数字逻辑

Chapter 1 数字系统与信息

信息的表示与数字系统

信息

- 信号是信息表示的物理载体
- 数字信号广泛采用两个离散值，称为二进制

数字系统

数字系统是一种离散信息处理系统，采用一组离散形式的信息作为输入，采用一组离散形式的内部信息操控**系统状态**，产生离散形式的信息作为**输出**。

数制与算术运算

数制

数的表示规则称为数制

- 基底(r)：一个数值所包含的数字符号的个数
- 权(r^i)：数字符号的位置所决定的值
- 任何一个数值，都是各位数字本身的**值与其权之积的总和**

数制转换

1. m进制转十进制

   按权展开

2. **十进制数转m进制**

   整数部分除m取余（得到的余数从右向左排列），小数部分乘m取整（得到的整数从左向右排列）

3. 二-八-十六进制互转

   分组转换

算数运算

加减乘与十进制时类似

编码

确定规则，然后按此规则编出代码，并给代码赋以一定的含义，就是编码

BCD码

用4位二进制表示十进制的10个数字，注意BCD码仍是 十进制数

字符编码

ASCII码、Unicode....

校验位

在二进制编码额外添加的一位，用于表示编码中1的个数是奇数还是偶数

- 偶校验：偶数个“1”，校验位是“0”
- 奇校验：奇数个“1”，校验位是“0”

无法确定哪一位出错

格雷码

在一组数的编码中，若任意两个相邻的代码只有一位二进制数不同，则称这种编码为格雷码

- 优点

  能够减少误差

- 编码方式

  前一半数值：左边最高位为0，往右各位由原二进制编码的每一位与它左边相邻位的偶校验构成

  后一半数值：前一半逆序排列，并将左边最高位置1

十进制数	自然二进制数	格雷码	十进制数	自然二进制数	格雷码
0	0000	0000	8	1000	1100
1	0001	0001	9	1001	1101
2	0010	0011	10	1010	1111
3	0011	0010	11	1011	1110
4	0100	0110	12	1100	1010
5	0101	0111	13	1101	1011
6	0110	0101	14	1110	1001
7	0111	0100	15	1111	1000

简单的，$(n+1)$位格雷码的前2^n个与n位格雷码相同（首位置0）；后2^n个与n位格雷码的逆序相同（首位置1）

Chapter 2 布尔代数

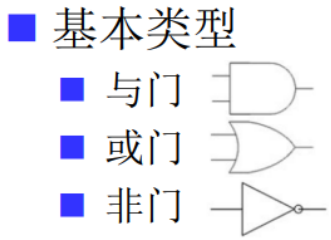
二值逻辑

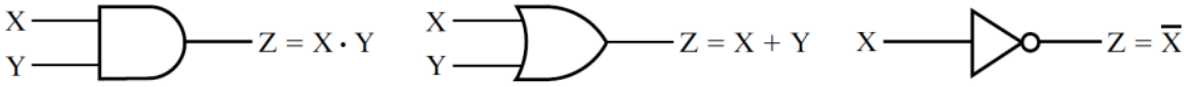
包括二值变量及对这些变量所施加的逻辑运算

- 与运算
- 或运算
- 非运算
- 异或运算

逻辑门

处理一个或多个输入信号，产生一个输出信号的数字电路称为逻辑门





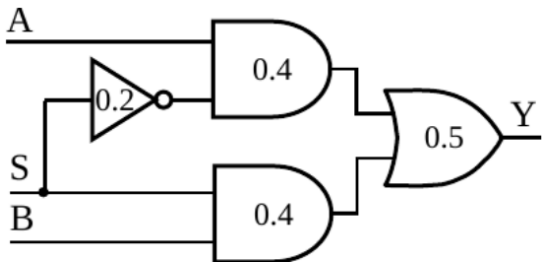
可以通过 定时图 的方式表示时序中的逻辑门输入和输出关系

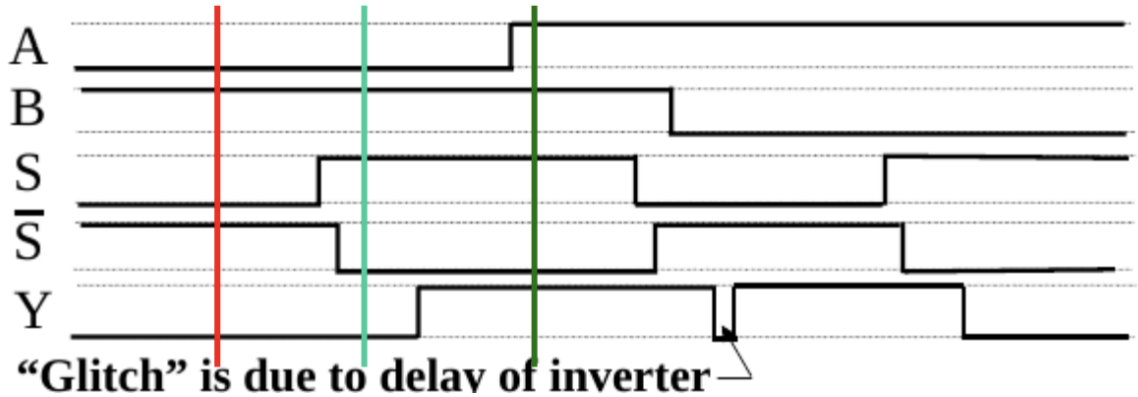
- 传播延迟

  输入信号变化引起输出信号相应变化所需要的时间，用t_G表示

毛刺现象

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





在箭头所指的地方，$A=1, B=0, S=0$，按理来说Y应该取1，但事实却是0，WHY?

观察电路，我们可以发现：S到Y的延迟时0.9ns，而$\overline{S}$到Y的延迟确实1.1ns，也就是说，若S发生变化（假设从1变0），那么0.9ns后Y收到的 S信号已经变为0 ，而其收到的 $\overline{S}$信号仍然为0 ！再过0.2ns后$\overline{S}$信号才变为1，电路恢复正常。

所以，毛刺现象产生的原因是：输入信号同时变化，但是他们经过组合电路到达输出信号的延迟不同，所以就会产生一个不稳定的一部分信号更新而一部分信号未更新。

布尔代数

基本概念

- 布尔代数：是一种处理逻辑变量和逻辑运算的代数方法
- 布尔表达式：是由二进制变量、常量0和1、逻辑运算符号和括号组成的代数运算式
- 布尔函数：是由函数值变量、等号和布尔表达式组成的函数

布尔恒等式

■ 与运算

$$X \cdot 1 = X$$
$$X \cdot 0 = 0$$
$$X \cdot X = X$$
$$X \cdot \overline{X} = 0$$

■ 或运算

$$X + 0 = X$$
$$X + 1 = 1$$
$$X + X = X$$
$$X + \overline{X} = 1$$
$$\overline{\overline{X}} = X$$

■ 异或运算

$$X \oplus 0 = X$$
$$X \oplus X = 0$$
$$X \oplus \overline{Y} = \overline{X \oplus Y}$$
$$X \oplus 1 = \overline{X}$$
$$X \oplus \overline{X} = 1$$
$$\overline{X} \oplus Y = \overline{X \oplus Y}$$

有且仅有一个输入变量为1，异或结果为1。

布尔代数性质

1. 对偶原则
2. 德摩根律等定律

   $\overline{X_1 + X_2 + \ldots + X_n} = \overline{X}_1 \cdot \overline{X}_2 \cdot \cdots \cdot \overline{X}_n$

   $\overline{X_1 \cdot X_2 \cdot \cdots \cdot X_n} = \overline{X}_1 + \overline{X}_2 + \ldots + \overline{X}_n$
3. 反函数，例如：

   $F = \overline{X}Y\overline{Z} + \overline{XY}Z$

   $\overline{F} = (X + \overline{Y} + Z) \cdot (X + Y + \overline{Z})$

布尔等式推导

例

$$AB+\overline{A}CD+\overline{A}BD+\overline{A}C\overline{D}+ABCD=B(A+D)+\overline{A}C$$

