数字逻辑

第四章 时序逻辑电路分析与设计

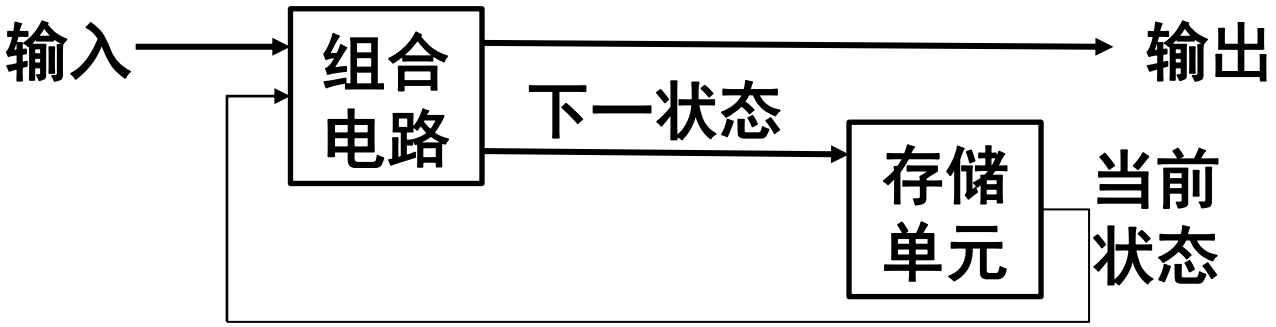
北京理工大学

计算机学院

黄永刚

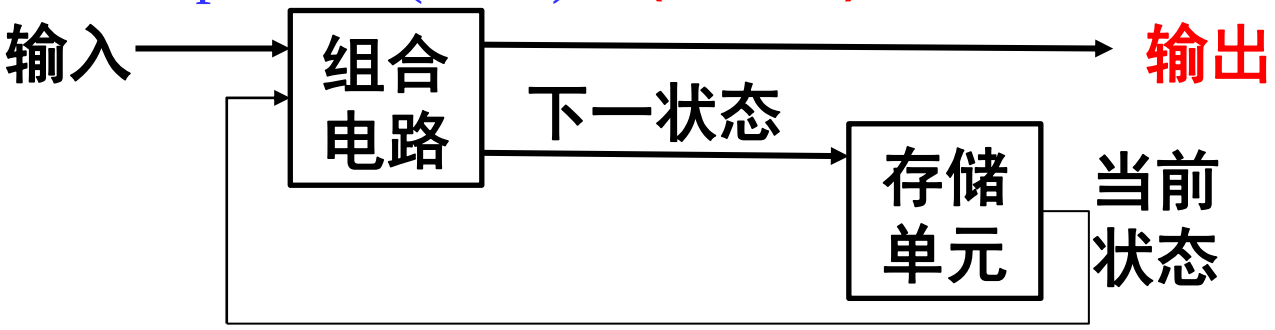
时序电路

- 输出，不仅取决于当前输入，且与当前状态有关
- 构成
  - 存储单元：实现状态存储的电路
  - 组合逻辑：转换函数



- 存储单元
  - 锁存器/触发器: 基于电路延时

- 组合电路
  - 下一状态函数
    - Next State = f(Inputs, State)
  - 输出函数
    - Outputs = g(Inputs, State) （Mealy）
    - Outputs = h(State) （Moore）



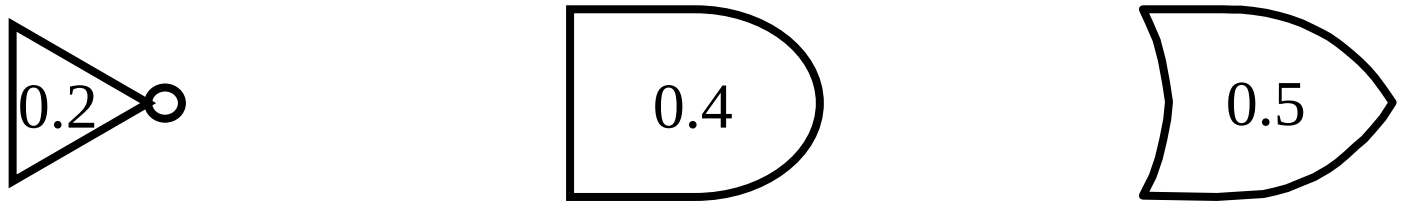
- 时序电路类型
  - 同步：全局时钟，存储单元状态改变只发生在全局时钟到来时
  - 异步：任何时刻可以改变存储单元状态
- 另一视角
  - 将时钟看作一个输入，则所有电路异步
  - 但，同步电路更易分析和设计

一. 存储单元

- 1. 状态存储
- 2. 锁存器
- 3. 触发器
- 4. 标准符号

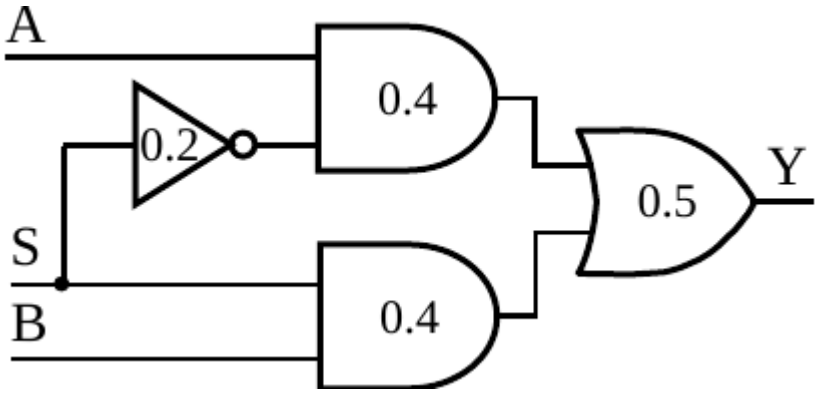
1. 状态存储

- 如何让电路能够存储状态?
  - 利用电路延时
- 门延时模型
  - 设门延时为 0.2 ns, 0.4 ns, 0.5 ns
  - 可表示为

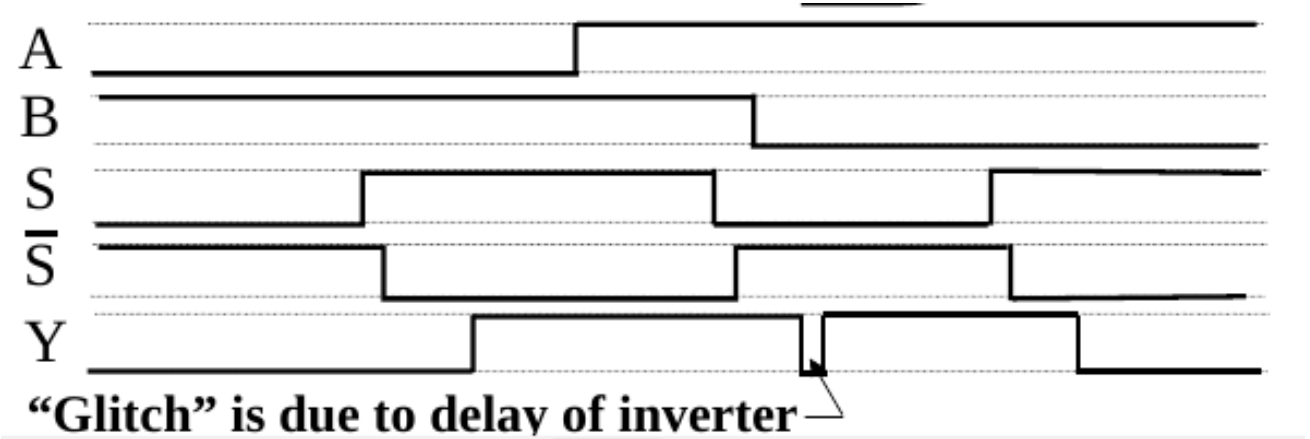


1. 状态存储

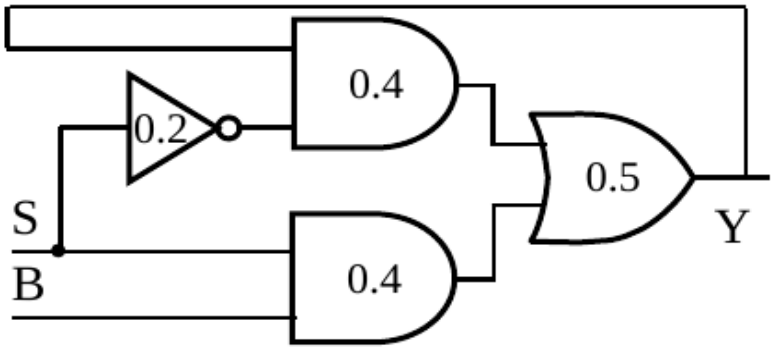
- 2 -1 多路复用器
- 电路函数
  - Y = A for S = 0
  - Y = B for S = 1



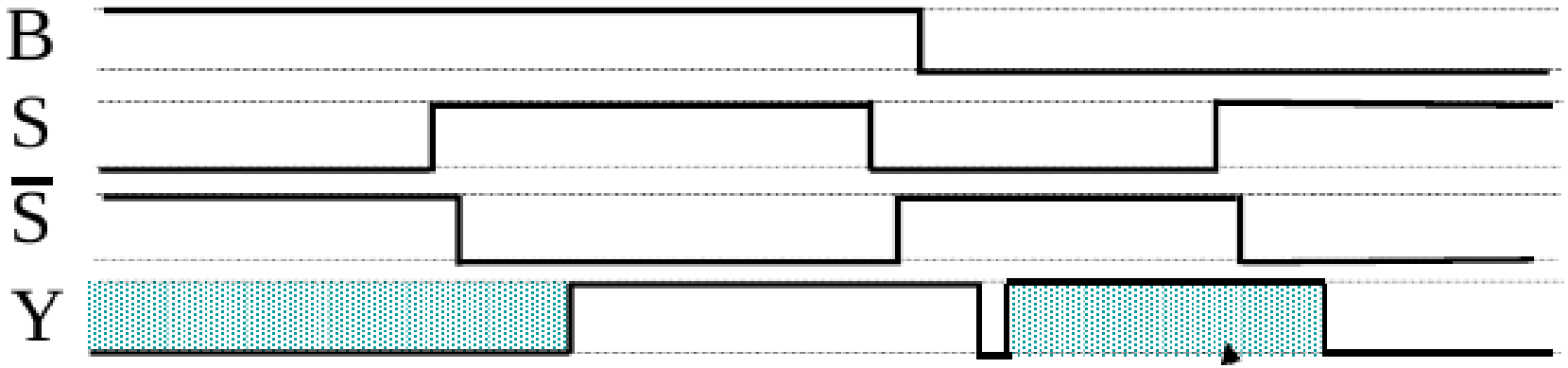
- S变为1： A1.1ns 消失，B 0.9 ns到达
- S变为0： A1.1ns 到达，B 0.9 ns消失
- 毛刺: 反相器延迟导致两个与门都关闭(或打开)



1. 状态存储

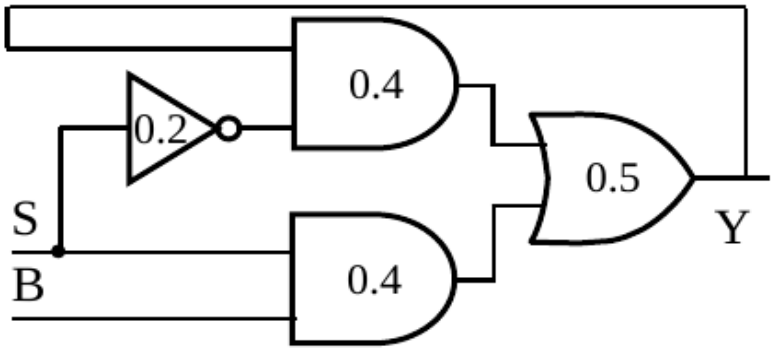


- A直连Y
- 电路函数
  - Y = B for S = 1
  - Y(t) =Y(t – 0.9) for S = 0
- 此时电路变成时序电路
  - 因为输出是输入信号时序函数



1. 状态存储

- 输入信号每100ns改变一次
  - 100ns vs 0.5ns
  - 门延时可忽略
- Y 不仅是输出，更是状态

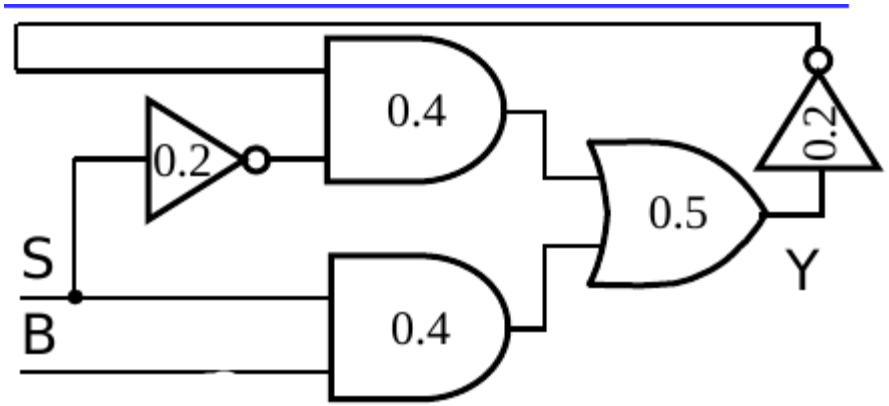


Time ↓

B	S	Y	Comment
1	0	0	Y “remembers” 0
1	1	1	Y = B when S = 1
1	0	1	Now Y “remembers” B = 1 for S = 0
0	0	1	No change in Y when B changes
0	1	0	Y = B when S = 1
0	0	0	Y “remembers” B = 0 for S = 0
1	0	0	No change in Y when B changes

1. 状态存储

- 加入1个反相器
- S=0时
  - 振荡器
  - 可作为原始时钟



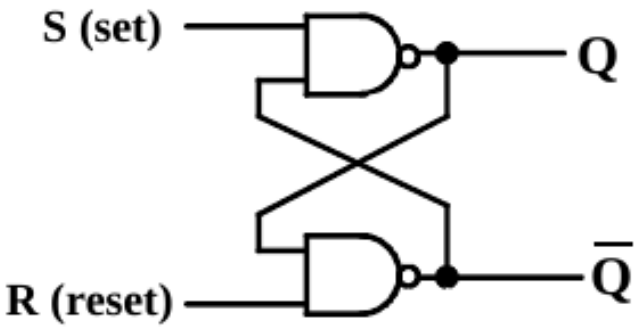
B	S	Y	Comment
0	1	0	Y = B when S = 1
1	1	1	
1	0	1	Now Y “remembers” A
1	0	0	Y, 1.1 ns later
1	0	1	Y, 1.1 ns later
1	0	0	Y, 1.1 ns later

2. 锁存器

- $\bar{S}$-$\bar{R}$锁存器
- S-R锁存器
- 时钟S-R锁存器
- D锁存器

2. 锁存器

- $\bar{S}$-$\bar{R}$锁存器
  - 交叉耦合的两个与非门
  - 一半为0，一半为1
- 名称: Set / Reset
  - 取反：0有效
- S=1, R=1 保持
- S=0, R=1 置位
- S=1, R=0 复位
- S=0, R=0 禁止
  - 两输出为1，不合逻辑
  - 回到S=1, R=1, S, R不完全同步, 结果不可预测

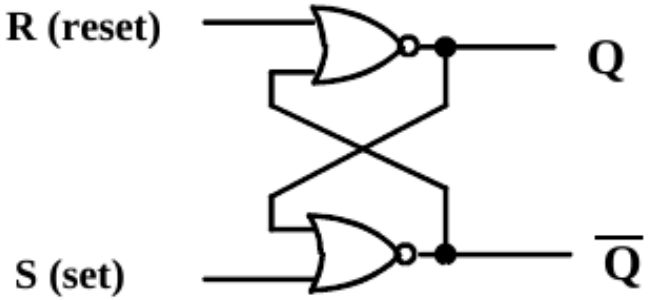


Time

R	S	Q	$\bar{Q}$	Comment
1	1	?	?	Stored state unknown
1	0	1	0	“Set” Q to 1
1	1	1	0	Now Q “remembers” 1
0	1	0	1	“Reset” Q to 0
1	1	0	1	Now Q “remembers” 0
0	0	1	1	Both go high
1	1	?	?	Unstable!

2. 锁存器

- S-R锁存器
  - 交叉耦合的两个或非门
  - 一半为0，一半为1
- 名称: Set / Reset
  - 取反：1有效
- S=0, R=0 保持
- S=1, R=0 置位
- S=0, R=1 复位
- S=1, R=1 禁止
  - 两输出为0，不合逻辑
  - 回到S=0, R=0, S, R不完全同步, 结果不可预测

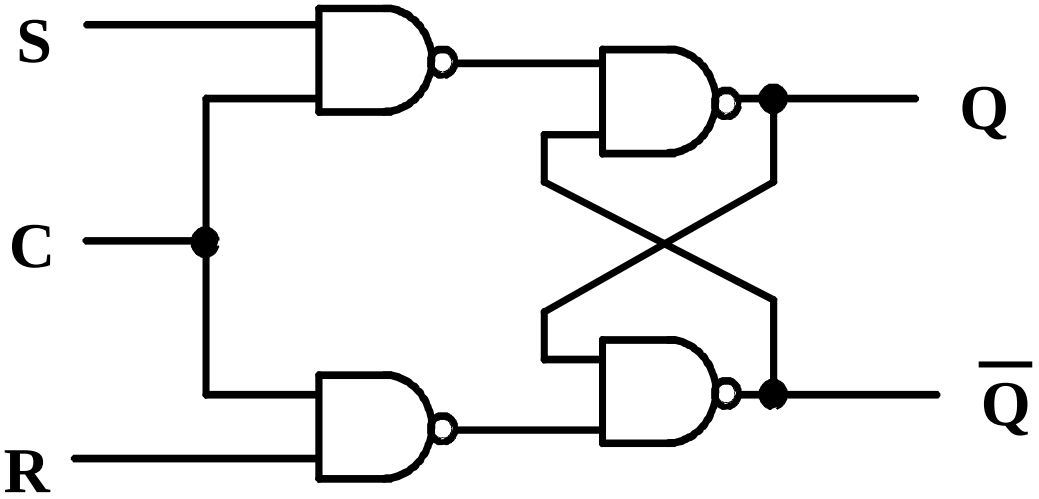


Time

R	S	Q	$\overline{Q}$	Comment
0	0	?	?	Stored state unknown
0	1	1	0	"Set" Q to 1
0	0	1	0	Now Q "remembers" 1
1	0	0	1	"Reset" Q to 0
0	0	0	1	Now Q "remembers" 0
1	1	0	0	Both go low
0	0	?	?	Unstable!

