

## 第四章 表达式

- 表达式由操作数和操作符组成；
- 可以用在期望数值出现的任何地方。

- 西北工业大学计算机学院韩 兵
- Email: hanbing@nwpu.edu.cn

## 4.1 操作数

- 1) 常数 // 256 (位数不确定的十进制数)、'hFB、4' b1011,
- 2) 参数 // parameter LOAD= 4' d12 (位数确定无符号)
- 3) 线网 // 若没有关键词signed, 值被解释为无符号数
- 4) 变量(寄存器) // 位数、有 / 无符号数
- 5) 位选择 : 从向量中抽取特定位 //  $State[x]$  值为x
- 6) 部分选择: 从向量中抽取相邻的若干位 //  $code[7: 4]$
- 7) 存储器元素(单元)  
reg [1:8] Ack, Dram[0 : 63] ; .....  
Ack=Dram[60];
- 8) 函数调用  $\$time + Sum\_Of\_Events(A, B)$
-

## 4.2 操作符

- 1) 算术操作符
- 2) 关系操作符
- 3) 相等操作符
- 4) 逻辑操作符
- 5) 按位操作符
- 6) 归约操作符
- 7) 移位操作符
- 8) 条件操作符
- 9) 连接和复制操作符
- 

优先级别

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

# 1) 算术操作符

- +、-、\*、/、%、\*\*幂运算 (2001版)
- 若操作数含x或z，其结果为x；
- 'b10x1 + 'b01111 结果 'bxxxxx
- 取模操作符  
求出与第一个操作符符号相同的余数
-

## 2) 关系操作符和相等关系操作符

- 比较两个已知的值，**返回一位**反映比较结果的 (T/F) 位 ；
- 如操作数**有一位是x或z**，**将返回x**；  
52 < 8 'hxFF     结果为x
- 如果操作数长度不同，**短操作数在左方添0补齐**
- 'b1000     >=   'b01110  
   'b01000   >=   'b01110
- ==, !=,     **===**全等,     **!==**非全等
- *Data* = 'b11x0; *Addr* = 'b11x0;
- *Data* == *Addr*     值为x
- *Data* === *Addr*   值为1
-

### 3) 逻辑操作符

- `&&` 逻辑与
- `||` 逻辑或
- `!` 逻辑非
- 非0向量按1对待
- 若  $A\_Bus = 'b0110$ ;  $B\_Bus = 'b0100$ ;  
 $A\_Bus \ || \ B\_Bus$  结果为1  
 $A\_Bus \ \&\& \ B\_Bus$  结果为1
- 若任意一个操作数包含x, 结果也为x  
 $!x$  结果为x
-

## 4) 按位操作符

- $\sim$  取反
- $\&$  按位与
- $|$  按位或
- $\wedge$  按位异或
- $\sim\sim$  按位异或非

这些操作符在输入操作数的对应位上按位操作，  
并产生向量结果。

$\&$ 与	0	1	x	z	$ $ 或	0	1	x	z
0	0	0	0	0	0	0	1	x	x
1	0	1	x	x	1	1	1	1	1
x	0	x	x	x	x	x	1	x	x
z	0	x	x	x	z	x	1	x	x

$\wedge$ 异或	0	1	x	z	$\sim\wedge$ 异或非	0	1	x	z
0	0	1	x	x	0	1	0	x	x
1	1	0	x	x	1	0	1	x	x
x	x	x	x	x	x	x	x	x	x
z	x	x	x	x	z	x	x	x	x

$\sim$ 非	0	1	x	z
	1	0	x	x

- 若:  $m = 4'b0101$ ;  $n = 4'b0011$ ;  $p = 4'b010z$ ;  
 $\sim m$  是  $4'b1010$      $m \& n$  是  $4'b0001$      $m \& p$  是  $4'b010x$   
 $\sim p$  是  $4'b101x$      $m \wedge n$  是  $4'b0110$      $m \sim \wedge n$  是  $4'b1001$   
 高阻位作为未知处理

## 5) 归约(缩减)操作符

归约操作符在单一操作数的所有位上操作，并产生 1 位结果。归约操作符有：

- & (归约与)

如果存在位值为 0，那么结果为 0；若如果存在位值为 x 或 z，结果为 x；否则结果为 1。

- ~& (归约与非)

与归约操作符 & 相反。

- | (归约或)

如果存在位值为 1，那么结果为 1；如果存在位 x 或 z，结果为 x；否则结果为 0。

- ~| (归约或非)

与归约操作符 | 相反。

- ^ (归约异或)

如果存在位值为 x 或 z，那么结果为 x；否则如果操作数中有偶数个 1，结果为 0；否则结果为 1。

- ~^ (归约异或非)

与归约操作符 ^ 正好相反。

如下所示。假定，

A = 'b0110;

B = 'b0100;

那么：

B	结果为 1
& B	结果为 0
~ A	结果为 1



归约异或操作符用于决定向量中是否有位为  $x$ 。假定,

```
MyReg = 4'b01x0
```

那么:

$\wedge$ MyReg 结果为  $x$

上述功能使用如下的 if 语句检测:

```
if (^MyReg == 1'bx)
```

```
    $display ("There is an unknown in the vector MyReg !")
```

注意逻辑相等(==)操作符不能用于比较; 逻辑相等操作符比较将只会产生结果  $x$ 。全等操作符期望的结果为值1。

## 6) 移位操作符有

\*移位算符完成在单次操作中将一个二进制数移  $n$  位

\*对移出的位丢掉, 移入的位填 0

Operator	Usage	Description
<<	$m \ll n$	Shift $m$ left $n$ times
>>	$m \gg n$	Shift $m$ right $n$ times

