

## 编写 testbench 及仿真

黑金动力社区 2017-12-21

### 1 文档简介

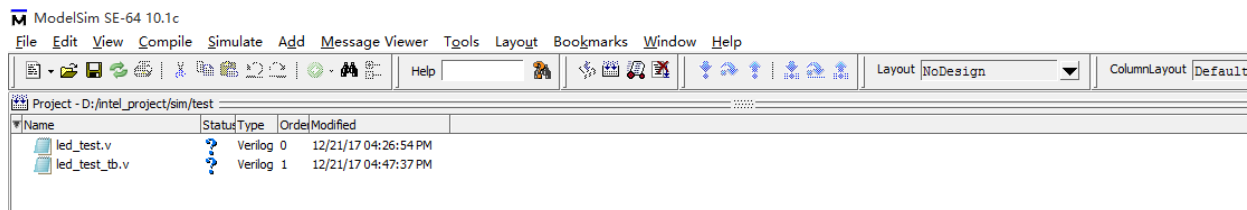
本章节主要了解如何编写激励文件，常用的系统任务和系统函数，并实现在 Modelsim 软件里仿真。

### 2 新建 Modelsim 工程

- (1) 新建文件夹，利用 01\_led\_test 实验文件进行仿真，将 verilog 文件及激励文件拷贝到新建文件夹中。注意路径不要出现中文或空格等非法字符。

本地磁盘 (D:) > intel_project > sim			
名称	修改日期	类型	大小
led_test.v	2017/12/21 16:26	V 文件	4 KB
led_test_tb.v	2017/12/21 16:47	V 文件	1 KB

- (2) 打开 Modelsim 软件，新建项目工程指定到所建的文件夹，并将两个文件导入到项目中。



- (3) 后面讲到的例子都放在了 demo/testbench 文件夹下，如果遇到问题可以点击“点我仿真.bat”自动运行仿真测试。但建议按照下面的步骤学习。

led_test.v	2018/3/28 17:48	V 文件	4 KB
led_test_tb.v	2018/3/28 17:51	V 文件	1 KB
start.do	2018/3/28 17:51	DO 文件	1 KB
点我仿真.bat	2017/8/16 15:09	Windows 批处理...	1 KB

### 3 分析激励文件

打开激励文件 led\_test\_tb.v 可以看到，激励文件主要由模块定义、信号初始化、执行模块、例化等几部分组成。下面对 testbench 逐条进行了解释：

```
`timescale 1ns/1ns
```

//timescale是verilog中的一种时间尺度预编译指令，用来定义  
//仿真时的时间单位和时间精度，/左边是时间单位，右边是时间精  
//度，时间单位是用于编写激励文件，时间精度是显示时的刻度，比  
//如#100也就是100ns。时间精度不能大于时间单位，  
//比如`timescale 1 ns/1 ps是正确的，  
//而`timescale 1 ps/1 ns是错误的。

```
module led_test_tb();
```

//端口定义，激励文件不需要声明输入输出端口  
//对于例化文件的输入端口，需要定义寄存器类型的信号

```
reg clk;
```

```
reg rst_n;
```

//对于例化文件的输出端口，需要定义wire类型的信号  
//每个initial或always模块之间是并行执行的  
//需要对输入信号进行初始化处理

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

```
initial
```

```
begin
```

```
clk = 1'b0;
```

```
rst_n = 1'b0;
```

```
#100 rst_n = 1'b1;
```

//# 代表延时，如时间单位是1ns，那么#100代表延迟100ns

```
end
```

```
always#10 clk = ~clk;
```

//根据时钟频率，如50MHz，每个周期是20ns，那么每10ns翻转  
//产生出时钟信号

```
led_test dut
```

```
(
```

```
    .clk      (clk),      // 开发板上输入时钟：50Mhz
```

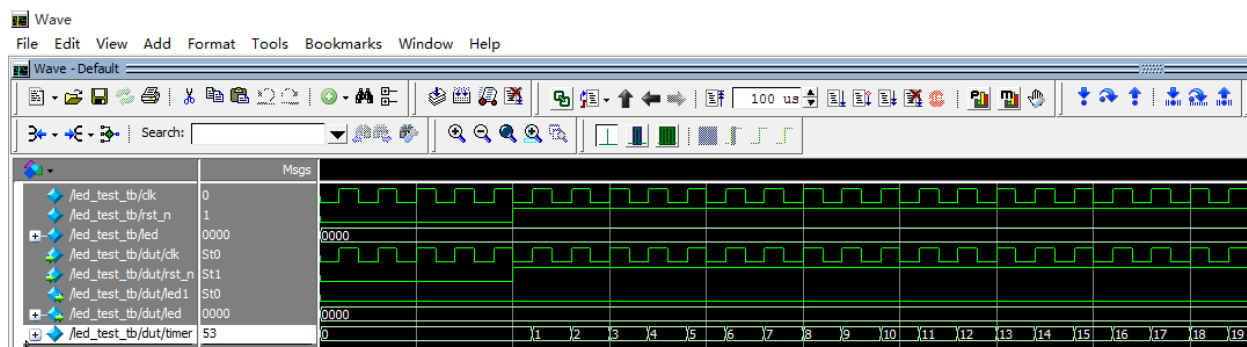
```
    .rst_n    (rst_n),    // 开发板上输入复位按键
```

```
    .led      (led)       // 输出LED灯,用于控制开发板上四个LED(LED0~LED3)
```

```
);
```

```
endmodule
```

