



布尔代数和Verilog基础

刘学

13604181486





QQ: 672297876

liuxue@cse.neu.edu.cn

布尔代数

- 逻辑值“1”和“0”
 - 表示真和假，无大小之分
- 逻辑变量
 - 参与逻辑运算，取值非“0”即“1”
- 逻辑运算符
 - 用门电路实现
 - 非门、与门、或门、与非门、或非门、异或门、同或门
- 逻辑表达式
 - 常量“1”和“0” + 逻辑变量 + 逻辑运算符

逻辑门

Gate	Symbol	Truth-Table	Expression															
NAND		<table><tr><th>X</th><th>Y</th><th>Z</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	X	Y	Z	0	0	1	0	1	1	1	0	1	1	1	0	$Z = \overline{X \cdot Y}$
X	Y	Z																
0	0	1																
0	1	1																
1	0	1																
1	1	0																
AND		<table><tr><th>X</th><th>Y</th><th>Z</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	X	Y	Z	0	0	0	0	1	0	1	0	0	1	1	1	$Z = X \cdot Y$
X	Y	Z																
0	0	0																
0	1	0																
1	0	0																
1	1	1																
NOR		<table><tr><th>X</th><th>Y</th><th>Z</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	X	Y	Z	0	0	1	0	1	0	1	0	0	1	1	0	$Z = \overline{X + Y}$
X	Y	Z																
0	0	1																
0	1	0																
1	0	0																
1	1	0																
OR		<table><tr><th>X</th><th>Y</th><th>Z</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	X	Y	Z	0	0	0	0	1	1	1	0	1	1	1	1	$Z = X + Y$
X	Y	Z																
0	0	0																
0	1	1																
1	0	1																
1	1	1																

逻辑门

XOR
 $(X \oplus Y)$



X	Y	Z
0	0	0
0	1	1
1	0	1
1	1	0

$Z = X \bar{Y} + \bar{X} Y$
X or Y but not both
("inequality", "difference")

XNOR
 $\overline{(X \oplus Y)}$

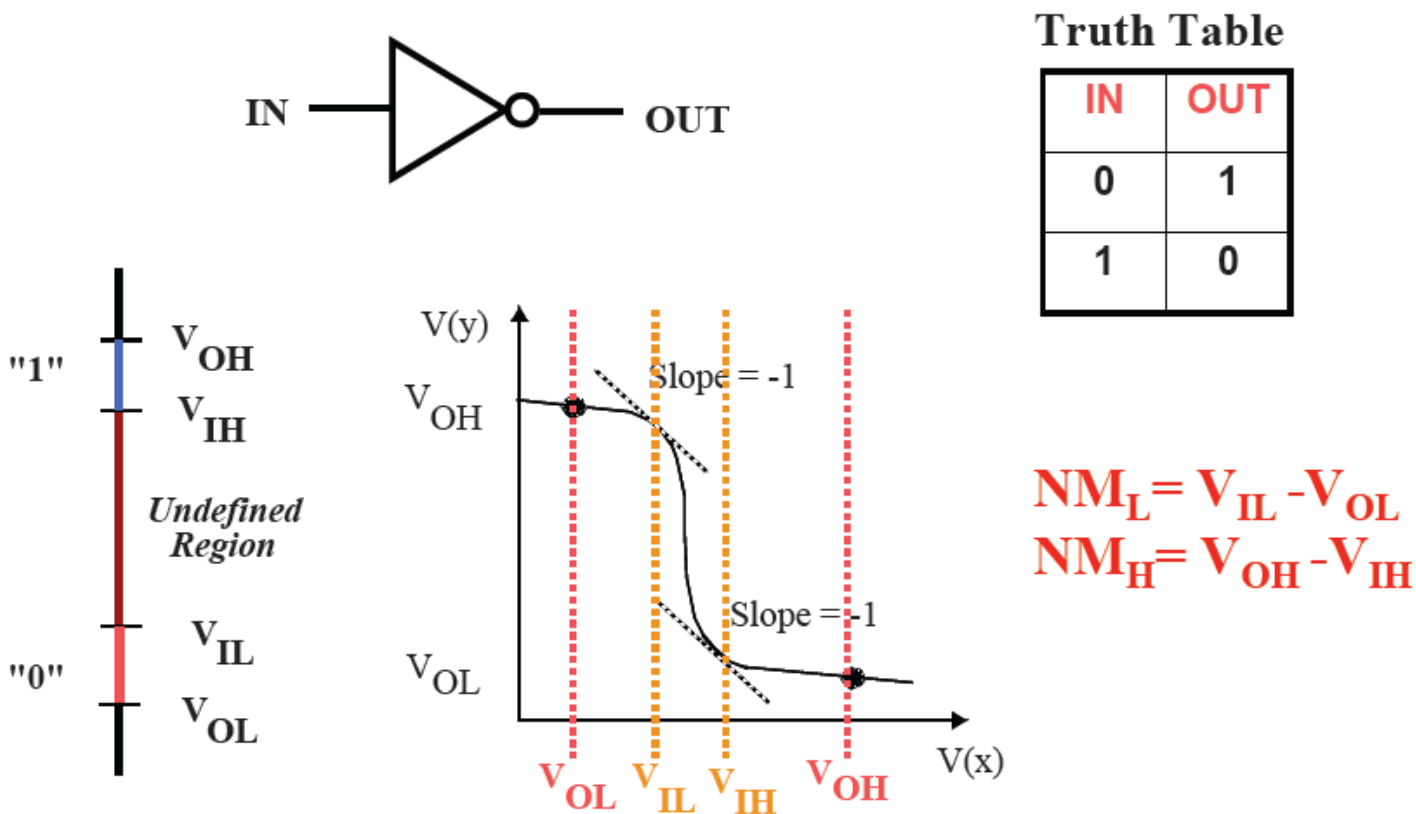


X	Y	Z
0	0	1
0	1	0
1	0	0
1	1	1

$Z = \bar{X} \bar{Y} + X Y$
X and Y the same
("equality")

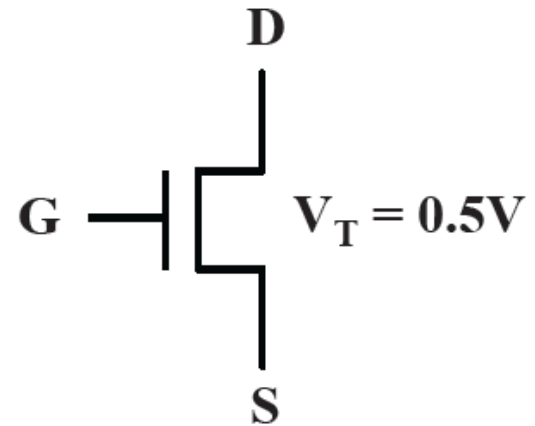
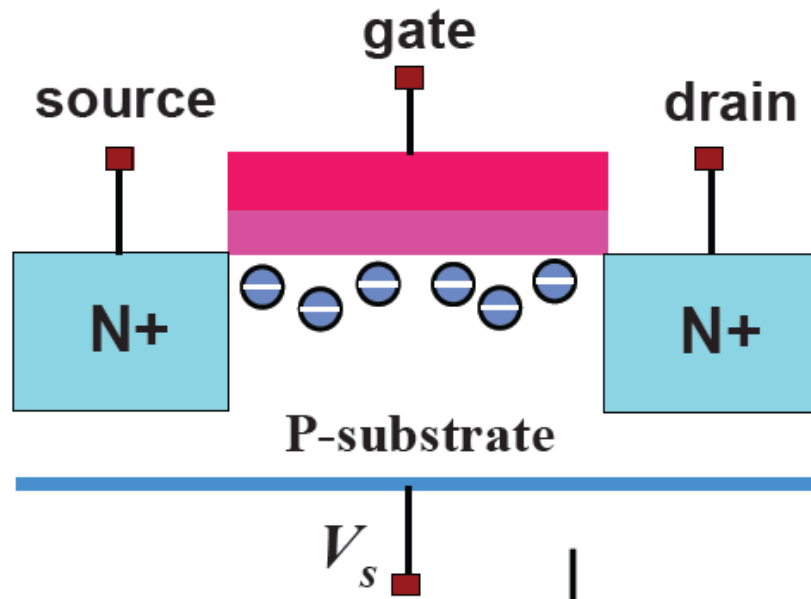
广泛用于加法器和乘法器等算术结构