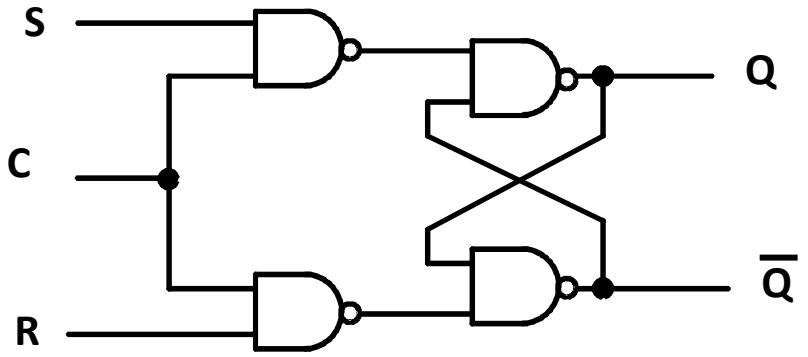
2. 锁存器

- 时钟S-R锁存器
  - 向$\overline{S}$-$\overline{R}$锁存器添加时钟使能
  - 当C=1时, S和R才有效
  - C: Control，Clock



2. 锁存器

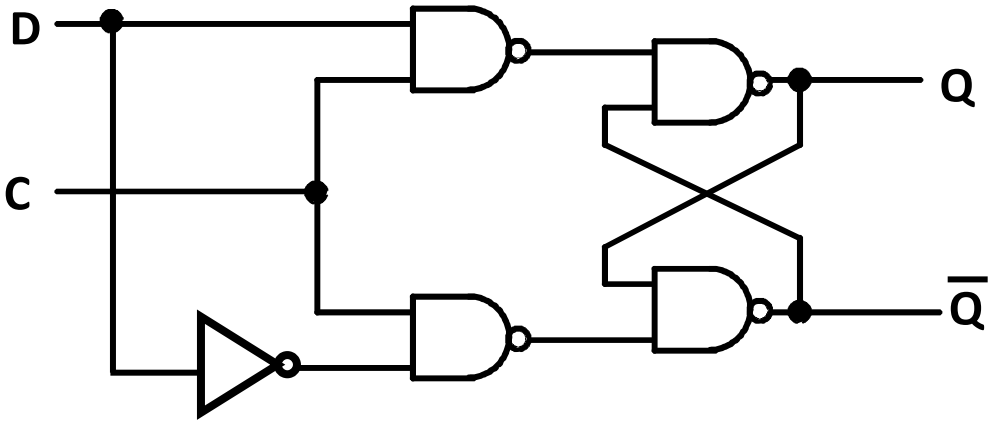
- C=1 时
  - S-R锁存器
- C=0 时
  - 保持状态
- 时序行为
  - 基于 t 时
    - S, R
    - Q(t)
  - t+1: 下个时钟
    - Q(t+1)



Q(t)	S	R	Q(t+1)	Comment
0	0	0	0	No change
0	0	1	0	Clear Q
0	1	0	1	Set Q
0	1	1	???	Indeterminate
1	0	0	1	No change
1	0	1	0	Clear Q
1	1	0	1	Set Q
1	1	1	???	Indeterminate

2. 锁存器

- D锁存器
  - 时钟S-R锁存器加反相器
- 名称意义
  - Data
- C=1时
  - D=1: S=1, R=0
  - D=0: S=0, R=1
- C=0 时
  - S=0，R=0
- 时序行为



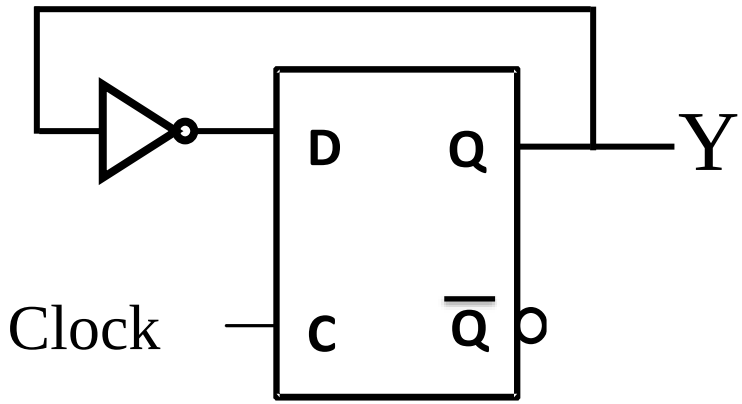
Q	D	Q(t+1)	Comment
0	0	0	No change
0	1	1	Set Q
1	0	0	Clear Q
1	1	1	No Change

3. 触发器

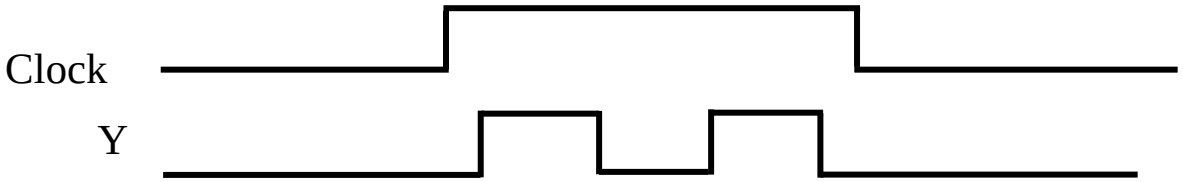
- 锁存器缺陷
- 主从触发器
- 边沿触发器
- 直接输入

3. 触发器

- 锁存器缺陷



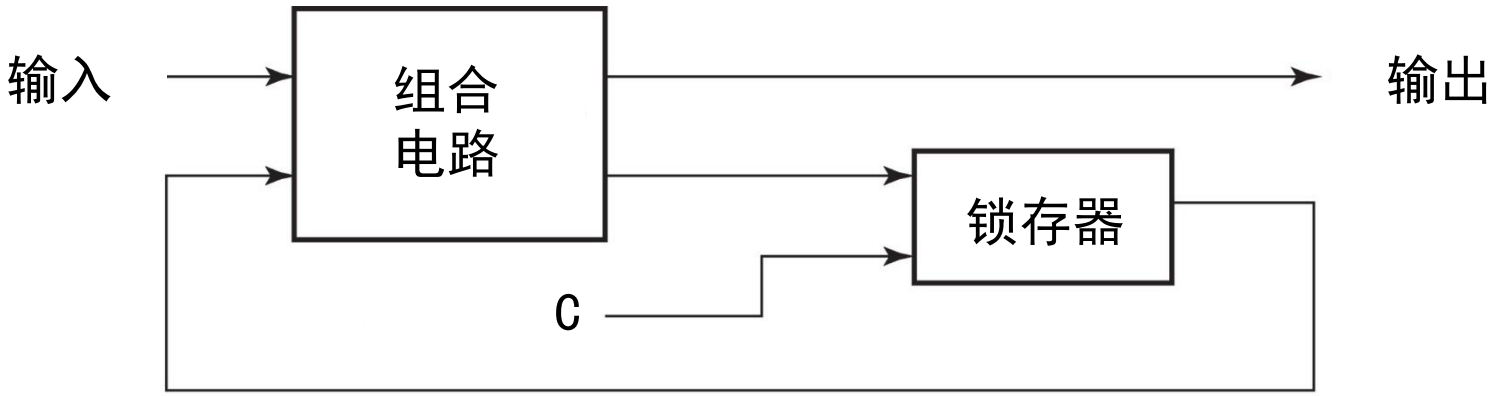
- 初始Y=0, 则



- 问题：C=1时，Y持续改变，最终状态不确定
- 正确：1次时钟，Y改变1次
- 原因：Y到Y回路的延迟

3. 触发器

- 一般情况，存在通过组合电路的回路
  - 1个锁存器到另外1个锁存器
  - 1个锁存器到自身
- C=1时，输入改变，引起锁存器输出持续改变

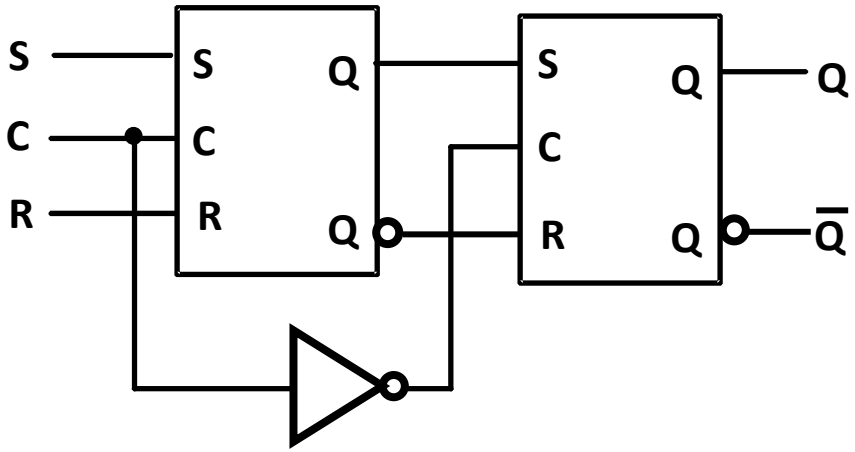


3. 触发器

- 怎么办?
  - 切断Y到Y的路径
  - 即采用触发器取代锁存器

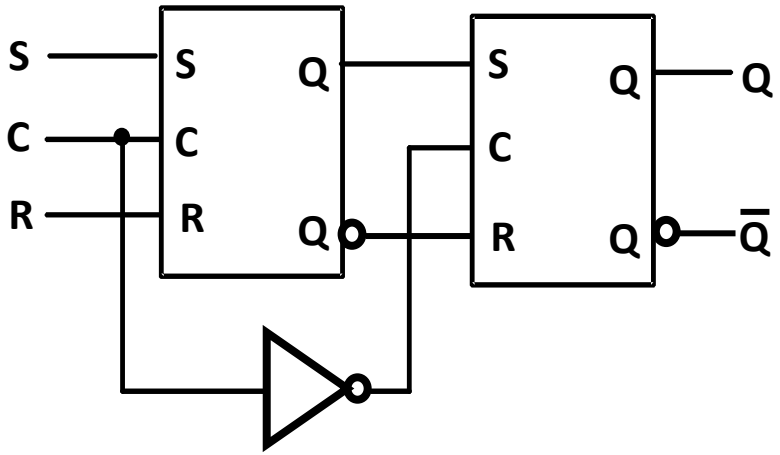
3. 触发器

- 主从S-R触发器
  - 包含两个时钟S-R锁存器
  - 左边，主；右边，从
  - 第二个锁存器的时钟反相
- 回路被时钟的不同值切断
  - 只有1个锁存器有效



3. 触发器

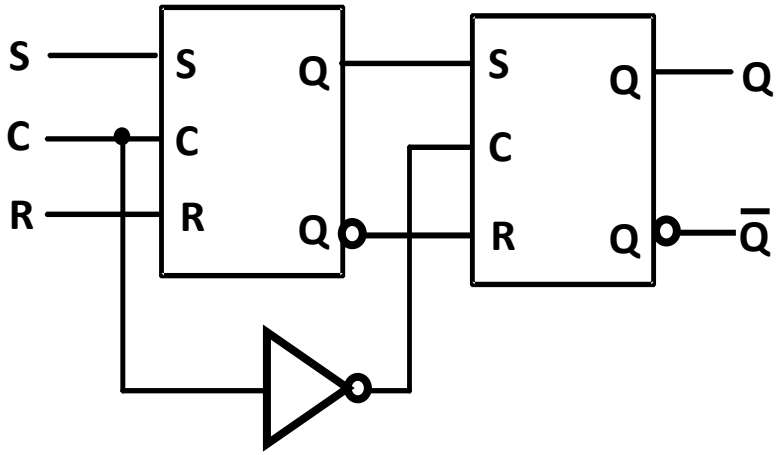
- 主从S-R触发器



- 又叫脉冲触发器
  - 有脉冲时，主锁存器状态改变
  - 无脉冲时，从锁存器状态改变

3. 触发器

- 脉冲触发器问题
- 易受干扰信号影响
- 例子
- 初始状态为0
- 无干扰时
  - 有脉冲时：S=0, R=0
  - 无脉冲时：Q=0
- 有干扰时
  - 有脉冲时：S=0, R=0；S=1 (尖峰干扰)；S=0
  - 无脉冲时：Q=1

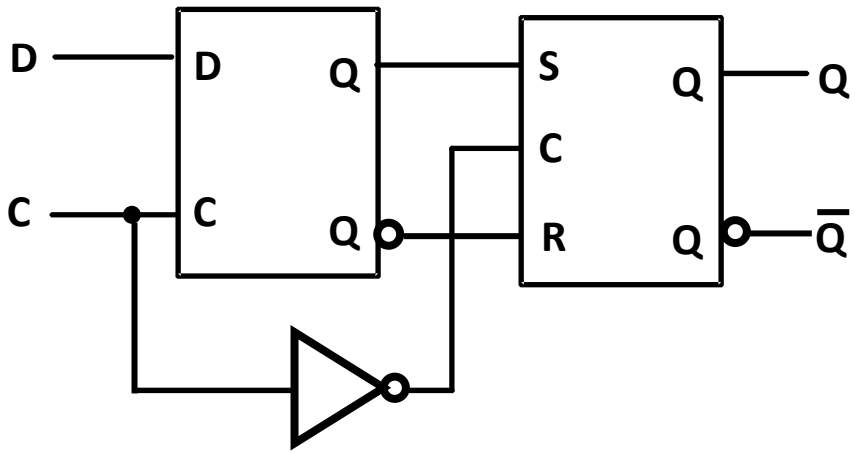


3. 触发器

- 怎么办?
- 采用边沿触发代替脉冲触发

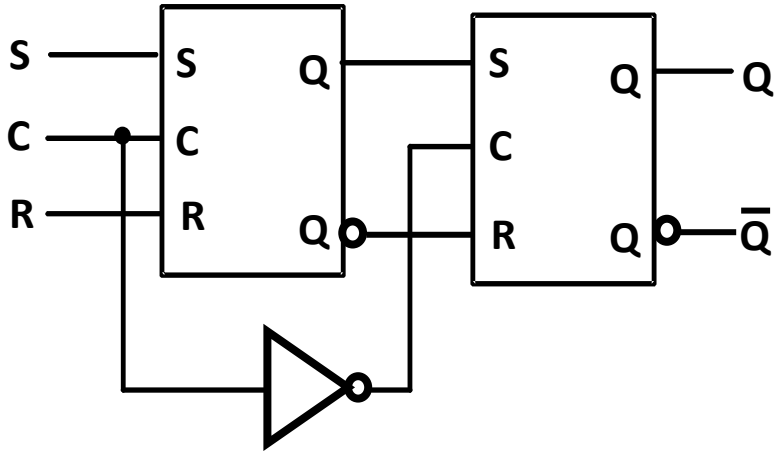
3. 触发器

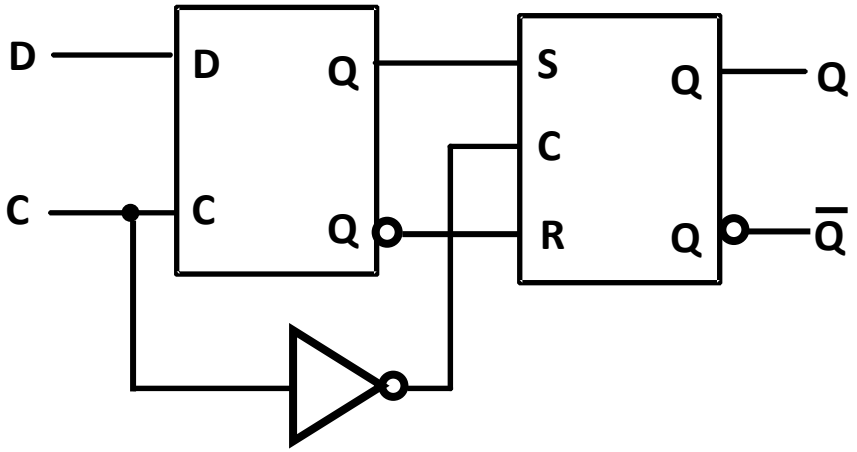
- 边沿D触发器
  - 将主从S-R的主锁存换成D锁存
  - 输出改变由负边沿触发
  - 称为负边沿D触发器



3. 触发器

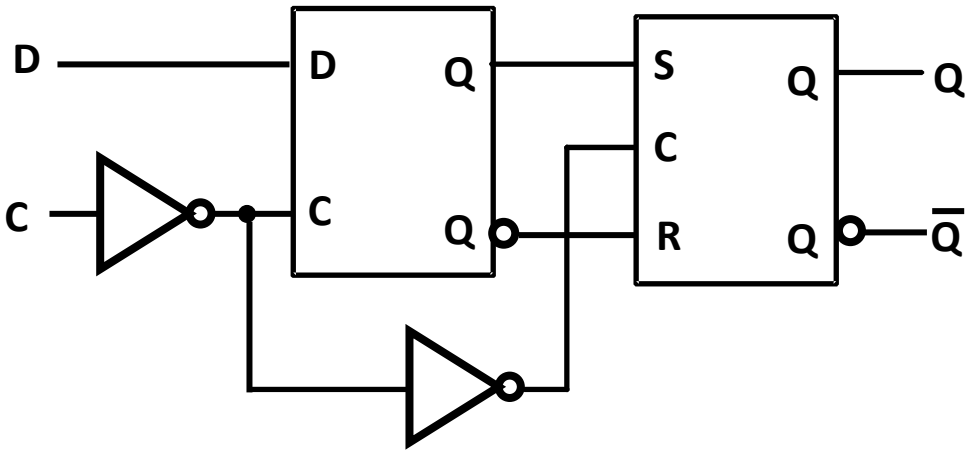
- 主从S-R触发器 vs 边沿D触发器
  - 主从S-R触发器：状态由脉冲有效的时间段决定
  - 边沿D触发器：状态由边沿的一瞬间决定





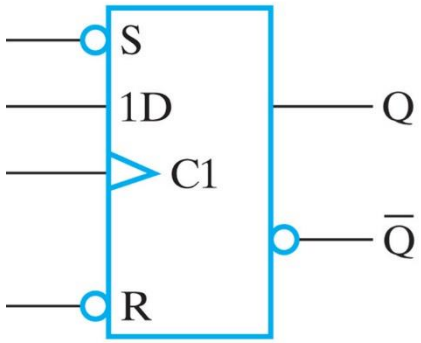
3. 触发器

- 正边沿D触发器
  - 给时钟加反相器
  - Q 在正边沿改变
  - 标准触发器



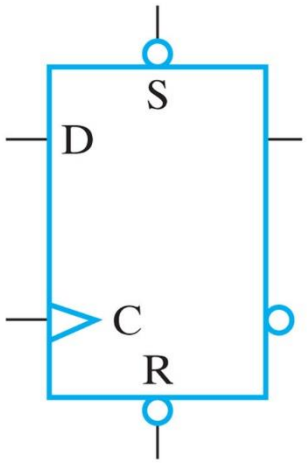
3. 触发器

- 直接输入
  - 当启动或Reset,时序电路全部或部分需初始化
  - 独立于时钟，异步输入
  - 1表示依赖关系
  - 具备直接置位和复位功能的正边沿D触发器



标准图形符号

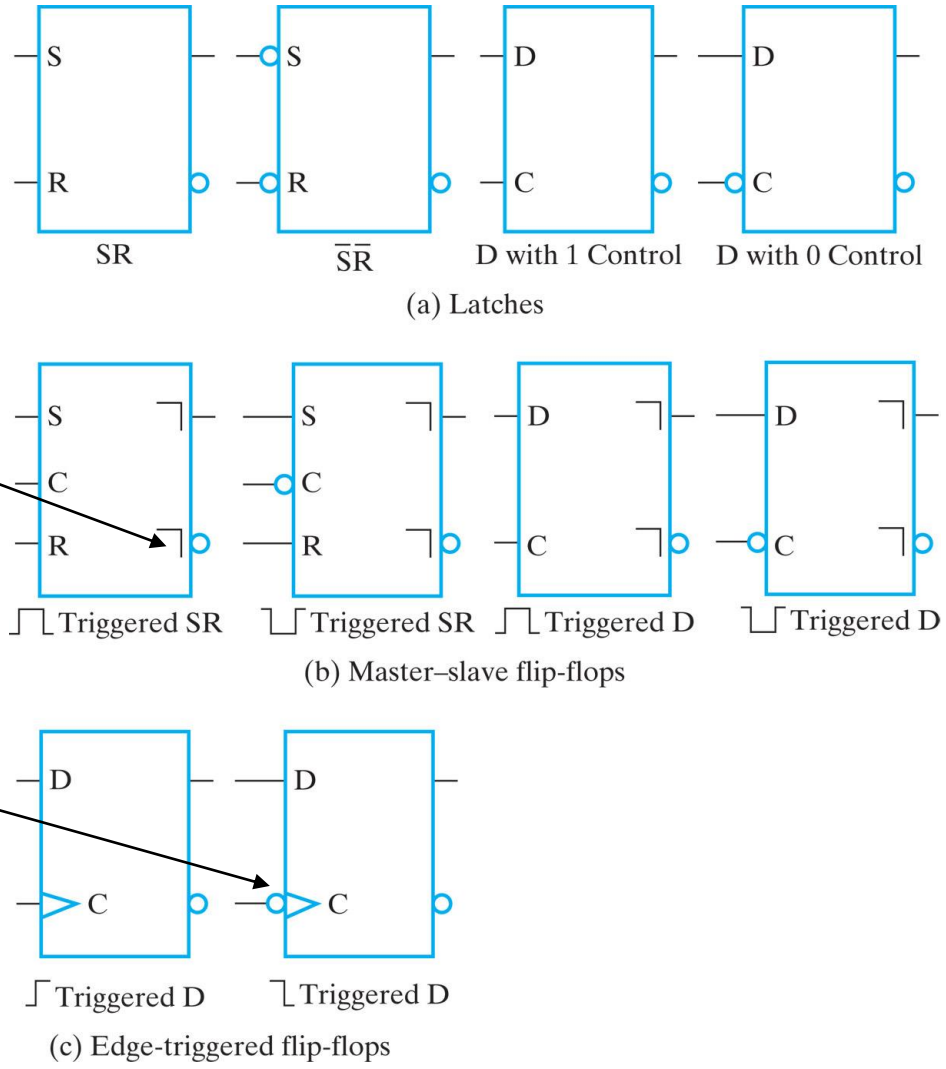
S	R	C	D	Q	$\overline{Q}$
0	1	X	X	1	0
1	0	X	X	0	1
0	0	X	X	Undefined	
1	1	↑	0	0	1
1	1	↑	1	1	0



