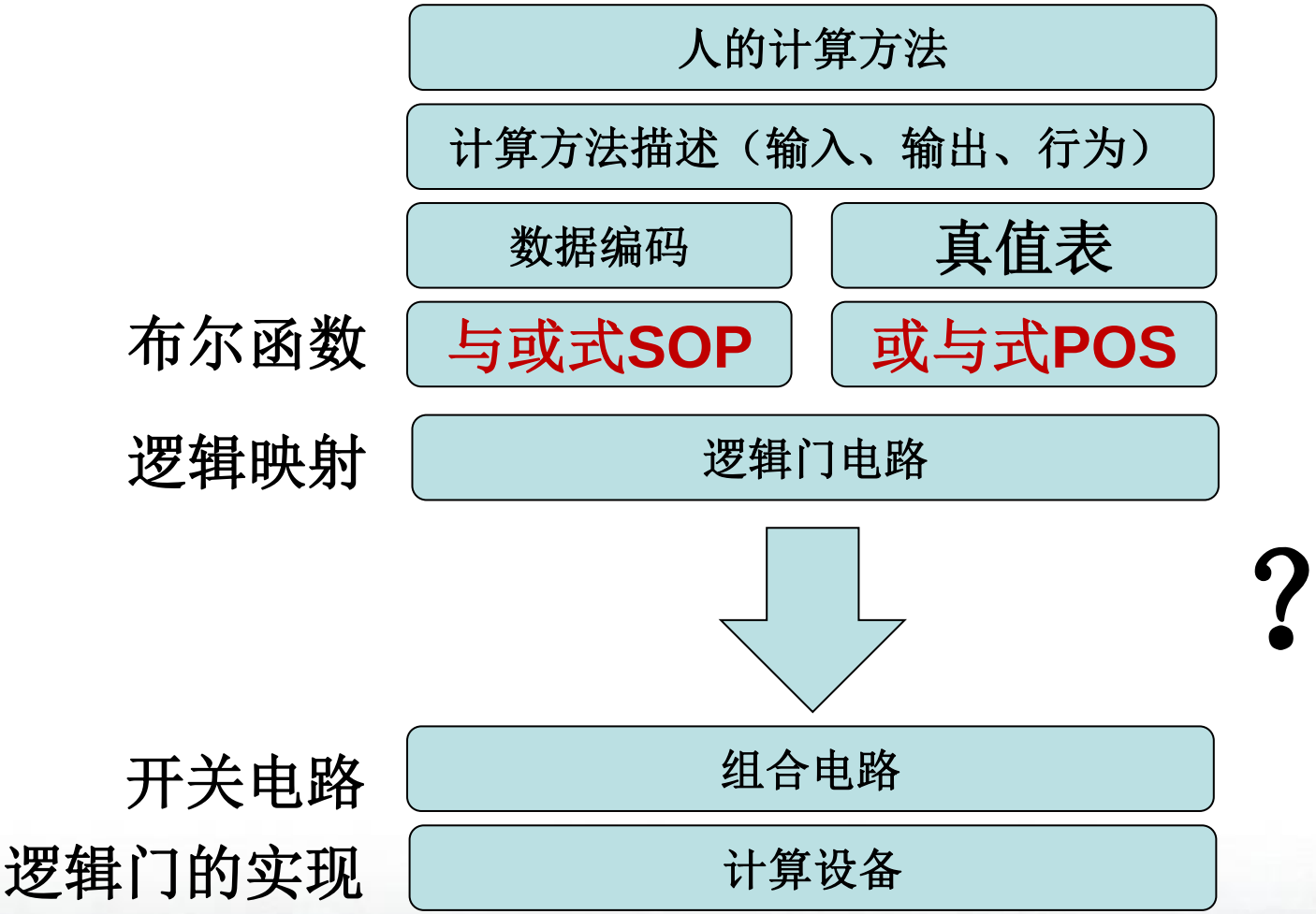
第四讲 组合逻辑
Combinational Logic

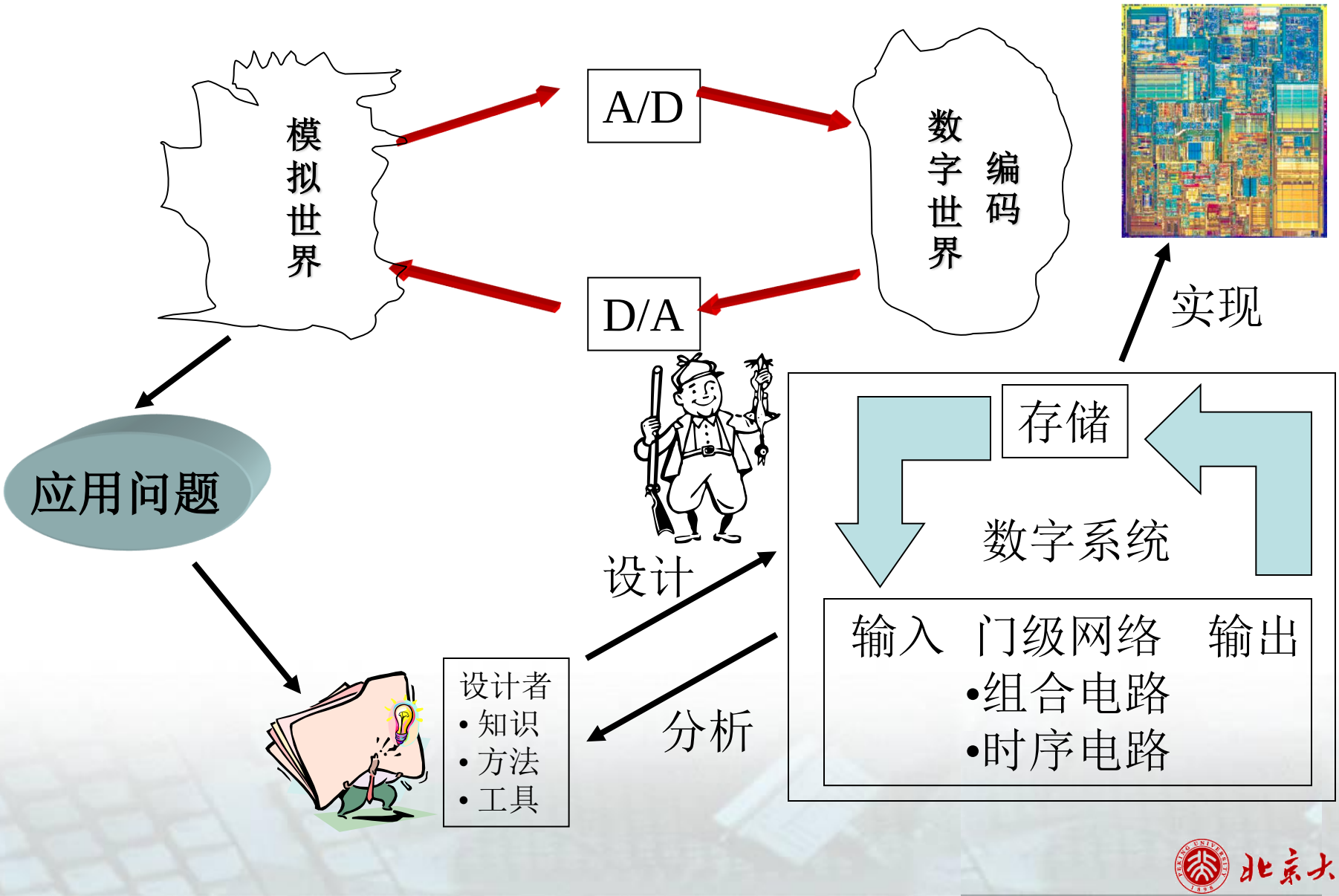
佟冬
tongdong@pku.edu.cn

微处理器研究开发中心（MPRC）
北京大学计算机学院

如何做一个能计算的设备?



数字系统的分析与设计



数字电路的分析与设计

- 电路设计：从一个电路功能的描述到一组开关函数进而到门级、PLD或其它逻辑单元实现的转化过程。
- 电路分析：从一个数字电路的实现出发，得到电路的某种形式的功能描述
  - 开关表达式(Switch expression)
  - 真值表(Truth table)
  - 时序图(Timing diagram)
  - 其它行为描述(behavioral description)
- 设计与分析是相反的过程

数字电路分析的目的

- 确定逻辑电路的行为功能
- 验证电路的行为和规范说明是否一致
- 协助将电路转变为另一种形式
- 减少电路中门的个数
- 采用不同的逻辑单元实现电路

- 获取前人电路设计的知识、方法和经验

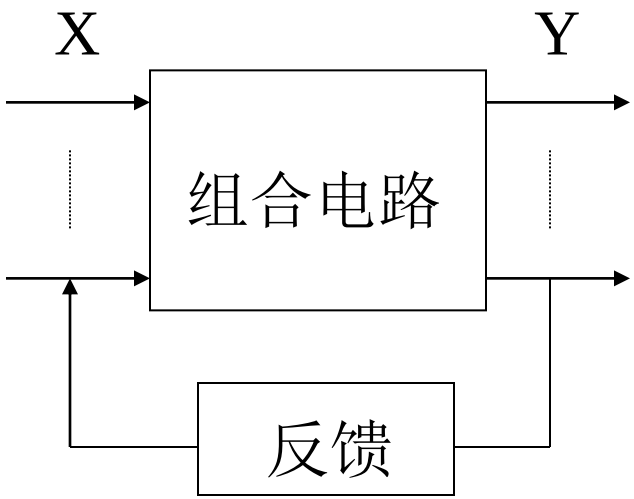
复习：组合电路和时序电路

- 组合电路
  - 输出值是输入值的逻辑函数
  - 输入值变化后一段时间后出现新的输出值
  - 电路没有记忆功能
  - 电路中没有循环反馈 (feedback loop)和时钟
- 时序电路
  - 输出值是输入值和电路状态的函数
  - 输入变化后新的输出出现在下一个时钟事件或者其他事件发生
  - 电路有循环反馈，存在存储元件

复习：组合电路和时序电路

- 时序电路是组合电路加上反馈电路

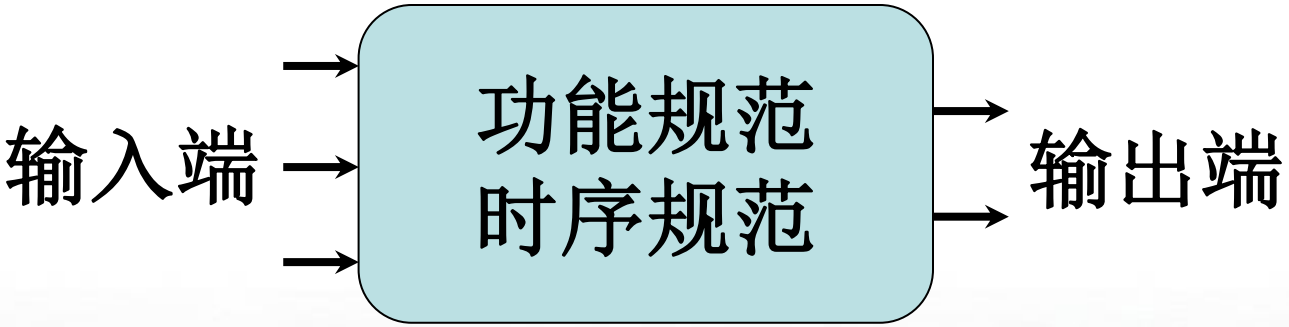
组合电路



时序电路

组合电路

- 组合电路规范
  - 输入端
  - 输出端
  - 功能规范：输入到输出逻辑关系（如真值表）
  - 时序规范：从输入改变到输出相应的延迟（如波形图）



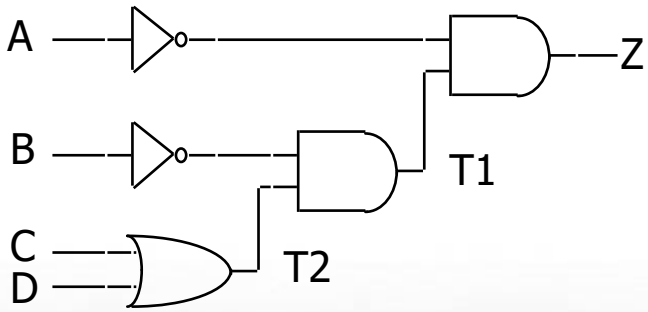
1. 数字电路分析方法

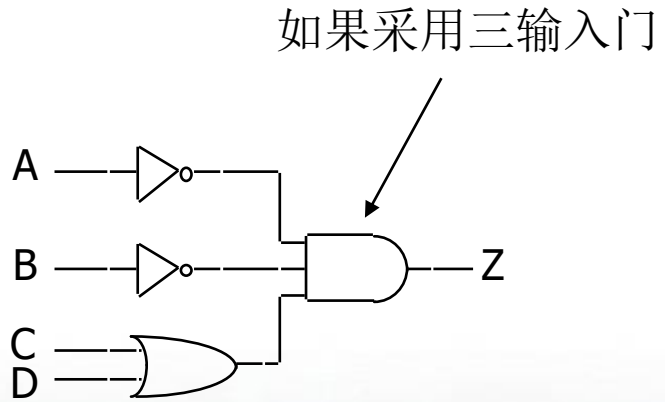
- 代数分析法：用开关代数来获取指定的功能形式
- 真值表分析法：逐次分析每一个门的真值表，直到输出
- 时序图分析法
  - 时序图(Timing Diagram)是一个开关网络的输入和输出信号关系在时间维度上的图形表示。
  - 时序图可以显示中间信号和传播延迟。
  - 时序图的获得
    - 示波器(oscilloscope)
    - 逻辑分析仪(logic analyzer)
    - 逻辑模拟程序(simulation program, simulator)
    - 许多电路的设计和说明文档中给出的时序图

布尔表达式到电路图的转换

- 可以有多种方法将表达式映射成电路

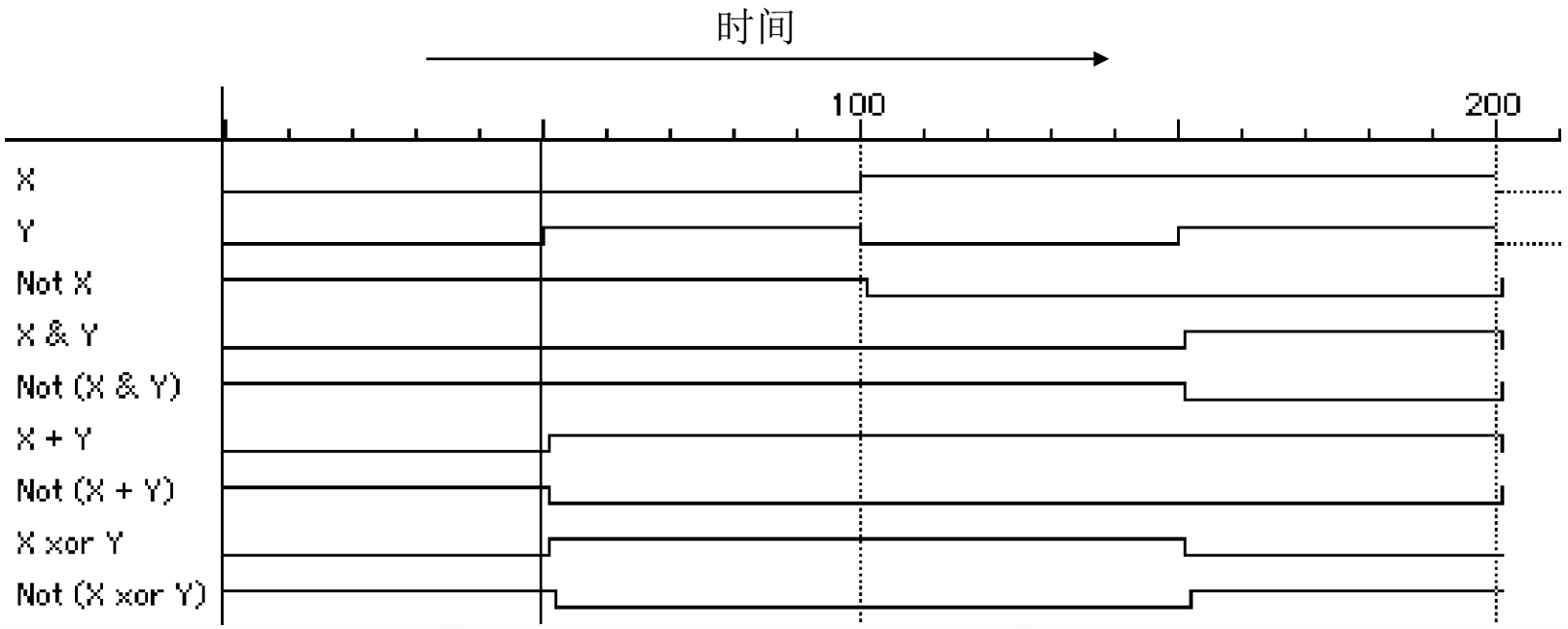
– e.g., $Z = A' \cdot B' \cdot (C + D) = (A' \cdot \overline{(B' \cdot (C + D))}^{T2})^{T1}$





逻辑函数的的波形图表示

- 真值表的另一种表示方法
  - 注意门的传播延迟



什么是好的电路?

- 减少输入的数目
  - 字母: 输入变量
    - 大概估计一个字母2个晶体管
    - 为什么不算非门?
  - 较少的字母意味着较少的晶体管
    - 更小的电路
  - 较少的输入意味着较快的门
    - 门越小越快
  - 扇入fan-ins （门的输入数）由所采用的工艺限制
- 减少门的个数
  - 较少的门数意味着更小的电路
    - 直接影响制造成本

什么是最好的实现?

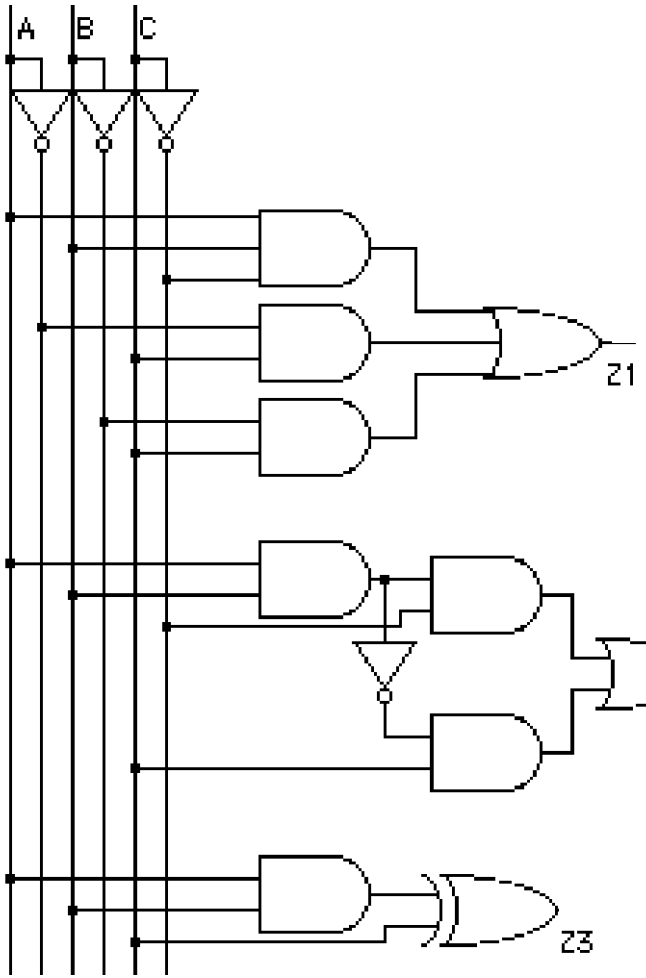
- 减少门的级数
  - 较少的门级数意味着减少信号传播延迟
  - 最少的延迟一般需要更多的门

- 需要在增加电路延迟和规模之间权衡
  - 自动工具可以产生多种实现
  - 逻辑化简：减少门的个数和复杂度
  - 逻辑优化：进一步在延迟方面权衡

实现同一函数的不同电路

Z=A'B'C+A'BC+AB'C+ABC'

A	B	C	Z
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0



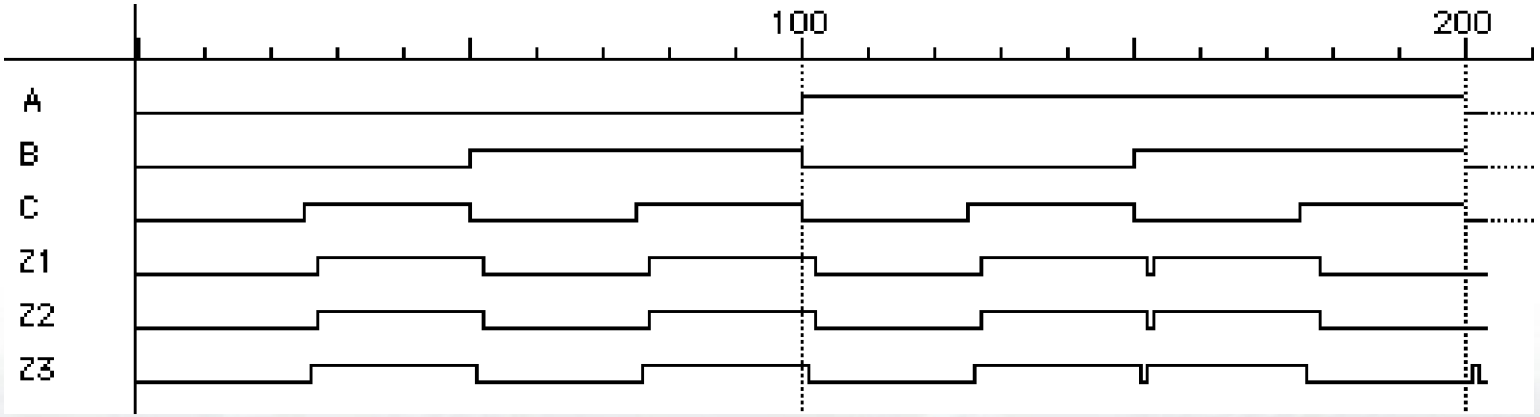
两级实现
（不算非门NOT）

多极实现
（门的输入较少）

异或门XOR （门最少，但是可能比较难实现）

所有实现的一致性?