Examples:    given:     $m = 4'b01x1;$   
                               $n = 4'b0010;$

$m \ll 1$  is  $4'b1x10$

$m \ll 2$  is  $4'bx100$

$m \gg 1$  is  $4'b001x$

$m \gg n$  is  $4'b0001$

## 7) 条件操作符

- *cond\_expr ? expr 1 : expr 2*
- 
- **wire** [0:2] *Student* = *Marks* > 18 ? *Grade\_A* : *Grade\_C*;
- **always**  
#5 *Ctr* = (*Ctr* != 25) ? (*Ctr* + 1) : 5;
-

## 8) 连接（拼接）操作

- $\{expr\ 1, expr\ 2, \dots, expr\ N\}$
- 
- **wire** [7:0] *Dbus*;
- **assign** *Dbus*[7:4]= $\{Dbus[0], Dbus[1], Dbus[2], Dbus[3]\}$
- **assign** *Dbus* =  $\{Dbus[3:0], Dbus[7:4]\}$  ;
- 由于 非定长 常数的 长度未知,  
不允许连接 非定长 常数;  
 $\{Dbus, 5\}$  //不允许连接操作非定长常数。

●

## 9) 复制

- 通过指定重复次数来执行操作
- $\{ \textit{repetition\_number} \{ \textit{expr1}, \textit{expr2}, \dots, \textit{exprN} \} \}$
- 
- $Abus = \{ 3 \{ 4'b1011 \} \};$   
// 位向量12 ' b1011\_1011\_1011)
- $Abus = \{ \{ 4 \{ Dbus[7] \} \}, Dbus \};$  /\*符号扩展\*/
- $\{ 3 \{ 1'b1 \} \}$  结果为111
- $\{ 3 \{ Ack \} \}$  结果  $\{ Ack, Ack, Ack \}$
-

## 4.3 表达式种类

常量表达式是在编译时就计算出常数值的表达式。通常，常量表达式可由下列要素构成：

- 1) 表示常量文字，如'b10和326。
- 2) 参数名，如RED的参数表明：

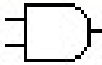
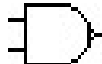


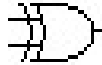

```
parameter RED = 4'b1110
```

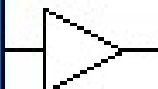
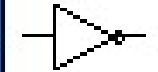
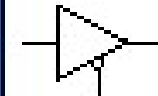
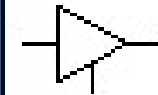
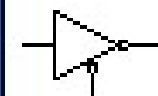
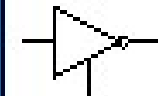
标量表达式是计算结果为1位的表达式。如果希望产生标量结果，但是表达式产生的结果为向量，则最终结果为向量最右侧的位值。



# 第五章 门电平模型化

## ● 5.1 预定义门级原型

	<u>and</u>	n-input AND gate
	<u>nand</u>	n-input NAND gate
	or	n-input OR gate
	nor	n-input NOR gate
	<u>xor</u>	n-input exclusive OR gate
	<u>xnor</u>	n-input exclusive NOR gate

	<u>buf</u>	n-output buffer
	not	n-output inverter
	bufif0	tri-state buffer; Lo enable
	bufif1	tri-state buffer; hi enable
	notif0	tri-state inverter; Lo enable
	notif1	tri-state inverter; hi enable



## 5.2 内置基本门

- 多输入门：
  - **and, nand, or, nor, xor, xnor**
- 多输出门：
  - **buf, not**
- 三态门：
  - **bufif0, bufif1, notif0, notif1**
- 上拉、下拉电阻：
  - **pullup, pulldown**
- MOS开关：
  - **cmos, nmos, pmos, rcmos, rnmos, rpmos**
- 双向开关：
  - **tran, tranif0, tranif1, rtran, rtranif0, rtranif1**
-

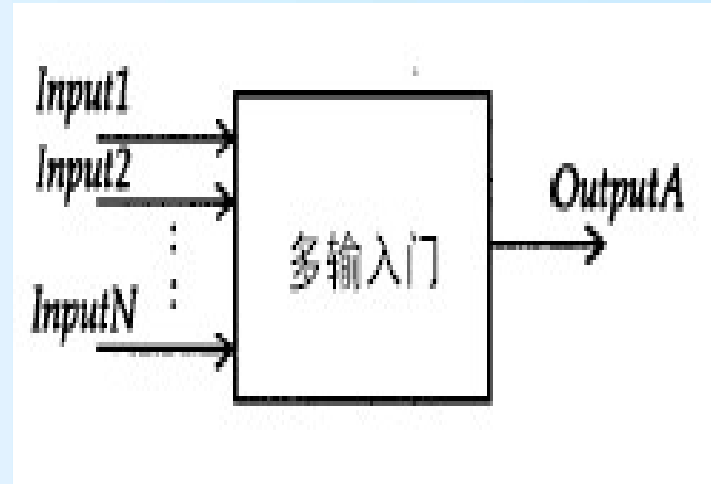
## 5.3 门例化

- 门实例语句：
- *gate\_type* [*instance\_name*] (*term1*, *term2*, ..., *termN*) ;
- *gate\_type* : 为设计需要的某种门类型；
- *Term* : 与门的输入/输出端口相连的线网或寄存器；
- 同一门类型的多个实例能够在一个结构形式中定义：

```
gate_type
```

```
[instance_name1] (term11, term12, . . ., term1N  
[instance_name2] (term21, term22, . . ., term2N  
. . .  
[instance_nameM] (termM1, termM2, . . ., termMN
```

