西安交通大学
XI'AN JIAOTONG UNIVERSITY

人工智能学院
College of Artificial Intelligence, XJTU

数字设计和计算机体系结构

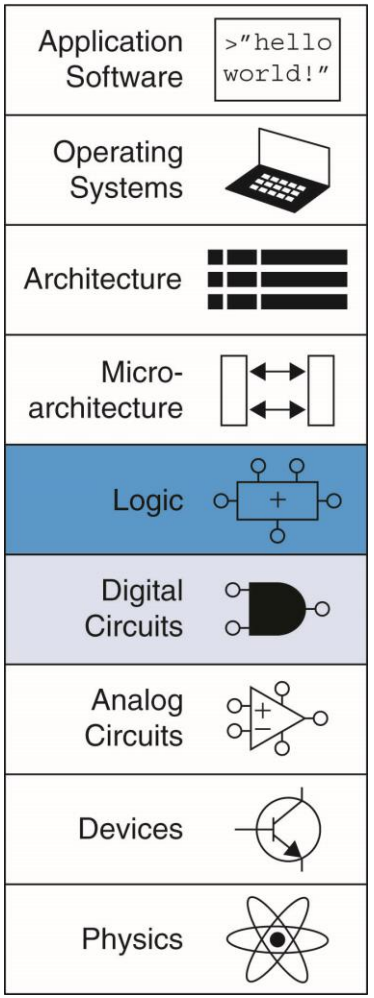
第三章 时序逻辑设计

刘龙军 副教授

2020年10月31日

第三章：时序逻辑设计

- 介绍
- 锁存器与触发器
- 同步逻辑设计
- 有限状态机
- 时序逻辑电路的时序
- 并行



介绍

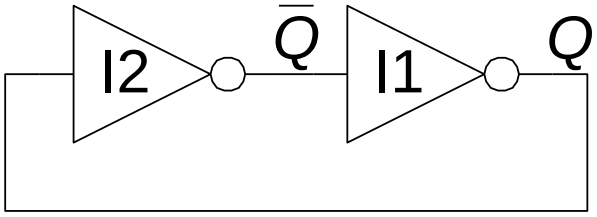
- 时序逻辑的输出与当前值和先前值有关，因此该电路具有记忆状态
- 一些定义:
  - **状态**: 所有先前的输入及关于过去输入的信息，这些信息被称为系统的状态。**数字时序逻辑电路的状态** 由一组称为状态变量的**位**构成
  - **锁存器与触发器**：基本的存储一位状态的简单时序逻辑电路的单元
  - **同步时序电路**: **由组合逻辑和表示电路状态的触发器组成（时序逻辑分析很复杂，简化）**

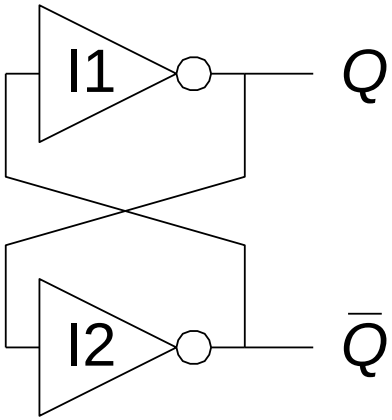
时序逻辑电路（字面 组合）

- 事件发生顺序
- 短期记忆
- 利用从输出到输入的反馈来存储信息

双稳态电路 Bistable Circuit

- 信息存储器件的基本模块 （记忆）
- 两个输出: Q, $\overline{Q}$
- 没有输入



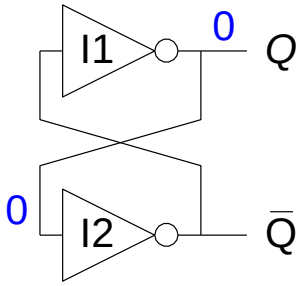


- 组成：连接成环的一对反相器，交叉耦合，输出到对方输入

双稳态电路分析

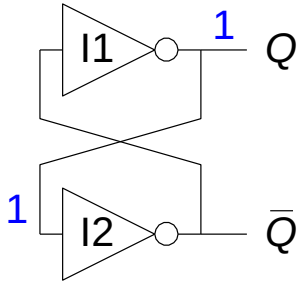
- 考虑两种可能的情况（稳态）：
  - **$Q = 0$:**

    then $Q = 0$, $\bar{Q} =$



  - **$Q = 1$:**

    then $Q = 1$, $\bar{Q} =$

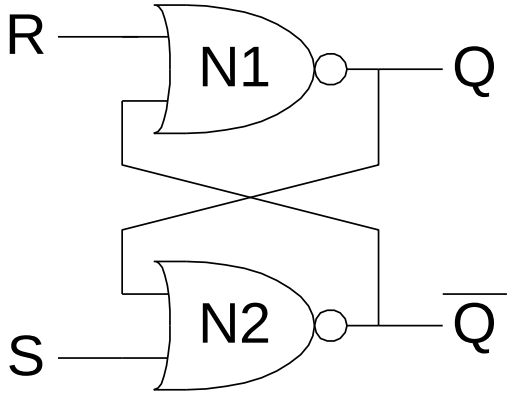


- 存储并记忆1bit 的状态, Q (or $\bar{Q}$)，体会记忆
- **没有输入输出状态**

- 交叉耦合反相器可进入两种稳态，所以该电路成为双稳态电路
- 会不会存在第三种状态？ 亚稳态
- 具有N个稳态的元件可以表示$\log_2 N$位的信息，所以双稳态元件可以存储1位信息。
- 交叉耦合反相器的状态包含在二进制状态变量Q中，Q的值保存了用于解释电路未来行为所需信息
- 该电路是否有实用价值？无输入

SR (Set/Reset) 锁存器

- SR 锁存器

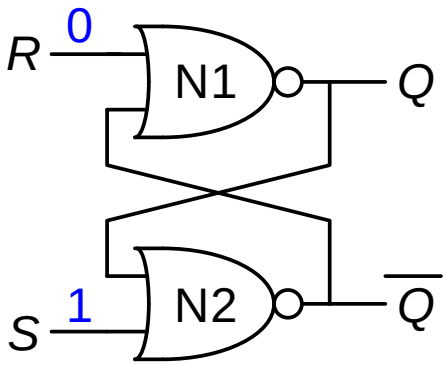


- 最简单的时序电路，一对交叉耦合的或非门组成。
- 两个输入set，reset。两个输出
- 与交叉耦合反向器相似，但可以通过S、R来控制
- $S = 1, R = 0$ ； $S = 0, R = 1$
- $S = 0, R = 0$ ； $S = 1, R = 1$

SR 锁存器状态分析（真值表）

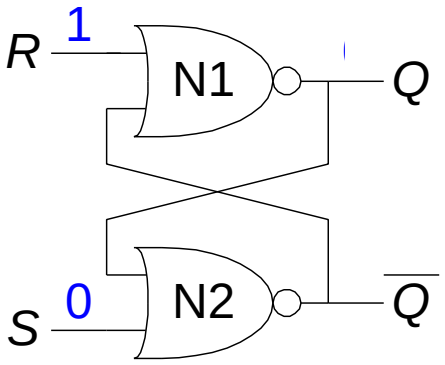
– ***S* = 1, *R* = 0:**

then Q = and $\overline{Q}$ =



– ***S* = 0, *R* = 1:**

then Q = and $\overline{Q}$ =

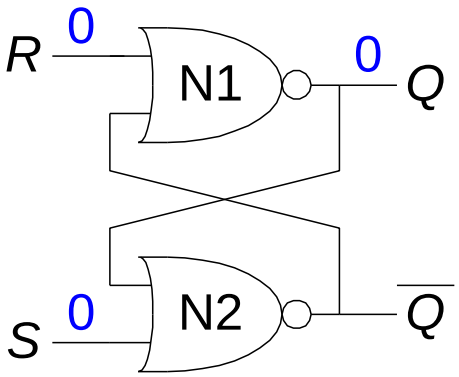


SR锁存器状态分析

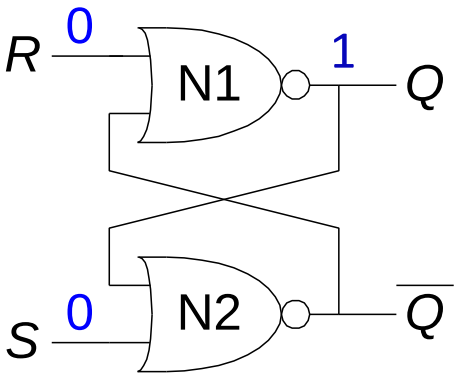
– ***S* = 0, *R* = 0:**

then $Q = Q_{prev}$

$Q_{prev} = 0$

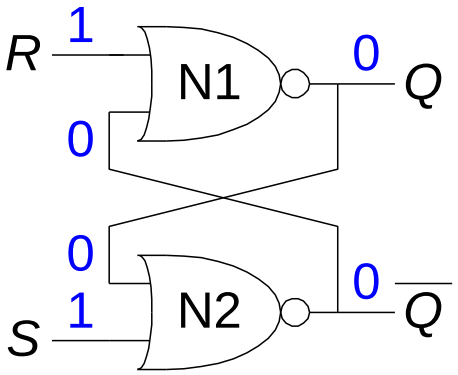


$Q_{prev} = 1$



– ***S* = 1, *R* = 1:**

then $Q = 0, \bar{Q} = 0$



与假设不一致，
自相矛盾

SR锁存器输出真值表

置位 S 为1的情况，

S为0的情况，

R为1的情况，（复位时不能置位）

R为0的情况

SR锁存器实现了1位状态的存储

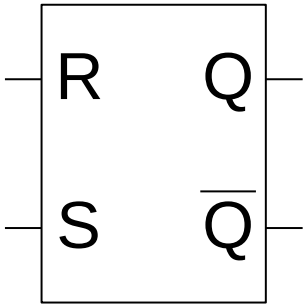
Case	S	R	Q	$\overline{Q}$
IV	0	0	Q_{prev}	$\overline{Q}_{prev}$
I	0	1	0	1
II	1	0	1	0
III	1	1	0	0

SR锁存器输入

- SR stands for Set/Reset Latch
  - Stores one bit of state (Q)
  - 通过S与R的输入控制着什么值被存储
  - **Set:** 置位即表示设置为1
    (S = 1, R = 0, Q = **1**)
  - **Reset:** 复位及表示输出设为 0
    (S = 0, R = 1, Q = **0**)
- **避免无效状态**
  (when S = R = 1)

SR锁存器符号

SR Latch
Symbol



使用符号表示SR锁存器是抽象化和模块化的一个应用。有很多方法可以构造SR锁存器，例如不同的逻辑门或者晶体管。

因此，只要满足功能真值表和构建相应的电路元件即可成为SR锁存器

SR锁存器小结

SR锁存器是一个在Q上存储1位状态的双稳态元件

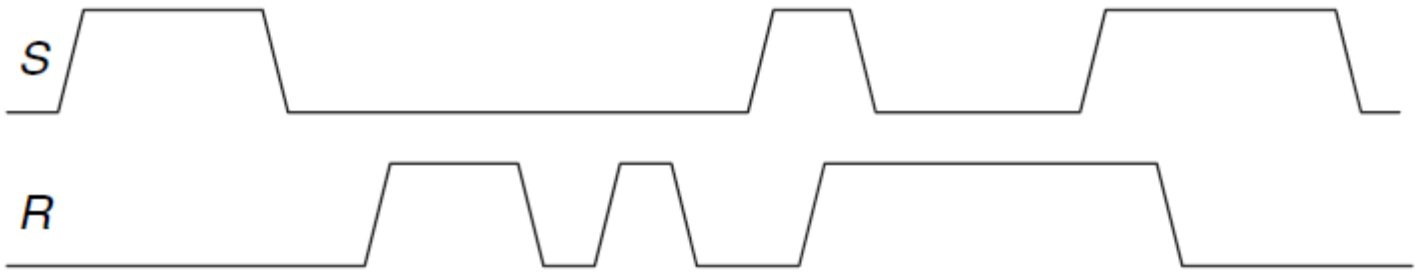
与交叉耦合反相器一样，但状态可通过输入S和R控制

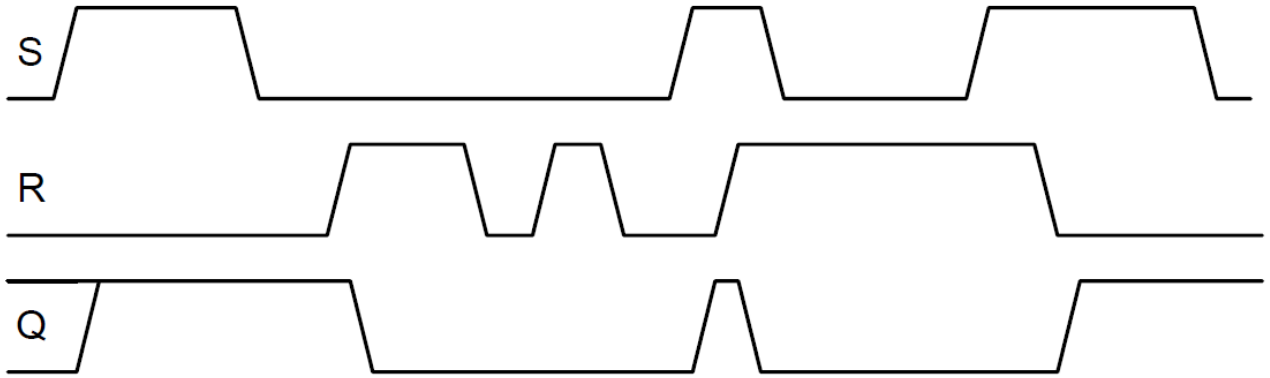
当R有效（复位了），状态复位为0
当S有效（重新置位）,状态位置位1
当S和R都无效（0），状态保持旧值不变。

