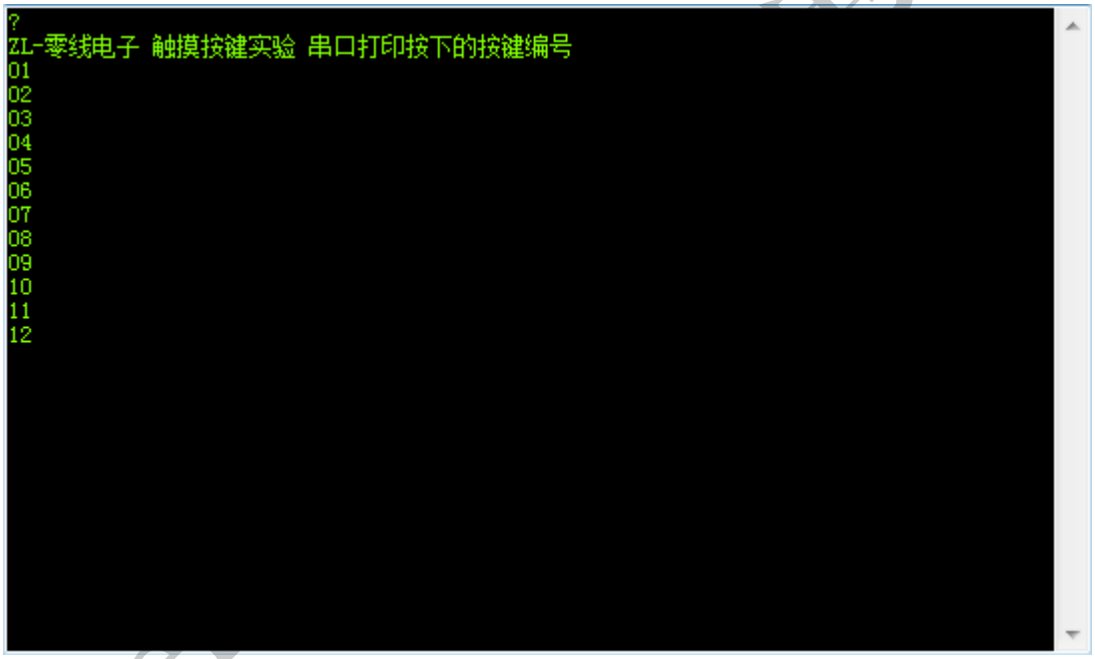
- 步骤 4：在主函数中调用 GET_KEY_BITMAP();获取按键值。
- 步骤 5：DATA_BUF[0] DATA_BUF[1]存储的就是按键状态，其中

  DATA_BUF[0] = KEY 8（MSB） ~ KEY1 （LSB）。

  DATA_BUF[1] = KEY16（MSB） ~ KEY9 （LSB）。

6.4 运行结果

上电后，顺序按下 12 个按键的打印信息。可以将开发板连接其他硬件，作为触摸按键模块使用。



7. 数码管显示

7.1 数码管驱动 TM1652

TM1652 是一款 LED（发光二极管、数码管、点阵屏）驱动控制专用芯片，内部集成了数字通讯电路、解码电路、数据锁存器、震荡器、LED 驱动电路。通讯方式采用异步串口通信（UART）协议，因芯片只接收单片机发来的数据，仅需要单片机的一个 TX 端口发送数据给芯片即可，实现单线通讯；在显示驱动方面，芯片采用动态扫描方式，两种显示模式可选，

