Verilog HDL

硬件描述语言

参考书

[1].(美)J.Bhasker 著 徐振林 等译，Verilog HDL 硬件描述语言，机械工业出版社

[2].王金明 杨吉斌 编著，数字系统设计与Verilog HDL,电子工业出版社

[3].夏宇闻 编著,从算法设计到硬线逻辑的实现-*复杂数字逻辑系统的Verilog HDL设计技术和方法*,高等教学出版社

引言

Verilog HDL是一种用于数字逻辑电路设计的硬件描述语言,可用于从算法级、门级到开关级的多种抽象层次的数字系统设计。

Verilog HDL语言最初是于1983年由Gateway Design Automation公司开发的逻辑模拟器——Verilog-XL及其硬件描述语言。1989年，该公司被Cadence公司收购。1990年， Cadence公司公开发表了Verilog HDL语言。由于这种语言具有简捷、高效、易学易用、功能强等优点，因此逐渐为众多设计者所接受和喜爱。

引言

Verilog HDL语言于1995年成为IEEE标准，称为IEEE Standard 1364-1995。

从语法结构上看，Verilog HDL语言与C语言有许多相似之处，并继承和借鉴了C语言的多种操作符和语法结构。它具有以下一些主要特点：

- 能形式化地表示电路的结构和行为。
- 借用高级语言的结构和语句，例如条件语句、赋值语句和循环语句等，在Verilog HDL中都可以使用，既简化了电路的描述，又方便了设计人员的学习和使用。

引言

- 能够在多个层次上对所设计的系统加以描述，从开关级、门级、寄存器级(RTL)到功能级和系统级，都可以描述。设计规模可以是任意的，语言不对设计的规模施加任何限制。
- Verilog HDL具有混合建模能力，即在一个设计中各个模块可以在不同设计层次上建模和描述。
- 基本逻辑门，例如and、or和nand等都内置在语言中；开关级结构模型，例如pmos和nmos等也内置在语言中，用户可以直接调用。

引言

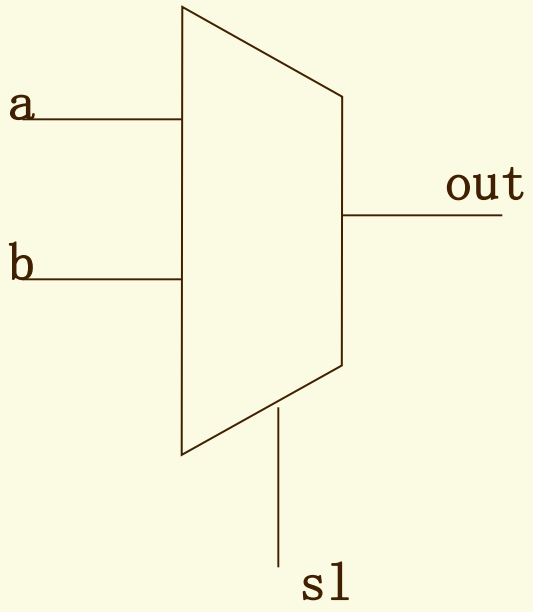
- 用户定义原语(UDP)创建的灵活性。用户定义的原语既可以是组合逻辑原语，也可以是时序逻辑原语。Verilog HDL还具有内置逻辑函数。

  Verilog HDL语言最大的特点是易学易用，通过学习和使用，可以在短时间内掌握该语言。另外，该语言功能强，可以满足各个层次设计人员的需要，从高层的系统描述到低层的版图设计，都能很好地支持。由于Verilog HDL巨大的优越性，使得它广泛流行，尤其是在ASIC设计领域，更是处于主流地位。

1. Verilog模块的基本概念

例1：

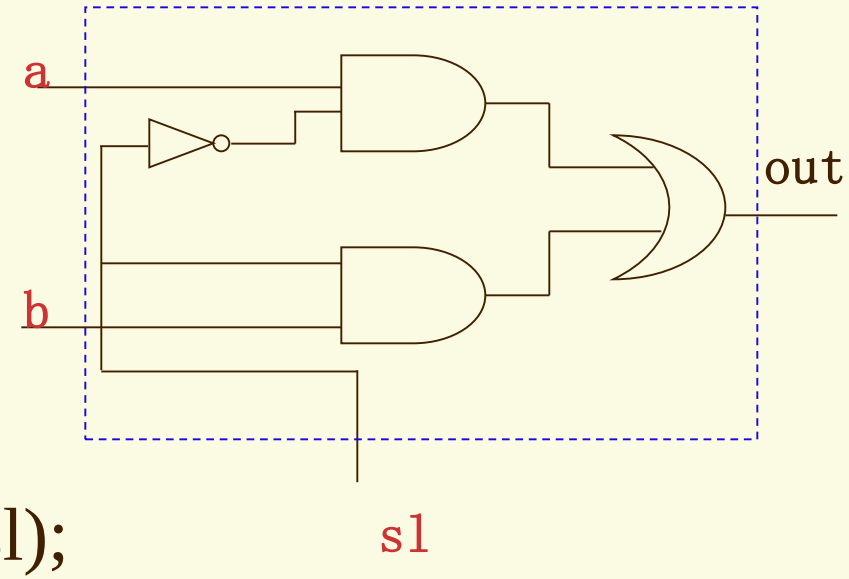
```
module muxtwo(out,a,b)
      input a,b,sl;
       output out;
        reg out;
              always @(sl or a or b)
                  if(!sl)   out=a;
                  else out=b;
endmodule
```



