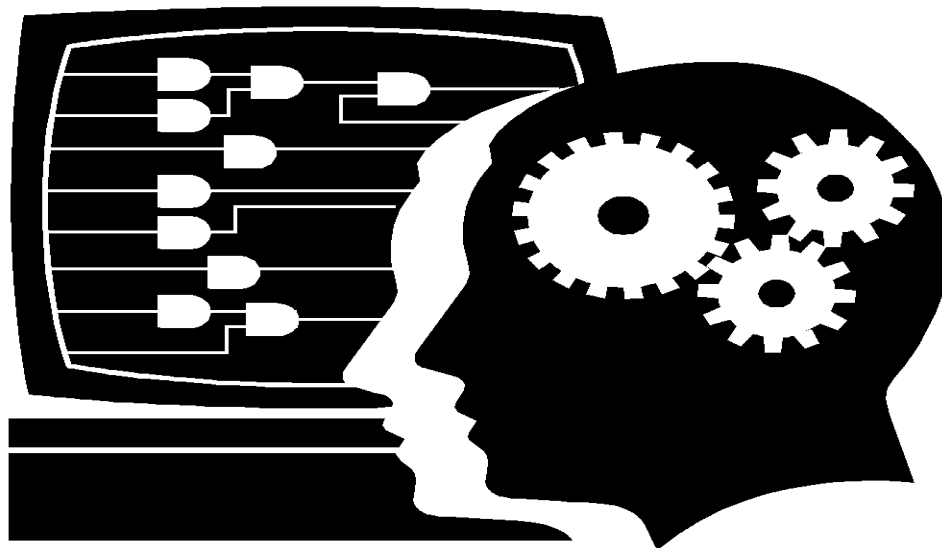


Verilog设计初步



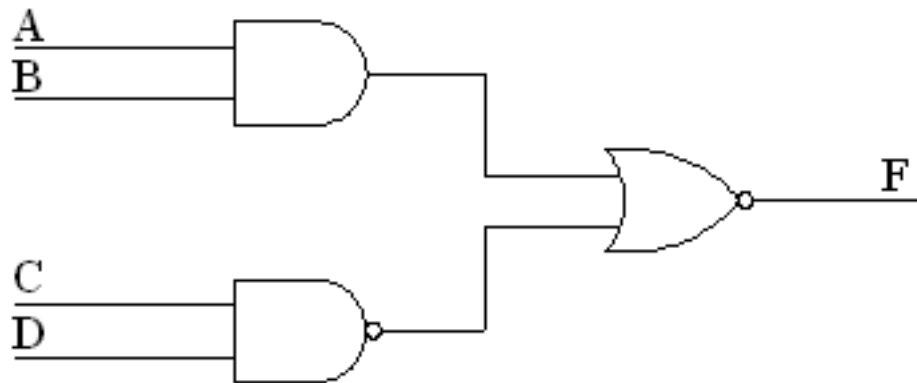


主要内容

- 1 Verilog模块的结构**
- 2 Verilog语言要素**
- 3 常量**
- 4 数据类型**
- 5 参数**
- 6 向量**
- 7 运算符**

1 Verilog模块的结构

```
module AOI (A,B,C,D,F); //模块名为AOI
input A,B,C,D;          //模块的输入端口为A, B, C, D
output F;                //模块的输出端口为F
wire A,B,C,D,F;         //定义信号的数据类型
assign F= ~( (A&B) | (~ (C&D)) ); //逻辑功能描述
endmodule
```



该程序的第**1**行为模块的名字、模块的端口列表；第**2**、**3**行为输入输出端口声明，第**4**行定义了端口的数据类型；在第**5**行中对输入、输出信号间的逻辑关系进行了描述。

Verilog 模块的模板

```
module <顶层模块名> (<输入输出端口列表>);
output 输出端口列表;    //输出端口声明
input  输入端口列表;    //输入端口声明
/*定义数据，信号的类型，函数声明*/
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
    //while, repeat, for循环语句
    //task, function调用
end
//调用其他模块
<调用模块名module_name> <例化模块名> (<端口列表port_list>);
//门元件例化
门元件关键字 <例化门元件名> (<端口列表port_list>);
endmodule
```