非门：噪声容限(Noise margin)



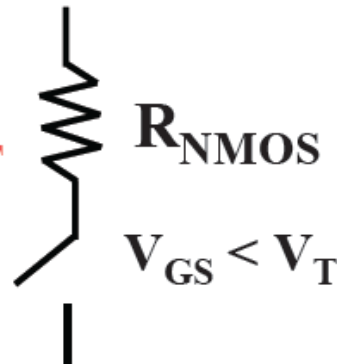
大噪声容限可防范各种噪声源

MOS Technology: NMOS Switch

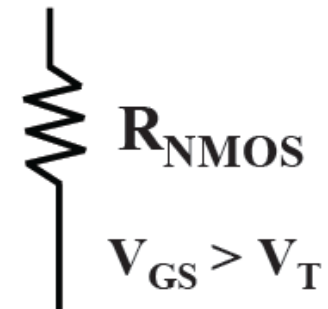


**Switch
Model**

OFF

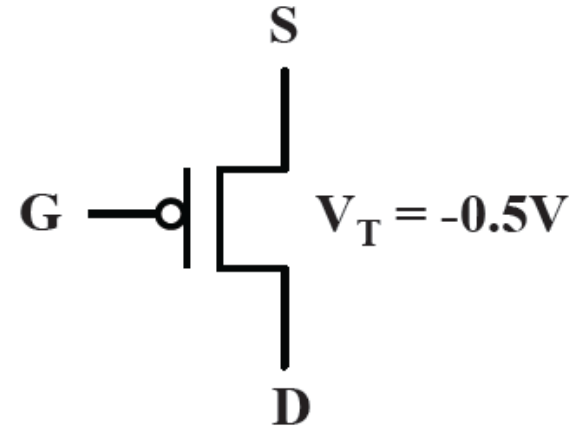
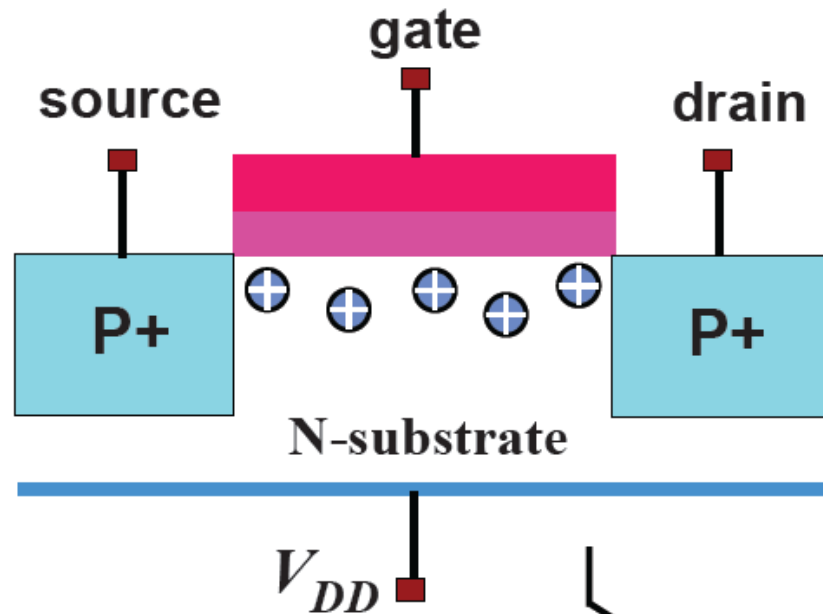


