

# 目录

<b>第 1 章 介绍</b>	<b>10</b>
1.1 关于 PROTEUS VSM	10
1.2 关于本文档	10
<b>第 2 章 使用指南</b>	<b>11</b>
2.1 交互式仿真指南	11
2.1.1 介绍	11
2.1.2 绘制电路图	11
2.1.3 编写程序	12
2.2 基于图表的仿真	15
2.2.1 介绍	15
2.2.2 开始	15
2.2.3 激励源	16
2.2.4 探针	17
2.2.5 图表	17
2.2.6 仿真	17
2.2.7 测量	18
2.2.8 使用电流探针	18
2.2.9 频率分析	18
2.2.10 扫描变量分析	19
2.2.11 噪声分析	19
<b>第 3 章 交互式仿真</b>	<b>21</b>
3.1 基本技能	21
3.1.1 仿真控制按钮	21
3.1.2 指示及执行	21
3.1.3 建立一个交互式仿真	22
3.2 测量	24
3.2.1 概况	24
3.2.2 动画效果	24
3.2.3 指示和显示参数信息	26
3.2.4 虚拟仪器	27
3.3 动态步进控制	27
3.3.1 概况	27
3.3.2 仿真帧率	27
3.3.3 帧步进时间	28
3.3.4 单步时间	28

3.4 亮点和提示.....	28
3.4.1 电路时间尺度.....	28
3.4.2 电压范围.....	28
3.4.3 接地.....	29
3.4.4 高阻抗点.....	29
<b>第4章 虚拟仪器.....</b>	<b>30</b>
4.1 虚拟示波器.....	30
4.1.1 总述.....	30
4.1.2 示波器的使用.....	31
4.1.3 使用模式.....	31
4.1.4 示波器的触发.....	32
4.1.5 输入耦合.....	32
4.2 逻辑分析仪.....	32
4.2.1 概况.....	32
4.2.2 使用逻辑分析仪.....	33
4.2.3 缩放和全局显示.....	34
4.2.4 测量.....	34
4.3 虚拟信号发生器.....	34
4.3.1 总述.....	34
4.3.2 使用信号发生器.....	35
4.3.3 使用 AM、FM 调制输入.....	35
4.4 模式发生器.....	36
4.4.1 总述.....	36
4.4.2 使用模式发生器.....	36
4.4.3 模式发生器的引脚说明.....	37
4.4.4 时钟模式.....	38
4.4.5 触发模式.....	38
4.4.6 外部保持.....	40
4.4.7 附加功能.....	40
4.5 定时/计数器.....	42
4.5.1 概述.....	42
4.5.2 使用定时器.....	43
4.5.3 使用频率计算模式.....	43
4.5.4 使用计数器.....	44
4.6 虚拟终端.....	44
4.6.1 概述.....	44
4.6.2 使用虚拟终端.....	45
4.6.3 MAX232 模型.....	46
4.7 I2C 调试器.....	47
4.7.1 概述.....	47
4.7.2 器件引脚界面说明.....	47

4.7.3 I2C 调试窗口.....	47
4.7.4 发送和接收数据.....	49
4.7.5 模型的特性.....	50
4.7.6 语法参考.....	50
4.8 SPI 调试器.....	50
4.8.1 简述.....	50
4.8.2 原理图部件: .....	51
4.8.3 SPI 调试窗口.....	51
4.8.4 模型的特性.....	53
4.8.5 使用 SPI 调试器.....	54
4.9 电压表和电流表.....	55
<b>第 5 章 微处理器仿真.....</b>	<b>56</b>
5.1 基于 Proteus VSM 源代码调试.....	56
5.1.1 概述.....	56
5.1.2 源代码窗口.....	56
5.1.3 支持的仿真文件格式.....	58
5.1.4 单步执行.....	59
5.1.5 使用断点.....	60
5.1.6 变量窗口.....	60
5.1.7 多个 CPU 仿真.....	62
5.2 源代码控制系统.....	63
5.2.1 概述.....	63
5.2.2 创建源代码文件.....	63
5.2.3 在源代码上工作.....	63
5.2.4 安装第三方目标代码生成工具.....	64
5.2.5 使用 MAKE 程序.....	64
5.2.6 使用第三方源代码编辑器.....	65
5.2.7 使用第三方 IDE.....	65
5.2.8 调试工具.....	66
5.3 观测窗口.....	70
5.3.1 给观测窗口增加项目.....	70
5.3.2 观测点条件.....	71
5.4 断点触发.....	72
5.4.1 建立硬件断点.....	72
5.4.2 电压断点触发 (RTVBREAK) .....	73
5.4.3 电流断点触发 (RTIBREAK) .....	73
5.4.4 数字断点触发 (RTDBREAK) .....	73
5.4.5 电压监测器 (RTVMON) .....	74
5.4.6 电流监测器 (RTIMON) .....	74
<b>第 6 章 基于图表的仿真.....</b>	<b>75</b>
6.1 介绍.....	75

6.2 建立一个基于图表的仿真.....	75
6.2.1 概况.....	75
6.2.2 绘制电路.....	75
6.2.3 放置探针和发生器.....	75
6.2.4 放置图表.....	76
6.2.5 在图表中添加轨迹.....	77
6.2.6 仿真过程.....	77
6.3 图表对象.....	78
6.3.1 概况.....	78
6.3.2 当前图表.....	78
6.3.3 放置图表.....	78
6.3.4 编辑图表.....	78
6.3.5 在图表中添加轨迹.....	78
6.3.6 添加曲线命令对话框.....	79
6.3.7 图表轨迹编辑.....	80
6.3.8 改变轨迹的顺序或颜色.....	80
6.3.9 手工 Y 轴设置.....	80
6.4 仿真过程.....	81
6.4.1 命令驱动仿真.....	81
6.4.2 执行仿真.....	81
6.4.3 仿真开始后发生的事件.....	82
<b>第 7 章 分析类型.....</b>	<b>84</b>
7.1 介绍.....	84
7.2 模拟信号瞬态分析.....	85
7.2.1 概述.....	85
7.2.2 计算方法.....	85
7.2.3 使用模拟图表.....	85
7.2.4 定义模拟轨迹表达式.....	87
7.3 数字信号瞬态分析.....	87
7.3.1 概述.....	87
7.3.2 计算方法.....	87
7.3.3 使用数字图表.....	89
7.3.4 数字数据的显示.....	89
7.4 混合模式瞬态分析.....	90
7.4.1 概述.....	90
7.4.2 计算方法.....	90
7.4.3 使用混合图表.....	91
7.5 频率分析.....	91
7.5.1 概述.....	91
7.5.2 计算方法.....	91
7.5.3 使用频率图表.....	91

7.6 DC 扫描分析.....	93
7.6.1 概述.....	93
7.6.2 计算方法.....	94
7.6.3 使用 DC 扫描分析图表.....	94
7.7 AC 扫描分析.....	95
7.7.1 概述.....	95
7.7.2 计算方法.....	95
7.7.3 使用 AC 扫描图表.....	95
7.8 DC 转移曲线分析.....	96
7.8.1 概述.....	96
7.8.2 计算方法.....	96
7.8.3 使用 AC 扫描图表.....	97
7.9 噪声分析.....	98
7.9.1 概述.....	98
7.9.2 计算方法.....	98
7.9.3 使用噪声分析图表.....	99
7.10 失真分析.....	99
7.10.1 概述.....	99
7.10.2 计算方法.....	99
7.10.3 使用失真分析图表.....	100
7.11 傅立叶分析.....	100
7.11.1 概述.....	100
7.11.2 分析方法.....	100
7.11.3 使用傅立叶分析图表.....	101
7.12 音频分析.....	101
7.12.1 概述.....	101
7.12.2 分析方法.....	101
7.12.3 使用音频分析图表.....	101
7.13 交互式分析.....	102
7.13.1 概述.....	102
7.13.2 计算方法.....	102
7.13.3 使用交互式分析图表.....	102
7.14 数字一致性分析.....	103
7.14.1 概述.....	103
7.14.2 分析方法.....	103
7.14.3 使用数字一致性分析图表.....	103
<b>第 8 章 激励源和探针.....</b>	<b>107</b>
8.1 激励源.....	107
8.1.1 概述.....	107
8.1.2 放置激励源.....	107
8.1.3 编辑激励源.....	108

8.1.4 直流激励源.....	108
8.1.5 正弦激励源.....	108
8.1.6 脉冲激励源.....	109
8.1.7 指数激励源.....	109
8.1.8 单频率调频激励源.....	110
8.1.9 分段线性激励源.....	110
8.1.10 文件激励源.....	111
8.1.11 音频激励源.....	112
8.1.12 数字激励源.....	112
8.2 探针.....	113
8.2.1 概述.....	113
8.2.2 探针放置.....	113
8.2.3 探针设置.....	113
<b>第 9 章 脚本化信号源.....</b>	<b>114</b>
9.1 前言.....	114
9.2 创建、编辑及调试 EasyHDL 脚本.....	115
9.3 程序结构.....	119
9.4 语法图解析.....	120
9.5 使用 EASYHDL 规定模拟信号.....	121
9.5.1 声明输出.....	121
9.5.2 程序流.....	121
9.5.3 规定输出转换.....	123
9.6 使用 EASYHDL 规定数字信号.....	124
9.6.1 声明输出.....	124
9.6.2 程序流.....	124
9.6.3 规定输出转换.....	125
9.7 变量.....	126
9.7.1 变量声明.....	126
9.7.2 数据类型.....	127
9.7.3 字符串类型.....	127
9.7.4 BOOL 及 PIN 类型.....	127
9.7.5 节点类型.....	128
9.7.6 指定变量.....	129
9.8 表达式.....	130
9.8.1 操作符.....	130
9.8.2 值.....	131
9.8.3 函数.....	132
9.9 系统常量及变量.....	133
9.9.1 数据常量.....	133
9.9.2 事件 ID 常量.....	133
9.9.3 网络状态常量.....	134

9.9.4 PIN 及 BOOL Pin 活动常量.....	134
9.9.5 TDSCALE 系统变量.....	134
9.9.6 RANDOM 系统变量.....	134
9.9.7 REALTIME 及 EVTTIME 系统变量.....	135
9.10 命令参考.....	135
9.10.1 ABORT 命令.....	135
9.10.2 ALIAS 命令.....	135
9.10.3 BREAK 命令.....	136
9.10.4 CALLBACK 命令.....	137
9.10.5 CONTINUE 命令.....	138
9.10.6 DATA 命令.....	138
9.10.7 END 命令.....	139
9.10.8 FOR...TO...STEP...NEXT 命令.....	139
9.10.9 GOTO 命令.....	140
9.10.10 IF...ELIF...ELSE...ENDIF 命令.....	141
9.10.11 MAP ON...ENDMAP 命令.....	142
9.10.12 ON BOOT...ENDON 命令.....	143
9.10.13 ON EVENT...ENDON 命令.....	144
9.10.14 ON TIMER...ENDON 命令.....	145
9.10.15 PRINT 命令.....	146
9.10.16 READ 命令.....	146
9.10.17 REM 命令.....	147
9.10.18 REPEAT...UNTIL 命令.....	147
9.10.19 RESTORE 命令.....	148
9.10.20 SEED 命令.....	149
9.10.21 SLEEP 命令.....	149
9.10.22 SWITCH...ENDSW 命令.....	150
9.10.23 TIMER 命令.....	151
9.10.24 WAIT 命令.....	152
9.10.25 WHILE...WEND 命令.....	152
<b>第 10 章 SPICE 模型的应用.....</b>	<b>154</b>
10.1 前言.....	154
10.2 使用一个 SPICE 模型（模型卡）.....	155
10.3 SPICE 模型库.....	155
10.4 *SPICE 脚本.....	156
10.5 模型仿真失败的处理.....	156
<b>第 11 章 进阶.....</b>	<b>158</b>
11.1 温度模型.....	158
11.2 固化模型数据.....	158
11.3 地和电源范围.....	158
11.3.1 为什么需要一个地.....	158

11.3.2 电源.....	159
11.4 初始条件.....	160
11.4.1 前言.....	160
11.4.2 设定一个网络的初始条件.....	161
11.4.3 设定器件的初始条件.....	161
11.4.4 NS (NODESET)属性.....	161
11.4.5 PRECHARGE 属性.....	162
11.5 数字仿真范例.....	162
11.5.1 九态逻辑.....	162
11.5.2 不确定状态.....	163
11.5.3 浮空输入行为.....	163
11.5.4 毛刺的处理.....	163
11.6 混合模式接口模型 (ITFMOD) .....	165
11.6.1 概述.....	165
11.6.2 使用 ITFMOD 属性.....	166
11.7 录音机和电路分区.....	167
11.7.1 概述.....	167
11.7.2 单个部分的仿真.....	167
11.7.3 录音机对象.....	168
11.7.4 录音机模式.....	168
11.8 仿真控制属性.....	169
11.8.1 概述.....	169
11.8.2 误差属性.....	169
11.8.3 MOSFET 属性.....	170
11.8.4 迭代属性.....	170
11.8.5 温度属性.....	170
11.8.6 数字仿真器属性.....	170
11.9 仿真模型的类型.....	171
11.9.1 如何描述器件含有模型.....	171
11.9.2 原型模型 (Primitive Models) .....	172
11.9.3 原理图型模型 (Schematic Models) .....	172
11.9.4 VSM 模型 (VSM Models) .....	173
11.9.5 SPICE 模型 (SPICE Models) .....	173
11.10 如何提高交互仿真的速度.....	174
11.10.1 前言.....	174
11.10.2 使用数字的电阻和二极管模型.....	174
11.10.3 优化外部 RAM 和 ROM 的访问.....	176
<b>第 12 章 错误处理.....</b>	<b>178</b>
12.1 仿真日志.....	178
12.2 网络表错误.....	178
12.3 链接错误.....	178



12.4 分区错误.....	179
12.5 电源范围定义错误.....	179
12.6 仿真错误.....	179
12.7 收敛问题.....	179

# 介绍

## 关于 PROTEUS VSM

传统的电路仿真是一个非交互性事件，网络表需人工准备，得到的仿真结果也仅仅是大量的数字代码，最多可以得到用符号构成的伪图形输出来显示电压和电流的波形。

现在的原理图输入及图形输出都已经有了标准可循，但是仿真的过程仍然是一个非交互的过程——绘制电路，运行并得到一些仿真输出结果。如果你测试的电路是一个静态过程的话，这是一个非常好的方式；比如 1MHZ 的振荡器。然而现在的很多设计是交互式的过程，比如说一个报警器，（当键盘上有错误的 PIN 输入时，报警器能进行报警工作）。这种仿真是没有办法实现的。

现在教育圈一直在进行一个尝试，就是将电路像现实一样做交互式仿真。其中最大的困难是用于交互的动态器件很难建模。现在只有一些简单的模型，比如开关、灯、电机等，在专业电路设计中这些模型还远远满足不了要求。再者，电路仿真缺少许多的信息，比如核心器件数字模型中的定时信息。

PROTEUS VSM 很好的提供了解决方案。它基于工业标准 SPICE3F5，含有动态器件的高级混合电路仿真器。另外还提供一个结构，用户可以据此创建动态模型。很多种类的动态模型无需编程直接建模。有了动态模型，使用者可以像实际设计一样对电路进行交互式仿真。

PROTEUS 有一大批仿真模型，包含现在主流的处理器、一系列接近现实元件的动态模型（比如 LCD、LED 显示，键盘，RS232 终端等）。这样，使用者无需架构硬件电路及仿真设备，直接设计的单片机系统，编写程序，运行仿真，减少产品的开发周期。

现在的 PC 的处理能力已经得到了飞速发展。一个 300MHZ 的奔腾 II 的 PC 可以实时的仿真一个简单的单片机系统。当然有些稍微复杂一些的设计对仿真环境要求苛刻一些，这时使用者应清楚仿真与现实实时性的差别。如果你注重实时性，你可以购买一个 2GHz 双核 PC 机。但是这仍然不足以能完全实时交互仿真。

## 关于本文档

这个手册包括一些背景知识及操作指南。帮助使用者更快的掌握 VSM 仿真。

PROTEUS VSM 有两种仿真方式：交互式仿真和基于图表仿真。一般来说，交互式仿真主要用来验证设计电路是否正常工作。高级图形仿真主要用来测量一些电路细节。当然设计中也可以同时使用两种仿真方式。

手册中有一些例子，使用者可以按照说明按部就班来学习软件的基本使用。

注：对于 VSM 模型构建，在 VSM SOFTWARE DEVELOPMENT KIT (SDK) 中有详细的说明，它作为一个在线资源提供给正版用户。

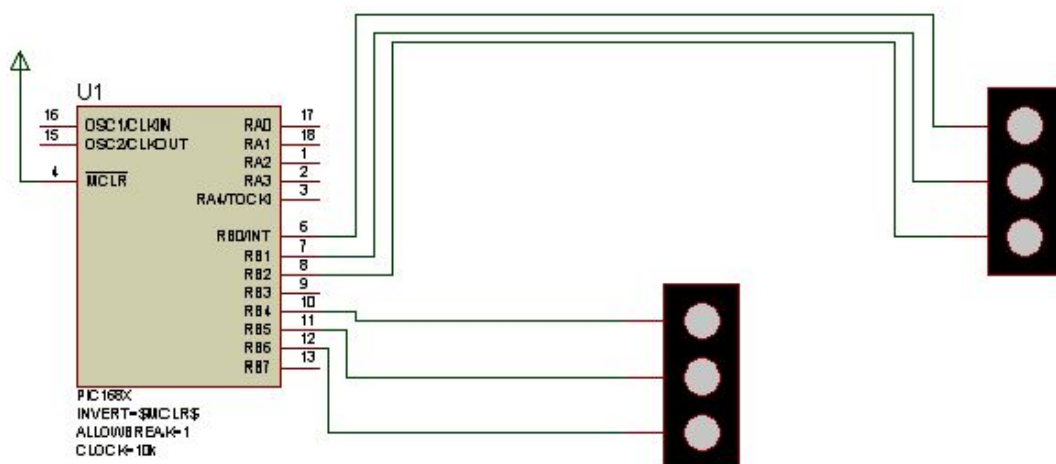
# 使用指南

## 交互式仿真指南

### 介绍

这个指南通过制作一个例子，展示怎样使用 PROTEUS VSM 进行交互式仿真。其中的重点是动态器件的使用及 ISIS 编辑器源码调试，ISIS 的一些基本操作在 ISIS 帮助手册中有详细的说明。

这个电路是一个交通灯控制系统，使用的芯片是 PIC16F84。原理图如下：



这个原理图可以在 PROTEUS 安装文件夹下的 Samples\Tutorials\下找到，也可以直接手工绘制。

### 绘制电路图

#### 1. 放置元件：

- 新建一个原理图设计，选择 component 模式按钮，再点击对象选择窗口上的‘P’钮进入元件库中。
- 在 KEYWORD 对话框中输入关键词，比如说输入 TRAFFIC，在结果窗口就会显示查找的结果，双击查询结果，对应元件就会添加到对象选择列表当中。用同样的方法添加 PIC16F84。
- 当把原理图所有的元件的都选择添加好以后，将元件摆放到原理图编辑窗口当中，其中 TRAFFIC LIGHTS 放置两次，PIC 放置一次。（选中元件，其呈高亮状态，在预览窗口将出现元件预览。）

## 2. 移动和旋转:

- 移动: 将鼠标移到元件上, 右键单击 (元件呈高亮状态), 按住鼠标左键并移动元件, 松开左键, 元件位置就确定下来。注意一点, 这时元件还处于选取状态, 再点击左键, 元件再次放置。
- 旋转: 右键选中元件, 元件呈高亮状态, 再在旋转按钮框中点击一种旋转方式, 元件就会以 90 度进行旋转。
- 缩放及捕捉:

缩放: 在 VIEW 菜单下含有四种缩放方式:

- ◆ 按 F6 或单击 ZOOM IN 按钮, 鼠标所在周围放大。
- ◆ 按 F7 或单击 ZOOM OUT 按钮, 鼠标所在周围缩小。
- ◆ 按 F8 或单击 ZOOM ALL 按钮, 返回整张设计图。
- ◆ 单击 ZOOM TO AREA, 可以选择一部分进行放大。
- ◆ 按住 SHIFT 键并用鼠标左键拖曳一个框, 释放 SHIFT 键后就会放大到所选区域。

捕捉: ISIS 中有一个强大个功能是实时捕捉。当鼠标靠近引脚时, 系统会自动进行捕捉。极大的方便原理图的绘制。该功能在 Tool Manu 中, 默认是打开的。

## 3. 连线:

导线绘制过程:

1. 选中连线模式: 导线还是总线;
2. 点击起点及终点, 系统自动拉出导线。
3. 导线复制: 双击起点, 系统走出和上一条导线相同的轨迹。
4. 对导线进行拖曳, 调整导线位置。

# 编写程序

## 1. 程序编辑

PROTEUS 自带多种汇编编辑编译系统, 使用编辑环境输入下列 PIC 汇编源代码:

```
LIST    p=16F84 ; PIC16F844 is the target processor
        #include "P16F84.INC"      ; Include header file
        CBLOCK 0x10                ; Temporary storage
        State
        11,12
        ENDC
        org      0                  ; Start up vector.
        goto     setports           ; Go to start up code.
        org      4                  ; Interrupt vector.
halt     goto     halt               ; Sit in endless loop and do nothing.
setports clrw                       ; Zero in to W.
```

```

movwf  PORTA      ; Ensure PORTA is zero before we enable it.
movwf  PORTB      ; Ensure PORTB is zero before we enable it.
bsf    STATUS,RP0 ; Select Bank 1
    clrw          ; Mask for all bits as outputs.
    movwf  TRISB  ; Set TRISB register.
    bcf    STATUS,RP0 ; Reselect Bank 0.
initialise  clrw          ; Initial state.
    movwf  state  ; Set it.
loop        call    getmask ; Convert state to bitmask.
    movwf  PORTB  ; Write it to port.
    incf   state,W ; Increment state in to W.
    andlw  0x04   ; Wrap it around.
    movwf  state  ; Put it back in to memory.
    call   wait   ; Wait :-)
    goto   loop   ; And loop :-)

```

; Function to return bitmask for output port for current state.  
 ; The top nibble contains the bits for one set of lights and the  
 ; lower nibble the bits for the other set. Bit 1 is red, 2 is amber  
 ; and bit three is green. Bit four is not used.

```

getmask    movf    state,W      ; Get state in to W.
    addwf  PCL,F      ; Add offset in W to PCL to calc. goto.
    retlw  0x41       ; state==0 is Green and Red.
    retlw  0x23       ; state==1 is Amber and Red/Amber
    retlw  0x14       ; state==3 is Red    and Green
    retlw  0x32       ; state==4 is Red/Amber and Amber.
    ; Function using two loops to achieve a delay.

```

```

wait        movlw   5
    movwf  l1
w1           call    wait2
    decfsz l1
    goto   w1
    return
wait2       clrf    l2
w2           decfsz  l2
    goto   w2
    return
END

```

注意：上面的程序故意设置了一些错误的代码，以便后面调试使用。

## 2. 链接编译源文件

编辑好源程序后，进行源程序链接编译工作，整个过程如下：

1. 点击 SOURCE MENU 菜单下 ADD/REMOVE SOURCE FILE，在代码生成工具下拉菜单选择 MPASM，新建一个汇编程序，导入刚才编好的 TL.ASM 汇编文件。
2. 单击 SOURCE 菜单下 BUILD ALL 命令，如果程序没有错误，MPASM 会生成 HEX 格式的文件。
3. 编辑处理器属性中的 PROGRAM 属性，将生成的 HEX 文件添加到该对话框中。

\*这样我们就完成了程序的编辑编译以及和处理器模型的连接。

## 3. 调试程序

### a. Simulating the Circuit (仿真电路)

点击仿真盘框中的运行按钮。电路进入仿真状态，观察电路运行效果，你会发现开始一个交通灯是绿色而一个交通灯为红色，此后交通灯再也不会改变显示状态，这说明是程序存在 BUG，需要经过调试改正。

### b. Debugging Mode(调试模式)

按 CTRL+F12 或点击暂停或单步使电路从仿真状态切换到调试状态。在默认设置下系统会弹出两个窗口，一个是源程序调试窗口，另一个是寄存器窗口。另外一些调试窗口可以通过 DEBUG 菜单选出显示；其中使用者可以直接在 WATCH WINDOW 中添加自己比较关心的变量进行实时监测。

程序执行到某处，在该行程序的最多边会有一个红色的箭头出现，这行程序呈处于高亮状态。

### c. Setting a Breakpoint (断点设置)

观察程序可以发现它以一个重复周期循环。因此我们首先在循环的起点设置一个断点是个好主意。用鼠标使该行（地址 000E）高亮，然后按 F9 就可以设置断点。然后按 F12 使程序运行。你就可以从 Status Bar 看到执行到数字断点的消息以及程序计数器（PC）地址。这和我们设置的第一个断点的地址是对应的。

在 Debug 菜单下有一系列的调试键，但是多数时候我们用 F11 来单步运行程序。现在单击 F11 并注意左边的红色箭头下移到下一条指令。我们所做的是运行到“clrw”指令并停下。通过观察寄存器窗口的 W 寄存器并注意它的值被清零，你可以校验指令的运行。

现在我们需要做的，是决定我们希望下一条指令做什么，并测试实际上做了没有。例如，下一条指令将吧“w”寄存器的内容传送到端口 A，端口 A 应该被清零。运行这个指令并检查寄存器窗口可知实际就是这样执行的。继续这个线索直至到达我们设置的第二个断点，你可以注意到两个端口都被清零为输出（如 TRISB 寄存器所命令的那样），状态变量正确地设为 0。

由于这是个函数调用，我们可以使用 Stepping Over 选项跳过函数（通过单击 F10 键），但是为了完整我们单步运行每条指令。在这里单击 F11 就跳到 getmask 函数的第一条指令。向前单步执行我们看到传送指令成功执行，我们落在正确的位置，为查表添加 0 偏移值。当我们返回主程序后，

我们得到期望的屏蔽值。继续单步执行把屏蔽值写到端口，我们可以在原理图上看到正确的结果。再次单步执行，在寄存器窗口 W 寄存器的值加 1，显然状态也成功地加 1。

单步执行，程序设计为增加到 3 后绕回 0。就像在 Watch Window 看到那样，这并没有按照应有的方式运行。为了在下一循环中正确地设置屏蔽值，状态在这里应该清楚地被加 1。

#### d. Finding the Bug (查找 Bug)

仔细地分析就暴露了问题的原因是和 4 相与而不是 3。我们希望的状态是 0, 1, 2, 3 任何这些数与 4 相与都是 0。这就是为什么我们运行仿真时交通灯没有变化。解决方法很简单，就是把错误指令改为和 3 相与而非 4。这意味着状态增加到 3 也就是 W 寄存器增加到 4 时状态返回到 0。另一种简单的解决办法是当“W”寄存器到 4 时把它清零。

通过这个 ProteusVSM 调试技术的小例子说明了基本的技巧，还有更多的功能可用。建议你参考“源代码调试”一节以获得更加详细的资料。

## 基于图表的仿真

### 介绍

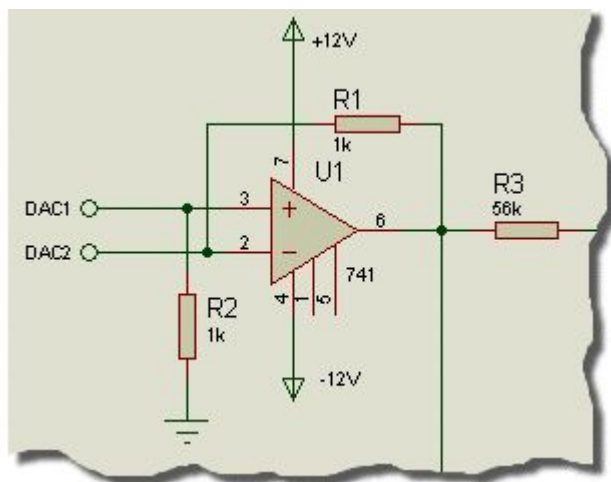
以一个放大电路的例子，展示怎样使用 PROTEUS VSM 做基于图形的仿真，基本过程：

1. 放置图表、探针及激励源。
2. 运行仿真。
3. 用图表显示仿真结果。
4. 对图形做测量分析。

### 开始

下图是我们准备仿真的基于 741 的音频放大电路，图中 741 工作在单 5V 供电的非正常状态。反馈电阻 R3 和 R4 决定了其增益为 10。R1、R2 和 C1 作为偏置器件，在反相端设置了虚地基准，并对信号去耦。

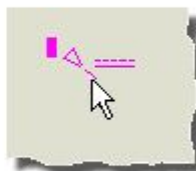
这里，我们先对这个电路做瞬态分析。瞬态分析是一种最常用的分析。在给顶激励信号的情况下，计算电路的时域响应，图表显示随时间变化的电压电流值。这种分析是非常有用的，它提供一个电路的大量的信息。其他的分析也可以以瞬态分析作为参照。



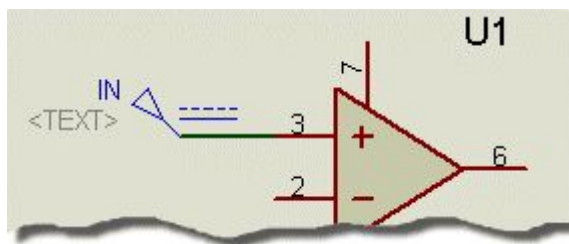
## 激励源

测试这个电路，需要合适的输入。PROTEUS 带的激励源可以提供所需要的信号。

点击 GENERATOR 按钮，对象选择器中会列出支持的激励源。对于这个电路，我们需要一个脉冲发生器。选择脉冲的类型，在编辑窗口中选择放置位置，点击左键确认，再进行连线。



激励源的操作和 ISIS 的元件操作是一样的，在选取放置时也可以进行编辑、移动、旋转、删除等操作；另外，激励源可以放置在已经存在的导线上，也可以放置好后再连线；在连线时激励源终端名称会自动命名。



最后，编辑激励源，得到想要的脉冲（此处设置高电平为 10mV, 脉冲宽度为 0.5S）。

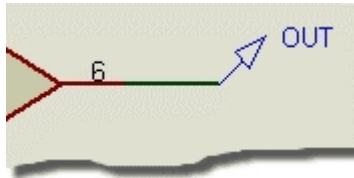
\*在 ISIS 手册中的 Generators and Probes 对各种激励源有一个详细的介绍。在原理图中激励源的数目



是有限制的。

## 探针

输入使用激励源，在需要监测的地方放置探针。在工具栏中选中探针类型（本例选用的是电压探针），放置在连线上，也可以放置好后再连线。



探针放置和 PROTEUS 中的其他元件是一样的，在选中探针后，可以进行编辑、移动、旋转等操作。

## 图表

图表在仿真中起了一个非常重要的作用：它不仅是一个显示媒介，还起着仿真中约束条件的作用。通过多个不同类型的图表（电压、数字、阻抗等）得到不同的测试图形，对电路进行不同侧面的分析。做瞬态分析需要一个模拟图表（取名模拟是为了与数字图表区分）。混合模式图表可以同时做模拟和数字的分析。

### 放置图表过程：

在工具栏中选择 GRAPH 按钮，对像选择器显示分析的图形列表。选择 ANALOGUE 类型，在编辑窗口中鼠标左键拖拽出一个合适大小的图表窗口。你也可以像往常那样选中图表、对它进行大小和位置调整。

在图表中添加激励及探针，有三种方式：

1. 选中探针/激励，将其拖曳进图表当中，系统会自动识别出添加的探针/激励。对于模拟分析，图线可能分别对应于左或右坐标，探针/激励要加在靠近坐标的一侧。
2. 使用 GRAPH 菜单中 ADD TRACE 命令，在对话框中探针选项中选择探针。（如果有多个图表，当前图表指的是选中的图表）。
3. 如果在原理图中已经有选中的探针，使用 ADD TRACE 命令时，系统会快速添加探针/激励。

在开始仿真前，有必要设定仿真时间，ISIS 会捕捉仿真时间内的波形。在此例中，电路输入是 10KHZ 的方波（周期为 100us），在图表编辑窗口中有 START、STOP 时间选项，将其中默认的 STOP 1S 改为 100us。

## 仿真

使用 GRAPH 菜单下的的 SIMULATE 命令（或使用快捷方式 SPACE）进行仿真，图表生成仿真

曲线。

仿真初始时间默认为 0，截止时间可以自己设置，在仿真中途可以按 ESC 退出。

仿真日志包含了最后一次仿真的信息。可以用 Graph—ViewLog 或按 CTRL+V 查看。仿真日志提供了一些图线上无法表示的一些有用信息。

## 测量

在图表未选中状态下，左键单击图表绿色标题栏，图表窗口最大化。图线的颜色与对应的标签颜色一致。要详细测量多条图线中的一条，可以将其标号拖曳到右边，此时，右坐标的刻度会更精细，图线也会被放大。

此例中测量两个量：

- 电压增益
- 增益对应的时间差

这些量都是通过坐标线来做的：

每一个图表都可以设置两条坐标线，第一条是绿色基准坐标线，用左键直接点击产生；第二条是红色参考坐标线，用 CTRL+左键显示。坐标线出现后，可以在图表下方读出即时量值及电压增益、时间差等。

## 使用电流探针

使用电流探针测量电路中流向 R4 的小电流。

电流探针和电压探针使用基本相似，但有一个小差别，就是电流探针有方向，放置探针时，其方向应该和实际电流方向呈水平状态，如果是垂直的话会产生错误。

选择电流探针按钮，调整好方向，置于流向 R4 的导线上，仿真运行后，我们可以看到反馈电流为  $9.1 \times 10^{-8} \text{A}$ ，是符合整个运放电路设计要求的。

## 频率分析

除了瞬态分析，应用在模拟电路场合还有其他好几种图表。它们的使用方法一样。频率分析就是其中一种，频率分析的作用是分析电路在不同频率工作状态下电路的运行情况。在 PROTEUS 频率分析图表中，X 轴表示频率，两个 Y 轴表示幅值和相位。

### 频率分析过程

#### 1. 放置频率分析图表

- 在工具箱中选择 SIMULATION GRAPH 按钮，在对象选择器中列出仿真分析图表。
- 选择 FREQUENCY 仿真图形。
- 在编辑窗口，按住左键，拖拽出图表。

#### 2. 添加探针

由于频率分析图表有两个 Y 轴，左边表示频率，右边为相位，所以选择电压探针后，得

在左右两边都添加，才能同时显示出幅相特性曲线。

在 PROTEUS 中幅相参考值的设定是通过 REFERENCE GENERATOR 来进行。一个参考激励源是 0dB（1 伏）、0 度。任何存在的激励源都可以指定为参考激励源，指定以后，电路中其他的激励源在频率分析过程中可以忽略不计。在这个电路当中，设定 IN 激励为参考，将其拖入图表当中。

3. 对图表属性进行编辑，在此例中，图表默认设置可以满足要求，无需做任何更改。
4. 按空格键运行仿真，观察得到的曲线，分析得到可用的频率范围是 50HZ 到 20KHZ。

## 扫描变量分析

在 ISIS 中可以进行 DC 扫描和 AC 扫描。

DC 扫描可以观察电路元器件参数值在使用者定义范围内发生变化时对应电路工作状态（电压或电流）的影响。

AC 扫描显示的是元件参数发生变化时对电路频率特性的影响。

由于两种扫描分析方法相似，这里我们只进行 DC Sweep。输入偏置电阻 R1 和 R2 影响着流入 U1 的微小电流。DC Sweep 用来分析它们的值如何影响偏置点。

在编辑窗空白处画出 DC Sweep 图表。将 U1(POS IP)探针选中并拖入图表。编辑图表，设置扫描变量名、起始值、终止值以及扫描步长。这里我们设置电阻值从 100k 到 5M。

R1 和 R2 的阻值也要从原来 470k 修改为 X，注意到图表中扫描变量也是 X。

这样就可以运行仿真进行扫描分析了。同样地可以进行 AC Sweep 分析，比如分析在 50Hz 上分析这些电阻值变化对低频特性的影响。

## 噪声分析

电阻或半导体元器件会产生温度噪声。ISIS 系统提供噪声分析，对每个器件产生的噪声计算出平方和，结果相对于噪声带宽呈现在图线上。

**噪声分析的一些特性：**

- 仿真时间和电压探针数目存在比例关系。
- 电流探针被忽略，不进行噪声分析。
- 在仿真日志中会产生大量的仿真信息。
- PROSPICE 同时计算输入和输出噪声。对于前者，需要通过添加参考激励来建立一个参考量。这样就能得出每个输出探针对应的等效输入噪声。

在 741 这个电路中，我们先把 R1、R2 阻值还原为 470 千欧。添加噪声图表到原理图中，添加参考激励源，再添加电压探针（这里我们只关注输出噪声，所以只添加输出探针）。我们需要设置输入参考为输入激励 IN。在 Edit Noise Graph 对话框中，显示的单位是 dB。如果使用这个选项，那么 0dB 等价于 1 伏。

运行仿真。在最大化的图表中可以看到噪声分析的结果非常小（本例中是 pV 级）。在生成的仿真日志当中，除了电路总噪声外，还有每个独立单元产生噪声的列表，事实上多数单元都位于运放内部。在 Edit Noise Graph 对话框中选择 Log Spectral Contributions 选项，可以获得更多的日志数据，

表示每个器件在每个频率点产生的噪声。

# 交互式仿真

## 基本技能

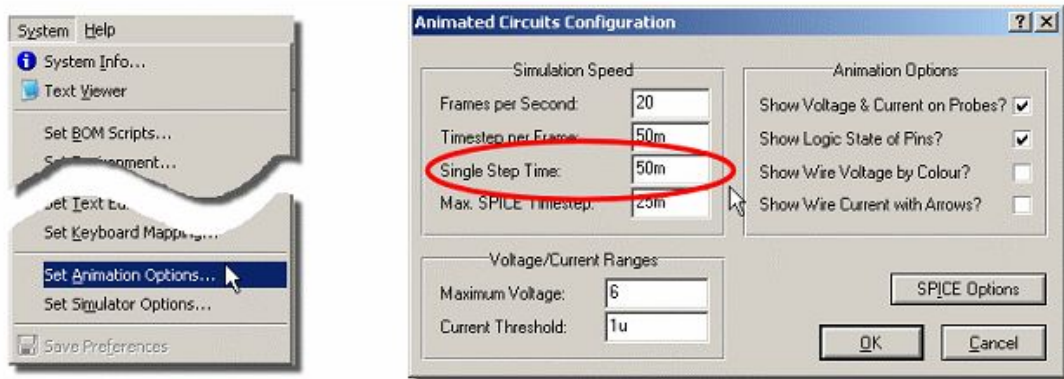
### 仿真控制按钮



交互式仿真通过上图中的一组按钮控制，它们位于编辑窗口的右下方。如果控制按钮没有显示，可以从 GRAPH 菜单中 CIRCUIT ANIMATION OPTION 进行设置。按钮包含四个：

1. PLAY：用来开始仿真。
2. STEP：用来步进仿真。步进时间由 Animated Circuit Configuration 对话框设置。
3. PAUSE：用来暂停仿真。（也可以通过 PC 键盘上的 PAUSE 按键暂停）
4. STOP：用来停止实时仿真。（也可以通过 PC 键盘上 SHIFT-BREAK 停止）

使用者可以设置仿真步进时间，通过相关设置可以更好的观察电路变化的细节。



注意一点，单步按钮并不是指程序的单步执行，而仅表示一段时间内电路变化。

在仿真过程当中，在状态栏中显示电路仿真时间及平均 CPU 负荷。如果 CPU 能力不足以进行实时仿真，CPU 负荷会显示 100%。如果没有错误，电路会按照设置的帧率自动进行仿真工作。

### 指示及执行

除了常规的电子元件，交互式仿真还需使用到动态器件。这些动态器件包含一组图形状态，两种方式：指示及执行。INDICATORS（指示）显示电路参数变化时的图形状态，而 ACTUATORS（执行）允许用户决定状态并修改一些电路的参数。

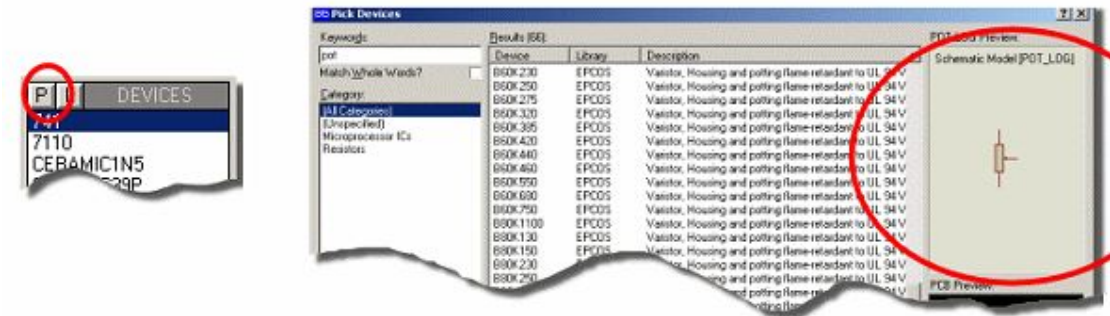
ACTUATORS 可以通过一个小红圈控制，通过鼠标点击来控制，也可以直接通过鼠标滚轮来确

定位置是否合适。

# 建立一个交互式仿真

在 ISIS 中绘制一个原理图，整个过程如下：

- 1. 在元件库中选择需要的元器件。



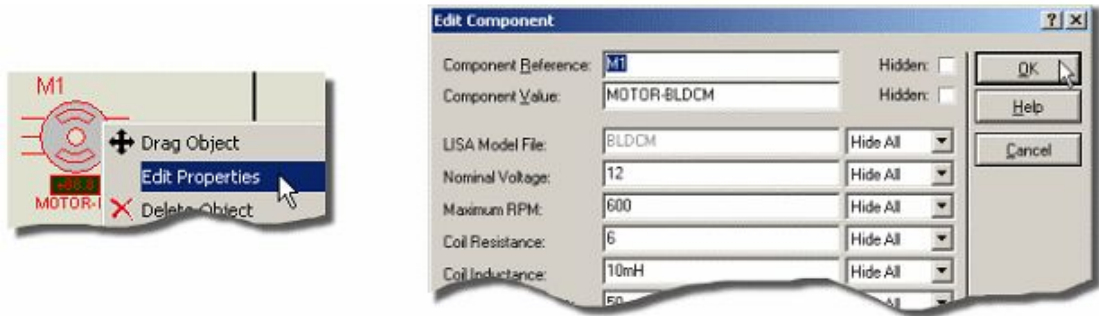
通过库浏览器选择有仿真功能的元器件

- 2. 在对象选择窗口中左键选中元件，在编辑窗口中放置元件。



在 ISIS 中放置元件

- 3. 选中元件，左键单击调出属性窗口，编辑属性。

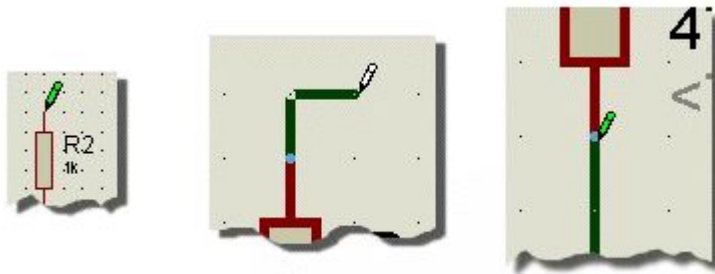


仿真前在 ISIS 中编辑元件属性

- 4. 处理器源代码调试：自带编译器编译的代码可以在 PROTEUS VSM 的 SOURCE 菜单下调出源码，其他可以通过装载第三方 IDE 生成的调试代码（COF，ELF，HEX 等）进行源码调试



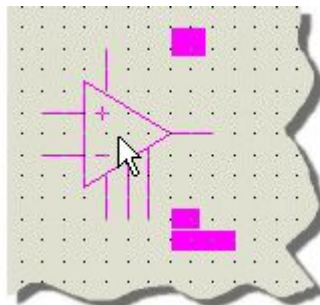
##### 5. 原理图连线



##### 6. 删除：右键单击元件，从下拉菜单中选择删除元件。



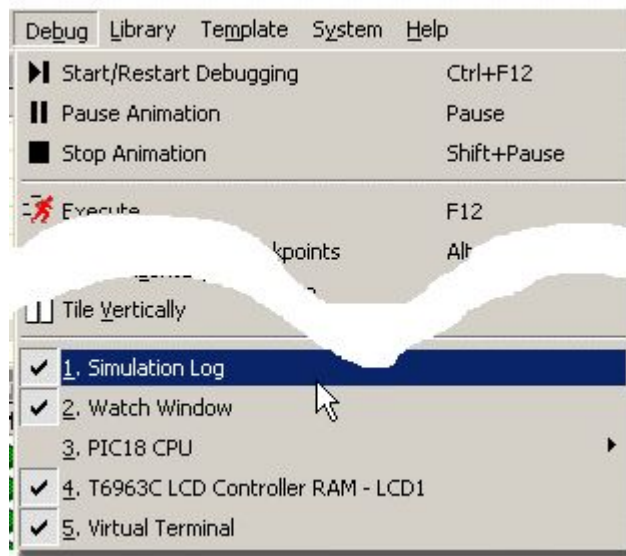
##### 6. 移动：选中元件，按住左键移动。



##### 8. 点击暂停键，仿真暂停，仿真时间归零。



##### 9. 在使用虚拟仪器、处理器 VSM 模型时，在仿真时 DEBUG 菜单下会多出若干命令，使用者可以添加自己关心的窗口进行观测。



## 测量

## 概况

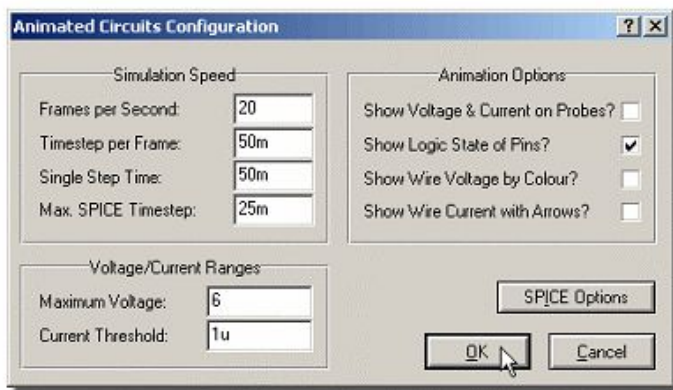
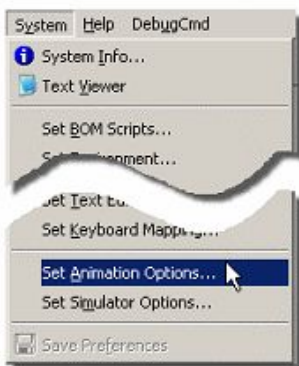
在交互式仿真中可以运用下列测量方法：

