|  |
| --- |
| 数字电路实验 |
| Verilog初学者手册 |
|  |

|  |
| --- |
| 梁润秋 滕思洁  2017-9-24 |

# Verilog初学者手册

目录

[Verilog初学者手册 1](#_Toc495402301)

[一、介绍 2](#_Toc495402302)

[1.1 结构 2](#_Toc495402303)

[1.2模块端口定义 2](#_Toc495402304)

[1.3模块内容： 3](#_Toc495402305)

[二、数据类型和运算符 7](#_Toc495402306)

[2.1间隔符与注释符 7](#_Toc495402307)

[2.2操作符 7](#_Toc495402308)

[2.3数值常量 9](#_Toc495402309)

[2.4字符串 10](#_Toc495402310)

[2.5标识符 10](#_Toc495402311)

[2.6关键字 10](#_Toc495402312)

[2.7变量 11](#_Toc495402313)

[三、基本语句 12](#_Toc495402314)

[3.1赋值语句 12](#_Toc495402315)

[3.2条件语句 13](#_Toc495402316)

[3.3循环语句 15](#_Toc495402317)

[3.4结构声明语句 16](#_Toc495402318)

[四、不同抽象级别的Verilog HDL模型 17](#_Toc495402319)

[4.1Verilog HDL门级描述方式 18](#_Toc495402320)

[4.2Verilog HDL行为级描述 18](#_Toc495402321)

[五、其他 19](#_Toc495402322)

[1.组合逻辑和时序逻辑的区别 19](#_Toc495402323)

[2.和C语言相比较 19](#_Toc495402324)

[3.作为一个助教的一些建议 19](#_Toc495402325)

# 一、介绍

Verilog HDL是一种标准通用的硬件描述语言，是数字电路这门课主要使用的语言。Verilog由于容易学习，所以是目前应用较为广泛的一种硬件描述语言。

## 1.1 结构

Verilog HDL程序由模块(module)组成，模块的基本结构如下：



module full\_addr(s,cout,a,b,cin);//模块端口定义

input a,b,cin; //I/O声明

output s,cout;

assign {cout,s} = a+b+cin;//功能描述

endmodule

**注意：**

1.模块内容位于module和endmodule之间；每个模块都有一个模块名(如上模块的模块名是full\_addr)。

2.除了endmodule意外，所有语句后面必须有“;”。

3.语句也可以通过用begin和end两个保留字包围起来的多条语句组成的复合语句。

## 1.2模块端口定义

模块端口定义用来生命设计模块的输入/输出端口，其格式如下：

module 模块名(端口1,端口2,端口3,端口4,......)

## 1.3模块内容：

**（1）I/O声明**

模块的I/O声明用来声明各端口信号流动方向，包括输入（input），输出（output）和双向（inout）。I/O声明格式如下：

输入口：input 端口名1，端口名2，……端口名n

输出口：output 端口名1，端口名2，……端口名n

输入、输出口：inout 端口名1，端口名2，……端口名n

**（2）信号类型声明**

信号类型声明是声明设计电路的功能描述中所用的信号的数据类型和函数。信号的数据类型主要有连线（wire）、寄存器（reg）、整型（integer）、实型（real）、和时间（time）等。

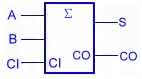
**（3）功能描述**

功能描述是Verilog HDL程序设计中最主要的部分，用来描述设计模块的内部结构和模块端口间的逻辑关系，在电路上相当于器件的内部电路结构。功能描述可以用assign语句、元件例化（instantiate）方式、always块语句、initial块语句等方法来实现，通常将设计模块描述的方法称为建模。