简化的符号

4. 标准符号

- 主从
  - 延迟输出指示
  - 输出脉冲结尾改变
- 边沿触发
  - 动态输入指示
  - 响应输入脉冲边沿

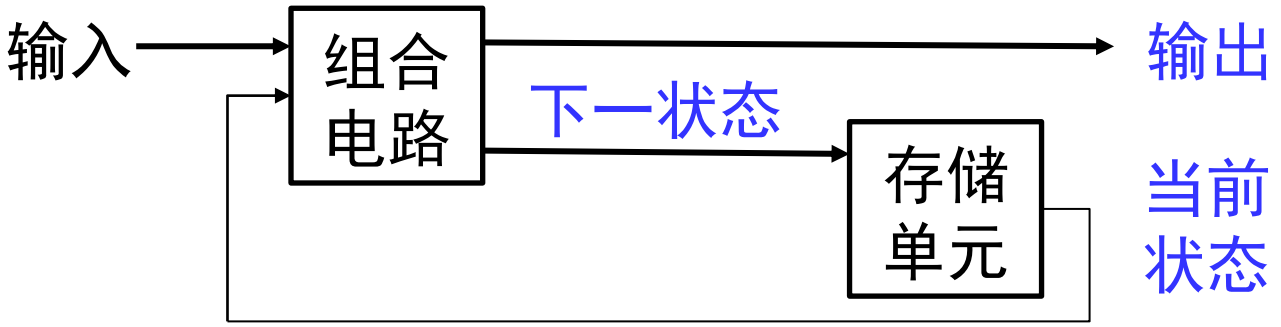


二. 时序电路分析

- 1. 方程
- 2. 状态表
- 3. 状态图
- 4. 等价状态
- 5. 输出类型

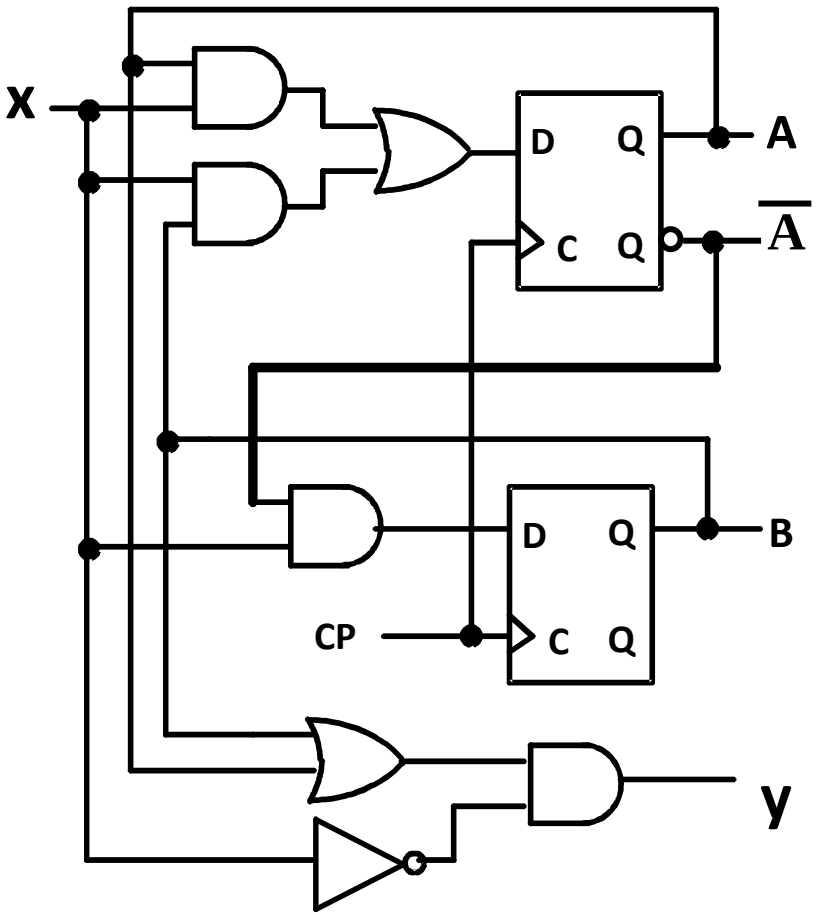
一般模型

- 状态：存储单元
- 下一状态函数
  - Next State = f(Inputs, State)
- 输出函数
  - Outputs = g(Inputs, State) （Mealy）
  - Outputs = h(State) （Moore）



1. 方程

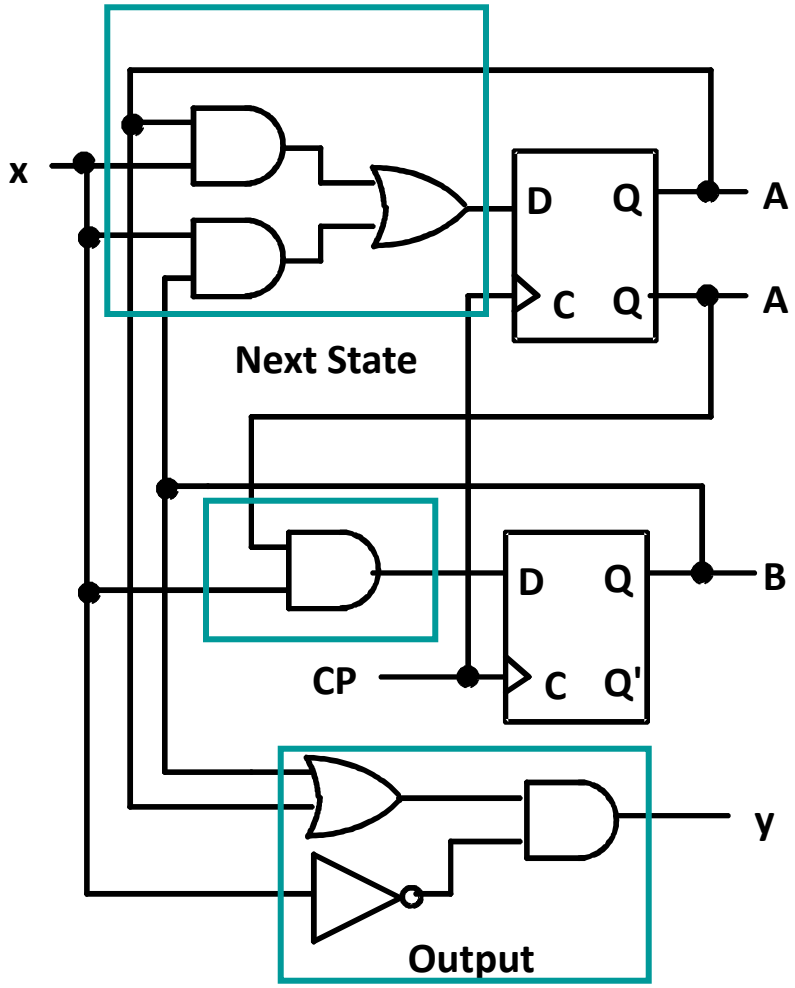
- 输入: X
- 输出: Y
- 状态: (D_A, D_B)
- 下一状态函数?
- 输出函数?



1. 方程

状态函数：$\begin{cases} D_A = AX + BX \\ D_B = \bar{A}X \end{cases}$

输出函数：$Y = (A + B)\bar{X}$



2. 状态表

- 四个部分
  - 当前状态: 状态变量合法值
  - 输入: 合法输入组合
  - 下一状态: 基于当前状态和输入的下一状态
  - 输出: 基于当前状态和输入(可选)的输出
- 可以视为真值表
  - 输入: 当前状态+输入
  - 输出: 下一状态+输出

2. 状态表

- 一维状态表
  - 可由方程推导

$$D_A = AX + BX$$

$$D_B = \bar{A}X$$

$$Y = (A + B)\bar{X}$$

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

2. 状态表

二维状态表

- 输入从左到右列表第一行

$$D_A = AX + B\bar{X}$$

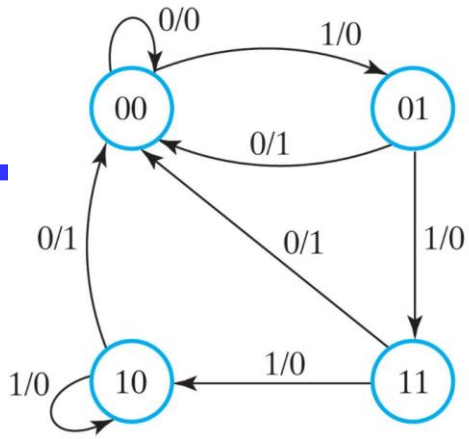
$$D_B = \bar{A}X$$

$$Y = (A + B)\bar{X}$$

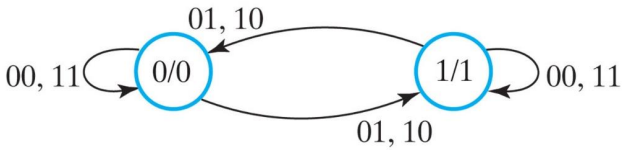
Present State		Next State				Output	
		X = 0		X = 1		X = 0	X = 1
A	B	A	B	A	B	Y	Y
0	0	0	0	0	1	0	0
0	1	0	0	1	1	1	0
1	0	0	0	1	0	1	0
1	1	0	0	1	0	1	0

交换后两行，状态和输入符合格雷码序，与卡诺图匹配

3. 状态图

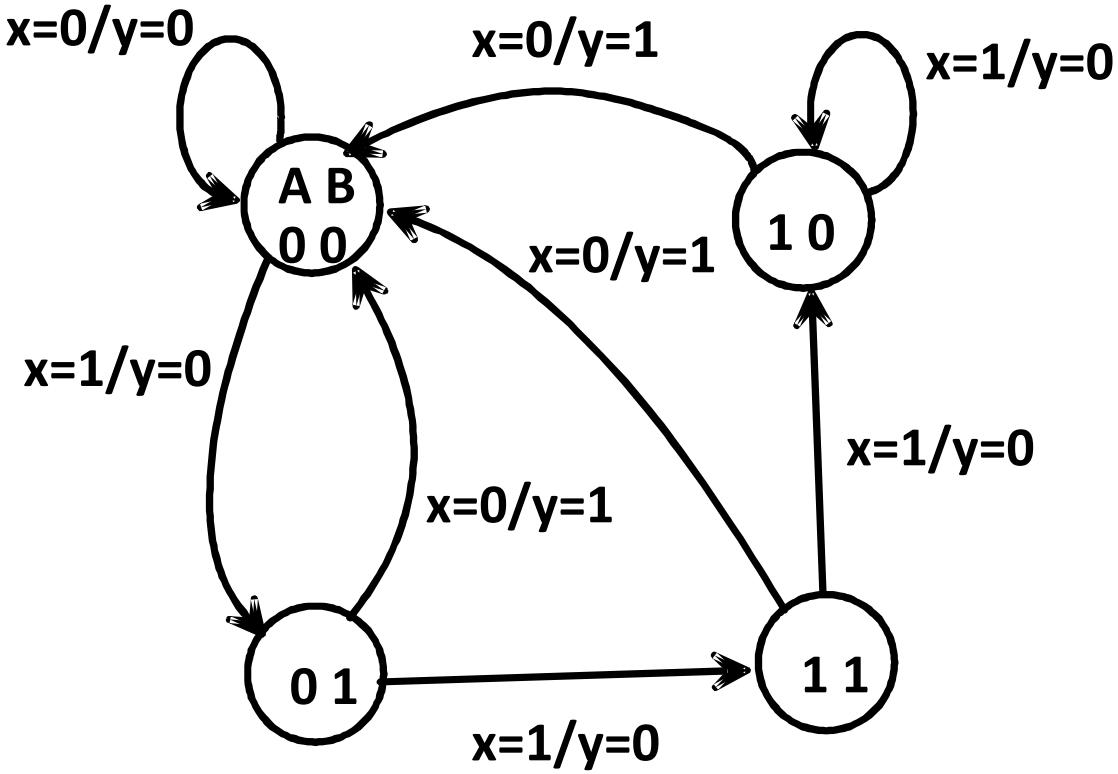


- 用图的形式表达时序电路函数
- 状态： 用圆圈表示
- 下一状态函数： Next State = f(Inputs, State)
  - 当前状态到下一状态弧线
  - 弧线上标注输入，即转移条件



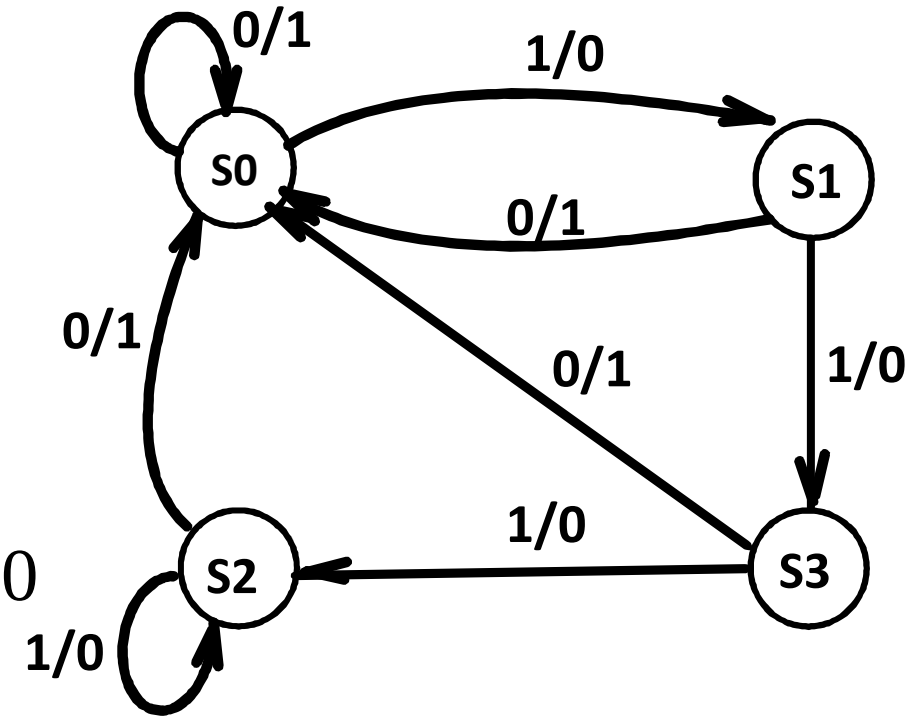
- 输出函数： Outputs = h(State) （Moore）
  - 状态上标注输出：状态/输出
- 输出函数： Outputs = g(Inputs, State) （Mealy）
  - 借用下一状态函数弧线：输入/输出

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



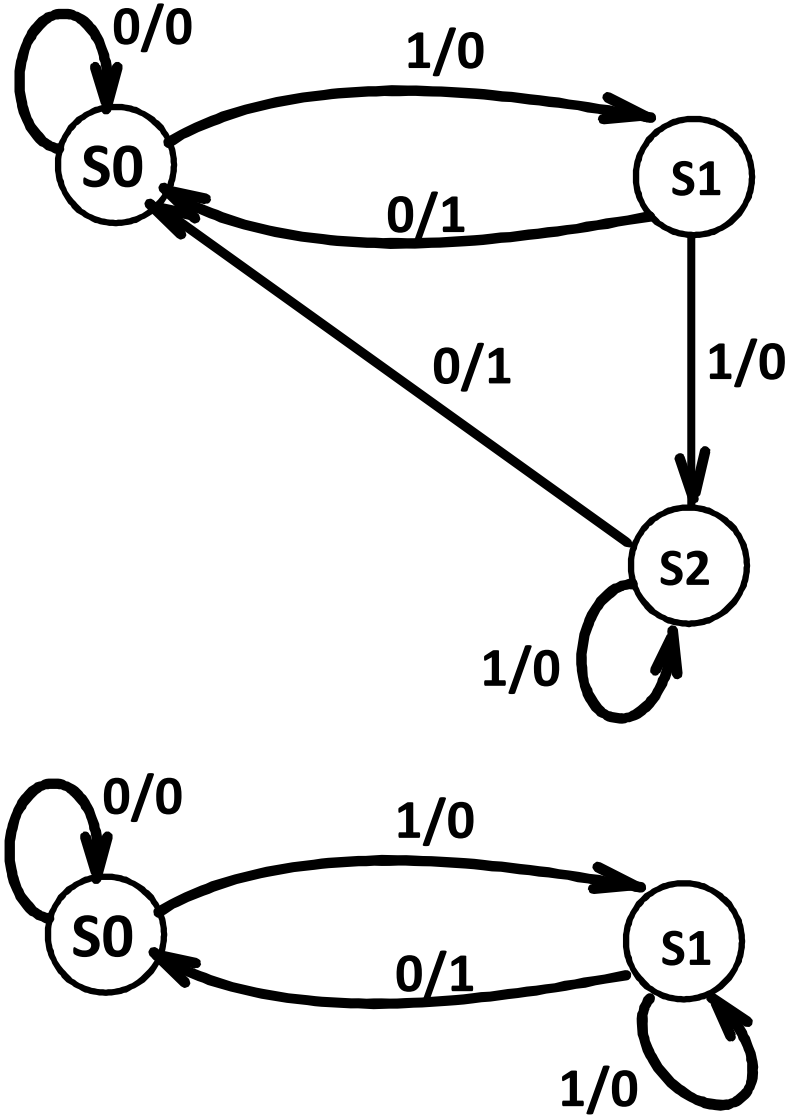
4. 等价状态

- 等价状态
  - 两个状态，对每一个输入
    - 相同的输出
    - 相同的下一状态
  - 则这两个状态等价
- 对于S2和S3
  - 输入0
    - 输出1，下一状态S0
  - 输入1
    - 输出0，下一状态S2
  - S2和S3等价



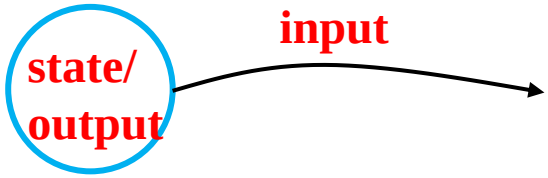
4. 等价状态

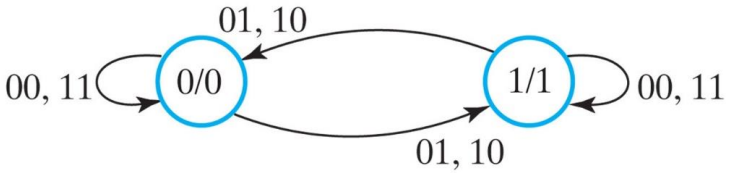
- 等价状态可以合并为1个
  - S2和S3合并为S2
- 新状态图上，S1和S2
  - 输入0
    - 输出1，下一状态S0
  - 输入1
    - 输出0，下一状态S2
  - S1和S2等价
  - S1和S2合并为S1

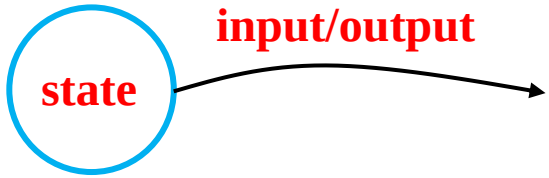


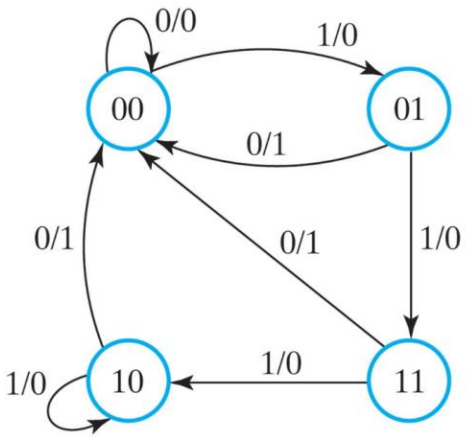
5. 输出类型

- 两种输出类型
  - Moore型
    - Outputs = h(State)
    - 状态上标注
    - 状态/输出
  - Mealy型
    - Outputs = g(Inputs, State)
    - 转移弧上标注
    - 输入/输出









5. 输出类型

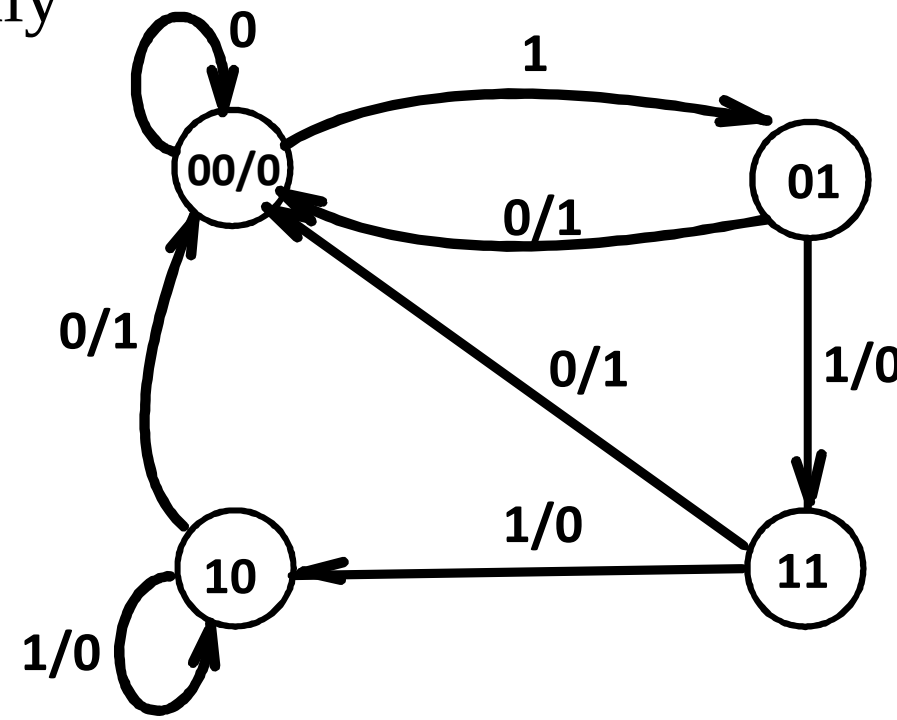
- 改造状态表
  - Moore 型
    - 不包含输出条件
  - Mealy 型
    - 包含输出条件

