阅读材料 2　单片机仿真方法

本篇将介绍 Proteus 三大主要功能中的第二大功能——单片机仿真功能。上篇已介绍过，系统给新建项目创建的第二个标签页是 Source Code，即源代码标签页。源代码标签页的主要功能是程序编辑、代码编译以及程序仿真调试。为了掌握 Source Code 的方法，需要从认识它的工作界面开始。

2.1 Source Code 工作界面

项目新创建后的 Source Code 工作界面如图 B.32 所示。

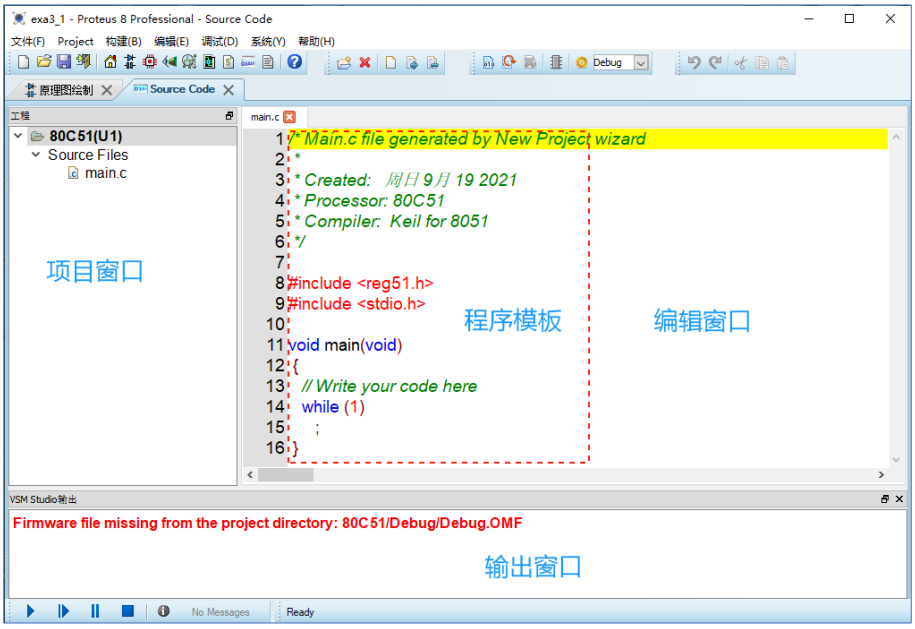


图 B.32　Source Code 标签页的最初界面

由图可知，Source Code 界面中有 3 个主要窗口，即项目窗口、编辑窗口和输出窗口。其实 Source Code 有两个工作界面，图 B.32 只是程序编辑界面，另一个是代码调试界面（图 B.45）。本节先介绍程序编辑界面。

1. 项目窗口

在项目窗口中，以项目树的形式列出了本项目包含的所有文件。项目树有 3 个层级，第一级为控制器（Controller），即项目创建时选定的微控器型号；第二级为文件类型，可以是 Design Files、Source Files、Header Files 和 Link Script Files 等类型；第三级为程序名。

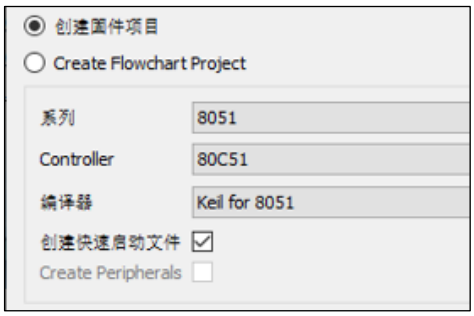


图 B.33　项目树的设置

如果在创建 80C51 的固件对话框中，将编译器指定为 Keil for 8051 且勾选了“创建快速启动文件”（如图 B.33 所示），系统会在项目窗口中添加 80C51 微控器、Source Files 类型名和 main.c 程序名的项目树（如

图 B.32 所示）。

如果原理图中有多个 80C51 微控器，则每个 80C51 都可有独立的项目树（如图 B.34 所示）。

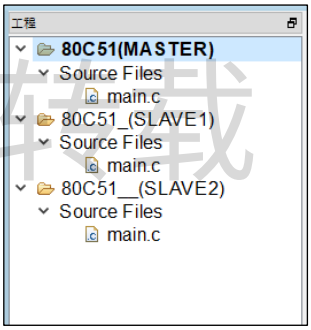


图 B.34 项目树

如果将编译器指定为 ASEM-51（系统内嵌的）时，程序名默认为 main.asm。

2. 编辑窗口

编辑窗口的主要功能是进行程序文本处理，其基本功能等同于普通记事本的文本查找、替换、复制、粘贴、撤销和恢复等功能。

除了上述功能外，程序文本中还具有许多普通记事本没有的特殊信息要求，如对变量、常量、运算符、关键字、注释等信息的不同显示，以及程序代码的缩进、高亮、行号、括号匹配等格式信息。

