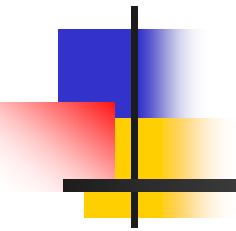


第10章

Verilog行为仿真

10.1 Verilog行为仿真流程

图 10-1 HDL 系统设计描述层次

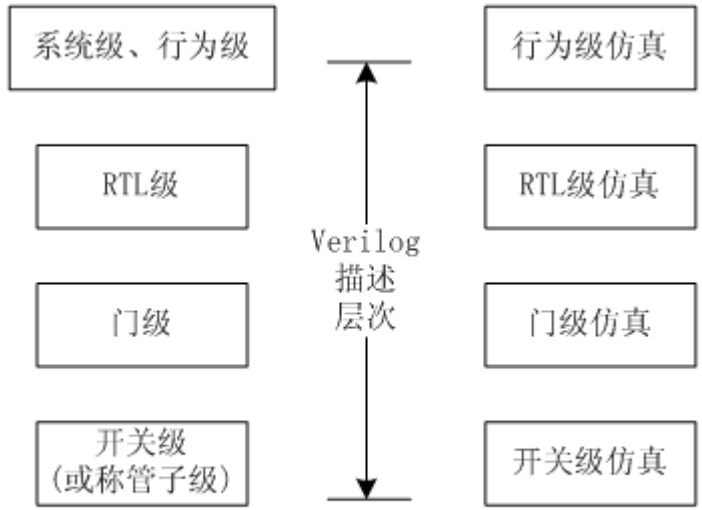


图 10-2 Verilog 描述层次与仿真层次

10.1 Verilog行为仿真流程

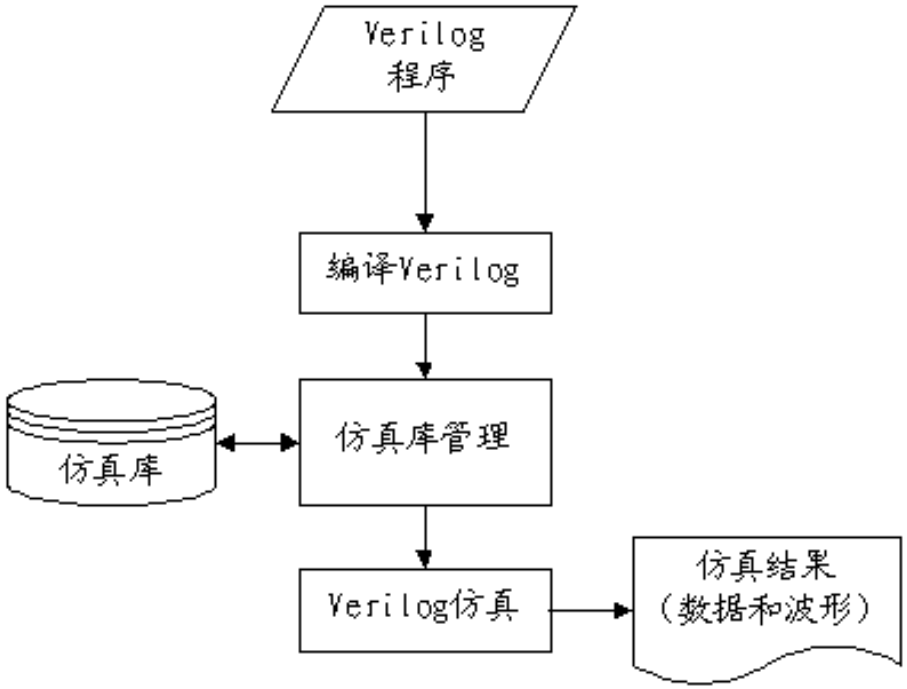


图 10-3　Verilog 仿真流程

10.2 ModelSim应用向导

【例 10-1】

```
`timescale 10ns/1ns
module cnt4( input clk,  input rst, input [3:0] d,
  input load,  output reg[3:0] q,  output reg cout);
  always @(posedge clk,posedge rst) begin
     if(rst)   {cout,q} <= 5'h0;  // 异步复位
     else  if(load)  q <= d;       // 同步装载
     else   {cout,q} <= {cout,q} + 1'b1;  // 上升沿计数器加 1
end
endmodule
```

