ModelSim-Altera 6.5 Example Guide in Quartus 11

Editor: Ourfpga www.Ourfpga.com

Note: I wrote this document with reference to documents and codes on the Internet, and I would like to express my thanks to them!

By default, you have installed the Quartus 11.0 software

1. Modelsim_Altera_ASE software installation

The Modelsim_Altera_ase installation package is provided in the CD-ROM of the Ruizhi FPGA development board. The ASE version is Altera start edition, that is, the entry version, which is free to use; AE is Altera edition, which needs to be cracked and supports more functions.

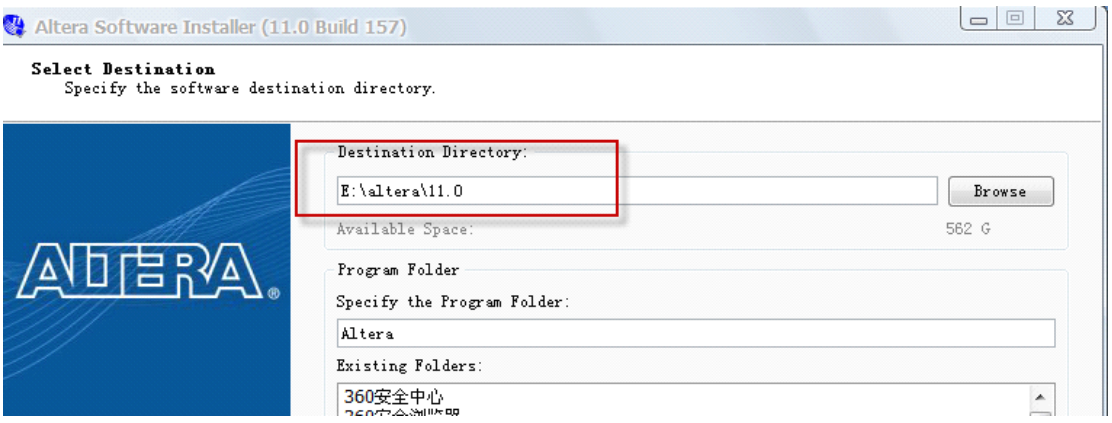
But for us, the ASE version is enough. Install it and use it. There's no need to break it down.

If you want to install the AE version, please refer to the tutorial written by Bingo, the website address is as follows:

http://www.cnblogs.com/crazybingo/archive/2011/02/21/1959893.html

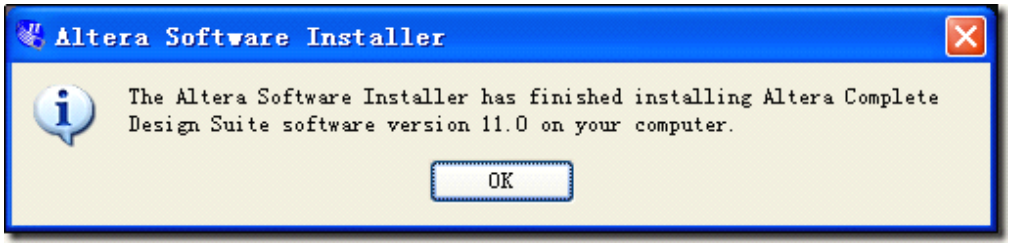
Install 11.0_Altera_Modelsim_ase_windows.exe, here. Specific steps are as follows:

(1) Open the setup in the installation directory, go all the way to next, until the path is selected, select the same path as the Quartus ii installation directory. As shown in the picture below, my computer is installed on the E drive, you need to change it according to your settings.



