沈阳工业大学

SHENYANG UNIVERSITY OF TECHNOLOGY

数字电子技术

沈阳工业大学
电子技术教研室

26

知识点26　Verilog HDL 行为建模

沈 阳 工 业 大 学

SHENYANG UNIVERSITY OF TECHNOLOGY

1. initial语句

initial语句是初始化语句，在仿真模拟开始时执行，即在0时刻开始执行，且只执行一次。initial语句一般用来设置某些对象或变量的初始状态，也可以用来产生特定的输出波形。

initial语句的语法格式如下：

```
initial
[timing_control] procedural_statement
```

initial语句实例如下所示：

```
reg Q;
...
initial
Q = 1'b0;
```

2. always语句

与initial语句不同，always语句重复执行，其语法格式如下：

always [tining_control] procedural_statement

过程语句procedural_statement和时延控制tining_control的描述方式与initial语句相同。例如：

```
always
#5 C1k= ~C1k;
```

上述语句将无条件的无限循环执行，产生一个无休止的方波信号，信号周期为10个时间单位，占空比50%。

当initail或always包含多个语句时，可以使用begin…end包含所有语句，所包含的语句都是顺序执行的。begin…end语句组合的作用就相当于c语言中的“{ }”，起到将多个语句合并为一个语句的作用。always后面的@(posedge CLK)起到控制后续语句在什么时候执行的目的，即当CLK发生一个上升沿事件时，后续的顺序语句才会执行。如果没有@(posedge CLK)条件，那么always引导的后续语句将无条件循环执行。

3. 事件控制

1.边沿触发事件控制

边沿触发事件控制的语法格式：@ event procedural_statement

例如：

@（posedge Clock）

Curr_State = Next_State;

3. 事件控制

1.边沿触发事件控制

也可使用如下形式；

@ event；

该语句触发一个等待，直到特定的事件发生。下面是确定时钟在initial语句中使用的一个例子。

```
initial
begin
@（posedge CLK）; //等待，直到在时钟CLK上发生正边沿
Y = D;
end
```

3. 事件控制

1）边沿触发事件控制

@ (Ctrl_A or Ctrl_B)

Dbus = ‘bz;

注意关键字or并不是表达式中的逻辑或，而是表示前后两个事件中的任何一个发生都可以。在Verilog HDL中，posedge和negedge分别是正沿和负沿的关键字，它们可能是表4-10中转换形式的一种。

边沿	转换形式				
正边沿	1 -> x	0 -> z	0 ->1	x -> 1	z -> 1
负边沿	1 -> x	1 -> z	1 -> 0	x -> 0	z -> 0

3. 事件控制

2） 电平敏感事件控制

在电平敏感事件控制中，进程语句或进程中的过程语句，一直延迟到条件变为真后才执行。电平敏感事件控制的语法格式如下所示：

```
wait（Condition）
procedural_Statement
```

在上面的表示形式中，过程语句是可选的。例如：

```
wait （Sum > 22）
Sum = 0;    //当Sum的值大于22时，才Sum清0
wait （DataReady）
Data =Bus;    //当DataReady为真时，将Bus赋给Data
wait （Preset）;    //延迟至Preset变为真时，执行后续语句
```

4. 语句块

1） 顺序语句块

顺序语句块的语法格式如下：

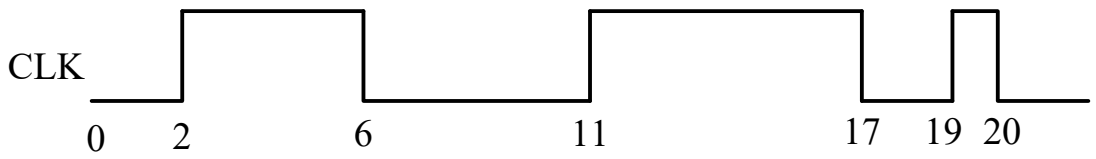
```
begin
[:block_id{declarations}]
procedural_statement(s)
end
```

4. 语句块

1） 顺序语句块

例如：

```
//产生波形：
Begin: CLKINT
#2 CLK = 1;
#4 CLK = 0;
#5 CLK = 1;
#6 CLK = 0;
#2 CLK = 1;
#1 CLK = 0;
end
```



