

第四章 验证与 VCS 使用

本章将讲述的内容：

第一节 验证

- 。什么是验证
- 。为什么需要验证
- 。验证的重要性
- 。如何进行验证

第二节 VCS 简单使用方法

2.1 什么是 VCS

2.2 VCS 可以做什么

2.3 怎样进行验证

2.4 VCS 的工作方式

2.5 VCS 使用方法 举个简单例子

2.6 VirSim 的图形方式和每个窗口的介绍

附录 A. VCS 的参数

附录 B. virsim 简明帮助

附录 C. simv 简明帮助

第一节 验证

当代码编写完之后，怎么确定是正确的呢，代码能不能符合设计要求，能不能完成所需要的功能，这就是验证所要做的工作。验证在设计中有很重要的地位，从设计流程中可以看到，几乎设计工作每前进一步，都要进行验证。

对验证的要求，大多数人认为只要编译通过之后，能实现功能就可以了，其实决不仅仅这么简单，验证的目的应该是尽量多的找到代码中的错误，不管是编写错误还是功能错误，找出的错误越多，验证工作就做的越好越好。

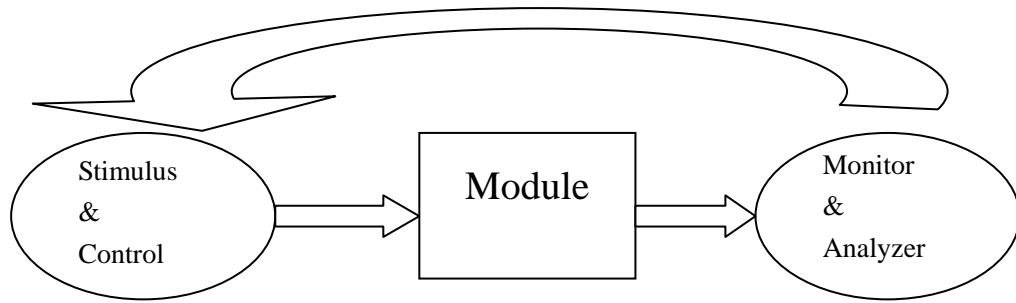
既然验证这么重要，如何进行验证呢？对于验证来说，不同等级的验证，它的方法是不一样的。什么是验证的等级，从设计流程（下图）可以看到，验证可以大致分为单独子模块验证、功能模块验证、系统顶级验证。

- 。单独子模块验证，需要做的工作是验证它的功能和逻辑是否符合设计要求
- 。功能模块验证，需要验证这个模块的功能可不可以满足要求，是否会有非法数据或不该有的输出，错误的状态等。
- 。系统顶级验证，更关注于系统整体的行为方式，模块间的联系和通讯，总线信号，数据流路径是否满足设计要求，数据处理或时序正确与否等。

验证需要一个支持的平台，这就是 `test_bench`，在这个测试平台上，有激励信号产生器、被测模块、响应分析和监测器，（下图）

激励与控制：输入端口设置，测试向量，测试模式设置，同步。

响应分析器和监测器：可以及时监控输出信号变化，可以判断输出信号是正确、合法、错误、非法等等。



testbench 可以用 verilog 描述语言搭建，也可以用 C 语言编写，如果用 C 语言编写，还需要相关的编译器并和与 verilog 的接口。

第二节 VCS 的简单使用方法

2.1 什么是 VCS

VCS 的全称是 Verilog Compile Simulator，是 Synopsys 公司的强有力的电路仿真工具，可以进行电路的时序模拟。

2.2 VCS 的工作方式

VCS 运行首先把输入的 verilog 源文件编译，然后生成可执行的模拟文件，也可以生成 VCD 或者 VCD+ 记录文件。然后运行这个可执行的文件，可以进行调试与分析；或者查看生成的 VCD 或者 VCD+ 记录文件。还生成了一些供分析和查看的文件，以便于调试。

2.3 怎样进行仿真和验证

仿真测试一个模块的大致步骤如下：

- (1) 首先需要编写好模块的 verilog 代码。
- (2) 搭建 testbench，充分了解被测模块的特性，编写测试向量，输入端口的激励，编写响应分析和监测部分。
- (3) 运行 VCS 进行模拟，查看输出或者波形。
- (4) 若发现错误，分析错误类型和原因，修改代码或者修正测试方法，直到符合测试要求。

2.4 VCS 的运行方式

VCS 的运行方式有两种，一种是交互模式 (interactive mode)，一种是批处理模式 (batch mode)，两种方式各有优劣，具体用在不同的情况下。在测试小模块或者底层模块，情况不太复杂的时候，而又需要很详细信息的时候，可以采用交互模式，交互性能更好，显示更直观；当进行复杂测试而关注于整体性能，而不必去查看每个信号的时候，只需要查看所需要关心的信号即可，这种情况可以用批处理模式。

2.5 VCS 简单使用例子

下面用一个简单的例子来说明如何使用 VCS

这是个四位全加器的 verilog 代码，存储为 add4.v,

```
module add4 (clk,in1, in2, sum, carry);
```

```
output [3:0]    sum;
```

```
output          carry;
```

```
input           clk;
```

```
input [3:0]     in1, in2;
```

```
reg  [3:0]      sum;
```

```
reg             carry;
```

```
integer         temp;
```

```
initial begin
```

```
    sum = 0;
```

```
    carry = 0;
```

```
end
```

```
always @(posedge clk) begin
```

```
    temp = in1+in2;
```

```
    sum=temp;
```

```
    if (temp > 15)
```

```
        carry = 1;
```

```
    else
```

```
        carry = 0;
```

```
end
```

```
endmodule
```