1. 仿真效果
2. 指示操作参数信息
3. 电压/电流探针
4. 虚拟仪器

## 动画效果

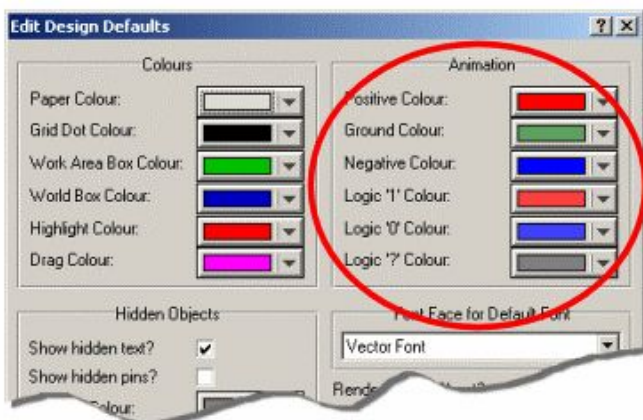
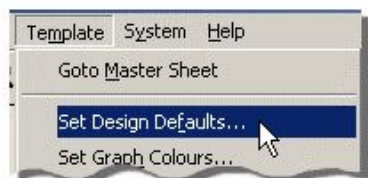
就像电路中使用的动态器件一样，动画效果也可以帮助使用者学习电路的动作。这些选项可以通过 SYSTEM 菜单下的 SET ANIMATION OPTION 来进行设置。



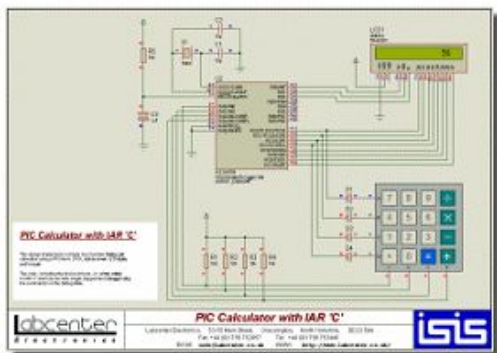


## 引脚逻辑状态

引脚状态通过有颜色的方块确定，方块蓝色表示逻辑 0，红色表示逻辑 1，灰色表示浮空状态。颜色方案也可以通过 TEMPLATE 菜单下的 SET DESIGN DEFAULTS 命令设置。

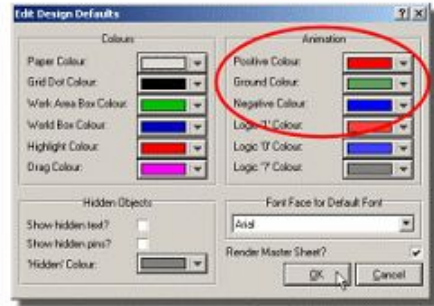
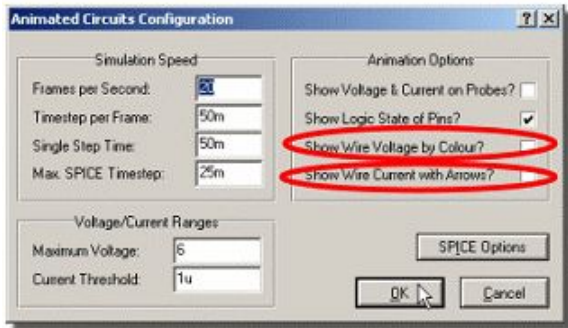


引脚状态显示在进行程序调试时非常有用，当使用断点、单步等各种调试手段时，使用者可以通过其观测各个口线的逻辑状态。



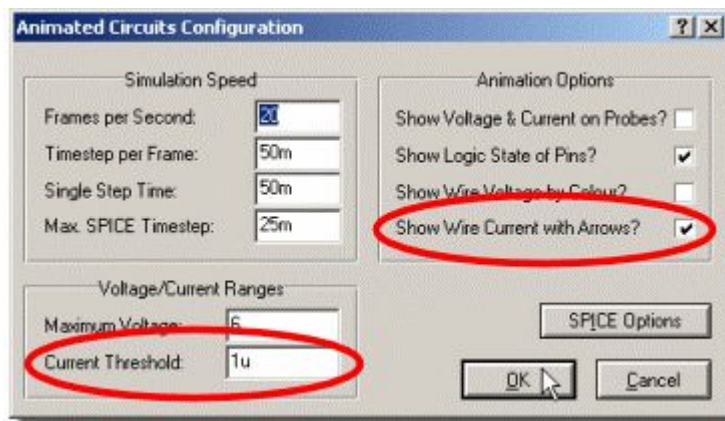
## 通过颜色显示线路电压

默认设置是：-6V 蓝色、0V 绿色、6V 红色。默认设置可以通过 SET ANIMATION 、SET DESIGN DEFAULTS 对话框进行相应更改。



## 显示电流方向

箭头显示实际电流流向，默认电流最小值是 1UA，可以通过 SET ANIMATION OPTION 对话框进行更改。



电压颜色和电流方向这些功能在电路分析上非常有用，可以帮助使用者动态的分析电路状态变化。

## 指示和显示参数信息

当仿真暂停时，使用者可以捕捉原理图上任一器件的电气参数信息。一般的元件都包含结点电压、引脚逻辑状态等信息，有些器件还有参考电压、功率信息。

### a. 示参数信息设置过程：

1. 仿真暂停，在 Animation Control Panel 上进行设置。

2. 在 ISIS 上选择 MULTIMETER ICON（万用表按钮）。

3. 左键点击元件显示信息。

#### **b. 直流工作点**

仿真暂停时，使用者可以观测直流工作点，这在分析耗散功率上是非常有益的；这个值不是平均值，它是瞬时电流乘以瞬时电压。

#### **c. 模拟还是数字？**

PROTEUS VSM 对电路网络表进行分析时，如果是数字信号，显示逻辑状态，如果是模拟信号，显示节点电压值。

#### **电压电流探针**

在 ISIS 中的电压探针主要应用于高级图形仿真，也可以和电流探针一起在交互式仿真中实时测量电压电流值。

探针相比虚拟电流电压表有两大优势：

可以在电路中随便移动位置，无需另外连线。

电压电流表在原理图中占用空间太大。

探针被放在纯数字网络上时，将显示逻辑状态而不是节点电压。

#### **探针放置方法**

选择探针类型——电压探针或电流探针。

在原理图中需要的探测点放置探针。

\*电流探针是有方向，放置时应注意箭头方向和实际电流方向平行。

## **虚拟仪器**

点击 ISIS 中万用表工具，对象选择窗口中会以列表的形式列出 PROTEUS 含有的虚拟仪器，详细的使用方法请查阅虚拟仪器使用章节。

## **动态步进控制**

### **概况**

两个参数决定交互式仿真是否实时， **ANIMATION FRAME RATE**（仿真速率）决定屏幕每秒中刷新的次数， **ANIMATION TIMESTEP** 决定每一帧的仿真量。在实时仿真中，这两个值呈倒数关系。

### **仿真帧率**

通常 20 帧/S 的采样速率已经能够提供一个平滑的动态效果，无须做任何的改变。除非当 CPU 的占有率溢出时，可以考虑降低采样帧率。

## 帧步进时间

这个量值可以改变仿真的快慢，它是仿真帧率的倒数。

当然，这个时间是一个期望值，是否能达到它还得看仿真使用的 PC 计算能力是否足够强。当 CPU 显示占有率达 100%时，说明其能力不足，实际帧步长将减慢。

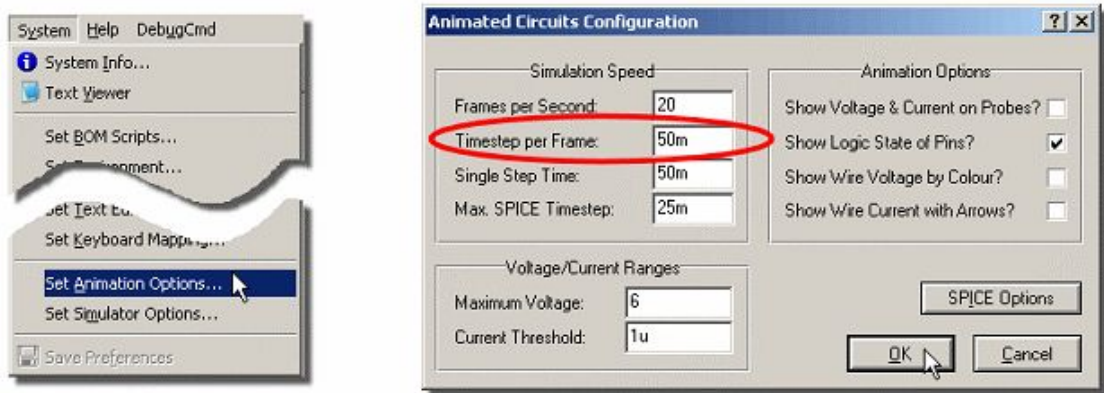
## 单步时间

这个量表示动画控制面板中步进的时间。

## 亮点和提示

## 电路时间尺度

交互式仿真视为实时行为，所以不必要用电路来产生时钟或频率信号，你只需调整 SYSTEM 菜单下 ANIMATED CIRCUITS COFIGURATION 对话框中 TIMESTEP PER FRAME 参数。



对于快速电路的仿真要注意以下几点：

1. 对于特定能力的 CPU，单位时间内仿真计算能力是一定的。PROTEUS VSM 设定了动态仿真帧率，降低仿真帧率带来的结果可能是不实时的，但会使整个仿真过程更加平滑。
2. 模拟元件模型仿真比数字仿真模型困难得多。一个高速 PC 可以实时仿真几 M 的数字事件，而模拟电路只能达到 10~20KHZ。

## 电压范围

使用有颜色的线表示电压高低通过 ANIMATED CIRCUITS CONFUIRATION 对话框进行设置。改变 Maximum Voltage value 可是设置显示电压范围。

## 接地

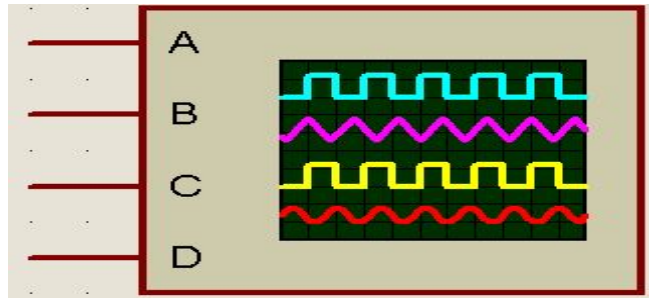
PROSPICE 对于一些动态电路无须定义地参考点，系统可以自动识别（通常被认为是电池的中点或者在双电源电路的中点）。但如果需要标示，可以从 ISIS 终端中选择地。

## 高阻抗点

对于任何未接地的元器件终端，为了保证 SPICE 仿真收敛性，会插入高阻值的电阻。这样，一些未连接的电路或有一部分未连的电路也可仿真（虽然有时得到的结果是不正确的）。

# 虚拟仪器

## 虚拟示波器

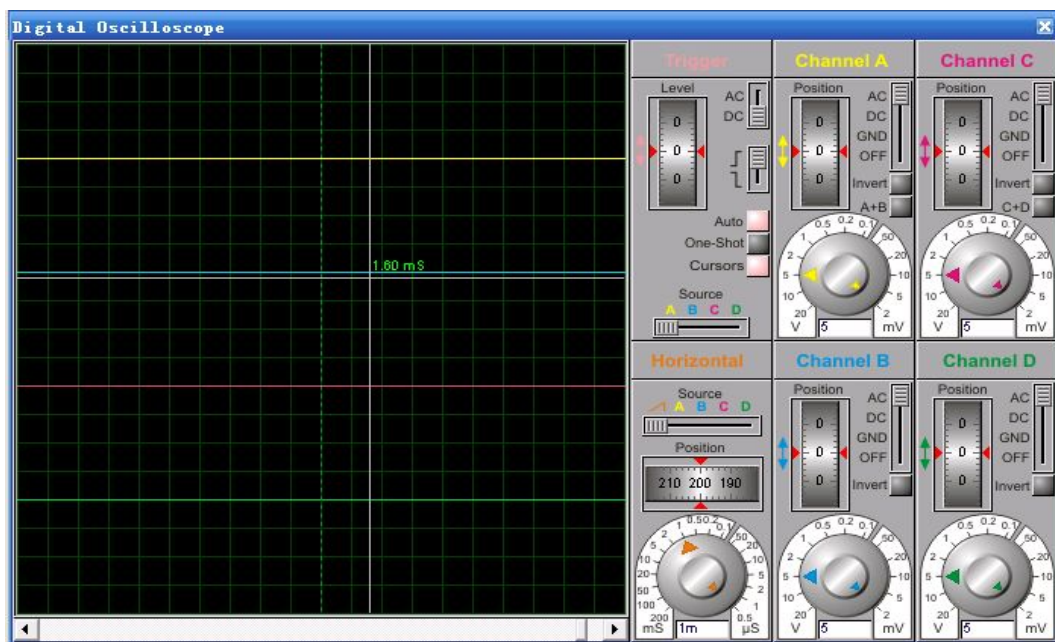


## 总述

VSM 示波器以 ProSPICE 版本为标准，模拟了基本的双通道（V7 以前）或四通道（V7 以后）单元并且有以下的特性：

- i、双通道/四通道，X—Y 模式
- ii、增益范围从 20V/每格到 2mV/每格
- iii、时基范围从 200ms/每格到 0.5us/每格
- iv、可以锁定任何通道的自动触发电压
- v、AC 或 DC 耦合输入





## 示波器的使用

单击工具箱中的 Virtual Instrument 按钮，则在对象选择器中列出所包含的选项。从对象选择器中选择 OSCILLOSCOPE，则在预览窗口显示出示波器的图标。

在编辑窗口中单击左键，则添加了示波器。将示波器输入端和欲采样的信号输出端连接上。

在仿真控制面板上按下“开始”按钮，进行交互式仿真。此时，出现示波器窗口  
如果显示双通道，选择 Dual 模式。

设置时基刻度盘到合适的值。此时，需要考虑电路中波形的频率，通过频率的倒数转换出周期。如果显示带直流偏移量的单通道，则选择 AC 模式。

调整 Y-gain 盘和 Y-pos 盘获得合适的波形大小和位置。当波形是具有直流电压偏移量的交流信号时，应该在被测节点和示波器之间加一电容。原因是 Y-pos 只能补偿一定数量的直流电压。

旋转触发按钮直到显示屏能捕捉到待测的输入波形。它锁定上升沿（拨上）或下降沿（拨下）

## 使用模式

有 3 种模式

- 单踪，即 Dual 和 X-Y 灯均不亮。在这一模式下。Ch1 和 Ch2 的 LED 显示灯表示在示波器中显示相应通道的信号
- 双踪，即 Dual 的 LED 灯亮。在这一模式下，Ch1 和 Ch2 的 LED 表示相应通道的信号被用作触发。
- X-Y 模式，即 X-Y 灯亮。在这一模式下。将分别以 Ch1 和 Ch2 通道数据作为 X 轴及 Y 轴的数据显示曲线

按动 Dual 模式与 X-Y 模式旁的按钮，可循环设置 3 种显示模式。

## 示波器的触发

虚拟示波器具有自动触发功能。这一功能使得输入波形可以与时基同步。Ch1 和 Ch2 的 LED 指示灯表示相应通道的信号被用于触发。连续旋转 Trigger 拨盘，可设置产生触发的电平和边沿。当拨盘指针指向上方时，锁定为上升沿触发；当拨盘指针指向下方时，锁定为下降沿触发，如果在多于一个时基周期内没有触发产生，则时基将自由运行

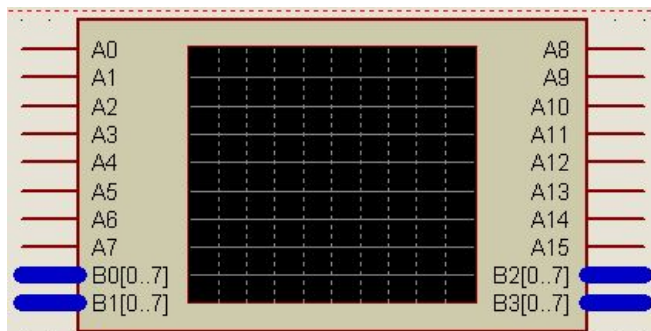
## 输入耦合

每一通道既可采用直接耦合方式，也可通过仿真电容采用交流耦合方式。其中，交流耦合方式的测量适用于带有较高直流偏压的交流小信号。

将输入端临时接地进行校准，这对于测量非常有用。

## 逻辑分析仪

### 概况



逻辑分析仪连续地将输入的数字信息记录到一个大的捕获缓冲区。这是一个采样的过程，因此有可调节的分辨率来定义可以被采样的最短脉冲。触发部件监测输入的数据并导致数据捕捉行动在触发条件已经生效后停止一定的时间；捕捉由 arming 按钮开始。最终显示的内容是在触发时间前后的捕捉器所捕捉到的。因为捕捉缓冲器非常大（可存放 10000 个采样数据），因此提供了放大/缩小和全局的显示。最后，可移动的测量标尺可以精确地测量脉宽。

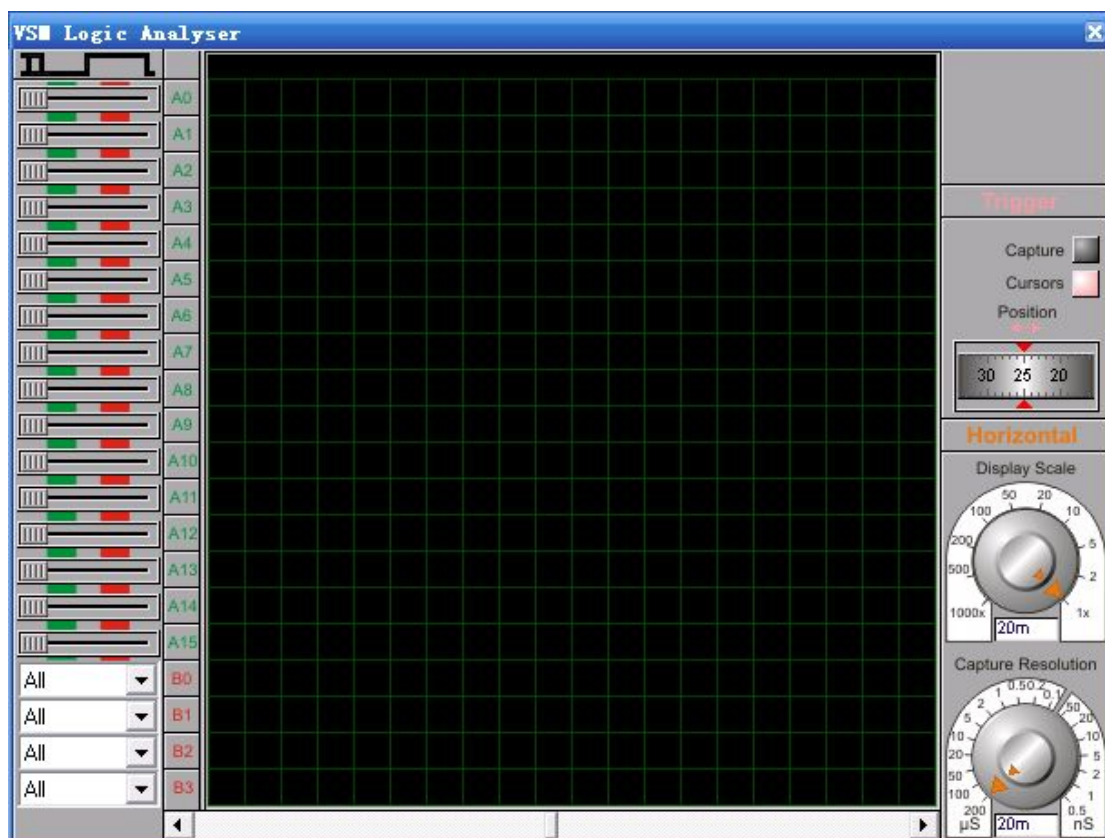
VSM 逻辑分析仪提供的特性有：

- 8\*1 位 Traces 和 2\*8 位 Bus Traces。
- 10000\*24 位捕获缓冲器
- 捕获分辨率从 0.5ns/采样点~200us/采样点，相应的捕捉时间是为 5ms~2ms.
- 显示的缩放范围为 1 个采样点/格~1000 个采样点/格
- 位输入信号的逻辑电平和/或边沿与总线值进行“与”操作后触发逻辑分析仪



- 触发位置在捕获缓冲器的 0%,25%,50%,75%和 100%处
- 提供两个坐标用于精确测量时间

## 使用逻辑分析仪



捕获和显示数字数据：

在 ISIS 中选中药表按钮并在对象选择器中选 LOGIC ANALYSER。放置在原理图部分上并连接到需要测试的地方。

按仿真控制盘的 PLAY 按钮启动交互式仿真。逻辑分析仪界面将会出现。

根据实际需要在分辨率刻度盘设置合适的分辨率。分辨率表示了能记录的最小的脉冲宽度。分辨率越高，捕获数据的时间间隔就越短。

在仪器的左边找到复选框并设置以满足要求的触发条件。例如，如果当连接到通道 1 的信号为高且连接到通道 3 的信号是上升沿信号时，想要驱动仪器，则需要设置第一位为高，第三位为“low-High”。

根据实际电路的需要，确定是否查看触发发生前后的主要数据，并且点击百分数旁的 LED 灯选择需要的触发位置

当设置完成后，点亮 ARMED 灯左边的按钮同时 TRIGGER 会熄灭。逻辑分析仪便开始连续捕捉输入的数据同时监测用于触发的输入信号。当触发发生时，TRIGGER 灯会亮。数据捕捉将一直进行，直至触发位置之后的捕捉缓冲器满为止。此时，ARMED 灯会熄灭并且将显示被捕捉的数据。

## 缩放和全局显示

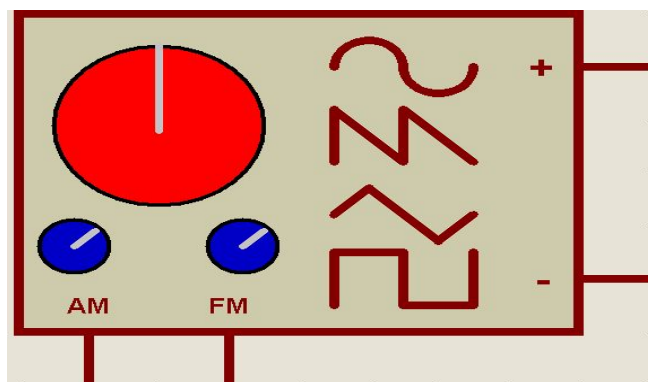
因为捕捉缓冲器可以捕捉到 10000 个采样点，但是显示屏仅能显示 250 个像素宽，因此需要在捕捉缓冲器进行缩放和全局操作。ZOOM 拨盘决定每格采样点的数量，同时滚动条可以实现左右移动。

注意在 ZOOM 拨盘设置下的每格以 s 为单位显示的是当前时间，而不是拨盘设置的实际值。每格的实际时间为缩放（ZOOM）设置值与分辨率（Resolution）设置值的乘积。

## 测量

仪器提供了两个可调标尺用于精确地测量。利用相应颜色的拨盘可以调整两标尺的位置。每一拨盘下面所显示的是相对于触发时间的 Marker 发生时间，而 Delta A-B 显示的是两个标尺的时间差。

## 虚拟信号发生器



## 总述

VSM 信号发生器模拟了一个简单的音频函数发生器，它具有以下特点：

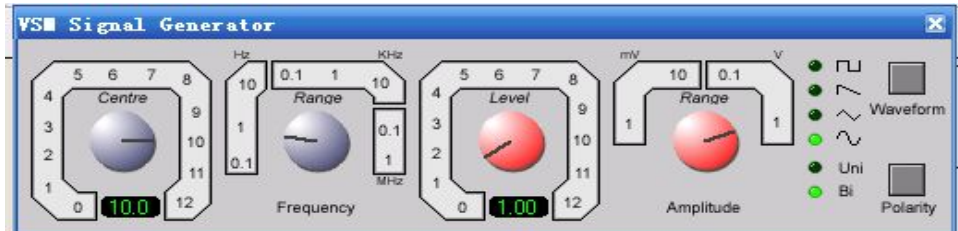
可输出方波，锯齿波，三角波和正弦波。

信号输出分 8 个波段，范围是 0~12MHZ

信号输出幅值分 4 个波段，范围是 0~12V

具有调频输入和调幅输入功能

## 使用信号发生器



在 ISIS 中选中仪表按钮并在对象选择器中选中 SIGNAL GENERATOR。放置在原理图部分上，然后引其输出端至电路。通常情况下（例如：电路需要的输入源为一个平衡的输入源的情况）需要信号发生器的“-”终端与接地终端相连接。利用接地终端很容易做到这一点。在不使用幅值与频率的调制输入时，可以不连接 AM、FM 这两个输入端口。

按仿真控制盘的 PLAY 按钮启动交互式仿真。信号发生器界面将会出现。

设置频率拨盘，以满足电路设计的需要。当 Center 指针等于 1 时，表示信号的频率为 1。频率单位以指针对应的标识为准。

设置幅值拨盘，以满足电路设计的需要。当 Level 指针等于 1 时，表示信号的幅度为 1。幅度单位以指针对应的标识为准。幅值为输出电平的值。

单击 Waveform 拨盘，代表相应波形类型的 LED 灯将会点亮，输出合适的信号。

## 使用 AM、FM 调制输入

信号发生器支持在幅度调制和频率调制的后的输出。AM 和 FM 有以下特性：

调制输入的增益由频率和幅度的 Range 拨盘分别按照 HZ/V 和 V/V 进行设置

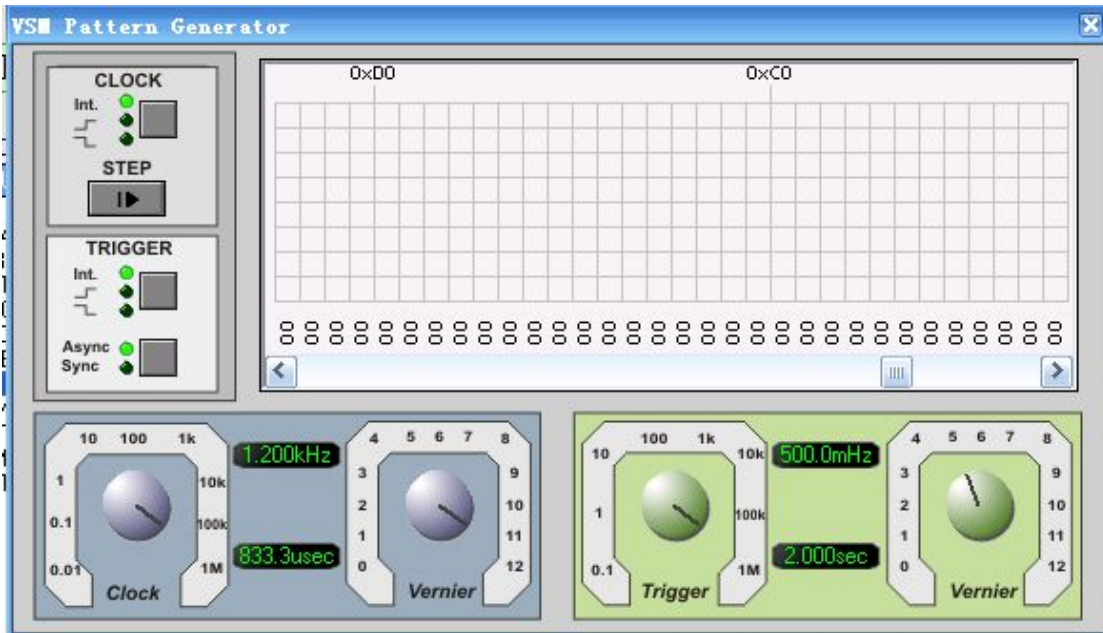
调制输入的电压范围是-12V~+12V.

调制输入有无限大的输入阻抗

调制输入电压值加上相应的游标值，然后与频率 Range 设置值相乘，决定了瞬时输出频率对应的幅度。

例如，如果频率的 Range 设为 1KHZ，频率的 Range 拨盘值设为 2，则 2V 的调频信号的输出频率为 4KHZ.

# 模式发生器



## 总述

VSM 模式发生器在数字方面等价于模拟信号发生器。在 Proteus 专业版仿真软件中都以此为模版发布。

VSM 模式发生器支持高达 1K 字节的模式信号，具有以下特性：

- 既可以在基于图表的仿真中使用，又可以在交互式仿真中使用
- 支持内部时钟和外部时钟模式以及触发模式
- 十六进制和十进制删格显示格式
- 直接输入具体值获得更为精确的值
- 加载和保存模式脚本文件
- 可以手动配置信号的周期
- 可以单步运行发生器
- 实时显示工具观测在栅格中的位置
- 可通过外部控制保持模式当前状态
- 栅格块编辑命令便于设置图案

## 使用模式发生器

在交互式方式下使用

在 ISIS 中选中仪表按钮并在对象选择器中选中 Pattern Generataor

按仿真控制盘的 PAUSE 按钮初始化交互式仿真。信号发生器窗口将会出现。

在栅格编辑窗口点击小方格切换逻辑状态，此即为输出信号的模式

确定使用内部时钟还是外部时钟，通过 CLOCK 按钮可以进行设置。

如果选择内部时钟，则可以通过 CLOCK 刻度盘来进行调整，以获得期望的时钟频率。

确定触发是内部触发还是外部触发，然后使用 TRIGGER 按钮设置相应的模式。如果使用了外部触发，则需要考虑是时钟同步触发还是时钟异步触发。

如果使用了内部触发，可以通过调整触发刻度盘来获得期望的触发频率。

按仿真控制盘的 PLAY 按钮启动交互式仿真，输出模式信号。

如果期望等到单时钟周期的模式信号，须在按仿真控制盘的 SUSPEND 按钮，后使用 STEP 按钮使栅格向左移动。

## 在基于图表的仿真下使用模式发生器

按通常的方法创建原理图

在原理图感兴趣的节点插入探针，然后把探针加进到图表。

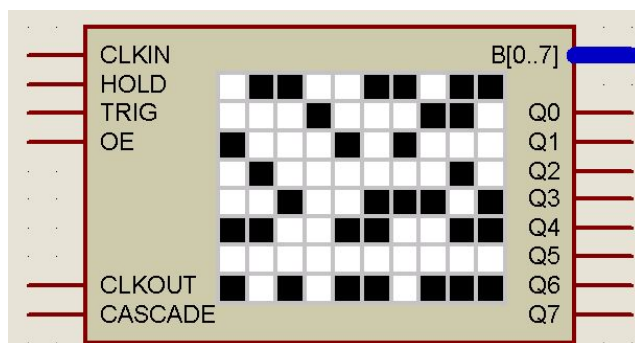
对准模式发生器先右键后左键调出器件属性对话框

设置触发和时钟参数

在模式发生器脚本区域加载期望的模式文件

退出模式发生器的属性编辑窗口，按空格键运行图表仿真

## 模式发生器的引脚说明



- ◆ **Data Output Pins:** 模式发生器可以以总线、引脚的方式输出模式信号
- ◆ **Clockout Pin:** 当使用内部时钟时，可以通过该引脚观测内部时钟脉冲。此性能可以在器件属性设置窗口中修改。该性能默认是关闭的，因为它会减弱系统的性能，尤其是在高频的系统中更为明显。
- ◆ **Cascade Pin:** 当模式信号的第一位正在受驱动时该引脚被驱动为高电平，并且一直保持高电平直到驱动下一位信号驱动（一个时钟周期以后）。这表明在开始仿真时的第一个时钟周期它呈现高电平，在接下来重新传输中的第一个时钟周期它又是高电平。

- ◆ **Trigger Pin:** 此引脚为输入引脚，用于接收外部触发脉冲。触发模式在下面将会详细讲述。
- ◆ **Clock-In Pin:** 此引脚为输入引脚，用于接收外部时钟信号，时钟信号的模式在下面将会详细讲述。
- ◆ **Hold Pin:** 当该引脚被驱动为高电平时，模式发生器将保持在暂停点，直至该引脚被释放。对于内部时钟或内部触发，时钟将从暂停点重新开始。例如，对于 1HZ 的内部时钟，如果发生器在 3.6s 处暂停，在 5.2S 处重新开始，则下一个下降时钟边沿将发生在 5.6S 处。
- ◆ **Output Enable Pin:** 使用高电平驱动该引脚。如果该引脚不为高，虽然模式发生器仍然工作在指定的模式，但并不驱动引脚输出模式信号。

## 时钟模式

### 1. 内部时钟

内部时钟是负沿脉冲。即每周期以低—高一低的形式。

内部时钟既可以在仿真仿真之前使用元器件属性对话框指定，也可以在仿真暂停期间使用时钟模式按钮进行指定。

Clockout 引脚使能后可以观测内部时钟信号。该性能默认是关闭的，因为它会减弱系统的性能，尤其是在高频的系统中更为明显。但是可以在器件属性设置窗口中启动此性能。

### 2. 外部时钟

模式发生器有 2 种外部时钟模式——负脉冲(LOW-HIGH-LOW)和正沿脉冲(HIGH-LOW-HIGH)将外部时钟脉冲连接到时钟输入引脚，并选择其中一种时钟模式

与内部时钟一样，外部时钟既可以在仿真仿真之前使用元器件属性对话框指定，也可以在仿真暂停期间使用时钟模式按钮进行指定。

## 触发模式

### 1. 内部触发

模式发生器的内部触发模式是按照指定的时间间隔触发的。如果时钟是内部时钟，则时钟脉冲在这一触发点复位。如图



例如，设定内部时钟脉冲是 1HZ，并且设定内部触发时刻是 3.75s。则 CASCADE 引脚在模式第一



位时为高，而在其他时间为低。

注意：在触发时间，内部时钟被异步复位。模式的首位被驱动至输出引脚（图中 CASCADE 被拉高）

2. 外部异步正脉冲触发

触发器由触发引脚的正边沿转换信号触发。当触发发生时，触发器立即动作，在下一个时钟边沿（即位时钟 1/2 处，与复位时间相同）发生由低到高转换。



例如，设定内部时钟脉冲是 1HZ，并且设定触发引脚在时刻 3.75s 处被拉高，则时钟立即在触发脚正边沿复位，模式的第一位驱动到引脚。

3. 外部同步时钟正脉冲触发

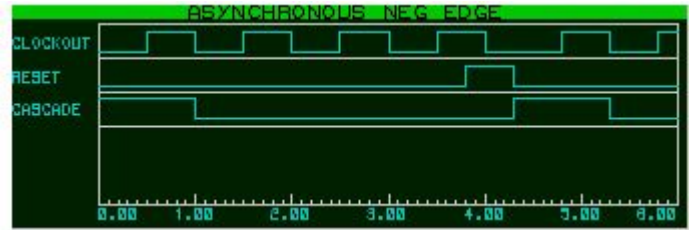
触发器由触发引脚的正边沿转换信号触发。触发立即使模式的第一位驱动到引脚。



例如，设定内部时钟脉冲是 1HZ。注意：时钟不受触发器的影响。在时钟下降沿时触发器立即动作，同时模式的第一位驱动到引脚。

4. 外部异步负脉冲触发器。

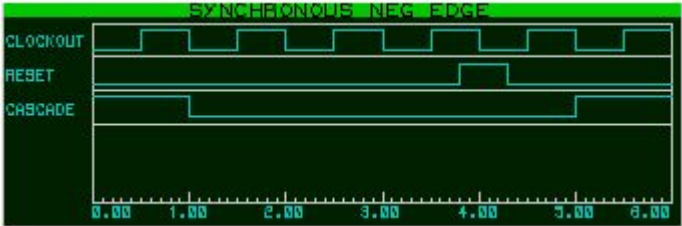
触发器由触发引脚的负边沿转换信号触发。当触发发生时，触发器立即动作，且模式第一位在输出引脚输出。



例如，设定内部时钟为 1HZ。从图中可以看到时钟在触发脉冲的负边沿复位，同时模式的第一位在此时被驱动。

### 5. 外部同步负脉冲触发器

触发器由触发引脚的负边沿转换信号触发。触发被锁定，并与下一个时钟的下降沿同步动作。

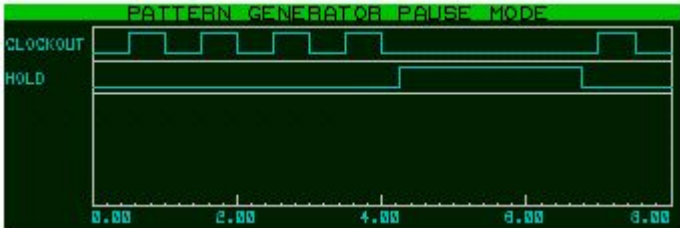


例如，设定内部时钟为 1HZ。注意，触发发生在触发脉冲的下降沿，模式直到时钟脉冲时钟脉冲的下降沿，与触发动作并发复位。

## 外部保持

### 保持模式发生器的当前状态：

想要在一段时间内保持状态，可以在该期间使保持引脚为高电平进入保持模式。如使用的是内部时钟，当在释放保持引脚的同时，模式发生器将同时重新启动。也就是说，在半个时钟周期里保持引脚应该为高电平。然后，当释放保持引脚时，下一位将要在以后的每半个时钟周期时驱动输出引脚。



当保持引脚为高时，内部时钟被暂停。当释放保持引脚时，时钟将在相对于一个时钟周期的暂停点重新启动。

## 附加功能

### 1. 加载和保存模式脚本

在栅格上单击右键，从菜单中选出相关的选项。这一方法对要在多个设计中使用特殊的模式非常有用。

脚本文件是纯文本，每个字节由逗号分隔。每个字节代表栅格上的一列，以分号起始的行被记作注释行，并且被解析器忽略。在默认状况下，字节格式为十六进制。当用户创建脚本时，输入值



可以是十进制、二进制或十六进制。

## 2. 为刻度盘设置指定值

用户可以通过双击合适的刻度值，指定位和触发频率的精确值。双击后，将出现浮动的编辑框。在默认状况下，输入值被认作频率，同时，也可以通过为输入值加上合适的后缀（如 sec、ms 等）来指定输入值的类型。此外，如果希望触发为时钟的精确乘倍乘，可以附加期望的倍乘后缀“bits”（如 5bits）。

按 Enter 键，或 Escape 键或者单击模式发生器件窗口的任何其他部位，用于确定输入。

## 3. 设定特定值到模式的栅格

左键在显示当前一栏值的文本上单击，可以指定栅格上该栏中任何一个值。出现的一个浮动的编辑框，用户可以在框中输入期望值。格式可以是十进制（如 135）、十六进制（如 0xA7）或二进制（如 0b01011101）。按 Enter 或 Escape 键，或单击模式发生器的任何其他部位，用于确认输入。为了方便，用户可以在想要编辑的栏，使用组合键 Ctrl+I 设置一栏，或者使用组合键 Ctrl+Shift+I 清除一栏。

## 4. 手动指定模式的周期长度

在期望模式结束的栏的栅格中，单击左键指定期在同样的区域，单击右键取消指定周期。

## 5. 单步执行模式

当仿真时间周期等与内部时钟或外部时钟指定的位时钟相等时，Step 按钮可以用于进一步仿真。仿真将一直进行到下个时钟周期结束并中止。

## 6. 切换栅格显示方式

栅格显示格式可以在十六进制和十进制中转换。此操作可以在栅格处点击右键，然后从菜单中选取设置的参数项或者通过快捷键 CTRL+X（十六进制的条件下）和 CTRL+D（十进制的条件下）进行转换。

## 7. 指定输出

调出模式发生器的属性编辑窗口。通过设置该窗口下方的属性可以指定模式信号在总线和引脚输出还是只以引脚或总线的方式输出。

## 8. 显示工具

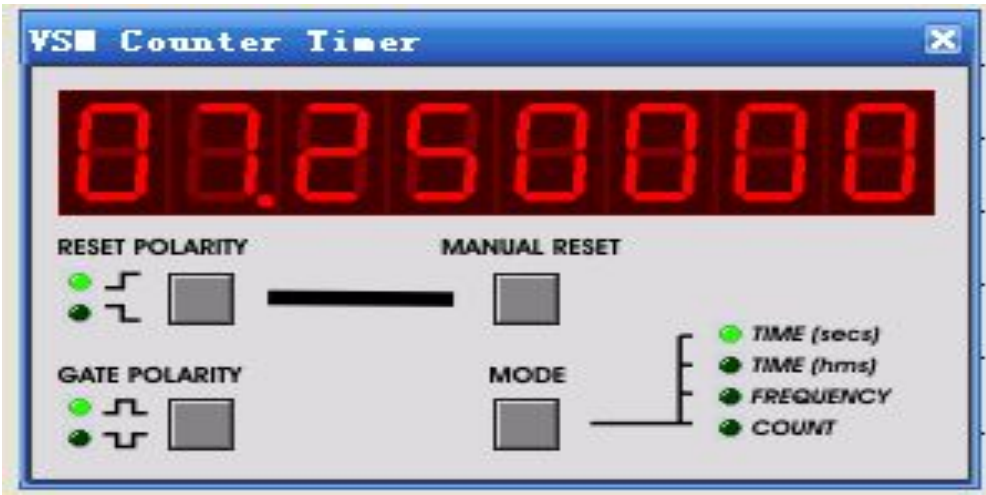
用户可以设定随光标移动显示当前行、当前列信息的工具。通过右击出现的文本，或组合键

CTRL+Q 控制。注意：在块设置或块清除时该工具不能使用。

### 9. 块编辑

用户可以通过块设置 (Block Set) 和块清除 (Block Clear) 命令快速设置栅格等到期望的模式。可以在右键出现的文本或组合键 (CTRL+S 是 Block set 命令的组合键, CTRL+C 是 Block clear 命令的组合键) 进行操作。注意：块编辑命令在 Tooltip Display 模式下不能使用。

## 定时/计数器



## 概述

VSM 计数/定时器是一个多用途的数字仪器。它可以用来测量时间间隔、信号频率和脉冲数。

计数/定时器有以下特点：

- ◆ 定时模式（显示秒），分辨率为 1 $\mu$ s
- ◆ 定时模式（显示小时、分、秒），分辨率为 1m
- ◆ 频率计模式，分辨率为 1HZ
- ◆ 计数模式—最大计数值到 99, 999, 999

时间、频率和脉冲值既可以在虚拟界面显示也可以在其弹出窗口界面显示。在仿真期间，用户可以定时/计数器单击左键或在 Debug 菜单调出。

## 使用定时器



用于测量时间间隔：

选中万用表按钮，在对象选择器中选 **COUNTER TIMER** 并把它放入原理图编辑窗口。

如果需要时钟使能信号则可以将使能控制信号连接到 **CE** 引脚。如果不需要时钟使能信号，可以将该引脚悬空。

如果需要定时器复位，可以将复位信号连接到 **RST** 引脚。如果不需要复位信号，可以将该引脚悬空。

将光标放在计数/定时器上，按组合键 **CTRL+E** 调出属性编辑窗口

选择要显示的时间模式、**CE** 和 **RST** 引脚的逻辑极性

启动交互式仿真

注意：

1. **RST** 引脚边沿触发方式非电平触发方式。如果使定时器保持为零，可以同时使用 **CE** 和 **RST** 引脚。
2. 计数/定时器提供了在手动按键复位的功能。该操作可以在仿真期间的任何时刻操作。可以利用该功能确定执行某一特定程序模块需要的时间。这种功能在嵌入式系统中非常有用。

## 使用频率计算模式

测量数字信号的频率：

1. 表按钮，在对象选择器中选 **COUNTER TIMER** 并把它放入原理图编辑窗口。将待测信号连接到 **CLK** 引脚。
2. 在频率模式下，不使用 **CE** 和 **RST** 引脚
3. 将光标放在计数/定时器上，然后按组合键 **CTRL+E** 调出属性编辑窗口。
4. 选中频率模式 (**Frequency Mode**) 关闭属性编辑窗口并启动交互式仿真。

注意：

- 在仿真每一秒中频率计计算信号上升的数量，因此要求输入信号稳定且在一秒内有效。同时，如果仿真不是在实时的速率下进行，频率计数器将在相对较长的时间内输出频率值。
- 计数器是个纯数字器件。在测量低电平模拟信号时，需要将待测信号通过 **ADC** 和逻辑开关后才输入计数器的 **CLK** 引脚。同时，因为模拟仿真比数字仿真仿真速率慢 1000 倍，所以计数器不是真正适合测量频率高于 10KHZ 的模拟振荡器电路。在这种情况下，可以利用虚拟示波器（或图表）测量信号的周期。

## 使用计数器

### 测量数字脉冲：

选中万用表按钮，在对象选择器中选中 COUNTER TIMER 并把它放入原理图编辑窗口，将待测信号连接到 CLK 引脚。

如果需要时钟使能信号则可以将使能控制信号连接到 CE 引脚。如果不需要时钟使能信号，可以将该引脚悬空。

如果需要定时器复位，可以将复位信号连接到 RST 引脚。如果不需要复位信号，可以将该引脚悬空。

将光标放在计数/定时器上，然后按组合键 CTRL+E 调出属性编辑窗口。

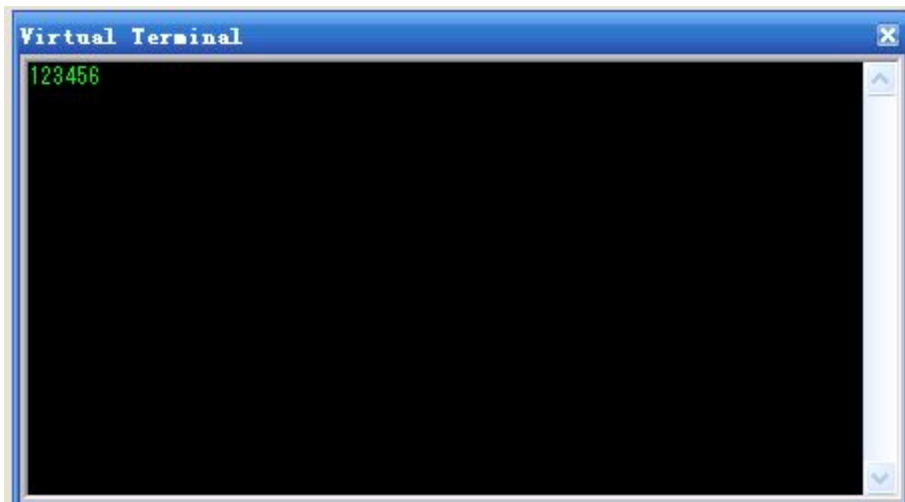
选择 Counter 模式、CE 和 RST 引脚的逻辑极性。

启动交互式仿真

注意：

- 当 CE 有效时，在 CLK 引脚的上升沿开始计数
- RST 引脚是边沿触发非电平方式触发。如果使定时器保持为零，可以同时使用 CE 和 RST 引脚。
- 计数/定时器提供了在手动复位的功能。该操作可以在仿真期间的任何时刻操作。

## 虚拟终端



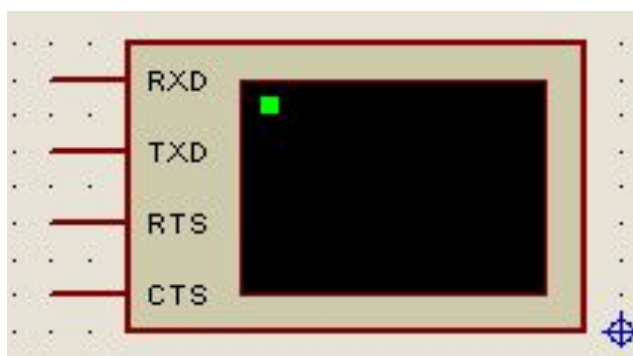
## 概述

VSM 虚拟终端允许用户通过 PC 的键盘和屏幕与仿真微处理器系统收发 RS232 异步串行数据。在显示用户编写程序产生的调试/跟踪信息时非常有用。

虚拟终端有以下特性：

- 全双工 以 ASCII 码方式显示接收的串行数据，同时以 ASCII 码方式传输键盘信号。
- 简单的两线传输。RXD 用于接收数据，TXD 用于传送数据。
- 简单的两线握手信号：RTS 表示发送准备好，CTS 表示清除发送数据。
- 波特率范围：300~57600
- 7 或 8 位的数据位
- 包含奇校验、偶校验和无校验
- 0、1 或 2 的停止位
- 除了硬件握手外，还提供了 XON/XOFF 软件握手
- 可对 RS/TS 和 RTS/CTS 的逻辑极性进行转换