- 在相同的输入激励下，3个不同的实现产生几乎相同的波形
  - 延迟不一样
  - 产生假信号glitches（“毛刺”，冒险hazards）
  - 不同的结果因为门的级数和结构不同
- 三种实现在功能上是一致的

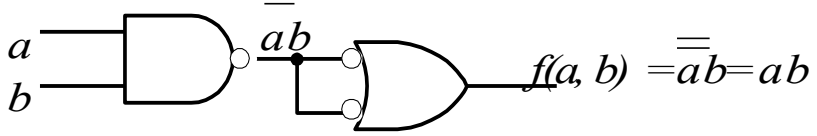


组合电路的实现

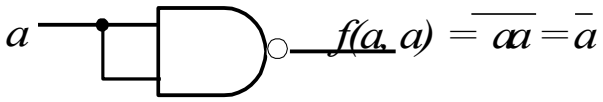
- 布尔表达式的规范表示
  - 与或式，积之和SOP
  - 或与式，和之积POS
  - 非确定函数
- 两级逻辑电路
  - 易于理解和逻辑实现
  - 与或两级逻辑电路、与非门两级逻辑电路
  - 或与两级逻辑电路、或非门两级逻辑电路

复习：与非门和或非门的应用

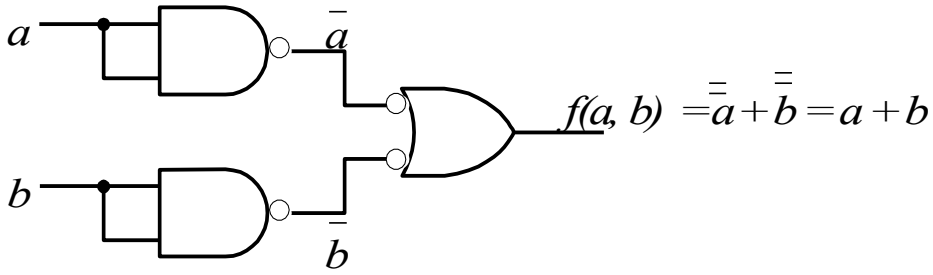
- AND, OR, NOT门可用NAND(或NOR)门构造
- 门级网络可以只用NAND(或NOR)门来实现。



AND gate

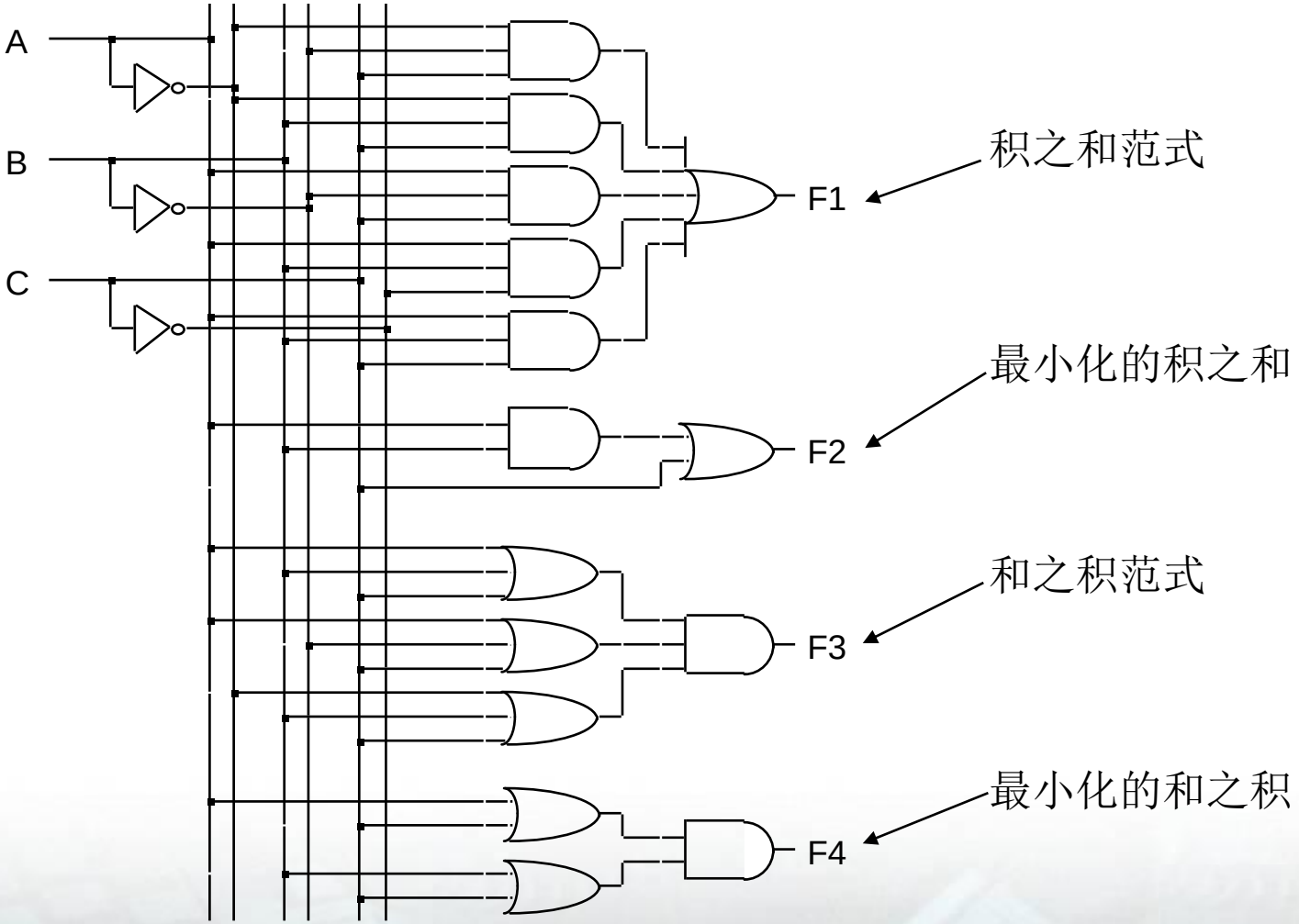


NOT gate



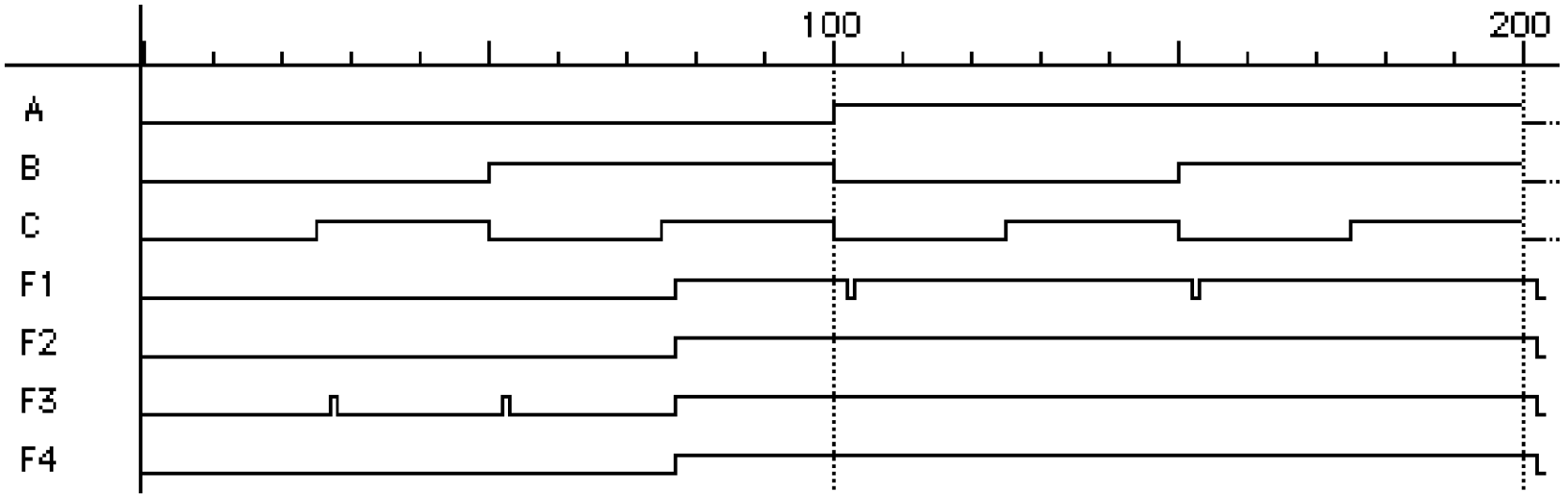
OR gate

F = AB + C的四种不同的两级逻辑电路实现



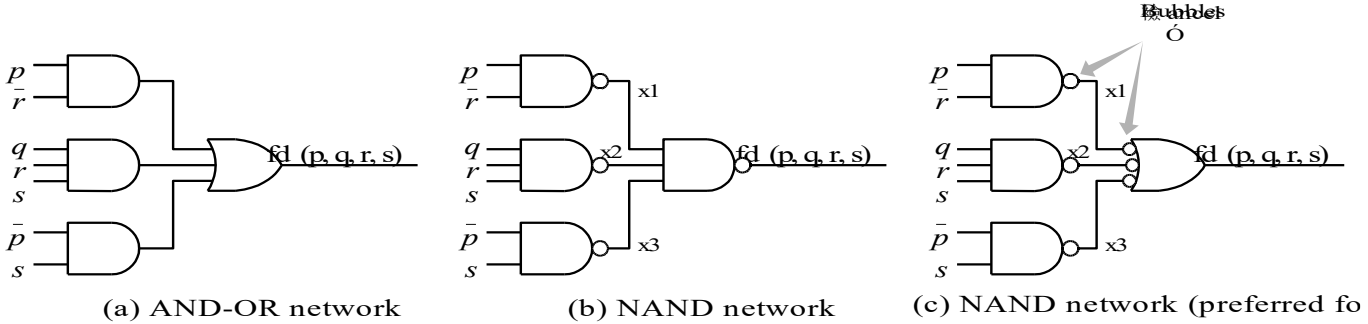
四种不同实现的波形图

- 波形图基本上是一致的
  - 存在以下冒险hazards（毛刺glitches）
  - 电路延时并不一致



两级组合逻辑电路

- AND-OR和NAND逻辑电路
  - 开关表达式采用SOP的形式
  - 例：$f_\delta(p,q,r,s) = p\bar{r} + qrs + \bar{p}s$



(a) AND-OR network　(b) NAND network　(c) NAND network (preferred fo

$$f_\delta(p,q,r,s) = \overline{\overline{p\bar{r} + qrs + \bar{p}s}}$$

$$= \overline{\overline{p\bar{r}} \cdot \overline{qrs} \cdot \overline{\bar{p}s}}$$

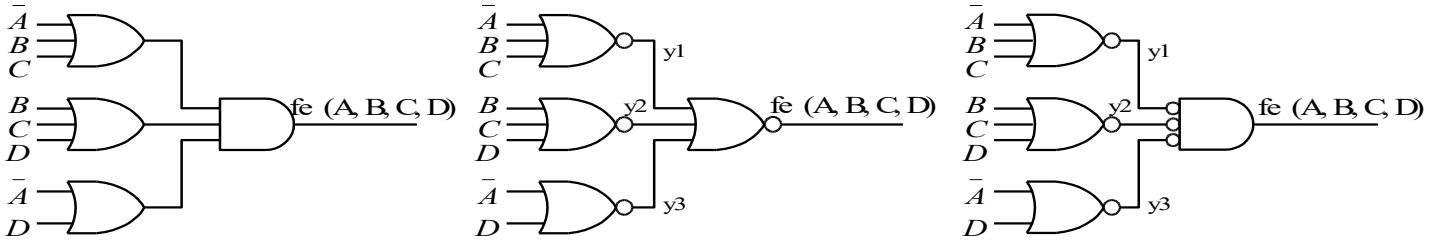
$$= \overline{x_1 \cdot x_2 \cdot x_3}$$

其中：$x_1 = \overline{p\bar{r}}, x_2 = \overline{qrs}, x_3 = \overline{\bar{p}s}$

两级组合逻辑电路

- OR-AND和NOR两级逻辑电路
  - 开关表达式采用POS的形式
  - 例：

$$f_\varepsilon(A,B,C,D)=(\overline{A}+B+C)(B+C+D)(\overline{A}+D)$$



(a) OR-AND network　(b) NOR network　(c) NOR network (preferred fc

$$f_\varepsilon(A,B,C,D)=\overline{\overline{(\overline{A}+B+C)(B+C+D)(\overline{A}+D)}}$$

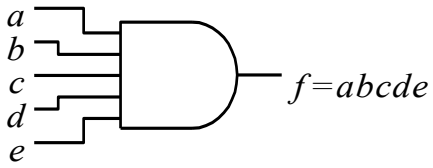
$$=\overline{\overline{\overline{A}+B+C}+\overline{B+C+D}+\overline{\overline{A}+D}}$$

$$=\overline{y_1+y_2+y_3}$$

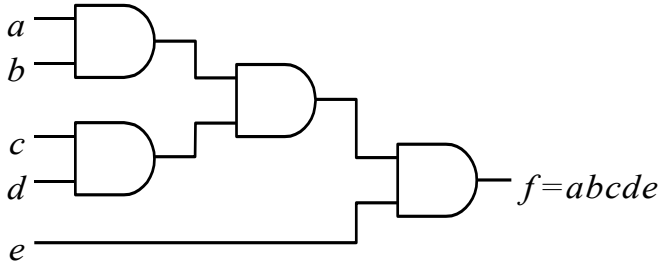
其中：$y_1=\overline{\overline{A}+B+C}, y_2=\overline{B+C+D}, y_3=\overline{\overline{A}+D}$

多级组合逻辑电路

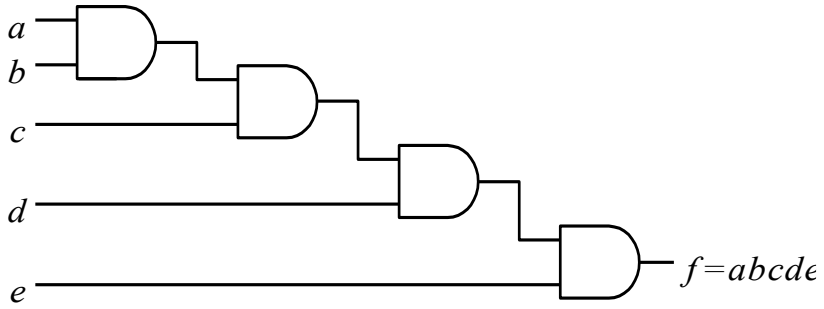
- 采用多于两级的电路经常是因为扇入(fan-in)的限制。
- 电路速度方面的考虑



(a) A single five-input AND gate



(b) Three-level network of two-input gates



(c) Four-level network of two-input gates.

组合逻辑电路的综合

- 例：用NAND逻辑实现 $f_\phi(X,Y,Z)=\sum m(0,3,4,5,7)$

1. $f_\phi(X,Y,Z)=\sum m(0,3,4,5,7)$

2. $f_\phi(X,Y,Z)=m0+m3+m4+m5+m7$

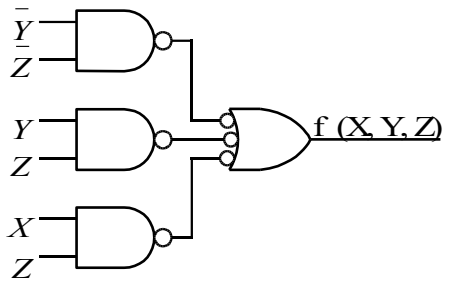
$$=\bar{X}\bar{Y}Z+\bar{X}YZ+X\bar{Y}\bar{Z}+X\bar{Y}Z+XYZ$$

3. $f_\phi(X,Y,Z)=\bar{Y}\bar{Z}+YZ+XZ$

4a. $f_\phi(X,Y,Z)=\overline{\overline{\bar{Y}\bar{Z}}+\overline{YZ}+\overline{XZ}}$

4b. $f_\phi(X,Y,Z)=\overline{\overline{\bar{Y}\bar{Z}}+YZ+XZ}$

$$=\overline{\overline{\bar{Y}\bar{Z}}\cdot\overline{YZ}\cdot\overline{XZ}}$$



(a) NAND implementatic

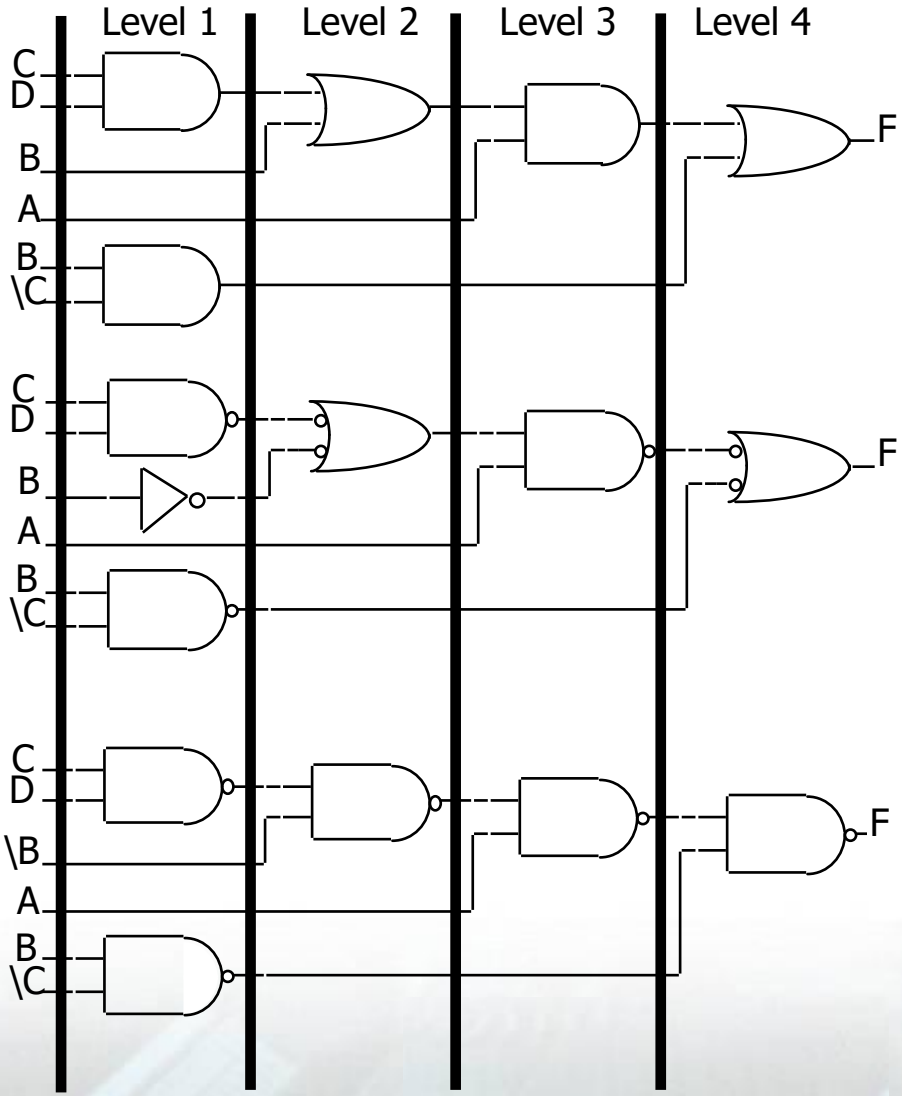
多级逻辑NAND变换

- F = A (B + C D) + B C’