输入的全部历史可以由状态变量Q解释，

无论过去置位或复位如何发生的，都需要通过最近一次置位或复位操作来预测SR锁存器的未来行为。

S和R同时有效时，其输出不确定，使用起来不方便，当输入有效时（非同时有效），得确定何时改变的内容。

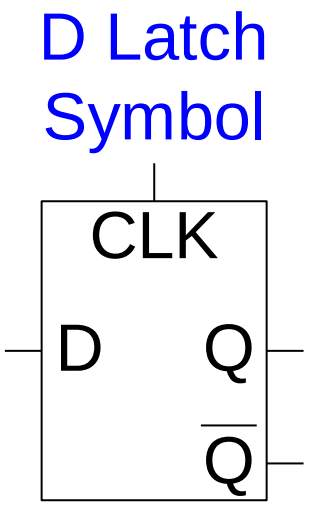

S
R


S
R
Q

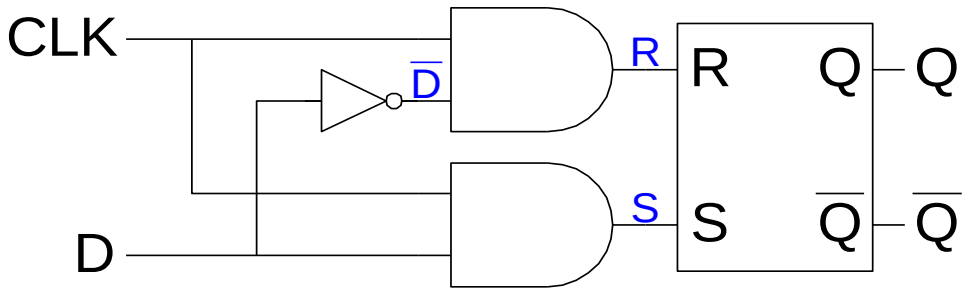
D 锁存器Latch

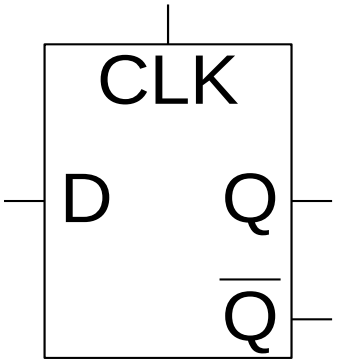
- 两个输入: ***时钟****clk*和数据*Data*
  - ***CLK***: 用来控制状态发生改变的时间
  - ***D*** (the data input): 用来控制下一个状态值
  - 功能
  - 当 ***CLK* = 1**,
    D 直通到*Q* (*透明*)
  - 当***CLK* = 0**,
    Q 保持以前的值 (*不透明，D被挡了*)
- 避免无效情况 when
  $Q \neq \text{NOT } \overline{Q}$

怎么设计？？？？自己练习想想！



D锁存器内部电路





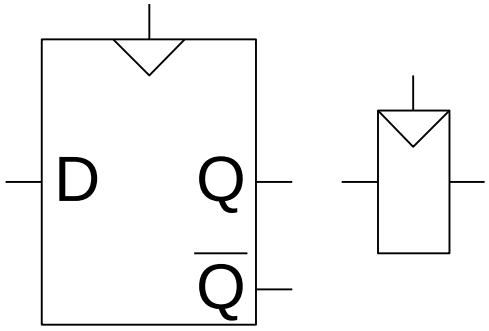
CLK	*D*	$\overline{D}$	*S*	*R*	*Q*	$\overline{Q}$
0	X					
1	0					
1	1					

思考D锁存器比起SR锁存器的优点有哪些？

D 触发器Flip-Flop

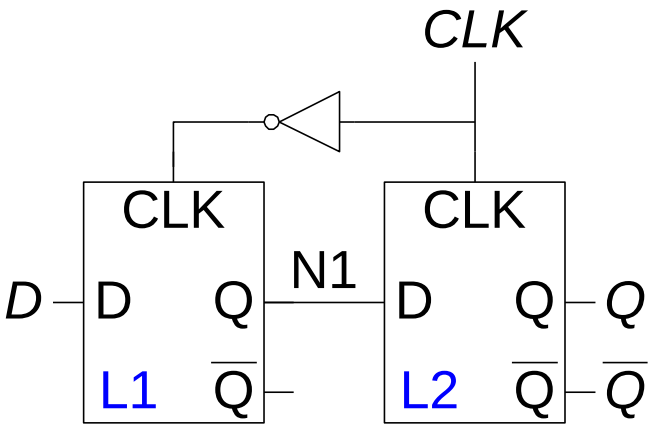
- **Inputs:** 时钟*CLK*, 数据*D*
- **Function**
  - 在CLK的上升沿采样数据*D*
    - 当*CLK* 从 0 to 1上升, *D* 传输到*Q*
    - 否则, *Q* 保持以前的值
  - *Q* 只会在 *CLK* 的上升沿改变
- 被称为上升沿触发
- 时钟沿被激活
- 怎么设计？？？自己想想！！

D Flip-Flop Symbols



D 触发器内部电路

- 一个D触发器由两个反向时钟控制两个串联的D锁存器(L1 and L2)
- 当 ***CLK* = 0**
  - L1 t
  - L2
  - *D*传输到 to N1
- 当***CLK* = 1**
  - L2 t
  - L1
  - N1 传输到 *Q*
- 因此, 在时钟的上升沿 (when ***CLK* rises from 0→1**)
  - *D* 传输到 *Q*



D 触发器

当CLK=0，主锁存器是透明的，从锁存器不透明，D被传输到N1

当CLK=1，主锁存器不是透明，从锁存器是透明，N1被传输Q。但N1到D被切断

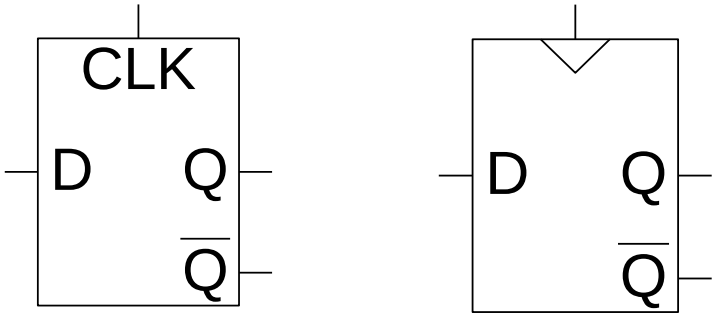
时钟CLK从0到1之前在时钟变为1之后，D值被复制到Q，其他任何时刻Q保持。

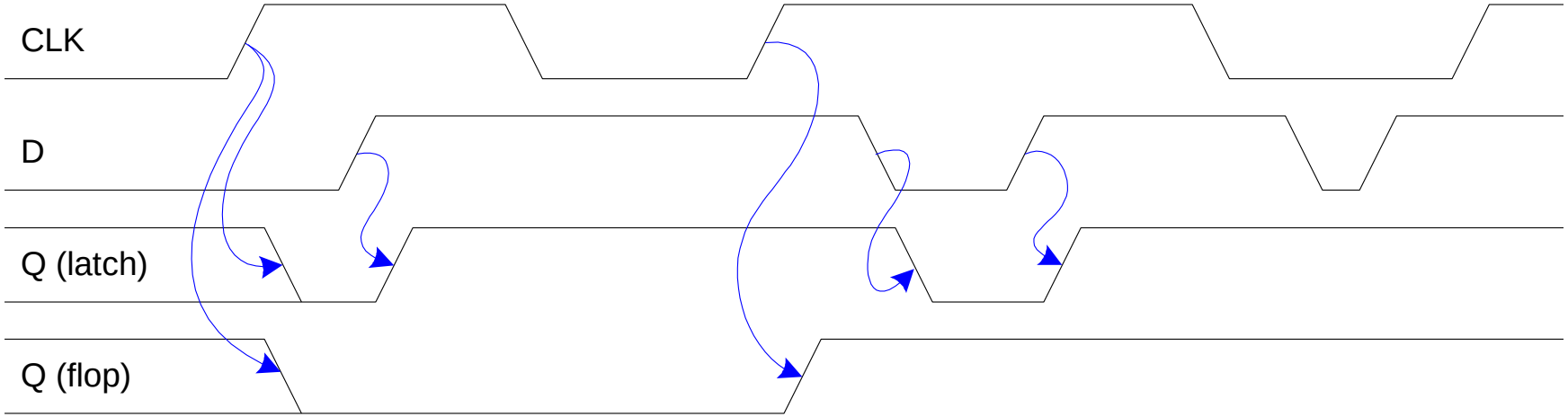
时钟上升沿采数据

输入D确定新的状态----D触发，时钟沿确定状态发生改变的时间

主从触发器、边缘触发器、正边缘触发器。

D 锁存器 vs. D 触发器

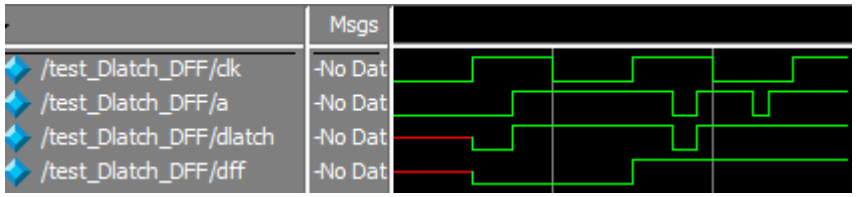




D锁存器D触发器systemverilog实现

主模块：

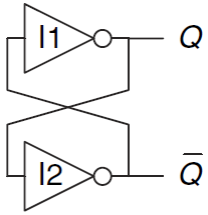
```
module DLatch_DFF(clk,a,dlatch,dff);
input  clk,a;
output dlatch,dff;
reg      dlatch,dff;
always @(clk or a)
   if(clk)  dlatch <= a;
always @(posedge clk)
   dff <= a;
endmodule
```

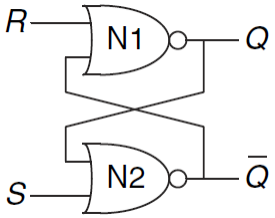


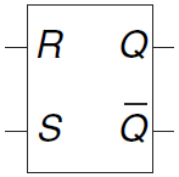
测试模块：

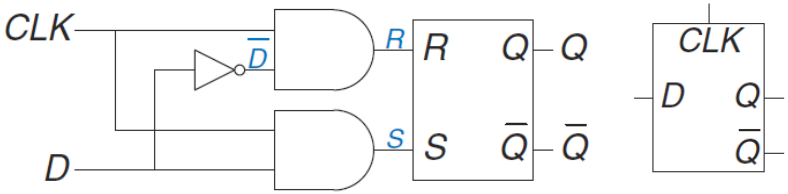
```
module test_Dlatch_DFF;
reg       clk,a;
initial  begin
    a = 1'b0; clk=1'b0; #10 a=1'b0;
    #5   a = 1'b1; #20   a = 1'b0;
    #3   a = 1'b1; #7     a = 1'b0;
    #2   a = 1'b1; #10   $stop;
end
always #10 clk = ~clk;
DLatch_DFF   n1(clk, a, dlatch, dff);
endmodule
```

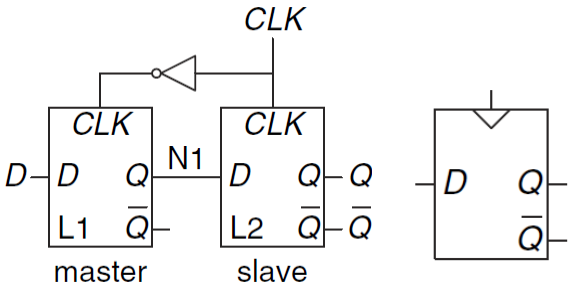
画出：交叉耦合双稳态电路，SR锁存器，D锁存器，D触发器，电路图及符号图（上课练习试试）



