EDA Technology

Chapter 10

Verilog HDL Simulation

Verilog HDL Simulation Flow

⌘

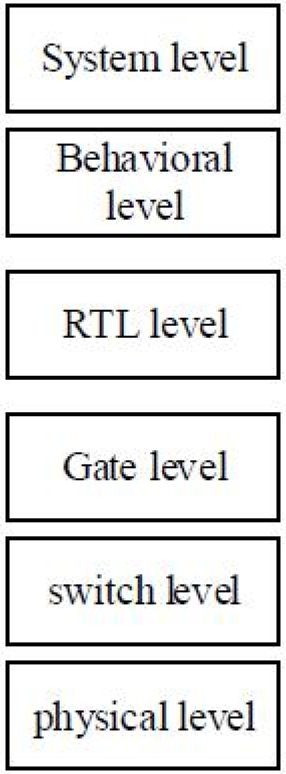


Figure: HDL System Design Description Level

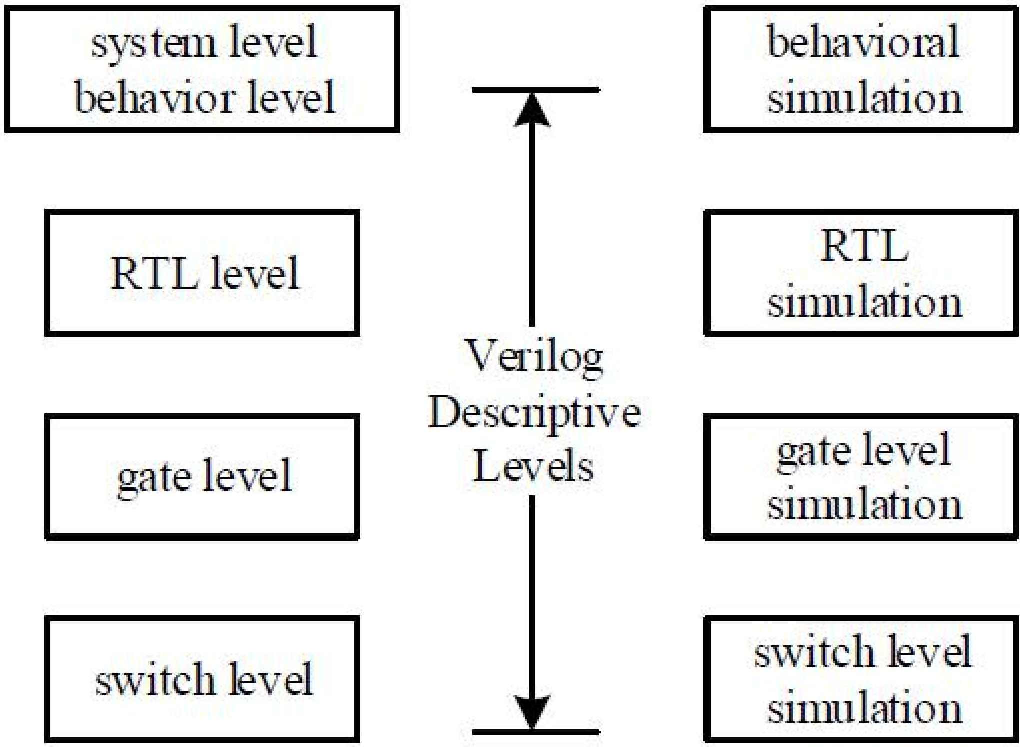


Figure: Verilog description hierarchy and corresponding simulation hierarchy

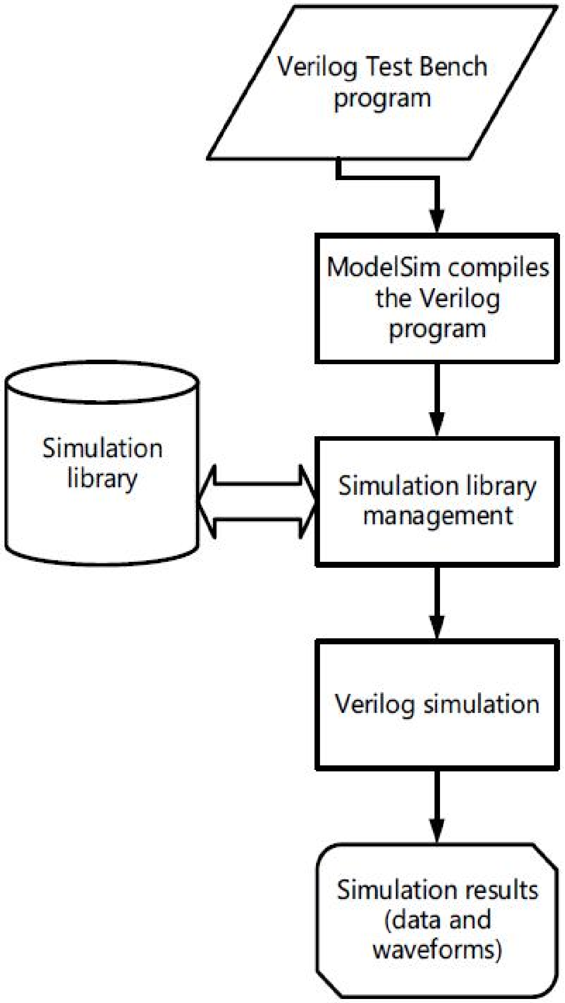

Verilog Test Bench program
ModelSim compiles the Verilog program
Simulation library