原始
AND-OR
网络

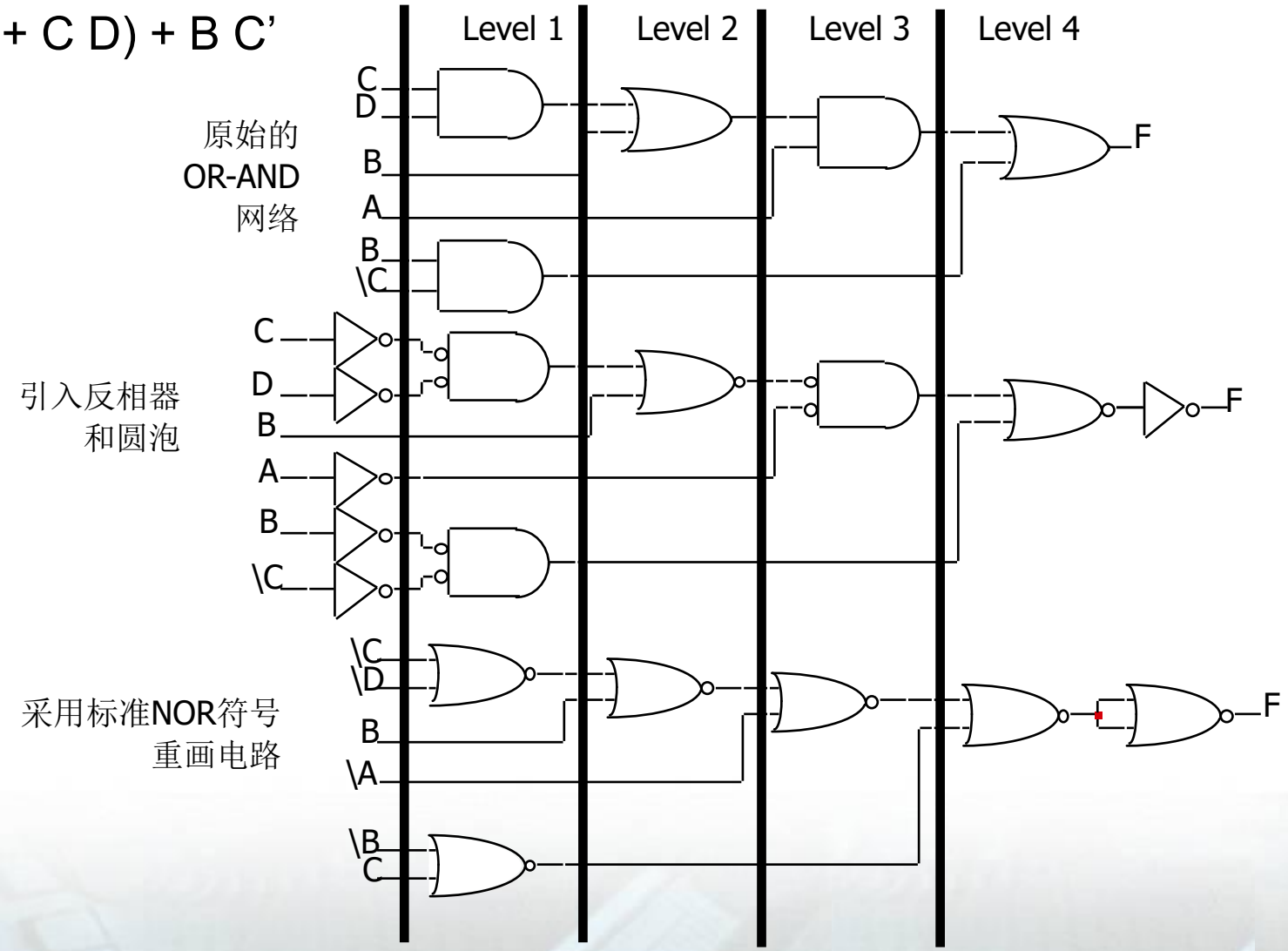
加入反相器
和圈泡

采用标准NAND符号
重画电路



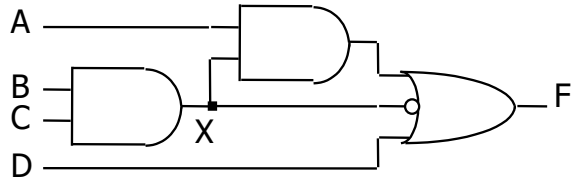
多级逻辑的NOR变换

- F = A (B + C D) + B C'

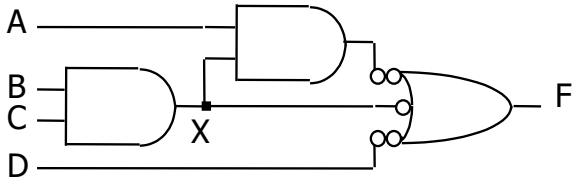


多级网络电路变换实例

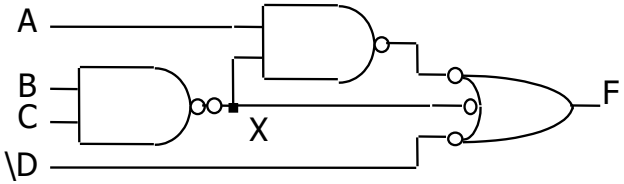
例子



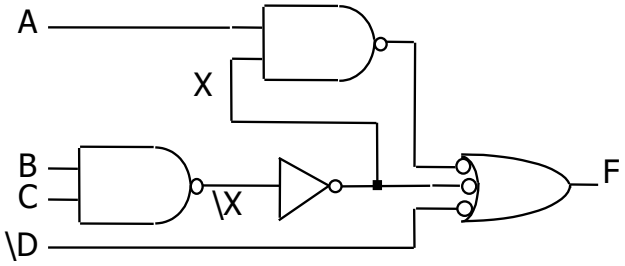
原电路



加入双圆泡
将或门的输入信号取反



加入双圆泡
将与门的输出信号取反

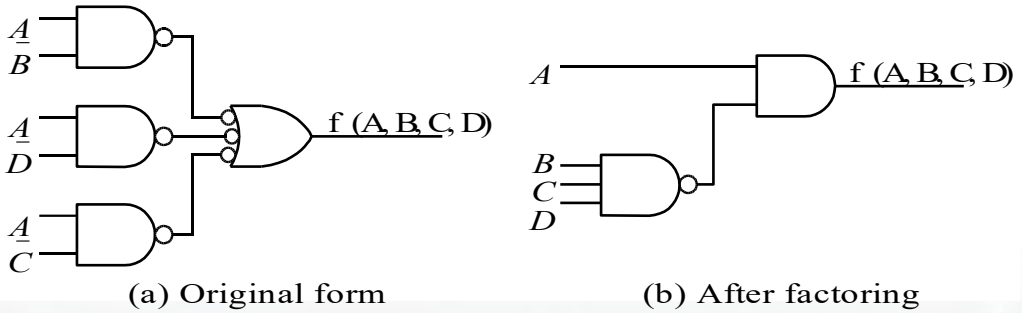


插入非门替换多余的圆泡

组合逻辑电路

- 因式分解(factoring)
  - 求高级开关形式的技术
  - 高级形式：
    - 可能需要更少的硬件
    - 可能受扇入的限制
    - 可能为设计带来困难
    - 可能更慢
- 例：

$$f(A,B,C,D) = A\overline{B} + A\overline{D} + A\overline{C} = A(\overline{B} + \overline{D} + \overline{C}) = A(\overline{BCD})$$



(a) Original form　　(b) After factoring

组合电路应用实例(1)

- **例1**：一个带有4个控制开关的防盗报警系统，每个开关产生逻辑1，当：
  - 开关*A*：保密开关关闭
  - 开关*B*：保险箱在储藏室的正常位置
  - 开关*C*：时钟在10:00和14:00小时之间
  - 开关*D*：储藏室门处于关闭状态

  命题：根据下列条件写出产生逻辑1的控制逻辑表达式

  保险箱被移动并且保密开关关闭 或者
  在银行下班后储藏室门被打开 或者
  控制开关打开并且储藏室门被打开

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

应用实例(2)

- 例2：写出计算两个2-bit二进制数$(A_1A_0)_2$和$(B_1B_0)_2$的和，产生结果$(S_1S_0)_2$和进为C_1

$$\begin{array}{r} A_1A_0 \\ +\,B_1B_0 \\ \hline C_1S_1S_0 \end{array}$$

应用实例(2)

- 真值表

A1	A0	B1	B0	C1	S1	S0
0	0	0	0	0	0	0
0	0	0	1	0	0	1
0	0	1	0	0	1	0
0	0	1	1	0	1	1
0	1	0	0	0	0	1
0	1	0	1	0	1	0
0	1	1	0	0	1	1
0	1	1	1	1	0	0
1	0	0	0	0	1	0
1	0	0	1	0	1	1
1	0	1	0	1	0	0
1	0	1	1	1	0	1
1	1	0	0	0	1	1
1	1	0	1	1	0	0
1	1	1	0	1	0	1
1	1	1	1	1	0	0

- 逻辑表达式

$$S_0 = \bar{A}_1\bar{A}_0\bar{B}_1B_0 + \bar{A}_1\bar{A}_0B_1B_0 + \bar{A}_1A_0\bar{B}_1\bar{B}_0 + \bar{A}_1A_0B_1\bar{B}_0 + A_1\bar{A}_0\bar{B}_1B_0 + A_1\bar{A}_0B_1B_0 + A_1A_0\bar{B}_1\bar{B}_0 + A_1A_0B_1\bar{B}_0$$

$$S_1 = \bar{A}_1\bar{A}_0B_1\bar{B}_0 + \bar{A}_1\bar{A}_0B_1B_0 + \bar{A}_1A_0\bar{B}_1B_0 + \bar{A}_1A_0B_1\bar{B}_0 + A_1\bar{A}_0\bar{B}_1\bar{B}_0 + A_1\bar{A}_0\bar{B}_1B_0 + A_1A_0\bar{B}_1\bar{B}_0 + A_1A_0B_1B_0$$

$$C_1 = \bar{A}_1A_0B_1B_0 + A_1\bar{A}_0B_1\bar{B}_0 + A_1\bar{A}_0B_1B_0 + A_1A_0\bar{B}_1B_0 + A_1A_0B_1\bar{B}_0 + A_1A_0B_1B_0$$

应用实例(2)

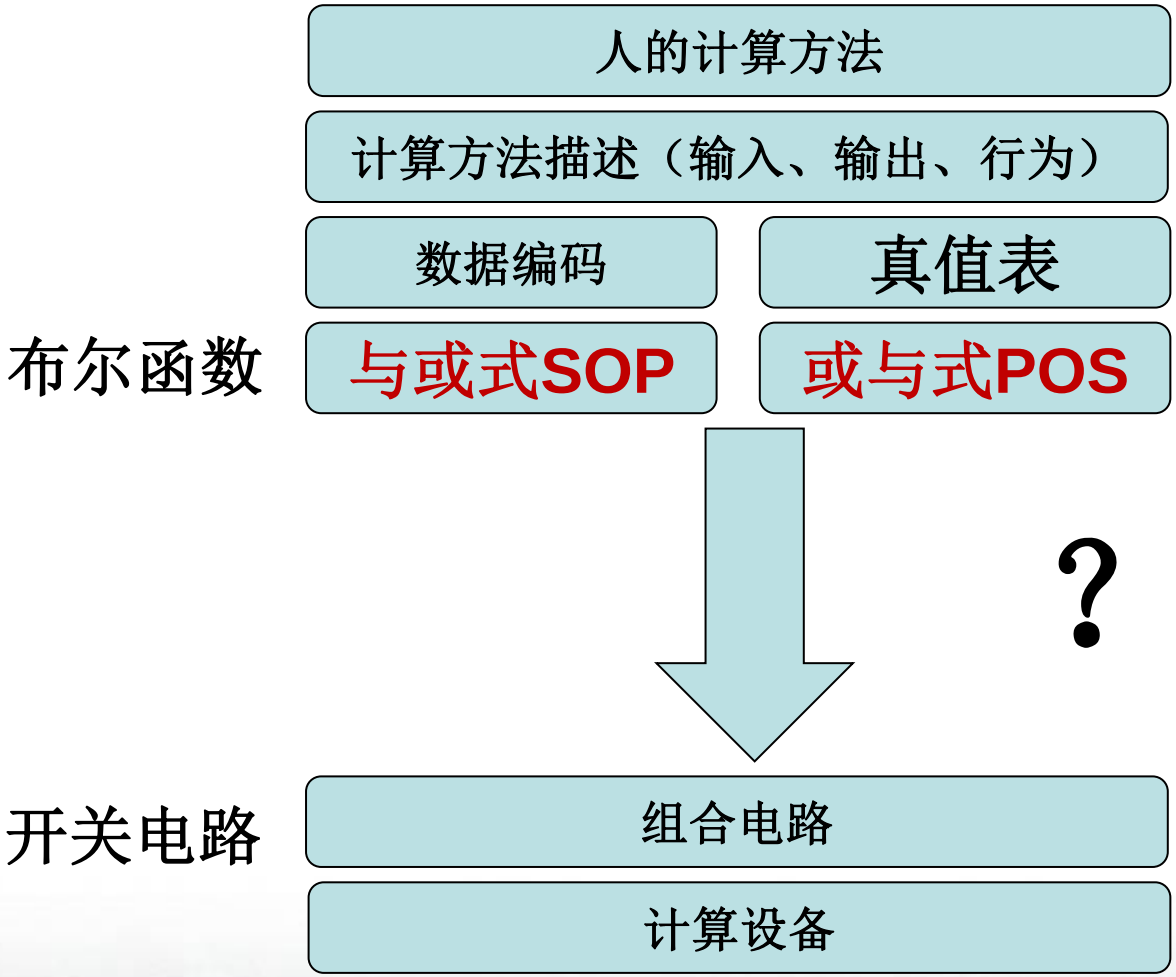
- 化简后的表达式

$$S_0 = A_0\overline{B}_0 + \overline{A}_0 B_0 = A_0 \oplus B_0$$

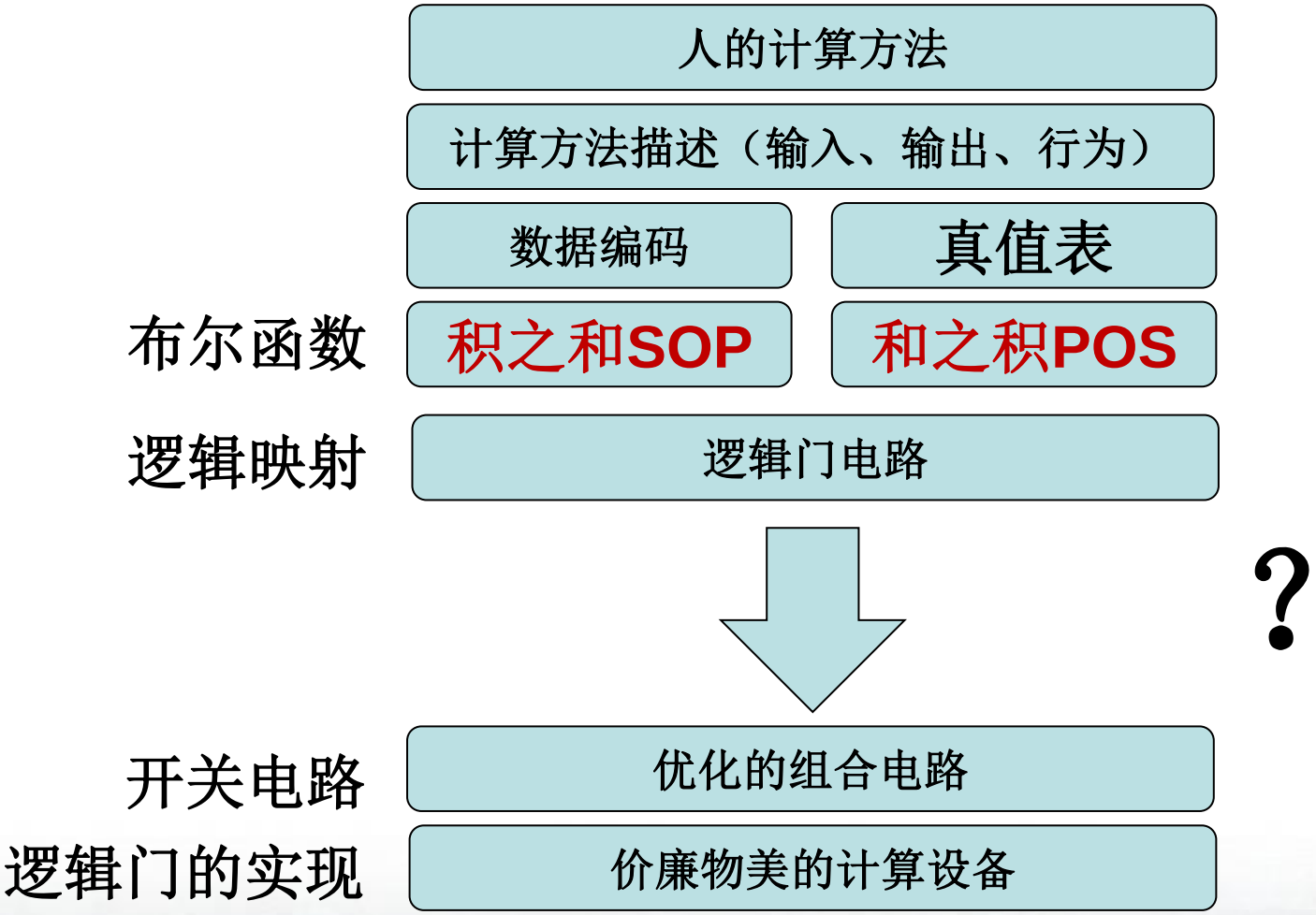
$$S_1 = \overline{A}_1\overline{A}_0 B_1 + \overline{A}_1 B_1\overline{B}_0 + \overline{A}_1 A_0\overline{B}_1 B_0 + A_1 A_0 B_1 B_0 + A_1\overline{B}_1\overline{B}_0 + A_1\overline{A}_0\overline{B}_1$$

$$C_1 = A_0 B_1 B_0 + A_1 A_0 B_0 + A_1 B_1$$

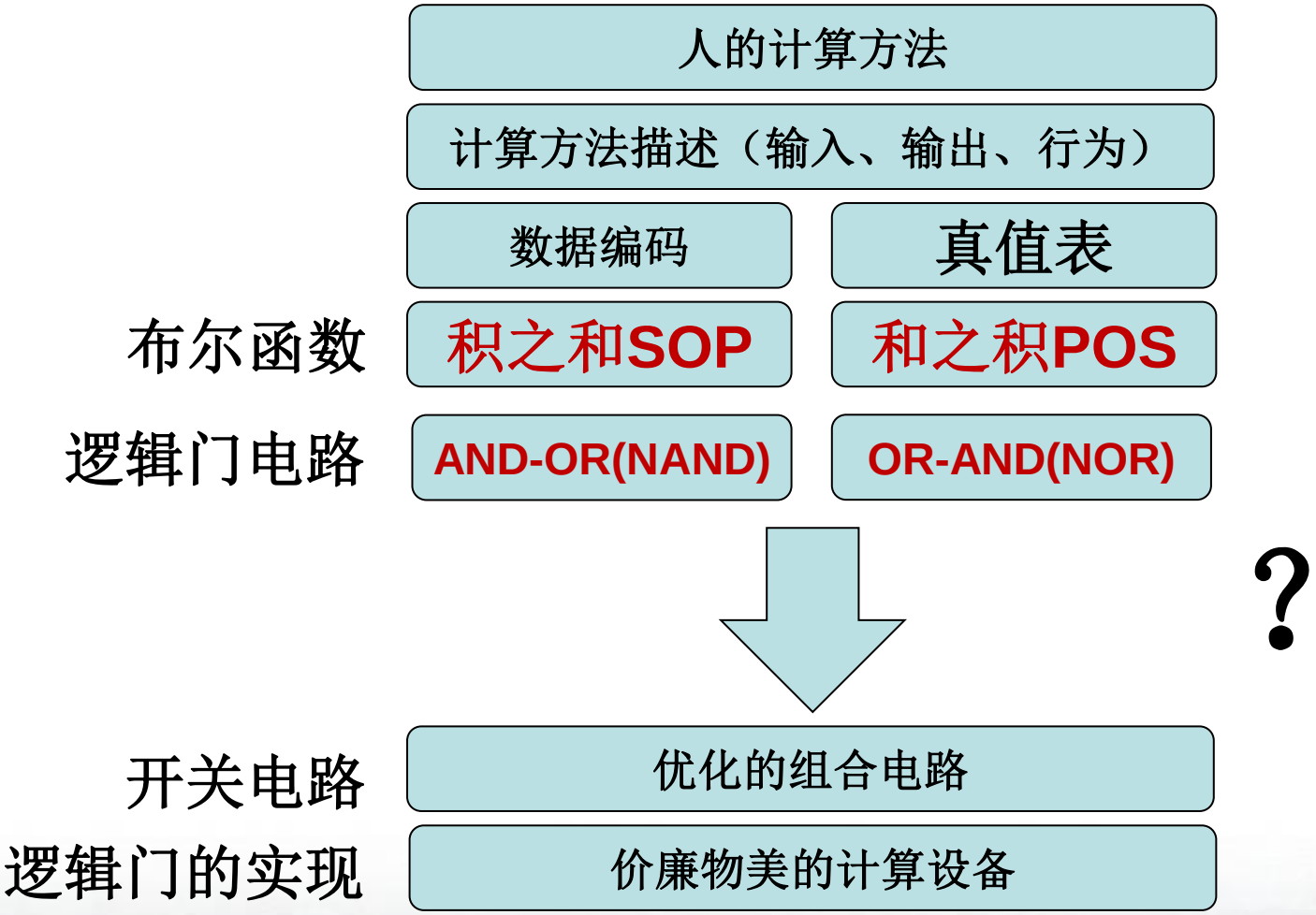
如何做一个能计算的设备?



如何做一个能计算的设备?



如何做一个能计算的设备?