①用assign语句建模

用assign语句建模的方法很简单，只需要在“assign”后面再加一个表达式。Assign语句一般适合对组合逻辑进行赋值，称为连续赋值方式。

**例一位全加器的设计**

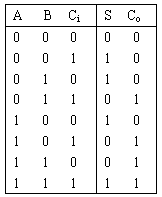


1位全加器的逻辑符号：

sum是全加器的和输出端，

cout是进位输出端，ina和inb是两个加数输入端，

cin是低位进位输入端。



全加器的Verilog HDL源程序如下：

module adder1 (sum, cout, ina, inb, cin);

input ina, inb, cin;

output sum, cout;

assign {cout, sum} = ina+inb+cin;

endmodule

“assign {cout, sum} = ina+inb+cin;”语句实现1位全加器的进位输出cout与和输出sum的建模。

在语句表达式中，用拼接运算符“{}”将cout、sum这两个1位操作数拼接为一个2位操作数。

②用元件例化（instantiate）方式建模

元件例化方式建模是利用Verilog HDL提供的元件库实现的。例如，用与门例化元件定义一个三输入端与门可以写为

and myand3(y,a,b,c); and是Verilog HDL元件库中与门元件名，myand3是例化出的三输入端与门名，y是与门输出端，a、b、c是输入端。

③用always块语句建模

always块语句可以产生各种逻辑，常用于时序逻辑的功能描述。一个程序设计模块中，可以包含一个或多个always语句。程序运行中，在某种条件满足时，就重复执行一遍always结构中的语句。

**例 8位二进制加法计数器的设计**

8位二进制加法计数器的逻辑符号如图。

OUT是8位二进制计数器的输出端（8位向量）；

COUT是进位输出端（1位）；

DATA是并行数据输入端（8位向量）；

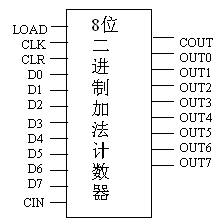
LOAD是计数器的预置控制输入端，

当LOAD=1时，OUT=DATA；

CLK是时钟控制输入端，上升沿为有效边沿；

CLR是同步复位输入端，当CLK的上升沿到来时且CLR=1，则计数器被复位，OUT=00000000。

（逻辑符号图是由计算机对计数器电路的Verilog HDL源代码编译后产生的元件符号，图中的输入/输出标识符自动被改为大写，而源程序中的标识符都是小写。）



8位二进制加法计数器的Verilog HDL 源程序如下：

module cnt8 (out, cout,data,load,cin,clk, clr);

input [7:0] data;

input load, cin, clk, clr

output [7:0] out;

output cout;

reg [7:0] out;

always @(posedge clk)

begin

if (load) out=data;

else if(clr) out=’b00000000;

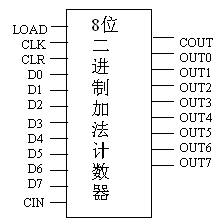
else out=out+1;

end

assign cout = &out;

endmodule

用always块语句实现8位二进制加法计数器的建模。@(posedge clk)是时间控制敏感函数，表示clk上升沿到来的敏感时刻。每当clk的上升沿到来时，always块语句中的全部语句就执行一遍。“assign cout = &out;”语句产生进位输出cout，在语句中“&out”是与的缩减运算式，只有out中数字全为1时，结果才为1。



④用initial块语句建模

Initial块语句与always语句类似，不过在程序中它只执行1次就结束了。

Initial块语句的使用格式 ：

Initial

Begin

语句1;

语句2;

:

end

例3 用Initial过程语句对测试变量赋值

initial

begin

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

memory[addr]=0; //对memory存储器进行初始化

end

从以上例子中可以看出

Verilog HDL程序设计模块的基本结构：

①Verilog HDL程序是由模块构成的。每个模块的内容都是嵌套在module和endmodule两语句之间，每个模块实现特定的功能，模块是可以进行层次嵌套的。