经过仿真，即可查看波形界面：



## 4 系统任务和系统函数

下面介绍几个激励文件中常用的系统任务和系统函数。

### 4.1 显示任务

#### 4.1.1 \$display

\$display 系统任务用于打印信息，类似于 C 语言的打印。未指定显示格式时，默认显示的格式是十进制，另外还有\$displayb, \$displayo, \$displayh 显示格式分别是二进制、八进制、十六进制。\$display 会在每次显示后自动换行。格式如下：

例：\$display("%b+%b=%b",a,b,c);

常用的显示格式如下表：

%b 或%B	二进制
%o 或%O	八进制
%d 或%D	十进制
%h 或%H	十六进制
%e 或%E	实数
%c 或%C	字符
%s 或%S	字符串
%t 或%T	时间
\n	换行
\t	制表符
\\	反斜杠\
\"	引号"
\%%	百分号%

举例说明，打开新建的 Modelsim 工程，打开 testbench 文件，添加两行代码：

```
$display("hello alinx");
```

```
$display("rst_n = %d", rst_n);
```

```

`timescale 1ns/1ns
module led_test_tb;
reg clk;
reg rst_n;
wire[3:0] led;

initial
begin
    $display("hello alinx") ;
    clk = 1'b0;
    rst_n = 1'b0;
    #100 rst_n = 1'b1;
    $display("rst_n = %d", rst_n) ;
end

always#10 clk = ~clk;//50Mhz
led_test dut
(
    .clk          (clk),
    .rst_n        (rst_n),
    .led          (led)
);
endmodule

```

编译，仿真、运行，可以在 Modelsim 主界面底部信息栏里看到打印信息。

```

Transcript
VSI6> vsim -voptargs=+acc work.led_test_tb
# vsim -voptargs=+acc work.led_test_tb
# ** Note: (vsim-3813) Design is being optimized due to module recompilation...
#
# ** Warning: D:/intel_project/sim/led_test_tb.v(22): (vopt-2685) [TFMPC] - Too few port connections for 'dut'. Expected 4, found 3.
#
# ** Warning: D:/intel_project/sim/led_test_tb.v(22): (vopt-2718) [TFMPC] - Missing connection for port 'led1'.
#
# Loading work.led_test_tb(fast)
# Loading work.led_test(fast)
VSI7> run 500ns
# hello alinx
# rst_n = 1
VSI8>

```

Project: test | Now: 500 ns Delta: 2 | sim:/led\_test\_tb

#### 4.1.2 \$write

\$write 与 \$display 类似，唯一的不同之处是 \$write 不像 \$display 一样自动换行。如将上面的两行代码改为 \$write (“hello alinx”); \$write (“rst\_n = %d”, rst\_n);

```

`timescale 1ns/1ns
module led_test_tb;
reg clk;
reg rst_n;
wire[3:0] led;

initial
begin
    $write("hello alinx") ;
    clk = 1'b0;
    rst_n = 1'b0;
    #100 rst_n = 1'b1;
    $write("rst_n = %d", rst_n) ;
end

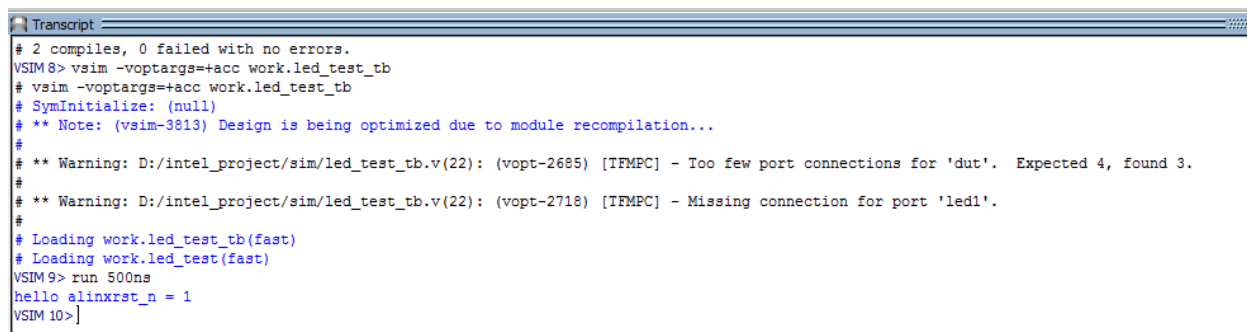
```

```

always#10 clk = ~clk;//50Mhz
led_test dut
(
    .clk            (clk),
    .rst_n          (rst_n),
    .led            (led)
);
endmodule