## 5.4 多输入门



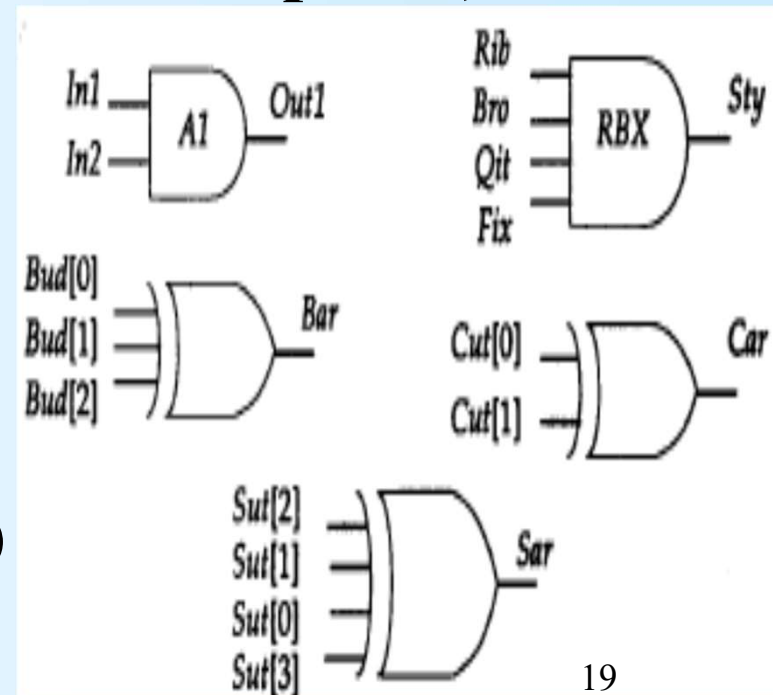
- **and , nand , nor , or , xor , xnor**

- 

- **multiple\_input\_gate\_type**

**[instance \_ name] (OutputA, Input1, ..., InputN) ;**

- **and** *A1* (*Out1*, *In1*, *In2*) ;
- **and** *RBX* (*Sty*, *Rib*, *Bro*, *Qit*, *Fix*) ;
- **xor** (*Bar*, *Bud*[ 0 ], *Bud*[1], *Bud*[ 2 ] ) ,  
(*Car*, *Cut* [0], *Cut*[ 1 ] ) ,  
(*Sar*, *Sut*[2], *Sut*[1], *Sut*[0], *Sut*[ 3 ] )



## ● 多输入门真值表

<b>nand</b>	0	1	x	z
0	1	1	1	1
1	1	0	x	x
x	1	x	x	x
z	1	x	x	x

<b>and</b>	0	1	x	z
0	0	0	0	0
1	0	1	x	x
x	0	x	x	x
z	0	x	x	x

<b>or</b>	0	1	x	z
0	0	1	x	x
1	1	1	1	1
x	x	1	x	x
z	x	1	x	x

<b>nor</b>	0	1	x	z
0	1	0	x	x
1	0	0	0	0
x	x	0	x	x
z	x	0	x	x

<b>xor</b>	0	1	x	z
0	0	1	x	x
1	1	0	x	x
x	x	x	x	x
z	x	x	x	x

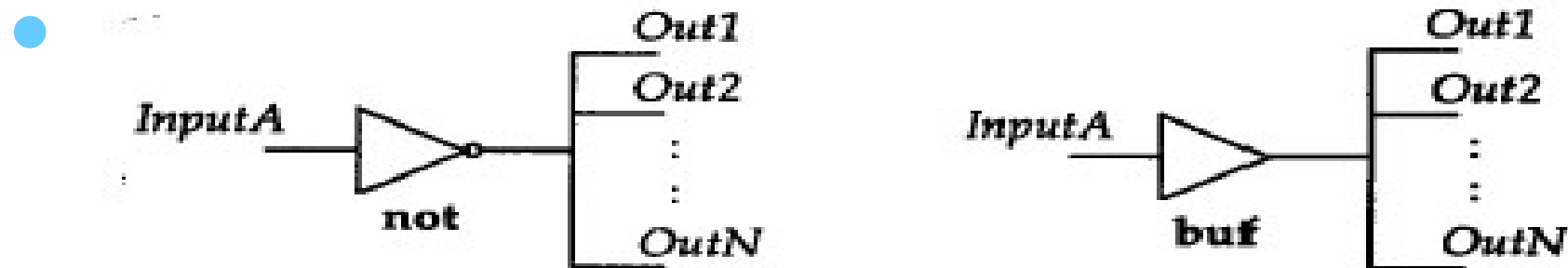
<b>xnor</b>	0	1	x	z
0	1	0	x	x
1	0	1	x	x
x	x	x	x	x
z	x	x	x	x

## 5.5 多输出门

- **buf , not**

- *multiple\_output\_gate\_type*

*[instance\_name] (Out1, Out2, ... OutN, InputA) ;*



```
buf B1 (Fan [0], Fan [1], Fan [2], Fan [3], Clk) ;
```

```
not N1 (PhA, PhB, Ready) ;
```

这些门的真值表如下::

buf	0	1	x	z	not	0	1	x	z
(输出)	0	1	x	x	(输出)	1	0	x	x

## 5.6 三态门



- **bufif0** , **bufif1**, **notif0** , **notif1**
- *tristate\_gate* [*instance\_name*]  
(*OutputA*, *InputB*, *ControlC*) ;
- 根据控制输入，输出可被驱动到高阻状态，即值z
- **bufif1** *BF1* (*Dbus*, *MemData*, *Strobe*) ;
- **notif0** *NT 2* (*Addr*, *Abus*, *Probe*) ;
-