②每个模块首先要进行端口定义，并声明输入（input）、输出（output）或双向（inouts），然后对模块的功能进行逻辑描述。

③Verilog HDL程序的书写格式自由，一行可以有一条或多条语句，一条语句也可以分为多行写。

④除了end或以end开头的关键字（如endmodule）语句外，每条语句后必须要有分号“；”。

⑤可以用/\*……\*/或//……对Verilog HDL程序的任何部分注释。一个完整的源程序都应当加上必要的注释，以加强程序的可读性。

# 二、数据类型和运算符

Verilog HDL的词法标识符包括：间隔符与注释符、操作符、数值常量、字符串、标识符和关键字。

## 2.1间隔符与注释符

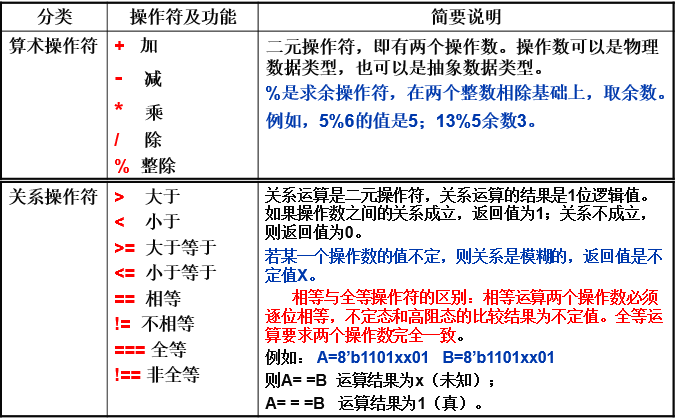
间隔符又称空白符，包括空格符、tab符号、换行符及换页符等。它们的作用是分隔其他词法标识符。在必要的地方插入间隔符可以增强源文件的可读性。但在字符串中空格符和tab符号（制表符）是有意义的字符。

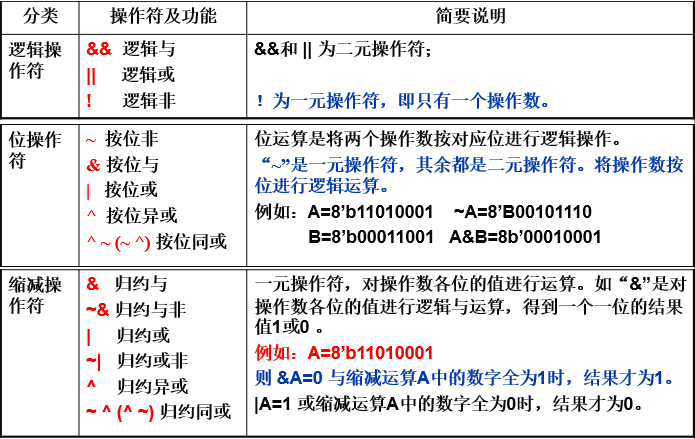
Verilog HDL 有单行注释和多行段注释两种注释形式。单行注释以字符“//”起始，到本行结束；而段注释则以“/\*”起始以“\*/”结束，在段注释中不允许嵌套，段注释中单行注释标识符“//”没有任何特殊意义。

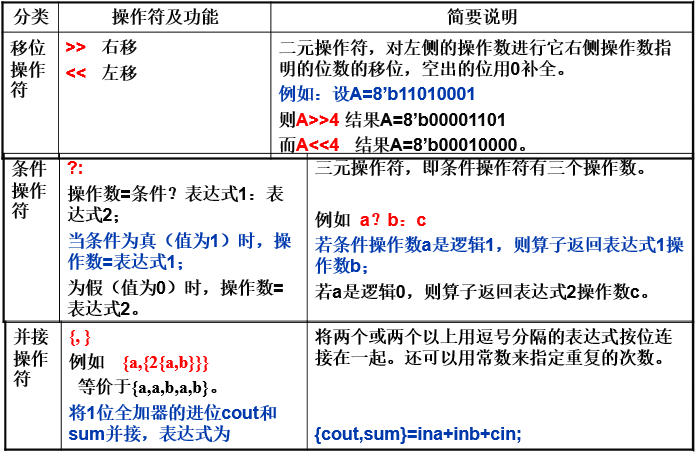
## 2.2操作符

Verilog HDL 中定义了操作符，又称运算符，按照操作数的个数，可以分为一元、二元和三元操作符；按功能可以大致分为算术操作符、逻辑操作符、比较操作符等几大类。