```

打印信息显示如下：



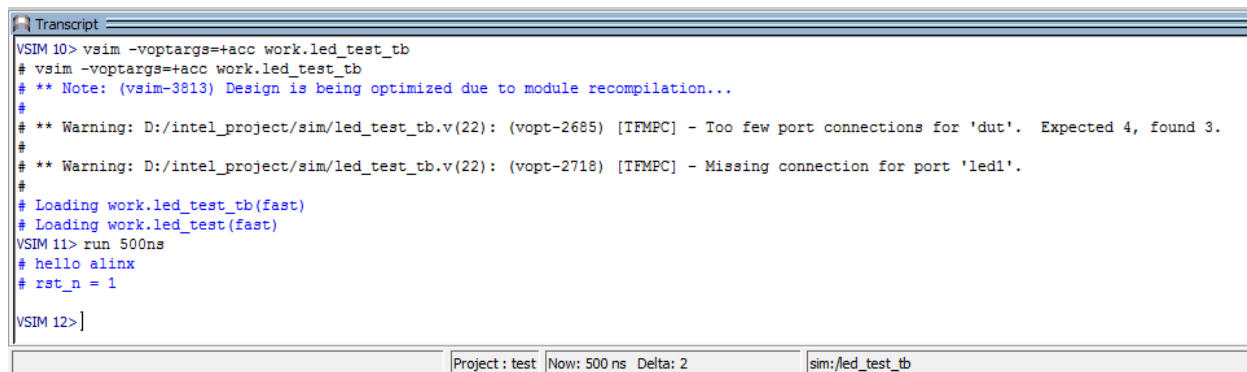
```

Transcript
# 2 compiles, 0 failed with no errors.
VSIM8> vsim -voptargs=+acc work.led_test_tb
# vsim -voptargs=+acc work.led_test_tb
# SymInitialize: (null)
# ** Note: (vsim-3813) Design is being optimized due to module recompilation...
#
# ** Warning: D:/intel_project/sim/led_test_tb.v(22): (vopt-2685) [TFMPC] - Too few port connections for 'dut'. Expected 4, found 3.
#
# ** Warning: D:/intel_project/sim/led_test_tb.v(22): (vopt-2718) [TFMPC] - Missing connection for port 'led1'.
#
# Loading work.led_test_tb(fast)
# Loading work.led_test_tb(fast)
VSIM9> run 500ns
hello alinxrst_n = 1
VSIM10>

```

可以看到两条打印信息是连在一起的，再加上换行符后即可。`$write("hello alinx\n");`

`$write("rst_n = %d\n", rst_n);`如下图所示：



```

Transcript
VSIM10> vsim -voptargs=+acc work.led_test_tb
# vsim -voptargs=+acc work.led_test_tb
# ** Note: (vsim-3813) Design is being optimized due to module recompilation...
#
# ** Warning: D:/intel_project/sim/led_test_tb.v(22): (vopt-2685) [TFMPC] - Too few port connections for 'dut'. Expected 4, found 3.
#
# ** Warning: D:/intel_project/sim/led_test_tb.v(22): (vopt-2718) [TFMPC] - Missing connection for port 'led1'.
#
# Loading work.led_test_tb(fast)
# Loading work.led_test_tb(fast)
VSIM11> run 500ns
# hello alinx
# rst_n = 1
VSIM12>

```

Project: test    Now: 500 ns    Delta: 2    sim:/led\_test\_tb

### 4.1.3 \$monitor

`$monitor` 用于持续监测指定变量，只要变量发生了变化，即会触发`$monitor`，显示对应的语句。格式如下：

例：`$monitor("%b+%b=%b",a,b,c);`

在代码中添加以下代码：

Initial

`$monitor("clk = %b", clk);`

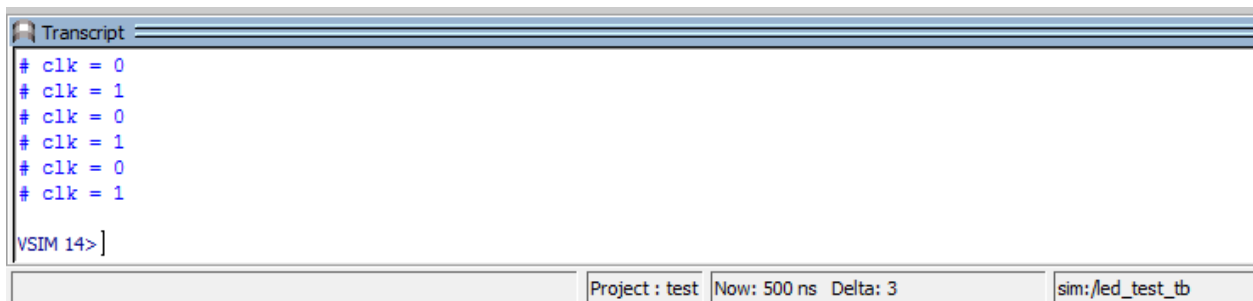
```
`timescale 1ns/1ns
module led_test_tb;
reg clk;
reg rst_n;
wire[3:0] led;

initial
begin
    $write("hello alinx\n") ;
    clk = 1'b0;
    rst_n = 1'b0;
    #100 rst_n = 1'b1;
    $write("rst_n = %d\n", rst_n) ;
end

initial
$monitor("clk = %b", clk) ;

always#10 clk = ~clk;//50Mhz
led_test dut
(
    .clk          (clk),
    .rst_n        (rst_n),
    .led          (led)
);
endmodule
```

编译、仿真执行打印信息如下，可以看到只要 clk 有变化就会打印信息。



还有\$monitoron 和\$monitoroff 用于打开关闭 monitor 功能。

#### 4.1.4 \$strobe

\$strobe 的语法跟显示任务\$display 一样，也是把信息打印出来，区别在于\$strobe 是在被调用时所有的活动都完成了，才打印出来，\$display 是在仿真器看到时就执行。同样也有\$strobeb, \$strobo, \$strobeh 代表二进制，八进制、十六进制。