$$XY+\overline{X}Z+YZ=XY+\overline{X}Z$$

一致律定律

$$YZ=YZ(X+\overline{X})$$

布尔代数的标准形式

- 最小项

  所有变量都以原变量或反变量的形式出现， 且仅出现一次，这样的**乘积项**叫做最小项。n个变量，共有2^n个不同的最小项

- 最大项

  所有变量都以原变量或反变量的形式出现， 且仅出现一次，这样的**和项**叫做最小项。n个变量，共有2^n个不同的最大项

最小项有唯一取真取值，最大项有唯一取假取值

任意布尔表达式都可以唯一的变为标准和之积或标准积之和的形式

与离散数学中的最大项最小项相同

布尔代数的化简

优化的逻辑电路成本

1. 文字成本 L

   布尔表达式中的文字的个数、最大/小项的个数要最少

2. 门成本 G

   G = 全部文字数 + 除单个文字之外的全部项数 = L + T

代数化简

利用布尔代数中基本的布尔恒等式、代数性质等对布尔函数进行约简，进而能够简化数字电路

卡诺图化简

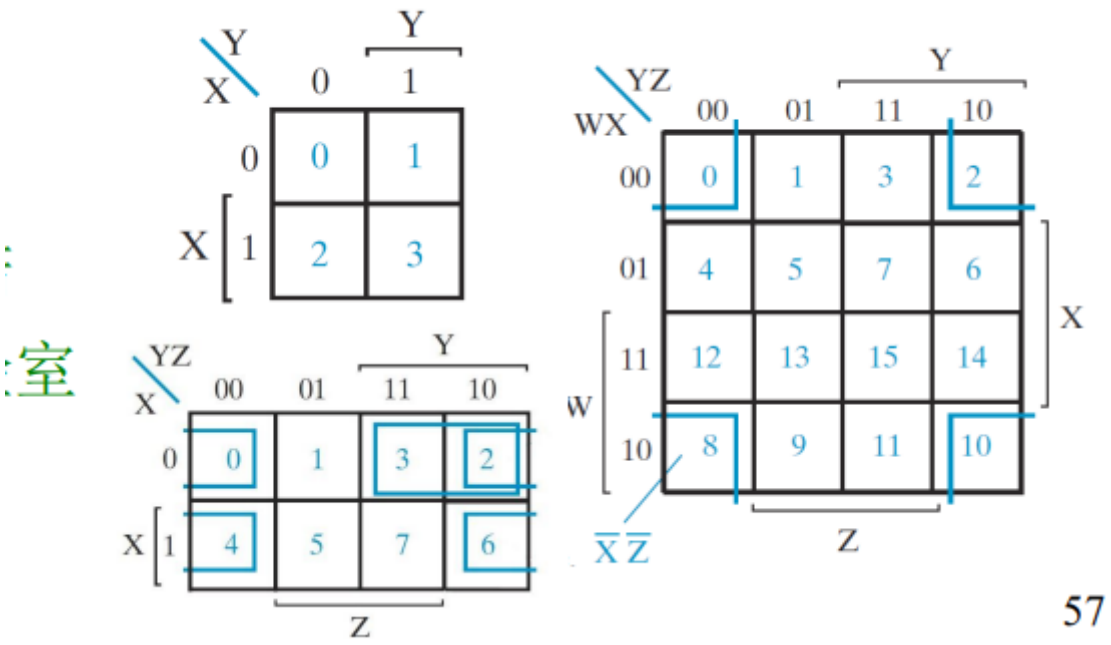
基本概念

卡诺图：

- 方格组成的集合
- 每个方格对应一个最小项
- 水平和竖直依次排列变量

卡诺图的构造方式

1. 小方格的数量等同于最小项的数量
2. 采用格雷码排列，相邻的项只有一处不同



对应取值为真的方格处，里面要置1

化简的一般原则

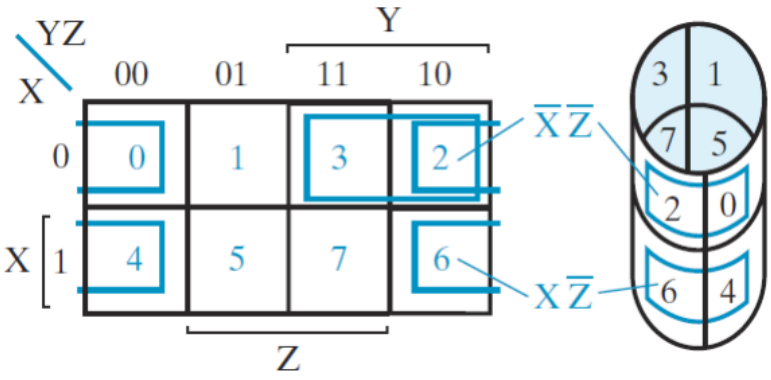
最基本的原则：

1. 矩形的方格数必须是2的幂次方：1、2、4、8
2. 找出最少的矩形来包含或覆盖所有标记为1的方格：矩形数尽可能少，矩形尽可能大
3. 根据矩形数写出最简式

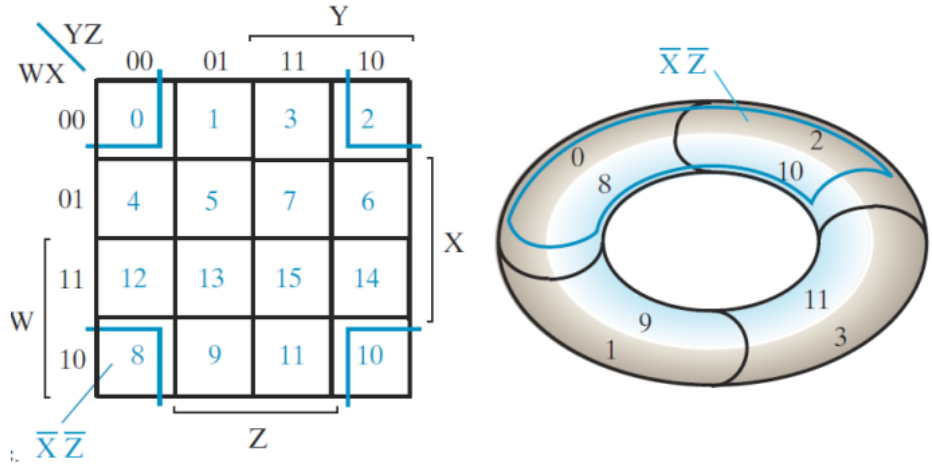
如何实现?

第一点容易实现

第二点要注意一些地方视觉不相连而实际相连，如：三个变量时有圆柱图，四个变量时有环图



卡诺图　　圆柱图



卡诺图　　环图

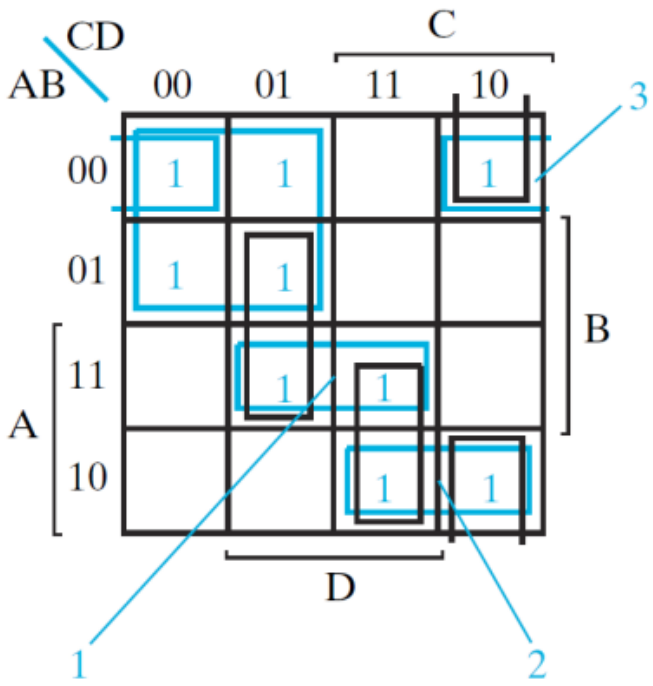
化简的具体方法：

1. 确定所有的主蕴涵项
2. 对全部质主蕴涵项进行求和
3. 加上其他主蕴涵项用来覆盖剩余的、不被质主蕴涵项所包含的最小项

主蕴含项：卡诺图中不能被更大卡诺圈包含的、由2^n个1方格组成的矩形

质主蕴含项：至少包括一个没有被任何其他主蕴含项覆盖的方格的矩形

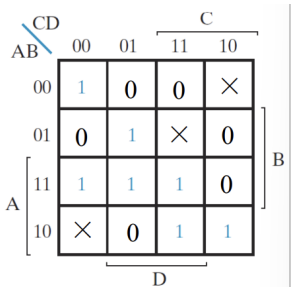
■ 例子 $F=\sum m(0,1,2,4,5,10,11,13,15)$



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

不完全确定函数的化简

无关最小项：在实际操作中不会出现或者取值对结果无影响的最小项，如：



在实际优化操作中，无关最小项可以取0也可以取1，怎么方便怎么取。对于上面的卡诺图，可以先假设X都是1，但是不认为X可以生成 质主蕴含项 ，针对质主蕴含项得出公式后，可以加上含X的公式，然后得出优化的方程。

和之积优化

很简单，上面的卡诺图操作的对象也都是卡诺图中的“1”，优化出的结果都是积之和。

只需要把操作的对象换成卡诺图中的“0”，就能轻松得到优化后的和之积。

Chapter 3 组合逻辑电路分析与设计

一个组合逻辑电路由：

- m个布尔输入
- n个布尔输出
- n个转换函数

构成

组合逻辑电路的设计过程

设计过程分为五步：

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

规范化

确定输入输出范围

形式化

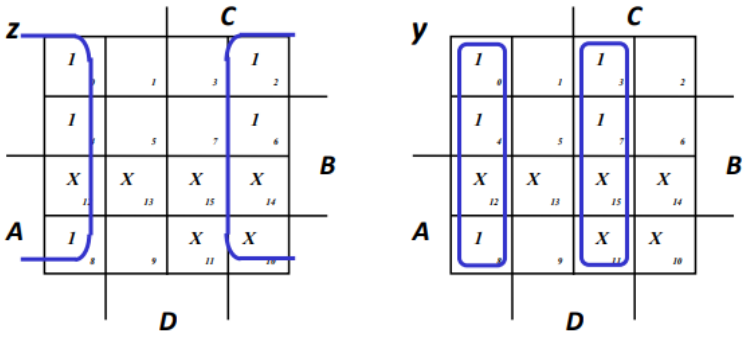
得出真值表或者布尔函数

优化

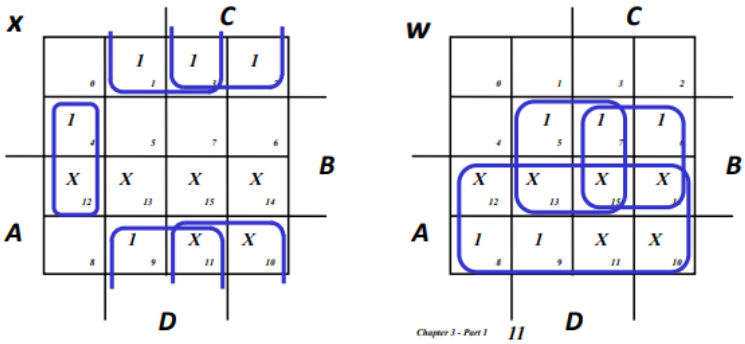
用卡诺图优化，降低门输入成本

例子：由该真值表

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



得出了4个卡诺图：



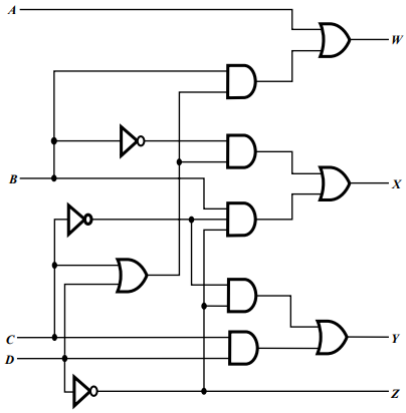
最后化简得到：

- $W = A + BC + BD$
- $X = \overline{B}C + \overline{B}D + B\overline{C}\overline{D}$
- $Y = CD + \overline{C}\overline{D}$
- $Z = \overline{D}$

还可以通过**共享电路**进行进一步优化，如令$\overline{C}\overline{D} = \overline{C + D} = \overline{T}$，则有：

- $W = A + BT$
- $X = \overline{B}T + T\overline{B}$
- $Y = CD + \overline{T}$
- $Z = \overline{D}$

最后得到下图所示电路：



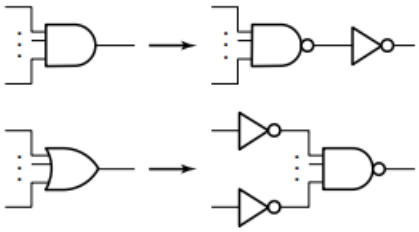
工艺映射

通过连结词完备集将电路用一种或两种门表示出来

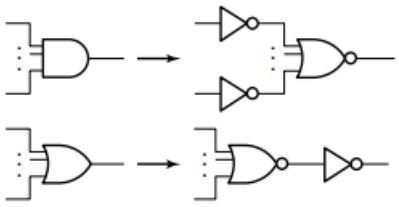
映射过程：

1. 用与非门/或非门替换掉与门和或门

与非门



或非门



2. 将反相器推过电路中的**扇出点**，其实就是一个变量不要用多个非门
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**：
    - 电路输入或或非门的输出
    - 或非门输入

验证

人工逻辑验证或者模拟验证

组合逻辑功能模块

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。

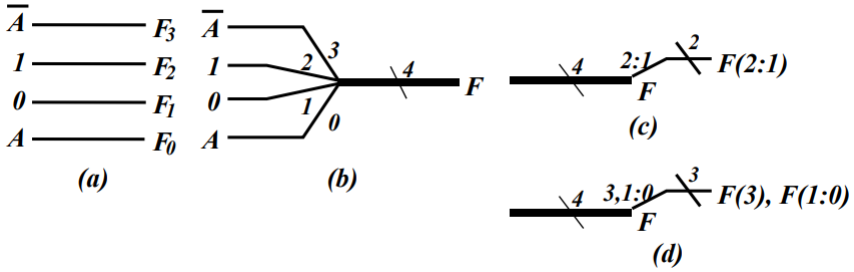
基本逻辑函数

单变量函数

一个变量X的函数，可在输入处作功能块

多位函数

一位函数的向量；粗线代表**总线**，是一个**向量信号**；可以从总线中分割出一个位子集。



使能函数

是否允许函数从输入传递到到输出

引入**使能信号EN**。EN为1时允许，为0时不允许并输出一个 固定的信号值 ，可能为0可能为1。

译码与编码

译码：输入n位，输出m位，其中$n \leq m \leq 2^n$

编码：输入最大m位，输出n位，其中$n \leq m \leq 2^n$

译码与编码互逆

译码器

实现译码功能的电路

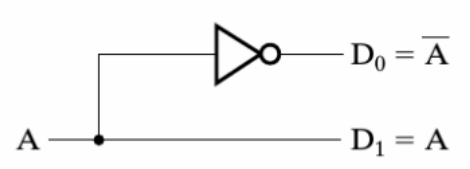
译码有n位输入m位输出，

1-2译码器

A	D_0	D_1
0	1	0
1	0	1

$$D_0 = \overline{A}$$

$$D_1 = A$$



2-4译码器

简单的来讲，2-4译码器可以由两个1-2译码器加上4个与门（作用于四种输出）构成

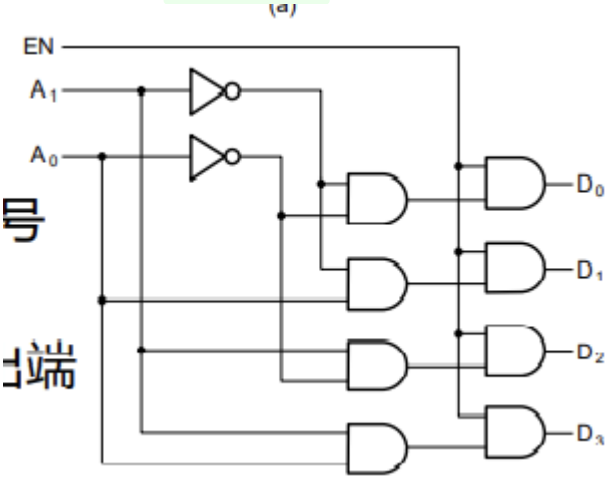
n-2^n译码器

需要2^n个与门，其中每个与门和两个译码器连接，这两个译码器的输入相等或者只相差1，并且输入之和为n。不断展开，直到展开到1-2译码器。

通过修改$n-2^n$译码器，可以得到$n-m$译码器，如7-128译码器可以由3-8译码器和4-16译码器和128个与门构成。

带使能的译码器

电路输出增加 使能信号-EN ，并且$n-2^n$译码器再增加2^n个输出与门以结合使能：



当EN为0时，A_0, A_1变为X，X可以表示0或者1

此时相当于EN为输入信号，A_0, A_1为输出端 选择信号

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

基于译码器的组合电路设计

- 实现1个函数，其中有n个变量
- 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（最小项之和）
- 简单来说，n个输入能输出所有的最小项或者最大项；将所有最小项和起来或者最大项或起来就能得到最后的电路