10.2 ModelSim应用向导

1. 启动**ModelSim**

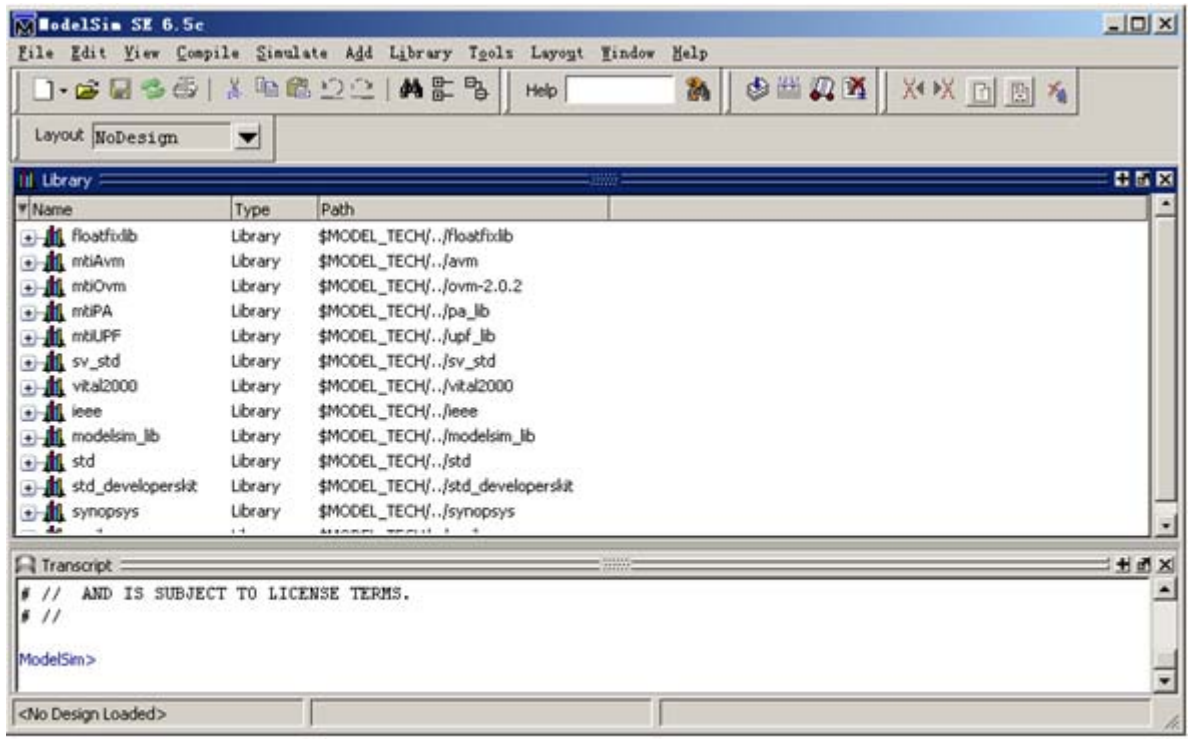


图 10-4 ModelSim 的启动界面

10.2 ModelSim应用向导

2. 建立仿真工程项目

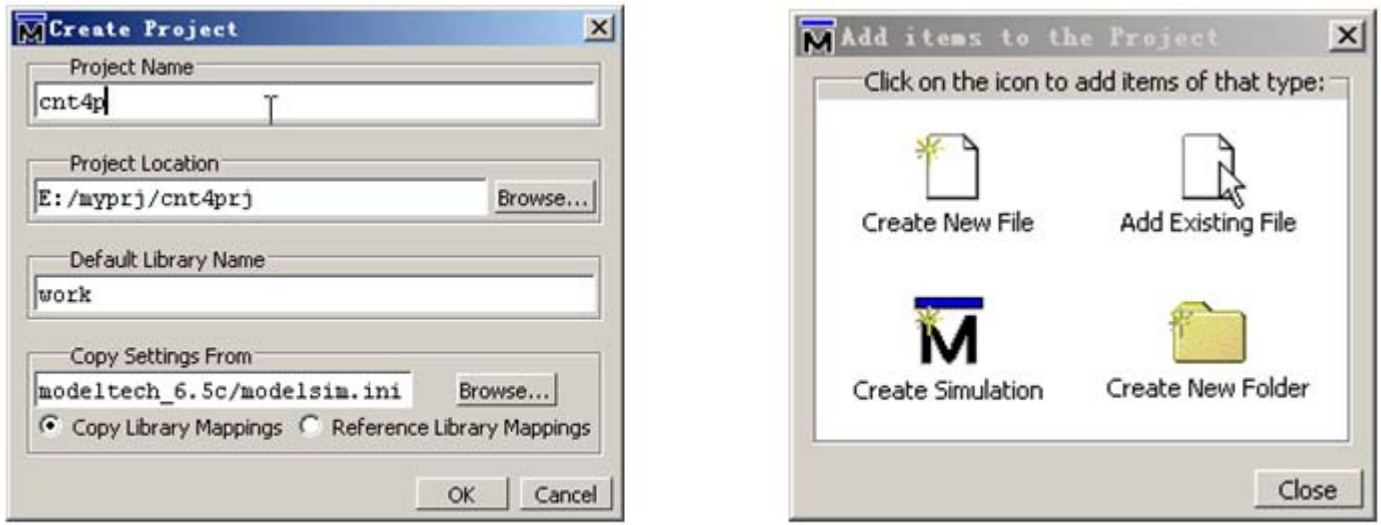


图 10-5　建立工程项目

10.2 ModelSim应用向导

2. 建立仿真工程项目

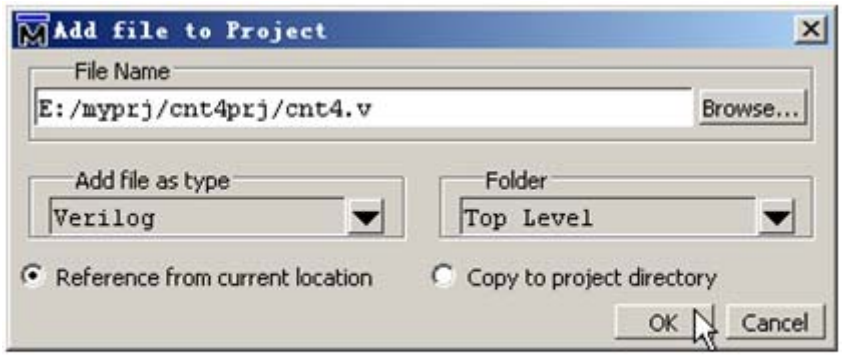


图 10-6　加入 HDL 文件

10.2 ModelSim应用向导

3. 编译仿真文件

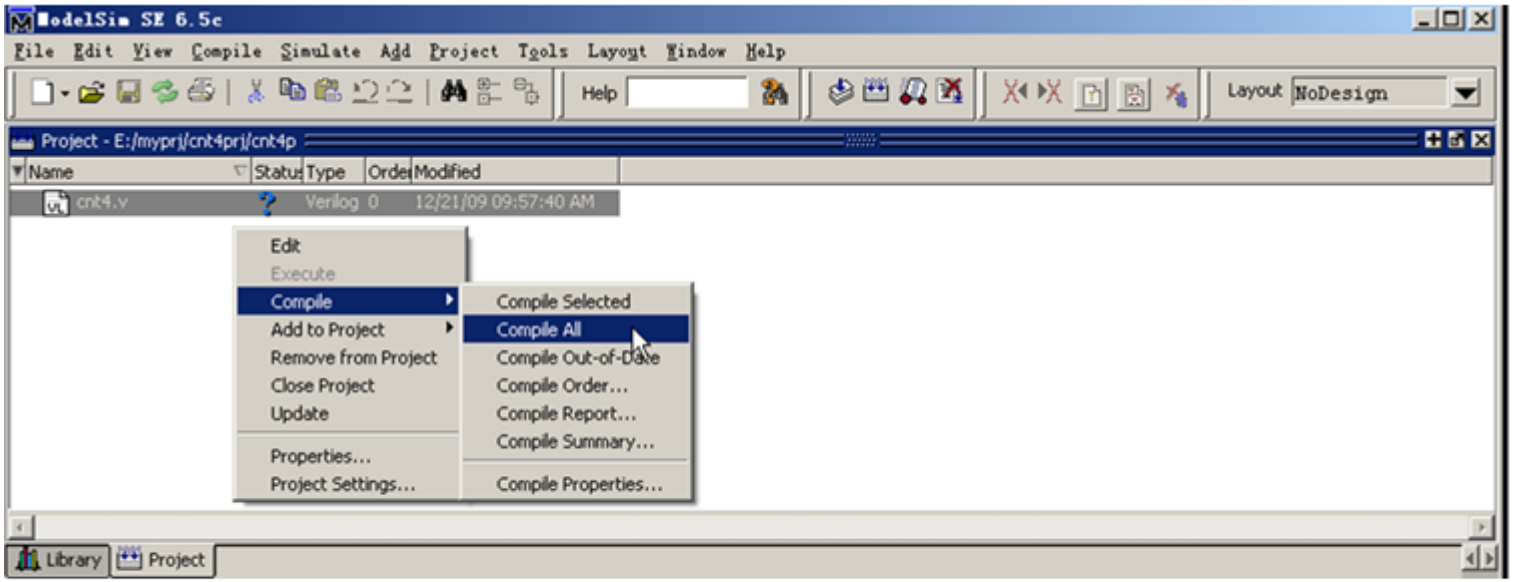


图 10-7 开始编译仿真文件

10.2 ModelSim应用向导

3. 编译仿真文件

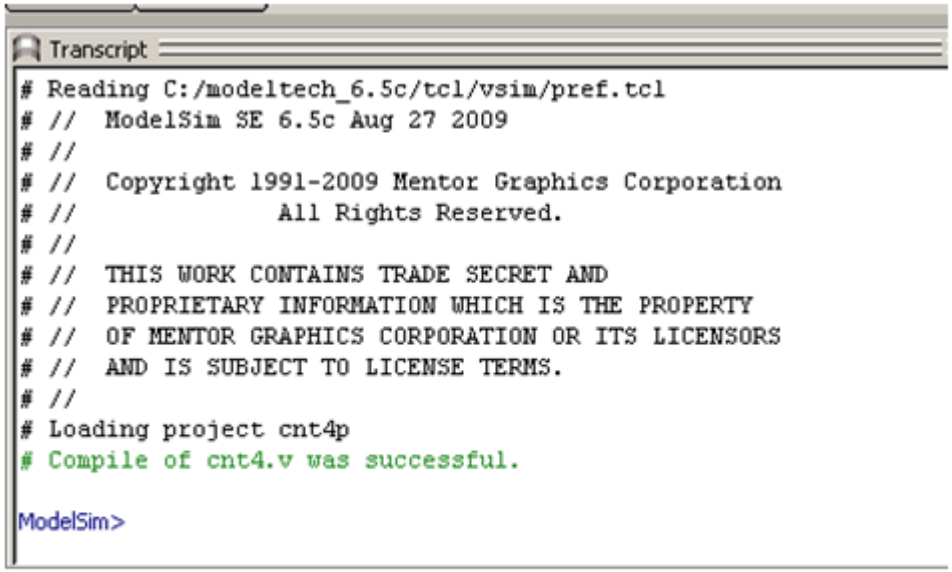


图 10-8 ModelSim 编译时的提示信息

10.2 ModelSim应用向导

4. 装载仿真模块和仿真库

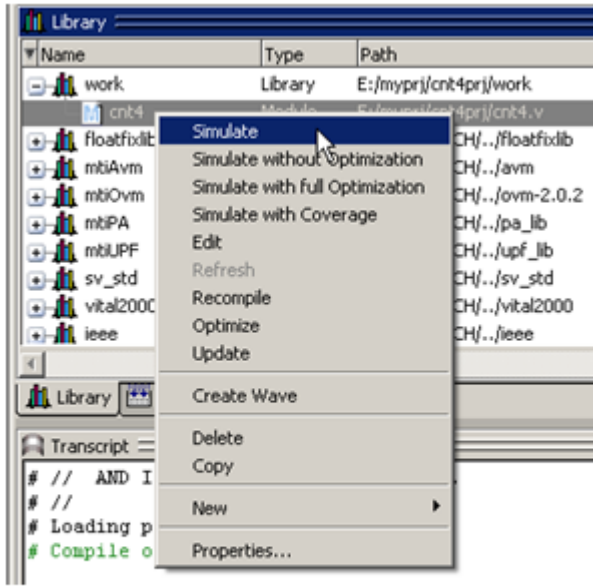


图 10-9 装载仿真模块

10.2 ModelSim应用向导

4. 装载仿真模块和仿真库

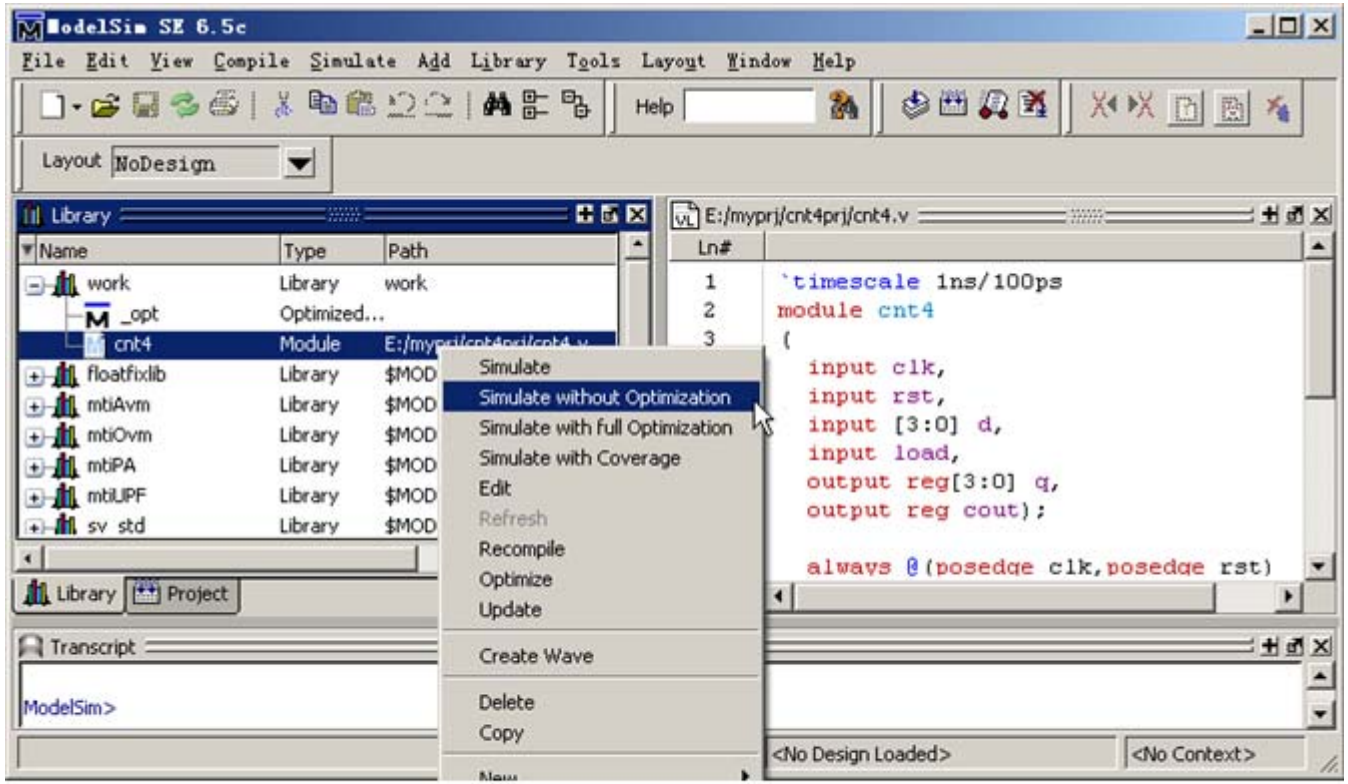


图 10-10 启动仿真（无优化模式）

5. 执行仿真

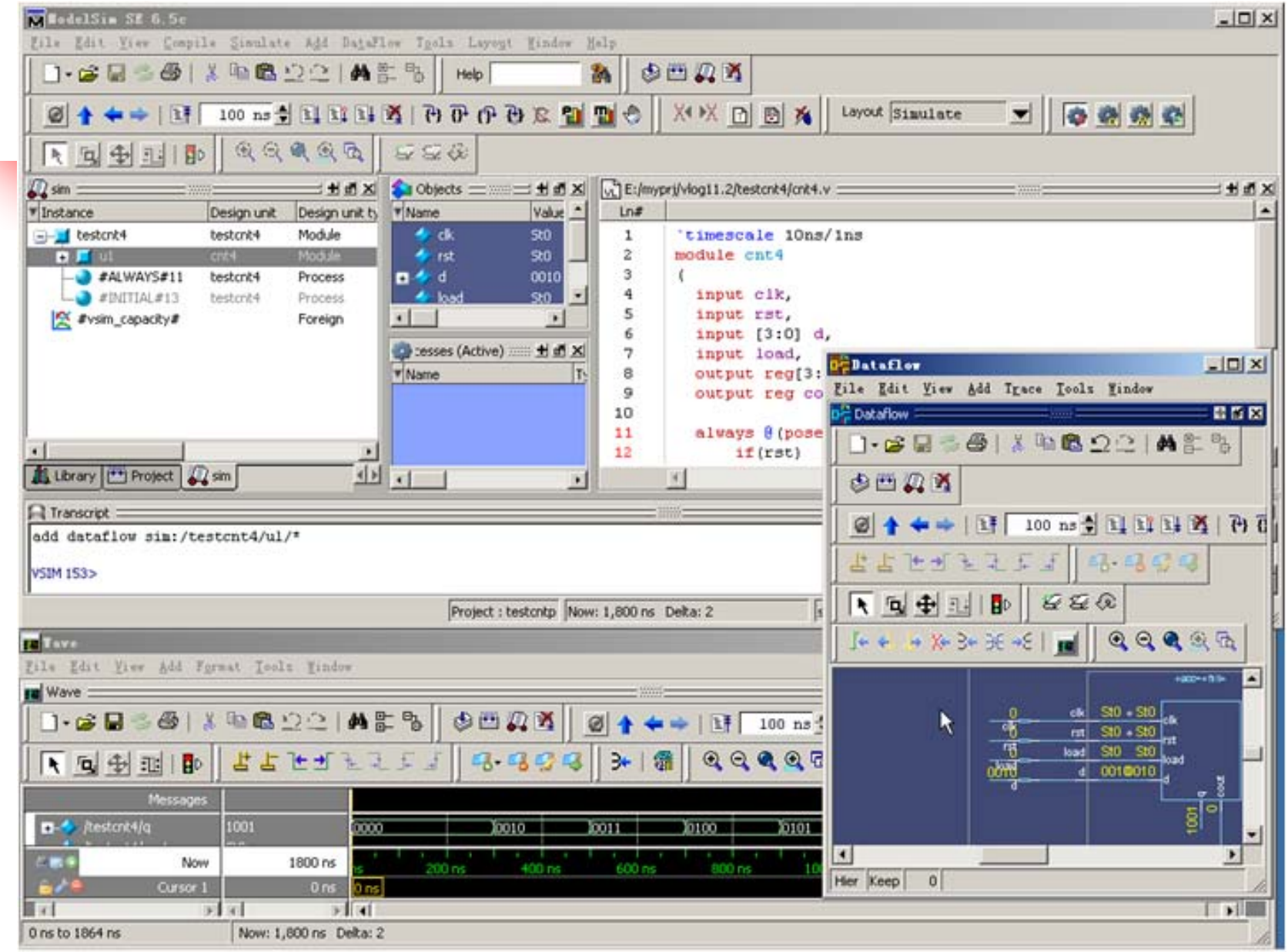


图 10-11　ModelSim 的仿真观察窗

10.2 ModelSim应用向导

5. 执行仿真

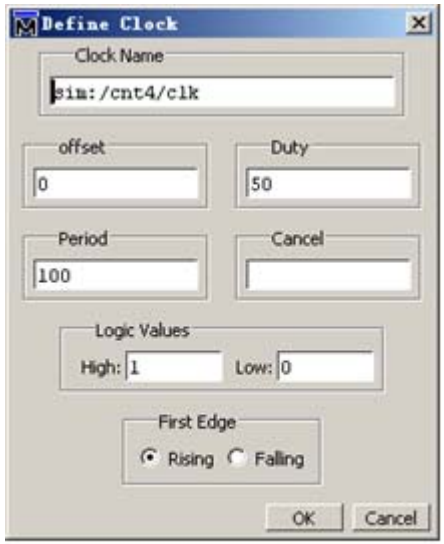


图 10-12 Clock 设置

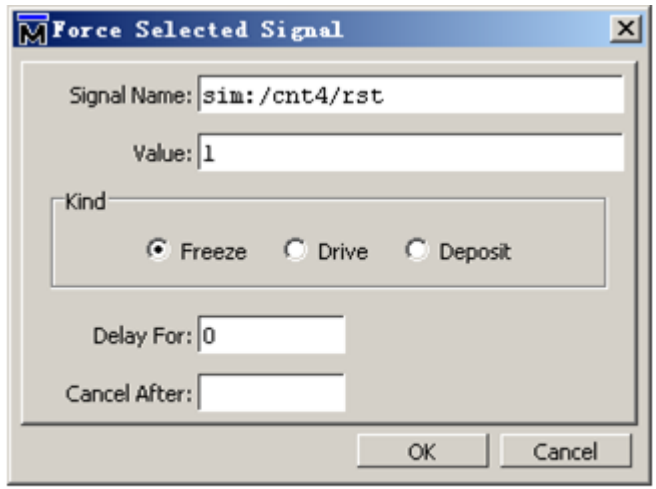


图 10-13 Force 对话框

10.2 ModelSim应用向导

5. 执行仿真

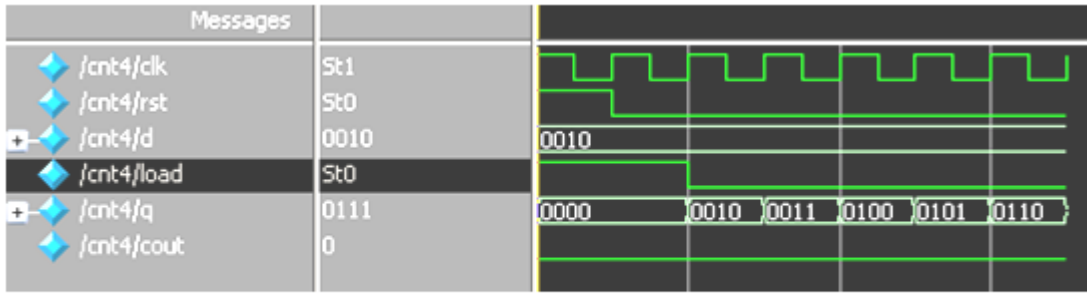


图 10-14 仿真结果

10.3 Verilog系统任务和预编译语句

10.3.1 系统任务、系统函数

1. $display

【例 10-2】

```
module sdisp1;
integer i;     // i 为整型
reg [3:0] x;   // x 为 4 位
initial begin // initial 块，只执行一次。initial 使用方法参见本章后面部分
i=21;          // i 赋值 21
x=4'he;        // x 赋值十六进制数 e
$display("1\t%d\n2\t%h\\",i,x);  // 输出显示
end  endmodule
```

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

10.3 Verilog系统任务和预编译语句

10.3.1 系统任务、系统函数

1. $display

表 10-1 Verilog 转义符

转义符	含义	转义符	含义	转义符	含义
\n	换行	%b	二进制格式	%v	显示信号强度
\t	Tab	%o	八进制格式	%m	显示层次名
\\	字符\	%d	十进制格式	%s	字符串格式
\"	字符"	%h	十六进制格式	%t	显示当前时间
\ddd	1～3 位八进制表示的 ASCII 字符	%l	显示：库绑定信息	%u	未格式化二值数据