## 使用虚拟终端



1. 中虚拟仪器按钮，在对象选择器中选中 **Virtual terminal** 并把它放入原理图编辑窗口。
2. 将 RX 和 TX 引脚分别接到测试电路的传送线和接收线。其中，RX 是输入，TX 是发送。
3. 如果待测系统使用硬件握手，则须将 RTS 和 CTS 引脚连接到数据流控制线上。RTS 是输出，发送信号，表明虚拟终端准备接受数据。CTS 是输入，此引脚在虚拟终端进行传送之前必须为高。
4. 调出终端的属性窗口，设置合适的波特率、字长、校验位、流控制和数据传输停止位。
5. 启动交互式仿真。终端立即显示接收到的数据；当传送数据时，在光标置于虚拟终端窗口的前提下，使用键盘输入数据。
6. 在仿真的过程中，在虚拟终端窗口单击右键，出现一系列可操作菜单。该菜单可以实现显示、复制和粘贴等操作。

注意：

虚拟终端支持 ASCII 控制代码 CR(0X08), BS(0X80)和 BEL(0X07)。其它代码包括 LF(0X0A)被忽略。

虚拟终端是纯数字模型因此在其引脚处不需要任何特别的电平要求。也就是说可以将其直接连到 CPU 或 UART 而不用通过 RS232 驱动器件例如具有逻辑电平转换功能的 MAX232。

RX 和 TX 引脚默认为高电平。因此空闲状态为高电平，起始位是低电平，停止位是高电平。数据位中，逻辑高代表“1”，逻辑低代表“0”。这与多数微控制器 UART 定义兼容，并且与诸如 6850 和 8250 的定义也兼容。当不是上述情况时（例如，因为将虚拟终端连接到 RS232 驱动元器件的输出端）需要将 RTS/CTS 极性反向。

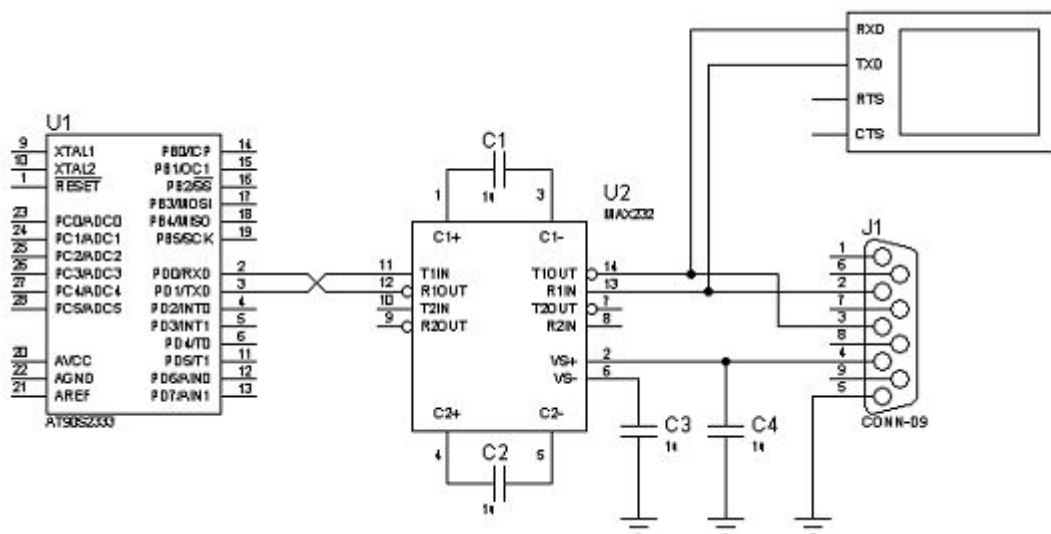
RTS 和 CTS 默认为高电平。如果想将两引脚连到反向控制线，则需要反向 RTS/CTS 的极性。

默认情况下，虚拟终端不显示用户键入的字符；也就是说，主系统将驱动输出终端显示用户键入的字符。如果想显示用户键入的字符，从快捷菜单中选择 Echo Typed Characters 选项。

用户可以用 TEXT 属性预先定义传输的数据。例如，定义好 TEXT=“Hello World”后，在电路启动时就传输数据，Hello World 就会被传送到电路中，除非 CTS 未激活。

## MAX232 模型

Proteus VSM 包含了 MAX232 这个 RS232 驱动器的模型，因此虚拟终端可以跟目标 CPU 进行连接。如下图所示：



然而要注意 MAX232 含有逻辑反相器，因此虚拟终端的 RX/TX polarity 须改为 inverted（反相）才能正确仿真。

需要注意的是 MAX232 也只是一个纯数字的器件，并没有模拟它内部电压转换器的工作。仿真这一特性可能带来巨大的仿真性能损耗。将任何模拟器件（电阻、电容、示波器等）连接到 TXOUT/RX 引脚也会造成同样的损耗。

# I2C 调试器



## 概述

I2C 协议调试器提供了监视 I2C 接口、与 I2C 接口交互的功能。该调试器有两个目的：一是用户查看沿 I2C 总线发送的数据，二是以主器件或从器件向总线发送数据。在编写 I2C 程序时既可以作为调试工具又可以作为开发和测试的辅助手段

## 器件引脚界面说明

I2C 调试器在 ISIS 中器件界面非常简单，如下图：



SCL 引脚：双向引脚，用于连接 I2C 总线的时钟线。

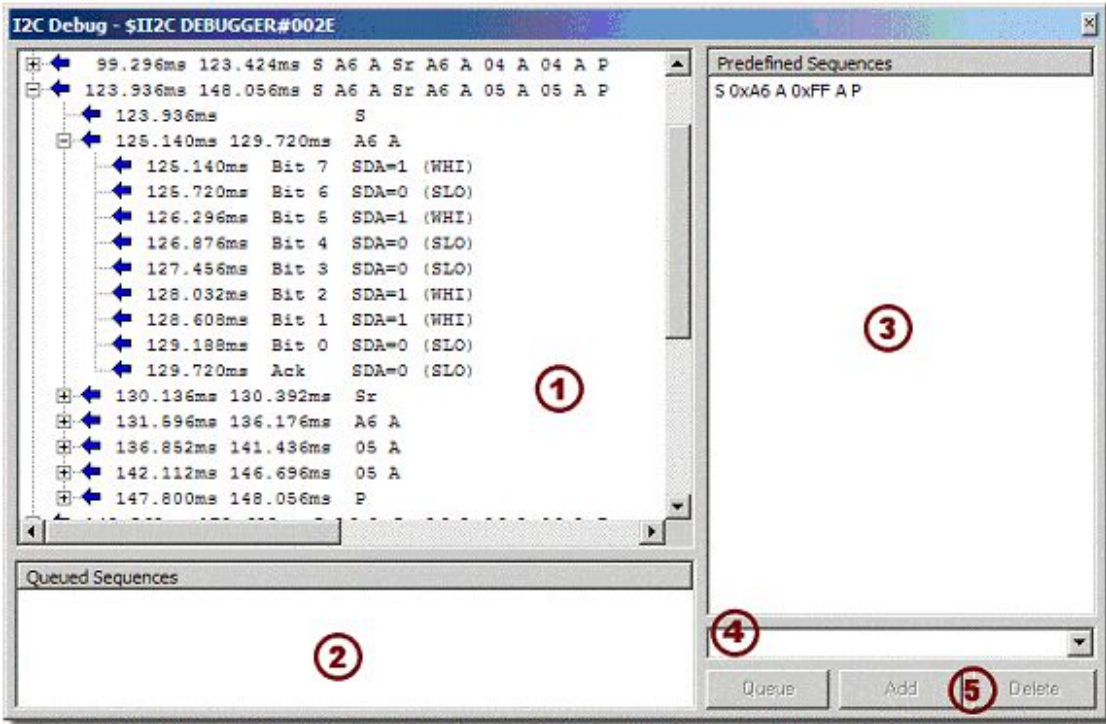
SDA 引脚：双向引脚，用于连接 I2C 总线的数据线。

TRIG 引脚：输入引脚。用于触发一系列连续的储存数据到输出队列

## I2C 调试窗口

当仿真暂停或运行时，I2C 调试器界面将会出现（如果界面没自动弹出，可以通过 Debug 菜单

调出)。界面可以分为几个区域。它不仅可以监视 I2C 总线上的数据还可以以主机或从机的身份仿真总线。



注意：当仿真的有很多 I2C 总线活动的电路时，I2C 调试器会导致很大的仿真消耗。

**a. 输入数据的显示**

主窗口区域显示在总线上所有活动记录。从上图可以知道，用户可以查看数据每一位的传输。

- 当以字节或更长数据为单位观察时，序列起止时间被记录。当以位为单位观察时，引脚的逻辑状态和位标号记录在时间旁。
- 当箭头上显示不可见时表明传输是双向的。如果有非法的序列，箭头上会出现问号标志。

**b. 预定义序列表**

预定义序列表在对话框的右上角。该表显示了在 SCL 引脚控制下的 SDA 引脚传送的预编制的一系列数据。用户可以在序列输入窗增加预编数据（利用 ADD 按钮完成向序列表添加数据）。

当仿真正在进行时，双击事先定义的序列会把该序列添加到缓冲器/队列表后用于总线上的传输。注意传送的时间决定于总线的特性，例如此时的总线是主机还是从机等。

**c. 缓冲/队列的序列表**

缓冲序列表位于窗口的左下角。用于显示在下一个有效时机 I2C 将要传送的一系列数据。



用户可以在序列输入窗增加预编数据。(利用 Queue 按钮完成向序列表添加数据序列)。

d. 序列输入窗口

该窗口（位于界面的右下角）允许用户输入序列，既可以放在预编程的序列窗或立即放在输出列队以便在下一个有效的时机传送到 SDA 引脚。支持以下语法规则：

语法	描述
‘0x’	表示十六进制值的前缀，例如 0xFE
‘\$’	表示十六进制值的前缀，例如 \$FE
‘h’	表示十六进制值的后缀，例如 FEh
‘%’	表示二进制值的前缀，例如 %1101b
‘b’	表示二进制值的后缀，例如 1101b
‘d’	表示十进制值的后缀，例如 47d

注意十六进制和二进制必须有限定的前缀或后缀。十进制的后缀是可忽略的。以上字符必须要用单引号或双引号括住。

控制按钮

控制按钮用于激励 I2C 总线。该调试器既可以作为主机又可以从机，此特性提供了一种强大的测试 I2C 执行情况的功能。

- ADD 按钮：**向预定义序列表输送在 Sequence Entry Box 定义的序列，为以后的操作使用。
- Delete 按钮：**删除在预定义序列表中的序列。
- Queue 按钮：**从 Sequence Entry Box 或 Pre-defined Sequence List 中排列已被选项。被排列的选项在会出现在 Buffered/Queued 案 Sequences List.

发送和接收数据

1. 发送数据

将 SDA 和 SCL 引脚连接到电路的相应引脚。

在按钮的上方的终端窗口输入要传输的数据。数据可以是单字，逗号或空格隔离的一系列字节，或用双引或单引号括起来的文本字符串。一个字可以是具体的数字或是具体的控制信号。

一旦完成了传输序列的输入，则可以直接排列数据准备发送或者将数据放到预定义序列表以便后面使用。点击 queue 按钮即可以立即将传输序列放到缓冲序列表中；点击 add 按钮即可以将传输序列放在预定义序列表里。

要想从预定义序列表中删除一个输入的数据，可以先在预定义序列表中选中要删除的数据然后按 delete。

2. 接收数据

将 SDA 和 SCL 引脚连接到电路的相应引脚。在 I2C 总线上的数据将以十六进制和特定的格式显示在调试窗口的左上方。

# 模型的特性

ADDRESS1	地址 1。如果使用此终端来仿真一个从元器件，则这一属性用于指定从元器件地址的第一个地址字节。主机使用最低有效位作为系统进行读操作或写操作的标志位，而在寻址时，这一位被忽略。如果属性设置框为默认值，或为空，此终端将不被认为是从元器件。
ADDRESS2	如果使用此终端来仿真一个从元器件并且期望 10 位的地址，则本属性用于指定从元器件的第二个地址字节。如果属性设置框为空，则假定地址为 7 位。
STOPONEMPTY	指示表明当输出缓冲器为空和一个字节需发送时，仿真是否应该暂停。
CLOCKFREQ	当处于主元器件时，用于控制 SCL 的时钟频率
SEQUENCE_FILE	用于指定存储预先定义的输出序列的文本文件名 如果此属性框为空，序列将被当做器件属性的一部分而保存
AUTOLOAD	
TIMEPREC	定义十进制的数量以显示时间
WRAPLENGTH	定义在新线创建之前在单一线上允许的项目的数量。

# 语法参考

以下内容详细介绍了调试器用到语法

语法	描述
S	表示开始条件
Sr	表示重新开始条件
P	表示停止条件
N	表示负应答条件
A	表示应答条件
L	表明仲裁已经丢失，转入到主机模式
*	表示接收到局部的数据
?	表示检测到无效的逻辑电平

# SPI 调试器

## 简述

SPI 调试器允许用户监测 SPI 接口并与 SPI 接口互动。该调试器不仅允许用户观测 SPI 传送的数据并且允许用户直接向总线发送数据。该终端可以工作在以下三种模式之一：

- ◆ 从模式：以从机的身份运行
- ◆ 主模式：以主机的身份运行
- ◆ 监测模式：只报告当前 SPI 总线传送的内容

## 原理图部件：

原理图部件如下图所示：



管脚说明：

**SCK 引脚：**该引脚与 SPI 总线的时钟线相连。它是双向引脚，当作为从机使用时是输入，当作为主机使用时是输出

**DIN 引脚：**数据接收引脚。接收来自 SPI 总线上的数据

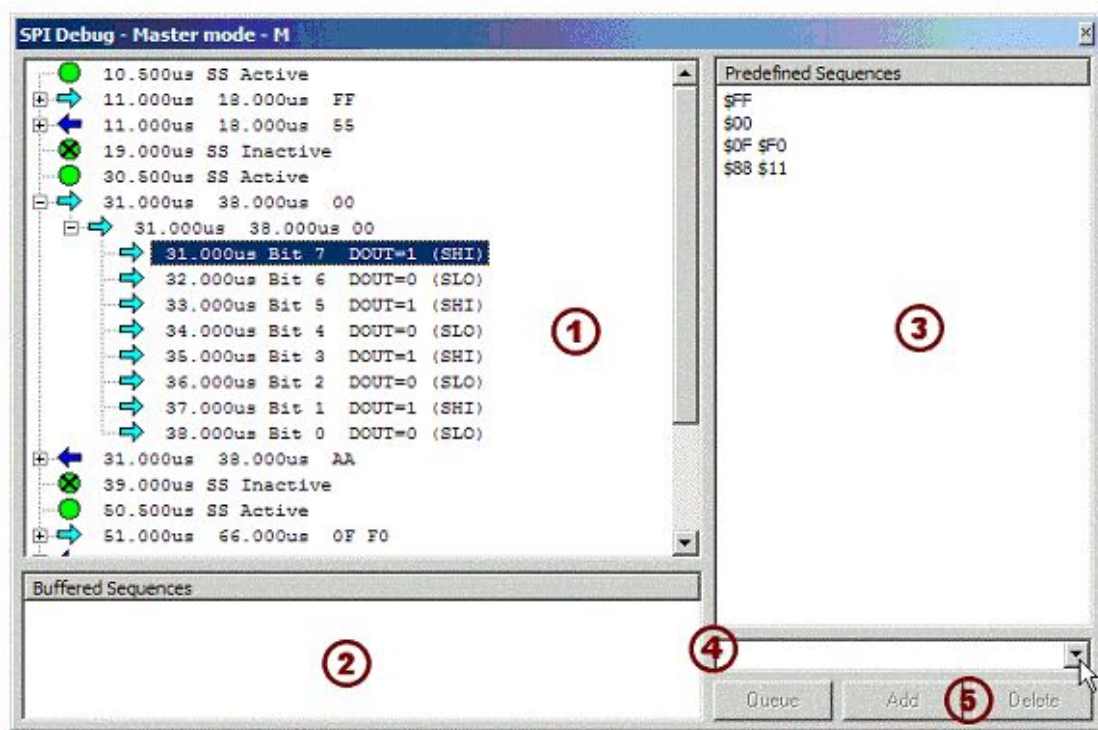
**DOUT 引脚：**数据输出引脚。向 SPI 总线发送数据

**SS 引脚：**选择从设备引脚，在从机模式时，该引脚必须有效以激活调试器。在主模式时，引脚激活的同时正在传送数据

**TRIG 引脚：**该引脚为输入引脚。用于将调试器存储的下一个序列输入到 SPI 输出序列

## SPI 调试窗口

当仿真处于暂停状态或全速运行时，调试窗口可用（如果该窗口不是自动弹出来，用户可以从 Debug 菜单中调出来）。该界面分为几大部分，不仅允许用户 SPI 的当前情况并且以主机或从机的模式仿真总线。



在主模式下的调试窗口

## 1. 输入数据的显示

主窗口（在左上角）显示了总线所有的运行情况。从上图可以看到，用户可以仔细观察单个位传输。

当以字节或更大单位查看时，序列传送的开始和结束时间都将给予记录。当以位单位查看时，引脚的逻辑电平和位指示器将在界面同时间一起显示

在上述界面中不可见时，箭头以双向的方式表示传输或接收数据（上述例子是接收的状态）。如果检测到一个无效的序列，箭头和旁边的问号一同出现。将此特点和以位观测数据的功能相结合，则在调试潜在的问题时将会发挥极大的作用。

## 2. 定义序列表

在右上角的是预定义序列表。它显示了一列可在总线上进行传输的预编程数据序列。用户可以利用在 Sequence Entry Box 中添加事先定义的序列，方法如下：（用 Add 按钮将序列递交到序列表）

当仿真正在进行时，在一预定义序列上双击左键（或先将预定义序列高亮选中后点击 Queue 控制按钮）将会把这一预定义序列添加到 Buffered/Queued Sequence 中，用于在总线上的传送。注意，无论是主机还是从机传输的时间取决于总线的活动并且很有可能不是瞬时的。

## 3. 缓冲/序列表

缓冲/序列表位于窗口的左下方。它显示了一队将在下一个有效的时机被 SPI 调试器传送的序列。

用户可以在 Sequence Entry Box 中增加一缓冲序列。用户用 Queue 按钮将序列递交到该序列表。

4. 序列输入框

该输入框（位于窗口的右下方附近）允许用户在可编程序列表中放置序列或立即将序列加到输出序列中，准备在下一有效时机发送到 DOUT 引脚。支持的语法如下表：

语法	说明
‘Ox’	前缀，表示十六进制的值（例如，OxFE）
‘\$’	前缀，表示十六进制的值（例如，\$FE）
‘h’	后缀，表示十六进制的值（例如，FEh）
‘%’	前缀，表示二进制的值（例如，%1101）
‘b’	后缀，表示二进制的值（例如，1101b）
‘d’	后缀，表示十进制的值（例如，47d）

注意十六进制和二进制的数值必须有一前缀或后缀，而十进制可以省略。字符串必须以双引号或单引号括住。

利用 Control 按钮去决定一个已给定的序列是加在预定义序列表还是加在缓冲/序列表。

5. 控制按钮

控制按钮允许用户为 SPI 总线提供激励。协议分析仪完全既可以做主机又可以做从机，因此为检测 SPI 执行情况提供了很好的方式。

A dd 按钮：使用此按钮将在序列输入框定义的序列递交到预定义序列表，以便以后使用。

Delete 按钮：使用此按钮删除预定义序列表的一个序列。

Queue 按钮：使用此按钮对直接在序列输入框或预定义序列表选中的项目进行排列。所排列的项目将会出现在缓冲/序列表中。

模型的特性

SPI 仿真器有很多个性化的特性.用户可以根据特定的需求进行相应的设置.所有特性都可以通过属性对话框进行设置(右键选中后点击左键弹出属性对话框)。具体个性属性说明如下表：

特性	说明
WORDLENGTH	字长，指定每一个传输数据的位数，可以指定从 1~16 任意一个数据
BITORDER	位顺序，指定每一个传输数据的位顺序，既可以先传输 MSB，也可以先传输 LSB
SAMPLEEDGE	采样边沿指定 DIN 引脚采样的边沿。当 SCK 从空闲到激活、或从活跃到空闲时进行采样。
IDLESTATE	SCK 空闲状态，指定 SCK 何时为空闲状态，或当 SCK 为高电平，或当低电平时为空闲。

STOPONEMPTY	输出缓冲器为空时停止。指定当输出缓冲器为空时是否停止仿真
SEQUENCE_FILE	序列文件。允许用户给用来保存预定义输出的序列的文件命名。如果该属性设为空，则序列将作为器件属性的一部分被保存。
MODE	主机或从机的模式选择控制
CLOCKFREQ	在主机模式下，用于控制 SCK 上的时钟频率
AUTOLOAD	控制是否在仿真进行前立即将预定义序列放置到序列表中
WRAPLENGTH	指定在建立一新线之前一个单线上允许的字数量。
TIMEPREC	某一时间显示十进制数值的数量
DATAINVERT	指示 DOUT 和 DIN 的数据是否被反向
LOOPBACK	当被使能时，任何接收到数据将会被排列以便发送

## 使用 SPI 调试器

### 1. 用 SPI 调试器接收数据

观测 SPI 总线上的数据要求将终端的 SCK 和 DIN 引脚连接电路中相应的引脚,然后设置终端模型的字长、位顺序、SCK 空闲状态和采样边缘的属性等。SPI 总线上的数据将以十六进制的格式显示在终端窗口的左上角。如果一个字长的接收数据位传输完毕（例如在接收整个字的数据之前 SS 无效），则显示显示的不在是十六进制而是星形符号。如果 DIN 上的接收数据位有因冲突不稳定，则显示的不在是十六进制而是问号。

### 2. 使用 SPI 调试器传输数据

使用 SPI 发送数据，用户首先需要将 SCK 和 DIN 引脚连接到电路的相应引脚上，然后设置终端模型的字长、位顺序、空闲状态和采样边沿。

在终端窗口的右下角输入需要传输的数据，就在按钮的上方。（见上面的序列输入框）输入的数据可以为单字、以逗号或空格键间隔开的一串字，也可以为用单引号或双引号括住的一串文本字符。数值可用十进制、十六进制或二进制表示。十六进制以一个“0x”或“\$”为前缀，或者以一个“h”为后缀表示。二进制以“%”为前缀或以“b”为后缀表示。十进制不需要前缀或后缀表示，但有时可以一个“d”为后缀。

一旦用户已经输入要传输的序列后，可以直接发送数据序列，也可以将数据序列保存在预定义序列表中以便以后使用。点击 **queue** 按钮可将传输序列立即保存在缓冲序列表中。点击 **add** 按钮可将传输序列保存在预定义序列表中。

如果需要将预定义序列表中的序列复制到缓冲序列表，首先确定序列输入框为空白，然后选择需要预定义的序列，最后点击 **queue** 键。

如果需要从预定义序列中移除数据，先选择需要移除的序列，然后点击 **delete** 键。

## 电压表和电流表

Protues VSM 提供了 AC 和 DC 的电压表、电流表。它们可以像其它元器件一样在在电路图连线。仿真开始后，它们通过自带的终端或以易读的数字格式显示电压或电流值。

仪表的 FSD 可以现实 3 位有效数字，最多可显示 2 位小数。显示电压范围可通过元器件编辑窗口的 Display Range 属性进行设置。

电压表模型含内阻属性，默认值是 100M，属性值可以通过属性编辑窗口进行设置。当内置电阻为空时，加载模型内阻选项无效。

交流电压表和交流电流表以用户可定义的时间常数显示有效值。

# 微处理器仿真

Proteus VSM 中的大多数微处理器模型会生成很多窗口。可以通过 Debug 菜单显示和隐藏这些窗口。这些窗口有以下四个主要的类型：

- 状态窗口—通常利用此窗口显示其寄存器的值。
- 寄存器窗口—对应于微处理器的结构中寄存器空间。寄存器设备（包括 RAMS 和 ROMS）同样会生成这些窗口。
- 源代码窗口—在原理图中每一个处理器都有对应的源代码窗口
- 变量窗口—假设装入程序支持可编程的变量显示，在原理图中每一个处理器都有对应的变量窗口

## 显示弹出窗口

- 按 CTRL+F12 启动仿真模式，或者当系统已经处于运行状态，点击仿真进程控制按钮 Pause。
- 按 ALT+D，将在 Debug 菜单显示所需窗口的序列号。

这些窗口类型只能在当仿真暂停显示。当仿真暂停时（手动或者因为断点）这些窗口将会重新出现。

仿真窗口都有右键菜单—如果在窗口上按右键，则出现一系列菜单。从中可以设置控制窗口中数据的显示/隐藏和格式等。

仿真窗口的位置和可见性将以 PWI 文件的格式以设计名称为文件名自动保存。该 PWI 文件同样包含了以前设置了的断点的位置以及观察窗口的内容。

## 基于 Proteus VSM 源代码调试

### 概述

PROTEUS VSM 支持基于汇编程序和编译器的源代码调试。调试下载器的设置包含在 LOADERS.DLL 中并且此类工具的数量正在以很快的速度增长。在我方网站“3<sup>rd</sup> Party Compilers”可以查询到最新信息。

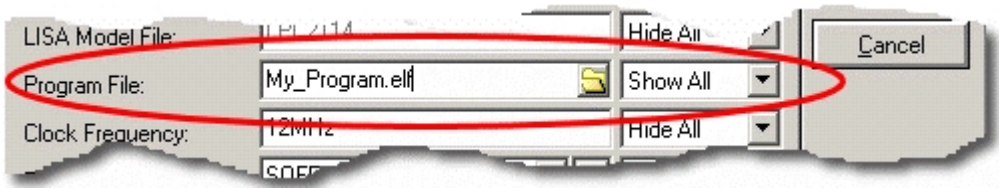
如果已经使用了支持的汇编器和编译器，则 PROTEUS VSM 会为项目中每一个源文件生产一个源代码窗口。当仿真暂停时，这些窗口会出现在 DEBUG 菜单上。

### 源代码窗口

源代码窗口是 Proteus 中用来仿真电路系统的最基本的工具。运行程序时允许客户单步仿真程序

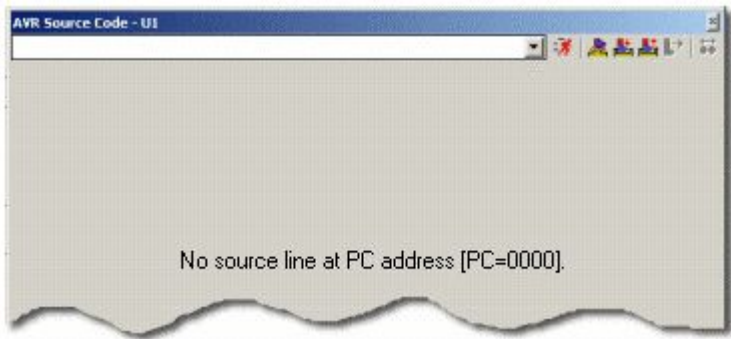


或同时运行整个嵌入式系统。程序可以通过两种方式加载到源代码窗口。一是通过编译器输出的仿真文件，二是通过由源代码控制系统生成的 SDI 文件。当使用编译器时，仿真文件应该通过如下所示的程序属性窗口载入到原理图部件。

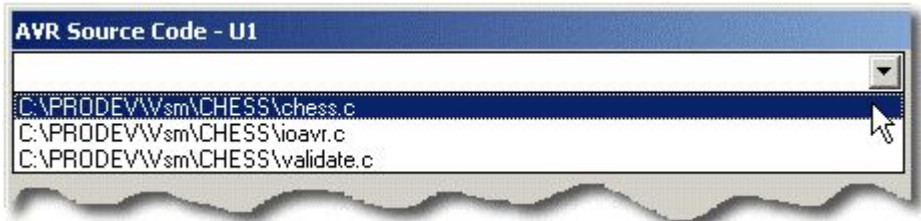


大家通常会犯一个错误就是载入到原理图部件的是 HEX 文件然后企图采用源代码仿真的方式。这样是不太正确的。为了能进行源代码仿真，仿真系统还得需要更多的有关地址、变量等信息。只有当用汇编程序编写或编译工具不能生成任何可支持的仿真格式时才使用 HEX 格式文件（这种情况的可能性很小）

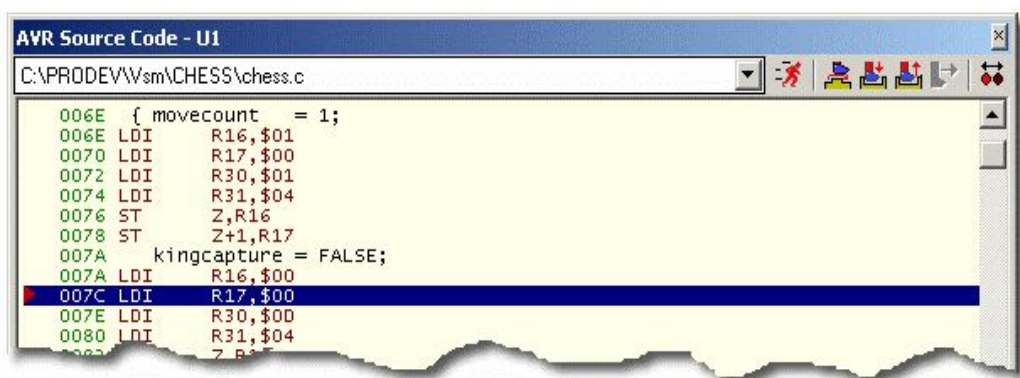
在进程控制按钮中按 PAUSE 启动仿真，在默认的情况下源代码窗口就会出现，界面如下图所示。



界面中所示信息的原因是仿真时间为 0，此时微处理器的程序计数器并不是指向程序可执行行。用户当然可以从源程序窗口顶端的下拉框选择想仿真的程序然后按需要进行单步仿真或设置端点，如下图。



如果使用高级语言编写的程序，则可以使用高级语言或机器码进行单步调试。如果想调出机器码，可以使用组合键 CTRL+D 或者在源代码窗口上单击右键后选择 Disassembly 即可。



在源代码窗口点击右键后选择合适的使能/禁止选项，用户便可以观测行号、代码和地址。

## 支持的仿真文件格式

### 1. ELF/DWARF

此格式是 Proteus 推荐的仿真文件格式。该格式一般为大多数模型编译器所支持，在可预见的未来它将流行于大多数编译器。在 Proteus 使用 ELF 文件，只要在微处理器的属性对话框指定该文件即可。

- 目前 ELF/DWARF 只支持 AVR 和 ARM7 模型，但不久将会支持 PIC 和其它系列的芯片。
- ELF（二进制文件）和 DWARF（调试文件）必须保存在同一个目录下——当在编译器选择 ELF/DWARF 作为仿真文件格式时，默认情况下 ELF、DWARF 保存在同一个目录。

### 2. COFF

此格式虽然为 Proteus 支持的格式，但在 ELF/DWARF 为有效格式的情况下还是应以 ELF/DWARF 为优先选项。这就意味着一个 PIC 编译器可能将 COF 中的浮点数描述成 IEEE 浮点数或 PIC 的浮点数并且不提供清晰的信息说明格式的种类。如果需要更多有关 COF 格式的信息请查阅相关网站。如果想使用 COF 格式，则需在编译器指定仿真格式然后在微处理器的属性对话框的 program property 中使用 COF 格式。

### 3. UBROF

UBROF 是 IAR 编译器一种可选的仿真格式。如果想使用 UBROF8 格式（非最新版本），则需在编译器指定仿真格式然后在微处理器的属性对话框的 program property 中使用 UBROF 格式。

用户还可以在 IAR Embedded Workbench 中直接驱动 proteus，这方面的信息可以在 Using Proteus as a Virtual ICE 章节找到。

### 4. OMF51

OMF51 是由 Keil 编译器生成的仿真格式。如果想使用 OMF51 格式，则需在编译器指定仿真格式然后在微处理器的属性对话框的 **program property** 中使用 OMF51 格式文件。

更新版本的 Keil 编译器同样可以生成 ELF/DWARF 格式文件，因此建议使用 ELF/DWARF 格式。

用户同样可以在 Keil 中直接驱动 proteus，这方面的信息可以在 Using Proteus as a Virtual ICE 章节找到。

## 5. COD

COD 格式由 Bytecraft 生成并广泛使用于 PIC 系列中。如果提供的信息不齐全并且有其它格式可以采用时，建议不使用 COD 格式。特别地，COD 符号仿真数据格式的限制是指 VSM 仿真支持局限于机器码仿真和只能观察指定的寄存器位置并且不支持变量显示。

如果想使用 COD 格式，则需在编译器指定仿真格式然后在微处理器的属性对话框的 **program property** 中使用 COD 格式文件。

## 6. BAS

目前 Proteus 针对 Crownhill Proton Development Suite 为客户提供一种解决方案。编译完程序后在 PIC 微处理器的属性编辑框的 **program property** 中指定 BAS 文件。

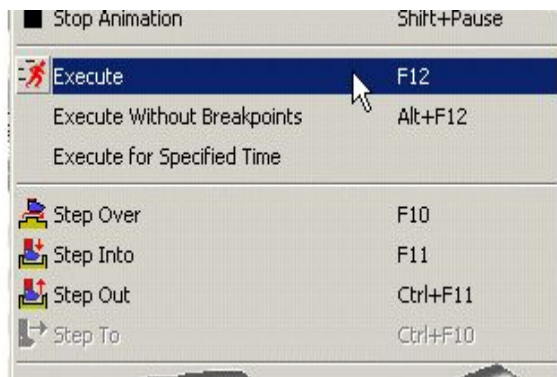
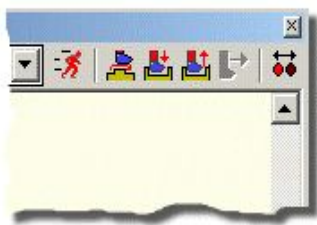
## 7. SDI

SDI 格式只在用汇编程序编写的程序能使用并且通过 ISIS 中的源代码控制系统进行组合生成。如果使用 SDI 格式，则需在 Code Generation Tools 对话框设置相应选项从而指定 Debug Extraxction tool（在安装软件时将会正常创建）然后在微处理器的属性对话框的 **program property** 中使用 SDI 格式文件。

注意：在微处理器的 **Program property** 选项中不要输入源程序文件的名字——Proteus VSM 不仿真“C”或“ASM”文件；CPU 从上述的仿真文件中载入和执行二进制机器码。

# 单步执行

Proteus 提供一系列用于单步执行的选项。全部选项可以在代码窗口本身选取或在 **Debug** 菜单操作。



- Step Over—单行执行程序。当遇到子程序时，以子程序为一行执行（即整个程序被 执行）
- Step Into—执行一条源代码指令。如果没有激活了的源代码窗口，将执行一条机器码指令。
- Step Out—执行到当前的子程序返回
- Step To—执行到程序运行到光标所处行。该功能只在有激活的源代码窗口的情况下才有效。

注意：除了 Step To 命令外，其它单步执行命令都可以在没有有效的源代码窗口的情况下使用。

Proteus 仍然可以仿真由 Proteus 不支持的下载工具生成的代码。当然这样做还是存在不少困难的。

## 使用断点

对于研究软件的问题或设计中软硬交互性，断点是研究这些特性的一种非常有用的方法。一般情况下，在发生问题的子程序开端设置一个断点，启动仿真，然后与设计进行交互式仿真一直到程序运行到断点处。在断点处，仿真将被挂起。之后，利用单步执行观测寄存器值，存储单元和其它电路中涉及到的条件。打开 Show Logic State of Pins 对于调试会更有帮助。

在有效的源代码窗口里，可以按 F9 或右键菜单的选项对断点进行设置或清除。只能在有结果代码的程序行设置断点。



如果源程序被更改，Proteus VSM 将会尽力重新导入基于子程序地址的断点。如果源代码更改的地方过多则导入过程将会发生错误，但是一般情况下程序仍能正常工作，用户不必考虑导入过程中发生的错误。

## 变量窗口

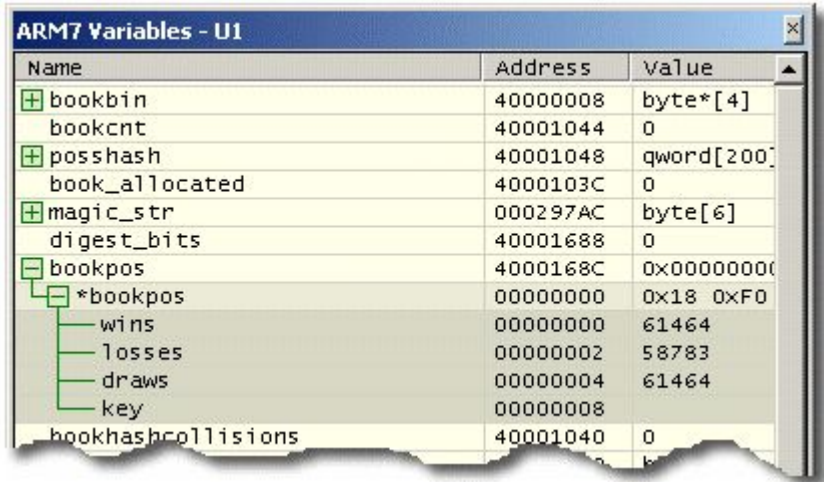
由 Proteus VSM 提供的大部分下载器可以读取程序变量的位置和源程序行的地址。Proteus 显示变量窗口和源代码窗口。

变量窗口是一个非常强大的仿真帮助工具。它提供了一系列可自定义的特性以便能更容易获取

和明白所显示的信息。

### 1. 混合类型和指针的扩展

变量窗口将会很好处理混合类型（结构，数组，枚举）和指针。它以扩展了的树型方式显示类型的内容或者指针的相关参考值。



载入变量窗口的信息由编译器生成的 Object Module File 提供。有可能的话推荐使用编译器生成的 ELF/DWARF 格式，在 ISIS 中的原理图部件指定使用 ELF 格式。在没有 ELF/SWARF 的情况下，可以使用 COFF 格式并且该格式应该提供完全的变量支持。

### 2. 更改信息和旧的数据值

当某变量值改变并且仿真暂停时，对应变量的值在变量窗口中以高亮显示。同样可以通过在变量窗口中点击右键选中 Show Previous Values 来查看旧的数据值。

如果一混合类型（例如一结构），它的任何元素发生变化时，只是对应的元素以高亮的方式显示。

### 3. 拖放到观察窗口

虽然在程序运行时变量窗口是隐藏的，但用户可以向观察窗口拖放变量。观察窗口中的变量在仿真的过程中一直是可见的。因为观察窗口的可见性，使得可以针对变量设立观测条件，并当条件匹配时暂停程序。

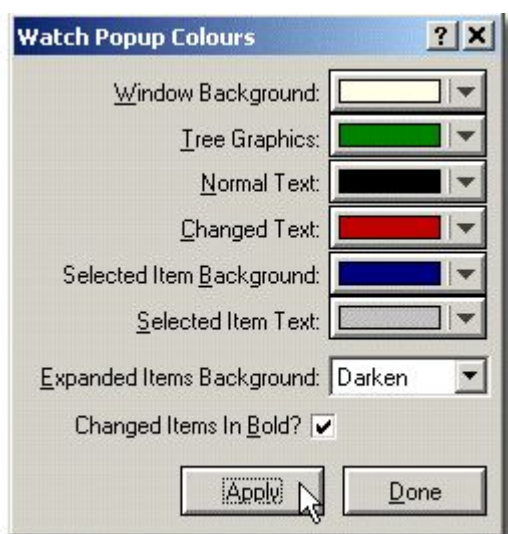
### 4. 隐藏和显示全局变量

在大型设计里可能有很多全局变量。在变量窗口点击右键从中选择 Show Globals 选项，可以隐藏全部局部变量，以便更加容易观测局部变量。



## 5. 颜色设置

在变量窗口点击右键，选择 Set Colours 选项后出现一对话框允许用户为变量窗口全部项目设置颜色。



## 多个 CPU 仿真

Proteus VSM 可以仿真含两个或两个以上 CPU 的电路系统。每一个 CPU 各自生成包括代码和变量的弹出式窗口。这些窗口在 Debug 菜单的子菜单整合在一起。

当单步执行时，光标所在的源代码窗口对应的 CPU 被认为是主处理器，其它 CPU 将自由运行。当主 CPU 已经完成一条命令时，将中止从 CPU 的执行。如果将光标从源代码窗口退出，则最后光标所在的源代码窗口的处理器为主 CPU；单击其他任意处理器的源代码窗口，则改变主 CPU。



# 源代码控制系统

## 概述

源代码控制系统有两大主要功能：

- 程序源代码寄存于 ISIS 中。这一功能使用户可以直接在 ISIS 编辑环境中直接编辑源代码，而无须手动切换应用环境。
- 在 ISIS 中定义了源代码编译为目标代码的规则。一旦程序启动，并执行仿真时这些规则将被执行，因此目标代码被更新。

注意并不一定需要利用源代码控制系统仿真微处理器。事实上，如果用户定义的汇编程序或编译器自带的 IDE，可直接在其中编译。生成了可执行程序后切换到 Proteus 即可。

## 创建源代码文件

- 将源代码程序添加到设计中
- 在 Source 菜单中选择 Add/Remove Source Files
- 为源程序文件选择 Code Generation Tool。如果第一次使用新的汇编程序或编译器，需要使用菜单项 Source—Code Generation Tool。
- 点击 NEW 按钮，用文件选择器为源程序文件选择或输入文件名。如果期望的文件名不存在，用户可以在文本框中键入期望的文件名。
- 在 Flags 框输入处理源文件所需的标记。当每次使用某一特殊工具并且在该工具已登记的前提下，可以在 Flags 框输入标记。
- 点击 OK 将源文件加载到设计中。

请记住编辑微处理机的属性并指定目标代码文件名（通常是一 HEX，S19 或带调试信息的数据文件）。ISIS 不能自动加载目标文件，因为原理图中有可能有多于一个微处理器而导致 ISIS 无法判断某一目标文件的所属取向。

## 在源代码上工作

### 1. 编辑源代码程序：

使用组合键 ALT+S

在 source 菜单中选择相应源文件的序号

### 2. 返回 ISIS，编译目标代码并运行仿真：

在源文件窗口中按组合键 Alt+Table 返回 ISIS 编辑环境。

按 F12 执行程序，或按组合键 Ctrl+F12 开始调试。

任何一种方法，ISIS 会指示文字编辑器保存其文件，检查源代码和目标代码文件的采样时间并唤醒适当的代码生成工具编译目标代码。

### 3. 重新编译目标代码：

选择 Source 菜单 Build All 命令重新编译全部目标代码

ISIS 将运行需要的代码生成工具来编译目标代码，命令行输出将在窗口显示。这样可以方便地检查编译中的警告和错误。

## 安装第三方目标代码生成工具

许多共享汇编软件和编译器可以从系统 CD 安装到 Proteus Tools 目录下，并且会被自动作为 Proteus 的代码生成工具。然而，如果想使用其他生成工具，则需要使用 Source 菜单的 Define Code Generation Tools 命令。

### 注册新的代码生成工具

在 Source 菜单中选择 Define Code Generation Tools 命令

点击 NEW 选项，利用文件选择器为新的代码生成工具导入可以执行文件。用户也可以注册批处理文件作为代码生成工具。

为源文件和目标代码文件输入扩展名。扩展名决定文件的类型，因为对于特定的源文件，ISIS 将会通过文件类型确定是否执行这一工具。如果选择了 Always Build，则该工具一直处于激活状态，可以省去目标代码的扩展名。

在 Command line 指定路径。%1 代表源代码文件，%2 代表目标代码文件。也可以使用 %\$ 代表 Proteus 目录路径，%~ 代表 DSN 文件放置目录。

如果想要使用 Proteus VSM 源代码调试功能，用户需要一个汇编器或编译器的 Debug Data Extractor (DDX)。调试数据提取程序是一个小的命令程序，它通过由汇编程序或编译器产生的列表文件提取目标代码/源代码行的交叉参考信息。

如果用户有一个 DDX 程序，为由代码生成工具生成的列表文件或符号调试文件键入扩展名，然后单击 Browse 选择 DDX 程序的路径和文件名。如果用户的编译器可以输出 COF，ELF/DWARF 等格式文件，则没有必要使用 DDX。

## 使用 MAKE 程序

在有些情况下，由 ISIS 执行的简单的编译规则不能满足用户的要求——特别是设计项目中包含多个源代码文件和目标代码文件。在这种情况下，用户需要使用外部 MAKE 程序，可按以下步骤建立项目：

安装 MAKE 程序（通常提供汇编程序/编译器环境）作为代码生成工具。设置源文件名的扩展名为 MAK，将 Always Build 选项选中。对于典型的 MAKE 程序命令行设置为 -f%1。

使用 Source—Add/Remove Source Files 命令添加 make 文件（例如。MYPROJECET.MAK）到设



计项目中。

添加源代码文件，但选择代码生成工具为<NONE>

每次项目的编译，ISIS 将会运行外部 MAKE 程序，并将项目的 make 文件作为参数。该参数将确定哪一代码生成工具将被运行。一个好的 MAKE 程序将会提供很大的灵活性。

## 使用第三方源代码编辑器

Proteus VSM 提供了一个简明的源代码文本编辑器 SRCEDIT。使用该编辑器可以编辑源代码文件。SRCEDIT 本质上是 NOTEPAD 的改进版本。它可以打开多个源文件，并且响应 DDE 的要求，保存任一修改后的缓存。

如果用户有更加先进的文本编辑器，例如 UltraEdit，便可以在 ISIS 中使用这一编辑器。注意 IDE 类型的环境有可能不响应 DDE 命令，因此以这种方式整合并不是那么合适。

### 建立第三方源代码编辑器

选择 Source—Setup External Text Editor 菜单项

点击 Browse 按钮，使用文件选择器定位文本编辑器的可执行文件。

ISIS 利用 DDE 协议指示文本编辑器打开并保存文件。参考文本编辑器的说明文档向供应商索取命令语法的详细资料。如果服务名称不确定的，不妨以诸如 ULTRAEDIT 的产品名一试。

## 使用第三方 IDE

大部分专业的编译器和汇编器都有自己的集成开发环境或称 IDE。例如，IAR's Embedded Workbench, Keil's uVision, Microchip's MP-LAB and Atmel's AVR studio。如果用户使用其中一个 IDE 开发程序代码，可以很容易地在 IDE 中进行编辑，生成可执行文件（HEX, ELF/DWARF, COF 文件）后切换到 Proteus VSM，然后进行仿真。

PROTEUS VSM 提供两总方式支持外部 IDE 运行：

- 将 Proteus 作为外部调试器

当 Proteus 作为外部调试器时，Proteus 中可用的下载器支持由编译器生成的带调试信息的目标文件格式。下载器提取的不仅是高级语言程序行的地址而且在允许的条件下是程序变量的位置。