Simulation library management
Verilog simulation
Simulation results (data and waveforms)

Figure: Verilog simulation process

Example of Verilog Test Bench

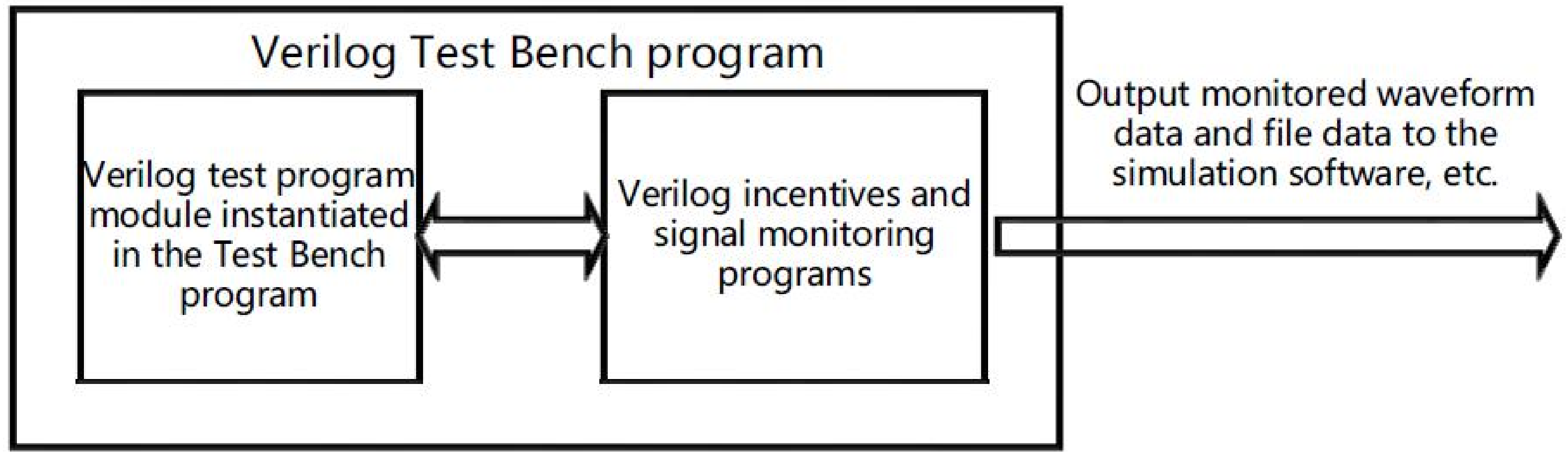

Verilog Test Bench program
Verilog test program module instantiated in the Test Bench program
Verilog incentives and signal monitoring programs
Output monitored waveform data and file data to the simulation software, etc.