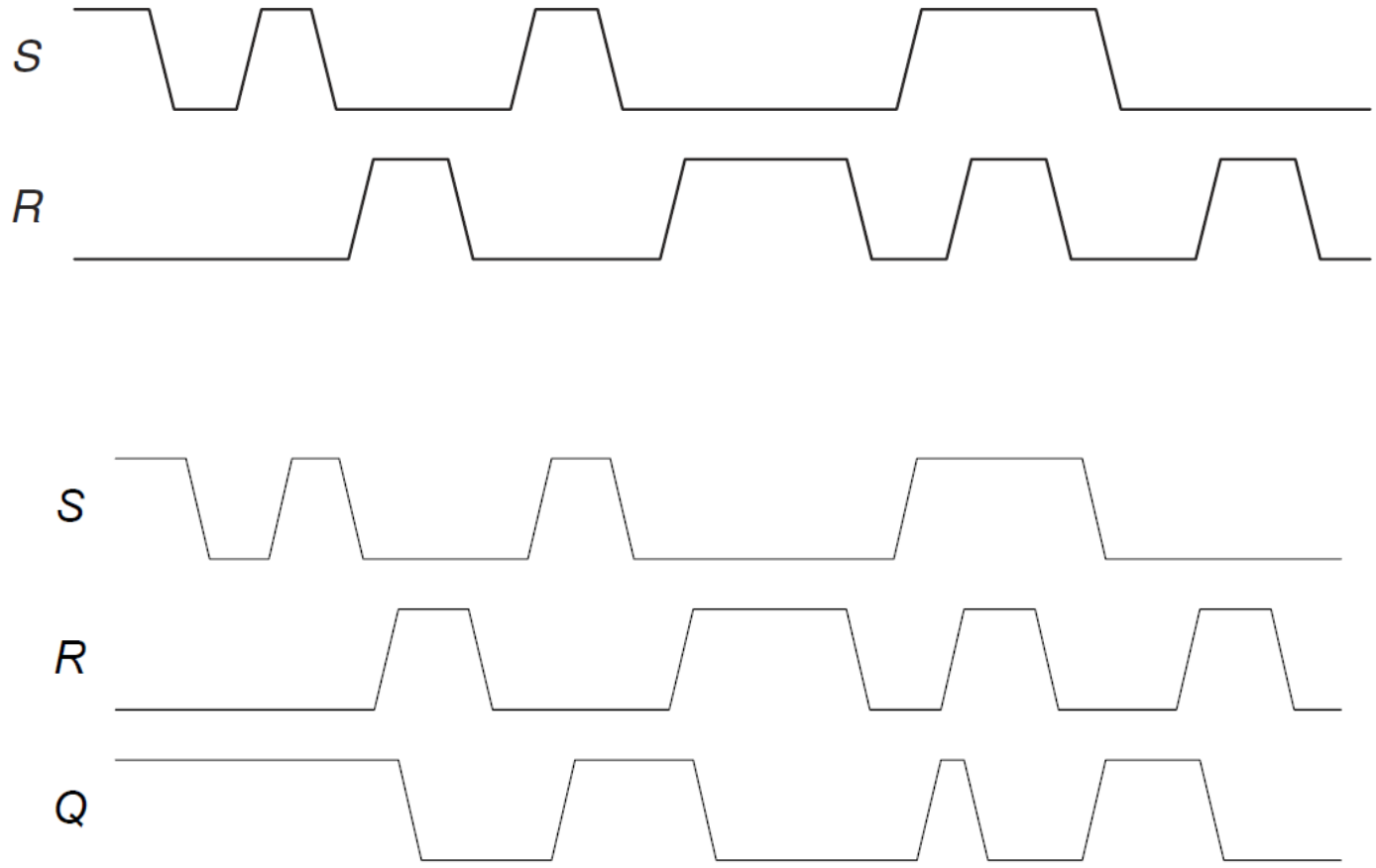




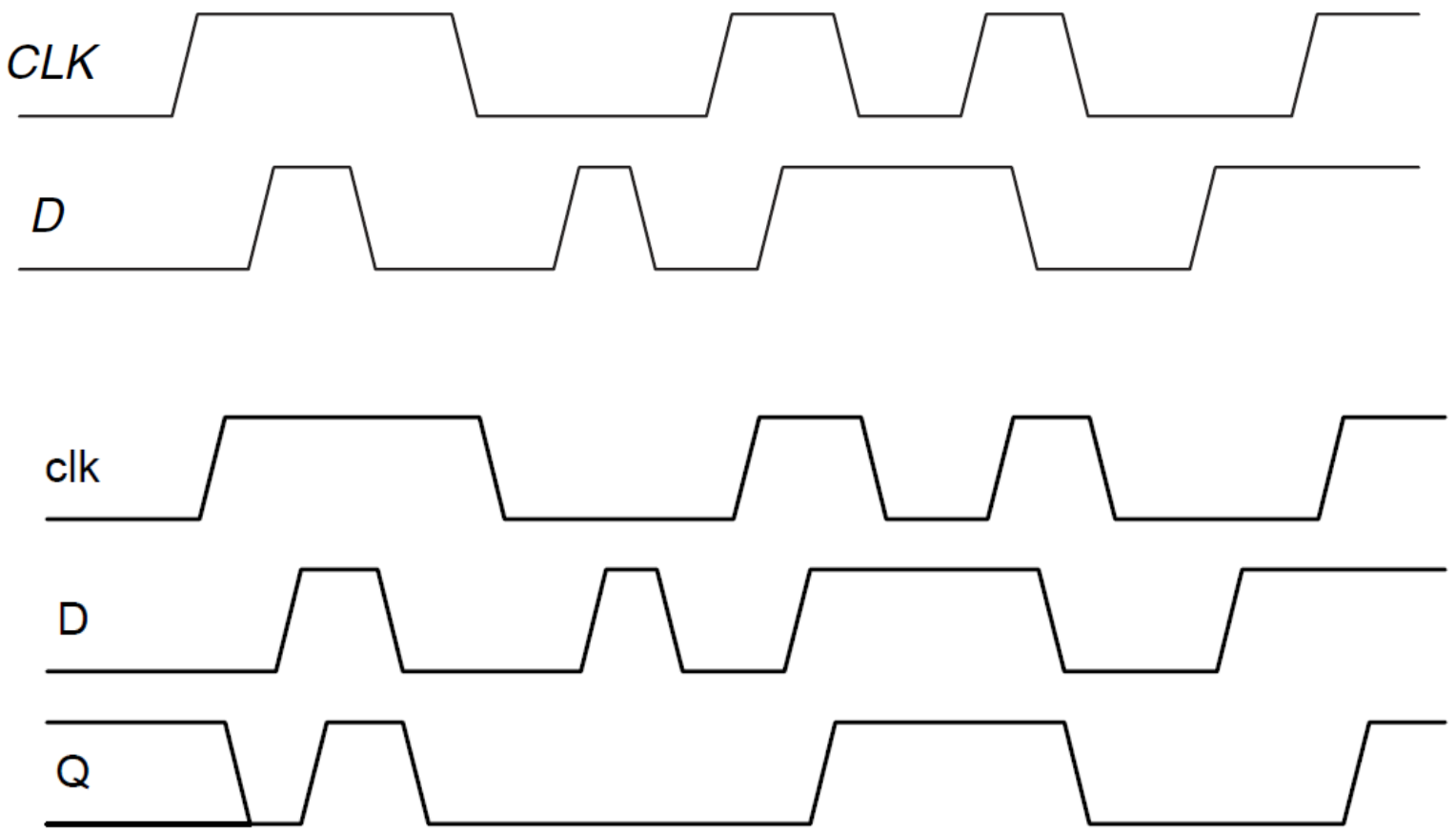




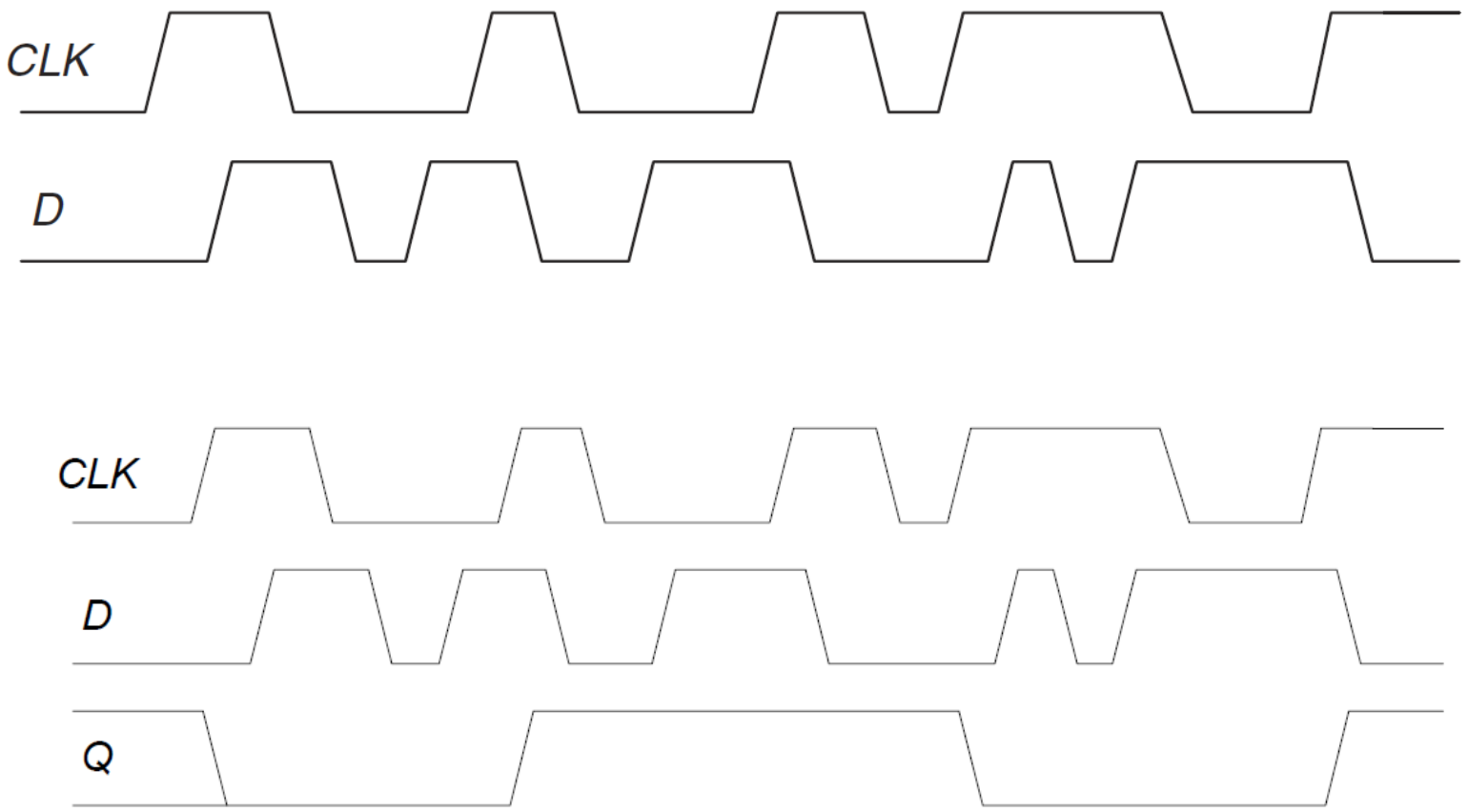
SR锁存器



D锁存器

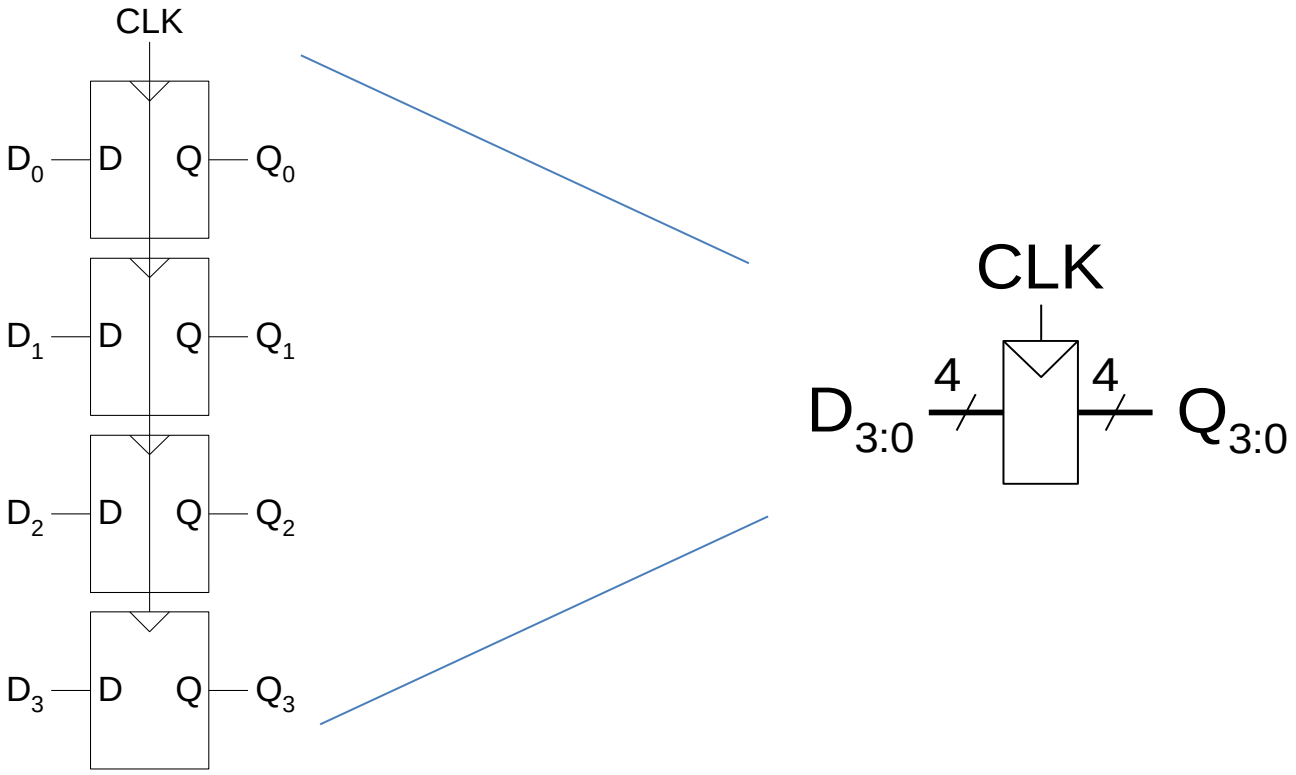


D触发器



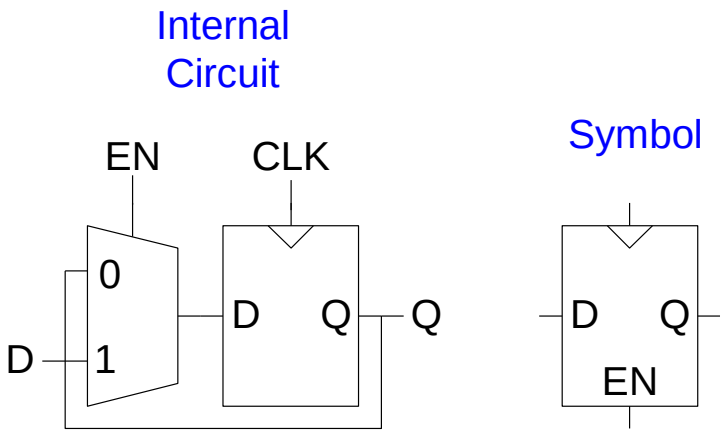
寄存器: 多比特的触发器

共享一个公共CLK输入的一排N个触发器组成，这样寄存器的所有位同时被更新。



带使能端的触发器

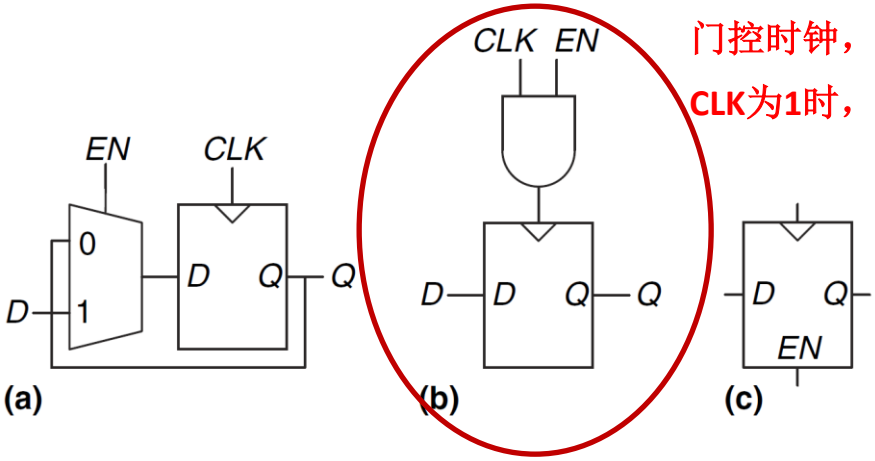
- **输入: *CLK, D, EN***
  - 输入使能信号 (*EN*) 控制着什么时候新数据 (*D*) 被存储
- **功能**
  - ***EN* = 1: 在时钟的边缘，数据*D*传输到 *Q***
  - ***EN* = 0: 保持触发器着以前的值 （环路）**



带使能端的触发器

- 当EN=1时，与普通D触发器一样。
- 当EN=0时，带使能触发器忽略时钟，保持原值，原因是主D触发器的输入是Q输出

当我们在某一时间而不是CLK上升沿载入一个新值到触发器中时，带使能的触发器非常有用.