通常的仿真格式是 COD(在 PIC 系列中使用)，UBROF（被全部 IAR 编译器使用），ELF/DWARF（普通格式），COFF（普通格式）和 OMF（在 8051 系列中使用）。我们同样提供下载器支持其它格式，例如支持由 Crownhills Proton Development Suite 生成的列表文件。

点击仿真进程控制盘的 PLAY 按钮开始实时仿真或者按 STEP 按钮运行到第一行命令行。在后一种情况，代码窗口将会出现，用户可以开始单步运行代码。

- 将 Proteus 作为虚拟 In-circuit 仿真器

到目前为止，虚拟的 ICE 只支持 Keil、IAR、MPLAB。虚拟 ICE 是个发展很快的领域，

用户应该经常访问公司网站专题”Supported Compilers”以获得最新的信息和说明资料。安装 Proteus 实现与这些工具一起运行的说明同样可以在这网站中找到。

## 调试工具

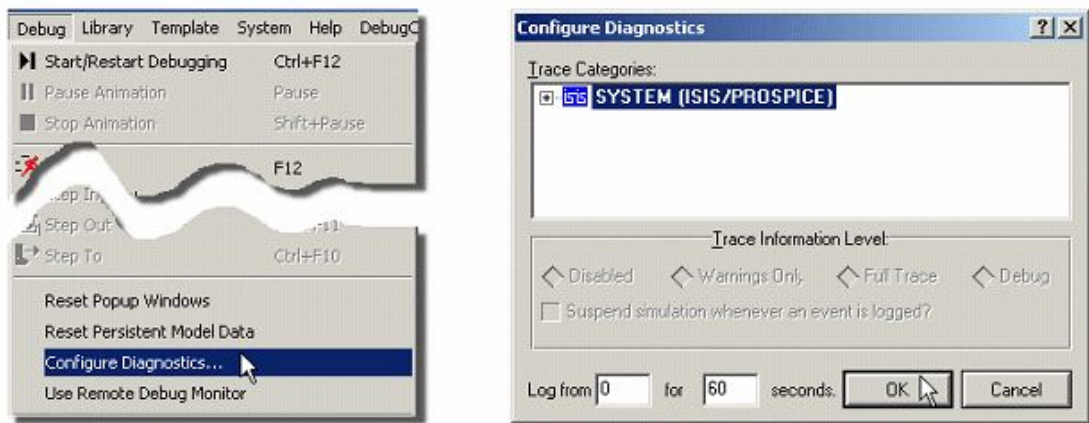
Proteus VSM 包含了扩展的调试工具和跟踪模式。它们在查找错误和验证系统运转方面很有作用。该机制使得全部仿真行为或某一指定的仿真行为在仿真过程中被记录下来并显示在仿真指示器上。

作为一个系统级的仿真器，Proteus VSM 的调试模式不仅仅可以调试微处理器还可以调试合适的外围设备（LCD 显示，I2C 寄存器，温度控制仪器等），这些跟踪模式可以通过调试对话框使能。

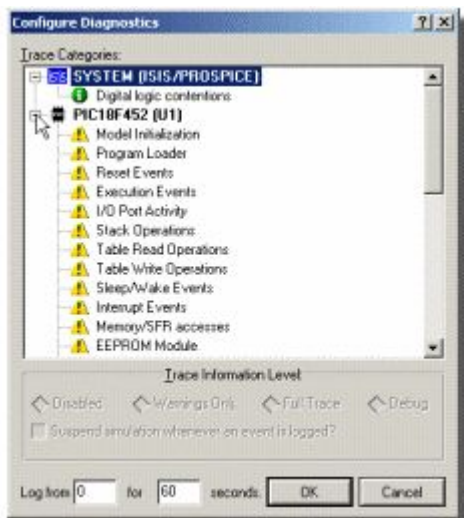
### 设置调试参数

#### 当调试功能使能时设置仿真

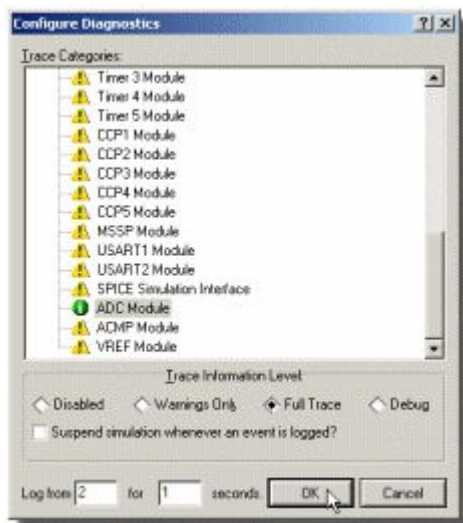
从 Debug 菜单中运行 Configure Diagnostics 对话框



在对话框里展开选项，从中找到欲使能的调试项。



点击左键选定感兴趣的项目，然后在 Trace Information Level 下选中 Full Trace。用户可以设置‘Arm’的时间和运行时间以控制调试功能激活的时间间隔。



重复选择感兴趣的其它选项，然后退出对话框  
仿真时调试功能将在 Arm 时刻被激活，运行指定的周期，所有调试结果将显示在仿真指示器上。

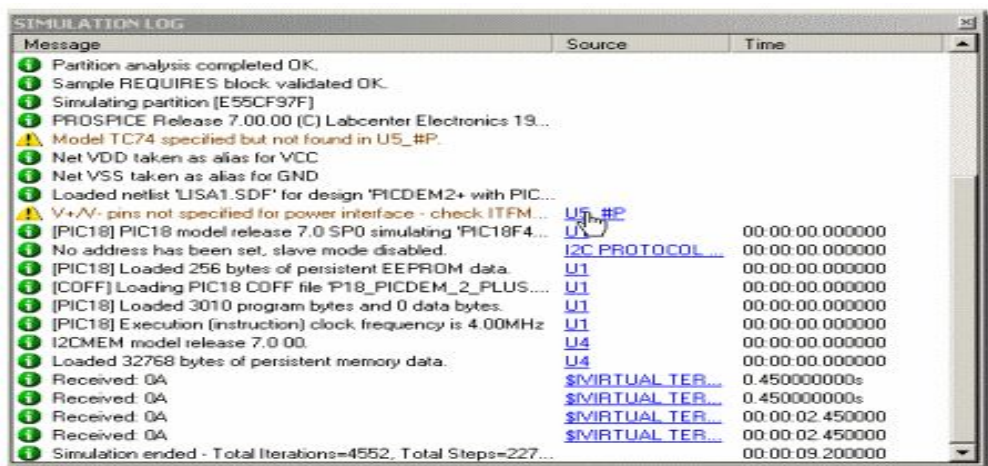
注意：使能了的跟踪调试增加了仿真的负荷。但是，因为它们是一般用来确定模棱两可的问题，达到仿真的目的，因此仿真非实时的情况不再是个问题。

## 仿真指示器

仿真指示器存放了在仿真期间产生的全部错误、警告和调试信息。该指示器在 ISIS 中的位置处于仿真进程控制盘附近的状态栏。指示器的状态显示栏在仿真过程中不断更新，不仅显示记录信息（包括错误、警告和调试信息）而且显示了此类信息的种类。用户可以在仿真过程的任何时刻或者在完成仿真后左键点击状态栏指示器所在处调出仿真指示器（可以显示全部信息）。



在由一个物理元器件产生的全部信息（信息量非常丰富，包含除了系统信息外的其它信息）的右边有一个与这些信息相关的栏目。这项功能可以显示产生信息的原理图器件。在 Source 列中单击左键将最小化仿真指示器并且可以放大/缩小、标记相应的元器件。

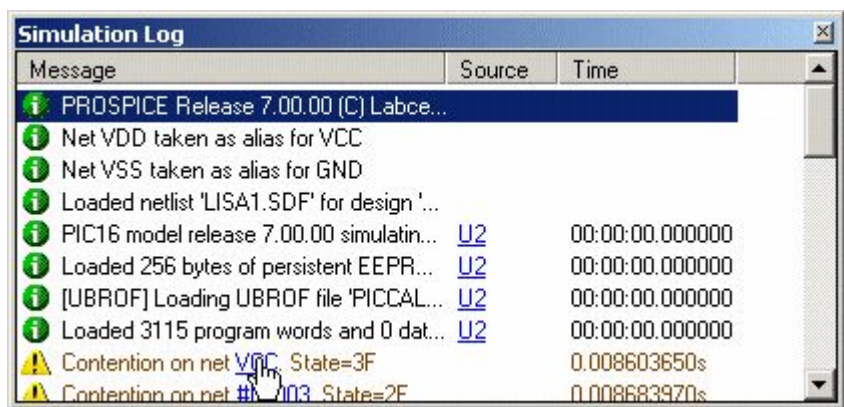


该功能特别有用，尤其是在复杂的设计中。利用它可以检查一器件周围的硬件设计以便解决问题。

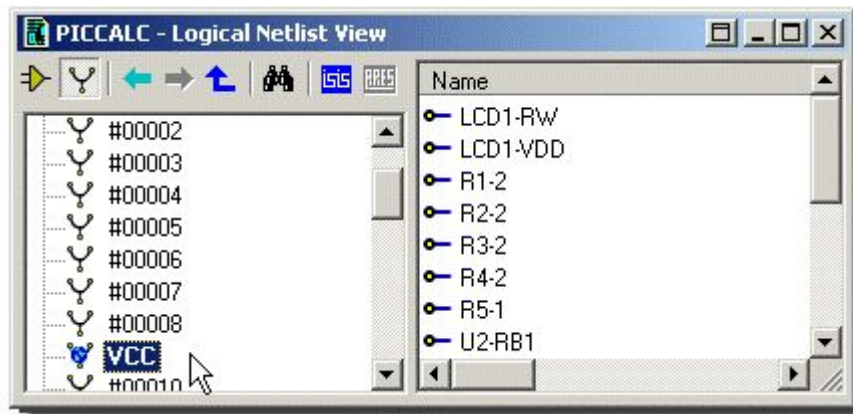
关于仿真错误信息方面最让人感到挫折的事情之一就是—一些问题（例如网点分歧，SPICE 单复数等）与网点相关而不是与某一具体的元器件相关，因此在原理图中将冲突电路隔离出来是非常困难。仿真指示器包含了在信息上的超链接，点击链接允许客户主导整个设计。这样就简化了仿真的任务。

## 从仿真指示器中操作一个网点

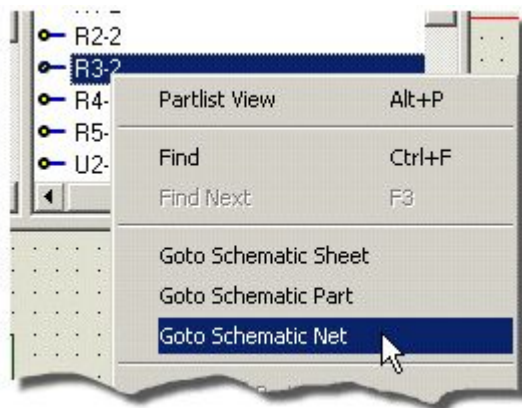
在仿真指示器中点击感兴趣的信息所包含的“NET” 链接。



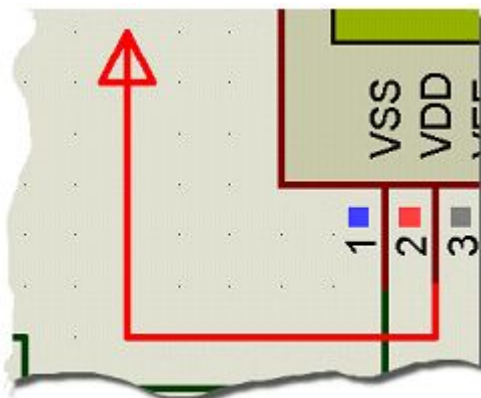
点击“NET”链接后，仿真指示器将会最小化回到状态栏并调出 ISIS 中 Design Explorer。Design Explorer 在窗口的左边显示一系列网点同时在窗口的右边显示了与冲突网点相关的连接点（引脚）。



在其中的一个引脚单击右键后从右键菜单中选择 Goto Schematic Net.



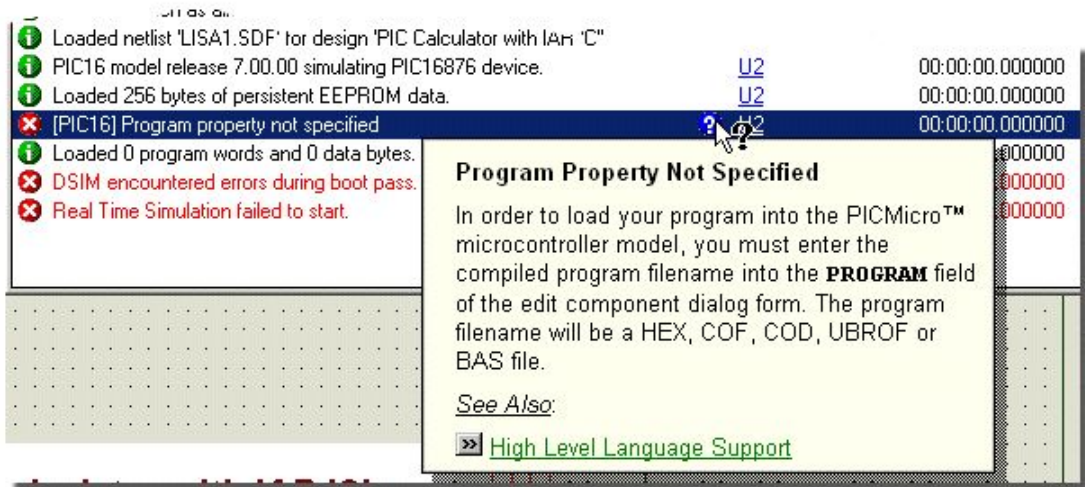
出错的网点将在原理图中以高亮方式显示



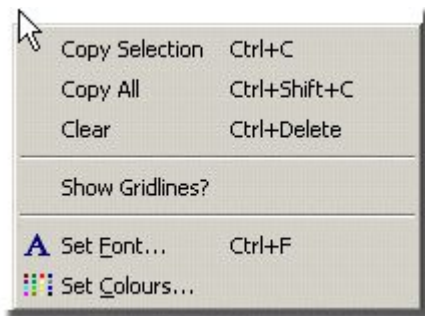
注意：该检测方法并不是绝对可行。当原理图部件通过等效电路（用户利用 ISIS 已有器件构成的与某一元器件等效的电路）进行仿真并且出错的网点在该等效电路里时，这种仿真方法就不可行了。然而，虽然存在着一些不能检测的情况，但是上述的技术在大多数正常条件下是可以正常起作用的并且是一个强大的分析工具。

一般的错误信息或者设计缺陷的错误信息包含了帮助文件。这些帮助文件详细介绍了解决问题的方法。这就不仅清晰显示了发生错误的地方而且提供了解决问题的相关建议。用户可以为某一错误信息调出相应的帮助文件。方法是在信息的右边左键点击“help”按钮。如下图所示：





在仿真指示器点击右键，产生带几个选项的菜单栏。利用这些菜单栏可以实现拷贝已有数据到粘贴板或设置窗口的颜色/网格、外观。



## 观测窗口

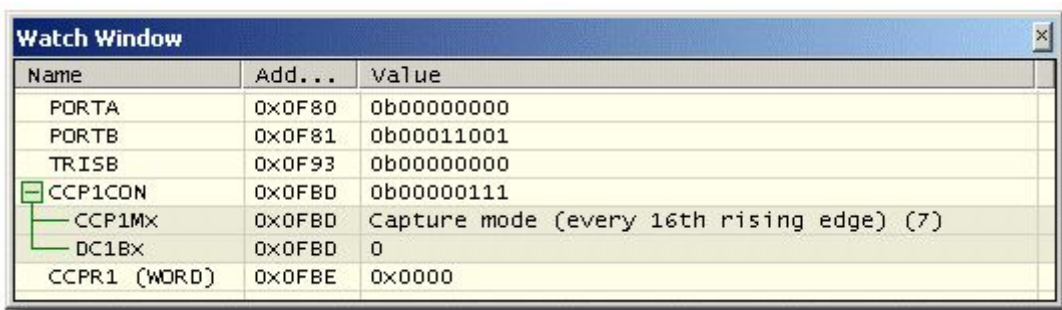
### 给观测窗口增加项目

微处理器模型包含的变量，存储器和寄存器窗口只能在仿真暂停时显示，但是观测窗口则可以实时更新显示值。它同时可以给独立存储单元指定名称，这比从存储器窗口查找要容易得多。

在观测窗口中添加项目的步骤如下：

1. 按组合键 CTRL+F12 开始仿真或系统处于仿真状态时，按 pause 按钮，可暂停仿真。
2. 单击 Debug 菜单中的窗口序号，显示包含被观测的项目的存储器窗口和观测窗口。
3. 使用鼠标左键定义存储单元的位置。所选定单元以反色显示。
4. 从存储器窗口中拖动选择的项目到观测窗口。
5. 用户同样可以使用鼠标右键菜单的“Add Item by Name ”或“ Add Item by Address”。对于微处理器的 SFR 来说，观测窗口特别有用。因为该窗口可以监测个别的位变量（PIC、AVR

和 ARM 模型)。



Name	Add...	Value
PORTA	0x0F80	0b00000000
PORTB	0x0F81	0b00011001
TRISB	0x0F93	0b00000000
CCP1CON	0x0FB0	0b00000111
CCP1MX	0x0FB0	Capture mode (every 16th rising edge) (7)
DC1Bx	0x0FB0	0
CCPR1 (WORD)	0x0FBE	0x0000

(观测窗口监测 PIC18 的寄存器 CCP1CON)

在观测窗口中修改项目：

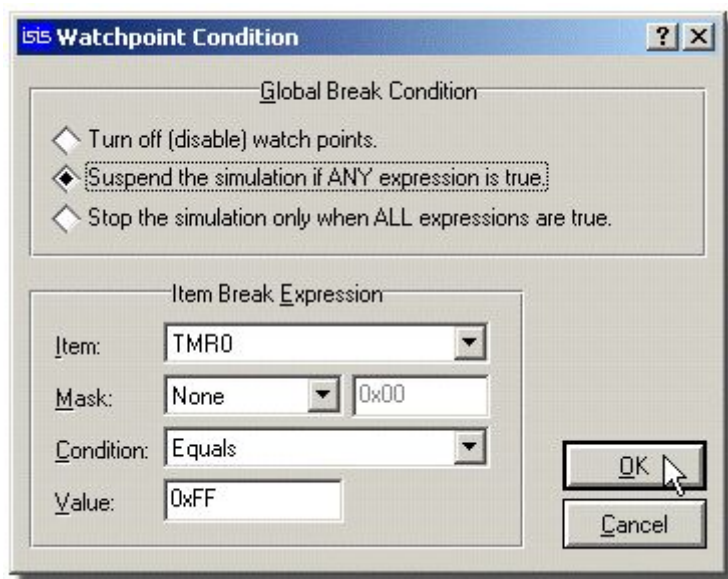
1. 已经将项目添加到观测窗口中，用户可以选择其中一个项目，然后进行以下编辑。
2. 按组合键 CTRL+R 或 F2 键重命名项目
3. 利用右键菜单的选项改变数据量。系统提供了几种数据类型（例如，16 或 32 位字，或字符串）。第二个和后续的字节假定是服从项目地址的，因此，选择多字节或字符串显示方式时，只须从存储器窗口拖出第一字节即可。
4. 改变数据的格式，如二进制、八进制、十进制或十六进制。

## 观测点条件

当项目的值出现特定情形时，观测窗口可以延缓仿真。当特定的项目值发生改变时，观测窗口可延缓仿真。当然，用户也可以定义更加复杂的情形。

设定观测点条件：

1. 按组合键 CTRL+F12 开始仿真，如果系统已经处于仿真状态，按暂停键，可暂停仿真。
2. 单击 Debug 菜单中的窗口序号，显示观测窗口。
3. 增加观测点
4. 选择观测点并右击，选择右键快捷菜单中的 Watchpoint Condition
5. 在 Global Break Condition 选项区进行设置。这一设置决定了当任一项目表达式为真时，或所有项目表达式为真时，系统是否延缓仿真
6. 对 Item Break Expressions 选项区进行设置。一个 Item Break Expressions 选项由 Item、Mask、Condition、Value 构成。



增加观测情形会给仿真器增加一定的负荷,这是因为 ProSPICE 在每次到达观测点时刻时必须重启观测情形。然而,当寻找模棱两可的设计漏洞时,该特性还是非常有用的。一般来说,系统多消耗一点时间到达测试点并不影响系统的正常运行。

## 断点触发

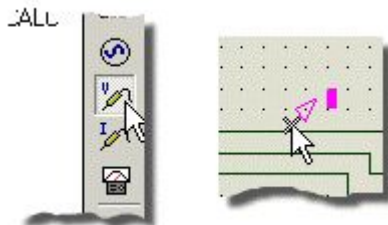
许多元器件都具有当一特定电路情形发生变化时触发仿真延缓的功能。与单步仿真结合使用时,这一功能非常有用,因为电路可以直至某一特定情形出现时,才可以正常仿真,然后使用单步执行,就可以看到电路下一步将会发生什么动作。

## 建立硬件断点

用户利用硬件断点可以在匹配硬件断点的条件下暂停仿真。当调试设计,尤其是异步仿真电路并且需要分析电路所受的影响时,该特性非常有用。

建立硬件断点的步骤如下:

1. 在欲触发断点的导线(总线)上放置电压探针

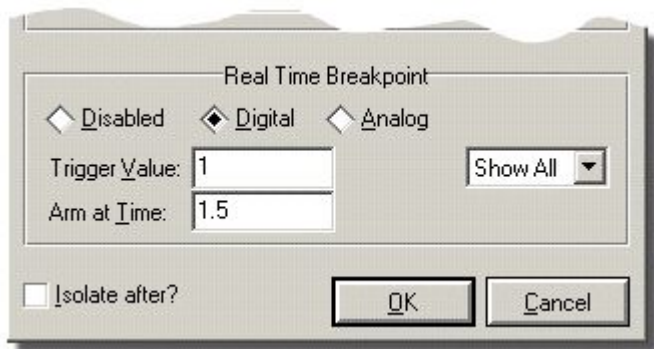


2. 在探针处点击右键,然后从菜单中选择 Edit Properties





3. 根据探针所在的网点，在对话框的底部选择 **Digital** 或者 **Analog** 并指定触发值。对于数字网点和单导线，1 或 0 对应的是逻辑高或逻辑低；对于模拟网点，将会是一具体的值。用户同样可以指定 **arm** 时间，使得断点在指定的一段时间后有效。



4. 点击 **OK** 退出对话框，然后按 **PLAY**（或组合键 **CTRL+F12**）运行仿真

## 电压断点触发（RTVBREAK）

电压断点触发元器件有单引脚和双引脚两重形式。当单引脚对地电压值，或两个引脚间的电压值比指定的值大时，触发断点。用户可以将一个元器件连接到任意控制电压源（AVCS）触发断点。

一旦触发电压超过指定值，元器件不会再次触发，直至电压回落到触发门限以下，并再次到达指定值。

## 电流断点触发（RTIBREAK）

电流断点触发元器件有两个引脚。当流过其电流超出指定值时，触发断点。

一旦触发电流超过指定值，元器件不会再次触发，直至电流回落触发门限以下，并再次到达指定值。

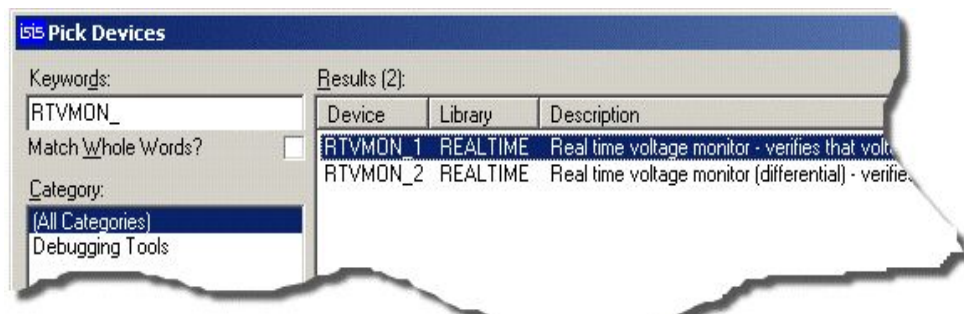
## 数字断点触发（RTDBREAK）

数字断点触发元器件的引脚有多种形式。用户可以根据需要进行选择。当输入引脚的二进制值等于 **Trigger Value** 时，触发断点。例如，指定 **RTDBREAK\_8** 的值为 **0x80**，那么当 **D7** 为高电平，**D0~D6** 为低电平时，元器件进行触发。

一旦触发触发发生，元器件不会再次触发直至输入引脚出现不同的值。

## 电压监测器（RTVMON）

该元器件有单引脚和双引脚两种形式。当输入电压在指定的最小值或最大值的范围外时，触发断点、警告或错误条件。将这些元器件整合到仿真模型里，则当模型里的电压超出指定的极限时，将会警告最终用户。



## 电流监测器（RTIMON）

该仪器有两个引脚。当流过仪器的电流在在指定的最小值或最大值的范围外时，触发断点、警告或错误条件。将这些元器件整合到仿真模型里，则当模型里的电流超出指定的极限值时，将会警告用户。



*Selecting the Current Monitor from the Library Browser.*

# 基于图表的仿真

## 6.1 介绍

交互式仿真有很多优势，但在很多场合需要捕捉图表来进行细节分析。基于图表的仿真是可以做很多的图形分析：比如小信号交流分析，噪声分析及扫描参数分析等。

\*PROTEUS 用户只有在购买 ASF 模块后才能做基于图表的仿真。

## 6.2 建立一个基于图表的仿真

### 6.2.1 概况

基于图表的仿真过程建立有 5 个主要阶段。

- 绘制仿真原理图。
- 在监测点放置探针。
- 放置需要的仿真分析图表，比如：用频率图表显示频率分析。
- 将信号发生器或检测探针添加到图表当中。
- 设置仿真参数（比如运行时间），进行仿真。

### 6.2.2 绘制电路

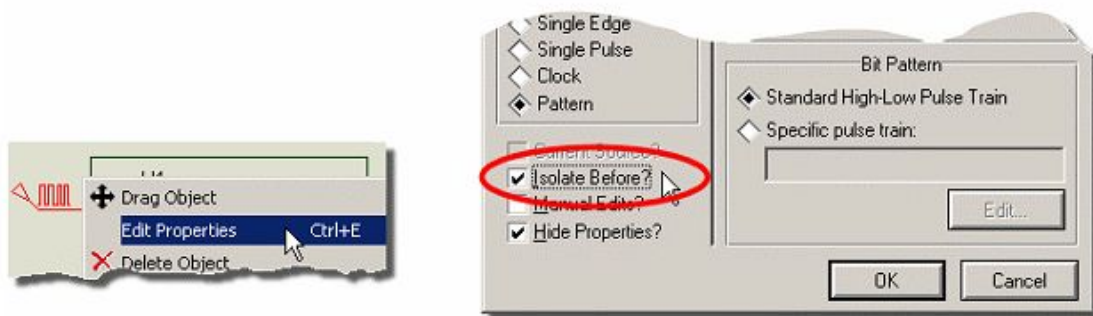
在 ISIS 中输入需要仿真的电路；电路图的绘制技巧在 ISIS 手册中有详细的介绍。

### 6.2.3 放置探针和发生器

仿真过程的第二阶段是在监测点放置信号发生器及探针。探针、信号发生器和其他元件，终端的放置方法是一样的。选择合适的对象按钮，选择信号发生器、探针类型，将其放置到原理图中需要的位置，可以直接放置到已经存在的连线上，也可以放置好后再连线。

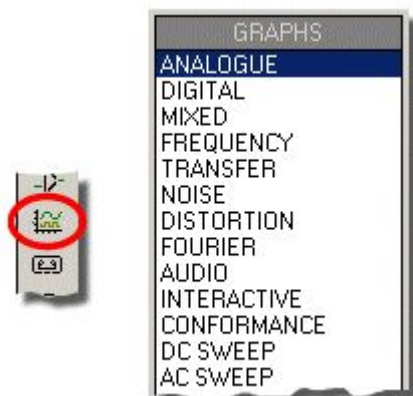


这个阶段中，使用者可以选择将信号发生器隔离出仿真范围。



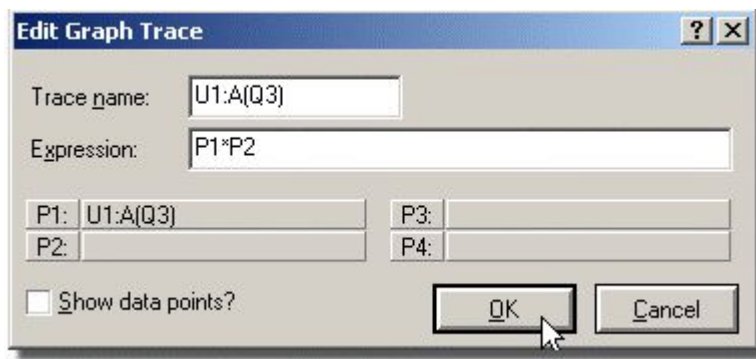
## 6.2.4 放置图表

仿真的第三个阶段放置分析图表。图表类型包括模拟、数字、转移、频率、扫描分析等等。选择图形按钮，选中需要分析的图表，用拖曳的方法放置在原理图中合适的位置。



## 6.2.5 在图表中添加轨迹

在原理图放置多个图表后，必须指定每个图表对应的探针/信号发生器。每一个图表也可以显示多条轨迹，这些轨迹数据来源一般是单个的信号发生器或者探针，但是 ISIS 提供一条轨迹显示多个探针，这些探针通过数学表达式的方式混合。举个例子，一个监测点既有电压探针也有电流探针，这个检测点对应的轨迹就会是功率曲线。



曲线显示对象的添加有两种方式：

- 在原理图中选中探针或激励源拖入到图表当中（在同一个图表当中可以添加多个探针或激励源）。
- 在 EDIT GRAPH TRACE 对话框中选中探针，需要多个探针时添加运算表达式。

## 6.2.6 仿真过程

基于图表的仿真是命令驱动的。这意味着整个过程是通过信号发生器，探针及图表构成的系统，设定测量的参数，得到图形，验证结果。其中，任何仿真参数都是通过 GRAPH 存在的属性定义（比如仿真开始及停止时间等），也可以自己手动添加其他的属性（比如对于一个数字仿真，你可以在仿真器系统当中添加一个‘RANDOMISE TIME DELAYS’属性）。

在仿真开始时系统做了以下工作：

- 产生网络表——网络表提供一个元件列表，引脚之间连接的清单，及元件所使用的仿真模型等。
- 分区仿真——ISIS 对网络表进行分析，将其中的探针分成不同的类，当仿真进行时，结果也保存在不同的分立文件当中。
- 结果处理——最后，ISIS 通过这些分立文件在图表中产生不同的曲线。将图表最大化进行测量分析。

如果在以上中的任何一步有错误，仿真日志会留下详细的记载。有一些错误是致命的，有一些是警告。致命的错误报告会直接弹出仿真日志窗口，不产生曲线；警告不会影响到仿真曲线的产生。大多数错误产生源于电路图绘制，也有一些是选择元件模型错误。

## 6.3 图表对象

### 6.3.1 概况

图表也是设计中的一部分。图表的目的是用来显示仿真结果。做什么样分析就用什么样的图表。仿真图形能否正常显示主要决定于激励源及探针是否正确的添加到图表当中。

### 6.3.2 当前图表

所有图表命令 GRAPH 菜单下。在这个菜单下包含一个图表清单，当前图表前有一个标志加以区分。

任何在 GRAPH 菜单下的命令在图表命令盘框中都有对应。



### 6.3.3 放置图表

1. 在模式选择工具栏中选择图表按钮。
2. 从对象选择器中选择合适的图表类型。
3. 在编辑窗口中左键拖曳放置图表。

### 6.3.4 编辑图表

所有的图表可以移动，重新定义大小，编辑图表属性等。

图表属性是通过 EDIT 按钮或 CTRL+ ‘E’ 进入对话框中进行编辑。

### 6.3.5 在图表中添加轨迹

每个图表都可以显示多条轨迹。每一条轨迹数据显示都是和信号发生器及探针相联系。然而，对于模拟及混合图表类型，一条轨迹最多可以表示 1~4 个探针，这些探针通过数学表达式合成。

每一条轨迹都有一个指定作用的标号，这个标号在 Y 轴旁边。有些图表只有一个 Y 轴，所以没有必要指定某条轨迹对应特定的 Y 轴。在默认设置中，轨迹名是和探针名是一致的（多个探针，取第一个探针名）-这些可以在轨迹编辑中修改。

轨迹可以通过三种方式定义：

- 在图表中放置单个激励源或者探针。
- 选取一部分探针，使用 ADD TRACE 命令快速添加。

- 使用 ADD TRACE COMMAND 对话框添加。

前两种方式可以快速添加激励源及探针，每一次添加都生成独立的曲线；第三种方式可以对探针、激励源灵活操作，特别是曲线需要几个不同监测量合成的场合。

#### 向图表中拖放探针或激励源：

1. 选中需要添加的激励源或者探针。
2. 按住左键，将探针/激励源移动到图表中，松开即添加完成。

在图表中添加一条新的曲线，曲线代表一个独立的探针/激励源。注意一点，图表中添加新的曲线时，原有曲线可能会有所变化。

有些图表有两条 Y 轴，这需要在图表的左右两边都放置探针/激励源。此外，对于数模混合图表，添加激励源/探针，系统会自动生成各自类型的曲线（一般在混合图表中的第一个探针/激励源通常会产生一个模拟曲线）。

#### 在一个图表中快速添加激励源/探针：

保证你想添加激励源/探针的对象是当前图表。当前图表在 GRAPH 菜单下有选中标志。

选中要添加的探针或激励源。

点击 GRAPH MENU 菜单下的 Add Trace command 命令。如果有探针/激励源选中，系统会弹出 “Quick add tagged probes?” 提示。

选择 Yes 按钮将选中的探针或激励源添加到当前图表中。

这种方式添加的每一个探针/激励源都会产生一条独立的曲线，这条曲线显示了所代表的探针/激励源数据。如果存在两个 Y 轴，曲线默认设置是左边的 Y 轴。

## 6.3.6 添加曲线命令对话框

如果使用者未采用快速添加模式，点击 Add Trace 命令会弹出 Add Trace 设置对话框（不同的图表类型有对话框会有一些小差别）。这个对话框允许您选择新建曲线名字，曲线的类型（模拟、数字等），Y 轴的位置（左或右），多达 4 个的探针，及探针表达式等。

使用 Add Trace command 命令在一个图表中添加一条曲线：

1. 在 GRAPH 菜单下选择 Add Trace 命令，如果原理图中有选中的探针，系统会弹出快速添加的提示，选择 NO，弹出 Add transient Trace 对话框。
2. 通过 TRACE TYPE 按钮选择图表允许范围内的合适曲线类型，添加到图表中。
3. 选中所添加曲线相适合的 Y 轴位置。只有在图表允许两条 Y 轴时有效。
4. 通过 P1 到 P4 的对话框选择探针，这些对话框有显示探针名的下拉菜单。选中探针，探针名会显示在探针选择对话框及名字对话框中。
5. 如果曲线表达式无效，那么只有探针 P1 有效—仿真后，系统会产生一条代表 P1 的曲线。
6. 重复步骤 3、4，添加曲线需要的所有探针或者激励源。
7. 在曲线表达式对话框输入表达式。添加的探针名由 P1、P2、P3、P4 表示，通过表达式完成探针叠加。
8. 如果曲线类型并不支持曲线表达式，表达式对话框无效。

9. 点击 OK 确定，在图表中添加曲线完成。

### 6.3.7 图表轨迹编辑

图表中一条轨迹可以通过编辑改变轨迹名称及表达式。

**选中，编辑及取消选择轨迹曲线：**

1. 保证要被编辑的曲线所在图表未被选中。
2. 右键单击曲线名选中曲线，曲线名呈高亮状态。
3. 在曲线名上点击鼠标左键，弹出 EDIT GRAPH TRACE 对话框。
4. 编辑曲线名或表达式。对于曲线不支持的表达式更改无效。
5. 选择 OK 完成改变。
6. 在图表中曲线名外位置点击右键释放曲线的选中。

### 6.3.8 改变轨迹的顺序或颜色

按住鼠标左键可以拖动标号来改变曲线在图表中的位置：

对于数字信号，移动它们仅仅改变了电平变化的顺序。

对于模拟信号，可以将曲线从左拖到右，也可以改变曲线颜色分配等。

### 6.3.9 手工 Y 轴设置

一个图表中 Y 轴的默认设置是提供曲线最大可视的 Y 值。在一些场合，当你需要在原理图中比较多个图表时，这种默认设置就需改变。ISIS 允许你设置并锁定左右 Y 轴的最大坐标值。

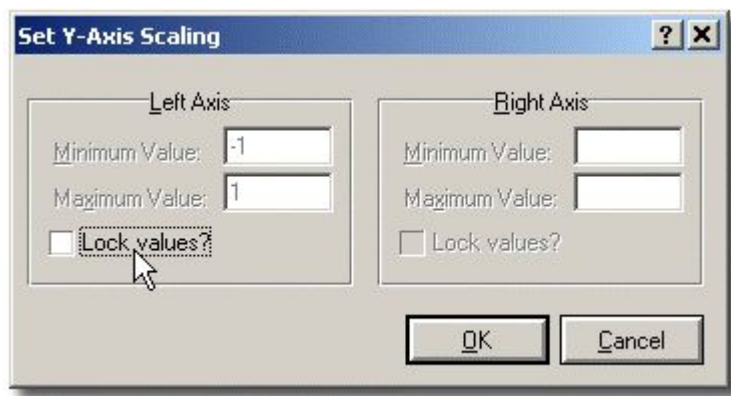
**手动改变 Y 轴范围：**

1. 放置图表，添加曲线，仿真图表。
2. 选中图表，进入 编辑窗口，在右下方即为 Y 轴范围更改选项。



3. 点击选项进入 Y 轴设置对话框





4. 选中 LOCK VALUES 复选框，输入合适的最大值及最小值。

注意：当这些选项呈灰色时表示它们在这个图表中是不可用的。比如有些图表只有左 Y 轴，这样右边 Y 轴不可用的。

5. 关闭对话框重新仿真。

## 6.4 仿真过程


### 6.4.1 命令驱动仿真

在设计 PROTEUS 时，一个主要的目标是将软件设计得更为人性化。以前的一些仿真平台都是按照标准输入一些数据，得到仿真结果。

这方面来说，PROTEUS 有其独到之处，绘制原理图，放置图表以后，每一次更新图表只需按空格键重新仿真，这种方式称为命令驱动仿真。ISIS 中也可以同时放置多个图表，进行多个实验。

对于给定图表，重要的一步是对放置合适的探针。ISIS 要利用这些信息来计算确定设计中的哪一部分需要仿真，无需手动分割，这对以后的 PCB 整板设计也是非常有益的。

### 6.4.2 执行仿真

一旦图表放置好，探针及激励源指定好以后。在图表中使用  命令或直接通过空格键执行仿真。ISIS 会自动分析设计，确定哪一部分需要仿真，更新图表，显示新的数据。通过这个仿真过程，产生一个仿真日志。仿真正确或只有警告时，日志不会弹出，只有存在致命错误时，仿真日志会弹出，图表也不进行更新。

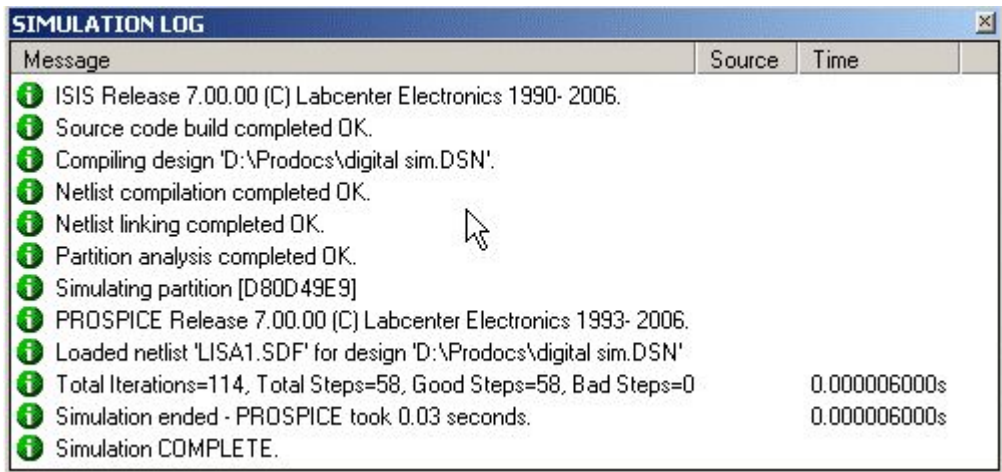
#### 更新图表观测仿真日志：

保证要更新的图表是当前图表，将图表最大化，选择 SIMULATE 命令

仿真结束时，如果有错误发生，会有提示显示这部分结果，选择 YES，仿真数据，如果选择 NO，

系统会弹出仿真日志。

没有错误发生或者有错误发生选择 YES，仿真日志可以通过 VIEW LOG 菜单下调出。



### 6.4.3 仿真开始后发生的事件

当使用者刷新图表，使用 GRAPH 菜单下的 SIMULATE 命令或者 SPACE 键。在这个过程中包含了以下内容：

- 网络表编辑——网络表提供一个元件列表，引脚之间连接的清单，及元件所使用的仿真模型等。网络表在进行引脚连线时非常有用。
- 网络表链接——有一些元器件是通过子网络表或模型文件建模，这些模型文件的保存路径可以通过 SET PATHS COMMAND 对话框中 MODULE PATH 设置。PROTUES 提供很多模型文件，使用者也可以创建自己的模型文件。一个模型文件看成一个分立的子电路设计，它的父电路就是那些需要建模的元器件。

每一次利用这种方式建模时候，在模型文件替换网络表中原始器件时，模型和原来器件的输入输出连接管脚应保持一致。

在网络表链接过程的最后，网络表中剩下的每一个元器件都有一个 PRIMITIVE 属性，这种模式可以直接由 PROSPICE 仿真。

- 分区仿真——ISIS 观测那些放置探针的点，曲线产生就由信号注入点和这些地方决定。一个信号激励视为一个输入，仿真过程中每一点析结果都会保存于一个独立的文件当中。
- 做更深入的分析，需要注意一些仿真前提：比如：一个分区电路 A 有一个输出到分区电路 B 的输入，这时，A 必须先进行仿真。
- 仿真器调用——ISIS 在对每一个分区电路进行仿真计算时，都需调用 PROSPICE。
- 每一次仿真器调用导致一个新的分区文件，这个过程信息同样添加到仿真日志当中，仿真日志中保留仿真过程中的每个动作。
- 结果处理——最后，ISIS 通过分区文件在图表当中建立不同的曲线，图表重新刷新，对曲线进行测量分析。

在整个过程中如果有错误发生，有致命错误也有警告，错误细节都会记录在仿真日志当中。致命错误时图表不刷新，立即弹出日志窗口；警告时图表仍刷新，使用者可以选择观察仿真日志。大

多数错误来源于绘制电路错误，模型文件链接错误等。

# 第 7 章分析类型

## 7.1 介绍

Proteus 中有 13 种基于图表的分析方法，每一种都展示了 PROSPICE 支持的各种电路分析的结果：

模拟信号分析	与示波器非常相似，模拟信号分析图用于绘制电压和/或电流随时间变化的图形。另外，包含几个探针值的表达式的图形也可以绘制，例如：电流乘以电压的表达式可以绘制即时功率的图形。此类分析通常称之为瞬态分析。
数字信号分析	与逻辑分析仪非常相似，用于绘制随时间变化的逻辑电平的图形。描记线可以通过单一的数据位或一条总线的二进制值来表示。数字图表使用事件驱动的仿真方法来计算。
混合模式分析	在同一个图表中包含模拟和数字信号。
频率分析	绘制小信号电压或电流随频率变化的图形，也即是波特图，可以显示幅度和相位的变化。另外，通过使用表达式，可以绘制输入和输出阻抗图。 注意：图中的值是相对于指定输入激励源的增益值。
DC 扫描分析	绘制稳态工作点的电压或电流随扫描变量变化的图形。与模拟分析类似，可以绘制由多个探针值构成的表示式的图形。
AC 扫描分析	绘制一簇对应于扫描变量每一个值的频率图。
转移特性分析	通过扫描一个或两个输入激励源来绘制特性曲线或曲线簇，并绘制稳态的电压和电流值。第一个激励源变量值为 X 轴，第二个激励源变量的每一个变化值将生成一条曲线。
噪声分析	输入或输出噪声电压随频率的变化图。也产生单个噪声贡献的列表。
失真分析	绘制 2 次谐波和 3 次谐波的频谱图。也可以绘制互调失真图。
傅立叶分析	显示瞬态分析的谐波成分。类似于频谱分析仪。
音频分析	执行瞬态分析，然后通过 PC 机的声卡播放结果。也可以把电路的输出生成 Windows 的 WAV 文件保存。
交互式分析	执行交互式仿真，在图表中显示结果。本分析方法可以让你结合交互式仿真和基于图表的仿真的优点。
一致性分析	执行数字仿真，然后把结果与先前保存的一系列结果相比较。特别用于制作软件测试工具，来对基于微控制器的应用进行测试。

此外，所有类型的分析都是从计算工作点开始的，也即是，在时刻 0 的所有节点的初始电压值、支路电流和态变数。关于工作点的帮助信息可以从 ISIS 中得到。

# 7.2 模拟信号瞬态分析

## 7.2.1 概述

本类图表显示的图形与你在示波器上看到的一样。X 轴显示时间，Y 轴显示电压值或电流值。我们经常称此类分析为瞬态分析，因为它是发生在时域的。

瞬态分析也许是最经常用到的分析类型。所有你能够从真实示波器中测量到的数据都可以从模拟图表中测量得到。它可以用来检查电路是否正常工作，可以快速测量增益值，可以直观地看到已经失真的信号，可以测量流过电源或元器件的电流值等等。

## 7.2.2 计算方法

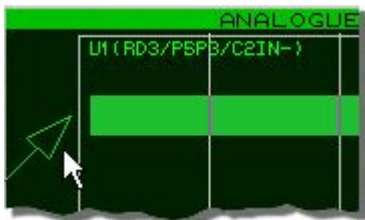
从学术上来说，本类分析与非线性节点分析有关，它使用了所有 SPICE 仿真器使用的计算方法。考虑到具体一个时刻，电路中的所有元器件都可以由电流源和/或电阻代替。这种安排可以通过根据欧姆定律和基尔霍夫定律建立的联立方程来描述，并通过高斯消除法来求解。每一次求得的解，包括电流源和电阻值，根据元件模型内部定义的规则进行调整，这个过程会一直重复，直到得到一个稳定的结果。

对于一个包含前导时间的仿真分析，有两个独立的阶段。先是计算电路的工作点——即在仿真一开始时电路的电压值。然后考虑前导时间对电路的影响，并对每一步进行重新计算（像前面说的迭代收敛）。每一步时间增量对于计算的稳定性是至关重要的，PROSPICE 将在用户定义的门限内自动进行调整。对于快速变化的电路，如高速开关线驱动器相对于平滑变化的电路，需要较小的时间步长，也更费事。使用时间步控制和迭代解都会增加大量的计算，使瞬态分析相当地慢。