3.2 组合电路实现技术

- 标准逻辑门Standard gates
  - 逻辑门封装芯片
  - 标准单元库
- 规整逻辑Regular logic
  - 多选器multiplexers
  - 译码器decoders
- 两级可编程逻辑Two-level programmable logic
  - PALs
  - PLAs
  - ROMs

随机逻辑(Random logic)

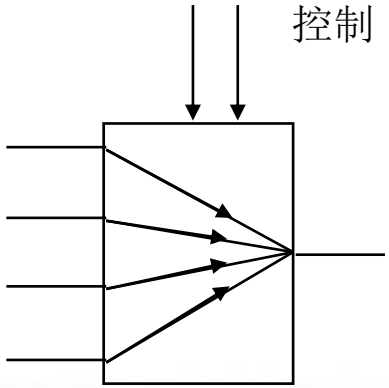
- 采用分立元器件
  - 一个封装内包含多个逻辑门单元
    - 14管脚 IC：6个非门, 4个与非门, 4个异或门
- 很难确定要采用的逻辑门类型
  - 从逻辑映射到与非门/或非门电路
  - 确定需使用的最少封装数目
    - 逻辑函数微小的改变可能会降低成本
- 很难修改已有电路实现
  - 需要重新布线
  - 可能需要新的部件
  - 设计可能剩下多余的逻辑门

规整逻辑(Regular logic)

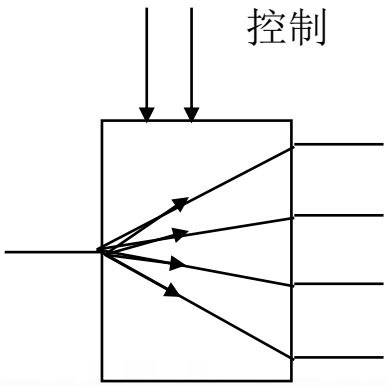
- 需要设计过程更快
- 需要工程师更容易更改设计
- 使设计人员理解和映射到功能更容易
  - 很难根据逻辑门思考
  - 很容易根据大规模多功能模块来思考

2 多路选择器

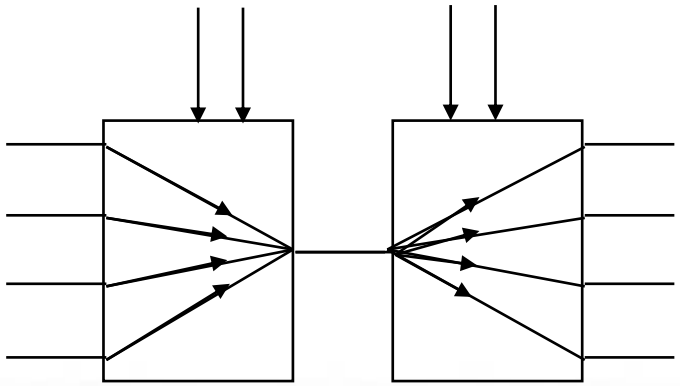
- 门之间直接的点对点连接
- 多个输入连接到一个单一输出
  - 多路选择器 multiplexer
- 一个输入连接到多个输出
  - 多路分配器 demultiplexer



多路选择器



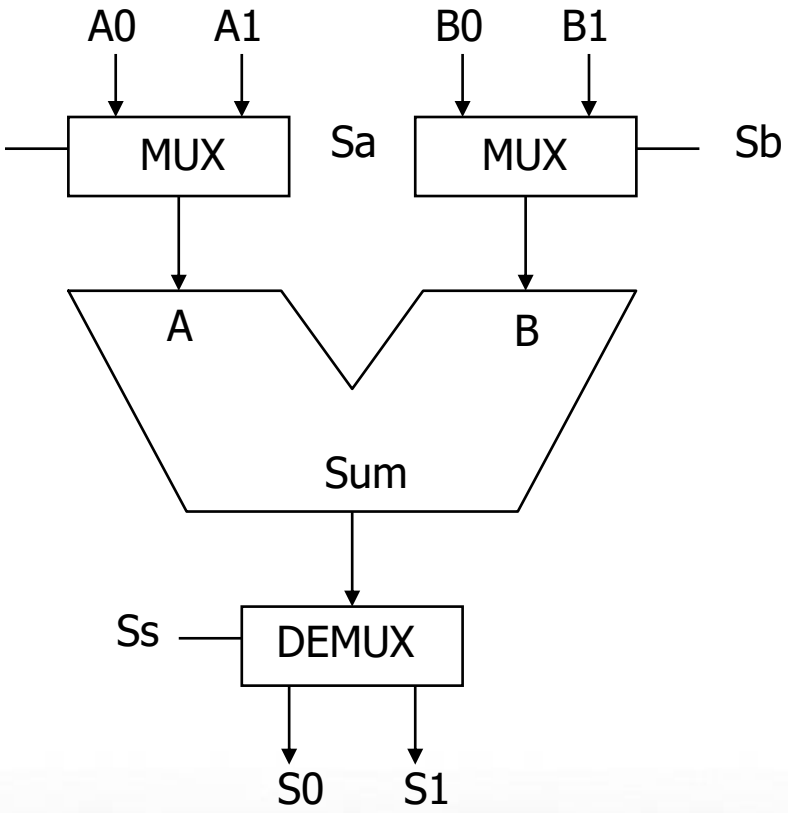
多路分配器



4x4 交换开关

多路选择器和多路分配器

- 在多点互连中使用多路选择器和多路分配器

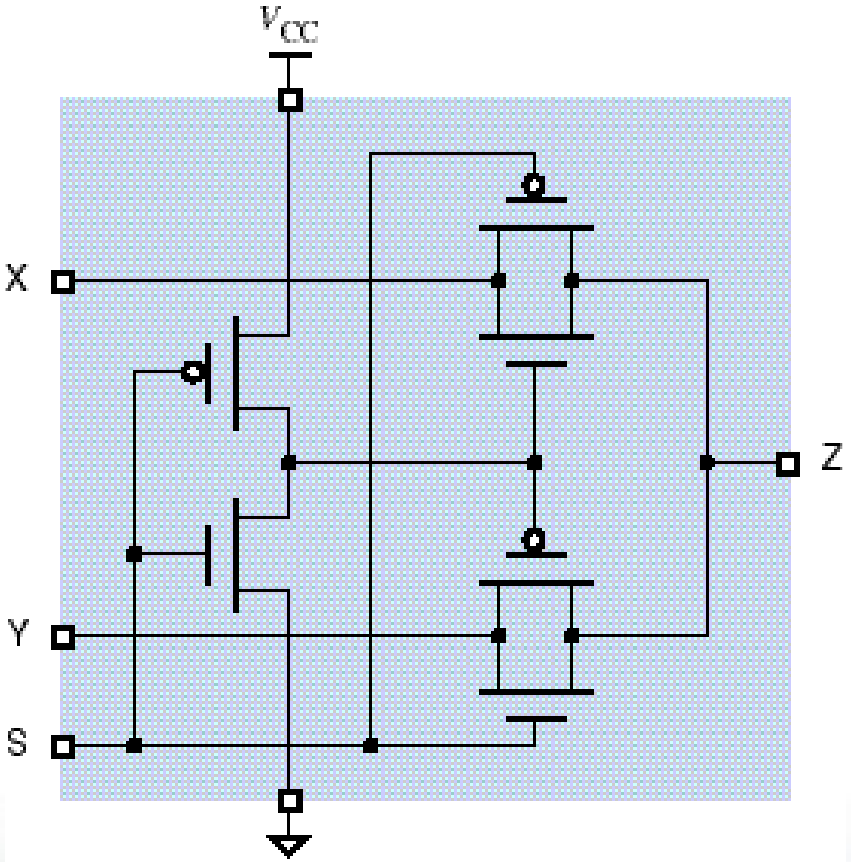


多个输入来源

多个输出目的

特殊的CMOS门：2选1 Mux

- 用传输门实现的2选1逻辑
  - S=0：Z=X
  - S=1：Z=Y
  - $Z = \bar{S}X + SY$



多路选择器

- 一般概念
  - 2^n 数据输入, n控制信号/选择信号, 1输出
  - 用在降2^n点连接到1个点
  - 控制信号是连接到输出的输入信号的二进制编码

$$Z = A' I_0 + A I_1$$

A	Z
0	I_0
1	I_1

功能形式

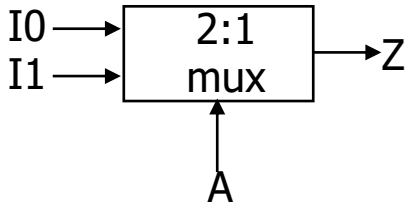
逻辑形式

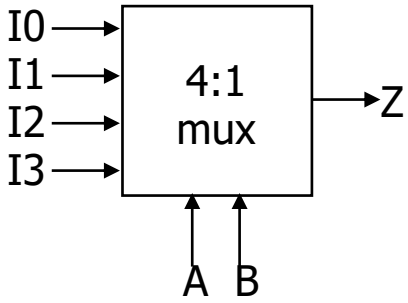
I_1	I_0	A	Z
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

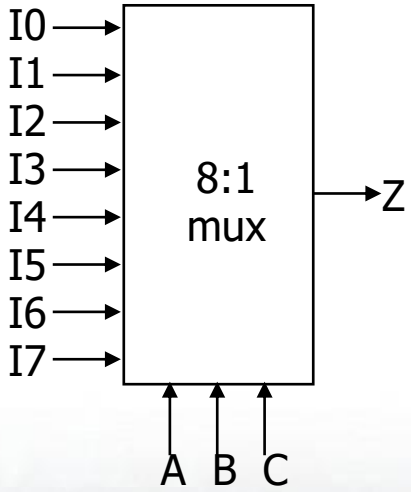
2选1的两种真值表形式

多路选择器

- 2:1 mux: $Z = A'I_0 + AI_1$
- 4:1 mux: $Z = A'B'I_0 + A'BI_1 + AB'I_2 + ABI_3$
- 8:1 mux: $Z = A'B'C'I_0 + A'B'CI_1 + A'BC'I_2 + A'BCI_3 + AB'C'I_4 + AB'CI_5 + ABC'I_6 + ABCI_7$

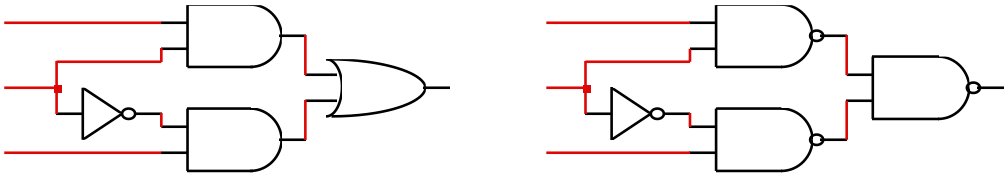




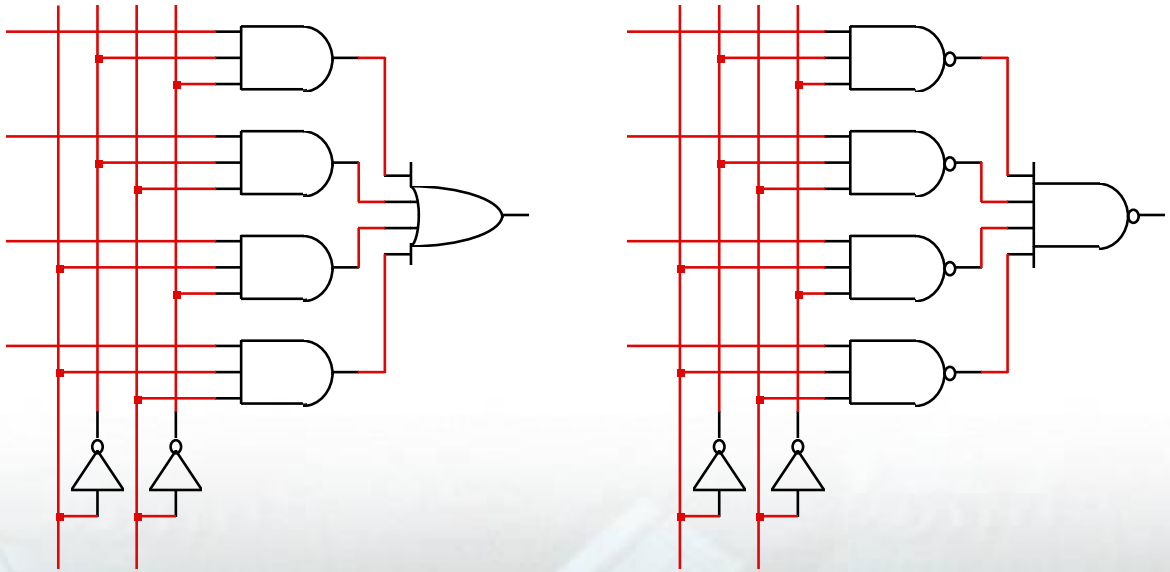


多选器的门级实现

- 2:1 mux

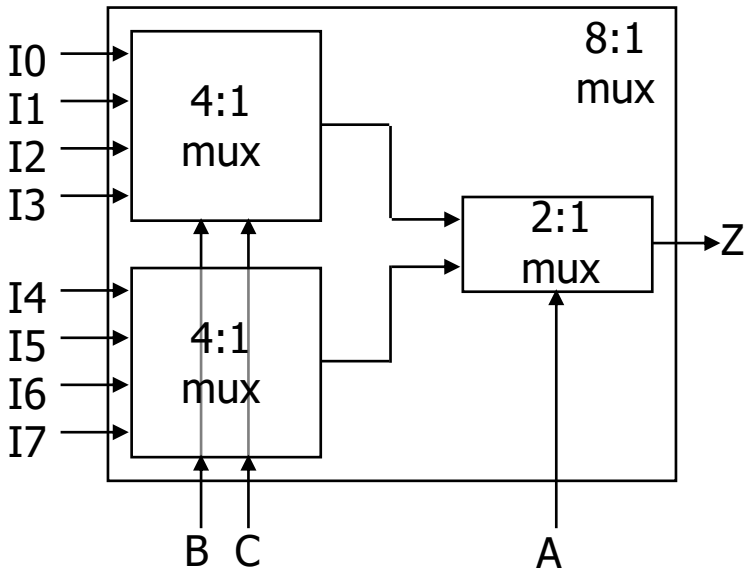


- 4:1 mux

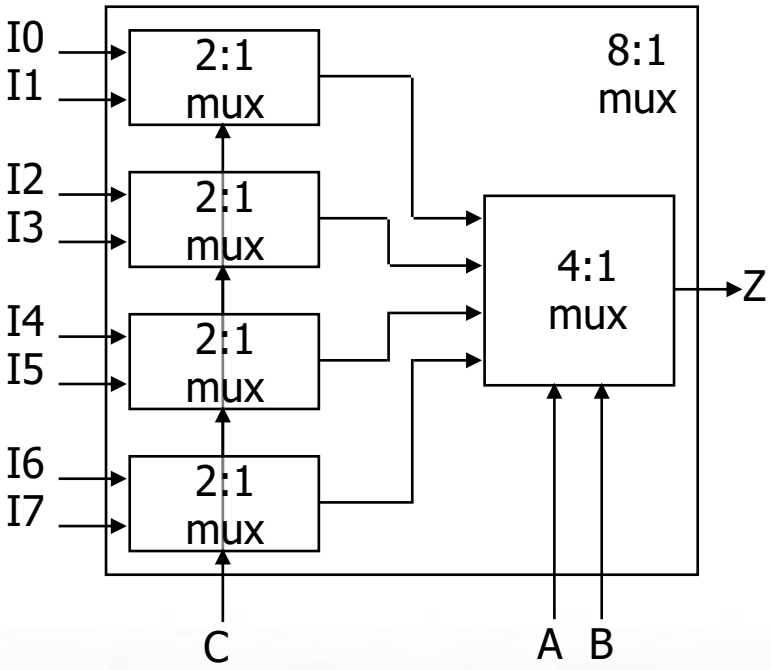


多选择的级联

- 大的多选器由多个小多选器级联形成

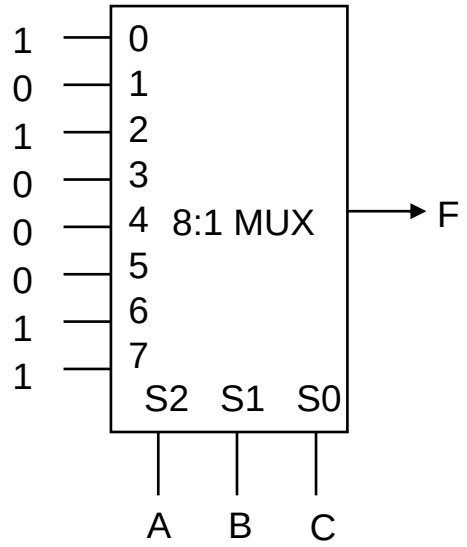


另一种实现



作为通用逻辑的多选器

- 一个2^n:1的多选器可以实现任意n输入变量的函数
  - 变量作为控制输入
  - 数据输入固定连接0或1
  - 本质上是一个查找表（真值表）



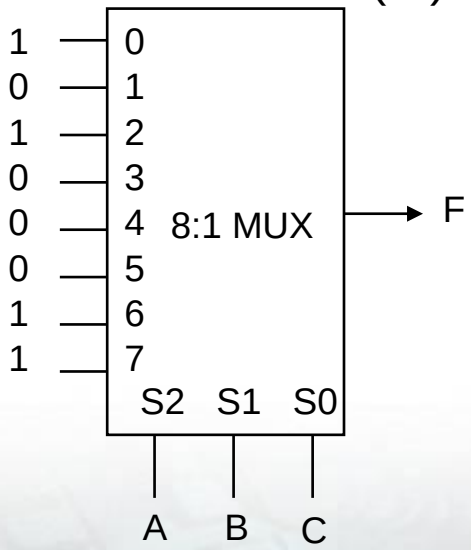
- 例子:
  - F(A,B,C) = m0 + m2 + m6 + m7

= A'B'C' + A'BC' + ABC' + ABC
= A'B'C'(1) + A'B'C(0)
+ A'BC'(1) + A'BC(0)
+ AB'C'(0) + AB'C(0)
+ ABC'(1) + ABC(1)

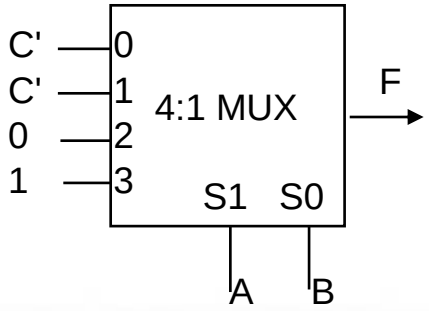
$$Z = A'B'C'I_0 + A'B'CI_1 + A'BC'I_2 + A'BCI_3 + AB'C'I_4 + AB'CI_5 + ABC'I_6 + ABCI_7$$

作为通用逻辑的多选器

- 一个2^{n-1}:1多选器可以实现任意n变量的函数
  - n-1个变量作为控制信号
  - 数据输入连接最后的变量或者它的反
- Example:
  - F(A,B,C) = A'B'C' + A'BC' + ABC' + ABC
    = A'B'(C') + A'B(C') + AB'(0) + AB(1)

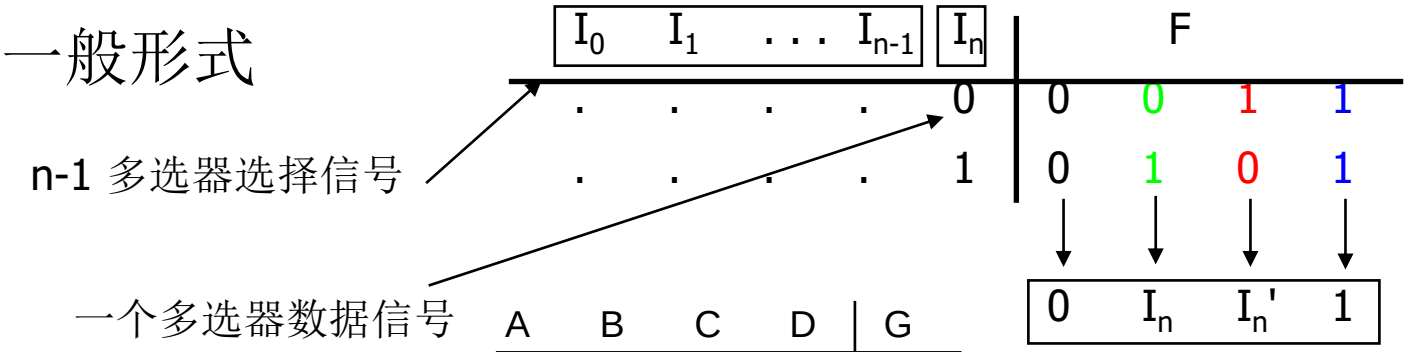


A	B	C	F	
0	0	0	1	C'
0	0	1	0	
0	1	0	1	C'
0	1	1	0	
1	0	0	0	0
1	0	1	0	
1	1	0	1	1
1	1	1	1	



多选器实现逻辑电路

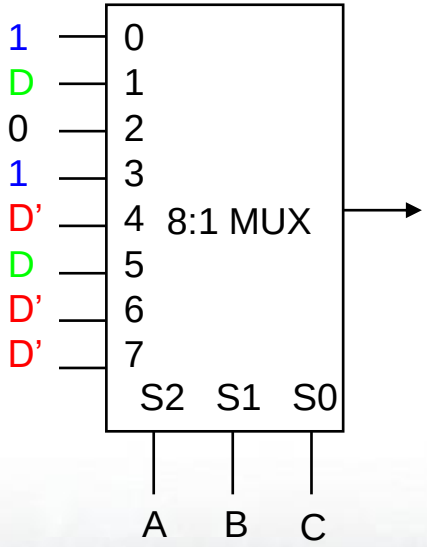
一般形式



例子

选择A,B,C作为选择信号

A	B	C	D	G	
0	0	0	0	1	1
0	0	0	1	1	
0	0	1	0	0	D
0	0	1	1	1	
0	1	0	0	0	0
0	1	0	1	0	
0	1	1	0	1	1
0	1	1	1	1	
1	0	0	0	1	D'
1	0	0	1	0	
1	0	1	0	0	D
1	0	1	1	1	
1	1	0	0	1	D'
1	1	0	1	0	
1	1	1	0	1	D'
1	1	1	1	0	



