

数字逻辑

第三章 组合逻辑电路分析与设计

北京理工大学

计算机学院

黄永刚

□ 一个组合逻辑电路

➤ m 个布尔输入

➤ n 个布尔输出

➤ n 个转换函数: 每一个转换函数将 2^m 个输入组合映射到一个输出, 输出依赖于当前的输入



m 个布尔输入

n 个布尔输出

一. 设计过程

- 1. 规范化
- 2. 形式化
- 3. 优化
- 4. 工艺映射
- 5. 验证

1. 规范化

- 指定组合电路行为

2. 形式化

- 用真值表对输入输出形式化

3. 优化

- 优化逻辑，减少门输入成本，如卡诺图优化

4. 工艺映射

- 将优化后逻辑映射到实现工艺

5. 验证

- 验证设计正确性

例子

□ 设计一个BCD码到余三码转换电路



1. 规范化

- 输入：0到9的BCD码表示，分别为0000-1001
- 输出：0到9的余三码表示，在BCD码上加0011

2. 形式化

□ 由于只有4位，用真值表很容易表示

□ 输入

BCD码

A,B,C,D

□ 输出

余三码

W,X,Y,Z

Input BCD A B C D	Output Excess-3 W X Y Z
0 0 0 0	0 0 1 1
0 0 0 1	0 1 0 0
0 0 1 0	0 1 0 1
0 0 1 1	0 1 1 0
0 1 0 0	0 1 1 1
0 1 0 1	1 0 0 0
0 1 1 0	1 0 0 1
0 1 1 1	1 0 1 0
1 0 0 0	1 0 1 1
1 0 0 1	1 1 0 0

3. 优化

□ 可直接从真值表观察标准形式

➤ 如何观察?

□ 但是:

➤ 不够优化

➤ 门输入成本高

□ 怎么办?

➤ 卡诺图优化

□ 优化目标

➤ 门输入成本G

➤ 即逻辑图中门的输入端的个数

Input BCD A B C D	Output Excess-3 WXYZ
0 0 0 0	0 0 1 1
0 0 0 1	0 1 0 0
0 0 1 0	0 1 0 1
0 0 1 1	0 1 1 0
0 1 0 0	0 1 1 1
0 1 0 1	1 0 0 0
0 1 1 0	1 0 0 1
0 1 1 1	1 0 1 0
1 0 0 0	1 0 1 1
1 0 0 1	1 1 0 0

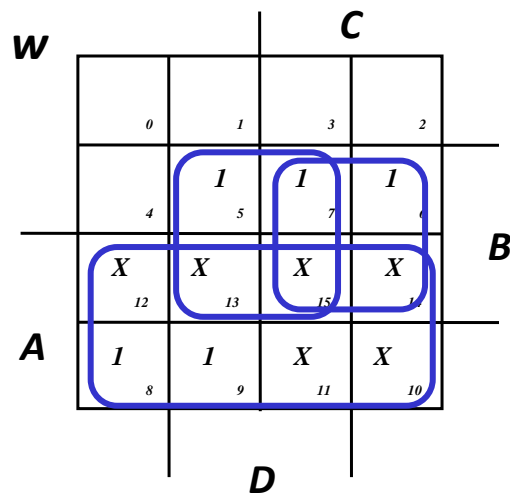
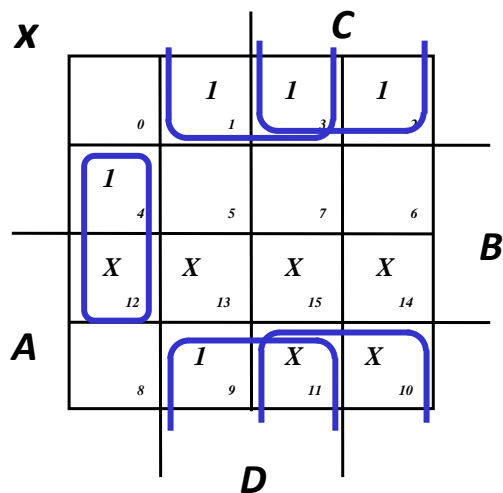
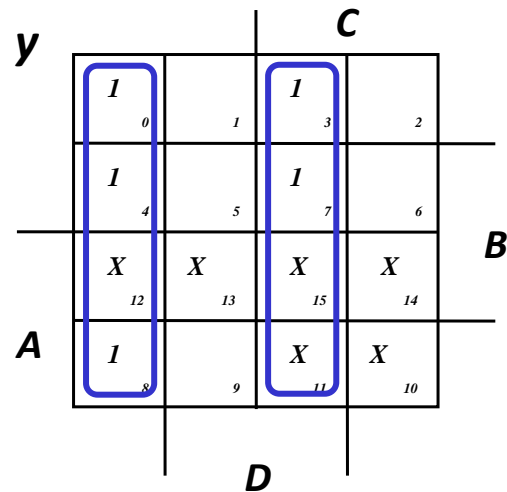
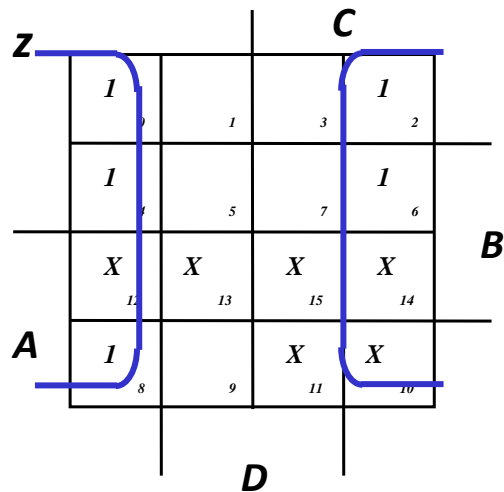
3. 优化

□ 如何由真值表得到卡诺图？

Input BCD A B C D				Output Excess-3 W X Y Z			
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0

		<i>C</i>				
<i>Z</i>		<i>1</i> <i>0</i>	<i>1</i>	<i>3</i>	<i>1</i> <i>2</i>	
		<i>1</i> <i>4</i>	<i>5</i>	<i>7</i>	<i>1</i> <i>6</i>	
<i>A</i>		<i>X</i> <i>12</i>	<i>X</i> <i>13</i>	<i>X</i> <i>15</i>	<i>X</i> <i>14</i>	<i>B</i>
		<i>1</i> <i>8</i>	<i>9</i>	<i>X</i> <i>11</i>	<i>X</i> <i>10</i>	
		<i>D</i>				

3. 优化



3. 优化

□ 系统化简

- ① 确定所有主蕴含项
- ② 对质主蕴含项进行逻辑和
- ③ 加上非质主蕴含项来覆盖未被包含的1
 - 选择规则
 - a. 尽可能减少主蕴含项重叠
 - b. 选择主蕴含项至少覆盖一个未被包含的1

3. 优化

□ 卡诺图优化结果

$$W = A + BC + BD$$

$$X = \bar{B}C + \bar{B}D + B\bar{C}\bar{D}$$

$$Y = CD + \bar{C}\bar{D}$$

$$Z = \bar{D}$$

□ 能不能进一步优化?

□ 优化目标是什么?

3. 优化

- 进一步**优化思想**：提取公因子，共享电路
- 有没有可以**共享**的？

$$W = A + BC + BD$$

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

$$Y = CD + \overline{C}\overline{D}$$

$$Z = \overline{D}$$

3. 优化

$$T_1 = C + D$$

$$W = A + BT_1$$

$$X = \overline{B}T_1 + B\overline{C}\overline{D}$$

$$Y = CD + \overline{C}\overline{D}$$

$$Z = \overline{D}$$

□ $G = 2 + 4 + 7 + 6 + 0 = 19$

□ 还能否进一步优化?

3. 优化

$$(\overline{C}\overline{D} = \overline{C + D} = \overline{T_1})$$

$$W = A + BT_1$$

$$X = \overline{B}T_1 + B\overline{T_1}$$

$$Y = CD + \overline{T_1}$$

$$Z = \overline{D}$$

□ $G = 2 + 4 + 6 + 4 + 0 = 16!$

4. 工艺映射

□ 优化的结果

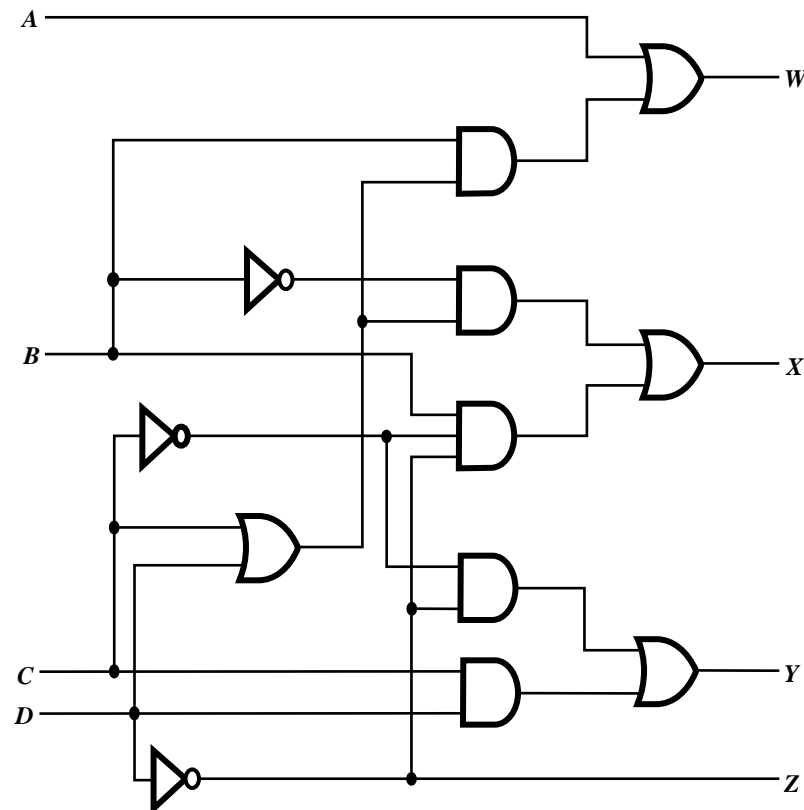
$$W = A + BT_1$$

$$X = \bar{B}T_1 + B\bar{T}_1$$

$$Y = CD + \bar{T}_1$$

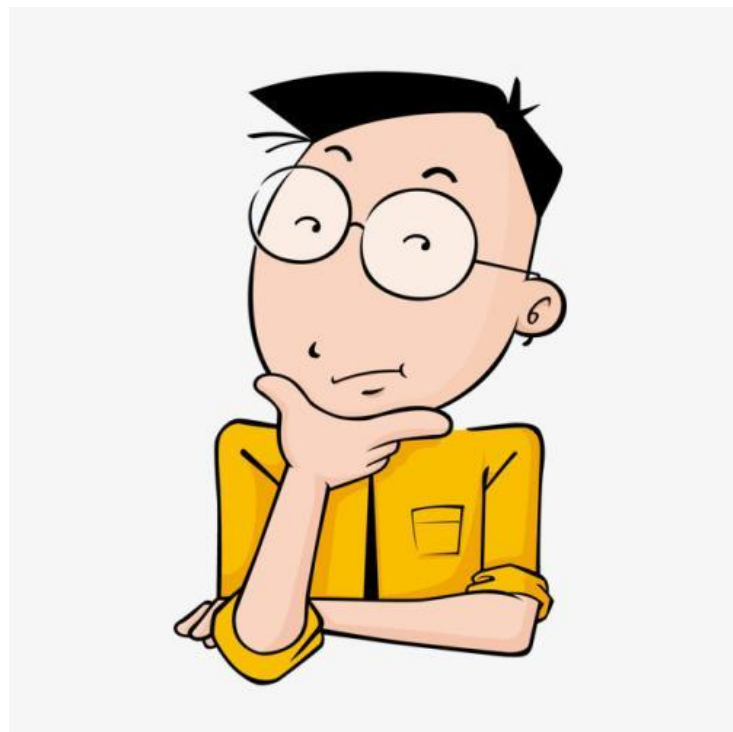
$$Z = \bar{D}$$

等价与



4. 工艺映射

能否用一种或两种门实现？



联结词完备集

$\{\neg, \vee\}$

$\{\neg, \wedge\}$

$\{\neg, \rightarrow\}$

$\{\uparrow\}$

$\{\downarrow\}$

4. 工艺映射

- 给定设计好的电路概要图
 - (含与门、或门、反相)
- → 到与非门
 - 库中含反相器和 n -输入与非门, $n = 2, 3, \dots$
- → 到或非门
 - 库中含反相器和 n -输入或非门, $n = 2, 3, \dots$