单击菜单【系统】→【编辑器配置】，可打开如图 B.35 所示的“编辑器配置”对话框。

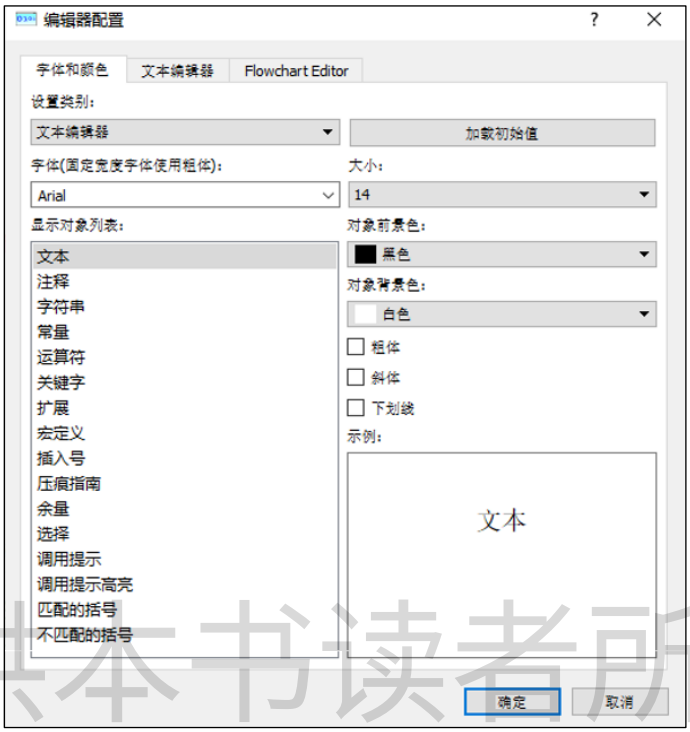


图 B.35 编辑器配置对话框

图中对话框有 3 个组成部分，其中“字体和颜色”标签页可对程序文本进行字体和颜色的设置，使得程序逻辑更加清晰便于理解；“文本编辑器”标签页可对程序代码进行格式设置，使得程序结构错落有致美观整齐；“Flowchart Editor（流程图编辑器）”标签页可对流程图风格进行设置，确保构建流程图项目时正确无误。

采用这些设置后，程序结构会更加规范有助于理解。一般情况下，源程序采用系统默认设置即可。

3. 输出窗口

输出窗口主要用于程序调试和编译过程中的显示信息，以及程序与原理图级联的状态信息。稍后会结合实例进行介绍，此处暂时跳过。

2.2 Source Code 编译设置

前文中介绍过，Proteus 可对电脑中安装好的编译器进行自检并能在后台调用其进行程序编译。如果创建新项目时指定编译器为 ASEM-51，则只能用于编译 A51 汇编语言的程序；如果指定编译器为 Keil for 8051，则既能编译 A51 汇编程序，也能编译 C51 程序。源程序经过编译后可产生微控制器的固件程序（firmware），又称为目标代码。

固件程序格式与编译器有关，如果编译器是 ASEM-51，则固件为 hex 格式，如果编译器是 Keil for 8051，则固件为 omf 格式。除了程序控制功能外，hex 和 omf 格式的目标代码文件中都包含有程序调试所需的信息，能借助 Source Code 的仿真调试手段进行程序逻辑错误排查。

然而 hex 和 omf 格式的固件程序中因包含了这些调试信息而使目标代码不够紧凑，若将其固化到微控制器中会影响代码运行效率。因而，最好能编译出两种目标代码，一种用于仿真调试，另一种用于固化到微控制器中。

从 Proteus 8 起编译器可配置为 Debug（调试）和 Release（发布）两种编译模式。前者能产生可仿真的 hex 和 omf 固件，后者则能产生高效运行但不能仿真的 hex 固件。编译器配置方法如下：单击菜单【Project】→【工程设置】→弹出“工程选项”对话框，如图 B.36 所示。

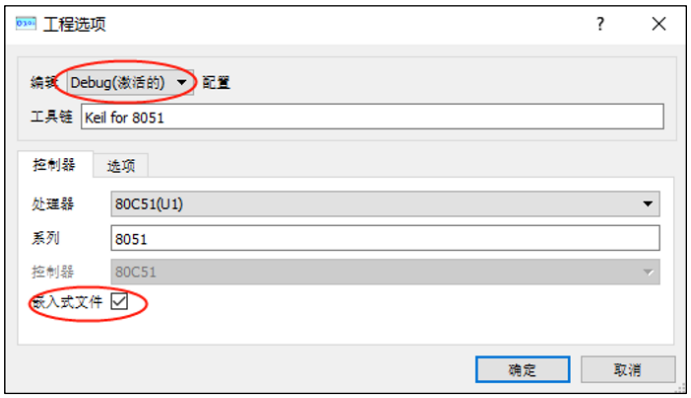
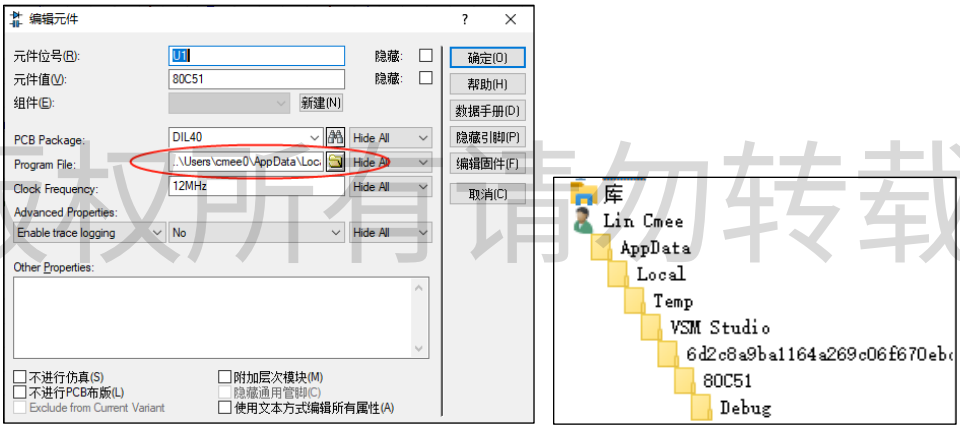


图 B.36 工程选项对话框

当在对话框的“编辑”下拉选项单中选择了“Debug（激活的）”时，编译器产生的是可仿真的固件；而选择了“Release（激活的）”时，编译器产生的是可发布的固件。因此，在源程序编写完成后，应当先采用 Debug 方式进行“构建工程”操作；而在程序仿真调试完成后，再用 Release 方式进行一次“重新构建工程”操作。

编译产生的目标代码文件 omf 是默认保存在 Windows 库文件的 Debug 文件夹里。双击原理图的单片机元件可打开“编辑元件”对话框，如图 B.37(a)所示→单击“Program File”的下拉框按钮，可看到 omf 的默认保存路径，如图 B.37(b)所示。



（a）单片机中加载的 omf　　（b）omf 的默认保存路径

图 B.37 目标代码文件 omf 的默认保存路径

由于 omf 是保存在系统的库文件夹中，相当于是将 omf 进行了仿真加载，因此无需再像早期版本那样进手工加载了，只要编译成功就能直接进行仿真调试。但由于 omf 不是保存在当前项目文件里，若将此项目在其他电脑上运行，就有可能出现找不到 omf 文件的错误。

