Verilog HDL是一种用于数字逻辑电路设计的硬件描述语言，可以用来进行数字电路的仿真验证、时序分析、逻辑综合。

* 用Verilog HDL描述的电路设计就是该电路的Verilog HDL模型。
* Verilog HDL既是一种行为描述语言也是一种结构描述语言。（行为指可以用电路的功能描述，结构指可以用元器件及其之间的连接来建立Verilog HDL模型）

Verilog HDL的抽象级别可以分为五级：

* 系统级：用高级语言结构（如case语句）实现的设计模块外部性能的模型；
* 算法级：用高级语言结构实现的设计算法模型（写出逻辑表达式）；
* RTL级：描述数据在寄存器之间流动和如何处理这些数据的模型；
* 门级：描述逻辑门以及逻辑门之间连接的模型；
* 开关级：描述器件中三极管和储存节点及其之间连接的模型。

Verilog HDL基本结构

1. 简单的Verilog HDL例子（8位全加器）

Module adder8(cout, sum, a, b, cin); 端口定义

output cout; //输出端口声明

output[7:0] sum;

input[7:0] a,b; //输入端口声明

input cin;

assign{cout,sum}=a+b+cin; //功能描述

end module

assign语句：无论右边表达式操作数何时发生变化，右边表达式都会重新计算，并且在指定的延迟后给左边赋值。

注：整个程序嵌套在module 和 endmodule 声明语句中。

每条语句相对module和endmodule最好缩进2格或4格。

//......表示注释部分，一般只占据一行。对编译不起作用。

8位计数器：  
module counter8(out,cout,data,load,cin,clk); 端口定义

output[7:0] out;

output cout;

input[7:0] data;

input load,cin,clk;

reg[7:0] out; 信号类型声明

always @(posedge clk)

begin

if(load)

out<=data; //同步预置数据

else

out<=out+1+cin; //加1计数

end

assign cout = &out &cin; //若out为8^hFF, cin为1，则cout为1

end module

位运算符

缩减运算符

2位比较器：  
module compare2(equal,a,b);

output equal;

input[1:0]a,b;

assign equal=(a==b)?1:0;

/\*如果a等于b，则equal为1，否则为0\*/

Endmodule

“<=”非阻塞过程性赋值，将想要赋给左式的值安排在未来时刻，不等上一个赋值结束就执行下个赋值语句。

“=”阻塞过程性赋值，按照顺序执行，前一个赋值结束才执行下一个赋值。

1. 门元件例化

例 三态驱动器

module trist2(out,in,enable);

output out;

input in,enable;

bufifl mybuf(out,in,enable);

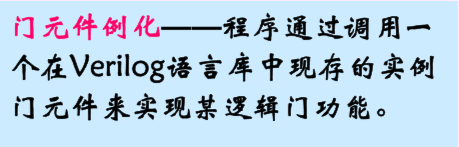
endmodule

例化元件名

表1 bufifl真值表

门元件关键字

|  |  |
| --- | --- |
| Inputs | Output |
| IN ENABLE | OUT |
| X 0   1. 1 2. 1 | Z  1  0 |



1. 模块元件例化

顶层模块（trist1）调用由某子模块（mytri）定义的实例元件（tri\_inst）来实现某功能。

之间的关系如图1:，

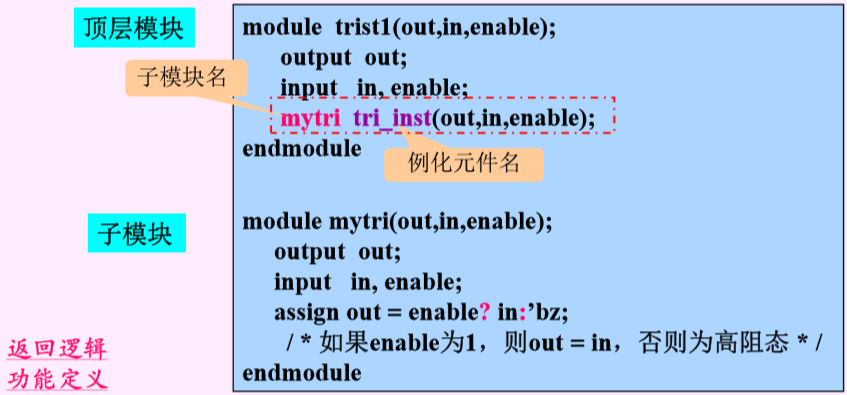


图1 模块元件例化示意

Verilog HDL基本结构总结：  
1.Verilog HDL程序是由模块构成的。每个模块嵌套在module和endmodule声明语句中。模块是可以进行层次嵌套的。