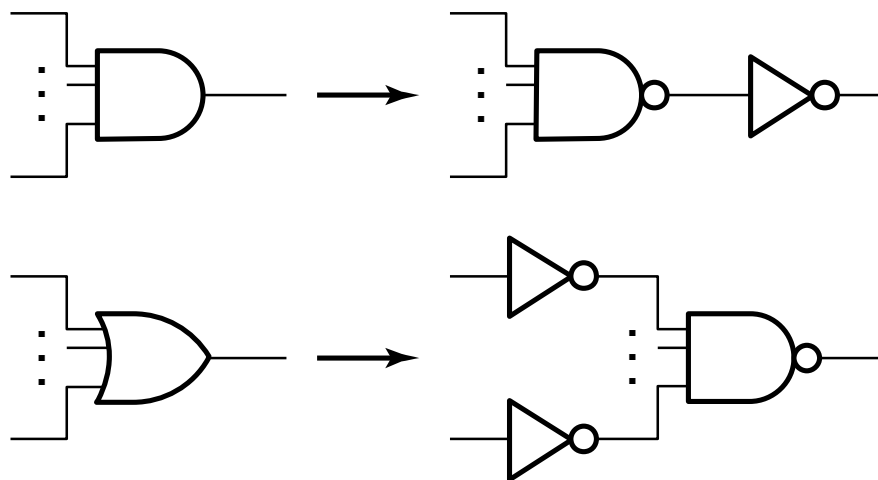
4. 工艺映射

□ 映射到与非门

- 库中含反相器和n-输入与非门, $n = 2, 3, \dots$

□ 映射过程

- ① 用与非门替换掉与门和或门



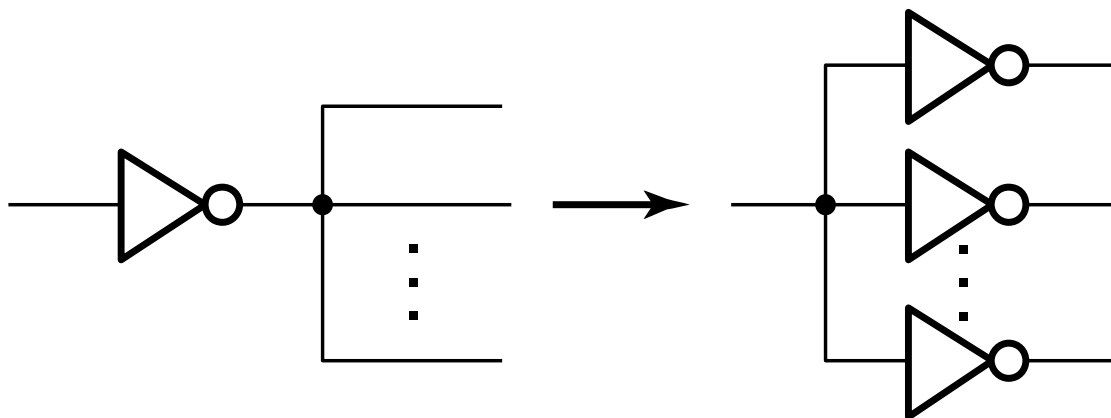
4. 工艺映射

□ 映射到与非门

- 库中含反相器和 n -输入与非门, $n = 2, 3, \dots$

□ 映射过程

- ② 将反相器推过电路中的扇出点



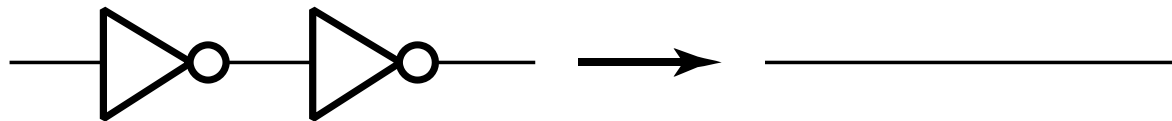
4. 工艺映射

□ 映射到与非门

- 库中含反相器和n-输入与非门, $n = 2, 3, \dots$

□ 映射过程

③ 抵消掉反相器对



4. 工艺映射

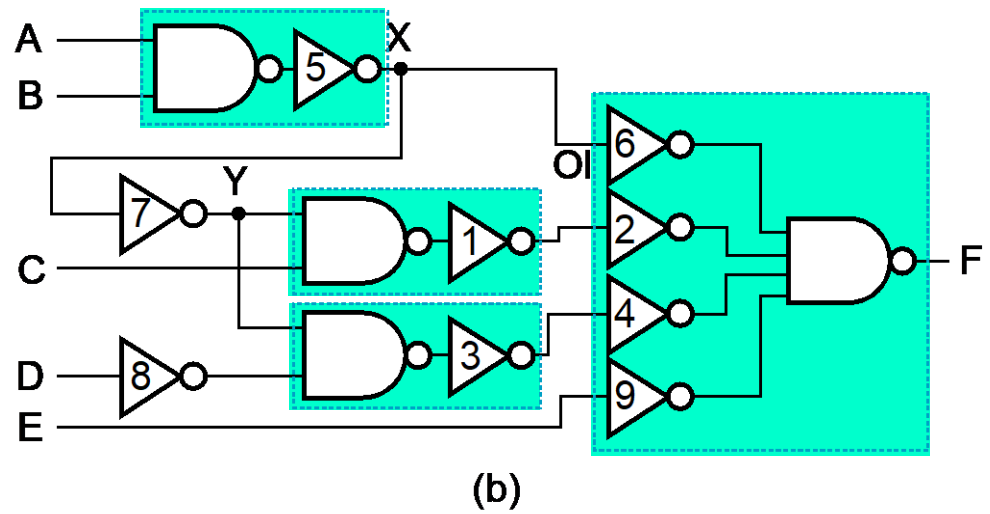
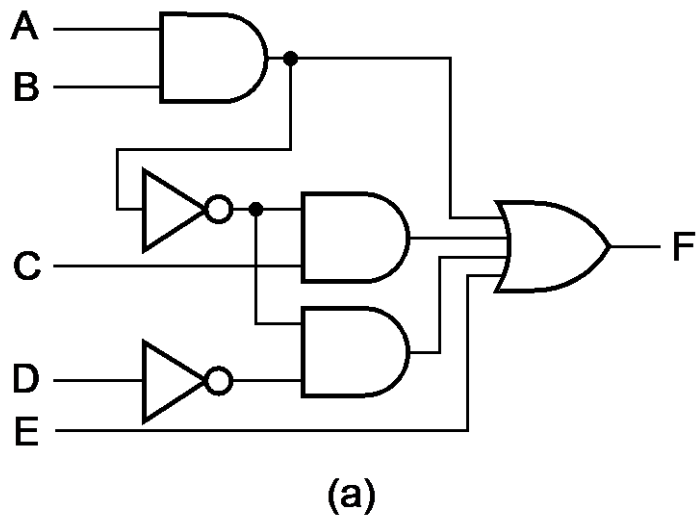
□ 映射到与非门

- 库中含反相器和 n -输入与非门, $n = 2, 3, \dots$

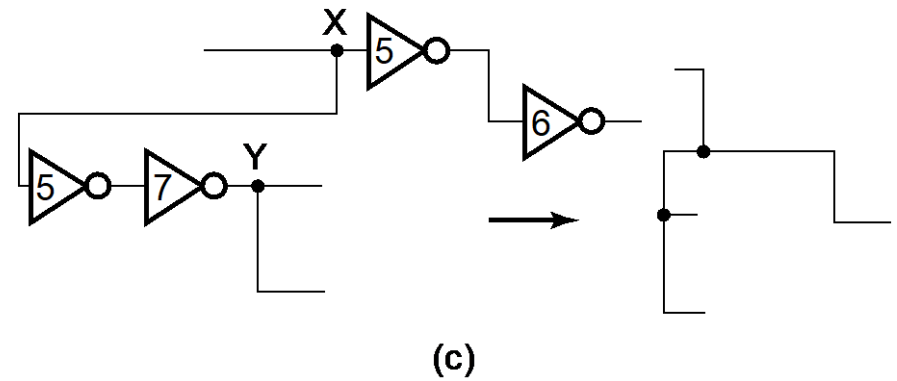
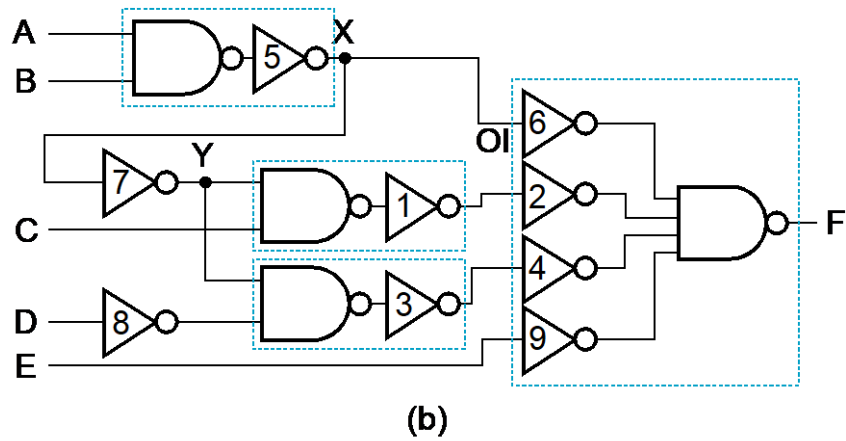
□ 映射过程

- ④ 重复②和③直到在 a 和 b 之间只存在1反相器
 - a. 电路输入或与非门的输出
 - b. 与非门输入
- 为什么要进行重复进行②和③?

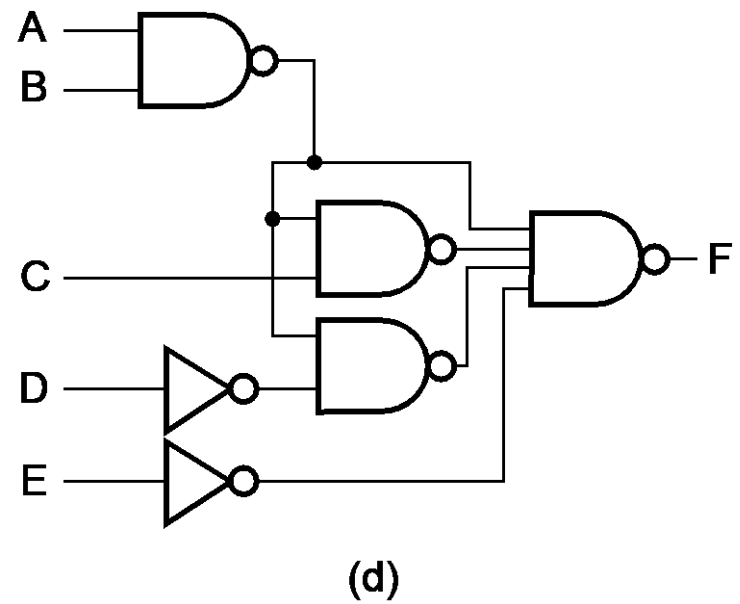
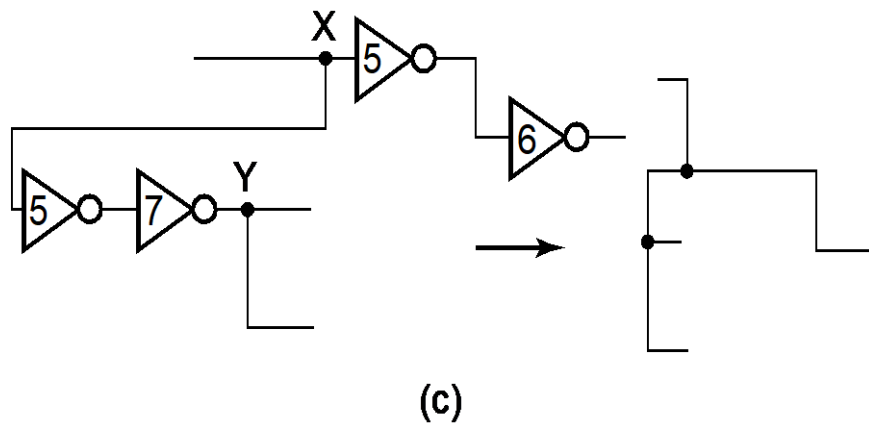
4. 工艺映射



4. 工艺映射



4. 工艺映射



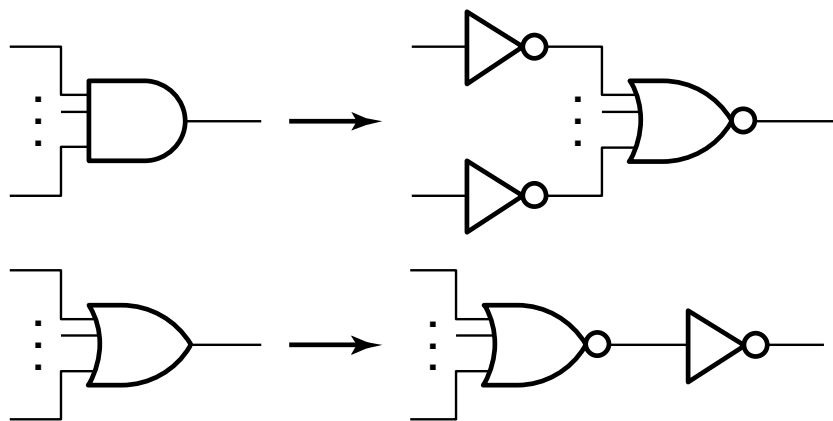
4. 工艺映射

□ 映射到或非门

➤ 库中含反相器和n-输入或非门, $n = 2, 3, \dots$

□ 映射过程

① 用或非门替换掉与门和或门



4. 工艺映射

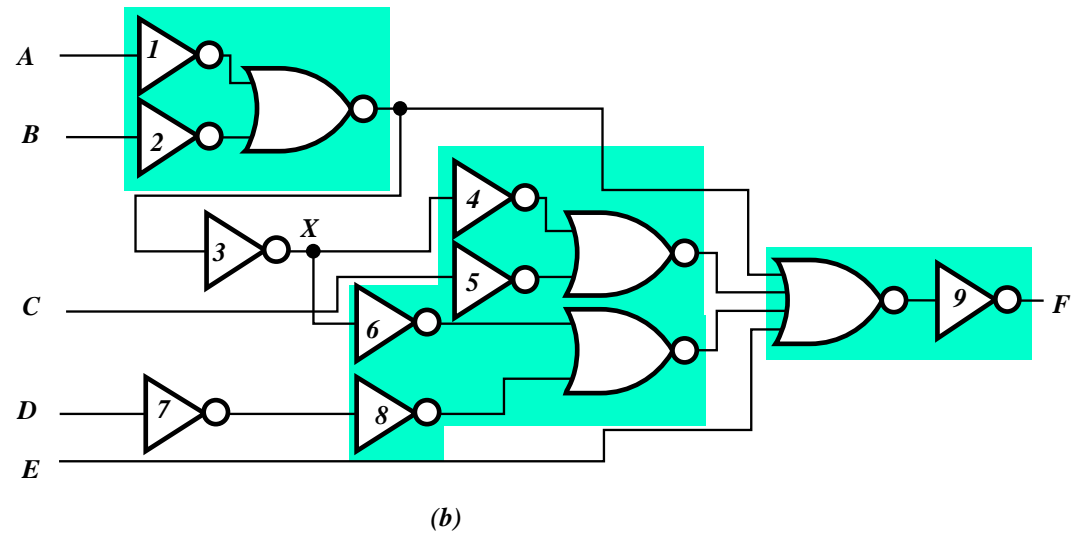
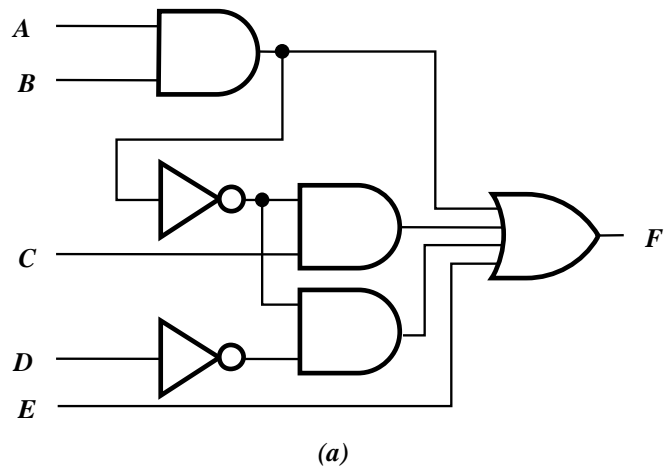
□ 映射到或非门