# 三态门真值表

- 2001版与1995版稍有不同，输出1/z、0/z的都为x

bufif0		控 制			
		0	1	x	z
数据	0	0	z	0/z	0/z
	1	1	z	1/z	1/z
	x	x	z	x	x
	z	x	z	x	x

bufif1		控 制			
		0	1	x	z
数据	0	z	0	0/z	0/z
	1	z	1	1/z	1/z
	x	z	x	x	x
	z	z	x	x	x

notif0		控 制			
		0	1	x	z
数据	0	1	z	1/z	1/z
	1	0	z	0/z	0/z
	x	x	z	x	x
	z	x	z	x	x

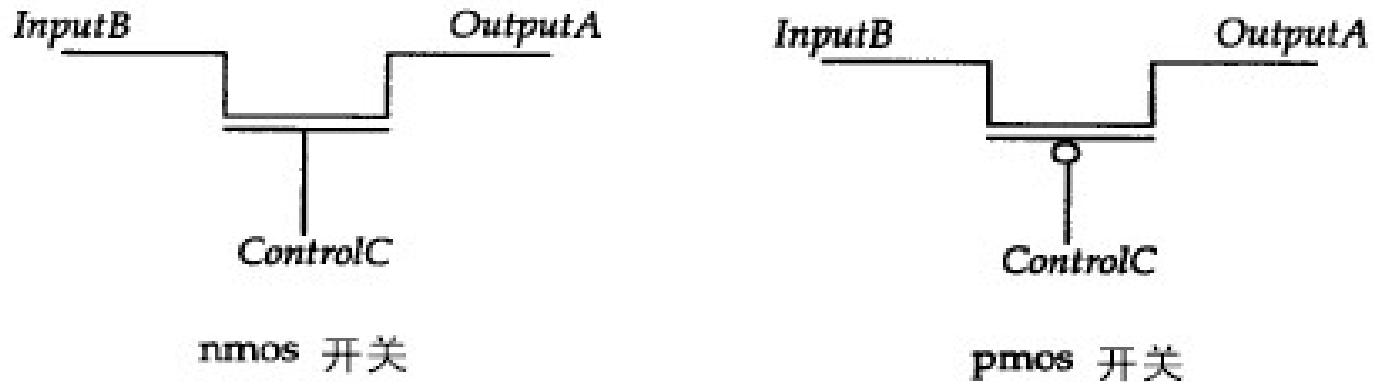
notif1		控 制			
		0	1	x	z
数据	0	z	1	1/z	1/z
	1	z	0	0/z	0/z
	x	z	x	x	x
	z	z	x	x	x

## 5.7 上拉、下拉电阻

- **pullup**                      **pulldown**
- *pull\_gate [instance\_name] (Output A) ;*
- 
- 端口表只包含1个输出
- 例: **pullup** *u0pullup* (core\_pwr) ;
-



## 5.8 MOS开关



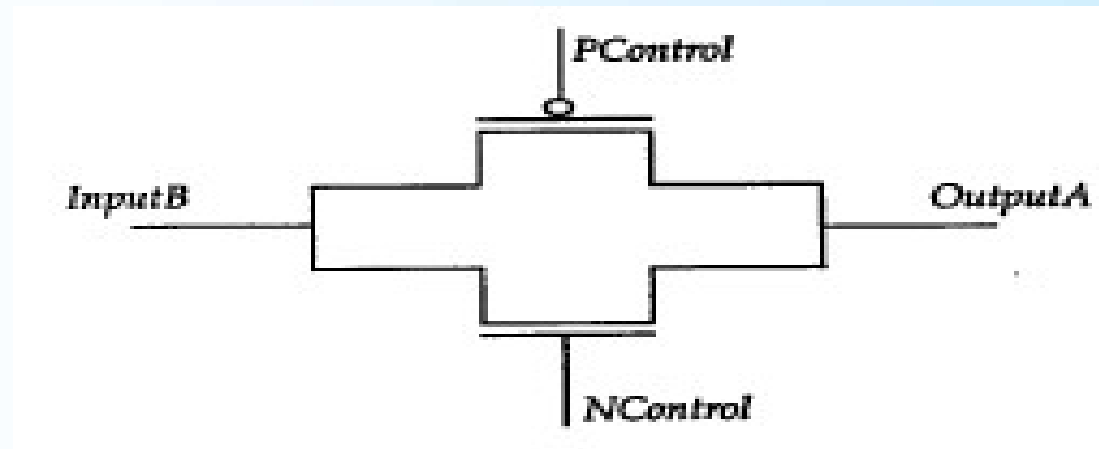
- **pmos nmos rpmos rnmos**
- 开关有一个输出、一个输入和一个控制输入
- *gate\_type [instance\_name] (OutputA, InputB, ControlC) ;*
- 带r的开关在输入引线和输出引线之间存在高阻抗。
- **pmos** *P1 (BigBus, SmallBus, GateControl) ;*
- **rnmos** *RN1 (ControlBit, ReadyBit, Hold) ;*
-

# MOS开关真值表

- 2001版与1995版稍有不同，输出1/z、0/z的都为x

pmos		控制				nmos		控制			
rmos						rmos					
		0	1	x	z			0	1	x	z
数据	0	0	z	0/z	0/z	数据	0	z	0	0/z	0/z
	1	1	z	1/z	1/z		1	z	1	1/z	1/z
	x	x	z	x	x		x	z	x	x	x
	z	z	z	z	z		z	z	z	z	z