逻辑行为的描述

例2：

```
module muxtwo(out,a,b)
      input a,b,sl;
      output out;
       reg out;
       not       u1(nsl,sl);
      and   #1  u2(sela,a,nsl);
      and   #1  u3(selb,b,sl);
      or     #2  u4(out,slea,selb);
endmodule
```



基于逻辑单元互连结构的描述

2. Verilog HDL基本结构

1. 简单的 Verilog HDL例子

(1)一个4位全加器的Verilog HDL源代码：

```
module  adder4 ( cout,sum,a,b,cin );
        output  cout;
        output [3:0] sum;
        input [3:0]  a,b;
        input   cin;
        assign {cout,sum}=a+b+cin;
endmodule
```

简单的 Verilog HDL例子

(2)一个比较器的Verilog HDL源代码：

```
module   compare ( equal,a,b );
   output  equal;     //声明输出信号equal
   input [1:0] a,b;    //声明输入信号a,b
   assign  equal=(a==b)? 1:0;
    /*如果两个输入信号相等,输出为1,否则为0*/
endmodule
```

简单的 Verilog HDL例子

(3)一个8位计数器的Verilog HDL源代码：

```
module   counter8(out,cout,data,load,clk);
    output[7:0]  out;
    output  cout;
    input[7:0]   data;
    input  load, clk;
    reg[7:0]  out;
    always @(posedge clk)
      begin
        if  (load)  out=data;
        else  out =out+1;
      end
    assign  cout=&out;
endmodule
```

从上面的例子可以看出:

1. Verilog HDL程序是由模块构成的。每个模块的内容都是嵌在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,并说明输入(input)和输出(output),然后对模块的功能进行逻辑描述。
3. Verilog HDL程序的书写格式自由,一行可以写几个语句,一个语句也可以分多行写。
4. 除了endmodule语句和begin/end语句外,每个语句和数据定义的最后必须有分号“；”。
5. 可以用/*.....*/和//.....对Verilog HDL程序的任何部分作注释。(一个完整的源程序都应当加上必要的注释,以增强程序的可读性和可维护性。)

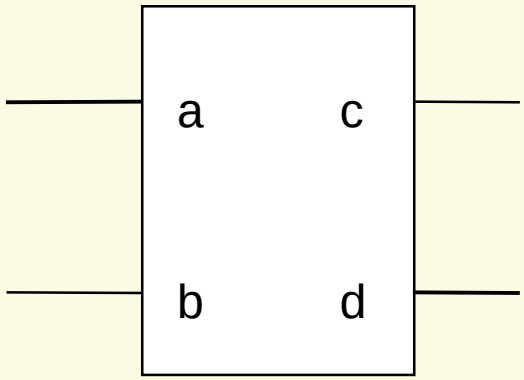
Verilog 是大小写敏感的(即区分大小写)。所有的Verilog 关键词都是小写的。

Verilog HDL模块的结构

Verilog的基本设计单元是“模块”(block)。一个模块是由两部分组成的：一）描述接口；二）描述逻辑功能，即定义输入是如何影响输出的。下面举例说明：

```
module  block (a,b,c,d);
input a,b;
output c,d;

assign c=a | b ;
assign d=a & b;
endmodule
```



Verilog HDL模块的结构

从上面的例子可以看出：

- Verilog模块结构完全嵌在module和endmodule声明语句之间；

- 每个Verilog程序包括四个主要部分：端口定义、I/O说明、信号类型声明、功能描述。

1. 模块的端口定义

模块的端口声明了模块的输入和输出口。其格式如下：

module 模块名(口1,口2,口3,……);

2. 模块内容

模块内容包括I/O说明，信号类型声明和功能定义。

(1)I/O说明的格式如下：

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;

I/O说明也可以写在端口声明语句里。其格式如下：

module module-name（input port1，input port2，..... output port1，output port2，.....）；

⑵信号类型声明：它是说明逻辑描述中所用信号的数据类型及函数声明。如前面计数器模块中：

reg[7:0]out；//定义out的数据类型为reg(寄存器)型

对于端口信号的缺省定义类型为wire（连线）型。

逻辑功能定义

模块中最重要的部分是逻辑功能定义。有3种方法可在模块中描述逻辑。

1.用“assign”语句

如：assign F=~((A&B)|(C&D));

这种方法的句法很简单，只须写一个“assign”,后面在
加一个方程式即可。“assign”语句一般适合于对组合逻辑进行赋值，称为连续赋值方式。

2.用实例元件

如：and myand3(f，a，b，c);

这个语句利用Verilog HDL提供的与门库，定义了一个三输入的与门。采用实例元件的方法象在电路图输入方式下，调入库元件一样，键入元件的名字和引脚的名字即可。要求每个实例元件的名字必须是唯一的。

逻辑功能定义

3.用“always”块语句

在前面的计数器模块中：

```
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;
    else
          out =out+1;
  end
```

“always”块可用于产生各种逻辑，常用于描述时序逻辑。这个例子中用“always”块生成了一个带同步置数的计数器。

数据类型及常量、变量

Verilog HDL中总共有19种数据型。数据类型是用来表示数字电路中的数据存储和传送单元的。其中4个最基本的数据类型是：reg型、wire型、integer型、parameter型。

