**Verilog HDL知识点**

**目录**

[1. VHDL的数字值 2](#_Toc521326019)

[1.1. 值集合 2](#_Toc521326020)

[1.2. 常量 2](#_Toc521326021)

[2. 变量 3](#_Toc521326022)

[2.1. wire 3](#_Toc521326023)

[2.2. reg 3](#_Toc521326024)

[2.3. 存储器 3](#_Toc521326025)

[2.4. 参数 4](#_Toc521326026)

[3. 运算符 4](#_Toc521326027)

[3.1. 运算符分类 4](#_Toc521326028)

[3.2. 运算符优先级 5](#_Toc521326029)

[4. VHDL的抽象级 6](#_Toc521326030)

[4.1. 门级器件 6](#_Toc521326031)

[4.2. 常用门的格式 7](#_Toc521326032)

[5. 模块 7](#_Toc521326033)

[5.1. 模块的结构 8](#_Toc521326034)

[5.2. 接口描述与逻辑功能描述 8](#_Toc521326035)

[6. 语句 9](#_Toc521326036)

[6.1. Verilog描述级 9](#_Toc521326037)

[6.1.1. 行为级描述 9](#_Toc521326038)

[6.1.2. 结构级描述 10](#_Toc521326039)

[6.2. 块语句 10](#_Toc521326040)

[6.2.1. begin-end语句 11](#_Toc521326041)

[6.3. 条件语句 11](#_Toc521326042)

[6.3.1. If-else语句 11](#_Toc521326043)

[6.3.2. Case语句 12](#_Toc521326044)

[6.4. 赋值语句 13](#_Toc521326045)

[6.4.1. 连续赋值语句 13](#_Toc521326046)

[6.4.2. 阻塞赋值语句 13](#_Toc521326047)

[6.4.3. 非阻塞赋值语句 14](#_Toc521326048)

[6.5. 结构说明语句 15](#_Toc521326049)

[6.5.1. initial语句 15](#_Toc521326050)

[6.5.2. always语句 15](#_Toc521326051)

[6.5.3. function语句 16](#_Toc521326052)

# VHDL的数字值

## 值集合

Verilog HDL中规定了四种基本的值类型：

0：逻辑0或“假”；

1：逻辑1或“真”；

**X：未知值；**

**Z：高阻。**

注意：**x 值和z 值都不区分大小写**，也就是说，值0x1z 与值0X1Z 相同。

## 常量

VHDL有三种常量：整型、实型、字符串型（主要介绍整型）整型的表示方法包括十进制表示和基数表示

**（1）十进制表示**

32 十进制数32

-15 十进制数-15

**（2）基数表示**

表示格式为：**[size ]’base value**  其中size是位宽 base为进制数，base为进制数value数base进制下的数字序列(base与value中间无空格)

base取值为o或O表示八进制，b或B表示二进制，d或D表示十进制，h或H表示十六进制.

**如：**

**5’o37 五位八进制数（二进制11111，十进制31）**

**4’d2 四位十进制数，等价于4’D0002（二进制10，十进制2））**

**32'h8CB10000 三十二位十六进制数（常用于表示指令）**

**8’ha2 八位十六进制数（十进制162）**

**使用方法：**

**assign ram[0] = 32'h00002820;**

# 变量

vhdl的变量类型包括wire（线型）、reg（寄存器）、mem存储器、parameter(参数)

## wire

wire用于对结构化器件之间的物理连线的建模。如器件的管脚，内部器件如与门的输出等。可以用作任意函数或方程的输出，也通常用assign进行赋值。

**定义格式：**

**wire [n-1:0] 数据名1，…，数据名m**；n表示wire变量的位宽（n-1等于31表示32位），如wire[31:0] T4

**赋值格式：**

input [31:0] A1;

input [31:0]A2;

wire [31:0] res

assign res = A1&A2 //或 res = A1&A2

## reg

reg是vhdl常用的寄存器类型， reg型数据常用来表示“always”模块内的指定信号，代表触发器。通常在设计中要由“always”模块通过使用行为描述语句来表达逻辑关系。**在“always”模块内赋值的每一个信号都必须定义成reg型。**reg缺省值时为默认为x