Present State	Next State x=0	Next State x=1	Output
0	0	1	0
1	0	2	0
2	0	2	1

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

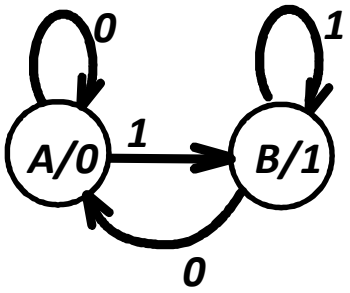
5. 输出类型

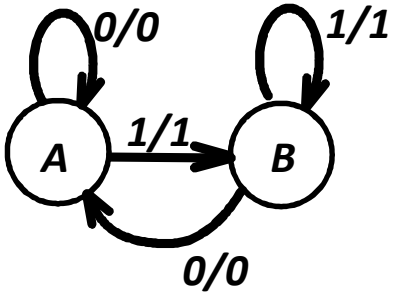
- 两种类型混合
  - 实际设计中, 有些输出Moore型，有些Mealy
  - 00上输出: Moore
  - 01, 10, 和11上输出: Mealy



5. 输出类型

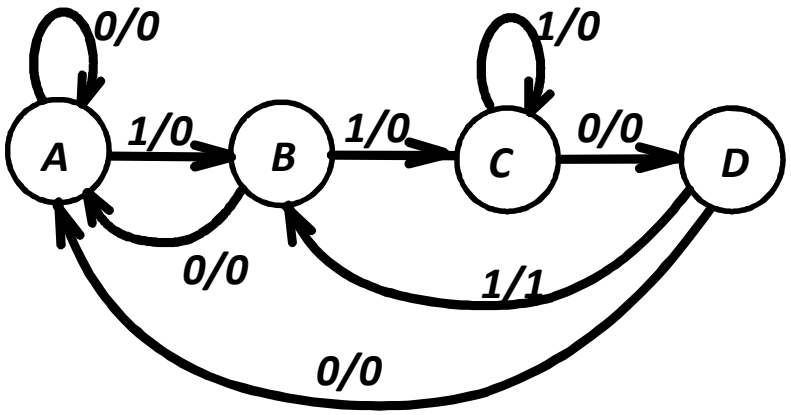
- Moore和Mealy 转换
- 基本思想：Moore 输出比Mealy慢一拍
  - Moore：先转移到次态，在次态(行)输出(状态表)
  - Mealy:　转移同时输出，在本态(行)输出(状态表)
- Moore 到Mealy
  - 当前状态输出改为输入弧上输出

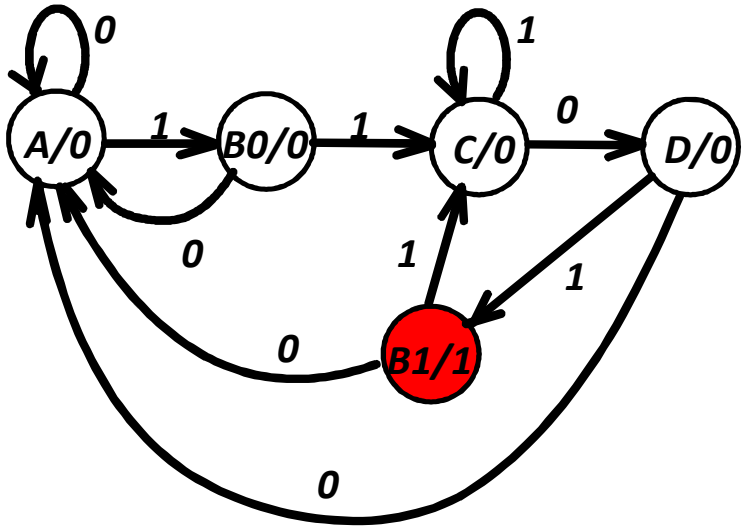




5. 输出类型

- Mealy 到Moore
  - 输入弧上输出改为当前状态输出
    - 当前状态输入弧上输出一致，直接修改
    - 当前状态输入弧上输出不一致，增加状态
      - 一个状态对应一个输出
      - 不同状态：不同输入弧，相同输出弧





三. 时序电路设计

- 1. 规范化
- 2. 形式化
- 3. 状态分配
- 4. 优化
- 5. 工艺映射
- 6. 验证
- 7. 例子

组合电路 vs. 时序电路

组合电路

- 规范化
- 形式化
- 优化
- 工艺映射
- 验证

时序电路

- 规范化
- 形式化
- 状态分配
- 优化
- 工艺映射
- 验证

① 规范化：规格说明
② 形式化：状态表或状态图
③ 状态分配：给状态编码
④ 优化
- 确定输入方程：下一状态函数
- 确定输出方程：输出函数
- 优化：对方程优化

⑤ 工艺映射：将方程映射到触发器和门工艺
⑥ 验证：验证设计正确性

1. 规范化

- 描述电路的时序行为
- 形式:
  - 文字描述
  - 数学描述
  - 硬件描述语言
  - 表格描述
  - 方程描述
  - 图表述
- 序列识别器
  - 文字描述：一个时序电路, 其能在输入序列中识别目标序列1101的出现

- 序列识别器
  - Verilog描述
    - 存储单元
    - 状态函数
    - 输出函数
    - 自动完成

```
// Sequence Recognizer: Verilog Process Description
// (See Figure 20(d) for state diagram)
module seq_rec_v(CLK, RESET, X, Z);
  input CLK, RESET, X;
  output Z;
  reg [1:0] state, next_state;
  parameter A = 2'b00, B = 2'b01, C = 2'b10, D = 2'b11;
  reg Z;
// state register: implements positive-edge-triggered
// state storage with asynchronous reset.
always @(posedge CLK or posedge RESET)
begin
  if (RESET == 1)
    state <= A;
  else
    state <= next_state;
end
// next state function: implements next state as function
// of X and state
always @(X or state)
begin
  case (state)
    A:   if (X == 1)
           next_state <= B;
         else
           next_state <= A;
    B:   if(X) next_state <= C;else next_state <= A;
    C:   if(X) next_state <= C;else next_state <= D;
    D:   if(X) next_state <= B;else next_state <= A;
  endcase
end
// output function: implements output as function
// of X and state
always @(X or state)
begin
  case (state)
    A: Z <= 0;
    B: Z <= 0;
    C: Z <= 0;
    D: Z <= X ? 1 : 0;
   endcase
end
endmodule
```

2. 形式化

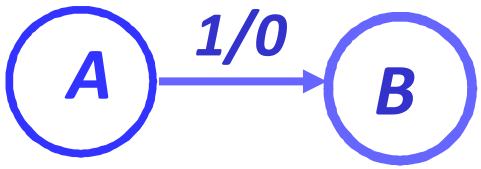
- 状态图或状态表
- 状态：加载到电路历史输入的抽象
- 序列识别器
  - 文字描述：一个时序电路, 其能在输入序列中识别目标序列1101的出现
  - 重叠情况：1101101中子序列重叠，均需识别
    - 1101101 或 1101101
    - 识别出最后1个1即识别出第1个1

2. 形式化

- 状态图或状态表
- 状态：加载到电路历史输入的抽象
- 序列识别器
  - 文字描述：一个时序电路, 其能在输入序列中识别目标序列1101的出现
- 分析
  - 初始状态, 没有任何符号识别
  - 增加一个状态识别第一个符号
  - 当有后续符号输入, 增加新状态来识别
  - 最终状态表示输入序列被完整识别

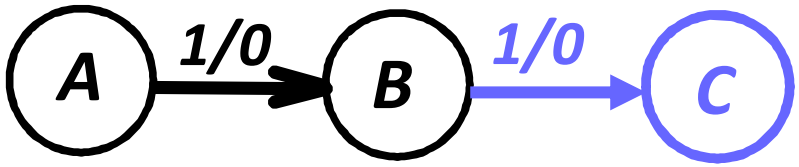
2. 形式化

- 序列识别器：1101
- 状态图(Mealy型)
  - 初始状态A
    - A意义：无子序列被识别
  - 识别出第1个1，增加状态B，输出0
    - B意义：识别出子序列1
    - 0：没有完整识别出目标序列



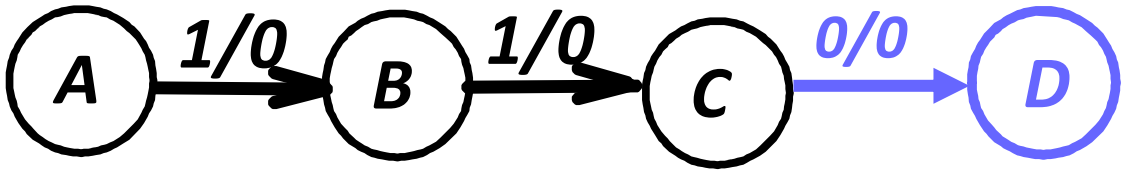
2. 形式化

- 序列识别器：1101
- 状态图(Mealy型)
  - 识别出第2个1，增加状态C，输出0
    - C意义：识别出子序列11



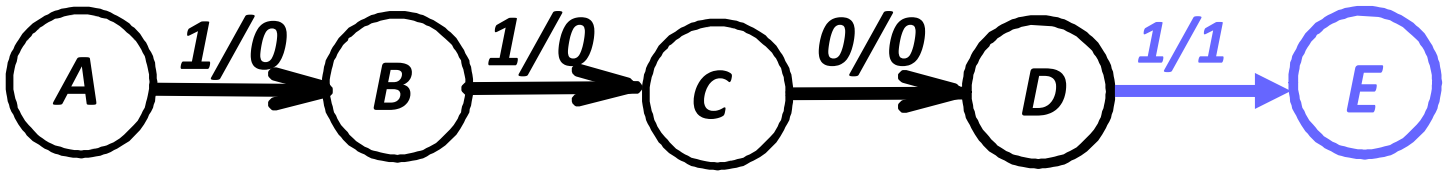
2. 形式化

- 序列识别器：1101
- 状态图(Mealy型)
  - 识别出0，增加状态D，输出0
    - D意义：识别出子序列110



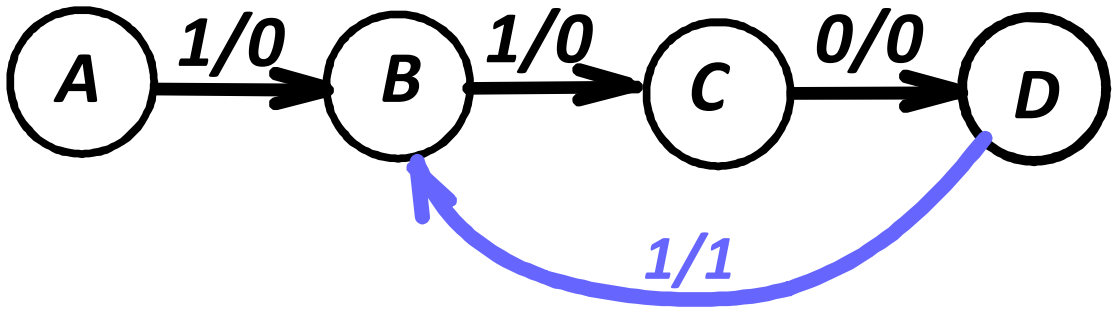
2. 形式化

- 序列识别器：1101
- 状态图(Mealy型)
  - 识别出1，增加状态E，输出1
    - E意义：完整识别出目标序列
    - 1：完整识别出目标序列

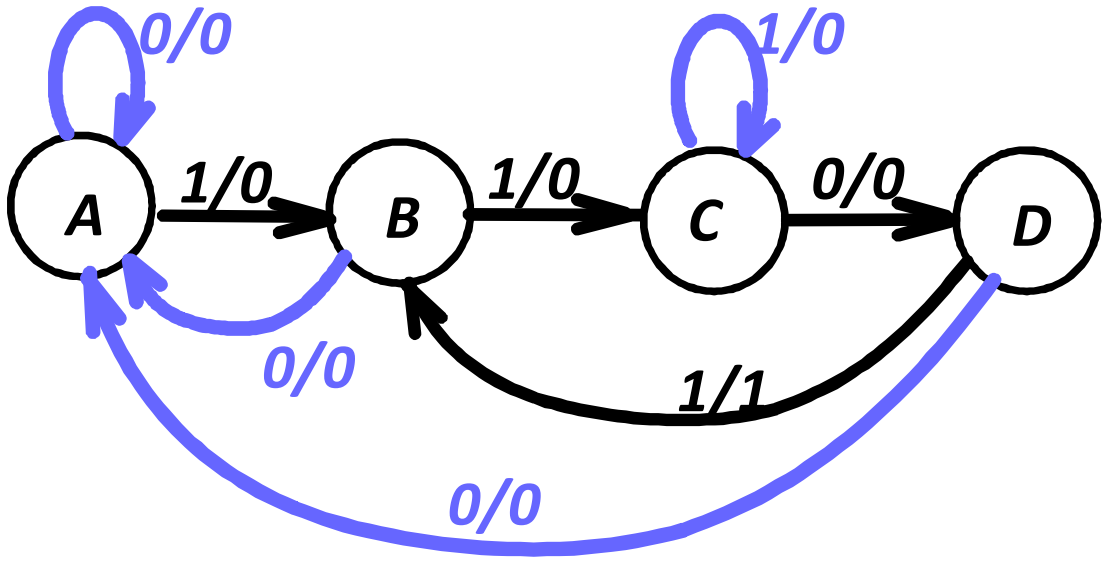


2. 形式化

- 序列识别器：1101
- 状态图(Mealy型)
  - 识别出1，复用状态B，输出1
    - B意义：识别出子序列1/完整识别出目标序列
    - E与B是等价状态
    - 识别出最后1个1即识别出第1个1
    - 1：完整识别出目标序列

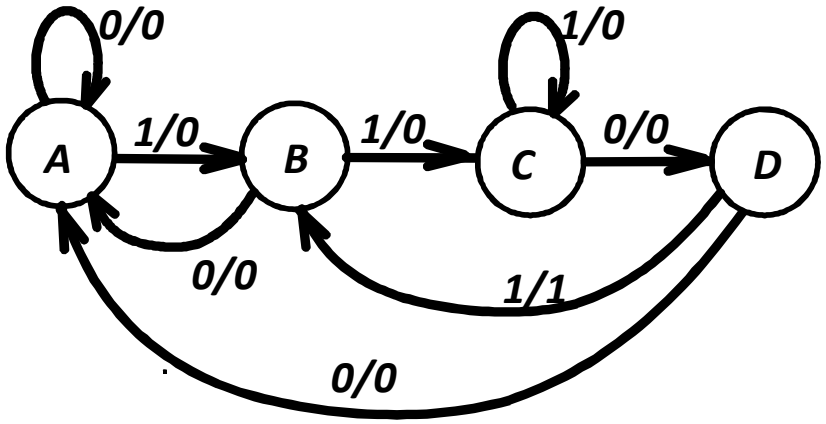


- 序列识别器：1101
- 增加未正常识别的转移弧



2. 形式化

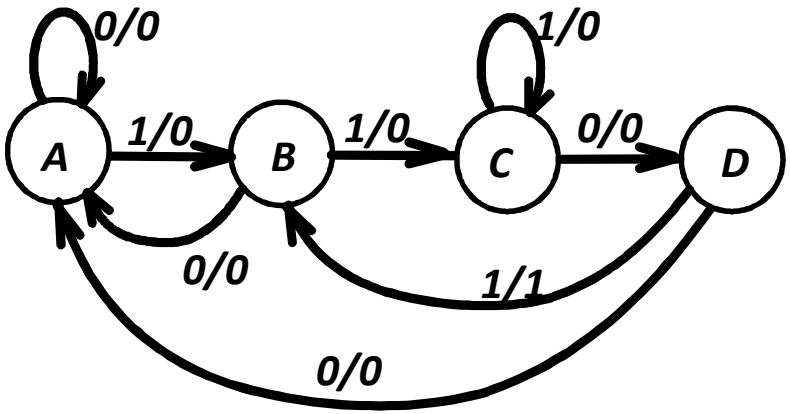
- 状态表
  - 输入：序列
  - 状态：A, B, C, D
  - 输出：结果

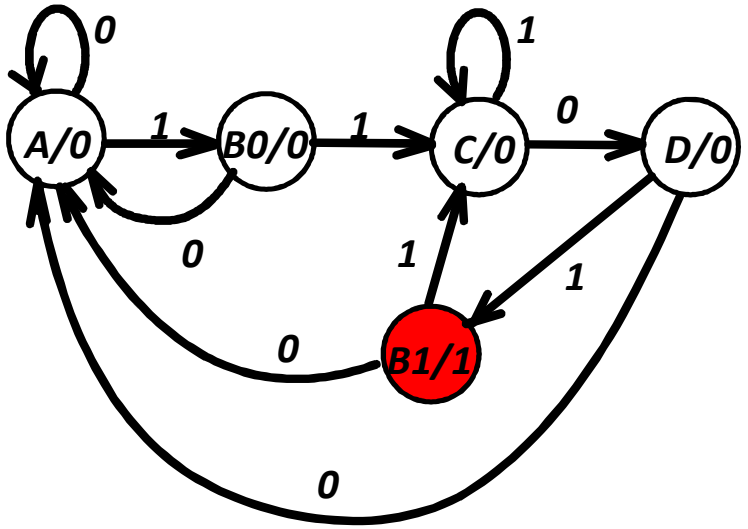


Present State	Next State x=0	Next State x=1	Output x=0	Output x=1
A	A	B	0	0
B	A	C	0	0
C	D	C	0	0
D	A	B	0	1

2. 形式化

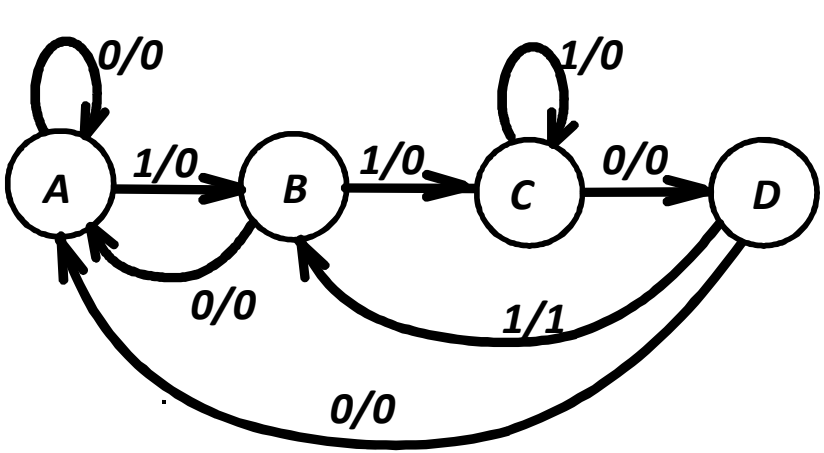
- Mealy 到Moore
  - 输入弧上输出改为当前状态输出
    - 当前状态输入弧上输出一致，直接修改
    - 当前状态输入弧上输出不一致，增加状态
      - 一个状态对应一个输出
      - 不同状态：不同输入弧，相同输出弧

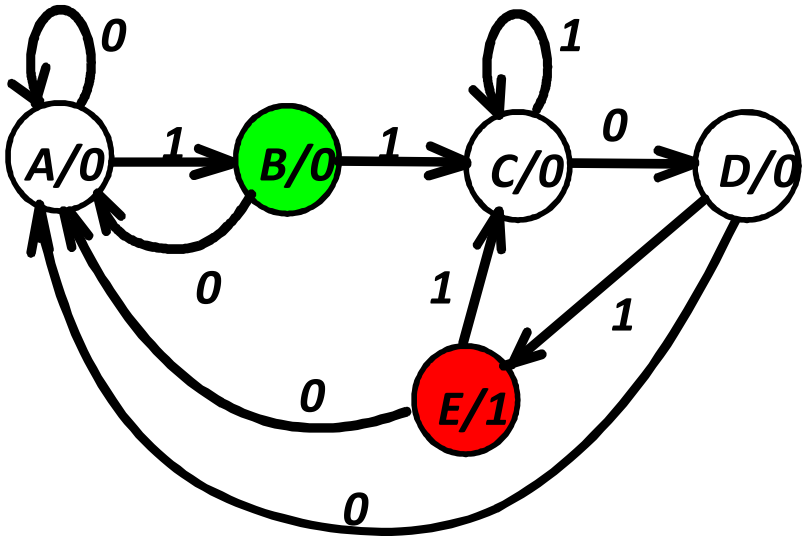




2. 形式化

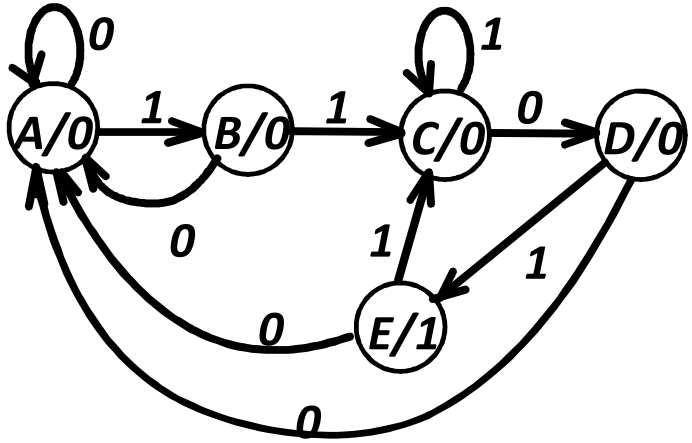
- Mealy到Moore
- 增加状态E
  - E意义：完整识别出目标序列
- Moore型具有更多状态
  - Moore is more