然后再根据这个模块写一个测试模块，也称之为 testbench，存为 top.v,

```
module top;
```

```
reg      clk_reg;
```

```
reg [3:0] in1_reg, in2_reg;
```

```
wire [3:0]    sum;
```

```
wire          carry;
```

```
addr4 a4 (clk_reg,in1_reg, in2_reg, sum, carry);
```

```
parameter d = 100;
```

```
initial    begin
```

```
    clk_reg=0;
```

```
    in1_reg = 0;
```

```
    in2_reg = 0;
```

```
    repeat (16*100)    begin
```

```
        #d in1_reg = in1_reg+1; in2_reg = in2_reg+1;
```

```
//        $display($stime,,"in1_reg +%d    in2_reg+ %d = sum %d    carry is    %d", in1_reg,
```

```

in2_reg, sum, carry);
    //      $display($time,," %b +  %b = %b and carry is %b", in1_reg, in2_reg, sum,
carry);

    end

    //      $strobe($time,,"in1_reg %b  in2_reg %b  sum %b  carry %b", in1_reg, in2_reg,
sum, carry);

    #1
    $finish(2);
    end

always
begin
    #50 clk_reg= ~clk_reg;
end
always @(sum) begin
    //$display($time,,"in1_reg +%d  in2_reg+ %d = sum %d  carry_reg is  %d", in1_reg,
in2_reg, sum, carry_reg);
    $display($time,,"now at a clock posedge,the operation is :: %d +  %d = %d and carry is
%d", in1_reg, in2_reg, sum, carry);
    //$stop;
end
endmodule

```

最简单的仿真，只要运行 `vcs filename` 即可，*filename* 可以有很多个，比如上面的例子：

`vcs top.v add4.v`

然后，如果没有发现编译错误，将会出现 VCS 的说明和一些信息，接下来的就是它的执行情况，翻译 `top.v` 和 `add4.v`，可以自动发现顶层模块，如果定义了 `timescale` 将显示用的 `timescale` 信息，如果没有，就显示 `No TimeScale specified`。然后将产生一个名为 `simv` 的可执行文件，这个就是模拟仿真文件。

下面是运行结果：

```

Parsing design file 'top.v'
Parsing design file 'add4.v'
Top Level Modules:
    top
No TimeScale specified
1 of 2 unique modules to generate
1 of 1 modules done
Invoking loader...
simv generation successfully completed

```

下面我们来运行一下这个 `simv` 可执行文件，

`simv <cr>`

结果将显示：

Chronologic VCS simulator copyright 1991-2001

Contains Synopsys proprietary information.

Compiler version 6.0; Runtime version 6.0; Jan 8 15:37 2002

```
0 now at a clock posedge,the operation is :: 0 + 0 = 0 and carry is 0
150 now at a clock posedge,the operation is :: 1 + 1 = 2 and carry is 0
250 now at a clock posedge,the operation is :: 2 + 2 = 4 and carry is 0
350 now at a clock posedge,the operation is :: 3 + 3 = 6 and carry is 0
450 now at a clock posedge,the operation is :: 4 + 4 = 8 and carry is 0
550 now at a clock posedge,the operation is :: 5 + 5 = 10 and carry is 0
650 now at a clock posedge,the operation is :: 6 + 6 = 12 and carry is 0
750 now at a clock posedge,the operation is :: 7 + 7 = 14 and carry is 0
850 now at a clock posedge,the operation is :: 8 + 8 = 0 and carry is 1
950 now at a clock posedge,the operation is :: 9 + 9 = 2 and carry is 1
1050 now at a clock posedge,the operation is :: 10 + 10 = 4 and carry is 1
1150 now at a clock posedge,the operation is :: 11 + 11 = 6 and carry is 1
1250 now at a clock posedge,the operation is :: 12 + 12 = 8 and carry is 1
1350 now at a clock posedge,the operation is :: 13 + 13 = 10 and carry is 1
1450 now at a clock posedge,the operation is :: 14 + 14 = 12 and carry is 1
1550 now at a clock posedge,the operation is :: 15 + 15 = 14 and carry is 1
1650 now at a clock posedge,the operation is :: 0 + 0 = 0 and carry is 0
1750 now at a clock posedge,the operation is :: 1 + 1 = 2 and carry is 0
1850 now at a clock posedge,the operation is :: 2 + 2 = 4 and carry is 0
1950 now at a clock posedge,the operation is :: 3 + 3 = 6 and carry is 0
2050 now at a clock posedge,the operation is :: 4 + 4 = 8 and carry is 0
2150 now at a clock posedge,the operation is :: 5 + 5 = 10 and carry is 0
2250 now at a clock posedge,the operation is :: 6 + 6 = 12 and carry is 0
2350 now at a clock posedge,the operation is :: 7 + 7 = 14 and carry is 0
2450 now at a clock posedge,the operation is :: 8 + 8 = 0 and carry is 1
..... (略)
```

```
$finish at simulation time 160001
```

V C S S i m u l a t i o n R e p o r t

Time: 160001

CPU Time: 0.100 seconds; Data structure size: 0.0Mb

Tue Jan 8 15:37:31 2002

验证输出的这些信息，就可以发现，模块的设计是正确的，满足和全加器的设计要求。
注意到 top.v 文件中的注释掉的几行，也可以用其他方式显示，特殊情况需要特殊处理。

2.6 图形方式的 VCS

下面用图形方式启动 VCS，可以进行更方便的控制和监测，
运行 `VCS -RI top.v add4.v` (-RI 的选项就是打开交互式图形界面，是指 Run Interactive)
运行上面的命令会打开 VirSim 主程序，Interactive 窗口会自动打开。

2.6.1 VirSim 概况

Virsim 是基于 OSF/Motif 的图形化仿真调试系统，Virsim 可以和 VCS、EPIC powermill 以及 Timemill 一起协同工作。利用 Virsim 与 VCS 交互式的工作方式可以在模拟的过程中显示仿真结果，结果可以存到一种叫做 VCD+的文件中。