8 级段驱动电流可调，16 级位占空比可调；TM1652 内置消隐处理优化电路。

在模拟串口实验中，我们已经介绍了 19200 模拟串口的使用，因此可以直接用来驱动 TM1652。

实验只使用 TM1652 的地址自动加 1 模式，进行控制。

3.1、地址自动加1模式

使用地址自动加1模式，设置地址实际上是设置传送的数据流存放的起始地址。起始地址命令字发送完毕，紧跟着发送数据，最多6BYTE，数据发送完毕后置高数据线。

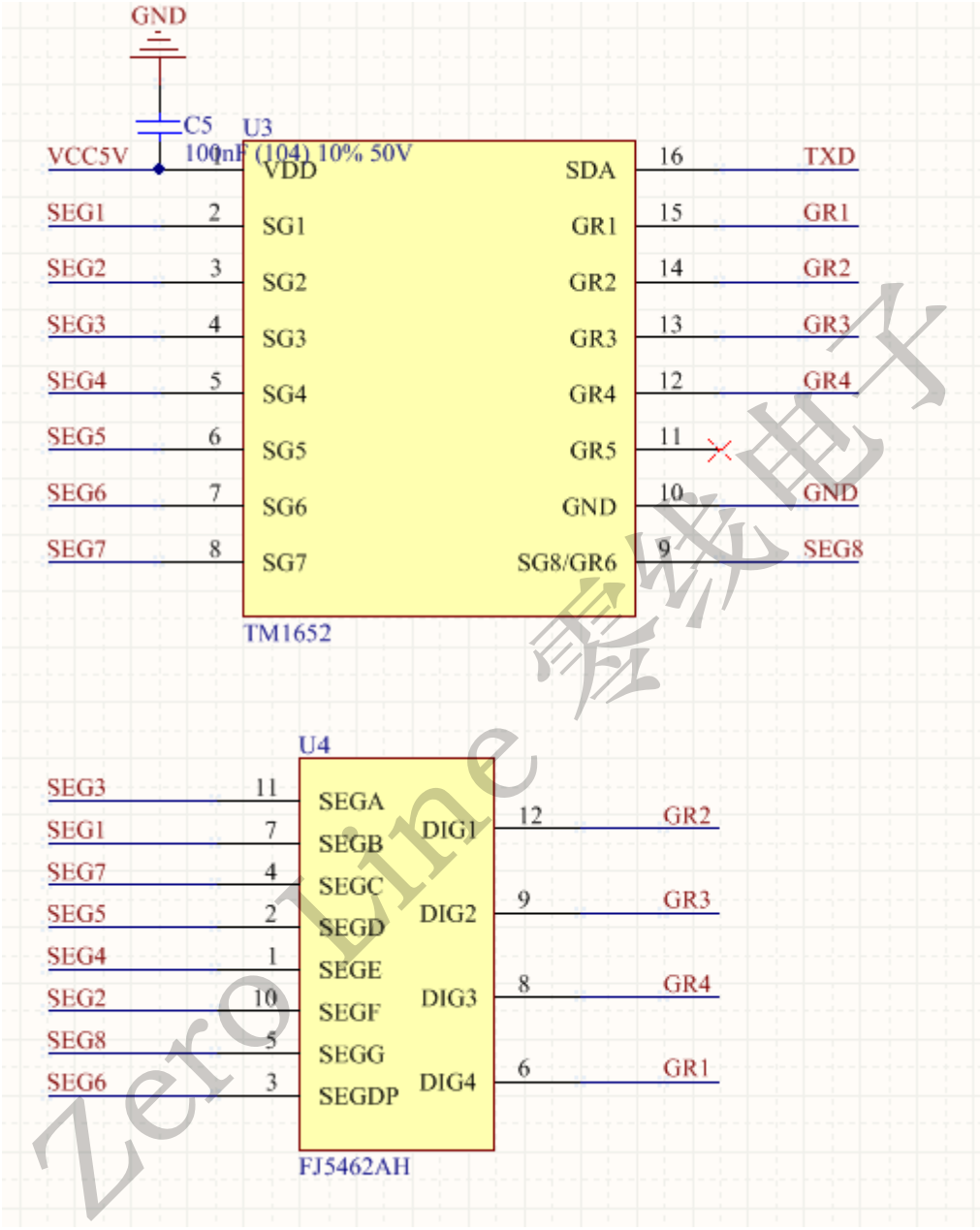
图6

Command1:选择显示地址命令（0x08）
Data1~Data n:发送显示数据（最多6bytes）
Time：数据线置高时间（最小时间为3ms）
CommandX：选择显示控制命令（0x18）
CommandY:发送显示控制调节命令（包括位占空比、段驱动电流以及显示模式设置）

