Synopsys Verilog Compiler Simulator (VCS) Tutorial

Synopsys Verilog Compiler Simulator is a tool from Synopsys specifically designed to simulate and debug designs. This tutorial basically describes how to use VCS, simulate a verilog description of a design and learn to debug the design. VCS also uses VirSim, which is a graphical user interface to VCS used for debugging and viewing the waveforms.

There are three main steps in debugging the design, which are as follows

1. Compiling the Verilog/VHDL source code.
2. Running the Simulation.
3. Viewing and debugging the generated waveforms.

You can interactively do the above steps using the VCS tool. VCS first compiles the verilog source code into object files, which are nothing but C source files. VCS can compile the source code into the object files without generating assembly language files. VCS then invokes a C compiler to create an executable file. We use this executable file to simulate the design. You can use the command line to execute the binary file which creates the waveform file, or you can use VirSim.

Below is a brief overview of the VCS tool, shows you how to compile and simulate a finite state machine which is Fibonacci series.

Tutorial Example

In this tutorial, we would be using a simple counter example . Find the verilog code and testbench at the end of the tutorial.

Source code file name	:	counter.v
Test bench file name	:	counter_tb.v

Setup

You need to copy the below scripts first before you invoke dc_shell. First one is "***create_dir_setup.tcl***", which will create the directory structure required to run the tutorial. You will see that it creates a directory called "project". This directory will contain all generated content including, VCS simulation, synthesized gate-level Verilog, and final layout. In this course we will always try to keep generated content from the tools separate from our source RTL. This keeps our project directories well organized, and helps prevent us from unintentionally modifying the source RTL. There are subdirectories in the project directory for each major step in the ASIC Flow tutorial. These subdirectories contain scripts and configuration files for running the tools required for that step in the tool flow. For this tutorial we will work exclusively in the ***vcs*** directory.

Second is “***synopsys_setup.tcl***” which sets all the environment variables necessary to run the Design Compiler.
Please copy these two files and source them at unix prompt.

```
[hkommuru@hafez ]$ source /packages/synopsys/setup/create_dir_setup.tcl
[hkommuru@hafez ]$ source /packages/synopsys/setup/synopsys_setup.tcl
```

In the “src” directory, look at the directory “counter” you will “main_counter.f “ file, it has only one line as shown below:

```
counter_tb.v  counter.v
```

Copy all the three files from the “counter” directory to your “vcs” working directory .

Compiling and Simulating

1. In the “vcs” directory, compile the verilog source code by typing the following at the machine prompt.

```
[hkommuru@hafez vcs]$ vcs –f main_counter.f +v2k
```

Please note that the –f option means the file specified (main_counter.f) contains a list of command line options for vcs. In this case, the command line options are just a list of the verilog file names. Also note that the testbench is listed first. The below command also will have same effect .

```
 [hkommuru@hafez vcs]$ vcs –f  counter_tb.v counter.v  +v2k
```

The +v2k option is used if you are using Verilog IEE 1364-2000 syntax; otherwise there is no need for the option. Please look at Figure 1 for output of compile command.