但是直接通过译码器得到组合电路，就相当于没有用卡诺图把最小项之和优化成积之和， 门输入成本太高

编码器

十进制——BCD编码器

- 输入: 10位代表从0到9，$(D_0, \ldots, D_9)$
- 输出: 4位BCD码
- 函数: 若输入位 Di 是1, 则输出(A_3, A_2, A_1, A_0) 是i的BCD码

步骤：真值表→卡诺图优化→优化

有布尔方程：

- $A_3 = D_8 + D_9$
- $A_2 = D_4 + D_5 + D_6 + D_7$
- $A_1 = D_2 + D_3 + D_6 + D_7$
- $A_0 = D_1 + D_3 + D_5 + D_7 + D_9$

但是如果输入有不止一个1，该编码器就无法工作

优先编码器

下面以五输入十进制-BCD编码器为例讲解如何解决这个问题

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

$$A_2 = D_4$$
$$A_1 = \overline{D}_4 D_3 + \overline{D}_4\overline{D}_3 D_2 = \overline{D}_4 F_1,\ F_1 = (D_3 + D_2)$$
$$A_0 = \overline{D}_4 D_3 + \overline{D}_4\overline{D}_3\overline{D}_2 D_1 = \overline{D}_4(D_3 + \overline{D}_2 D1)$$
$$V = D_4 + F_1 + D_1 + D_0$$

显然该编码器只人他看到的“第一个1”，后面的不管，然后输出对应BCD码

也就是说，D4的优先级最高，D3次之，D0优先级最低

选择和复用器

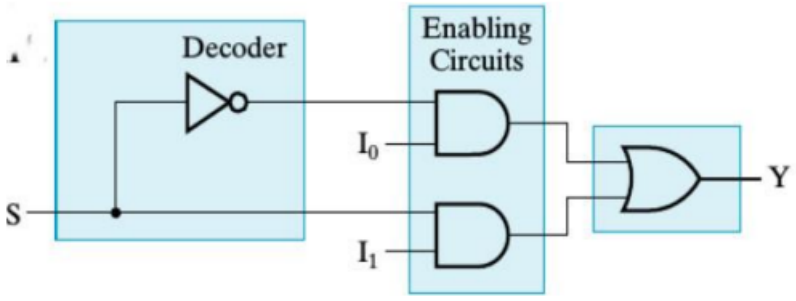
多路复用器

多路复用器是执行选择的逻辑电路，输入输出如下：

- 输入：一组待选择的数据，最多2^n个($I_0 \dots I_{2^n}$);
  一组用来进行选择的选择信号n个($S_{n-1} \dots S_0$)
- 输出：一个输出Y

多路复用器通过改变选择信号S，就可以实现对带选择的数据I的选择输出

1. 2-1多路复用器



当$S = 0$时，输出I_0；当$S = 1$时，输出I_1

实际上，该多路复用器由：

- 一个1-2译码器
- 两个使能信号
- 2-输入或门

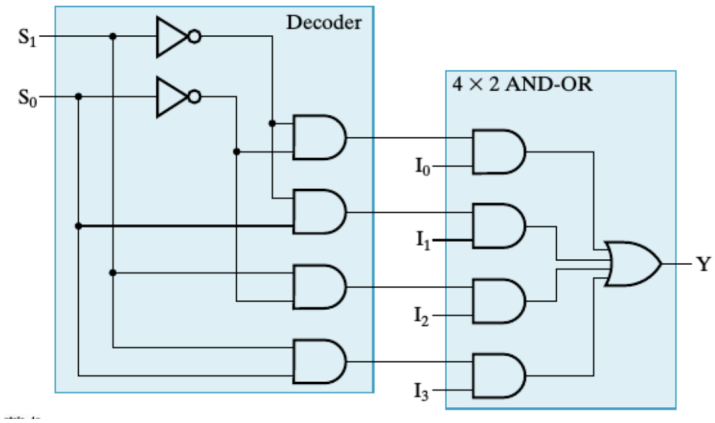
组成，由此可以得出下列的多路复用器

2. $2^n - 1$多路复用器

由：

- 一个$n-2^n$译码器
- 2^n个使能信号
- 2^n-输入或门

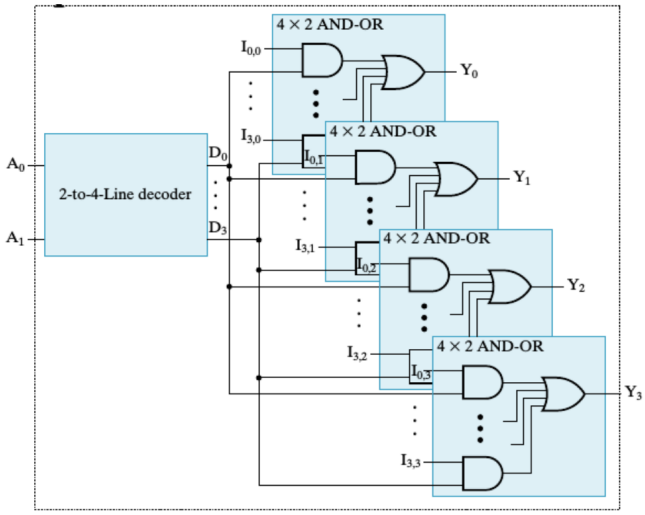
组成。其中2^n个使能信号和2^n个输入或门看作 $2^n \times 2$个与或门



位宽展开

上面所说的多路复用器的输入都是 单个位 ，多位的多路复用器输入采用 位向量

也就是说n位的多路复用器要实现n个函数，有n个输出；在译码器之后要平行地使用n个$2^n \times 2$个与或门



基于复用器的组合电路

实现**m个函数**，包含n个变量

1. m位宽2^n-1多路复用器
    - 得到函数真值表
    - 根据真值表进行如下操作
        - 将函数输入$S_n-1,\ldots,S_0$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

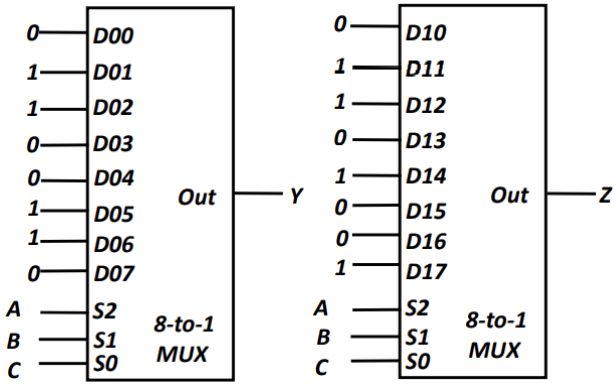
    例子，格雷码到二进制码的转换：

1. 重新排列真值表使输入按 计数顺序

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

2. y和z通过一个双位8-1多路复用器实现：

   A，B，C连到选择信号；y和z连到输出信号，各自的真值连到待选择数据：



2. m位宽的 $2^{n-1}-1$ 多路复用器

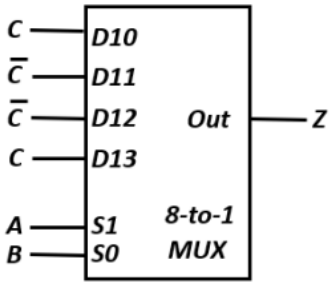
先得到函数的真值表：

- **重排真值表**，使得输入按照计数升序
- 基于n-1个变量值，将真值表中的 行配对 ，其中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$(0,1,X,\overline{X})$

根据真值表：

- 将n-1个变量作为选择信号
- $(0,1,X,\overline{X})$作为待选择数据
- 将多路复用器的输出标识成函数输出

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		



行配对：在上图右侧的多路复用器中,我们可以看到:对于某个AB的组合,Z的值仅由C决定.所以行配对就是要得出AB相同时的函数输出与C的关系,以构建最终的多路复用器.