**定义格式：**

**reg [n-1:0] 数据名1，….，数据名n；**

**如：**

reg [3:0] Sat; // Sat为4 位寄存器。

reg Cnt; //1 位寄存器。（默认为1位）

## 存储器

**用寄存器数组类型来建立存储器的模型**，如对2个8位的RAM建模如下：

**reg [7：0] Mem[0：1] ；表示2个8位的存储器**

对存储单元的赋值必须一个个赋值，如上2个8位的RAM的赋值必须用两条赋值语句：

Mem[0] = ’ h 55；

Mem[1] = ’ haa； //缺省位宽，默认为32位

**注意：**

**reg [n-1:0] rega; //一个n位的寄存器**

**reg mem [n-1:0]; //一个由n个1位寄存器构成的存储器**

## 参数

parameter：参数数据类型，其实就是个常量，通常出现在module内部，常被定义为状态机状态、数据位宽和延迟大小，参数的定义是局部的，只在当前模块有效。

**定义格式：**

parameter N = 8'd5;

parameter P = 4'b0001;

# 运算符

## 运算符分类

Verilog HDL语言的运算符范围很广，按功能可以分为以下几类：

1. 算术运算符（**+、-、X、/、%**）

表示加减乘除、取余；

1. 赋值运算符（**=、<=**）；

表示阻塞和非阻塞赋值

1. 关系运算符（**>、<、>=、<=**）；

大于，小于，大于等于，小于等于

1. 逻辑运算符（**&&、||、!**）；

逻辑与、或、非

1. 条件运算符（**? :**）；

cond\_expr ? expr1 : expr2 Cond\_expr为真则取expr1，假则取expr2

1. 位运算符（**~、|、^、&、^~**）；

按位取非、或、异或、与、同或

1. 移位运算符（**<<、>>**）；

按二进制左移和右移

1. 拼接运算符（**{}**）；

将小表达式合并成大表达式

1. 其他（**~|**）；

或非（规约）操作符，~|A 将A的每一位一起取或，最后取非，如~|01 = 0

## 运算符优先级

表3-1 运算符优先级

|  |  |
| --- | --- |
| 运算符 | 优先级别 |
| !、~ | 高优先级别 |
| \*、/、 % |  |
| +、- |  |
| <<、>> |  |
| <、<=、>、>= |  |
| ==、!=、===、!== |  |
| & |  |
| ^、^~ |  |
| | |  |
| && |  |
| || |  |
| ? : | 低优先级别 |

***说明：自上而下优先级逐步降低。***

# VHDL的抽象级

**1、系统级（System-Level）：**用高级语言结构实现设计模块的模型。

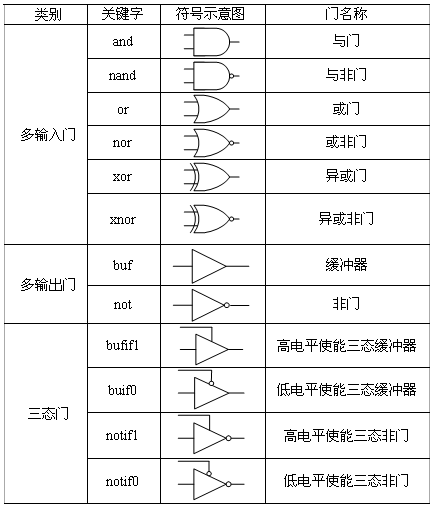
**2、算法级（Algorithem-level）**：用高级语言结构实现设计算法的模型。

**3、RTL级（Register-Transfer-Level）：**描述数据在寄存器之间流动和如何处理这些数据的模型。

**4、门级（Gate-Level）：**描述逻辑门以及逻辑门之间连接的模型。

## 门级器件

表4-1 门级器件



## 常用门的格式

**多输入门的格式：**

**gate [Name](outputA,input1,input2,…,inputN)**

gate是门的类型，Name为实如名，**只有第一个参数是输出**，后续参数都是输入

如：

**and A1(out1,In1,In2)**

表示一个而输入与门。

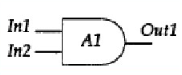


图4-1 与门

**多输出门的格式：**

**gate [Name](input1,input2,input3,…,outputA)**

gate是门的类型，Name为实如名，**只有最后一个参数是输入**，之前的参数都是输出

如：

**not (out1,out2,…,outN,InputA)**

表示一个多输出非门

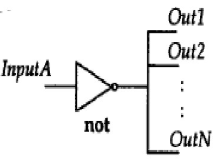


图4-2 非门

以上门级语句对应verilog的结构级描述。

# 模块

一个模块是由两部分组成的，一部分描述**接口**，另一部分描述**逻辑功能**，即定义输入是如何影响输出的。如：

**module addr (a, b, cin, count, sum);**

input [2:0] a;

input [2:0] b;

input cin;

output count;

output [2:0] sum;

assign {count,sum} = a +b + cin;

**endmodule**

## 模块的结构

（1）模块内容是嵌在**module** 和**endmodule**两个语句之间。每个模块实现特定的功能，模块可进行层次的嵌套，即一个模块调用了另一个模块。因此可以将大型的数字电路设计分割成大小不一的小模块来实现特定的功能。