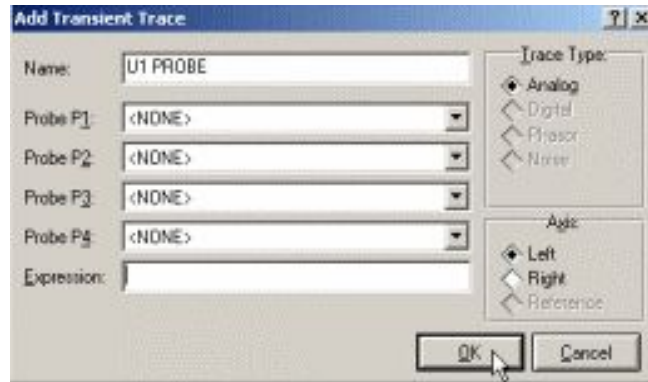
因为算法包含了迭代运算，所以结果有可能不会收敛。这种情况最经常发生在初始时间点，这意味着仿真不能在特定的时刻为工作点建立一个稳定的或唯一的一组值。偶尔也会出现在电路行为变得严重不可测的特定时刻。PROSPICE 使用了一系列的技术去避免这类问题，但不可能完全避免。也即是说，基于 Berkeley 的 SPICE3F5，你可以尽可能地得到你想要的。

## 7.2.3 使用模拟图表

1. 模拟图表既可以只使用左边的 Y 轴，也可以使用左边和右边的 Y 轴。探针可以通过两种方式放置到特定的轴上：
2. 选取然后拖动探针到图形的对应边上。



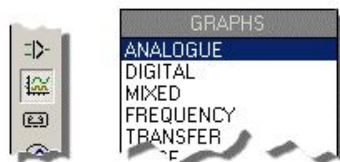
3. 通过“增加瞬态踪迹（Add Transient Trace）”对话框。



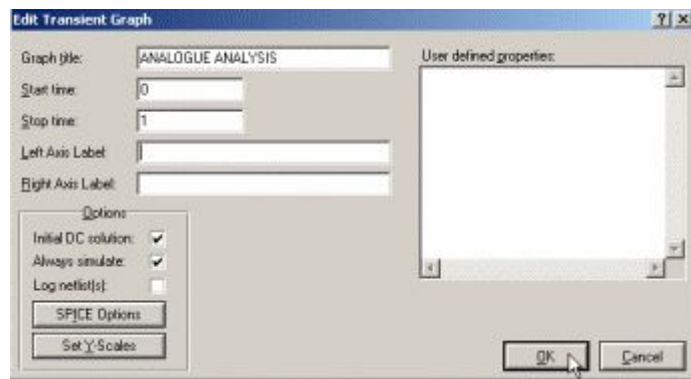
通常使用左右两边的 Y 轴去分离不同单位的踪迹。例如：如果要同时显示电压和电流踪迹，通常可以使用左边的 Y 轴显示电压，而右边的 Y 轴显示电流。也可以把数字探针放到模拟图表中，但混合图表更适合于此类应用。

#### 使用瞬态分析图表分析模拟电路的步骤：

1. 为电路增加需要的激励源，驱动电路的输入端。
2. 在电路中放置探针，可以在电路中的各个节点放置，也可以在输出端放置。
3. 绘制模拟图表。



4. 把探针（如果需要，包括激励源）放入到图表中。
5. 编辑图表（用鼠标指向图表，然后按 CTRL+E 键，设置仿真停止时间、标签和需要的控制属性）。



6. 开始仿真，可以从 Graph 菜单选择 Simulate 命令或使用空格键。

# 7.2.4 定义模拟轨迹表达式

通常，我们只是把电压和电流探针加入到图表中，来观察电路的工作情况。但是不要忘了，我们也可以建立基于探针值的数学表达式来进行模拟瞬态分析。例如：我们在电路的输出端放置了电压和电流探针，把它们相乘，就可以得到这一点的输出功率图。

## 绘制输出功率图的步骤：

- 1. 在电路的输出端放置电压和电流探针。
- 2. 打开“增加瞬态踪迹（Add Transient Trace）”，可以通过快捷键“CTRL+A”打开。
- 3. P1 赋值为电压探针。
- 4. P2 赋值为电流探针。
- 5. 把表达式改为  $P1 \cdot P2$ 。
- 6. 单击“OK”按钮。
- 7. 单击空格键开始仿真。



# 7.3 数字信号瞬态分析

## 7.3.1 概述

数字图表正如逻辑分析仪一样，X 轴显示时间，Y 轴显示竖直信号堆，可以是单一信号，也可以是总线信号的二进制值。

## 7.3.2 计算方法

数字瞬态分析使用事件驱动仿真技术进行计算。这与模拟瞬态分析不同之处在于，只有当电路元器件的状态发生改变时才进行计算。另外，只有离散的逻辑电平才会考虑，因此，这使得我们可以在更高级别对元器件的功能进行描述。例如：对于一个计数器，我们是把它作为一个寄存器值来

考虑的，在每个时钟周期对它的值加 1，而不是把它作为几百个晶体管来考虑。这使得对同一个电路，事件驱动的仿真比模拟仿真的速度快几个数量级。

## 引导入口

引导入口的作用是在正常仿真之前定义电路所有网络的初始状态。

通过以下几点定义引导入口：

所有连接到 VCC 和/或 VDD 网络的输入引脚被认为是高电平的。

所有连接到 GND 和/或 VSS 网络的输入引脚被认为是低电平的。

所有连接到与激励源相连的网络的输入引脚的初始态与激励源的 INIT 属性定义的一致。

所有剩下的引脚被初始化为不固定的电平。

所有模型都需要输入引脚电平进行检测，并依此设置输出引脚电平，当每个输出引脚设置完成，该引脚连接的网络的状态将会被重新计算。

当网络改变了状态，连接到该网络的模型将会被要求重新计算输出状态。这个过程会一直持续到出现新的稳定状态。

## 事件处理循环

紧跟着引导入口，DSIM 开始进行仿真。仿真是通过一个循环来实现的，这个循环重复执行以下两步：

- 当前时间的所有状态改变事件顺序读完并应用到相应的网络上，这个过程将产生一组新的网络状态。
- 当一个网络改变了状态，所有连接到这个网络的模型将重新进行仿真运算。当它们的输出改变了状态，将产生新的事件，并放置到事件队列中。

当然，不同的模型在不同的时刻将产生处理到期的事件。DSIM 内核因此必须为所有在每个循环周期尾产生的新事件进行排序。

另外值得注意的是，我们的原理图支持零延时的模型。在这种情况下，在同一个时间码产生的事件将会依据产生它的方法分批处理。

## 停止条件

仿真将在遇到以下情形时停止：

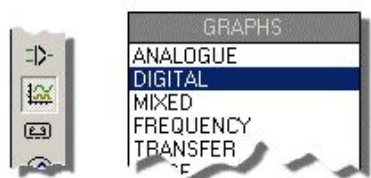
- 设定的停止时间到了。
- 事件队列是空的。这意味着电路达到一个恒定的状态。
- 出现零延时的逻辑矛盾，导致当前时间停止前进。
- 系统错误，例如事件队列超出内存范围，一般不会出现这种情况，除非电路设计不稳定，很可能是由于在电路的某些地方产生了高频（例如 100MHz）振荡。



### 7.3.3 使用数字图表

使用瞬态分析图表分析数字电路的步骤:

- 1. 为电路增加需要的激励源，驱动电路的输入端。
- 2. 在电路中放置探针，可以在电路中的各个节点放置，也可以在输出端放置。
- 3. 绘制模拟图表。



- 4. 把探针（如果需要，包括激励源）放入到图表中。
- 5. 编辑图表，设置合适的仿真停止时间，和图表的标题、控制参数等。
- 6. 从菜单调用“仿真图表”命令，或者使用空格键进行仿真。

注意:

激励源要使用数字激励源，特别要注意的是脉冲（Pulse）激励源是模拟的。

在数字网络中只能使用电压探针，如果使用电流探针，将会迫使 PROSPICE 进行混合信号仿真分析，将会大大降低仿真速度。

### 7.3.4 数字数据的显示

#### 一般的描绘线

所有 DSIM 仿真器的输出值使用以下六种数字状态:

状态类型	关键字	图形外观
强高	SHI	高电平青色
弱高	WHI	高电平蓝色
浮动	FLT	中间电平白色
冲突	CON	中间电平黄色
弱低	WLO	低电平蓝色
强低	SLO	低电平青色

因此可以通过描绘线的颜色和位置来确定描绘线在具体时刻的状态，或者通过放大图表，把鼠标指向需要查看的位置，从而读出状态。另外，在描绘线标号的右端将显示鼠标所指处的逻辑电平。

#### 总线描绘线

总线描绘线（通过在总线上放置电压探针）在交叉线的中间，显示了总线状态的 16 进制值:



当总线的一位或者多位在中间状态时，总线描绘线也将处于中间状态。另外，当两边的交叉线太接近，没有空间显示十六进制值时，将会省略掉十六进制值，这时可以通过把鼠标指向该位置来显示。

## 7.4 混合模式瞬态分析

### 7.4.1 概述

混合图表可以在相同的时间轴（X 轴）上同时显示数字信号描绘线和模拟信号波形。混合模式分析用于同时包含数字和模拟模型的电路分析中，而且这是唯一可用的方式。

### 7.4.2 计算方法

混合模式瞬态分析既包含了 SPICE3F5 的非线性节点分析方法，也包含了 DSIM 的事件驱动仿真分析方法。详细的实现细节非常复杂，但对基本的算法，可以简单地概括一下：

- 在仿真之前，对每一个网络进行分析，以确定该网络连接的是数字模型还是模拟模型的器件。当某个网络使用模拟电压去驱动数字输入时，PROSPICE 将自动插入 ADC 接口对象，当数字输出驱动模拟器件时，它将自动插入 DAC 接口对象。
- 当几个数字输入由同一个网络驱动时，将建立多个 ADC 接口对象，从而每个输出都可以模拟不同的逻辑转换电平和负载特性。类似地，虽然很少几个数字输出会同时驱动同一个模拟器件，但遇到此类情况时，也会建立多个 DAC 接口对象。
- 建立工作点，将在下面详细描述建立过程。
- 然后进行标准的模拟信号分析，直到当 ADC 输出的电压超过数字信号的转换阈值，从而产生数字事件，此时模拟分析将被挂起，将进行数字仿真。
- 当数字仿真导致 DAC 接口对象输出的状态改变时，在开关时间内强制进行模拟仿真。实际上，DAC 的输出模拟了转换时间（上升或者下降），并且在这期间将模拟几个时间点。

#### 建立工作点

混合模式仿真工作点的建立特别复杂，因为模拟电路的状态将影响数字器件，反之亦然。基本上，PROSPICE 的处理过程如下：

- 初始条件（IC）值将应用到模拟和数字网络上。其它网络的开始电压假设为 0。
- 像标准的 SPICE 仿真一样，建立节点矩阵并进行运算。
- 在矩阵运算的每个重复过程中，数字电路根据 ADC 的变化不断重复进行计算。这些变化通过几个数字模型进行传递，数字电路不断重复计算直到达到稳定状态。

- DAC 接口对象将根据数字电路状态变化而重新调整输出，如果数字电路没有达到稳定状态，将会被忽略，因为它可能是一个瞬时的引用。
- 循环重复，直到找到一个稳定的状态，或者已经达到重复的限制。

### 7.4.3 使用混合图表

混合图表的使用与模拟图表的使用完全一样。除了还需要添加数字探针。添加第一个数字探针时，要使用“添加探针对话框”。然后，就可以通过拖拉的方式添加数字探针和模拟探针。数字探针拉到图表的数字部分，模拟探针拉到图表的模拟部分。

## 7.5 频率分析

### 7.5.1 概述

通过频率分析，可以检查电路在不同频率上的行为。本类分析只考虑单一频率的行为，而不像频谱分析一起考虑所有的频率。本类分析在效果上就像在输入端接上频率发生器，然后通过 AC 电压表去观察输出结果。可以检查信号幅度和相位。

频率分析产生电路的频率响应或者波特图。对于检查滤波器是不是像设计的那样工作或放大器在要求的频率范围内有没有正常工作时很有用。

频率分析图表也可以用来绘制电路在某个频率上的小信号输入和输出阻抗。

### 7.5.2 计算方法

频率分析先进行电路工作计算，然后把所有的有源器件用线性模型代替。有源器件的内部电容是在工作点上计算得到的，并假设在工作电压波动的时候不会变化。所有的激励源，除了频率参考激励源（见下面）之外，都使用它们的内阻代替，这将使电源线能有效地连接在一起。

频率分析使用复数执行线性计算。关键频率将从开始频率到最后频率根据事件增量逐渐增强。频率分析的线性特性使得它比瞬态分析快很多，虽然它使用了复数进行计算。

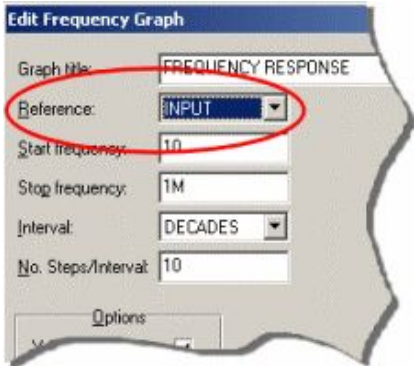
必须注意的是，频率分析把电路假设为线性的，这意味着，在输入端加纯正弦信号，在所有的频率上，输出也必定是纯正弦信号。当然，没有任何实际的器件是纯线性的，但是有很多器件都非常接近线性，而适用于本类分析。也有些电路并非都是线性的，例如带施密特触发输入端的线中继器。对于非线性电路，“频率响应”没有实际的数学意义，所以在任何频率上进行仿真都不会产生有意义的结果。如果对此类电路的频域响应感兴趣，就应该使用傅立叶分析方法。

### 7.5.3 使用频率图表

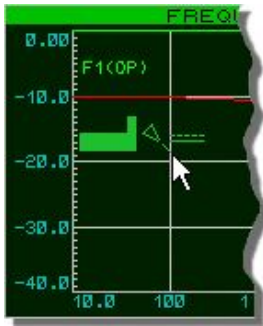
为了计算输出的正弦信号的幅度，必须在输入端加入一个参考的正弦信号。PROSPICE 将会自

动进行这个操作，但必须要知道电路的输入端是哪个。所以必须要对每个频率分析图表设置一个“参考激励源”，指示仿真器在哪里加入参考信号。这里有三种方法去进行设置：

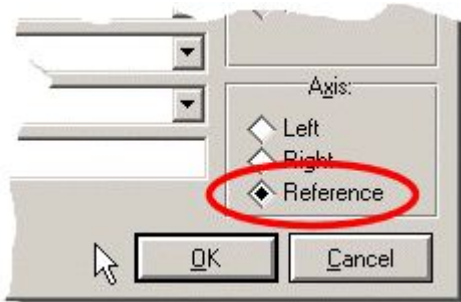
- a. 使用“编辑频率图表”对话框的“参考”字段去选择输入激励源。这个激励源可以是任何一个单脚的激励源，也可以是 ASIMMDLS 库中的 FREQREF 原型。



- b. 拖曳模拟激励源到频率分析图表中。



- c. 使用“Add Trace”对话框添加参考激励源。



FREQREF 仿真原型在定义一个新的模型时，经常用来驱动新模型的内部电路。如在麦克风模型中，当需要考虑模型其它部分的作用时，要包含一个明确指定的激励源作为参考。

第二和第三种方法比较经常使用。拖拉一个激励源到频率分析图表与到瞬态分析图表有所不同，到瞬态分析图表是添加一个激励源探针，而到频率分析图表是把激励源作为电路的参考点。这个激励源不一定是正弦激励源，直流、脉冲和 PWLIN 激励源也一样能正常工作。如果想把激励源作为一个探针来添加，可以通过“Add Trace”对话框来进行。

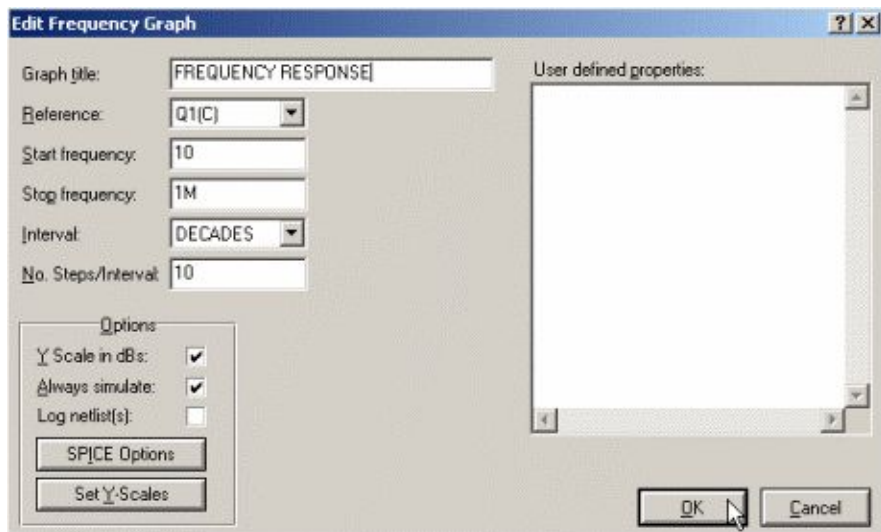
参考激励源的幅值总是为 1 伏，相位总是为 0 度。参考激励源的内阻与第一处定义的激励源的一致。它是用于 dB 计算，即  $0\text{dB}=1\text{V}$ 。ISIS 会限制一个非常小的值（ $-200\text{dB}$ ）去代替  $\log(0)$ 。

频率分析图表有左右两个 Y 轴。左边的 Y 轴是用于显示被探测信号的幅值，右边的 Y 轴是用于显示相位。如果拖拉探针到图表的左边，它将显示幅值，如果拖拉探针到图表的右边，将显示相位。

X 轴用于显示参考激励源的频率。在频率分析图表中总是使用对数尺度，左边的 Y 轴可以显示 dBs 或正常的单位，右边的 Y 轴总是使用度为单位。

### 使用频率响应分析电路的步骤：

1. 在需要测试的节点放置探针。
2. 绘制频率分析图表。
3. 把探针添加到图表中。左边是幅值，右边是相位。对同一个探针可以同时分析幅值和相位。
4. 编辑图表（用鼠标指向图表，然后按 CTRL+E 键）设置仿真开始、停止频率、标签和需要的控制属性。
5. 按下空格键调用 PROSPICE 进行仿真。



例子 ZIN.DSN 和 ZOUT.DSN 显示了怎样使用表达式进行输入和输出阻抗分析。

## 7.6 DC 扫描分析

### 7.6.1 概述

通过 DC 扫描分析，可以看到电路参数的改变会怎样影响电路的工作。这是通过 PROSPICE 仿真器的属性表达式评估（Property Expression Evaluation）来实现。扫描图表分析定义了一个变量，可以在用户定义范围内以平均的步距进行扫描。扫描变量可以是电路的任何属性，例如电阻值、晶体管的放大系数，甚至是电路的温度。

DC 扫描曲线显示了探测点随扫描变量变化时稳态的电压或电流。它可以用来绘制电路的 DC 传输特性曲线，只要把激励源的值作为扫描变量，也可以绘制电路在某个工作点下，元器件参数变化对电路影响的曲线。

## 7.6.2 计算方法

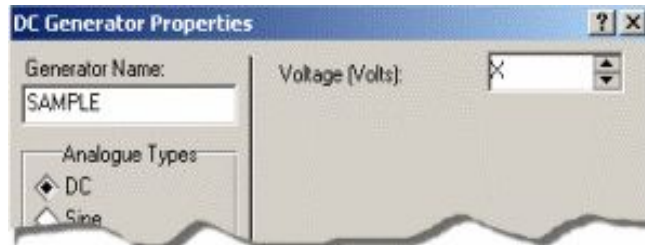
在 DC 扫描中，PROSPICE 会在寻找电路的工作点，然后增加扫描变量进行计算中循环。PROSPICE 会重复计算步与步之间所有的变量值，所以扫描变量会经常被使用，基于扫描变量的变量也可以使用。所有的电路初始化参数都会在每次重新计算工作点时使用。

## 7.6.3 使用 DC 扫描分析图表

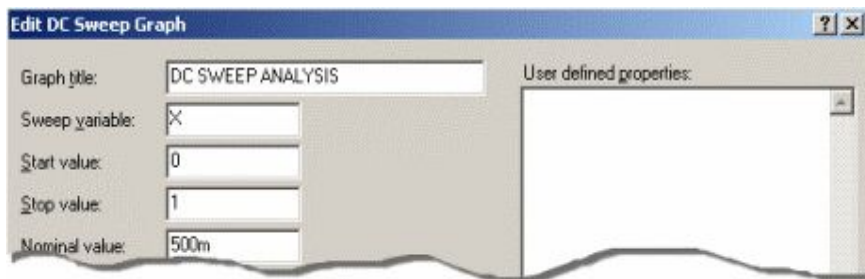
与模拟瞬态分析类似，可以使用两个 Y 轴进行分析。X 轴显示的是扫描变量值。可以使用追踪表达式（Trace expressions）。当选择步数时，记住仿真的时间只是大概与步数成正比。

绘制电路的传输函数的步骤：

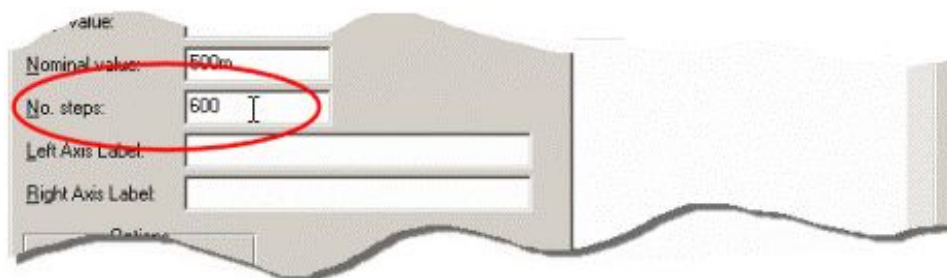
1. 添加 DC 激励源到电路的输入端，设置它的值为变量 X。



2. 放置一个或多个探针到电路的输出端。
3. 绘制 DC 扫描图表，添加探针到图表。
4. 编辑图表（CTRL+'E'），设置输入扫描变量的开始和停止值，设置扫描变量为 X。



5. 按下空格键调用 PROSPICE。
6. 如果放大图表后，结果看上去是脱节的或有角的，请在“Edit DC Sweep Graph”对话框中增加步数。



### 绘制电路参数变化的步骤:

1. 绘制原理图，添加探针和激励源到电路中。
2. 编辑需要改变参数的元件，把它的值设置为都变量 X 的表达式，可以设置一个或多个元件。
3. 绘制 DC 扫描图表。添加探针到图表中。
4. 编辑图表，依照上一个例子设置扫描参数。
5. 按下空格键调用 PROSPICE。

提示:

Proteus 中所有的对话框都可以通过“Point and shoot”接口和对话框窗口右上角的问号来得到快速的帮助提示。

## 7.7 AC 扫描分析

### 7.7.1 概述

本类分析将为扫描变量的每一个不同值建立一簇频率响应曲线。

AC 扫描图表主要用于测试一个元器件变化对电路频率响应的影响。

### 7.7.2 计算方法

本分析方法与通常计算频率响应完全一样，对扫描变量的每个取值进行频率响应计算。因此也要满足线性电路的约束。

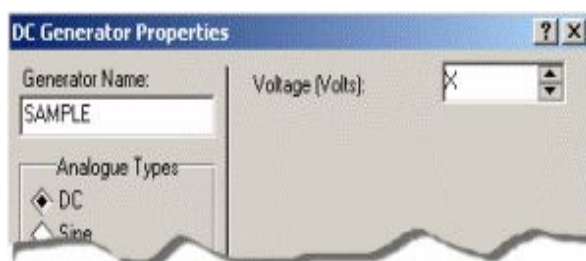
### 7.7.3 使用 AC 扫描图表

像频率分析一样，左边和右边的 Y 轴分别显示的是增益和相位。当然也要定义一个激励源作为参考点，用于计算增益。

电路参数变化对频率响应的影响:



1. 绘制原理图
2. 编辑要测试的元器件，设置属性值包含扫描变量 X，可以设置一个或多个元器件。



3. 绘制 AC 扫描图表，添加探针到图表中。
4. 添加激励源到电路输入端。
5. 拖拉电路输入端的激励源到图表中，作为参考激励源。



6. 编辑图表 (CTRL+E)，设置扫描参数，设置频率参数到需要测试的频率上。
7. 按下空格键，调用 PROSPICE 进行仿真。

## 7.8 DC 转移曲线分析

### 7.8.1 概述

本类图表特别用于测试半导体器件，产生特性曲线簇，当然也偶尔会用于其它用途。每条曲线都绘制了根据指定输入激励源变化时的一组电压或电流工作点。另外一个激励源值的变化用于产生一簇曲线。

### 7.8.2 计算方法

本类分析方法与 DC 扫描分析类似，但这里可以使用两个变量进行扫描。先找到激励源初始值时的工作点，然后第一个变量一步步在指定范围内增加，绘制一条曲线。在第一个变量的每一个扫描过程完成后，第二个扫描变量增加，用于绘制另一条曲线。

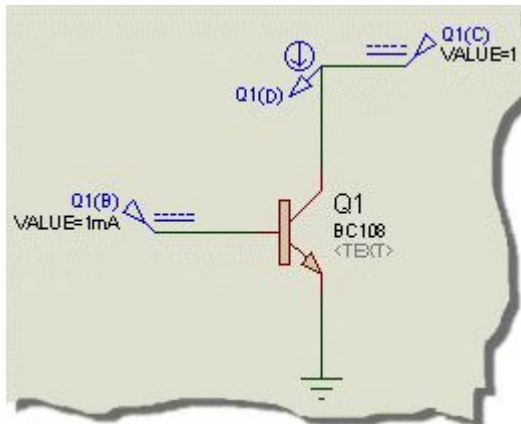


### 7.8.3 使用 AC 扫描图表

与模拟瞬态分析一样，可以使用左右两边的 Y 轴。X 轴显示的是第一个扫描变量，每一条曲线对应第二个变量的扫描值。在这里也可以使用变量表达式进行扫描。

绘制晶体管转移特性曲线的步骤：

1. 绘制类似于下图的电路。

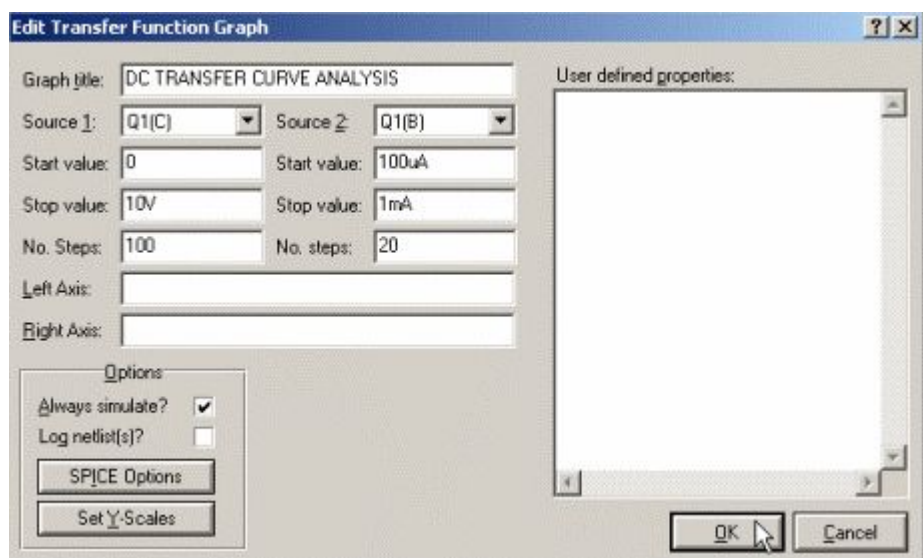


Q1(B)是电流源，Q1(C)是电压源，Q1(D)是需要测试的集电极电流。

2. 绘制转移特性图表，添加探针 Q1(D)到图表中。



3. 编辑图表 (CTRL+E)。
4. 指定 Source 1 为 Q1(C)，Source 2 为 Q1(B)。
5. 设置电压和电流范围在合理的范围内，如 Q1(C)为 0 到 10V，Q1(B)为 100uA 到 1mA。
6. 调整 Q1(B)的步数，使用中间的间隔比较合理，应该注意，绘制出的曲线将比设定的步数多一条，因为一步有两个离散值。
7. 单击 OK，关闭对话框，然后进行仿真。



## 7.9 噪声分析

### 7.9.1 概述

SPICE 仿真器可以模拟电阻和半导体器件中产生的热噪声。电路中在某个频率范围内独立的噪声成分由电压探针收集到一起。噪声电压，归一化为噪声带宽的平方根，可以绘制相对于频率的图谱。噪声图表描绘线的单位总是  $V/\sqrt{Hz}$ 。

将计算两种类型的噪声——输出噪声和等效输入噪声。计算等效输入噪声将对噪声和输入信号或输入噪声进行比较。输入噪声代表了输入端的噪声水平，将在输出端产生噪声，考虑电路在特定频率的增益。

放置探针到左轴将显示输出噪声，而放置探针到右轴将显示等效输入噪声。

噪声分析可能会产生特别小的值（在纳伏在数量级），因此，可以选择用 dB 来显示结果。

#### 警告：

正确噪声模型的电路调用 IC 宏模型是没有任何保证的，因为这些模型可能使用线性控制源仅仅是去模拟器件的基本行为。

### 7.9.2 计算方法

像通常一样，先计算电路的工作点，然后修改电路模型以正确地合计噪声分量。在噪声分析过程中（除了计算工作点时），除了输入参考激励源，其它所有的激励源都将被忽略。因为没有办法知道，在分析完成后，将查看哪个探针，PROSPICE 仿真器将为电路中的每个电压探针计算热噪声，包括与激励源和录音机有关的探针。不支持噪声电流，因此 PROSPICE 将忽略所有的电流探针。在

噪声分析过程中，每个探针都将进行单独的仿真，因此仿真的时间大约与放置的电压探针数量成正比。

### 7.9.3 使用噪声分析图表

当使用噪声分析图表时，要记住外部电磁场引起的噪声是没有模型的，不能进行仿真。在很多情况下，输入端的噪声是最关键的，对系统的性能有决定性的影响。

如果使用录音机把电路分割成几块，必须很清楚一点，就是 PROSPICE 仿真器只考虑电流的分割。孤立的噪声图虽然也很有用，但必须去掉录音机，才能观察整个电路的等效噪声。

为了确定电路的噪声源，系统自动把独立的噪声电压被记录仿真日志中。轮流考虑每个探针，然后给出每个元件的噪声分量。这些分量通过计算得到，并以平方值的形式显示出来。这个过程在开始和结束频率上进行。如果电路中有大量的探针，将会得到大量数据。

#### 进行噪声分析的步骤：

1. 绘制噪声分析图表，并进行编辑，设置频率范围和输入参考激励源。
2. 添加电压探针到电路的输出端或者其它节点，然后把它们拖进图表中。
3. 按下空格键去调用 PROSPICE 仿真器。
4. 如果要降低噪声电平，检查仿真日志 (CTRL+V)，确定噪声源。

## 7.10 失真分析

### 7.10.1 概述

失真分析确定了被测试电路产生的失真谐波的电平。这些失真谐波是单频率信号或互调信号的 2 次和 3 次谐波。

失真是由电路传输函数的非线性引起的，如果一个电路只是由线性器件（电阻、电容、电感、线性控制源），是不会产生任何的失真的。SPICE 失真分析图表可以对二极管、双极型晶体管，JFET 和 MOSFET 的模型的失真情况进行分析。

#### 警告：

正确失真模型的电路调用 IC 宏模型是没有任何保证的，因为这些模型可能使用线性受控源仅仅是去模拟器件的基本行为。

类似的信息可以通过使用傅立叶分析得到，但失真分析也可以显示当基频变化时，失真是怎样变化的。

### 7.10.2 计算方法

本类分析是基于电路中器件的小信号模型的，因此，第一步要计算电路的工作点。每一个非线

性器件的模型随着输入基频得到的分量，对相应的谐波产生复杂的失真。这些谐波在输出信号中的程度将决定图表中的值。这个过程会在指定的频率范围内不断重复。事实上，本类分析数学方法特别复杂，调用了泰勒级数的构造和处理过程，去表达器件的非线性。

注意，因为这里使用了复数，所以分析产生了每个谐波的幅度和相位信息。

对于单频率的谐波失真，图表中将绘制两条曲线，一条是 2 次谐波的，另一条是 3 次谐波。

对于互调失真，使用了两个输入频率，在基频（F1）和第二个频率（F2）站定一个比例参数。图表中将显示三条曲线  $F1+F2$ ， $F1-F2$  和  $2F1-F2$ ，显示互调的结果。

选择比例  $F2/F1$  时要小心，否则会出现奇怪现象。例如：当  $F2/F1$  为 0.5 时， $F1-F2$  的值是  $F2$ ，而  $F1-F2$  的曲线图将变得没有任何意义，因为与第二个基频相同。通常，应该选择一个无理数，如  $F2/F1=49/100$ ，可能会更好点。

**注意：**

这里有一个约束条件： $F2/F1 < 1$ 。

## 7.10.3 使用失真分析图表

失真分析图表在左轴显示的是谐波的幅度，右边显示的是相位。测试频率(F1)绘制到 X 轴上。

对于谐波失真分析（单输入频率为 F1），会为每个探测点绘制两条曲线，分别是  $2F1$  和  $3F1$ 。

对于互调失真分析（两个输入频率（F1 和 F2）），会为每个探测点绘制三条曲线，分别是  $F1+F2$ 、 $F1-F2$  和  $2F1-F2$ 。

在上面两种情形中，可以把鼠标指向相应的曲线，而把曲线分辨出来。当前所指的曲线的信息会显示在状态栏的右边。

## 7.11 傅立叶分析

### 7.11.1 概述

傅立叶分析是把时域数据转换为频域数据的过程，转换的结果类似于把示波器用频谱分析仪代替。在分析信号的谐波成分时非常有用，可以用来寻找特殊部分的失真，当然还有其它诸多用途。

### 7.11.2 分析方法

傅立叶分析先执行瞬态分析，然后对结果数据进行快速傅立叶变换。这个过程调用了在时域进行数据的离散采样的命令，即依据奈奎斯特准则进行。简单来说，就是可以观察到的最高频率是采样频率的一半。然而，可能会出现其它令人误解的结果，因为，容易把采样频率与输入信号的谐波频率混淆，谐波频率高于采样频率的一半。为了使这种可能最小化，可以对输入 FFT 的数据使用各种类型的窗口滤波。

## 7.11.3 使用傅立叶分析图表

在这个例子里，需要先绘制一个电路图，与用于瞬态分析时的一样，除了把瞬态分析图表用傅立叶分析图表代替。当然，也可以同时使用两种图表。

**使用傅立叶分析分析信号的频率成分：**

1. 像瞬态分析中一样绘制电路图。
2. 绘制傅立叶分析图表，拖曳探针到图表中。
3. 选择开始和停止时间、频率/分辨率的值以适合所分析的信号。如果可能，选择时间间隔（time interval）和频率分辨率对应到所分析信号的基频。
4. 按下空格键调用 PROSPICE 进行仿真。

## 7.12 音频分析

### 7.12.1 概述

Proteus VSM 集成了一系列的特性让你可以“听”到你的电路的输出结果（当然你必须安装了声卡）。主要的部件是音频分析图表。它与模拟分析基本上是一样的，只不过在模拟分析以后，会从分析结果的时域数据中生成一个 Windows 的 WAV 文件，然后用声卡播放出来。

WAV 文件可以导出给其它应用程序使用。

### 7.12.2 分析方法

音频分析与模拟分析基本上是一样的，只不过在模拟分析以后，数据以标准的 PC 采样率（11025、22050 或 44100Hz）进行重新采样，然后调用标准的 Windows 函数写成 WAV 格式的数据，最后调用命令去播放 WAV 文件。

### 7.12.3 使用音频分析图表

在这个例子中，需要先绘制一个电路图，与用于瞬态分析时的一样，除了把瞬态分析图表用音频分析图表代替。当然，也可以同时使用两种图表。

**进行音频分析的步骤：**

1. 绘制电路图。
2. 绘制音频分析图表，拖拉电路输出端的探针到图表中

3. 选择开始、停止时间和循环时间，以产生合理的波形长度，并且符合真实仿真的最小值。通过分析 1 毫秒的数据，然后循环 1000 次，建立 1 秒时间的音频比完全分析 1 秒的数据要快很多。
4. 选择采样分辨率和速率，以适合信号的特性。建议使用 16 位的分辨率，否则你会产生一个大文件，导致磁盘空间不足。大多数的 PC 机的声卡并不会因为使用 44.1KHz 的采样率而声音得到改善。所以完全没必要使用 44.1KHz 的采样率。
5. 按下空格键调用 PROSPICE 进行仿真。
6. 按下 CTRL-SPACE 可以进行重放，而不进行重新仿真。
7. 可以导出音频数据到 .WAV 文件中，如果需要，可以导入到其它设计图中（通过音频激励源）。

## 7.13 交互式分析

### 7.13.1 概述

交互式分析图表集合了交互式仿真和图表分析的优点。交互式分析在交互式仿真环境下进行，但它可以把结果记录，并像瞬态分析图表一样显示出来。这种分析方法在检查某个控制操作时非常有用，可以把它想像成是存储示波器和逻辑分析仪的集合体。

### 7.13.2 计算方法

计算方法与混合模式瞬态分析的完全一样，只是仿真分析是在互动的模式下进行。因此，电路中开关、键盘和其它可动作器件的操作会影响结果。仿真会依照动画时间步长（Animation Timestep）定义的速度进行，而不是使用可以达到的最大速度。

必须清楚，此类仿真会捕捉到大量的数据。如果处理器使用真实的速度运行，每秒会产生数百万的事件，将这些事件显示在图表中，将占用几兆的存储量，特别是当你探测数据或地址总线的时候，将更加明显。如果 ISIS 载入的数据超过 20 或 30MB 时，可能很快就让系统崩溃。

如果在相对短的时间里，不可能捕捉到需要的数据，最好使用逻辑分析仪进行分析。

像正常的互动仿真一样，不支持电路的多部分仿真，所有的没有设置成播放模式的录音机对象将会自动从电路中移除。

### 7.13.3 使用交互式分析图表

通常你会使用互动分析图表去测试一个在进行互动仿真时，发现当进行某部分的控制操作时，发生了奇怪的事情。在下面这个例子里，你可以使用虚拟仪器去观察，但是在某些情形下，有必要捕捉结果，用图表显示出来，然后再慢慢研究它。

**进行互动分析的步骤：**

1. 添加探针到节点。
2. 绘制互动分析图表，把探针拖进图表中。
3. 编辑图表属性，选择合适的开始和停止时间。注意，PROSPICE 为了减少数据量将不会在开始时间之前捕捉数据。
4. 在原理图中设置所有的互动控制到合适的初始状态。
5. 准备进行任何的互动操作，然后按下空格键。你必须在第 4 步设置的时间内操作动态器件。如果很难做到，你既可以增加停止时间，也可以减少每帧的时间步长（Timestep Per Frame）的设置项。

## 7.14 数字一致性分析

### 7.14.1 概述

一致性分析把一组数字仿真结果与另外一组进行比较。这是对修改前后的两个电路的工作是否一致的快速检查方法，用来证明修改后的电路没有出现意想不到的副作用。这种分析方法特别适用于基于微控制器的电路，当固件程序改变后，需要对整个电路进行重新测试。

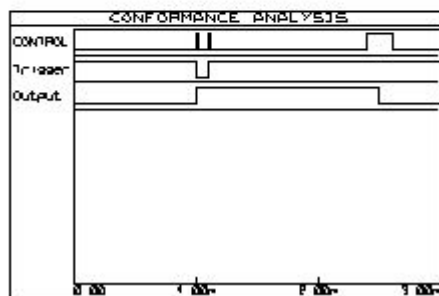
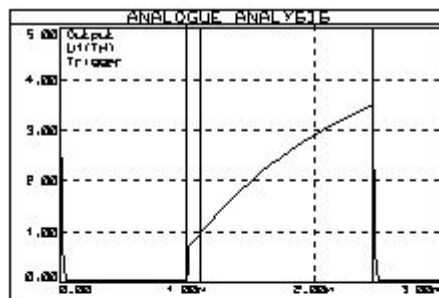
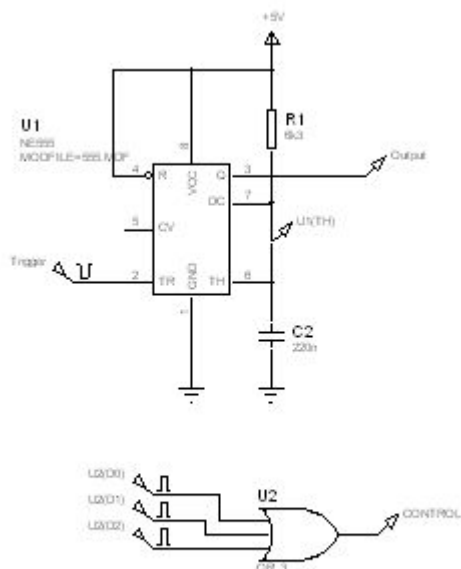
### 7.14.2 分析方法

除了能把两组结果数据存储在图表中，其它的计算方法与数字瞬态分析一致。这里把这两组数据称为测试数据和参考数据。

一致与不一致是通过在第一条描绘线的每个边沿对测试数据和参考数据进行比较得到的。这里把这条描绘线称为控制线，它以不同的颜色显示在图表中，以区别于其它描绘线。更加重要的是，测试数据和参考数据的边沿不必要一致。这意味着在结果数据中的绝对时间的改变并不是不一致的体现。这特别是在基于微处理器的电路中，任何代码的改变都会影响到事件的绝对时间。在这种情况下，控制线可以由代码来生成，从被测试程序的入口到出口。

### 7.14.3 使用数字一致性分析图表

通常，一致性分析用于测试策略的一部分，可用于测试仿真模型，但最常用于测试嵌入式系统。在下面这个简单的例子里，将详细描述使用一致性分析图表测试 555 单稳态触发器电路的操作过程。



被测试电路包含了 U1、R1 和 C2，上图显示了经典的 555 单稳态触发器的连接图。电路在 1ms 的时候被数字脉冲激励源触发，输出波形显示在上图的模拟瞬态分析图表中。为了测试电路是否正确工作，先考虑以下几个因素：

- 输出在触发脉冲前是低电平的。
- 输出在触发脉冲跳到低电平后，很快将跳到高电平。
- 输出在触发脉冲回到高电平后，仍然维持在高电平。
- 输出维持在高电平大约 1.5ms。

在经历上述时间后，输出回到低电平。

为了通过一致性分析图表达达到上述结果，需要在触发脉冲信号的每个边沿转换时，和输出信号的每个边沿对输出信号进行采样。可以通过为最后两国采样点选择间隔，而为单稳态触发器的延时时间设置一个容差。

可以通过一系列的数字脉冲激励源，把它们的输出通过或门连接在一起，而达到这个效果。每个脉冲激励源的脉冲宽度决定了每个意料中的事件的时间容差。例如，第三个脉冲激励源设置在 2.4ms 到 2.6ms 之间触发，将给输出脉冲提供  $\pm 100\mu s$  的时间容差。

注意，可以使用数字时钟或模式激励源去产生控制信号；使用简单的时钟信号将满足大多数的应用，它会在有规律的时间间隔进行检验。

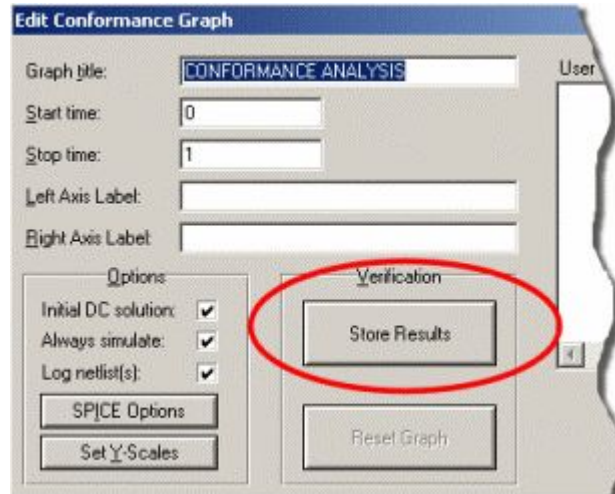
通常的操作过程总结如下：

### 进行一致性分析的步骤：

1. 确定想进行测试的部件和时间。
2. 设计一个测试原理图，可以产生想要检验的输出和控制信号。
3. 绘制一致性分析图表，像数字分析图表一样进行设置，并确保控制信号轨迹是在图表的最顶端。

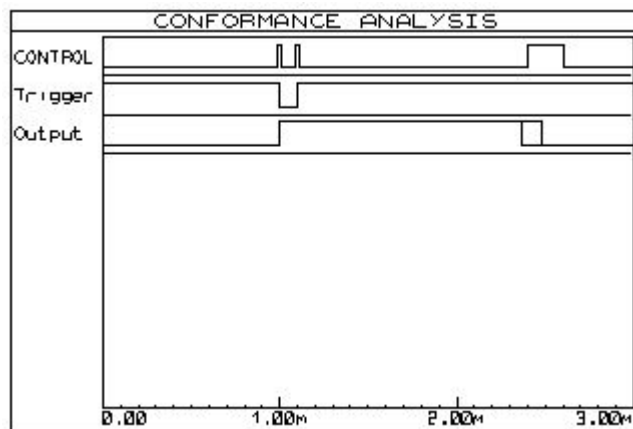


4. 运行仿真，检验结果是不是像意想的那样，控制轨迹的转变时间是不是在想要的时间点上。
5. 编辑一致性分析图表，按下“保存结果”按钮，这将会把当前显示的结果变为参考结果数据。这将以暗淡的颜色显示，当重新进行仿真时，图表将对任何新的结果与它进行比较。



6. 保存设计，它将用来检验新数据。

假设因为元件的缺乏，R1 和 C2 的值必须要改变。如果把 C2 改为 100nf，R1 改为 15k，这个电路还能像以前一样工作吗？实验的结果如下图所示：



显然，答案是否定的，输出脉冲变短了，在仿真日志（simulation log）中可以找到以下信息：

Comparing Results...

Data mismatch at time 2.40m in trace 'Output'.

Data signatures were 'L' and 'H'.

Conformance analysis FAILED.