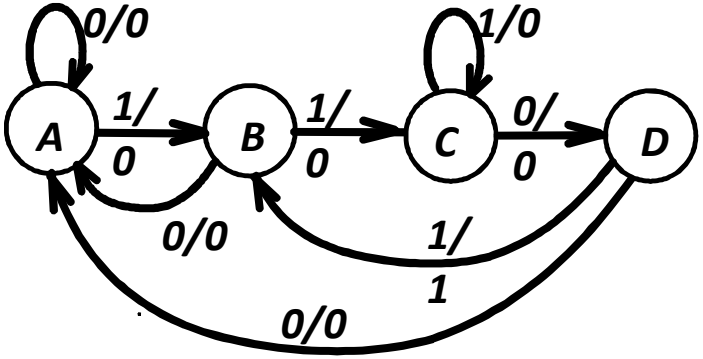


2. 形式化

形式化最终成果



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



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

3. 状态分配

- 给状态分配唯一二进制编码
  - m个状态至少需要n位二进制：$n \geq \lceil log_2 m \rceil$
  - 至多存在$2^n - m$个未使用状态
- 分配方式
  - 计数赋值：A(00)、B(01)、C(10)、D(11)
  - 格雷码赋值：A(00)、B(01)、C(11)、D(10)
  - 单热点赋值：
    - A(0001)、B(0010)、C(0100)、D(1000)
    - 每个状态一个触发器
    - 需要m位长
    - 只有1位为1

3. 状态分配

计数赋值

- A(00)、B(01)、C(10)、D(11)

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

3. 状态分配

格雷码赋值

- A(00)、B(01)、C(11)、D(10)

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 1	0	0
1 1	1 0	1 1	0	0
1 0	0 0	0 1	0	1

3. 状态分配

- 单热点赋值
  - A(0001)、B(0010)、C(0100)、D(1000)

Present State	Next State		Output	
	x = 0	x = 1	x = 0	x = 1
0001	0001	0010	0	0
0010	0001	0100	0	0
0100	1000	0100	0	0
1000	0001	0010	0	1

4. 优化

- 确定输入方程：下一状态函数
- 确定输出方程：输出函数
- 优化：对方程优化
- 工具：卡诺图

4. 优化

- 计数赋值
  - 两个D触发器
    - 状态：D_1, D_2
    - 输出：Y_1, Y_2
  - 输出Z
  - 交换状态表后两行

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

D_1

Y_1 \ Y_2 X	0	1
0	0	0
1	0	1

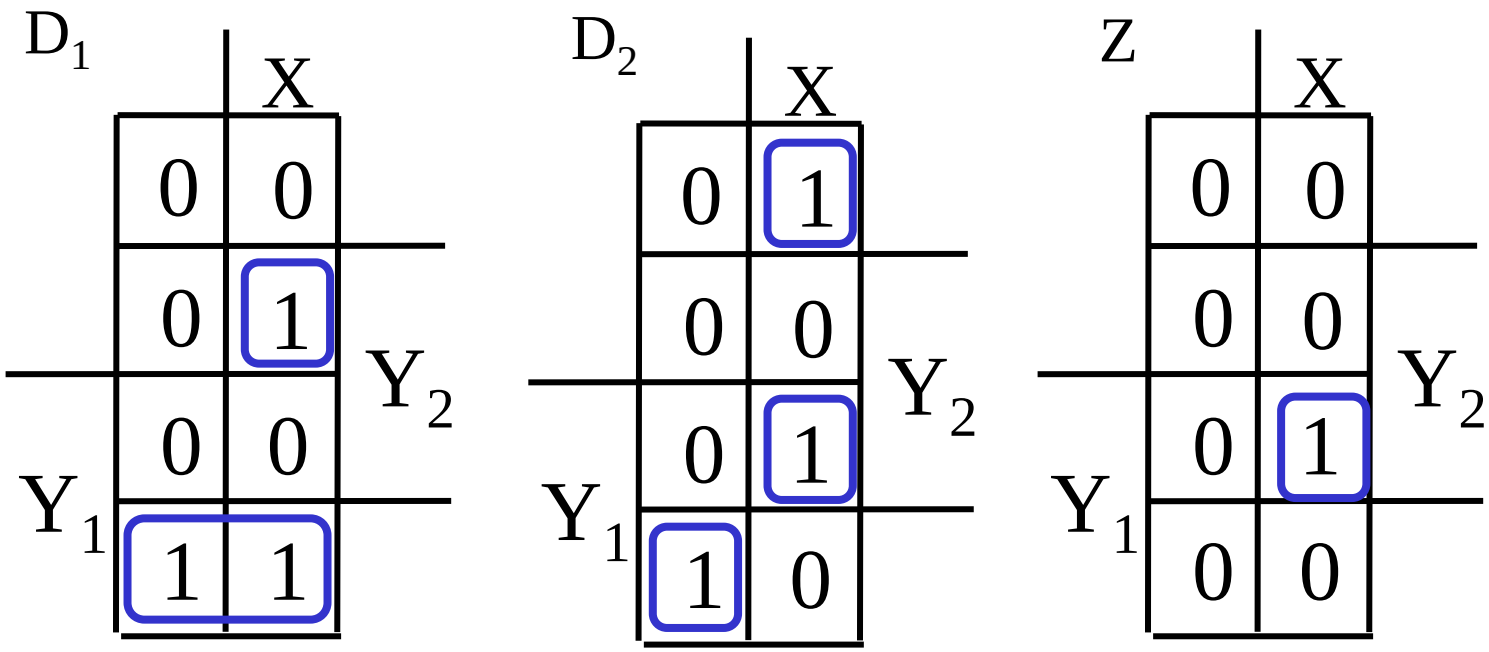
D_2

Y_1 \ Y_2 X	0	1
0	1	0
1	0	1

Z

Y_1 \ Y_2 X	0	1
0	0	0
1	0	0

4. 优化



$$D_1 = Y_1\bar{Y}_2 + X\bar{Y}_1Y_2$$
$$D_2 = \bar{X}Y_1\bar{Y}_2 + X\bar{Y}_1\bar{Y}_2 + XY_1Y_2$$
$$Z = XY_1Y_2$$

门输入成本G=22

4. 优化

- 格雷码赋值
  - 两个D触发器
    - 状态：D_1, D_2
    - 输出：Y_1, Y_2
  - 输出Z
  - 不用交换状态表后两行

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 1	0	0
1 1	1 0	1 1	0	0
1 0	0 0	0 1	0	1

D_1

Y_1 \ X	0	1
0	0	0
1	1	1
0	0	0

Y_2

D_2

Y_1 \ X	0	1
0	0	1
1	0	1
0	0	1

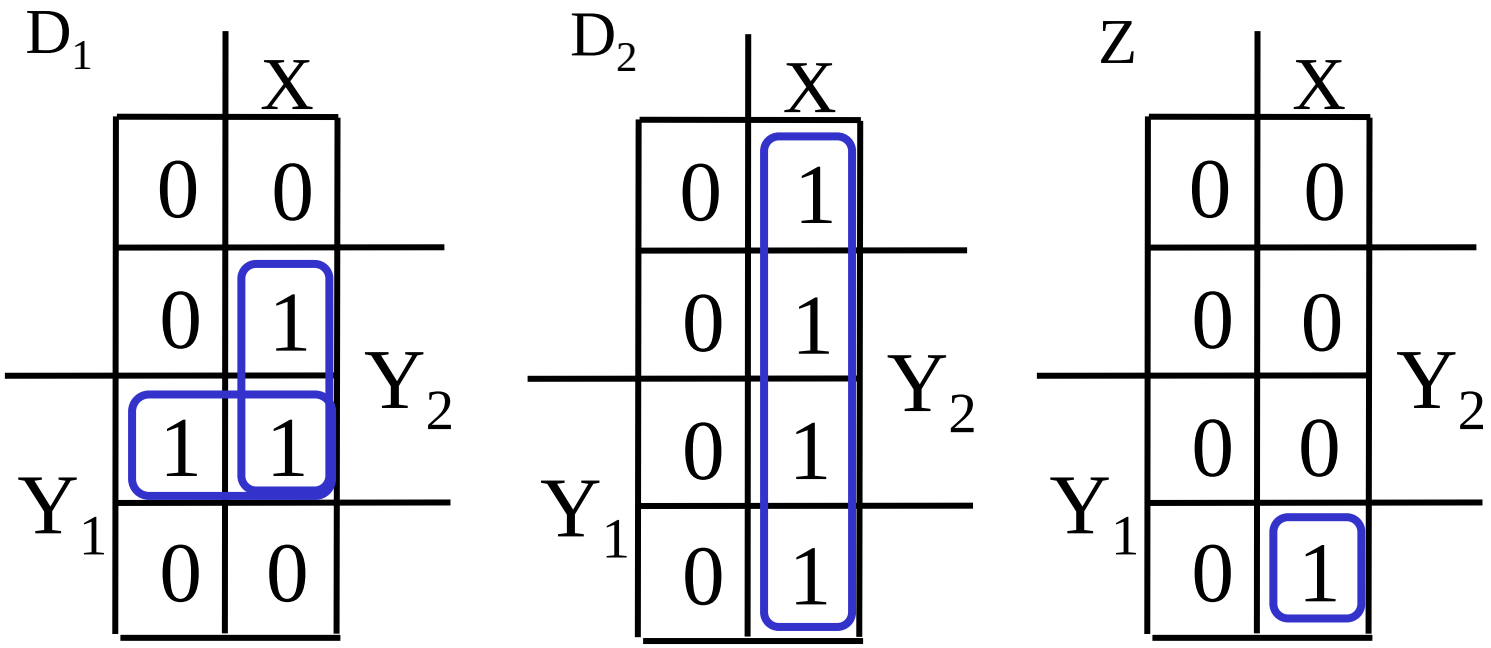
Y_2

Z

Y_1 \ X	0	1
0	0	0
1	0	0
0	0	0

Y_2

4. 优化



$$D_1 = Y_1Y_2 + XY_2$$
$$D_2 = X$$
$$Z = XY_1\overline{Y}_2$$

门输入成本G=9

4. 优化

- 单热点赋值
  - 四个D触发器
    - 状态：D_3~D_0
    - 输出：Y_3~Y_0
    - 序号和之前相反
  - 输出Z
  - 直接观察方程
    - 最小项之和
  - 仅有1位是1
    - 简化最小项
    - Y_0等价$\overline{Y}_3\ \overline{Y}_2\ \overline{Y}_1\ Y_0$
    - 全0或两个及以上1是无关项

Present State	Next State x = 0	Next State x = 1	Output x = 0	Output x = 1
0001	0001	0010	0	0
0010	0001	0100	0	0
0100	1000	0100	0	0
1000	0001	0010	0	1

$$D_0 = \overline{X}(Y_0 + Y_1 + Y_3) \text{ or } \overline{X}\,\overline{Y}_2$$

$$D_1 = X(Y_0 + Y_3)$$

$$D_2 = X(Y_1 + Y_2) \text{ or } X\overline{(Y_0 + Y_3)}$$

$$D_3 = \overline{X}\,Y_2$$

$$Z = XY_3$$

4. 优化

- 单热点赋值
  - 四个D触发器
    - 状态：D_3~D_0
    - 输出：Y_3~Y_0
    - 序号和之前相反
  - 输出Z
  - 可简化设计
    - 简单组合逻辑
    - 更多的触发器
  - 综合门成本不低

Present State	Next State x = 0	Next State x = 1	Output x = 0	Output x = 1
0001	0001	0010	0	0
0010	0001	0100	0	0
0100	1000	0100	0	0
1000	0001	0010	0	1

$$D_0 = \overline{X}(Y_0 + Y_1 + Y_3) \text{ or } \overline{X}\,\overline{Y_2}$$

$$D_1 = X(Y_0 + Y_3)$$

$$D_2 = X(Y_1 + Y_2) \text{ or } X\overline{(Y_0 + Y_3)}$$

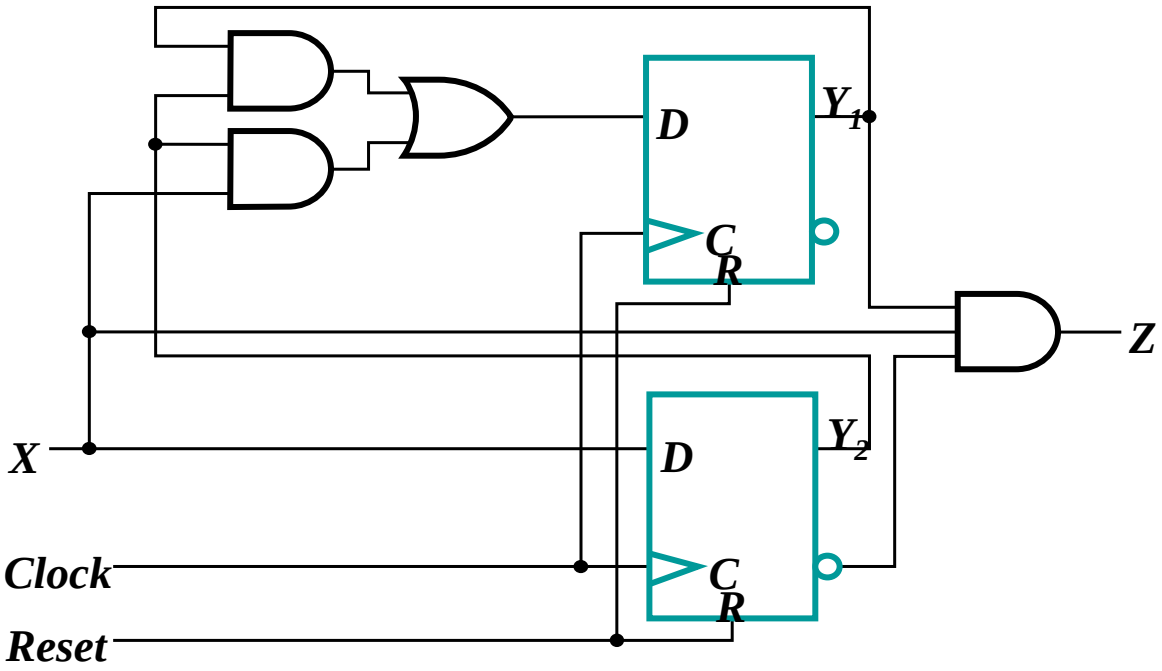
$$D_3 = \overline{X}\,Y_2$$

$$Z = XY_3$$

5. 工艺映射

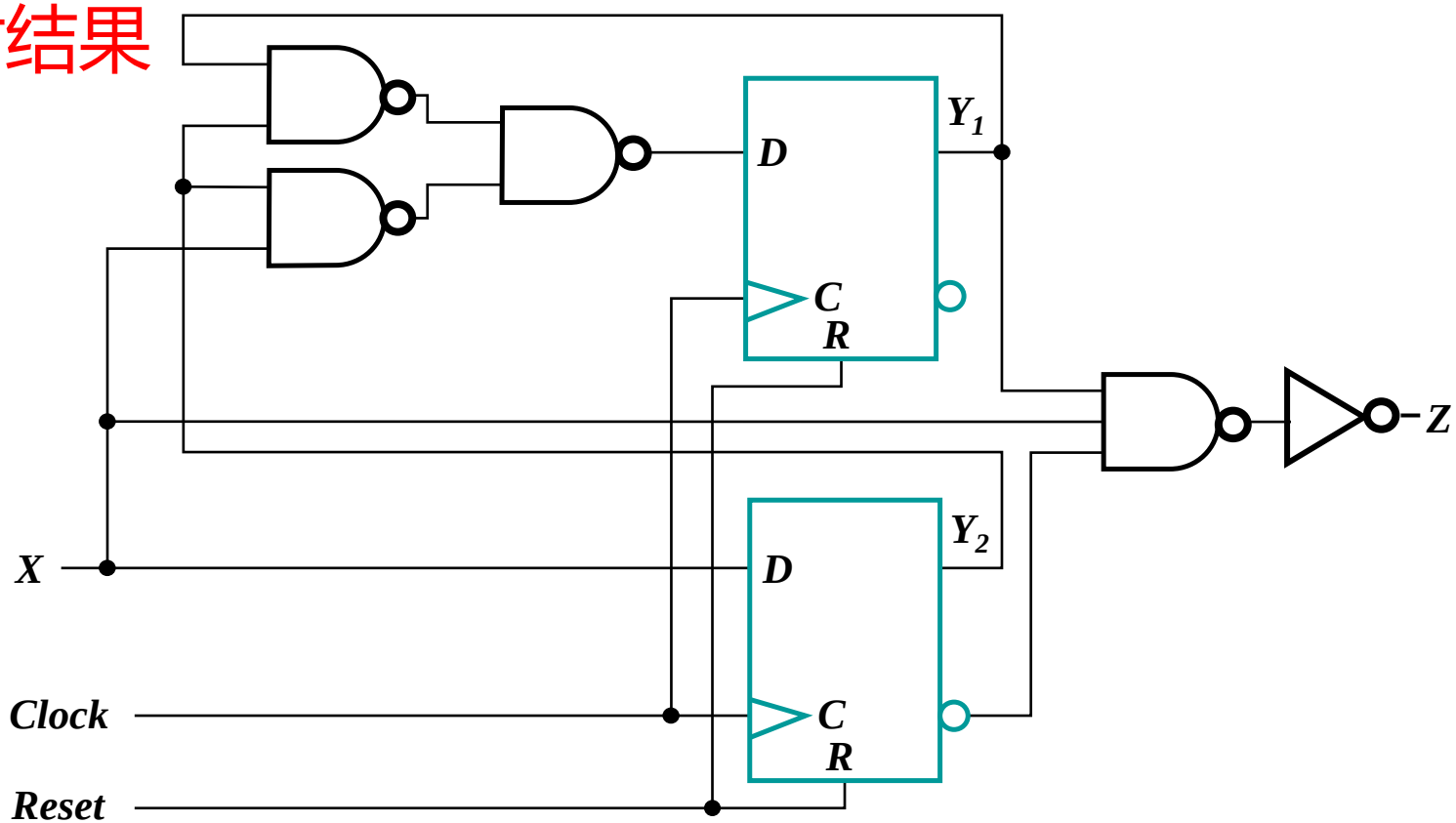
- 格雷码赋值优化结果
- 考虑系统启动时初始化
  - 带有Reset的D触发器
- 优化结果

$$D_1 = Y_1Y_2 + XY_2$$
$$D_2 = X$$
$$Z = XY_1\overline{Y}_2$$



5. 工艺映射

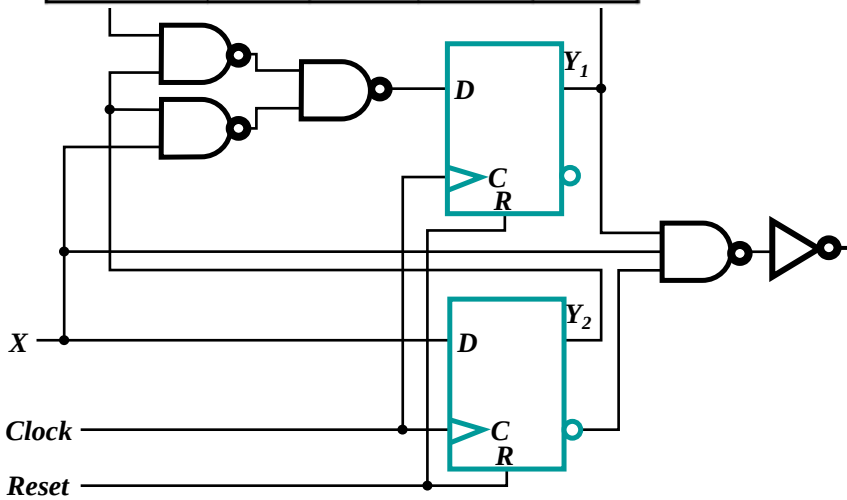
- 库中包含
  - 带有Reset的D触发器
  - 反相器和n-输入与非门， $n = 2, 3, 4$
- 映射结果



6. 验证

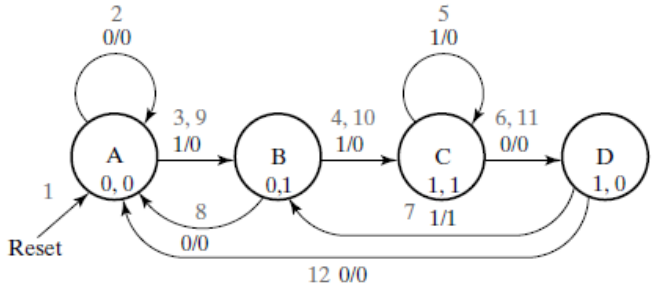
- 可通过呈现原始状态图或状态表进行验证
  - 手工验证：对于小电路，可加载各种状态与输入组合，验证输出和下一状态是否正确。
- 4状态x2输入=8组合
  - 复位：状态(0,0)
  - (0,0)：输入0，输出0，下一状态(0,0)
  - (0,0)：输入1，输出0，下一状态(0,1)
  - ……

