

3 Verilog HDL基础语法与组合逻辑电路建模

3.1 Verilog HDL基本语法规则

3.2 Verilog HDL门级建模

3.3 Verilog HDL数据流建模与运算符

3.4 组合电路的行为级建模

3.5 分层次的电路设计方法

3.1 Verilog HDL基本语法规则

3.1.1 词法规定

3.1.2 逻辑值集合

3.1.3 常量及其表示

3.1.4 数据类型

3.1.1 词法规定

为对数字电路进行描述, Verilog语言规定了一套完整的语法结构。

1. **间隔符**: Verilog 的间隔符主要起分隔文本的作用, 可以使文本错落有致, 便于阅读与修改。

间隔符包括空格符 (\b)、TAB 键 (\t)、换行符 (\n) 及换页符。

2. **注释符**: 注释只是为了改善程序可读性, 编译时不起作用。

多行注释符(用于写多行注释): /* --- */;

单行注释符 :以//开始到行尾结束为注释文字。

3. 标识符和关键词

标识符:给对象（如模块名、电路的输入与输出端口、变量等）

取名所用的字符串。以英文字母或下划线开始

如，`clk`、`counter8`、`_net`、`bus_A`。

关键词:用Verilog语言本身规定的特殊字符串定义语言的结构。

例如，`module`、`endmodule`、`input`、`output`、`wire`、`reg`、`and`等都是关键词。关键词都是小写，关键词不能作为标识符使用。

3.1.2 逻辑值集合

为了表示数字逻辑电路的逻辑状态，Verilog语言规定了4种基本的逻辑值。

0	逻辑0、逻辑假
1	逻辑1、逻辑真
x或X	不确定的值（未知状态）
z或Z	高阻态

3.1.3 常量及其表示

三种类型的常量 { 整数型常量
实数型常量
字符串型常量

整数型常量 { 十进制数形式的表示方法: 表示有符号常量
例如: 30、-2
带基数形式的表示方法:
格式为: $\langle + / - \rangle \langle \text{位宽} \rangle' \langle \text{基数符号} \rangle \langle \text{数} \rangle$
例如: $\overset{\text{3 位 2 进制}}{3}' \text{b}101$ 、 $\overset{8}{5}' \text{o}37$ 、 $\overset{16}{8}' \text{h}e3$, $8' \text{b}1001_0011$

实数型常量 { 十进制记数法 如: 0.1、2.0、5.67
科学记数法 如: $23_5.1\text{e}2$ 、 $5\text{E}-4$
 23510.0 、 0.0005

3.1.3 常量及其表示

字符串常量

字符串是用双撇号括起来的字符序列，它必须包含在同一行中，不能分成多行书写。例如：

```
"this is a string"
```

```
"hello world!"
```

符号常量

Verilog 允许用参数定义语句定义一个标识符来代表一个常量，称为**符号常量**。定义的格式为：

```
parameter 参数名1 = 常量表达式1, 参数名2 = 常量表达式2.....
```

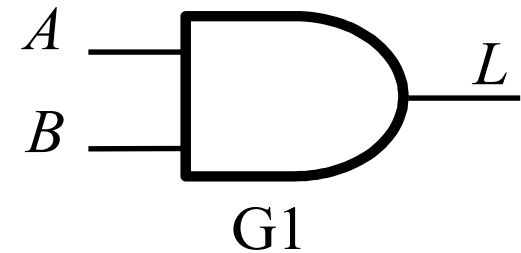
如 parameter BIT=1, BYTE=8, PI=3.14;

3.1.4 数据类型

变量的数据类型 { 线网型
寄存器型

线网类:是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.

例:网络型变量 L 的值由与门的驱动信号 a 和 b 所决定, 即 $L = a \& b$ 。 a 、 b 的值发生变化, 线网 L 的值会立即跟着变化。



常用的网络类型由关键词`wire`定义,格式如下:

`wire [n-1:0] 变量名1, 变量名2, ..., 变量名n;`

63:1
64:0
都
可以

变量宽度

表3.1.3 线网类型变量及其说明

线网类型	功能说明
wire, <u>tri</u>	用于表示单元（元件）之间的连线，wire为一般连线；tri用于描述由多个信号源驱动的线网，并没有其他特殊意义，两者的功能完全相同。
wor, trior	具有线或特性的线网，用于一个线网被多个信号驱动的情况
wand, riand	具有线与特性的线网，用于一个线网被多个信号驱动的情况
triereg	具有电荷保持特性的线网类型，用于开关级建模
tri1	上拉电阻，用于开关级建模
tri0	下拉电阻，用于开关级建模
supply1	用于对电源建模，高电平1
supply0	用于对地建模，低电平0

关于“多重驱动”

- 在写可综合的Verilog代码时，建议不要对同一个变量进行多次赋值（简称多重驱动），以避免出现多个信号同时驱动一个输出变量的情况。
- 例如， A 、 B 、 C 三个内部信号同时接到（驱动）一个输出端 L 。
或者说，输出 L 同时被三个内部信号所驱动。
此时 L 的逻辑值可能无法确定。