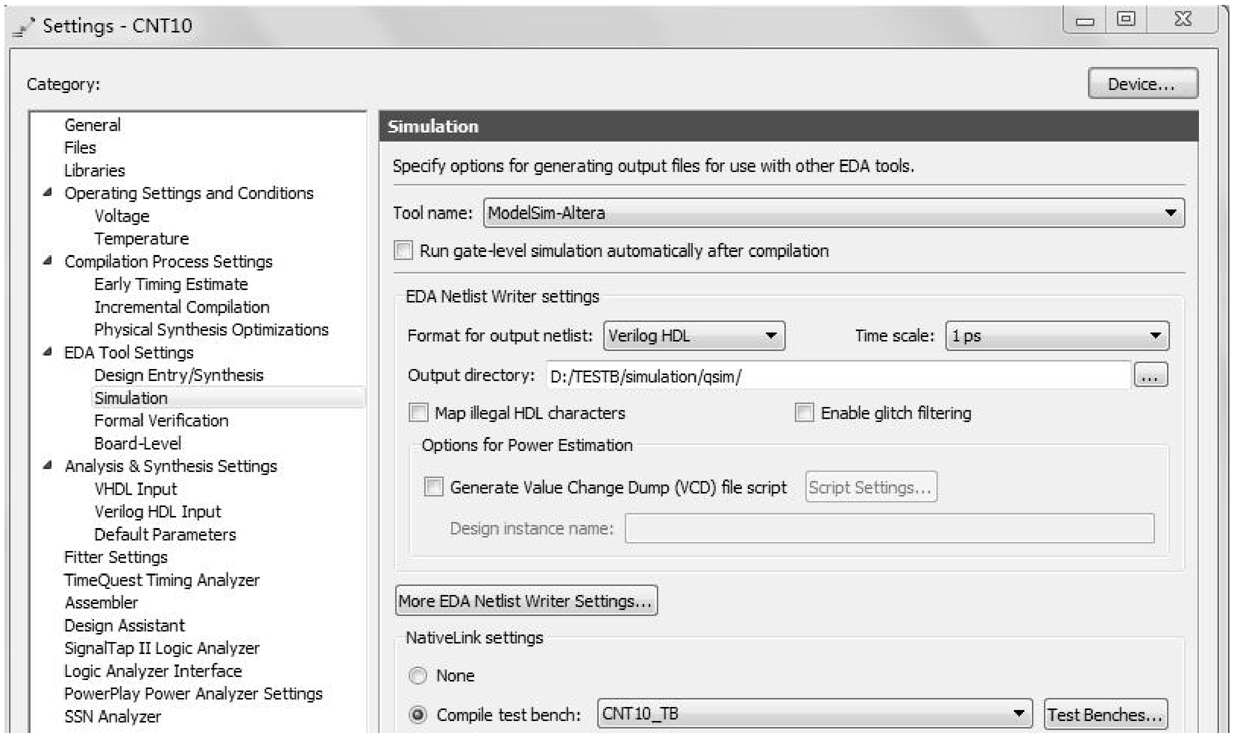
Figure: Verilog test bench structure

Example: //Test Bench file name:CNT10_TB.v

```
'timescale 10ns/1ns    // In Test Bench, this simulation time scale statement must exist
module CNT10_TB ();    // Note that this module does not have to give the port descriptionreg
clk, en, rst, load; reg [3:0] data ; // The data type that defines the stimulus signal
                                        //is reg
wire [3:0] dout ;  wire cout ; // The data type that defines the stimulus signal is reg
always #3 clk=~clk;  // The statement that generates the clock, every 3 time units, ie
                        // 30ns, clk toggle once
initial
$monitor ("DOUT=%h",dout); //Print the output data of the module to be tested DOUT in
                              //hexadecimal
initial begin                          // One-time procedure statement
   #0  clk=1'b0;                        //0 time unit, set the clk level to 0
   #0  rst=1'b1;   #20 rst=1'b0;  #2  rst=1'b1;  // Note that it is a sequential
                                                  //assignment statement
end
initial begin
   #0   en  = 1'b0;  #5   en  = 1'b1;
end
initial begin
  #0  load=1'b1; #49 load=1'b0; #3  load=1'b1;
end
initial begin
  #0  data=4'h7; #30 data=4'h2; #30 data=4'h5; #30 data=4'h4;
end
CNT10 U1 (.CLK(clk), .RST(rst), .DATA(data), .LOAD(load),
          .EN(en), .COUT(cout), .DOUT(dout)); // Instantiation statement endmodule
```

Testing Flow of Verilog Test Bench

- 1. Install ModelSim
- 2. Set simulation parameters for Test Bench



Settings - CNT10

Category:

- General
- Files
- Libraries
- Operating Settings and Conditions
  - Voltage
  - Temperature
- Compilation Process Settings
  - Early Timing Estimate
  - Incremental Compilation
  - Physical Synthesis Optimizations
- EDA Tool Settings
  - Design Entry/Synthesis
  - Simulation
  - Formal Verification
  - Board-Level
- Analysis & Synthesis Settings
  - VHDL Input
  - Verilog HDL Input
  - Default Parameters
- Fitter Settings
- TimeQuest Timing Analyzer
- Assembler
- Design Assistant
- SignalTap II Logic Analyzer
- Logic Analyzer Interface
- PowerPlay Power Analyzer Settings
- SSN Analyzer

Simulation

Specify options for generating output files for use with other EDA tools.

