测试平台描述

- **编写Testbench目的**

  对用硬件描述语言设计的电路进行仿真验证，测试设计电路的逻辑关系是否正确、验证电路功能和部分性能是否与预期相符。

- **编写Testbench进行测试的过程**
  - 产生模拟激励（波形）
  - 将产生的激励加入到被测试模块并观察其输出响应
  - 将输出响应与期望进行比较，从而判断设计的正确性

● 基本的Testbench结构

虽然测试模块也是通过Verilog编程实现，但是它与一般功能模块的编写规则却不大相同。基本的Testbench结构如下：

```
`timescale  1ns/1ps  //定义时间单位与时间精度，不可缺少
module Test_bench;  //通常Testbench没有输入与输出端口
    信号或变量定义声明
    使用initial或always语句来产生激励波形
    实例化设计模块
    监控和比较输出响应
endmodule
```

● 基本的Testbench结构

例：

```
1   `timescale 1ns/1ns
2
3   module logic_gates_tb;
4
5   reg i_a;
6   reg i_b;
7   wire o_and;
8   wire o_or;
9   wire o_not;
10
11  initial
12  begin
13    i_a = 0;
14    #40 i_a = 1;
15    #40 i_a = 0;
16    #40 i_a = 1;
17    #40 i_a = 0;
18  end
```

```
20  initial
21  begin
22    i_b = 0;
23    #40 i_b = 0;
24    #40 i_b = 1;
25    #40 i_b = 1;
26    #40 i_b = 0;
27  end
28
29  logic_gates logic_gates_inst(
30    .iA(i_a),
31    .iB(i_b),
32    .oAnd(o_and),
33    .oOr(o_or),
34    .oNot(o_not)
35  );
36
37  endmodule
```

● 基本的Testbench结构

与可综合Verilog代码所不同的是，testbench Verilog是在计算机主机上的仿真器中执行的。testbench Verilog的许多构造与C语言相似，我们可在代码中包括复杂的语言结构和顺序语句的算法。

1 always块和initial块

```
always
begin
   clk=1;
   #20;
   clk=0;
   #20;
end
```

```
initial
begin
  进程语句;
end
```

always块可用来模拟抽象的电路，包括：用以指定与不同结构之间的传播延迟等同的时序结构；或等待指定事件的时序结构。敏感列表有时可忽略。

initail块仅在仿真之初执行，常用于设置变量的初始值。

● 基本的Testbench结构

1 always块和initial块

- 当always块用来描述组合逻辑时，应当使用阻塞赋值。例如状态机中状态转移的always块。
- 对于时序逻辑的描述和建模，应当使用非阻塞赋值，其大多数always块都是用非阻塞赋值。
- 同一个always块不要混合使用阻塞和非阻塞赋值，尤其在可综合电路的同一个always块。

● 基本的Testbench结构

2 进程语句

进程语句应用于initial块、always块、function和task之中。testbench最常用的进程语句为：

- 阻塞赋值
- 非阻塞赋值
- if表达式
- case表达式
- 循环表达式(for、while、repeat和forever)

```
integer i;
. . .
repeat(16)
begin
  [procedural_statements;]
end
```

```
initial
begin
  clk=1'b0;
  forever
    #10 clk=~clk;
end
```

● 基本的Testbench结构

3 时序控制

在testbench中，必须指定不同信号有效和无效或等待某事件或条件的时间。有三种时序控制结构：

- 时延控制：#[delay_time]
- 事件控制：@([event], [event], …]
- 等待语句：wait([boolean_expression])

```
1  localparam delta=1;
2  . . .
3  @(posedge clk); // wait for the rising edge of clk
4  #delta;         // wait for delta to avoid hold-time violation
5  en=1'b1;        // assert en to 1
6  @(posedge clk); // wait for the next rising edge of clk
7  #delta;         // wait for delta to avoid hold-time violation
8  en=1'b0;        // assert en to 0
```

● 基本的Testbench结构

3 时序控制

此外还有一个编译器指令，`timescale，也与时序规范有关。

`timescale [time_unit] / [time_precision]

time_unit指定计时和延时的测量单位，time_precision则是指定仿真器的精度。

```
1 `timescale 10ns/1ns
2 #5 y = a & b;
```

✓ 精度越小，仿真的准确性越高，但是会减慢仿真的速度。

✓ time_unit和time_precision的数字部分可以为1、10和100，时间单元可以是s（秒）、ms（毫秒）、us（微秒）、ns（纳秒）和ps（皮秒）。

● 基本的Testbench结构

4 系统控制函数和任务

Verilog有一组预定义的系统函数，以$打头，执行与系统相关的操作，如仿真控制、文件读取等。

- 输出控制：$display,$write,$monitor
- 模拟时标：$time
- 进程控制：$finish,$stop
- 文件读写：$readmem
- 其它：$random $signed $unsigned
  $fopen $fclose $fdisplay $fwrite $fmonitor……

● 产生激励的一些描述方式

1 产生时钟的几种方式

- 使用**forever**方式产生占空比为**50%**的时钟

```
1    initial
2    begin
3        clk = 0;
4        #50;
5        forever
6            #20 clk = ~clk;
7    end
```

注意：一定要给时钟赋初始值。信号的缺省值为**Z**，如果不赋初值，则反相后还是**Z**。时钟就一直处于高阻状态**Z**。

● 产生激励的一些描述方式

1 产生时钟的几种方式

- 使用**always**方式

```
1    initial    clk = 0;
2    always     #20 clk = ~clk;
```

- 使用**repeat**产生确定数目的时钟脉冲

```
1    initial
2    begin
3        clk = 0;
4        repeat(6)
5            #20 clk = ~clk;
6    end
```

● 产生激励的一些描述方式

2 产生复位信号的几种方式

- 异步复位

```
1    initial
2    begin
3        Rst = 1;
4        #100;
5        Rst = 0;
6        #500;
7        Rst = 1;
8    end
```

- 同步复位

```
1    initial
2    begin
3        Rst = 1;
4        @(negedge clk);
5        Rst = 0;
6        #30;
7        @(negedge clk);
8        Rst = 1;
     end
```