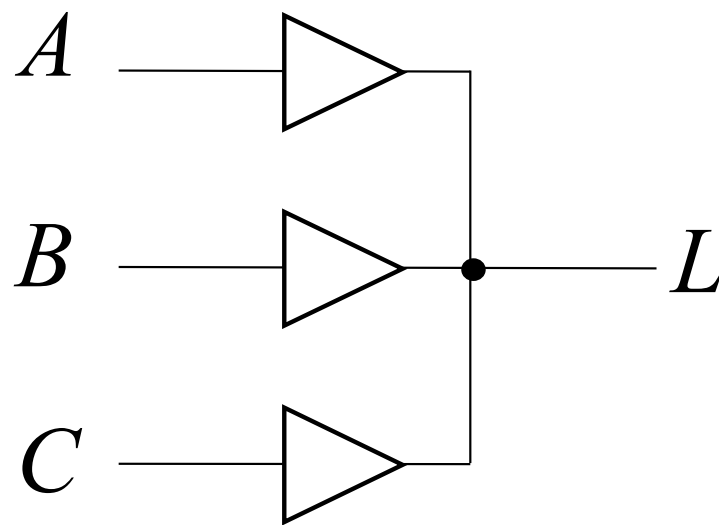


图3.1.2 多重驱动示意图

3.1.4 数据类型

变量的数据类型 { 线网型
寄存器型

寄存器型变量对应的是具有状态保持作用的电路等元件,如触发器寄存器。寄存器型变量只能在initial或always内部被赋值。

寄存器变量类型

表3.1.5 寄存器变量类型及其说明

寄存器类型	功能说明
<u>reg</u>	常用的寄存器型变量
<u>integer</u>	32位带符号的整数型变量
real/realtime	64位带符号的实数型变量
time	64位无符号的时间变量

抽象描述,
不对应具
体硬件

例: `reg clock;` //一个1位寄存器变量的声明
`reg [3:0] counter;` //一个4位寄存器变量的声明

3.2 Verilog HDL门级建模

3.2.1 多输入门

3.2.2 多输出门

3.2.3 三态门

3.2.4 门级建模举例

基本概念：

- **结构级建模：**就是根据逻辑电路的结构（逻辑图），实例引用Verilog HDL中内置的基本门级元件或者用户定义的元件或其他模块，来描述结构图中的元件以及元件之间的连接关系。
- **门级建模：**Verilog HDL中内置了12个基本门级元件（Primitive，有的翻译为“原语”）模型，引用这些基本门级元件对逻辑图进行描述，也称为门级建模。

Verilog HDL 基本门级元件 (Primitive : 原语)

- 多输入门: **and**、**nand**、**or**、**nor**、**xor**、**xnor**
只有单个输出, 1个或多个输入
与门 *异或* *同或*
- 多输出门: **not**、**buf**
允许有多个输出, 但只有一个输入
- 三态门: **bufif0**、**bufif1**、**notif0**、**notif1**
有一个输出, 一个数据输入和一个控制输入

- 上拉电阻**pullup**、下拉电阻**pulldown**

6.4.1 多输入门

多输入门的一般引用格式为：

Gate_name <instance> (OutputA, Input1, Input2,..., InputN);

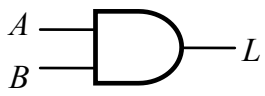
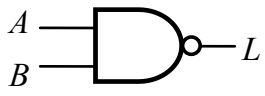




□ 共6个：

➤ and、nand、

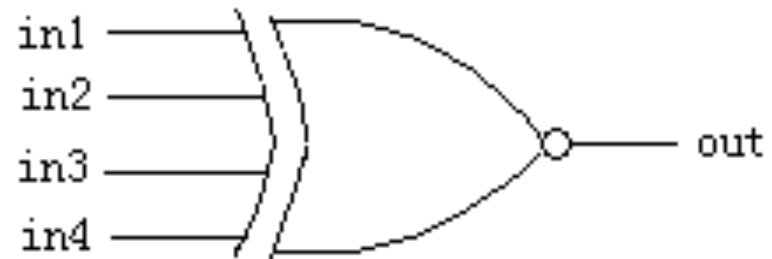
➤ or、nor、

➤ xor、xnor

□ 特点：只有1个输出，
有多个输入

原语名称	图形符号	逻辑表达式
and (与门)		$L = A \& B$
nand (与非门)		$L = \sim(A \& B)$
or (或门)		$L = A B$
nor (或非门)		$L = \sim(A B)$
xor (异或门)		$L = A \wedge B$
xnor (同或门)		$L = A \sim \wedge B$

基本门的调用方法举例：



先输出 后输入

```
and    A1 (out, in1, in2, in3) ;  
xnor   NX1 (out, in1, in2, in3, in4) ;
```

- 对基本门级元件，调用名A1、NX1可以省略。
- 若同一个基本门在当前模块中被调用多次，可在一条调用语句中加以说明，中间以逗号相隔。

真值表举例

表3.2.2 and、nand真值表

and		输入1			
		0	1	x	z
输入2	0	0	0	0	0
	1	0	1	x	x
	x	0	x	x	x
	z	0	x	x	x

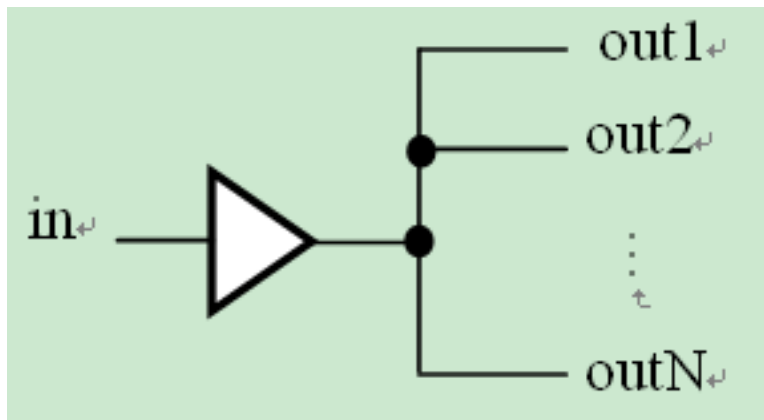
nand		输入1			
		0	1	x	z
输入2	0	1	1	1	1
	1	1	0	x	x
	x	1	x	x	x
	z	1	x	x	x