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 1	0	0
1 1	1 0	1 1	0	0
1 0	0 0	0 1	0	1



6. 验证

- 可通过呈现原始状态图或状态表进行验证
  - 模拟验证：输入能够检验所有状态和输入组合的序列(越短越优)、时钟信号，在时钟上升沿后验证输出和下一个状态是否正确。



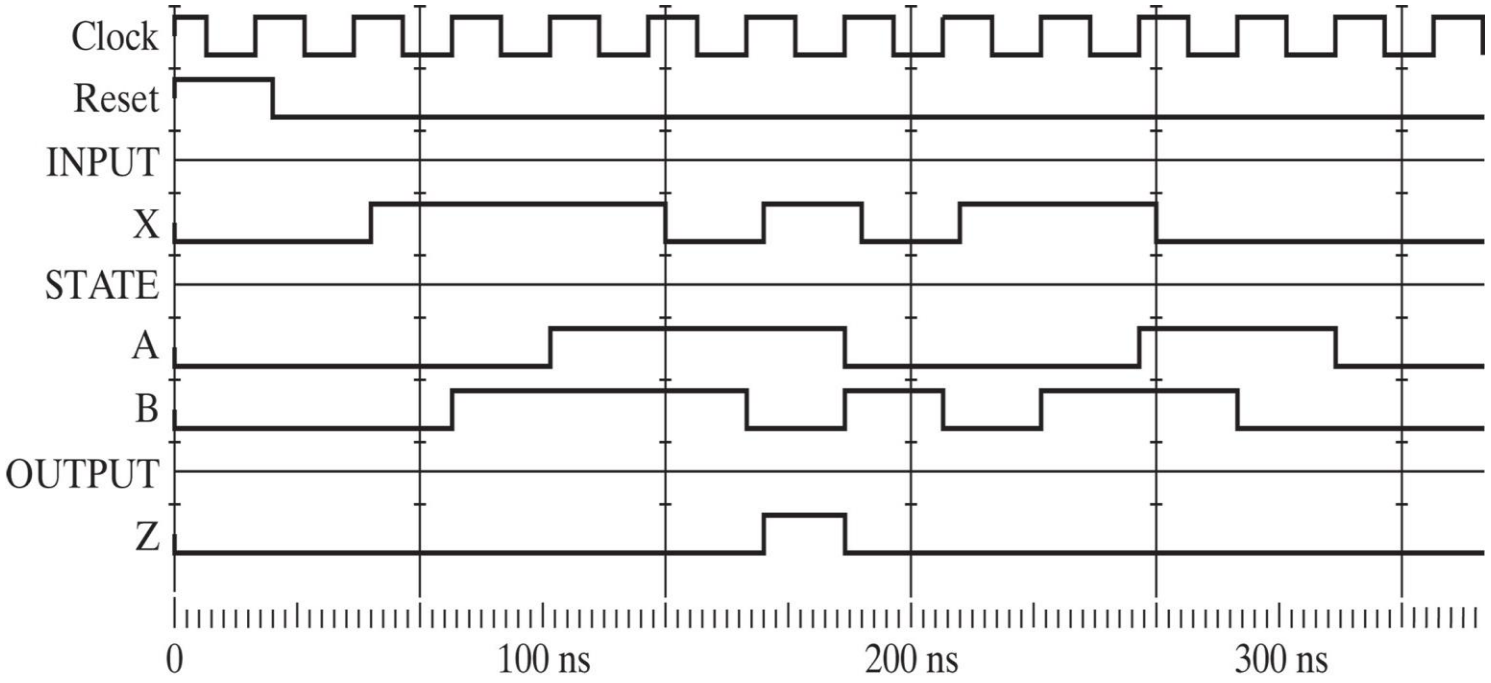
(a)

Clock Edge:	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Input R:	X	1	0	0	0	0	0	0	0	0	0	0	0	
Input X:	X	0	0	1	1	1	0	1	0	1	1	0	0	
State (A, B):	X, X	0, 0*	0, 0	0, 0	0, 1	1, 1	1, 1	1, 0	0, 1	0, 0	0, 1	1, 1	1, 0	0, 0
Output Z:	X	0	0	0	0	0	0	1	0	0	0	0	0	

(b)

6. 验证

- 可通过呈现原始状态图或状态表进行验证
  - 模拟验证：输入能够检验所有状态和输入组合的序列(越短越优)、时钟信号，在时钟上升沿后验证输出和下一个状态是否正确。



6. 验证

- Verilog
  - Testbench
    - 时钟
    - 初始化
    - 调用

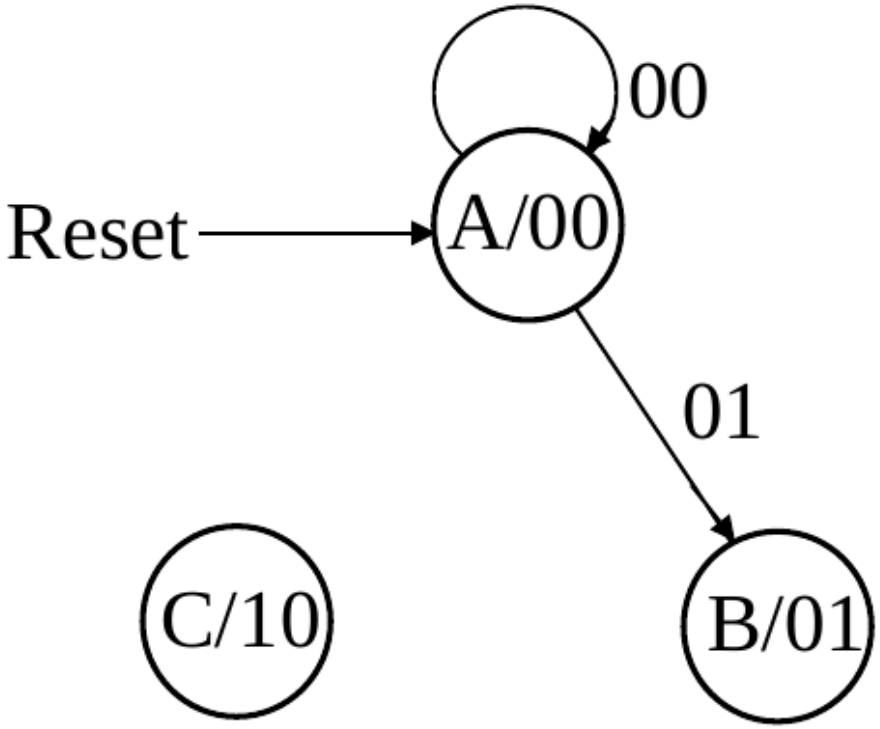
```
// Testbench for Verilog sequence recognizer
module seq_req_v_testbench();
    wire Z;
    reg clock, X, reset;

    reg [0:10] test_sequence = 11'b011_1010_1100;
    integer i;
    parameter PERIOD = 100;

    seq_rec_v DUT(clock, reset, X, Z);

// This initial block initializes the clock, applies reset,
// and then applies the test sequence to input X.
    initial
    begin
       reset = 1'b1;
       X = 1'b0;
       // Ensure that inputs are applied
       // away from the active clock edge
       #(5*PERIOD/4);
       reset = 1'b0;
       for (i = 0; i < 11; i = i+1)
       begin
          X = test_sequence[i];
          #PERIOD;
       end
       // Stop the simulation after all the inputs
       // in the sequence have been applied
          $stop;
       end
 // This always block provides the clock pulses
     always
     begin
        clock = 1'b1;
        #(PERIOD/2);
        clock = 1'b0;
        #(PERIOD/2);
     end
 endmodule
```

7. 例子

- 规范化
- 设计1个2位数时序模3累加器
  - 模n加法器：加法器的结果为和的余数
    - 2 + 2 模 3 = 余数 of 4/3 = 1
  - 累加器：初始0，将输入不断和存储值累加
- 状态: 结果(Y_1,Y_0)
- 输入: (X_1,X_0) [可取00, 01, 10]
- 输出: 结果(Z_1,Z_0)

7. 例子

- 形式化
  - 状态图和状态表
- 状态分配

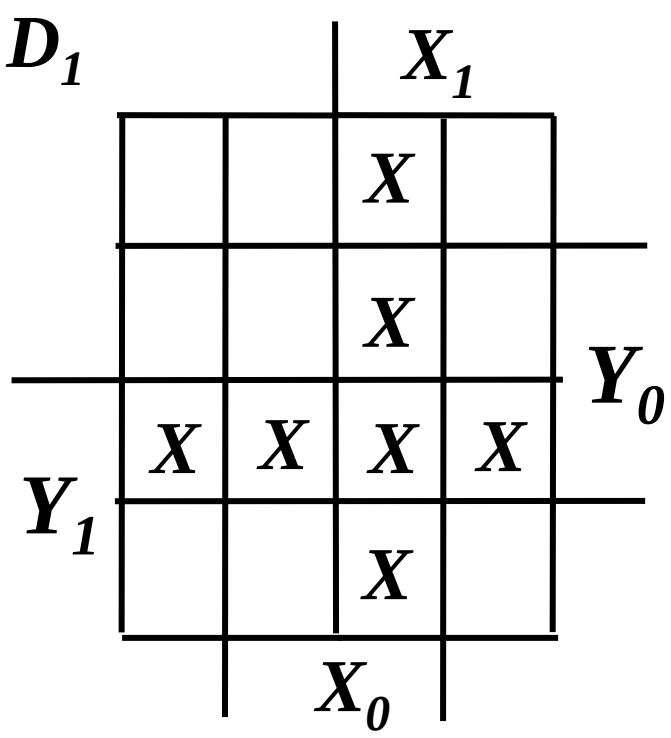


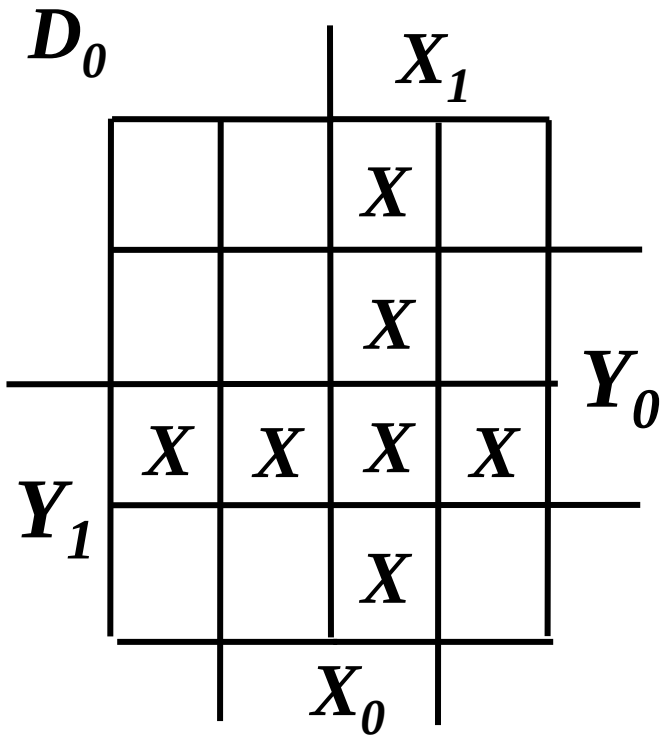
7. 例子

Present State	Next State				Output
	X_1X_0=00	X_1X_0=01	X_1X_0=11	X_1X_0=10	
A (00)	00		X		00
B (01)			X		01
- (11)	X	X	X	X	11
C (10)			X		10

7. 例子

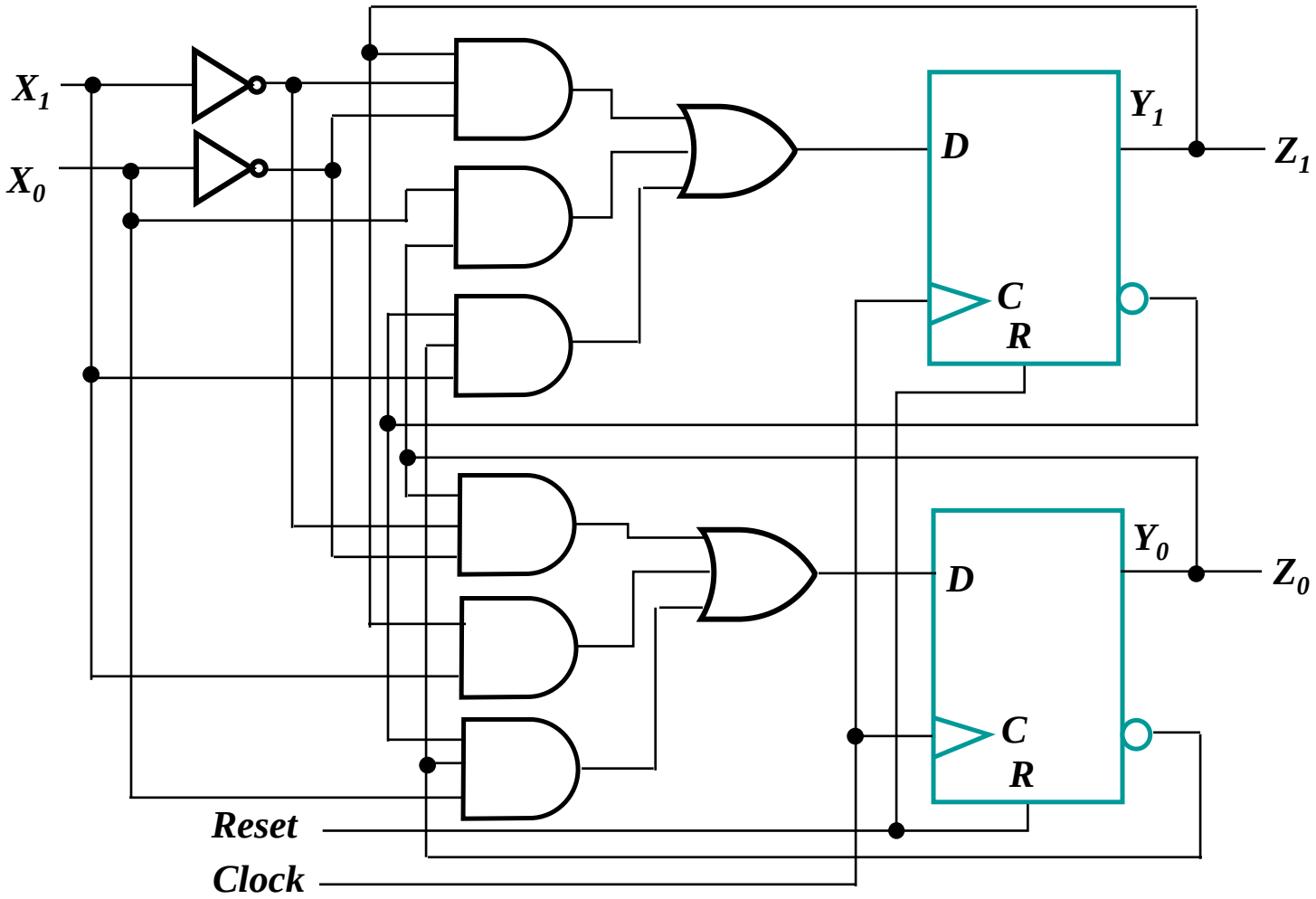
- 优化





7. 例子

- 工艺映射
- 验证



四. 状态机图及应用

- 1. 状态图的问题
- 2. 状态机图模型
- 3. 约束检查
- 4. 状态机图设计

1. 状态图的问题

- 数学基础：有限状态机
  - 3个集合
    - I：输入组合集合
    - O: 输出组合集合
    - S：状态集合
  - 2个函数
    - f(I, S)：下一个状态函数
    - h(S) [Moore 型]：输出函数
    - g(I, S) [[Mealy 型]：输出函数
- 状态图和状态机是表达有限状态机两种方式

1. 状态图的问题

- 状态图和状态表需要：
  - 问题1：表达下一状态函数，需枚举所有输入组合
    - 1输入： 2
    - 2输入： 4
    - n输入： 2^n

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

1. 状态图的问题

- 状态图和状态表需要：
  - 表达输出函数时：
    - 问题2：Mealy 型：每个状态枚举所有输入组合
    - 问题3：Mealy 型：每个(状态-输入)对写出所有输出
      - 输出为UVWXYZ，U=1，100000
    - 问题3：Moore型：写出所有输出
    - Moore比Mealy表达高效：不用枚举所有输入组合

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

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

1. 状态图的问题

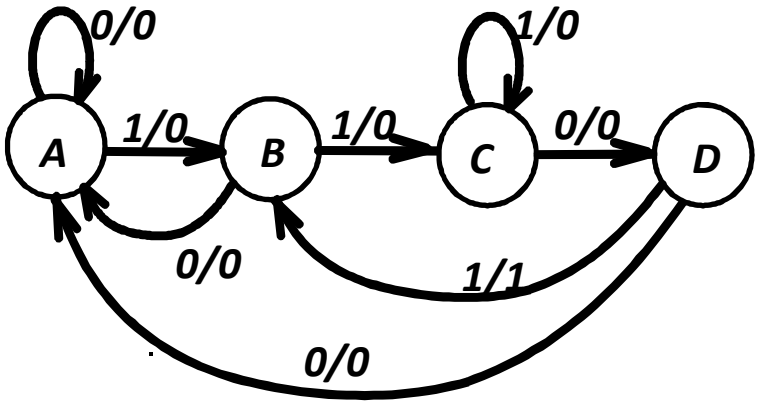
- 状态图和状态表需要：
  - Mealy输出
    - 问题4：只能表示在转移弧上
    - 不能表达在状态上

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

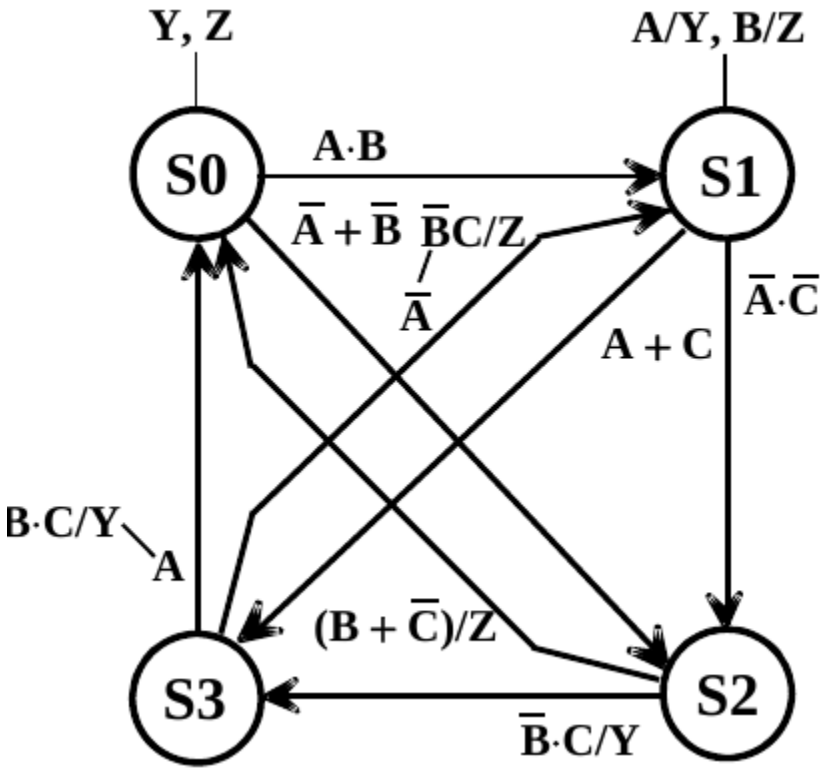
2. 状态机图模型

- 怎么办?
- 一个现象
  - 输入为X, Y, Z
  - 输入为100,101,110,111，状态A从转移到B
  - 即输入为X时，状态从A转移到B
- 可解决问题1，问题2
- 问题3：默认值，不是默认值才显示表达
- 问题4：划分转移条件和输出条件

2. 状态机图模型



状态图



状态机图

2. 状态机图模型

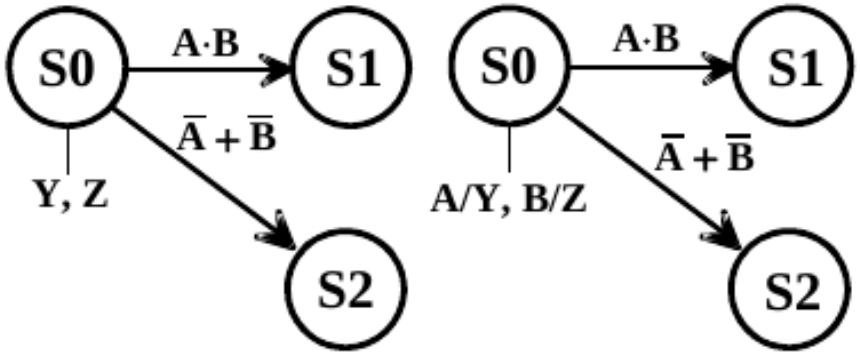
- 条件
  - 输入条件
    - 输入变量的布尔表达式或方程，值为0或1
  - 转移条件(Transition condition，TC)
    - 转移弧上表达状态转移的输入条件，为1则转移
  - 输出条件(Output condition，OC)
    - 表达输出的输入条件，为1则发生输出行为
  - 1个输入条件可同时为TC和OC