这种图形化的调试系统可以支持三种基本的调试方式：波形、结构和代码。这几种方式可以同时地使用。支持标准 Verilog 的所有函数、语法、系统调用和编程语言接口（PLI, Programming Language Interface）。Virsim 是一个多窗口调试系统，可以允许用户根据需要打开很多个调试窗口。

VirSim 可以有两种方式运行：交互模式（interactive mode）和后处理模式（post-processing mode）。第一种模式允许实时的控制仿真的进行，允许在模拟的过程中改变寄存器的值或者设置，这些改变会实时地影响到模拟的结果。第二种模式先输出用户指定选择的信号及其变化过程到一个文件中，然后可以用 VirSim 来分析这个文件。该文件是 VCD+类型的，VCD+文件是一种二进制的格式，里面记录了 VCS 模拟的结果，和信号的变化历史等信息。

下面这就是VirSim的主窗口：



VirSim是一个整体集成包，有六个相互有关联关系的工具，主要的工具就是交互环境 interactive窗口，还有可以查看模块结构的hierarchy窗口，查看源代码的source窗口，查看逻辑连接的logic窗口，查看波形状况的有waveform窗口，查看寄存器、变量等的register窗口。通过点击VirSim主窗口的按钮可以打开这几个窗口，也可以在这几个窗口里面通过菜单或者快捷键打开其他的窗口。

2.6.2 interactive 窗口

下面这个是 interactive 窗口,可以看到，模拟刚开始先停在 0 时刻。

在 interactive 窗口，在这个窗口中，可以进行随时的交互式的控制，随时监控电路的模拟状况。如果已经写了输入信号序列，如果在 testbench 中写了 \$stop，可以执行到这个硬断点。

这里说明几种调试中的断点类型：硬断点，是调用了 verilog 的系统函数 \$stop 的这类断点；软断点，是在交互环境中用 tbreak 命令产生的断点；信号变化断点，是在模拟过程中定义了敏感信号，当这个信号一旦有变化，就会中断模拟过程。设置合适的断点是调试的技巧和重要的手段。



2.6.3 hierarchy 窗口

点击主窗口中的 **hierarchy** 按钮打开 **hierarchy** 窗口，在这个窗口中系统会用不同的颜色来表示设计的层次结构，可以表示出来的有：模块、任务、函数、有名块、信号、寄存器、线网、输入输出等。这个窗口可以认为是一个查看器（brower），用户可以用鼠标把需要查看的对象拖动到其他相应的窗口中，后面会讲述到怎么拖动和如何查看。

这个窗口有三个区域，结构查看区域、搜索区域、选择区域，如下图所示。



- (1) 结构查看区域：在这里可以查看设计中的各个模块层次结构，选择、拖拉，这个区域提供两种查看方式：单级（one level）和多级（multi level）。点击菜单的display可以切换。在单级方式下可以依次地打开每一级，在此区域中只显示当前的模块下属的子模块。如下所示，根模块是top，下面只有一个子模块uar，在uar的下面有五个子模块，下面三幅图是依次打开三级模块：




如果在多级方式下，就会在这个区域里面显示多级结构，这样结构化更明显，但是


如果设计太复杂，会显得比较杂乱，显示的情况如下所示。两种方式各有优劣。



在这个窗口中，可以定义一些书签（book mark），比如在一个模块上点右键，然后选择

create bookmark，然后点击 ，就可以看到增加了一个刚才所添加的模块的项，这样的方法在模块比较复杂的时候很方便就可以到所要查看的模块了。

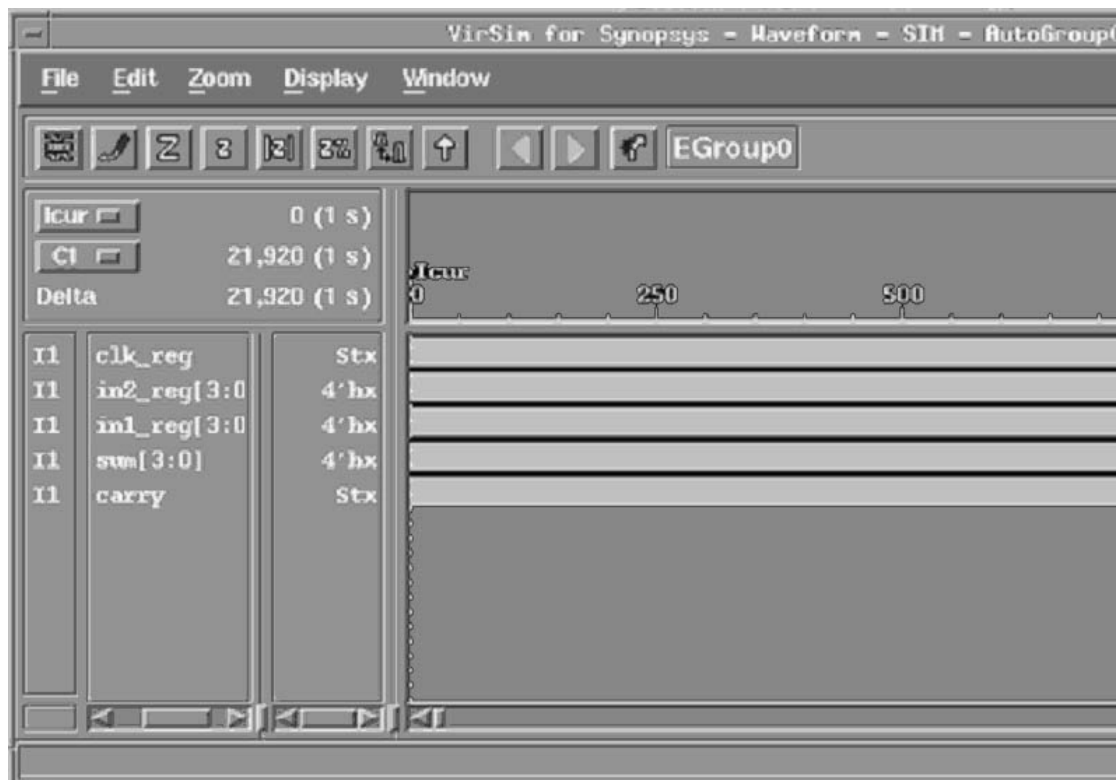
- (2) 搜索区域：在这个区域里面，可以寻找符合指定样式的信号或者模块名。搜索的范围有三种，所有（All）、子模块（children）、当前所选模块（selected）。搜索到的会在搜索区域中显示出来。可以用*匹配所有字符，用?可以匹配一个字符
- (3) 选择区域，在这里，有一个过滤器，可以指定显示哪些的信号或者端口名。用*可以匹配所有字符，用?可以匹配一个字符。