练习

- 实现 F = B'CD' + ABC' 采用4:1多选器和最少的门

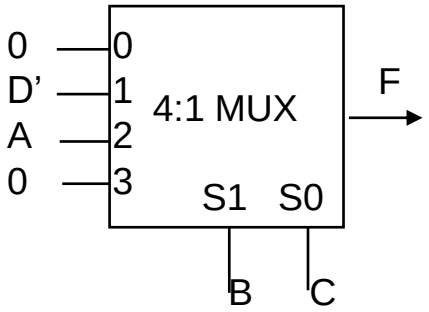
A	B	C	D	Z
0	0	0	0	0
0	0	0	1	0
0	0	1	0	1
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	1
1	1	1	0	0
1	1	1	1	0

0 when B'C'

D' when B'C

A when BC'

0 when BC



Z = B'C'(0) + B'C(D') + BC'(A) + BC(0)

3 多路分配器/译码器

- 译码器/多路分配器：一般概念
  - 一个数据输入，n个控制输入，2^n个输出
  - 控制输入(称为选择 (S, “selects”)) 代表输入需连接输出的二进制索引码
  - 数据输入一般称为使能端(G, “enable”)

1:2 Decoder:

O0 = G • S′
O1 = G • S

2:4 Decoder:

O0 = G • S1′ • S0′
O1 = G • S1′ • S0
O2 = G • S1 • S0′
O3 = G • S1 • S0

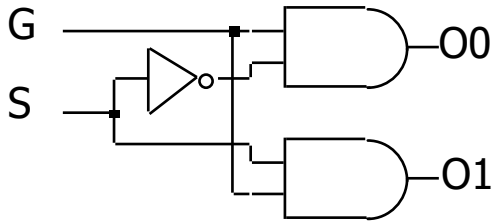
3:8 Decoder:

O0 = G • S2′ • S1′ • S0′
O1 = G • S2′ • S1′ • S0
O2 = G • S2′ • S1 • S0′
O3 = G • S2′ • S1 • S0
O4 = G • S2 • S1′ • S0′
O5 = G • S2 • S1′ • S0
O6 = G • S2 • S1 • S0′
O7 = G • S2 • S1 • S0

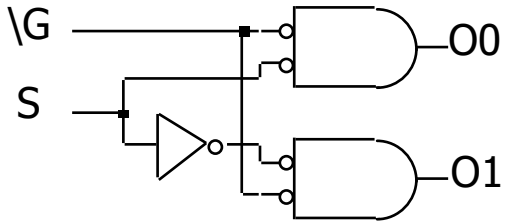
多路分配器的门级实现

1:2 译码器

高有效使能

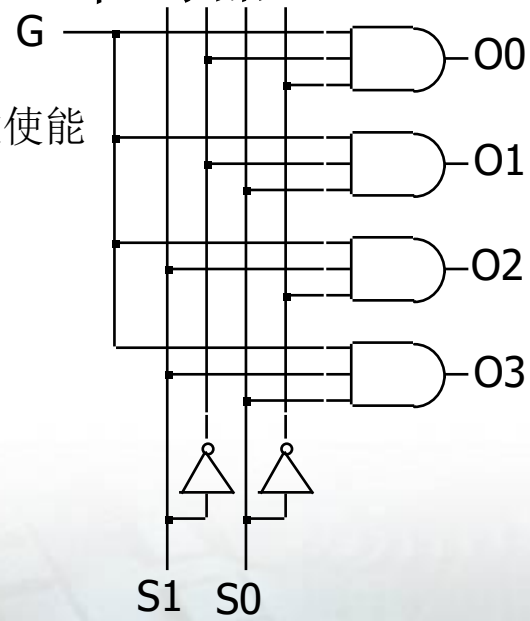


低有效使能

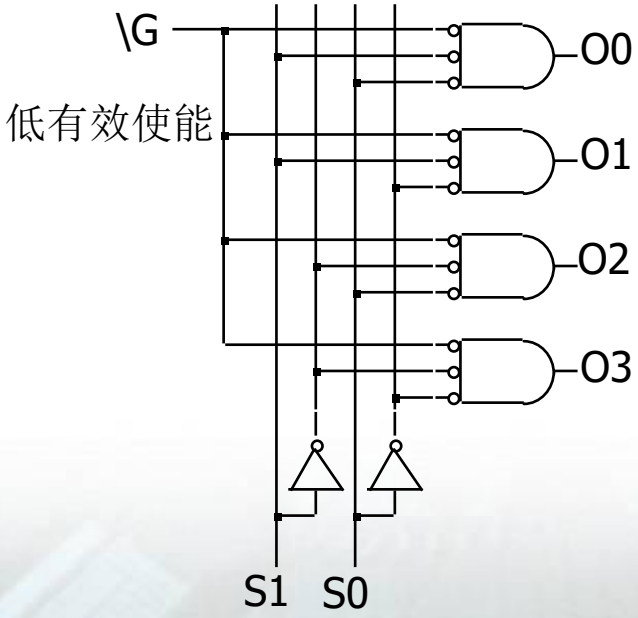


2:4 译码器

高有效使能

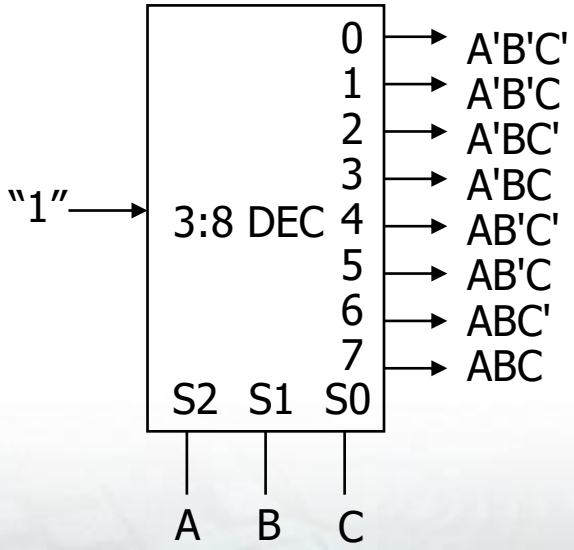


低有效使能



作为通用逻辑的多路分配器

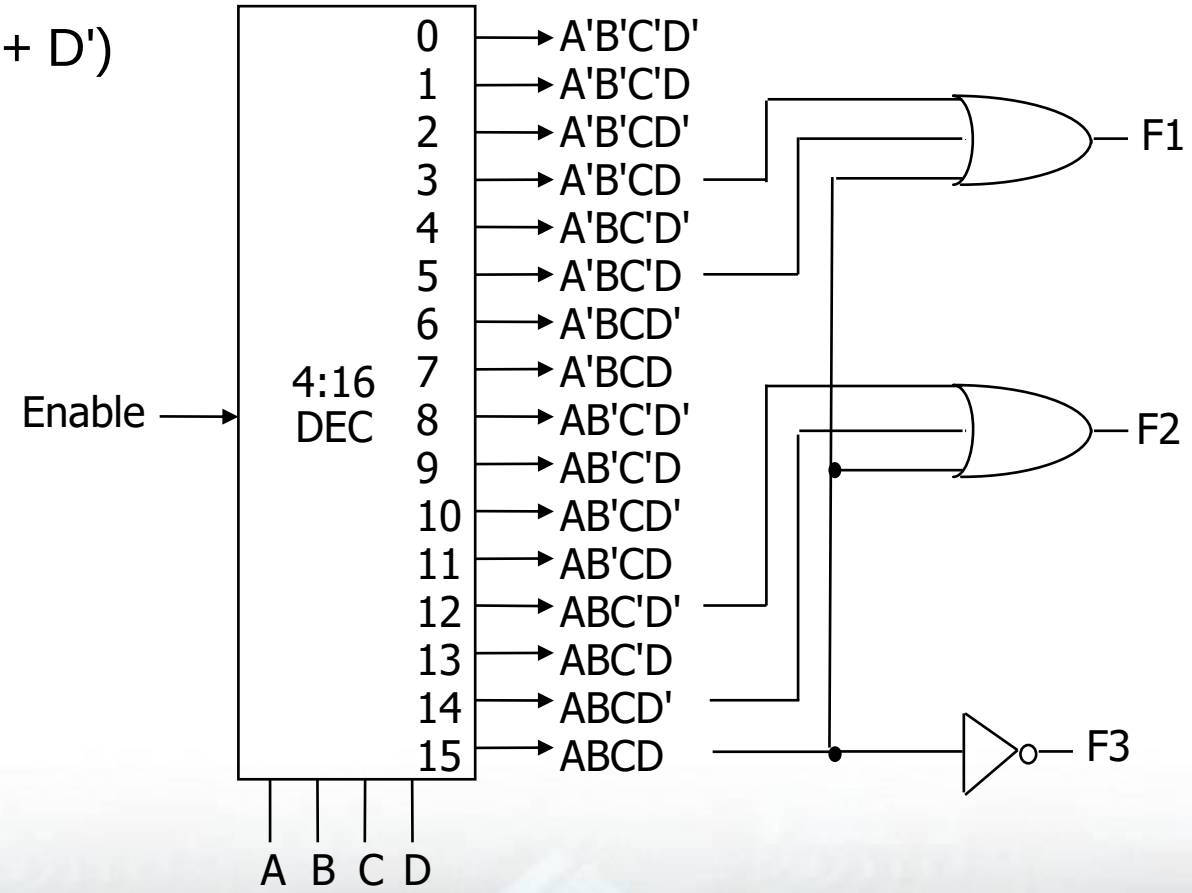
- 一个n:2^n 译码器可以实现任意n个变量的函数
  - 变量作为控制信号
  - 使能信号固定连接1
  - 相应的最小项的和形成函数实现
  - 相当于，最小项的生成器



多路分配器基于输入信号
生成相应的最小项
（对控制信号译码）

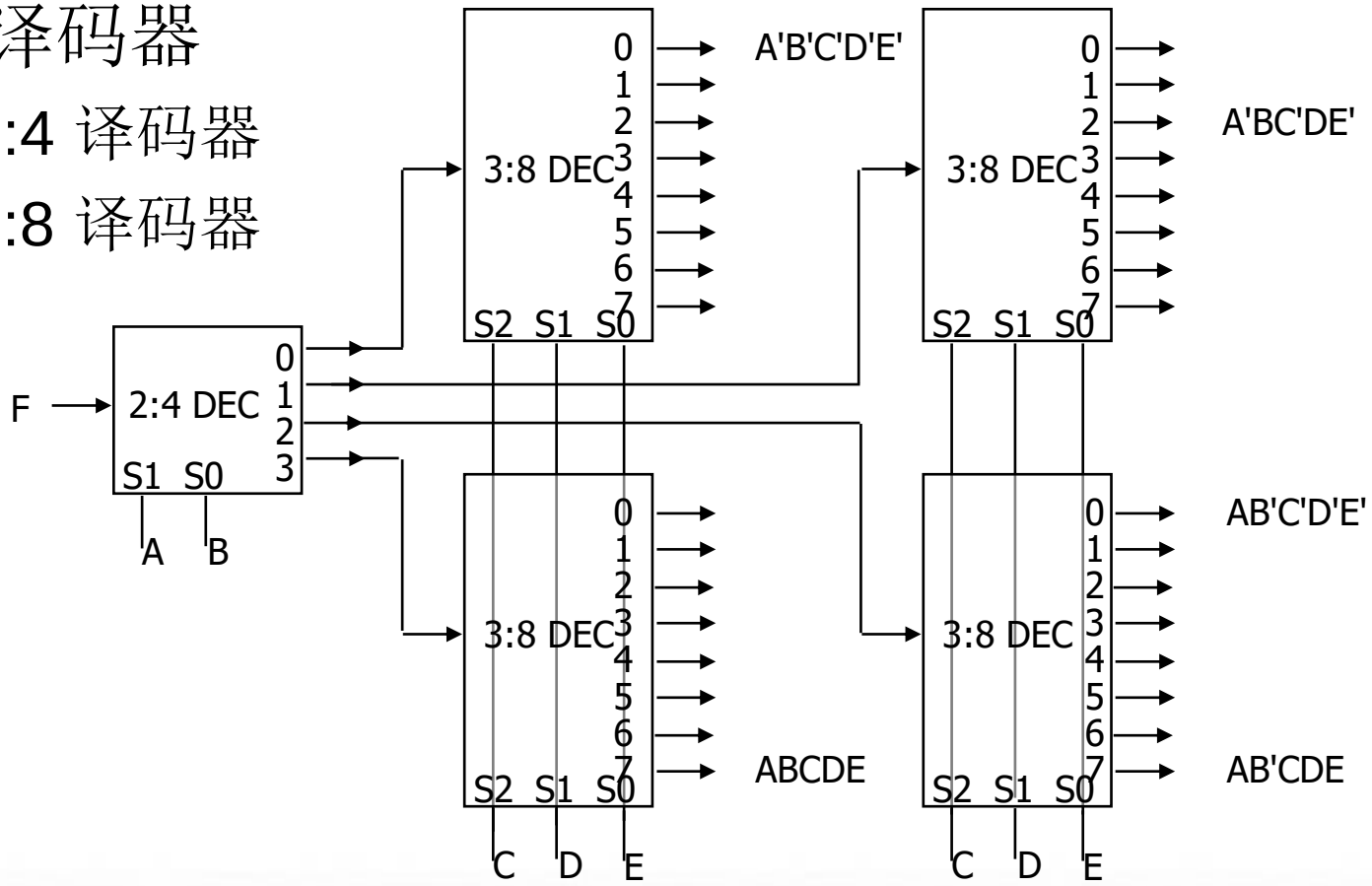
作为通用逻辑的多路分配器

- F1 = A'BC'D + A'B'CD + ABCD
- F2 = ABC'D' + ABC
- F3 = (A' + B' + C' + D')



译码器级联

- 5:32 译码器
  - 1x2:4 译码器
  - 4x3:8 译码器

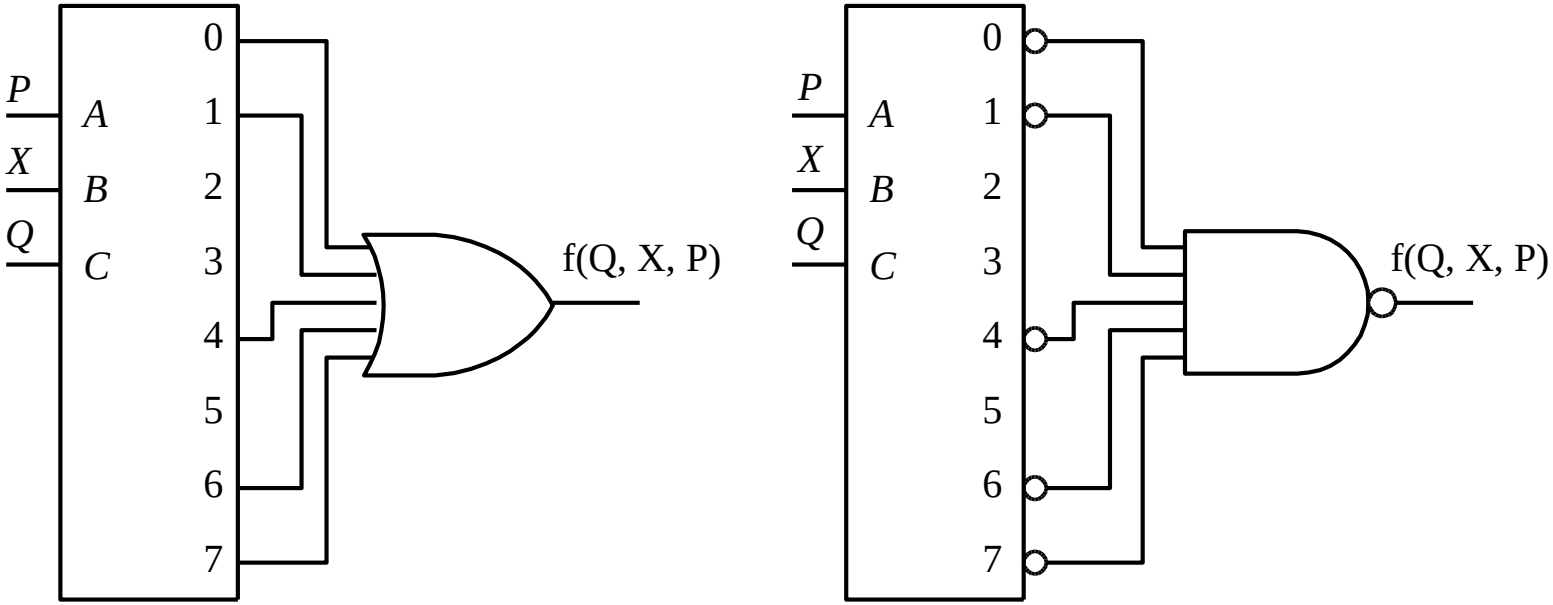


例：实现$f(Q,X,P)=\sum m(0,1,4,6,7)=\prod M(2,3,5)$

- 输出低电平有效译码器：最大项生成器

$$(a)\quad f(Q,X,P)=m_0+m_1+m_4+m_6+m_7$$

$$(b)\quad f(Q,X,P)=\overline{m_0}\cdot\overline{m_1}\cdot\overline{m_4}\cdot\overline{m_6}\cdot\overline{m_7}$$

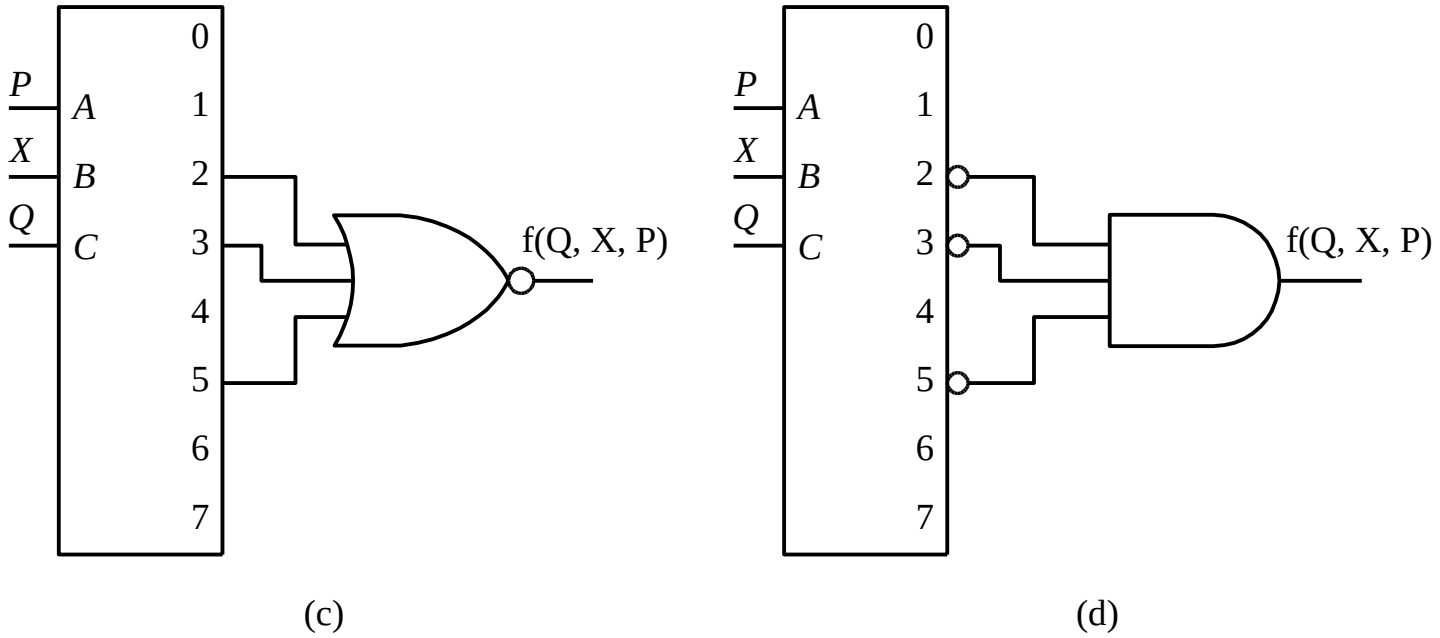


例（续）

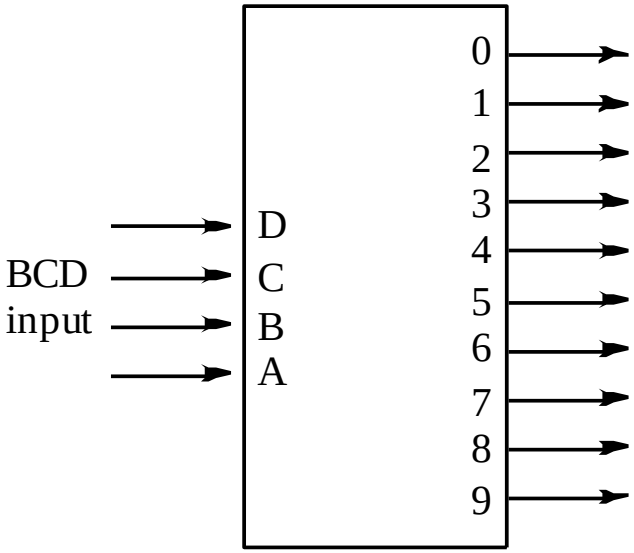
- 输出低电平有效译码器：最大项生成器

$$(c)\quad f(Q,X,P)=\overline{m_2+m_3+m_5}$$

$$(d)\quad f(Q,X,P)=\overline{m_2}\cdot\overline{m_3}\cdot\overline{m_5}=M_2\cdot M_3\cdot M_5$$