2. 状态机图模型

- 状态转移
  - 无条件转移
    - 弧上无转移条件或转移条件含常量1
  - 条件转移
    - 转移弧上有一或多个转移条件，任何一个转移条件为1，则转移发生

2. 状态机图模型

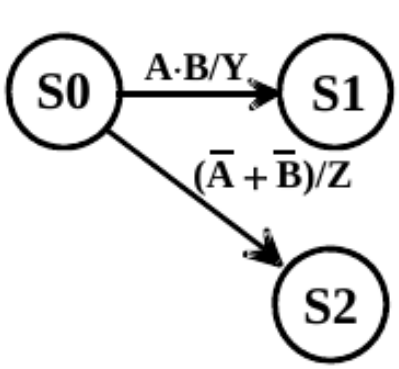
- 输出行为
  - Moore输出
    - 只依赖于状态，用一条线和相应状态连接
  - 非转移条件依赖(Transition condition-independent, TCI)
    - 输出行为由输出条件驱动，和相应状态连接



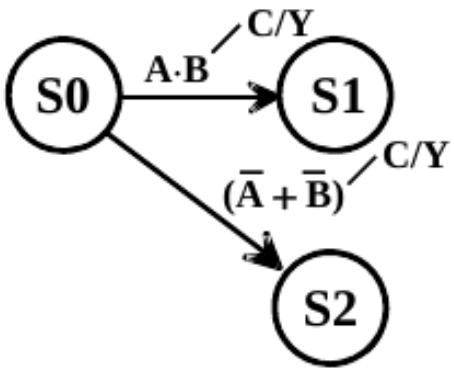
Ex. 1: Moore Outputs　　Ex. 2: TCI Outputs

2. 状态机图模型

- 输出行为
  - 转移条件依赖(Transition condition-dependent, TCD)
    - 输出行为由转移条件驱动
  - 转移和输出条件依赖(Transition and output condition-dependent，TOCD)
    - 输出行为由输出条件驱动，和转移条件连接



Ex. 3: TCD Outputs



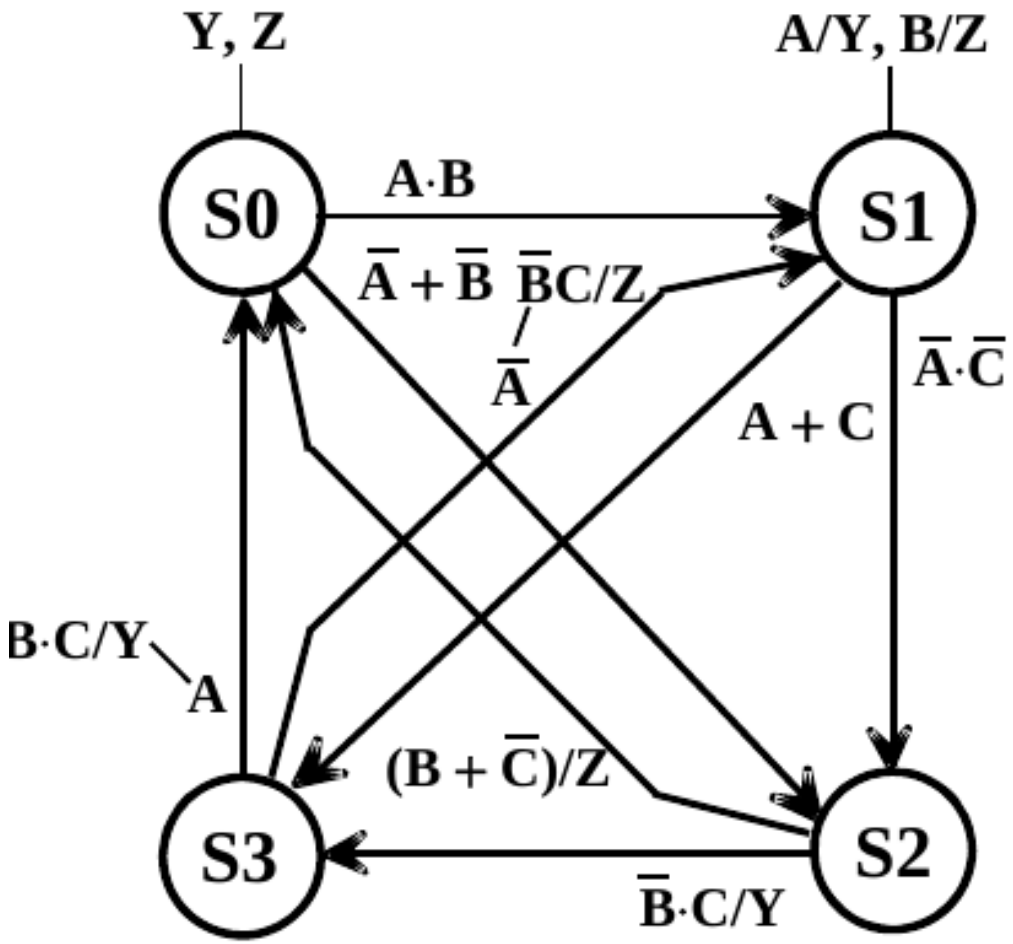
Ex. 4: TCOD Outputs

2. 状态机图模型

- 输出行为
  - 单变量Z的出现，表示Z=1
  - 向量变量Z=向量值，指定了Z的值
  - 否则Z为默认值
  - 默认值语句可指定Z默认值为0或1

2. 状态机图模型

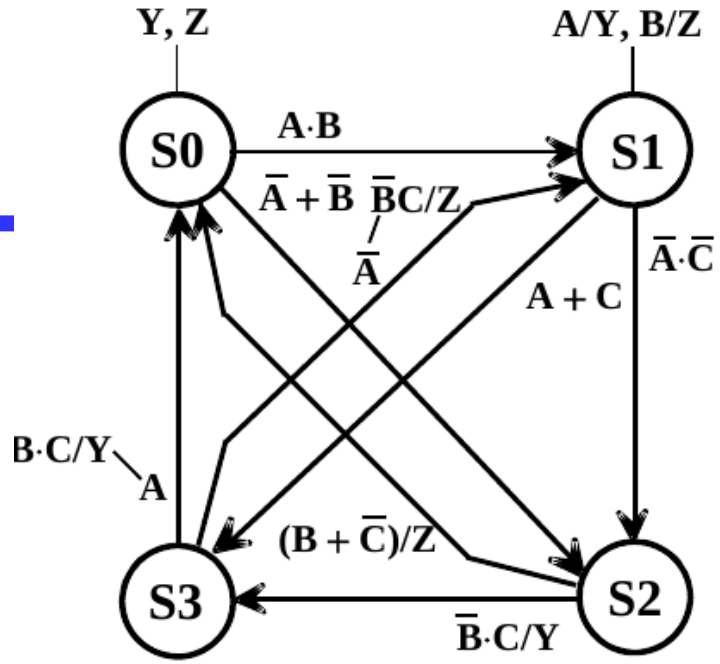
- 状态机图



2. 状态机图模型

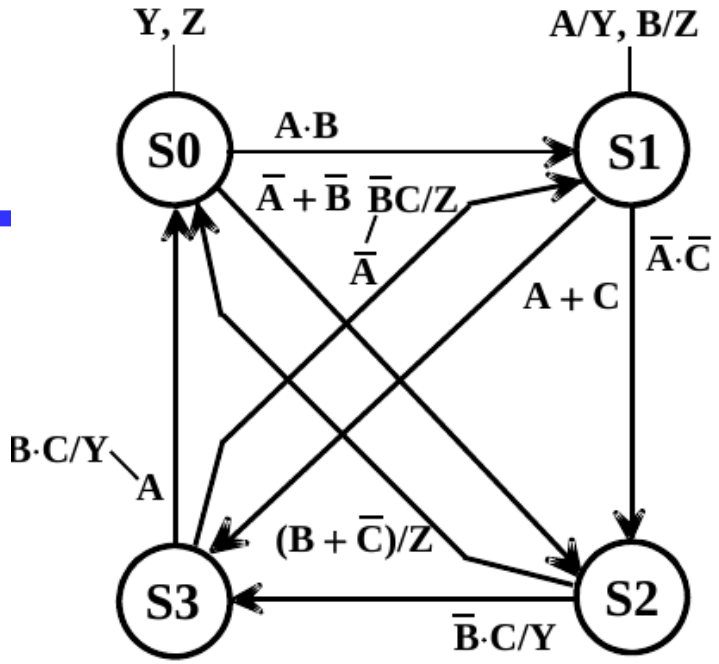
- 状态机表

State	State Code	Transition Condition	Next State	Next State Code	Output Actions (and OCs)
State Name 1	State Code 1	Unused	Unconditional Next State 1	Next State Code 1	Moore or TCI Output (and OC)
		Transition Cond. 11	Next State 11	Next State Code 11	TCD or TOCD Output (and OC)
		Additional Transition Conditions and Entries for State Name 1			
State Name i	Entries for State Names i, i = 2, …n				



State	State Code	Transition Condition	Next State	Next State Code	Output Actions (OCs)
S0	00				Y,Z
		A·B	S1	01	
		$\overline{A} + \overline{B}$	S2	10	
S1	01				A/Y, B/Z
		$\overline{A}\cdot\overline{C}$	S2	10	
		A + C	S3	11	

- *: 依赖转移条件



State	State Code	Transition Condition	Next State	Next State Code	Output Actions (OCs)
S2	10				
		$\overline{B}\cdot C$	S3	11	Y*
		$B+\overline{C}$	S0	00	Z*
S3	11				
		A	S0	00	B·C/Y*
		$\overline{A}$	S1	01	$\overline{B}$·C/$\overline{Z}$*

3. 约束检查

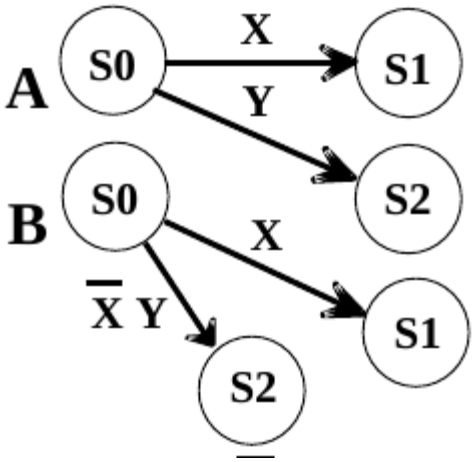
- TC 约束
  - 约束 1: 对状态 S_i, 从S_i出发所有TC 对 (T_{ij}, T_{ik})：

$$T_{ij} \times T_{ik} = 0$$

  - 约束 2: 对状态S_i，对所有TC, T_{ij}:

$$\Sigma\, T_{ij} = 1$$

- 例 A: $X \times Y \neq 0$ 且 $X + Y \neq 1$, 两个约束都违反
- 例 B: $X \times X'Y = 0$, 但 $X + X'Y \neq 1$, 违反约束2



3. 约束检查

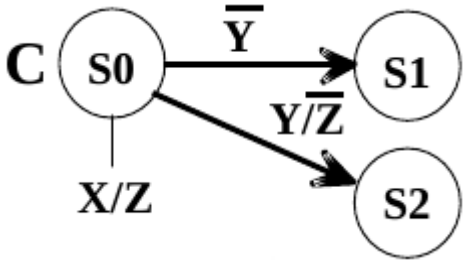
- OC 约束
  - 约束 1: 对状态S_i，在其上或者其状态转移上有一致输出变量但不同值的输出行为，相应的输出条件对 (O_{ij}, O_{ik}) 互斥:

$$O_{ij} \times O_{ik} = 0$$

  - 约束2: 对每个输出变量，在状态 S_i上或者在S_i状态转移上的输出条件必须覆盖所有可能的输入变量组合:

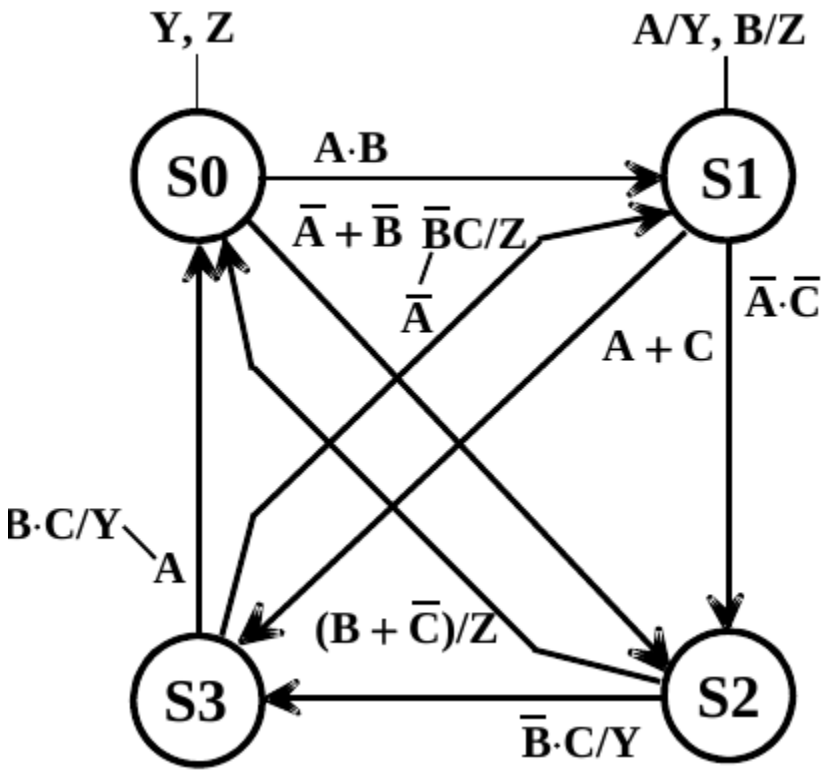
$$\Sigma\, O_{ij} = 1$$

- 例 C: 对 Z = 1 和 Z = 0, $X \times Y \neq 0$, 因此违反约束1
- X + Y + Y' = 1, 在Y'上输出默认值Z'，因此约束2 满足



3. 约束检查

Defaults: Y = 0, Z = 0

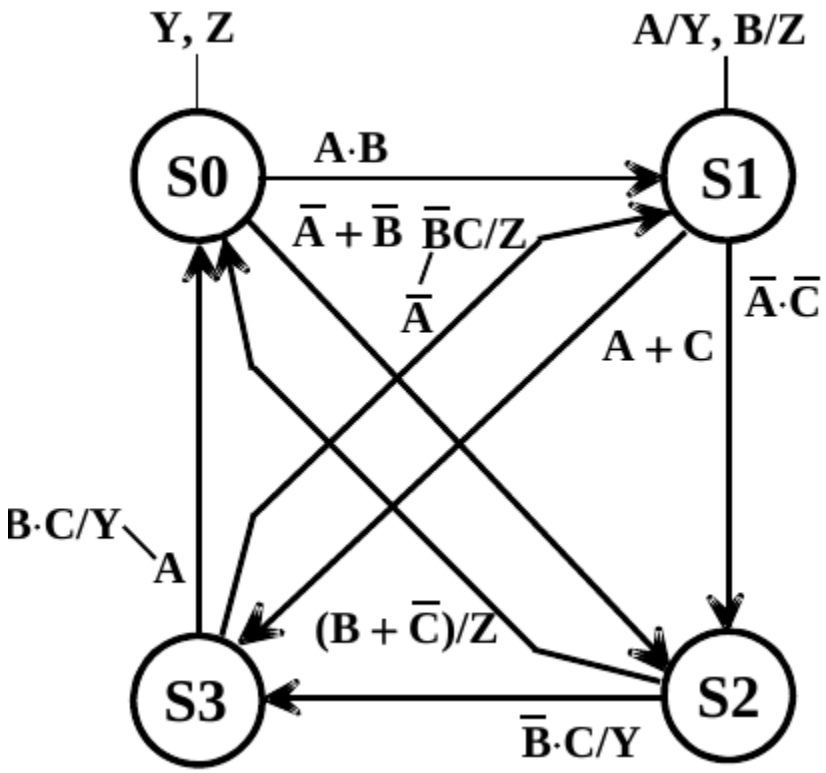


转移约束

- S0: $A\cdot B\cdot(\overline{A}+\overline{B})=0$;
  $A\cdot B+(\overline{A}+\overline{B})=1$
- S1: $\overline{A}\cdot\overline{C}\cdot(A+C)=0$;
  $\overline{A}\cdot\overline{C}+(A+C)=1$
- S2: $\overline{B}\cdot C\cdot(B+\overline{C})=0$;
  $\overline{B}\cdot C+(B+\overline{C})=1$
- S3: $A\cdot\overline{A}=0$;
  $A+\overline{A}=1$

3. 约束检查

Defaults: Y = 0, Z = 0



输出约束

- S0: 没有具有不同输出值的相同输出变量，输出约束满足
- S1:
  - 对于输出条件A，Y=1；对于输出条件A’，Y默认为0
  - 对于B，Z=1；对于B’，Z默认0
- S2:
  - 对于B+C’，Z=1； B ’ C，Z默认0
  - 对于B ’ C，Y=1； B+C’，Y默认0
- S3:
  - 对于ABC，Y=1；其他，Y=0
  - 对于A ’ B ’ C，Z=1；其他，Z=0

3. 约束检查

- 通过状态机表进行约束检测更直观

State	State Code	Transition Condition	Next State	Next State Code	Output Actions (OCs)
S0	00				Y,Z
		A·B	S1	01	
		$\overline{A} + \overline{B}$	S2	10	
S1	01				A/Y B/Z
		$\overline{A} \cdot \overline{C}$	S2	10	
		A + C	S3	11	

3. 约束检查

State	State Code	Transition Condition	Next State	Next State Code	Output Actions (OCs)
S2	10				
		$\overline{B}\cdot C$	S3	11	Y*
		$B + \overline{C}$	S0	00	Z*
S3	11				
		A	S0	00	B·C/Y*
		$\overline{A}$	S1	01	$\overline{B}\cdot C$/Z*

4. 状态机图设计

① 定义输入输出变量，并定义每个变量0和1意义
② 画出电路状态机图或者写出状态机表
③ 若使用状态机图，将其转换成状态机表
④ 从状态机表，推导出电路下一状态和输出方程

4. 状态机图设计

- 例子：控制滑动门
  - 单向滑动门
  - 有人接近、门框内时自动打开
  - 手动按钮可以打开
  - 没人接近、没人在门框内时自动关闭
  - 关门时有人接近、门框内、门里时自动打开
  - 键控锁，用电控门闩锁门
  - 两个限位开关，确定门完全打开/关闭

4. 状态机图设计

输入输出

Input Symbol	Name	Meaning for Value 1	Meaning for Value 0
LK	锁	Locked	Unlocked
DR	门阻力传感器	Door resistance ≥ 15 lb	Door resistance < 15 lb
PA	接近传感器	Person/object approach	No person/object approach
PP	存在传感器	Person/object in door	No person/object in door
MO	手动打开按钮	Manual open	No manual open
CL	关限位	Door fully closed	Door not fully closed
OL	开限位	Door fully open	Door not fully open

Output Symbol	Name	Meaning for Value 1	Meaning for Value 0
BT	门闩	Bolt closed	Bolt open
CD	关闭门	Close door	Null action
OD	打开门	Close door	Null action