%%	字符%	%c	字符格式	%z	未格式化四值数据
%e	以科学计数法显示实数值	%f	以十进制显示实数值	%g	取%e、%f 格式中最短的显示

10.3 Verilog系统任务和预编译语句

10.3.1 系统任务、系统函数

1. $display

【例 10-3】

```
module sdisp2;          // 无输入输出端口
reg [31:0] rval;        // 32 位 reg 类型
pulldown (pd);          // pd 接下拉电阻，pll down 用法见本章后续内容
initial begin           // initial 块
rval = 101;             // 赋整数 101
$display("rval = %h hex %d decimal",rval,rval); // 十六进制、十进制显示
$display("rval = %o octal\nrval = %b bin",rval,rval);// 八进制、二进制显示
$display("rval has %c ascii character value",rval); // 字符格式显示输出
$display("pd strength value is %v",pd);             // pd 信号强度显示
$display("current scope is %m");                    // 当前层次模块名显示
$display("%s is ascii value for 101",101);          // 字符串显示
$display("simulation time is %t", $time);           // 显示当前仿真时间
end
endmodule
```

10.3 Verilog系统任务和预编译语句

10.3.1 系统任务、系统函数

1. $display

```
# 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
```

10.3 Verilog系统任务和预编译语句

2. $write

```
$write ("带格式字符串", 参数 1, 参数 2, ... );
```

3. $strobe和$monitor

```
$strobe ("带格式字符串", 参数 1, 参数 2, ... );
$monitor ("带格式字符串", 参数 1, 参数 2, ... );
```

10.3 Verilog系统任务和预编译语句

3. $strobe和$monitor

【例 10-4】

```
module sdisp3;                                    // 无输入输出信号
reg [1:0]a;                                       // a 为 2 位 reg
reg b;
initial $monitor("\$monitor: a = %b", a); // $monitor 监测 a 的变化
initial begin                                     // initial 块，只执行一次
b = 0;                                            // b 赋值 0，阻塞赋值
a = 0;                                            // a 赋值 0，阻塞赋值