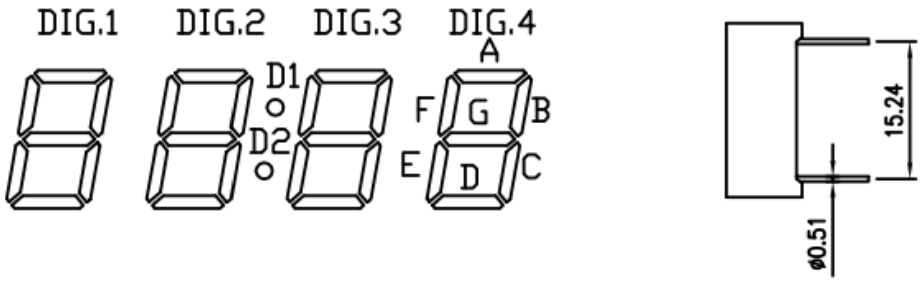
对应的显示代码如图，就是对所有的段进行点亮。具体调节亮度可以在数据手册中查找。

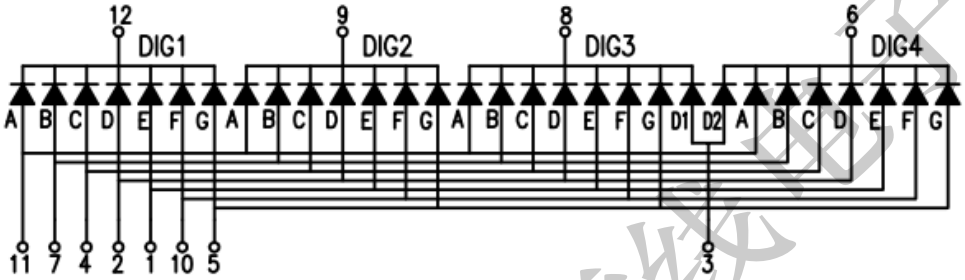
```
SimUART_TxByte(0x08); //选择显示地址命令

SimUART_TxByte(0xff);   //GRID 1 连接到第4位数码管
SimUART_TxByte(0xff);   //GRID 2 连接到第1位数码管
SimUART_TxByte(0xff);   //GRID 3 连接到第2位数码管
SimUART_TxByte(0xff);   //GRID 4 连接到第3位数码管
SimUART_TxByte(0xff);   //GRID 5  没有连接

GCC_DELAY(10000);GCC_CLRWDT();  //延时3ms
GCC_DELAY(10000);GCC_CLRWDT();

SimUART_TxByte(0x18);//选择显示控制命令
SimUART_TxByte(0x1C);//使芯片工作在位占空比8/16、段驱动电流4/8，显示模式为8段5位
```

7.2 硬件设计



数码管如图





- TXD 连接到 PD7。代码中修改模拟串口发送脚的宏定义即可。
- TM1652 的 GR1 接数码管 DIG4，既 4 位数码管的第 4 位。
- TM1652 的 GR2 接数码管 DIG1，既 4 位数码管的第 1 位。
- TM1652 的 GR3 接数码管 DIG2，既 4 位数码管的第 2 位。
- TM1652 的 GR4 接数码管 DIG3，既 4 位数码管的第 3 位。
- 点亮数码管的一个段。

  假设需要点亮 SEGA，即 TM1652 的 SEG3 需要输出有效电平，对应的数值 0x04

  所有段对应 TM1652 输出的数值如下图

```
/**********************************
//共阴七段数码管编码
#define _seg_a   0x04
#define _seg_b   0x01
#define _seg_c   0x40
#define _seg_d   0x10
#define _seg_e   0x08
#define _seg_f   0x02
#define _seg_g   0x80
#define _seg_dp  0x20
```