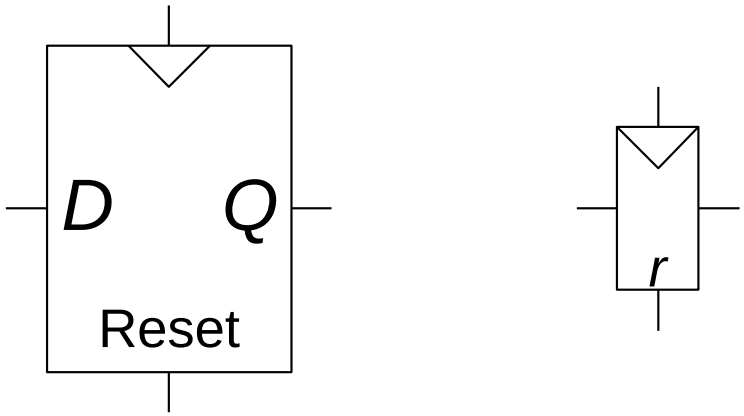
门控时钟，常用于低功耗设计

CLK为1时，EN不能变化，以免产生时钟毛刺

带复位的触发器

- 输入: *CLK*, *D*, *Reset*
- 功能:
  - ***Reset* = 1:** *Q* 被强制置0
  - ***Reset* = 0:** 触发器的功能和一般的 D 触发器一样

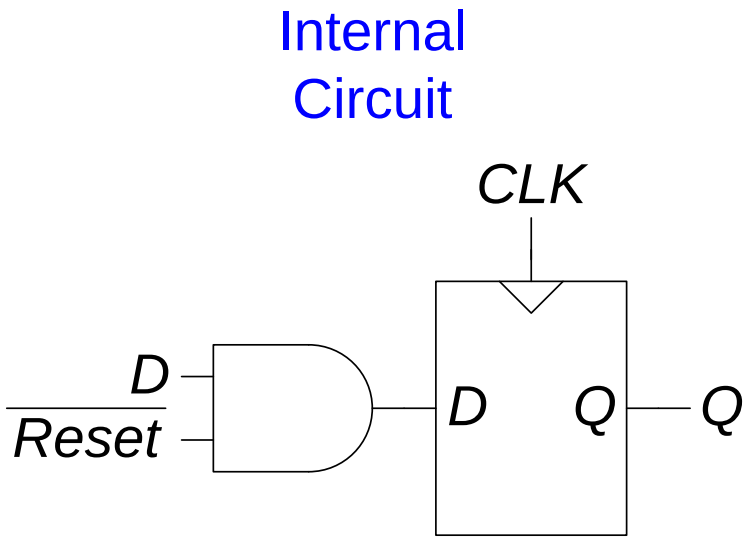
Symbols



复位

- 两种方式:
  - **同步（Synchronous）**：只有在时钟的边缘触发，与时钟同步
  - **异步（Asynchronous）**：一旦 *Reset* = 1，立即复位，与时钟无关
- 异步复位触发器需要修改触发器内部的电路
- Synchronously resettable flip-flop? 不需要修改D触发器内部。

同步复位

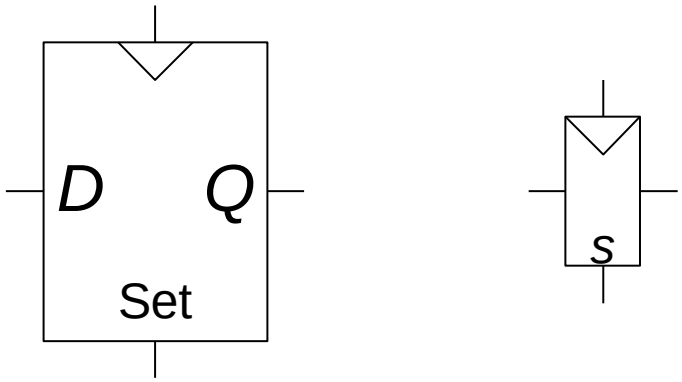


利用D触发器和与门构造带同步复位功能的触发器。

带置位的触发器

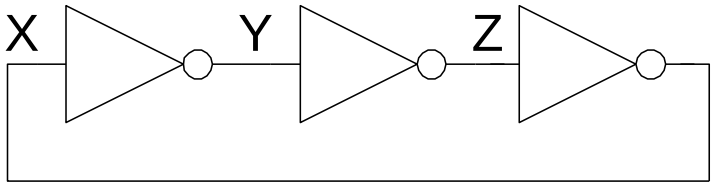
- 输入: *CLK*, *D*, *Set*
- 功能:
  - ***Set* = 1:** *Q* 被设置为1
  - ***Set* = 0:** **触发器的行为与**D 触发器一样

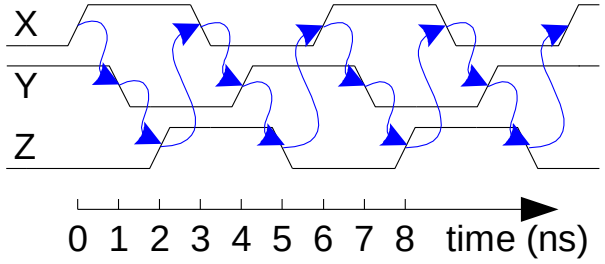
Symbols



时序逻辑电路设计

- 时序电路包括所有不是组合电路的电路
- 怎样设计时序逻辑电路？
- 非稳态电路:





- 无输入
- 振荡
- 振荡周期受制于反相器的延迟
- 环路--输出接到输入

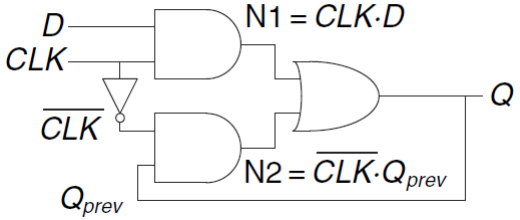
6ns为周期的环形振荡器

时序逻辑电路设计

- 某同学设计了一个新的D锁存器，觉得比以前的锁存器好：使用的门数更少！

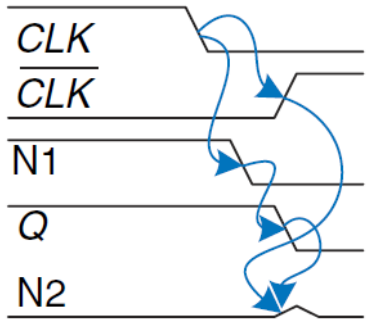
  他写出了真值表和布尔表达式如下，

  请问该锁存器能否正常工作？（延迟不同）



$$Q = CLK \cdot D + \overline{CLK} \cdot Q_{prev}$$

CLK	D	Q_{prev}	Q
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1



假设CLK=D=1，则Q=1；当CLK由1变到0，理论上Q保持1；但如果时钟反相器的延迟比与门和或门的延迟大，则$\overline{CLK}$上升前，节点N1和Q下降为0，节点N2为0，Q将无法变为1.

异步电路（输出反馈到输入）设计，容易出现竞争，因为电路行为取决于两条通过逻辑门的路径中哪条最快。由于延迟问题很多，电路将难以分析。

同步时序电路

避免环路与竞争例子的情况；

- 插入寄存器来**断开环路**，将电路变为组合逻辑与寄存器的电路；
- 寄存器包含系统的状态，状态仅在时钟沿到达时变化，系统被同步到了时钟，状态同步于时钟信号。
- 如果时钟足够慢，下一个时钟沿到来时，输入寄存器的信号都可以稳定下来，这样所有的竞争将被消除。

同步时序电路

- **同步时序电路定义：输入、输出、功能规范、时序规范**
- **同步时序电路（Synchronous Sequential Circuit）：**
  - **输入**: 时钟（上升沿表示状态变化时间）、输入值、当前状态（current state）
  - **功能规范**：对于当前状态和输入值的各种组合，决定下一个状态（next state）和输出值
  - **时序规范**：上界时间(传播延迟)t_{pcq}和下界时间(最小延迟)t_{ccq}（从时钟上升沿直到输出发生改变的时间），建立时间t_{setup}和保持时间t_{hold}（输入相对于时钟上升沿的稳定时间）
  - **输出**：输出值、下一个状态（next state）

同步时序电路

- **同步时序电路的规则：**
  - 每一个电路元件是寄存器或组合电路
  - 至少有一个电路元件是寄存器
  - 所有寄存器都接收同一个时钟信号
  - 每个环路至少包含一个寄存器

非同步的时序电路则称为异步电路（Asynchronous）

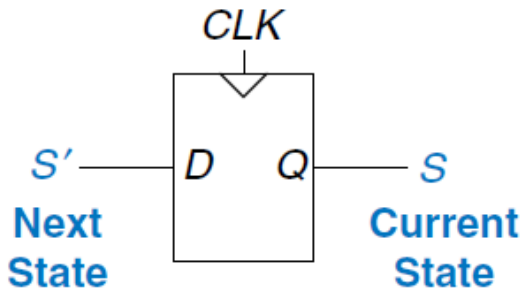
最简单的同步时序电路：单个触发器：

输入：CLK/D；输出：Q；两个状态{0,1}

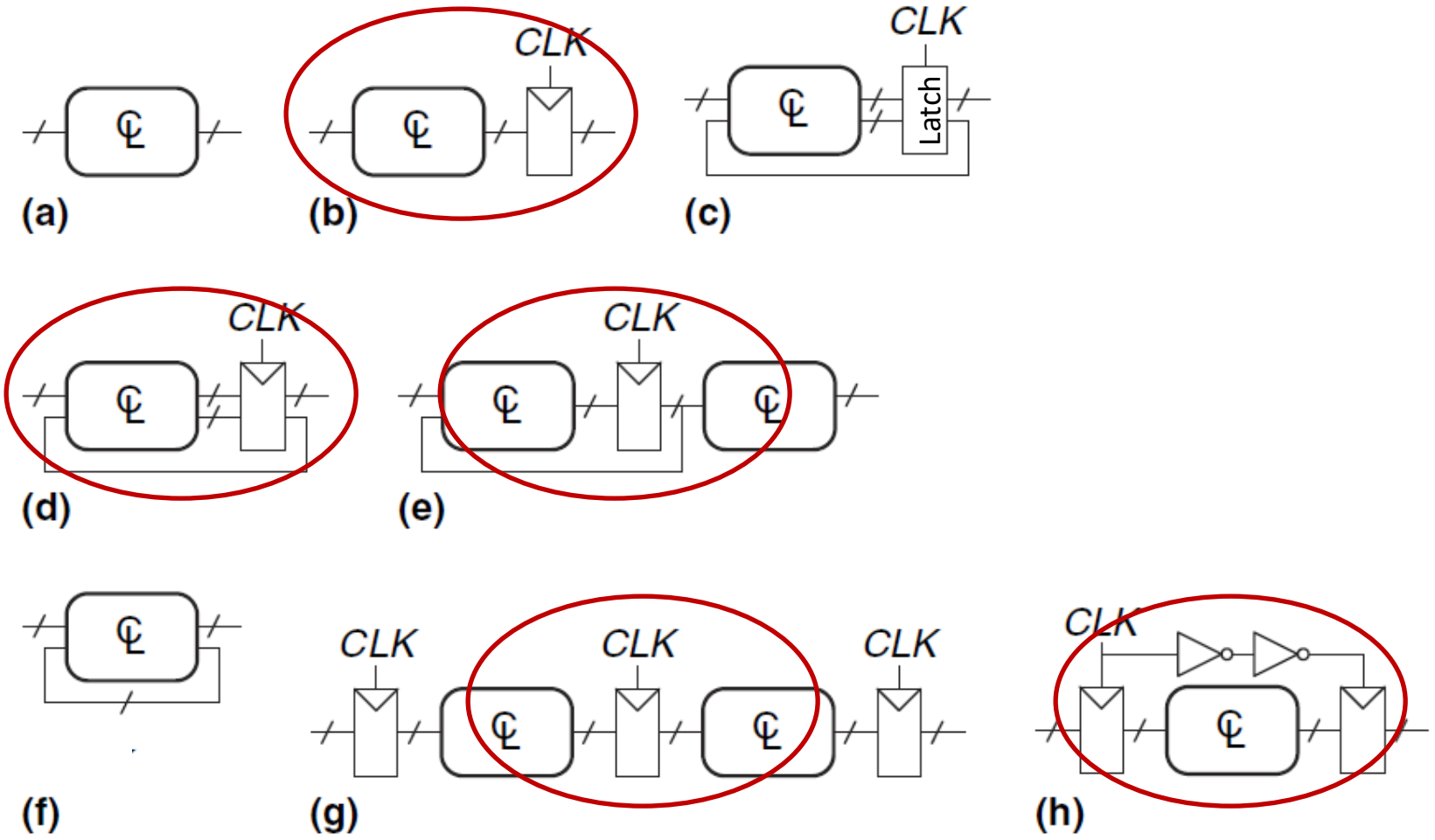
下一个状态是D，输出Q是当前状态。

变量S表示当前状态

变量S‘表示下一个状态



判断同步时序电路



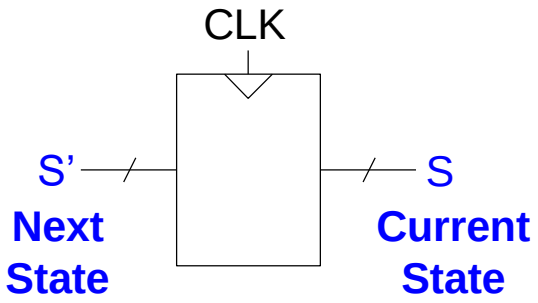
同步电路 VS. 异步电路

- 异步电路设计更通用，时序不受时钟控制的寄存器约束；
- 同步电路设计更容易，更容易设计成大规模的数字电路；
- 目前几乎所有的系统本质上是同步的；
- 跨时钟域（不是同一个时钟源）信号处理涉及异步电路。