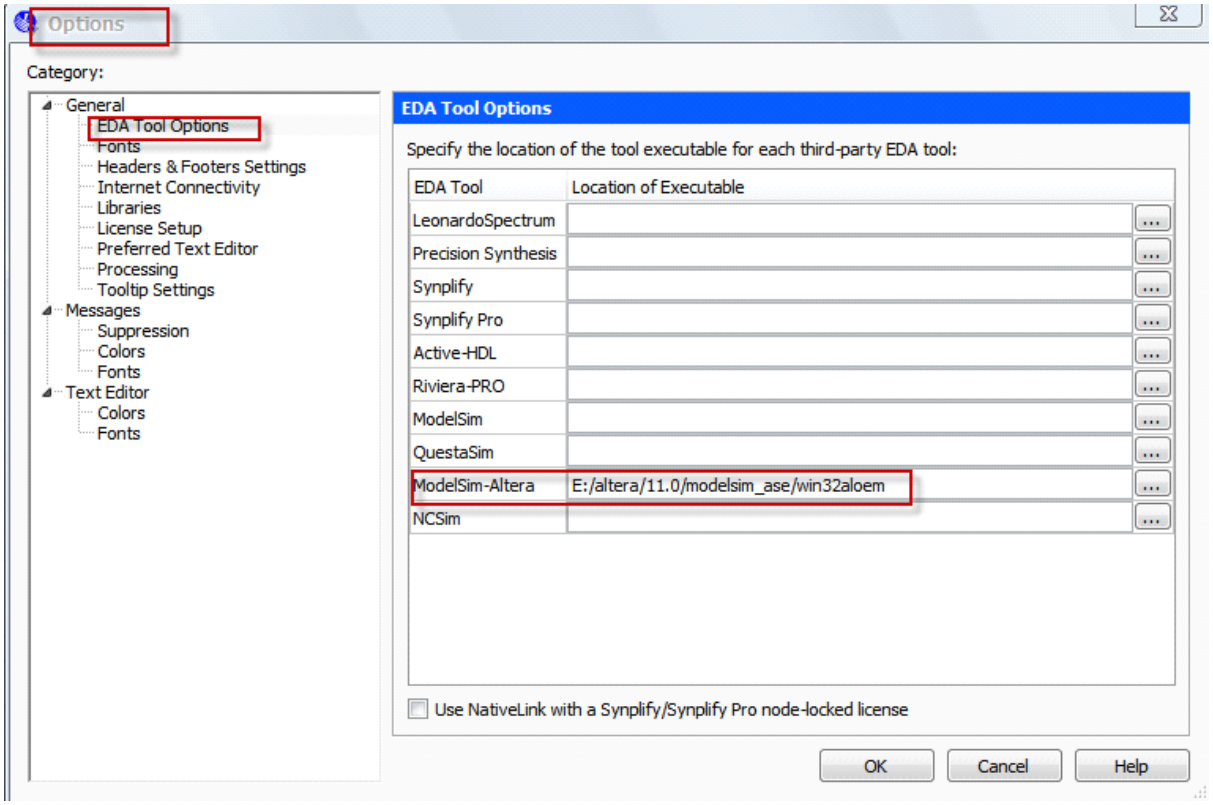
(2) Click on Next, and wait for the installation to complete

(3) After the installation is complete, the following interface appears:



(4) OK, Finish!

(5) Open Quartus II, open the menu Tool-Options, select ModelSim-Altera in EDA Tool Options, select the root directory of the ModelSim-Altera application, and configure the third-party software path of the ModelSim-Altera application. As shown in the figure below: In this tab, the following ModelSim-Altera item specifies the installation path as E:/Altera/11.0/modelsim_ae/win32aloem (where E:/Altera/11.0/modelsim_ae/ is ModelSim-Altera 6.5 in my computer installation path)