输出只有0、1、x三种状态

3.2.2 多输出门

允许有多个输出，但只有一个输入。

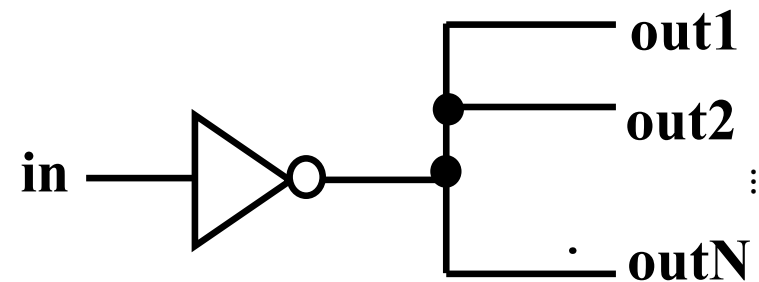
buf B1 (out1, out2, ..., in) ;



buf真值表

buf	输 入			
	0	1	x	z
输 出	0	1	x	x

not N1 (out1, out2, ..., in) ;

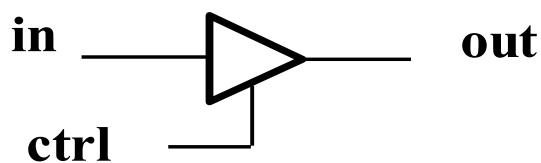


not真值表

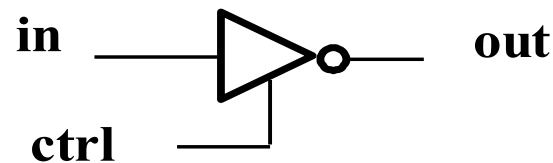
not	输 入			
	0	1	x	z
输 出	1	0	x	x

3.2.3 三态门

有一个输出、一个数据输入和一个输入控制。
如果输入控制信号无效，则三态门的输出为高阻态z。



(a) **bufif1** 高电平使能



(b) **notif1**

bufif1		控制输入			
		0	1	x	z
数据输入	0	z	0	0/z	0/z
	1	z	1	1/z	1/z
	x	z	x	x	x
	z	z	x	x	x

notif1		控制输入			
		0	1	x	z
数据输入	0	z	1	1/z	1/z
	1	z	0	0/z	0/z
	x	z	x	x	x
	z	z	x	x	x

3.2.4 门级建模举例 — 2选1数据选择器

//Gate-level description

module _2to1muxtri (a,b,sel,out);

input a,b,sel;

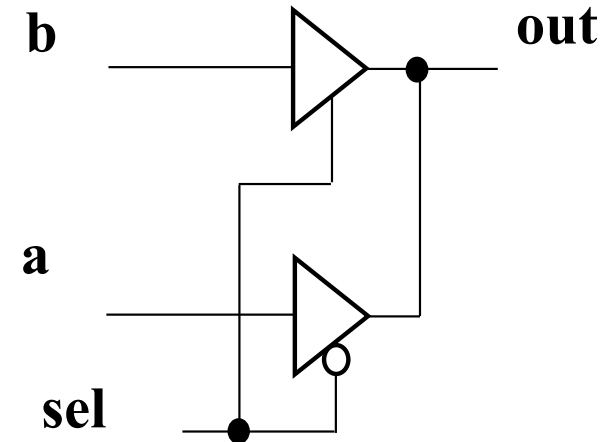
output out;

tri out;

bufif1 (out,b,sel);

bufif0 (out,a,sel);

endmodule



一些Verilog原型(Primitive)

always reg 行为描述

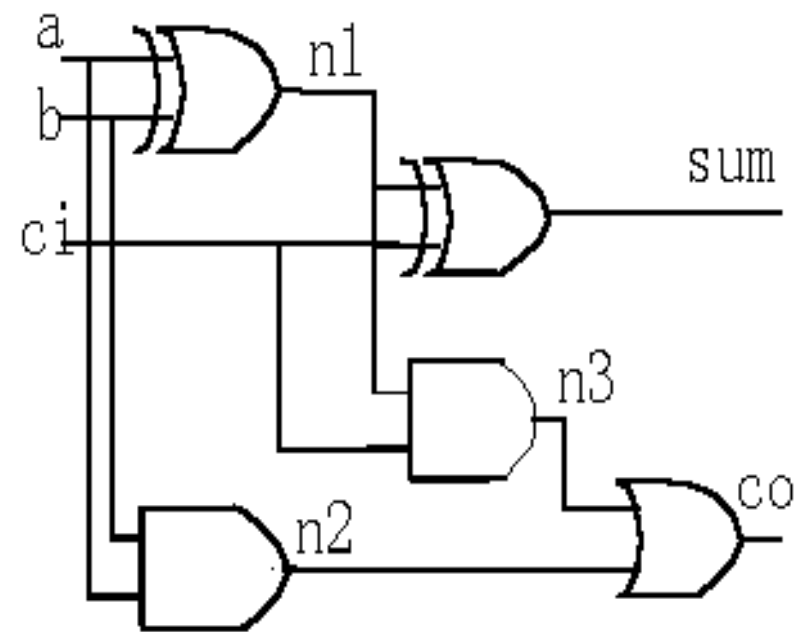
小结:

门级建模就是列出电路图结构中的元件，并按网表连接。

3.2.4 门级建模举例 — 1位全加器

$$\begin{aligned} \text{Sum} &= A \oplus B \\ \text{co} &= A \& B \end{aligned}$$

```
module addbit (a, b, ci, sum, co);  
input  a, b, ci;  
output sum, co;  
wire  a, b, ci, sum, co, n1, n2, n3;  
{  
  xor  u0(n1, a, b),  
    u1(sum, n1, ci);  
  and  u2(n2, a, b),  
    u3(n3, n1, ci);  
  or   (co, n2, n3);  
}  
endmodule
```



门级描述小结:

1. 给电路图中的每个输入输出引脚 赋以 端口名.
2. 给电路图中每条内部连线 取上各自的连线名.
3. 给电路图中的每个逻辑元件取一个编号 (即 “调用名”).
4. 给所要描述的这个电路模块确定一个模块名.
5. 用 **module** 定义相应模块名的结构描述, 并将逻辑图中所有的输入输出端口名列入端口名表项中, 再完成对各端口的输入输出类型说明.
6. 依照电路图中的连接关系, 确定各单元之间端口信号的连接, 完成对电路图内部的结构描述.
7. 最后用 **endmodule** 结束模块描述全过程.