Tool name: ModelSim-Altera

☐ Run gate-level simulation automatically after compilation

EDA Netlist Writer settings

Format for output netlist: Verilog HDL   Time scale: 1 ps

Output directory: D:/TESTB/simulation/qsim/

☐ Map illegal HDL characters   ☐ Enable glitch filtering

Options for Power Estimation

☐ Generate Value Change Dump (VCD) file script

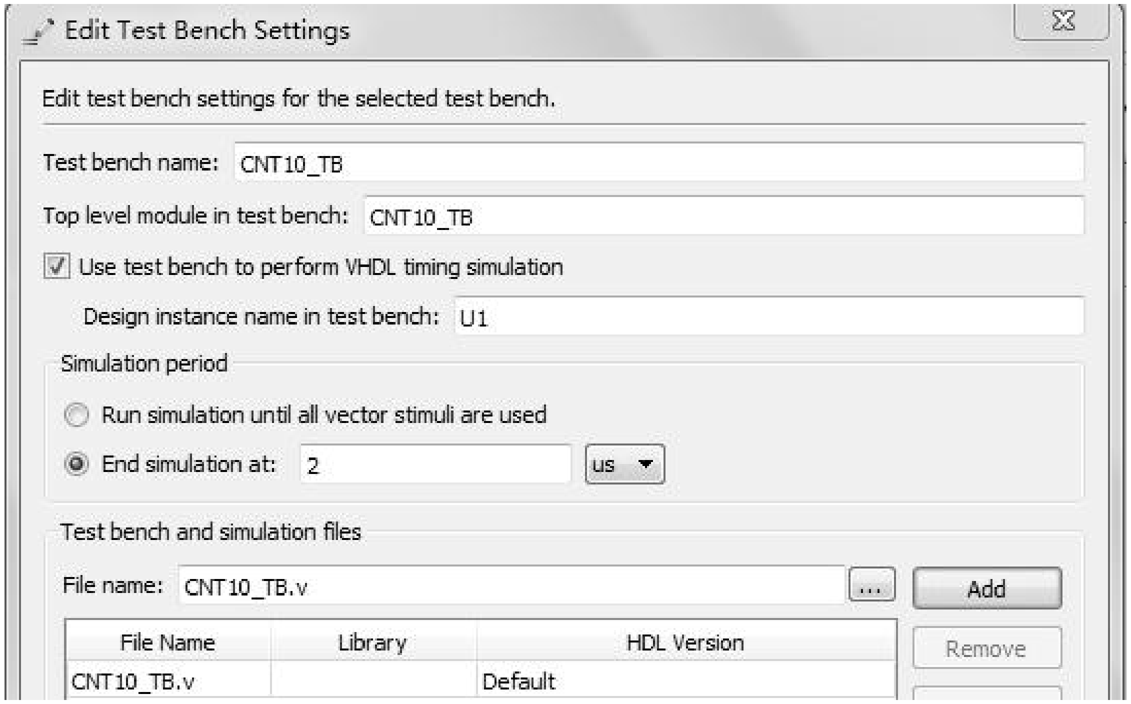
Design instance name:

NativeLink settings

○ None

● Compile test bench: CNT10_TB

Figure: Select simulation tool name and output netlist language form



Edit Test Bench Settings

Edit test bench settings for the selected test bench.