- 库中含反相器和 n -输入或非门, $n = 2, 3, \dots$

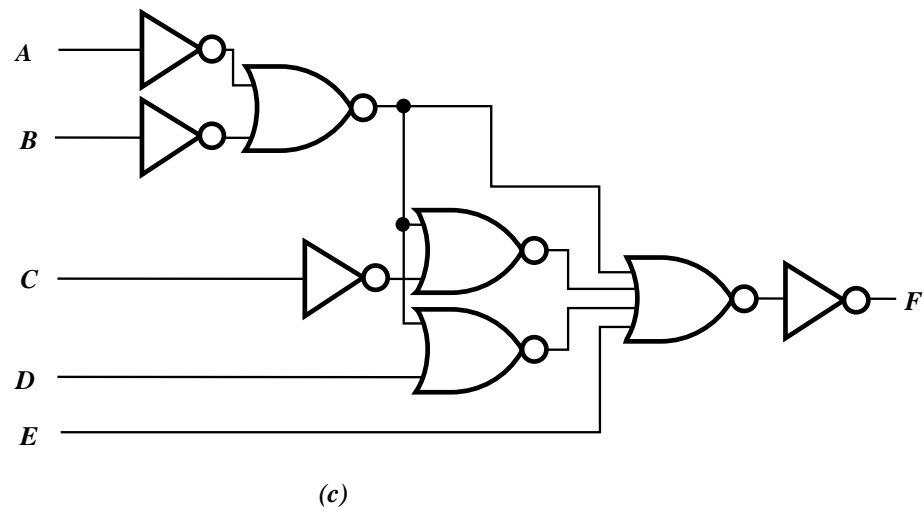
□ 映射过程

- ② 将反相器推过电路中的扇出点
- ③ 抵消掉反相器对
- ④ 重复②和③直到在 a 和 b 之间只存在1个反相器
 - a. 电路输入或或非门的输出
 - b. 或非门输入

4. 工艺映射



4. 工艺映射



5. 验证

- 人工逻辑分析
- 模拟

5. 验证

■ 人工逻辑分析

- 找出实现电路真值表或方程式
- 将实现电路真值表和规范真值表比较，或
- 证明实现电路的方程式和规范方程式等价

$$T_1 = \overline{\overline{C + D}} = C + D$$

$$W = \overline{A} (\overline{T_1 B}) = A + B T_1$$

$$X = (T_1 B) (B \overline{C} \overline{D}) = \overline{B} T_1 + B \overline{C} \overline{D}$$

$$Y = \overline{C \overline{D}} + \overline{C} D = CD + \overline{C} \overline{D}$$

Input BCD A B C D	Output Excess -3 WXYZ
0 0 0 0	0 0 1 1
0 0 0 1	0 1 0 0
0 0 1 0	0 1 0 1
0 0 1 1	0 1 1 0
0 1 0 0	0 1 1 1
0 1 0 1	1 0 0 0
0 1 1 0	1 0 0 1
0 1 1 1	1 0 1 0
1 0 0 0	1 0 1 1
1 0 0 1	1 0 1 1

5. 验证

□ 模拟

➤ 以Verilog为例：Verilog是一种硬件描述语言

① 使用Verilog 对电路进行编程实现

② 编写测试程序，即TestBench

➤ 产生模拟信号

➤ 将产生信号加到实现电路上

③ 将输出与期望值比较

5. 验证

□ 例子：一个加法器

① 使用Verilog 对电路进行编程实现

```
1  module add(a,b,c,d,e); // 模块接口
2  input [5:0] a; // 输入信号a
3  input [5:0] b; // 输入信号b
4  input [5:0] c; // 输入信号a
5  input [5:0] d; // 输入信号b
6
7  output[7:0] e; // 求和输出信号
```

5. 验证

□ 例子：一个加法器

② 编写TestBench: 产生模拟信号

```
initial
begin    // initial 是仿真用的初始化关键词
    a=0 ; // 必须初始化输入信号
    b=0 ;
    c=0 ;
    d=0
    for(i=1;i<31;i=i+1)
        begin
            #10 ; a = i; b = i; c = i; d = i;
        end
    end
end
```

5. 验证

□ 例子：一个加法器






② 编写TestBench: 将产生信号加到实现电路上

```
// 调用被仿真模块模块
add uut (
    .a(a),
    .b(b),
    .c(c),
    .d(d),
    .e(e));
```

5. 验证

□ 例子：一个加法器

③ 将输出与期望值比较

Name	Value	0 ns					20 ns					40 ns				
 a[5:0]	4	0	1	2	3	4	0	1	2	3	4	0	1	2	3	4
 b[5:0]	4	0	1	2	3	4	0	1	2	3	4	0	1	2	3	4
 c[5:0]	4	0	1	2	3	4	0	1	2	3	4	0	1	2	3	4
 d[5:0]	4	0	1	2	3	4	0	1	2	3	4	0	1	2	3	4
 e[7:0]	16	0	4	8	12	16	0	4	8	12	16	0	4	8	12	16

二. 组合逻辑功能模块

- 1. 组合功能模块
- 2. 基本逻辑函数
- 3. 译码和译码器
- 4. 基于译码器的组合电路
- 5. 编码和编码器
- 6. 选择和复用器
- 7. 基于复用器的组合电路

1. 组合功能模块

□ 组合功能模块

- 在电路设计中经常使用的公共模块
- 每个功能模块对应一个组合电路实现

□ 按集成度高低的不同

- 小规模集成电路 (SSI): 10-100 个晶体管
- 中规模集成电路 (MSI): 100-1000
- 大规模集成电路 (LSI): 1000-100000
- 超大规模集成电路 (VLSI): 100000以上

1. 组合功能模块

□ 芯片(集成电路)工艺 (nm)

- 晶体管栅极的宽度, 也称栅长
- 栅长越短
 - 同尺寸的硅片可集成更多晶体管
 - 频率越高, 功耗更低

□ 目前水平: 7nm, 迈向5nm

2. 基本逻辑函数

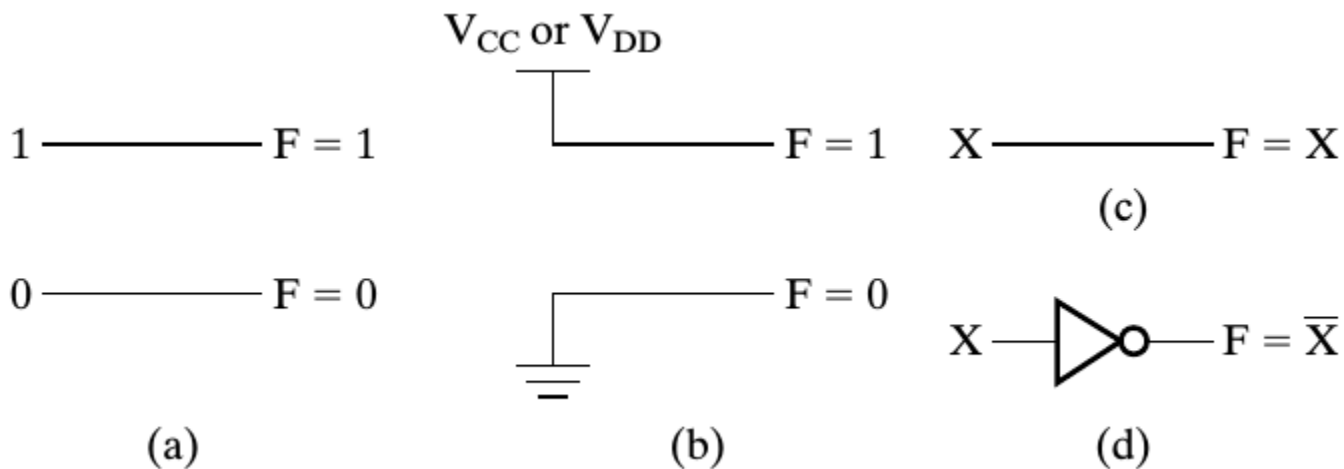
- 单变量函数
- 多位函数
- 使能函数

2. 基本逻辑函数

□ 单变量函数

- 一个变量 X 的函数
- 可以在输入处用作功能块

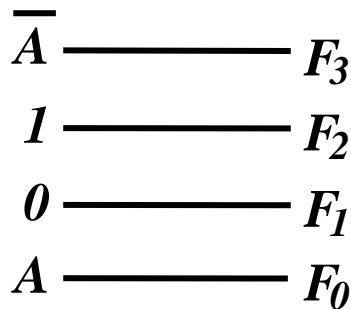
X	$F = 0$	$F = X$	$F = \bar{X}$	$F = 1$
0	0	0	1	1
1	0	1	0	1



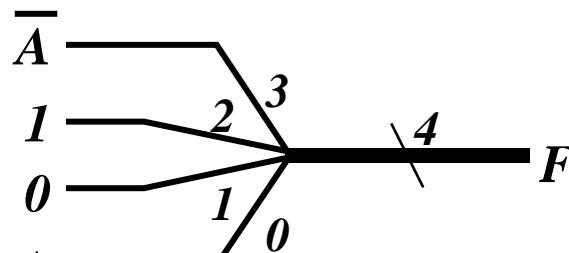
2. 基本逻辑函数

□ 多位函数

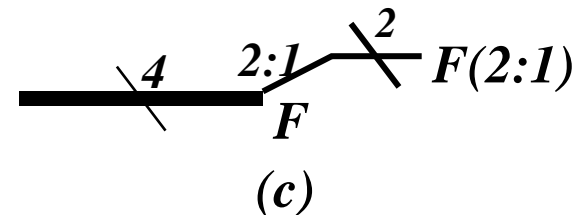
- 1位函数的向量
- 粗线代表总线，其是一个向量信号, 如图(b)
- 可以从总线中分割出一个位子集, 如 (c, d)



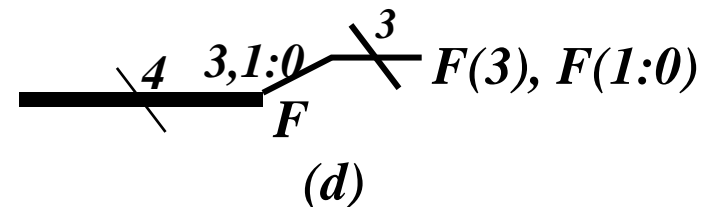
(a)



(b)



(c)



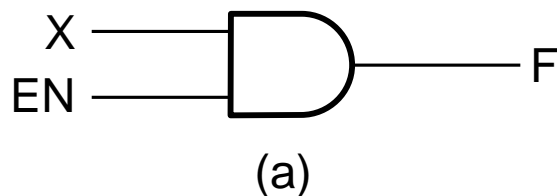
(d)

2. 基本逻辑函数

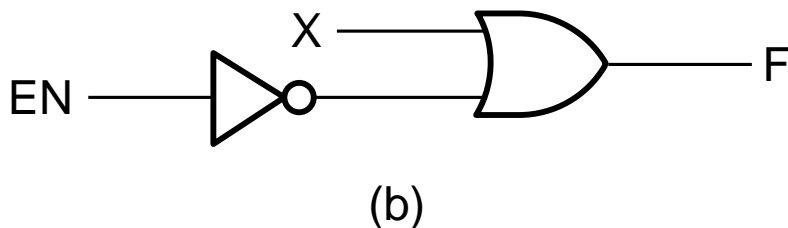
□ 使能函数

- 是否允许信号从输入传到输出
- 引入使能信号EN
 - $EN=1$ 允许信号传输
 - $EN=0$ 阻止信号传输
 - 输出用固定值替代，可能是0或者1

□ 固定值是 0



□ 固定值是 1



3. 译码和译码器

□ 译码

- 输入 n 位, 输出 m ($n \leq m \leq 2^n$) 位
- 例子: 输入二进制码, 在输出中将对应位置1
 - $010 \rightarrow 00000100$

□ 编码

- 输入最大 m ($n \leq m \leq 2^n$) 位, 输出 n 位
- 例子: 输入中某位为1, 输出中编码出位置
 - $00000100 \rightarrow 010$

□ 译码和编码互逆

3. 译码和译码器

□ 译码

- 输入 n 位, 输出 m ($n \leq m \leq 2^n$) 位
- 例子: 输入二进制码, 在输出中将对应位置1
 - 010 → 00000100