Figure 1:

```
                Chronologic VCS (TM)
        Version B-2008.12 -- Wed Jan 28 20:08:26 2009
          Copyright (c) 1991-2008 by Synopsys Inc.
                 ALL RIGHTS RESERVED

This program is proprietary and confidential information of Synopsys Inc.
and may be used and disclosed only as authorized in a license agreement
controlling such use and disclosure.


Warning-[ACC_CLI_ON] ACC/CLI capabilities enabled
  ACC/CLI capabilities have been enabled for the entire design. For faster
  performance enable module specific capability in pli.tab file

Parsing design file 'counter_tb.v'
Parsing design file 'counter.v'
Top Level Modules:
       timeunit
```

```
        counter_testbench
TimeScale is 1 ns / 10 ps
Starting vcs inline pass...
2 modules and 0 UDP read.
recompiling module timeunit
recompiling module counter_testbench
Both modules done.
gcc  -pipe -m32 -O -I/packages/synopsys/vcs_mx/B-2008.12/include   -c -o rmapats.o rmapats.c
if [ -x ../simv ]; then chmod -x ../simv; fi
g++  -o ../simv -melf_i386 -m32  5NrI_d.o 5NrIB_d.o IV5q_1_d.o blOS_1_d.o rmapats_mop.o rmapats.o
SIM_l.o          /packages/synopsys/vcs_mx/B-2008.12/linux/lib/libvirsim.a    /packages/synopsys/vcs_mx/B-
2008.12/linux/lib/librterrorinf.so              /packages/synopsys/vcs_mx/B-2008.12/linux/lib/libsnpsmalloc.so
/packages/synopsys/vcs_mx/B-2008.12/linux/lib/libvcsnew.so                    /packages/synopsys/vcs_mx/B-
2008.12/linux/lib/ctype-stubs_32.a -ldl -lz -lm  -lc -ldl
../simv up to date

VirSim B-2008.12-B Virtual Simulator Environment
Copyright (C) 1993-2005 by Synopsys, Inc.
Licensed Software. All Rights Reserved.
```

By default the output of compilation would be a executable binary file is named simv. You can specify a different name with the -o compile-time option. VCS compiles the source code on a module by module basis. You can incrementally compile your design with VCS, since VCS compiles only the modules which have changed since the last compilation.

2. Now, execute the simv command line with no arguments. You should see the output from both vcs and simulation and should produce a waveform file called counter.dump in your working directory.

```
[hkommuru@hafez vcs]$simv
```

Please see Figure 2 for output of simv command

Figure 2:

```
Chronologic VCS simulator copyright 1991-2008
Contains Synopsys proprietary information.
Compiler version B-2008.12; Runtime version B-2008.12;  Jan 28 19:59 2009

time=    0 ns, clk=0, reset=0, out=xxxx
time=   10 ns, clk=1, reset=0, out=xxxx
time=   11 ns, clk=1, reset=1, out=xxxx
time=   20 ns, clk=0, reset=1, out=xxxx
time=   30 ns, clk=1, reset=1, out=xxxx
time=   31 ns, clk=1, reset=0, out=0000
time=   40 ns, clk=0, reset=0, out=0000
time=   50 ns, clk=1, reset=0, out=0000
time=   51 ns, clk=1, reset=0, out=0001
time=   60 ns, clk=0, reset=0, out=0001
time=   70 ns, clk=1, reset=0, out=0001
time=   71 ns, clk=1, reset=0, out=0010
```

```
time=   80 ns, clk=0, reset=0, out=0010
time=   90 ns, clk=1, reset=0, out=0010
time=   91 ns, clk=1, reset=0, out=0011
time=  100 ns, clk=0, reset=0, out=0011
time=  110 ns, clk=1, reset=0, out=0011
time=  111 ns, clk=1, reset=0, out=0100
time=  120 ns, clk=0, reset=0, out=0100
time=  130 ns, clk=1, reset=0, out=0100
time=  131 ns, clk=1, reset=0, out=0101
time=  140 ns, clk=0, reset=0, out=0101
time=  150 ns, clk=1, reset=0, out=0101
time=  151 ns, clk=1, reset=0, out=0110
time=  160 ns, clk=0, reset=0, out=0110
time=  170 ns, clk=1, reset=0, out=0110
All tests completed sucessfully


$finish called from file "counter_tb.v", line 75.
$finish at simulation time  171.0 ns
          V C S   S i m u l a t i o n   R e p o r t
Time: 171000 ps
CPU Time:      0.020 seconds;       Data structure size:   0.0Mb
Wed Jan 28 19:59:54 2009
```

If you look at the last page of the tutorial, you can see the testbench code, to understand the above result better.

VIRSIM TUTORIAL

3. We can now re-invoke vcs to view the waveform. At the unix prompt, type:

```
[hkommuru@hafez vcs]$ vcs –RPP counter.v
```

The –RPP option tells vcs that we are opening in the post processing mode. This should open a new window as below.