为此 Proteus 在图 B.36 的 “工程选项”对话框中设置了一个“嵌入式文件”的可选项。勾选了这个选项，omf 文件就被嵌入到系统库文件里；取消勾选，则将 omf 文件保存到当前项目文件夹里。当项目在其他电脑中运行出现找不到 omf 文件的错误时，便可从该项目文件夹中进行手动加载了。

需要说明一点，可能是软件存在 bug 的缘故，利用【Project】→【工程设置】打开“工程选项”对话框，进行 Debug 和 Release 状态切换时，常有无法切换问题出现，但如果是用系统工具栏的“工程设置”选项单进行切换，则能顺利进行。工程设置选项单如图 B.38 所示。

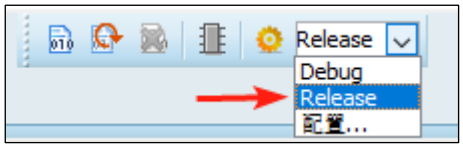


图 B. 38 工程设置选项单

2.3 编辑与编译功能

下面我们将利用阅读材料 1 中完成的电路原理图（图 B.28）和包含快速启动文件的【Source Code】标签页（图 B.32）进行 C51 程序的编辑、编译与仿真调试等工作。

由于创建新项目时选择了 80C51 单片机、Keil for 8051 编译器和快速启动文件，因而 Source Code 界面中包含了 80C51 项目树、main.c 的程序模板，以及添加的 reg51.h 和 stdio.h 两个头文件的引用语句。

1. 建立源程序

待完成的这个实例具有如下运行功能：能将 BUT 按键的单击次数以十进制数形式显示在数码管上，且统计次数应能在 00-99 之间循环进行。

我们已经写好了相应的 C51 源程序，它由由主函数、延时函数以及变量声明等语句所组成，源程序以 txt 本文形式保存，如图 B.39 所示。

```
*test.txt - 记事本
文件(F)  编辑(E)  格式(O)  查看(V)  帮助(H)
#include <reg51.H>                    //51头文件
sbit P3_7=P3^7;
unsigned char code table[]={0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f,0x6f};
unsigned char count;
void delay(unsigned int time)   //延时
{       unsigned int j = 0;
        for(;time>0;time--)
                for(j=0;j<125;j++);
}


void main()
{ count1=0;                          //计数器赋初值
  P0=table[count/10];                //P0口显示初值
  P2=table[count%10];                //P2口显示初值
  while(1)                           //进入无限循环
   {      if(P3_7==0)                          //软件消抖,检测按键是否压下
          {       delay(10);
                  if(P3_7==0)                  //若按键压下
                  {       count++;                     //计数器增1
                  if(count==100)               //判断循环是否超限
                          count=0;
                          P0=table[count/10];      //P0口输出显示
                          P2=table[count%10];      //P2口输出显示
                          while(P3_7==0);          //等待按键松开，防止连续计数
                  }
          }
   }
}
```

图 B.39 已完成的 C51 源程序

下面将这段程序粘贴到 Source Code 界面的程序模板中，具体步骤如下。

①由于本例中没有使用 stdio 头文件，故可将图 B.32 中的头文件引导语句“#include stdio.h”删掉（也可继续保留但会使固件变大）；

②将 txt 程序文本全选后，用快捷键“Ctrl+C”（复制）拷贝到裁剪版中；

③用快捷键“Ctrl+V”（粘贴）覆盖掉图 B.32 中的 C51 程序模板内容；

④单击工具栏“保存工程”按钮保存当前结果。完成替换后的 C51 程序如图 B.40 所示。

```
1  /* Main.c file generated by New Project wizard
2  *
3  * Created:  周六 8月 7 2021
4  * Processor: 80C51
5  * Compiler:  Keil for 8051
6  */
7  #include <reg51.h>
8
9  sbit P3_7=P3^7;
10 unsigned char code table[]={0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f,0x6f};
11 unsigned char count;
12 void delay(unsigned int time)   //延时
13 {    unsigned int j = 0;
14      for(;time>0;time--)
15              for(j=0;j<125;j++);
16 }
17
18 void main(void)
19 {
20 { count1=0;                  //计数器赋初值
21   P0=table[count/10];        //P0口显示初值
22   P2=table[count%10];        //P2口显示初值
23   while(1)                   //进入无限循环
24   {    if(P3_7==0)                  //软件消抖,检测按键是否压下
25        {    delay(10);
26             if(P3_7==0)             //若按键压下
27             {    count++;           //计数器增1
28             if(count==100)          //判断循环是否超限
29                     count=0;
30                     P0=table[count/10]; //P0口输出显示
31                     P2=table[count%10];     //P2口输出显示
32                     while(P3_7==0);    //等待按键松开，防止连续计数
```

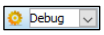
图 B.40 完成替换后的 C51 程序

当然，不采用上述替换方法，而是直接在编辑窗口中录入源程序也是完全可以的。

2. 源程序编译

如果源程序编译后要进行仿真调试且不考虑异地运行，则编译前应将编译器设为 Debug 状态，并将固件嵌入到库文件中；反之如果程序编译后要进行固件发布，则应将编译器设为 Release 状态，并将

固件存入当前项目中。采用前者目的的设置方法是：

①在系统工具栏的“工程设置” 