**Verilog HDL 的操作符及简要说明如下图：**







**操作符优先级如下图：**



**注意：**列表顶部是最高优先级，底部是最低优先级。列在同一行中的操作符具有相同的优先级。所有操作符（？：除外）在表达式中都是从左向右结合的。圆括号（）用于改变优先级或使得表达式中运算顺序更加清晰，提高源文件的可读性。

2.3数值常量

Verilog HDL中的数值常量有整型和实型两大类，分为十进制、十六进制、八进制或二进制。若在前面加上一个正“+”或负“—”号就表示有符号数，否则所代表的就是无符号数。在数值常量的任意位置可以随意插入下划线“\_”以提高可读性。

常量定义格式为：

parameter 常量名1=表达式，常量名2=表达式，…，常量名n=表达式；

parameter是常量定义关键字，常量名是用户定义的标识符，表达式为常量赋值。

例如 parameter Vcc=5, fbus=8’b11010001;

Verilog HDL中的整型数值常量就是整数，有两种书写格式：一种是无位宽的十进制表示法，如-132。第二种是定义位宽和进制的表示法，这种表示方法通常是无符号数。

常数书写格式是：

[size]’base value

其中size是位宽，定义了数值常量的位数（长度）；base 代表这个数据的进制，取值范围和相应的进制如下表；value是一个数值常量的值，书写格式与进制base相对应。

例如 4’h6a8c，表示一个4位十六进制数。

2’hf5 等于8’b11110101； 8’b1111xxxx 等价2’hfx；

8’b1101zzzz 等价2’hdz

Verilog HDL中的进制

|  |  |
| --- | --- |
| **base进制代码取值** | **对应的进制** |
| **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** |

Verilog HDL中的实型数值常量就是浮点数，可以用十进制与科学计数法两种形式书写。如果采用十进制格式，小数点两边必须都有数字。

Verilog HDL的编程最终是与硬件相对应的。由于硬件电路中信号的逻辑状态具有特殊性，即不仅有0（低电平）和1（高电平），还有可能是X（未知状态）和Z（高阻态），因此Verilog HDL数值集合有四个基本值：

0：逻辑0或假状态；

1：逻辑1或真状态；

X：逻辑不确定；

Z：高阻态。

## 2.4字符串

字符串是双引号“”括起来的字符序列，必须包含在同一行中，不能多行书写。在表达式或赋值语句中作为操作数的字符串被看作ASCII值序列，即一个字符串中的每一个字符对应一个8位ASCII值。

## 2.5标识符

标识符是模块、寄存器、端口、连线、示例和begin-end块等元素的名称，是赋给对象的唯一的名称。标识符可以是字母、数字、$符和下划线“\_”字符的任意组合序列。

定义标识符规则：

①首字符不能是数字，必须以字母或下划线“\_”开头。

②字符数不能多于1024。

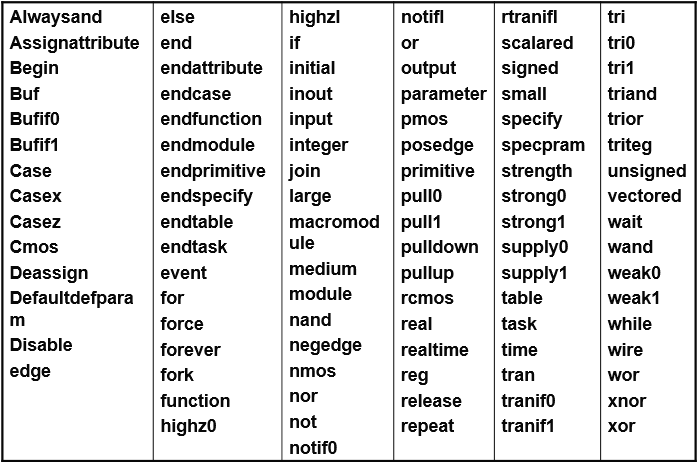
③标识符区分大小写。

④不要与关键字同名。

例如 ina、inb、adder、adder8、name\_adder是正确的，而1a ?b是错误的。

## 2.6关键字

关键字是Verilog HDL预先定义的专用词。在IEEE标准——Verilog HDL 1364-1995中规定了102个关键词，都采用小写形式。关键词有其特定和专有的语法作用，用户不能再对它们做新的定义。