这里有个选择框 ，S表示的是信号（signal），P表示的是端口（port），表示显示的包含信号还是端口或者都包括。

总之在这个 hierarchy 窗口中可以查看模块的层次结构，还可以查看到每个层次的寄存器、变量、线、输入输出端口等。这个窗口中的内容有的可以拖动到其他的窗口中，比如，打开 waveform 窗口，在 hierarchy 窗口，把想要查看的变量选中，然后点 add 或者用鼠标中键拖到 waveform 窗口中，因为 virsims 启动的时候电路模拟是停在 0 时刻的，所以看到 waveform 现在的值都还是 X。如果把其他的窗口也打开了，比如把模块或者信号拖到 source 窗口，就会显示包含这个模块或者信号的代码文件；把模块或者信号或者端口拖到 logic 窗口，就可以显示出连接状况；把信号或者寄存器拖到 register 窗口中，可以显示出这个信号或者寄存器的值以及他的变化情况。这样就可以提供了很多种方便而有力的调试和监测方式

2.6.4 waveform 窗口

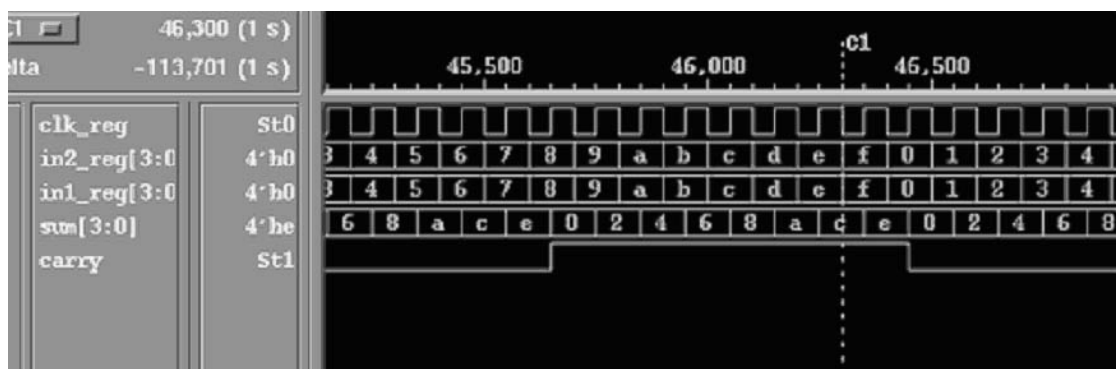
这个窗口可以显示每个信号的波形状况。可以提供指针来指明时刻，可以定义标签来定义断点，用表达式可以指定和跟踪模拟中的事件。还可以允许用户定义表达式、信号组、断点组。这几种方式提供了方便又强大的手段来控制仿真模拟。



waveform 窗口是基于时间线的，可以查看波形，也可以进行即时模拟，让电路运行到某一个信号发生变化然后暂停，也可以一直运行到预先定义的硬断点。比如某个选中信号，然后再在波形上点右键按住不放，选 **next edge**，或者敲击键盘的 **n** 键，即运行到这个信号的下一个变化沿。



如果我们点击一下 interactive 窗口的 ，模拟就会一直执行下去，直到硬断点或者以前设置过了的断点处，同时，看到 waveform 窗口的波形已经出来了。因为注意到在 top.v 中没有设置硬断点，所以会一直执行到结束。




可以看到，每一个时钟上升沿的时候， $\text{sum} = \text{in1} + \text{in2}$ ，carry 为进位输出。

下面介绍几个比较有用的功能，信号分组，标记和断点分组

- (1) 信号分组，可以把几个信号定义成一个组，方法是选择几个信号，然后选择菜单 **edit->Group**，可以定义一个新的组名，再点击 **add** 就创建了一个新的组。也可以把信号加入到原来已经存在的组中，还可以更改组名。