3.3 Verilog HDL数据流建模与运算符

3.3.1 数据流建模

3.3.2 运算符及其优先级

3.3 Verilog HDL数据流建模与运算符

- 对于基本单元逻辑电路，使用Verilog语言提供的门级元件模型描述电路非常方便。
- 但随着电路复杂性的增加，使用的逻辑门较多时，使用HDL门级描述的工作效率就很低。
- 本节介绍的**数据流建模**能够在较高的抽象级别描述电路的逻辑功能，并且通过**逻辑综合**软件，能够自动地将数据流描述转换成为门级电路。
- **数据流建模**主要使用逻辑表达式，所以要了解各种运算符和表达式。

3.3.1 数据流建模

□ 数据流建模使用的连续赋值语句，由关键词**assign**开始，后面跟着由**操作数**和**运算符**等组成的**逻辑表达式**。

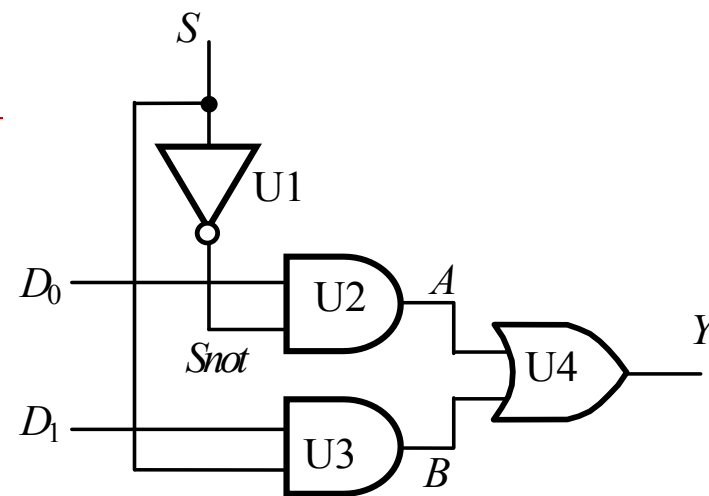
□ 一般用法如下：

```
wire [位宽说明] 变量名1, 变量名2, ....., 变量名n;  
assign 变量名 = 表达式;
```

□ 注意，**assign** 语句只能对**wire**型变量进行赋值，所以等号左边变量名的数据类型必须是**wire**型。

例 用数据流描述方式建立模型

$$Y = D_0 \cdot \bar{S} + D_1 \cdot S$$



```
module mux2to1_dataflow(D0, D1, S, Y );
```

```
  input D0, D1, S;
```

```
  output Y;
```

```
  wire Y ;
```

端口类型说明

数据类型说明

//下面是逻辑功能描述

```
  assign Y = (~S & D0) | (S & D1); //表达式左边Y必须是wire型
```

assign 必须接成网型

```
endmodule
```

电路结构描述

例：用条件运算符描述了一个2选1的数据选择器。

```
module mux2x1_df (D0,D1,S,L);  
    input D0,D1,S;  
    output L;  
  
    assign L = S ? D1 : D0;  
  
endmodule
```

条件运算符：如果S = 1，则输出L = D1；
否则L = D0。

例：用数据流建模方法对2线-4线译码器的行为进行描述。

```
module decoder_df (A1,A0,E,Y);
```

```
  input A1,A0,E;
```

```
  output [3:0] Y;
```

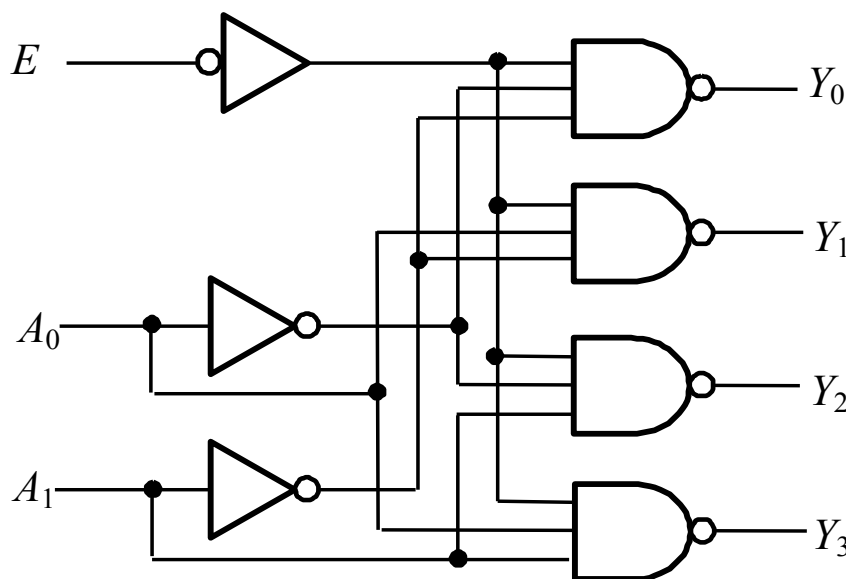
```
    assign Y[0] = ~(~A1 & ~A0 & ~E);
```

```
    assign Y[1] = ~(~A1 & A0 & ~E);
```

```
    assign Y[2] = ~(A1 & ~A0 & ~E);
```

```
    assign Y[3] = ~(A1 & A0 & ~E);
```

```
endmodule
```



3.3.2 运算符及其优先级

1.运算符 (9类)