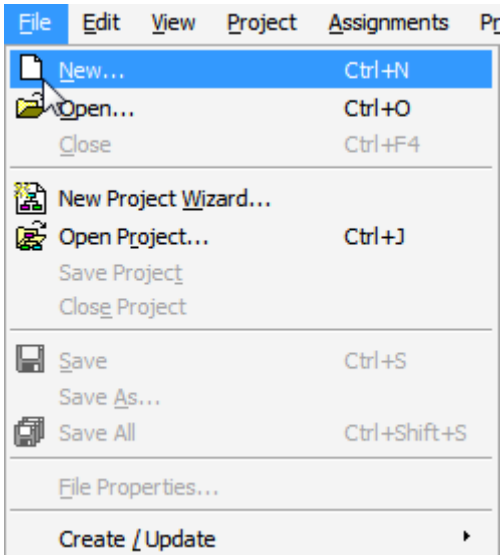


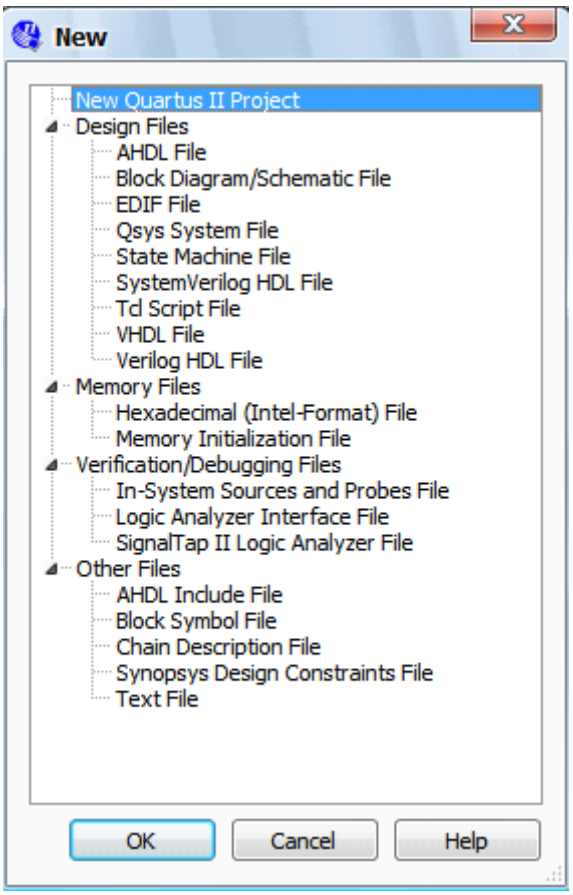
So far, the Altera-ModelSim ASE version is installed

2. How to Run ModelSim-Altera in Quartus II 11.0

I'll describe the whole process with a simple example:

First, get a project, open Quartus II, menu file---new, create a new project





Create a new Verilog HDL File, code:

```
module modelsim_test(clk,rst_n,div);
input clk;
input rst_n;
output div;
reg div;
always@(posedge clk or negedge rst_n)
if(!rst_n)div<=1'b0;
else div<=~div;
endmodule
```

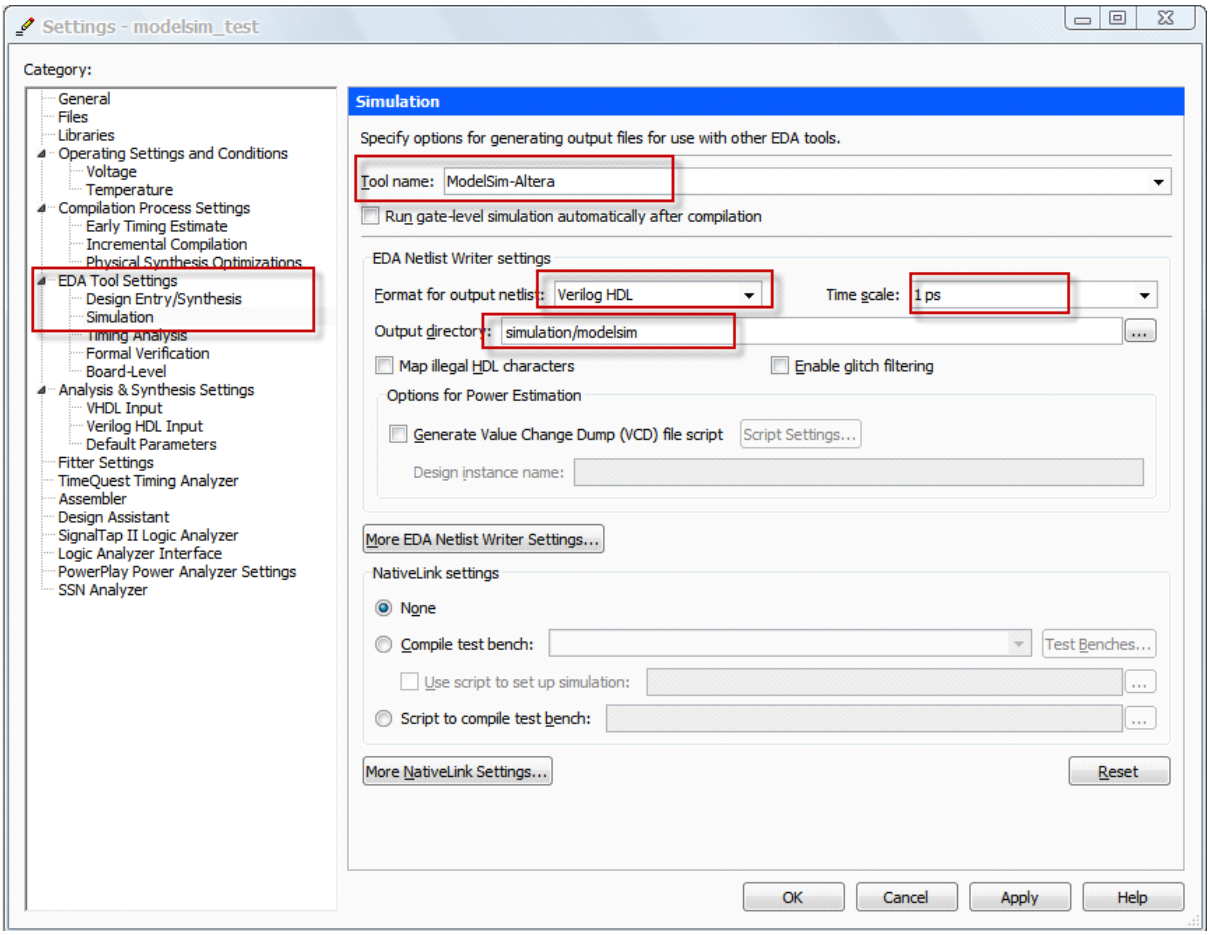
A very simple code is a divide-by-two circuit.