□ 译码器

- 实现译码功能的电路
- n - m 译码器
- 例子:
 - 输入: 1的位置的编码, 如010
 - 输出: 只有1位是1的输出, 如 00000100

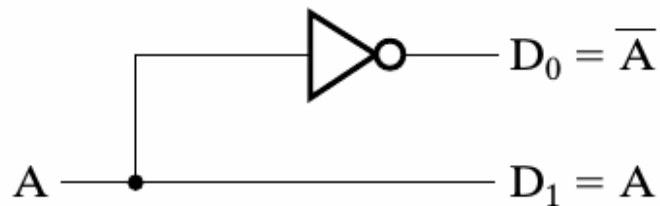
3. 译码和译码器

□ 如何设计一个1-2 译码器？

A	D₀	D₁
0	1	0
1	0	1

$$D_0 = \overline{A}$$

$$D_1 = A$$



3. 译码和译码器

□ 如何设计一个2-4译码器？

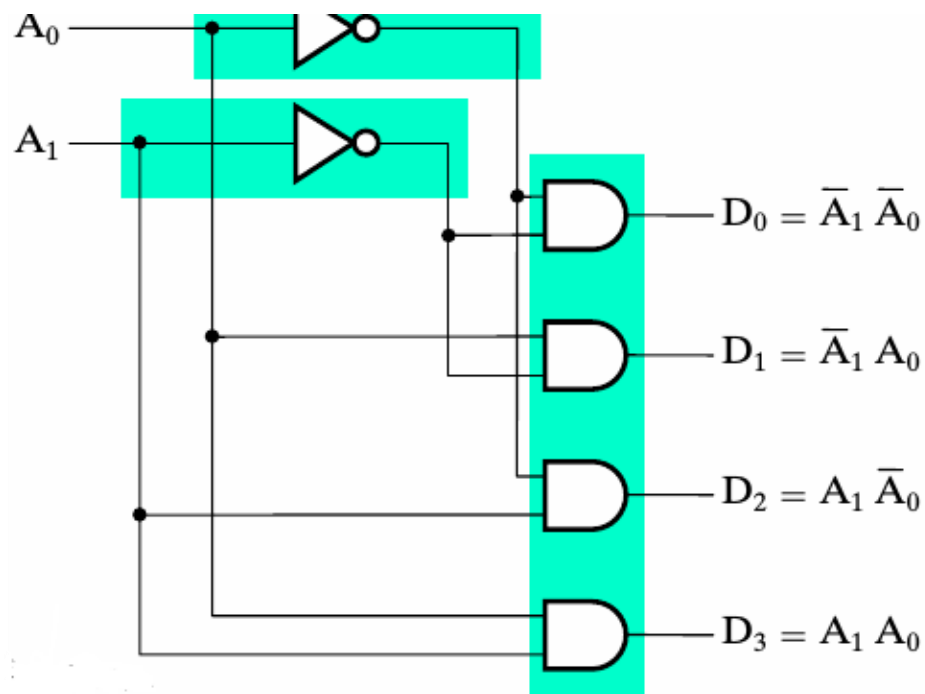
A_1	A_0	D_0	D_1	D_2	D_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

$$D_0 = \bar{A}_1 \bar{A}_0$$

$$D_1 = \bar{A}_1 A_0$$

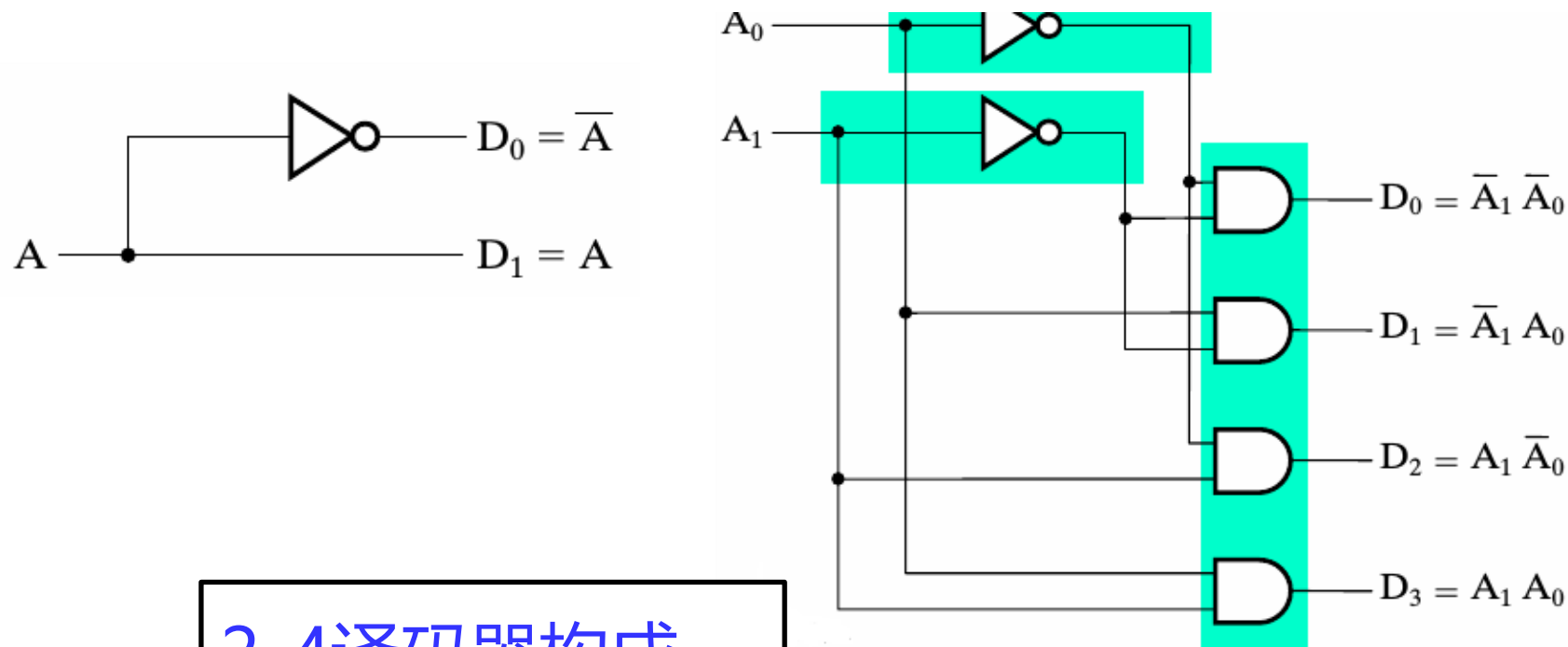
$$D_2 = A_1 \bar{A}_0$$

$$D_3 = A_1 A_0$$



3. 译码和译码器

□ 1-2译码器和2-4译码器有什么联系？



2-4译码器构成

- 2个1-2译码器
- 4个与门

3. 译码和译码器

□ $n-2^n$ 译码器展开

- 需要 2^n 个与门
- 每个输出与门被两个译码器驱动
 - 这两个译码器输入相等或相差1
- 将这两个译码器按照同样过程展开
- 直至到1-2译码器

□ 上述过程可经修改应用到输出 $\neq 2^n$ 的译码器

3. 译码和译码器

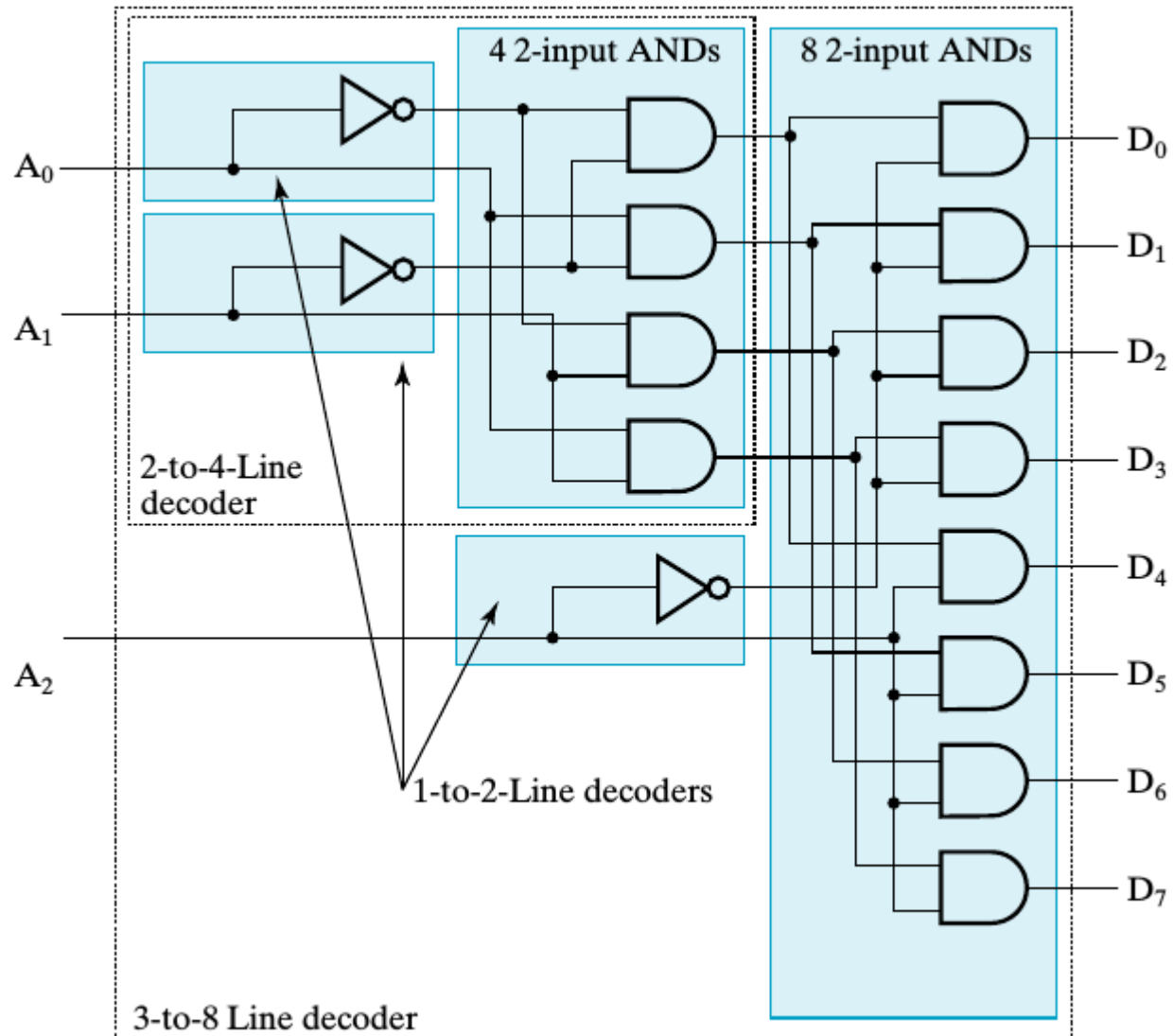
□ 例子: 3-8 译码器

- 需要8个输出与门
- 每个输出与门被两个译码器驱动
 - 最相近的两个译码器
 - 2-4 译码器
 - 1-2 译码器

□ 2-4译码器

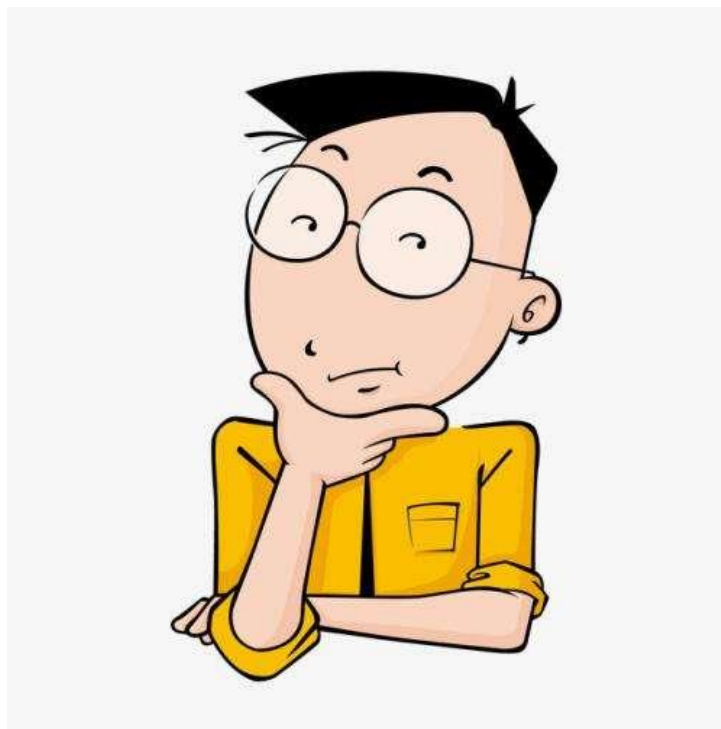
- 需要4个输出与门
- 每个输出与门被两个译码器驱动
 - 最相近的两个译码器
 - 2个 1-2 译码器

3. 译码和译码器



3. 译码和译码器

□ 如何构建一个7-128 译码器？



3. 译码和译码器

□ 7-128 译码器

- 需要128个输出与门
- 每个输出与门被两个译码器驱动
 - 最相近的两个译码器
 - 4-16 译码器
 - 3-8 译码器

□ 4-16译码器

- 需要16个输出与门
- 每个输出与门被两个译码器驱动
 - 最相近的两个译码器
 - 2个 2-4 译码器

3. 译码和译码器

□ 带有使能的译码器

- 电路输出增加使能信号-EN

□ 真值表

- 注意X可以表示0和1

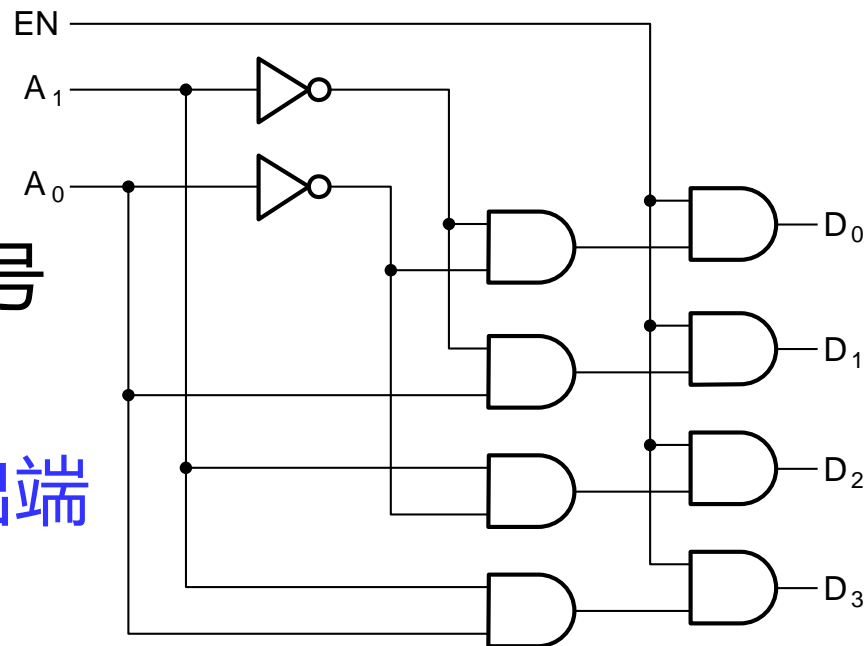
□ 也被称为1-4 多路分配器

- EN为输入数据
- A_1A_0 为输出端选择信号
 - 可以看作使能信号
- 将数据输出到选择输出端

□ 同一电路，两个视角

EN	A_1	A_0	D_0	D_1	D_2	D_3
0	X	X	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1