BCD到十进制译码器



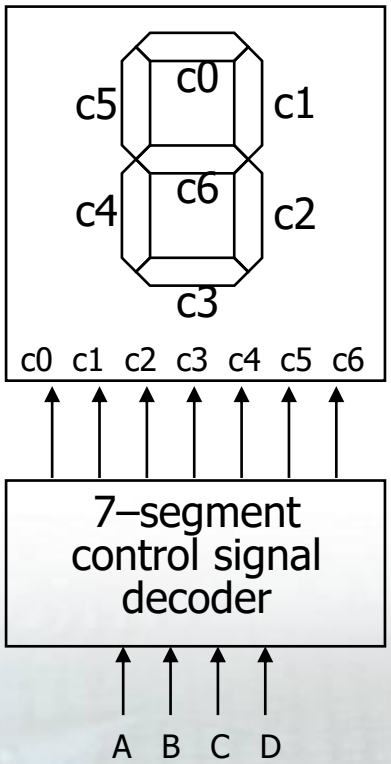
(a)

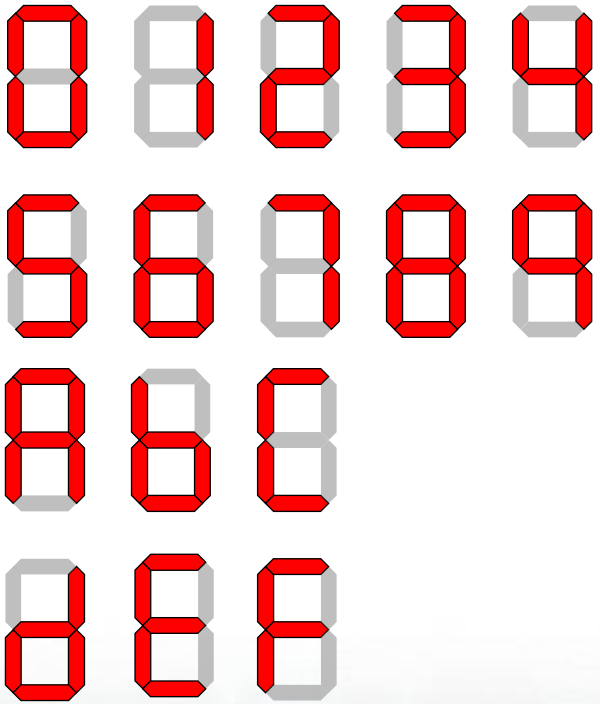
BCD code DCBA	Decimal digits
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9

(b)

7段显示译码器

- 理解问题
  - 输入是4位十六进制数字 (A, B, C, D)
  - 输出是7段显示的控制信号
    (7个输出 C0 – C6)
- 结构图





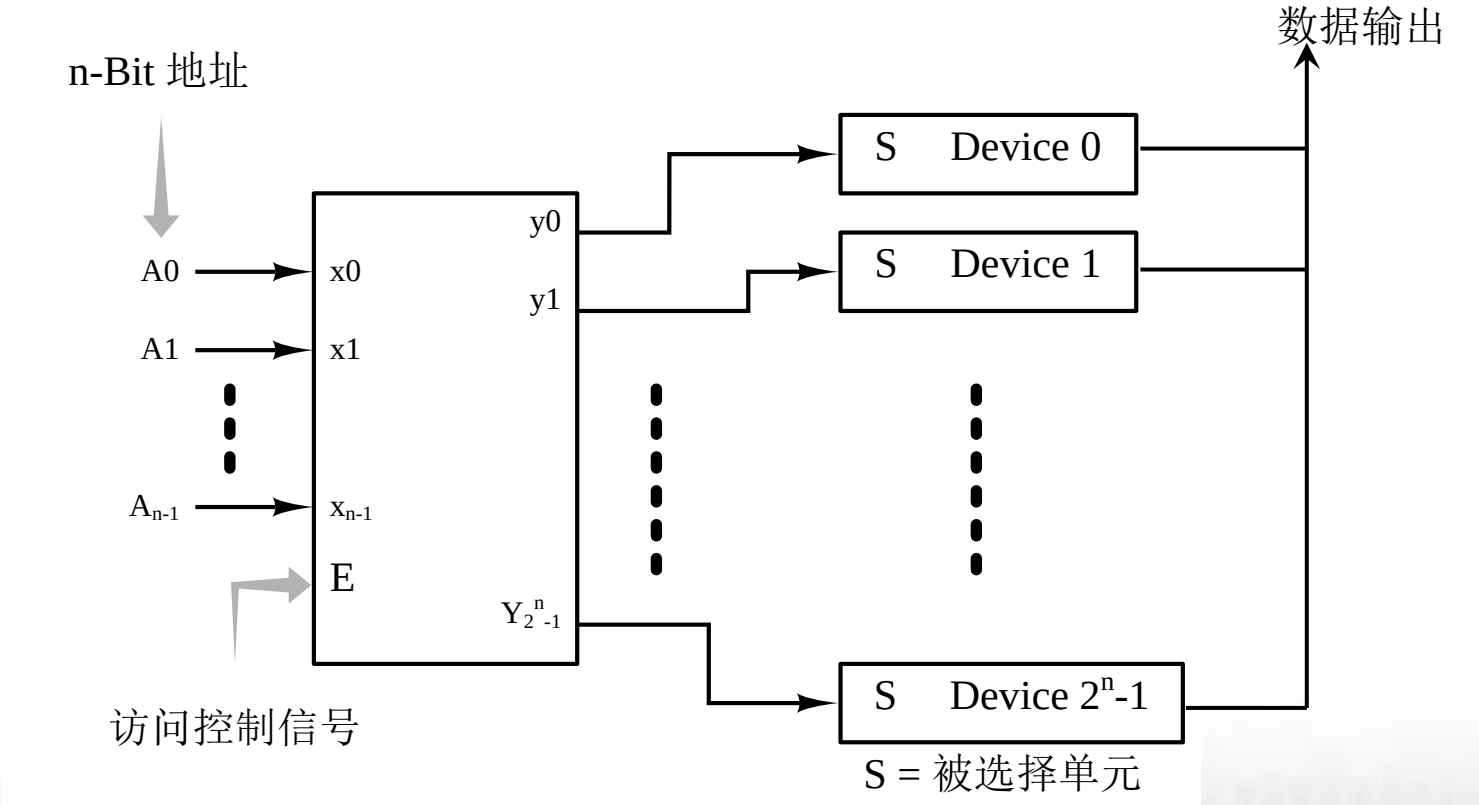
问题形式化描述

- 卡诺图
  - 无关项
- 选择实现目标
- 进行实现流程
  - 化简卡诺图
- **直接Verilog描述真值表**

A	B	C	D	C0	C1	C2	C3	C4	C5	C6
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	0	0	1	1
1	0	1	0	1	1	1	0	1	1	1
1	0	1	1	0	0	1	1	1	1	1
1	1	0	0	1	0	0	1	1	1	0
1	1	0	1	0	1	1	1	1	0	1
1	1	1	0	1	0	0	1	1	1	1
1	1	1	1	1	0	0	0	1	1	1

存储地址译码器

- 处理器的指令译码器
- 存储器地址译码——用于控制总线的共享输出



指令译码器：RISC-V 指令编码

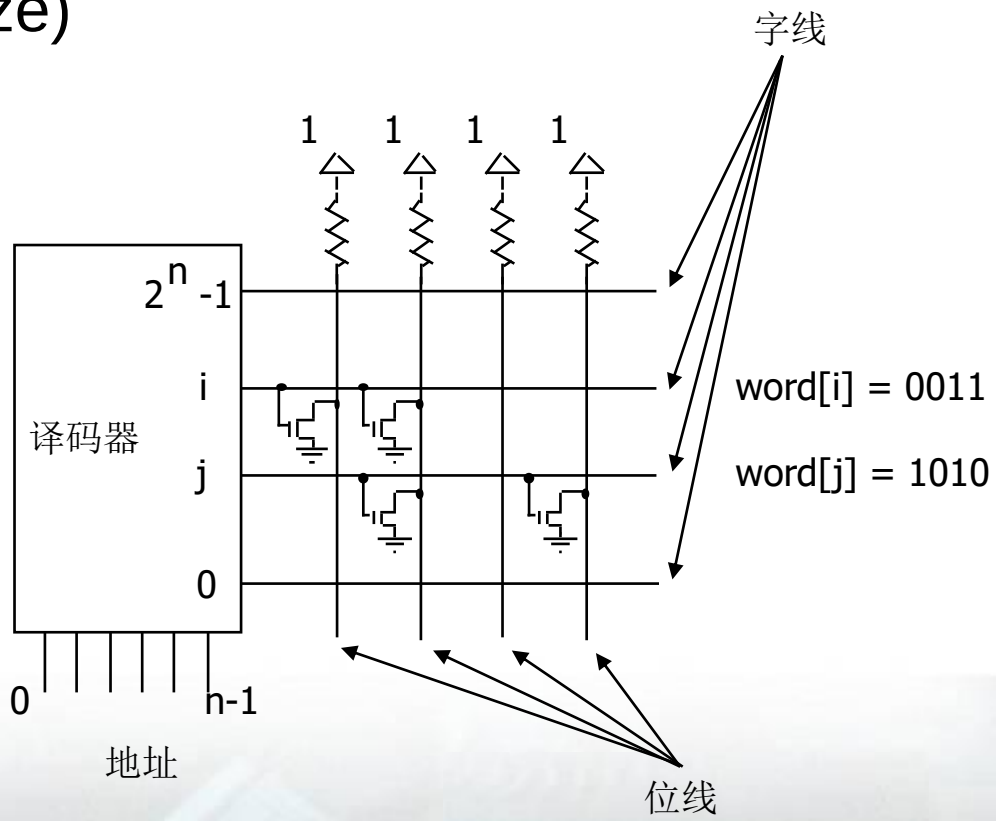
Func			OP1		OP	
imm[11:0]		rs1	000	rd	0010011	ADDI
imm[11:0]		rs1	010	rd	0010011	SLTI
imm[11:0]		rs1	011	rd	0010011	SLTIU
imm[11:0]		rs1	100	rd	0010011	XORI
imm[11:0]		rs1	110	rd	0010011	ORI
imm[11:0]		rs1	111	rd	0010011	ANDI
0000000	shamt	rs1	001	rd	0010011	SLLI
0000000	shamt	rs1	101	rd	0010011	SRLI
0100000	shamt	rs1	101	rd	0010011	SRAI
0000000	rs2	rs1	000	rd	0110011	ADD
0100000	rs2	rs1	000	rd	0110011	SUB
0000000	rs2	rs1	001	rd	0110011	SLL
0000000	rs2	rs1	010	rd	0110011	SLT
0000000	rs2	rs1	011	rd	0110011	SLTU
0000000	rs2	rs1	100	rd	0110011	XOR
0000000	rs2	rs1	101	rd	0110011	SRL
0100000	rs2	rs1	101	rd	0110011	SRA
0000000	rs2	rs1	110	rd	0110011	OR
0000000	rs2	rs1	111	rd	0110011	AND

RISC-V 指令编码（续）

imm[31:12]				rd	0110111	LUI
imm[31:12]				rd	0010111	AUIPC
imm[20|10:1|11|19:12]				rd	1101111	JAL
imm[11:0]		rs1	000	rd	1100111	JALR
imm[12|10:5]	rs2	rs1	000	imm[4:1|11]	1100011	BEQ
imm[12|10:5]	rs2	rs1	001	imm[4:1|11]	1100011	BNE
imm[12|10:5]	rs2	rs1	100	imm[4:1|11]	1100011	BLT
imm[12|10:5]	rs2	rs1	101	imm[4:1|11]	1100011	BGE
imm[12|10:5]	rs2	rs1	110	imm[4:1|11]	1100011	BLTU
imm[12|10:5]	rs2	rs1	111	imm[4:1|11]	1100011	BGEU
imm[11:0]		rs1	000	rd	0000011	LB
imm[11:0]		rs1	001	rd	0000011	LH
imm[11:0]		rs1	010	rd	0000011	LW
imm[11:0]		rs1	100	rd	0000011	LBU
imm[11:0]		rs1	101	rd	0000011	LHU
imm[11:5]	rs2	rs1	000	imm[4:0]	0100011	SB
imm[11:5]	rs2	rs1	001	imm[4:0]	0100011	SH
imm[11:5]	rs2	rs1	010	imm[4:0]	0100011	SW

存储器直接实现真值表

- 存储器：两维的0，1整列
  - 每行一个字（word）
  - 行大小 = 字长(word-size)
  - 地址（address）索引
  - 地址(address) 是输入
  - 所选的字是输出
- ROM只读存储器
- RAM随机访问存储器



ROM内部组织

存储器和组合逻辑

- 用存储器实现组合逻辑

F0 = A' B' C + A B' C' + A B' C

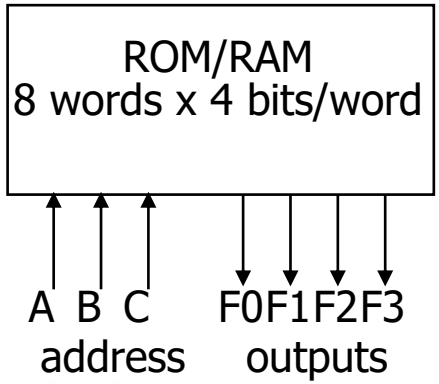
F1 = A' B' C + A' B C' + A B C

F2 = A' B' C' + A' B' C + A B' C'

F3 = A' B C + A B' C' + A B C'

A	B	C	F0	F1	F2	F3
0	0	0	0	0	1	0
0	0	1	1	1	1	0
0	1	0	0	1	0	0
0	1	1	0	0	0	1
1	0	0	1	0	1	1
1	0	1	1	0	0	0
1	1	0	0	0	0	1
1	1	1	0	1	0	0

真值表



框图

3.3 卡诺图化简

- 布尔表达式的化简：组合最小项（最大项）
- 卡诺图以图形的方式进行布尔表达式化简
- 合并定理
  - $PA + PA' = P$

A	B	C	Y
0	0	0	**1**
0	0	1	**1**
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

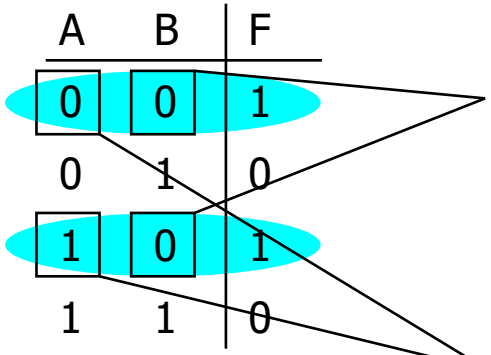
Y \ AB / C	00	01	11	10
0	**1**	0	0	0
1	**1**	0	0	0

Y \ AB / C	00	01	11	10
0	$\bar{A}\bar{B}\bar{C}$	$\bar{A}B\bar{C}$	$AB\bar{C}$	$A\bar{B}\bar{C}$
1	$\bar{A}\bar{B}C$	$\bar{A}BC$	ABC	$A\bar{B}C$

合并定理

- 化简的关键工具：A (B’ + B) = A
- 两级逻辑化简的基础

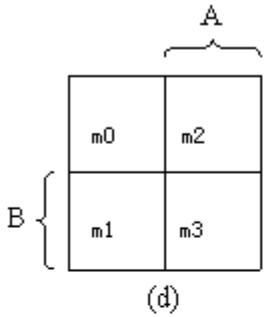
$$F = A'B'+AB' = (A'+A)B' = B'$$



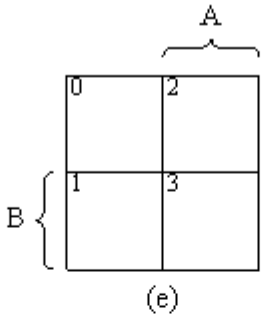
B在真值表的两行有相同的值
– B被保留

A在真值表的两行有不同的值
– A被去除

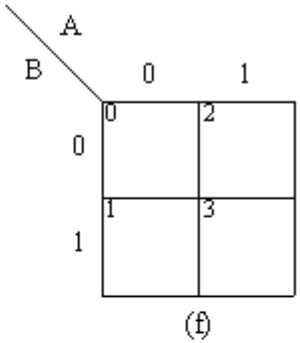
2变量的卡诺图



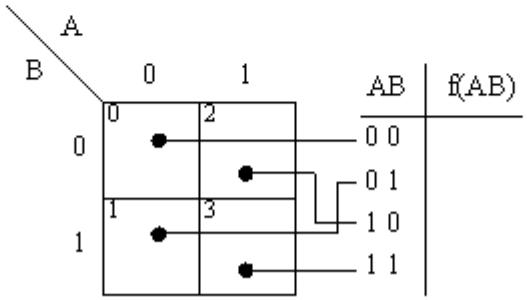
(d)



(e)

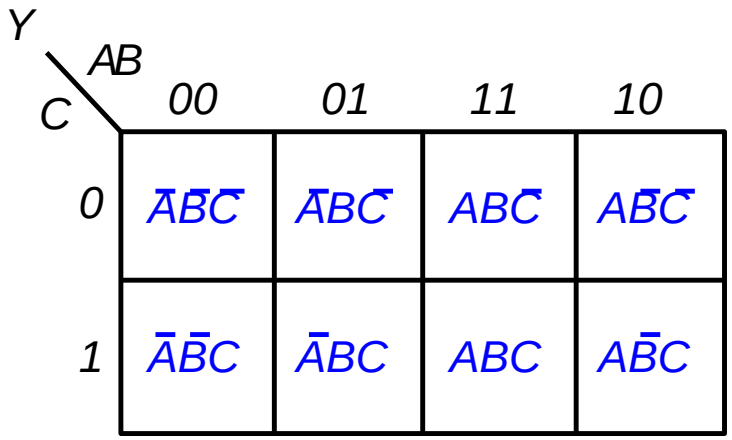


(f)



(g)

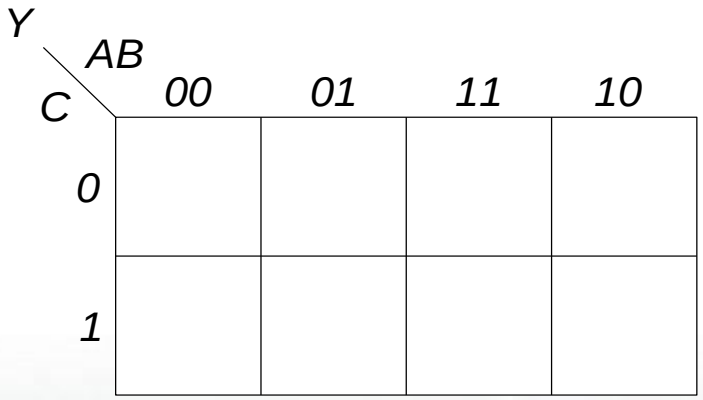
2变量卡诺图



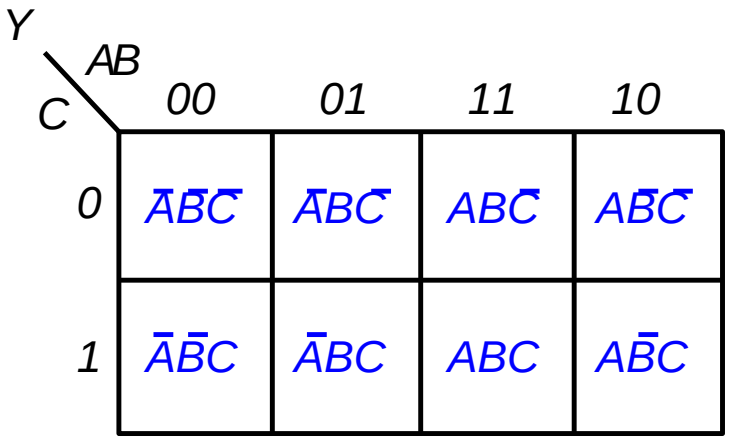
Truth Table

A	B	C	Y
0	0	0	0
0	0	1	0
0	1	0	**1**
0	1	1	**1**
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	**1**

K-Map



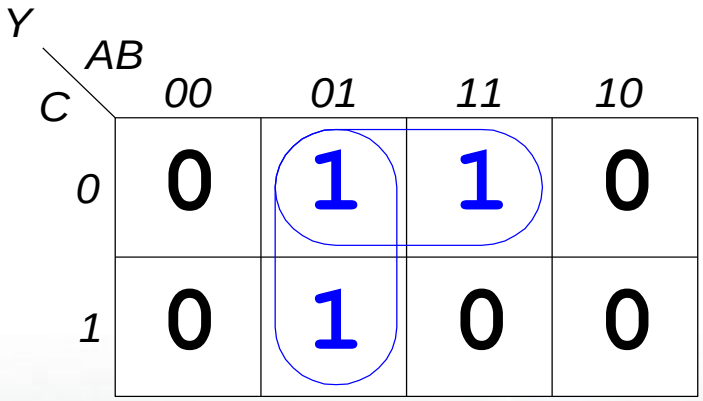
3变量卡诺图化简



Truth Table

A	B	C	Y
0	0	0	0
0	0	1	0
0	1	0	**1**
0	1	1	**1**
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	**1**