4. 语句块

2） 并行语句块

并行语句块语法如下：

```
frok
[:block_id{declarations}]
procedural_statement(s);
join
```

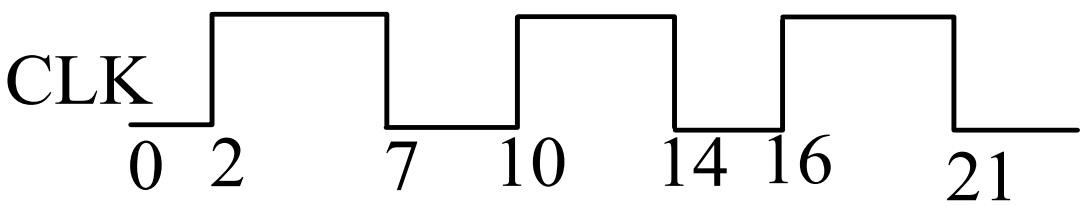
4. 语句块

2） 并行语句块

例如：

//生成波形：

```
frok
#2 CLK = 1;
#7 CLK = 0;
#10 CLK = 1;
#14 CLK = 0;
#16 CLK = 1;
#21 CLK = 0;
join
```



5. 过程性赋值

过程性赋值是在initial语句或always语句内的赋值，它只能对寄存器数据类型的变量赋值。表达式的右端可以是任何表达式，例如：

```
reg[3:0] En,A,B;
...
#5 En = A | B;
```

5. 过程性赋值

过程性赋值分两类：阻塞性过程赋值和非阻塞性过程赋值。

1） 阻塞性过程赋值

赋值操作符是“=”的过程赋值是阻塞性过程赋值。阻塞性过程赋值在其后所有语句执行前执行，或者说下一条语句必须等待前一条语句执行结束后才能执行。

如下所示：

```
reg T1,T2,T3;
always  @(A or B or Cin)
begin
        T1 = A & B;
        T2 = B & Cin;
        T3 = A & Cin;
end
```

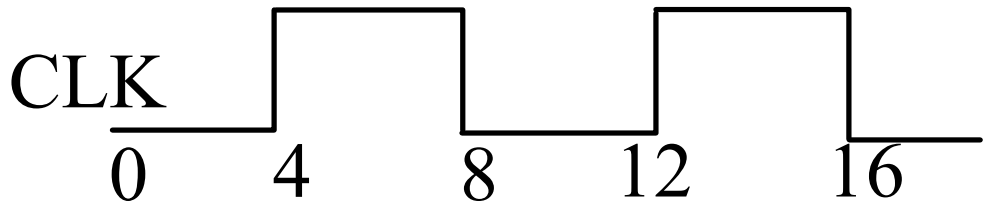
5. 过程性赋值

过程性赋值分两类：阻塞性过程赋值和非阻塞性过程赋值。

1） 阻塞性过程赋值

下例是使用语句内部延时控制的阻塞性过程赋值语句：

```
initial
begin
CLK = #4 1;
CLK = #4 0;
CLK = #4 1;
CLK = #4 0;
end
```



5. 过程性赋值

过程性赋值分两类：阻塞性过程赋值和非阻塞性过程赋值。

2） 非阻塞性过程赋值

在非阻塞性过程赋值中，使用赋值符号“<=”。前一条非阻塞性过程赋值语句是否执行完不影响后续语句的执行。当非阻塞性过程赋值被执行时，计算右端表达式，右端值被赋于左端目标，并继续执行下一条语句。例如：

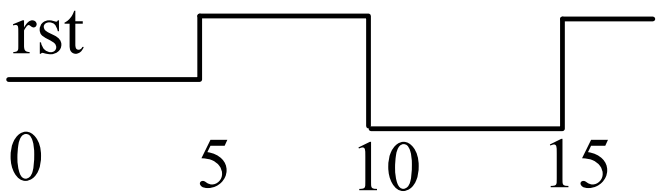
```
begin
        Y <= 1'b1;
        C <= Y;
end
```