如下面代码所示，定义四个变量 a,b,c,d，\$random 是产生随机数的函数，后面会讲。在 always 语句里将\$strobe 放到最前面，\$display 放到最后面。可将以下代码拷贝到 modelsim 中。

```
`timescale 1ns/1ns
module led_test_tb;
reg clk;
reg rst_n;
wire[3:0] led;
reg [7:0] a ;
reg [7:0] b ;
reg [7:0] c ;
reg [7:0] d ;

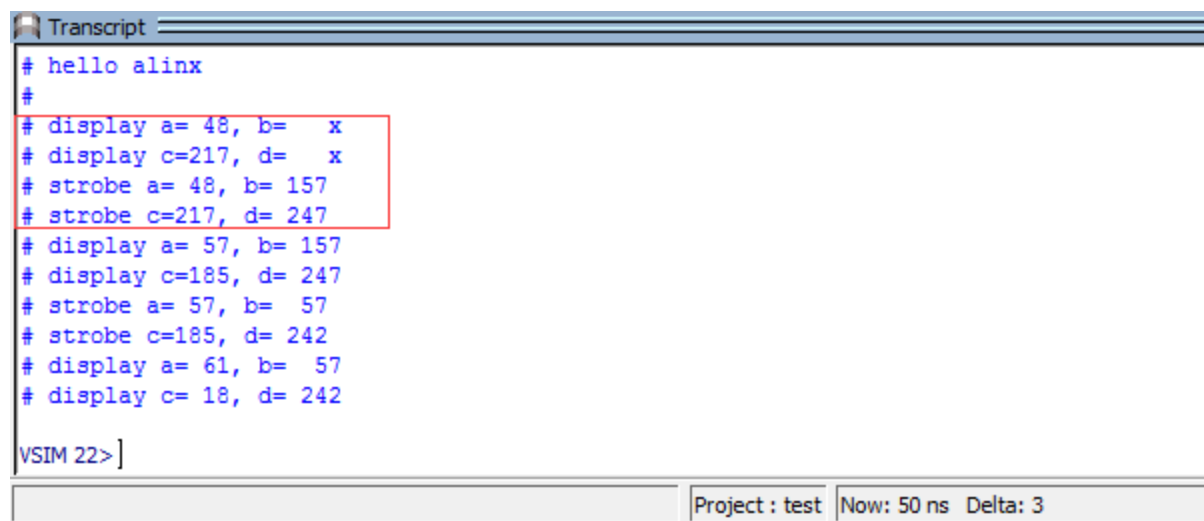
initial
begin
    $write("hello alinx\n") ;
    clk = 1'b0;
    rst_n = 1'b0;
    #100 rst_n = 1'b1;
    $write("rst_n = %d\n", rst_n) ;
end

//initial
//$monitor("clk = %b", clk) ;
always@(posedge clk)
begin
    $strobe("strobe a=%d, b= %d", a,b) ;
    a = $random%100 ;
    $display("display a=%d, b= %d", a,b) ;
    b = $random%100 ;
end

always@(posedge clk)
begin
    $strobe("strobe c=%d, d= %d", c,d) ;
    c = $random%100 ;
    $display("display c=%d, d= %d", c,d) ;
    d = $random%100 ;
end

always#10 clk = ~clk;//50Mhz
led_test dut
(
    .clk            (clk),
    .rst_n          (rst_n),
    .led            (led)
);
endmodule
```

观察打印出的信息，结果\$display 在第一个周期只显示了 a 和 c 的值，而没有 b 和 d 的值，\$display 是在出现时就执行。而\$strobe 是等到所有工作结束后才打印信息。

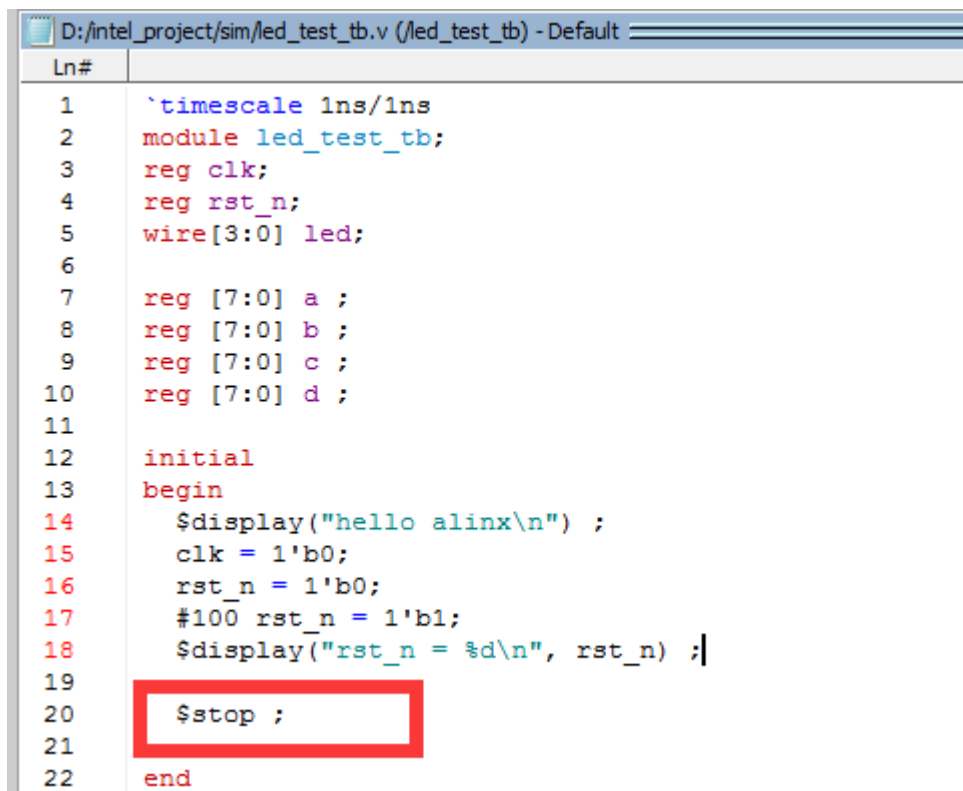


```
Transcript
# hello alinx
#
# display a= 48, b= x
# display c=217, d= x
# strobe a= 48, b= 157
# strobe c=217, d= 247
# display a= 57, b= 157
# display c=185, d= 247
# strobe a= 57, b= 57
# strobe c=185, d= 242
# display a= 61, b= 57
# display c= 18, d= 242
VSIM 22> ]
Project : test Now: 50 ns Delta: 3
```