## cmos rcmos



- **cmos** ( mos求补)和**rcmos** ( cmos的高阻态版本)开关
- 有一个数据输出，  
一个数据输入和两个控制输入。
- **(r)cmos** [*instance \_name*]  
(*OutputA*, *InputB*, *NControl*, *PControl*);
-

## 5.9 双向开关

- **tran rtran tranif0 rtranif0 tranif1 rtranif1**
- 数据可以**双向流动**，  
并且在开关中传播时**没有延时**；
- 后4个开关能够通过 设置合适的控制信号来 关闭；
- tran或rtran ( tran 的阻抗型)开关实例语句的语法：
- **( r ) tran [instance \_ name] (SignalA, SignalB) ;**
- 只有两个端口，并且无条件地**双向流动**；
- 其它双向开关的实例语句的语法：
- **gate \_ type [instance \_ name] (SignalA, SignalB, ControlC) ;**
-

## 5.10 门延迟

- 可定义门从任何输入到其输出的信号传输延迟；
- **gate\_type** [*delay*] [*instance \_ name*] (*terminal \_ list*) ;
- 由三类延迟值组成：
- 上升延迟    下降延迟    截止延迟
- **notif1** #(2,8,6) ( *Dout, in, cont*) ;
- 各种具体的时延取值情形。

	无时延	1个时延(d)	2个时延(d1, d2)	3个时延 (dA, dB, dC)
上升	0	d	d1	dA
下降	0	d	d2	dB
to_x	0	d	min① (d1, d2)	min (dA, dB, dC)
截止	0	d	min (d1, d2)	dC

## 5.11 实例数组

- 当需要重复性的实例时，在实例描述语句中能够有选择地定义范围说明
- *gate \_type* [*delay*] *instance\_name* [*leftbound:rightbound*] (*list\_of\_terminal\_names*);

- ```
wire [3:0] Out, InA, InB;
. . .
nand Gang [3:0] (Out, InA, InB;

nand
    Gang3 (Out[3], InA[3], InB[3]),
    Gang2 (Out[2], InA[2], InB[2]),
    Gang1 (Out[1], InA[1], InB[1]),
    Gang0 (Out[0], InA[0], InB[0]);
```

## 5.12 隐式线网

- 一个线网没有被特别说明，  
被缺省声明为1位线网；
- 
- 编译指令格式：`default\_nettype net\_type`
- 
- 例：`default\_nettype wand`
- 
- 在模块定义外出现，  
并且在下一个相同编译指令  
或`resetall`编译指令 出现前 一直有效。
-

## 例：2-4解码器举例

```
module DEC2x4 (A,B,Enable,Z);
```

```
    input A,B,Enable;
```

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

```
    wire Abar, Bbar;
```

```
    not # (1,2)
```

```
        V0 (Abar,A),
```

```
        V1(Bbar, B);
```

```
    nand # (4,3)
```

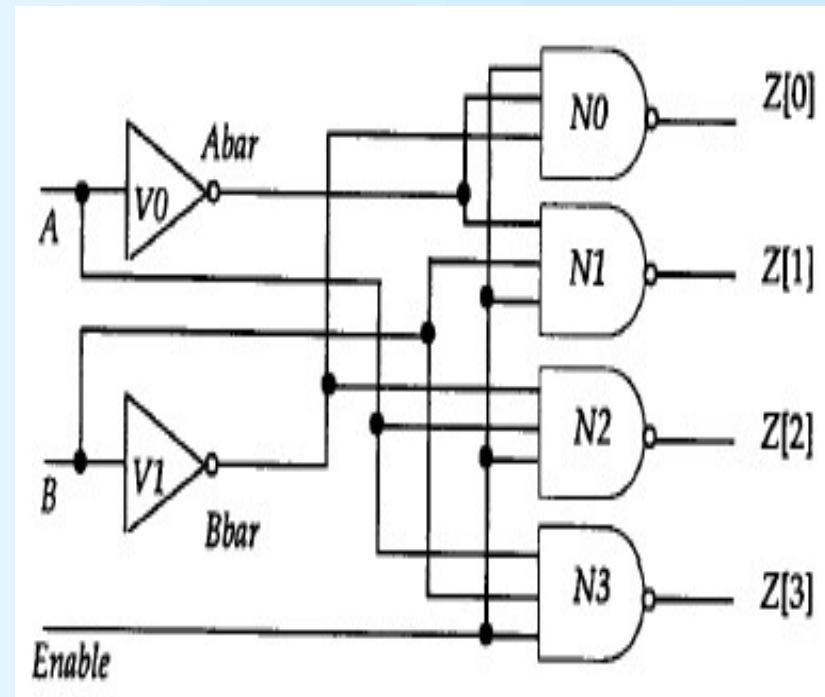
```
        N0 (Z[3], Enable, A,B,
```

```
        N1 (Z[0], Enable, Abar,Bbar,
```

```
        N2 (Z[1], Enable, Abar,B,
```

```
        N3 (Z[2], Enable, A,Bbar,
```

```
endmodule
```







## 第六章 用户定义原语

- UDP的实例语句  
与内置基本门的实例语句语法相同；
- **primitive** *UDP\_name* (*OutputName, List\_of\_inputs* )
- *Output\_declaration*
- *List\_of\_input\_declarations*
- [*Reg\_declaration*]
- [*Initial\_statement*]
- **table**
- *List\_of\_table\_entries* //UDP的行为以表的形式描述
- **endtable**
- **endprimitive**
-