ON



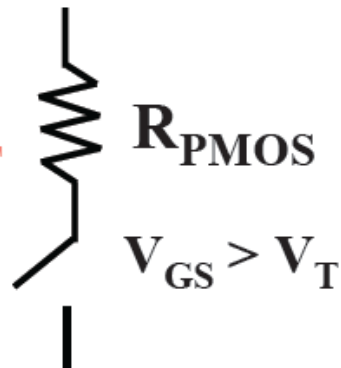
NMOS ON when Switch Input is High

PMOS: Complementary Switch

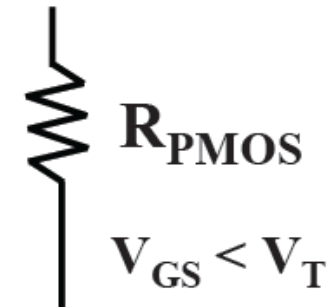


**Switch
Model**

OFF

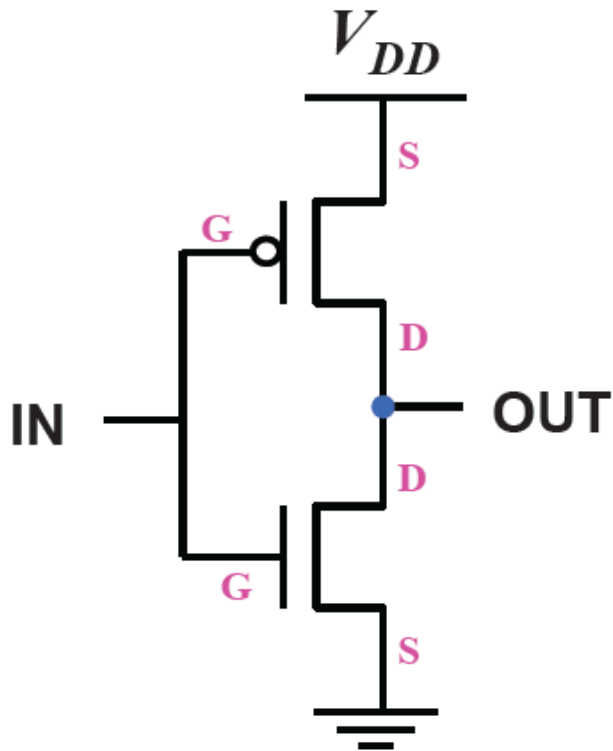


ON



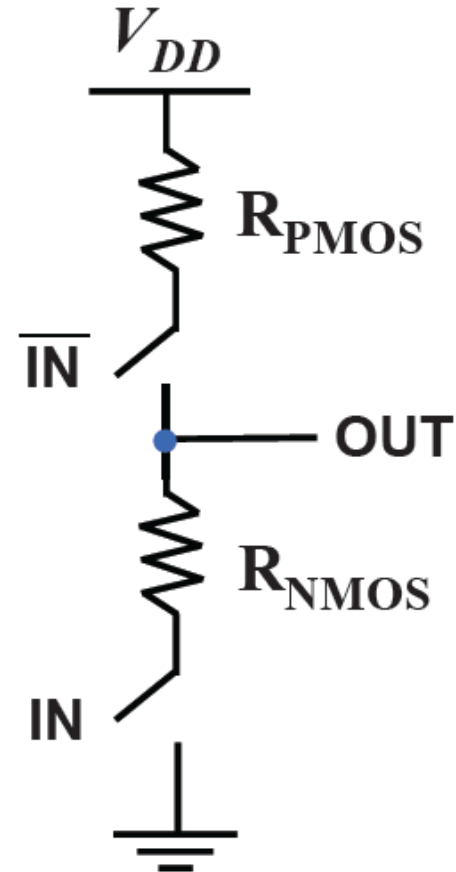
PMOS ON when Switch Input is Low

CMOS Inverter



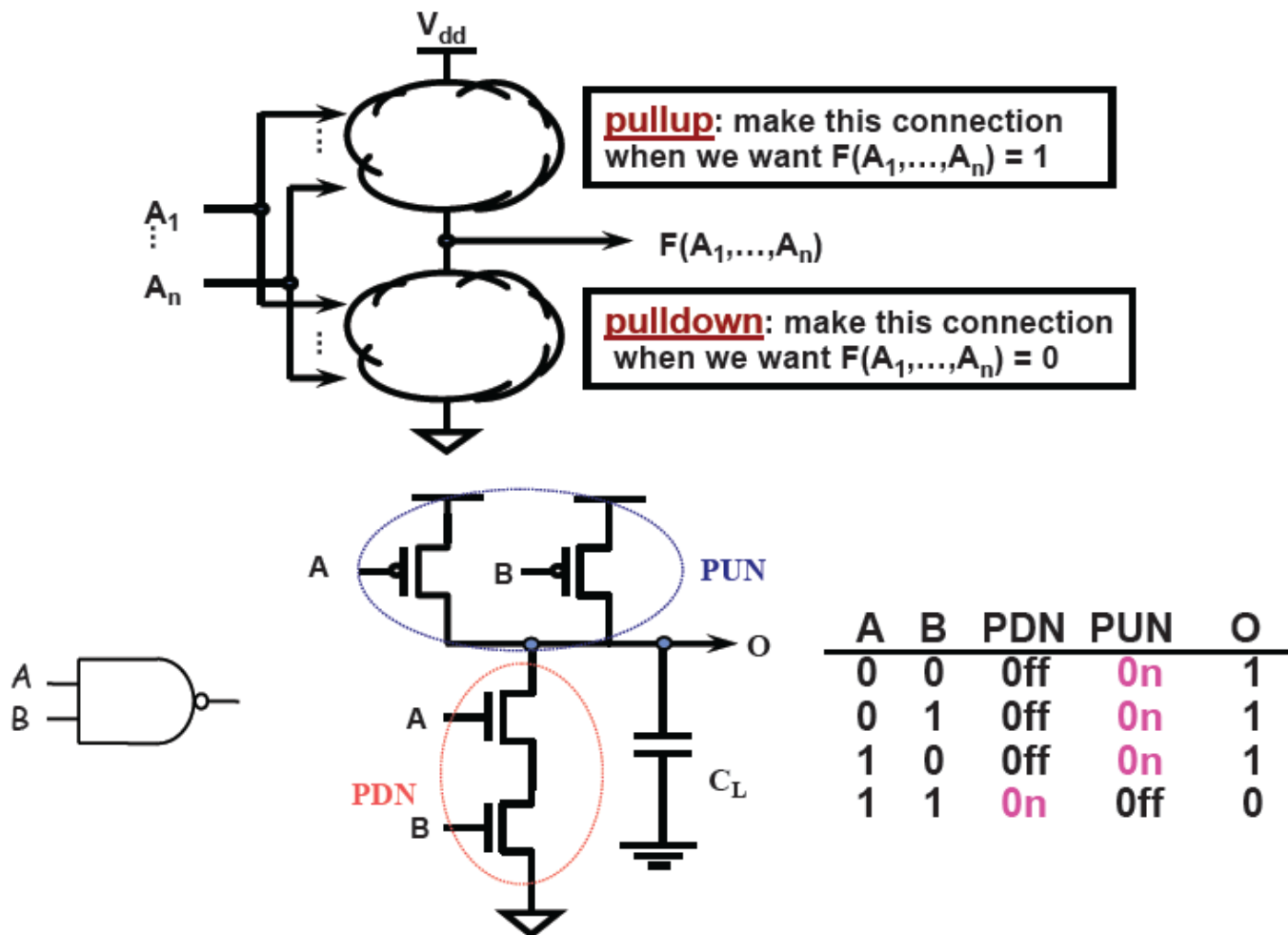
Rail-to-rail Swing in CMOS

Switch Model



CMOS逻辑门

与门?
或非门?
或门?



2输入逻辑函数

- 2输入变量对应16种可能的逻辑函数：



X	Y	16 possible functions (F_0 – F_{15})															
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	0	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
		X AND Y		X		Y		X XOR Y		X OR Y		X NOR Y NOT (X OR Y)		X = Y		NOT Y	
																NOT X	
																X NAND Y NOT (X AND Y)	

- n 输入变量对应 $2^{(2^n)}$ 种可能的逻辑函数

布尔代数基本定律

- 公理1 如 $A \neq 1$, 则 $A = 0$
 如 $A \neq 0$, 则 $A = 1$
- 公理2 $\overline{1} = 0$
 $\overline{0} = 1$
- 公理3 $0 \cdot 0 = 0$
 $1 + 1 = 1$

布尔代数基本定律

- 公理4 $0 \cdot 1 = 1 \cdot 0 = 0$
 $1 + 0 = 0 + 1 = 1$
- 公理5 $1 \cdot 1 = 1$
 $0 + 0 = 0$
- 交换律 $A \cdot B = B \cdot A$
 $A + B = B + A$
- 结合律 $A \cdot (B \cdot C) = (A \cdot B) \cdot C$
 $A + (B + C) = (A + B) + C$

布尔代数基本定律

- 分配律 $A(B + C) = A \cdot B + A \cdot C$
 $A + B \cdot C = (A + B)(A + C)$
- 控制律 $A \cdot 0 = 0$
 $A + 1 = 1$
- 自等律 $A \cdot 1 = A$
 $A + 0 = A$
- 重叠律 $A \cdot A = A$
 $A + A = A$

布尔代数基本定律

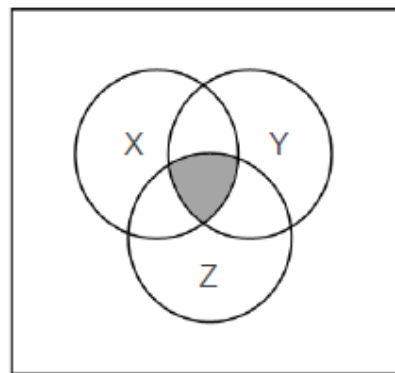
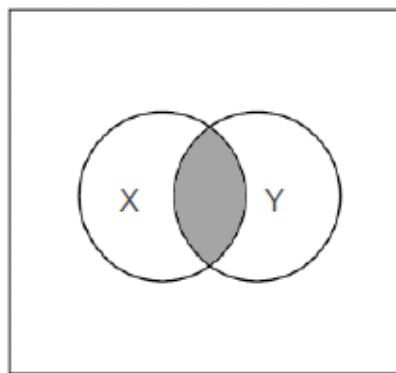
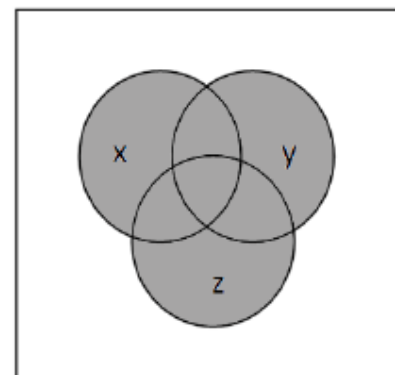
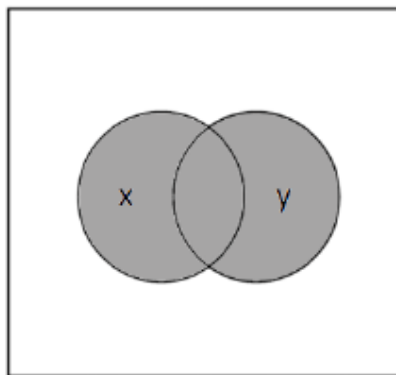
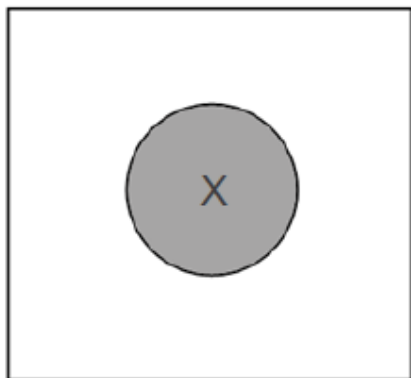
- 吸收律 $A + AB = A$
 $A \cdot (A + B) = A$