7.3 软件设计

程序任务为：使用数码管显示按键值。

```
if(usTimerCnt > 200)    //刷新显示
{
    usTimerCnt = 0;

    _emi = 0;       //关总中断

    SimUART_TxByte(0x08); //选择显示地址命令

    SimUART_TxByte(segCode[data1%10]);  //GRID 1 连接到第4位数码管
    SimUART_TxByte(segCode[0]); //GRID 2 连接到第1位数码管
    SimUART_TxByte(segCode[0]); //GRID 3 连接到第2位数码管
    SimUART_TxByte(segCode[data1/10]);   //GRID 4 连接到第3位数码管
    SimUART_TxByte(00);   //GRID 5  没有连接

    GCC_DELAY(10000);GCC_CLRWDT();  //延时3ms
    GCC_DELAY(10000);GCC_CLRWDT();

    SimUART_TxByte(0x18);//选择显示控制命令
    SimUART_TxByte(0x1C);//使芯片工作在位占空比8/16、段驱动电流4/8，显示模式为8段5位

    _emi = 1;       //开总中断
}
```

在刷新显示的时候，需要关掉中断，避免发送数据被打断，显示乱码。

8. 实时时钟

8.1 实时时钟芯片 BL5372

低功耗实时时钟芯片(RTC)BL5372

1. 概述

BL5372是一款低功耗实时时钟电路，通过I²C两线接口电路可以与CPU实时通信，主要用于一切需要提供时基的系统中。该芯片能够产生多种周期性中断脉冲（最长周期可长达1个月），还具有两套报时系统。BL5372内部集成一低功耗的稳压电源，故能够使恶劣的环境条件下仍能保持振荡器正常在很低的功耗工作（典型值：400nA@3.6V）。BL5372具有晶振停振检测锁存的功能，通过检测该位可以检测内部时钟数据的有效性。BL5372内置数字时间调整电路，可以保证时钟走时的高精度，并且有32KHz和32.768KHz两种晶振选择模式。该产品与理光RS5C372A完全兼容。

2. 主要特点

- 超低功耗（典型值400nA@3.6V）
- 实时时钟（12时制或者24时制两种计时方式）
- 自动识别闰年、平年（2000~2099）
- BCD码表示的时钟计数（包括时、分、秒）和万年历（包括闰年、平年、月、日、周）
- 30秒数字校时功能
- 可控的32.768KHz（或者32KHz）输出
- 两个可编程闹钟输出
- 两路可编程方波输出，为CPU提供多种中断（一个月至一秒的周期性中断）
- 通过I²C两线接口与CPU相连（最大数据时钟频率为100KHz）
- 晶振停振检测锁存功能保证了时钟数据有效性
- 32KHz和32.768KHz晶振选择
- 高精度的时间调整电路，保证了时钟走时的精确
- 超低电压工作（计时电压最低可至1.8V，通讯电压最低可至1.8V）
- SOP8或TSSOP8封装

8.2 IIC 协议

对于嵌入式开发的朋友来说，I2C 协议实在是再熟悉不过了，有太多的器件，采用的都是通过 I2C 来进行相应的设置。今天，我们就随便聊聊这个 I2C 协议。

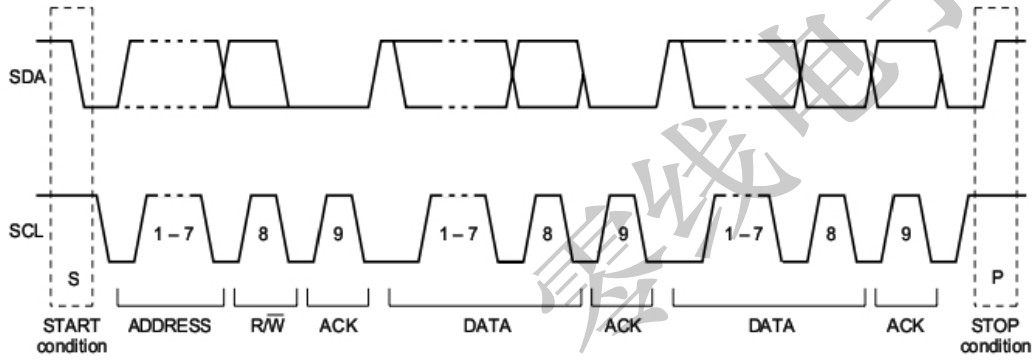
I2C 协议中最重要的一点是 I2C 地址。这个地址有 7 位和 10 位两种形式。7 位能够表示 127 个地址，而在实际使用中基本上不会挂载如此多的设置，所以很多设备的地址都采用 7 位，所以本文接下来的说明都是基于此。