$strobe ("\$strobe : a = %b", a);          // $strobe 显示 a 的赋值
a = 1;                                           // a 赋值 1
$display ("\$display: a = %b", a);          // $display 显示 a 的当前赋值
a = 2;                                           // a 赋值 2
$monitor("\$monitor: b = %b", b);          // $monitor 取代前一个$monitor
a = 3;                                          // a 赋值 3
#30 $finish;                                    // 延时 30 个时间单位后，仿真终止
end
always #10 b = ~b;                  // b 每隔 10 个时间单位，值反转，Clock 信号
endmodule
```

10.3 Verilog系统任务和预编译语句

3. $strobe和$monitor

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

4. $finish和$stop

```
$finish;
$stop;
```

10.3 Verilog系统任务和预编译语句

4. $finish和$stop

【例 10-5】

```
module sdisp4();
reg [3:0]a,b;    // a，b 都为 4 位 reg
initial $monitor($time," \$monitor:a=%0d,b=%d",a,b);//显示变化及当前时间
initial begin    // initial 块，只执行一次
b = 0;           // b 赋值 0
$strobe ($time," \$strobe : a = %0d", a);   // 显示 a 的赋值结果
$monitoron;      // 开启$monitor
a = 1;           // a 赋值 1，阻塞赋值
a <= 2;          // a 赋值 2，非阻塞赋值
$display ($time," \$display: a = %d", a);   // 显示 a 的当前值
a = 3;           // a 赋值 3，阻塞赋值
#25 $monitoroff;// 关闭$monitor
#10 $stop;       // 10 个时间单位后，暂停仿真器仿真
end
always #10 b = b+1; // b 每过 10 个时间单位，加 1
endmodule
```

10.3 Verilog系统任务和预编译语句

4. $finish和$stop

```
#                    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
```

10.3 Verilog系统任务和预编译语句

5. $time

$time 返回一个 64 位整数时间值。
$stime 返回一个 32 位整数时间值。
$realtime 返回一个实数时间值。
$timeformat 控制时间的显示方式。

```
$monitor("%d d=%b,e=%b", $stime, d, e); //$time
```