2 Verilog语言要素

Verilog 程序由符号流构成，符号包括

- 空白符（**White space**）
- 注释（**Comments**）
- 操作符（**Operators**）
- 数字（**Numbers**）
- 字符串（**Strings**）
- 标识符（**Identifiers**）
- 关键字（**Keywords**）等



空白符和注释

■ 空白符 (White space)

空白符包括：空格、**tab**、换行和换页。空白符使代码错落有致，阅读起来更方便。在综合时空白符被忽略。

■ 注释 (Comment)

- ◆ 单行注释：以 “//” 开始到本行结束，不允许续行
- ◆ 多行注释：多行注释以 “/*” 开始，到 “*/” 结束

标识符 (Identifiers)

■ 标识符 (Identifiers)

Verilog中的**标识符**可以是任意一组字母、数字以及符号“\$”和“_”（下划线）的组合，但标识符的第一个字符必须是字母或者下划线。另外，标识符是区分大小写的。

■ Examples:

`count`

`COUNT`

`_A1_d2`

`R56_68`

`FIVE`

//COUNT与count是不同的

//以下划线开头



关键字 (Keywords)

- Verilog语言内部已经使用的词称为**关键字**或**保留字**，这些保留字用户不能作为变量或节点名字使用。
- 关键字都是**小写的**。



3 常量

程序运行中，值不能被改变的量称为常量（**constants**），Verilog中的常量主要有如下3种类型：

- ◆ 整数
- ◆ 实数
- ◆ 字符串



整数 (integer)

- 整数按如下方式书写:

+/-<size> '<base><value>

即 **+/-<位宽>'<进制><数字>**

size 为对应二进制数的宽度；**base**为进制；**value**是基于进制的数字序列。

进制有如下4种表示形式:

- ◆ 二进制 (**b**或**B**)
- ◆ 十进制 (**d**或**D**或缺省)
- ◆ 十六进制 (**h**或**H**)
- ◆ 八进制 (**o**或**O**)

整数 (integer)

Examples:

8'b11000101 //位宽为八位的二进制数11000101

8'hd5 //位宽为八位的十六进制数d5;

5'O27 //5位八进制数

4'D2 //4位十进制数2

4'B1x_01 //4位二进制数1x01

5'Hx //5位x（扩展的x），即xxxxx

4'hZ //4位z，即zzzz

8□'h□2A /*在位宽和'之间，以及进制和数值

之间允许出现空格，但'和进制之间，数值间是不允许出现空格的，比如8'□h2A、8'h2□A等形式都是不合法的写法 */



实数 (Real)

■ 实数 (Real) 有下面两种表示法。

◆ 十进制表示法。例如：

2.0

0.1 //以上2例是合法的实数表示形式

2. //非法：小数点两侧都必须有数字

◆ 科学计数法。例如：

43_5.1e2 //其值为43510.0

9.6E2 //960.0 (e与E相同)

5E-4 //0.0005



字符串 (Strings)

- 字符串是双引号内的字符序列。
- 字符串不能分成多行书写。
- 字符串的作用主要是用于仿真时，显示一些相关的信息，或者指定显示的格式。

4 数据类型

数据类型（Data Type）是用来表示数字电路中的物理连线、数据存储和传输单元等物理量的。

- Verilog有下面四种基本的逻辑状态。
 - ◆ 0：低电平、逻辑0或逻辑非
 - ◆ 1：高电平、逻辑1或“真”
 - ◆ x或X：不确定或未知的逻辑状态
 - ◆ z或Z：高阻态
- Verilog中的所有数据类型都在上述4类逻辑状态中取值，其中x和z都不区分大小写，也就是说，值0x1z与值0X1Z是等同的。



数据类型 (Data Type)