Verilog HDL语言中也有常量和变量之分,它们分别属于以上这些类型。下面对最常用的几种进行介绍。

常量

在程序运行过程中, 其值不能被改变的量称为常量。

1.数字

(1)整数

在Verilog HDL中,整数型常量即整常数有以下4种进制表示形式:

1)二进制整数(b或B);

2)十进制整数(d或D);

3)十六进制整数(h或H);

4) 八进制整数(o或O)。

数字

完整的数字表达式为：

<位宽>’<进制><数字>

位宽为对应二进制数的宽度，如：

8’b11000101 //位宽为8的数的二进制表示，’b表示二进制

8’hc5 //位宽为8的数的十六进制，’h表示十六进制

十进制的数可以缺省位宽和进制说明，如：

197 //代表十进制数197

(2)x和z值

x代表不定值,z代表高阻值。每个字符代表的宽度取决于所用的进制，例如：

8’b1001xxxx 等价于8’h9x

8’b1010zzzz 等价于8’haz

在case语句中使用x进行匹配，可增强程序的可读性。在较长的数之间可用下划线分开，如16’b1010_1101_0010_1001。当变量不说明位数时，默认值为32位。此外，“？”是高阻态z的另一种表示

符号。

Parameter常量

在Verilog HDL中，用parameter来定义常量，即用parameter来定义一个标识符，代表一个常量，称为符号常量。其说明格式如下：

parameter 参数名1＝表达式，参数名2＝表达式，……， 参数名n＝表达式;

例如：parameter sel=8, code=8'ha3;

//分别定义参数sel为常数8(十进制)，参数code为常数a3(十六进制)

又如： parameter datawidth=8,
addrwidth=datawidth*2;

变量

变量是在程序运行过程中其值可以改变的量。变量分为两种：

网络型(nets)

寄存器型(register)

Nets型变量

nets型变量指输出始终根据输入的变化而更新其值的变量，它一般指的是硬件电路中的各种物理连接。Verilog HDL中提供了多种nets型变量，具体见下表：

类　型	功　能　说　明
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
tri1, tri0	分别为上拉电阻和下拉电阻
supply1, supply2	分别为电源(逻辑1)和地(逻辑0)

Nets型变量

Wire是一种最常用的nets型变量， wire型数据常用来表示以assign语句赋值的组合逻辑信号。

Verilog HDL模块中的输入/输出信号类型缺省时自动定义为wire型。 wire型信号可以用作任何方程式的输入，也可以用作“assign”语句和实例元件的输出。取值为0, 1, x, z。

wire型变量的定义格式如下：

wire 数据名1,数据名2,……,数据名n;

Wire型变量

例如：

wire a，b；//定义了两个wire型变量a，b

上面两个变量a，b的宽度为1位，若定义一个向量(vectors)，可按以下方式：

wire[n-1:0] 数据名1,数据名2,……,数据名n;

wire[n:1] 数据名1,数据名2,……,数据名n;

它们定义了数据宽度为n位。如下面定义了8位宽的数据总线，20位宽的地址总线：

wire[7:0] databus;

wire[19:0] addrbus;

或

wire[8:1] databus;

wire[20:1] addrbus;

register型变量

register型变量对应的是具有状态保持作用的电路元件，如触发器、寄存器等。 register型变量与nets变量的根本区别在于：

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。在设计中必须将寄存器型变量放在过程块语句(如initial，always)中，通过过程赋值语句赋值。在always， initial等过程块内被赋值的每一个信号都必须定义成寄存器型。

Verilog HDL中，有4种寄存器型变量，具体见下表：

类　型	
reg	常用的寄存器型变量
integer	32位带符号整数型变量
real	64位带符号实数型变量
time	无符号时间变量

register型变量

integer、real、time 3种寄存器型变量都是纯数学的抽象描述，不对应任何具体的硬件电路。reg型变量是最常用的一种寄存器型变量，下面着重对其进行介绍。

reg型变量的定义格式类似于wire型，具体格式为：

reg 数据名1,数据名2,……,数据名n;

例如：

reg a，b; //定义了两个reg型变量a，b

下面的语句定义了8位宽的数据：

reg[7:0] data; //定义data为8位宽的reg型向量

或 reg[8:1] data;

数组

若干个相同宽度的向量构成数组，reg型数组变量即为 memory型变量，即可定义存储器型数据。如：

```
reg[7:0] mymem[1023:0];
```

上面的语句定义了一个1024个字节、每个字节宽度为8位的存储器。通常，存储器采用如下方式定义：

```
parameter  wordwidth=8,memsize=1024;
reg[wordwidth-1:0]  mymem[memsize-1:0];
```

Verilog与C语言相对应的运算符

C语言	Verilog语言	功能
*	*	乘
/	/	除
+	+	加
-	-	减
%	%	取模
!	!	逻辑反
&&	&&	逻辑与
‖	‖	逻辑或
>	>	大于
<	<	小于
>=	>=	大于等于

Verilog与C语言相对应的运算符

C语言	Verilog语言	功能
<=	<=	小于等于
= =	= =	等于
!=	!=	不等于
~	~	位取反
&	&	按位逻辑与
|	|	按位逻辑或
^	^	按位逻辑异或
~^	~^	按位逻辑同或
>>	>>	右移
<<	<<	左移
?:	?:	等同于if-else

运算符及表达式

Verilog HDL的运算符范围很广，其运算符按其功能可分为以下几类:

1. 算术运算符

+，－，×，/，%

以上的算术运算符都属于双目运算符。前面4种用于常用的加、减、乘、除四则运算，%是求模运算符，或称为求余运算符。

2.逻辑运算符

&&　逻辑与

||　逻辑或

!　逻辑非

运算符及表达式

3.位运算符

位运算符是将两个操作数按对应位进行逻辑运算，位运算符包括：

~	按位取反
&	按位与
|	按位或
^	按位异或
^~，~^	按位同或(^~与~^是等价的)

要注意的一点是两个不同长度的数据进行位运算时，会自动将两个操作数按右端对齐，位数少的操作数会在高位用0补齐。

运算符及表达式

4.关系运算符

<	小于
<=	小于或等于
>	大于
>=	大于或等于

在进行关系运算时，如果声明的关系是假，则返回值是0；如果声明的关系是真，则返回值是1；如果某个操作数的值不确定，则其结果是模糊的，返回值是不确定的。

运算符及表达式

5.等式运算符

==	等于
!=	不等于
===	全等
!==	不全等

这4种运算符都是双目运算符，得到的结果是1位的逻辑值。如果得到1，说明声明的关系为真；如得到0，说明声明的关系为假。

“===”和“==”运算符的不同是,它在对操作数进行比较时对某些位的不定值x和高阻值z也进行比较,两个操作数必需完全一致，其结果才是1，否则为0。

比如：a=4’b1x01，b=4’b1x01，则“a==b”的结果为不定值x,而“a===b”的结果为1。

运算符及表达式

6. 缩减运算符

缩减运算符是单目运算符，它有：

&	与
~&	与非
|	或
~|	或非
^	异或
~^, ^~	同或

缩减运算符与位运算符的逻辑运算法则一样，但缩减运算符是单个操作数逐位进行与、或、异或等递推运算。

运算符及表达式

7. 移位运算符

>>　　右移

<<　　左移

Verilog HDL的移位运算符只有左移和右移两个，其用法为：A>>n或A<<n，表示把操作数A右移或左移n位，同时用0填补移出的位。

例： A=5’b11001

则 A>>2的值为 5’b00110。

运算符及表达式

8. 条件运算符

?:

它是一个三目运算符，对3个操作数进行运算，其意义同C语言中的定义一样，方式如下：

信号=条件？表达式1：表达式2；

当条件成立时，信号取表达式1的值，反之取表达式2的值。

例如：对2选1的MUX可描述如下：

out=sel?in1:in0;

即sel=1时out=in1; sel=0时out=in0。

运算符及表达式

9. 位拼接运算符

{ }

它将两个或多个信号的某些位拼接起来。用法如下：

{信号1的某几位，信号2的某几位，.....，
信号n的某几位}

如：**{a,b[3:0],w,3'b101}** 也可以写成为

{a, b[3], b[2], b[1], b[0], w, 1'b1, 1'b0, 1'b1}

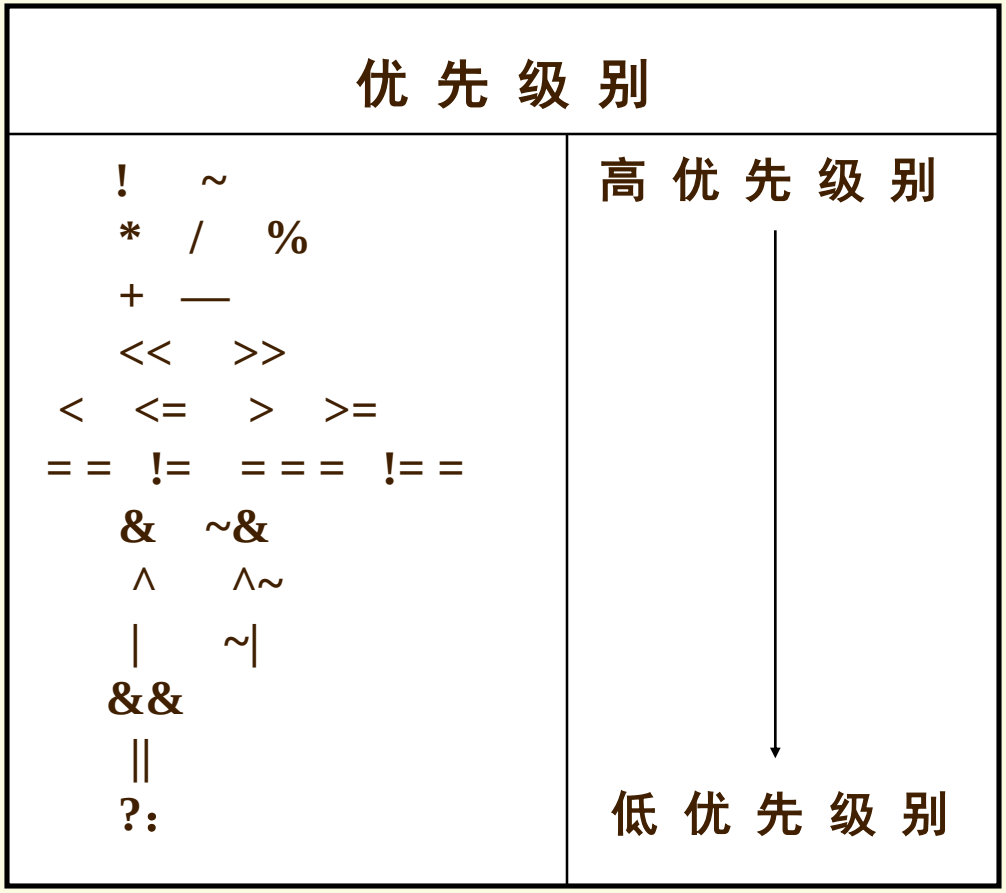
位拼接还可以用重复法来简化表达式。

如：**{4{w}}** //等同于{w,w,w,w}

位拼接还可以用嵌套的方式来表达。如：

{b, {3{a, b}}} //这等同于{b, a, b, a, b, a, b}

运算符的优先级



语句

赋值语句

- 1. 连续赋值语句

assign为连续赋值语句，它用于对wire型变量进行赋值。如：

assign c=a&b;