- 在UDP中可以描述：
- 1) 组合电路
- 2) 时序电路(边沿触发和电平触发)

```

primitive MUX2x1 (Z, Hab, Bay, Sel) ;
    output Z;
    input Hab,Bay, Sel;

    table
// Hab Bay Sel : Z
    0  ?   1   : 0 ;
    1  ?   1   : 1 ;
    ?  0   0   : 0 ;
    ?  1   0   : 1 ;
    0  0   x   : 0 ;
    1  1   x   : 1 ;
    endtable
endprimitive

```

- 注：
- “?” 不必关心相应变量的具体值，0、1或x
- “-” 表示值“无变化”
- 如何由2选1多路器组成4选1多路器？

# 表项汇总

出于完整性考虑，下表列出了所有能够用于 UDP 原语中表项的可能值。

| 符 号 | 意 义        | 符 号  | 意 义                |
|-----|------------|------|--------------------|
| 0   | 逻辑0        | (AB) | 由A变到B              |
| 1   | 逻辑1        | *    | 与(??)相同            |
| x   | 未知的值       | r    | 上跳变沿，与(01)相同       |
| ?   | 0、1或x中的任一个 | f    | 下跳变沿，与(10)相同       |
| b   | 0或1中任选一个   | p    | (01)、(0x)和(x1)的任一种 |
| —   | 输出保持       | n    | (10)、(1x)和(x0)的任一种 |

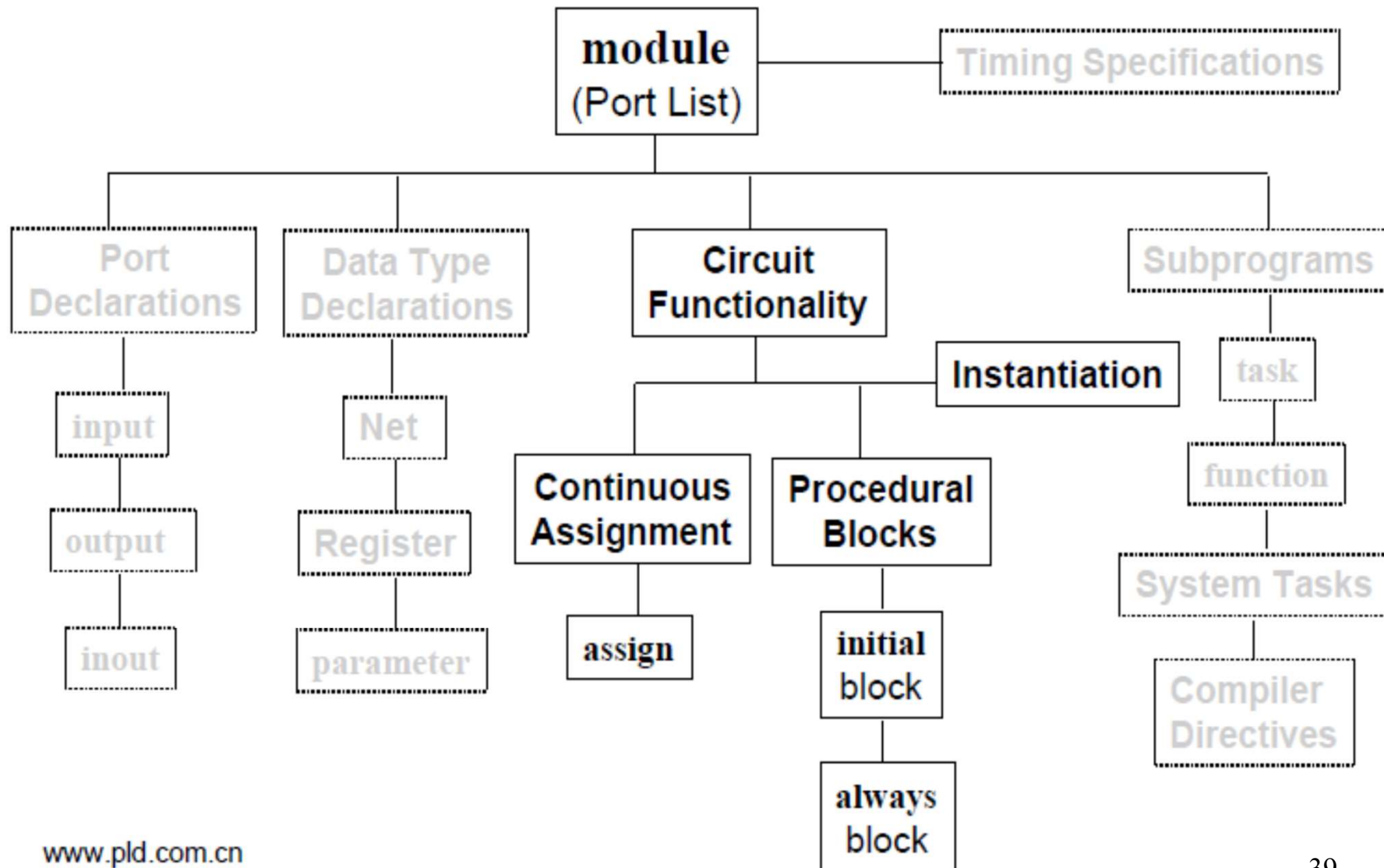
- 作业
- Verilog数字系统设计教程 (第3版), 夏宇闻, 北航
- 第4章 2、4、6
- 第8章 自学自查
- 第9章 1、3、5、7、10、11
- Verilog HDL入门(第3版), (美)巴斯克 著, 夏宇闻译, 北航
- 第4章 2、4、8
- 第5章 1、2、
- 第6章 1、3、