■ Verilog中的变量分为如下两种数据类型：

◆ net型

◆ variable型

net型中常用的有wire、tri；

variable型包括reg、integer等。

■ 注意：在Verilog-1995标准中，variable型变量称为register型；在Verilog-2001标准中将register一词改为了variable，以避免初学者将register和硬件中的寄存器概念混淆起来。

net型

- Net型数据相当于硬件电路中的各种物理连接，其特点是输出的值紧跟输入值的变化而变化。对连线型有两种驱动方式，一种方式是在结构描述中将其连接到一个门元件或模块的输出端；另一种方式是用持续赋值语句assign对其进行赋值。
- wire是最常用的Net型变量。
- wire型变量的定义格式如下：

wire 数据名1, 数据名2,数据名n;

例如: wire a,b; //定义了两个wire型变量a和b

Examples:

```
wire[7:0] databus;           //databus的宽度是8位  
wire[19:0] addrbus;          //addrbus的宽度是20位
```


Variable型

- **variable**型变量必须放在过程语句（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信号必须定义成**variable**型。
- 注意：**variable**型变量并不意味着一定对应着硬件上的一个触发器或寄存器等存储元件，在综合器进行综合时，**variable**型变量会根据具体情况来确定是映射成连线还是映射为触发器或寄存器。
- **reg**型变量是最常用的一种**variable**型变量。定义格式如下：
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

Examples:

```
reg[7:0] qout; //定义qout为8位宽的reg型向量  
reg[8:1] qout;
```



5 参数(parameter)

■ 在**Verilog**语言中，用参数**parameter**来定义符号常量，即用**parameter**来定义一个标志符代表一个常量。参数常用来定义时延和变量的宽度。

■ 其定义格式如下：

parameter 参数名1=表达式1， 参数名2=表达式2， 参数名3=表达式3，；

■ 例如：

parameter sel=8,code=8'ha3;

//分别定义参数**sel**代表常数8（10进制），参数**code**代表常量**a3**（16进制）



6 向量

- 1. 标量与向量

宽度为**1**位的变量称为标量，如果在变量声明中没有指定位宽，则默认为标量（**1**位）。举例如下：

wire a; **//a为标量**

reg clk; **//clk为标量reg型变量**

- 线宽大于**1**位的变量（包括**net**型和**variable**型）称为向量（**vector**）。向量的宽度用下面的形式定义：

[msb : lsb]

- 比如：

wire[3:0] bus; **//4位的总线**



■ 2. 位选择和域选择

在表达式中可任意选中向量中的一位或相邻几位，分别称为位选择和域选择，例如：

A=mybyte[6]; //位选择

B=mybyte[5:2]; //域选择

■ 再比如：

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



7 运算符

- 1. 算术运算符 (**Arithmetic operators**)

- 常用的算术运算符包括:

+	加
-	减
*	乘
/	除
%	求模



■ 2. 逻辑运算符 (**Logical operators**)

&& 逻辑与

|| 逻辑或

! 逻辑非



■ 3. 位运算符 (**Bitwise operators**)

- 位运算，即将两个操作数按对应位分别进行逻辑运算。

~ 按位取反

& 按位与

| 按位或

^ 按位异或

^~, ~^ 按位同或（符号 \wedge 与 \vee 是等价的）



■ 4. 关系运算符 (**Relational operators**)

< 小于

<= 小于或等于

> 大于

>= 大于或等于



■ 5. 等式运算符 (**Equality Operators**)

== 等于

!= 不等于

=== 全等

!== 不全等



■ 6. 缩位运算符 (**Reduction operators**)

& 与

~& 与非

| 或

~| 或非

^ 异或

^~ , ~^ 同或



■ 7. 移位运算符 (**shift operators**)

>> 右移

<< 左移



■ 8. 条件运算符 (**conditional operators**) ?:

- 三目运算符，其定义方式如下：

**signal=condition>true_expression:false
_expression;**

- 即：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。



9. 位拼接运算符 (**concatenation operators**)

{ }


该运算符将两个或多个信号的某些位拼接起来。

使用如下：

{信号1的某几位, 信号2的某几位,, 信号n的某几位}

运算符的优先级

在书写程序时建议用括号（）来控制运算的优先级

运 算 符	优先级
! ~	 高优先级
* / %	
+ -	
<< >>	
< <= > >=	
= != == !=	
& ~&	
^ ~	
~	
&&	
?:	
	低优先级