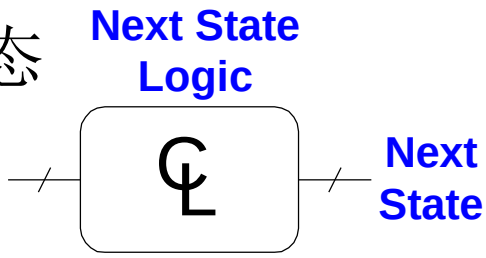
- 两种常见的同步时序电路：
  - 有限状态机（**Finite State Machine (FSM)**）
  - 流水线（**Pipelines**）

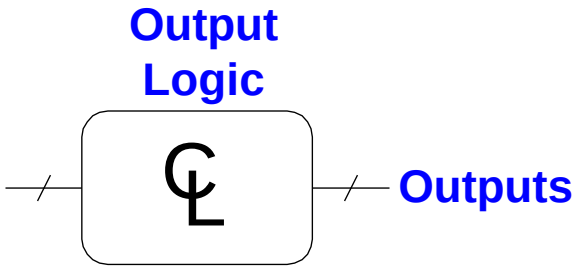
有限状态机 (FSM)

- 有K位寄存器的电路可处于2^K种状态中的某个状态
- 组成:
  - **状态寄存器**
    - S存储当前状态
    - 在时钟沿加载下一状态S‘



  - **组合逻辑**
    - 计算下一状态
    - 计算输出

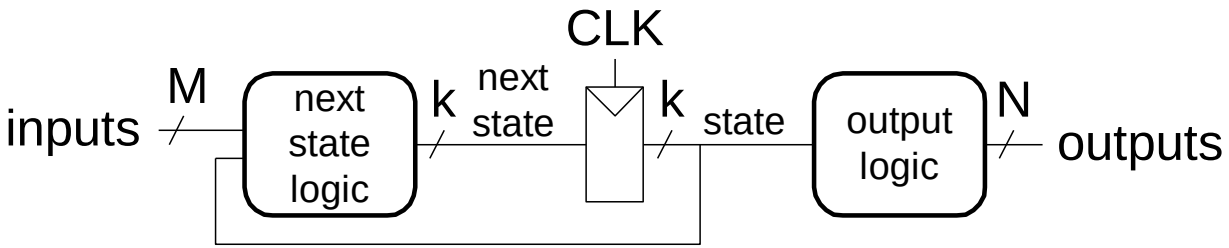




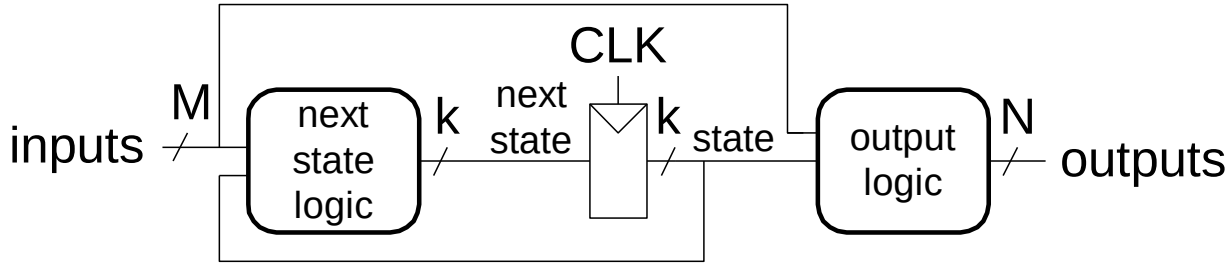
有限状态机 (FSM)

- 根据生成输出值的不同，可分为两类有限状态机：
  - **Moore FSM:** 输出仅取决于当前状态
  - **Mealy FSM:** 输出取决于当前状态和输入值

Moore FSM



Mealy FSM



上节课回顾

- 时序逻辑电路
- 发展过程，工程实际意义？典型电路，典型器件，典型功能
- 内部实现
- 自己设计

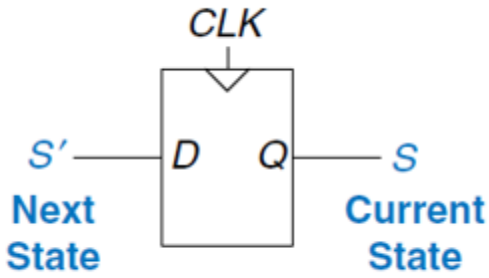
最简单的同步时序电路：单个触发器：

输入：CLK/D；输出：Q；两个状态{0,1}

下一个状态是D，输出Q是当前状态。

变量S表示当前状态

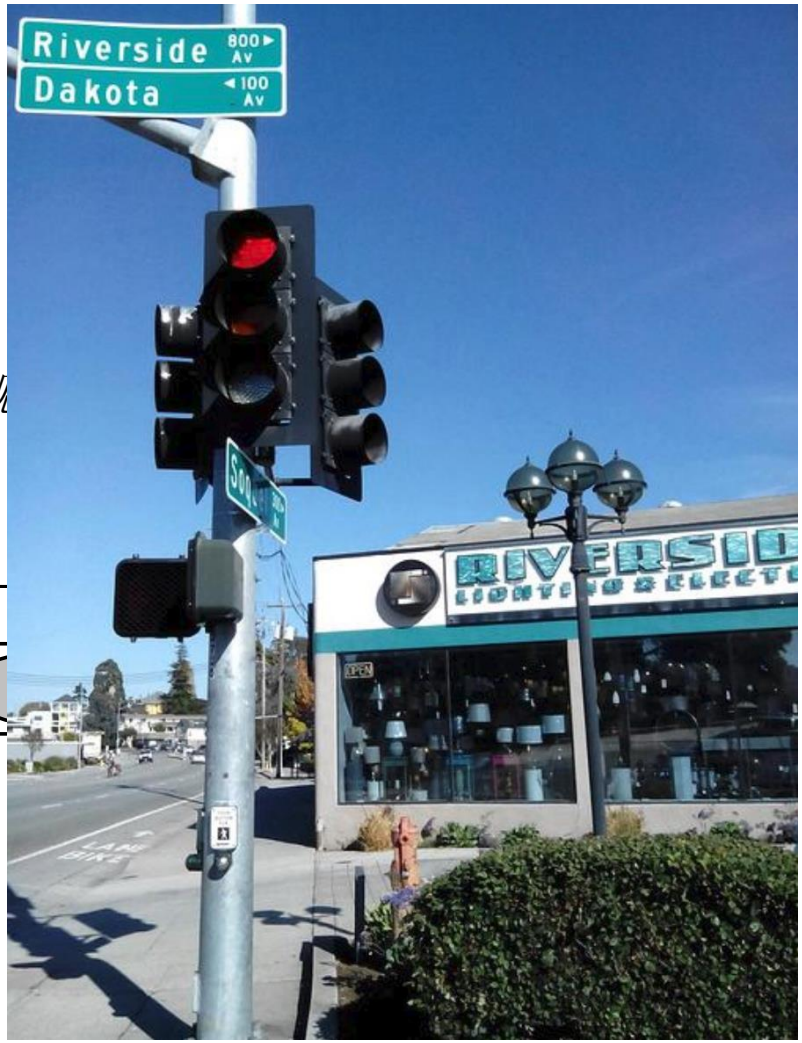
变量S‘表示下一个状态



FSM1 例子

- 交通灯控制器
  - 交通传感器: T_A, T_B
  - 灯: L_A, L_B
  - *传感器输入为TRUE时，表示路上有人出现*
    否则路上无人
  - *交通灯显示 红 绿 黄*

采用一个周期为5秒，每一个时钟周期上升沿，灯将根据交通传感器来改变。

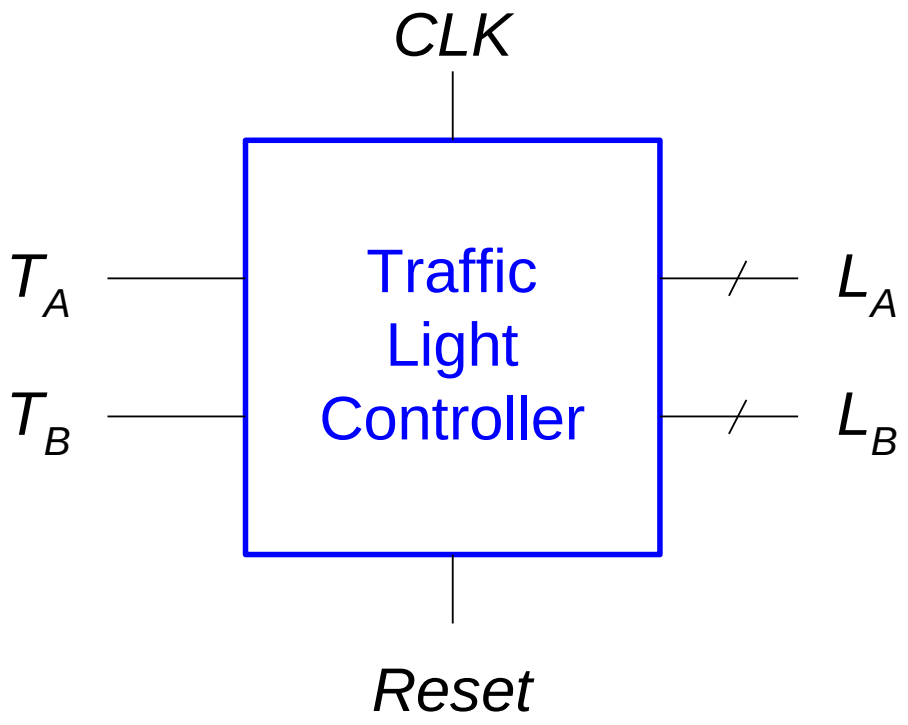


Academic

Labs

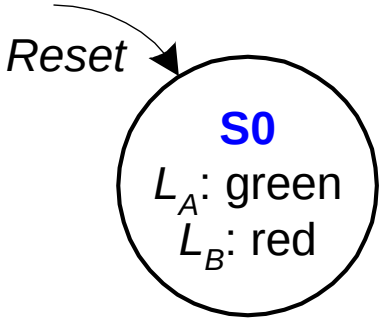
FSM1 黑盒子

- 输入: *CLK*, *Reset*, T_A, T_B
- 输出: L_A, L_B



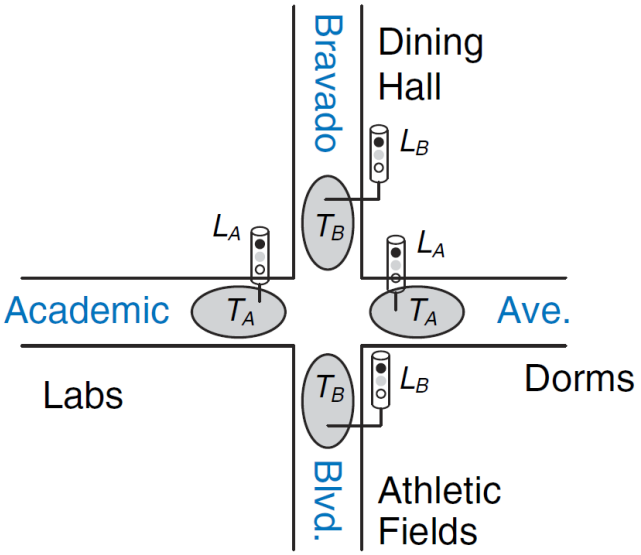
FSM1 状态转换图

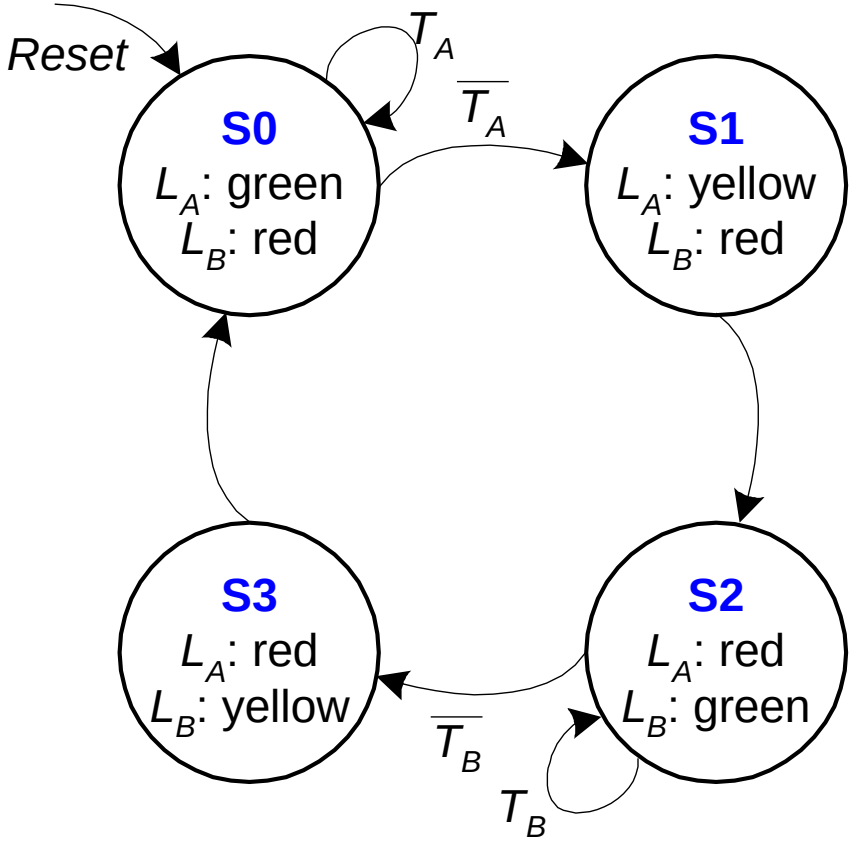
- **Moore FSM:** 输出在每个状态上
- **状态:** 圆圈里
- **转换:** 箭头