## 4.2 模拟控制任务

### 4.2.1 \$stop

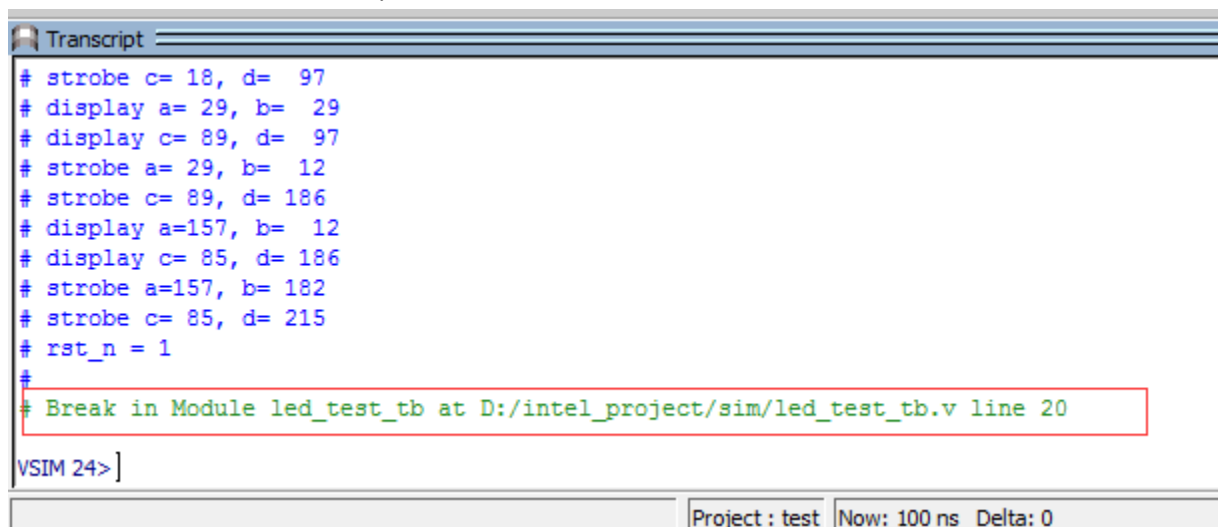
\$stop 任务是暂停仿真，返回软件操作主窗口，由使用者决定下一步操作。如下图所示。



```
D:/intel_project/sim/led_test_tb.v (/led_test_tb) - Default
Ln#
1  `timescale 1ns/1ns
2  module led_test_tb;
3  reg clk;
4  reg rst_n;
5  wire[3:0] led;
6
7  reg [7:0] a ;
8  reg [7:0] b ;
9  reg [7:0] c ;
10 reg [7:0] d ;
11
12 initial
13 begin
14     $display("hello alinx\n") ;
15     clk = 1'b0;
16     rst_n = 1'b0;
17     #100 rst_n = 1'b1;
18     $display("rst_n = %d\n", rst_n) ;|
19
20     $stop ;
21
22 end
```