运算符分类	所含运算符
算术运算符	$+, -, *, /, \%$
位运算符	$\sim, \&, , ^, \wedge \sim \text{or} \sim \wedge$
缩位运算符(单目)	$\&, \sim \&, , \sim , ^, \wedge \sim \text{or} \sim \wedge$
逻辑运算符	$!, \&\&, $
关系运算符 (双目)	$<, >, <=, >=$
相等与全等运算符	$==, !=, ===, !==$ <i>相等</i> <i>全等</i> <i>a 01x0 a==b b 01x0 a===b ⇒ 1</i>
逻辑移位运算符	$<<, >>$
连接运算符	$\{ \}$
条件运算符	$?:$

位拼接运算符

作用是将两个或多个信号的某些位拼接起来成为一个新的操作数，进行运算操作。

设 $A=1'b1$, $B=2'b10$, $C=2'b00$

则 $\{B,C\} = 4'b1000$

$\{A, \underline{B[1]}, C[0]\} = 3'b110$

$\{A, B, C, \underline{3'b101}\} = 8'b11000101$ 。

对同一个操作数的重复拼接还可以双重大括号构成的运算符 $\{\{\}\}$

例如 $\{4\{A\}\} = 4'b1111$, $\{2\{A\}, 2\{B\}, C\} = 8'b11101000$ 。

位运算符与缩位运算的比较

A: 4'b1010、
B: 4'b1111,

位运算	$\sim A = 0101$ $\sim B = 0000$	$A \& B = 1010$	$A B = 1111$	$A \wedge B = 0101$	$A \sim \wedge B = 1010$
缩位运算	$\&A = 1 \& 0 \& 1 \& 0 = 0$ $\sim \&A = 1$	$\&B = 1$	$ A = 1$ $\sim B = 0$	$\wedge A = 0$ $\wedge B = 0$	$\sim \wedge A = 1$ $\sim \wedge B = 1$

1010
01010

相等与全等运算符

$==$ (逻辑相等), $!=$ (逻辑不等)
 $===$ (条件全等), $!==$ (条件不全等)

a、b的初值同为4'b0100, c和d的初值同为4'b10x0

$a == b$	$a != b$	$a === b$	$a !== b$
1	0	1	0
$c == d$	$c != d$	$c === d$	$c !== d$
x	x	1	0

条件运算符

是三目运算符，运算时根据条件表达式的值选择表达式。


一般用法：

`condition_expr?expr1:expr2;`

首先计算第一个操作数`condition_expr`的值，如果结果为逻辑1，
则选择第二个操作数`expr1`的值作为结果返回，结果为逻辑0，
选择第三个操作数`expr2`的值作为结果返回。

2. 运算符的优先级

优先级的顺序从下向上依次增加。

类型	符号	优先级别
取反	! ~ -(求2的补码)	最高优先级
算术	* / + -	
移位	>> <<	
关系	< <= > >=	
等于	== !=	
缩位	& ~& ^ ^~ ~	
逻辑	&& 	
条件	?:	最低优先级

always 寄存器 reg

3.4 组合电路的行为级建模

3.4 Verilog HDL行为级建模

行为级建模就是描述数字逻辑电路的**功能和算法**。

在Verilog中，行为级描述主要使用由关键词initial或always定义的两种结构类型的语句。一个模块的内部可以包含多个**initial或always语句**。

initial语句是一条初始化语句，仅执行一次，经常用于测试模块中，对激励信号进行描述，在硬件电路的行为描述中，有时为了仿真的需要，也用initial语句给寄存器变量赋初值。

initial语句主要是一条面向**仿真**的过程语句，**不能用于逻辑综合**。这里不介绍它的用法。

在always结构型语句内部有一系列过程性赋值语句，**用来描述电路的功能（行为）**。

3.4.1 行为级建模基础

下面介绍行为级建模中经常使用的语句：

1. always语句结构及过程赋值语句

2. 条件语句 (if-else)

3. 多路分支语句 (case-endcase)

4. for循环语句 (例如 for等)

1. always语句的一般用法:

always @ (事件控制表达式)

begin: 块名

块内局部变量的定义;

过程赋值语句 (包括高级语句);

end

- “@”称为事件控制运算符，用于挂起某个动作，直到事件发生。“事件控制表达式”也称为敏感事件表，它是后面begin和end之间的语句执行的条件。当事件发生或某一特定的条件变为“真”时，后面的过程赋值语句就会被执行。
- begin...end 之间只有一条语句时，关键词可以省略;
- begin...end 之间的多条语句被称为顺序语句块。可以给语句块取一个名字，称为有名块。

2、条件语句（if语句）

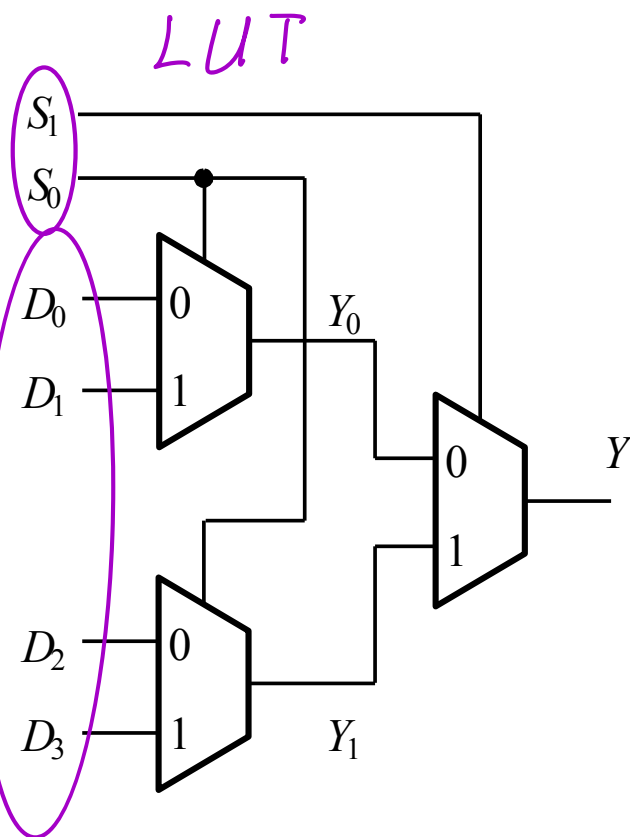
条件语句就是根据判断条件是否成立，确定下一步的运算。

Verilog语言中有3种形式的if语句：

- (1) `if (condition_expr) true_statement;`
- (2) `if (condition_expr) true_statement;
else false_statement;`
- (3) `if (condition_expr1) true_statement1;
else if (condition_expr2) true_statement2;
else if (condition_expr3) true_statement3;
.....
else default_statement;`