在上面的赋值中，a、b、c三个变量皆为wire型变量，a和b信号的任何变化，都将随时反映到c上来，因此称为连续赋值方式。

2. 过程赋值语句

过程赋值语句用于对寄存器类型(reg)的变量进行赋值，主要用在“always”模块内。过程赋值有两种方式：

(1)非阻塞赋值

<= 如b<=a;

非阻塞赋值在块结束时才完成赋值操作，即b的值并不是立即就改变的。

(2)阻塞赋值

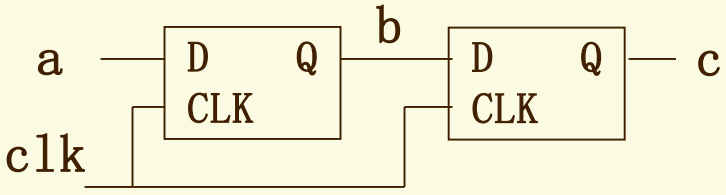
= 如b=a;

阻塞赋值在该语句结束时就完成赋值操作，即b的值在该赋值语句结束后立刻改变。

非阻塞赋值和阻塞赋值的区别

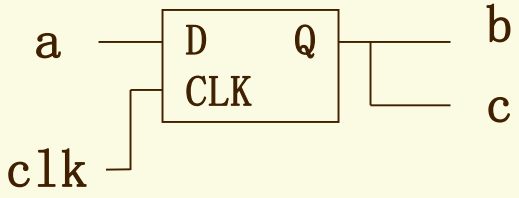
例：非阻塞赋值：

```
module  non_block(c,b,a,clk);
output c,b;
input clk,a;
reg c,b;
always @(posedge clk)
  begin
     b<=a;
     c<=b;
  end
endmodule
```



阻塞赋值：

```
module  block(c,b,a,clk);
output c,b;
input clk,a;
reg c,b;
always @(posedge clk)
   begin
      b=a;
      c=b;
   end
endmodule
```



Verilog与C语言相对应的控制结构与关键字

C语言	Verilog语言
sub-function	Module,function,task
if –then-else	if –then-else
case	case
{,}	begin,end
for	for
while	while
break	disable
define	define
int	int
printf	monitor,display,strobe

条件语句

条件语句有if-else语句和case语句两种。它们都是顺序语句，应放在“always”块内。

1.if-else语句

其格式与C语言中的if-else语句类似，有3种：

①if(表达式) 语句1;

②if(表达式) 语句1;
else 语句2;

③if(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n+1;

- 如执行的语句为多句时，要用“begin-end”语句括起来。

条件语句

2. case语句

相对if语句只有两个分支而言，case语句是一种多分支语句，故case语句多用于多条件译码电路，如描述译码器、数据选择器、状态机及微处理器的指令译码等。case有case、casez、casex三种表示方式。

(1)case语句

```
case(敏感表达式)
    值1：语句1;          //case分支项
    值2：语句2;
    ……
    值n：语句n;
    default：语句n+1; //default语句可以省略
endcase
```

(2)casez与casex语句

在**casez**与**casex**语句中，值**1~**值**n**中的某些位可以是**z**和**x**，比较时对这些位不予考虑。另外,还可以用无关值“**?**”来表示**z**。

这两种语句的格式与**case**语句相同。

循环语句

在verilog HDL中有4种类型的循环语句，用来控制语句的执行次数，这4种语句分别是：

(1)forever 连续地执行语句,多用在“initial”块中，

以生成周期性输入波形。

(2)repeat 连续执行一条语句n次。

(3)while 执行一条语句，直到某个条件不满足。

(4)for语句。

1. for语句，格式同C语言：

for(表达式1；表达式2；表达式3)语句

即for(循环变量赋初值；循环结束条件；循环变量增值)执行语句

循环语句

2.repeat 语句

repeat(循环次数表达式)语句；

或　repeat(循环次数表达式) begin

……

end

3.while语句

while(循环执行条件表达式)语句；

或　while(循环执行条件表达式) begin

……

end

4. forever语句

forever　语句；

或　forever　begin

……

end

结构说明语句

Verilog HDL中的任何过程模块都从属于以下4种结构说明语句：

- initial
- always
- task
- function

在一个模块(module)中，使用initial和always语句的次数是不受限制的。 Initial说明语句一般用于仿真中的初始化，仅执行一次；always块内的语句则是不断重复执行的；task和function语句可以在程序模块中的一处或多处调用。

always块语句

格式： always @(<敏感信号表达式event-expression>)
begin
//过程赋值
//if语句
//case语句
//while，repeat，for循环
//task，function调用
end

- 敏感信号表达式“event-expression”