编译、执行仿真。设定的\$stop 是在#100 个单位之后，在窗口输入 run 200ns 后，跳出信息 break，

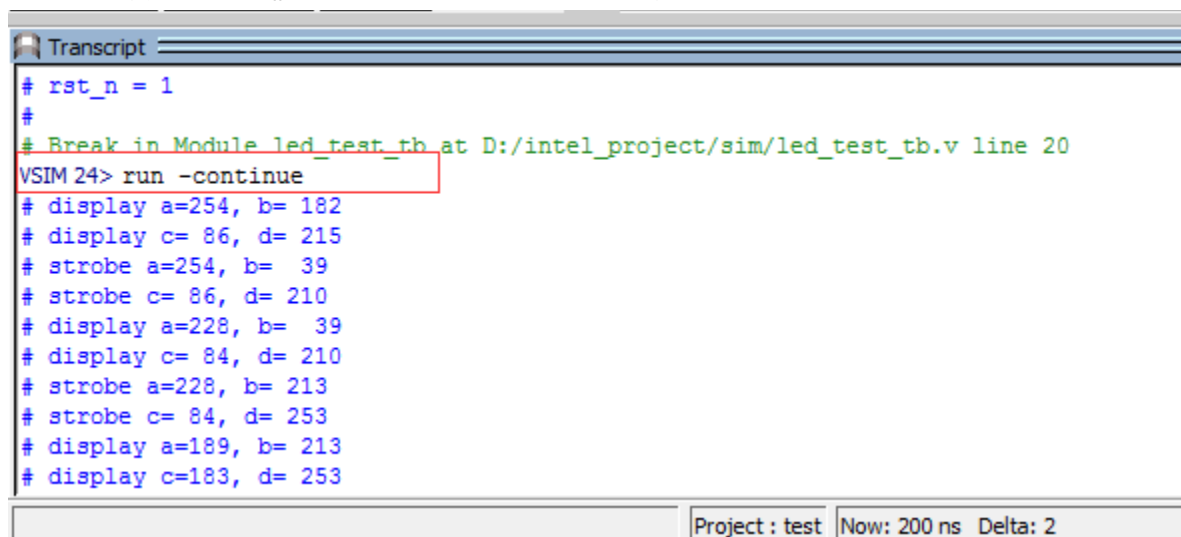


The screenshot shows the VSIM Transcript window with the following text:

```
# strobe c= 18, d= 97
# display a= 29, b= 29
# display c= 89, d= 97
# strobe a= 29, b= 12
# strobe c= 89, d= 186
# display a=157, b= 12
# display c= 85, d= 186
# strobe a=157, b= 182
# strobe c= 85, d= 215
# rst_n = 1
#
# Break in Module led_test_tb at D:/intel_project/sim/led_test_tb.v line 20
VSIM 24> ]
```

At the bottom of the window, the status bar shows: Project : test Now: 100 ns Delta: 0

如果想继续运行，可输入 run -continue，直到前面设定的 200ns。



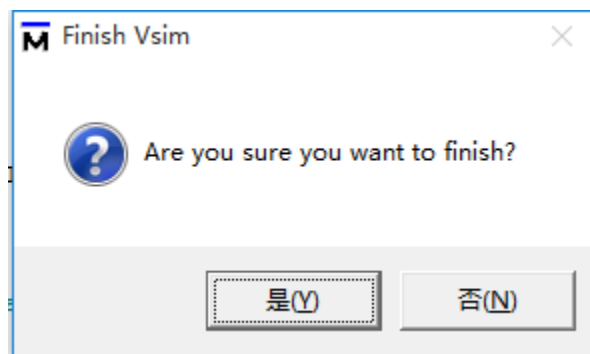
The screenshot shows the VSIM Transcript window with the following text:

```
# rst_n = 1
#
# Break in Module led_test_tb at D:/intel_project/sim/led_test_tb.v line 20
VSIM 24> run -continue
# display a=254, b= 182
# display c= 86, d= 215
# strobe a=254, b= 39
# strobe c= 86, d= 210
# display a=228, b= 39
# display c= 84, d= 210
# strobe a=228, b= 213
# strobe c= 84, d= 253
# display a=189, b= 213
# display c=183, d= 253
```

At the bottom of the window, the status bar shows: Project : test Now: 200 ns Delta: 2

## 4.2.2 \$finish

\$finish 的功能是仿真停止，将\$stop 改成\$finish，仿真运行 run 200ns，会跳出窗口，是否终止仿真，可根据需要选择，选择“是”之后软件直接关闭。



## 4.3 模拟时间函数

\$time 功能为读取当前仿真时间。有以下几种

\$time 返回一个 64 位整数时间值

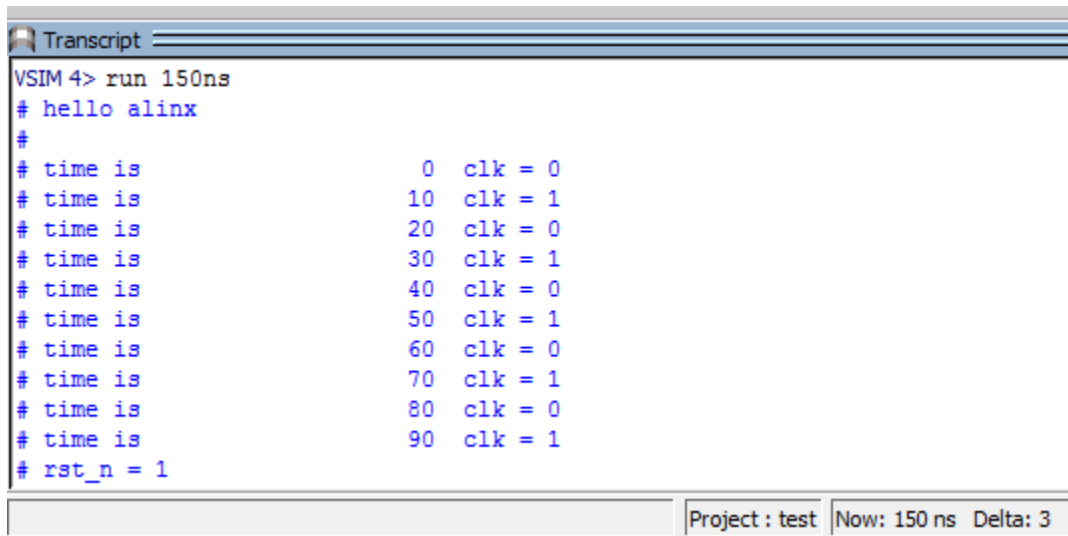
\$stime 返回一个 32 位整数时间值

\$realtime 返回一个实数时间值

举例说明，如下图，将其他功能注释，用\$monitor 功能显示 clk 翻转的时间。

```
$monitor("time is %d\tclk= %b", $time, clk) ;  
D:/intel_project/sim/led_test_tb.v (/led_test_tb) - Default  
Ln#  
2 module led_test_tb;  
3 reg clk;  
4 reg rst_n;  
5 wire[3:0] led;  
6  
7 reg [7:0] a ;  
8 reg [7:0] b ;  
9 reg [7:0] c ;  
10 reg [7:0] d ;  
11  
12 initial  
13 begin  
14 $display("hello alinx\n") ;  
15 clk = 1'b0;  
16 rst_n = 1'b0;  
17 #100 rst_n = 1'b1;  
18 $display("rst_n = %d\n", rst_n) ;  
19 // $finish ;  
20 end  
21 initial  
22 $monitor("time is %d\tclk = %b", $time, clk) ;  
23 /*  
24 always@(posedge clk)  
25 begin  
26 $strobe("strobe a=%d, b= %d", a,b) ;  
27 a = $random%100 ;  
28 $display("display a=%d, b= %d", a,b) ;  
29 b = $random%100 ;  
30 end
```