if后面的条件表达式一般为逻辑表达式或关系表达式。执行if语句时，首先计算表达式的值，若结果为0、x或z，按“假”处理；若结果为1，按“真”处理，并执行相应的语句。

例：使用if-else语句对4选1数据选择器的行为进行描述



```
module mux4to1_bh(D, S, Y);  
    input [3:0] D; //输入端口  
    input [1:0] S; //输入端口  
    output reg Y; //输出端口及变量数据类型  
    always @(D, S) //电路功能描述  
    if (S == 2'b00)    Y = D[0];  
    else if (S == 2'b01) Y = D[1];  
    else if (S == 2'b10) Y = D[2];  
    else                Y = D[3];  
endmodule
```

注意，过程赋值语句只能给寄存器型变量赋值，因此，输出变量Y的数据类型定义为reg。

3、多路分支语句（case语句）

begin 是一种多分支条件选择语句，一般形式如下

case (case_expr)

item_expr1: statement1;

item_expr2: statement2;

.....

default: default_statement; //default语句可以省略

endcase

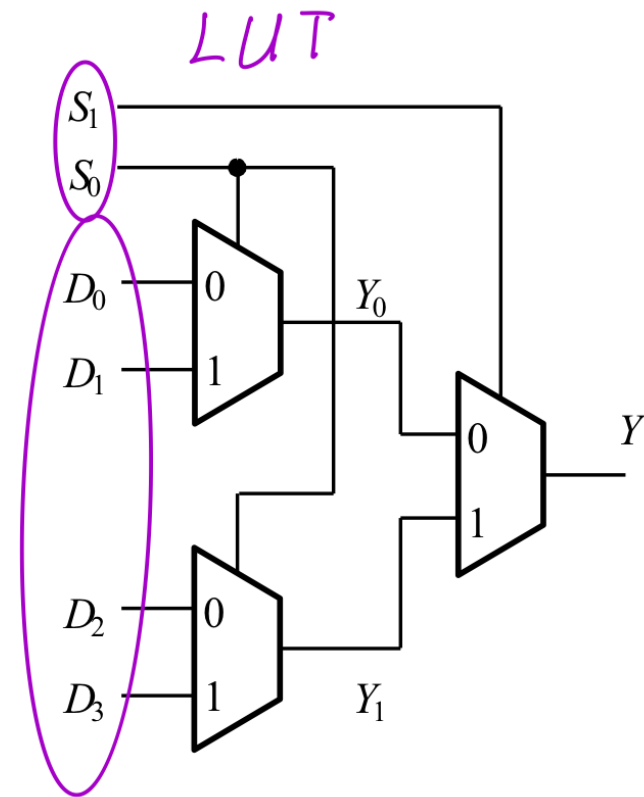
end

注意：当分支项中的语句是多条语句，必须在最前面写上关键词**begin**，在最后写上关键词**end**，成为顺序语句块。

另外，用关键词**casex**和**casez**表示含有无关项x和高阻z的情况。

例：对具有使能端En 的4选1数据选择器的行为进行Verilog描述。
当En=0时，数据选择器工作， En=1时，禁止工作，输出为0。

```
module mux4to1_bh (D, S, Y, En);  
    input [3:0] Assign D, [1:0] S;    input En;  
    output reg Y;  
    always @(D, S, En) //2001, 2005 syntax  
    begin  
        if (En==1) Y = 0; //En=1时，输出为0  
        else //En=0时，选择器工作  
            case (S)  
                2'd0: Y = D[0];  
                2'd1: Y = D[1];  
                2'd2: Y = D[2];  
                2'd3: Y = D[3];  
            endcase  
        end  
    end  
endmodule
```



4、for循环语句 *在 begin 之间 end*

一般形式如下

```
for (initial_assignment; condition; step_assignment)  
    statement;
```

□ initial_assignment 为循环变量的初始值。

□ condition 为循环的条件，

➤ 若为真，执行过程赋值语句 statement，

➤ 若不成立，循环结束，执行for后面的语句。

□ step_assignment 为循环变量的步长，每次迭代后，循环变量将增加或减少一个步长。

试用Verilog语言描述具有高电平使能的3线-8线译码器。

```
module ecoder3to8_bh(A,En,Y);
```

```
    input [2:0] A,  En;
```

```
    output reg [7:0] Y;
```

```
    integer k;      //声明一个整型变量k
```

```
    always @(A, En) //
```

```
        begin
```

```
            Y = 8'b1111_1111; //设译码器输出的默认值
```

```
            for(k = 0; k <= 7; k = k+1) //下面的if-else语句循环8次
```

```
            { if ((En==1) && (A== k) )
                Y[k] = 0; //当En=1时, 根据A进行译码
              else
```

```
                Y[k] = 1; //处理使能无效或输入无效的情况
```

```
            end
```

```
        endmodule
```

3位二进制译码器真值表

输 入			输 出							
A ₂	A ₁	A ₀	Y ₇	Y ₆	Y ₅	Y ₄	Y ₃	Y ₂	Y ₁	Y ₀
0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	1	0	0
0	1	1	0	0	0	0	1	0	0	0
1	0	0	0	0	0	1	0	0	0	0
1	0	1	0	0	1	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0