Test bench name: CNT10_TB

Top level module in test bench: CNT10_TB

[✓] Use test bench to perform VHDL timing simulation

Design instance name in test bench: U1

Simulation period

○ Run simulation until all vector stimuli are used

● End simulation at: 2 us

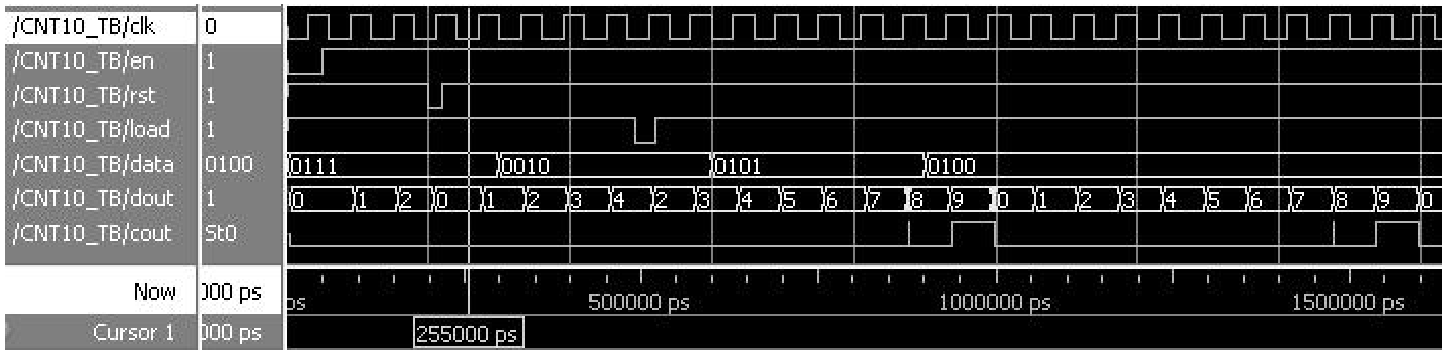
Test bench and simulation files

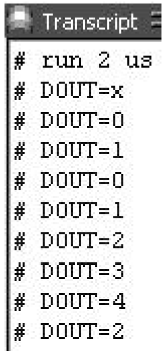
File name: CNT10_TB.v

File Name	Library	HDL Version
CNT10_TB.v		Default

Figure: Set parameters for test bench simulation

- 3. Start test bench simulation
- 4. Analyze test bench simulation results





```
# run 2 us
# DOUT=x
# DOUT=0
# DOUT=1
# DOUT=0
# DOUT=1
# DOUT=2
# DOUT=3
# DOUT=4
# DOUT=2
```

Figure: Test Bench output simulation waveforms and data (on the right)

Verilog System Tasks and System Functions

System Tasks and System Functions

⌘ **1. $display**

```
$display ("with format string", parameter 1, parameter 2, ... );
```

```
    module sdispl;
    integer i;                          // i is an integer
    reg [3:0] x;                               // x is 4 digits
    initial begin                       // The initial block, only executed once
    i=21;       x=4'he;
    $display("1\t%d\n2\t%h\\",i,x);   //show output
  end   endmodule
```

Simulate in ModelSim and enter in the Transcript command line window: run.

Start the simulation and the result returned in the command line window is:

```
# 1    21
# 2    e\
```

Table: Verilog Escapes

escapes	meaning	escapes	meaning	escapes	meaning
\n	Newline	%b	Binary format	%v	Display signal strength
\t	Tab	%o	Octal format	%m	Display hierarchy name
\\	charcter\	%d	Decimal format	%s	String format
\"	character"	%h	Hexadecimal format	%t	Display present time
\ddd	1 to 3 octal ASCII characters	%l	Display: Library Binding information	%u	Unformatted binary data
%%	charcer%	%c	Charcter format	%z	Unformatted four-valued data
%e	Scientific notation Display real value	%f	Decimal display Real value	%g	Take the shortest display in the %e, %f format

```
module sdisp2;                        // notice no input or output port
  reg [31:0] rval;                    //32 bit reg type
  pulldown (pd); // Pd pull-down resistor, plldown usage see the rest of this chapter
  initial begin    //initial block
  rval = 101;       // assignment integer101
  $display("rval = %h hex %d decimal",rval,rval);//display decimal and hexadecimal
  $display("rval = %o octal\nrval = %b bin",rval,rval); //display binary and octal
  $display("rval has %c ascii character value",rval);//display character format output
  $display("pd strength value is %v",pd);       // display pd signal strength
  $display("current scope is %m"); // Current hierarchical module name display
  $display("%s is ascii value for 101",101);  //display string
  $display("simulation time is %t", $time);         //display present simulation time
  end
  endmodule
```

- Simulate in ModelSim, the result returned in the command line window is:
- # rval = 00000065 hex   101 decimal
- # rval = 00000000145 octal
- # rval = 00000000000000000000000001100101 bin
- # rval has e ascii character value
- # pd strength value is StX
- # current scope is sdisp2
- #   e is ascii value for 101
- # simulation time is   0