What we're going to do is run a ModelSim simulation of this circuit,

Next, let's set it up:

Select Assignments->Settings in the Quartus II 11.0 interface menu bar.

1. Select the Simulation item in the EDA Tool settings under this interface;
2. Select ModelSim-Altera in Tool name;
3. In Format for output netlist, select the type of development language Verilog or VHDL, etc.,
4. Time scale specifies the time unit level
5. Output directory specifies the output path of the test file template (the path is the relative path of the project file).



Generate simulation test files

Select Processing->Start->Start Test Bench Template Writer under the menu bar of the Quartus II 11.0 development interface, and it will prompt when the generation is successful. This generated simulation test file (find the file with the suffix ".vt" in the directory under modelsim under the modelsim_test project file) and edit it according to your needs. The following is the original file generated, it has not been changed

```
`timescale 1 ps/ 1 ps
module modelsim_test_vlg_tst();
// constants
// general purpose registers
reg eachvec;
// test vector input registers
reg clk;
reg rst_n;
// wires
wire div;

// assign statements (if any)
modelsim_test i1 (
// port map - connection between master ports and signals/registers
   .clk(clk),
   .div(div),
   .rst_n(rst_n)
);
initial
begin
// code that executes only once
// insert code here --> begin

// --> end
$display("Running testbench");
end
always
// optional sensitivity list
// @(event1 or event2 or .... eventn)
begin
// code executes for every event on sensitivity list
// insert code here --> begin

@eachvec;
// --> end
end
endmodule
```

Note: The suffix of the testbench file in QuartusII is .vt, and the generated template file only contains port mapping, port declaration, etc. The specific functions still need to be written by the designer. Let's modify the template and write the testbench. The code is as follows:

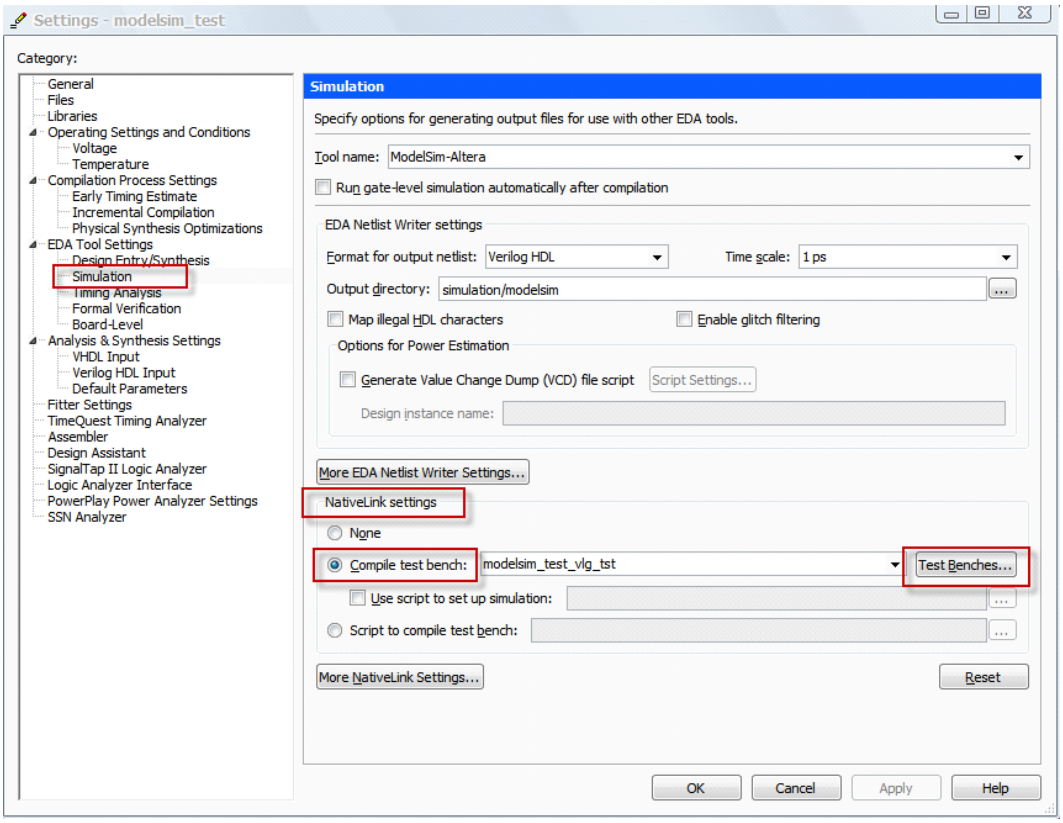
```
timescale 1 ps/ 1 ps
module modelsim_test_vlg_tst();
reg clk;
reg rst_n;
wire div;
modelsim_test i1(
.clk(clk),
.div(div),
.rst_n(rst_n)
);
Initial
begin
clk=0;
forever
#10 clk=~clk;
end
initial
begin
rst_n=0;
#1000 rst_n=1;
#1000;
$stop;
end
endmodule
```

The reason why the code is written this way involves learning how to write testbench, which is not within the scope of this document, but at the end of the document, a testbench writing tutorial found on the Internet is attached for your reference.

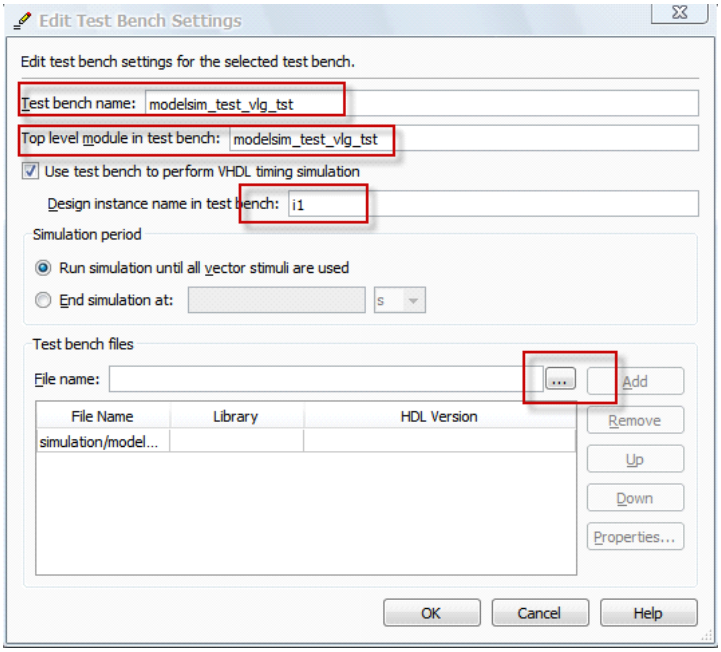
The following are very critical steps, please be optimistic, if you set it wrong, you will not succeed.

Select Assignments->Settings in the Quartus II 11.0 interface menu bar.

Select the Simulation item in EDA Tool settings under this interface; select Compile test bench in Native Link settings and click Test Benches behind



Click New in Test Benches. See below:



In the Testbench name column, fill in the entity name of the testbench file we just created i.e. modelsim_test_vlg_tst;