2.每个Verilog HDL源文件中只准有一个顶层模块，其他为子模块。

3.每个模块要进行端口定义，并说明输入输出端口，然后对模块的功能进行行为逻辑描述。

4.程序书写格式自由，一行可以写几个语句，一个语句也可以分行写。

5.除了endmodule语句、begin\_end语句和fork\_join语句外，每个语句和数据定义的最后必须有分号。

6.可用/\*......\*/和//...对程序的任何部分作注释。加上必要的注释，以增强程序的可读性和可维护性。

1. Verilog HDL模块的结构

Verilog 的基本设计单元是“模块”。

Verilog 模块的结构由在module和endmodule关键词之间的4个主要部分组成：

端口定义 module block1(a,b,c,d);

I/0说明 Input a,b,c;

Output d;

信号类型说明 Wire x;

功能描述 Assign d = a | x;

Assign x = (b & ~c);

endmodule

1. 逻辑功能定义

有3种方法描述电路的逻辑功能：  
（1）用assign语句

assign x = (b & ~c); 连续赋值语句，常用于描述组合逻辑

（2）用元件例化（instantiate）

and myand3(f,a,b,c); 门元件例化

注：元件例化包括门元件例化和模块元件例化

每个实例元件的名字必须唯一！以避免与其他调用元件的实例相混淆。

（3）用always块语句（结构说明语句）

always @(posedge clk) //每当时钟上升沿到来时执行一遍块内语句

begin

if(load)

out = data; //同步预置数据

else

out = data + 1 + cin; //加1计数

end

注：“always”块语句与assign语句是并发执行的，assign语句一定要放在“always”块语句之外。

Verilog HDL模块的模板（仅考虑用于逻辑综合的部分）见图2

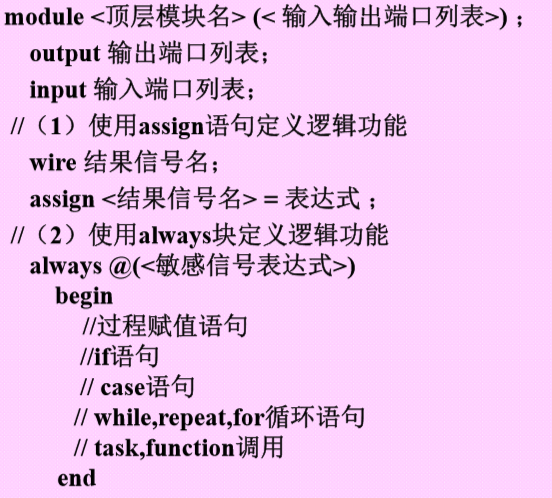


图2 Verilog HDL模块的模板

1. 关键字

用小写字母定义，如图3



图3

1. 标识符

标识符可以由字母、数字、下划线和$符号构成，但第一个字符必须是字母或下划线，不能是数字或$符号！

变量名区分大小写！

1. 代码编写标准
2. 每个源文件中只准编写一个顶层模块，不能把一个顶层模块分成几部分写在几个源文件中。
3. 源文件名字应与文件内容有关，最好与顶层模块同名
4. 每行只写一个声明语句或说明。
5. 源代码用层层缩进的格式来写。
6. 定义变量名的大小写应始终保持一致
7. 通过注释对源代码做必要的说明。
8. 常量尽可能多地使用参数定义和宏定义，而不要在语句中直接使用字母、数字和字符串。（注：参数定义格式：parameter 参数名1=表达式, 参数名2=表达式, .....;

宏定义格式：’define 标识符（宏名）字符串（内容））

注意事项：  
（1）避免生成不想要的触发器

在时钟沿触发的always块中，如果用非阻塞赋值语句对reg型变量赋值，或者当reg型变量经过多次循环其值仍保持不变，则会在综合中生成触发器。

用reg型变量生成触发器例子：  
module rw2(clk, d, out1);

input clk,d;

output out1;

reg out1;

always@(posedge clk)

out1<=d;

endmodule

若不想生成触发器，而是希望用reg型变量生成组合逻辑，则应使用电平触发：  
module rw2(clk,d,out1);

input clk,d;

output out1;

reg out1;

always@(d)

out1<=d;