- **2. $write**

```
$write ("with format string", parameter 1, parameter 2, ... );
```

- **3. $strobe and $monitor**

```
$strobe ("with format string", parameter 1, parameter 2, ... );
$monitor ("with format string", parameter 1, parameter 2, ... );
```

```
module sdisp3;                                        //no output and input signal
reg [1:0]a;                                           //a is 2 bit reg
reg b;
initial $monitor("\$monitor: a = %b", a) //$monitor monitor changes in a
initial begin                                   //initial block, only carry out once
b = 0;a = 0;                                    //b,a is 0, blocking assignment
$strobe ("\$strobe : a = %b", a);               //$strobe display a
a = 1;                                          //a is 1
$display ("\$display: a = %b", a);              //$display display present a
a = 2;                                          //a is 2
$monitor("\$monitor: b = %b", b); //$monitor replace the previous $monitor
a = 3;                                          //a is 3
#30 $finish;        //after delay 30 time unit, stop simulation
end
always #10 b = ~b;// b every 10 time units, the value is reversed, Clock signal
endmodule
```

The output of ModelSim simulation results is as follows:

```
# $display: a = 01
# $strobe : a = 11
# $monitor: b = 0
# $monitor: b = 1
# $monitor: b = 0
```

4. $finish and $stop

```
module sdisp4();
 reg [3:0]a,b;                      //a, b both are 4 bit reg
 initial $monitor($time," \$monitor:a=%0d,b=%d",a,b); //display change and time
initial begin                       //initial block, only excute once
  b = 0;                            //b is 0
  $strobe ($time," \$strobe : a = %0d", a); //display the result of a
  $monitoron;                       //start $monitor
  a = 1;                            //a is 1, Blocking assignments
  a <= 2;                           //a is 2, non-blocking assignments
  $display ($time," \$display: a = %d", a); //display the value of present a
  a = 3;                            //a is 3,blocking assignments
  #25 $monitoroff;                  //stop $monitor
  #10 $stop;                        //after 10 time units, time out the simulation
  end
  always #10 b = b+1;               // b For every 10 time units, add 1
 endmodule
```

The output of ModelSim simulation is as follows:

```
#                    0 $display: a =  1
#                    0 $strobe : a = 2
#                    0  $monitor:a=2,b= 0
#                   10  $monitor:a=2,b= 1
#                   20  $monitor:a=2,b= 2
```

5. $time

```
$time //Return a 64-bit integer time value
$stime //Return a 32-bit integer time value
$realtime //Return a real time value
$timeformat //Control how time is displayed
```

```
  $monitor("%d d=%b,e=%b", $stime, d, e); //$time shows
another form of current time
```

6. File operation

The complex Test Bench used for large-scale digital system simulation verification often needs to read or write files during simulation. Verilog also has system functions and system tasks for file operations. Their syntax is:

```
   File handle = $fopen("file name") //Open file
   $fstrobe(file handle, "formatted string", parameter list) //strobe to file
   $fdisplay(file handle, "formatted string", parameter list t) //display to file
   $fmonitor(file handle, "formatted string", parameter list t) //monitor to file,
can be multiple processes
   $fwrite(file handle, "formatted string", parameter list) //write to file
   $fclose(file handle); //Close file
   $feof(file handle); //Query whether the end of the file has been reached
```

The following are system tasks and functions for exporting VCD files for simulation output. The syntax is:

```
   $dumpfile("file name"); //Export to file where the file suffix is vcd
   $dumpvar; //Export all variables of the current design
   $dumpvar(1, top); //Export all variables in the top-level module
   $dumpvar(2, top); //Export all variables for the top 1 and 2 level module
   $dumpvar(n, top); //Export all variables from the top 1 to top n-1th level
module
   $dumpvar(0, top); //Export all variables of the top level and all hierarchy
modules
   $dumpon; //Export initialization
   $dumpoff; //Stop export
```

```
module fileio_demo;                              //file write and read
integer fp_r, fp_w, cnt;                 // Defining file handles, integers
reg [7:0] reg1, reg2, reg3;                      //3 8-bit reg values
initial begin
fp_r = $fopen("in.txt", "r");                    // Open in.txt as read-only
fp_w = $fopen("out.txt", "w");                   //open out.txt as write-only
while(!$feof(fp_r)) begin  // Loop reading and writing files until end of in.txt
  cnt = $fscanf(fp_r, "%d %d %d", reg1, reg2, reg3);   // Read a line
  $display("%d %d %d", reg1, reg2, reg3);        // Display the read value
  $fwrite(fp_w, "%d %d %d\n", reg3, reg2, reg1);// Write one line in reverse order
end
 $fclose(fp_r);                                  //close file in.txt
 $fclose(fp_w);                                  //close file out.txt
end
 endmodule
```