下拉框中选择 Debug；

②单击工程设置按钮  →在“工程选项”对话框，勾选“嵌入式文件”选项。

下面开始程序编译。

单击菜单【构建】→【构建工程】→系统将在后台调用 Keil for 8051 进行编译。编译信息会显示在输出窗口中如图 B.41 所示。

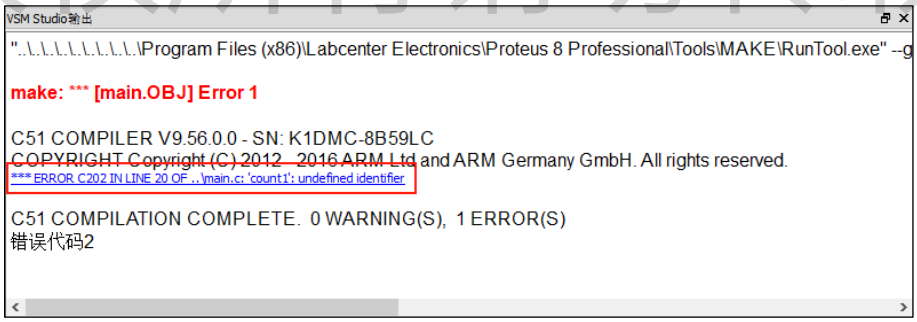


图 B.41 编译信息显示

根据信息提示，“*** ERROR C202 IN LINE 20 OF.main.c:count 1：undefined identifier”可知，在语句第 20 行处有一个未定义的变量 count1，错误代码 C202。

据此查看源程序，发现源程序中将变量名 count 误写为 count1 了。修改此错误，再次编译后的提示信息为图 B.42 所示。

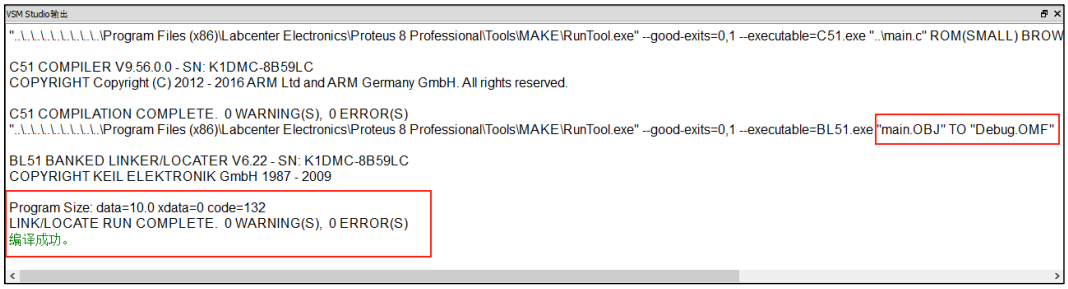


图 B.42 编译信息显示

可见这次编译成功了，致命错误 ERROR(S)和警告错误 WARNING(S)都为零，固件名为 Debug.OMF，代码长度为 132 字节，接下来可以进行仿真了。

3. 仿真控制

单击菜单【调试】→【运行仿真】或直接单击快捷键 F12 均可启动仿真运行；单击菜单【调试】→【停止仿真】可停止仿真。也可通过一组仿真运行按钮实现仿真运行控制，它们位于 Source Code 界面的左下角处，如图 B.43 所示。

图 B. 43 仿真运行按钮

按照从左到右的顺序，仿真运行按钮的功能分别为：PLAY 按钮——开始仿真；STEP 按钮——单步仿真；PAUSE 按钮——暂停仿真；STOP 按钮——停止仿真。

单击 PLAY 按钮启动仿真运行，仿真运行效果如图 B.44 所示。

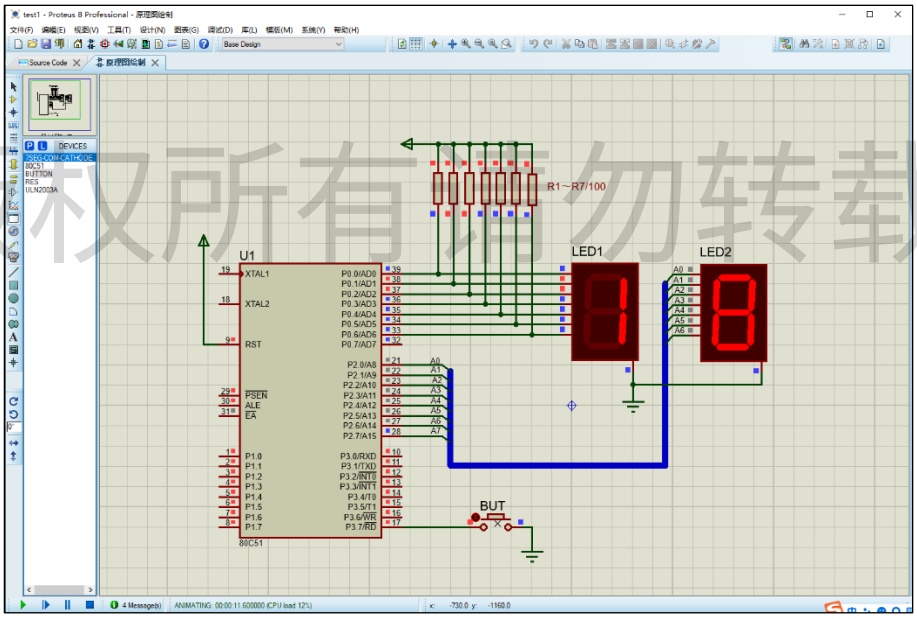


图 B.44 仿真运行效果

仿真运行时，单击按键 BUT，数码管 LED1 和 LED2 会实时显示按键次数；当按键值超过 99 后会自动从 0 开始；电路图中元件的引脚处会有小方块状的电平指示，红色为高电平，蓝色为低电平，灰色为不确定电平。

这说明，此时程序和电路都是正常的，已无逻辑错误，并且实现了预期功能要求。

不过应该知道，这种一次就能成功的情形是不多见的，程序虽然排除了语法错误，也成功进行了编译，但这不等于也没有逻辑错误了。如果仿真运行还有问题，那就需要进行仿真调试了。

2.4 代码仿真调试

Source Code 最强大的功能就是程序代码和电路原理图的联合仿真调试。下面分别介绍有关内容。

1. 代码调试界面

在未启动仿真运行的前提下，单击仿真工具的 STEP 按钮，Source Code 程序编辑界面会变为代码调试界面（单击菜单【调试】→【开始仿真】也可进入这一界面），如图 B.45 所示。