## 2.7变量

变量是在程序运行时其值可以改变的量。在Verilog HDL中，变量分为网络型（nets type）和寄存器型（register type）两种。

①网络型变量

nets型变量是输出值始终根据输入变化而更新的变量，一般用来定义硬件电路中的各种物理连线。Verilog HDL提供了多种nets型变量。

常见的nets型变量及说明



②寄存器型变量

register型变量是用来描述硬件系统的基本数据对象。作为一种数值容器，可以容纳当前值，也可以保持历史值。与寄存器的记忆功能相对应，可以作为模块各器件间的信息传递通道。

register型变量与wire型变量的区别在于register型变量需要被明确的赋值， 并且在重新赋值前一直保持原值。register型变量是在always、initial等过程语句种定义，并通过过程语句赋值。

常见的register型变量及说明



# 三、基本语句

Verilog HDL的语句包括赋值语句、条件语句、循环语句、结构声明语句和编译预处理语句等类型，每一类语句又包括几种不同的语句。在这些语句中，有些语句属于顺序执行语句，有些语句属于并行执行语句。

## 3.1赋值语句

（1）赋值语句

在Verilog HDL中，赋值语句常用于描述硬件设计电路输出与输入之间的信息传送，改变输出结果。Verilog HDL有4种赋值方法：门基元、连续赋值、过程赋值和非阻塞赋值。

①门基元赋值语句

门基元赋值语句的格式为：

基本逻辑门关键字 （门输出，门输入1，门输入2，…，门输入n）；

例如 4输入与非门的门基元赋值语句为

nand (y,a,b,c,d); //与语句assign y=~(a&b&c&d)等效

②连续赋值语句

连续赋值语句的关键字assign，赋值符号是“=”，赋值语句的格式为

assign 赋值变量=表达式；

例如 assign y=~(a&b&c&d)；

连续赋值语句的“=”两边的变量都应该是wire型变量。在执行中，输出y的变化跟随输入a、b、c、d的变化而变化，反映了信息传送的连续性。连续赋值语句用于逻辑门和组合逻辑电路的描述。

**例 四输入端与非门的Verilog HDL源程序**

module example1(y,a,b,c,d);

output y;

input a,b,c,d;

assign #1 y=~(a&b&c&d);

//#1表示输出与输入信号之间具有1个单位的时间延迟

Endmodule

③过程赋值语句

过程赋值语句出现在initial和always块语句中，赋值符号是“=”，语句格式为：

赋值变量=表达式；

过程赋值语句“=”左边的赋值变量必须是（寄存器）reg型变量，其值在该语句结束时即可得到。

如果一个块语句中包含若干条过程赋值语句，按顺序一条一条执行，前面的语句没完成，后面的语句就不能执行。

因此，过程赋值语句也称为阻塞赋值语句。

④非阻塞赋值语句

非阻塞赋值语句也是出现在initial和always块语句中，赋值符号是“<=”，语句格式为：

赋值变量<=表达式；

非阻塞赋值语句“<=”左边的赋值变量必须是（寄存器）reg型变量，其值在块语句结束时才可得到，与过程赋值语句不同。

例如 下面的块语句包含4条赋值语句

always @(posedge clock)

m=3; n=75; n<=m; r=n;