Precompiled Statements

1. *'define* macro definition

The usage of the macro definition statement 'define mentioned earlier is very similar to the #define in C. Usage is as follows:

```
    'define dnand(dly) nand #dly
    'dnand(2) g121 (q21, n10, n11);
    'dnand(5) g122 (q22, n10, n11);
```

2. Translate_on and translate_off

The specific implementation is to insert a special comment in Verilog source code, the format is:

```
    //synthesis translate_off
    //synthesis translate_on
```

Delay Model

# Delay and Gate Delay

In the simulation, the most common is the assignment delay, such as:

```
#10 rout = cin;
```

The unit of time can be defined with the statement 'timescale. The gate delay representation has three formats:

- # Number of delay time units.
- # (rise delay, drop delay).
- # (rise delay, fall delay, delay in transition to z).

The basic gate can also be delayed during instantiation. For example:

```
Nand #20 inand2(a,b,c);
```

```
module  dnot ();
   reg in; wire out;
   not #(3,4) (out,in);    // Instantiate not, also explain gate delay
initial begin
     $monitor ("%g in = %b out=%b", $time, in, out); // Monitor in, out
     in = 0;        // Initial assignment 0
     #10 in = 1;        //After 10 time units, assignment 1
     #10 in = 0;        //After 10 time units, assigned 0
     #10 $stop;         //After 10 units of time, pause simulation
end
endmodule
```

Simulate on ModelSim and enter: run –all in the command line window to observe the following output information:

```
# 0 in = 0 out=x
# 3 in = 0 out=1
# 10 in = 1 out=1
# 14 in = 1 out=0
# 20 in = 0 out=0
# 23 in = 0 out=1
```

Delay Description Block

```
module veridelay(output out,  input a,b,c,d);
   wire e,f;
   specify          //specify delay description block
       (a=>out)=3;  //from a to out delay three time units
       (b=>out)=3;  // from b to out delay three time units
       (c=>out)=5;  // from c to out delay five time units
       (d=>out)=51;    // from a to out delay 51 time units
   endspecify
   and U1(e,a,b); and U2(f,c,d);  and U3(out,e,f); // Instantiate 3 components
endmodule
```

Other Simulation Statements

fork-join Block Statements

```
module forkA(clk,a,b);
    input clk;
    output reg a, b;
    initial begin
      a=0;  b=0;  end
  always @(posedge clk)
    fork
      #30 a = 1;
      #10 b = 1;
    join
endmodule
```

```
    module
forkB(clk,a,b);
    input clk;
    output reg a, b;
    initial begin
  a=0;   b=0;   end
always @(posedge clk)
    begin
      #30 a = 1;
      #10 b = a;
end
endmodule
```

```
module forkC(clk,a,b);
    input clk;
    output reg a, b;
    initial begin
  a=0;   b=0;  end
always @(posedge clk)
    fork
      #30 a = 1;
      #10 b = a;
    join
endmodule
```