注意：图表的光标指向了第一个出现差异的时间 2.4ms

事实上，有三种方法可以调用一致性分析：

使用通常的方法重新仿真图表，例如：通过按下空格键或者使用“Graph”菜单中的“Simulate Graph”命令。

使用“Graph”菜单中的“Conformance Analysis command”命令，这将对设计中的所有一致性图表重新仿真，然后报告任何的错误。

高级用户使用一使用命令行，输入：

**ISIS <filename> /V**

将像第二种方法一样进行整体一致性分析。如果有任何一个图表仿真失败，ISIS 将显示错误 level 1。这允许通过批处理文件自动进行多个测试。

# 第 8 章激励源和探针

## 8.1 激励源

### 8.1.1 概述

激励源是一种用来产生信号的对象。有很多类型的激励源，每一种都产生不同种类的信号。

激励源	描述
DC（直流）	产生恒定的直流电压源。
Sine（正弦）	正弦波信号发生器，可以对幅度、频率和相位进行设置。
Pulse（脉冲）	模拟脉冲信号激励源，可以对幅度、周期和升降时间进行设置。
Exp（指数）	指数脉冲信号激励源，产生的波形与 RC 充放电电路相同。
SFFM（单频率调频）	单频率调频信号激励源，产生的波形与一个正弦信号对另一个正弦信号进行频率调制得到的波形一致。
Pwlin（分段线性）	分段线性激励源产生任意形状的脉冲或其它信号。
File（文件）	跟上面的激励源一样，但波形数据是从 ASCII 文件输入的。
Audio（音频）	使用 Windows 的 WAV 文件作为输入的波形信号。这在与音频分析图表一起使用时，将非常有用，可以听到电路的输出信号。
Dstate（稳态）	输出稳态逻辑电平。
Dedge（单边沿）	单个逻辑电平转换或边沿。
Dpulse（单脉冲）	单个数字时钟脉冲。
Dclock（数字时钟）	数字时钟信号。
Dpattern（图形）	逻辑电平的任意序列。

### 8.1.2 放置激励源

步骤：

1. 选择“Generator”图标，激励源类型列表将出现在对象选择器中。
2. 选择激励源类型，预览窗口将出现相应激励源的图形。
3. 使用旋转和镜像按钮调整激励源的放置方向。
4. 鼠标放到编辑窗口中，按下左键并拖拉激励源到合适的位置，然后释放鼠标左键。

提示：

可以直接把激励源的连接点放置到连线上，也可以先放在空的地方，以后再连线。

当激励源没有连接到任何网络时，它将得到一个默认的名称（?），以表示它没有进行标注。当它连接到一个网络（把它直接放置到连线上）时，它将以所连接的网络名进行标注。如果该网络没有标注，它将以元件的参考名称和第一个连接到该网络的引脚名称的组合进行标注。激励源的名称会根据所连接的网络自动进行更新。也可以编辑激励源的属性，给激励源命名，在这种情况下，激励源的名称不会自动更新。

### 8.1.3 编辑激励源

所有的激励源都可以使用 ISIS 通用的编辑方法进行编辑，最简单的方法是单击右键（选择）然后再单击左键（编辑），或者用鼠标指向激励源然后按 CTRL+E。

激励源编辑对话框提供了一系列的通用属性和各激励源的特有属性。通用属性的解释如下：

属性	描述
Name（名称）	激励源的名称。 如果是手动输入的名称，ISIS 将不会对它进行自动更新。 如果希望激励源名称会自动更新，打开属性对话框，去掉手动输入的名称，单击 OK 确定。
Type（类型）	激励源的类型。 激励源的类型改变，对话框右边的属性选项也跟着改变。
Current Source（电流源）	除了数字激励源，其它激励源都可以作为电压源或电流源使用。勾选此选项将使目标激励源成为电流源。
Isolate Before（隔离之前的网络）	当激励源放置到连线的中间时，本控制选项用于控制是否把连线打断，分成两个网络，即只驱动激励源所指的方向。相反方向上不接激励源。
Manual Edits（手动编辑）	选中此选项后，激励源的属性以文本方式显示，这主要是为了方便软件向前兼容。然而，高级用户也可以使用属性表达式来设置属性，这种方法只有在手动编辑模式下是可行的。

### 8.1.4 直流激励源

直流激励源用于产生恒定的模拟直流电压或电流。它只有一个属性，用于设置输出值。

### 8.1.5 正弦激励源

正弦激励源用于产生固定频率的连续正弦波形，它有几个属性：

- 输出值（Output level）指定为峰值振幅（VA）和可选的偏移电压（VO）。幅度也可以以有效值（RMS）或峰峰值的形式表示。
- 频率可以使用 Hz（FREQ），或周期（PER），或总的周期数来表示。

- 可以设定相位偏移，用度表示（PHASE），或者使用时间延迟（TD）。在后一种方式中，到达指定时间后才会输出正弦波（开始振动）。
- 可以通过阻尼因数（THETA）设置开始振动后的波形的指数式衰减。

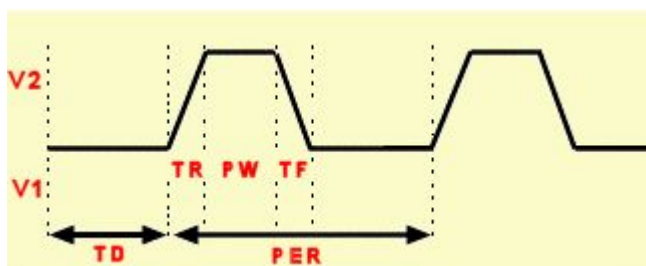
正弦波的输出公式为：

$$V = VO + VAe^{-(t-TD)THETA} \sin(2\pi FREQ(t + TD)) \quad (t \geq TD)$$

$$V = VO \quad (t < TD)$$

## 8.1.6 脉冲激励源

脉冲激励源用于产生各种在模拟分析中使用到的周期性信号。如矩形波、锯齿波和三角波等，也可以产生单个短的脉冲信号。必须注意，上升时间和下降时间不能为 0，所以不可能产生一个真正的矩形波。因为在 PROSPICE 中，不允许有跳变。下图描述了脉冲激励源的工作情况：

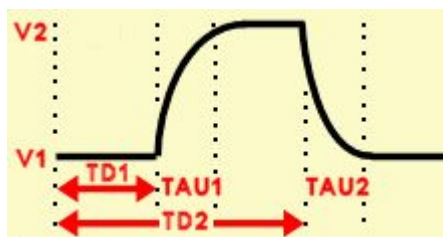


这里：

符号	描述
PER	波形的周期，如果没有指定，将使用 FREQ 的参数。
FREQ	波形的频率，在瞬态分析中默认为一个周期。
V1	输出的低电平值。
V2	输出的高电平值。
PW	每个周期中，输出为 V2 的时间。不包括 TR 和 TF。
TR	上升时间——每个周期中，输出电平从低到高所花的时间。默认为 1ns。
TF	下降时间——每个周期中，输出电平从高到低所花的时间。默认与 TR 相同。
TD	延迟时间。激励源输出电平开始为 V1，然后保持 TD 秒的时间。

## 8.1.7 指数激励源

指数激励源产生的波形与 RC 电路在充放电情况下产生的波形是一样的。参数描述如下图所示：



符号	描述
V1	输出的低电平值。
V2	输出的高电平值。
TD1	上升曲线的开始时间。
TAU1	上升曲线的时间常数。这个时间大约等于达到 0.63 满电压值时的时间。
TD2	下降曲线的开始时间。注意，TD2 是从 0 开始计算的。
TAU2	下降曲线的时间常数。

数学上，波形由以下三部分定义：

0 到 TD1  $V1$

TD1 到 TD2  $V1 + (V2 - V1)(1 - e^{\frac{-(t-TD1)}{TAU1}})$

TD2 到 TSTOP  $V1 + (V2 - V1)(1 - e^{\frac{-(t-TD1)}{TAU1}}) + (V1 - V2)(1 - e^{\frac{-(t-TD2)}{TAU2}})$

## 8.1.8 单频率调频激励源

单频率调频激励源产生的波形是一个正弦波对另一个正弦波进行频率调制的结果波形。数学上，定义如下：

$$V = VO + VA \sin(2\pi FCt + MDI \sin(2\pi FSt))$$

参数解释如下：

符号	描述
VO	直流偏转电压。
VA	载波幅度。
FC	载波频率。
FS	信号频率。
MDI	调制系数。

## 8.1.9 分段线性激励源

分段线性激励源用于产生复杂的模拟信号，或者重现测量过的波形。输出波形使用时间或输出幅度来描述。激励源的输出值在规定时间间隔内是线性的。

分段线性激励源对话框由曲线图表编辑器组成，在上面可以拖拉放置数据点。操作简单描述如下：

- 单击左键放置新的数据点。
- 使用左键拖拉数据点，可以移动数据点。
- 单击右键删除数据点。

须符合下面的限制：

- 在时刻 0 必须有数据点，虽然它的 Y 值可以任意改变。也即是，只能对时刻 0 的数据点进行拖拉，而不能删除。
- 如果试图创建垂直的边沿，图表编辑器将在两个数据点之间自动插入最小的上升/下降时间值。

如果有表列数据，使用手动编辑模式也许会更加容易。在这种情形下，每个数据点由  $V(t)$  属性定义。请看下面的例子：

$V(0)=0$

$V(2n)=0$

$V(3n)=1$

$V(5n)=1$

$V(6n)=0$

如果数据量非常大，使用文件激励源将更方便。

## 8.1.10 文件激励源

文件激励源产生的信号是由保存在 ASCII 文件中的一系列时间点和数据值定义的模拟信号，用来驱动电路。因此它将与分段线性激励源非常类似，除了它使用外部文件定义数据点，而不是使用器件的属性来定义数据点。

它的对话框只有一个字段，定义了数据文件的名称。这些文件没有默认的扩展名，而且文件应该与设计文件保存在同一个目录下，否则要输入整个路径。

### 数据文件格式

ASCII 文件的格式定义为：每一行定义一个时间和电压对，之间用空格或 tab 键隔开，但不能用逗号。时间值必须是逐渐增大的，而且必须是简单的浮点数（不能使用后缀）。

### 例子：

下面的示例数据文件定义了三个周期的锯齿波，上升时间为 0.9ms，下降时间为 0.1ms，幅度为 1V。

0	0
9E-4	1
1E-3	0
1.9E-3	1
2E-3	0
2.9E-3	1
3E-3	0

### 8.1.11 音频激励源

音频激励源使用 Windows 的 WAV 文件驱动电路。与音频分析图表一起，可以听到仿真电路处理声音信号的结果。

- 文件名具有默认的扩展名 WAV，文件应该与设计文件保存在同一个目录下，否则要输入整个路径。
- 幅度可以通过最大的幅度绝对值来设定，也可以通过峰峰值进行设置。
- 可以设置直流偏移电压。如果此电压为 0，则输出电压将在 0 周围震荡。
- 对于立体声 WAV 文件，可以选择播放的通道，或者把数据作为单声道的对待。

### 8.1.12 数字激励源

有五种子类型的数字激励源：

单边沿——从低到高或从高到低的单调信号。

单脉冲——在相反方向上的一对信号转换，共同形成一个正的或负的脉冲波形。既可以设置每个边沿的时间（开始时间和停止时间），也可以设置开始时间和脉冲宽度。

时钟——一个连续的脉冲间隔相等的脉冲队列。可以设置开始值、第一个脉冲边沿到达的时间、周期或者频率。周期定义的是整个周期的时间。

图案——这是最灵活的激励源，事实上可以产生所有其它类型的信号。图案激励源由以下参数定义：

符号	描述
Initial State 开始状态	在 0 时刻的电平值，在混合信号仿真过程中，用来寻找电路的工作点。
First Edge 第一个边沿	图案激励源真正开始的时间，输出在此时间之前将一直保持在开始状态。
Timing 时间	图案的每一步可以使用相同的时间（通过勾选 Equal mark/space timing），也可以为高低电平设置不同的时间。在这种情况下，脉冲宽度(Pulse width)定义逻辑“1”的时间值，而间隔时间(Space time)定义逻辑“0”的时间值。
Transitions 转换	输出类型可以是持续重复输出图案，直到仿真结束的。也可以是达到固定数量边沿转换后，自动停止的。
Bit Pattern 位图图案	默认的模式是简单的高低序列。另外，也可以设定图案字符串。图案字符串可以包含以下字符： 0, L—输出波形为强的低电平（注意使用大写的“L”）。 1, H—输出波形为强的高电平（注意使用大写的“H”）。 l—输出电平为弱的低电平（注意使用小写的“l”）。 h—输出电平为弱的高电平（注意使用小写的“h”）。 F,f—输出电平为浮动电平。

脚本——激励源由数字 BASIC 脚本控制。在脚本文件中通过声明一个与激励源参考名相同的



PIN 变量，就可以访问该激励源。

## 8.2 探针

### 8.2.1 概述

探针用于记录所连接到的网络的状态。有两种类型的探针：

- 电压探针——可用于模拟仿真和数字仿真中，在模拟仿真中，它记录了真实的电压值，而在数字仿真中记录的是逻辑电平和强度。
- 电流探针——只能用于模拟仿真中，测量的方向由电流探针上的箭头表示。

**警告：**不能把电流探针放到数字仿真中，或者放置到总线上。

探针最常用于基于图表的仿真中，但也可用在交互式仿真中，用来显示工作点的数据和分割电路。

### 8.2.2 探针放置

**步骤：**

1. 选择电压探针或电流探针图标。在预览窗口将见到探针的图形。
2. 使用旋转和镜像按钮确定探针的方向。  
注意：电流探针的方向非常重要，系统是根据电流探针上的箭头来测量电流的。
3. 把鼠标移到编辑窗口，按下鼠标左键并保持不放，拖拉探针到合适的位置，然后释放鼠标左键，把探针放到连线上。

既可以直接把探针放置到连线上，也可以先放置探针，然后再连线。

当探针没有连到网络上时，它默认的名称是一个问号（?），表示它没有被标注。当探针连接到某一个网络时，它将标注成该网络的名称，如果该网络也没有标注，探针的名称将是最接近的元件的名称和连接到该网络的引脚名的组合。如果以此种方式命名，在更改网络的时候，探针名将会自动更新。也可以给探针手动命名，在这种情况下，探针的名称是固定的。

### 8.2.3 探针设置

通过编辑探针对话框，可以调整探针的两个参数：

**负载电阻（Load Resistance）**

电压探针可以在原理图中设置一个负载电阻，当探测点对地没有直流通路时，这项设置非常有用。

**记录文件名（Record Filename）**

电压探针和电流探针都可以把数据记录到文件中，然后通过录音机激励源（Tape Generator）把记录的数据重新播放出来。这个特性可以让你把一个电路的输出波形记录到文件中，然后作为另一个电路的输入播放出来。

请参考录音机和分区章节以获得更多信息。

## 第 9 章脚本化信号源

### 9.1 前言

PROSPICE 包含了支持可编程的脚本语言，该语言可以用于编写复杂的测试信号。我们命名该语言为“EasyHDL”，因为它比学习通用的硬件描述语言像 Verilog 或 VHDL 更容易些。不过，尽管其相对简单，但它却可以用来模拟及数字波形，并且可以用来产生复杂的测试向量，通过这种方式，在原理图上，一个脚本可以规定多个信号源对象的特性。

EasyHDL 是轻微的基于 BASIC 编程语言的，不过它添加了另外的专门用于规定测试向量的特性。本章的其余部分将向你展示如何创建 EasyHDL 脚本，及如何在电路原理图上连接它们到一个或

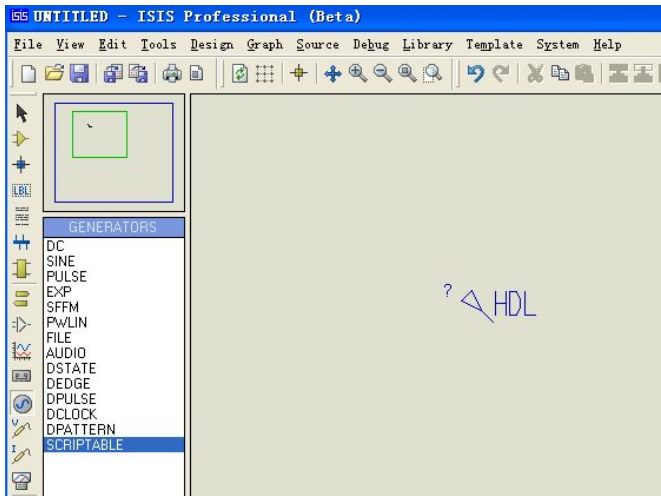
多个脚本化信号源对象上。不过本手册并没有给你一个关于 BASIC 语言的详细解释， 除非在以下情况下才解释 EasyHDL 提供的特殊功能，即为了成为一个有用的仿真模型和控制语言。对标准 BASIC 关键字的描述是简洁的且只限于解释语言的语法。

EasyHDL 脚本对一个特殊的脚本化信号源对象是局部的，或为了规定一个复杂的测试向量集，它们可以被放置到原理图上，然后同多个脚本化信号源建立联系。

## 9.2 创建、编辑及调试 EasyHDL 脚本

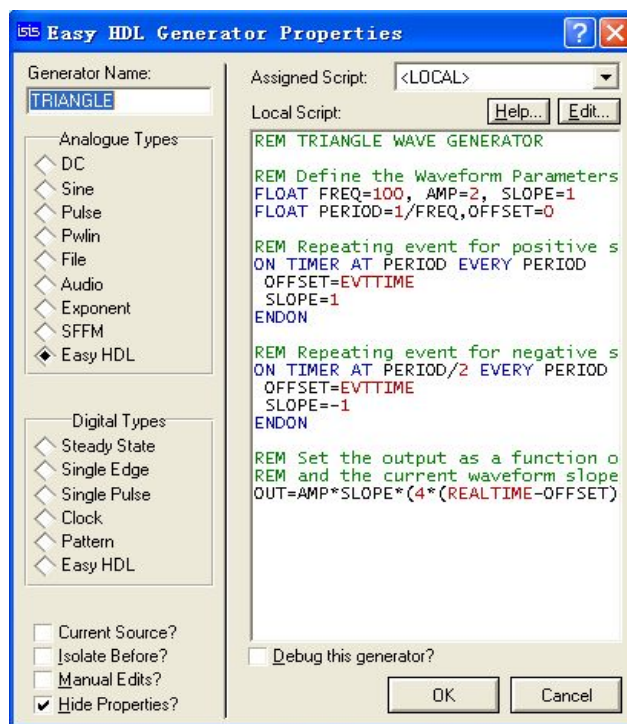
为了对一个脚本化信号源产生局部的 EasyHDL 脚本：

1. 选择一个脚本化信号源图标放置一个脚本化信号源，或单击鼠标右键编辑一个已经存在的脚本化信号源（具体方法见下图）。



放置一个脚本化信号源图示

2. 设置脚本化信号源类型为 Easy HDL – 依据你想创建的脚本化信号源类型，它能够是模拟或数字的。
3. 输入脚本到本地脚本编辑域中，可选的，你也可以点击“Edit”按钮在外部的文本编辑器中编辑脚本。你也可以通过菜单上的“Source/ Setup External Text Editor”命令，规定哪一个文本编辑器被使用（下图示出了其属性编辑界面）。

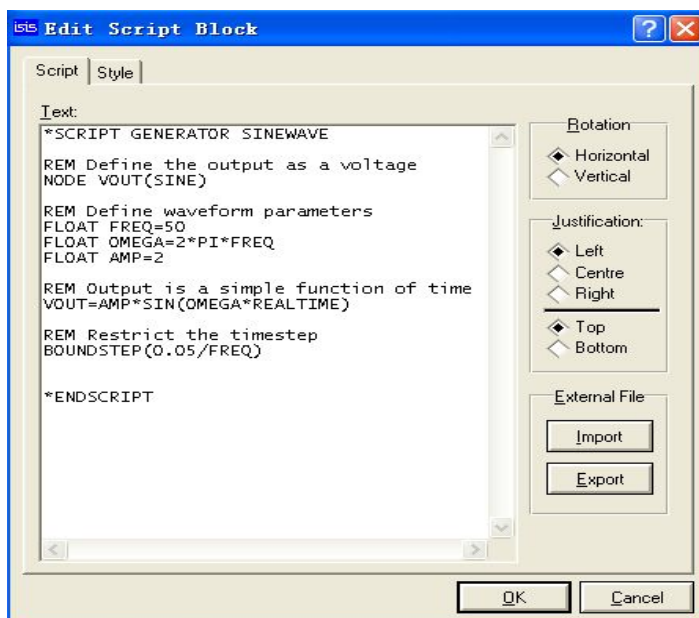


脚本化信号源属性编辑窗

输出引脚被自动指定名字为“OUT”，假如你在第2步规定为模拟信号源，那么“OUT”将是一个节点变量，假如你在第2步规定为数字信号源，那么“OUT”将是一个引脚变量。

为了在原理图上产生一个 EasyHDL 脚本：

1. 选取脚本图标然后在 ISIS 绘图域中点击鼠标左键放置一个脚本对象。则将弹出编辑脚本对话框。其界面如下图所示



在 ISIS 中查看“SCRIPTS”的有关放置及编辑文本脚本的常用指令。

2. 脚本的第一行必须是（在“text”域中）：

**\*SCRIPT GENERATOR script\_name**

关键词“**SCRIPT**”指示在脚本块中的文本将被包含在网表中，同时关键词“**GENERATOR**”告知 PROSPICE 有关脚本的类型（为了连接的目的），在“**GENERATOR**”关键词之后的名字是由你提供的确定脚本的名字。任何顺序的阿拉伯字符及下划线('\_')都可以被使用。空格或任何其他字符都被认为是脚本名的结束。

3. 在脚本的最后一行必须是：

**\*ENDSCRIPT**

这通知 PROSPICE 中的网表装载机脚本的结束位置。假如在网表编辑中没有发现它 ISIS 将报告错误。在之后的任何行将被忽略。

4. 在开头的“**\*SCRIPT...**”及结尾的“**\*ENDSCRIPT**”中间输入你的脚本，你必须声明发生器为“**NODE**”变量或“**PIN**”变量。

5. 当你做完成时，关闭脚本对话框窗口。

为了连接原理图中基于脚本的信号发生器对象：

1. 选择一个脚本化信号源图标放置一个脚本化信号源，或单击鼠标右键编辑一个已经存在的脚本化信号源。

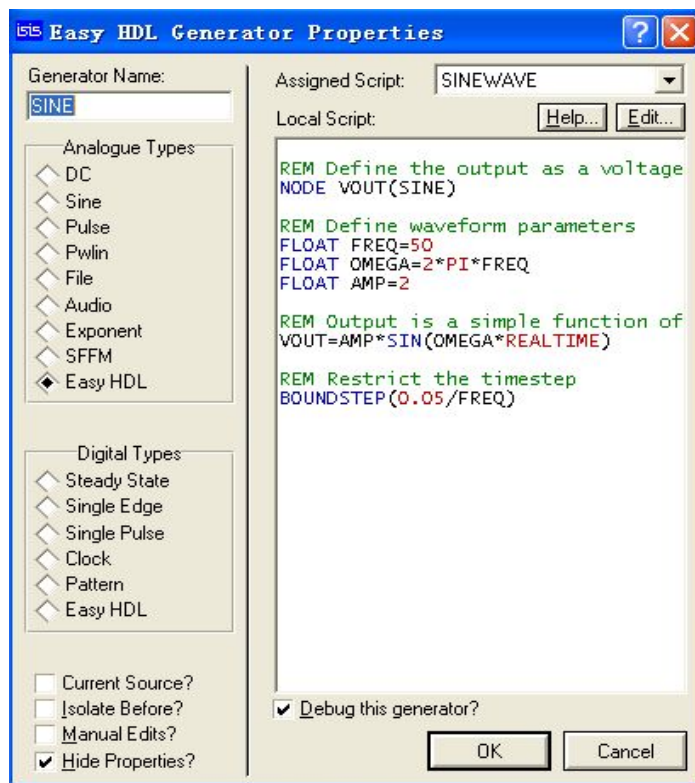
2. 设置脚本化信号源类型为 Easy HDL – 依据你想创建的脚本化信号源类型，它能够是模拟或数字的。

3. 使用指定的脚本选择器规定将要驱动信号发生器的脚本。发生器的名字必须同在控制脚本中的“**NODE**”或“**PIN**”的变量相匹配。

在仿真过程中调试发生器脚本：

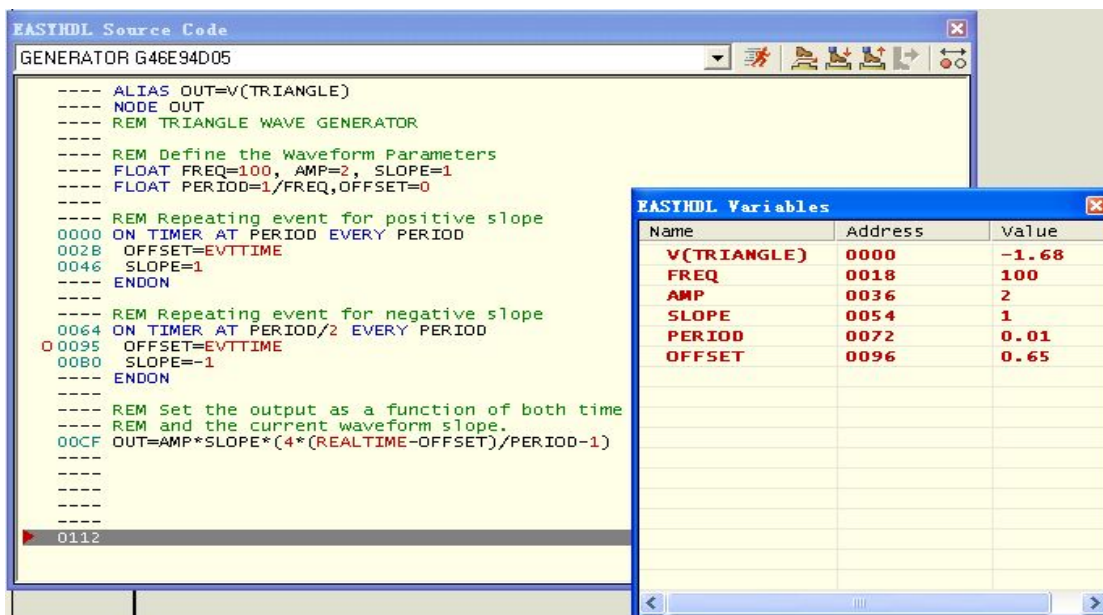
1. 以常规方法放置脚本化信号源，设置信号源类型“EasyHDL”并写脚本。

2. 检查对话框窗口中“**Debug this generator**”的选项使之处于调试之中(见下图所示)。



选中“Debug this generator”选项图示

3. 暂停仿真以观察脚本的源代码及变量。



暂停仿真时弹出的代码窗及变量窗

4. 在代码框中，设置断点然后仿真运行（在源代码窗的问题行中双击可设置断点）。
5. 当断点被命中时，仿真将暂停，你可以常规的方式单步运行脚本，监视变量的变化等。

规定一个 EasyHDL 三角波脚本发生器被激活用来仿真。

从用户接口的视角来看，调试一个 EasyHDL 接口是完全等同于在 Proteus VSM 中调试一个微控制器固件。假如你对这个基本的操作不能明确的话，我们建议你在这点上可以参考上述的调试（微控制器的调试）。除此之外，你必须牢记在大脑里的有一点点不同的是：

- EASYHDL 描述执行 0 时间。当你单步执行 EASYHDL 代码时，电子仿真在每一步之后。这与微控制器的单步执行代码显著不同。在那里，每一步指令花费的时间有限，仿真接近每一步代码。

- 由于时间落后于每一个描述之后，任何在脚本中所做的指定在电路仿真时都不会立即看到，直到每一个脚本的最后后一个描述或 “a SLEEP” 或 or “WAIT” 描述到达时。

- 一旦仿真开始时，所有的脚本被立即执行。因此从脚本的开始处开始可以到达的任何断点在电路的操作点被计算以前都可以到达。这可能导致不正常的电压及逻辑状态显示在电路上。在这种情形下，操作点将在每一步之后计算，直到脚本的结束为止。当脚本被定时器或其它调用事件调用时调试才唯一激活。

- 当一个模拟脚本在正常的模拟时间点被调用时，断点及单步并不使能。当脚本被定时器或其它呼叫事件调用时调试才唯一激活。

几个简单的“EasyHDL scripts”可以在你的 Proteus 被安装后的样本例子目录下的“Generator Scripts ”子目录中发现。它们的范围覆盖了从一个简单的正弦波到一个完整的 QPSK 调制器等。它们是值得很好研究的。

## 9.3 程序结构

EasyHDL 由声明、指定及命令组成。声明用来声明变量的名字及可能的别名，指定用来指定表达式对变量的结果，命令用来控制程序脚本的流向。

每一个声明或命令有一个或多个关键字与之相联系，不过并不是所有的关键字都需要规定，也就是说，命令的某些部分是可选的，例如 FOR...TO...STEP...

**NEXT** 命令有四个关键字，在其中，只有关键字 FOR, TO 及 NEXT 是必须的 (STEP 关键字及紧跟其后的表达式是可选的。

某些命令有块语法及线语法，块语法容许几个命令在命令的开始及结束关键字之间被嵌套（见下面的例子）。线语法只容许其他的命令在同一行作为命令本身被规定。也没必要规定结束命令关键字，它假定命令行本身为结束行。下面的例子显示了 IF...ELSE...ENDIF 命令在块语法及线语法中的格式。

块语法格式：

```

IF i=0 THEN
    j=1      // IF... 块
    k=2
ELSE
    j=3      // ELSE... 块
    k=4
ENDIF

```

线语法格式

IF i=0 THEN j=1:k=2 ELSE j=3:k=4 // 不需要 ENDIF 作为结尾。

## 9.4 语法图解析

下面的术语及语法符号在随后的命令参考材料部分被使用，强烈建议用户在理解后面的参考材料时需要熟悉这些术语。

在命令参考部分的每一个命令，其语法以一个简单的“BNF”格式被给，解释如下：

1. 在语法定义及例子中，所有的命令关键字以“**courier**”字体表示，在语法定义内所有的命令关键字也处于加粗状态，以示命令关键字同一般文本描述的区别。例如：

**IF** expression **THEN** ...

2. 圆点('...') 指示一个或多个其他命令，由冒号分开

3. 方括号('[ ' 及 ' ]') 指示在括号内的文本是可选的。在那里，括号中的文字被嵌套，内部的文字只有在外部的被输入后才容许提供，也就是，假如你省略了外部的文字，那么你必须省略紧跟其后的括号内的所有内部的文字。例如：

**COMMAND** [[A[.B]] ]]

意指：假如你选择了可选的命令“A”，那么你可以选择使用可选的逗号及“B”，假如你没有选择命令“A”，则其后的任何命令你就不能选择。

4. 符号('|') 用来指示多个中的一个，互斥的选项可以输入，例如：

**COMMAND** A | B | C

意指：命令可以按照中的'A', 'B', or 'C'任何一个。



5. 字母 **etc** 用来指示前面参数的重复。在 **etc** 前面的特性意指，在前面的文本中，从那里，你可以重复参数，例如：

```
INT variable=expression[[,variable=expression]][[,etc]]
```

“etc ” 意指前面的参数 `[[,variable=expression]]` 可以无限的重复。

6. 字或短语以更底情形的 Times Roman 斜体表示，用于某些任意的表达式或变量，常常以文本的形式出现，其后紧跟语法定义，在下面的例子中：

```
IF test_expression THEN ...
```

短语 “test\_expression” 表示任意数字的表达式。

## 9.5 使用 EASYHDL 规定模拟信号

### 9.5.1 声明输出

模拟脚本化发生器的输出脚应该被声明为一个节点变量，例如：

```
NODE V(OUT)          // 单电压输出 Single voltage output

NODE V(POS,NEG)       // 差分电压输出 Differential voltage output
NODE I(OUT)           // 单点电流输出 Single ended current output
NODE I(POS,NEG)       // 电流输出 Current output
```

一个局部的发生器脚本包含了一个隐式声明， `NODE V(OUT)` 用于电压发生器， `NODE I(OUT)` 用于电流发生器。

### 9.5.2 程序流

当一个 EasyHDL 脚本用于规定一个模拟信号时， 整个脚本在每一个时间点，或每一次调用事件发生时都被从头到尾地执行。命令像 “SLEEP” 不容许，也假定程序将对所有的节点变量进行指定。

事件调用可以通过使用 “CALLBACK” 或 “TIMER” 命令来安排。这些命令在规定的时间内暂停一个模拟的时间点，脚本以一个相应的事件 ID 号方式被调用。

典型地，你可以使用数学公式规定在特殊的时间点输出，使用调用事件以规定的次数来改变某些参数，假如这听起来有点让人糊涂，下面的例子将澄清这一概念：

## REM TRIANGLE WAVE GENERATOR

REM Define the Waveform Parameters

FLOAT FREQ=100, AMP=2, SLOPE=1

FLOAT PERIOD=1/FREQ, OFFSET=0

REM Repeating event for positive slope

ON TIMER AT PERIOD EVERY PERIOD

    OFFSET=EVTTIME

    SLOPE=1

ENDON

REM Repeating event for negative slope

ON TIMER AT PERIOD/2 EVERY PERIOD

    OFFSET=EVTTIME

    SLOPE=-1

ENDON

REM Set the output as a function of both time

REM and the current waveform slope.

OUT=AMP\*SLOPE\*(4\*(REALTIME-OFFSET)/PERIOD-1)

当脚本在任意时间点被调用时，只有最后一行被执行。这一行规定了输出的形式为电压，它将按照“SLOPE”的值，随时间线形变化，升高或降低。“SLOPE”的值被交替地设为 1 或-1，它通过两个“ON TIMER”块实现，这两个“ON TIMER”交替地全或半周触发以便三角波被产生。

从上述的例子可以明显看到，变量在调用之间保持它们自己的值。不过，你必须小心的，不能假定在仿真时间内你的脚本被调用多少次。例如：

REM !!!BAD CODE!!!

FLOAT VOUT=0

VOUT = VOUT + 1.0

OUT=VOUT

这将产生不可预料的结果，因为 VOUT 的上升速率完全取决于由 SPICE 内核所选择的时间步长。

## 9.5.3 规定输出转换

你能够以以下两种方式中的一种规定模拟信号产生器的输出：

- 作为一个连续值。

OUT=vout

- 作为一个时间量化的转换。

OUT=vout AFTER time\_interval

在时间量化的转换中，输出将按照规定的时间间隔，从它的初始值到终止值线性上升。你也可以规定一个绝对的目的时间用于转换。

OUT=vout AT time

下面的例子，显示了输出转换如何用利用存储的数据来产生一个分段线性的波形。

REM PIECEWISE LINEAR GENERATOR

REM Define time/value pairs for the output

DATA 2m,5

DATA 3m,5

DATA 5m,0

DATA 0,0 // End of data

FLOAT PERIOD=10m

FLOAT v,t,td

TIME OFFSET=0

REM Reset the data pointer to

REM start a new cycle

ON TIMER AT PERIOD EVERY PERIOD

RESTORE

OFFSET=EVTTIME

ENDON

ON EVENT

READ t,v

```

IF t > 0
  t = t + OFFSET
  OUT=v AT t // 输出在时间“t”被设置为电压“v” The output is set to voltage v at time t
  CALLBACK AT t
ENDIF
ENDON

```

## 9.6 使用 EASYHDL 规定数字信号

### 9.6.1 声明输出

数字脚本化发生器的输出脚应该被声明为一个引脚变量，例如：

```
PIN Q //单数字输出 Single bit output
```

```
PIN D[0..7] //8 位数据输出 8 bit data bus
```

一个局部的发生器脚本包含了一个隐式声明 **PIN OUT**。

在那里，发生器有一个名字，但它是不合理的 EasyHDL 变量名，你可以使用一个别名命令。

```
ALIAS OUT=Test Signal
```

```
PIN OUT
```

### 9.6.2 程序流

当一个 EasyHDL 脚本用于规定一个数字信号时，整个脚本都被从头到尾地执行直到下列之一条件满足：

- 一个 **END** 描述被执行
- 在脚本中最后的描述被执行。
- 一个 **“SLEEP”** 或 **“WAIT”** 命令引起脚本执行的终止。

一旦 **“END”** 描绘或脚本的结尾到达，脚本将不再被调用，直到一个 **“CALLBACK”** 或 **“TIMER”** 事件被计划或安排。这两个命令将引起脚本被重新执行（假如被暂停）或从一个对应的事件 ID 从头开始执行。注意：只有一个唯一的执行线索；假如一个事件时脚本是暂停的或睡眠的，那么执行总是从紧跟的描述之后开始的。

典型地，一个数字化信号产生器脚本将由输出指定组成，这些指定中包含了 **“SLEEP”** 或 **“WAIT”** 描述，用以产生所需的定时。例如下面的代码将会对一个变量激发出一个串行的数据流。

```

// Bit Bang a single character

// 1 Start bit, 1 Stop bit and no
// parity are assumed.
// Start bit
OUT = 0
SLEEP FOR BITTIME

// Data bits
FOR j=0 TO 7
  OUT = d & (1 << j)
  SLEEP FOR BITTIME
NEXT j

// Stop bit
OUT = 1
SLEEP FOR BITTIME

```

### 9.6.3 规定输出转换

你能够以以下几种方式中的一种规定数字信号产生器的输出：

- 作为一个连续值（转换附加 0 延时）。

OUT=dout

- 作为一个将来值

OUT=vout AT time

多个将来的指定可以立即被安排。例如一个脉冲可以被描述为：

OUT=1 AT time

OUT=0 AT time+width

- 作为一个时间量化的转换

OUT=dout AFTER delay

OUT=dout AFTER tdlh,tdhl

在时间量化转换的情况下，有两个参数，当输出上升时(low->high transition)，tdlh 被使用，当输出下降时(high->low transition)，tdhl 被使用。

## 9.7 变量

### 9.7.1 变量声明

变量可以被声明为单一的或数组的，数组变量提供了一个产生查找表的方法。更重要的是，支持总线。

变量的声明有以下几种形式：

**TYPE** *name*[[*[lower..upper]*]] [[*=expression*]] [[,etc]]

**TYPE** *name*[[*[width]*]] [[*=expression*]] [[,etc]]

声明表现为你希望声明的变量类型、名字，紧接着由一个逗号分隔一系列变量名。假如变量代表一个数组，则在名字之后需有数组包含的两周期分离的高、地边界值或数组的宽度，它们被包含在方括号之中。数组的宽度或高低边界必须是整型常数，表达式不容许。假如你通过宽度规定了一个数组，那么低边界为 0、高边界为 1 被假定。变量的初始化可以通过其后的一个等号及之后的初始化表达式来实现。

例如，

```
PIN data[2..5]=10, output
```

```
INT i
```

```
INT j=0,k[2]
```

```
TIME t1,t2=10n
```

声明两变量的类型 **PIN** (data and output), 三变量的类型 **INT** (i, j, and k)及两变量的类型 **TIME** (t1 and t2). 数据“data”和“k”变量都被声明为数组，“data”有四个元素 data[2], data[3], data[4] and data[5], 而“k”有两个元素 k[0] and k[1]。在被声明的变量中，“data”，“j”，及“t2”被相应地初始化为 10, 0 及 1E-8。

带有初始化表达式的变量的值只是在导入阶段一次被指定初值，之后，变量将依据调用脚本来决定其值，相对每一次调用来讲，而不再被指定为它们的初始化值。

关于声明 **PIN** 变量数组，请看 **BOOL** 及 **PIN** 类型获取更多的信息。

请看变量中有关指定变量为表达式的解释。

## 9.7.2 数据类型

在 EasyHDL 中有三种数据类型：

**INT** 无符号或有符号的整数，范围：0 ~ 2,147,483,648.

**FLOAT** 无符号或有符号的浮点数，范围：2.225074 x 10<sup>-308</sup> ~ 1.767693 x 10<sup>-308</sup>.

**TIME** 无符号或有符号的整数，有能力维持一个事件的时间，有 1ps 的分辨率。在模拟仿真中的当前时间可以来自系统变量“REALTIME”，它将返回一个浮点数。

## 9.7.3 字符串类型

EasyHDL 支持简单字符串类型。

**STRING** 简单的 ASCII 字符串.

不同的操作可以通过字符串变量实现：

**STRING** S1 = "Hello", S2 = "World", S3

**INT** C

S3 = S1+" "+S2 // S3 <= "Hello World"

S3 = S1[1,4] // S3 <= "Hell"

C=ASC(S1) // C <= 48h ('H')

C=ASC(S1[2]) // C <= 65h ('e')

S3 = CHR(C+1) // S3 <= "f"

## 9.7.4 BOOL 及 PIN 类型

EasyHDL 的 **PIN** 是用来访问数字信号发生器的引脚，该数字信号发生器由其所连接的脚本所驱动。被声明为这种类型的变量被连接到被命名为相同的数字信号发生器上，该数字信号发生器由其所指定的脚本所驱动。

看声明输出获取有关如何声明 **PIN** 变量的更多信息。

变量声明的顺序是不重要的，对于其它的 EasyHDL 类型，被声明的变量的初始化也是容许的。

当一个 **PIN** 类型的变量被读或测试时，它返回这个引脚的以前值及当前值。在这里，引脚的活动取决于引脚被连接的网络状态、引脚的电器类型、逻辑器件系列的转换表、以及通过拥有的器件或发生器所指定的引脚的任何特性。类型为 **PIN** 的变量只能通过等价于下列的系统常数之一来测试：

**TRUE, ACTIVE** - 引脚当前是激动的，它以前的状态不管。

**FALSE, INACTIVE** - 引脚当前是非激动的，它以前的状态不管。

**FF** - 引脚当前是非激动的，并且保持非激动的。

**FT, POSEDGE** - 引脚是非激活的，已被激活。

**TF, NEGEDGE** - 引脚是激活的，变为非激活的。

**TT** - 引脚是激活的，保持为激活的。

直接驱动输出引脚连接的网络是可能的，而不须考虑引脚的电器类型，或所指定的任何其他特性。这可以通过指定 **PIN** 变量为下列系统常量之一来实现。

**FLT** - 使用漂浮态驱动网络。

**SLO** - 使用强低态驱动网络。

**SHI** - 使用强高态驱动网络。

**WLO** - 使用弱低态驱动网络。

**WHI** - 使用弱高态驱动网络。

由于引脚活动状态的存储是复杂的，特殊类型的“**BOOL**”被用来存储引脚变量的状态。**BOOL** 类型等同于 **PIN**，除非在声明的 **BOOL** 类型没有连接到一个外部引脚的情况下例外。

## 9.7.5 节点类型

EasyHDL 中的类型 **NODE** 用来访问模拟信号发生器的管脚，该信号发生器被脚本所连接。声明为这种类型的变量被连接到由脚本所指定的拥有相同名字的信号发生器上。

看声明输出获取有关如何声明 **NODE** 变量的更多信息。

变量声明的顺序是不重要的，对于其它的 EasyHDL 类型，被声明的变量的初始化也是容许的。

当一个 **NODE** 类型的变量被读或测试时，它返回这个变量正在驱动电压或电流值。在时间量化转换的情况下，它将是当前仿真时间点的内插值。



## 9.7.6 指定变量

指定单一变量有常规的语法：

```
[[LET]] variable = expr'n[[,variable=expr'n]][[,etc]]
```

例如：

```
INT VAR1, VAR2, VAR3
```

```
...
```

```
VAR1=10, VAR2=20, VAR3=30
```

通常，表达式的结果总是在分配数据之前，指派变量的类型。例外的是，指定一个网络状态给一个 **PIN** 类型的变量，这个变量连接到了一个外部的发生器或输出器件的管脚上，在那里状态被直接应用到发生器或管脚的网络上。

数组可以被指定。假如表达式产生一个数组结果，并且在数组表达式中的元素的数目同所指定的数组变量中的元素的数目一样，那么指定可以从每一个数组的低界开始，依次利用表达式来实现对数组变量中各个元素的指定工作。假如表达式中的元素不足以指定数组变量中的元素，那么一个实时运行错误将导致。

例如：

```
INT A[10..19]
```

```
A[10..12]=A[15..19]    // A[10]=A[11], A[11]=A[16], A[12]=A[17]
```

```
A[10..15]=A[16..17]    // Error, insufficient elements in
```

```
// A[16..17].
```

假如表达式的结果是一个单一值，数组变量被指定为非“**PIN**”类型，那么这个值被首先传给数组变量的类型，然后被指定给其中的每一个元素。对于“**PIN**”类型的数组变量，这个值首先传送的是一个“**INT**”，然后 **PIN** 数组中的每一个元素被设置为激活或非激活，这取决于该 **INT** 的结果的相应位是“1”还是“0”。在数组中的最低位元素被设置为表达式值的最低位（第 0 位），下一位元素被设置为表达式值的第一位，依次类推。指定并不是依据位的值所做的（该位对应的元素号）。例如：

```
INT A[3]
```

PIN D[2..3]

A=4 // A[0]=4, A[1]=4, A[2]=4

D=10 // D[2]=Bit 0 of 10, D[3]=Bit 1 of 10.

对一个 **PIN** 类型的变量进行指定时可以使用扩展语法以提供定时信息，如下：

*pin\_var = expression AFTER time\_expression*

*pin\_var = expression AFTER tlh\_expression, thl\_expression*

*pin\_var = expression AT|@ time\_expression*

第一个语法使用关键字 “**AFTER**” 和一个表达式来强制引脚的输出在规定的时间内改变。第 2 个语法也是用关键字 “**AFTER**”，但有两个时间表达式，“tlh\_expression” 和 “thl\_expression” 表达式强制输出引脚或者在 “tlh\_expression” 后发生改变，假如引脚输出涉及一个从低到高的转变；或者在 “thl\_expression” 后发生改变，假如引脚输出涉及一个从高到低的转变。注意术语 “low-to-high” 及 “high-to-low” 通常是：一个输出开关从一个悬浮态到一个弱低态被认为是完成一个高到低的转换。最后一句语法使用 “**AT**” 或 “**@**” 符号规定一个绝对时间，在该时间点，输出引脚被改变。

应该注意到，对一个引脚的指定在规定的时间内到达前不会发生，在脚本结束后以前也不会发生，被暂停的情况也是如此，甚至在指定的时间同脚本调用的时间一样的情况下也是如此。因此，在同一个脚本调用期，对引脚的指定或随后的测试将不产生指定值。例如：

PIN p

p=TRUE // Assume pin was inactive, we assign it active.

IF p=TRUE THEN // Pin is still inactive, therefore test fails.

...

关于指定 PIN 变量的更多信息请查看 **BOOL** 及 **PIN** 类型。

## 9.8 表达式

### 9.8.1 操作符

下列操作符在 EasyHDL 中是合理的：

Parentheses     ()

Unary minus -  
 Unary logical NOT NOT or !  
 Unary bitwise NOT ~  
 Logical AND AND or &&  
 Logical OR OR or ||  
 Bitwise AND &  
 Bitwise OR |  
 Bitwise XOR ^  
 Less than or equal <=  
 Less than <  
  
 Equal = or ==  
 Not Equal <> or !=  
 Greater than >  
 Greater than or equal >=  
 Left Shift >>  
 Right Shift <<  
 Addition +  
 Subtraction -  
 Multiplication \*  
 Division /  
 Modulus %

在减过程中，下列操作符优先：

Unary operators (!, NOT, -).  
 Parentheses.  
 Modulus, Division, Multiplication.  
 Addition, Subtraction.  
 Left and Right Shift.  
 Bitwise AND, OR and XOR.  
 Less than, Less than or equal, Greater than, Greater than or equal.  
 Equals, Not equals  
 Logical AND and OR.