endmodule

（2）一般情况下，在赋值语句中不能使用延迟，否则是不可综合的。

（3）不要使用integer型和time型寄存器。

九、数据类型

Verilog HDL中共有19种数据类型；

其中4个最基本的数据类型为：  
integer、parameter、reg、wire

常量

（1）整数型常量的4种进制表示形式：  
二进制整数（b或B）；

十进制整数（d或D）；

十六进制整数（h或H）；

八进制整数（o或O）

整数常量的3种表达方式，见图 4



图 4

（2）x和z值

X表示不定值，z表示高阻值

当用二进制表示时，已标明位宽的数若用x或z表示某些位，则只有在最左边的x或z具有扩展性！例：  
8’bzx = 8’bzzzz\_zzzx

8’b1x = 8’b0000\_001x

“？”是z的另一种表示符号，建议在case语句中使用?表示高阻态z

（3）负数

在位宽前加一个减号，即表示负数，如：-8’d5表示十进制数5的补码

为提高程序的可读性，在较长的数字之间用下划线\_隔开，但不可以用在进制和数字之间。

（4）parameter常量（符号常量）

用parameter来定义一个标识符，代表一个常量——称为符号常量。

每个赋值语句的右边必须为常数表达式，即只能包含数字或先前定义过的符号常量

parameter addrwidth = 16;

常用参数来定义延迟时间和变量宽度。

可用字符串表示的任何地方，都可以用定义的参数来代替。

参数是本地的，其定义只在本模块内有效。

模块实例引用时参数的传递：

1. 利用defparam定义参数声明语句！

格式 defparam 例化模块名.参数名1 = 常数表达式,

例化模块名.参数名2 = 常数表达式, ...;

defparam语句在编译时可重新定义参数值。

一般不使用defparam语句！在模块的实例引用时可用“#”后跟参数的语法来重新定义参数。

1. 利用特殊符号“#”

格式 被引用模块名#(参数1,参数2, ...)例化模块名（端口列表）;

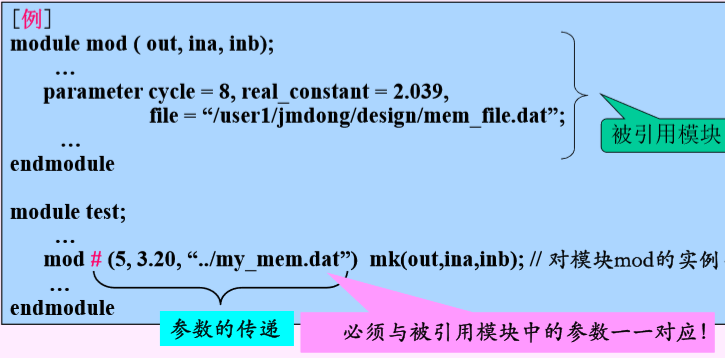


图5

变量

数据类型有19种，常用的有3种：  
网络型（nets type）、寄存器型（register type）、数组（memory type）

1.nets型变量——输出始终随输入变化而变化的变量。

表示结构实体之间的物理连接。

常用nets型变量：  
wire, tri：连线类型

wor, trior：具有线或特性的连线

wand, triand：具有线与特性的连线

supply1, supply0：电源（逻辑1）和地（逻辑0）

wire型变量

最常用的nets型变量，常用来表示以assign语句赋值的组合逻辑信号。

模块中的输入/输出信号类型缺省为wire型。

可用做任何方程式的输入，或“assign”语句和实例元件的输出

格式 wire 数据名1，数据名2，......，数据名n;

wire型向量（总线）

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;（总线位宽为n，共有m条总线）

2.register型变量

对应具有状态保持作用的电路元件（如触发器，寄存器等），常用来表示过程块语句（如initial, always, task, function）内的指定信号。

register型变量与nets型变量的根本区别是：register型变量需要被明确地赋值，并且在被重新赋值前一直保持原值。Register型变量必须通过过程赋值语句进行赋值！不能通过assign语句赋值！在过程块内被赋值的每个信号必须定义成register型！

例 用reg型变量生成组合逻辑

module rw1(a,b,out1,out2);

input a,b;

output out1,out2;

reg out1;

wire out2;

assign out2 = a;

always@(b)

out1 <= ~b;