4. In this window, click on "open" under the "File" menu option. Change the file type that you want to open to VCD (not VCD+). (VCD has .dump file extension and VCD+ has .vcd file extension). They are both waveform files but VCD files are text files, and VCD+ are condensed binary files. Take a look at the "Open File Dialog" below.

5. Select and open the file counter.dump and then click OK (also click O.K) on the information pop-up screen). Click on the Testbench button and you should see all signals instantiated in the signal window as shown below.

6. Click on New Waveform Window to open waveform window as below:

7. In the Hierarchy window, highlight all signals in the signal list with the left mouse button. Then with the middle mouse button, drag the selected signals over to the black

space in the waveform window. At this point, you should see the waveforms starting at the time 0 of the simulation as shown below.

8. In the waveform window, the menu option Display → Time Scale can be used to change the display unit and the display precision. You can also use Zoom on the menu to change the appearance as shown below:

9. Since we used system command $dumpvars in the verilog simulation, you should be able to view all signals at any hierarchical level of the design. Hence if you go back to the hierarchy window and click on the green arrow next to the Testbench button, you can traverse down the hierarchy and select more signals to view. Before exiting the waveform viewer, you can save your settings in a configuration file under the File → Save Configurations option.

Compiling in Interactive Mode

You can compile and simulate the same design in interactive mode. You can exit VirSim, and recompile the source code with the following command line:

```
vcs –RI –Mupdate –f main_counter.f
```

The –Mupdate is a compile-time option that tells vcs to compile incrementally. The – RI means we are going to simulate in the interactive mode.

As soon, as the code is compiled, Virsim is invoked and the simulation will start.

Verilog Code

File : Counter.v

```
module counter ( out, clk, reset ) ;

input        clk, reset;
output [3:0] out;

reg [3:0]    out;

wire [3:0]   next;

// This statement implements reset and increment
assign       next = reset ? 4'b0 : (out + 4'b1);

// This implements the flip-flops
always @ ( posedge clk ) begin
   out <= #1 next;
end
```

```
endmodule // counter
```

File : Counter_tb.v [Test Bench]

```
// This stuff just sets up the proper time scale and format for the
// simulation, for now do not modify.
`timescale 1ns/10ps
module timeunit;
   initial $timeformat(-9,1," ns",9);
endmodule


// Here is the testbench proper:
module counter_testbench ( ) ;

    // Test bench gets wires for all device under test (DUT) outputs:

   wire [3:0]        out;

   // Regs for all DUT inputs:
   reg               clk;
   reg               reset;


   counter dut (// (dut means device under test)
                // Outputs
                .out            (out[3:0]),
                // Inputs
                .reset          (reset),
                .clk            (clk));


  // Setup clk to automatically strobe with a period of 20.
   always #10 clk = ~clk;


   initial
     begin

        // First setup up to monitor all inputs and outputs
        $monitor ("time=%5d ns, clk=%b, reset=%b, out=%b", $time, clk,
reset, out[3:0]);

        // First initialize all registers
        clk = 1'b0;            // what happens to clk if we don't
                               // set this?;
        reset = 1'b0;

        @(posedge clk);#1;     // this  says wait for rising edge
                               // of clk and one more tic (to prevent
                               // shoot through)

        reset = 1'b1;

        @(posedge clk);#1;

        reset = 1'b0;
        // Lets watch what happens after 7 cycles
        @(posedge clk);#1;
        @(posedge clk);#1;
        @(posedge clk);#1;
        @(posedge clk);#1;
        @(posedge clk);#1;
        @(posedge clk);#1;
        @(posedge clk);#1;


        // At this point we should have a 4'b0110 coming out out
because
        // the counter should have counted for 7 cycles from 0
        if (out != 4'b0110) begin
           $display("ERROR 1: Out is not equal to 4'b0110");
           $finish;
        end


        // We got this far so all tests passed.
        $display("All tests completed sucessfully\n\n");

        $finish;
     end


   // This is to create a dump file for offline viewing.
   initial
     begin
        $dumpfile ("counter.dump");
        $dumpvars (0, counter_testbench);
     end // initial begin


endmodule // counter_testbench
```