算术功能模块

迭代组合电路

基本思想：利用规律性来简化设计

在实现算数功能时，一般是对二进制向量进行操作，对向量中的每一位进行**同样**的子函数操作

所以可以重复使用一些功能

单元：子函数模块

迭代阵列：相互连接的单元的阵列

二进制加法器

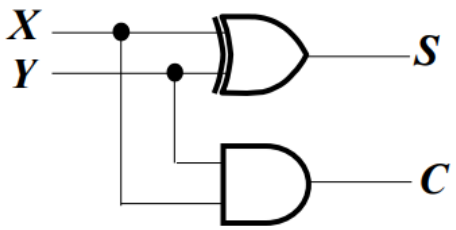
半加器

输入：X，Y

输出：和位S，进位C

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

公式：$S = X \oplus Y$，$C = X \cdot Y$



全加器

输入：X，Y，进位Z

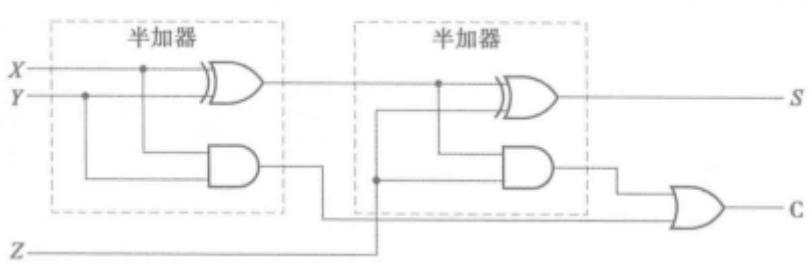
输出：和位S，进位C

真值表：

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

公式：$S = X \oplus Y \oplus Z$，$C = XY + (X \oplus Y)Z$

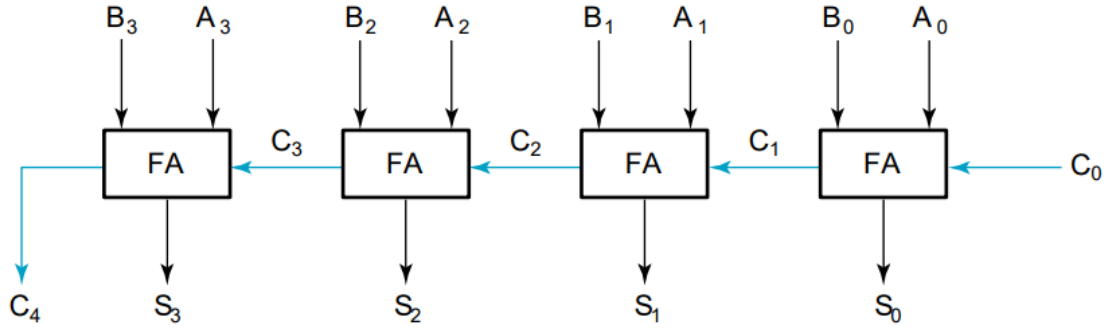
电路图：



实际上全加器由两个半加器和一个或门组成

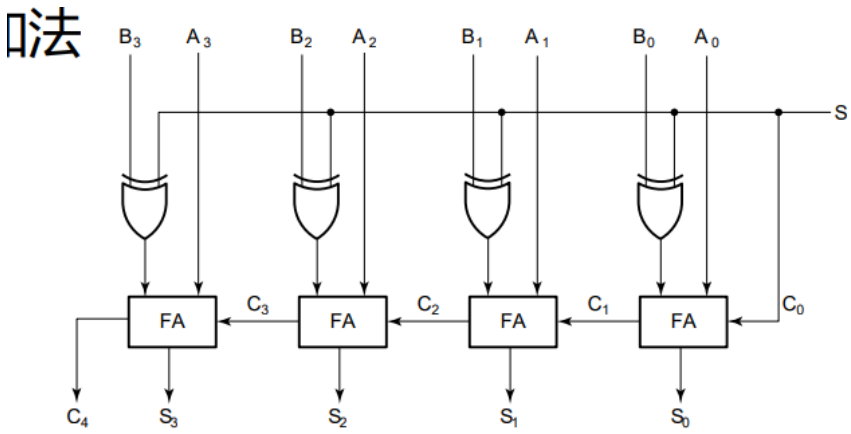
4位行波进位加法器

单元是1位全加法器



二进制加减法器

按照补码的加法执行



当S=1时，对B按位取反（反码），$C_0 = 1$使得B变成补码（反码+1）。注意，最终计算得到的结果如果 没有产生进位 ，那么需要对结果 求补并添加负号 ；如果产生进位则计算结果就是真实结果，进位不用管他。

当S=0时，等同于二进制加法

有符号的二进制加减法

负数的合理表示

在二进制加减法器中我们发现，减法在该电路模块中还没有完全完成，还需要根据是否有进位来进行校正；并且上面的加减法器只适用于无符号数加减。为解决这一问题，需要引进新的负数表示方法：`符号-二进制补码表示法`。

符号-二进制补码表示法：`负数用其绝对值的补码来表示，包括绝对值的符号位0`

有符号的二进制加法

用补码表示负数的两个有符号二进制数的加法运算，只需要将其包含符号位在内的两个数相加，然后丢弃进位

有符号的二进制减法

用补码表示负数的两个有符号二进制数的减法运算，对减数取补，再将其与被减数相加，然后丢弃进位

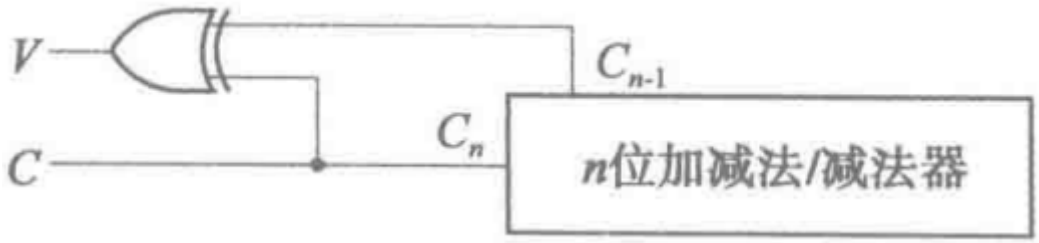
所以，只要采用**符号-二进制补码表示法**，上面的二进制加减法器的电路仍然可以正常使用，而此时不需要在进行计算后的校正操作

溢出

溢出：两个n位数产生的结果需要用n+1位数保存，那么称其发生了溢出

溢出检测

如果**符号位的进位输入与进位输出**不相等，那么发生了溢出



压缩

从一个基本电路（如二进制加法器或乘法器）出发，将已有的电路转换成有用的、较简单的电路来简化设计，从而代替直接设计电路。针对特定应用将已有电路简化成一个简单电路，我们称这个过程为`压缩`

对于已经设计好的功能块，通过将其输入端的值固定、传递和取反，即可实现新的功能

递增/递减

由加减法器压缩可得

Chapter 4 时序电路分析与设计

时序电路简介

概念

时序电路：电路任意时刻的稳态输出，不仅取决于该时刻的输入，而且与前一时刻电路的状态有关，即输出=F(输入，状态)

由存储元件（存储当前状态）和组合电路（决定下一状态与输出）连接而成

组合电路：

- 输入：从外部输入信号+当前状态
- 输出：往外部输出信号+下一状态

存储单元：

- 输入：**下一状态**
- 输出：新的当前状态

时序电路类型

- 同步时序电路

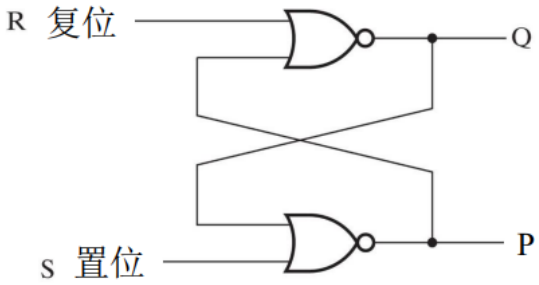
  触发器 作为基本元件，状态只会在时钟上升/下降的边沿发生

- 异步时序电路

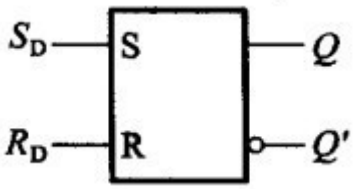
  锁存器 作为基本元件，状态可以在任意时间发生变化

锁存器

S-R锁存器



图形符号：



由两个交叉耦合的或非门构成，禁止S=1且R=1的输入

可以观察到Q与P的取值总是相反的，所以P也可以写为$\overline{Q}$

得到真值表

S	R	Q	$\overline{Q}$	
1	0	1	0	置位状态
0	0	1	0	
0	1	0	1	复位状态
0	0	0	1	

进行时序行为分析，得：

S_D	R_D	Q	Q*
0	0	0	0
0	0	1	1
1	0	0	1
1	0	1	1
0	1	0	0
0	1	1	0

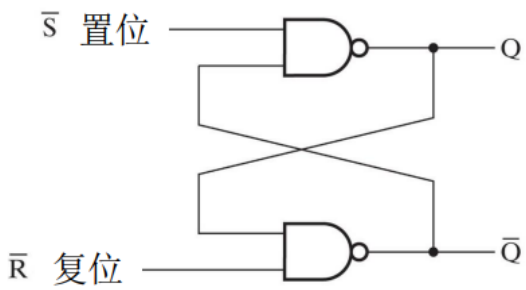
01变0，10变1，00不变

即$Q_{t+1}=\overline{\overline{S}_t\cdot\overline{\overline{R}_t\cdot Q_t}}$，$\overline{Q}_{t+1}=\overline{\overline{R}_t\cdot\overline{\overline{S}_t\cdot\overline{Q}_t}}$

为什么R=1且S=1是不稳定的状态?

简单来说，当输入从$R=1,S=1$变为$R=0,S=0$时，因为传播延迟，R和S不会同时变为0，就会产生“10”或“01”的**中间态**，导致Q的值改变，而Q改变的值是未知的（你不知道RS是变成10还是01）。

$\overline{S}-\overline{Q}$锁存器



由两个交叉耦合的或非门构成，禁止$\overline{S}=0$且$\overline{R}=0$的输入

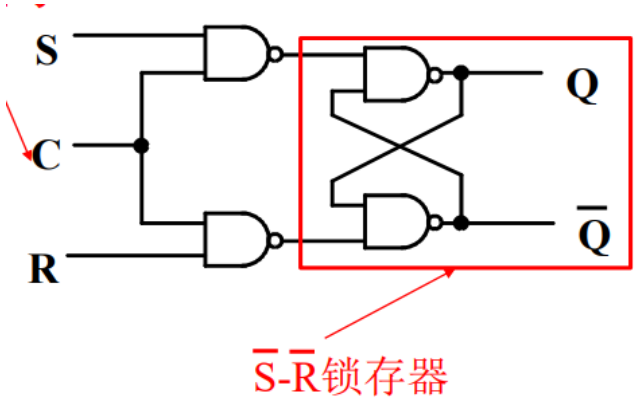
真值表：

S'_D	R'_D	Q	Q*
1	1	0	0
1	1	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0

时钟S-R锁存器

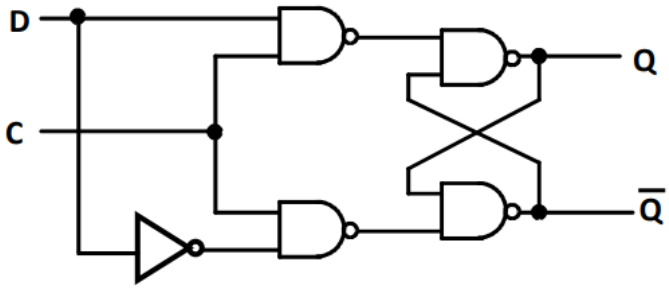
在$\overline{S}-\overline{R}$锁存器基础上添加额外控制信号C，以此开启或关闭（触发）锁存器的功能

- 当C=1时：实现S-R锁存器的功能(因为相当于对S和R取反了,$\overline{S}-\overline{R}$变成$S-R$)
- 当C=0时：锁存器不受输入的影响(相当于输入为11的$\overline{S}-\overline{R}$)



D锁存器

在时钟$S-R$锁存器的基础上添加了一个反相器，**消除了未定义状态**.



- D：数据信号
- C：控制信号

可以看到，D锁存器与SR锁存器的区别就是输入信号从S和R变成了D，其中S就是D而R是$\overline{D}$，这样确保了不会出现$S\cdot R=1$的情况

锁存器作为存储单元的问题

时钟信号C=1时，锁存器开始改变状态；锁存器的输出端通过某些组合电路与一些锁存器的输入端相连。而在时钟信号保持为1的这段时间，如果锁存器的输入信号发生变化，锁存器会响应这一变化进入新的状态然后将其输出，而不是保持原来的状态，这会引起一系列的状态变化。最终锁存器会进入一个不可预测的状态。

触发器

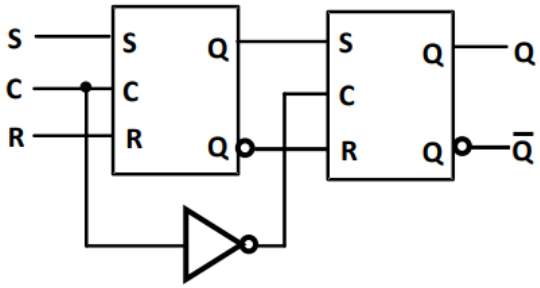
触发：输入信号值的改变可以控制内部锁存器的状态

触发器泛指用于记忆1位二进制信号的基本单元电路，其中锁存器是触发器的基本构成部分

触发器解决了锁存器中存在的上述问题

脉冲触发式触发器（主从触发器）

由两个时钟S-R锁存器串联而成, 其中第二个锁存器的时钟信号由反相器取反



从左到右的输入到输出路径被时钟信号不同取值切断：

- 左：主锁存器，C=1时能够根据输入改变状态
- 右：从锁存器，C=0时能够根据输入改变状态

从该电路图可以看到，当C=1时，触发器的输出不会立即改变；等到C=0时输出信号才会发生改变

缺点: 1箝位

当$C=1$时，S变为1再变为0，主锁存器Q被置1，然后当C=0时，从锁存器输出1，一切正常；

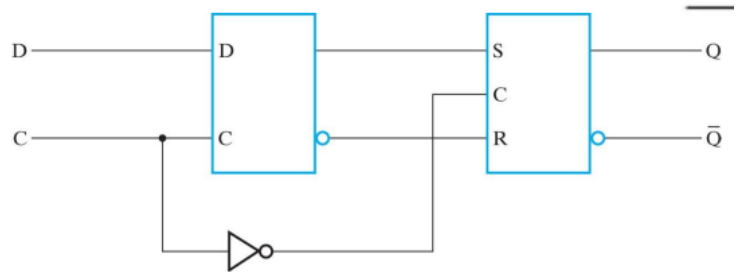
可是如果当$C=1$时，S变为1再变为0之后，R再变为1然后恢复为0，此时原本已被置1的主锁存器就又被置0，原先置1的操作没有传到从锁存器就被顶掉了，这就是1箝位