endmodule

3.memory型变量

Verilog HDL通过reg型变量建立数组来对存储器建模，memory型变量可以描述ram, rom和reg文件。

reg型变量与memory型变量的区别

1. 含义不同 reg[n-1:0] rega; //一个n位的寄存器

reg mema[n-1:0]; //由n个1位寄存器组成的存储器

1. 赋值方式不同

一个n位的寄存器可用一条赋值语句赋值，一个完整的存储器则不行！若要对某存储器中的存储单元进行读写操作，必须指明该单元在存储器中的地址！

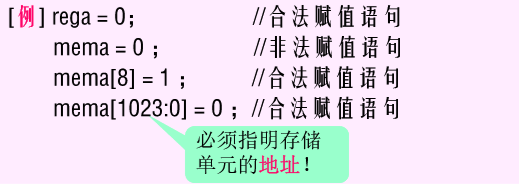


图6

1. 语句

赋值语句：  
（1）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。

（2）过程赋值语句——用于对reg型变量赋值，有两种方式：  
非阻塞赋值方式：赋值符号为<=，如 b <= a；

阻塞赋值方式：赋值符号为=，如 b = a

块语句：  
用来将两条或多条语句组合在一起，使其在格式上更像一条语句，以增加程序的可读性。块语句有两种：  
begin\_end语句——标识顺序执行的语句

fork\_join语句——标识并行执行的语句

1. 顺序块（用begin\_end标识的块）

块内的语句顺序执行

每条语句延迟时间相对于前一条语句的仿真时间而言

直到最后一条语句执行完，程序流程控制才跳出该顺序块

例：begin

b = a;

#10 c = b; //在两条赋值语句之间延迟10个时间单位

end

1. 并行块

块内的语句同时执行

块内的每条语句延迟时间是相对于程序流程控制进入到块内的仿真时间而言的

延迟时间用于给赋值语句提供时序

当按时间排序在最后的语句执行完或一个disable语句执行时，程序流程控制跳出该并行块

条件语句：

1.if-else语句和case语句，放在“always”块内

if-else语句有3种形式：

其中表达式为逻辑表达式或关系表达式，或1位的变量

语句可为单句，也可以为多句；多句时要用“begin\_end”语句括起来，形成一个复合块语句

if语句可以嵌套;

若if与else数目不一样，注意用“begin\_end”语句来确定if 与 else的配对关系！

注意：  
if(reset)

else if(load)

else if(cin)

不要写成3个并列的if语句：  
if(reset)

If(load)

If(cin)

2.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

default项可有可无，一个case语句里只能有一个default项！

值1~值n必须互不相同，否则矛盾。

值1~值n的位宽必须相等，且与控制表达式的位宽相同。

1. 注意事项

应注意列出所有条件分支，否则当条件不满足时，编译器会生成一个锁存器保持原值！

在组合电路设计中，应避免生成锁存器！有效的方法是在if语句最后写上else项；在case语句最后写上default项。

循环语句

1. for语句——通过执行3个步骤来决定语句的循环执行；
2. 给控制循环次数的变量赋初值。
3. 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。
4. 修改循环变量的值，返回第（2）步。
5. repeat语句——连续执行一条语句n次。
6. while语句——执行一条语句，直到循环执行条件不满足
7. forever语句——无限连续地执行语句，可用disable语句中断！

repeat语句格式：repeat（循环次数表达式）语句

forever语句格式： forever 语句

无条件连续执行后面的语句或语句块。（常用在测试模块中产生周期性的波形，作为激励信号；常用disable语句跳出循环；不同与always语句，不能独立写在程序中，一般用在initial语句块中！）

结构说明语句

initial说明语句——只执行一次

always说明语句——不断重复执行，直到仿真结束

task说明语句——可在程序模块中的一处或多处调用

function说明语句——可在程序模块中的一处或多处调用

1.always语句：

包含一个或一个以上的声明语句，在运行全过程中，在定时控制下被反复执行。  
在always语句块中被赋值的只能是register型变量（如reg, integer, real, time）。

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。（注：1.如果always块中包含了一个以上的语句，则这些语句必须放在begin\_end或fork\_join块中！2.always语句必须与一定的时序控制结合在一起才有用，如果没有时序控制，则易形成仿真死锁！）

always语句模板：

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

在敏感信号表达式中应列出影响块内信号取值的所有信号！

敏感信号可以为单个信号，也可为多个信号，中间需用关键字or连接！

敏感信号不能为x或z，否则会阻挡进程！

当always块有多个敏感信号时，一定要采用if-else if语句

通常采用异步清零！

2.initial语句

利用initial语句来生成激励波形，如图7

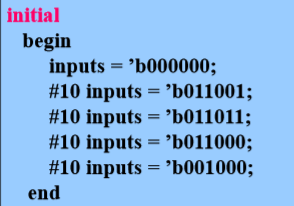


图7

在仿真的初始状态对各变量进行初始化；

在测试文件中生成激励波形作为电路的仿真信号。

3.task和function语句

task和function语句分别用来由用户定义任务和函数，利用任务和函数可将一个很大的程序模块分解为许多较小的任务和函数，便于理解和调试。

输入、输出和总线信号的值可以传入、传出任务和函数。

task:当希望能够对一些信号进行一些运算并输出多个结果时，宜采用任务结构。常常利用任务来帮助实现结构化的模块设计，将批量的操作以任务的形式独立出来，使设计简单明了。

task<任务名>

端口及数据类型声明语句;