（2）模块包括**接口描述部分**和**逻辑功能描述部分。**

**对每个模块都要进行端口定义**，并说明输入、输出口，然后对模块的功能进行逻辑描述，对测试模块，可以没有输入输出口。

（3）Verilog HDL 的书写格式自由，一行可以写几个语句，也可以一个语句分几行写。具体由代码书写规范约束。

（4）除endmodule 语句外，每个语句后面需**有分号表示该语句结束。**

## 接口描述与逻辑功能描述

**接口描述：**

模块定义声明了模块的输入、输出口，格式如下：

**module 模块名（接口1，接口2，接口3，….）;**

如：

module addr (a, b, cin, count, sum); 其中module 是模块的保留字，addr 是模块的名字。**（）内是该模块的输入输出**，是该模块与其他模块通讯的外部接口。

**逻辑功能描述：**

模块的内容包括I/O说明，内部信号、调用模块等声明语句和功能定义语句,这些语句共同描述了模块的逻辑功能。

I/O说明的格式：

**输入口：input[n-1:0] 端口名1，端口名2，…；**

**输出口：output[n-1:0] 端口名1，端口名2，…；**

其中的input 、output是保留字，定义了管脚信号的流向，[n-1:0]表示该信号的位宽。

**三种描述逻辑功能的方法**

（1）用assign声明语句 如：**assign a = b & c；**

（2）用实例原件 如：**and and1(q, a, b);**

（3）用always块，always常用与描述时序逻辑

如：

always @(posedge clk or posedge clr)

begin

if (clr) q<=0;

else if(en) q <=d;

end

这是一个带一部清除端的D触发器。

# 语句

## Verilog描述级

Verilog的语句用于描述硬件电路，既可以进行行为级描述，也可已进行结构级描述：

1. 行为描述级。侧重对模块**行为功能**的抽象描述。
2. 结构描述级。侧重对模块**内部结构实现**的具体描述。**又被认为是寄存器级(RTL级)，数据流多从一个寄存器流向另一个寄存器。**

### 行为级描述

**描述体的组成：**

（1）描述体由多个并行运行的过程块组成。

（2）过程块由过程语句（**initial和always**）和块语句（串行块**begin-end**和并行块fork-join）组成。

（3）块语句由过程赋值语句和高级程序语句构成。其中过程赋值语句包括阻塞与非阻塞式赋值语句。高级程序语句包括**if-else、case**等语句。

如：

module fadder(sum,a,b,cin);

output sum,cout;

input a,b,cin;

reg sum,cout;

always @(a or b or cin)

begin

sum=a^b^cin;

cout=(a&b)|(b&cin)|(a&cin);

end

endmodule

这段Verilog代码用行为级描述了一个全加器。

### 结构级描述

**描述体的组成**

（1）门级描述：由**基本逻辑门**（and、or、not、xor等）互连而成的具有一定功能的电路模块的描述。

（2）结构级描述：将门级描述的逻辑门用一个个功能模块替换，就拓展到一般意义的结构级描述。

如：

module fadder(sum,a,b,cin);

output sum,cout;

input a,b,cin;

xor U0(sum,a,b,cin);

and U1(net1,a,b);

and U2(net2,a,cin);

and U3(net3,b,cin);

or U4(cout,net1,net2,net3);

endmodule

这段代码用结构级描述了一个全加器，多用利用门级器件的组合实现功能。

## 块语句

块语句用于**行为级**描述，通常用来将两条或多条语句组合在一起，使其在格式上更象一条语句，**有点类似c语言的{ }**。**块语句有两种，一种是begin-end语句，用它来标识的块称为顺序块；一种是fork-join语句，通常用来标识并行执行的语句，用它来标识的块称为并行块。（只介绍begin-end块）**

### begin-end语句

格式如下：

**begin**

**语句1；**

**语句2；**

**……**

**end**

用begin和end关键字包括语句

如：

begin

areg = breg;

creg = areg; // creg的值即为breg值

end

## 条件语句

条件语句用于**行为级**描述。

### If-else语句

Verilog HDL语言提供了3种形式的if语句。

**（1） if（表达式）语句**

如：if (a>b) out <= int1;

**（2） if（表达式）语句1；**

**else 语句2；**

**如：**

if(Sum < 60)

begin

Grade = C;

Total\_C = Total \_c + 1;

end

else if(Sum < 75)

begin

Grade = B;

Total\_B = Total\_B + 1;

end

else

begin

Grade = A;