## 9.8.2 值

值可以是变量，系统变量、系统常量、函数的结果、子表达式的结果（通过在括号中的表达式估计）、数字。数字可以是十进制、十六进制（在数字前面加\$符号）、二进制（在数字前面加%符号）。所有的值可以通过一个一元操作符修改。下面的例子给出了这些情况。

23 - 十进制值 Decimal value, 23 decimal

%1010 - 二进制值 Binary value 1010 (10 decimal)  
\$010A - 十六进制值 Hexadecimal value 010A (266 decimal)  
(10\*5) - 子表达式 Sub-expression, 10\*5 (50 decimal)  
TRUE - 系统常量 System constant, TRUE  
!TRUE - 一元的操作符及常数 Unary operator and constant, NOT TRUE  
max\_count - 变量 Variable, max\_count  
EVTTIME - 系统变量 System variable, EVTTIME  
~%1010 - 一元的按位取反 Unary Bitwise NOT (5 decimal)  
!\$A - 一元的逻辑非 Unary Logical NOT (0 decimal)  
"Hello" - 字符串常数 String constant

评估优先操作的例子

下面是几个表达式的例子：

2 \* -3 + 4 - 结果为 -2  
  
2 \* -(3 + 4) - 结果为 -14  
5 << 3 - 结果为 40  
10 >> 2 - 结果为 2  
3 << 3+4 - 结果为 28  
20-3\*4 - 结果为 8.  
2<1+3 - 结果为 1 (i.e. TRUE, 2<4)  
1+2>=8 - 结果为 0 (i.e. FALSE, 3>=8)  
40%16 - 结果为 8  
!(1<3) - 结果为 0 (i.e. FALSE, !TRUE)  
!5<2 - 结果为 1 (i.e. TRUE, 0<2)

### 9.8.3 函数

EasyHDL 支持一些常用的数学函数。所有的函数由函数名及随后的圆括号中的参数（可以是表达式）所表达。函数的结果作为值被返回。下面的表达式将平方根表达式的结果指定给变量 Z。

```
z = sqrt (a*b*c)
```

EasyHDL 支持下列表达式：

DEG - 将弧度转换为度  
 RAD - 将度转换为弧度  
 SIN - 返回参数的正弦值，参数为弧度  
 COS - 返回参数的余弦值，参数为弧度  
 TAN - 返回参数的正切值，参数为弧度  
 ASIN - 反正弦函数，结果单位为弧度  
 ACOS - 反余弦函数，结果单位为弧度  
  
 ATAN - 反正切函数，结果单位为弧度  
 SQRT - 返回平方根。  
 ROUND - 将参数整形为最近的整数  
 FRAC - 返回浮点数的小数部分  
 ASC - 返回在字符串中的第一个字符的 ASCII 码  
  
 CHR - 将 ASCII 码转换为字符  
  
 LEN - 返回字符串的长度

SUBSTR - 返回一个子字符串，第二个参数规定其开始位置，可选的第三个参数规定其长度

## 9.9 系统常量及变量

系统常量是使用 DSIM 定义的常数值，以便使脚本简洁。其对于一个特定版本的模拟引擎是固定的，它不会随脚本的变化而变化。系统变量是只读的，它返回仿真脚本运行时某个特定的值，因此，系统变量是可以随运行状况而改变的。DSIM 系统常数定义如下：

### 9.9.1 数据常量

下列的数据常量被定义：

PI - 3.141582654?

### 9.9.2 事件 ID 常量

常量 EI\_ANY, EI\_BOOT, EI\_SETTLE, EI\_INPUT 及 EI\_CALLBACK 用于测试系统变量 EVTID，以确定当前的脚本调用是否是系统事件的结果，以及使用在使用命令“SLEEP 及 WAIT”

的情况下。

当脚本被调用时，引起调用的事件 ID 被放置到系统常量 **EVTID** 中，**EI\_INPUT** 值指示是调用网络状态改变的结果，该网络是脚本正在模型化的器件的管脚之一。**EI\_CALLBACK** 值指示调用是用户定义调用事件的结果，该事件在没有用户提供事件 ID 号时被初始化。

有关 **SLEEP** 命令的信息查看 **SLEEP** 命令

有关 **WAIT** 命令的信息查看 **WAIT** 命令

## 9.9.3 网络状态常量

常量 **SHI, SLO, FLT, WLO, WHI** 用来指定 **PIN** 类型的变量，以直接驱动连接在发生器或相应器件引脚上的网络。

有关这些常量的使用信息，请查看 **BOOL** 及 **PIN** 类型。

## 9.9.4 PIN 及 BOOL Pin 活动常量

引脚活动状态 **FALSE, INACTIVE, FF, FT, TRUE, ACTIVE, TF, FT, POSEDGE**, 及 **NEGEDGE** 用来测试 **PIN** 类型的变量，以获得引脚以前及当前活动的状态。

有关这些常量的使用信息，请查看 **BOOL** 及 **PIN** 类型。

## 9.9.5 TDSCALE 系统变量

**TDSCALE** 变量返回仿真引擎 **TDSCALE** 仿真控制特性的当前值，它提供了定时的度量值。假如 **TDSCALE** 特性被初始化以产生一个随机的时间度量值，那么 **TDSCALE** 变量将在每次被读时返回一个新的随机时间度量值。

## 9.9.6 RANDOM 系统变量

**RANDOM** 变量从 **DSIM** 脚本随机数发生器返回下一个随机数。随机数的范围为 1~32767 (215-1)。当由发生器产生的数字序列看起来是随机的时候，序列本身是确定的，它由种子的值确定。你可以在脚本的任何地方通过使用种子命令放置种子放生器。**DSIM** 保证：对一个给定的种子值，由后来的 **RANDOM** 所读的的数据序列是确定的。

有关使用种子命令种植脚本的随机数发生器，请查看 **SEED** 命令。

## 9.9.7 REALTIME 及 EVTTIME 系统变量

REALTIME 变量在模拟仿真时以浮点类型返回当前时间值，这不同于 EVTTIME，它在数字仿真模拟时，以时间类型返回当前事件时间，事件时间以 1ps 的分辨率量化。

## 9.10 命令参考

### 9.10.1 ABORT 命令

#### 语法

**ABORT** idcode\_expr

#### 用法

**ABORT** 命令提供一个放弃当前仿真的方法，这个命令的执行引起仿真引擎送出一个 EasyHDL 过程运行时间错误。错误信息被插入到仿真日志中，它包括紧随 **ABORT** 关键字之后的由表达式 “idcode\_expr” 所给出的 ID 码。

#### **ABORT** 关键字

你应该注意到 **ABORT** 命令不仅放弃了当前的 EasyHDL 调用，而且放弃了整个仿真会话。

#### 例子

```
IF input=TRUE THEN PRINT "Input Is TRUE!" :ABORT 0
```

### 9.10.2 ALIAS 命令

#### 语法

**ALIAS** new\_name=existing\_name *[[,new\_name=existing\_name]] [[,etc]]*

#### 用法

**ALIAS** 命令提供一个对一个变量名提供另外一个名字的方法。别名在某些地方是非常有用的，在这些地方器件的管脚名字使用一些字符，而这些字符在 EasyHDL 变量名中是不容许的；而对于发生器上 **PIN** 或 **NODE** 类型的变量也是不可能直接声明的。new\_name 是你希望使用变量或外部管脚，它必须符合 EasyHDL 变量命名规则。existing\_name 是一个存在的变量、属性、或管脚名-所有的字符包括空格或括号都是合理的。假如你需要别名某个事情，在其名字中使用了空格或括号，你需要在

existing\_name 上加双引号。在接下来的别名声明中, new\_name 可以认为是 existing\_name 的同义词, 可以用在你必须使用 existing\_name 的任何地方

例, 如 7493 TTL 计数器, 有复位管脚包含括号- R0(1)及 R0(2), 你不能在 EasyHDL 中包含带有括号的变量名, 你必须给 7493 管脚一个别名, 例如 RS0 及 RS1。之后, 声明一个带有这些别名的 **PIN** 类型变量, 看下面的例子:

有关使用 **ALIAS** 命令声明 **PIN** 变量的情况, 请查看 **BOOL** 及 **PIN** 类型获取更多的信息。

### 例子

```
ALIAS RS1=R0(1),RS2=R0(2)
...
PIN RS1,RS2      // RS1 is the synonymous with R0(1), etc.
...
IF RS1 AND RS2 THEN ...
```

## 9.10.3 BREAK 命令

### 语法

#### **BREAK**

### 用法

**BREAK** 命令引起当前 **FOR...TO...NEXT**, **WHILE...WEND**, **REPEAT...UNTIL** 循环的终止或 **MAP ON** 或 **SWITCH** 当前情况的终止。

- 在 **WHILE...WEND** 循环内的命令执行引起 **WEND** 关键字之后第一个命令连续执行。
- 在 **REPEAT...UNTIL** 循环内的命令执行引起 **UNTIL** 关键字之后第一个命令连续执行。
- 在 **FOR...TO...NEXT** 循环内的命令执行引起 **NEXT** 关键字之后第一个命令连续执行。
- 在 **MAP ON...ENDMAP** 循环内的命令执行引起 **ENDMAP** 关键字之后第一个命令 连续执行。
- 在 **SWITCH...ENDSW** 循环内的命令执行引起 **ENDSW** 关键字之后第一个命令 连续执行。

### 例子

```
WHILE pin_enable=TRUE
...
IF EVTTIME > 1 THEN BREAK
```



...  
WEND

## 9.10.4 CALLBACK 命令

### 语法

```
CALLBACK [[eventid]] AFTER time_expression  
CALLBACK [[eventid]] AT time_expression [[EVERY repeat_expression]]
```

### 用法

**CALLBACK** 命令容许一个回调事件被当作消息，当事件时间到达时，EasyHDL 脚本被调用。

这个命令也容许事件的 ID 值被规定。当事件被处理及脚本被调用时，这个 ID 值将被放置到系统变量 **EVTID** 中，当所有系统事件 ID 码是零或负时，只有正的非零 ID 码可以使用 **CALLBACK** 命令。假如没有 ID 值被规定，当脚本被调用时，**EVTID** 将会包含系统变量值 **EI\_CALLBACK**。

事件 ID 表达式，总是其后紧跟关键字 **AFTER** 或关键字 **AT**，然后是一个时间表达式，它规定在那一个时间回调将发生。**AFTER** 关键字指示时间表达式是一个添加到当前事件时间中的延时值。**AT** 指示时间表达式是绝对时间，它指出了回调事件将发生的时间。

一个重复的回调通过使用 **AT... EVERY** 结构实现。

有关 **EVTID** 系统变量及事件 ID 码的解释，请看事件 ID 常量

### 例子

```
IF EVTID = EI_BOOT THEN  
    REM BOOT Sequence; post call back for 1u.  
    CALLBACK 2 AT 1u  
ELIF EVTID = 2 THEN  
    REM Process out call-back.  
    ...  
ENDIF
```

## 9.10.5 CONTINUE 命令

### 语法

**CONTINUE**

### 用法

**CONTINUE** 命令引起当前的循环表达式 **WHILE...WEND**, **REPEAT...UNTIL** 循环被重新评估, 或同命令限定的表达式相比, 当前的 **FOR...TO...NEXT** 循环变量被增加或重新测试。

- 在 **WHILE...WEND** 循环内的命令的执行引起 **WHILE** 命令的连续执行。
- 在 **REPEAT...UNTIL** 循环内的命令的执行引起 **UNTIL** 命令的连续执行。
- 在 **FOR...TO...NEXT** 循环内的命令的执行引起 **NEXT** 命令的连续执行。

### 例子

```
REPEAT
...
IF EVTTIME <= 100n THEN CONTINUE
...
UNTIL pin_enable=F
```

## 9.10.6 DATA 命令

### 语法

**DATA** *constant* *[[,constant]]* *[[,etc]]*  
**DATAREPEAT**

### 用法

**DATA** 命令规定了一个或多个常数值, 它们可以通过 **READ** 命令依次读出。典型地, 存储的数据将会决定测试信号发展过程, 其通过脚本在时间上的控制实现。例如为了定义分段线性的波形, 你可以定义时间/电压对, 或你可以存储数据及地址以输出到总线上。

常数值可以是任何合理的 EASYHDL 数据类型包括字符串。

关键字 **REPEAT** 引起数据表被循环考虑。也就是说，在最后的的数据项被读后，下一个命令将会在表的开始又重新 **READ**。

### 例子

```
DATA "Hello World ", 250m
DATA "- EASYHDL Test Program\r", 500m
DATA REPEAT
```

## 9.10.7 END 命令

### 语法

**END**

### 用法

**END** 命令引起当前 BASIC 脚本的终止及对模拟引擎控制的返回。没有必要使用一个 **END** 命令结束一个 BASIC 脚本，这里是暗指。

### 例子

```
IF EVTTIME < 100n THEN END
```

## 9.10.8 FOR...TO...STEP...NEXT 命令

### 语法

```
FOR variable=init_expr'n TO limit_expr'n [[STEP step_expr'n]]
...
NEXT variable
```

### 用法

**FOR...TO...STEP...NEXT** 命令容许在 **FOR** 及 **NEXT** 之间的命令执行给定的次数。**FOR** 后的变量用于循环，并且必须事先声明为一个 **INT** 或 **FLOAT** 类型，并且在 **NEXT** 之后也必须被规定。

循环变量被初始化为的 *init\_expression* 值，在 **FOR...NEXT** 块中命令于是被执行。当到达 **NEXT** 关

键字时，循环变量的值被增加，其或是表达式规定的值，或加 1，假如新的循环变量的值小于或等于 `limit_expression` 的值，那么将连续执行 **FOR** 关键字之后的第一个命令，否则，将连续执行 **NEXT** 关键字之后的第一个命令。

在 **FOR...NEXT** 块中的命令总是至少执行一次。而不管初始 *init\_expression* 及 *limit\_expression* 表达式的值。在 **FOR...NEXT** 块中的循环变量是合理的，也可以象其他变量一样操作，不过注意，可能象这样操作循环变量以至于 **FOR...NEXT** 循环从不终止。例如：

```
FOR i=0 TO 10 STEP 2
    i=0
NEXT i
```

循环变量总是设置为 0，尽管当 **NEXT** 相遇时，它在通过步长指令被增加，循环变量的值永远不会到达 10。

### 例子

```
INT v
FOR v=0 TO 10 STEP 2
    PRINT v;"*";v;"=";v*v
NEXT v
```

## 9.10.9 GOTO 命令

### 语法

**GOTO** label\_name

### 用法

**GOTO** 容许程序执行顺序被切换到由标号 `label_name` 所规定的标号之后的命令行开始执行。在 BASIC 脚本中同 `label_name` 有一样名字的标号必须被声明，前后方向的跳转都是容许的。

标号必须在一行的开始处被声明，并且由不超过 30 个阿拉伯字符及下化线('\_')组成的字符串组成，其不应同 EasyHDL 脚本中的任何关键字相同，进一步，命令之后紧跟终止冒号，如果希望的话。

### 例子

```
LBL_START:           // Declare the label.
PRINT "START OF SCRIPT"
```

```
...
IF !done THEN GOTO LBL_START // Jump to PRINT... command.
```

## 9.10.10 IF...ELIF...ELSE...ENDIF 命令

### 语法

```
IF expression [[THEN]]
    ...
    ...
[[ELIF expression [[THEN]]
    ...
    ...]]
[[ELIF expression [[THEN]]
    ...
    ...]]
etc
[[ELSE
    ...
    ...]]
ENDIF
```

*IF expression [[THEN]] ... [[ELIF expression [[THEN]] ... ]][[ELSE ... ]][[ENDIF]]*

### 用法

**IF...ELIF...ELSE...ENDIF** 命令容许在每一个块中的命令条件执行。有一个唯一的 **IF** 关键字及可选的一个 **ELSE** 关键字。在 **IF** 及 **ELSE** 或 **ENDIF** 之间可能会有多个 **ELIF** 关键字。

跟在 **IF** 及 **ELIF** 之后的表达式被连续地评估，假如这些中的任何一个评估为非零，在 **IF** 及 **ELIF** 之后的命令，在下一个 **ELIF**, **ELSE** 或 **ENDIF** 关键字之前的命令被执行。假如没有 **IF** 或 **ELIF** 表达式的值被评估为非零，那么在 **ELSE** 关键字之后的命令被执行。执行将连续 **ENDIF** 命令之后的命令。

### 例子

```
IF i=0 THEN
    PRINT "Value is zero"
    zero = TRUE
ELSE
    PRINT "Value is non-zero"
```

```
zero = FALSE
ENDIF
```

```
IF i THEN j=0 : WHILE (i>0) j=j*2:i=i-1 : WEND : i=0 ELSE j=-1
```

## 9.10.11 MAP ON...ENDMAP 命令

### 语法

**MAP ON** property\_name

```
CASE string1 :    ...
    ...
    [[BREAK]]
[[CASE string2    :    ...
    ...
    [[BREAK]] ]]
[[DEFAULT :    ...
    ...
    [[BREAK]] ]]
```

**ENDMAP**

### 用法

**MAP ON...ENDMAP** 命令容许一个置位命令按照一个或多个条件被选择执行。条件将按照 **MAP ON** 命令后的属性值同标号值的比较结果而被选择。

在 **MAP ON...ENDMAP** 块内的是多个条件。 每一个条件有由：要么是 **DEFAULT+冒号**，要么是 **CASE +标号+冒号**组成。在 **CASE** 关键字之后的前十五个字符直到一个空格、一个 TAB 字符、或冒号被认为是 **CASE** 的标号。冒号之后是 **CASE** 相联系的命令。

在 **MAP ON** 关键字之后的文本属性值同声明的 **CASE** 标号值依次比较，**Case** 是无关的。假如匹配找到了，然后连续执行 **CASE** 标号之后的命令。假如匹配没找到，一个默认 **DEFAULT** 的条件被设置，然后连续执行 **DEFAULT** 标号之后的命令。执行命令直到 **ENDMAP** 关键字位置；假如执行只到下一个下一个 **CASE** 被要求，那么在 **CASE** 中的最后一个命令应该是 **BREAK**。假如没有任何的 **CASE** 被发现且只有一个 **DEFAULT case**，执行 **ENDMAP** 之后的命令将被连续执行。

**MAP ON...ENDMAP** 一般用于按照器件的特性状态初始化模型脚本时使用。 为了这个目标，只是

在 **MAP ON...ENDMAP** 块内特性被指定。在该块内的命令及有关的条件只是在脚本的导入期间被调用。对于所有其它的调用，命令被自动忽略，没有必要把它放置在一个 **IF...ENDIF** 块内。

### 例子

```
// Assign timing properties depending on what family
// of device we are modelling.
MAP ON VALUE

    CASE 74259    :   TDRQ=16n    : TDLHDQ=14n
                  TDHLDQ=11n : TDLHAQ=15n
    BREAK
    CASE 74LS259 :   TDRQ=12n    : TDLHDQ=19n
                  TDHLDQ=13n : TDLHAQ=17n
    BREAK

ENDMAP
```

## 9.10.12 ON BOOT...ENDON 命令

### 语法

**ON BOOT**

...

...

**ENDON**

### 用法

**ON BOOT** 容许一个置位命令在仿真的开始被执行一次，准确地说，等价于写。

```
IF EVTID = EI_BOOT THEN
```

```
...
```

```
...
```

```
ENDIF
```

### 例子

```
ON BOOT
```

```
    REM BOOT Sequence; post call back for 1u.  
    CALLBACK AT 1u  
ENDON
```

### 9.10.13 ON EVENT...ENDON 命令

#### 语法

```
ON EVENT [[event_id]]  
    ...  
    ...  
ENDON
```

#### 用法

**ON EVENT** 命令容许无论什么时候一个特殊的事件发生时，一个置位命令被执行。准确地说，等价于：

```
IF EVTID = event_id THEN  
  
    ...  
    ...  
ENDIF
```

假如一个 *event\_id* 没有定义，那么它假定为 EI\_ANY.

在正常的模拟时间点期间，在 **ON EVENT** 块内的任何代码将不被执行。

#### 例子

```
ON TIMER AT PERIOD EVERY PERIOD  
    RESTORE  
    OFFSET=EVTTIME  
ENDON
```

```
ON EVENT  
    READ t,v  
    IF t > 0  
        t = t + OFFSET  
        OUT=v AT t
```



```
CALLBACK AT t
ENDIF
ENDON
```

**ON EVENT** 句子在 startup (EI\_BOOT), 每一个 **TIMER** 周期, 以及每一次 callback (EI\_CALLBACK) 时被执行。

## 9.10.14 ON TIMER...ENDON 命令

语法

```
ON TIMER event_id AFTER delay_expr
```

```
...
```

```
...
```

```
ENDON
```

```
ON TIMER event_id AT time_expr [/EVERY period_expr ]
```

```
...
```

```
...
```

```
ENDON
```

用法

ON TIMER 命令设置定时器并且容许当它被触发时一个置位被运行。准确地说, 等价于:

```
TIMER event_id AT time_expr EVERY period_expr
```

```
IF EVTID = event_id
```

```
...
```

```
...
```

```
ENDIF
```

假如 *event\_id* 没有被指定, 那么一个唯一的 id 被自动定位。

当使用 **TIMER** 命令时, 一旦一个事件已被安排, 则 **ON TIMER** 命令不能被触发。

例子

```
ON TIMER AT PERIOD EVERY PERIOD
```

```
    OFFSET=EVTTIME
```

```
    SLOPE=1
```

## 9.10.15 PRINT 命令

### 语法

**PRINT** *expression* | "*text*" *[[,|; expression | "text"]][[etc]][[,|;]]*

### 用法

**PRINT** 命令提供了插入文本、变量/表达式进入仿真日志中的方法。

**PRINT** 关键字后面由一系列参数组成，它们由逗号、分号等隔离。每一个参数由一个表达式或文本加上双引号。每一个参数以从左到右的方式被插入到仿真日志中。由逗号分割的参数插入到仿真日志中以空格分开，由分号分割的参数插入到仿真日志中直接位于前一个参数之后。假如 **PRINT** 命令参数单以参数结束，那么在仿真日志中，在最后一个参数被插入时将换行。假如 **PRINT** 命令参数单以逗号或分号结束，那么没有新行被插入，来自随后的 **PRINT** 命令的任何输出将在最后的一个参数之后，它们都将在一行内。

### 例子

```
PRINT var1;"+";var2;"=";var1+var2;"."
```

## 9.10.16 READ 命令

### 语法

**READ** *var1* *[[,var2]] [[,etc]]*

### 用法

**READ** 命令从常数值中获取下一个项目，这些常数值在程序中以 **DATA** 描述方式被声明，并以规定的变量存储。

### 例子

```
REM Define data to be output
```

```
DATA "Hello World", 250m
```

```
REM Read the data
```

```
READ s,td
```

## 9.10.17 REM 命令

### 语法

**REM** *text*

### 用法

**REM**(remark) 命令一些解释信息包含在 EasyHDL 脚本中。在 **REM** 关键字之后的所有内容将被编译器所放弃。

注意：由于在 **REM** 关键字之后的所有内容将被编译器所放弃，**REM** 命令应该在由冒号分开的顺序命令结尾之前。

也应该注意：你也可能需要“/”，“//”这样的优点是不需关键字，它不要求在它之前有一个冒号。对于 **REM** 命令，“//”之后的所有内容将被 EasyHDL 编译器放弃。

### 例子

```
REM This is a comment, and will be ignored.
```

```
REM So is this.
```

```
PRINT "A=",a      // Display the value of A
```

## 9.10.18 REPEAT...UNTIL 命令

### 语法

**REPEAT**

...

...

**UNTIL** *expression*

### 用法

**REPEAT...UNTIL** 命令容许在 **REPEAT** 及 **UNTIL** 关键字之间的命令被重复执行，直到在 **UNTIL** 关键字之后的表达式的值为非零值为止。

在 **REPEAT** 及 **UNTIL** 关键字之间的命令被执行，当 **UNTIL** 关键字到达时，循环表达式的值被评估，假如结果为非零，执行将在 **UNTIL** 关键字之后的第一个命令处继续，否则将在 **REPEAT** 关键字的第一个命令处继续执行。

注意：在循环的末尾，当循环表达式被测试时，在 **REPEAT** 和 **UNTIL** 之间的命令总是被至少执行一次。

### 例子

```
REPEAT
    WAIT
    count=count+1
UNTIL enable_pin=TRUE
```

## 9.10.19 RESTORE 命令

### 语法

**RESTORE**

### 用法

**RESTORE** 命令复位由命令所使用的数据指针，以便它指向 程序中的第一个数据描述。这也容许数据再次被读。

### 例子

```
// Define data to be output

DATA "Hello World", 250m
// Read the data
READ s,td
// Read the data again
RESTORE
READ s,td
```

## 9.10.20 SEED 命令

### 语法

**SEED** (*value*)

### 用法

**SEED** 命令被用来种植脚本的随机数发生器。对一个给定的种子值，DSIM 保证：当由 **RANDOM** 系统变量的并发读所返回的系列值看起来是随机的情况下，系列本身是有限和确定的。

例如，规定一个种子值为 1，则由 **RANDOM** 系统变量所产生的随机系列的前 10 个随机数总是 909, 3365, 5030, 24170, 3305, 10539, 20191, 14244, 15844, 21839, 9277. 类似地，假如一个种子值为 500，其产生的前 10 个数的随机系列总是 14616, 21006, 7672, 12031, 9888, 2966, 3985, 24555, 30757, 30317, 27030.

有关 **RANDOM** 系统变量的解释，请看 **RANDOM** 系统变量

### 例子

```
SEED (500)
FOR i=1 TO 10
    PRINT "Random Value",i,"=",RANDOM
NEXT I
```

## 9.10.21 SLEEP 命令

### 语法

**SLEEP** *[[eventid]]* **FOR** *time\_expression*  
**SLEEP** *[[eventid]]* **UNTIL** *time\_expression*

### 用法

**SLEEP** 命令使得当前被调用的数字脚本被暂停要么一个固定的延时，要么暂停到直到一个设定的时间为止。当结束时间到达时，在 **SLEEP** 命令之后的第一个命令将继续执行。

**SLEEP** 命令容许一个事件的 ID 值被规定，这个表达式的值决定脚本的暂停是否可以被其他事件所中断。如，使得以前的被声明的回调变为正常，或使得输入引脚所连接的网络中的一个或多个发生改变。事件的 ID 表达式，假如规定了，应该是一个唯一的正非零值，或系统常量中的一个：**EI\_ANY**,

**EI\_INPUT**, 或 **EI\_CALLBACK**. 使用唯一的整数、或 **EI\_CALLBACK** 常数保证暂停将不被中断。**EI\_ANY** 或 **EI\_INPUT** 容许暂停被任何事件或在输入引脚上的变化所中断。当暂停完成时, **EVTID** 变量将包含所规定的事件 ID 的值。默认的 ID 值, 假如没有表达式所指定, 则是 **EI\_CALLBACK**。

事件 ID 表达式, 在任何情况下, 总是其后为关键字 **FOR** 或关键字 **UNTIL** , 并且表达式用于表达在那一个时间点暂停结束。**FOR** 关键字指示时间表达式是一个被添加到当前时间中的延时值, **UNTIL** 关键字指示时间表达式是绝对时间, 在该时间点暂停结束。

**SLEEP** 命令不能用在模拟信号产生器脚本中。

有关 **EVTID** 系统变量及事件 ID 码的解释请看 Event Identification Constants 部分

## 例子

PIN output

```
...
// Generate 100 periods of mark=20n, space=10n clock
FOR i=0 TO 100
    output = TRUE      // Output active
    SLEEP 1 FOR 20n    // Wait 20ns (mark time)
    output = FALSE     // Output inactive
    SLEEP 1 FOR 10n    // Wait 10ns (space time)
NEXT i
...
```

## 9.10.22 SWITCH...ENDSW 命令

### 语法

**SWITCH** *switch\_expression*

**CASE** *value1*:

```
...
    [[BREAK]]
[[CASE value2 : ...
...
    [[BREAK]] ]]
[[DEFAULT : ...
...
    [[BREAK]] ]]
```

## ENDSW

### 用法

SWITCH...ENDSW 命令容许一个置位命令按照 CASE 中的一个或多个条件执行。条件的选择将依据评估 SWITCH 表达式的值而定。

在 SWITCH...ENDSW 块中是一个或多个条件。每一个条件由关键字 CASE+固定的浮点数值+冒号或关键字 DEFAULT+冒号。在冒号之后是同 CASE 有关的命令。

在 SWITCH 关键字之后的 switch 表达式被作为一个浮点值被评估并且依次同每一个所声明的 CASE 值相比较。假如相匹配，那么将在相匹配的 CASE 值之后的命令处继续执行，假如不匹配，并且一个 DEFAULT 条件被规定，那么将在 DEFAULT 之后的命令处继续执行。执行继续到 ENDSW 命令为止。假如要求只执行到下一个 CASE 处，那么在 CASE 中的最后一个命令将是 BREAK 命令。假如在任何 CASE 中没有发现匹配，并且有一个 DEFAULT 条件，或假如一个 BREAK 在任何 CASE 中被碰到，那么执行将在 ENDSW 关键字后的第一个命令处继续执行。

### 例子

```
SWITCH bus[0..2]
    CASE 0 : REM Fall through to next case...
    CASE 1 : REM Fall through to next case...
    CASE 2 : PRINT "Bus is 0,1,or 2" : BREAK
    DEFAULT : PRINT "Unknown bus value!" : BREAK
ENDSW
```

## 9.10.23 TIMER 命令

### 语法

```
TIMER [[event_id]] AFTER delay_expr
TIMER [[event_id]] AT time_expr [[EVERY period_expr]]
```

### 用法

TIMER 命令以类似于 CALLBACK 命令的流行方式安排事件。不过，不同的是，当以前的计划事件发生时定时器才唯一被触发。换句话说，一旦定时描述已经被执行且发出了一个事件通知后，在事件的时间到达以前，定时器将不做任何事情。

## 例子

```
PIN OUT = 0
TIMER 1 AT PERIOD EVERY PERIOD
OUT = !OUT
```

## 9.10.24 WAIT 命令

### 语法

**WAIT** *[[eventid]]*

### 用法

WAIT 命令引起当前脚本的调用暂缓执行一个新的事件。当且仅当当前这个事件发生后，WAIT 命令之后的第一个命令才继续执行。

可选地，命令之后可以是用于等待的事件 ID 值，假如这样的值被规定的话。当表达式对应的事件到达以后，脚本才继续执行。假如没有 ID 值被规定的话，默认值即为系统常量 EI\_ANY，脚本将在任何事件到达以后继续执行。

有关事件 ID 码的解释，请看事件 ID 常量。

## 例子

```
REM Post call-back, and then wait for it and only it - input
REM   pin transitions are ignored.
CALLBACK 2 AFTER 100n
WAIT 2
...
REM Post call-back; wait for it or an input pin transition
CALLBACK 3 AFTER 200n
WHILE EVTID != 3 AND EVTID != EI_INPUT WAIT
```

## 9.10.25 WHILE...WEND 命令

### 语法

**WHILE** *expression*

...



...

**WEND**

**WHILE** *expression* ... **[/WEND]**

## 用法

**WHILE...WEND** 命令容许在 **WHILE** 和 **WEND** 关键字之间的命令，在 **WHILE** 关键字之后的循环表达式的值为非零时重复执行。

在 **WHILE** 关键字之后的循环表达式的值被评估，假如结果为零时，在 **WEND** 关键字之后的第一个命令将继续执行。否则在 **WHILE** 和 **WEND** 关键字之间的命令被执行。当 **WEND** 关键字被遇到时，执行返回到 **WHILE** 命令且重新评估表达式的值。

注意，当在循环顶部的的循环表达式的值被评估时，假如第一次测试时表达式的值为零值，则在在 **WHILE** 和 **WEND** 关键字之间的命令可以不被执行。

## 例子

```
WHILE i<j
    PRINT "i is less than j"
    i=i+2;j=j+1
WEND
```

```
WHILE enable_pin=F WAIT
```

# 第 10 章 SPICE 模型的应用

## 10.1 前言

目前许多主流器件厂商都提供产品线的与 SPICE 兼容的仿真模型。由于 PROSPICE 是基于 Berkley SPICE 源码的，因此使用这些模型几乎没有问题。然而，在使用第三方模型时，有一些问题必须明白：

- 一个 SPICE 模型是一个 ASCII 网表文件。里面不含任何图形信息，故不能直接放在原理图上。这就意味着应该使用 ISIS 器件库中相关 SPICE 模型的器件。遗憾的是，对于原理图器件库的部件没有标准的文件格式，因此通常需要自己绘图设计。
- 在一个 SPICE 网表中，模型是参考使用 X 卡（X card）的子电路来调用的。电路节点与模型节点的关联由 X 卡中电路节点的序号决定。

例如：

```
XU1 46 43 32
```

意思是电路节点 46 连接到模型节点 1。类似的电路节点 46 连到模型节点 2、电路节点 32 连接到模型节点 3。不幸的是，这种结构意味着模型节点未被命名。更糟的是没有将模型节点号和实际器件引脚号相联系，假如实际的封装含有 NC（空脚），这在 SPICE 模型中是无法描述的。对于多元器件如 TL074 也很复杂：模型是实现单个运放还是四个？

实际上，这意味着需要通过某种准确的交叉参考信息告诉 ISIS 器件引脚所使用的 SPICE 模型节点号，因为 ISIS 的引脚名和引脚号都不能用于表达此信息，所以引入 SPICEPINS 这个器件属性。

- SPICE 存在诸多变种。最早期的是 SPICE 2，许多模型依据此写成。然而，较多厂家公布了他们兼容 PSPICE 的模型，这是一个在基本标准下扩展的专有版本。PSPICE 同时支持 SPICE3F5 的很多功能，它也有些不同的原型类别，并且使用了不同的语法描述一些新的内容。因此为 PSPICE 设计的模型可能不适用于 PROSPICE

简单地说，如果第三方模型工作不正常，首先应该检查它是不是基于 SPICE2 或 SPICE3。

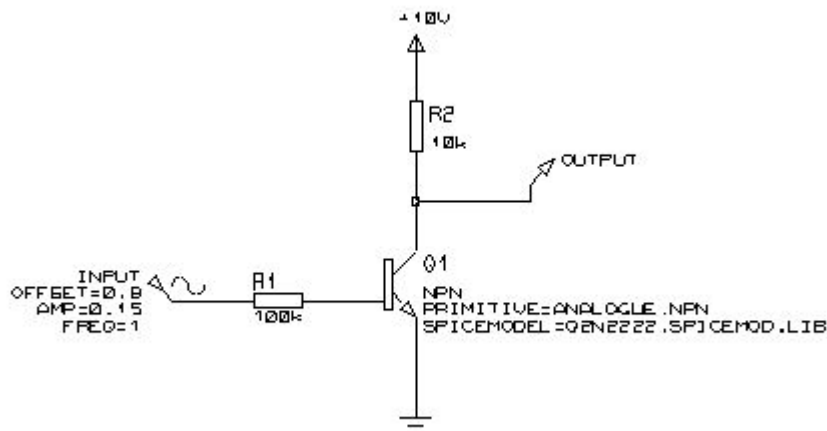
一个好消息是我们自己搜集了大量（目前超过 1500 个）模型并用此生成了 ISIS 器件。

**您必须明白，我们并不对这些模型的精度甚至功能负责，不论是我们还是厂家都不会承担由于使用这些模型而造成的任何损失。任何情况下都不能保证电路仿真能完全准确地反映实际硬件的性能，所以我们忠告在量产之前最好搭试实际电路进行试验。**

## 10.2 使用一个 SPICE 模型（模型卡）

对于半导体器件，特别是二极管，三极管，JFET 和 MOSFET，SPICE 模型通常用单个模型卡来定义。它列出各个 SPICE 原型的参数值。这个过程类似于 SUBCKT 模型，只是由于没有引脚名的转换而简单一些。需要补充的是，在 ISIS 器件库中已经存在不同类型的 SPICE 原型——存在于 ASIMMDLS.LIB 中。

第二个例子是一个 2N2222 三极管（ASCII 文件 SPICEMOD.LIB 中的器件），设计文件是 SPICE2.DSN，在 samples 目录下。



三极管有两个属性——PRIMITIVE 和 SPICEMODEL。  
分别加以解释：

`PRIMITIVE=ANALOGUE,NPN`

这个属性表示该器件由 PROSPICE 使用 NPN 型原型直接仿真。如果你从 ASIMMDLS.LIB 中选取的一个二极管，三极管，JFET 或 MOSFET，就已经有了 PRIMITIVE 的属性赋值。

`SPICEMODEL=Q2N2222,SPICEMOD.LIB`

SPICEMODEL 的赋值与使用 SUBCKT 模型时一样。它指定模型的名称以及包含该名称的 SPICE 网表文件。大多数厂家在一个文件中提供很多的模型定义。

SPICE 模型文件搜索路径是本级目录或者在 Set Paths 对话框中的 Module Path 所设置的路径。

## 10.3 SPICE 模型库

PROSPICE 提供的第三方 SPICE 模型库是与 ISIS 器件和符号库一样的二进制文件。这样做有两个理由：

许多文件非常小且是数字表示，对于大容量硬盘，如果分别存放会浪费很多硬盘空间。

如果许多模型保存于一个 ASCII 文件，PROSPICE 会因被迫分析整个文件而变慢。

根据 SPICE 模型在 SPICE 模型库（SML）中的位置，ISIS 库器件与模型绑定的语法略有不同。例如，第一个例子中的 LMC660 可以替换为如下属性：

```
PRIMITIVE=ANALOGUE,SUBCKT (as before)
SPICEMODEL=LMC660
SPICEPINS=POS IP,NEG IP,V+,V-,OP(as before)
SPICELIB=NATSEMI
```

SPICELIB 属性给出包含该器件模型的 SPICE 库文件。它的搜索路径是本级目录或者在 Set Paths 对话框中的 Module Path 所设置的路径。

SPICE 模型库可以使用命令行工具 PUTSPICE.EXE 和 GETSPICE.EXE 来管理，虽然我们不期望初学者用很多创建或者搜集的模型来获取授权。这两个程序不带参数运行会出现有用的信息。

## 10.4 \*SPICE 脚本

实际上也可以在 ISIS 中直接键入 SPICE 模型定义，然后调用原理图中的相关器件。例如，三极管 2N2222 的模型可以用如下 ISIS 脚本表示：

```
*SCRIPT SPICE
.MODEL Q2N2222 NPN(IS=3.108E-15 XTI=3 EG=1.11 VAF=131.5 BF=300 NE=1.541
+ISE=190.7E-15 IKF=1.296 XTB=1.5 BR=6.18 NC=2 ISC=0 IKR=0 RC=1
+CJC=14.57E-12 VJC=.75 MJC=.3333 FC=.5 CJE=26.08E-12 VJE=.75
+MJE=.3333 TR=51.35E-9 TF=451E-12 ITF=.1 VTF=10 XTF=2)
*ENDSCRIPT
```

三极管的 SPICEMODEL 属性改为：SPICEMODEL=Q2N2222

不需要指定文件名称。

这个功能非常适用于书面得到的模型或者想试验一下参数值的影响的情形。

## 10.5 模型仿真失败的处理

由于子电路模型具有很高的复杂性和设计难度。结果导致某些模型比其他模型易于仿真。作为一般性准则，如果 PROSPICE 不能通过它翻译的模型仿真你的电路，那就没更多其他工具能够仿真了。

我们经常发现一些问题是因为较差的电路设计所致。如果你有器件的应用笔记，可以对照检查一下自己的电路，例如，你是否成功地在输入级馈入了足够的偏置电流。模型的设计者总是参照应用笔记设计而不是一般情形来设计模型。

通过提高 GMIN 可以使 ProSPICE 可以变得更加稳定（但是牺牲了精度）。默认值是 1E-14，因

此最好试试  $1\text{E-}13$  和  $1\text{E-}12$ 。如果高于  $1\text{E-}6$ ，仿真结果将与实际情况相去甚远，所以应该尽量取最小值。

# 第 11 章进阶

## 11.1 温度模型

SPICE3F5 提供了温度效应的扩展模型支持。运行机制如下：

有一个应用于电路中所有器件的全局属性 TEMP。

单个器件的温度属性可以由设置其自己的 TEMP 属性来设定。

如果器件参数与测试温度密切相关，可以全局或单独地通过 TNOM 属性设置为该温度。

电阻、二极管、JFET、MOSFET、BJT 和 1、2、3 级 MOSFET 都具有温度模型。BSIM 模型是在特定的温度下测得。数字原型没有应用温度特性。

然而，重要的是要意识到大多数 IC 的等效电路模型没有正确地对温度效应建模。因为这些模型通常使用理想受控源和其他宏建模原型，而不是包含实际的器件。一旦使用这些原型，模型是不可能显示正确的温度特性的，除非有人不厌其烦地使其能做到。

## 11.2 固化模型数据

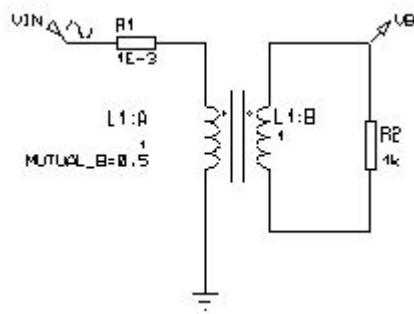
特定的仿真模型，如 EPROM 和带 EEPROM 的单片机或者 Flash 存储器，可以在仿真中“记忆”数据，正如它们在实际中那样。这个动作来自于 MODDATA 属性。固化模型数据块保存在 DSN 文件中，因此在保存设计之前请不要切换 Proteus 任务。

要将固化模型数据还原到初始值，可以使用 Debug 菜单下面的 Reset Persistent Model Data 命令。

## 11.3 地和电源范围

### 11.3.1 为什么需要一个地

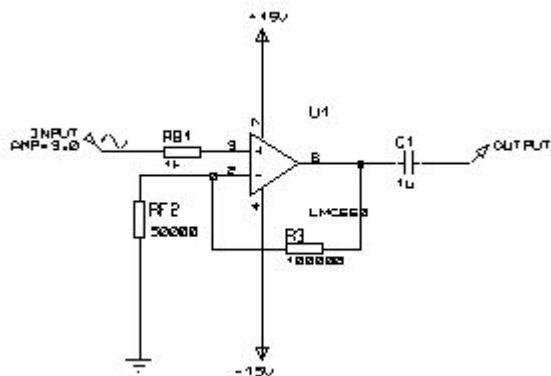
所有仿真都要求定义一个地网络——否则在特定网络上放置探针就没有意义，因为该网络的电压必有须固定的参考点。实际上，更深一层的要求是所有电路元件都应该有对地直流通路，用探头接在浮空的网络上同样没有意义。例如，下面电路中的 VB 是没有意义的：



因为次级是浮空的。理论上，我们可以使用双端探头（比如实际的万用表），但是对没有地的电路求解在数学上是非常困难的，再者，伯克莱大学在 SPICE3F5 中也没有考虑这个。

关键就在于电路中的各个部分都必须有对地直流通路。值得庆幸的是 ProSPICE 会检查这个问题，当网络中存在违反规则将发出警告。多数情况下仿真会因此失败。

下面是另一个导致过问题的电路：



这里，由于耦合电容 C1 的存在，使得 OUTPUT 探针没有对地直流通路，因而仿真器无法求解输出的工作点，因为工作点是在所有电容开路的情形计算的。我们使电容轻微漏电来解决此问题。因此，在没有任何直流通路的情况下，工作点基于 C1 完全充电状态计算。

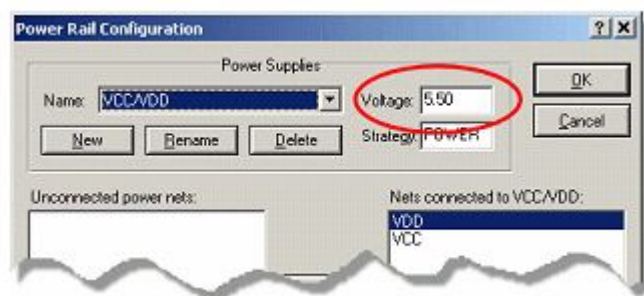
电容的漏电特性由仿真控制属性 GLEAK 定义，默认值是 1E-12Mho。传统的 SPICE 仿真器中，GLEAK 设置为 0 表示电容不漏电。

除了仿真 DRAM 存储器内部构造，我们无法发现这个电路有什么问题，这也避免初学者面对许多奇怪的错误信息。任何情况下，实际的电容器通常有愈一百万兆欧的漏电阻。

地线网络在电路中既可以显式定义也可以隐含定义。显式定义是放置一个无网络标号的地线端子。比如上图 RF2 的下端。如果空间紧张，也可以通过放置 GND 网络标号来定义。隐含定义在使用电源范围、单端信号发生器、负载探头或数字输出时产生。在使用包含内部接地节点的模型也会用到隐含接地。

## 11.3.2 电源

一般可以使用 ISIS 中的 Design 菜单中的 Power Rail Configuration 对话框来指定仿真电源电压。



使用这个表格有两个重要步骤：

1. 使用 NEW 按钮生成一个或多个电源。每生成一个电源，可以在 Voltage 栏设置其电压。
2. 连接一个或多个电源网络到电源，在左边列表选中高亮的网络名并点击 Add 按钮。

同时需注意：

电源网络通过在原理图中放置电源端子或者设计中使用隐含电源脚的器件时形成。

不能对一个存在未连接电源网络的设计进行仿真——仿真将导致网表编译器错误。

也可以放置名称为 +5, -5 或 +10, -10 的电源端子来定义电源。“+”和“-”符号是必需的，例如 5.0V 是不允许的。

ISIS 提供了默认电源范围：VCC/VDD 为 +5V，VEE 为 -5V。VSS 和 GND 设定为内部短路的。这样设定使得像多数 TTL、CMOS 和处理器件仿真时不需要显式定义任何电源连接。

更多关于电源范围设置的信息可以查阅 ISIS 手册。

## 11.4 初始条件

### 11.4.1 前言

无论任何时候进行任何仿真，PROSPICE 首先做的是计算电路的工作点——那是加入任何输入信号前应用的稳态条件。这些值的计算有两种不同的操作模式：

工作点在所有电容充电和所有电感放电情形获得。这时，瞬态分析将显示电路加电足够时间以后的行为，仿真开始时电路已处于稳态。这是默认的操作模式，我们可以指定特定器件和（或）节点电压的初始条件，这是常见的情形。

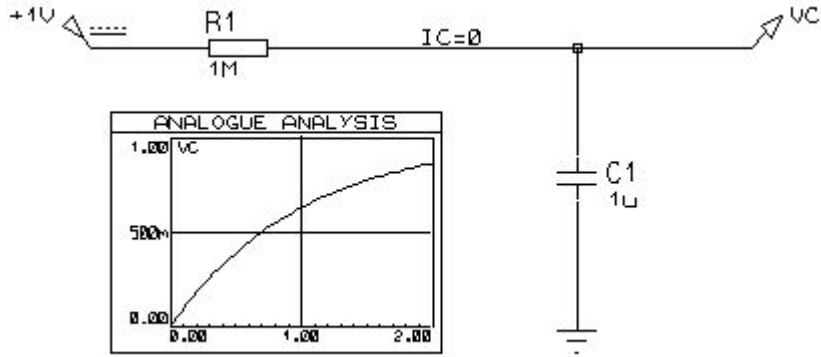
工作点在所有电容短路和所有电感开路情形获得。这时，瞬态分析将显示电路加电瞬间的行为。这种操作模式可以通过不选图表属性的 Compute Operating Point 选项来实现。

在两种情况下，都可以指定特定器件或节点电压的初始条件。这在电路为振荡器，或者某些依赖于特定电容处于放电状态而工作的电路中特别有用。确实，对于振荡器来说，稳态工作点是没有意义的，仿真也会因为缺少某些初始条件而完全失败。



## 11.4.2 设定一个网络的初始条件

最容易的做法是标示一个网络的起始电压。



上面电路中，初始条件用加在探针网络的  $IC=0$  的网络标号实现。如果没有这个设置，PROSPICE 会计算 C1 的稳态电压（如 1V），图表中 VC 就会是一条水平直线。

对于纯数字器件的网络，应该使用逻辑状态作为初始条件，例如：1,0,H,L,HIGH,LOW,SHI,WHI,SLO,WLO 或 FLT，并赋予 BS（Boot State 启动状态）属性。对于混合网络，应该设定初始电压——对于数字器件会自动转换成逻辑电平。

## 11.4.3 设定器件的初始条件

该选项只用于 PROSPICE 不计算初始工作点的情形——如图标中 Initial DC Solution 选项未被选择时。此时除了用上述方法设定了初始条件的外所有节点电压初始为 0。在此条件下，就有可能设定特定器件的初始条件。例如，可以在前面例子中添加属性

$IC=1$

到 C1 使得 C1 从它的稳态电压 1V 开始仿真。

关于 IC 属性所支持的不同 SPICE 原型参见 Modelling 有关章节。

工作点的计算方面没有这个选项是比较难堪，但是 Berkeley 的 SPICE3F5 确实这样编程的，我们增加 PRECHARGE 属性对此作为补救。

## 11.4.4 NS (NODESET)属性

有时候，当因为无法确定工作点而失败，有益的做法是给 SPICE 一个特定网络初始值的提示。这于设置初始条件不同，这些值只用于首次迭代运算，然后该网络“浮动”到矩阵求解的收敛值上。这样有助于收敛并且不会对实际的工作点求解产生影响。

这样的收敛提示可以通过 NS 网络属性设定，因此放置一个  $NS=10$  的网络标号就假定该网络起始值为 10V。

## 11.4.5 PRECHARGE 属性

PRECHARGE 是设定初始条件的另一个选择。它可以设置给电路中的电流或电感，用以分别设定跨越器件的电压或流经器件的电流。

PRECHARGE 属性是 Labcenter 对 SPICE 专门补充的，与 IC 属性不同，PRECHARGE 属性的应用与 Initial DC Solution 选项的选择与否无关。

## 11.5 数字仿真范例

### 11.5.1 九态逻辑

你也许认为数字仿真器只需要对高、低电平建模，而事实上 DSIM 是对总共 9 种状态进行建模的。

State Type	Keyword	Description
Power High	<b>PHI</b>	Logic 1 power rail.
Strong High	<b>SHI</b>	Logic 1 active output.
Weak High	<b>WHI</b>	Logic 1 passive output.
Floating	<b>FLT</b>	Floating output - high-impedance.
Undefined	<b>WUD</b>	Mid voltage from analogue source.
Contention	<b>CON</b>	Mid voltage from digital conflict.
Weak Low	<b>WLO</b>	Logic 0 passive output.
Strong Low	<b>SLO</b>	Logic 0 active output.
Power Low	<b>PLO</b>	Logic 0 power rail.