K-Map



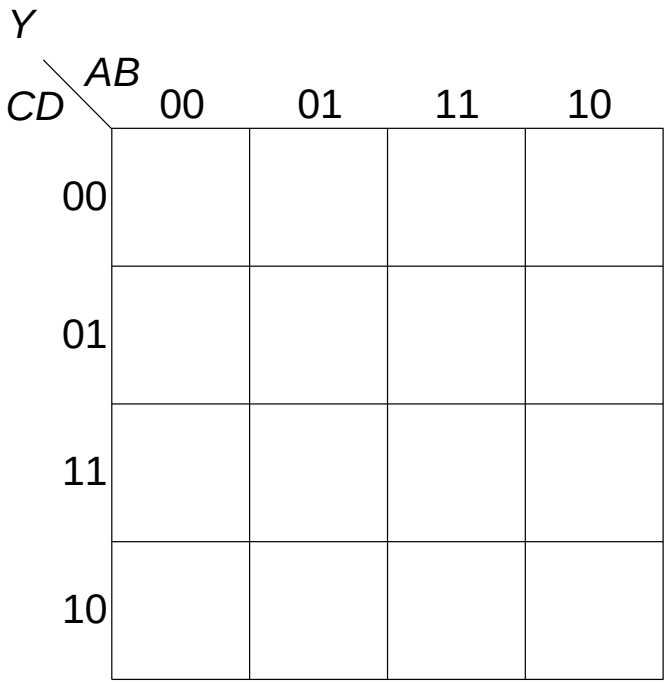
$$Y = \bar{A}B + B\bar{C}$$

卡诺图化简规则：最小项化简，NAND电路

- 每个‘1’至少被圈在一个方形组中
- 每个方形组在每个维度必须是2幂 (如 1, 2, 4,…)
- 每个方形组必须尽可能的大
- 方形组可以在边界循环
- “无关项(X)”只有在有利于化简的情况下才能被圈在方形组中。

4变量卡诺图化简

A	B	C	D	Y
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0



4变量卡诺图化简

A	B	C	D	Y
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

Y	AB 00	01	11	10
CD 00	1	0	0	1
01	0	1	0	1
11	1	1	0	0
10	1	1	0	1

卡诺图化简指导原则

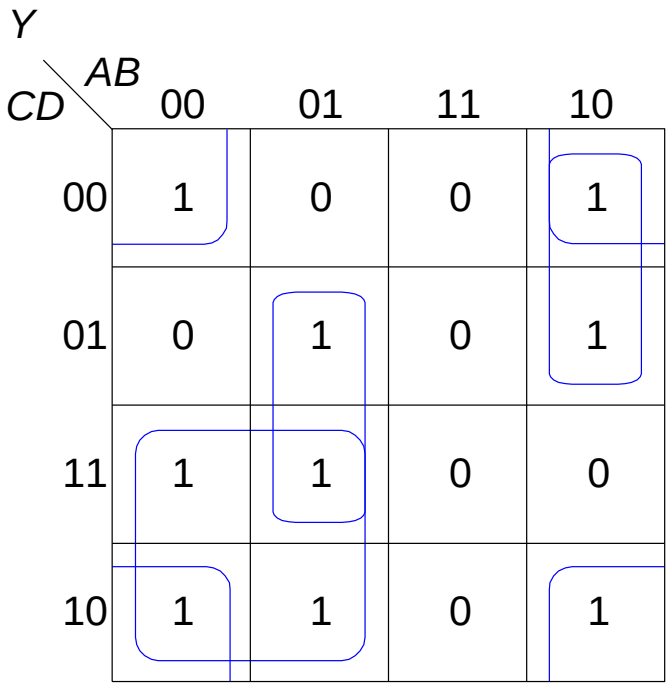
- n变量卡诺图中的每个单元都有n个逻辑相邻的单元。
- 单元可以被合并为大小为 $2,4,8,\ldots,2^k$ 的方形组。
- 每个被合并组中包含的所有单元都对一些变量有相同的值。
- 合并尽可能多的单元，这将导致组所对应的项内字母的个数最少（扇入）。
- 尽可能用最少的组覆盖所有的最小项，这将导致结果中包含最少的积项（门数和最后一级或门的扇入）。
- 应该从最“孤立”单元开始。

卡诺图化简：算法1

- 1. 计算每个最小项的相邻度
- 2. 选择未被覆盖的相邻度最小的最小项，如果有多个相同的选项，任选一个。
- 3. 生成包含该最小项的尽可能大的方形组，如果存在多个相同规模的组，任选一个。
- 4. 重复步骤2和步骤3，直到所有的最小项都被覆盖。

4变量卡诺图化简

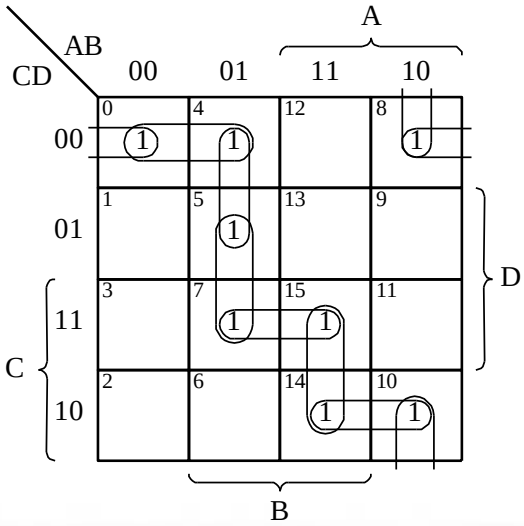
A	B	C	D	Y
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0



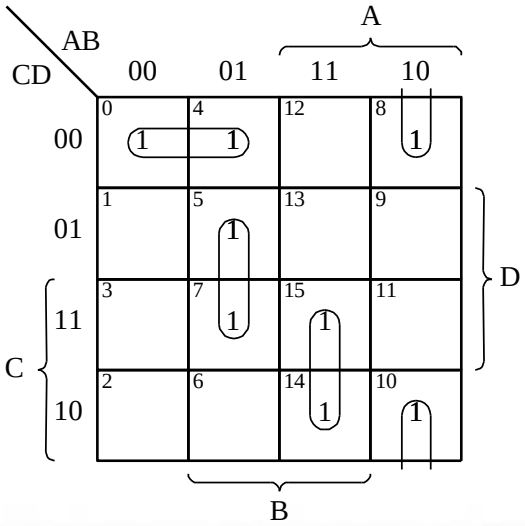
$$Y = \bar{A}C + \bar{A}BD + A\bar{B}\bar{C} + \bar{B}\bar{D}$$

相邻度相同的函数例子

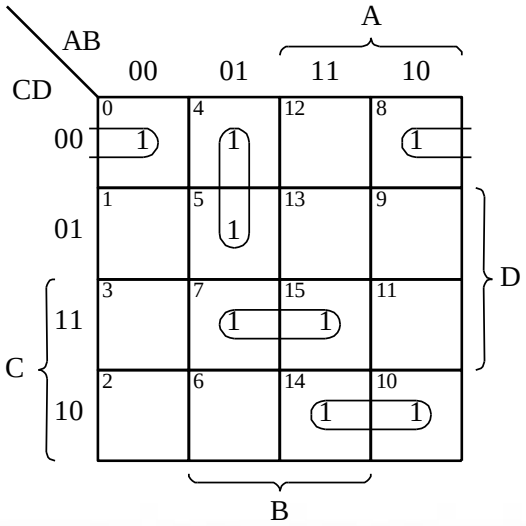
$$f(A,B,C,D) = \sum m(0,4,5,7,8,10,14,15)$$



(a)

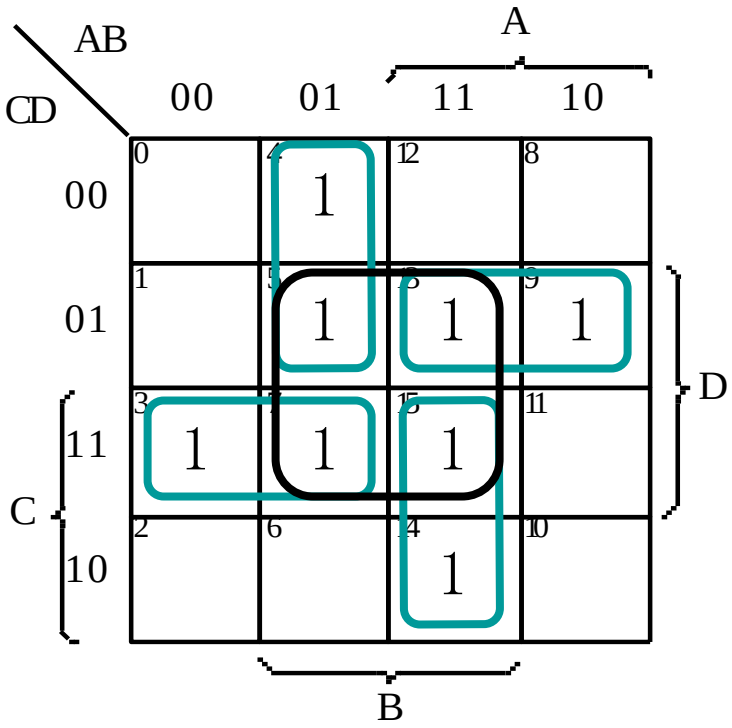


(b)



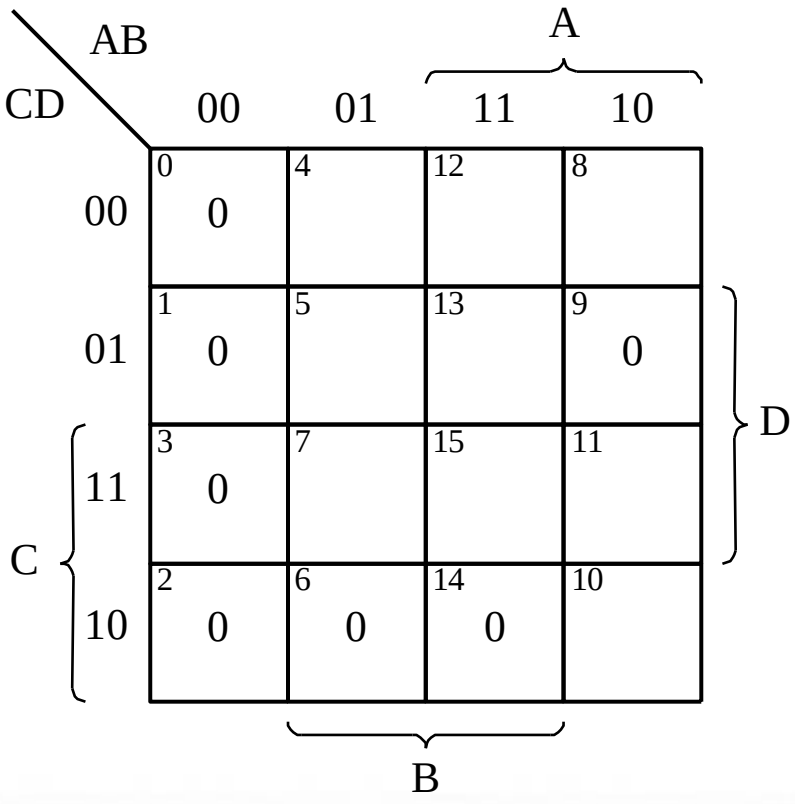
(c)

典型的例子

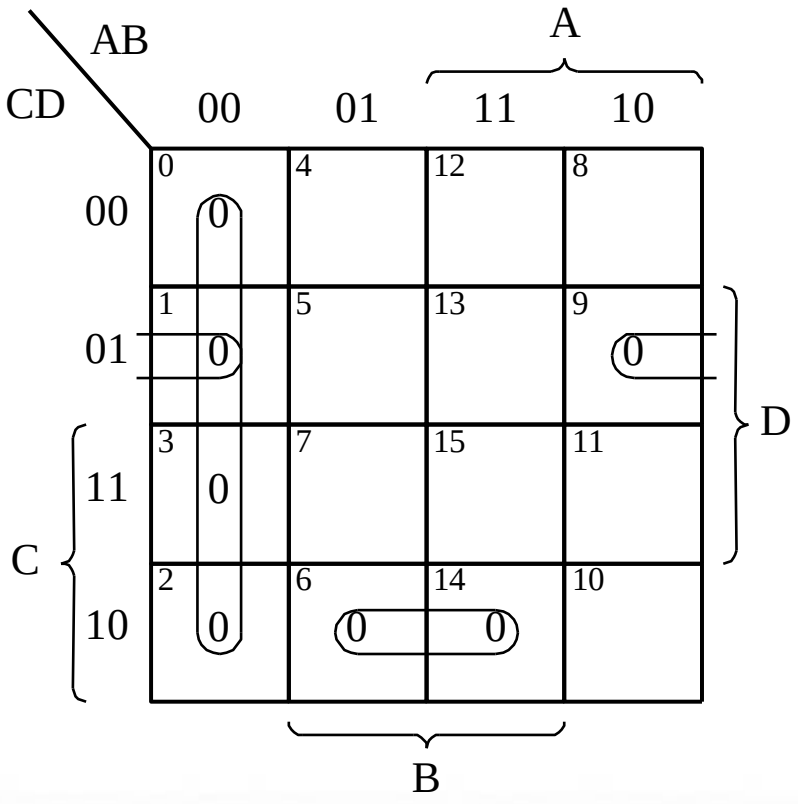


图中蓝色的组不在最后的覆盖中

最大项卡诺图化简：NOR电路



(a)



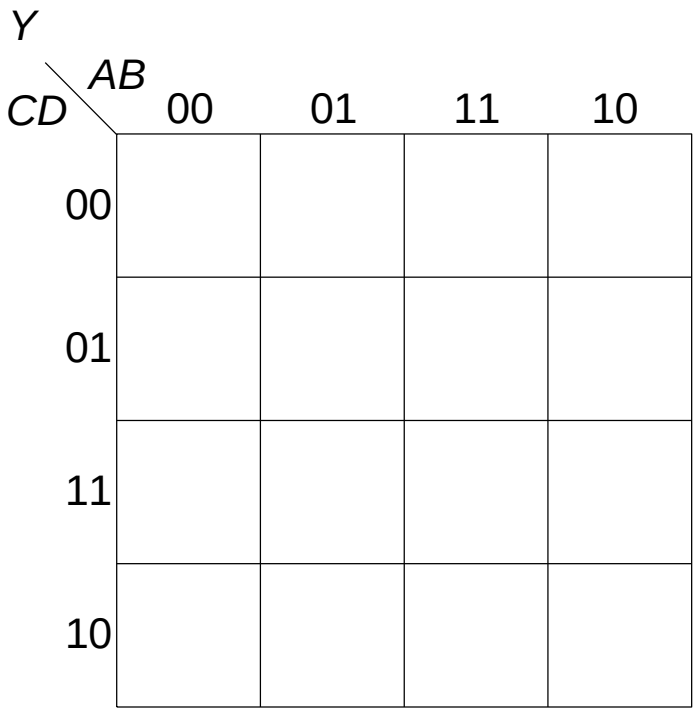
(b)

回忆：非确定函数

- 开关函数非完全确定的原因
  - 一些确定的输入组合不会产生
  - 一些输出为0或1只对特定的输入组合成立
- 无关(don't care)最小项：忽略一些最小项
- 无关最大项：忽略一些最大项
- 表示
  - 无关最小项：d_i
  - 无关最大项：D_i
- 例子：
  - 无关最小项：$f(A,B,C)=\Sigma m(0,3,7)+d(4,5)$
  - 无关最大项：$f(A,B,C)=\Pi M(1,2,6)\cdot D(4,5)$

带无关项的卡诺图

A	B	C	D	Y
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	X
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	X
1	0	1	1	X
1	1	0	0	X
1	1	0	1	X
1	1	1	0	X
1	1	1	1	X



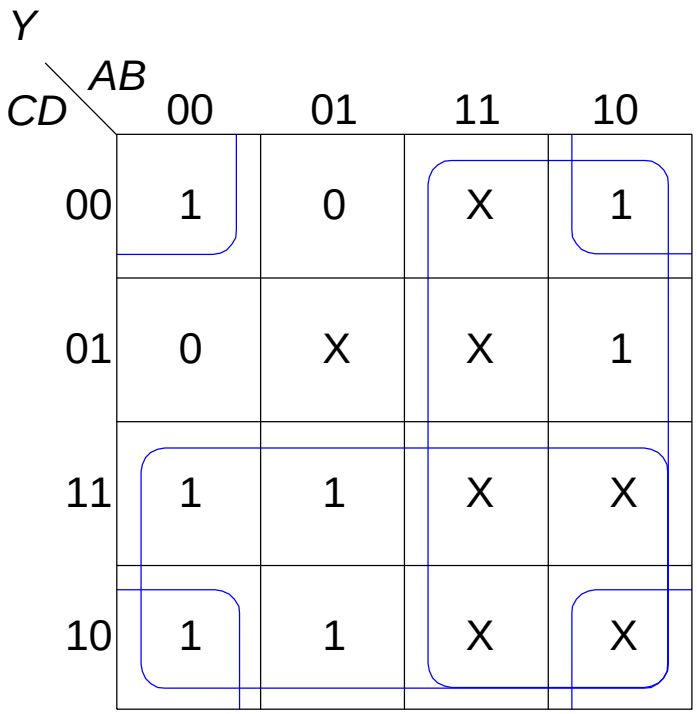
带无关项的卡诺图

A	B	C	D	Y
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	X
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	X
1	0	1	1	X
1	1	0	0	X
1	1	0	1	X
1	1	1	0	X
1	1	1	1	X

Y	AB 00	01	11	10
CD 00	1	0	X	1
01	0	X	X	1
11	1	1	X	X
10	1	1	X	X

带无关项的卡诺图

A	B	C	D	Y
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	X
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	X
1	0	1	1	X
1	1	0	0	X
1	1	0	1	X
1	1	1	0	X
1	1	1	1	X



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

非确定函数的化简

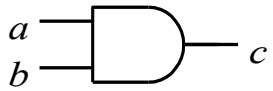
- 在化简非确定函数时，可以选择无关项为1或者0的确定值。
- 原则是使卡诺图中划分的方形组大于不包括无关项的卡诺图的方形组。
- 在选择最小覆盖时，忽略那些没有被选择为确定值的非确定项。
- 可以根据无关项是否对函数的化简有帮助来决定无关项的确定值。
- 包含无关项的函数的SOP化简函数和POS化简函数并不一定在形式上互补。
- 产生非确定结果的输入组合，在真实的电路中有确定的结果。
- **非确定性函数卡诺图化简中，最终覆盖中的每个组应该至少包含一个确定性最小项（最大项）。**