其他语句;

endtask

<任务名>(端口1, 端口2, ...);

注：任务的定义与调用必须在一个module模块内；任务被调用时，需列出端口名列表，且必须与任务定义中的IO变量一一对应；一个任务可以调用其他任务和函数！

我们通过图8了解任务的定义和调用方法，

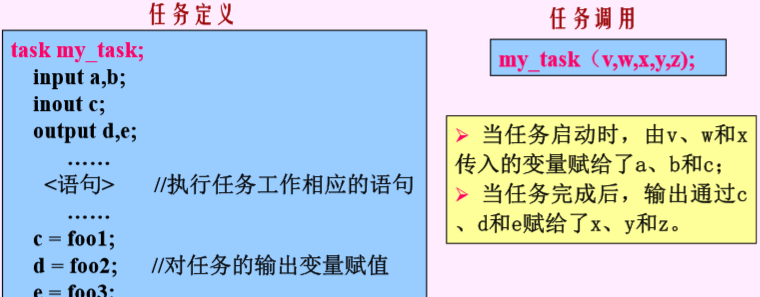


图8

函数（function）

函数的目的是通过返回一个用于某表达式的值，来响应输入信号。适于对不同变量采用同一运算的操作。

函数在模块内部定义，通常在本模块中调用，也能根据按模块层次分级命名的函数名从其他模块调用。而任务只能在同一模块内定义与调用。

函数的调用是通过将函数作为调用函数的表达式中的操作数来实现的！

函数在调用时被理解为具有独立运算功能的电路，每调用一次函数，相当于改变此电路的输入，输出一次运算值。

注：  
1.函数的定义不能包含任何时间控制语句——用延迟#、事件控制@或等待wait标识的语句！

2.函数不能调用任务。

3.定义函数时至少要有一个输入参量，且不能有任何输出/输入输出双向变量！

4.函数定义中必须有一条赋值语句，给函数中的一个内部寄存器赋以函数的结果值，该内部寄存器与函数同名。

例 用函数对一个8位二进制数中为0的位进行计数

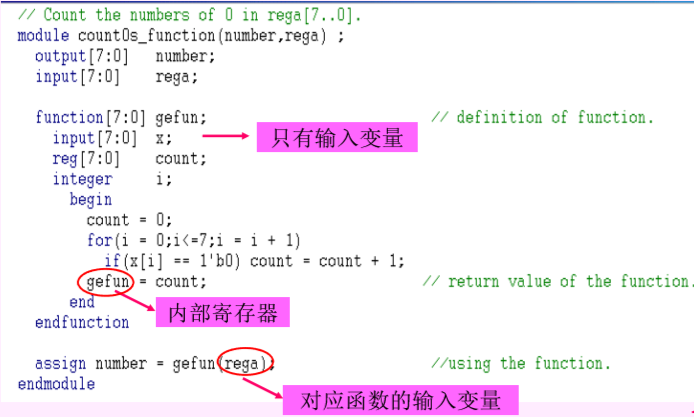


图9

运行结果如图10



图10

Verilog语言中任务和函数的区别，见表2

|  |  |  |
| --- | --- | --- |
|  | 任务（task） | 函数（function） |
| 目的或用途 | 可计算多个结果值 | 通过返回一个值，来响应输入信号 |
| 输入与输出 | 可为各种类型（包括inout型） | 至少有一个输入变量，但不能有任何output或inout型变量 |
| 被调用 | 只可在过程赋值语句中调用，不能在连续赋值语句中调用 | 可作为表达式中的一个操作数来调用，在过程赋值和连续赋值语句中均可调用 |
| 调用其他任务和函数 | 任务可以调用其他任务和函数 | 函数可调用其他函数，但不能调用任务 |
| 返回值 | 不向表达式返回值 | 向调用它的表达式返回一个值 |

编译预处理语句

`define语句/`include语句/`timescale语句

编译预处理语句以西文符号“ ` ”开头，注意不是单引号！

1. 宏定义语句——`define 标识符（宏名）字符串（宏内容）

宏展开——在编译预处理时将宏名替换为字符串的过程。

`define语句可以写在模块定义的里面或外面。有效范围为定义命令之后到源文件结束。

在引用已定义的宏名时，必须在其前面加上符号“ ` ”

1. 文件包含语句——`include “文件名”

可将多个`include语句写在一行，在该行中，只可出现空格和注释行

1. 时间尺度语句——`timescale <时间单位>/ <时间精度>

时间单位——用于定义模块中仿真时间和延迟时间的基准单位；

时间精度——用来声明该模块的仿真时间和延迟时间的精确程度。