- 互补律 $A \cdot \bar{A} = 0$
 $A + \bar{A} = 1$

- 反演律（摩根定律）
 $\overline{A \cdot B} = \bar{A} + \bar{B}$
 $\overline{A + B} = \bar{A} \cdot \bar{B}$

- 双重否定律
 $\overline{\bar{A}} = A$

文氏图表示法



课堂练习

- 用逻辑代数的公理或定律证明下列等式

$$1. AB + \overline{A}C + \overline{B}C = AB + C$$

$$2. \overline{A\overline{B} + B\overline{C} + C\overline{A}} = ABC + \overline{A}\overline{B}\overline{C}$$

布尔代数基本规则

- 置换(Replacement)规则
 - 对于逻辑表达式中的任一变量X，若将所有出现它的地方都用逻辑函数G替换，等式仍然成立

$$A + AB\bar{C}(D + E) \rightarrow A + AG = A$$

布尔代数基本规则

- 对偶(Dual)规则
 - 所有逻辑常量和逻辑符号分别作1与0、+与·的变换
 - 必须保持原函数变量之间的运算顺序不变
- 反演(Invert)规则
 - 求一个函数的反函数，反演规则也称求反规则或求补规则
 - 逻辑常量、逻辑符号和逻辑变量分别作1与0、+与·、原变量与反变量的变换

求逻辑函数 $F = (X_1 \overline{X_2} + X_3)X_4$ 的反函数 \overline{F}

注意事项

- 不属于单个变量上的反号应保留不变
 - $F = \overline{AB} + \bar{C}(D + A)$ 中的 \overline{AB} 属于非运算，不是反变量，因而在反演时，非号须保留，该函数的反函数是 $\bar{F} = \overline{\overline{A} + \overline{B}}(C + \bar{D} \cdot \bar{A})$
而不是 $\bar{F} = (\bar{A} + \bar{B})(C + \bar{D} \cdot \bar{A})$
- 注意： $F + \bar{F} = 1$
- F 与 F' 是两个相互独立的函数，只是形式上的对偶

课堂练习

求下列逻辑函数的反函数 \bar{F} 和对偶函数 F'

$$1. F = \bar{A} \cdot \bar{B} + \overline{CD}$$

$$2. F = \overline{\bar{A} + \bar{B} + C}$$

布尔代数常用公式

- 并项公式

$$AB + \bar{A}B = B$$

- 消冗余因子公式

$$A + \bar{A}B = A + B$$

- 消冗余项公式

$$AB + \bar{A}C + BC = AB + \bar{A}C$$

推论:

$$AB + \bar{A}C + BCD = AB + \bar{A}C$$

- 消冗余因子公式

$$A \cdot \overline{A \cdot B} = A \cdot \bar{B}$$

$$\overline{A \cdot A \cdot B} = \bar{A}$$

逻辑函数及其描述方法

- 任何对 n 个逻辑变量 x_1, x_2, \dots, x_n 进行有限次逻辑运算的逻辑表达式，称为 n 变量的逻辑函数或简称函数，记作： $F = f(x_1, x_2, \dots, x_n)$
 - 逻辑表达式
 - 逻辑图
 - 真值表
 - 卡诺图
 - 标准表达式

逻辑函数描述示例

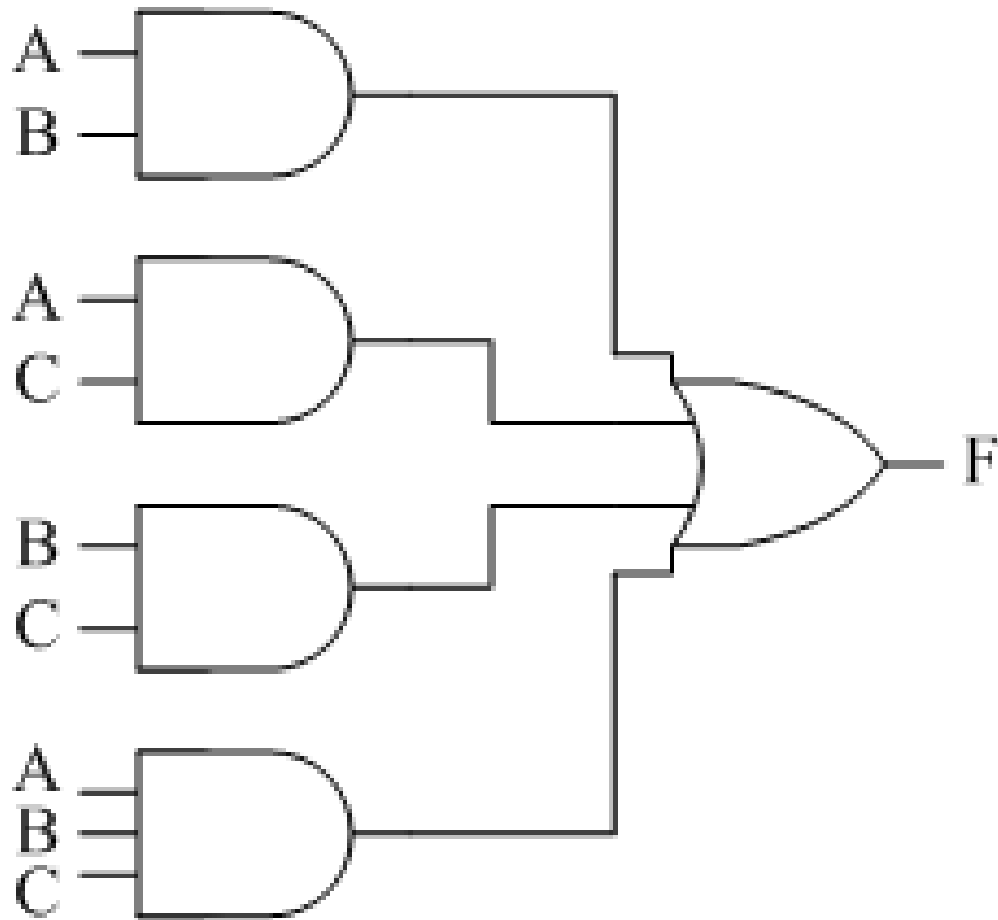
- 举重比赛有三名裁判，当运动员将杠铃举起后，须有两名或两名以上裁判认可，方可判定试举成功。用A、B、C分别表示三名裁判的意见输入，同意为1，否定为0；用F表示裁判结果输出，试举成功时为1，试举失败时为0。则F与A、B、C之间的关系可以用以下几种方式表示。

逻辑表达式

- 把逻辑函数的输入、输出关系写成与、或、非等逻辑运算的组合式，称为逻辑表达式
- 在上例中，用AB代表“裁判A、B都同意”，用BC和AC表示另两种有两名裁判同意的情况，ABC表示三名裁判都同意，所以

$$F = f(A, B, C) = AB + BC + AC + ABC$$

逻辑图



真值表

A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

唯一性

N个输入变量-> 2^n 种组合

最小项(minterm)