I2C 还有一个很重要的概念，就是“主—从”。对于从设备来说，它是啥都不干的，更

不会自动发送数据；而主设备，则是起到控制作用，一切都是从它开始。

除了 GND 以外，I2C 有两根线，分别是 SDA 和 SCL，所有的设备都是接到这两根线上。那么，这些设备如何知道数据是发送给它们呢？这就得依靠前面所说到的地址了。设备 I2C 的地址是固定的，比如 0x50，0x60 等等。因为只能有 127 个地址，地址冲突是很常见的，所以一般设备都会有一个地址选择 PIN，比如拉高时候为 0x50，接地为 0x60。如果无论拉高还是接地，都和别的芯片有冲突，那该怎么办呢？答案是：凉拌，没办法。遇到这种情况，只能换芯片了。

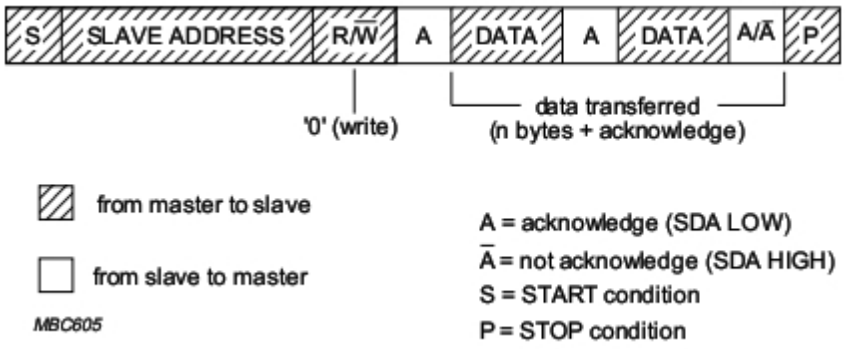
我们来看 I2C 协议中的数据传输时序图：



SCL 是时钟，SDA 承载的是数据。当 SDA 从 1 变动到 0，而 SCL 还是 1 时，表示开始数据传输。接下来的 7 位，就是设备的地址。紧接着的是读写标志，其为 1 时是读取，为 0 则是写。如果 I2C 总线上存在着和请求的地址相对应的设备，则从设备会发送一个 ACK 信号通知主设备，可以发送数据了。接到 ACK 信号后，主设备则发送一个 8 位的数据。当传输完毕之后，SCL 保持为 1，SDA 从 0 变换到 1 时，标明传输结束。

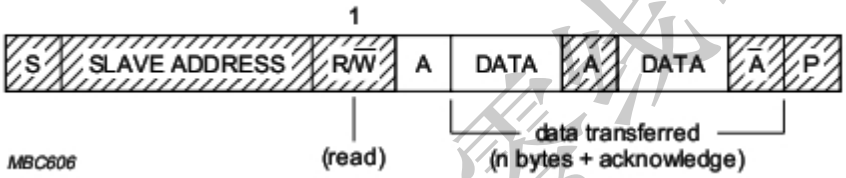
从这个时序图中可以看到，SCL 很重要，并且哪个时钟沿是干嘛的，都是确定好的。比如，前面 7 个必定是地址，第 8 个是读写标志，数据传输必须是 8 位，必须接个 ACK 信号等等。

