**Verilog中Generate语句的使用**

WARNING: for and while loop statements are valid in Verilog (and can be simulated), but cannot always be mapped to hardware. Generate statements are the appropriate use for for loops.

设计中有时需要编写很多结构相同但是参数不同的赋值语句或者逻辑语句。C语言中常用for语句来解决此类问题，Verilog-2001则为我们提供了generate语句。generate语句有generate\_for、generate\_if、generate\_case三种语句。

generate语句格式如下：

genvar 循环变量名；

generate

// generate循环语句

// generate 条件语句

// generate 分支语句

// 嵌套的generate语句

endgenerate

1、generate\_for语句

（1）必须使用genvar申明一个正整数变量，用作for循环的判断。

（2）需要复制的语句必须写到begin\_end语句里面，即使只有一句！

（3）begin\_end需要有一个类似于模块名的名字。

下面举例进行说明：

例1:

module buffer\_1 (

input wire in,

output wire out

);

assign out = ~in;

endmodule

//8bit width buffer

module buffer\_8 (

input wire [7:0] din,

output wire [7:0] dout

);

//Generate block

genvar i;

generate

for(i = 0; i < 8; i = i + 1) begin: BLOCK

buffer\_1 buffer\_1\_inst (.in(din[i]), .out(dout[i]));

end

endgenerate

endmodule

在buffer\_8中例化buffer\_1 8次，这里有几点需要注意：

① 循环变量i必须是genvar类型的，不可以是reg型，integer型；

② for循环之后的的begin最好加上一个标号（如BLOCK1，例化模块的名字）；

例2:

module nbit\_xor #(

parameter SIZE=16

)(

input(SIZE-1:0) a,b,output[SIZE-1:0] y

);

genvar gv\_i;

generate

for(gv\_i=0;gv\_i<SIZE;gv\_i++)

begin:sblka

xor uxor(y[gv\_i],a[gv\_i],b[gv\_i]);

end

endgenerate

endmodule

for循环中使用的循环变量gv\_i被称为genvar变量，这种变量必须用genvar来声明，并且只能在generate循环语句中使用；此外，generate块需要标签，用来表示循环的实例化名称，在上例中是sblka.



例3:

module test (bin, gray);

       parameter SIZE=8;

       output [SIZE-1:0] bin;

       input [SIZE-1:0] gray;

       genvar i; //genvar i;也可以定义到generate语句里面

       generate

              for(i=0;i<SIZE;i=i+1)

              begin: bit

                     assign bin[i]=^gray[SIZE-1:i];

              end

       endgenerate

endmodule

等同于下面语句：

assign bin[0]=^gray[SIZE-1:0];

assign bin[1]=^gray[SIZE-1:1];

assign bin[2]=^gray[SIZE-1:2];

assign bin[3]=^gray[SIZE-1:3];

assign bin[4]=^gray[SIZE-1:4];

assign bin[5]=^gray[SIZE-1:5];

assign bin[6]=^gray[SIZE-1:6];

assign bin[7]=^gray[SIZE-1:7];

例4:

generate

       genvar i;

       for(i=0;i<SIZE;i=i+1)

       begin:shifter

              always@(posedge clk)

                     shifter[i]<=(i==0)?din:shifter[i-1];

       end

endgenerate

相当于：

always@(posedge clk)

       shifter[0]<=din;

always@(posedge clk)

       shifter[1]<=shifter[0];

always@(posedge clk)

       shifter[2]<=shifter[1];

.................

       ......................

always@(posedge clk)

       shifter[SIZE]<=shifter[SIZE-1];

2. generate-conditional条件语句

generate条件语句最常见的格式如下：

if(condition)

statements

else

statements

condition必须是一个静态的条件，即在细化（Elaborated）期间计算得出。statements可以是任何能够在模块中出现的语句，例如always语句。注意，由于条件的值可能取决于从上层模块中传递过来的参数，因此条件的值可能不能再细化期间被完全算出。

例1：

module adder

#(parameter SIZE=4)

(input[SIZE-1:0] a,b,

output[SIZE-1:0] sum,

output carry\_out);

wire [SIZE-1:0] carry;

genvar gv\_k;

generate

for(gv\_k = 0; gv\_k < SIZE; gv\_k ++)

begin: gen\_blk\_adder

if(gv\_k == 0)

half\_adder u\_ha (

.a(a[gv\_k]),

.b(b[gv\_k]),

.sum(sum[gv\_k]),

.carry\_out(carry[gv\_k]),

);

else

full\_adder u\_ha (

.a(a[gv\_k]),

.b(b[gv\_k]),

.sum(sum[gv\_k]),

.carry\_in(carry[gv\_k-1]),

.carry\_out(carry[gv\_k]),

);

end

endgenerate

endmodule

例2：



3. generate-case分支语句

generate分支语句与条件语句类似，只不过分支语句是用分支来进行条件选择。

例1：

module adder #(

parameter SIZE=4

parameter IMPLEMENTATION\_LEVEL=0

)(

input[SIZE-1:0] arg1,arg2,

output[SIZE-1:0] result,

);

generate

case(IMPLEMENTATION\_LEVEL)

0: assign result=arg1+arg2;

1:.....;

2:.....;

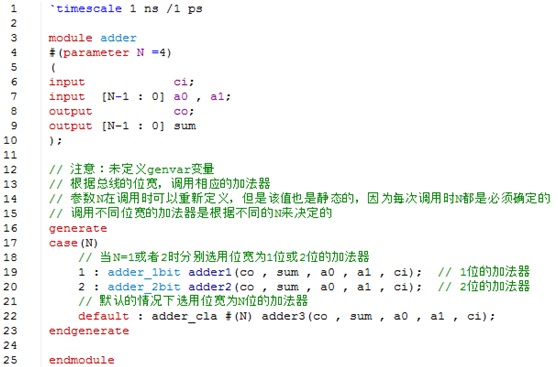
3:.....;

default:......;

endgenerate

endmodule

例2：



转载自： http://xilinx.eetrend.com/blog/2019/100045571.html

这篇帖子也介绍的很好： https://www.cnblogs.com/nanoty/archive/2012/11/13/2768933.html