初始为0

试用Verilog语言描述具有高电平使能的3线-8线译码器。

```
module ecoder3to8_bh(A,En,Y);  
    input [2:0] A, En;  
    output reg [7:0] Y;  
    integer k; //声明一个整型变量k  
    always @(A, En) //  
        begin  
            Y = 8'b0000_0000; //设译码器输出的默认值  
            for(k = 0; k <= 7; k = k+1) //下面的if-else语句循环8次  
                if ((En==1) && (A== k) )  
                    Y[k] = 1; //当En=1时，根据A进行译码  
                else  
                    Y[k] = 0; //处理使能无效或输入无效的情况  
        end  
endmodule
```

3位二进制译码器真值表

输入			输出							
A ₂	A ₁	A ₀	Y ₇	Y ₆	Y ₅	Y ₄	Y ₃	Y ₂	Y ₁	Y ₀
0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	1	0	0
0	1	1	0	0	0	0	1	0	0	0
1	0	0	0	0	0	1	0	0	0	0
1	0	1	0	0	1	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0

3.5 分层次的电路设计方法

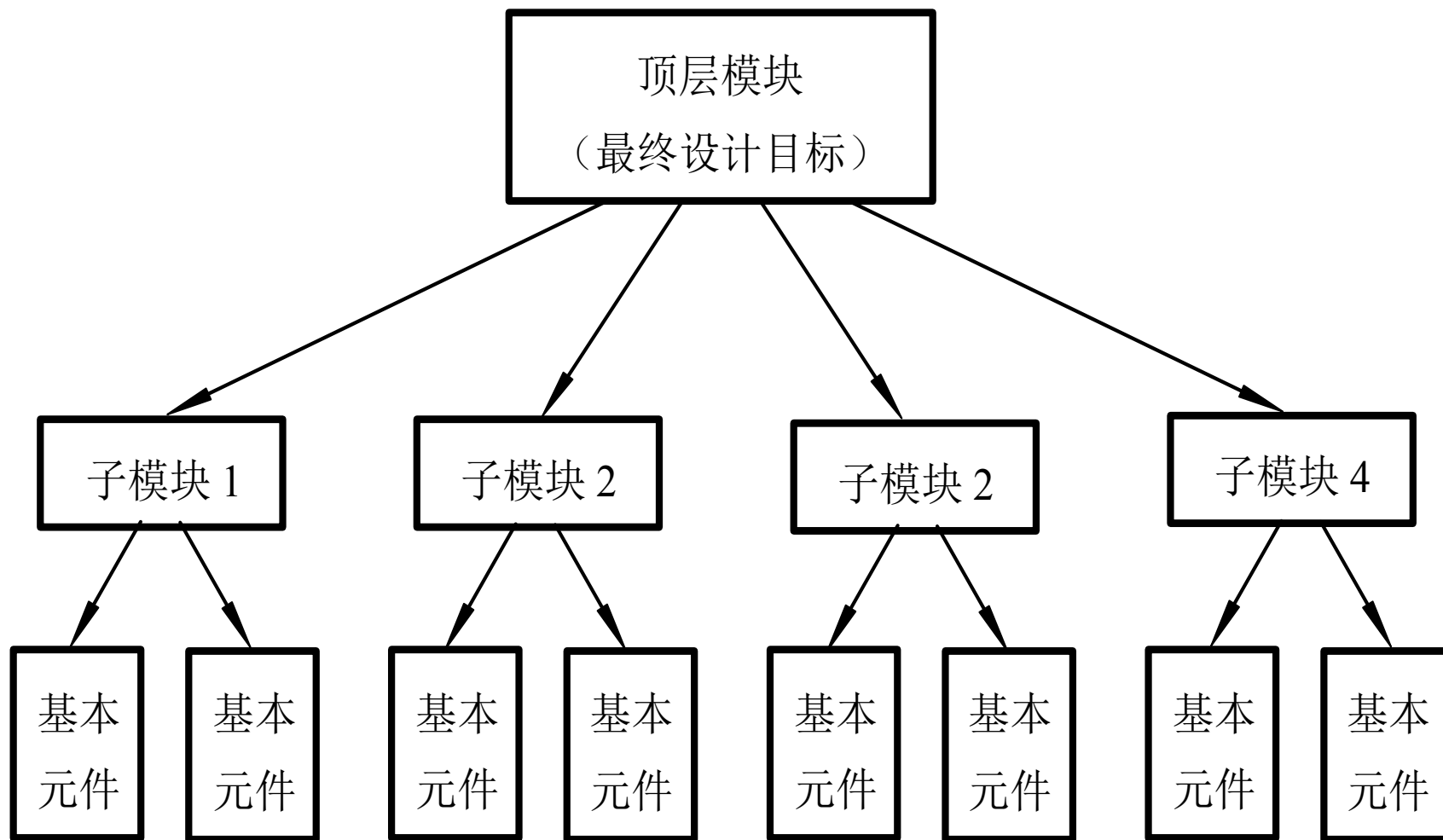
3.5.1 设计方法

3.5.2 模块实例引用语句

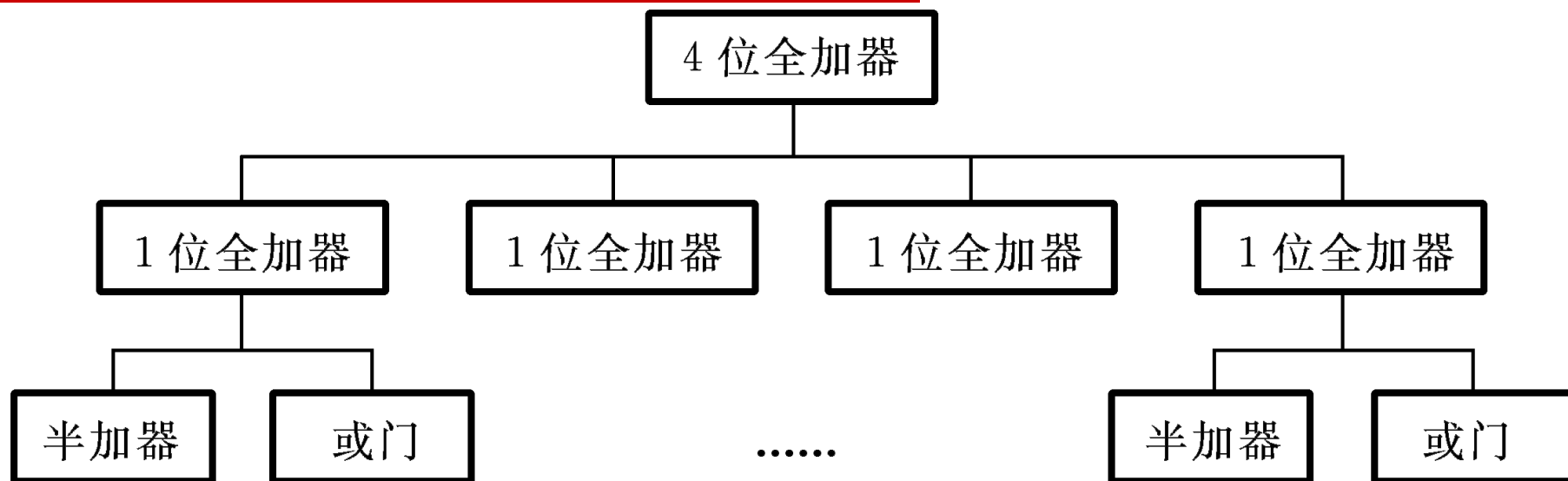
3.5.1 设计方法

- ❑ 分层次建模就是将一个比较复杂数字电路划分为多个组成模块，分别对每个模块建模，然后将这些模块组合成一个总模块，完成所需的功能。
- ❑ 通常有自顶向下（top-down）和自底向上（bottom-up）
- ❑ 自顶向下:先将最终设计目标定义成顶层模块，再按一定方法将顶层模块划分成各个子模块，然后对子模块进行逻辑设计。
- ❑ 自底向上:由基本元件构成的各个子模块首先被确定下来，然后将这些子模块组合起来构成顶层模块，最后得到所要求的电路。

3.5.1 设计方法



全加器电路设计举例



使用自下而上的方法：

- ❑ 实例引用基本门级元件xor、and定义底层的半加器模块halfadder;
- ❑ 实例引用两个半加器模块halfadder和一个基本或门元件or组合成为全加器模块fulladder;
- ❑ 实例引用4个1位的全加器模块fulladder构成4位全加器的顶层模块。

全加器电路设计举例

//***** 一位半加器的描述(参考图3.4.26) *****

module halfadder (S,C,A,B); //IEEE 1364—1995 Syntax

input A,B; //输入端口声明

output S,C; //输出端口声明

xor (S,A,B); //实例引用逻辑门原语

and (C,A,B);

endmodule

忽略数字类型的声明. 默认线网型