边沿触发式触发器（使用最为广泛）

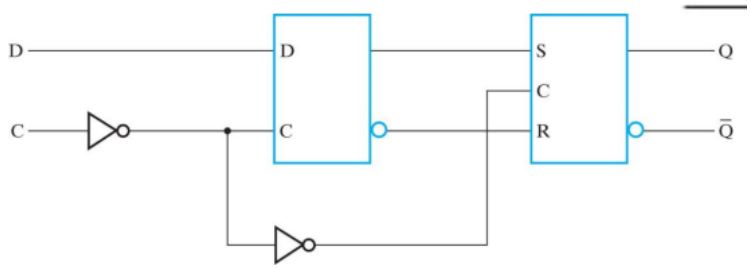
采用**边沿触发**来代替主从触发

边沿触发器忽略在常数水平的脉冲，只有在**时钟信号转换**的时候被触发

负边沿触发的D触发器：1跳变到0触发. 相当于把主从$S-R$触发器的第一个时钟$S-R$锁存器换成了时钟D锁存器



正边沿触发的D触发器：0跳变到1触发



正边沿比负边沿多一个取反

为什么边沿触发可以避免1箝位?

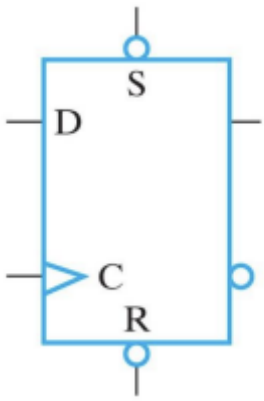
因为发生1箝位的原因究其根本在于时钟信号C保持高电平1时，输入的变化还未反应到输出就可能被下一个输入顶掉

而边沿触发保证了只有“一次”输入，自然就消除了1箝位的问题

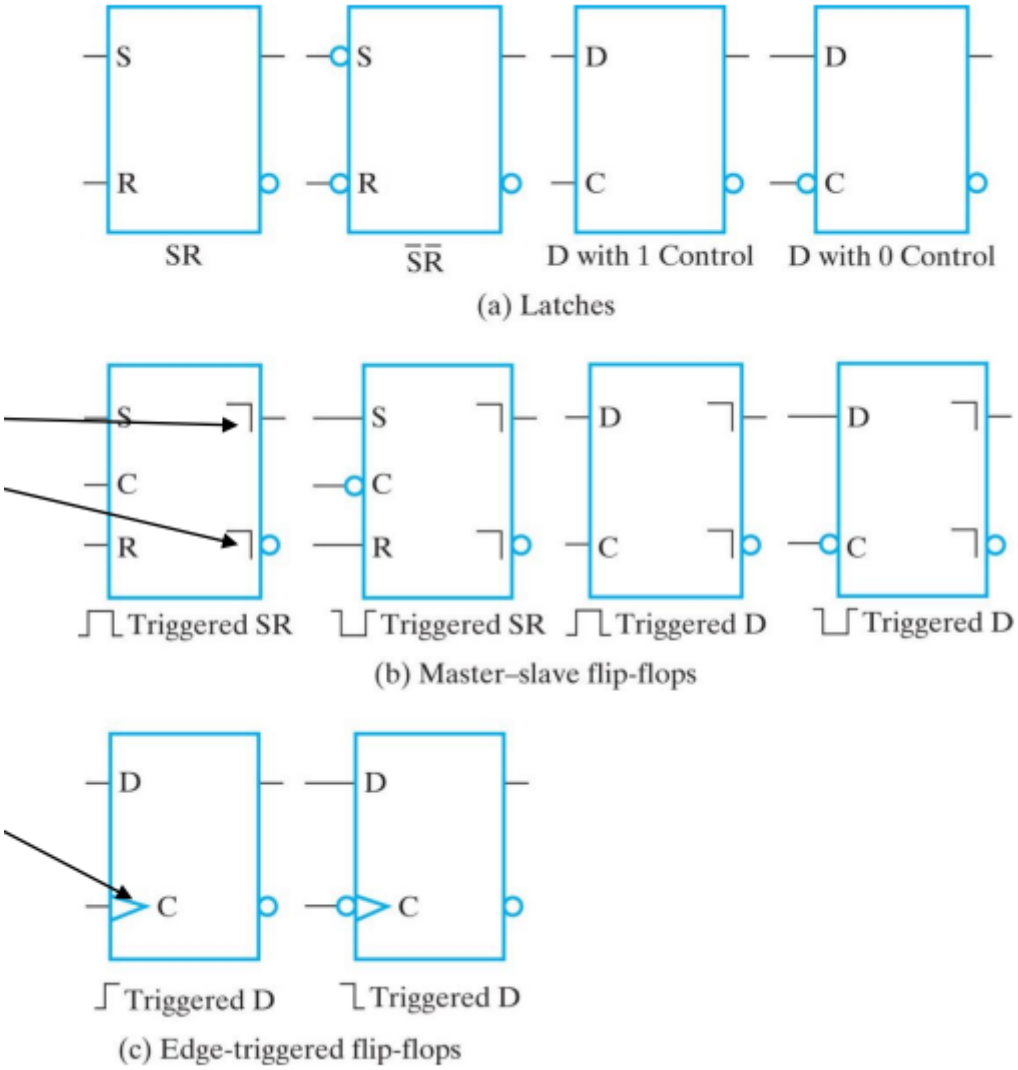
直接输入

直接置位或者预置，以及直接复位或者清零

在时钟正常运行之前，将触发器设置成一个**初始状态**



存储单元的标准图形符号



时序电路分析

基本模型

电路输入、输出及当前状态的时序模型

- 当前状态存储在触发器存储元件
- 下一状态是当前状态和输入的布尔函数
- 时间t时的输出是t时状态的布尔函数

状态表

输入、输出、状态构成的多变量表

- 当前状态：状态变量的合法值
- 输入：合法的输入组合
- 下一状态：基于当前状态和输入的（t+1）时的状态值
- 输出：基于当前状态和输入的输出值

从真值表的角度理解：输入=输入+当前状态，输出=输出+下一状态

状态表可以用下一状态和输出函数推导

一维状态表

输出栏是在当前时刻状态和输入的组合下的输 出值；当前状态和输入合并在一个组合栏中

$$D_A = AX + BX$$
$$D_B = \bar{A}X$$
$$Y = (A + B)\bar{X}$$

Present State		Input	Next State		Output
A	B	X	A	B	Y
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	1
0	1	1	1	1	0
1	0	0	0	0	1
1	0	1	1	0	0
1	1	0	0	0	1
1	1	1	1	0	0

二维状态表

当前状态列在表的左侧，输入则从左到右列在表的第一行组成二维表描述下一状态及输出

$$D_A = AX + BX$$
$$D_B = \bar{A}X$$
$$Y = (A + B)\bar{X}$$

Present State		Next State				Output	
		X = 0		X = 1		X = 0	X = 1
A	B	A	B	A	B	Y	Y
0	0	0	0	0	1	0	0
0	1	0	0	1	1	1	0
1	0	0	0	1	0	1	0
1	1	0	0	1	0	1	0

如果交换后两行顺序，状态行和输入列符合格雷码的顺序，与卡诺图匹配！

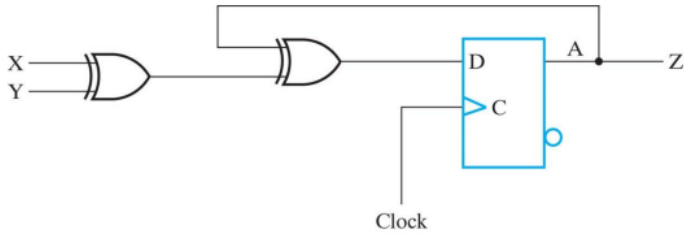
37

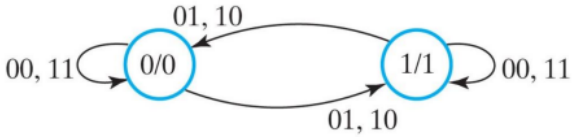
状态图

采用图形式表示状态表中的输入输出信息

- 每个**状态**用一个**圆圈**表示
- 每个**状态转移**用当前状态和下一状态的**有向线段**表示
- 在有向线段上标注出输入表示造成状态转移的**原因**

例子：





Present State	Inputs		Next State	Output
A	X	Y	A	Z
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	0
1	0	0	1	1
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Moore型和Mealy型状态图

状态图分为**Moore**型和**Mealy**型

Moore型的输出放在圆圈里，Mealy型的输出放在有向线段上

Moore型　　Mealy型

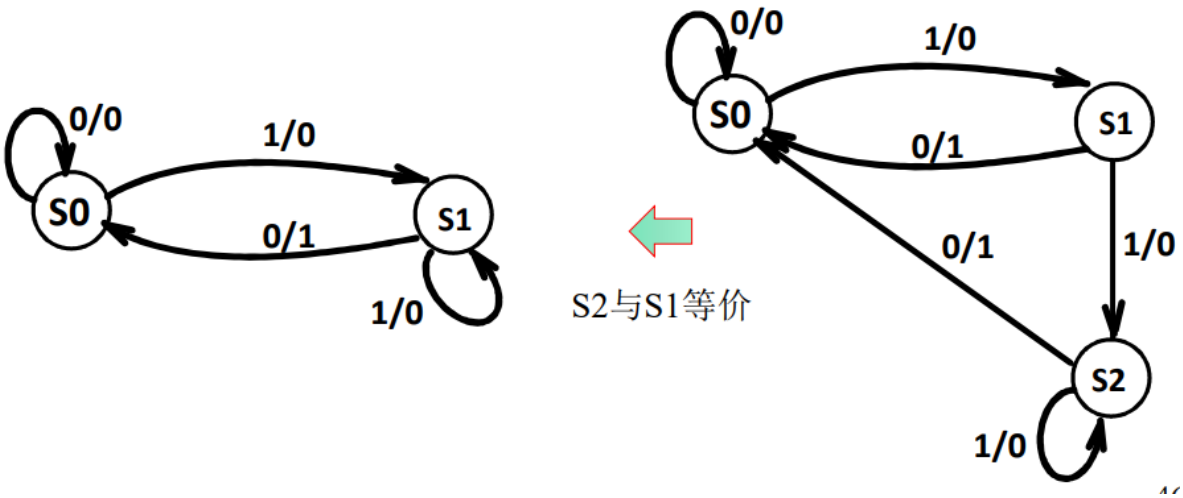
通过观察可以发现：Moore型输出**只依赖于状态**，Mealy型输出依赖于状态和输入