FSM1 状态转换图

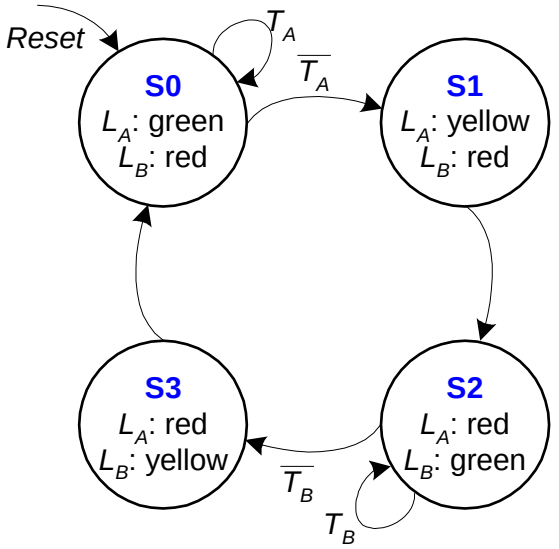
- **Moore FSM:** 输出在每个状态上
- **状态:** 圆圈里
- **转换:** 箭头





FSM1 状态转换表

Current State	Inputs		Next State
S	T_A	T_B	S'
S0	0	X	
S0	1	X	
S1	X	X	
S2	X	0	
S2	X	1	
S3	X	X	



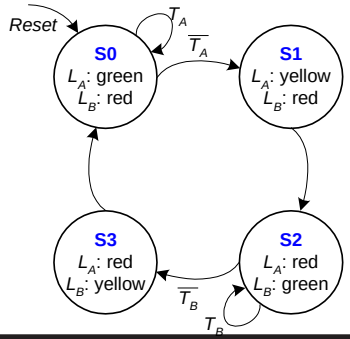
FSM1 编码的状态转换表

Current State		Inputs		Next State	
S_1	S_0	T_A	T_B	S'_1	S'_0
0	0	0	X		
0	0	1	X		
0	1	X	X		
1	0	X	0		
1	0	X	1		
1	1	X	X		

$S'_1 =$

$S'_0 =$

State	Encoding
S0	00
S1	01
S2	10
S3	11



FSM1 输出表

Current State		Outputs			
S_1	S_0	L_{A1}	L_{A0}	L_{B1}	L_{B0}
0	0				
0	1				
1	0				
1	1				

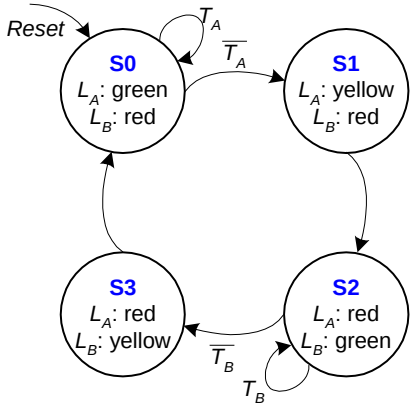
$$L_{A1} =$$

$$L_{A0} =$$

$$L_{B1} =$$

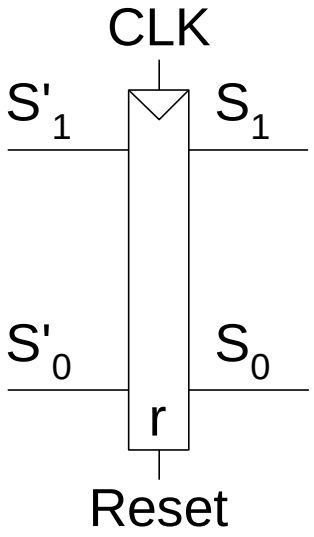
$$L_{B0} =$$

Output	Encoding
green	00
yellow	01
red	10



FSM1：状态寄存器

Current State		Inputs		Next State	
S_1	S_0	T_A	T_B	S'_1	S'_0
0	0	0	X	0	1
0	0	1	X	0	0
0	1	X	X	1	0
1	0	X	0	1	1
1	0	X	1	1	0
1	1	X	X	0	0

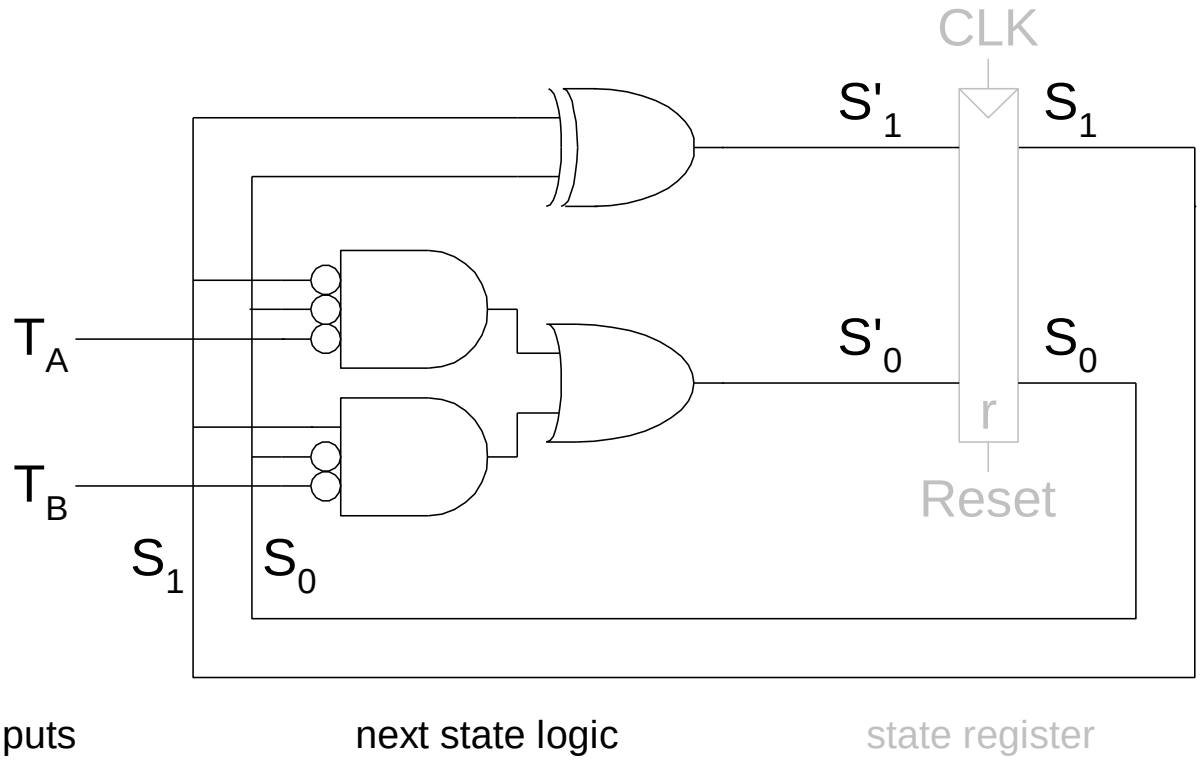


state register

FSM1：下一个状态逻辑

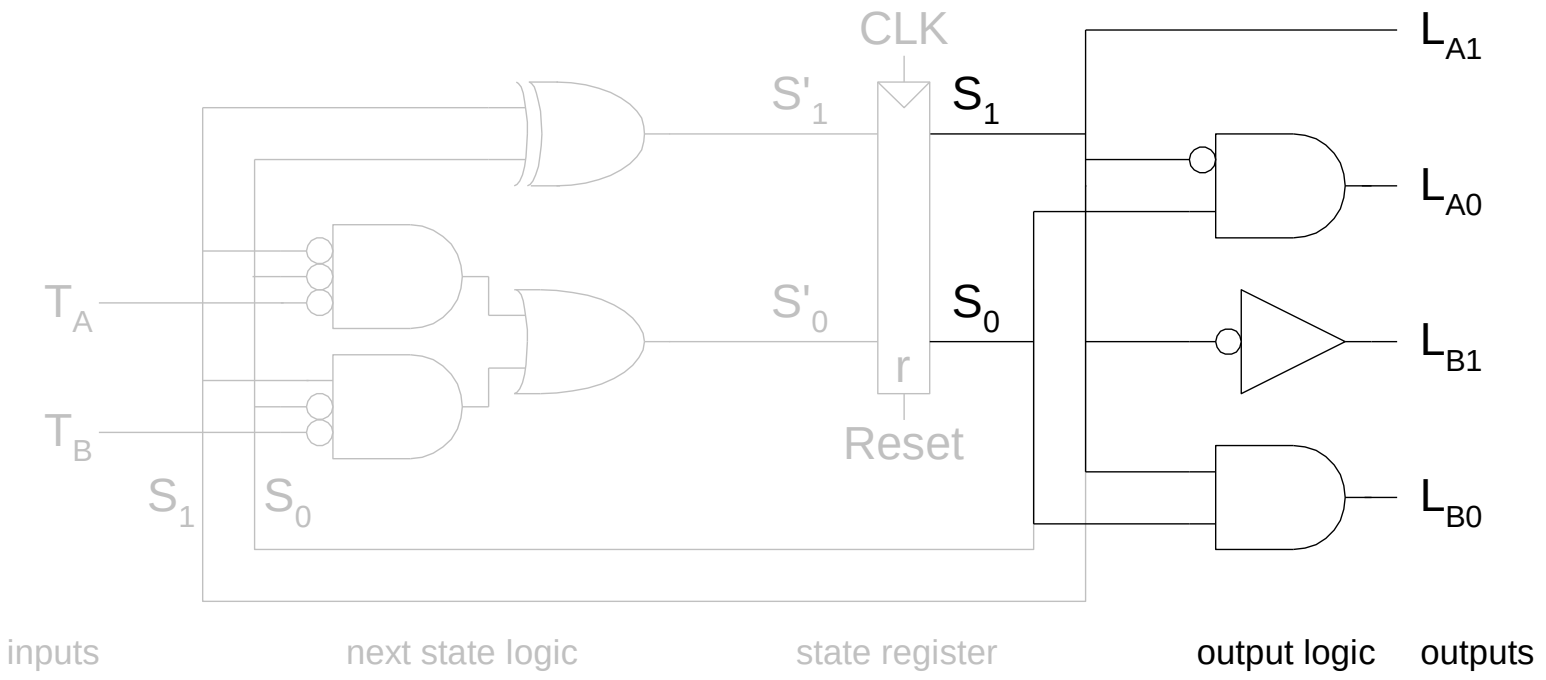
$$S_1' = S_1 \oplus S_0$$

$$S_0' = \overline{S}_1 \overline{S}_0 \overline{T}_A + S_1 \overline{S}_0 \overline{T}_B$$