(a)



(b)

4. 基于译码器的组合电路

□ 实现1个函数，其中有 n 个变量

- 最小项之和的表达式，即标准型
 - 一个 $n-2^n$ 译码器，译码器输出对应最小项
 - 1个或门，将最小项或起来

□ 方法1:

- 得到函数的真值表
- 如果1在真值表中，就连接译码器输出和或门

□ 方法2:

- 得到输出函数的最小项
- 将最小项用或门连接起来

4. 基于译码器的组合电路

□ 例子

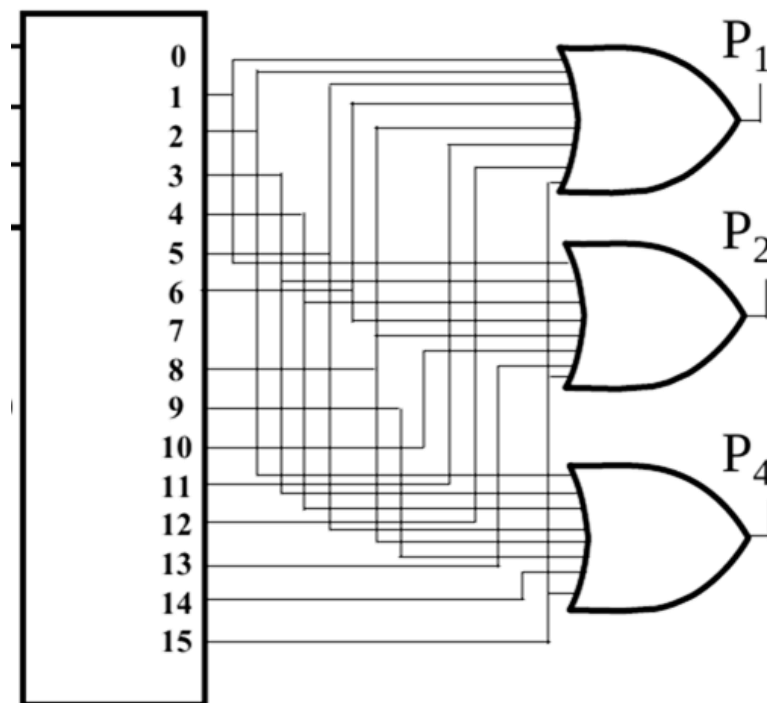
$$P_1 = \Sigma_m(1,2,5,6,8,11,12,15)$$

$$P_2 = \Sigma_m(1,3,4,6,8,10,13,15)$$

$$P_4 = \Sigma_m(2,3,4,5,8,9,14,15)$$

□ 有没有问题？

□ 门输入成本太高



5. 编码和编码器

译码

- 输入 n 位, 输出 m ($n \leq m \leq 2^n$) 位
- 例子: 输入二进制码, 在输出中将对应位置1
 - $010 \rightarrow 00000100$

编码

- 输入最大 m ($n \leq m \leq 2^n$) 位, 输出 n 位
- 例子: 输入中某位为1, 输出中编码出位置
 - $00000100 \rightarrow 010$

译码和编码互逆

5. 编码和编码器

□ 编码

- 输入最大 m ($n \leq m \leq 2^n$) 位, 输出 n 位
- 例子: 输入中某位为1, 输出中编码出相应位
 - 00000100 → 010

□ 编码器

- 实现编码功能的电路
- m - n 编码器
- 例子:
 - 输入: 只有1位是1的输入, 如 00000100
 - 输出: 1的位置的编码, 如010

5. 编码和编码器

□ 例子：十进制-BCD编码器

- 输入: 10位代表从0到9, (D_0, \dots, D_9)
- 输出: 4位BCD码
- 函数: 若 D_i 是1, 则输出(A_3, A_2, A_1, A_0)是iBCD码

□ 如何得到电路?

- 真值表→卡诺图优化→优化
- 但是10个输入?

5. 编码和编码器

□ 思考: A_j 什么时候是1?

➤ i 的二进制中 A_j 位是1

➤ 输入 D_i 是布尔方程 A_j 的一项

□ 布尔方程

➤ $A_3 = D_8 + D_9$

➤ $A_2 = D_4 + D_5 + D_6 + D_7$

➤ $A_1 = D_2 + D_3 + D_6 + D_7$

➤ $A_0 = D_1 + D_3 + D_5 + D_7 + D_9$

5. 编码和编码器

- 如果输入中不止一位为1
 - 如0100010000
 - 则编码器输出：多个位编码的或
 - 不能正常工作
- 怎么办？
- 思想：选择最重要的输入位编码
 - 能接受所有的输入组合
 - 能产生有意义的输出
 - 优先编码器

5. 编码和编码器

□ 例子：5输入优先编码器

➤ 输入：(D_4, D_3, D_2, D_1, D_0)

➤ 输出： A_2, A_1, A_0 和 V ， V 表示是否有1出现

No. of Min-terms/Row	Inputs					Outputs			
	D4	D3	D2	D1	D0	A2	A1	A0	V
1	0	0	0	0	0	X	X	X	0
1	0	0	0	0	1	0	0	0	1
2	0	0	0	1	X	0	0	1	1
4	0	0	1	X	X	0	1	0	1
8	0	1	X	X	X	0	1	1	1
16	1	X	X	X	X	1	0	0	1

□ X表示0或1，表条目对应乘积项而不是最小项

5. 编码和编码器

□ 如何得到电路？

➤ 真值表→卡诺图优化→优化

➤ 但是5个输入？

□ 可以直接从表中读出方程，并进行优化

□ 观察法

5. 编码和编码器

No. of Min-terms/Row	Inputs					Outputs			
	D4	D3	D2	D1	D0	A2	A1	A0	V
1	0	0	0	0	0	X	X	X	0
1	0	0	0	0	1	0	0	0	1
2	0	0	0	1	X	0	0	1	1
4	0	0	1	X	X	0	1	0	1
8	0	1	X	X	X	0	1	1	1
16	1	X	X	X	X	1	0	0	1

$$A_2 = D_4$$

$$A_1 = \overline{D}_4 D_3 + \overline{D}_4 \overline{D}_3 D_2 = \overline{D}_4 F_1, F_1 = (D_3 + D_2)$$

$$A_0 = \overline{D}_4 D_3 + \overline{D}_4 \overline{D}_3 \overline{D}_2 D_1 = \overline{D}_4 (D_3 + \overline{D}_2 D_1)$$

$$V = D_4 + F_1 + D_1 + D_0$$

6. 多路复用器

□ 选择

- 计算机系统的**关键功能模块**

□ 执行选择的电路

➤ 输入

- 一组待选择的数据
- 一组用来进行选择的选择信号

➤ 一个输出

□ 执行选择的逻辑电路被称为**多路复用器**

6. 多路复用器

□ **多路复用器**：从输入选择信息并输出

➤ 输入

➤ 待选择数据：最多 2^n 个输入 ($I_{2^n - 1}, \dots, I_0$)

➤ 选择信号： n 个, (S_{n-1}, \dots, S_0)

➤ 1个输出： Y

6. 多路复用器

□ 例子：2-1多路复用器

□ 输入

□ 待选择数据： 2^1 个， I_0, I_1

□ 选择信号： $n = 1$ 个， S_0

□ $S_0 = 0$ 选择输入 I_0

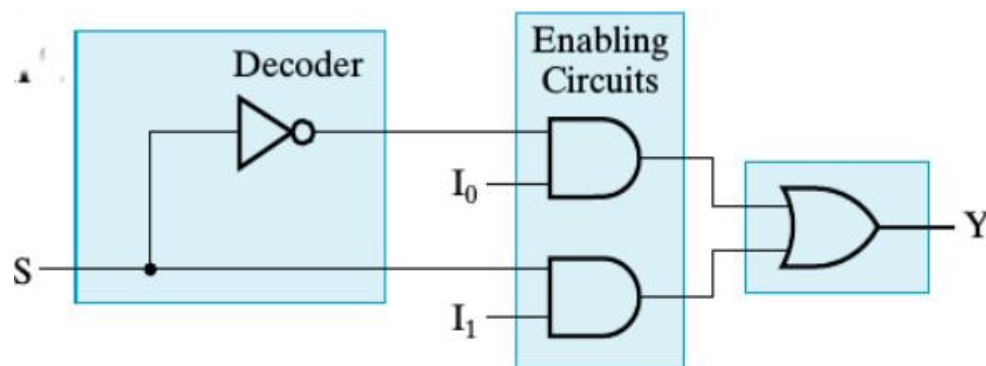
□ $S_0 = 1$ 选择 I_1

□ 输出： Y

□ 方程

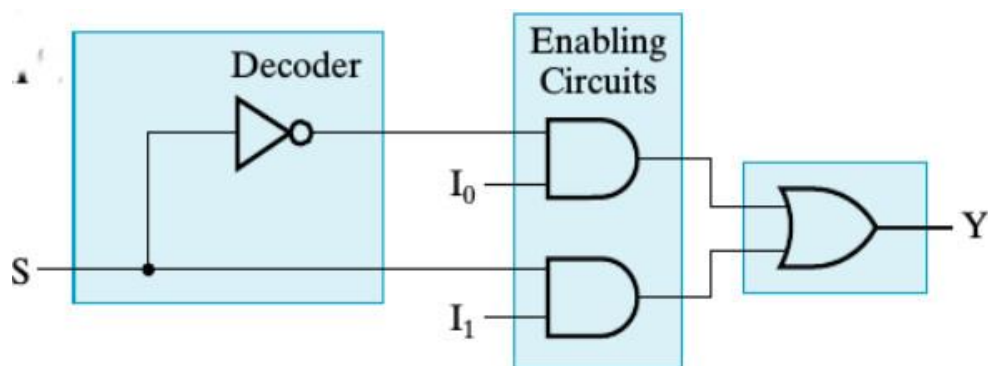
$$Y = \bar{S}I_0 + SI_1$$

□ 电路：



6. 多路复用器

□ 例子：2-1多路复用器



□ 1-2 译码器

□ 2个 使能 (2输入与门)

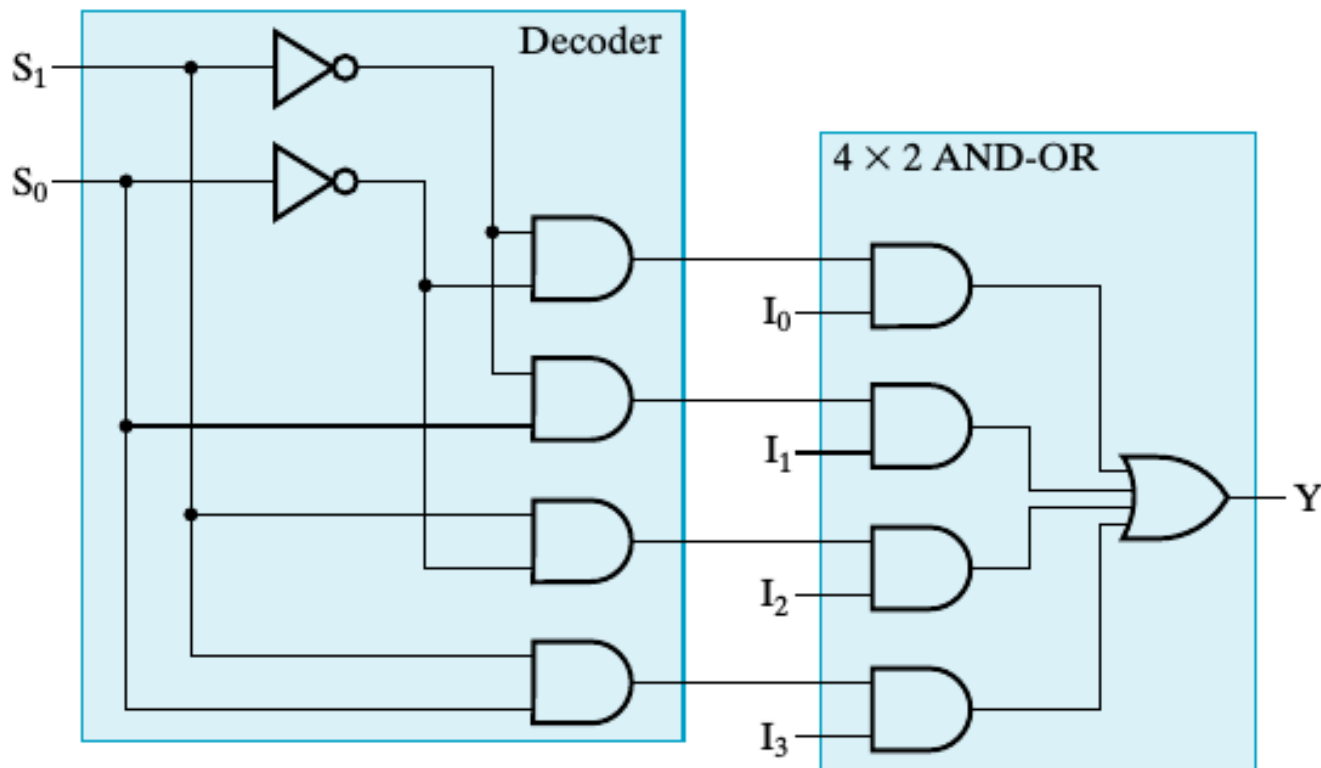
□ 2-输入或门

6. 多路复用器

- $2^n - 1$ 多路复用器
 - $n - 2^n$ 译码器
 - 2^n 使能 (2输入与门)
 - 2^n 输入或门
- 后面两个看作 $2^n \times 2$ 与或门
 - 2 表示与门输入数量
 - 2^n 表示与门的数量
 - 1个 2^n 输入的或门