敏感信号表达式又称事件表达式或敏感表，当该表达式的值改变时，就会执行一遍块内语句。因此在敏感信号表达式中应列出影响块内取值的所有信号，若有两个或两个以上信号时，它们之间用“or”连接。

always块语句

- **posedge与negedge关键字**

  对于时序电路，事件是由时钟边沿触发的。为表达边沿这个概念，verilog HDL提供了posedge与negedge两个关键字来描述，分别表示信号的上升沿和下降沿。

如： always @(posedge clk) //表示时钟信号clk的上升沿

always @(negedge clk) //表示时钟信号clk的下降沿

语句的顺序执行与并行执行

用Verilog HDL模块来设计电路，首先应该清楚哪些操作是同时发生的，哪些是顺序发生的。

在“always”模块内,逻辑是按照指定的顺序执行的，“always”块内的语句称为顺序语句，因为这些语句完全按照书写的顺序来执行。

“always”模块之间，是同时执行的，或者说
是并行执行的。两个或更多个“always”模块、“assign”语句、实例元件等是同时执行的。

语句的顺序执行与并行执行举例

例1.顺序执行模块1

```
module  serial1(q,a,clk);
output q,a;
input clk;
reg q,a;
always @(posedge clk)
  begin
    q=~q;
    a=~q;
  end
endmodule
```

例2. 顺序执行模块2

```
module  serial2(q,a,clk);
output q,a;
input clk;
reg q,a;
always @(posedge clk)
  begin
    a=~q;
    q=~q;
  end
endmodule
```

语句的顺序执行与并行执行举例

上面的两个例子，其区别只是在“always”模块内，把两个赋值语句的顺序相互颠倒。分别对上面的两个模块用MAX+PLUS II软件进行仿真,得到的波形分别如图1和图2所示。

图1

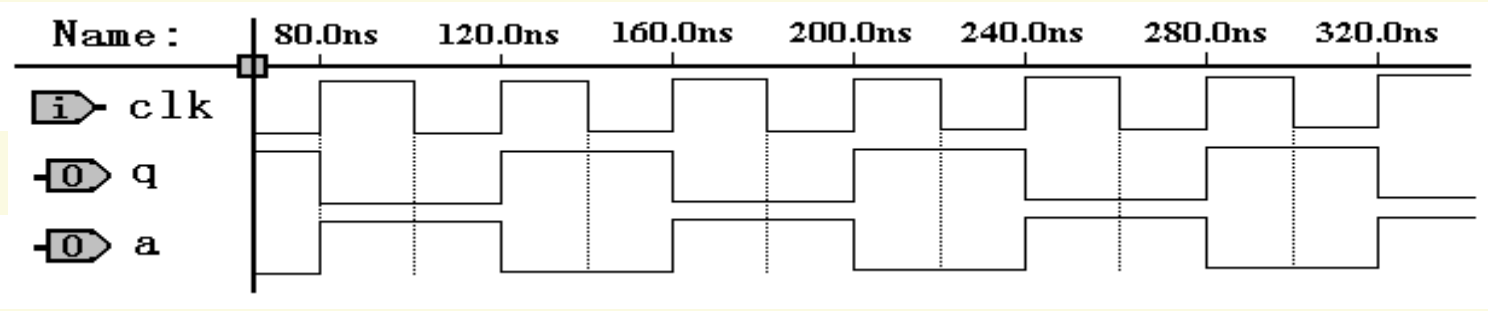
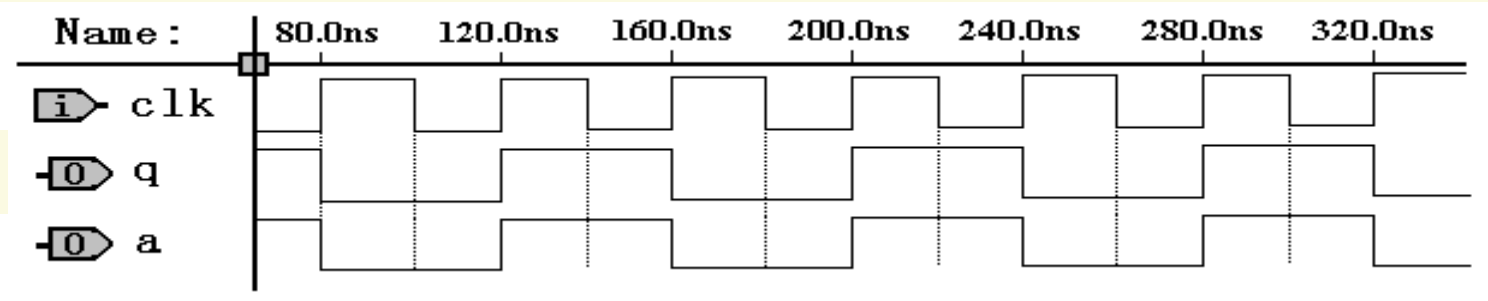


图2



语句的顺序执行与并行执行举例

如果将上述两句赋值语句分别放在两个“always”模块中，如例3
和例4两个程序，经过仿真可以发现：这两个“always”模块放置的顺

例3.并行模块1

```
module  paral1(q,a,clk);
output q,a;
input clk;
reg q,a;
always @(posedge clk)
  begin
     q=~q;
  end
always @(posedge clk)
  begin
     a=~q;
   end
endmodule
```