4. 状态机图设计

状态机图

Input Symbol	Name
LK	锁
DR	门阻力传感器
PA	接近传感器
PP	存在传感器
MO	手动打开按钮
CL	关限位
OL	开限位

Output Symbol	Name
BT	门闩
CD	关闭门
OD	打开门

步骤

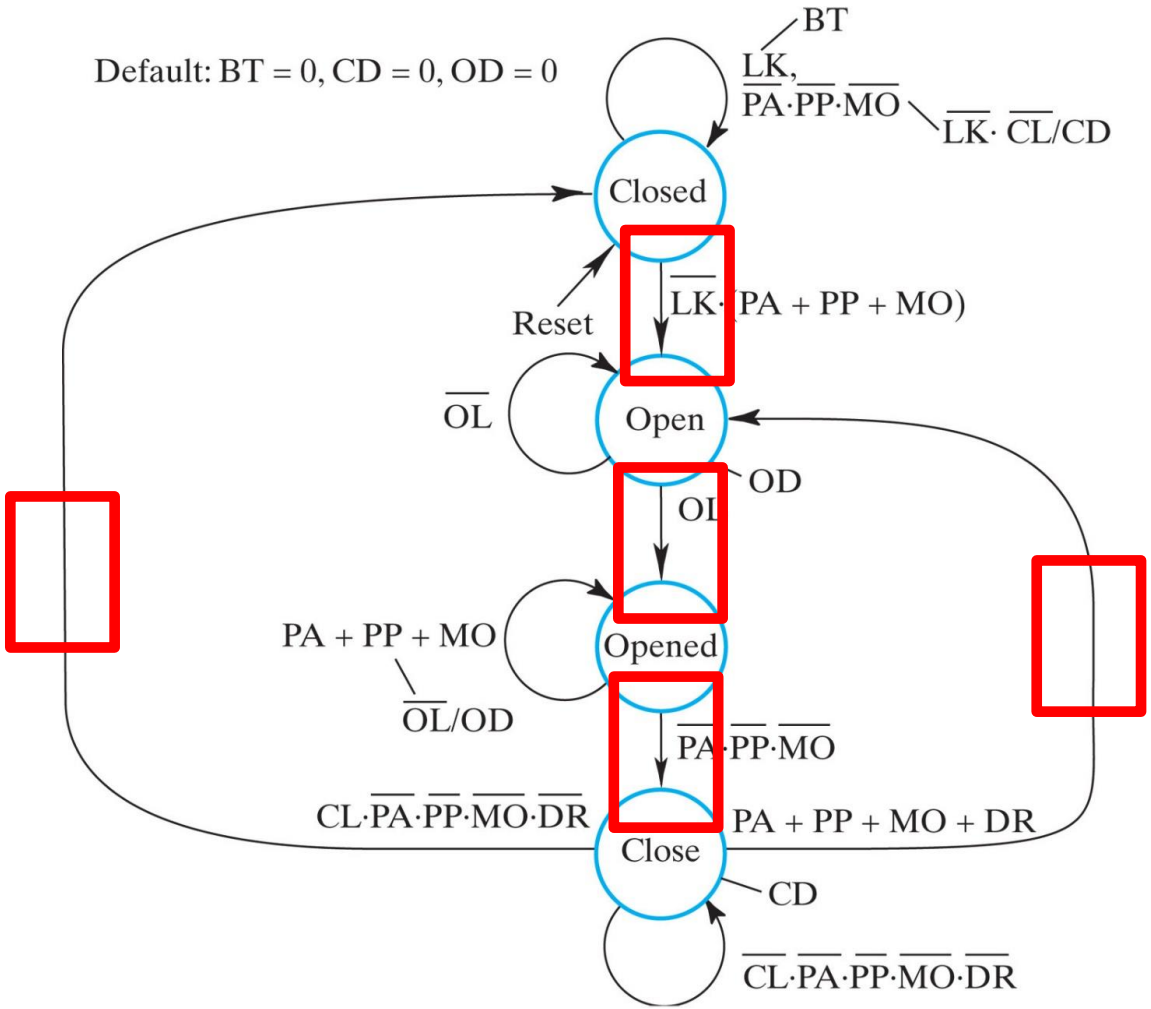
- 先定义状态
  - Closed Open Opened Close
- 主转移路径
  - Closed→Open
  - Open→Opened
  - Opend→Close
  - Close→Closed
  - Close→Open
- 约束检查添加其他转移
- 添加输出

4. 状态机图设计

状态机图

Input Symbol	Name
LK	锁
DR	门阻力传感器
PA	接近传感器
PP	存在传感器
MO	手动打开按钮
CL	关限位
OL	开限位

Output Symbol	Name
BT	门闩
CD	关闭门
OD	打开门



4. 状态机图设计

- 状态机表

State	State Code	Input Condition	Next State	State Code	Non-Zero Outputs (Including TCD and TOCD Output Actions and Output Conditions*)
Closed	00	LK	Closed	00	BT*
	00	$\overline{PA}\cdot\overline{PP}\cdot\overline{MO}$	Closed	00	$\overline{LK}\cdot\overline{CL}/CD^*$
	00	$\overline{LK}\cdot(PA + PP + MO)$	Open	01	
Open	01				OD
	01	$\overline{OL}$	Open	01	
	01	OL	Opened	11	
Opened	11	PA + PP + MO	Opened	11	$\overline{OL}/OD^*$
	11	$\overline{PA}\cdot\overline{PP}\cdot\overline{MO}$	Close	10	
Close	10				CD
	10	$\overline{CL}\cdot\overline{PA}\cdot\overline{PP}\cdot\overline{MO}\cdot\overline{DR}$	Close	10	
	10	$CL\cdot\overline{PA}\cdot\overline{PP}\cdot\overline{MO}\cdot\overline{DR}$	Closed	00	
	10	PA + PP + MO + DR	Open	01	

4. 状态机图设计

- 方程

State	State Code	Input Condition	Next State	State Code	Non-Zero Outputs (Including TCD and TOCD Output Actions and Output Conditions*)
Closed	00	LK	Closed	00	BT*
	00	$\overline{PA}\cdot\overline{PP}\cdot\overline{MO}$	Closed	00	$\overline{LK}\cdot\overline{CL}/CD^*$
	00	$\overline{LK}\cdot(PA + PP + MO)$	Open	01	
Open	01				OD
	01	$\overline{OL}$	Open	01	
	01	OL	Opened	11	
Opened	11	PA + PP + MO	Opened	11	$\overline{OL}/OD^*$
	11	$\overline{PA}\cdot\overline{PP}\cdot\overline{MO}$	Close	10	
Close	10				CD
	10	$\overline{CL}\cdot\overline{PA}\cdot\overline{PP}\cdot\overline{MO}\cdot\overline{DR}$	Close	10	
	10	$CL\cdot\overline{PA}\cdot\overline{PP}\cdot\overline{MO}\cdot\overline{DR}$	Closed	00	
	10	PA + PP + MO + DR	Open	01	

$$X = PA + PP + MO$$

$$Y_1(t+1) = \overline{Y}_1\cdot Y_2\cdot OL + Y_1\cdot Y_2 + Y_1\cdot\overline{Y}_2\cdot\overline{CL}\cdot\overline{X}\cdot\overline{DR}$$

$$Y_2(t+1) = \overline{Y}_1\cdot\overline{Y}_2\cdot\overline{LK}\cdot X + \overline{Y}_1\cdot Y_2 + Y_1\cdot Y_2\cdot X + Y_1\cdot\overline{Y}_2\cdot(X + DR)$$

4. 状态机图设计

- 方程

State	State Code	Input Condition	Next State	State Code	Non-Zero Outputs (Including TCD and TOCD Output Actions and Output Conditions*)
Closed	00	LK	Closed	00	BT*
	00	$\overline{PA}\cdot\overline{PP}\cdot\overline{MO}$	Closed	00	$\overline{LK}\cdot\overline{CL}$/CD*
	00	$\overline{LK}\cdot(PA + PP + MO)$	Open	01	
Open	01				OD
	01	$\overline{OL}$	Open	01	
	01	OL	Opened	11	
Opened	11	PA + PP + MO	Opened	11	$\overline{OL}$/OD*
	11	$\overline{PA}\cdot\overline{PP}\cdot\overline{MO}$	Close	10	
Close	10				CD
	10	$\overline{CL}\cdot\overline{PA}\cdot\overline{PP}\cdot\overline{MO}\cdot\overline{DR}$	Close	10	
	10	$CL\cdot\overline{PA}\cdot\overline{PP}\cdot\overline{MO}\cdot\overline{DR}$	Closed	00	
	10	PA + PP + MO + DR	Open	01	

$$BT = \overline{Y}_1\cdot\overline{Y}_2\cdot LK$$

$$CD = Y_1\cdot\overline{Y}_2 + \overline{Y}_1\cdot\overline{Y}_2\cdot\overline{LK}\cdot\overline{CL}\cdot\overline{X}$$

$$= (Y_1 + \overline{LK}\cdot\overline{CL}\cdot\overline{X})\cdot\overline{Y}_2$$

$$OD = \overline{Y}_1\cdot Y_2 + Y_1\cdot Y_2\cdot\overline{OL}\cdot X$$

$$= (\overline{Y}_1 + \overline{OL}\cdot X)\cdot Y_2$$

4. 状态机图设计

- 例子：电梯控制
- 输入
  - C1(C2)：到1层的Call（梯内)，1 – Call
  - G1(G2)： 到1层的Go，1 – Go
  - F1(F2)：电梯在1层，1 – 电梯在
  - S1(S2)：电梯在接近1层，1 – 电梯在接近
  - DO：门开限位，1 – 门打开
  - TO：等待时间间隔结束，1 – 间隔结束
  - DC：门关限位，1 –门关闭

4. 状态机图设计

- 例子：电梯控制
- 输出
  - Up：控制电梯上行，1-上行
  - Down：控制电梯下行，1-下行
  - TS：计时器开始，1 –启动计时器
  - SD：降速，1 – 电梯降速
  - OD：开门，1 – 开门
  - CD：关门，1 –关门

4. 状态机图设计

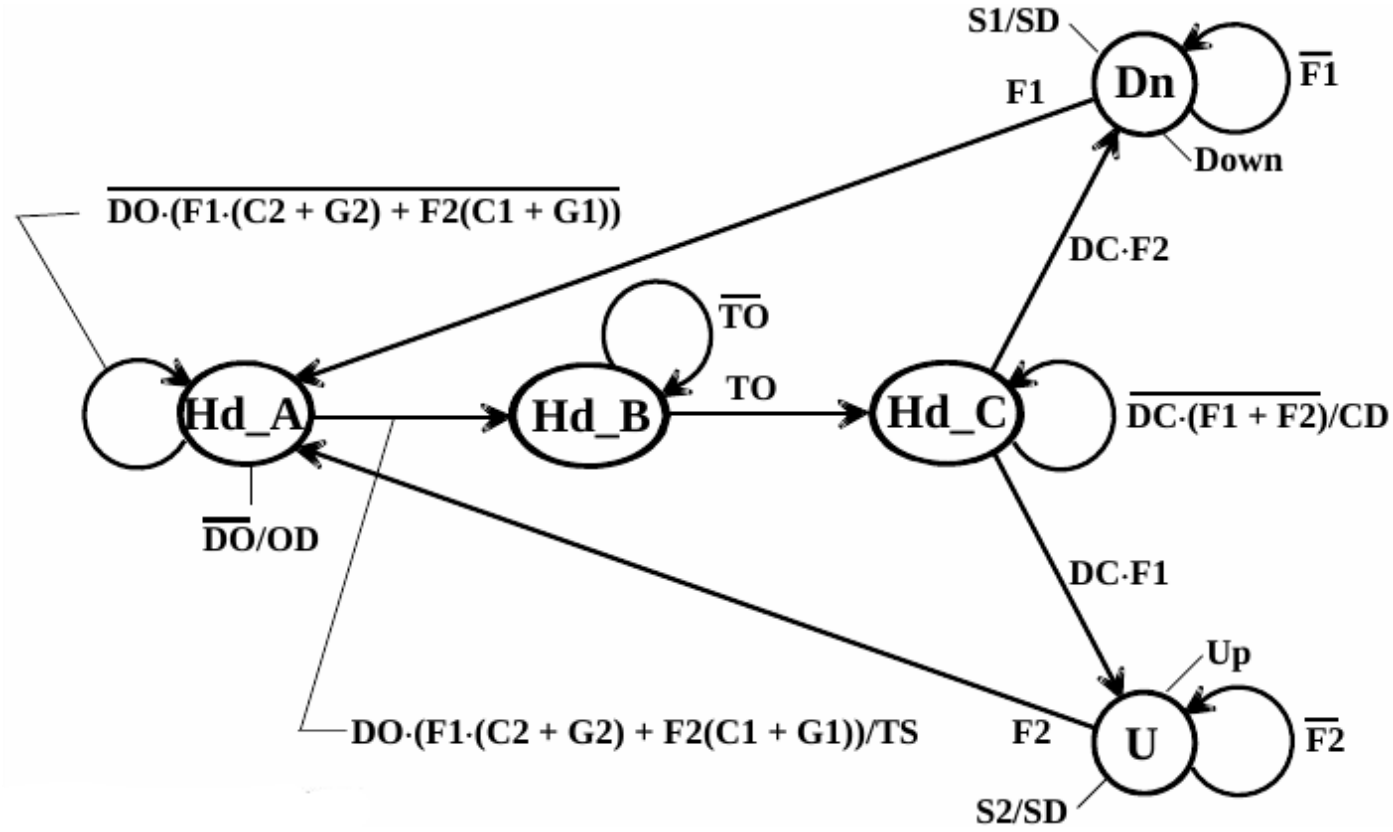
- 例子：电梯控制
- 定义状态
  - 上升：U
  - 下降：Dn
  - 开门：Hd_A
  - 等人：Hd_B
  - 关门：Hd_C

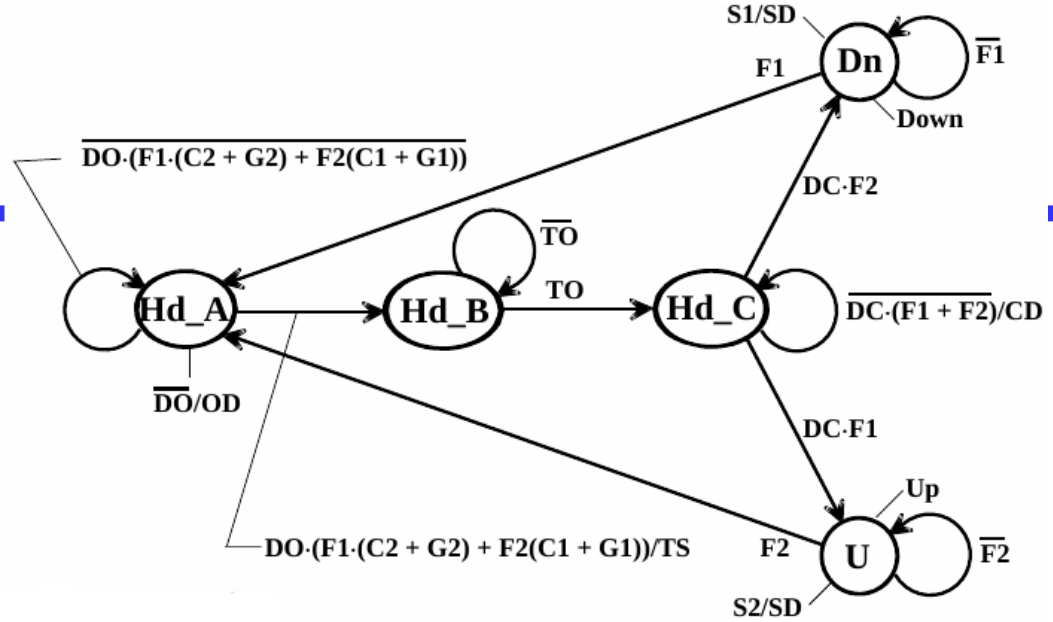
4. 状态机图设计

- 例子：电梯控制
- 主转移路径
  - Hd_A（开门）→Hd_B（等人）
  - Hd_B→Hd_C（关门）
  - Hd_C→Dn
  - Hd_C→U
  - Dn →Hd_A
  - U →Hd_A
- 约束检查添加其他转移
- 添加输出

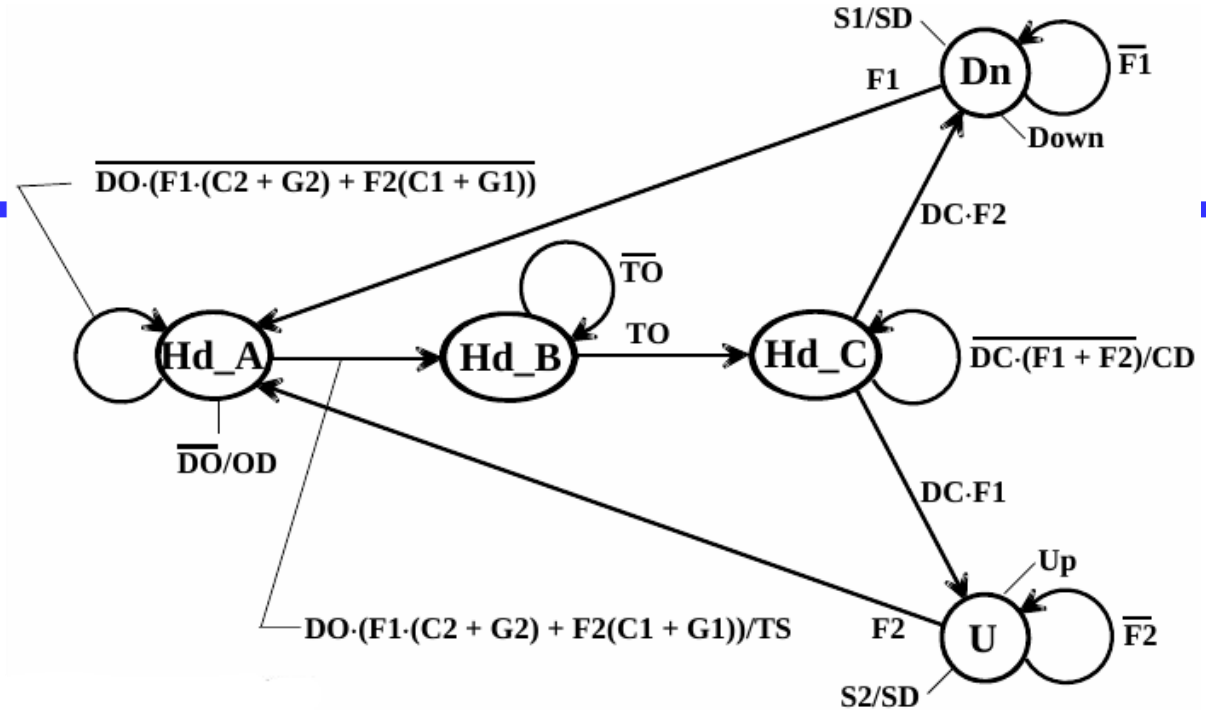
4. 状态机图设计

- 例子：电梯控制
  - 状态机图





State	State Code	Transition Condition	Next State	Next State Code	Output Actions (OCs)
Hd_A	00001				$\overline{DO}$/OD
		$\overline{(DO \cdot (F1 \cdot (C2 + G2) + F2 \cdot (C1 + G1))}$	Hd_A	00001	
		DO·(F1·(C2 + G2) + F2 ·(C1 + G1)	Hd_B	00010	TS*
Hd_B	00010	$\overline{TO}$	Hd_B	00010	
		TO	Hd_C	00100	
Hd_C	00100	$\overline{DC \cdot (F1 + F2)}$	Hd_C	00100	CD*
		DC·F2	Dn	01000	
		DC·F1	Up	10000	



State	State Code	Transition Condition	Next State	Next State Code	Output Actions (OCs)
Dn	01000				Down, S1/SD
		$\overline{F1}$	Dn	01000	
		F1	Hd_A	00001	
U					Up, S2/SD
		$\overline{F2}$	Up	10000	
		F2	Hd_A	00001	

State	State Code	Transition Condition	Next State	Next State Code	Output Actions (OCs)
Hd_A	00001				$\overline{DO}$/OD
		$\overline{(DO \cdot (F1 \cdot (C2 + G2) + F2 \cdot (C1 + G1))}$	Hd_A	00001	
		DO·(F1·(C2 + G2) + F2 ·(C1 + G1)	Hd_B	00010	TS*
Hd_B	00010	$\overline{TO}$	Hd_B	00010	
		TO	Hd_C	00100	
Hd_C	00100	$\overline{DC \cdot (F1 + F2)}$	Hd_C	00100	CD*
		DC·F2	Dn	01000	
		DC·F1	Up	10000	

- $X = DO \cdot ((F1 \cdot (C2 + G2) + F2 \cdot (C1 + G1))$
- $Y = DC \cdot (F1 + F2)$
- $D_{Hd\_A} = Hd\_A \cdot \overline{X} + Dn \cdot F2 + U \cdot F1$

State	State Code	Transition Condition	Next State	Next State Code	Output Actions (OCs)
Hd_A	00001				$\overline{DO}$/OD
		$\overline{(DO\cdot(F1\cdot(C2 + G2) + F2\cdot(C1 + G1))}$	Hd_A	00001	
		DO·(F1·(C2 + G2) + F2 ·(C1 + G1)	Hd_B	00010	TS*
Hd_B	00010	$\overline{TO}$	Hd_B	00010	
		TO	Hd_C	00100	
Hd_C	00100	$\overline{DC\cdot(F1 + F2)}$	Hd_C	00100	CD*
		DC·F2	Dn	01000	
		DC·F1	Up	10000	

State	State Code	Transition Condition	Next State	Next State Code	Output Actions (OCs)
Dn	01000				Down, S1/SD
		$\overline{F1}$	Dn	01000	
		F1	Hd_A	00001	
U					Up, S2/SD
		$\overline{F2}$	Up	10000	
		F2	Hd_A	00001	

- $X = DO\cdot((F1\cdot(C2 + G2) + F2\cdot(C1 + G1))$
- $Y = DC\cdot(F1 + F2)$
- $D_{Hd\_A} = Hd\_A\cdot\overline{X} + Dn\cdot F2 + U\cdot F1$
- $D_{Hd\_B} = Hd\_A\cdot X + Hd\_B\cdot\overline{TO}$
- $D_{Hd\_C} = Hd\_B\cdot TO + Hd\_C\cdot\overline{Y}$
- $D_{Dn} = Hd\_C\cdot DC\cdot F2 + Dn\cdot\overline{F1}$
- $D_{U} = Hd\_C\cdot DC\cdot F1 + U\cdot\overline{F2}$

- Down = Dn
- Up = U
- $SD = Dn\cdot S1 + U\cdot S2$
- $TS = Hd\_A\cdot X$
- $OD = Hd\_A\cdot\overline{DO}$
- $CD = Hd\_C\cdot\overline{Y}$