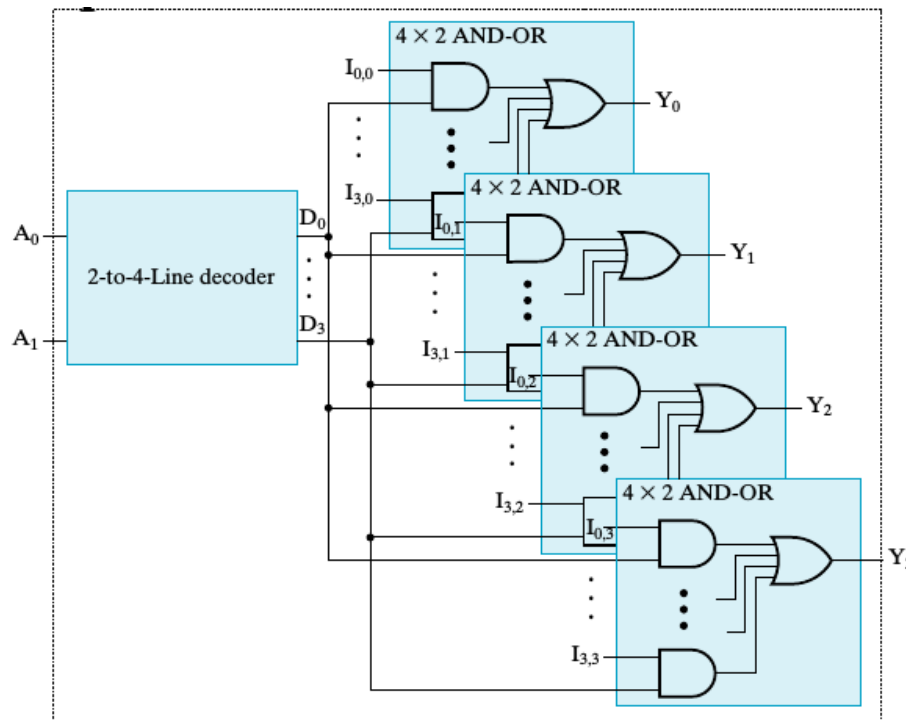
6. 多路复用器

- 2^2-1 多路复用器
 - $2-2^2$ 译码器
 - $2^2 \times 2$ 与-或门



6. 多路复用器

- 位宽展开：4-1 四位多路复用器
 - 选择位向量而不是单个位
 - 同一个译码器
 - 平行的使用四个 $2^n \times 2$ 与-或



7. 基于复用器的组合电路

- 实现m个函数，包含n个变量
- 方法1： m位宽 2^n-1 多路复用器
- 方法2： m位宽 $2^{n-1}-1$ 多路复用器

7. 基于复用器的组合电路

- 方法1: m 位宽 2^n-1 多路复用器
 - 得到函数的真值表
 - 根据真值表
 - 将函数输入 S_{n-1}, \dots, S_0 作为选择信号
 - 真值表中的值作为多路复用器的待选择数据
 - 将多路复用器的输出标识成函数输出

7. 基于复用器的组合电路

□ 例子：格雷码到二进制码转换

□ 真值表如图所示

□ $x = C$

□ y 和 z 比较复杂

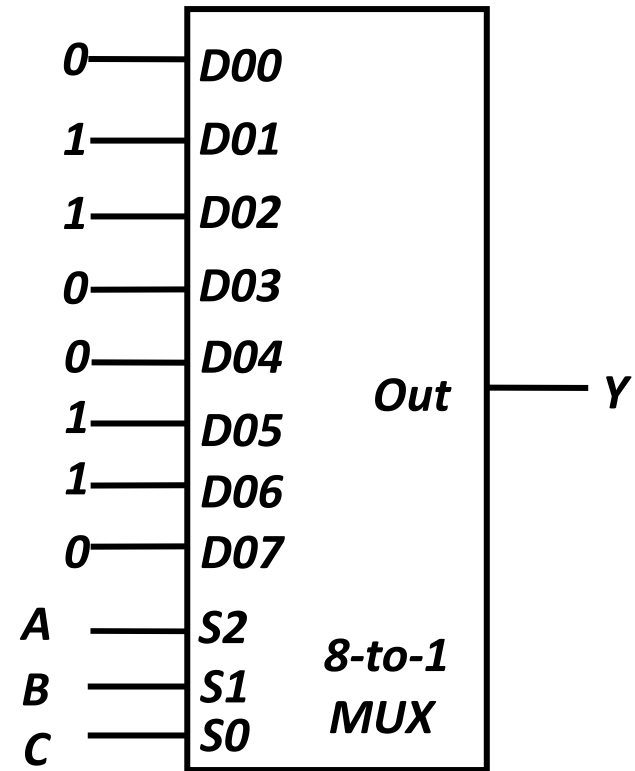
Gray A B C	Binary x y z
0 0 0	0 0 0
1 0 0	0 0 1
1 1 0	0 1 0
0 1 0	0 1 1
0 1 1	1 0 0
1 1 1	1 0 1
1 0 1	1 1 0
0 0 1	1 1 1

7. 基于复用器的组合电路

- 重新排列使得输入按计数顺序
- y和z 可通过双位8-1多路复用器实现
 - 将A, B, C连接到选择信号
 - 将y和z连接到两个输出
 - 将各自真值连接到待选择数据

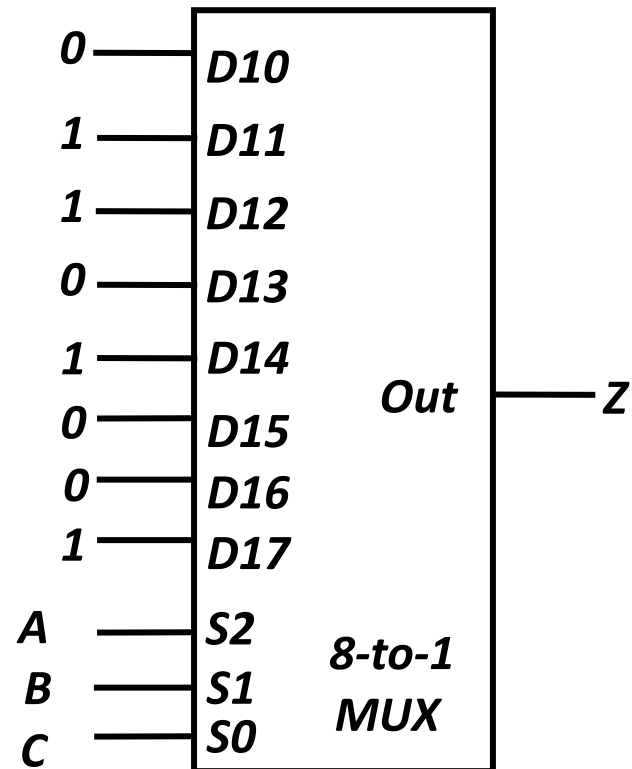
7. 基于复用器的组合电路

Gray A B C	Binary x y z
0 0 0	0 0 0
0 0 1	1 1 1
0 1 0	0 1 1
0 1 1	1 0 0
1 0 0	0 0 1
1 0 1	1 1 0
1 1 0	0 1 0
1 1 1	1 0 1



7. 基于复用器的组合电路

Gray A B C	Binary x y z
0 0 0	0 0 0
0 0 1	1 1 1
0 1 0	0 1 1
0 1 1	1 0 0
1 0 0	0 0 1
1 0 1	1 1 0
1 1 0	0 1 0
1 1 1	1 0 1



7. 基于复用器的组合电路

- 方法2: m 位宽 $2^{n-1}-1$ 多路复用器

- 得到函数的真值表

- 基于 $n-1$ 个变量值, 将真值表中的行配对

- $n-1$ 个变量一致

- 设剩下的变量为 X

- 每一配对中, 将输出表达成 $(0, 1, X, \bar{X})$

- 根据真值表

- 将 $n-1$ 个变量作为选择信号

- $(0, 1, X, \bar{X})$ 作为待选择数据

- 将多路复用器的输出标识成函数输出

7. 基于复用器的组合电路

□ 例子：格雷码到二进制码转换

□ 真值表如图所示

□ $x = C$

□ y 和 z 比较复杂

Gray A B C	Binary x y z
0 0 0	0 0 0
1 0 0	0 0 1
1 1 0	0 1 0
0 1 0	0 1 1
0 1 1	1 0 0
1 1 1	1 0 1
1 0 1	1 1 0
0 0 1	1 1 1

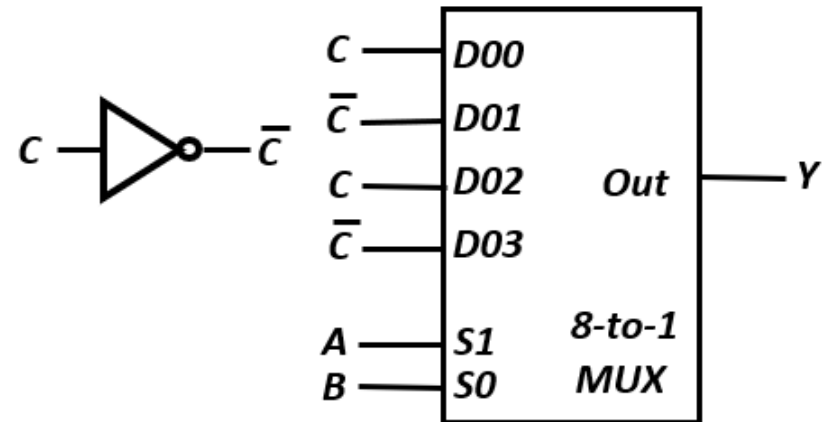
7. 基于复用器的组合电路

- 重排真值表，使得输入按照计数升序

Gray A B C	Binary x y z	Rudimentary Functions of C for y	Rudimentary Functions of C for z
0 0 0	0 0 0	F = C	F = C
0 0 1	1 1 1		
0 1 0	0 1 1	F = \overline{C}	F = \overline{C}
0 1 1	1 0 0		
1 0 0	0 0 1	F = C	F = \overline{C}
1 0 1	1 1 0		
1 1 0	0 1 0	F = \overline{C}	F = C
1 1 1	1 0 1		

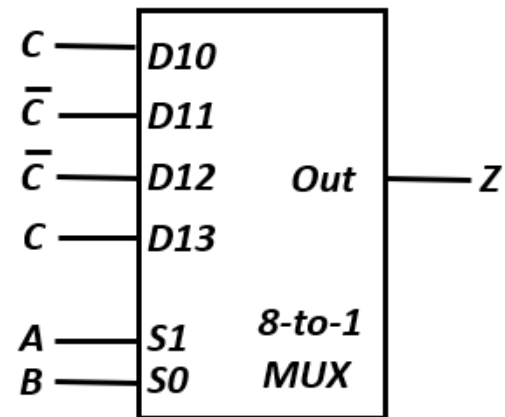
7. 基于复用器的组合电路

Gray A B C	Binary x y z	Rudimentary Functions of C for y	Rudimentary Functions of C for z
0 0 0	0 0 0	F = C	F = C
0 0 1	1 1 1		
0 1 0	0 1 1	F = \bar{C}	F = \bar{C}
0 1 1	1 0 0		
1 0 0	0 0 1	F = C	F = \bar{C}
1 0 1	1 1 0		
1 1 0	0 1 0	F = \bar{C}	F = C
1 1 1	1 0 1		



7. 基于复用器的组合电路

Gray A B C	Binary x y z	Rudimentary Functions of C for y	Rudimentary Functions of C for z
0 0 0	0 0 0	F = C	F = C
0 0 1	1 1 1		
0 1 0	0 1 1	F = \bar{C}	F = \bar{C}
0 1 1	1 0 0		
1 0 0	0 0 1	F = C	F = \bar{C}
1 0 1	1 1 0		
1 1 0	0 1 0	F = \bar{C}	F = C
1 1 1	1 0 1		



7. 基于复用器的组合电路

- 方法1 VS 方法2

- 方法1简单，方法2复杂

- 方法2 的门成本约是方法1的一半

□ 3-42 使用2个8-1多路复用器来构造一个15-1多路复用器。两个多路复用器应该相互连接，这样用于产生选择码0000至1110上的附加逻辑就最少。

□ 提示：

□ 两个多路复用器：1号，2号

□ 1号多路复用器可以选择8个数据：0000-0111

□ 2号多路复用器

□ 选择7个数据：1000-1110

□ 1号多路复用器的输出→1111

□ 将最高位作为复用器选择信号

三. 算术功能模块

- 1. 迭代组合电路
- 2. 二进制加法器
- 3. 二进制减法器
- 4. 其他算术模块

1. 迭代组合电路

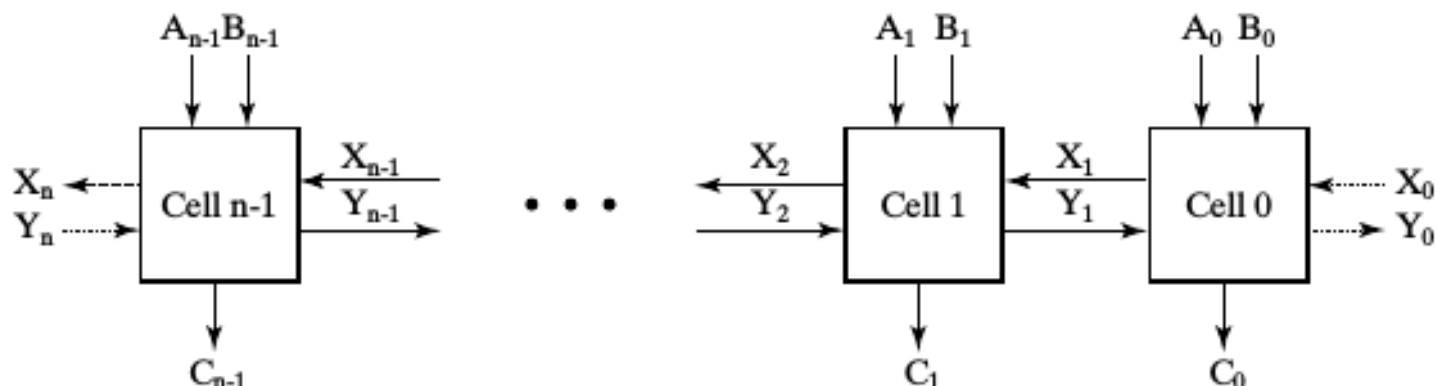
- 设计一个电路来处理32位二进制加法
 - 输入数量 = ?
 - 真值表行数 = ?
 - 布尔方程的输入变量个数 ?
 - 布尔方程包含非常多项
- 实际中不可行
- 那怎么办?
- 基本思想：利用规律性来简化设计