一般认为大型电路用状态表，小型电路用状态图更直观

等价状态

如果两个状态对每一个输入都产生相同的输出，并且下一状态也是一致的，那么这两个状态是等价的

彼此等价的两个状态可以**合并**为一个新状态

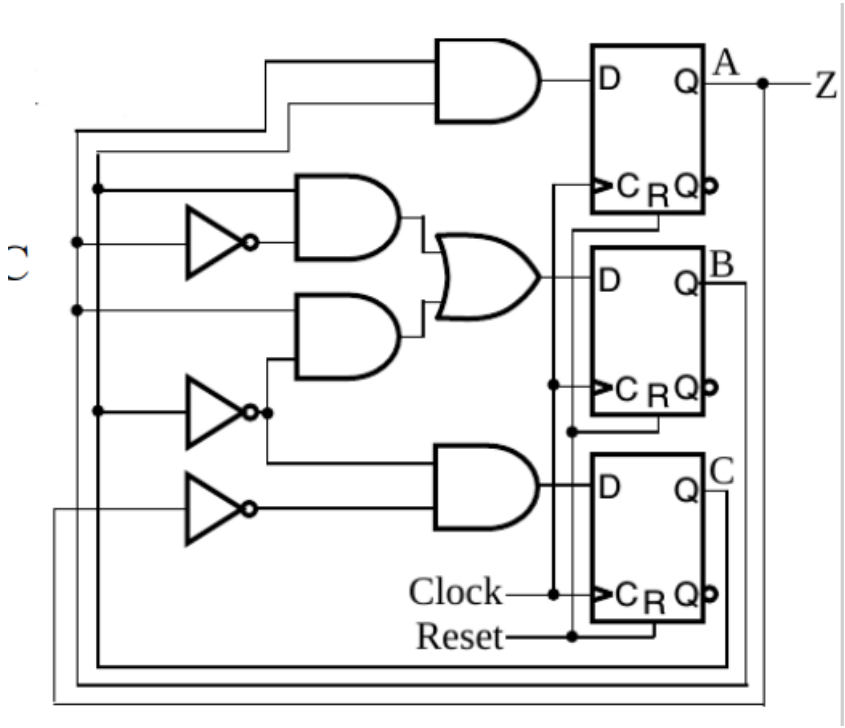


混合型

在实际设计中，一个电路中可能有些输出是Moore型的，有些是Mealy型。

两种类型可以相互转换

时序电路分析



易得：

- 输入：无
- 输出：Z
- 状态：A、B、C

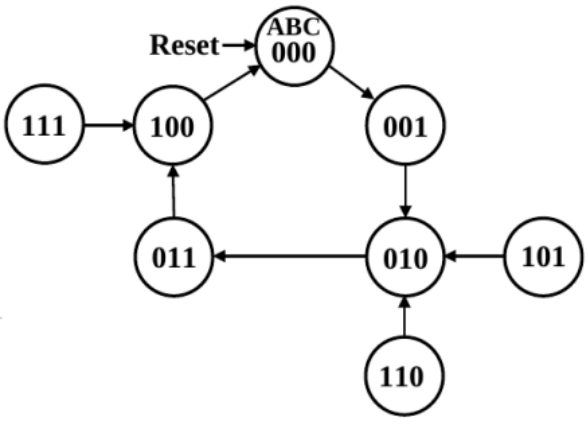
之后要去找状态函数和输出函数，即：

- A(t+1) = ?
- B(t+1) = ?
- C(t+1) = ?
- Z = ?

很容易能看出有Z = A

而 $D_A = BC$，$D_B = \overline{B}C + B\overline{C}$，$D_C = \overline{A}\overline{C}$，根据这个式子得出状态表和状态图

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



时序电路设计

设计过程

具体步骤：

1. 规范化：规格说明
2. 形式化：得到状态表或状态图
3. 状态分配：给存储单元的状态分配二进制码
4. 确定触发器的输入方程：
    - 选择触发器类型，并且根据状态表中的下一状态推导出触发器输入方程，也就是状态方程
5. 确定输出方程：
    - 根据状态表中的输出推导出输出方程
6. 优化：对方程进行优化
7. 工艺映射：
    - 从方程中得到电路并且映射到触发器和门工艺
8. 验证：验证最终设计的正确性

规范化

描述电路的时序行为

例：序列识别器 001101011101.....

- 一个时序电路, 其能在输入序列中发现目标序列时产生特定的输出，如识别一个输入序列1101的出现

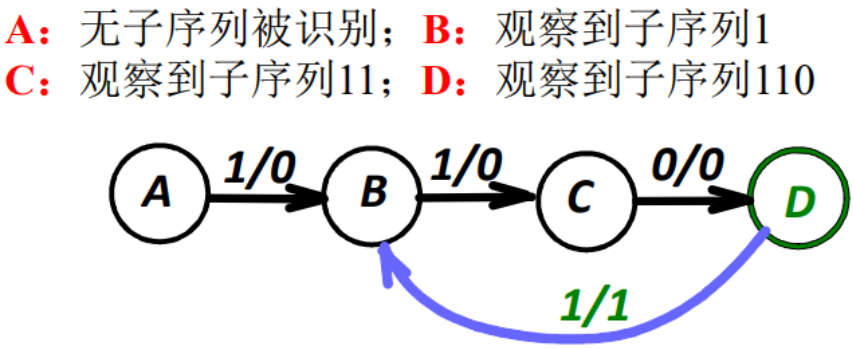
形式化

状态：一系列加载到电路的历史输入的抽象

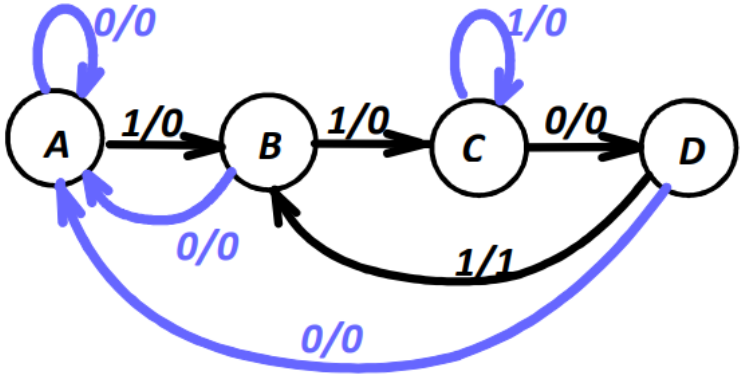
初始状态：提供一个硬件机制使电路从任何未知状态进入初始状态

根据状态分析获得状态图或状态表

仍以上面的序列识别器为例，其可以形式化出4个状态



发现对于一些输入，没有相应的转移弧，如状态A输入0。补充转移弧后：



得到了状态图之后，分析得到状态表：

Present State	Next State x=0	Next State x=1	Output x=0	Output x=1
A	A	B	0	0
B	A	C	0	0
C	D	C	0	0
D	A	B	0	1

Mealy型的转化和Moore型的转化并不相同，Moore型的状态更多

状态分配（状态赋值）

通过编码将形式化得到的所有状态分配**唯一**的代码

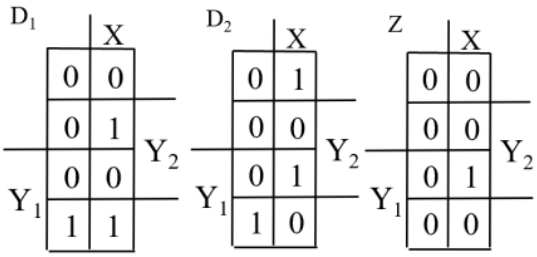
n个状态需要至少$log_2 n$位的编码

编码的方式一般有：计数编码、格雷编码、单热点编码等

确定触发器的状态方程、确定输出方程与优化

画出状态表与卡诺图：

Present State	Next State x = 0	Next State x = 1	Output x = 0	Output x = 1
0 0	0 0	0 1	0	0
0 1	0 0	1 0	0	0
1 0	1 1	1 0	0	0
1 1	0 0	0 1	0	1



74

然后得出状态方程：

$$D_1 = Y_1\overline{Y}_2 + X\overline{Y}_1Y_2$$
$$D_2 = \overline{X}Y_1\overline{Y}_2 + X\overline{Y}_1\overline{Y}_2 + XY_1Y_2$$
$$Z = XY_1Y_2$$ 门输入成本G=22

经过优化后，得到：

$$D_1 = Y_1Y_2 + XY_2$$
$$D_2 = X$$
$$Z = XY_1\overline{Y}_2$$ 门输入成本G=9

工艺映射

根据优化后的方程得到时序电路

验证

手工验证或模拟验证

状态机设计

有限状态机(FSM)包含：三个集合I,O,S和两个函数f和g

- I，O，S分别是输入、输出和状态集合
- f是下一状态函数f(I,S)，g是输出函数f(S)[Moore]或者f(I,S)[Mealy]

状态图和状态表都可以用于表示FSM，但是难以处理具有大量输入和输出的电路

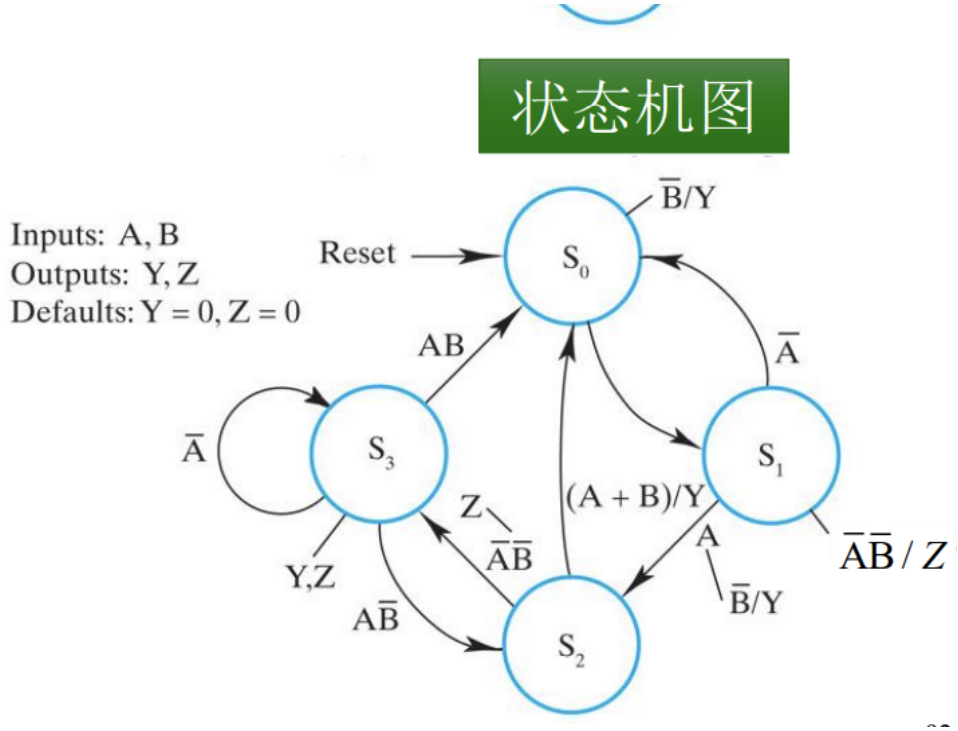
当输入输出过多时，采用 状态机图 (SMD)

状态机图模型

增加了在状态上定义Mealy输出的标识（直接连到状态节点上）

基于**输入条件、转移条件、输出条件**产生输出行为：

- 输入条件：采用输入变量的布尔表达式或方程来表示，其值为 0 或 1
- 转移条件(TC)： 在转移弧上 的输入条件
- 输出条件(OC)：如果其等于1，则造成输出行为发生；否则，输出行为不发生