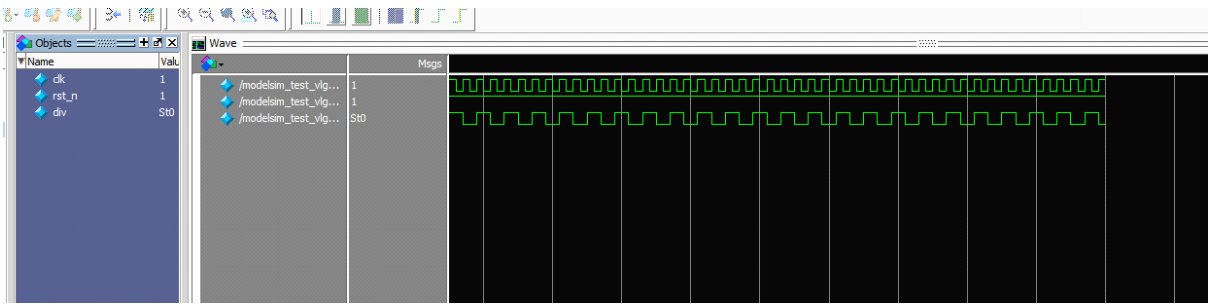
Also fill in modelsim_test_vlg_tst in Top level modele in test bench;

In Design Instance name in test bench

Here you can copy it directly from the testbench file to avoid writing mistakes by hand.

Then browse to add the testbench file in Test bench files, then remember to click add, step by step.

Everything is ready, select Run EDA Simulation Tool->EDA RTL Simulation in the menu bar Tools in the Quartus II 11.0 interface menu bar to perform behavioral-level simulation, then you can see the running interface of ModelSim-Altera 6.5e, observe the simulation waveform.



You can see that the div signal is divided by two of clk

How to write a testbench summary (very useful summary)

1. Incentive settings

The input type corresponding to the module under test is set to reg type, and the output is set to wire type. The two-way port inout needs to be processed during the test.

Method 1: Set the intermediate variable inout_reg for the two-way port as the output register of the inout. The inout port should be defined as a wire-type variable in the testbench, and then use the output to enable and control the transmission direction.
eg：

```
inout [0:0] bi_dir_port;
wire [0:0] bi_dir_port;
reg [0:0] bi_dir_port_reg;
reg bi_dir_port_oe;

assign bi_dir_port=bi_dir_port_oe?bi_dir_port_reg:1'bz;
```

Use bi_dir_port_oe to control the port data direction and use the intermediate variable register to change its value. equal to between two modules.
Use the inout bidirectional port to interconnect. Write to the port (that is, input into the module)

Method 2: Use force and release statements, this method cannot accurately reflect the signal change of the bidirectional port, but this method can reflect the signal change within the block. Specifically as shown:

```
module test();
wire data_inout;
reg data_reg;
reg link;
#xx; //延时
force data_inout=1'bx; // Force as input port
.....
#xx;
release data_inout; // release input port
endmodule
```

Read and write vectors from text files

1) Read a text file: Use the $readmemb system task to read a binary vector (which can contain input stimulus and output expectation) from a text file. $readmemh is used to read hex files. For example:

```
reg [7:0] mem[1:256] // a 8-bit, 256-word define memory mem
initial $readmemh ( "mem.data", mem ) // read .dat file into register mem
initial $readmemh ( "mem.data", mem, 128, 1 ) // The parameter is the address where the register loads the data always
```

2) Output text file: open the output file with ?$fopen

For example:

integer out_file; // out_file is a file description and needs to be defined as integer type
out_file = $fopen (" cpu.data "); // cpu.data is the file that needs to be opened, that is, the signal value in the final output text design can be passed through $fmonitor, $fdisplay,

2. Verilog and Ncverilog commands use library files or library directories

ncverilog -f run.f -v lib/lib.v -y lib2 +libext+.v //Generally compiled files are in run.f, library files are in lib.v, and the .v file system in the lib2 directory is automatically searched

Use library files or library directories to compile only the modules you need, not all of them

3. System tasks for Verilog Testbench signal logging:

1) The SHM database can record signal changes during design simulation. It only records your probes during the valid time

Changes in the signal of set probe on.

ex). $shm_open("waves.shm"); //Open the waveform database
$shm_probe(top, "AS"); // set probe on "top",
Second parameter: A -- signals of the specific scrape

S -- Ports of the specified scope and below, excluding library cells
C -- Ports of the specified scope and below, including library cells
AS -- Signals of the specified scope and below, excluding library cells
AC -- Signals of the specified scope and below, including library cells

There is also an M, which indicates the memories of the current scope, which can be used in combination with the above, "AM" "AMS" "AMC" nothing to indicate the ports of the current scope;

$shm_close //Close the database

2) The VCD database can also record the changes of signals during the design simulation process. It only records the changes of the signals you select. Example:

$dumpfile("filename"); //Open the database

$dumpvars(1, top.u1); //scope = top.u1, depth = 1

The first parameter indicates the depth, when it is 0, all depths are recorded; the second parameter indicates the scope, and the current scope is displayed when omitted.