前面的时序图并没有标明数据传输的方向，我们现在看看写操作的数据流向：



网格的是主设备发送的，白色格子是从设备发送的。从图示中可以看到，对于写操作，从设备都只是发送 ACK 进行确认而已。

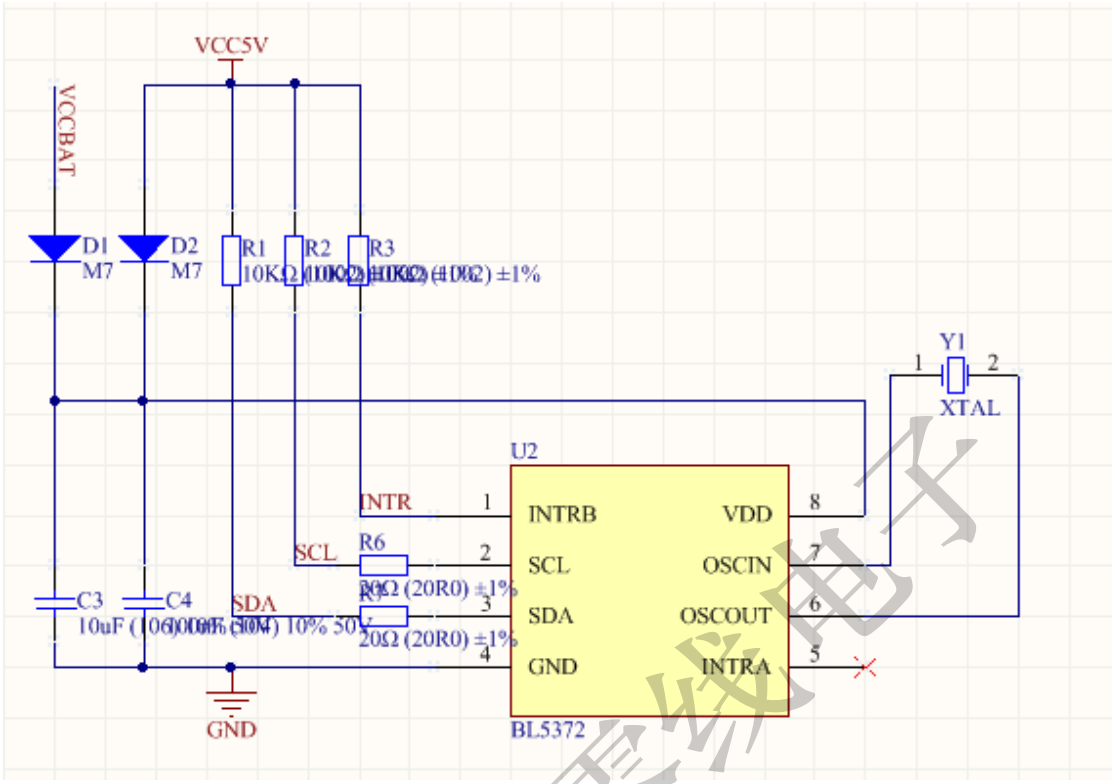
而读操作的数据流向，就有所不同，如图：

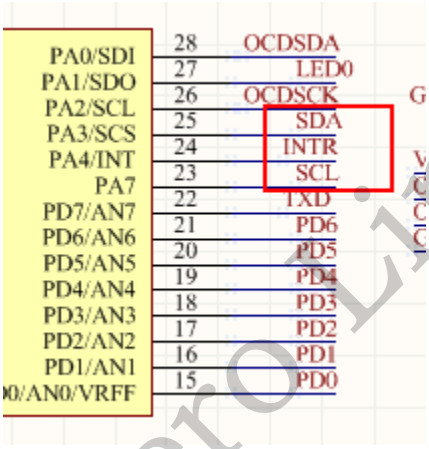


这时候，从设备除了发送 ACK 以外，紧跟着的还有数据。

8.3 硬件设计

使用 IO 口模拟 IIC 协议，和 BL5372 通信。
将中断 INTRB 连接到单片机外部中断口。





8.4 软件设计

本程序涉及的内容包括：

- 使用软件模拟 IIC 控制实时时钟，对时间进行设置和读写。
- 使用软件模拟 UART 控制数码管显示。
- 使用内部 EEPROM 来保存变量，对是否进行时钟初始化进行控制。
- 使用单片机的外部中断 IO 口 INTR，进行时钟刷新周期控制。INTR 中断信号由实时时钟产生。
- 修改实时时钟的时间的方法：修改 TIME_SET_FLGA 标志为 1 字节其他数字，修改时间