1. 迭代组合电路

- 类似的算术功能有以下规律：
 - 对二进制向量进行操作
 - 对每一位进行同样的子函数操作
- 设计子函数功能模块，重复使用得到总体功能
- 单元
 - 子函数模块
- 迭代阵列
 - 相互连接的单元的阵列

1. 迭代组合电路

- 单元
- 迭代阵列



2. 二进制加法器

- 半加器：2输入按位加功能模块
- 全加器：3输入按位加功能模块
- 行波进位加法器：二进制加法迭代阵列

2. 二进制加法器

- 半加器
- 输入: X, Y
- 输出: 和位 S , 进位 C

X	0	0	1	1
$+ Y$	$+ 0$	$+ 1$	$+ 0$	$+ 1$
<hr/>	<hr/>	<hr/>	<hr/>	<hr/>
$C\ S$	$0\ 0$	$0\ 1$	$0\ 1$	$1\ 0$

2. 二进制加法器

□ 半加器

□ 输入: X, Y

□ 输出: 和位 S , 进位 C

X	Y	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

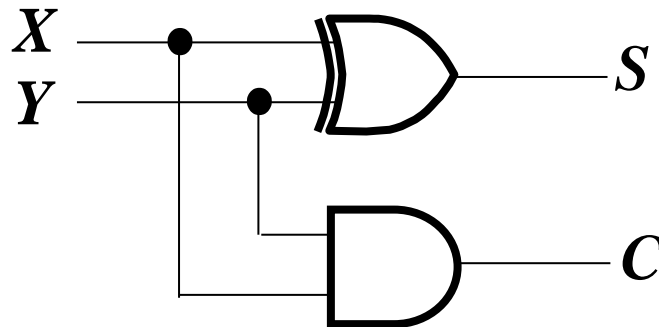
S		Y
	0	1
X	1	
	2	3

C		Y
	0	1
X		1
	2	3

2. 二进制加法器

- 可以得到多种表示
- 最常见实现：

$$S = X \oplus Y$$
$$C = X \cdot Y$$



2. 二进制加法器

□ 全加器

□ 输入：X, Y, 进位Z

□ 输出：和位S, 进位C

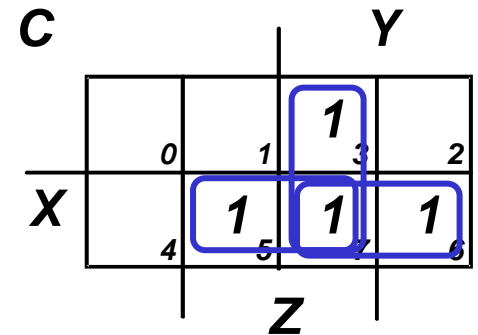
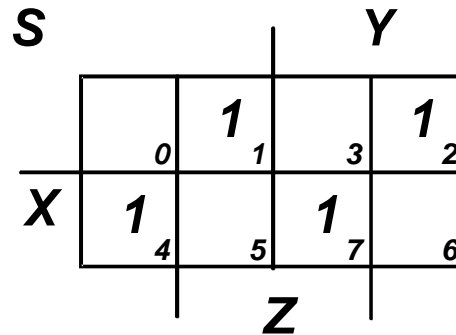
Z	0	0	0	0
X	0	0	1	1
<u>+ Y</u>	<u>+ 0</u>	<u>+ 1</u>	<u>+ 0</u>	<u>+ 1</u>
C S	0 0	0 1	0 1	1 0

Z	1	1	1	1
X	0	0	1	1
<u>+ Y</u>	<u>+ 0</u>	<u>+ 1</u>	<u>+ 0</u>	<u>+ 1</u>
C S	0 1	1 0	1 0	1 1

2. 二进制加法器

- 全加器
- 输入：X, Y, 进位Z
- 输出：和位S, 进位C

X	Y	Z	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1



2. 二进制加法器

□ S 是三位异或函数 (奇函数)

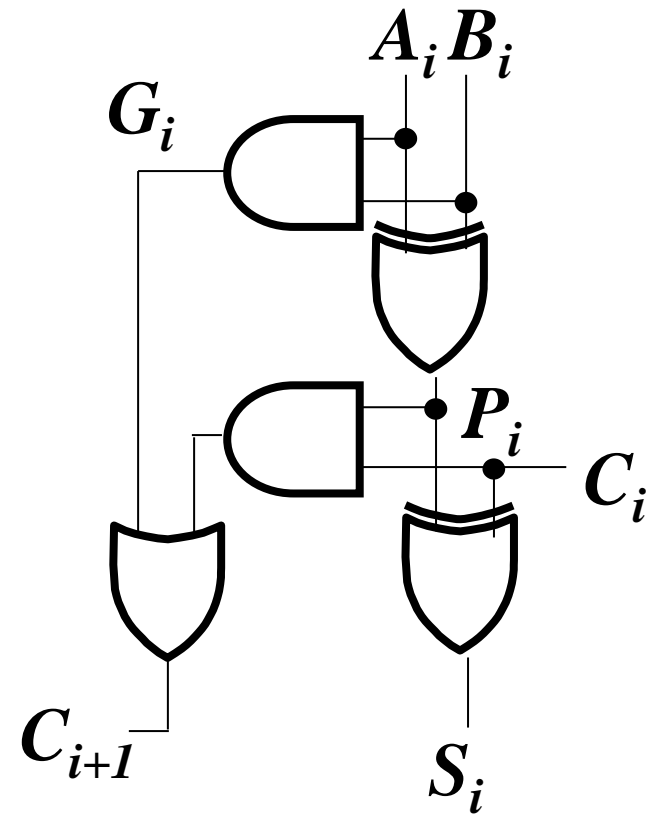
$$S = X \oplus Y \oplus Z$$

□ C是 $C = XY + (X \oplus Y)Z$

- 项 $X \cdot Y$ 是进位生成
- 项 $X \oplus Y$ 是进位传播

2. 二进制加法器

- 全加器概要图
- X, Y, Z (上页) 是 A, B, C
- 同时
 - G = 进位生成
 - P = 进位传播
- 组成
 - 两个半加器
 - 一个或门



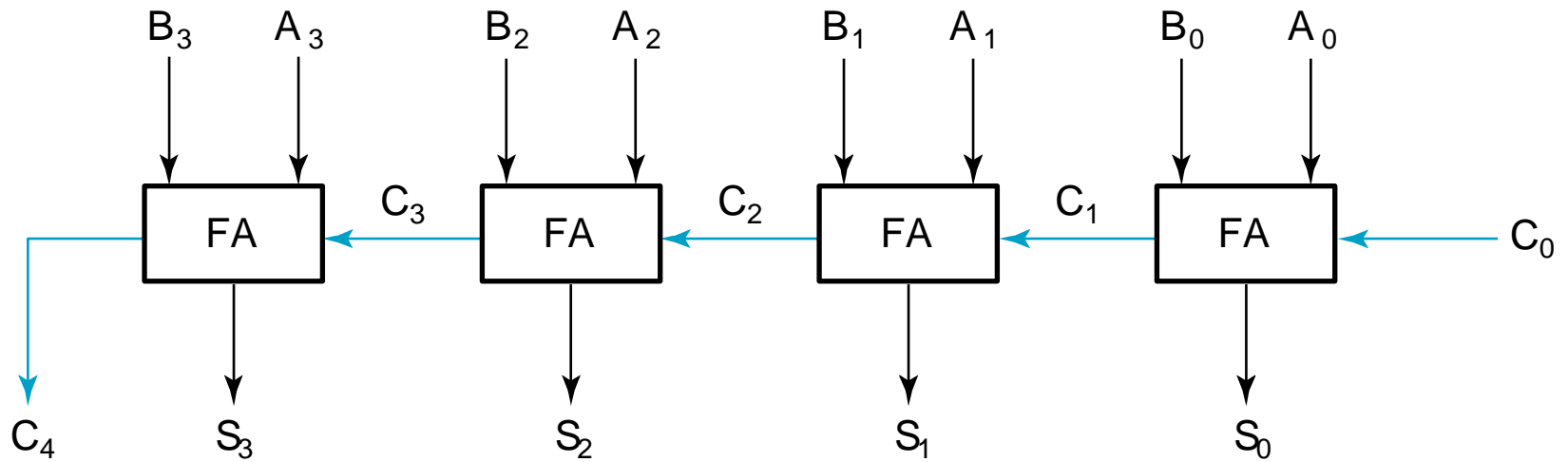
2. 二进制加法器

□ 4位行波进位加法器

➤ 迭代阵列

➤ 单元

➤ 1位全加法器



3. 二进制减法器

- 二进制补码

- B的补码为 $2^n - B$

- 二进制反码

- B的反码为 $2^n - 1 - B$

- 按位取反

- 二进制补码

- 按位取反加1

3. 二进制减法器

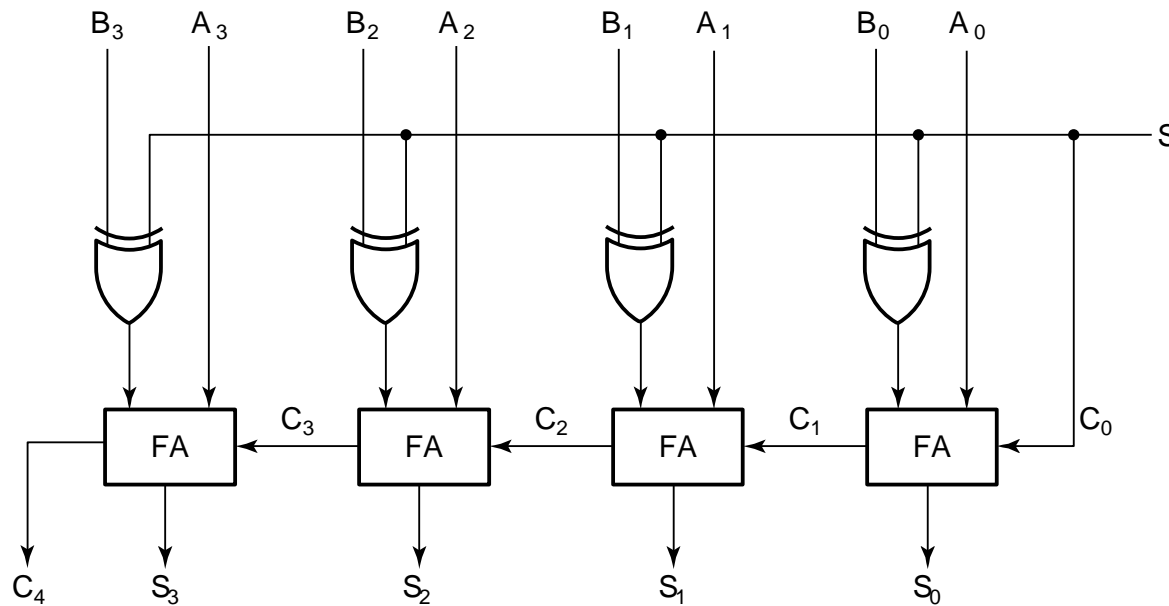
- 减法可以按照补码的加法执行
- $A-B$ 即是
- $A \geq B$
 - $2^n + A - B$
 - 得到 $A-B$
- $A < B$
 - $2^n - (B - A)$
 - 得到 $B-A$ 的补码
- 无符号数加减运算规则
- 采用符号-补码表示有符号数运算规则
- 两者一致

3. 二进制减法器

□ 电路如图所示，计算 $A+B$ 和 $A-B$

➤ $S=1$ ，减法

➤ $S=0$ ，加法



3. 二进制减法器

□ 溢出

- 进位 C_n
- $V = C_n \oplus C_{n-1}$

□ 无符号数

➤ 加法

- C_n 为1发生溢出
- C_n 为0无溢出

➤ 减法

- C_n 为1：结果为正，无需校正
- C_n 为0：结果为负数补码，需要校正

3. 二进制减法器

□ 溢出

- 进位 C_n
- $V = C_n \oplus C_{n-1}$

□ 有符号数 (符号-补码)