Total\_A = Total\_A + 1;

end

**注意：若if或else内包括多条语句，则包含在begin-end块内**

**（3）if（表达式1）语句1；**

**else if（表达式2）语句2；**

**else if（表达式3）语句3；**

**………………….**

**else if（表达式n）语句n；**

**else 语句m;**

if语句说明：

（1）表达式一般为逻辑表达式或关系表达式，若为0、X、Z按“假”处理；若为1，按“真”处理。

（2）在if和else后面可以包含一个内嵌的操作语句。

### Case语句

case语句常用于多分支选择语句，它的一般形式如下：

**（1）case（表达式）< case 分支项> endcase**

**（2）casez（表达式）< case 分支项> endcase**

**（3）casex（表达式）< case 分支项> endcase**

分支项的一般格式如下：

**分支表达式：语句；**

**缺省项（default项）：语句；**

如：

reg [15:0] rega;

reg [9:0] result;

case (rega)

16’d0: result =10’b0111111111;

16’d1: result =10’b1011111111;

16’d2: result =10’b1101111111;

16’d3: result =10’b1110111111;

16’d4: result =10’b1111011111;

16’d5: result =10’b1111101111;

16’d6: result =10’b1111110111;

16’d7: result =10’b1111111011;

default: result =’bx;

endcase

说明：rega取16’d0时，result =10’b0111111111，取16’d1时，result =10’b1011111111，以此类推。

## 赋值语句

### 连续赋值语句

连续赋值语句的格式：

**assign net\_type = 表达式；**

连续赋值语句用于组合逻辑的建模。等式左边是wire 类型的变量。等式右边可以是常量、由运算符如逻辑运算符、算术运算符参与的表达。

*说明： assign可以用于initial和always语句中为寄存器变量赋值，称为过程性连续赋值（了解就好）*

如：

wire [3:0] Z, Preset, Clear; //wire声明

assign Z = Preset & Clear; //连续赋值语句

wire Cout, C i n ;

wire [3:0] Sum, A, B;

assign {Cout, Sum} = A + B + Cin;

assign Mux = (S = = 3)? D : 'bz;

### 阻塞赋值语句

格式：**b=a；**说明：

（1）赋值语句未结束，后面的语句不能执行；

（2）顺序执行。

如：

always@(posedge clk)

begin

b = a;

c = b;

end

**在时钟上升沿到来时，a被赋给b，b又被赋给c，最后a=c，在一个上升沿完成赋值。clk是输入时钟，posedge表示时钟上升沿，另外还有negedge表示下降沿。**

### 非阻塞赋值语句

格式：**b<=a**

说明：常用于时序电路

如：

always@(posedge clk)

begin

b <= a;

c <= b;

end

**一次上升沿到来时，b的上一个值b0被赋给c，a被赋给b。等待下一次上升沿到来时，b再次赋给c，c才能等于a。**

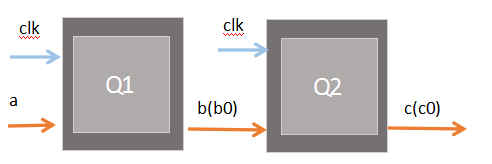


图6-1 非阻塞赋值

## 结构说明语句

### initial语句

initial和always说明语句**在仿真的一开始就开始执行。Initial语句只执行一次。**相反always语句则是不断地重复执行，直到仿真过程结束。在一个模块中，使用Initial和always语句的次数是不受限制的。**initial语句用于行为级描述。**

格式：**Initial**

**begin**

**语句1；**

**语句2；**

**end**

如：

Initial

begin

areg = 0; //初始化寄存器areg

for (index = 0; index <size; index = index + 1)

memory[index]=0; //初始化一个memory

end

用initial语句在仿真的初始状态对各变量初始化。

### always语句

always语句在仿真过程中是不断重复执行的。格式如下：

**always <时序控制> <语句> //Always用于行为级描述**

always语句由于其不断重复执行的特征，只有和一定的时序控制结合在一起才有用，表示：不断输入时钟信号，当时钟为上升或下降沿时执行程序。如：

reg[7:0] counter;

reg tick;

always @(posedge areg)

begin

tick = ~tick;

counter = counter +1;

end

### function语句

函数通常是大的程序在不同地点多次应用的相同程序段，函数可以简化程序结构，是程序更加简单易懂。函数通常有着输入和输出，其目的是返回一个用于表达式的值。

**function <返回值的类型或范围> （函数名）；**

**<端口说明语句>**

**<变量类型说明语句>**

**begin**

**<语句>**

**end**

**endfunction** *//说明：返回值类型可以缺省，若缺省返回一位寄存器数据。*

如：

function [31:0] select;

input [31:0] A0;

begin

<说明语句>

getbyte = A0;

end

endfunction

**函数的调用：**

格式：

**<函数名>（<表达式><,<表达式>>）**

如;

Input [15:0]A0;

Assign y = getbyte(A0);