回忆：传播延迟

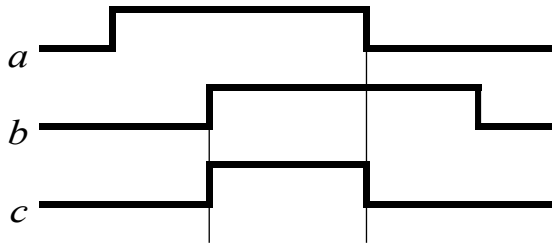
- 两种典型的传播延迟参数
  - t_{PLH} = 低电平到高电平输出的传输延迟。
  - t_{PHL} = 高电平到低电平输出的传输延迟。
- 近似传播延迟

$$t_{PD} = \frac{t_{PLH} + t_{PHL}}{2}$$

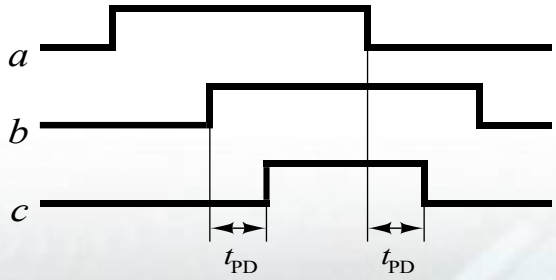
- 逻辑门的传播延迟



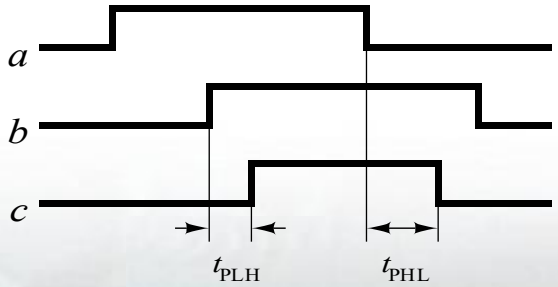
(a) Two-input AND gate



(b) Ideal (zero) delay



(c) $t_{PD} = t_{PLH} = t_{PHL}$

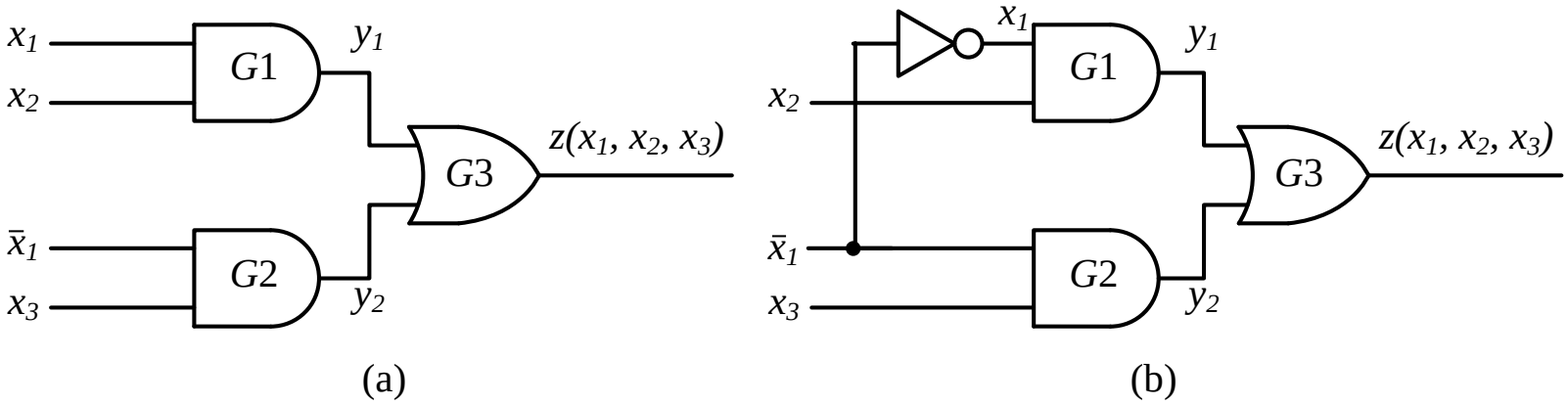


(d) $t_{PLH} < t_{PHL}$

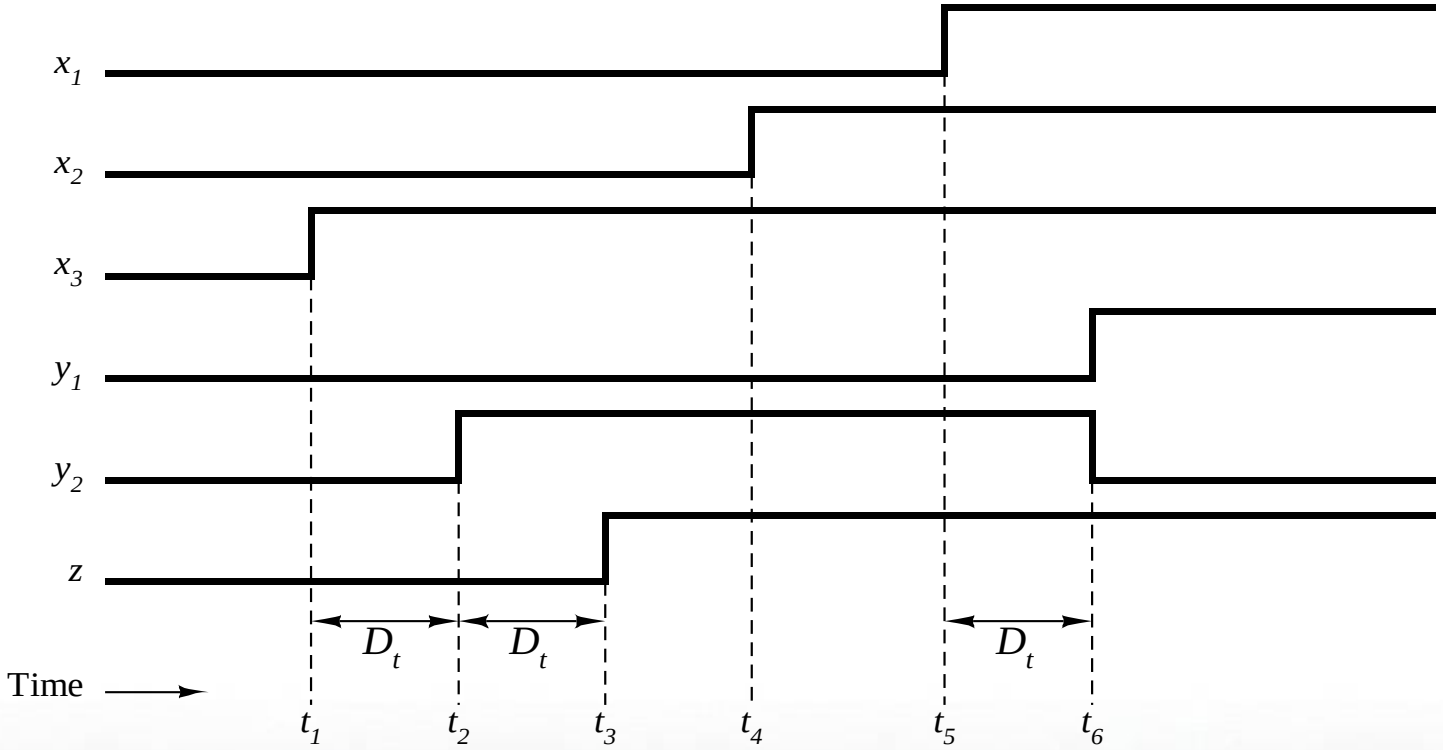
组合电路中的时序冒险

- 冒险(*Hazards*) 是由于组合电路中不相同的传输延时导致的不正确的输出。
- 静态冒险 *Static hazard* (glitch) – 输出相对正确值的瞬间变化。
  - 静态1冒险 – 输出从1变为0又变回到1
  - 静态0冒险 – 输出从0变为1又变回到0
- 动态冒险 *Dynamic hazard* (bounce) – 输出在状态变化时变化多次。
  - 动态0到1冒险 – 输出从0到1到0到1的变化
  - 动态1到0冒险 – 输出从1到0到1到0的变化

静态1冒险实例



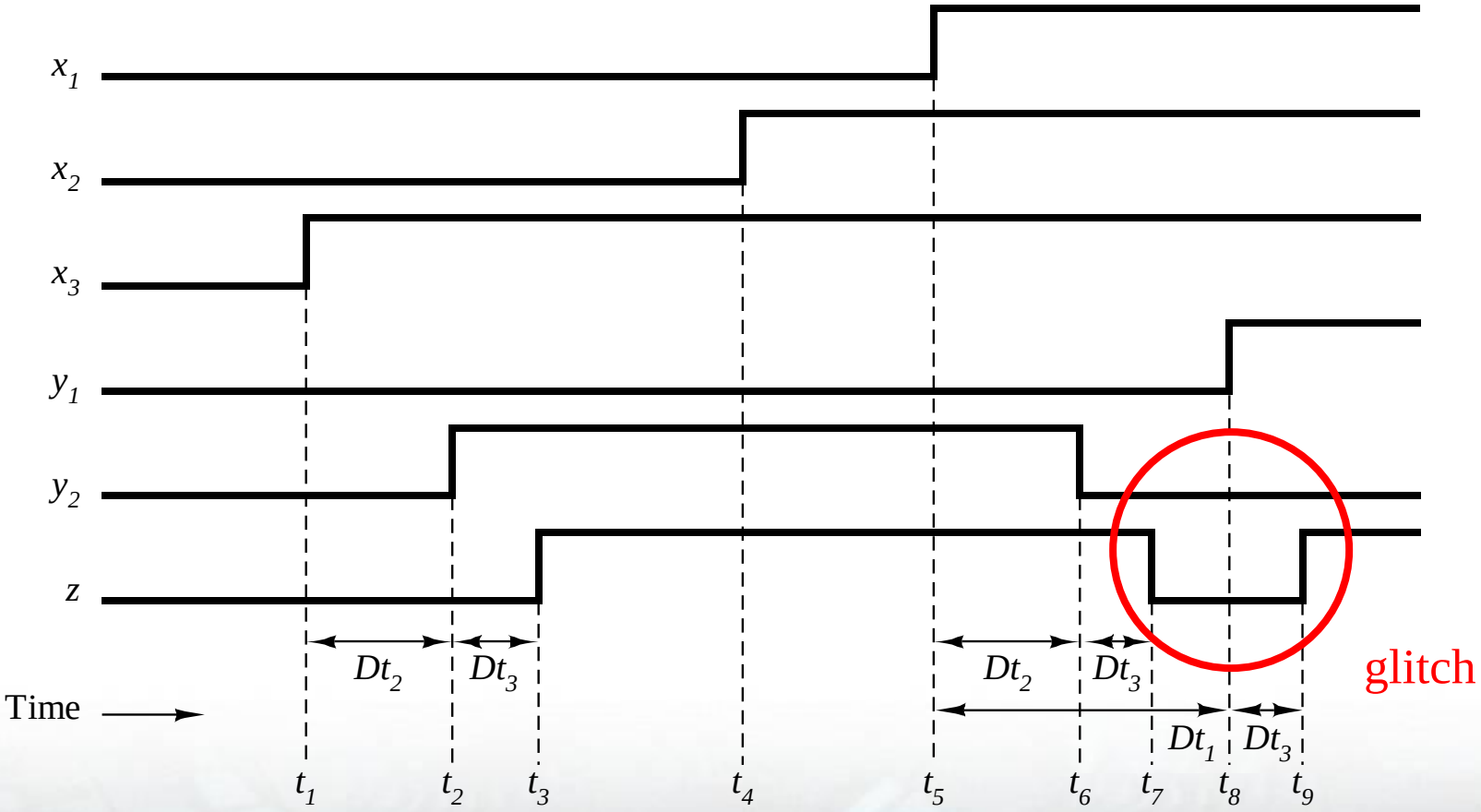
静态1冒险的实例（续）



(c)

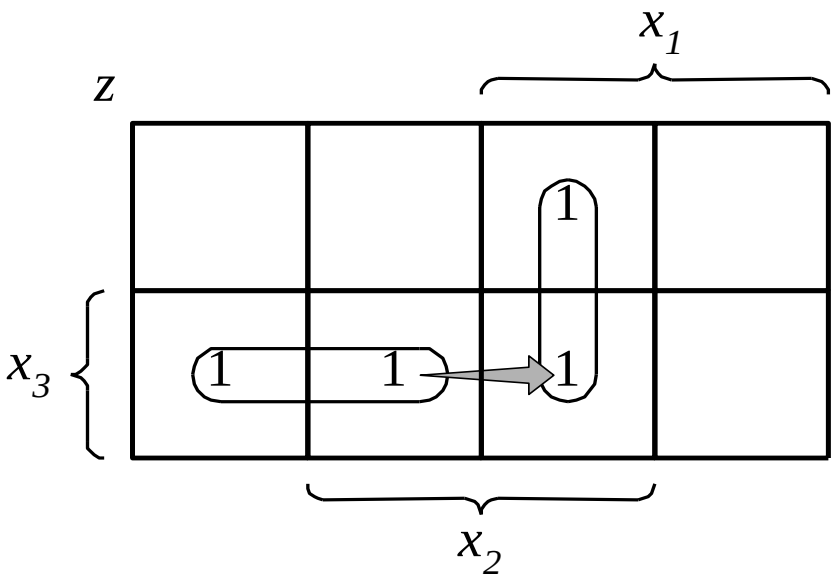
静态1冒险的实例（续）

$$Dt_1 > Dt_2 > Dt_3; Dt_1 = 2Dt_2; Dt_2 = 2Dt_3$$

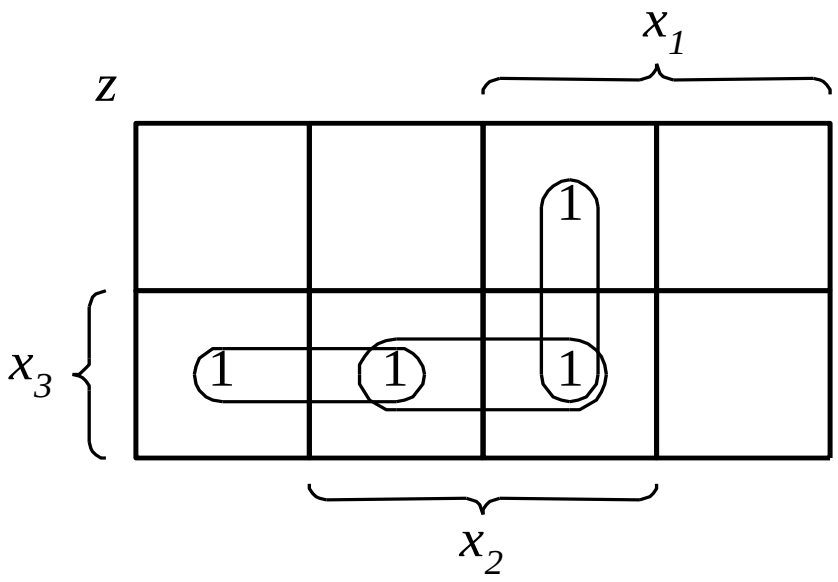


(d)

卡诺图中的冒险

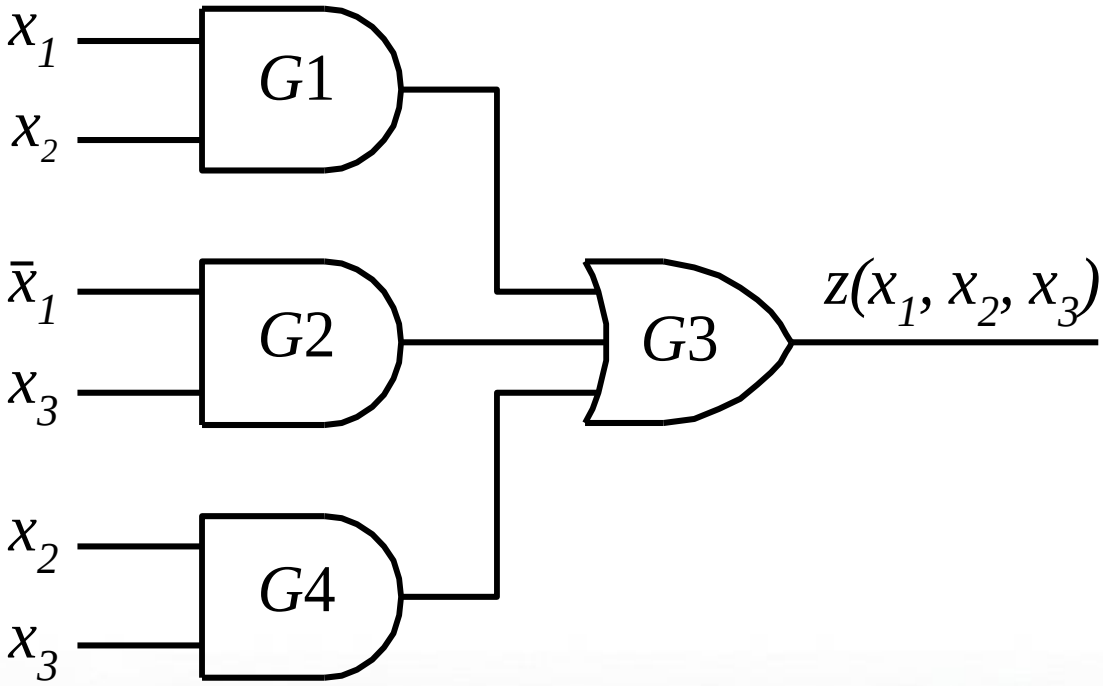


(a)

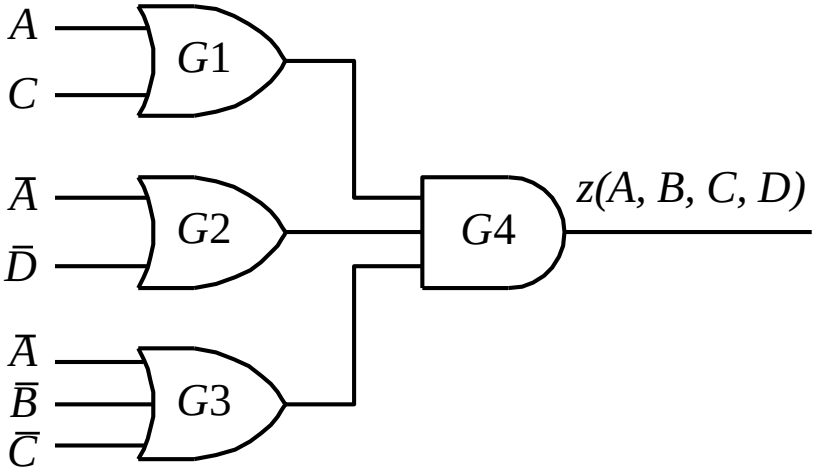


(b)

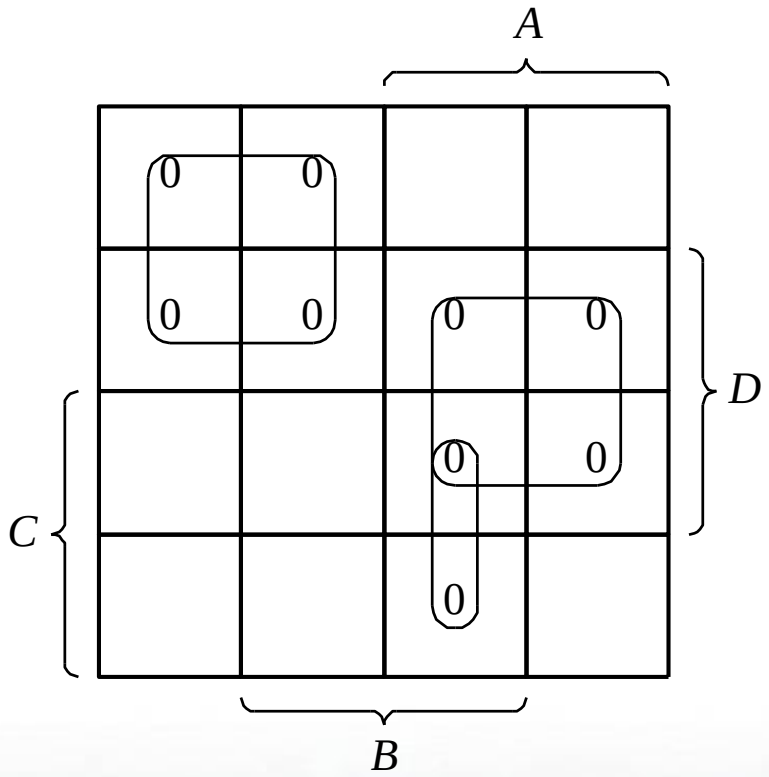
无冒险电路



静态0冒险的例子

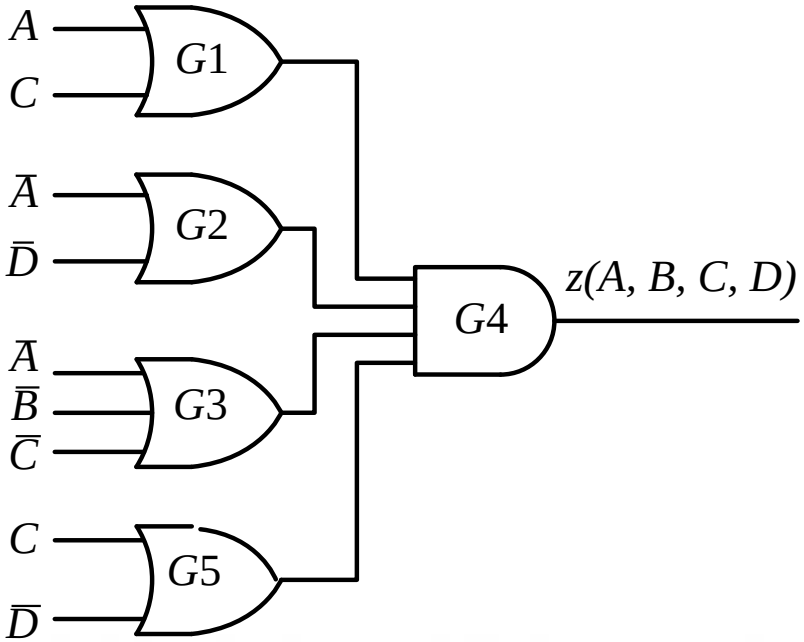


(a)

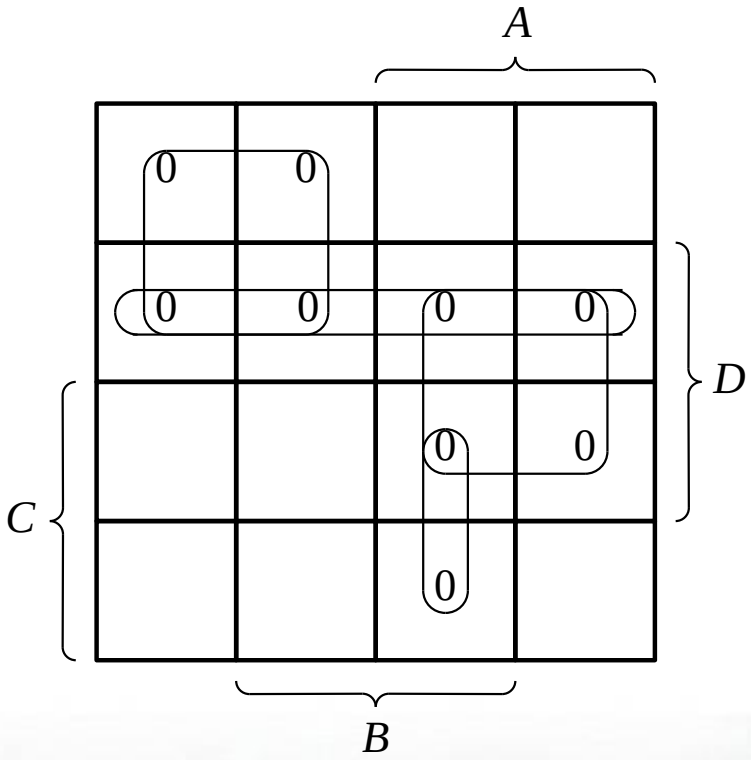


(b)

静态0冒险的例子(续).



(c)



(d)

静态冒险的解决

- 与或电路(SOP的实现)产生静态1冒险
- 或与电路(POS的实现)产生静态0冒险

- 一般的，可以通过添加相邻最小项（最大项）的公共积项（和项）的方法消除静态冒险。

函数最小化和电路化简

- 组合电路的分析与综合