# 第七章 数据流模型化

- 对数据传递的直接描述；
- 对信号与数据之间的逻辑关系、与结构非常熟悉；
- 主要适用于算法描述，  
尤其在设计有关通信、数字或图像处理类电路时；
- 连续赋值 用于 数据流行为建模；
- 组合逻辑电路的建模。

●

# Components of a Verilog Module



## 7.1 连续赋值语句

- 连续赋值语句将值赋给线网（不能为寄存器赋值）
- `assign LHS_target = RHS_expression ;`
- 
- `wire [3:0] Z, Preset, Clear;`  
`assign Z = Preset & Clear;`
- 
- `assign Z = ~ (A | B) & (C | D) & (E | F) ;`  
`//何时赋值?`
- 
-



## 连续赋值的目标类型

- 1) 标量线网 `wire a;`
- 2) 向量线网 `wire [7:0] a;`
- 3) 向量的常数型位选择 `a[1]`
- 4) 向量的常数型部分选择 `a[3:1]`
- 5) 上述类型的任意的拼接形式。
-

# 例：1位全加器

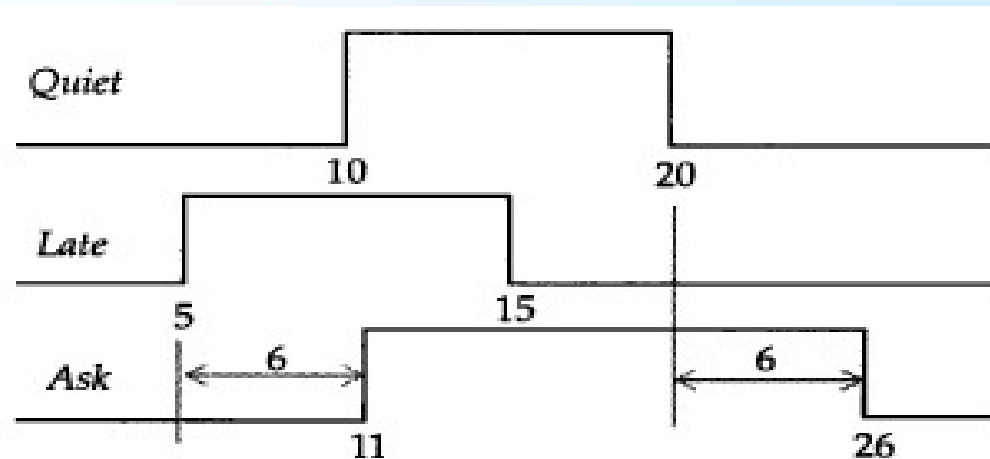
- module `full_adder_dataflow` (a, b, c\_in, sum, c\_out) ;
- input a, b, c\_in;
- output sum, c\_out ;
- 
- assign sum = a ^b ^c\_in;
- assign c\_out = (a & c\_in) | (b & c\_in) | (a&b) ;
- endmodule
- 
- 数据类型？ 位数？
- 连续赋值语句之间执行关系？
- 
-

## 7.2 线网声明赋值

- 连续赋值可作为线网声明本身的一部分，这样的赋值被称为**线网声明赋值**；
- `wire [3:0] Sum = 4'b0;`  
`wire Clear = 'b1;`  
`wire A_GT_B = A > B, B_GT_A = B > A;`
- 
- 线网声明赋值 **声明线网与连续赋值**：
- `wire Clear ;`  
`assign Clear = 'b1;`
- `wire Clear = 'b1;` //线网声明赋值：
-

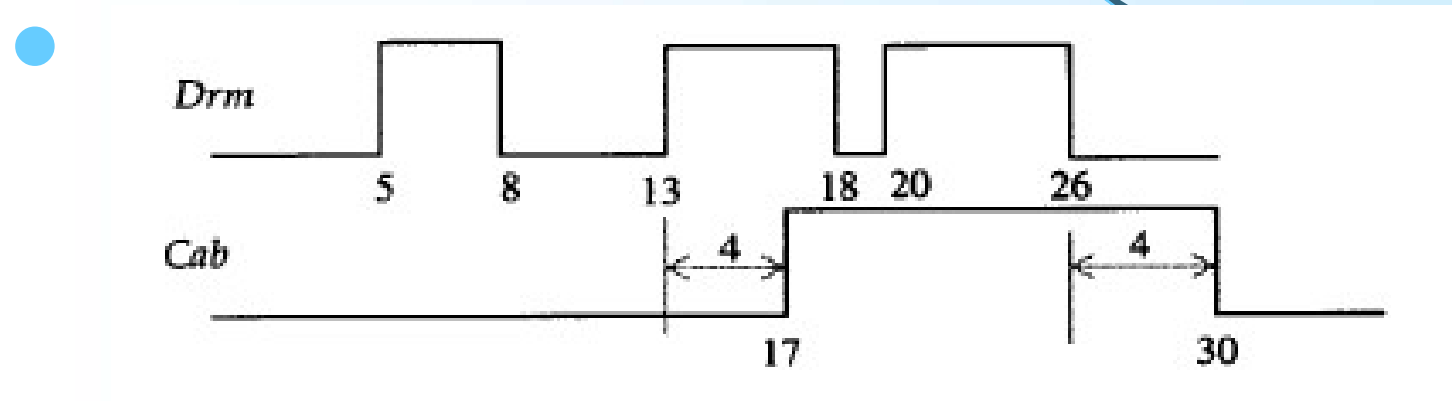
## 7.3 延迟

- 若在连续赋值语句中没有定义延迟，则右端表达式的值立即赋给左端表达式，时延为0；
- `assign #6 Ask = Quiet || Late;`
- 如果在时刻5，*Late*值发生变化，则赋值的右端表达式被计算，并且*Ask*在时刻11( = 5 +6)被赋予新值；