- 基本概念

- 由A、B、C三个逻辑变量构成的许多乘积项中，有八个被称为最小项，它们的特点是
 - 每项都只有三个因子
 - 每个变量都是它的一个因子
 - 每个变量以原变量或反变量形式出现一次
- 一般情况下，对n个变量来说，最小项共有 2^n 个，当n=3时，最小项有8个，它们是 $\overline{A}\overline{B}\overline{C}$, $\overline{A}\overline{B}C$, $\overline{A}B\overline{C}$, $\overline{A}BC$, $A\overline{B}\overline{C}$, $A\overline{B}C$, $AB\overline{C}$, ABC

最小项(minterm)

- 三变量函数的最小项

变量坐标	最小项	最小项符号	函数F
0 0 0	$\overline{A}\overline{B}\overline{C}$	m0	f0
0 0 1	$\overline{A}\overline{B}C$	m1	f1
0 1 0	$\overline{A}B\overline{C}$	m2	f2
0 1 1	$\overline{A}BC$	m3	f3
1 0 0	$A\overline{B}\overline{C}$	m4	f4
1 0 1	$A\overline{B}C$	m5	f5
1 1 0	$AB\overline{C}$	m6	f6
1 1 1	ABC	m7	f7

最小项(minterm)

- 性质

- 在输入变量的任何取值下，有且只有一个最小项的值为1
- 全体最小项之和为1
- 任意两个最小项的乘积为0
- 具有相邻性的两个最小项之和可以合并成一项并消去一个因子

- 编号

- 最小项用符号 m_i 表示，下标 i 即为最小项编号，用十进制数表示

真值表

A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

唯一性

$ABC\bar{C}$

标准与-或表达式

- 真值表上使函数值为1的变量取值组合对应的最小项相或，即可构成一个函数的标准与-或表达式

$$F = 0 \cdot \overline{A}\overline{B}\overline{C} + 0 \cdot \overline{A}\overline{B}C + 0 \cdot \overline{A}B\overline{C} + 1 \cdot \overline{A}BC + 0 \cdot A\overline{B}\overline{C} + 1 \cdot A\overline{B}C + 1 \cdot AB\overline{C} + 1 \cdot ABC$$

$$= \overline{A}BC + A\overline{B}C + AB\overline{C} + ABC$$

$$\begin{aligned} F &= 0 \cdot m_0 + 0 \cdot m_1 + 0 \cdot m_2 + 1 \cdot m_3 + 0 \cdot m_4 + 1 \cdot m_5 + 1 \cdot m_6 + 1 \cdot m_7 \\ &= m_3 + m_5 + m_6 + m_7 \\ &= \sum m(3, 5, 6, 7) \end{aligned}$$

标准与-或表达式

- 推广：一个n变量函数有 2^n 个最小项，每个最小项是n个变量构成的乘积项，每个变量在此与项中只出现一次。函数F的标准表达式就是这些最小项与对应的函数值乘积的总和，即

$$F = \sum_{i=0}^{2^n-1} f_i m_i$$

最大项和标准或-与表达式

$$F = \sum m_i$$

则 $\sum m_i$ 以外的那些最小项之和必为 \overline{F}

$$\overline{F} = \sum_{k \neq i} m_k$$

$$F = \overline{\sum_{k \neq i} m_k}$$

反演



$$F = \prod_{k \neq i} \overline{m_k} = \prod_{k \neq i} M_k$$

- 逻辑函数可以表示为若干最大项(maxterm)的乘积, 最小项之和与最大项之积都是唯一的, 所以也称为标准或-与式

最大项(maxterm)

- 三变量函数的最大项

变量坐标	最大项	最大项符号	函数F
0 0 0	$A+B+C$	M_0	f_0
0 0 1	$A+B+\bar{C}$	M_1	f_1
0 1 0	$A+\bar{B}+C$	M_2	f_2
0 1 1	$A+\bar{B}+\bar{C}$	M_3	f_3
1 0 0	$\bar{A}+B+C$	M_4	f_4
1 0 1	$\bar{A}+B+\bar{C}$	M_5	f_5
1 1 0	$\bar{A}+\bar{B}+C$	M_6	f_6
1 1 1	$\bar{A}+\bar{B}+\bar{C}$	M_7	f_7

$$M_i = \overline{m_i}$$

标准或-与表达式

$$F = \sum_i m_i \quad (i = 3, 5, 6, 7)$$

$$F = \prod_{k \neq i} \overline{m_k} = \prod_{k \neq i} M_k = M_0 M_1 M_2 M_4$$

$$= (A + B + C)(A + B + \overline{C})(A + \overline{B} + C)(\overline{A} + B + C)$$

- 根据函数的真值表求得反函数的标准与-或表达式，对其求反

标准或-与表达式

$$\begin{aligned} F &= (0 + M_0)(0 + M_1)(0 + M_2)(1 + M_3)(0 + M_4)(1 + M_5)(1 + M_6)(1 + M_7) \\ &= M_0 \cdot M_1 \cdot M_2 \cdot M_4 \\ &= (A + B + C)(A + B + \bar{C})(A + \bar{B} + C)(\bar{A} + B + C) \end{aligned}$$

- 找出函数真值表中所有等于0的行，根据每行的坐标求得对应的最大项（方法是坐标变量为1的取反变量，坐标变量为0的取原变量，然后相加），将这些最大项相与

卡诺图

- 一个逻辑函数的卡诺图就是将此函数的与-或表达式中的各最小项相应地填入一个特定的方格图内，此方格图称为卡诺图

A \ BC	BC			
	00	01	11	10
0	m_0	m_1	m_3	m_2
1	m_4	m_5	m_7	m_6

AB \ CD	CD			
	00	01	11	10
00	m_0	m_1	m_3	m_2
01	m_4	m_5	m_7	m_6
11	m_{12}	m_{13}	m_{15}	m_{14}
10	m_8	m_9	m_{11}	m_{10}