10.3 Verilog系统任务和预编译语句

6. 文件操作

```
文件句柄 =$fopen("文件名")  // 打开文件
$fstrobe(文件句柄,"带格式字符串", 参数列表) //strobe 到文件
$fdisplay(文件句柄,"带格式字符串", 参数列表 t) //display 到文件
$fmonitor(文件句柄,"带格式字符串", 参数列表 t) //monitor 到文件，可以多个进程
$fwrite(文件句柄,"带格式字符串", 参数列表) //write 到文件
$fclose(文件句柄);      // 关闭文件
$feof(文件句柄);        //查询是否已到文件末尾
```

10.3 Verilog系统任务和预编译语句

6. 文件操作

```
$dumpfile("文件名");   // 导出到文件，这里文件后缀为 vcd
$dumpvar;             //导出当前设计的所有变量
$dumpvar(1, top);     // 导出顶层模块中的所有变量
$dumpvar(2, top);     // 导出顶层模块和顶层下第 1 层模块的所有变量
$dumpvar(n, top);     //导出顶层模块到顶层下第 n-1 层模块的所有变量
$dumpvar(0, top);     //导出顶层模块和所有层次模块的所有变量
$dumpon;              // 导出初始化
$dumpoff;             //停止导出
```

10.3 Verilog系统任务和预编译语句

6. 文件操作

【例 10-6】

```
module fileio_demo;              // 文件读写
integer fp_r, fp_w, cnt;         // 定义文件句柄，整型
reg [7:0] reg1, reg2, reg3; //  3 个 8 位 reg 值
initial begin
  fp_r = $fopen("in.txt", "r"); // 以只读方式打开 in.txt
  fp_w = $fopen("out.txt", "w");// 以写方式打开 out.txt
  while(!$feof(fp_r)) begin      // 循环读写文件，直到 in.txt 末尾
    cnt = $fscanf(fp_r, "%d %d %d", reg1, reg2, reg3);  // 读一行
    $display("%d %d %d", reg1, reg2, reg3);             // 显示读到的值
    $fwrite(fp_w, "%d %d %d\n", reg3, reg2, reg1);      // 反序写一行
  end
  $fclose(fp_r);     // 关闭文件 in.txt
  $fclose(fp_w);     // 关闭文件 out.txt
end
endmodule
```

10.3 Verilog系统任务和预编译语句

10.3.2 预编译语句

1. `define 宏定义

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

2. `include 文件包含

```
`include "abc.v"
```

3. Translate _on与translate _off

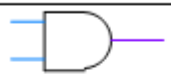
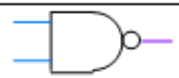
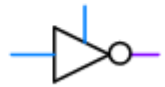
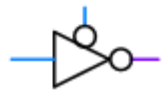
```
// synthesis translate_off

// synthesis translate_on
```

10.4 基本元件与用户自定义元件（UDP）

10.4.1 基本元件及其用法

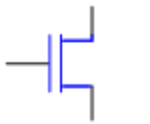
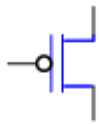
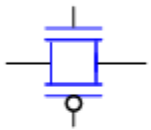
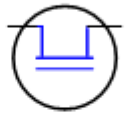
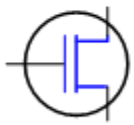
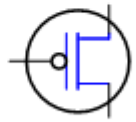
表 10-2 基本元件——基本门

符号	图形	功能	符号	图形	功能
and		与门	nand		与非门
or		或门	nor		或非门
xor		异或门	xnor		同或门
buf		缓冲门	not		非门
bufif1		三态门高电平使能	notif1		三态反向门 高电平使能
bufif0		三态门 低电平使能	notif0		三态反向门 低电平使能

10.4 基本元件与用户自定义元件（UDP）

10.4.1 基本元件及其用法

表 10-3 基本元件——开关级

图形	符号	功能	图形	符号	功能
	nmos	n 型 mos 开关		pmos	p 型 mos 开关
	rnmos	带电阻 n 型 mos 开关		rpmos	带电阻 p 型 mos
	cmos	cmos 开关		tran	双向传输管
	rcmos	带电阻 cmos 开关		rtran	带电阻 双向传输管
	tranif1	双向传输开关 高电平使能		tranif0	双向传输开关 低电平使能
	rtranif1	带电阻高电平使 能双向传输开关		rtranif0	带电阻低电平使 能双向传输开关
	pullup	上拉电阻		pulldown	下拉电阻

10.4 基本元件与用户自定义元件（UDP）

10.4.1 基本元件及其用法

表 10-4 信号值的强度

等级	1 强度	1 强度表示	0 强度	0 强度表示
7	supply1	Su1	supply0	Su0
6	strong1	St1	strong0	St0
5	pull1	Pu1	pull0	Pu0
4	large1	La1	large0	La0
3	weak1	We1	weak0	We0
2	medium1	Me1	medium0	Me0
1	small1	Sm1	small0	Sm0
0	highz1	HiZ	highz0	HiZ

10.4 基本元件与用户自定义元件（UDP）

10.4.1 基本元件及其用法

【例 10-7】 反相器开关级建模

```
module mosinv(a,b); // 开关级反相器建模
  input a;   output b;
supply1 vcc;          // 电源信号 vcc
supply0 gnd;          // 电源地信号 gnd
pmos p1(b,vcc,a);     // pmos
nmos n1(b,gnd,a);     // nmos
endmodule
```

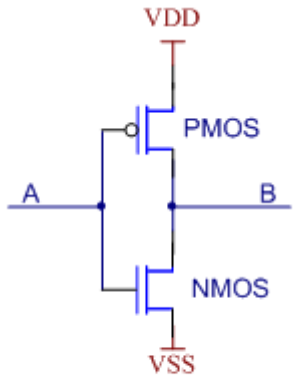


图 10-15 Verilog 开关级建模

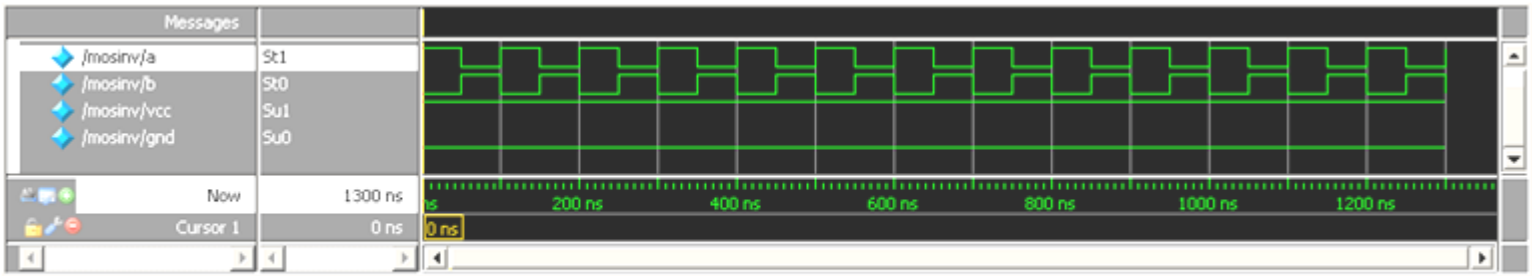


图 10-16 反相器开关级建模 ModelSim 仿真波形图

10.4 基本元件与用户自定义元件（UDP）

10.4.2 用户自定义元件（UDP）

```
primitive 元件名(输出端口名, 输入端口名1, 输入端口名2,…)
    output 输出端口名;
    input 输入端口名1, 输入端口名2,…;
    reg 输出端口名;
    initial begin
        // 输出端口寄存器或时序逻辑内部寄存器赋初值(0,1,X);
        ……
    end
    table
    // 输入1    输入2    …    :    输出
    逻辑值      逻辑值          :   逻辑值;
    逻辑值      逻辑值          :   逻辑值;
    ……
    endtable
endprimitive
```

10.4 基本元件与用户自定义元件（UDP）

【例 10-8】 mux41 UDP 元件

```
primitive  pmux41(out,in1,in2,in3,in4,ctrl1,ctrl2); // UDP
input  in1,in2,in3,in4,ctrl1,ctrl2;
output out;  // 只有一个输出端口
table        // 真值表
// in1  in2  in3   in4   ctrl1  ctrl2   :   out
0     ?     ?    ?     0     0    :   0;      // ?  表示任意值
1     ?     ?    ?     0     0    :   1 ;
?     0     ?    ?     0     1    :   0 ;
?     1     ?    ?     0     1    :   1 ;
?     ?     0    ?     1     0    :   0 ;
?     ?     1    ?     1     0    :   1 ;
?     ?     ?    0     1     1    :   0 ;
?     ?     ?    1     1     1    :   1 ;
endtable
endprimitive
```