例4. 并行模块2

```
module  paral2(q,a,clk);
output q,a;
input clk;
reg q,a;
always @(posedge clk)
  begin
     a=~q;
  end
always @(posedge clk)
  begin
     q=~q;
  end
endmodule
```

initial语句

格式：initial
begin
语句1；
语句2；
……
end

例： initial
begin
reg1=0;
for(addr=0;addr<size;addr=addr+1)
memory[addr]=0;
end

在上面的例子中，使用initial语句首先将一个变量reg1初始化为0，然后又用for循环语句将memory存储器进行初始化，将其所有的存储单元都置为“0”。

task和function语句

task和function语句分别用来定义任务和函数。利用任务和函数可以把一个大的程序模块分解成许多小的任务和函数，以方便调试，并且能使写出的程序清晰易懂。

1. 任务(task)

定义：task〈任务名〉；

端口及数据类型声明语句；

其他语句；

endtask

调用：〈任务名〉(端口1，端口2，……);

说明：①任务的定义和调用须在一个module模块内。

②定义任务时，没有端口名列表，但需要在后面进行端口和数据类型的说明。

③当任务被调用时，任务被激活。任务的调用与模块调用一样通过任务名调用实现，调用时，需列出端口名列表，端口名的排序和类型必须与任务定义中的排序和类型一致。

④一个任务可以调用别的任务和函数，调用的个数不限。

task和function语句

2. 函数(function)

定义：function<返回值位宽和类型说明> 函数名；

端口声明；

局部变量定义；

其他语句；

endfunction

<返回值位宽和类型说明>是一个可选的项，如果省略，则返回值为一位关于寄存器类型的数据。

函数的定义中蕴含了一个与函数同名的、函数内部的寄存器。在函数定义时，将函数返回值所使用的寄存器设为与函数同名的内部变量，因此函数名被赋予的值就是函数的返回值。

函数的调用是通过将函数作为表达式中的操作数来实现的。调用格式如下：

<函数名>(<表达式> <表达式>)；

函数的使用与任务相比有更多的限制和约束。例如，函数不能启动任务，在函数中不能包含有任何的时间控制语句，同时定义函数时至少要有一个输入参量等。这些需要在使用时注意。

task和function语句

3. 任务与函数的区别

	任务(task)	函数(function)
输入与输出	可有任意个各种类型的参数	至少有一个输入，不能将inout类型作为输出
调用	任务只可在过程语句中调用，不能在连续赋值语句assign中调用	函数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	任务可调用其他任务和函数	函数可调用其他函数，但不可以调用其他任务
返回值	任务不向表达式返回值	函数向调用它的表达式返回一个值

上面介绍了Verilog HDL的结构说明语句。在使用这些语句时，另外需要注意的一点是有的编译器对某些结构说明语句是不支持的。如 MAX+PLUS II，该软件支持always、function语句，但不支持task和initial语句。

编译预处理语句

- Verilog HDL语言和C语言一样也提供了编译预处理功能。
- 编译引导语句用主键盘左上角小写键 “ ` ”起头。
- 在编译时,先对这些特殊语句进行“预处理”，然后再将预处理的结果和源程序一起进行编译。
- 常用的编译预处理语句有：
  - a) `define
  - b) `include
  - c) `timescale

`define语句

- 使用`define 编译预处理语句能提供简单的文本替代功能，其形式为：
  `define〈宏名〉〈宏文本〉
  在编译时会用宏文本来替代源代码中的宏名。
- 合理地使用`define可以提高程序的可读性

举例说明：

```
`define on 1'b1
`define off 1'b0
`define and_delay #3
```

在程序中可以用有含义的文字来表示没有意思的数码,提高了程序的可读性，上面程序中用 `on，`off，`and_delay 分别表示 1，0，和 #3 。

- 宏定义语句行末不加分号。
- 在引用已定义的宏名时，前面要加符号“`”。

`include语句

- `include是文件包含语句，它可将一个文件全部包含到另一个文件中。其形式为：

  `include “文件名”

举例说明：

`include “global.v”

`include “parts/counter.v”

`include “../../library/mux.v”

- 合理地使用`include 可以使程序简洁、清晰、条理清楚、易于查错。
- 一个`include语句只能指定一个被包含的文件。
- `include语句可以出现在源程序的任何地方。
- 文件包含允许多重包含。比如文件1包含文件2，文件2包含文件3等。
- MAX+PLUS II软件不支持`include语句。

`timescale语句

- **`timescale** 用于定义模块中的时间单位和精度，其格式如下：
  `timescale <时间单位>/<时间精度>
- 时间单位有：s、ms、us、ns、ps和fs, 分别表示秒、10^{-3}秒、10^{-6}秒、10^{-9}秒、10^{-12}秒和10^{-15}秒。
- 举例说明： **`timescale 1ns/100ps**
  上面的语句表示时间单位是**1ns**，精度为**100ps**。
- **`timescale** 语句必须放在模块边界前面

如：
```
`timescale 1ns/100ps
 module MUX2_1(out,a,b,sel);
 … …
  not #1 not1(nsel, sel);
  and #2 and1(a1, a, nsel);
 … …
 endmodule
```

- 尽可能地使精度与时间单位接近，只要满足设计的实际需要就行。
- 在上例中所有的时间单位都是**1ns**的整数倍。

不同抽象级别的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。也就是说，既可以用电路的逻辑功能描述也可以用元器件和它们之间的连接来建立所设计电路的Verilog HDL模型。

Verilog HDL是一种能够在多个级别对数字电路和数字系统进行描述的高级语言， Verilog HDL模型可以是对实际电路的不同级别的抽象。这些抽象级别一般可分为5级：