可以从信息栏里看到时间刻度以十进制显示出来。



```
Transcript
VSIM 4> run 150ns
# hello alinx
#
# time is          0  clk = 0
# time is         10  clk = 1
# time is         20  clk = 0
# time is         30  clk = 1
# time is         40  clk = 0
# time is         50  clk = 1
# time is         60  clk = 0
# time is         70  clk = 1
# time is         80  clk = 0
# time is         90  clk = 1
# rst_n = 1

Project : test  Now: 150 ns  Delta: 3
```

## 4.4 \$random

\$random 是 verilog 中产生随机数的系统函数，在调用时返回一个 32 位的随机数，是带符号的整形数。有几种用法：

- (1) \$random 和 \$random() 意义一样，都是产生随机数
- (2) \$random%100 在 -99 到 99 之间产生随机数
- (3) {\$random}%100 采用位拼接符，在 0 到 100 之间产生随机数
- (4) 如 seed = 10, \$random(seed) 根据 seed 值产生随机数，而后 seed 值也会更新

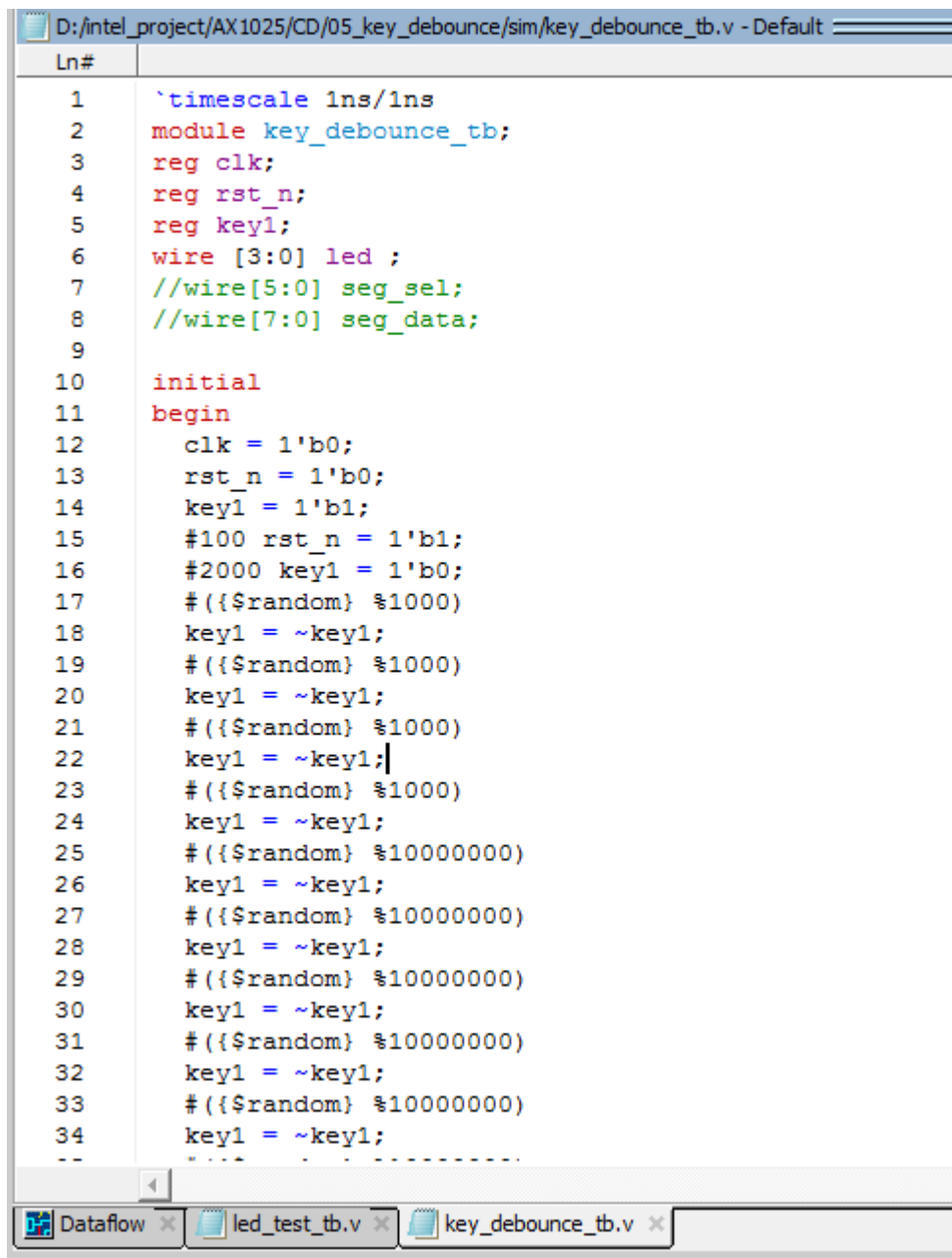
如前面讲到的用 \$random 产生 a,b,c,d 的值

```
always@(posedge clk)
begin
    $strobe("strobe a=%d, b= %d", a,b) ;
    a = $random%100 ;
    $display("display a=%d, b= %d", a,b) ;
    b = $random%100 ;
end

always@(posedge clk)
begin
    $strobe("strobe c=%d, d= %d", c,d) ;
    c = $random%100 ;
    $display("display c=%d, d= %d", c,d) ;
    d = $random%100 ;
end
```