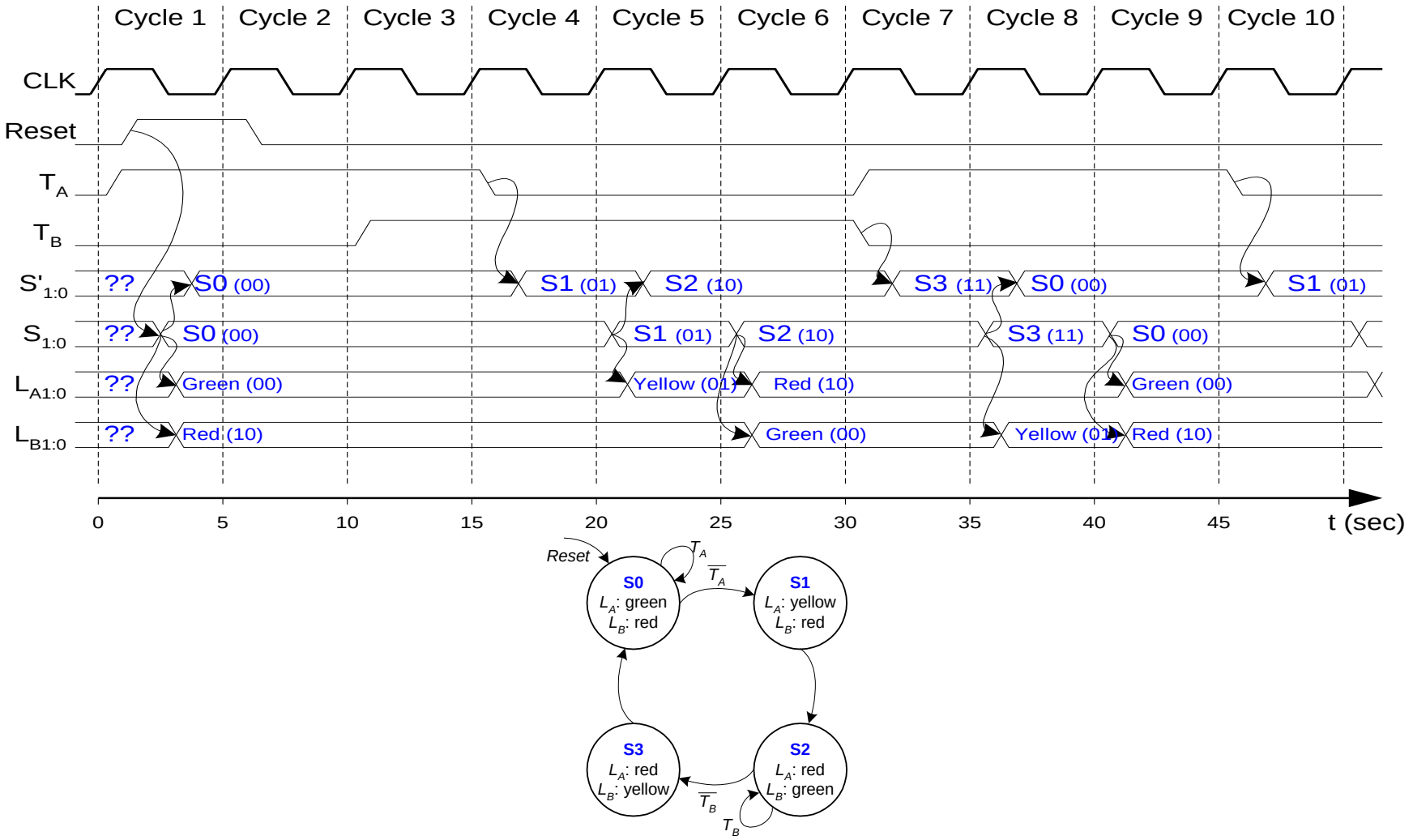


FSM1：输出逻辑

$$L_{A1} = S_1$$
$$L_{A0} = \overline{S}_1 S_0$$
$$L_{B1} = \overline{S}_1$$
$$L_{B0} = S_1 S_0$$



FSM 1时序图 （时序图画法）



FSM1例子的systemverilog实现

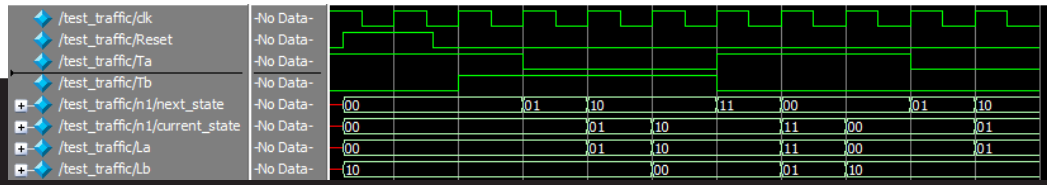
主模块

```
module traffic(clk,Reset,Ta,Tb,La,Lb);
input  clk,Reset,Ta,Tb;
output[1:0]  La,Lb;
parameter S0=2’b00,S1=2’b01,S2=2’b10,S3=2’b11;
reg[1:0]      current_state,next_state,La,Lb;
always_ff @(posedge clk or posedge Reset)
   if(Reset)  current_state <= S0;
   else  current_state <= next_state;
always_comb
   case(current_state)
   S0: if(Ta)  next_state = S0;
        else  next_state = S1;
   S1:  next_state = S2;
   S2:  if(Tb) next_state =  S2;
         else  next_state =  S3;
   S3:  next_state = S0;
   endcase
```

```
always_comb
   case(current_state)
   S0: begin La = 2’b00; Lb = 2’b10; end
   S1: begin La = 2’b01; Lb = 2’b10; end
   S2: begin La = 2’b10; Lb = 2’b00; end
   S3: begin La = 2’b11; Lb = 2’b01; end
   endcase
endmodule
```

测试模块

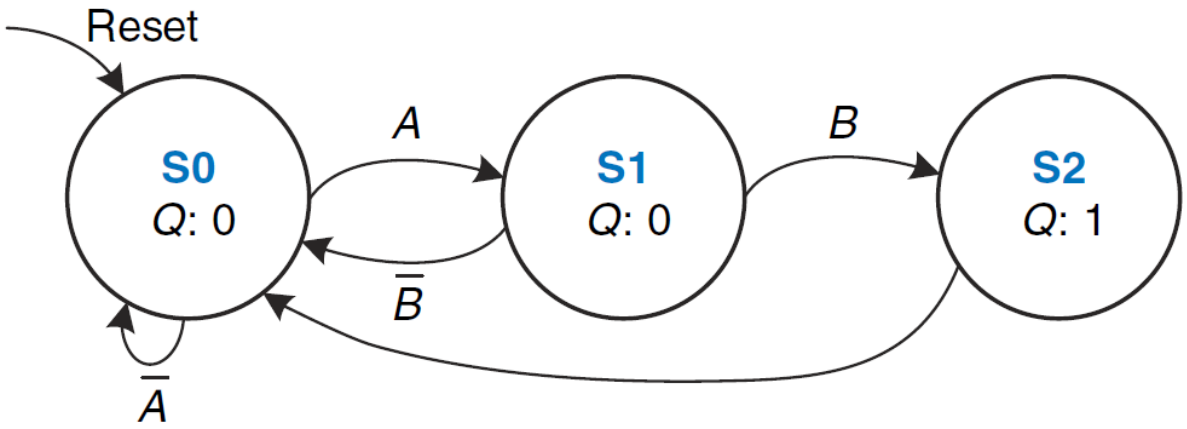
```
`timescale 1ns/10ps
module test_traffic;
reg       clk,Reset,Ta,Tb;
wire[1:0] La,Lb;
initial  begin
    clk = 1'b1; Reset=1'b0;
    #1 Reset = 1'b1; #7 Reset = 1'b0; end
initial  begin
          Ta = 1'b1;Tb=1'b0;
    #10 Ta = 1'b1;Tb=1'b1;
    #5   Ta = 1'b0;Tb=1'b1;
    #15 Ta = 1'b1;Tb=1'b0;
    #15 Ta = 1'b0;Tb=1'b0;
    #10   $stop;   end
always #2.5 clk = ~clk;
traffic  n1(clk, Reset, Ta,Tb,La,Lb);
endmodule
```



FSM1 小结

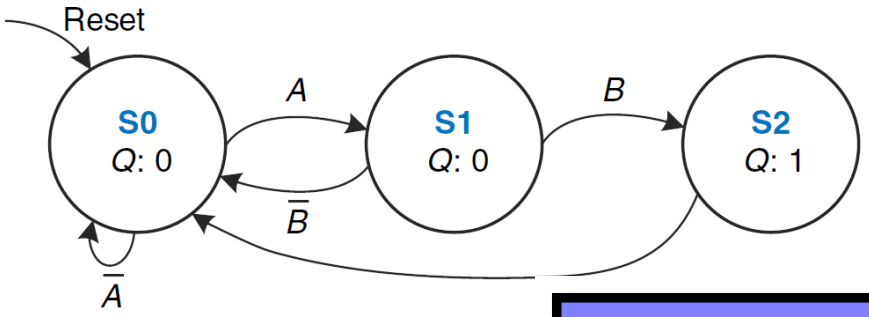
- 列出所有输入/输出信号
- 列出所有状态
- 画出状态图，即状态之间跳变关系
- 列出状态转换表，布尔表达式
- 列出输出表（状态、输入），布尔表达式
- 画出原理图

FSM2例子



1：说明状态机的功能
2：FSM的状态转换表、输出表
3：写成下一个状态和输出的布尔表达式
4：画出这个状态机的原理图

FSM2状态转换

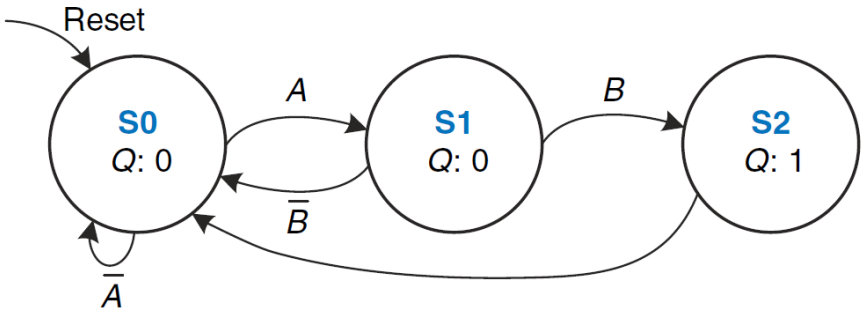


FSM的状态转换表

state	encoding $s_{1:0}$
S0	00
S1	01
S2	10

current state		inputs		next state	
s_1	s_0	a	b	s'_1	s'_0
0	0	0	X	0	0
0	0	1	X	0	1
0	1	X	0	0	0
0	1	X	1	1	0
1	0	X	X	0	0

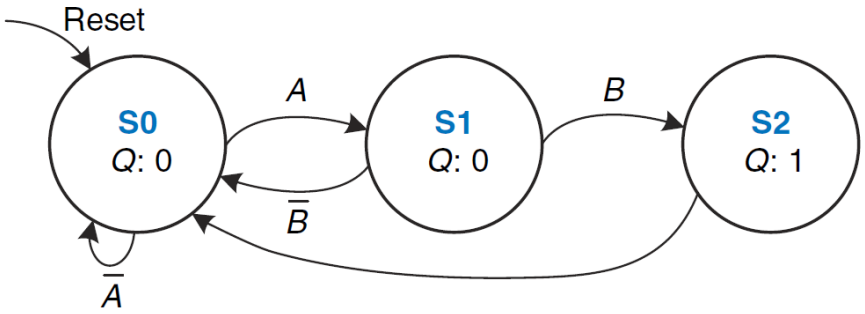
FSM2输出值



FSM的状态输出表

current state		output
s_1	s_0	q
0	0	0
0	1	0
1	0	1

FSM2布尔表达式



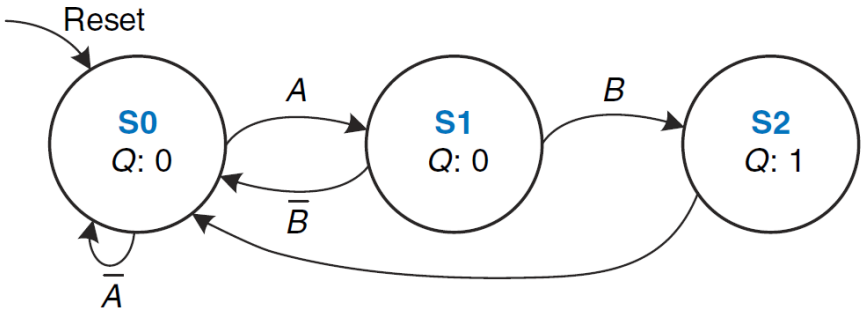
FSM的布尔表达式

$$S'_1 = S_0 B$$

$$S'_0 = \overline{S_1}\,\overline{S_0} A$$

$$Q = S_1$$

FSM2原理图

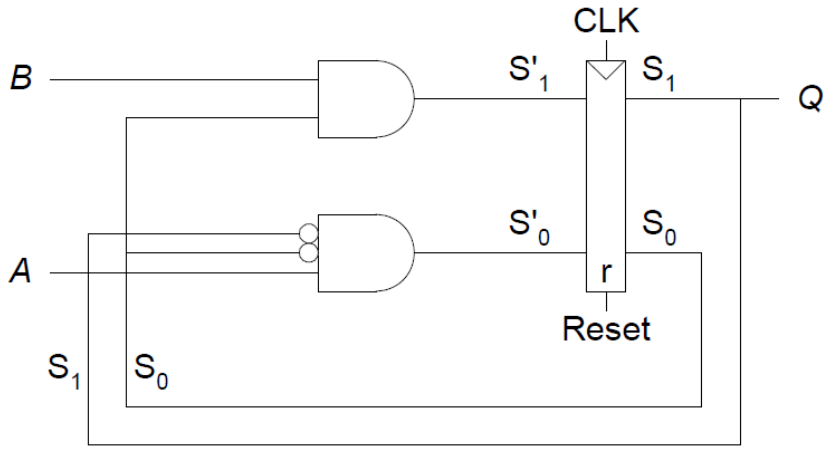


FSM的原理图

$$S'_1 = S_0 B$$

$$S'_0 = \overline{S_1}\,\overline{S_0} A$$

$$Q = S_1$$

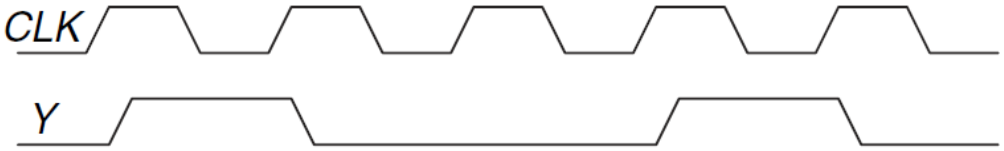


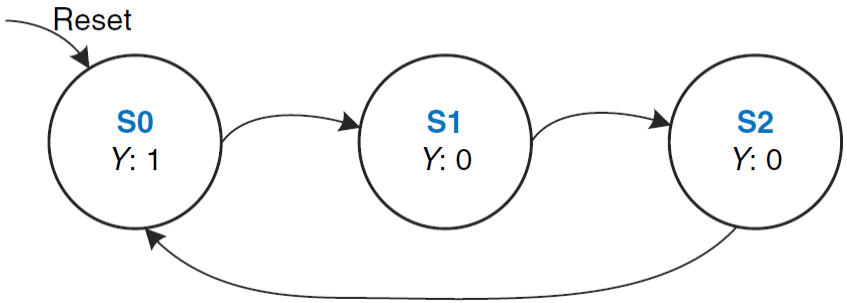
FSM 状态编码

- **二进制编码**:
  - i.e., for four states, 00, 01, 10, 11
- **One-hot** 码
  - 一个状态位表示一个状态
  - 每次只有一位为TRUE，如 0001, 0010, 0100, 1000
  - 需要更多的触发器
  - 但下一状态或输出逻辑更简单