$dumpvars; //depth = all scope = all
$dumpvars(0); //depth = all scope = current
$dumpvars(1, top.u1); //depth = 1 scope = top.u1
$dumpoff //Pause recording data changes, signal changes are not written to the library file
$dumpon //restore the record

3) Debussy fsdb database can also record signal changes, its advantage is that it can be combined with debussy for easy debugging.

If you want to record signals during ncverilog simulation, first set debussy:

a. setenv LD_LIBRARY_PATH LD_LIBRARY_PATH
   (path for debpli.so file (/share/PLI/nc_xl//nc_loadpli1))

b. while invoking ncverilog use the +ncloadpli1 option.
   ncverilog -f run.f +debug +ncloadpli1=debpli:deb_PLIPtr

The recording method of the fsdb database file is to use the $fsdbDumpfile and $fsdbDumpvars system functions, see VCD for the usage method

Note: When using ncverilog, in order to record the waveform correctly, use the parameter: "+access+rw", otherwise there is no read and write permission

When recording a signal or waveform, you need to point out the path of the recorded signal, such as: tb.module.u1.clk.

Notes on system tasks for signal logging:

Using the system task of signal recording in the testbench, you can record the results and waveform files you need (you can use the sigalscan tool to view), which is suitable for simulating larger systems, and is faster than global simulation. Simple to use, add in testbench:

```
initial begin
$shm_open("waves.shm");
$shm_probe("The path to record the signal", "AS");
# 10000
$shm_close; That's it.
```

4. The order of ncverilog compilation: ncverilog file1 file2 ....
Sometimes these files have dependencies. For example, if the variables defined in file1 are used in file2, it should be noted that the order of compilation is from back to front, and file2 is compiled first and then file2.

5. Forced assignment of signals force
First of all, the force statement can only appear in the process statement, that is, in the middle of initial or always. To remove the force, use the release statement.

```
initial begin force sig1 = 1'b1; ... ; release sig1; end
```

force can be assigned to wire, then the entire net is assigned; it can also be assigned to reg.

6. When loading the test vector, avoid changing on the upper and lower edges of the clock

In order to simulate the behavior of the real device, when loading the test vector, avoid changing on the upper and lower edges of the clock, but after delaying the rising edge of the clock for one time unit, the loaded test vector change. Example:

```
assign #5 c=a^b
……
@(posedge clk) #(0.1*`cycle) A=1;
```

```
********************************************************************************
// waveform output of testbench
module top;
...
initial
begin
$dumpfile("./top.vcd"); //The file name and path to store the waveform, usually in .vcd format.
$dumpvars(1,top); //Store all signal data of the top layer
$dumpvars(2,top.u1); //Store all data signals two layers below top.u1 (including top.u1)
$dumpvars(3,top.u2); //Store all data signals of the three layers below top.u2 (including the top.u2
layer)
$dumpvars(0,top.u3); //Store all data signals of all layers below top.u3
end
endmodule
//Generate random numbers, seed is the seed
$random(seed);
ex: din <= $random(20);
//Simulation time, 64-bit data of unsigned type
$time
ex:
...
time condition_happen_time;
...
condition_happen_time = $time;
...
$monitor($time,"data output = %d", dout);
...
//parameter
parameter para1 = 10,
para2 = 20,
para3 = 30;
// show tasks
$display();
// monitor task
$monitor();
// delay model
specify
...
//describe pin-to-pin delay
endspecify
ex:
module nand_or(Y,A,B,C);
input A,B,C;
output Y;
AND2 #0.2 (N,A,B);
OR2 #0.1 (Y,C,N);
specify
(A*->Y) = 0.2;
```

```
(B*->Y) = 0.3;
(C*->Y) = 0.1;
endspecify
endmodule
//timescale
`timescale unit time/time precision
//file I/O
```

1. Open the file

```
integer file_id;
file_id = fopen("file_path/file_name");
```

2. Write to file

```
//$fmonitor keeps logging as long as there are changes
$fmonitor(file_id, "%format_char", parameter);
eg fmonitor(file_id, "%m: %t in1=%d o1=%h", $time, in1, o1);
//$fwrite needs to trigger conditions to record
$fwrite(file_id, "%format_char", parameter);
//$fdisplay needs to trigger conditions to record
$fdisplay(file_id, "%format_char", parameter);
$fstrobe();
```