一般来说，一个给定的状态包含的信息有它的极性——高、低或中间态，和它的强度。它的强度是以两个或更多输出接在同一网络上时的灌出电流或吸入电流的量来计算的。

例如，如果一个集电极开路输出经一电阻接 VCC，那么当输出被拉低时，网络会同时出现弱高电平和强低电平。强低电平获胜，网络变为低电平。另一方面，如果两个三态输出在一个网络上同时生效，并且分别向相反的逻辑方向驱动，那么双方都不能获胜，并导致一个竞争状态。

这种设计保证 DSIM 可以仿真有集电极开路或发射极开路的电路以及上拉电阻，也可以仿真通过电阻相连的三态电路。重要的是要记住 DSIM 只是一个数字仿真器，它不能对模拟行为建模。比如用很大的电阻接到 TTL 输入时，DSIM 仿真是正确的，而可能在实际中因为没有足够的驱动电流而失败。

## 11.5.2 不确定状态

何时一个数字模型的输入是不确定状态，这是数字模型根据共同的准则而产生的。例如，如果一个与门有一个低电平输入，那么它的输出将是低电平，然而当除了一个输入外其他输入为高电平时，它的输入是不确定状态，其输出也将是不确定状态。

- 边缘触发器件需要一个从逻辑 0 到逻辑 1（或相反）的边沿。从逻辑 0 或 1 到不确定状态并不是一个有效的边沿。
- 更加复杂的时序逻辑器件（计数器、锁存器等）由他们的内部逻辑的设计决定了会经常不确定输入是 0 还是 1，这与实际中是不同的。

## 11.5.3 浮空输入行为

这是共同的，如果不是把未接的 TTL 输入都看成接逻辑 1。这种情况发生在忽略连线或者一个输入接在一个未激活的三态输出上。因为内部模型设定需要高电平或低电平的真正逻辑行为，DSIM 不得不做出一些行为。

这个通过 FLOAT 属性来设置。它既可以赋给一个器件，也可以赋给一个接口模型。特别的，对 TTL 部件的赋值是：

FLOAT=HIGH

这样将浮空输入状态翻译成逻辑电平 1。

要设定浮空输入状态为逻辑 0，可赋值：

FLOAT=LOW

否则，浮空输入会被识别为不确定状态。参见前述。

## 11.5.4 毛刺的处理

在设计 DSIM 过程中，关于如何在非常窄的脉冲下的模型进行仿真，我们讨论了很久。根本的问题是，在这种情况下，一个主要的 DSIM 范例的假设即模型纯的数字行为——开始被破坏了。例如，对一个真实的 7400 施加一个 5ns 的输入脉冲会在其输出产生某种脉冲，但是不符合 TTL 逻辑电平的定义。

该输出脉冲是否能够驱动下一级的计数器，这是一个模拟现象的问题了。

最好的办法是考虑极端情况，命名：

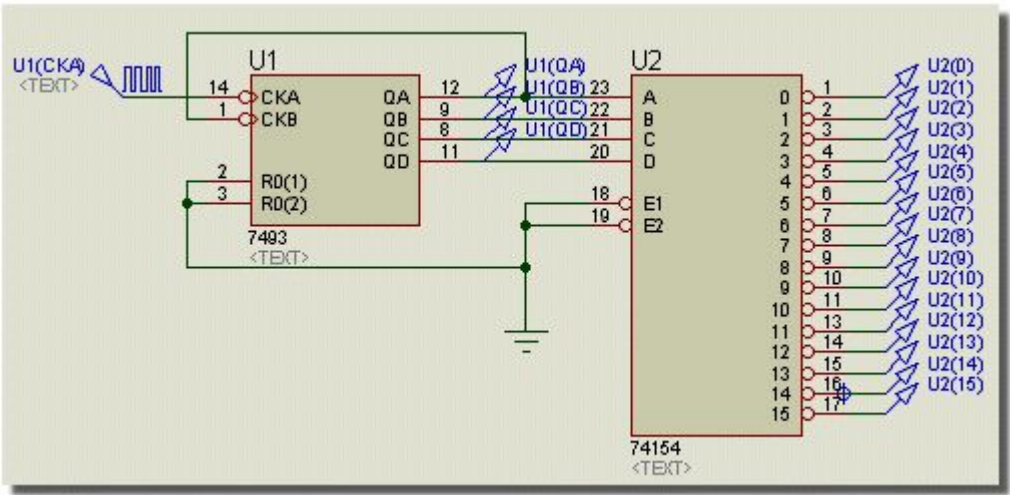
一个 1ns 的输入脉冲将不产生输出。

一个 20ns 输入脉冲可以完美地输出。

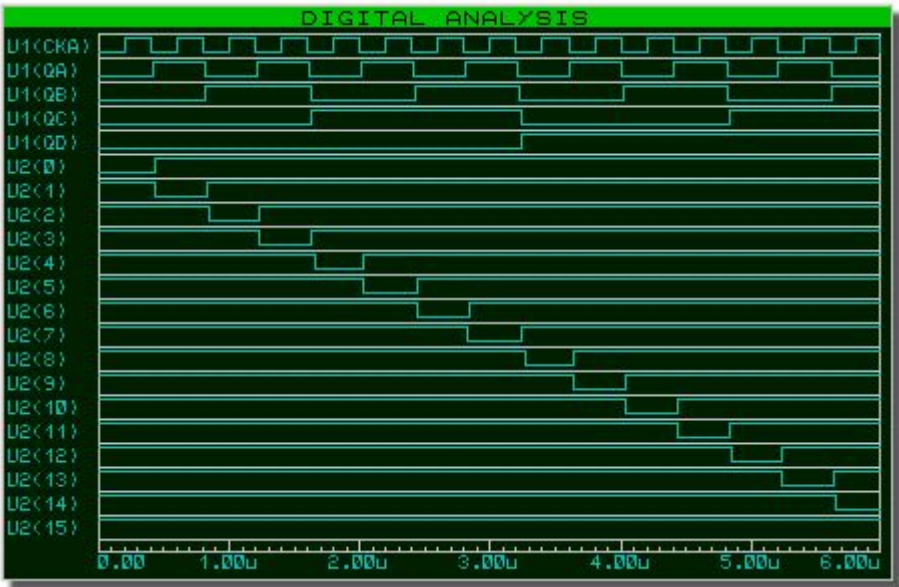
在其间，门电路停止输出脉冲，可以被认为抑制毛刺。这给我们一个 Glitch Threshold Time 的

概念，作为模型除 TDLH 和 TDHL 外的补充属性。

模型是否抑制毛刺的值得关注。为了解决这个问题，考虑下面一个由跳跃计数器驱动的 4-16 译码器：



计数器的输出是阶梯状的，出现这样的可能：译码器会在中间状态产生杂散的脉冲输入，见下面图表：



以这种现象的第一个毛刺为例，在 U1 (QA) 第一个下降沿，它驱动 U1 (QB) 的上升沿，一个中间输入状态 0 送到译码器约 10ns。问题在于译码器是否对此响应，更重要一点，如果输入步长变成 1ns 或 1ps 后会出现什么情况？显然，在后面两种情况实际器件是不会作出反应的，这告诉我们必须在输出处理毛刺而不是在输入。这是因为在上面例子中，输入脉宽都是相对较长而不会被当作毛刺。

特别有趣的是，如果你创建上述电路，也许没有毛刺。当然这是一个差劲的设计，但是 74154 的 TDLH 和 TDHL 大约 22ns 使其阻止的对 10ns 的输入的响应。对于我们单独试过的器件都没有输出，除了电源的细微抖动是可测的。

为了提供对毛刺的控制，所有 DSIM 原型提供给用户可定义的毛刺门限时间属性，叫做 TGxx，xx 表示相对应的输出。我们的 TTL 模型定义这些属性可以在 TTL 器件上取其默认值，因此毛刺门限时间是高一低和低一高传输延时的平均值。把毛刺门限时间设置为零将允许所有毛刺通过。上图中，设置了 74154 的属性 TGQ=0。

最后需要重点指出，如果毛刺门限时间比低一高或高一低传输延时大，那么它将被忽略。这是因为在一个输入边沿后，当相对传输延时过了以后，门输出必须转换其状态——不可能看到将来是否有另外一个输入事件（那将会取消当前输出）。考虑一个对称的门电路，其产生的延时是 10ns，毛刺门限时间是 20ns。在  $t=0\text{ns}$  输入变高， $t=15\text{ns}$  时输入变低。你也许期待能有这样的输出：输出在  $t=10\text{ns}$  时为高然后在  $t=25\text{ns}$  变低。因此产生一个 15ns 将会被抑制的脉冲输入，因为它小于毛刺门限时间。这个脉冲没有被抑制，因为在  $t=10\text{ns}$  时输出必须变高——无法在后续的 20ns 保持为低。

## 11.6 混合模式接口模型 (ITFMOD)

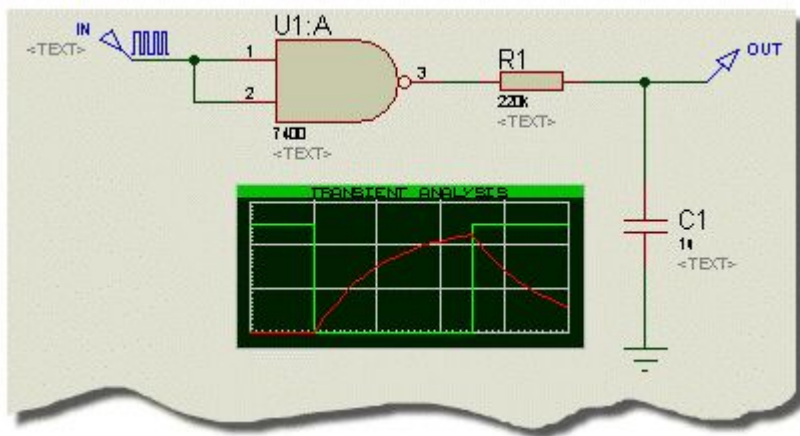
### 11.6.1 概述

在设计 PROSPICE 下混合模式仿真架构时，我们深思了如何确定数字器件的模拟特性这个问题，这些特性包括：

- 器件的输入输出阻抗
- 器件输入的逻辑门限
- 高低电平的电压
- 器件输出的上升和下降时间
- 浮空输入的默认逻辑状态

对于 TTL 库中所有器件都能涵盖以上特性的架构是极其臃肿的。

再者，一个关于电源指标的严重问题产生了（至少对于初学者）——如果低于某个敏感值将会使电路失效。该问题在于我们隐含了 7400 有通过隐含电源脚接到 VCC/GND 而获得 5V 电源。



所有这些问题因 ITFMOD 属性的引入迎刃而解。很像 MODEL 属性，它提供了一系列属性参考值，还在网表编译器中激活一个特殊的机制。它实质上是这样工作的：

- 对于任何具有 ITFMOD 属性的器件，一个附加的模型定义在网表中被调用，用以设定 ADC

和 DAC 对象的控制参数，甚至正负电源的引脚名称。在上面电路中 U1:A 具有属性 ITFMOD=TTL。

- 获得电源引脚名称后（本例中的 VCC，GND），ISIS 生成一个特殊的原型跨接在电源引脚上。ISIS 像子电路或模型那样命名此对象，因此在上电路中，电源对象被命名为 U1:A\_#P。
- 当 PROSPICE 仿真一个混合模式电路时，生成 ADC 和 DAC 对象并认为它们属于相连接的器件。在上面电路中，将产生一个名为 U1:A\_DAC#0000 的 DAC 对象以形成 U1:A 的输出接口。
- 这是一个明智的做法，它以名称前缀来定位电源接口对象如 U1: A 的对象是 U1: A\_#P。然后指引 U1:A\_DAC#0000 从 U1:A\_#P 中依次获得其属性，这些属性继承了模型在原始 ITFMOD 设定的属性。这样 DAC 对象便以 TTL 逻辑定义的参数运作。
- 电源接口对象也像在上层部件那样连接在其他同电压的电源引脚，从器件的电源引脚吸入或流出电流，使得数字器件的阶梯状输出的建模更加接近真实情形。

## 11.6.2 使用 ITFMOD 属性

已有的接口模型如下：

Property	Description
TTL	Standard TTL (74 series)
TTLS	Low power Schottky TTL (74LS series)
TTLS	Standard Schottky TTL(74S series)
TTLHC	High Speed CMOS TTL (74HC series)
TTLHCT	High Speed CMOS TTL with TTL outputs (74HCT series)
CMOS	4000 series CMOS.
NMOS	Microprocessor type MOS circuits with TTL logic levels.
PLD	PLD type MOS circuits.

此后任何新的数字模型通过设置以下属性赋予一个器件系列：

ITFMOD=TTL

各器件系列的定义在 models 目录中的文件 ITFMOD.MDF。

每个定义可以包含任何或所有 ADC 和 DAC 接口原型的属性定义。补充一点，下述属性可能需要给定：

Primitive	Value Example	Description
V+	-	Name of the positive power supply pin.
V-	-	Name of negative power supply pin.
VOLTAGE	5V	Specifies the default operating voltage.
RINT	1m $\Omega$	Specifies the impedance of the internal battery. A value of zero will disable the battery.
FLOAT	-	Specifies HIGH or LOW value for floating inputs.

最后值得指出的是任何特定的属性，如 **TRISE**，可以被上层器件改写，因此如果你想以低上升时间仿真一个 4000 系列的 IC，可以在器件属性列表添加属性 **TRISE=10u**。

## 11.7 录音机和电路分区

### 11.7.1 概述

VSM 系统的一个基本功能是可以把一个大型的设计划分为一个或多个部分或单元并分别进行仿真。

这样做有以下两个主要优点：

- 在完全集成的 CAD 实验中，产品整个设计电路中可能含有某些你不想或者不能仿真的部分。为了避免割裂整个设计，需要某些机制来定义哪些元件需要参与仿真。  
ISIS 通过判断测量点来决定仿真区域，测量点由测试信号源或/和电源驱动。
- 如果设计是多级的，一般要求观察后级根据前级的输出而作出的表现。当然可以让所有级电路一起仿真，但这样会非常的慢。电路分区允许录音机捕获前级的仿真结果，然后作为后级电路的输入重方出来。

ISIS 包含可通过人工或自动的方式这样做的逻辑。在后一种情形，它检测给定区间的电路变化，并对变化部分或者受变化影响的部分进行再仿真。

### 11.7.2 单个部分的仿真

许多仿真实验需要摆脱其他器件测试设计中的局部电路。一般可以通过在该部分注入信号，然后观察信号通过的情况。

显然，信号可以通过放置激励源和相应的导线来注入，电路行为可以通过放置探针来监视。然而，这些并不能使该部分电路脱离设计的其他部分。

要想做到这样，你需要设置激励源的 **Isolate Before** 对话框和探针的 **Isolate After** 对话框。一旦

这样做了，那么仅对分区出的部分电路建立网表并仿真。

了确定对那些部分仿真, ISIS 分析放置在图表的探针，依据以下条件提取器件和导线（包括内部端口连接）：

- 没有其他需要处理的器件。
- 遇到一个隔离的探针或激励源。
- 遇到录音机的输入或者输出。
- 遇到特定电源范围。一个网络通过连接到 POWER 或 GROUND 端口被划分到特定电源范围。GND 和 VCC 网络标号没有此功能。

### 11.7.3 录音机对象

录音机对象在系统中有两个独特的用法：

- 确定仿真点，只对左边的电路进行仿真，隔离右边电路。这些仿真点一般是低阻抗输出驱动高阻抗输入。这个也可以通过放置隔离探针实现。
- 表示可捕获设计中一级电路的输出，用来驱动后一级电路，使后级电路的仿真时就没有前级仿真的开销。

**放置一个录音机：**

1. 选择录音机图标；
2. 使用旋转和镜像按钮根据需要确定录音机的方向。录音机上的小箭头符号表示其方向；在正常方向下，输入在左输出在右。
3. 单击左键进入编辑窗，移动并再次单击将录音机放置在需要的位置上。
4. 可以直接把录音机放在现有导线上，录音机的连接点与导线相触。你也可以把若干录音机放在设计窗的空余地方，然后用导线连接到电路具体位置。

***录音机必须放置在敏感位置——必须是低阻抗驱动高阻抗。如果你将录音机放置在其他位置，将会改变电路行为而导致无效的仿真结果。***

### 11.7.4 录音机模式

为了提供用户最大限度的可控性，录音机有三中工作模式：AUTO, PLAY and RECORD

在后面的讨论中，我们使用左、右来描述电路分区的逻辑关系。设计的输入、输出描述为左、右。

#### **AUTO 模式**

这是默认架构，根据 ISIS 判定哪个部分需要再仿真来选择操作模式，可以使用预先保存的数据。在大多数需要使用录音机的情形，AUTO 模式是最有用的。

自动检测是基于相关部分的子网表中出现的文本。如果任何部件名称、值、属性、连线等发生变化，就进行再仿真，除非电路分区文件已经含有那些信息。



注意所有模型、脚本、全局设计属性等都包括在子网表中，因此这些对象的改变会导致自动分区单元的再仿真。ISIS 不会考虑分区部分的器件是否使用特定的模型、脚本等等。如果想克服此问题，可以使用人工的 RECORD 和 PLAY 模式。

还要注意 AUTO 模式的录音机在电路的交互仿真时会被移除。

## PLAY 模式

这个模式是你可以重放先前录制的文件，该文件既可以通过录音机，也可以通过带 RECORD 属性的探针录制。将需要重放的文件名设置到录音机的 Filename 中；只有输入文件名后才能使用 PLAY 模式。

当录音机处于 PLAY 模式，其左边电路被断开并忽略，除非存在应用于当前图表的探针。

记住你只能播放几路的电路分析数据，尝试播放其他文件将导致错误。

## RECORD 模式

这个模式使得出现在录音机输入的数据以 Filename 中设定的文件名被记录下来；只能在设定了文件名才能使用 RECORD 模式。

RECORD 模式的另一个影响是强制对录音机左边的部分进行再仿真，而与其是否发生变化无关。如果录音机右边电路被放置探针，那么它也将被再仿真，因为隐含认为它依赖于左边部分。

当你想记录一段仿真波形，作为未来后级电路仿真的输入信号时，录音机的 RECORD 模式显得非常有用。

# 11.8 仿真控制属性

## 11.8.1 概述

有大量的参数影响着仿真运行的细节。包括搜索工作点的最大允许迭代次数、判定结果收敛的误差、使用的积分方法等等。

这是所有分析的通用的选项，可以分别在每个图表中通过点击 SPICE Option 来进行调整。

## 11.8.2 误差属性

这一类参数决定了 SPICE 求解的精度。精度越高则仿真时间越长，如果要求误差太高经常会导致无法收敛。

这里最为有用的值是截断误差系数，在传统 SPICE 术语中叫 TRTOL。如果你的结果显得非常糟糕或者粗糙，你可以尝试减小这个值。

最小电导值 GMIN 定义了反偏 PN 结和一些理论上电阻无穷大的漏电特性。减小这个值也许会有助于仿真失败电路的收敛，但是以牺牲仿真精度为代价。参见 Convergence Problems 。

漏电导 GLEAK 我们提到过是用来求解含隔直电容的电路，它定义电容器的直流漏电，通常可以不用管它，除非你仿真像 CMOS 存储器单元那样的特殊的情形。

### 11.8.3 MOSFET 属性

MOSFET 的 SPICE 仿真是基于假定你在做 IC 设计，必然应用规则几何图案的框架。实际上这意味着有很多的参数来定义 MOSFET 器件单元的默认物理尺寸,这些就是这里定义的值。

补充一点,某些模型根据旧版本 SPICE 行为来创建,如果你使用它们,可以开关切换其 MOSFET 行为。

### 11.8.4 迭代属性

Iteration 选项卡下的属性定义 SPICE 如何处理难以收敛的电路。

积分方法可以是 Gear 或者 Trapezoidal。后者主要向下兼容早期版本的 SPICE，Gear 积分方法则在给定步长下获得更高的精度的结果。在 Gear 积分中，高于 2 的阶数是允许的，意味着 SPICE 使用某个时间点更多的历史数据来预测下一个时间点。

当 SPICE 在工作点无法收敛时，它做两个尝试：Gmin 单步和 Source 单步。每个算法的尝试的步数在这里设置。

跟着 3 个选项定义每个工作点的最大迭代次数、传输分析步长、瞬态分析时间点。这些值的提高有助于求解难以稳定或接近不稳定的电路。

最后，有两个选项用来获得快速的仿真。LTRA 压缩仅用于含传输线损耗（LOSSYLINE 模型）的电路。数据点因数据流水线上舍弃接近相同的数据而被处理。绕开未变化的单元以免 SPICE 再计算那些节点电压从上次分析后没发生变化的器件。

### 11.8.5 温度属性

有两个全局的温度属性：TEMP — 默认的工作温度, TNOM — 参数测量温度。TEMP 定义电路的实际温度，TNOM 则是器件参数测试条件时的温度。关于 PROSPICE 下温度建模的讨论参见温度建模部分。

### 11.8.6 数字仿真器属性

#### TDSCALE, TDSEED, TDLOWER 和 TDUPPER

TDSCALE 变量用于控制仿真运行时模型的全部定时属性值的大小，这些属性并没有被模型明确地引用为非量化的。TDSCALE 变量既可以赋予一个浮点常数，也可以赋予关键字 RANDOM。

如果指定了一个常数值，那么模型设定的所有温度属性值将乘以此值；小于 1.0 的常数减小了定时属性值，反之提高属性值。例如，变量赋值：

`TDSCALE = 1.1`

仿真时扩大了所有定时属性值 10%。`TDSCALE` 的默认值是 1.0; 这使得所有定时属性为原值。

如果 `TDSCALE` 被赋予关键字 `RANDOM`, `DSIM` 引擎将使用一个随机的浮点数乘以每个定时属性值进行随机定标。仿真引擎选择的定时定标数值范围受限于 `TDLOWER` 和 `TDUPPER` 变量; 它们分别定义随机浮点数的允许最大和最小值。`TDLOWER` 和 `TDUPPER` 的默认值分别是 0.9 和 1.1, 因此限制了随机定标值在 10% 之内。

产生的随机数序列被称为伪随机, 因为每产生一个成功的值, 看上去是随机的, 其实是从前面数值预测而来的 (用复杂的公式)。任何随机序列都是由一个种子值决定的, 也就是说, 对于同样的种子值, 产生的随机序列都是一样的。`TDSEED` 变量允许你指定一个种子值来产生随机的定时定标序列, 以保证各个仿真运行一致。这样避免在成功的仿真运行时由于随机产生延时的出现和消失带来定时设计错误。

`TDSEED` 变量只能设置 1—32767 以内的正整数。`TDSEED` 的默认值本身是一个随机数 (由时间和日期决定), 这意味着一个仿真运行产生的一系列值相对于下一仿真是随机的。

例如对如下赋值:

`TDSCALE = RANDOM`

`TDLOWER = 2.00`

`TDUPPER = 3.00`

`TDSEED = 723`

使所有定时属性值随机地增加了 200—300%。种子值是 723, 它使得每次仿真运行都产生同样的伪随机序列。

## INITSEED

`INITSEED` 属性用于作为随机初始化值发生器的种子, 被那些初始化属性设定为 `RANDOM` 关键字的 `DSIM` 原型所使用。

与前面提到的 `TDSCALE` 和 `TDSEED` 属性一起, 由随机初始化值产生器生成的序列是随机的, 序列整体是有限和确定的。`INITSEED` 属性意味着 `DSIM` 仿真器选择哪个序列随机数。

`INITSEED` 属性只能被赋予 1—32767 范围内的正整数。其默认值本身是一个随机数 (由时间和日期决定), 这意味着一个仿真运行产生的一系列值相对于下一仿真是随机的。

## 11.9 仿真模型的类型

### 11.9.1 如何描述器件含有模型

`PROTEUS` 有超过 8000 个库部件, 其中约 6000 个含有仿真模型。没有仿真模型的器件适用于 PCB 设计, 要求所有器件都有仿真模型是不现实的。也就是说, 设计类似 68020 处理器模型是没有太多实际价值的。

为了达到电路仿真的目的，你必须明白器件是否具有仿真模型。在第一个实例中，如果仿真就会失败，并有如下提示信息：

```
ERROR [PSM] : No model specified for 'U1'.
```

该错误是在电路分区阶段（PSM）检测出的，因为仿真电路部件中不允许存在没有模型的器件。有时也会出现一下信息：

```
ERROR [U1] : Value '74F00' of VALUE not found in parameter mapping table.
```

这意味着该器件存在模型文件，但是你改变的器件类型值与 MDF 文件不相符。在上例中，我们把 74LS00 门电路（已经建模）改成了 74F00。

器件适用的仿真模型的类型显示在器件库浏览器预览窗口的右上角，例如，如果你打开器件库浏览器并在 74LS 库中选择 74LS00 器件，你会看到

```
Schematic Model [74NAND.MDF]
```

更多关于不同的模型类型的信息将在后面讨论。

## 11.9.2 原型模型（Primitive Models）

PROSPICE 中有大量的基本器件模型。这些器件类型被称为原型（Primitives），包括电阻、电容、二极管、三极管、逻辑门、计数器、锁存器、存储器等等。

原型模型（Primitive models）仿真时不需要额外的文件，只需要一个 PRIMITIVE 属性来识别。其他属性直接在器件上指定并通过网表传递给 PROSPICE。例如一个电阻的属性：

```
PRIMITIVE=ANALOG,RESISTOR
```

这定义此部件是一个 SPICE 电阻原型。

仿真原型的标准集存在于 ASIMMDLS 和 DSIMMDLS 库。这些部件都有属性的文字帮助，它们使用的实例参见 VSM SDK 文档。

## 11.9.3 原理图型模型（Schematic Models）

当仿真一个更加复杂的器件时，通用的做法是用仿真原型画一个描述其行为的电路。这个电路可以是器件的内电路，但是为了加速仿真，我们通常使用理想电流源、电压源和开关。

原理图模型由 MODFILE 属性来设定，为了方便我们创建了这个只读的属性。例如，对于 741 运放有如下设置：

```
MODFILE=OA_BIP
```

从这里你会发现一个模型文件可用于若干个器件——通过参数映射表（Parameter Mapping Table）来区分它们的功能。

原理图模型是在 ISIS 中绘制原理图创建的，通过 Model Compiler 生成 MDF 文件。更多的信息参见 VSM SDK 文档。

## 11.9.4 VSM 模型（VSM Models）

VSM 模型也是原型模型，不过它们使用 PROSPICE 以外的 DLL。这使得我们可以使用特定的编程语言来描述仿真器件的功能，通常最容易的是使用 C++。

一个 VSM 模型既有 PRIMITIVE 属性（因为 ISIS 和 PROSPICE 均把它当作原型），又有指定器件相关 DLL 文件名的 MODDLL 属性。

例如，8052 模型属性：

```
PRIMITIVE=DIGITAL,8052
MODLL=MCS8051
```

注意模型 DLL 文件可以用于多个原型类型——MCS8051.DLL 可应用于许多 8051 系列的处理器。

VSM 模型还可以应用于与动画相关的模型，这样使得器件操作上电气的和图形方面的表现惊人的一致。LCD 显示器就是最好的印证。

创建 VSM 模型涉及一系列 C++接口类（与 COM 类似）。这些参见 VSM SDK 文档。

## 11.9.5 SPICE 模型（SPICE Models）

由于 PROSPICE 基于伯克莱的 SPICE3F5，因此直接兼容标准的 SPICE 模型，PROTEUS 中许多器件都是由制造商提供的 SPICE 文件来创建的。SPICE 模型既可以通过 SUBCKT 块设定，也可以通过 MODEL 记录中一系列参数来设定。以 CA3140 为例，SUBCKT 模型将具有以下属性设置：

```
PRIMITIVE=ANALOG,SUBCKT
SPICEMODEL=CA3140
```

三极管 BC108 的 SPICE 原型模型有以下属性：

```
PRIMITIVE=ANALOG,NPN
SPICEMODEL=BC108
```

模型本身可以保存在一个 ASCII 文件中或一个 SPICE 模型库中。这些文件的名字在 SPICEFILE 或 SPICELIB 属性中设定。

更多内容参见“SPICE 模型的应用”一节。

## 11.10 如何提高交互仿真的速度

### 11.10.1 前言

虽然 Proteus VSM 可以进行许多实时交互仿真，但是显然不是对所有电路均如此，我们可以很完美地画出一个振荡在 1GHz 的电路，但是它不可能在一台机器指令周期超过 1ns 的计算机上实时仿真。在这个单元，我们将解释仿真复杂性的细节以及如何优化电路来最大限度地加速仿真。

### 11.10.2 使用数字的电阻和二极管模型

首先，我们必须知道 ProSPICE 中模拟和数字仿真的差别。因为数字电路仿真比模拟电路仿真快 2~3 个数量级（几乎 1000 倍）。因此，ProSPICE 包含一个数字仿真器，它以事件驱动过程来表示数字器件的操作，从而避免了大量不必要的计算。

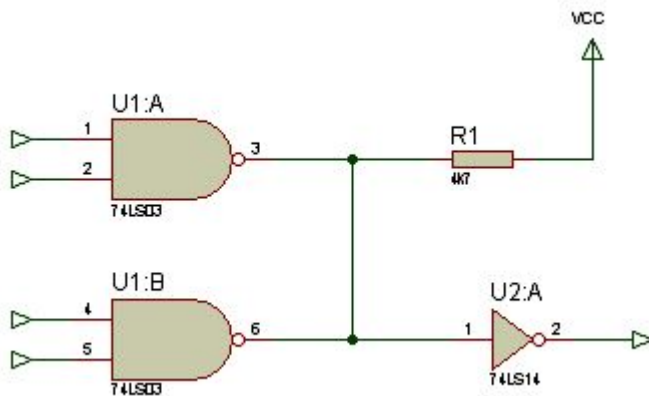
例如，一个 600MHz 的奔腾 III 处理器可以每秒仿真大约二百万个数字事件，同样的电脑在 CPU 负荷达到 100% 时却只能仿真 2kHz 下的正弦波振荡器。这样的波形需要每秒计算 60,000 个模拟测试点，每个测试点又需要通过电路节点方程的一组收敛解来获得——这比处理数字事件复杂很多。

对于许多器件，要认真考虑需要的是数字的还是模拟的仿真。例如，几乎所有的 TTL 和 CMOS 部件都是用数字模型表示的，而模拟 IC 如运放、比较器等等是用模拟模型表示的。所有用标准 SPICE 模型表示的器件都需要模拟仿真。

然而却出现一个灰色地带，某些器件现实是模拟器件，但是在特定情形可以表示为数字模型。二极管甚至电阻就落入了这个范围。这与线或逻辑、上拉电阻、集电极开路输出器件以及二极管——电阻逻辑网络密切相关。

#### 实例 1 线或逻辑

下面电路表示了一个典型的线或逻辑网络：



U1:A 和 U1:B 具有集电极开路输出，只能吸入电流。逻辑 1 输出导致一个高阻状态。按照 DSIM，

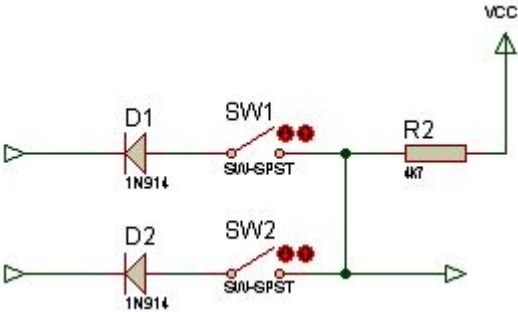
这意味着门输出驱动一个 SL0（强低电平）或者一个 FLT（浮空）逻辑状态。如果电阻 R1 是模拟模型，PROSPICE 必须在电阻和逻辑门之间插入一个数模接口对象，并在电阻和 U2:A 的输入之间插入模数接口对象。这将导致对该节点起伏波形和流经 R1 电流的精确仿真，但是也会造成每当 U1:A 或 U1:B 输出状态改变时巨大的计算量。

所有这些可以通过对 R1 使用数字模型来避免。在这种情形，它的行为是将 VCC 的 SHI 逻辑状态转换为 WHI（弱高电平）逻辑电平。当两个与非门拉低时，SL0 压倒电阻上的 WHI 状态，网络状态变成低电平。但是当两个门都不吸入电流，WHI 状态击败 FLT 状态，网络升高为逻辑高电平。所有这些受控于数字仿真阶段，不需要模拟仿真。

因此，任何时候的上拉电阻都应该是数字模型的。

### 实例 2 二极管电阻网络

另一种模拟电路可以使用数字模型的情况就是下面所示的二极管电阻网络。这经常出现在矩阵扫描键盘电路中，二极管用于防止多个按键同时按下时列驱动电路之间发生短路。



就像在线或逻辑例子那样，ProSPICE 会很自然地用模拟的方式建模仿真，但是必将导致巨大的计算量。如果上面使用数字电阻模型，二极管被当作只从阴极传递逻辑低电平到阳极和只从阳极传递高电平到阴极，那么上面电路同样就以数字模型建立了。

由于键盘扫描程序是以固定时间间隔进行的，常常使用大量的二极管和电阻，因此这种优化是极其重要的。

### 如何选择数字电阻模型

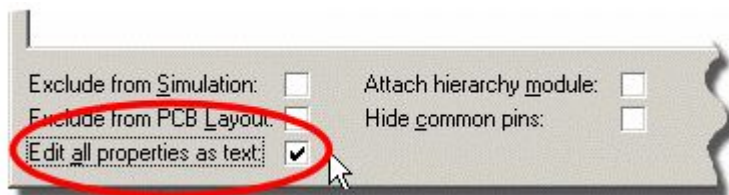
1. 鼠标指向需要改变属性的电阻，按 CTRL+E。
2. 点击 Edit All Properties as Text 选择框。
3. 改变 PRIMITIVE 属性为：

PRIMITIVE=DIGITAL,RESISTOR

如果是新建电路并且知道那些电阻需要数字模型的，你也可以通过从元件库中直接使用 PULLUP 或 PULLDOWN 模型。这些器件已经具有上述属性。

### 如何选择数字二极管模型

1. 鼠标指向需要改变属性的二极管，按 CTRL+E。
2. 点击 Edit All Properties as Text 选择框。

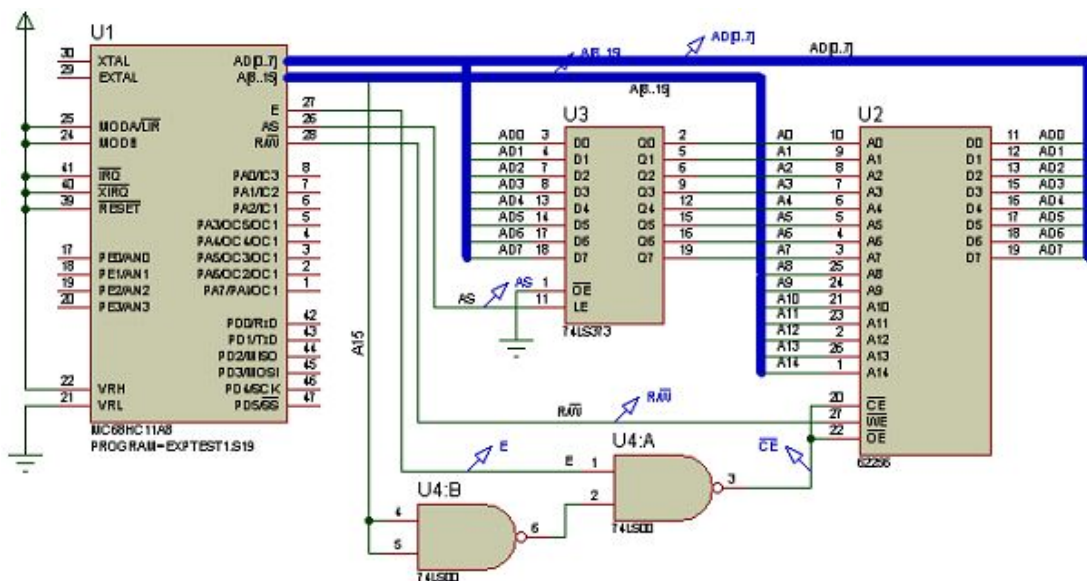


3. 改变 PRIMITIVE 属性为:
4. PRIMITIVE=DIGITAL,DIODE
5. 删除所有 MODEL 属性, 因为 SPICE 参数此时没有意义。

如果是新建电路并且知道那些二极管需要数字模型的, 你也可以通过从元件库中直接使用 DIODE-DIGITAL 器件。这个器件已经具有上述属性。

### 11.10.3 优化外部 RAM 和 ROM 的访问

许多较大型的单片机设计会用到外部的 ROM, RAM 或 EEPROM。它们用于存储代码或扩展单片机的 SRAM。给定一个正确的地址译码电路, 假设存储器有仿真模型, Proteus VSM 可以正确仿真如下电路。当 CPU 访问外部存储器时, 模型将驱动相应的地址、数据和控制线, 外部译码逻辑和存储器将对相应位置的读写出反应。



如果你希望校验你的译码电路工作, 那么这是非常有用的, 但是同样会带来很大的计算量开销。设置一个 16 位地址将产生最少 16 个数字事件, 从总线上读或写一个字的数据又会产生 8 个数字事件。由于地址线和数据线是复用的, 如上 HC11 的设计, 将会产生更多的数字事件。所有这些与 VSM CPU 每机器周期运行一条指令的能力相比是极不适宜的。

因此, 我们提供的 CPU 模型能够以内部访问的形式来仿真外部存储器的访问。目前应用在 8051、HC11 和较大的 AVR CPU 模型。外部存储器可以用 EXTRAM 或 EXTROM 属性加以声明, 这些属性设定了存储器的每个存储块的范围。全部细节见特定芯片的相关模型帮助文档, 可以通过 Edit Component 对话框或开始菜单访问。

一旦外部存储器这样定义了, 对特定区间外部存储器访问的指令就可以不需要产生大量的数字



事件来仿真。对存储器映射的外设访问也仍然可以完全仿真，因为它们位于 EXTRAM 和 EXTROM 设定的存储范围之外。

# 第 12 章 错误处理

## 12.1 仿真日志

每当运行一个仿真就会生成一个仿真日志（Simulation Log）。仿真日志文件包括所有有关仿真器和个别模型的信息、警告和错误消息。当一个模型产生一个消息，消息前缀是以方括号包括的模型器件参考名称，因此你也许会看到：

[U1] Loaded 26 files from PROGRAM.HEX

在交互仿真期间，这个文件的内容可以从 Debug 菜单下的弹出窗口看到，在图形仿真期间，这个文件可以通过鼠标指向图表并按下 CTRL+V 看到。

当仿真失败时，日志文件会自动出现使你可以立即知道问题所在。

## 12.2 网络表错误

网络表错误发生在 ISIS 为原理图创建网络表时出现问题，你也会在试图从原理图到 ARES 进行 PCB 设计时遇到，一般问题在于：

- 有两个同名的器件，或未命名器件，例如两个电阻为 R？
- 脚本文件格式（如 MAP ON 表）不对。如果你不能立即判断出问题，请参考 ISIS 手册中的语法定义。

## 12.3 链接错误

模型链接发生在 ISIS 为基于等效电路建模的器件调用 MDF 文件。目前共同的问题是指定的器件没有模型文件。模型文件必须在当前设计目录下，或在 Module Paths 中 Set Paths 设置的路径下。

其他常见的链接错误包括：

- 值在参数映射表中找不到。意思是部件类型——如 CA3140 没有列在特定模型文件的映射表中。像 OA\_MOS.MDF 这样的模型文件是为若干个有相同电路但不同参数的电路设计的。这个错误意味着该模型文件没有包括你指定的器件类型的参数，你可以用文本编辑器打开 MDF 文件看看建模了哪些器件。
- 无法解释的模块引脚。这是个警告而不是错误，意味着父器件体有一个引脚在模型中不存在。这个一般没有问题，例如，许多运放没有对调零和空脚建模。但是对一个新模型不能工作，这也许是个常见的错误，特别是同时出现 “No DC path to Ground” 的警告时。

## 12.4 分区错误

分区是一种机制，通过分区 ISIS 决定哪些部分电路需要仿真。这部分可能出现的问题有：

- 检测到循环依赖。意思是录音机的布置使得前后的分区相互依赖。假定你已经正确地设置了探针和激励源的 Isolate Before 和 Isolate After 标志，最简单的处理办法就是删去录音机，对全电路仿真。
- 没有指定模型。意思是提示的器件没有 PRIMITIVE、MODFILE 属性并出现在你的仿真电路中。如果器件与仿真无关（如连接器），那么你可以指定其属性 PRIMITIVE=NULL，否则你就需要一个模型！

详细信息参见“仿真模型的类型”。

## 12.5 电源范围定义错误

所有设计中的电源网络必须通过原理图或 Power Rail Configuration 绑定到电源上。如果存在未绑定的电源网络，仿真日志就会出现相应的错误信息。

注意连接电池或 SPICE 电压源到电源网络也不足以避免这个错误，你必须定义一个电源并绑定到 Power Rail Configuration 对话框中的网络。

## 12.6 仿真错误

仿真错误产生于 PROSPICE 而不是 ISIS，故常发生在网表文件成功产生之后。常见问题包括：

- 器件类型无法识别—意味着你设定了一个不支持的原型类型，或者一个模型文件使用了那样的原型。
- 没有对地直流通路—在“地和电源范围”中讨论。
- 没有发现探针—你试题参考一个不存在的探针或电压发生器。记住你必须使用 ASIMMDLS.LIB 中的 IPROBE 对象—你不能参考一个电流探针元件。
- 无法打开 SPICE 源文件—SPICEMODEL 属性设定的源文件无法找到。它必须在当前设计目录下或在 Module Path 中 the Set Paths 设置的路径下。
- 找不到模型库—你设定的 SUBCKT 或 MODEL 在磁盘指定的目录中找不到。
- 模型 DLL 文件找不到—设定的 VSM 模型 DLL 无法定位。它必须在当前设计目录下或在 Module Path 中 the Set Paths 设置的路径下。

## 12.7 收敛问题

最后一些问题与 SPICE 自己仿真电路时失败的原因有关。有一下三个基本的错误消息：

- 奇异矩阵。等价于未知数数目超过方程数，通常和绘图错误有关，或者某些需要设定初始值来确定起始状态的场合。

**i** 这个错误经常出现在 “No DC path to ground” 警告之前，你需要调查消息后面列出的引脚的连线情况。如果你的某部分电路没有接地，仿真器会解释为其电压是对地的——就这么简单。

- 多次迭代也不能收敛。这意味着电路的解是不稳定的。带有 VSWITCH 或 CSWITCH 原型的电路容易出现此问题，但是任何具有不连续传输函数的电路都会造成 SPICE 严重的问题。
- 时间步长太小。这意味着电路切换在这样一种状态：在充分小的时间步长下（典型如  $1\text{E}-18$  秒）仍然无法产生可接受的足够小的电路电压变化。

**i** 这个经常由模型设计不当产生，或者没有提供给二极管或三极管足够的参数。特别是，如果没有正确地选择结电容值，这些器件会呈现零开关时间，那将直接导致这个错误消息。

大多数收敛错误是因为绘图错误或错误的模型——我们收到很多“不能仿真”的电路，却只发现某些地方少了连线。请以仿真日志为线索，在得出 PROSPICE 出错的结论前重新检查你的电路。

- 当使用第三方 SPICE 或 VSM 模型时，除非你提供简单的说明电路，否则我们是不可能花时间去调试它们的。

还要防止使用那些功能不被 SPICE2 或 SPICE3 支持的第三方 SPICE 模型。为 PSPICE 开发的模型具有的元件风格和语法结构都不是标准的 SPICE。

振荡器因为工作点的初始解失败而产生特殊的问题。总之，振荡器是没有稳态的！在“初始条件”部分讨论了使用 IC、NS 和 OFF 属性确定初始状态的方法。

如果问题确实在于数值收敛上，你可以尝试一下办法：

- 增加 GMIN 的值。这是反偏 PN 结的漏电阻，降低该值使电路看上去更接近于电阻网络（肯定有解）。但是以降低精度为代价。默认值是  $1\text{E}-12$ ；超过  $1\text{E}-9$  的值将导致没有意义的结果。注意在任何情况下如果开始电路不收敛，SPICE3F5 会尝试 GMIN stepping。意思是较大的 GMIN 用于寻找初始解，然后 GMIN 回到原值以获得精度。
- 增加 ABSTOL 和/或 RELTOL 的值。这些值控制着仿真收敛的精度。然而，你使得误差变大，结果的精度实际上就降低了。
- 如果电路中使用了运放，尝试设置 MODFILE=OA\_IDEAL，取代早期器件设定的类型。
- 降低 TRTOL 的值。这将使 SPICE 使用更小的步长，便不容易漏掉收敛解，但是代价是更长的运行时间。这个方法只用于瞬态分析仿真部分失败的情形。
- 如果绘出来的图线过于“锯齿”或含有数学噪声，你也可以尝试降低 TRTOL。这经常表现为某个值在快速电平变化时产生的振荡。