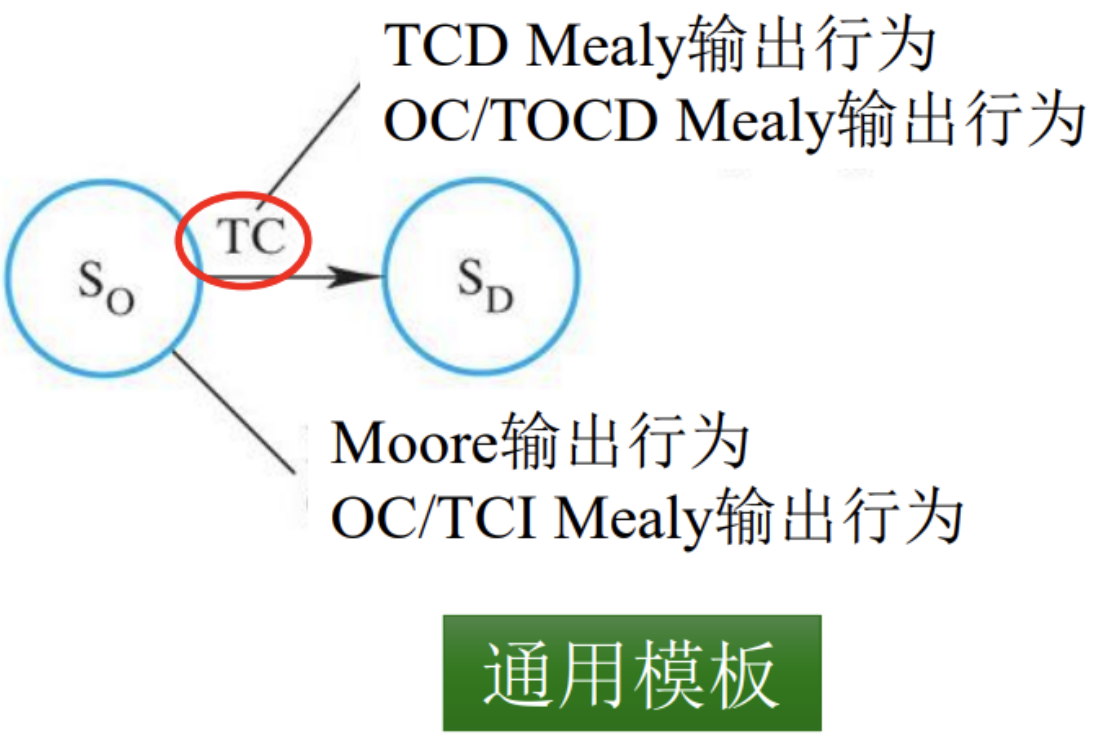


比如在上图中，S_1到S_0的弧上的$\overline{A}$即为转移条件,满足$\overline{A}=1$时条件转移

具体输出行为：

- Moore输出行为： 无条件 ，只依赖于状态。如上图连接在S_3的Y,Z，输出$Y,Z=1$
- 非转移条件依赖(TCI)：Mealy输出行为由 输出条件 和斜线驱动，和相应 状态 连接。当输出条件为1时输出行为发生。如上图连接在S_1的$\overline{AB}/Z$，若$\overline{AB}=1$，则产生输出Z=1
- 转移条件依赖(TCD)：Mealy输出行为和相应的 状态转移 连接，如果转移条件为1，则输出行为发生。如上图连接在S_2到S_3的转移条件上的Z，若$\overline{AB}=1$，则有输出$Z=1$

- 转移和输出条件依赖(TOCD)：Mealy输出行为由输出条件和斜线驱动，与相应的转移状态相连接。转移条件和输出条件 都为1 ，则输出行为发生。如上图连接在S_1到S_2转移条件上的$\overline{B}/Y$，若$A=1$且$\overline{B}=1$，则有输出$Y=1$



输出行为在以下情况发生：

- 无条件(Moore)
- TCI并且输出条件(OC)为1
- TCD并且转移条件(TC)为1
- TOCD并且TC和OC为1

约束检查

约束检测

- TC约束
  1. 对状态S_i，从S_i出发的所有可能TC对(T_{ij},T_{ik})，有$T_{ij}\cdot T_{ik}=0$

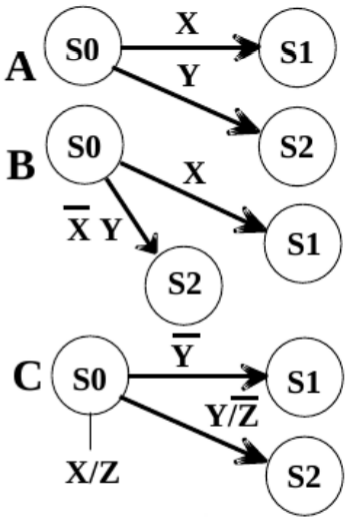
     状态不能同时向多个状态转移
  2. 对状态S_i，对所有的可能的TC，T_{ij}，有$\sum T_{ij}=1$

     状态总要满足一个状态转移条件（即使是转移到自身）
- OC约束
  1. 对状态S_i，在其上或者其状态转移上有一致的输出变量但不同值的输出行为，相应的输出条件对(O_{ij},O_{ik})是互斥的，即$O_{ij}\cdot O_{ik}=0$

     肯定不能同时输出一个变量的两个值
  2. 对每个输出变量，在状态S_i上或者在状态S_i转移上的输出条件必须**覆盖所有可能的输入变量组合**，即$\sum O_{ij=1}$

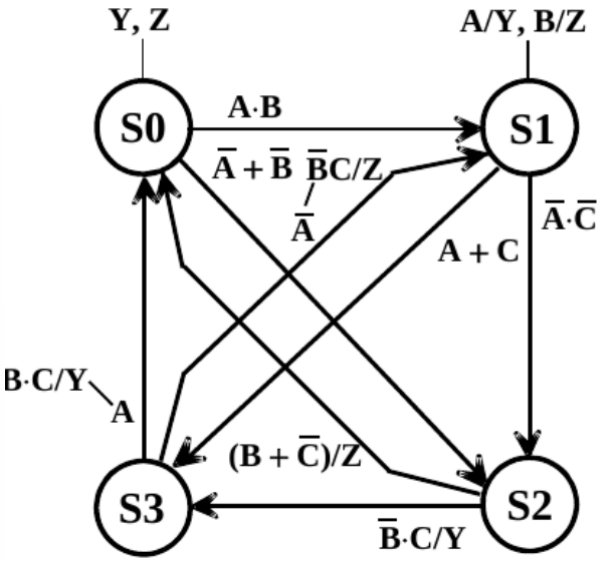
违反约束

- 转移约束
  - 例 A: $X \cdot Y \neq 0$ 且 $X + Y \neq 1$, 因此两个约束都违反
  - 例 B: $X \cdot \overline{X}Y = 0$, 但 $X + \overline{X}Y \neq 1$. 违反约束2
- 输出约束
  - 例 C: 对 $Z = 1$ 和 $Z = 0$, $X \cdot Y \neq 0$, 因此违反约束1
  - 约束$X + Y + \overline{Y} = 1$, 由于在$\overline{Y}$上输出默认值Z， 因此约束2 满足。



约束检查例子

Defaults: Y = 0, Z = 0



- 转移约束:
  - S_0: $(A \cdot B) \cdot (\overline{A} + \overline{B}) = 0$;$(A \cdot B) + (\overline{A} + \overline{B}) = 1$
  - S_1:
  - S_2:
  - S_3:
- 输出约束:

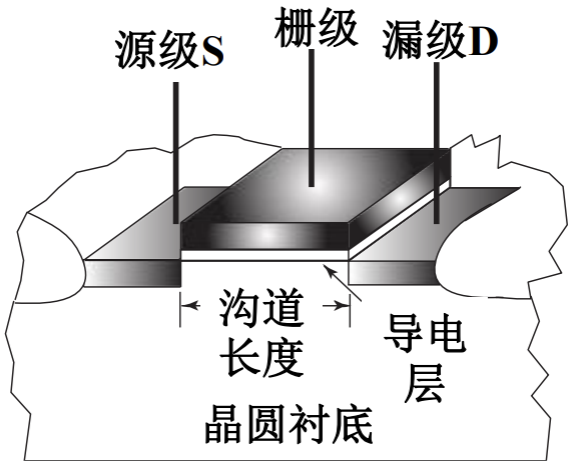
状态机图应用和设计

与时序电路的设计过程基本一致

Chapter 5 数字硬件实现——寄存器

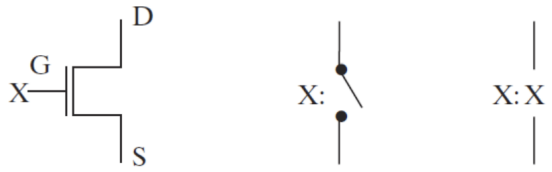
CMOS电路

工艺基础：MOS晶体管

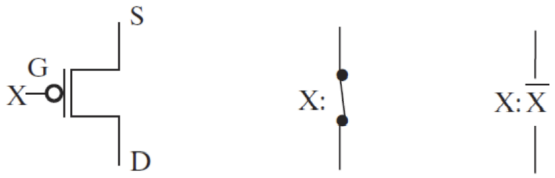


有两种模型，n沟道和p沟道:

- n沟道：触点习惯上为常开

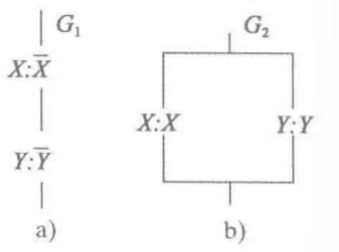


- p沟道：源级和漏级位置互换，触点习惯上为常闭



开关电路

CMOS电路就是用模拟晶体管的开关实现函数F, 当$F=1$时电路要有一个通路, $F=0$时电路没有通路



设计空间

电路设计是将逻辑门实现的逻辑电路映射到晶体管实现的电子电路，是从逻辑级到电路级的转换过程

相关参数：

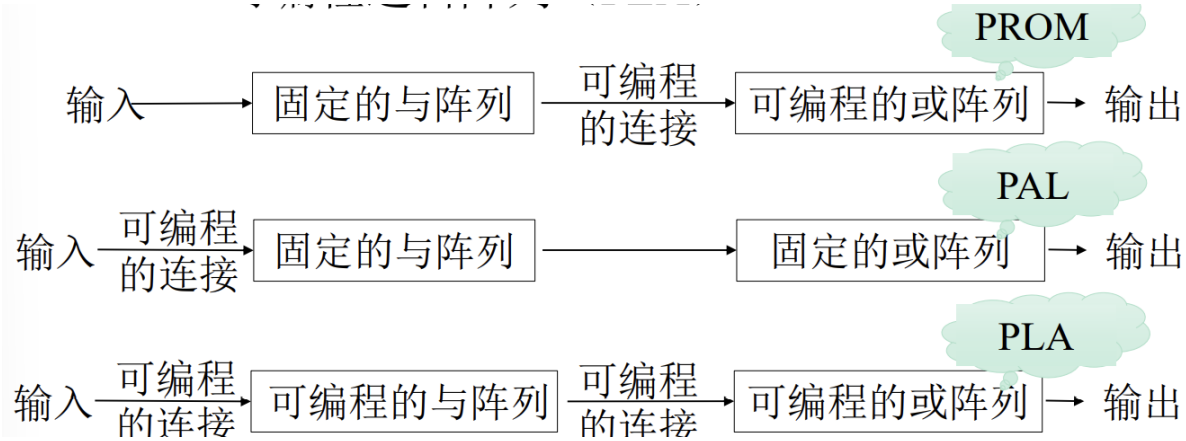
- 扇入：一个门可能的输入数
- 扇出：一个门输出驱动的标准负载数
- 成本：晶体管大小、数目等因素

可编程逻辑器件

可编程逻辑器件（PLD）包含了用来实现逻辑功能的结构和控制内部连接或存储所需的信息

- 只读存储器(ROM)
- 可编程逻辑阵列(PLA)
- 可编程阵列逻辑(PAL)
- 现场可编程门阵列(FPGA)