3. Read the file

```
integer file_id;
file_id = $fread("file_path/file_name", "r");
```

4. Close the file

```
$fclose(fjile_id);
```

5. Set the initial value of the memory from the file

```
$readmemh("file_name", memory_name); //Initialize data as hexadecimal
$readmemb("file_name", memory_name); //Initialize data to binary
//Simulation control
$finish(parameter); //parameter = 0,1,2
$stop(parameter);
//read in SDF file
$sdf_annotate("sdf_file_name", module_instance, "scale_factors");
//module_instance: The instance name corresponding to the sdf file.
//scale_factors: Adjust the delay parameters for the minimum delay min, typical delay typ, and
maximum delay max in timming delay
//generate statement, defined in Verilog-2001. Used to express repetitive actions
//The genvar type variable must be declared in advance as the index of the generate loop
eg:
genvar i;
generate for(i = 0; i < 4; i = i + 1)
begin
assign = din = i % 2;
end
```

```
endgenerate
//Resource Sharing
always @(A or B or C or D)
sum = sel ? (A+B) C+D);
//The above example uses two adders and a MUX, which has a large area
//The following example uses an adder and two MUXs, with a small area
always @(A or B or C or D)
begin
tmp1 = sel ? A:C;
tmp2 = sel ? B ;
end
always @(tmp1 or tmp2)
sum = tmp1 + tmp2;
```

****

template:

```
module testbench; //Define a module with no input and output
reg …… //Define the input of DUT as reg type
...
wire... //Define the output of DUT as wire type
...
//Instantiate the DUT here
initial
begin
... //add incentives here (there can be multiple such structures)
end
always... //usually define the clock signal here
initial
//add comparison statement here (optional)

end
initial
//Add output statement here (display simulation results on screen)
end
endmodule
```

Here are some tips for writing Testbench:

1. If there are some repetitive items in the incentive, you can consider writing these statements into a task, which will bring great convenience to writing and simulation. For example, the stimulus for a memory testbench can include write, read, etc. tasks.
2. If the DUT contains bidirectional signals (inout), pay attention when writing the testbench. It needs a reg variable to represent its input, and a wire variable to represent its output.
3. If the initial block statement is too complicated, consider dividing it into several complementary and related parts and describe it with several initial blocks. During simulation, these initial blocks run concurrently. This makes it easier to read and modify.
4. Each testbench preferably contains a $stop statement to indicate when the simulation ends.

Finally, a simple example is provided (from the Xilinx documentation):

DUT：

```
module shift_reg (clock, reset, load, sel, data, shiftreg);
input clock;
input reset;
input load;
input [1:0] sel;
input [4:0] data;
output [4:0] shiftreg;
reg [4:0] shiftreg;
always @ (posedge clock)
begin
if (reset)
shiftreg = 0;
else if (load)
shiftreg = data;
else
case (sel)
2’b00 : shiftreg = shiftreg;
2’b01 : shiftreg = shiftreg << 1;
2’b10 : shiftreg = shiftreg >> 1;
default : shiftreg = shiftreg;
endcase
end
endmodule
```

Testbench：

```
module testbench; // declare testbench name
reg clock;
reg load;
reg reset; // declaration of signals
wire [4:0] shiftreg;
reg [4:0] data;
reg [1:0] sel;
// instantiation of the shift_reg design below
shift_reg dut(.clock (clock),
.load (load),
.reset (reset),
.shiftreg (shiftreg),
.data (data),
.sel (sel));
//this process block sets up the free running clock
initial begin
clock = 0;
forever #50 clock = ~clock;
end
```

```
initial begin// this process block specifies the stimulus.
reset = 1;
data = 5’b00000;
load = 0;
sel = 2’b00;
#200
reset = 0;
load = 1;
#200
data = 5’b00001;
#100
sel = 2’b01;
load = 0;
#200
sel = 2’b10;
#1000 $stop;
end
initial begin// this process block pipes the ASCII results to the
//terminal or text editor
$timeformat(-9,1,"ns",12);
$display(" Time Clk Rst Ld SftRg Data Sel");
$monitor("%t %b %b %b %b %b %b", $realtime,
clock, reset, load, shiftreg, data, sel);
end
endmodule
```