- $V=0$, 无溢出
- $V=1$, 有溢出, 表明运算结果有 $n+1$ 位, 溢出位为符号位

4. 其他算术模块

- 压缩
- 递增
- 递减
- 常数乘法
- 常数除法
- 零扩充和符号扩展

4. 其他算术模块

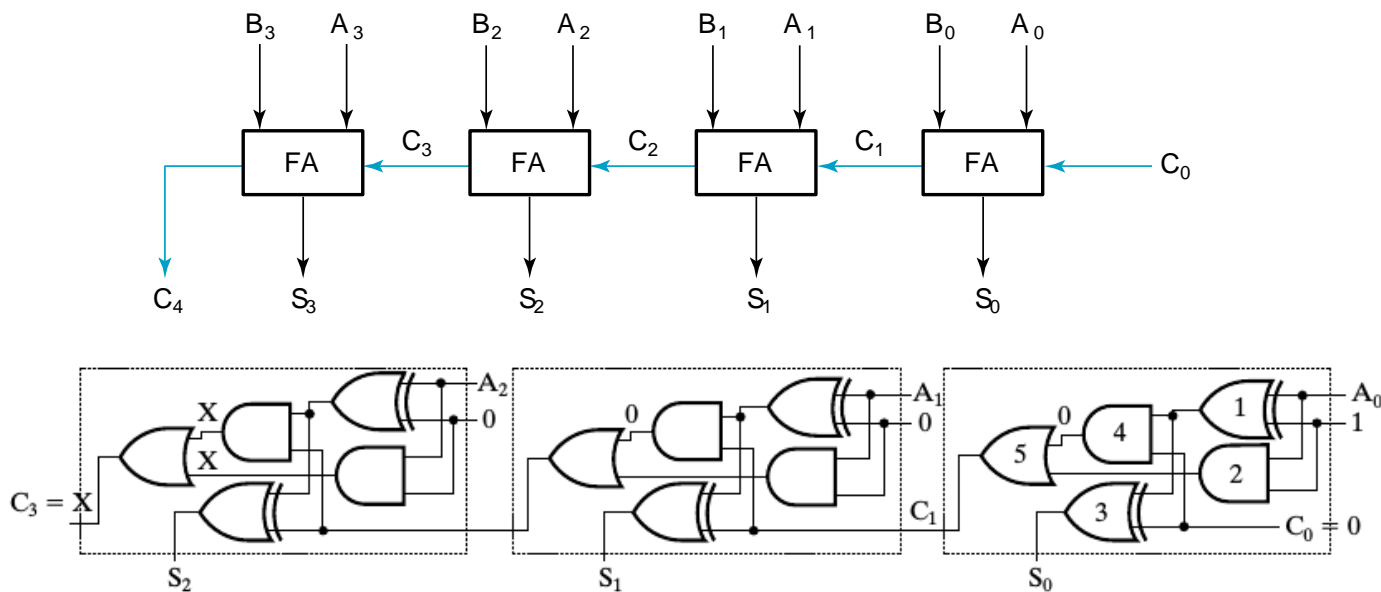
□ 压缩

- 针对特定应用将已有电路化简成一个简单电路
- 简化电路设计, 代替直接设计电路
- 通过将输入端的值固定、传递和取反
- 将未使用的输出端置为X
 - ✓ 该输出门和仅驱动该输出门的门可移去

3. 其他算术模块

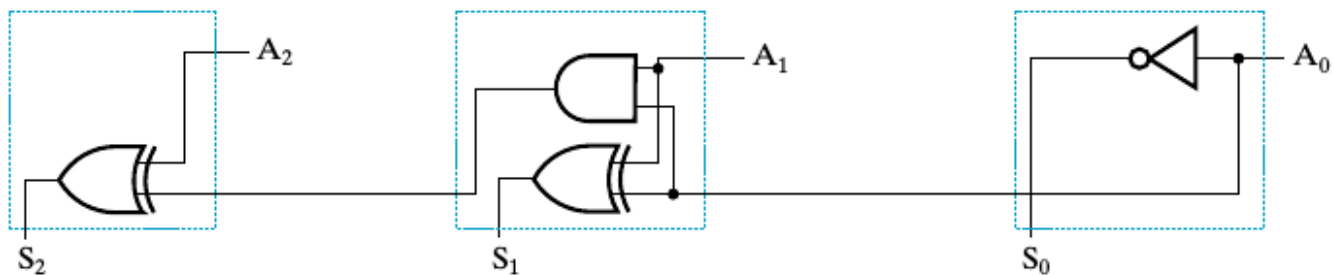
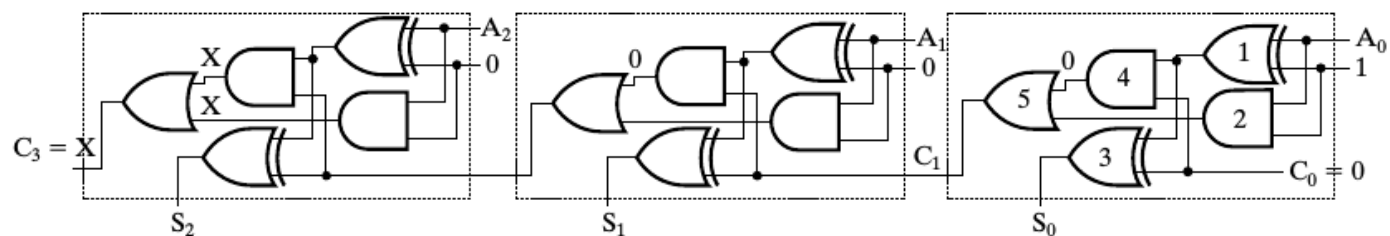
□ 递增器

- $C_0=0$
- $B_0=1 \ B_1=0 \ B_2=0$
- $C_3=X$



4. 其他算术模块

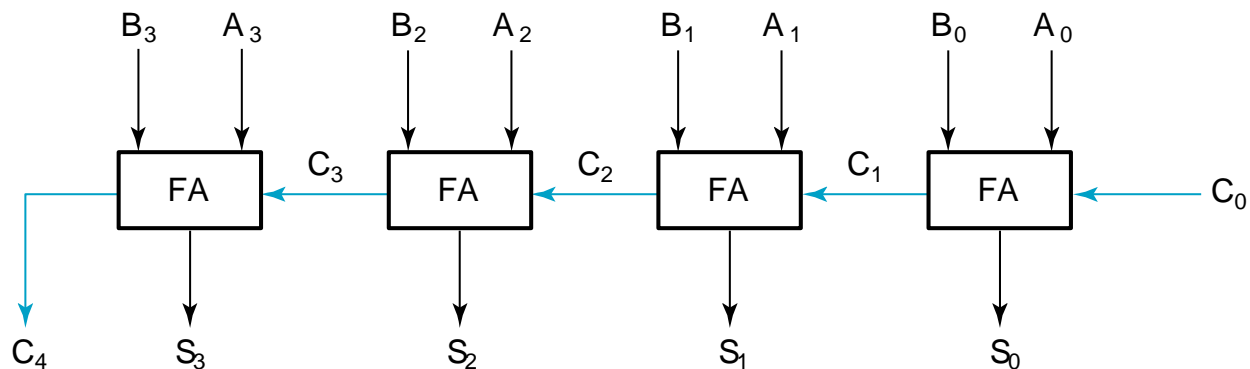
□ 递增器



4. 其他算术模块

□ 递减器

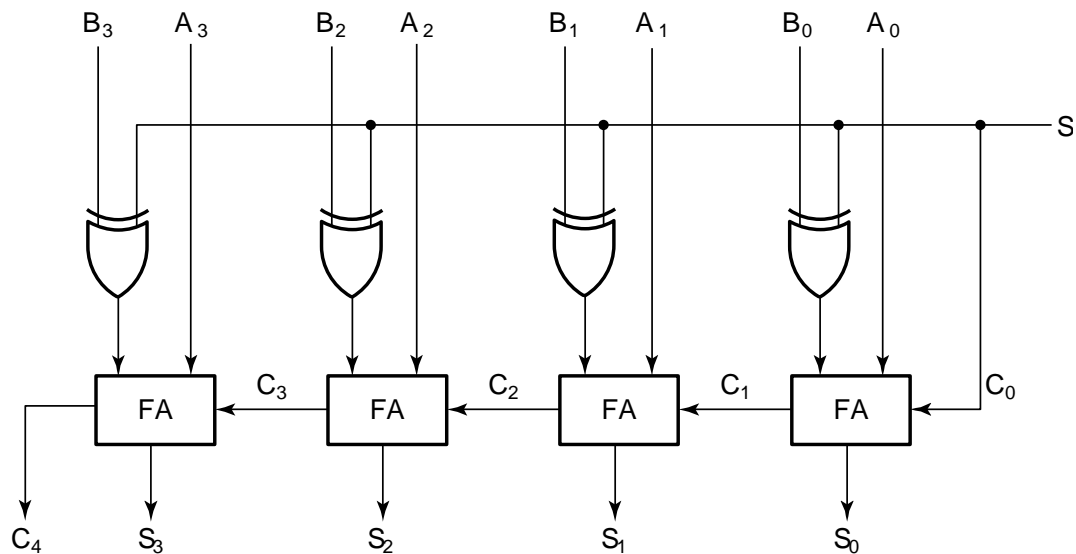
- $C_0=0$
- $B_0=1 \ B_1=1 \ B_2=1 \ B_3=1$
- $C_4=X$



4. 其他算术模块

□ 递减器

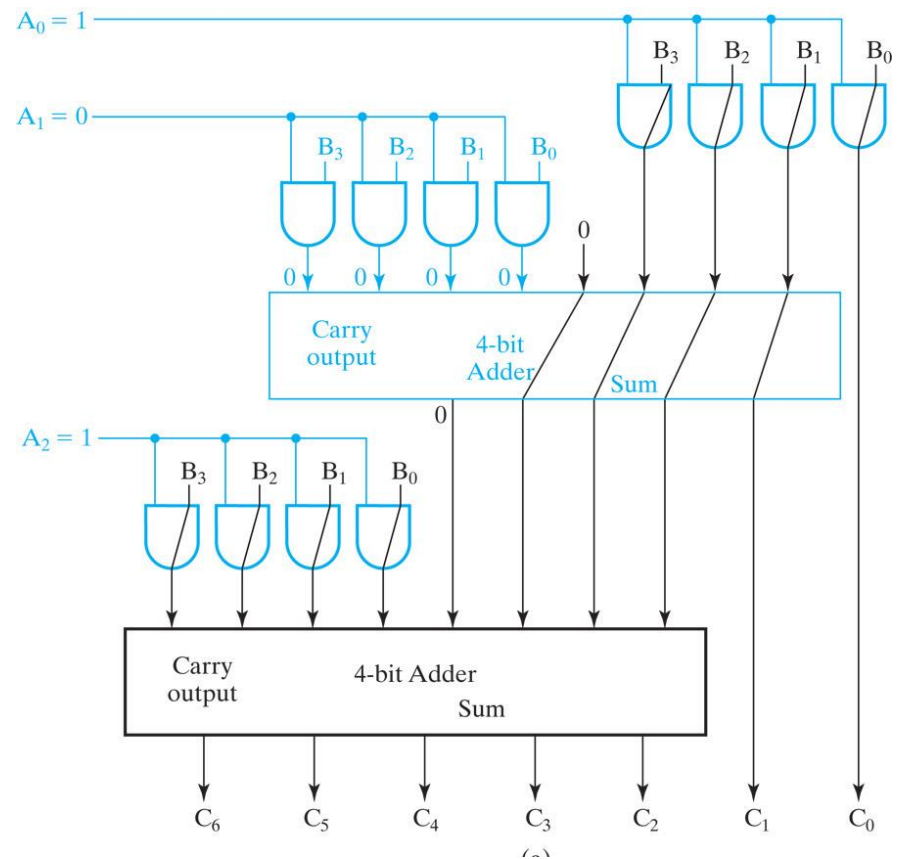
- $S=1$
- $B_0=1 \ B_1=0 \ B_2=0 \ B_3=0$
- $C_4=X$



4. 其他算术模块

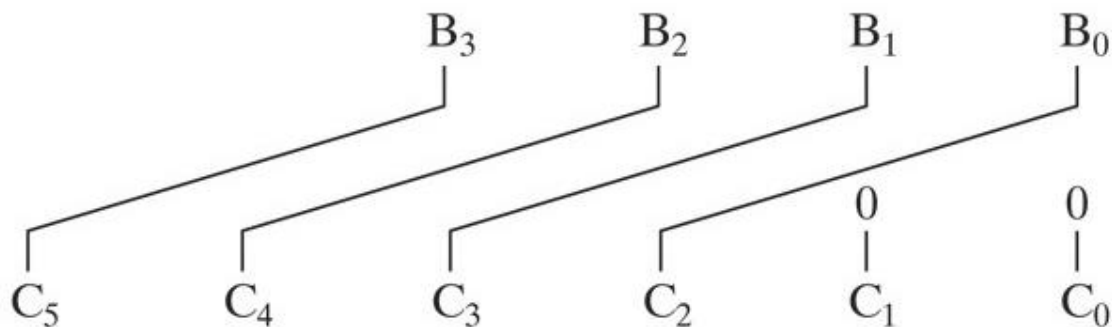
□ 常数乘法(101)

$B_3 B_2 B_1 B_0$
 $B_3 B_2 B_1 B_0$
 $B_3 B_2 B_1 B_0$



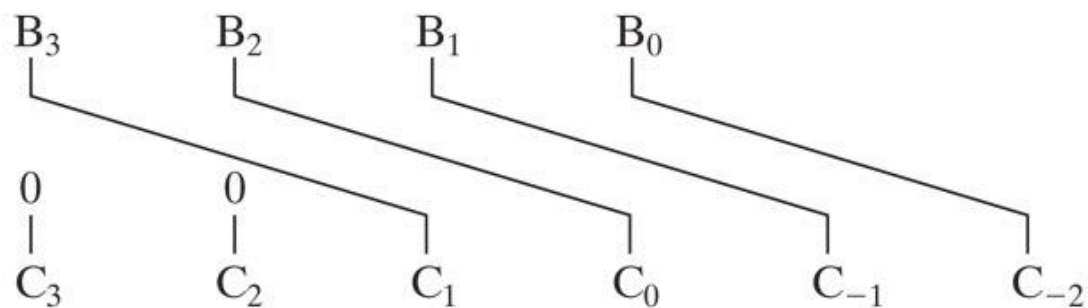
4. 其他算术模块

□ 常数乘法(100)



4. 其他算术模块

□ 常数除法



4. 其他算术模块

- 零填充：用于增加操作数的位数
- 11110101填充成16位
 - 左填充：0000000011110101
 - 右填充：1111010100000000

4. 其他算术模块

- 符号扩展：增加符号-补码表示符号数位数
- 01110101 符号扩展到16位
 - 0000000001110101
- 11110101 符号扩展到16位
 - 1111111111110101
- 11110101 零填充到16位
 - 0000000011110101
 - 不正确