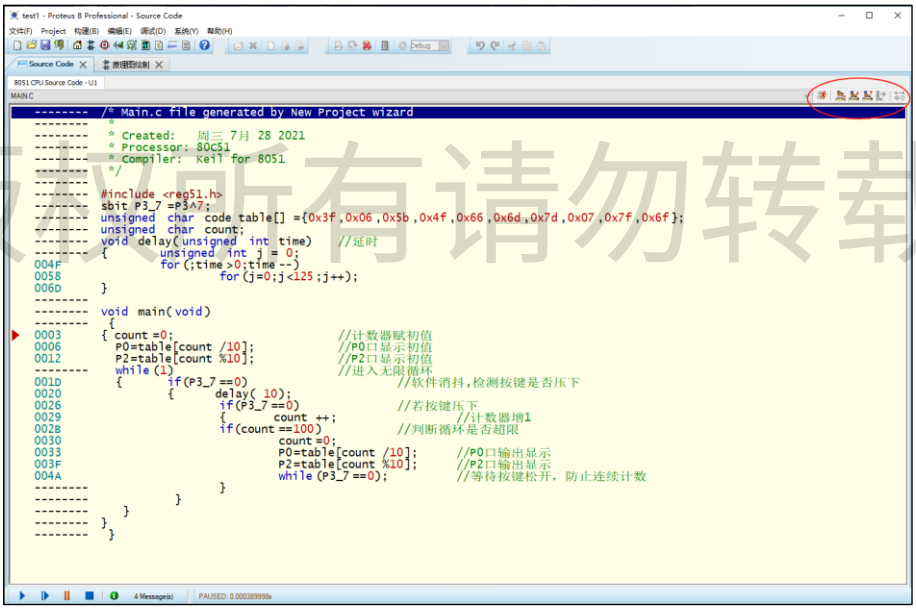


图 B. 45 代码调试界面

可见，代码调试界面的标签名仍然是 Source Code，但却没有了项目窗口和输出窗口，原来白色的编辑窗口已变为淡黄色，窗口中显示的是名为“8051 CPU Source Code-U1”的程序代码，窗口右上角还多了一组调试工具栏（稍后介绍）。

单击菜单【调试】-【8051 CPU】中的各个选项，可打开一组调试显示窗，其中有 Registers（寄存器窗）、SFR Memory（SFR 存储器窗）、Internal(IDATA) Memory（片内 RAM 存储器窗）、Source Code（源代码窗，默认打开）、Variables（变量监视窗）。这些调试显示窗首次打开时是水平折叠状态且位于源代码调试窗的下部，如图 B.46 为 5 个已打开的调试显示窗。

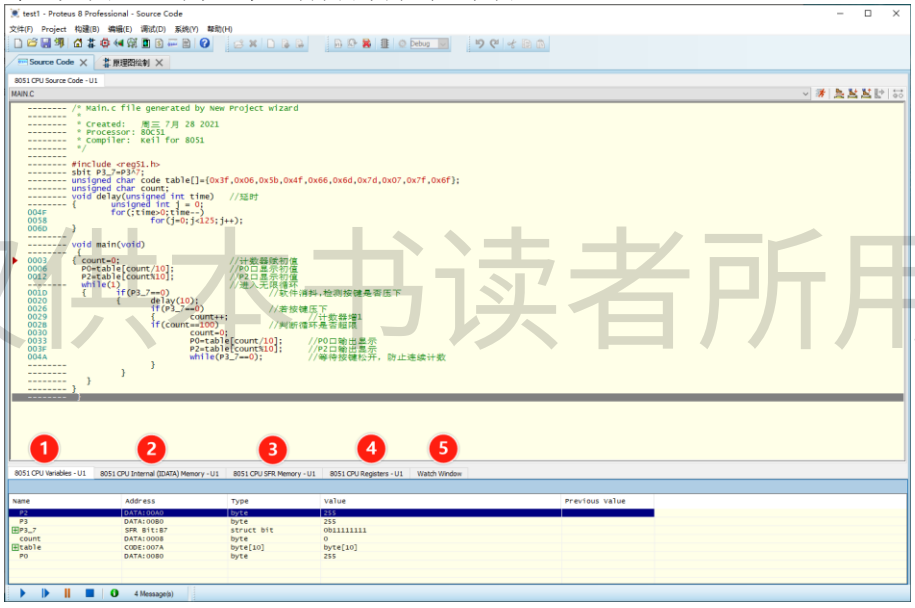


图 B. 46 多个折叠状态的显示窗

2. 调试显示窗的调整

最初的调试显示窗字体较小且窗口折叠在一起不便查看，最好对其进行一些调整。

①调整窗口字体大小

右键单击代码调试窗的空白处可弹出快捷命令弹窗，如图 B.47 所示。

单击“Set Font…”，可打开图 B.48 所示的“字体”设置对话框，其中的“字体”、“字形”和“大小”3 个选项列表都可分别进行设置。例如可将字体大小改为 14，单击“确定”退出后，代码调试窗的字体会立即生效。同理，可对每个调试显示窗的字体都进行修改。