卡诺图

- 卡诺图的每一个方块代表一种输入组合，对应的输入组合注明在阵列图的上方和左方

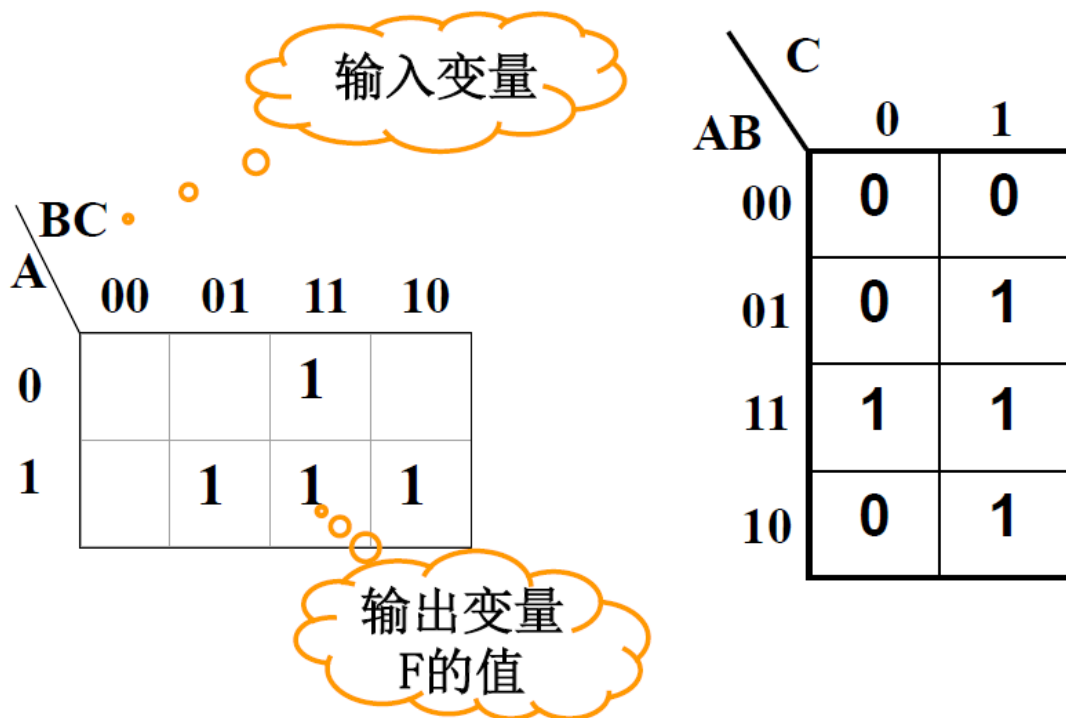
A \ BC				
	00	01	11	10
0			1	
1		1	1	1

逻辑相邻：相邻单元输入变量的取值只能有一位不同

AB \ CD				
	00	01	11	10
00	1	1		1
01	1	1		1
11	1		1	1
10	1			1

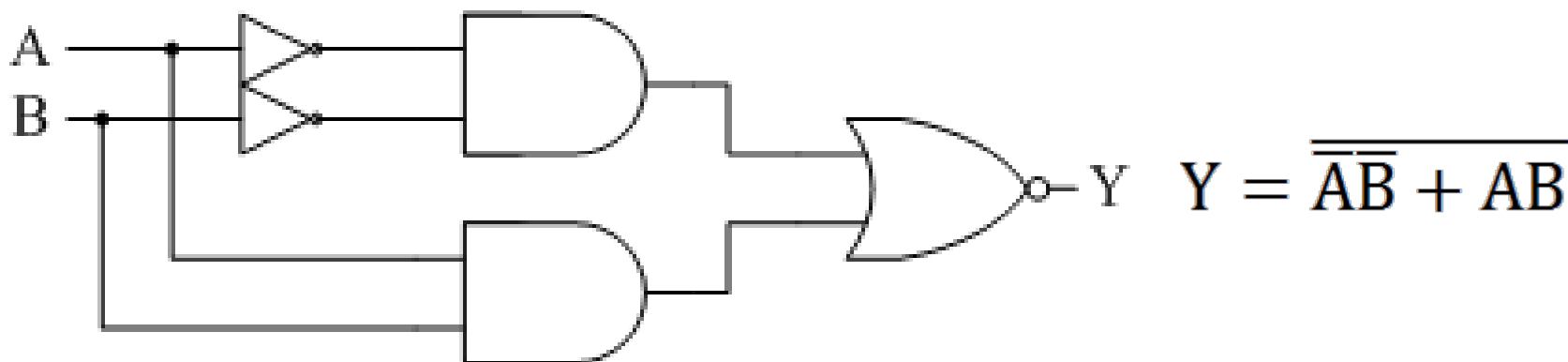
卡诺图

- 在卡诺图中变量组合按照循环邻接的原则进行排列，这个特点使得卡诺图可用于逻辑函数的简化



表示方法间的转化

- 逻辑图 \leftrightarrow 逻辑表达式

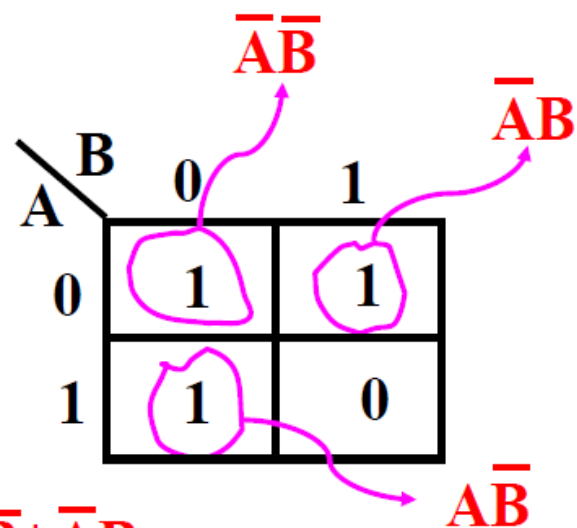


- 已知逻辑函数为 $F = \overline{A + \overline{B}C} + \overline{A}B\overline{C} + C$,
画出对应的逻辑图

表示方法间的转化

- 真值表、卡诺图 \leftrightarrow 逻辑代数式

A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0



$$Y = \overline{A}\overline{B} + A\overline{B} + \overline{A}B$$

- 此逻辑代数式并非是最简单的形式，实际上此真值表是与非门的真值表，因此有一个化简问题

逻辑函数的化简

- 化简意义
 - 降低成本
 - 减少功耗
 - 提高速度
- 化简目的
 - 消去多余的乘积项
 - 消去每个乘积项中多余的因子
- 化简标准
 - 与—或式中与项最少且每项的变量最少

公式法化简

- 并项法 $AB + \overline{A}\overline{B} = A$
- 试用并项法化简下列逻辑函数：

$$Y_1 = \overline{\overline{A}\overline{B}CD} + \overline{A}\overline{B}CD$$

$$Y_2 = \overline{A}\overline{B} + ACD + \overline{A}\overline{B} + \overline{A}CD$$

$$Y_3 = \overline{A}B\overline{C} + A\overline{C} + \overline{B}\overline{C}$$

$$Y_4 = B\overline{C}D + BC\overline{D} + \overline{B}\overline{C}\overline{D} + BCD$$

公式法化简

- 吸收法 $A + AB = A$
- 试用吸收法化简下列逻辑函数：

$$Y_1 = (\overline{\overline{AB}} + C)ABD + AD$$

$$Y_2 = AB + AB\overline{C} + ABD + AB(\overline{C} + \overline{D})$$

$$Y_3 = A + \overline{\overline{A} \cdot \overline{BC}}(\overline{A} + \overline{\overline{BC}} + D) + BC$$

公式法化简

- 消因子法 $A + \overline{A}B = A + B$
- 试用消因子法化简下列逻辑函数：

$$Y_1 = \overline{B} + ABC$$

$$Y_2 = \overline{A}\overline{B} + B + \overline{A}B$$

$$Y_3 = AC + \overline{A}D + \overline{C}D$$

公式法化简

- 消项法 $AB + \overline{A}C + BC = AB + \overline{A}C$
 $AB + \overline{A}C + BCD = AB + \overline{A}C$
- 试用消项法化简下列逻辑函数：

$$Y_1 = AC + \overline{A}\overline{B} + \overline{B} + C$$

$$Y_2 = \overline{A}\overline{B}\overline{C}\overline{D} + \overline{A}\overline{B}\overline{E} + \overline{A}\overline{C}\overline{D}\overline{E}$$

$$Y_3 = \overline{A}\overline{B}\overline{C} + \overline{A}BC + \overline{A}\overline{B}\overline{D} + \overline{A}B\overline{D} + \overline{A}BC\overline{D} + BC\overline{D}\overline{E}$$

公式法化简

• 配项法 $A + A = A$ 重叠律

$$A + \overline{A} = 1 \quad \text{互补律}$$

• 试用配项法化简下列逻辑函数：

$$Y_1 = \overline{A}\overline{B}\overline{C} + \overline{A}BC + ABC$$

卡诺图化简

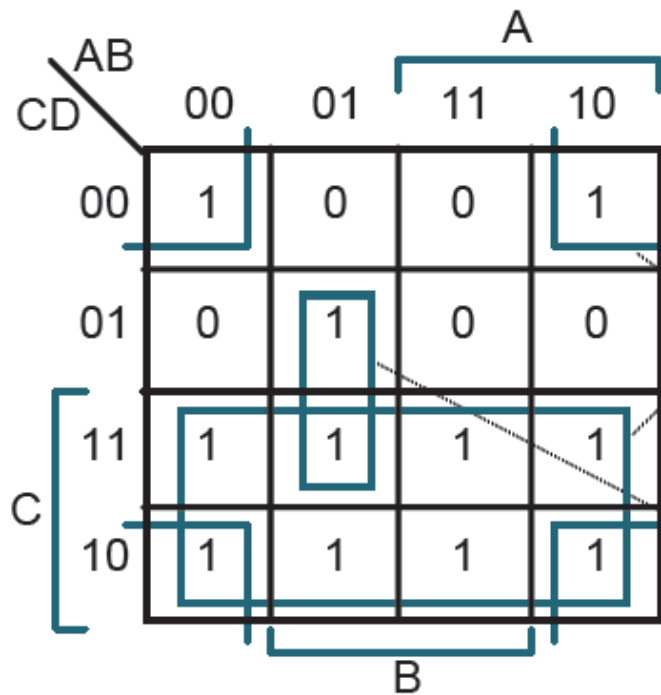
- 化简步骤
 - 将逻辑函数写成与-或表达式
 - 按与-或表达式填卡诺图，凡式中包含了的最小项，其对应方格填1，其余方格填0
 - 合并最小项，即将相邻的方格圈成一组，每一组含 2^n 个方格，对应每个包围圈写成一个新的乘积项
 - 将所有包围圈对应的乘积项相加
 - 也可由真值表直接填卡诺图，则前两个步骤可合并