宏定义。重新下载程序即可。

```
68
69    #define TIME_SET_FLGA    0xf5     //EEPROM中保存的时钟设定标志
70
71    #define YEAR     0x17
72    #define MONTH    0x09
73    #define DAY      0x28
74    #define WEEK     0x04
75    #define HOUR     0x22
76    #define MIN      0x12
77    #define SEC      0x00
78
```

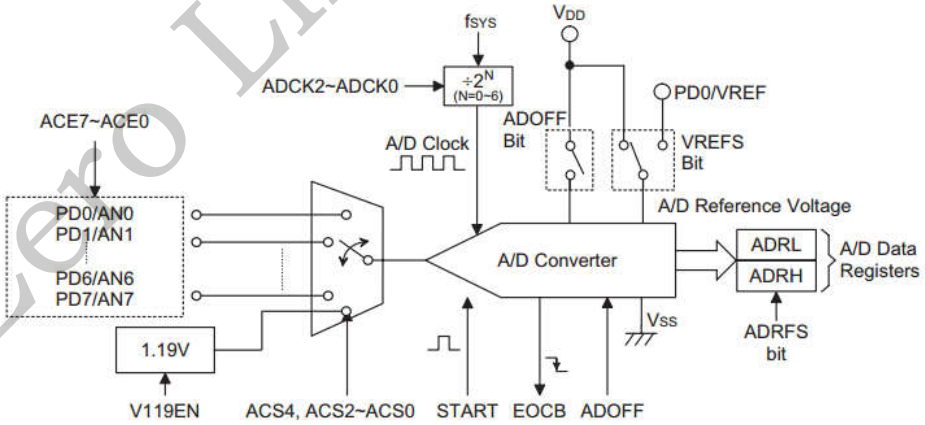
程序主要是为了让用户熟悉单片机资源，比如这个例子的重点是模拟 IIC 和外部中断以及单片机内部 EEPROM 的使用。

另外对于实时时钟 BL5372 的其他应用，这里不做展开，有需要的可以和卖家联系。

9 .ADC 模数转换

9.1 A/D 转换器简介

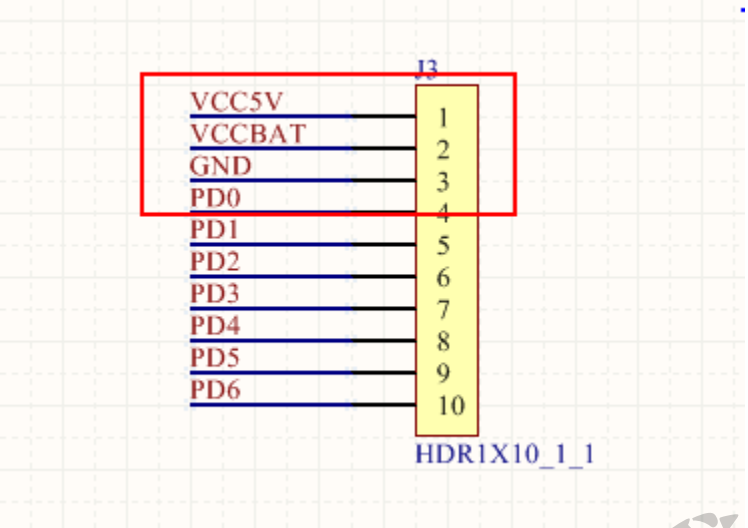
此单片机都包含一个多通道的 A/D 转换器， 它们可以直接接入外部模拟信号（来自传感器或其它控制信号） 并直接将这些信号转换成 12 位的数字量。