10.4 基本元件与用户自定义元件（UDP）

【例 10-9】 UDP 元件例化

```
module xmux41( input da, db, dc, dd, c1, c2, // 4 选 1 多路选择器
 output  dout);
pmux41 m4(dout,da,db,dc,dd,c1,c2);   // 例化 UDP
endmodule
```

【例 10-10】 DFF UDP 元件

```
primitive primdff (Q, Clk, Data) ;      // DFF UDP
  output Q ;   input Data, Clk;   // 只有一个输出
  reg Q ;
  initial Q = 0;       // Q 的初始化
  table
    // Clk Data Q (当前状态) Q(下一个状态)
      (01) 0 : ? : 0 ;        // 上升沿，Data 为 0
      (01) 1 : ? : 1 ;        // 上升沿，Data 为 1
      (0x) 1 : 1 : 1 ;        // 其他状态，保持
      (0x) 0 : 0 : 0 ;
      (?0) ? : ? : - ;
      ? (??): ? : - ;
  endtable
endprimitive
```

10.4 基本元件与用户自定义元件（UDP）

10.4.2 用户自定义元件（UDP）

表 10-5 UDP Table 中符号含义

符号	表示	说明
1	逻辑 1	
0	逻辑 0	
x	未知	允许在所有 UDP 的输入域以及时序 UDP 的当前状态域
?	指代 0、1 和 x	不允许在输出域
b	指代 0、1	允许在所有 UDP 的输入域以及时序 UDP 的当前状态域，不允许在输出域
-	不改变	仅允许在时序 UDP 的输出域
(vw)	值从 v 到 w 变化	v 和 w 可以是 0、1、x、？或 b 其中之一，不允许在输出域
*	同(??)	任何输入值变化
r	同(01)	输入上升沿
f	同(10)	输入下降沿
p	指代(01)、(0x)和(x1)	输入可能的正边沿
n	指代(10)、(x0)和(x0)	输入可能的负边沿

10.4 基本元件与用户自定义元件（UDP）

10.4.2 用户自定义元件（UDP）

【例 10-11】 D-Latch UDP 元件

```
primitive primlatch (q, clock, data);
  output q;
  input clock, data;
  reg q;
  table
  // clock data q q+
  0 1 : ? : 1 ;  // 直通
  0 0 : ? : 0 ;  // 直通
  1 ? : ? : - ;  // 锁存
  endtable
  endprimitive
```

【例 10-12】 DFF UDP 元件

```
primitive dff1 (q, clk, d);
input clk, d;
output q; reg q;
initial q = 1'b1;
table
// clk d q q+
r 0 : ? : 0 ;  // 上升沿
r 1 : ? : 1 ;  // 上升沿
f ? : ? : - ;  // 下降沿
? * : ? : - ;  //其他状态
endtable
endprimitive
```

10.4 基本元件与用户自定义元件（UDP）

【例 10-13】 双边沿 D 触发器元件

```
primitive prim_ddff (q, clk, d);    // 双边沿 D 触发器 UDP
input clk, d;
output q; reg q;
initial q = 1'b1;   // q 初始化
table
// clk d q q+
r 0 : ? : 0 ;   // 上升沿
r 1 : ? : 1 ;   // 上升沿
f 0 : ? : 0 ;   // 下降沿
f 1 : ? : 1 ;   // 下降沿
? * : ? : - ;   // 其他状态
endtable
endprimitive
// 例化 prim_dff
module test_pddff(q, clk, d);
input clk, d;
output q;
prim_ddff U1(q,clk,d);  // 例化 UDP
endmodule
```

10.4 基本元件与用户自定义元件（UDP）

10.4.2 用户自定义元件（UDP）

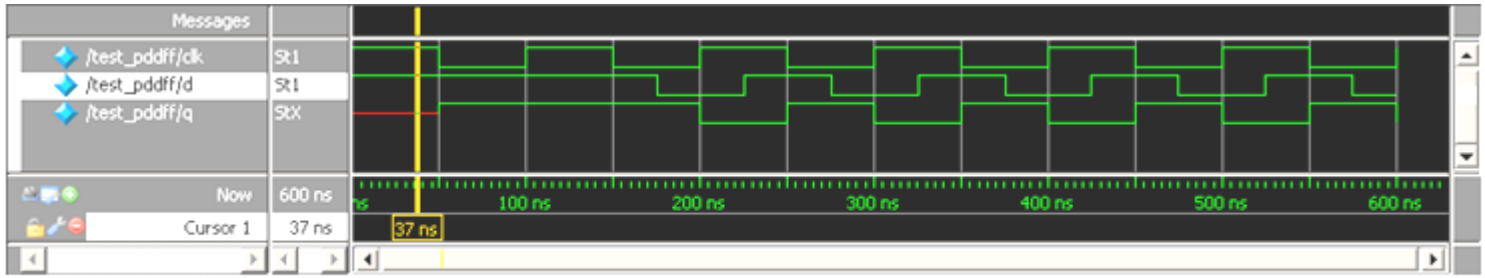


图 10-17　双边沿 D 触发器 UDP 的仿真图

10.5 延时模型

10.5.1 # 延时

【例 10-14】

```
module  d2x ();
reg [1:0] a;      reg [1:0] c;
initial #3 c=a;                              // 3 个时间单位后 c 赋值 a
initial begin
  $monitor ("%g a = %0d c=%0d", $time, a, c);   // 监控 a、c
  a = 1;                                     // a 赋值 1
  #10 a = 2;                                 // a 赋值 2
  #0 a = 0;                                  // 在 0 个时间单位后，a 赋值 0
  #10 a = 3;                                 // 在 10 个时间单位后，a 赋值 3
  #10 $stop;                                 // 在 10 个时间单位后，暂停仿真
end     endmodule
```

```
# 0 a = 1 c=x
# 3 a = 1 c=1
# 10 a = 0 c=1
# 20 a = 3 c=1
```

10.5 延时模型

10.5.2 门延时

【例 11-15】

```
module  dnot ();
reg in; wire out;
not #(3,4) (out,in);     // 例化 not，同时说明门延时
initial begin
  $monitor ("%g in = %b out=%b", $time, in, out); // 监控 in、out
  in = 0;                 // 初始赋值 0
  #10 in = 1;             // 10 个时间单位后，赋值 1
  #10 in = 0;             // 10 个时间单位后，赋值 0
  #10 $stop;              // 10 个时间单位后，暂停仿真
end    endmodule
```

```
# 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
```

10.5 延时模型

10.5.3 延时说明块

【例 10-16】

```
module veridelay(out,a,b,c,d);
output out;    input a,b,c,d;
wire e,f;
specify  // specify 延时说明块
    (a=>out)=3;   //  a 到 out 延时 3 个时间单位
    (b=>out)=3;   //  b 到 out 延时 3 个时间单位
    (c=>out)=5;   //  c 到 out 延时 5 个时间单位
    (d=>out)=51; //  d 到 out 延时 51 个时间单位
endspecify
and U1(e,a,b);   // 逻辑图描述
and U2(f,c,d);
and U3(out,e,f);
endmodule
```

10.6 其他仿真语句

10.6.1 initial语句

【例 10-17】

```
module rstclk (clk,rst);
output reg clk, rst;
initial begin                  // initial 块，初始化，只执行一次
  clk = 0;    rst = 0;         // 初始 clk、rst 都为 0
  #20 rst = 1;                 // 20 个时间单位后 rst 为 1
  #50 rst = 0;                 // 50 个时间单位的 rst 复位脉冲
  #100 $stop;                  // 暂停仿真
end
always  #10 clk = !clk; // 产生 clk 时钟
endmodule
```