除了用于赋值外，也可用于产生随机数延时，如按键消抖实验中的激励文件。

# ( {random}%1000 ) 表示产生随机延迟时间。



```
D:/intel_project/AX1025/CD/05_key_debounce/sim/key_debounce_tb.v - Default
Ln#
1  `timescale 1ns/1ns
2  module key_debounce_tb;
3  reg clk;
4  reg rst_n;
5  reg key1;
6  wire [3:0] led ;
7  //wire[5:0] seg_sel;
8  //wire[7:0] seg_data;
9
10 initial
11 begin
12     clk = 1'b0;
13     rst_n = 1'b0;
14     key1 = 1'b1;
15     #100 rst_n = 1'b1;
16     #2000 key1 = 1'b0;
17     #({$random} %1000)
18     key1 = ~key1;
19     #({$random} %1000)
20     key1 = ~key1;
21     #({$random} %1000)
22     key1 = ~key1;
23     #({$random} %1000)
24     key1 = ~key1;
25     #({$random} %10000000)
26     key1 = ~key1;
27     #({$random} %10000000)
28     key1 = ~key1;
29     #({$random} %10000000)
30     key1 = ~key1;
31     #({$random} %10000000)
32     key1 = ~key1;
33     #({$random} %10000000)
34     key1 = ~key1;
--     .....

```

## 4.5 \$readmemb

\$readmemb 和 \$readmemh 可用于从文件中读入数据

格式如下

```
$readmemb("<数据文件名>",<存储器名>);
```

```
$readmemb("<数据文件名>",<存储器名>,<起始地址>);
```

```
$readmemb("<数据文件名>",<存储器名>,<起始地址>,<结束地址>);
```

```
$readmemh("<数据文件名>",<存储器名>);
```

```
$readmemh("<数据文件名>",<存储器名>,<起始地址>);
```

```
$readmemh("<数据文件名>",<存储器名>,<起始地址>,<结束地址>);
```

例：reg [7:0] mem[255:0] //定义 256 个 8 位数据的 ROM

```
$readmemb("filename.txt",mem); //将 filename.txt 中的数据读入到 mem 中
```

以下面的程序举例，将 top.mif 的信息读入到 mem 里，并显示 mem 的内容，注意下面的写法中要将 top.mif 放在同一文件夹，如果 top.mif 放在其他文件夹，可通过路径显示，如

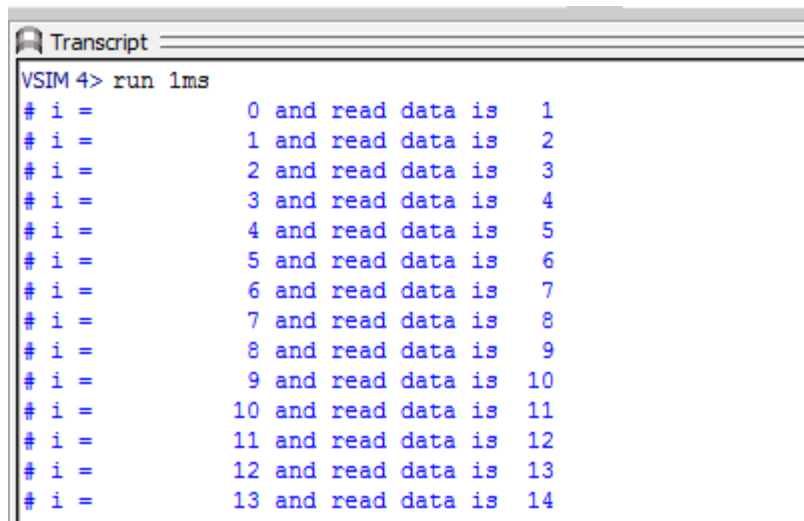
"D:/intel\_project/readmem/top.mif"

```
`timescale 1ns/1ns
module readmem ;
integer i ;
reg [7:0] mem[255:0] ; //定义深度为256，宽度为8的存储器mem

initial
begin
    $readmemb("top.mif", mem) ; //读取top.mif到mem存储器
    i = 0 ;
    repeat(256) //重复256次
    begin
        $display("i = %d and read data is %d", i,mem[i]) ; //显示mem的数据
        i = i+1 ;
        #20 ;
    end
end

endmodule
```

经过仿真，打印信息如下



```
Transcript
VSIM 4> run 1ms
# i =      0 and read data is  1
# i =      1 and read data is  2
# i =      2 and read data is  3
# i =      3 and read data is  4
# i =      4 and read data is  5
# i =      5 and read data is  6
# i =      6 and read data is  7
# i =      7 and read data is  8
# i =      8 and read data is  9
# i =      9 and read data is 10
# i =     10 and read data is 11
# i =     11 and read data is 12
# i =     12 and read data is 13
# i =     13 and read data is 14
```

## 5 总结

本节介绍了激励文件的结构以及常用的系统任务和系统函数，希望通过这节的讲解能够使大家学会如何编写激励文件，并能够在实践中多练习。