1. 系统级(System Level)
2. 算法级(Algorithm Level)
3. 寄存器传输级(RTL, Register Transfer Level)
4. 门级(Gate Level)
5. 开关级(Switch Level)

其中，前3种属于高级别的描述方法，又称为行为级的描述。门级模型是描述逻辑门以及逻辑门之间连接关系的模型。而开关级的模型则是描述器件中三极管和存储节点以及它们之间连接关系的模型。

对于数字系统的设计而言，主要掌握高层描述方法。在这里，我们主要讨论基于行为级和门级的逻辑描述。

Verilog HDL门级描述

Verilog HDL中提供了丰富的门类型关键字，用于电路的门级描述。 Verilog HDL有关门类型的关键字共有26个，比较常用的有下面几个：

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

Verilog HDL门级描述

- 调用门原语的句法如下：

  门类型关键字<例化的门名称>(<端口列表>)

  端口列表按下列顺序列出：

  (输出，输入1，输入2，输入3，……)；

  对于三态门，则按如下顺序列出端口：

  (输出，输入，使能控制端)；

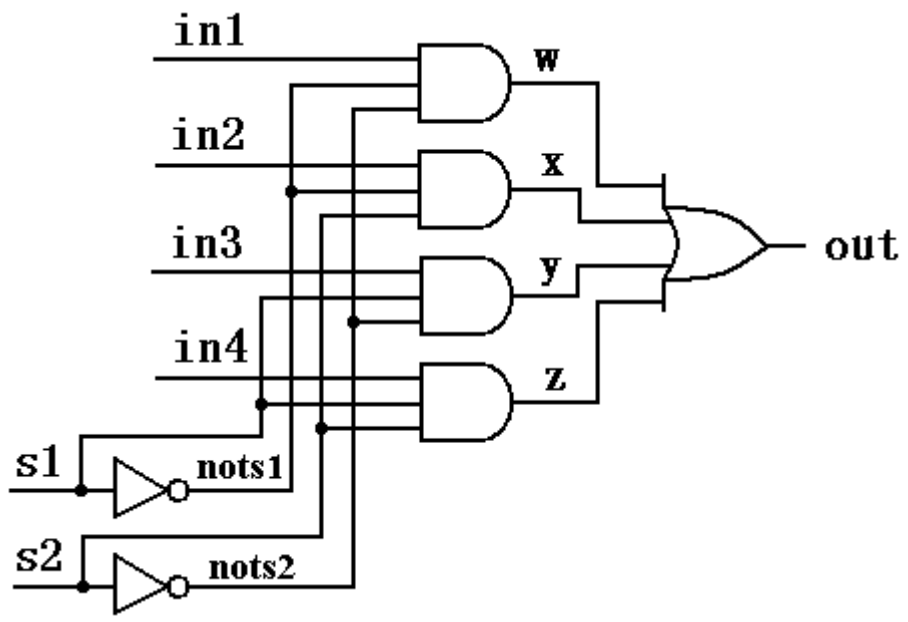
例：**and myand(out,in1,in2,in3);** //三输入与门

and myand2(out,in1,in2); //二输入与门

bufif1 mytri(out,in,enable); //高电平使能的三态门

【例1】调用门原语实现的4选1 MUX

```
module mux4_1(out,in1,in2,in3,in4,s1,s2);
output out;
input in1,in2,in3,in4,s1,s2;
wire nots1,nots2,w,x,y,z;
not (nots1,s1);
not (nots2,s2);
and (w,in1,nots1,nots2);
and (x,in2, nots1,s2);
and (y,in3, s1,nots2);
and (z,in4, s1, s2);
or (out,w,x,y,z);
endmodule
```



Verilog HDL的行为级描述

【例2】采用功能描述的4选1 MUX

```
module mux4_1(out,in1,in2,in3,in4,s1,s2);
output out;
input in1,in2,in3,in4,s1,s2;
assign out=(in1 & ~s1 & ~s2)|(in2 & ~s1 & s2)|
                (in3 & s1 & ~s2)|(in4 & s1 & s2);
endmodule
```

【例3】用case语句描述的4选1 MUX

```
module mux4_1(out,in1,in2,in3,in4,s1,s2);
output out;
input in1,in2,in3,in4,s1,s2;
reg out;
always @(in1 or in2 or in3 or in4 or s1 or s2)
  begin
   case({s1,s2})
    2'b00:out=in1;
    2'b01:out=in2;
    2'b10:out=in3;
    2'b11:out=in4;
    default:out=2'bx;
   endcase
  end
endmodule
```

【例4】用条件运算符描述的4选1 MUX

```
module mux4_1(out,in1,in2,in3,in4,s1,s2);
output out;
input in1,in2,in3,in4,s1,s2;
assign out=s1? (S2? in4:in3): (s2? in2:in1);
endmodule
```

- 对于设计者而言，采用的描述级别越高，设计越容易。但对于特定的综合器而言，有可能无法将某些抽象级别高的描述转化成电路。所以，设计者应该了解所用综合器的性能，以选择适当的描述级别，或者选择更好的综合软件。

小结

1. Verilog HDL语言基本概念及特点。
2. Verilog HDL模块的基本结构。
3. Verilog HDL语言的数据类型及常量、变量。
4. Verilog HDL语言的运算符及表达式。
5. Verilog HDL语句。语句的顺序执行和并行执行。
6. 不同抽象级别的Verilog HDL模型。