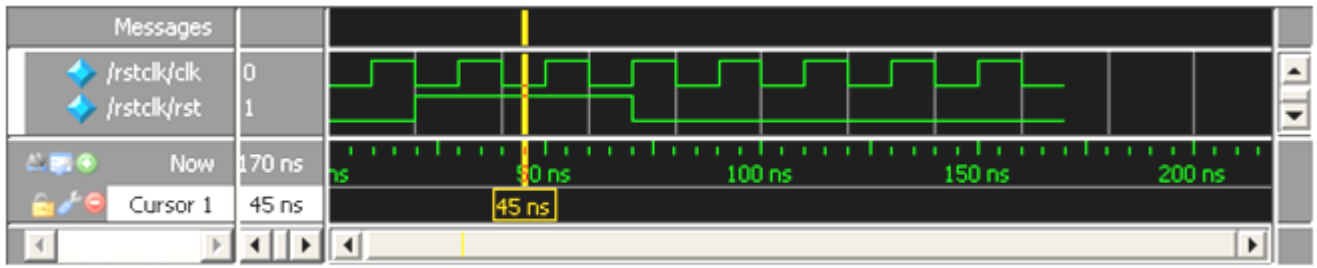


图 10-18　例 11-17 的仿真结果

10.6 其他仿真语句

10.6.2 fork-join块语句

【例 10-18】

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

【例 10-19】

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

【例 10-20】

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

10.6 其他仿真语句

10.6.2 fork-join块语句

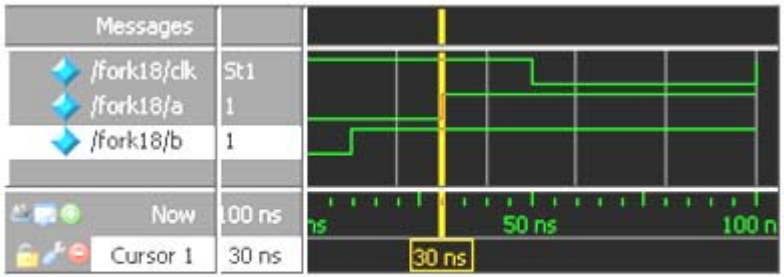


图 10-19 例 10-18 仿真波形

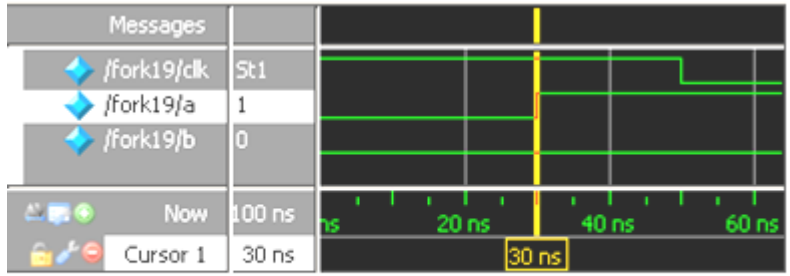


图 10-20 例 10-19 仿真波形

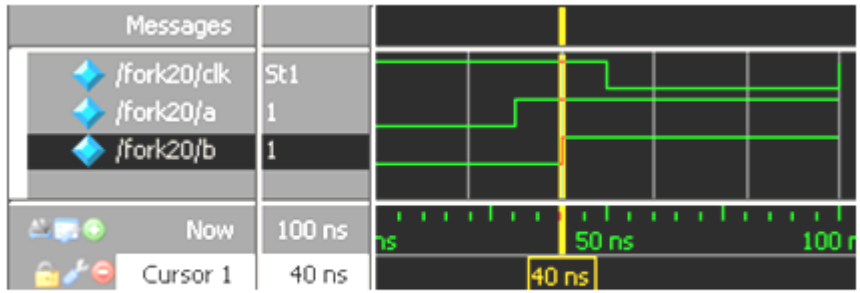


图 10-21 例 10-20 仿真波形

10.6 其他仿真语句

10.6.2 fork-join块语句

【例 10-21】

```
module fork21(clk, a, b,
  c, d, e, f);
    input clk;
    output reg a, b, c, d, e, f;
     initial begin // 初始化
        a = 0;     b = 0;
        c = 0;     d = 0;
        e = 0;     f = 0;
     end
     always @(posedge clk)
      fork // fork 块中语句并行执行
          #20 a = 1;
          #20 b = 1;
          begin // 块中语句顺序执行
             #20 c = 1;
             #20 d = 1;
             #20 e = 1;
          end
          #20 f = 1;
     join
    endmodule
```

【例 10-22】

```
module fork21(clk, a, b,
   c, d, e, f);
  input clk;
  output reg a, b, c, d, e, f;
  initial begin // 初始化
     a = 0;    b = 0;
     c = 0;    d = 0;
     e = 0;    f = 0;
  end
  always @(posedge clk)
    begin  // 块中语句顺序执行
      #20 a = 1;
      #20 b = 1;
      fork // fork 块中语句并行执行
         #20 c = 1;
         #20 d = 1;
         #20 e = 1;
       join
        #20 f = 1;
    end
endmodule
```

10.6 其他仿真语句

10.6.2 fork-join块语句

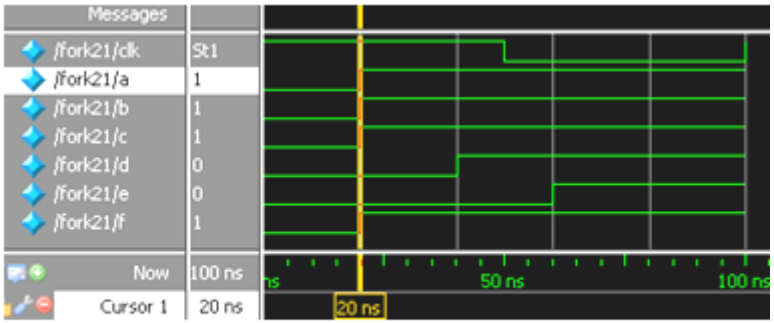


图 10-22 例 10-21 的仿真波形

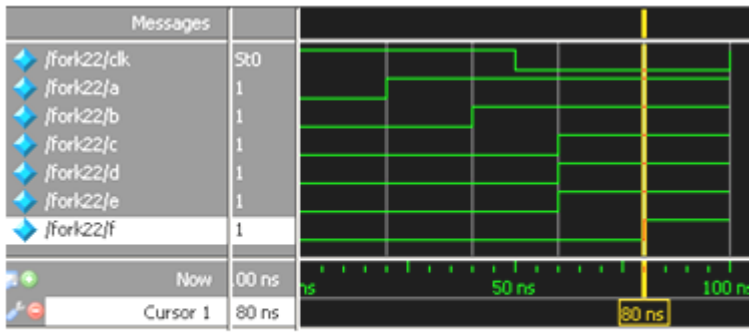


图 10-23 例 10-22 的仿真波形

10.6 其他仿真语句

10.6.3 wait语句

wait (条件表达式) 语句;

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

10.6 其他仿真语句

10.6.4 force、release语句

【例 11-23】

```
module testforce;  // force 语句测试示例
reg a, b, c, d;
wire e;
and and1 (e, a, b, c);
initial begin  // 监控 d、e 的变化
$monitor("%d d=%b,e=%b", $stime, d, e);
assign d = a & b & c;    // 连续赋值 d
a = 1;  b = 0;   c = 1;
#10;                     // 延迟 10 个时间单位
force d = (a | b | c);   // 强制赋值 d
force e = (a | b | c);   // 强制赋值 e
#10 $stop;               // 暂停仿真
release d;               // 释放 d
release e;               // 释放 e
#10 $stop;               //暂停仿真
end
endmodule
```

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


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

10.6 其他仿真语句

10.6.5 deassign语句

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

10.7 仿真激励信号的产生

【例 10-24】// 4 位加法器

```
module adder4(
   input[3:0] a,        input[3:0] b,
   output reg[3:0] c,   output reg co
                );
always @*                          // 组合逻辑
  {co,c} <= a + b;                 // co 为进位，c 为和
endmodule
```

10.7 仿真激励信号的产生

【例 10-25】

```
`timescale 10ns/1ns // 时间设置
module signal_gen(output reg [3:0] sig1,output reg [3:0] sig2);
initial begin   // 只执行一次
  sig1 <= 4'd10; // 依序列出输入信号变化
  sig2 <= 4'd3;
  #10 sig2 <=4'd4;
  #10 sig1 <=4'd11;
  #10 sig2 <=4'd6;
  #10 sig1 <=4'd8;
  #10 $stop;
end
endmodule
```