语句执行结束后，r的值是75，而不是3，因为第3行是非阻塞赋值语句“n<=m”，该语句要等到本块语句结束时，n的值才能改变。

块语句的“@(posedge clock)”是定时控制敏感函数，表示时钟信号clock的上升沿到来的敏感时刻。

**例 上升沿触发的D触发器的Verilog HDL源程序**

module D\_FF(q,d,clock);

input d,clock; //属于wire型变量，由隐含规则定义。

output q; //属于reg型变量

reg q;

always @(posedge clock)

q=d;

endmodule

## 3.2条件语句

条件语句包含if语句和case语句，它们都是顺序语句，应放在always块中。

①if语句

完整的Verilog HDL的if语句结构如下：

if（表达式）

begin

语句；

end

else if（表达式）

begin

语句；

end

else

begin

语句；

end

在if语句中，“表达式”是逻辑表达式或关系表达式，也可以是位宽为1位的变量。

系统对表达式的值进行判断， 若为0，x，z按“假”处理；若为1，按“真”处理，执行指定的语句。

**例 8线-3线优先编码器的设计**

8线-3线优先编码器的功能表

module code(y,a);

input[7:0] a;

output[2:0] y;

reg[2:0] y;

always @(a)

begin

if(~a[7]) y<=3’b111;

else if(~a[6]) y<=3’b110;

else if(~a[5]) y<=3’b101;

else if(~a[4]) y<=3’b100;

else if(~a[3]) y<=3’b011;

else if(~a[2]) y<=3’b010;

else if(~a[1]) y<=3’b001;

else y<=3’b000;

end

endmodule

真值表：

|  |  |
| --- | --- |
| **输入** | **输出** |
| **in0 in1 in2 in3 in4 in5 in6 in7** | **out2 out1 out0** |
| **x x x x x x x 0**  **x x x x x x 0 1**  **x x x x x 0 1 1**  **x x x x 0 1 1 1**  **x x x 0 1 1 1 1**  **x x 0 1 1 1 1 1**  **x 0 1 1 1 1 1 1**  **0 1 1 1 1 1 1 1** | **1 1 1**  **1 1 0**  **1 0 1**  **1 0 0**  **0 1 1**  **0 1 0**  **0 0 1**  **0 0 0** |

②case语句

case语句是一种多分支的条件语句，case语句的格式为：

case（表达式）

选择值1： 语句1；

选择值2： 语句2；

…

选择值n： 语句n；

default: n+1

endcase

执行case语句时，首先计算表达式的值，然后执行条件句中相应的“选择值”的语句。当所有的条件句的“选择值”与表达式的值不同时，则执行“default”后的语句。default可以省略。

case语句多用于数字系统中的译码器、数据选择器、状态机及微处理器的指令译码器等电路的描述。

**例：用case语句描述4选1数据选择器**

控制信号s1,s2有4种组合，控制a,b,c,d中的一个数据送到输出端。

4选1数据选择器Verilog HDL的描述如下：

module mux41(z,a,b,c,d,s1,s2);

input s1,s2;

input a,b,c,d;

output z;

reg z;

always @(s1 or s2)

begin

case({s1,s2})

2’b00: z=a;

2’b01: z=b;

2’b10: z=c;

2’b11: z=d;

endcase

end

endmodule

case语句还有两种变体形式，casez和casex语句。与case语句的区别是不考虑语句中的高阻z和未知x的那些位，只关注其他位的比较结果。

## 3.3循环语句

循环语句包含for语句、repeat语句、while语句和forever语句4种。

①for语句

for语句的格式为：

for（循环指针=初值；循环指针< 终值；循环指针=循环指针+步长值）

begin

语句；

end

for语句可以是一组语句重复执行，语句中的参数一般属于整型变量或常量。语句重复执行的次数由语句中的参数确定。即

循环重复次数=（终值-初值）/步长值

②repeat语句

repeat语句的语法格式为：

repeat（循环次数表达式）语句；

③while语句

while语句的语法格式为：

while（循环执行条件表达式）

begin

重复执行语句；

修改循环条件语句；

end

while语句在执行时，首先判断循环执行条件表达式是否为真。若为真，则执行其后面的语句；若为假，则不执行 ，表示循环结束。为了使语句能够结束，在循环执行的语句中必须包含一条能改变循环条件的语句。