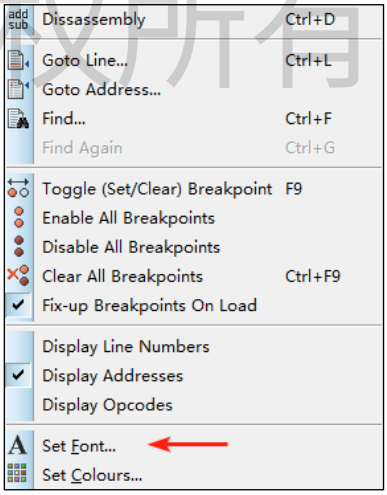


图 B. 47 右键快捷命令弹窗

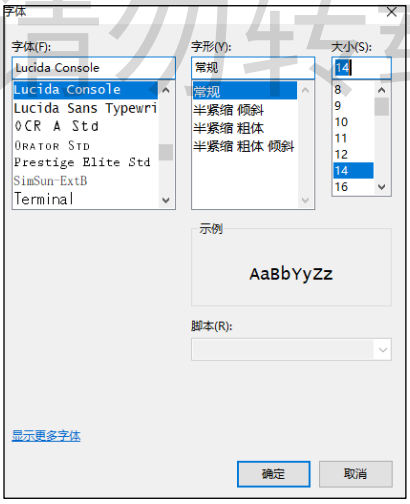


图 B. 48 字体设置对话框

②改为浮动窗

具体做法是，左键按住显示窗的标题栏向上拖动，显示窗即可脱离原先位置变为浮动窗。

将鼠标移到浮动窗的边沿，光标会变成双向空心箭头，再次按住左键拖动便可改变窗口的大小。如此可将图 B.62 中的显示窗都改为浮动窗，如图 B.49 所示。

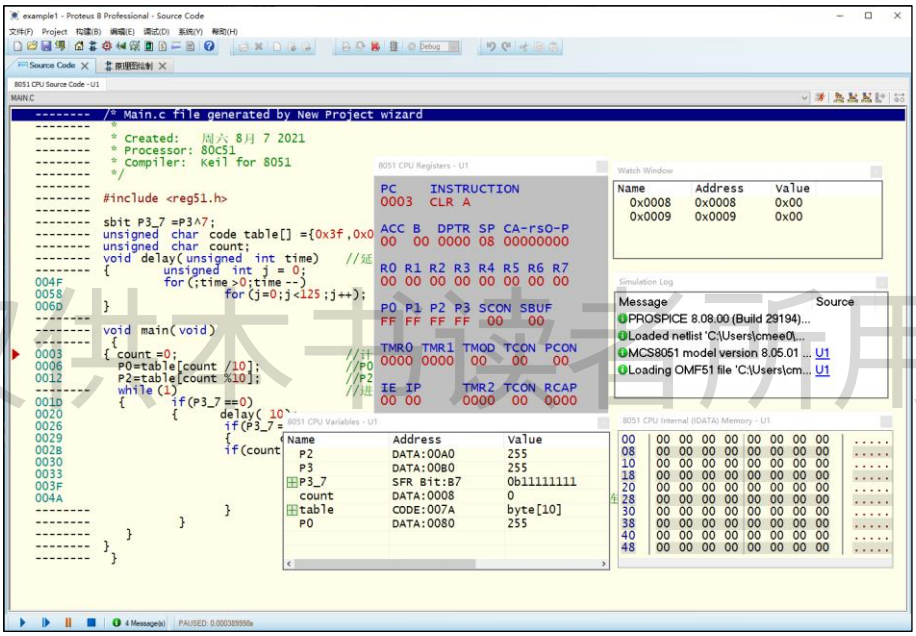


图 B. 49 浮动形式的显示窗

左键双击浮动窗的标题栏，浮动窗将会重新恢复到调整前的状态。

3.调试命令

单击菜单【调试】可在看到所有调试命令，其快捷键和功能如下：

①开始仿真，快捷键 Ctrl+F12，进入代码调试界面等待运行，等同于 STEP 按钮

②暂停仿真，无快捷键，暂停仿真运行，等同于 PAUSE 按钮

③停止仿真，无快捷键，终止仿真运行，等同于 STOP 按钮

④运行仿真，快捷键 F12，全速运行程序，遇到断点会停止运行

⑤不加断点仿真，快捷键 Alt+F12，全速运行程序，遇到断点也不停止运行

⑥运行仿真（时间断点），无快捷键，全速运行程序，直到预设的时间才停止运行

⑦单步执行，快捷键 F10，一行程序执行完后就停止运行

⑧跳进函数，快捷键 F11，单步运行到被调用的函数时，进入函数后继续单步执行

⑨跳出函数，快捷键 Ctrl+F11，单步运行到被调用的函数时，将函数看作是一条语句，整体执行

⑩跳到光标处，快捷键 Ctrl+F10，从当前程序行开始高速运行，到达光标所在行时停止运行

⑪连续单步，快捷键 Alt+F10，一行程序执行完后不停下来慢速执行下一行程序

除上述菜单调试命令外，还可使用位于代码调试界面右上角处的调试工具栏，如图 B.50 所示。

图 B.50 调试工具栏

按照从左到右的顺序，调试工具按钮的名称分别是：

运行仿真、单步运行、跳进函数、跳出函数、运行到光标、断点切换（激活→禁止→撤销）断点。

4. 调试方法

①常用调试方法

启动单步运行后，代码调试窗最左侧会出现一个可随单步运行而移动的红色箭头光标，它会始终指向下一步将要执行的程序行。

单步执行对发现程序错误很有用，但单步执行的效率较低，通常会与其他调试手段配合使用。

例如先执行“运行到光标”，然后再单步执行；也可先运行到断点再单步执行；或采用连续单步加快运行速度。如果单步执行时遇到调用的函数，可以酌情采用“跳进函数”或“调出函数”。