//***** 一位全加器的描述(参考图3.4.28) *****

module fulladder (Sum,Co,A,B,Ci);

input A,B,Ci; output Sum,Co;

wire S1,D1,D2; //内部节点信号声明

halfadder HA1 (.B(B),.S(S1),.C(D1),.A(A)); //实例引用底层模块halfadder

halfadder HA2 (.A(S1),.B(Ci), .S(Sum),.C(D2)); //端口信号按照名称对应关联

or g1(Co,D2,D1);

endmodule

全加器电路设计举例

//*****四位全加器的描述(参考图3.4.29)*****

```
module _4bit_adder (S,C3,A,B,C_1);
```

```
    input [3:0] A,B;
```

```
    input C_1;
```

```
    output [3:0] S;
```

```
    output C3;
```

```
    wire C0,C1,C2; //声明模块内部的连接线
```

```
    fulladder U0_FA (S[0],C0,A[0],B[0],C_1) ;//实例引用模块fulladder
```

```
    fulladder U1_FA (S[1],C1,A[1],B[1],C0) ; //端口信号按照位置顺序对应关联
```

```
    fulladder U2_FA (S[2],C2,A[2],B[2],C1) ;
```

```
    fulladder U3_FA (S[3],C3,A[3],B[3],C2);
```

```
endmodule
```

3.5.2 模块实例引用语句

模块实例引用语句的格式如下：

```
module_name instance_name (port_associations) ;
```

设计模块名

实例引用名

父、子模块端口的关联方式

位置关联法: 父模块与子模块的端口信号是按照位置（端口排列次序）对应关联的

名称关联法:

```
halfadder HA2 (.A(S1),  
                .B(Ci),  
                .S(Sum),  
                .C(D2)  
                );
```

实际名称
(父模块端口名)

形式名称
(子模块端口名)

关于模块引用的几点注意事项:

(1) 模块只能以实例引用的方式嵌套在其他模块内，嵌套的层次是没有限制的。但不能在一个模块内部使用关键词`module`和`endmodule`去定义另一个模块，也不能以循环方式嵌套模块，即不能在`always`语句内部引用子模块。

(2) 实例引用的子模块可以是一个设计好的Verilog HDL设计文件（即一个设计模块），也可以是FPGA元件库中一个元件或嵌入式元件功能块，或者是用别的HDL语言（如VHDL、AHDL等）设计的元件，还可以是IP（Intellectual Property，知识产权）核模块。

(3) 在一条实例引用子模块的语句中，不能一部分端口用位置关联，另一部分端口用名称关联，即不能混合使用这两种方式建立端口之间的连接。

关于模块引用的几点注意事项:

(4) 关于端口连接时有关变量数据类型的一些规定。