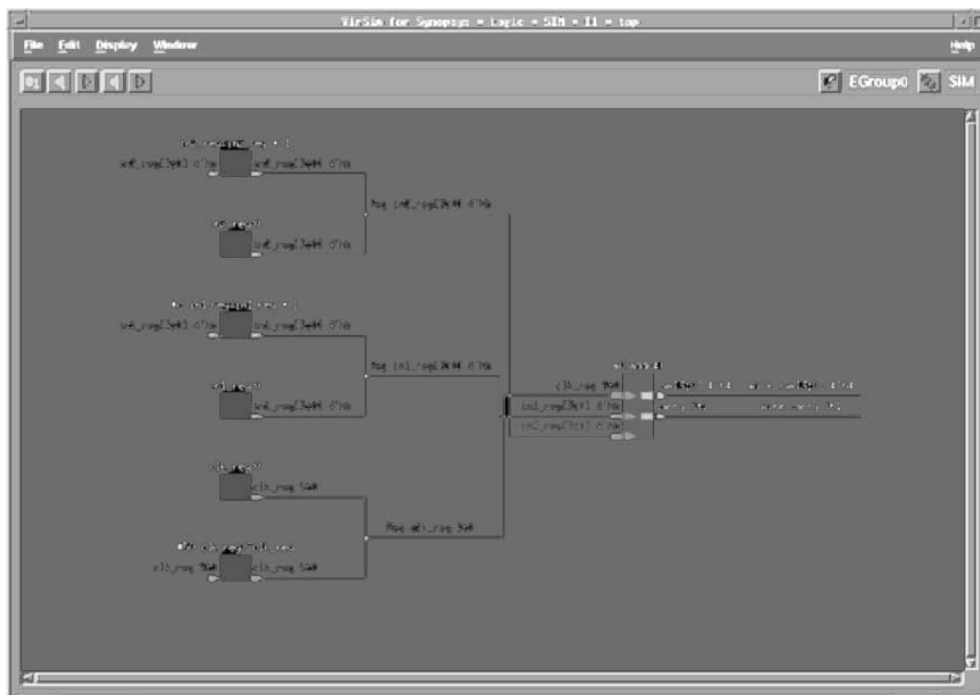




定义好了组之后，在 waveform 窗口中有一个  按钮，按下这个按钮，就可以看到所有的组，可以很快的查看需要的信号。

- (2) 标记，marker 是一个时间标签，可以定义一个时刻的别名。在波形图上，在需要加入 marker 的时刻点击鼠标左键，然后选择菜单 edit->Markers，在弹出的对话框中填写一个名字然后点 add，这时候可以看到波形图上有一条线，线上是 marker 名，如图所示：
- (3) 断点，breakpoint 是定义的一套基于表达式搜索的集合，当表达式成立的时候，VirSim 就会暂停下来。断点可以进行分组，这样可以方便的控制模拟的进行。

2.6.5 logic 窗口

下面看看 logic 窗口，把信号，或者模块用中键从 hierarchy 窗口拖到 logic 窗口中，就可以看的电路的拓扑结构，可以顺着信号线查看他的连接，当模块结构比较复杂的时候，这种方法很方便就可以检查到是否有连接错误。



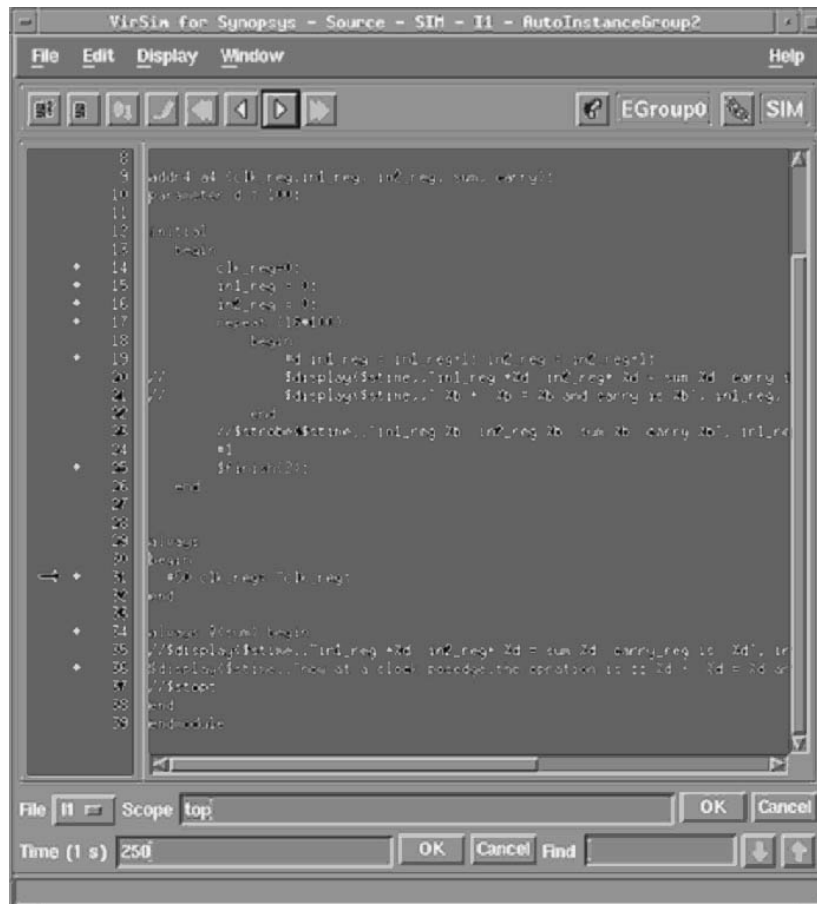
在这个窗口中，有个很方便的功能，就是信号变化软中断，如果选中了某一个信号线或者寄存器，在模拟开始或者任何一种暂停状态下，点击绿色的 ，就会模拟到这个信号的下一个变化时刻，并且暂停下来；倘若没有选择信号，点击红色的 ，就模拟运行到当前视图上的任意一个信号发生变化的时候暂停模拟。两种方式中，waveform 窗口也会同步变化。这样就给了很方便的方式随时监测信号的变化和电路工作方式。


2.6.6 source 窗口



下面再来打开 source 窗口，把信号或者模块从 hierarchy 窗口拖到这里，就马上可以看到相关的源代码了。VCS 还提供单步执行的调试方式，这需要在运行 VCS 的时候加上选项 -line:

`vcs -RI -line filename`


下面显示的是 source 窗口的样子，



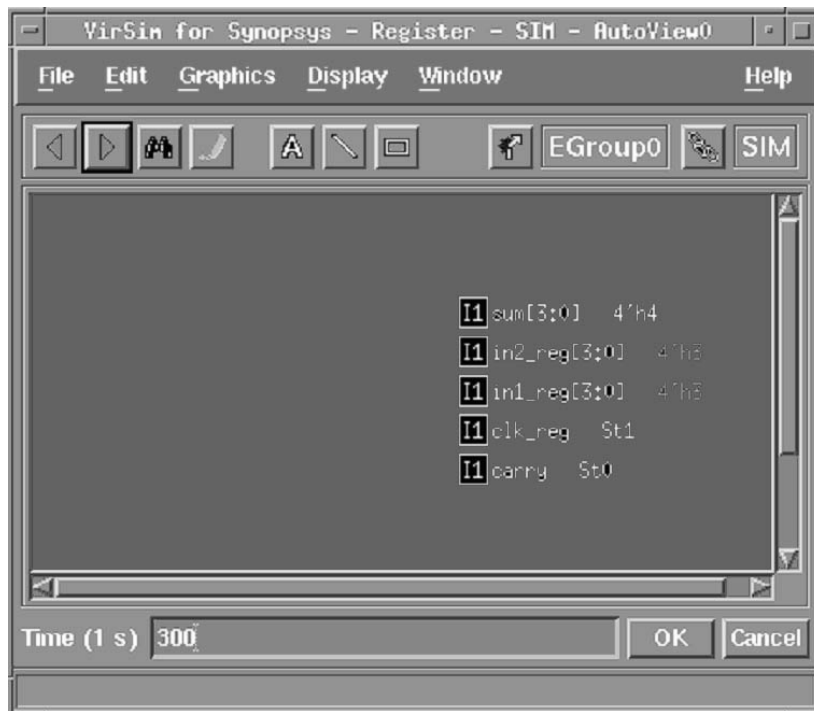
这样，在 source 窗口中，点击黄色的 ，就可以单步执行，可以看到，每一步执行的是哪一条语句，同时，如果打开了 waveform、logic、register 窗口，还可以看到信号也是即时变化的。


在 source 窗口中，可以直接定义断点，在行的前面点击一下，就会有红色的圆点，，这样很方便就设置了一个断点。如果设置了断点，可以看到红色的按钮 ，按下这个按钮，将执行到断点的位置。断点可以设置很多个。

2.6.7 register 窗口

下面打开 register 窗口，在 hierarchy 中选择几个信号或者寄存器，用中键拖到这个窗口中，上面有个红色的  按钮，按下这个按钮，可以让模拟执行到选定的这些信号中的任意一个发生变化，并且会红色高亮显示出来。在这个窗口中，还可以自己画一些辅助图形，比如把信号编组然后放在一起，用矩形括起来，并加上一个标签，这在当模块复杂，需要监

视的信号很多的情况下是很方便的。这些辅助也可以被存储下来，下一次模拟的时候可以方便地直接调用出来。



这几个窗口之间的关系是相互关联的，他们的关联关系都是通过  关联起来的，（当然也可以断开其中之一的关联）。在实际的调试过程中，可以综合运用多种方式，结合起来进行调试，综合运用各种断点，随时监视信号的变化是不是符合设计要求。

总之，VCS 提供了多种调试方法，功能非常强大。另外说明一点，调试工作是很烦杂的，需要对 verilog 很熟悉，还要有很丰富的调试经验，出了问题的时候需要分析是什么问题，再逐步缩小范围，问题查找需要耐心和方法。

附录 A. VCS 的参数:

一般的, 如果不知道参数怎样使用, 可以用命令帮助来看大多数的参数的使用
下面是 vcs -h 的输出结果:

Note: Bumping stack limit from 8192 to 65536 Kbytes.

Below is a summary of the commonly used vcs options and environment variables.
Many of these options have important modifiers. Please refer to the vcs manual
for more information.

Summary of vcs compile options:

```
-----
-ASFLAGS "opts"    pass 'opts' to the assembler
-B                generate long call instructions in native assembly code (HP only)
-CC "opts"         pass 'opts' to C compiler
-CFLAGS "opts"     pass 'opts' to C compiler
-LDFLAGS "opts"    pass 'opts' to C compiler on load line only
-I               enable interactive/postprocessing debugging capabilities
-ID             get host identification information
-M              enable incremental compilation (see manual)
-Mupdate         enable incremental compilation and keep the Makefile up-to-date
-Marchive[=N]    create intermediate libs to reduce link line length; N objs per lib
-P plitab        compiles user-defined pli definition table 'plitab'
-PP             enable optimizer postprocessing capabilities for vcd+
-R              after compilation, run simulation executable
-RI             after compilation, run simulation under xvcs (Implies -I)
-RIG            run simulation under xvcs without compiling (executable has to exist)
-RPP            run xvcs in postprocessing mode (requires file created by vcdpluse)
-V[t]           verbose mode; with 't', include time information
-as foo         use foo as the assembler
-cc foo         use foo as the C compiler
-cpp foo        use foo as the C++ compiler
-e <new_name>    specify the name of your main() routine.
                  (see manual section 7-11 for more details).
-f file         reads 'file' for other options
-gen_c          generate C code (for HP and Sun, default is -gen_obj)
-gen_asm        generate native assembly code (HP and Sun only)
-gen_obj        generate native object code (HP and Sun only)
-ld foo         use foo as the linker. (refer vcs manual for compatibility with -cpp
option)
-line          enable single-stepping/breakpoints for source level debugging
-lmc-swift      include lmc swift interface
-lmc-hm         include lmc hardware modeler interface
```

