基本结构

1. Verilog HDL 语言

（1）Verilog HDL：既是一种行为描述语言；也是一种结构描述语言

（2）作用：进行数字电路的仿真验证、时序分析、逻辑综合

2. 语言结构

（1）Verilog HDL 程序嵌套在 module 和 endmodule 声明语句中；

（2）基本结构：模块声明/端口定义/IO 口说明/信号类型/功能描述/

a.模块声明：包含模块名字和模块输入、输出端口列表

```
module  模块名（端口 1，端口 2.....）
.....
endmodule
```

b.端口定义：三种端口类型：输入、输出、输入输出

```
input  端口 1，端口 2；
output  端口 1，端口 2；
inout  端口 1，端口 2；
```

c.信号类型：连线型、寄存器型（未定义默认连线型）

```
reg cout;
wire a,b,c;
```

d.功能描述：连续赋值语句（组合）；元件例化；always 语句(组合、时序)

3. 门类型关键字

not，and，nand，or，nor，xor，xnor

原语名称	逻辑表达式
and（与门）	L = A & B
nand（与非门）	L = ~(A & B)
or（或门）	L = A | B
nor（或非门）	L =~(A | B)
xor（异或门）	L = A ^ B
xnor（同或门）	L = A ~^ B

数据类型

1. wire 型变量：

(1)格式：wire 数据名 1,数据名 2, ……,数据名 n;

(2)向量：

wire[n-1:0] 数据名 1,数据名 2, ……,数据名 m;

或 wire[n:1] 数据名 1,数据名 2, ……,数据名 m;

2. reg 型变量：

(1)格式：reg 数据名 1,数据名 2, ……,数据名 n;

(2)向量：

reg[n-1:0] 数据名 1,数据名 2, ……,数据名 m;

或 reg[n:1] 数据名 1,数据名 2, ……,数据名 m;

注明：reg 型变量既可生成触发器，也可生成组合逻辑； wire 型变量只能生成组合逻辑

语句介绍

1. 结构说明语句

always 语句：不断重复执行，直到仿真结束

（1）在 always 块中被赋值的只能是 register 型变量（如 reg，integer，real，time）。

（2）每个 always 块在仿真一开始便开始执行，当执行完块中最后一个语句，继续从 always 块的开头执行

（3）格式：always<时序语句><语句>

（4）always 的时间控制可以为沿触发，也可为电平触发，posedge 表示上升沿；negedge 表示下降沿。

（5）如果 always 块中包含一个以上的语句，则这些语句必须放在 begin_end

（6）always 语句必须与一定的时序控制结合在一起才有用

（7）模板：

```
always @ (<敏感信号表达式>)
  begin
    //  过程赋值语句
    // if 语句
    // case 语句
    // while，repeat，for 循环
    // task，function 调用
  end
```

2. 循环语句

for 语句——通过 3 个步骤来决定语句的循环执行：

（1）给控制循环次数的变量赋初值。

（2）判定循环执行条件，若为假则跳出循环；若为真，则执行指定的语句后，转到第（3）步。

（3）修改循环变量的值，返回第（2）步。

形式：

一般形式：for （表达式 1；表达式 2；表达式 5）语句

简单形式：for（循环变量赋初值；循环执行条件；循环变量增值）执行语句

3. 条件语句

if-else 语句：

方式 1：if（表达式） 语句 1；

方式 2：

```
if（表达式 1）  语句 1；
else            语句 2；
```

方式 3：

```
if（表达式 1）  语句 1；
else if（表达式 2）语句 2；
     …
else if（表达式 n）语句 n；
```

case 语句：

```
case（敏感表达式）
    值 1：语句 1；
    值 2：语句 2；
      …
    值 n：语句 n；
    default: 语句 n+1；
endcase
```