④forever语句

forever语句的语法格式为：

forever

begin

语句；

end

forever是一种无穷循环控制语句，它不断地执行其后的语句或语句块，永远不会结束。Forever语句常用来产生周期性的波形，作为仿真激励信号。例如产生时钟clk的语句为：

#10 forever #10 clk=!clk;

//#10表示输出与输入信号之间具有10个单位的时间延迟，!clk取非

## 3.4结构声明语句

Verilog HDL的任何过程模块都是放在结构声明语句中，结构声明语句包括always、initial、task和function等4种结构。

①always块语句

在一个Verilog HDL模块（module）中，always块语句的使用次数是不受限制的，块内的语句也是不断重复执行的。always块语句的语法结构为：

always @()

begin

//过程赋值语句；

//if 语句，case语句；

//for语句， while语句，repeat语句；

//task语句、 function语句；

end

在always块语句中，敏感信号表达式应该列出影响块内取值的所有信号（指设计电路的输入信号），多个信号之间用“or”连接。当表达式中任何信号发生变化时，就会执行一遍块内的语句。块内语句可以包括：过程赋值、if、case、for、while、repeat、task和function等语句。

在进行时序逻辑电路的描述中，敏感信号表达式中经常使用“posedge”和“negedge”这两个关键字来声明事件是由输入信号的正边沿（上升沿）或负边沿（下降沿）触发的。如“always @(posedge clk) ”表示模块的事件是由clk的上升沿触发的；而“always @(negedge clk) ”表示模块的事件是由clk的下降沿触发的。

②initial语句

initial语句的语法格式为：

initial

begin

语句1；

语句2；

…；

end

initial语句的使用次数是不受限制的，但块内的语句仅执行一次，因此initial语句常用于仿真中的初始化。

③function语句

在Verilog HDL模块中，function语句用来定义函数。函数类似高级语言中的函数，用来单独完成某项具体操作，并可以作为表达式中的一个操作数，被模块或任务及其他函数调用，函数调用时返回一个用于表达式的值。

可以被调用的函数必须事先定义，函数定义格式如下：

function[最高有效位：最低有效位] 函数名；

端口声明语句；

类型声明语句；

begin

语句；

end

endfunction

在函数定义语句中，“[最高有效位：最低有效位]”是函数调用返回值位宽或类型声明。

函数调用的格式如下

函数名 （关联参数表）；

函数调用一般是出现在模块、任务或函数语句中。通过函数的调用来完成摹写数据的运算或转换。例如，调用上例中求最大值函数的语句为

peak<=max(data, peak);

其中，data和peak是与函数定义的两个参数a、b相关联的关联参数。通过函数的调用，求出data和peak中的最大值，并用函数名max返回。

# 四、不同抽象级别的Verilog HDL模型

Verilog HDL具有行为描述和结构描述功能。

行为描述是对设计电路的逻辑动能的描述，并不关心设计电路使用哪些元件及这些元件之间的连接关系。行为描述属于高层次的描述方法，在Verilog HDL中，行为描述包括系统级（System Level）、算法级（Algorithm Level）和寄存器传输级（RTL:Register Thansfer Level）等3种抽象级别。

结构描述是对设计电路的结构进行描述，既描述设计电路使用的元件及这些元件之间的连接关系。结构描述属于低层次的描述方法，在Verilog HDL，结构描述包括门级（Gate Level）和开关级（Switch Level）两种抽象级别。

应重点掌握高层次描述方法，结构描述也可以用来实现电路的系统设计。

## 4.1Verilog HDL门级描述方式

Verilog HDL提供了丰富的门类型关键字，用于门级的描述。常用的：

not非门、and与门、nand与非门、or或门、nor或非门、xor异或门、xnor同或门、buf缓冲器、及bufif1、bufif0、 notif1、notif0各种三态门。

