Verilog 模块模板

module <顶层模块名> （<输入输出端口列表>）;

input 输入端口列表; //输入端口声明

output 输出端口列表; //输出端口声明

/\*定义数据，信号的类型，函数声明，用关键字wire、reg、task、function等定义\*/

wire 信号名

reg 信号名

//逻辑功能定义

assign <结果信号名>=<表达式>; //使用assign语句定义逻辑功能

//用always块描述逻辑功能

always @(<敏感信号表达式>)

begin

//过程赋值

//if-else,case语句，for循环语句

//task，function调用

end

//调用其它模块

<调用模块名> <例化模块名> (<端口列表>);

//门元件例化

门元件关键字<例化元件名> (<端口列表>);

endmodule

数值表示

整数

二进制：b或B

十进制：d或D或默认

十六进制：h或H

八进制：o或O

+/-<位宽><进制><数字>

没有定位位宽则默认32位

例如：

8'b11000101

8'hd5

5'o27

4'D2

reg

Verilog使用reg存储字符串

例如：

reg[8\*12,1] stringvar

initial

begin

stringvar="hello world!";

end

特殊符号

\n换行 \t制表 \\符号\ \”符号” \ddd八进制ddd对应符号

数据类型

四种逻辑状态（四值逻辑）

0：低电平

1：高电平

z：高阻态

x：不确定或未知逻辑状态

——可综合设计中，只有端口变量可赋值z，因为仅在FPGA的IO引脚中可物理实现高阻态

两种数据类型

net型：wire、tri等，相当于各种物理连接，特点是输出值紧随输入值变化而变化。

net型数据的值取决于驱动的值，如果没有连接到驱动，则为高组态z，两种驱动方式：

1.在结构描述中将其连接到一个门元件或模块的输出端

2.用持续赋值语句assign对其进行赋值

wire：最常见，模块的输入输出信号没有明确指定数据类型时都为wire型，如果wire型没有连接到驱动则为高阻态z。

tri：功能及使用方法与wire型完全一样，对于综合器来说tri和wire型完全相同

variable型：必须放在过程语句（initial、always等）中，通过过程赋值语句赋值

在initial、always等过程块内被赋值的信号也必须定义成variable型

reg：常用的寄存器型变量，不一定对应硬件的寄存器或触发器，综合器会根据情况将其映射为寄存器或连线

module abc(input a,b,c

output f1,f2);

reg f1,f2;//在always过程块中赋值的变量需定义为reg型

always @(a or b or c)

begin

f1=a|b;//f1、f2综合时不会映射为寄存器

f2=f1^c;

end

endmodule

integer：32位有符号整型变量，多用于表示循环变量，如循环次数。不能作为位向量访问，综合时，integer变量初始值为x。

integer i,j;

integer[31:0] d;

real：64位有符号整型变量

time：64位无符号整型变量

参数

parameter：定义符号常量，常用来定义延时和变量的宽度，只能被赋值一次。

参数名通常用大写字母表示：

parameter SEL=8,CODE=8’ha3

parameter还具有参数传递（重载）的功能：1.用“#”号隐式地重载；2.在线显式重载参数方式；3.使用defparam语句显式地重载

localparam

局部参数，只能在本模块中使用，不可用于参数传递

//采用localparam的加法器

module add\_localp

#(parameter MSB=15,LSB=0)//parameter参数定义

(input[MSB:LSB] a,b,

output[MSB:LSB] sum);

localparam HSB=MSB+1;//localparam参数定义

assign sum=a+b;

endmodule

标量：宽度为1位的变量

wire a;

reg clk;

向量：线宽大于1位的变量

[MSB:LSB]

左边的数字表示向量最高有效位（Most Significant Bit, MSB），右边的表示向量最低有效位（Least Significant Bit, LSB）

wire[3:0] bus;//4位总线

reg[7:0] ra,rb;//定义了两个8位寄存器，其中ra[7],rb[7]分别为最高有效位

reg[0:7] rc;//rc[0]为最高有效位，rc[7]为最低有效位

位选择和域选择

表达式中选择向量中的一位或相邻几位

A=mybyte[6];//将mybyte的第6位赋值给变量A，位选择

B=mybyte[5:2];//将mybyte的第5、4、3、2位的值赋给变量B，域选择

reg[7:0] a,b;reg[3:0] c;reg d;

d=a[7]&b[7];//位选择

c=a[7:4]+b[3:0];//域选择

使用时，运算符左右两端宽度应一致

向量分标量类向量和向量类向量，默认为标量类向量

标量类向量支持位选择和域选择，用scalared关键字说明

向量类向量不支持位选择和域选择，只能作为整体操作，用vectored关键字说明

reg scalared [31:0] rega;//rega为32位标量类向量

wire vectored [7:0] databus;//向量类向量

存储器

可视为二维向量，由一组宽度相同的寄存器构成的阵列

reg[7:0] mymem[63:0];

//定义了一个有64个单元的存储器，每个单元宽度为8b，存储器名叫mymem

reg[3:0] Amem[63:0];//Amem是容量为64、字长为4位的存储器

reg Bmem[5:1];//Bmem是容量为5、字长为1位的存储器

parameter WIDTH=8,MEMSIZE=1024;

reg[WIDTH-1:0] mymem[MEMSIZE-1:0];

//用parameter参数定义存储器的尺寸

//定义了一个宽度为8b、容量为1024个存储单元的存储器

存储器赋值：只能对某一单元整体赋值

reg[7:0] mymem[63:0];//存储器的定义

mymem[8]=8'b10001001;//mymem存储器第8个单元被赋值为二进制数10001001

mymem[25]=65;//mymem存储器第25个单元被赋值为十进制数65

与寄存器的区别

reg[1:8] rega;//定义了一个8位的寄存器

reg mema[1:8];//定义了一个字长为1、容量为8的存储器

rega[2]=1'b1;//对寄存器rega的第2位赋值1，合法

mema[2]=1'b1;//对存储器mema的第2个单元赋值1，合法

rega=8'b01011000;//对寄存器rega整体赋值，合法

mema=8'b01011000;//不允许对存储器的多个或所有单元一次性赋值，非法

运算符

算术运算符：+ - \* / %

逻辑运算符：&& || !

位运算符：~ & | ^：异或 ^~或~^：同或

关系运算符：< <= > >=

等式运算符：== != ===：全等 !==：不全等

缩减运算符：& ~&与非 | ~|或非 ^ ^~,~^

移位运算符：>> << >>>算术右移 <<<算术左移

指数运算符：\*\*\*\*\*\* 如2的n次幂为2\*\*\*\*\*\*n

条件运算符：?:

位拼接运算符：{}拼接两个或多个信号

加法运算将进位与和拼接：

input[3:0] ina,inb;input cin;

output[3:0] sum;output cout;

assign {cout,sum}=ina+inb+cin;//进位与和拼接在一起

嵌套使用&复制法：

{3{a,b}}//复制3次，等价于{{a,b}{a,b}{a,b}}或{a,b,a,b,a,b}

{2{3'b101}}//复制2次，结果为101101

符号位扩展：

wire[7:0] data;

wire[11:0] s\_data;

s\_data={4{data[7]},data};//将data的符号位扩展

运算符优先级：



过程语句

一个模块中的过程语句次数不受限制，initial语句常用语执行仿真中的初始化，只执行一次；always语句则不断执行

always @(<敏感信号列表>)

begin

//过程赋值

//选择语句

//循环

//调用

//等等

end