4. 赋值语句

连续赋值语句：（assign 语句）用于对 wire 型变量赋值，是描述组合逻辑最常用的方法之一。

过程赋值语句：用于对 reg 型变量赋值，有两种方式：

（1）非阻塞（non-blocking)赋值方式：赋值符号为<=，如 b <= a ；

（2）阻塞（blocking)赋值方式：赋值符号为=，如 b = a ；

非阻塞（non-blocking)赋值方式 (b<= a)：

（1）b 的值被赋成新值 a 的操作, 并不是立刻完成的，而是在块结束时才完成；

（2）块内的多条赋值语句在块结束时同时赋值；

（3）硬件有对应的电路。

阻塞（blocking)赋值方式 (b = a)：

（1）b 的值立刻被赋成新值 a；

（2）完成该赋值语句后才能执行下一句的操作；

（3）硬件没有对应的电路，因而综合结果未知。

运算符

1. 逻辑运算符

逻辑运算符	说明
&&(双目)	逻辑与
||(双目)	逻辑或
!(单目)	逻辑非

（1）逻辑运算符把它的操作数当作布尔变量：

（2）非零的操作数被认为是真(1‘b1)；

（3）零被认为是假(1‘b0)；

（4）不确定的操作数如 4’bxx00，被认为是不确定的(可能为零，也可能为非零)（记为 1’bx)； 但 4’bxx11 被认为是真（记为 1’b1，因为它肯定是非零的)。

2. 位运算符

（1）位运算其结果与操作数位数相同。

（2）位运算符中的双目运算符要求对两个操作数的相应位逐位进行运算。

（2）两个不同长度的操作数进行位运算时，将自动按右端对齐，位数少的操作数会在高位用 0 补齐。

[例] 若 A = 5’b11001，B = 5’b101，

则 A & B = （5’b11001）&（5’b00101）= 5’b00001

位运算符	说明
~	按位取反
&	按位与
|	按位或
^	按位异或
^~，~^	按位同或

3. 关系运算符

（1）运算结果为 1 位的逻辑值 1 或 0 或 x。

（2）关系运算时，若关系为真，则返回值为 1；若声明的关系为假，则返回值为 0；若某操作数为不定值 x，则返回值为 x。

关系运算符	说明
<	小于
<=	小于或等于
>	大于
>=	大于或等于

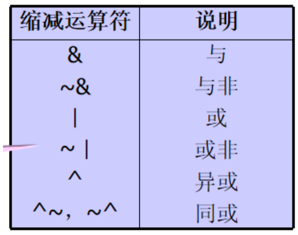
4. 等式运算符

（1）使用等于运算符时，两个操作数必须逐位相等,结果才为 1；若某些位为 x 或 z，则结果为 x。

（2）使用全等运算符时，若两个操作数的相应位完全一致，则结果为 1；否则为 0。

等式运算符	说明
==	等于
! =	不等于
===	全等
! ==	不全等

5. 缩减运算符



缩减运算符	说明
&	与
~&	与非
|	或
~ |	或非
^	异或
^~，~^	同或

（1）运算法则与位运算符类似，但运算过程不同！

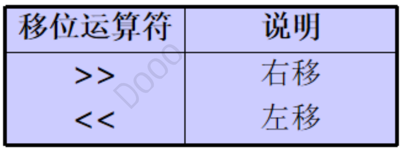
（2）对单个操作数进行递推运算,即先将操作数的最低位与第二位进行与、或、非运算，再将运算结果与第三位进行相同的运算，依次类推，直至最高位 。

（3）运算结果缩减为 1 位二进制数。

[例]reg[3:0] a;

b=|a;　　　　//等效于 b =((a[0] | a[1]) | a(2)) | a[3]

6. 移位运算符



移位运算符	说明
>>	右移
<<	左移

（1）用法：A>>n　 或 A<<n

（2）将操作数右移或左移 n 位，同时用 n 个 0 填补移出的空位。

[例] 4’b1001>>3 = 4’b0001；　4’b1001>>4 = 4’b0000

4’b1001<<1 = 5’b10010；　　4’b1001<<2 = 6’b100100；

1<<6 = 52’b1000000

7. 条件运算符

（1）当条件为真，信号取表达式 1 的值；为假，则取表达式 2 的值。

（2）条件运算符为 ？：

（3）用法：信号 = 条件？表达式 1：表达式 2