这些可编程器件的不同之处在于与-或阵列的可编程位置：

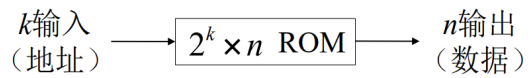


可编程实现技术

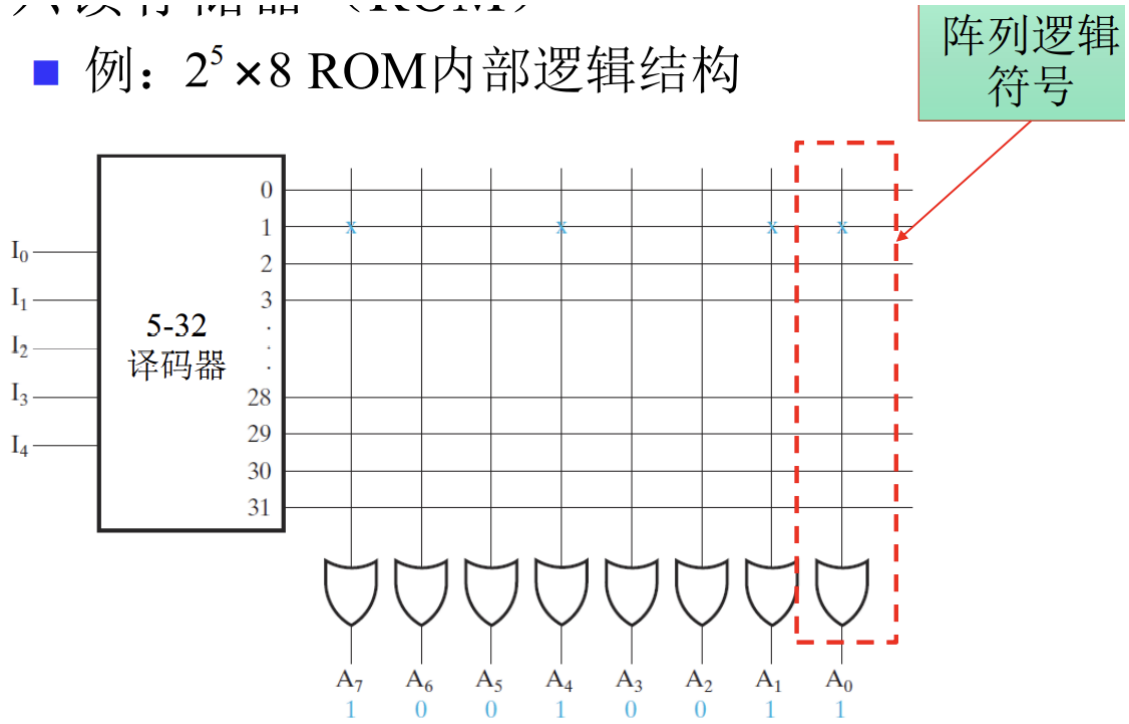
- 永久性固化编程：熔丝、反熔丝、掩膜编程
- 可重编程：编程点的存储单元、晶体管开关

只读存储器ROM

是非易失的存储二进制信息的器件

输入提供地址，输出提供定制选定的存储字的数据位： 

实例：

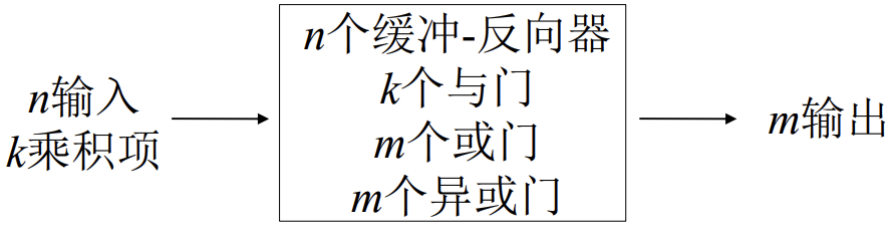


对于一个$2^k \times n$的ROM, 包括一个$k-2^k$译码器和n个或门, 每个或门有2^k个输入,他们通过可编程连接点连接到译码器的每个输出

上面ROM中译码器的每一个输出都代表一个**存储地址**, 该地址上存储有数据, 根据那些数据，编程决定他们是否与输出的或门直接连接, 就能最终将这些数据输出

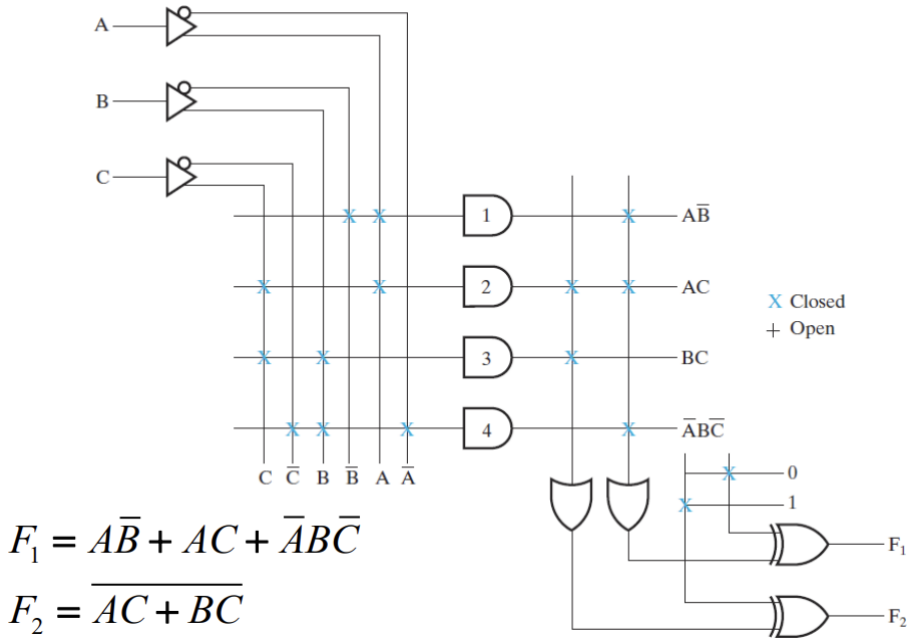
可编程逻辑阵列PLA

相比ROM, PLA使用**与门阵列**代替译码器，通过**编程**产生输入变量的乘积项, 然后把这些乘积项可选的连到或门以便生成 **布尔表达式的积之和**



- 输入和与门之间由$2n \times k$可编程连接点, 经过与门之后产生所有需要的乘积项
- 与门和或门之间有$k \times m$可编程连接点, 经过或门之后产生布尔表达式
- 与门和异或门之间有m可编程连接点, 有时得到的布尔表达式需要取反, 就用异或门

- 例：3个输入、2个输出的内部逻辑图



$$F_1 = A\bar{B} + AC + \bar{A}B\bar{C}$$

$$F_2 = \overline{AC + BC}$$

使用PLA设计电路，要尽可能地复用乘积项，以减小电路的规模

可编程阵列逻辑PAL

或门阵列固定，与门阵列可编程

通过与门得到最小项然后或门将最小项相加，最终得到积之和

- 可编程阵列逻辑器件（PAL）
  - 例：

$$W = \sum m(2,12,13)$$

$$X = \sum m(7,8,9,10,11,12,13,14,15)$$

$$Y = \sum m(0,2,3,4,5,6,7,8,10,11,15)$$

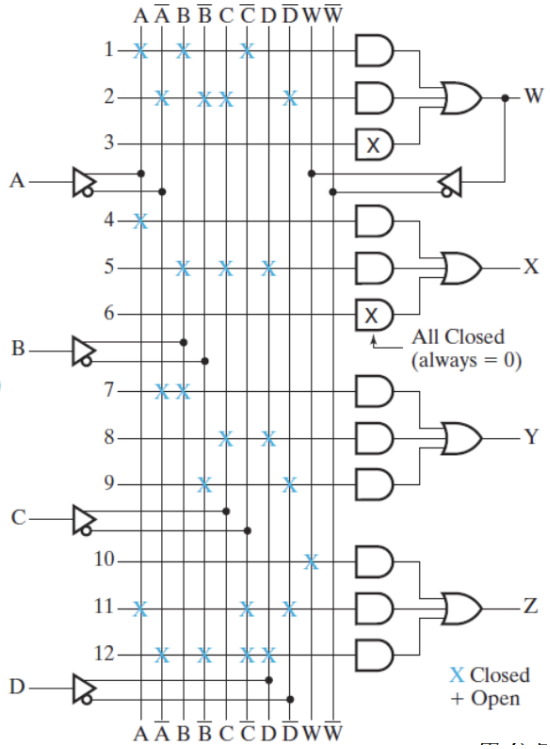
$$Z = \sum m(1,2,8,12,13)$$

$$W = AB\bar{C} + \bar{A}\bar{B}CD$$

$$X = A + BCD$$

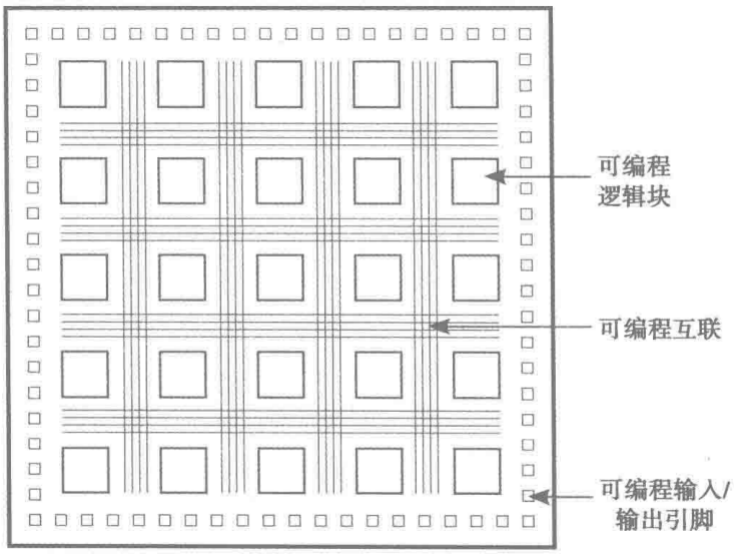
$$Y = \bar{A}B + CD + \bar{B}\bar{C}$$

$$Z = W + A\bar{C}\bar{D} + \bar{A}B\bar{C}D$$



现场可编程门阵列FPGA

由**可编程逻辑块**、**可编程互联**和**可编程输入/输出引脚**等三个可编程部件组成



可编程逻辑块

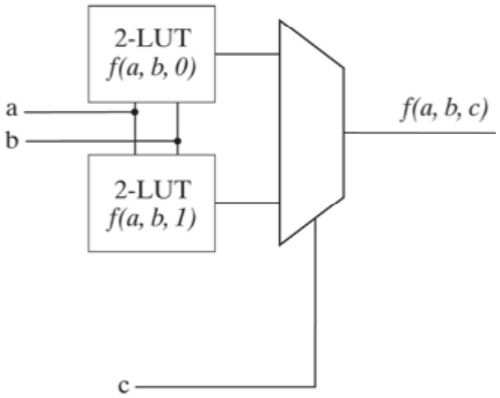
采用 查找表 实现组合逻辑函数, 一个查找表是一个$2^k \times 1$的 存储器 , 记录带有k个变量的函数的真值表, 称之为 k-LUT . 用多路复用器将多个k-LUT连接起来, 就能实现多于k个变量的函数

如何连接? 根据香农展开式定理: $f(x_1, x_2, \ldots, x_n) = x_n f(x_1, x_2, \ldots, 1) + \overline{x}_n f(x_1, x_2, \ldots, 0)$

比如对于一个三输入的函数$f(a, b, c)$, 可以用两个2-LUT实现$f(a, b, 0)$和$f(a, b, 1)$, 再连接c实现，例如:

用多路复用器实现布尔函数

$$F(A, B, C) = \sum m(3, 5, 6, 7)$$



当$C = 1$时最小项为m_3, m_5, m_7，所以得：$F(a, b, 1) = \overline{A}B + A\overline{B} + AB = A + B$; 当$C = 0$时，最小项为$m_6$，所以得：$F(a, b, 0) = AB$。

得到两个函数之后，在两个2-LUT电路相应的配置位中存放两个函数的真值表即可