卡诺图化简

- 画包围圈原则

- 包围圈内的方格数必定是 2^n 个， n 等于0、1、2、...
- 相邻方格包括上下底相邻，左右边相邻和四角相邻
- 同一方格可以被不同的包围圈重复包围，但新增包围圈中一定要有新的方格，否则该包围圈为多余
- 包围圈内的方格数要尽可能多，包围圈的数目要尽可能少
- 包围圈越大，所得乘积项中的变量越少，结果包围圈个数也就会少，使得消失的乘积项个数也越多

卡诺图化简



$$F(A,B,C,D) = \sum m(0,2,3,5,6,7,8,10,11,14,15)$$

$$F = C + \bar{A} B D + \bar{B} \bar{D}$$

化简逻辑函数 $L(A, B, C, D) = \sum m(0 \sim 3, 5 \sim 11, 13 \sim 15)$

Verilog: 模块(module)

module module_name (port_list); 注意点:

port declarations

data type declarations

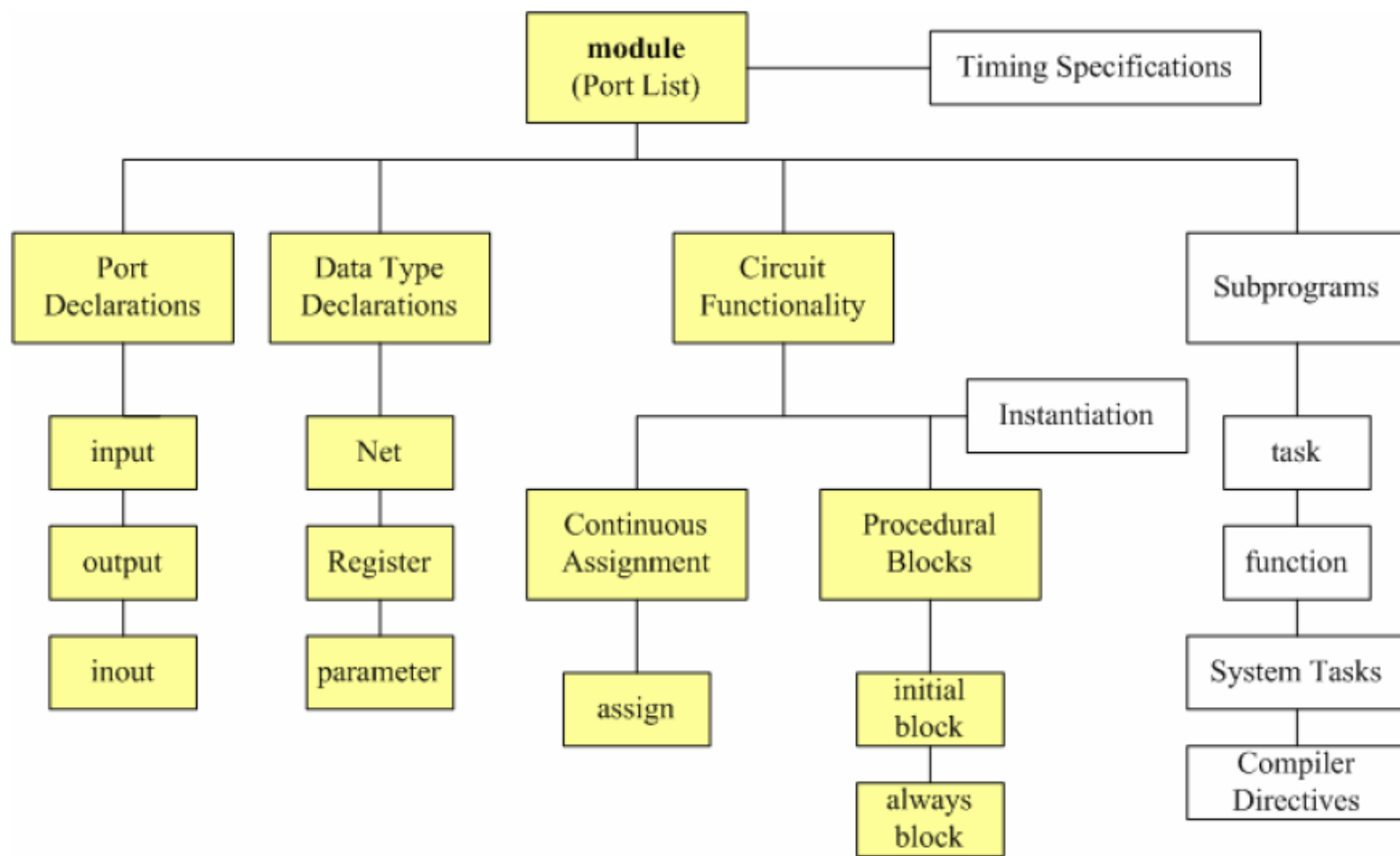
circuit functionality

timing specifications

endmodule

- 大小写敏感
- 所有关键字须小写
- 空格用于增加可读性
- 分号是语句终结符
- 单行注释: //
- 多行注释: /* */
- 时序规范用于仿真

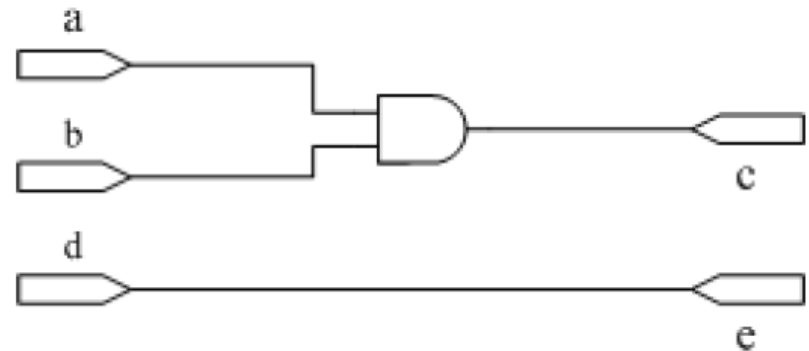
Verilog: 模块(module)



Verilog: 模块(module)

- 模块声明
module <模块名> (端口列表);
 <模块内容>
endmodule
- 端口列表
 – 列出所有端口名称
- 端口声明
 <端口类型> <端口名>;
- 端口类型
 – input->输入端口
 – output->输出端口
 – inout->双向端口

```
module hello_world(a,b,c,d,e);  
input a, b, d;  
output c, e;  
  
    assign c = a & b;  
    assign e = d;  
  
endmodule
```



Verilog: 模块(module)

- Verilog设计由互连的模块组成
- 模块可以是元件或者低层次模块的组合
- 一个简单的组合逻辑模块可能如下所示:

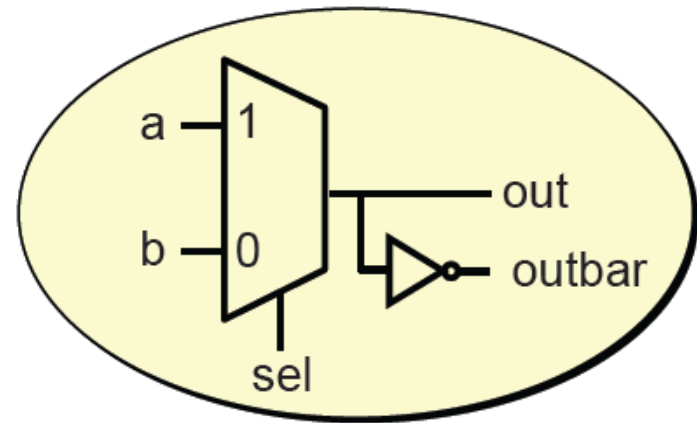
```
module mux_2_to_1(a, b, out,  
                  outbar, sel);
```

```
// This is 2:1 multiplexor
```

```
input a, b, sel;  
output out, outbar;
```

```
assign out = sel ? a : b;  
assign outbar = ~out;
```

```
endmodule
```

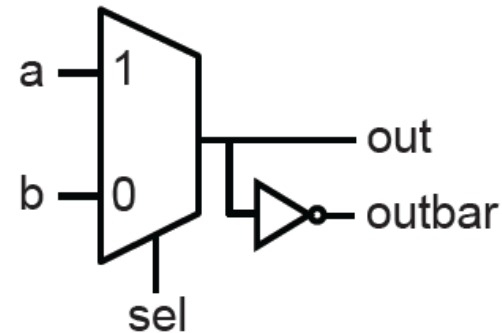


$$\text{Out} = \text{sel} \bullet a + \overline{\text{sel}} \bullet b$$

2-to-1 multiplexer with inverted output

连续赋值(continuous assignment)

```
module mux_2_to_1(a, b, out,  
                  outbar, sel);  
  
    input a, b, sel;  
    output out, outbar;  
  
    assign out = sel ? a : b;  
    assign outbar = ~out;  
  
endmodule
```



- 等号左侧必须是线网(Net), 右侧可以是寄存器(register)或线网
- 运算符
 - 条件运算: conditional_expression ? value-if-true : value-if-false;
 - 布尔逻辑: ~, &, |
 - 算术: +, -, *
- 嵌套条件运算(4:1 mux)
 - assign = s1 ? (s0 ? i3 : i2) : (s0 ? i1 : i0);

数据类型(Data Types)

- 线网型
 - 用关键字wire等声明
 - 相当于硬件电路里的物理连接

```
wire [7:0] in, out;  
assign out = in;
```
- 寄存器型
 - 用reg或integer声明
 - 数据存储单元的抽象
 - 不表示必将综合成硬件寄存器
 - 用在过程语句(always, initial)中

```
reg [2:0] a, b, c;  
reg d;
```


数据类型(Data Types)

- 二维寄存器数组（存储器）

reg [15:0] ROM[7:0];

reg [n-1:0] rega;

reg memb[n-1:0];

rega = 0; //合法赋值

memb = 0; //非法赋值

memb[0] = 1; //合法赋值

- parameter：用来定义常量

parameter size = 8;

reg [size-1:0] a, b;

- 其他类型：real, realtime, wand, wor, tri, triand, trior, trireg等

数字(Numbers)

- sized, unsized: <位宽>'<进制><数字>
 - sized: 3'b010 //3位二进制数字，值为010
 - unsized:
 - <进制>默认为十进制
 - <位宽>默认为32-bit
- 进制
 - 十进制('d或'D)
 - 十六进制('h或'H)
 - 二进制('b或'B)
 - 八进制('o或'O)

数字(Numbers)

- 负数——在<位宽>前加负号
 - 例：-8'd3
- 特殊字符
 - ‘_’（下划线）：增加可读性
 - ‘x’或‘X’（未知数）
 - ‘z’或‘Z’（高阻）
- 若定义的位宽比实际位数长
 - 如果高位是0或1，高位补0
 - 如果高位是x、z，高位分别补x、z

运算符(Operators)

- 算术运算符(Arithmetic)
 - +, -, *, /, %, **
- 逻辑运算符(Logical)
 - &&, ||, !
- 位运算符(Bitwise)
 - ~, &, |, ^, ^~, ~^
- 关系运算(Relational)
 - <, <=, >, >=
 - <=也用于表示一种赋值操作

运算符(Operators)

- 等价运算(Equality)
 - ==, !=, ===, !==
- 缩位运算(Reduction)
 - &, ~&, |, ~|, ^, ^~, ~^
- 移位(Shift)
 - >>, <<, >>>, <<<
- 条件(Conditional)
 - ? :
- 位拼接(Concatenation)
 - { }

运算符优先级

- 缺省操作符优先级

+, -, !, ~ (单目操作符)

*, /, %

+, - (双目操作符)

<<, >>

<, <=, >, >=

==, !=, ===, !==

&, ~&

^, ^~

|

&&

||

?:

高优先级

低优先级

- () 可用于调整优先级

课堂练习

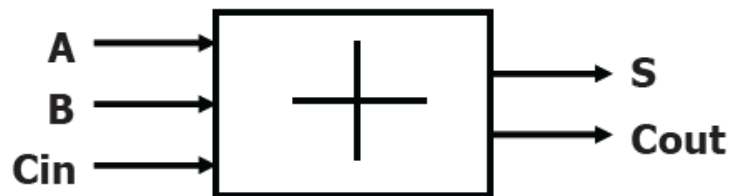
- 对之前举重比赛裁判评分逻辑进行Verilog HDL编程实现
 - 举重比赛有三名裁判，当运动员将杠铃举起后，须有两名或两名以上裁判认可，方可判定试举成功
 - 用A、B、C分别表示三名裁判的意见输入，同意为1，否定为0
 - 用F表示裁判结果输出，试举成功时为1，试举失败时为0

课堂练习

- 对1-bit加法器进行Verilog HDL编程实现

— 输入：A、B、Cin

— 输出：Sum、Cout



A	B	Cin	S	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

$$S = \overline{A} \overline{B} \text{Cin} + \overline{A} B \overline{\text{Cin}} + A \overline{B} \overline{\text{Cin}} + A B \text{Cin}$$

$$\text{Cout} = \overline{A} B \text{Cin} + A \overline{B} \text{Cin} + A B \overline{\text{Cin}} + A B \text{Cin}$$

1-bit加法器逻辑简化

$$\begin{aligned}\text{Cout} &= \overline{A} B \text{Cin} + A \overline{B} \text{Cin} + A B \overline{\text{Cin}} + A B \text{Cin} \\&= \overline{A} B \text{Cin} + \textcolor{red}{A B \text{Cin}} + A \overline{B} \text{Cin} + \textcolor{red}{A B \text{Cin}} + A B \overline{\text{Cin}} + \textcolor{red}{A B \text{Cin}} \\&= (\overline{A} + A) B \text{Cin} + A (\overline{B} + B) \text{Cin} + A B (\overline{\text{Cin}} + \text{Cin}) \\&= B \text{Cin} + A \text{Cin} + A B \\&= (B + A) \text{Cin} + A B\end{aligned}$$

$$\begin{aligned}S &= \overline{A} \overline{B} \text{Cin} + \overline{A} B \overline{\text{Cin}} + A \overline{B} \overline{\text{Cin}} + A B \text{Cin} \\&= (\overline{A} \overline{B} + A B) \text{Cin} + (A \overline{B} + \overline{A} B) \overline{\text{Cin}} \\&= \overline{(A \oplus B)} \text{Cin} + (A \oplus B) \overline{\text{Cin}} \\&= A \oplus B \oplus \text{Cin}\end{aligned}$$