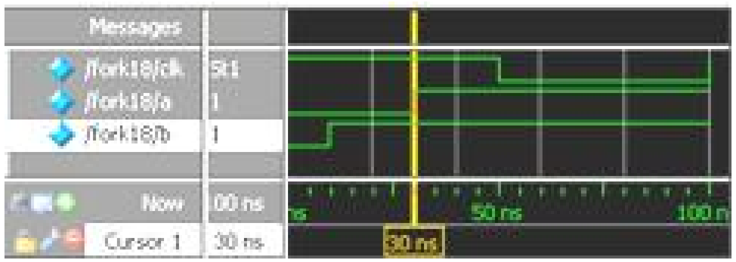


Figure: Simulation waveform of Example 10-9

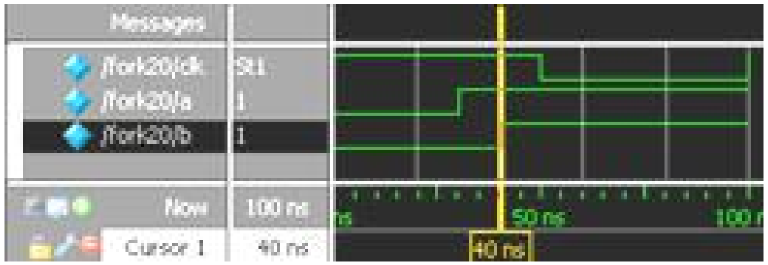


Figure: Simulation waveform of Example 10-10

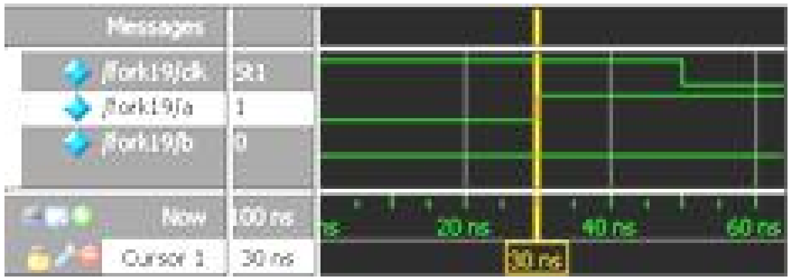


Figure: Simulation waveform of Example 10-11

wait Statement

The wait statement is also an unsynthesizable statement that can only be used for simulation. Its syntax is as follows:

```
wait (conditional expression) statement;
```

It can be used in the following way:

```
Forever wait(start) #10 go = ~go;
```

force, release Statement

```
 module testforce;                    // Force statement test example
     reg a, b, c, d;    wire e;
     and and1 (e, a, b, c);
      initial begin                   // Monitor changes in d and e   $monitor("%d
      d=%b,e=%b", $stime, d, e);
     assign d = a & b & c;            // Continuous assignment d
     a = 1;  b = 0;  c = 1;
     #10;                             // Delay 10 units of time
     force d = (a | b | c);           // Forced assignment d
     force e = (a | b | c);           // Forced assignment e
     #10 $stop;                       // Suspend simulation
     release d;                       // Release d
     release e;                       // Release e
     #10 $stop;                       // Suspend simulation
     end
endmodule
```

Simulate on ModelSim, first enter in the command line window:

```
run -all
```

Output results:

```
#                    0 d=0,e=0
#                   10 d=1,e=1
```

Then the simulator pauses, after entering run –all again, and the results output:

```
#                   20 d=0,e=0
```

deassign Statement

```
always @(clear or preset)
  if (clear)
    assign q = 0;
  else if (preset)
    assign q = 1;
  else
    deassign q;
always @(posedge clock) q = d;
```

Generation of Simulation Excitation Signals

```
  module adder4(input[3:0] a, input[3:0] b, output reg[3:0] c, output reg co);
                                                  //4-bit adder
  always @ *
{co,c} <= a + b;   // Co for carry, c for and
```

1. Method one

```
        'timescale 10ns/1ns                            //setting time
        module signal_gen(output reg [3:0] sig1,output reg [3:0] sig2);
            initial begin
              sig1 <= 4'd10;           // Input signal changes in sequence
              sig2 <= 4'd3;
              #10 sig2 <=4'd4;      #10 sig1 <=4'd11;    #10 sig2 <=4'd6;
              #10 sig1 <=4'd8;      #10 $stop;
            end
      endmodule
```

```
module test_adder4();                               // Top-level files for simulation
   wire [3:0] a,b,c;   wire co;
   adder4 U1(.a(a),.b(b),.c(c),.co(co));            // Instantiate the DUT
 signal_gen TU1(.sig1(a),.sig2(b)); // Instantiated incentive generation module
endmodule
```

2. Method two

Use the simulator's waveform setup command to apply the stimulus signal. In ModelSim, the force command can be used to interactively apply incentives. The format of the force command is as follows:

```
    Force <signname> <value> [<time>][, <value> <time> ...] [-repeat
  <cycle>]
```

For example:

```
 force a 0          // The current value of the forced signal is 0
 force b 0 0, 1 10// The forced signal b has value 0 at time 0 and value 1
at time 10
 force clk 0 0, 1 15 -repeat 20 // Clk is a periodic signal with a period
of 20
```

You can use the structure of module adder4 to simulate directly. After initializing the simulation process, enter the following command in the command line of ModelSim:

```
    force a 10 0, 5 200, 8 400
    force b 3 0, 4 100, 6 300
```

Then execute the run command or run 500 command continuously to obtain the simulation waveform.