10.7 仿真激励信号的产生

【例 10-26】

```
module test_adder4(); // 用于仿真的顶层文件
wire [3:0] a,b,c;
wire co;
adder4 U1(.a(a),.b(b),.c(c),.co(co));   // 例化被测元件 DUT
signal_gen TU1(.sig1(a),.sig2(b));      // 例化激励发生模块
endmodule
```

10.7 仿真激励信号的产生

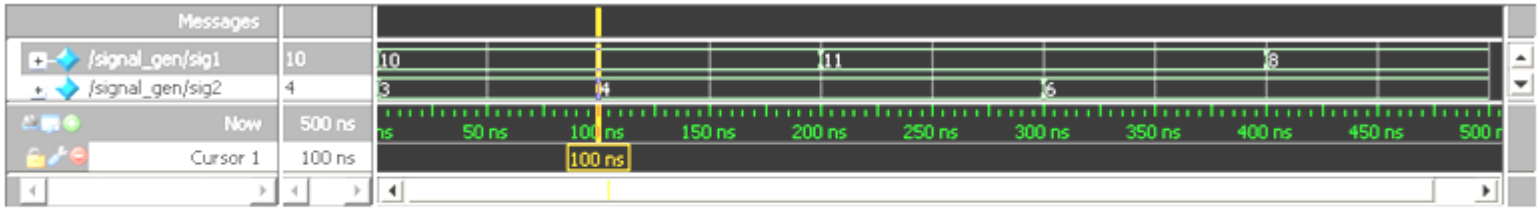


图 10-24 signal_gen 的仿真输出波形

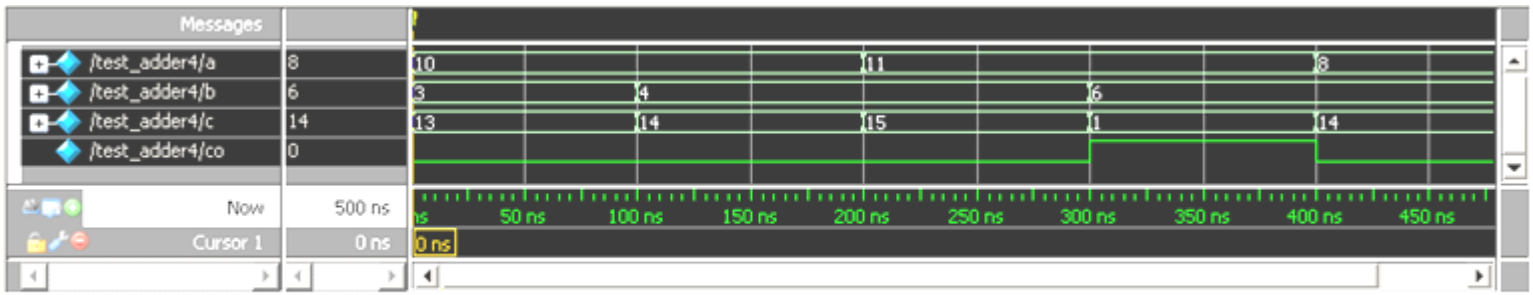


图 10-25 test_adder4 仿真波形图

10.7 仿真激励信号的产生

```
force <信号名> <值> [<时间>][, <值> <时间> …] [-repeat <周期>]
```

```
force a 0                    （强制信号的当前值为 0）
force b 0 0, 1 10            （强制信号 b 在时刻 0 的值为 0，在时刻 10 的值为 1）
force clk 0 0, 1 15 -repeat 20  （clk 为周期信号，周期为 20）
```

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

10.8 Verilog测试基准

【例 10-27】

```
`timescale 10ns/1ns
module testcnt4( output [3:0] q, output cout);
reg clk; reg rst,load; reg[3:0] d;
always #10 clk = ~clk;
initial begin
 rst = 1'b1;     clk = 1'b0;     // 初始化
 #15 rst = 1'b0; // 复位信号发生
 d=4'h2;         // d 赋值 2
 #10 load = 1'b1;    // 生成 load 信号
 #10 load = 1'b0;
end
// 例化被测元件 DUT
cnt4 u1(.clk(clk),.rst(rst),.d(d),.load(load),.q(q),.cout(cout));
 endmodule
```

10.8 Verilog测试基准

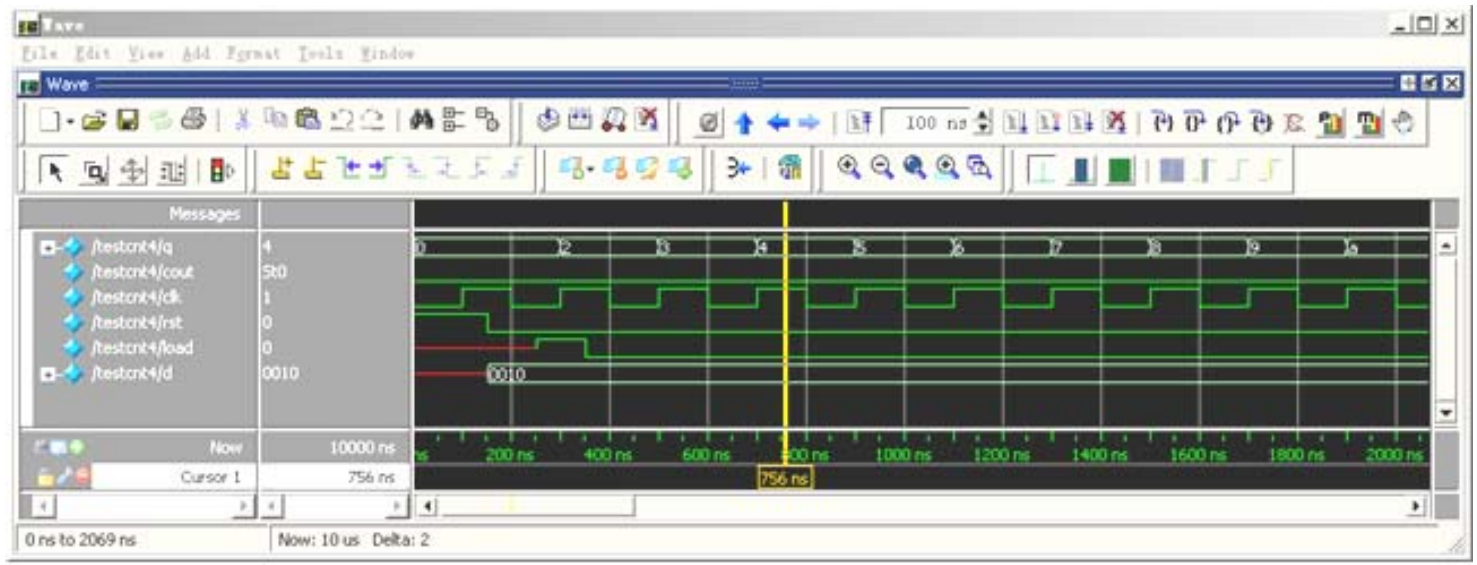


图 10-26　例 10-27 仿真结果

10.9 Verilog数字系统仿真

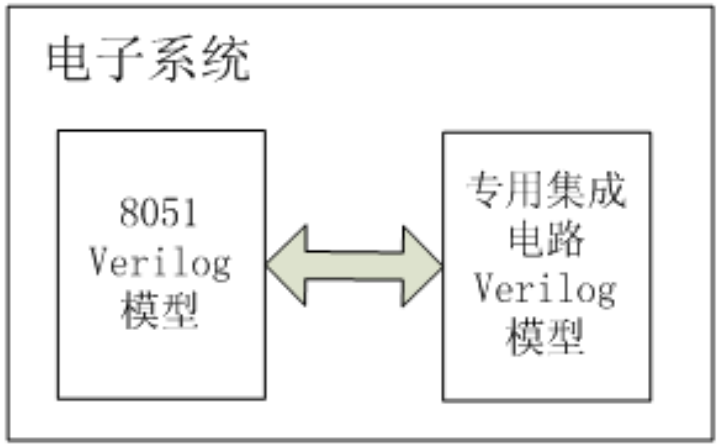


图 10-27　Verilog 系统仿真模型示意图

实 验

【例 10-28】

```
module acc16(
  input [15:0] a,  input rst,   input clk,
  output reg [15:0] c );
   always @(posedge clk,negedge rst)
    if(!rst) c = 0;
  else c=c+a;
endmodule
```