若右端在传输给左端前变化，结果如何？

- **assign #4 Cab = Drm;**
- 右端发生在延迟间隔内的变化被滤掉



与连续赋值语句延迟不同，过程赋值可以有以下两种延迟：

1) 语句间延迟：

$Sum = (A \wedge B) \wedge Cin;$

**#4**  $T1 = A \& Cin;$  // 开始执行本条语句前需要等待的时间

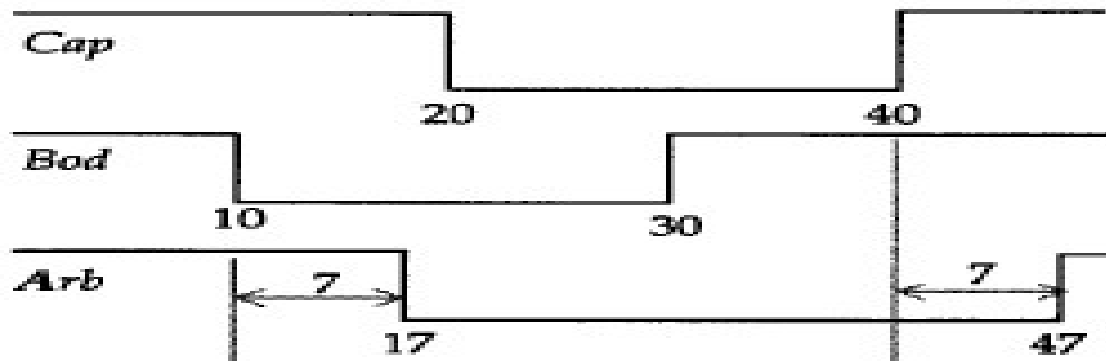
2) 语句内延迟：

$Sum = \text{\#3 } (A \wedge B) \wedge Cin;$  // 右式计算后到左式被赋值之间的时间

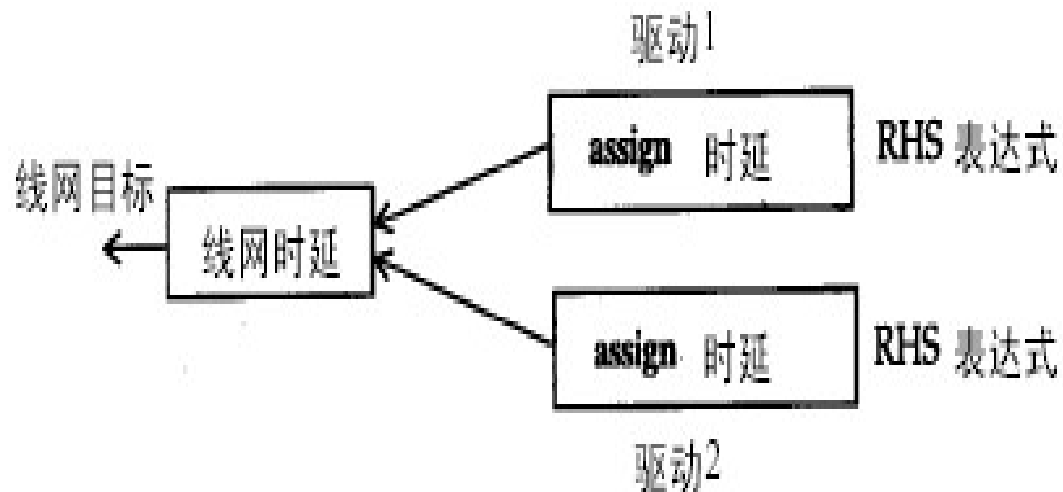
- 在每个延迟定义，总共能有三类延迟值可以被指定：
  - 上升延迟
  - 下降延迟
  - 截止延迟
- **assign** # (*rise, fall, turn-off*) *LHS\_target* = *RHS\_expression*;
- **assign** #4 *Ask* = *Quiet* || *Late*; // One delay value.
- **assign** # (4,8) *Ask* = *Quick* ; // Two delay values.
- **assign** # (4,8,6) *Arb* = & *DataBus*; // Three delay values.
- **assign** *Bus* = *MemAddr* [7:4]; // No delay value.
-

## 7.4 线网延迟

- 延迟也可以在线网说明中定义
- **wire #5 Arb;**
- **assign # 2 Arb = Bod & Cap;**
  - 假定在时刻10, *Bod*上发生事件, 右端表达式计算, 若新值, 则在2个时间单位后赋值给*Arb*, 即时刻12。但是因为定义了线网时延, *Arb*的赋值发生在时刻17( $= 10 + 2 + 5$ )。



# 线网延迟的效果



- 首先 使用赋值延迟，  
然后 增加任意线网延迟。
- 
- wire #2 A = B-C; // ? 延迟
- // 赋值延迟
-



# 例: 8位(参数化的)赋值比较器的数据流模型

- **module** *MagnitudeComparator* (*A*, *B*, *AgtB*, *AeqB*, *AltB* );
- **parameter** *BUS* = 8;                   // 参数定义
- **parameter** *EQ\_DELAY* = 5, *LT\_DELAY* = 8,  
                  *GT\_DELAY* = 8;
- **input** [1 : *BUS*] *A*, *B*;
- **output** *AgtB*, *AeqB*, *AltB*;
- 
- **assign** #*EQ\_DELAY* *AeqB* = *A* == *B*;
- **assign** #*GT\_DELAY* *AgtB* = *A* > *B*;
- **assign** #*LT\_DELAY* *AltB* = *A* < *B*;
- **endmodule**
-