-vera	add VERA 4.5+ libraries
-vera_dbind	add VERA 4.5+ libraries for dynamic binding
-location	display full pathname to vcs installation for this platform
-vhdlobj <name>	generate a vhdl obj for simulating in a vhdl design
-mixedhdl	include MixedHDL-1.0 interface
-mhdl	include MixedHDL-2.0 interface and library
-q	quiet mode
-platform	display name of vcs installation subdirectory for this platform
-syslib 'libs'	specify system libraries (placed last on the link line) eg -lm
-o exec	name the executable simulation model 'exec' (default is 'simv')
-u	treat all non text string characters as uppercase
-v file	search for unresolved module references in 'file'
-y libdir	search for unresolved module references in directory 'libdir'
+acc	enable pli applications to use acc routines (see manual)
+ad	include anlog simulation interface and library
+adfmi="files"	ADFMI support for vcs-ace
+cliedit	enable command line edit/recall (see doc/readline.ps)
+cli	enable command line interactive debugging (see manual)
+cmod	Enabling cmodule feature
+cmoext+cmodext	Changing cmodule extension to cmodext
+cmoindir+cmoindir	Cmodule Include directory
+cmodefine+macro	define cmodule source 'macro' in the form of XX=YY
+define+macro	define hdl source 'macro' to have value "macro"
+plusarg_save	hardwire the plusargs, which follow this flag, into simv
+plusarg_ignore	turn off +plusarg_save
+prof	tells vcs to profile the the design and generate vcs.prof file
+race	tells vcs to generate a report of all race conditions during simulation and write this report in the race.out file
+rad+1	enable level 1 radiant optimizations (See Release Notes)
+rad+2	enable level 2 radiant optimizations (See Release Notes)
+libext+lex	use extension 'lex' when searching library directories
+librescan	search from beginning of library list for all undefined mods
+incdir+idir	for `include files, search directory 'idir'
+nospecify	suppress path delays and timing checks
+notimingchecks	suppress timing checks
+optconfigfile+foo	use 'foo' as the optimization config file (See Release Notes)
+vcsd	enable the VCS Direct sim kernel interface
-cmhelp	enable CoverMeter help. CoverMeter should be installed and environment variable CM_HOME should be set.
-cm	enable VCS to first run cmSource to instrument the Verilog source files on the command line, and then to compile the instrumented source files
-cm_all	enable VCS to link CoverMeter into the VCS executable in a

way that enables line, condition, and FSM coverage and establishes the direct link. Enabling all types of coverage and the direct link is the default condition when you include the -cm option so you can omit this option

-cm_lineonly enable VCS to link CoverMeter into the VCS executable in a way that only enables line coverage when it also establishes the direct link. Use this option for faster simulation and when you only need line coverage

Summary of vcs runtime options:

-V	verbose mode
-grw file	send \$gr_waves output to 'file'
-i file	upon entering interactive mode, read 'file' as list of commands
-k file off	rename (or disable) the 'vcs.key' file
-l logfile	write output to 'logfile'
-q	quiet mode
-s	stop at time 0 and enter interactive mode
-vcd file	send \$dumpvars output to 'file'
-vhdlrun "opts"	pass 'opts' to Scirocco (scsim) for MixedHDL simulation
+cliecho	enable echoing of cli commands when sourcing a command file
+vcs+dumpon+t	turn on \$dumpvars at time 't'
+vcs+dumppoff+t	turn off \$dumpvars at time 't'
+vcs+finish+t	terminate simulation at time 't'
+vcs+stop+t	stop simulation at time 't'

Summary of vcs environment variables:

VCS_HOME	indicate alternate vcs distribution directory
VCS_CC	indicate the C compiler to be used
VCS_LOG	indicate a runtime log file
VCS_CMODLIB	Specify Cmodule library

如果需要更详细的 vcs 的参数说明，可以用参看 vcs 的 manual 手册用 man vcs 命令可以看到。

下面列举一下常用的 vcs 的参数：

-h 列举 vcs 简明帮助

-Moption=value ,M 后面的选项有很多个，下表说明

Option	Default	Comment
directory	csrc	产生的 C 语言文件和目标文件的目录位置
filename	Null	基本的 C 源文件和目标文件的基

		名
makefile	Makefile	产生的 makefile 的文件名
srclist	Filelist	包含有 makefile 的文件名列表
makeprogram	make	用 make 来产生目标文件
update	null	如果不为空,就重新产生 makefile
swift	null	如果不为空,就用 swift 来做接口
loadlist	null	如果是 yes,直接用 ld 来链接程序
ldcmd	null	格式化字符串来调用直接调用 ld

+acc ?

+cli ?

-o *file* 指明 vcs 的输出的文件名,或者是一系列文件名的基名。如果没有指定,默认得输出文件是 simv,输出的目标都是类似于 *vpidpattern.o*, pid 是当前的进程号, pattern 是一系列的随机数。

-c 翻译/编译源文件,但是不链接。

-C 翻译成 C 代码,但是不进行编译。

-V 显示详细的编译过程。在编译的过程中显示使用了哪些程序,有 C 编译器,汇编器,链接器等。

-Vt 同上,但是还增加了每一个程序占用 CPU 的时间。

-line 可以在源程序级进行单步执行。

-R 在完成了翻译、编译和链接之后,调用最近的那个可执行的目标文件,比如 simv 这个文件。

-I 这个选项使 vcs 自动加入+cli 和默认得 xvcs.tab 文件到编译命令行中,这个文件支持 VCS+的系统函数,比如\$vcpluson,\$vcplusoff,\$vcplusraceon 等。

-CC

-lname

-D *option* 用 *option* 来设置 vcs1 内部调试标识号

-P *file* 用 *file* 做为 PLI (programming Language Interface) 表文件

-L *file* 用 *file* 来做为并行布局表文件,这需要用户映射不同的程序线程,需要 VCS-MT 的支持,

-q 用安静的方式编译,在编译过程中不显示注释等信息。

-U 用非加速的方式运行 Vcs。Vcs 默认是用加速方式的。这种方式用来和 Verilog-XL 的非加速方式对应。

-a 用加速的方式运行 Vcs。默认的是打开这个选项的。

-c 这个选项指示 Vcs 编译源文件、进行语义分析、然后产生输出的文件类似于 *vpattern.o*

-f *file* *file* 这个文件中包含一些另外附加的参数,一般用在比较大的模型中,有时候操作系统可能会限制命令行的长度,这样可以把过长的参数写在文件中然后用 -f 参数指明这个文件。