A/D 转换器结构

从图上可以看到，AD 转换器可以采样从 ANX 输入的模拟电压，或者对内部的 V119EN 进行采样。

9.2 硬件设计



通过跳线将 VCC5V 或者 VCCBAT 或者 GND 连接到 PD0 口，进行 ADC 测量。

9.3 软件设计

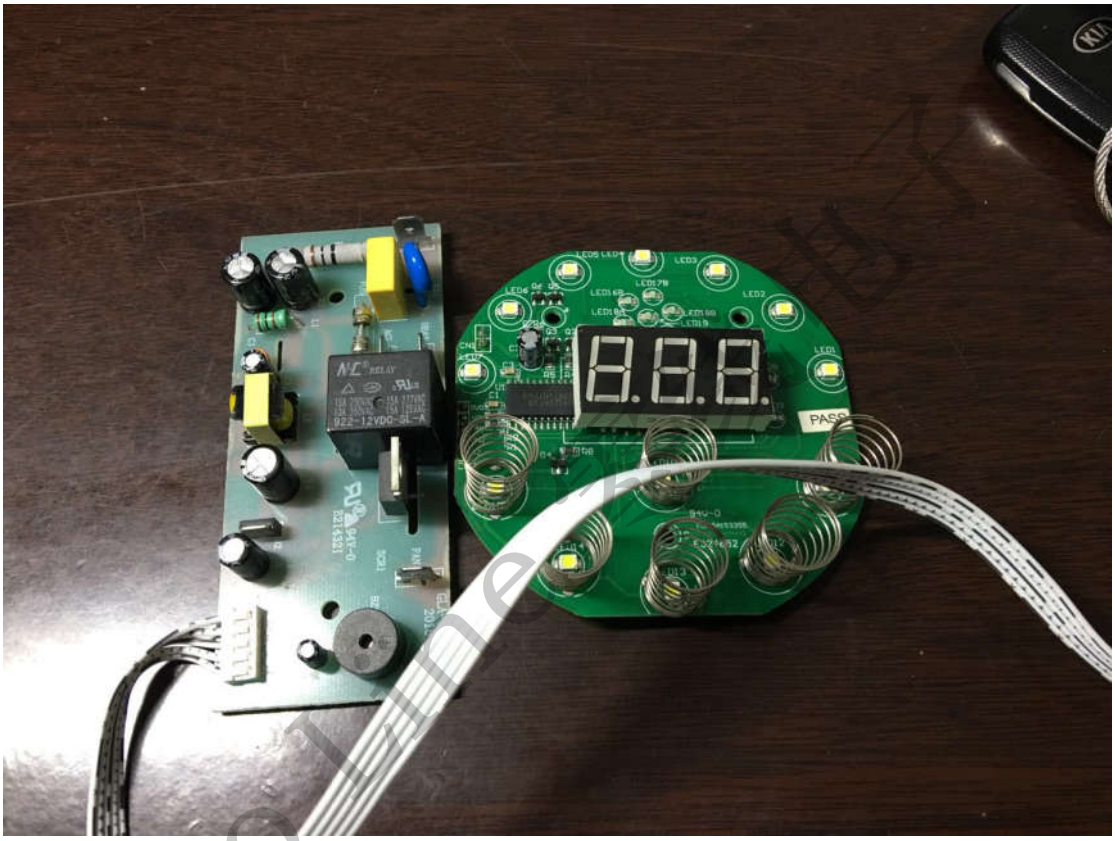
测量 PD0 口的 ADC 值，并且在数码管显示 ADC 采样值。

当 PD0 连接到 GND 时，显示 0。
当 PD0 连接到 VCC 时，显示 4095。
当 PD0 连接到 VBAT 时，显示 2500 左右。

后序

在实际项目中，本人使用 BS84C12 单片机做过空气炸锅和煮水器，均已经量产。特别是空气炸锅，出货量也比较大，功能稳定。后面重新整理资料，可以开源给各开发板买家。

用户通过开发板学习，再学习实际产品的项目程序，能有更好的进步。



上图是空气炸锅项目对应的已经批量的硬件。

上图是淘宝找的对应的显示效果。