门级描述语句格式为

门类型关键字 <例化门的名称> （端口列表）；

其中，“例化门的名称”是用户定义的标识符，属可选项；端口列表按：(输出、输入，使能控制端)的顺序列出。例如：

nand nand2 (y,a,b); //二输入端与非门

xor myxor(y,a,b) //异或门

bufif0 mybuf (y,a,en); //低电平使能的三态缓冲器

## 4.2Verilog HDL行为级描述

Verilog HDL行为描述方式是通过行为语句来描述电路要实现的功能，表示输入输出间转换的行为，不涉及具体结构。

行为建模是一种“高级”的描述方式。行为描述既可以描述简单的逻辑门，又可以描述复杂的数字系统，乃至微处理器。既可以描述组合逻辑电路，又可以描述时序逻辑电路。

例11 2选1数据选择器，用Verilog HDL对它做行为描述。

module mux\_2\_to\_1 (a, b, out,outbar,sel);

//这是一个2选1数据选择器，名为mux\_2\_to\_1

input a, b, sel; //定义模块的输入端口为a, b和sel

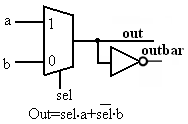
output out, outbar; //定义模块的输出端口为out和outbar

assign out=sel? a:b; //如果sel=1，将a赋值给out

//如果sel=0，将b赋值给out

assign outbar =~out; //将out取反后赋值给outbar

endmodule //模块描述结束



# 五、其他

## 1.组合逻辑和时序逻辑的区别

1.简单说，组合逻辑只跟当前的状态有关系，例如一根导线 + 逻辑门的组合，输出取决于当前时刻的输入，考虑门延时的话另说。

2.时序逻辑的状态跟之前状态、边沿信号有关，边沿信号没来之前，输出保持之前的输入；边沿信号来时，输出状态跟随输入信号变化。

组合逻辑：always@（敏感信号）或者always@（\*），组合逻辑相当于组合电路，与或非门组成的电路，其输出至于当前状态有关，与其他输入状态的函数无关，不涉及信号调变处理（组合逻辑竞争冒险：只要输入信号同时变化，组合逻辑必然产生毛刺）。

时序逻辑：always@（时钟边沿）是时序电路，输出不仅与当前状态有关，在时钟跳变时才变化，如D触发器。

## 2.和C语言相比较

C语言中的函数调用通过函数名相关联，函数之间的传值是通过端口变量实现的。相应地，Verilog HDL中地模块调用也通过模块名相关联，模块之间的联系同样通过端口之间的连接实现，所不同的是，它反映的是硬件之间的实际物理连接。

C语言中，整个程序的执行从main函数开始。每一个module模块都是等价的，但必定存在一个顶层模块，它的端口中包含了芯片系统与外界的所有I/O信号，这个顶层模块从程序的组织结构上讲，类似于C语言中的main函数，但Verilog HDL中所有module模块都是并发的，这一点必须从本质上与C语言加以区别。

## 3.作为一个助教的一些建议

大一的时候，课程就只讲了C语言，可能这是你们很多人第一次接触HDL（Hardware Description Language），可能一开始觉得有点困难。不过没关系，通过一学期的学习，会对对硬件描述语言有着清楚的认识。它和C语言在逻辑上有些不同，这可能是大家一开始学习的难点（当时我也是过了一段时间才理解），写Verilog时，千万不要完全按照C语言的思路写，这是Verilog中每个语句，每个模块在电路中都是有体现的，Verilog的程序，最后会变成实际的电路。所以写之前，一定要明确元件的输入和输出和它们之间逻辑的关系，电路图做到心中有数。

对于初学者来说，我还是推荐大家多看一些例子，看着例子理解其中的道理，这样会学的比较快。

如果发现文档有错误，可以跟我联系，我的邮箱是815556875@qq.com。

**最后祝大家新学期生活愉快~！**