单步运行效果如图 B.51 所示。

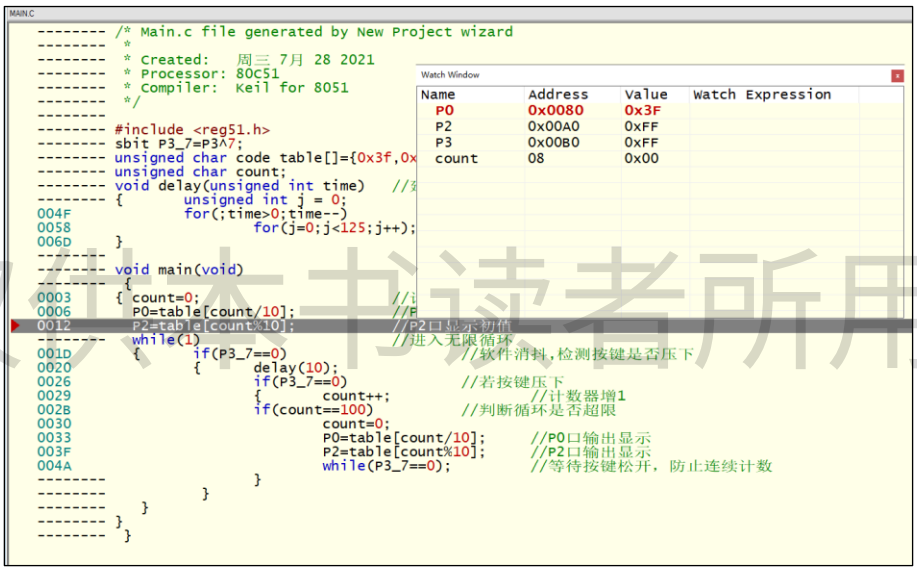


图 B.51 单步运行效果

②断点调试方法

断点调试的基本思想是，提前在程序的关键行处设置一个或多个停止点，当程序运行到此后便会自行停下来，便于通过查看中间运行结果，判断程序是否符合预想运行。

设置断点有 4 种方法，即双击欲设断点程序行、单击工具按钮、单击快捷键 F9 和右键快捷菜

单。

断点可以有 3 种状态，即在程序行前面出现红色“●”符号，表示断点已激活；出现红色“o”，表示断点已禁止；“o”符号消失，表示断点已撤销。重复进行上述设置操作可使断点在 3 种状态之间切换。

断点运行效果如图 B.52 所示。

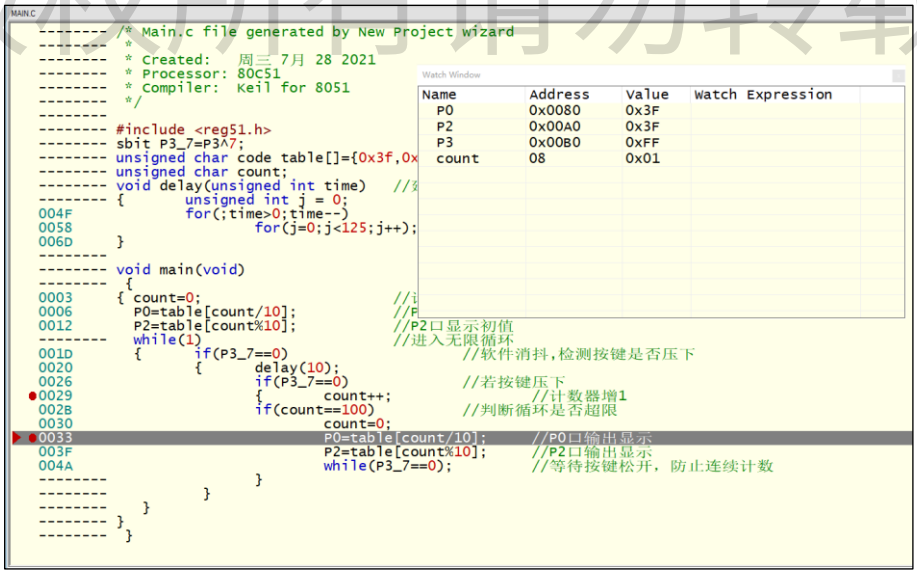


图 B.52 断点运行效果

③Active Popups 调试方法

Proteus 中新增了一个动态调试工具——Active Popups（调试弹出模式）。调试时用该工具在原理图中划定的监视区，会以弹窗形式出现在调试界面中，并能动态反映程序的变化。尤其是当监视区中包含有数码管、示波器或按钮等动态元件时，不仅可在调试中看到数码管显示值的变化，还能单击按钮与仿真程序互动，使程序调试更加直观和灵活。

添加 Active popus 监视区的方法是，在原理图界面中，先单击“模式工具栏”的“Active popus Mode”按钮，然后按住鼠标左键在电路图中拖出一个矩形框作为监视区。例如，可将本实例中的数码管和按钮选为两个监视区，如图 B.53 所示。

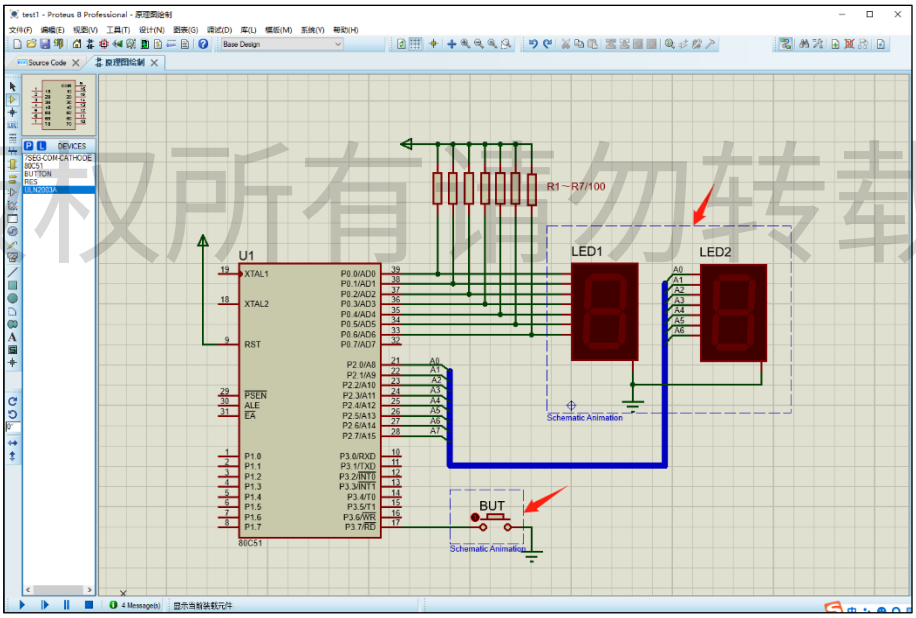


图 B. 53 设置监视区

将 Active Popups 工具和单步或断点结合起来也可实现仿真调试。单击 STEP 按钮使仿真处于待机状态后，监视框内的元器件将出现在 Schematic animation（动画图）窗口里。仿真调试期间通过单击按钮，可改变数码管的变化，就仿佛是在原理图中一样的操作，如图 B.54 所示。