-F *file* 和-f 参数是一样的,但是区别在于, *file* 中的内容是参数文件所在的目录。

-i *file* 提供给 *file* 中的内容做为 Vcs 的输入,当到达文件结尾,就把标准输入即键盘做为输入了。这个选项生效必需要有-R 选项。

-l *file* 所有的编译过程都记录到 *file* 文件中,同时也输出到标准输出,也就是显示器

上。

-s 停止选项，在模拟的开始就显暂停下来，然后进入交互模式。这个选项生效必须要有-R 选项。

-u

-v *file* 搜索库文件 *file* 中所有的模块来支持那些没有明文说明的模块。

-y *directory* 指明一个目录来进行搜索。

+define+macro+ 定义一个宏名，可以在源文件中用`ifdef 来检测，提供了预编译的功能。

+incdir+directory+ 指定`include 语句搜索的目录

+libext+extention+ 定义当搜索的时候，库文件的扩展名。比如+libexr+.v+.V+，这样就会搜索.v 和.V 的文件做为库文件。

+librescan 强制为每一个没有定义过的符号进行指定库的搜索。

附录 B. VirSim 的简明帮助:

VirSim 是 VCS 的一个图形化的集成调试环境，可以提供波形、结构、代码级的调试方法。有两种模式运行，交互方式 (interactive) 和后执行方式 (post-processing)。交互方式允许你在模拟的过程中进行控制。后执行方式提供在模拟结束后再检验完整的模拟过程的记录。VirSim 可以由 VCS 的命令来调用，下面的命令就可以在编译完 verilog 文件之后调用 VirSim 来显示，并且用交互方式，

```
%vcs -RI myfile.v
```

如果还想用单步执行，就加上-line 的参数，

```
%vcs -RI -line myfile.v
```

有时候可能不必要再编译 verilog 源文件，这在有些情况下比较有用，可以节省时间，下面的命令可以不用再编译源文件：

```
%vcs -RIG myfile.v #用交互方式
```

```
%vcs -RPP myfile.v #用后执行方式
```

当你在交互方式运行的时候，你是在一边模拟一边调试。然而在后执行方式，就要先用批处理方式，这时候产生需要的信号到 VCD+的文件中，然后再分析这个 VCD+文件。

举个例子，下面的命令就是用批处理方式编译并且运行 myfile.v，

```
%vcs -R -I myfile.v
```

-R 的选项就是用批处理方式，-I 的选项指明了 vcs 自动包含+cli(command line interface)、-P(PLI table)和-lm(数学库 math library)。PLI 的表定义了关于 VCD+的系统任务，默认的 PLI 表在\$VCS_HOME/virsims_support/xvcs.tab 这个文件中。

如果模拟已经完成了，就可以用-RPP 参数来调用调试器，比如：%vcs -RPP myfile.v 再在菜单中选择加载已经产生了的 VCD+文件。

如何产生 VCD+文件

VCD+类型的文件是通过在源代码中加入相应的系统函数来产生的，比如\$vcpluson()，或者\$vcplusraceon()，下面举例说明，源代码如下：

```
module example;
  reg a,b;
  wire y;
```

```

        some_sub_module n1(y,a,b);
initial begin
    $vcdpluson(example.n1); //产生 VCD+文件
    $vcdplustraceon;        //允许源代码级的调试
end
endmodule

```

VCS 所支持的关于 VCD+的新的系统任务有以下几种：

\$vcdpluson

\$vcdplusoff

\$vcdplustraceon

\$vcdplustraceoff

一般的默认的产生 VCD+文件名为 vcdplus.vpd，可以用参数指定文件名，比如

```
%vcs -R -I example.v +vcdfile+example.vcd
```

VirSim 的参数选项有：

```

-v file      在 file 中搜索没有引用到的模块
-y directory 在 directory 中的所有文件中搜索没有引用到的模块
-f file      从 file 中读取命令行的参数
-F file      和-f 的选项类似，但是 file 中指定了参数文件的文件名以及目录
-sim+file    指定产生的可执行文件的文件名
“+simargs+arglist” 指明了一些参数
+vpdfile+file 指定产生的 VCD+文件名
+cfgfile+file 指定读取已经保存过的配置文件
-V           用详细方式编译
-q           用安静方式编译

```

附录 C. simv 的简明帮助：

simv 是 vcs 所产生的可执行的模拟结果文件，运行这个可执行文件，可以进行查看信号变化以及调试等工作。simv 有一些参数和选项，下面介绍一下。

```

-grw file    设置$gr_waves 输出的文件名，可以用“|<program>”来过滤输出文件，
              默认的文件名是 grw.dump
-l file      所有的模拟结果都记录到 file 这个文件中，同时向标准输出显示。
-a file      用增加的方式把结果加到到 file 这个文件中。
-i file      从 file 文件中输入命令，当到文件末尾转为标准输入。
-q           输出 VCS 的头部信息和一些总结信息。
-s           在模拟的一开始就暂停，进入交互模式。
-V           显示版本和扩展的总结信息。
-vcd file    指明$dumpvars 这个系统任务的输出文件名，可以用|<program>”来过滤
              输出文件。如果在源文件内部用$dumpfile()指明了输出文件，这个参数就
              被忽略了。

```

+vcs+dump+off+<t>+<ht>

设置从开始到多少时刻之后停止输出 vcd 文件。<ht>是 64bit 的时间量，可选。

+vcs+dump+on+<t>+<ht>

设置从开始到多少时刻之后开始输出 vcd 文件。<ht>是 64bit 的时间量，可选。

+vpdfile+file 默认的 VCD+文件名是 vcdplus.vpd，可以用这个选项指定 VCD+文件名。

+vpdfilesize+MB 设置 VCD+文件的大小限制，如果超过这个限制，就停止输出 VCD+。

+vcs+finish+<t>+<ht>

设置多长时间之后结束模拟。<ht>是 64bit 的时间量，可选。

+vcs+stop+<t>+<ht>

设置多长时间之后结束模拟。<ht>是 64bit 的时间量，可选。