5. 过程性赋值

过程性赋值分两类：阻塞性过程赋值和非阻塞性过程赋值。

2） 非阻塞性过程赋值

```
initial
        begin
                rst <= #10 0;
                rst <= #5 1;
                rst <= #15 1;
        end
```



5. 过程性赋值

过程性赋值分两类：阻塞性过程赋值和非阻塞性过程赋值。

3）连续赋值与过程赋值的比较

特征	连续赋值	过程赋值
使用位置	模块内直接使用	always语句和initial语句
执行特性	并行执行	与周围环境有关，可能顺序执行 也可能并行执行
驱动对象	线网	寄存器
赋值符号	“=”	“=” “<=”
前导词	assign	无

6. 常用过程语句

1） if语句

使用语法格式如下：

```
if（condittion_1）
        procedural_statement_1
{else if(condition_2)
        procedural_statement_2}
        {else
                procedural_statement_3}
```

6. 常用过程语句

1） if语句

```
if(s==1'b0)
        y = a;
```

上述语句只有一个if条件，没有对应的else，那么在if条件不成立的情况下，语句将不发生任何操作。再例如：

```
if(s==1'b0)
        y = a;
else
        y = b;
```

上述语句包含的是一个if-else结果，在if条件成立（为真）的情况下，执行“y = a;”，否则执行“y = b;”。

6. 常用过程语句

2） case语句

case语句如下语法格式：

```
case（case_expr）
case_item_expr{,case_item_expr}:procedural_statement
…
…
[default：procedural_statement]
endcase
```

6. 常用过程语句

2） case语句

case语句如下所示：

```
case(sel)
        2'b00:y = a; //分支1
        2'b01:y = b; //分支2
        2'b10:y = c; //分支3
        2'b11,
        2'b1x: y = d; //分支4
        default: y = 1'b0; //分支5
endcase
```

6. 常用过程语句

3） 循环语句

Verilog HDL中有四类循环语句，它们分别是：forever、repeat、while和for。

（1） forever循环语句

这一形式的循环语句语法如下：

```
forever
        procedural_statement
```

6. 常用过程语句

3） 循环语句

Verilog HDL中有四类循环语句，它们分别是：forever、repeat、while和for。

（1） forever循环语句

```
initial
begin
        Clock = 0;
        # 1 forever
        # 5 Clock = ~Clock;
end
```

6. 常用过程语句

3） 循环语句

Verilog HDL中有四类循环语句，它们分别是：forever、repeat、while和for。

（2）repeat循环语句

repeat循环语句形式如下：

repeat（loop_count）

procedural_statement

6. 常用过程语句

3） 循环语句

Verilog HDL中有四类循环语句，它们分别是：forever、repeat、while和for。

（2）repeat循环语句

```
parameter Count = 10;
integer sum = 1;
integer data=1;
repeat(Count)
begin
        data = sum* data;
        sum = sum + 1;
end
```

6. 常用过程语句

3） 循环语句

Verilog HDL中有四类循环语句，它们分别是：forever、repeat、while和for。

（3）while循环语句

while循环语句结构和用法都类似于C语言中的while语句，具体语法如下：

```
while（condition）
        procedural_statement
```

6. 常用过程语句

3） 循环语句

Verilog HDL中有四类循环语句，它们分别是：forever、repeat、while和for。

（3）while循环语句

```
integer Count = 10;
integer sum = 1;
integer data=1;
while (Count > 0)
begin
        data = sum* data;
        sum = sum + 1;
        Count = Count - 1;
end
```

6. 常用过程语句

3） 循环语句

Verilog HDL中有四类循环语句，它们分别是：forever、repeat、while和for。

（4）for循环语句

for循环语句格式和用法类似于C语言中的for语句，具体的形式如下：

```
for(initial_assignment;condition;step_assignment)
    procedural_statement
```

6. 常用过程语句

3） 循环语句

Verilog HDL中有四类循环语句，它们分别是：forever、repeat、while和for。

（4）for循环语句

```
integer Count = 10;
integer sum = 1;
integer data=1;
for (Count = 0; Count<10; Count = Count +1)
begin
        data = sum* data;
        sum = sum + 1;
end
```

知识点小结

知识要点：VerilogHDL行为流建模方法

知识难点： VerilogHDL行为建模适用环境