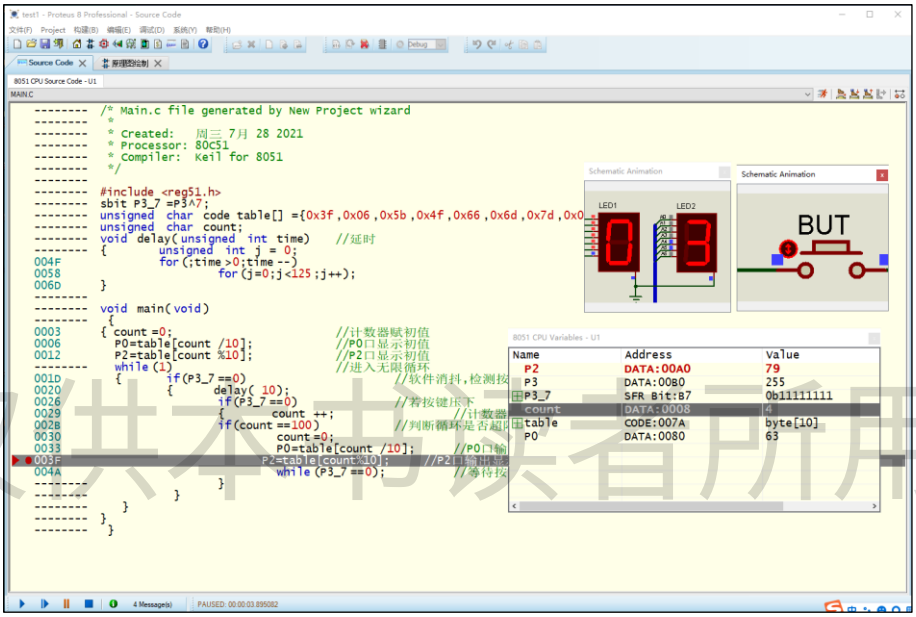


图 B. 54 利用 Active Popups 实现在原理图中的操作效果

此外，单击 Active popups 上的元器件，可弹出该元器件引脚的当前状态值。例如，单击 LED1 数码管，可显示数码管引脚当前状态，这会对了解程序运行状态有帮助，如图 B.55 所示。

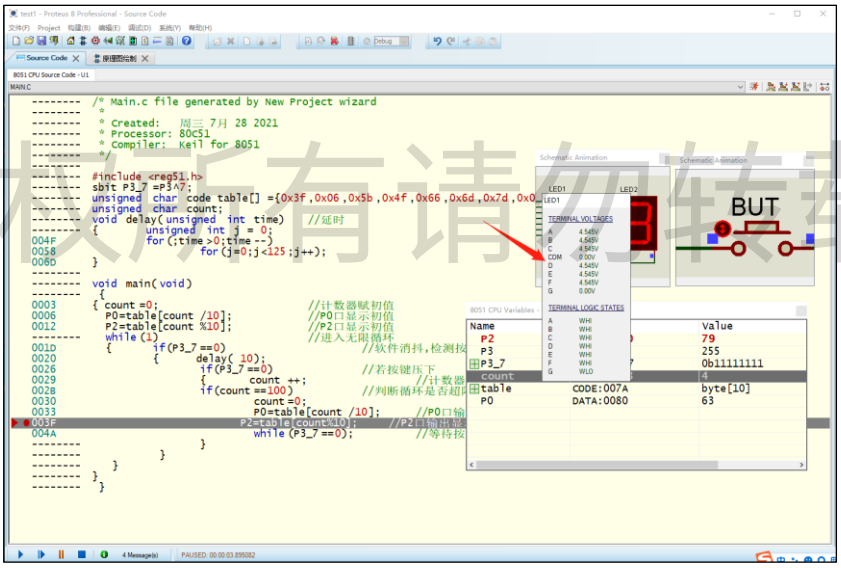


图 B.55 利用 Active popups 显示元器件的工作状态

④发挥调试显示窗的作用

在前述过的调试显示窗中变量监视窗（Variables）对程序的调试最为实用，其特点是能将当前程序中的所有变量自动添加到变量监视窗中。

例如实例程序中用到的变量 P0、P2、P3_7、Ccounter 和数组变量 table 都会被自动加载到变量监视窗，这将给程序调试带来了极大方便。当程序运行时，变量监视窗的内容会随之发生变化。如果当前值有变化的变量还会用红色字体显示，如图 B.56 所示。

8051 CPU Variables - U1

Name	Address	Type	Value
P2	DATA:00A0	byte	63
P3	DATA:00B0	byte	255
P3_7	SFR Bit:B7	struct bit	0b11111111
	SFR Bit:B7	byte bitfield	1
count	DATA:0008	byte	0
table	CODE:007A	byte[10]	byte[10]
table[0]	CODE:007A	byte	63
table[1]	CODE:007B	byte	6
table[2]	CODE:007C	byte	91
table[3]	CODE:007D	byte	79
table[4]	CODE:007E	byte	102
table[5]	CODE:007F	byte	109
table[6]	CODE:0080	byte	125
table[7]	CODE:0081	byte	7
table[8]	CODE:0082	byte	127
table[9]	CODE:0083	byte	111
P0	DATA:0080	byte	63

图 B.56 调试运行时变量监视窗的内容会发生变化

除了变量监视窗，其它几个 51 调试显示窗也都各有特点。例如 Registers（寄存器）窗中列出了所有特殊功能寄存器 SFR，观察 SFR 的内容可以了解到 CPU 资源的变化；SFR Memory（SFR 地址）窗是以存储器形式展示 SFR 的内容，这对将 SFR 作为一般存储器使用时的调试很方便；Internal(IDATA) Memory（IDATA 地址）窗列出了片内 RAM 从 00 至 80H 的存储单元，这对了解数据块的转存结果很方便。可见这些调试显示窗对 A51 汇编程序或 C51 反汇编程序的调试尤为有用。

综上可见，仿真调试是单片机程序开发中的重要环节，优秀的程序员不仅要有丰富的编程知识，还要有敏锐的错误判断能力，这些都是长期实践积累的财富，也是广大学习者努力的方向。