FSM3例子

三分频器分别用二进制码和One-hot码实现。





状态机功能图

FSM3状态编码

Current State	Next State
S0	S1
S1	S2
S2	S0

状态转换表的非编码形式

Current State	Output
S0	1
S1	0
S2	0

输出表的非编码形式

State	One-Hot Encoding			Binary Encoding	
	S_2	S_1	S_0	S_1	S_0
S0	0	0	1	0	0
S1	0	1	0	0	1
S2	1	0	0	1	0

编码表

FSM3布尔表达式

Current State S_1	Current State S_0	Next State S_1'	Next State S_0'
0	0	0	1
0	1	1	0
1	0	0	0

状态转换表及布尔表达式：

$$S_1' = \overline{S}_1 S_0$$
$$S_0' = \overline{S}_1 \overline{S}_0$$

Current State S_2	Current State S_1	Current State S_0	Next State S_2'	Next State S_1'	Next State S_0'
0	0	1	0	1	0
0	1	0	1	0	0
1	0	0	0	0	1

状态转换表布尔表达式：

$$S_2' = S_1$$
$$S_1' = S_0$$
$$S_0' = S_2$$

Current State	Output
S0	1
S1	0
S2	0

Current		Y
0	0	1
0	1	0
1	0	0

输出转换表及布尔表达式：$Y = \overline{S}_1 \overline{S}_0$

Current State	Output
S0	1
S1	0
S2	0

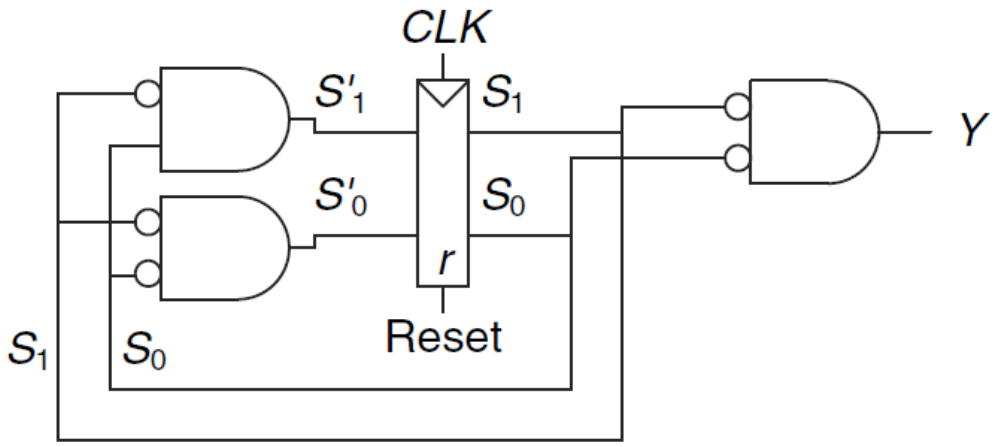
Current			Y
0	0	1	1
0	1	0	0
1	0	0	0

输出转换表及布尔表达式：$Y = S_0$

FSM3原理图

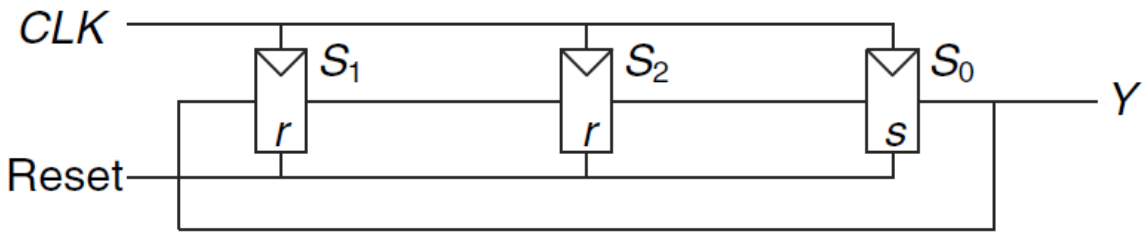
二进制码

$$S_1' = \overline{S}_1 S_0$$
$$S_0' = \overline{S}_1 \overline{S}_0$$
$$Y = \overline{S}_1 \overline{S}_0$$



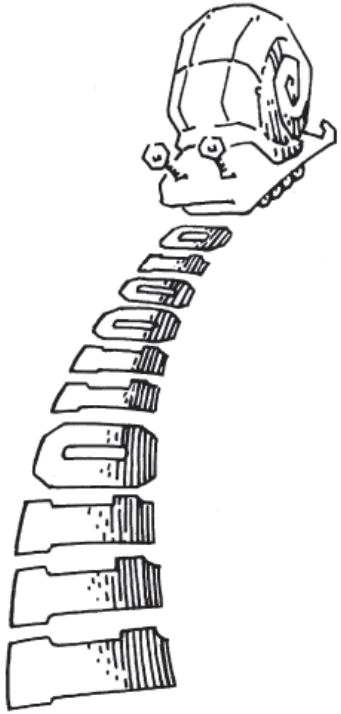
One-hot码

$$S_2' = S_1$$
$$S_1' = S_0$$
$$S_0' = S_2$$
$$Y = S_0$$



Moore vs. Mealy 状态机

- Mealy型状态机：输出取决于输入和当前的状态；
  输出被标记在弧上而不是圈内。
- 蜗牛沿着纸带从左向右爬行，纸带是1和0的序列每个时钟周期，蜗牛爬行到下一位。蜗牛爬行，最后经过的2位是01时，蜗牛会高兴得笑起来。
- 设计一个有限状态机计算蜗牛何时会笑？输入A是蜗牛触角下面的位。当蜗牛笑时，输出Y为TURE，比较Moore型状态机和Mealy型状态机设计。画出包含输入、状态和输出的每种机器的时序图，蜗牛爬行序列是01_0011_0111

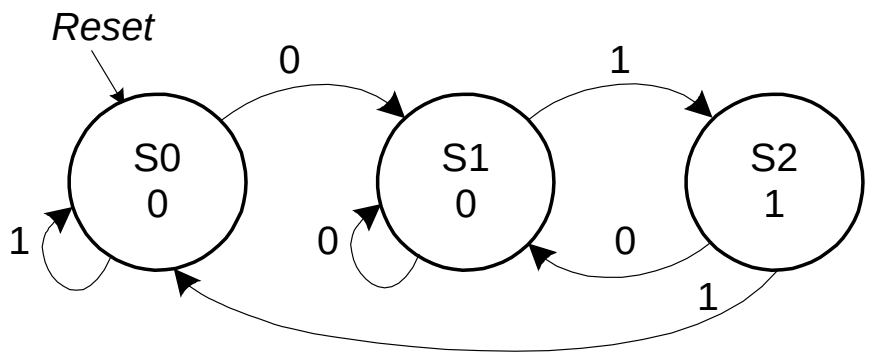


状态转换图

Moore状态机可设置3个状态1/0/01，或设置4个状态00/01/10/11

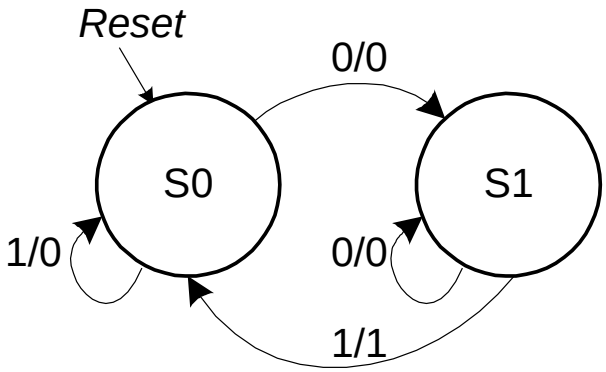
Mealy型状态机只需要2个状态（当前1/0）。每一个弧被标记为A/Y，A是引起转换的输入，Y是相应的输出。

Moore FSM



S0(1)： 11 S1(0)： 10 /00 S2(01): 01

Mealy FSM



S0(1)： 01,11 S1(0)： 00,10

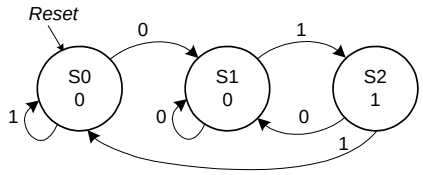
Mealy FSM: arcs indicate input/output

Moore FSM State Transition Table

Current State S_1	Current State S_0	Inputs A	Next State S'_1	Next State S'_0
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		

State	Encoding
S0	00
S1	01
S2	10

Moore FSM



$S_1' =$

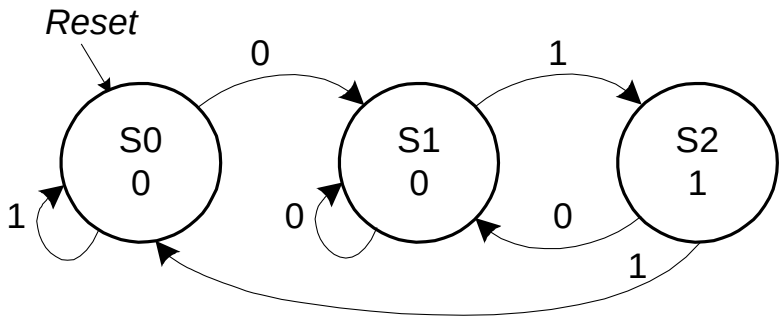
$S_0' =$

Moore FSM Output Table

Current State S_1	Current State S_0	Output Y
0	0	
0	1	
1	0	

$Y =$

Moore FSM

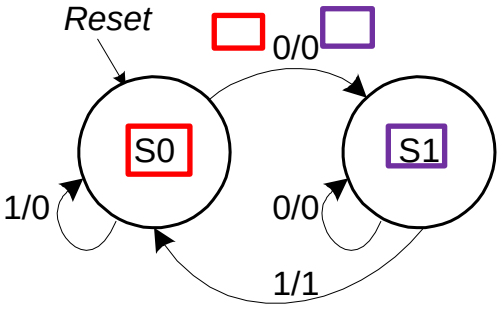


Mealy FSM State Transition & Output Tab

Current State	Input	Next State	Output
S_0	A	S'_0	Y
0	0		
0	1		
1	0		
1	1		

State	Encoding
S0	0
S1	1

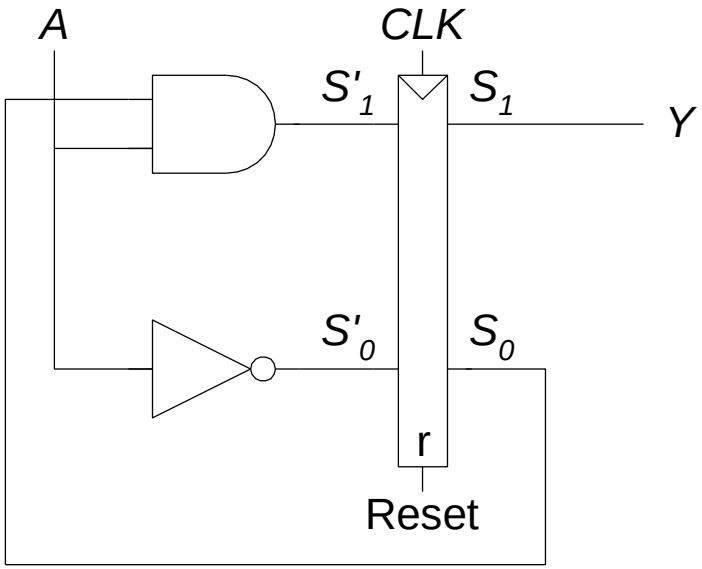
Mealy FSM



$S_0' =$

$Y =$

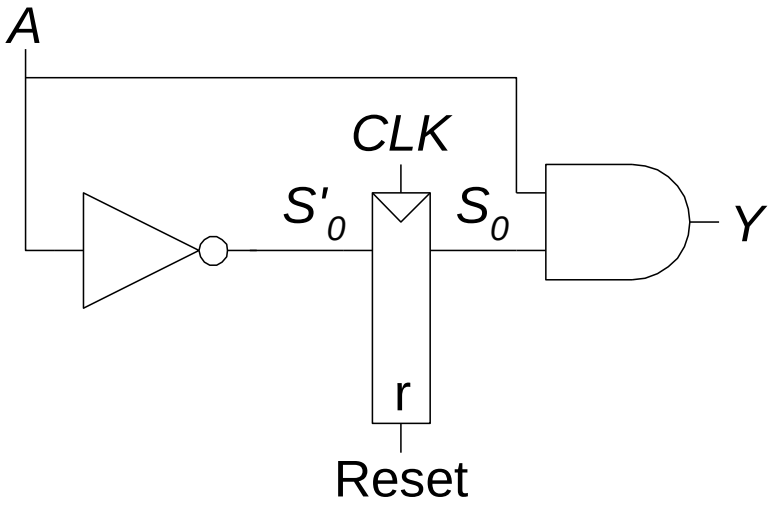
Moore



$$S_1' = S_0 A$$
$$S_0' = \overline{A}$$

$$Y = S_1$$

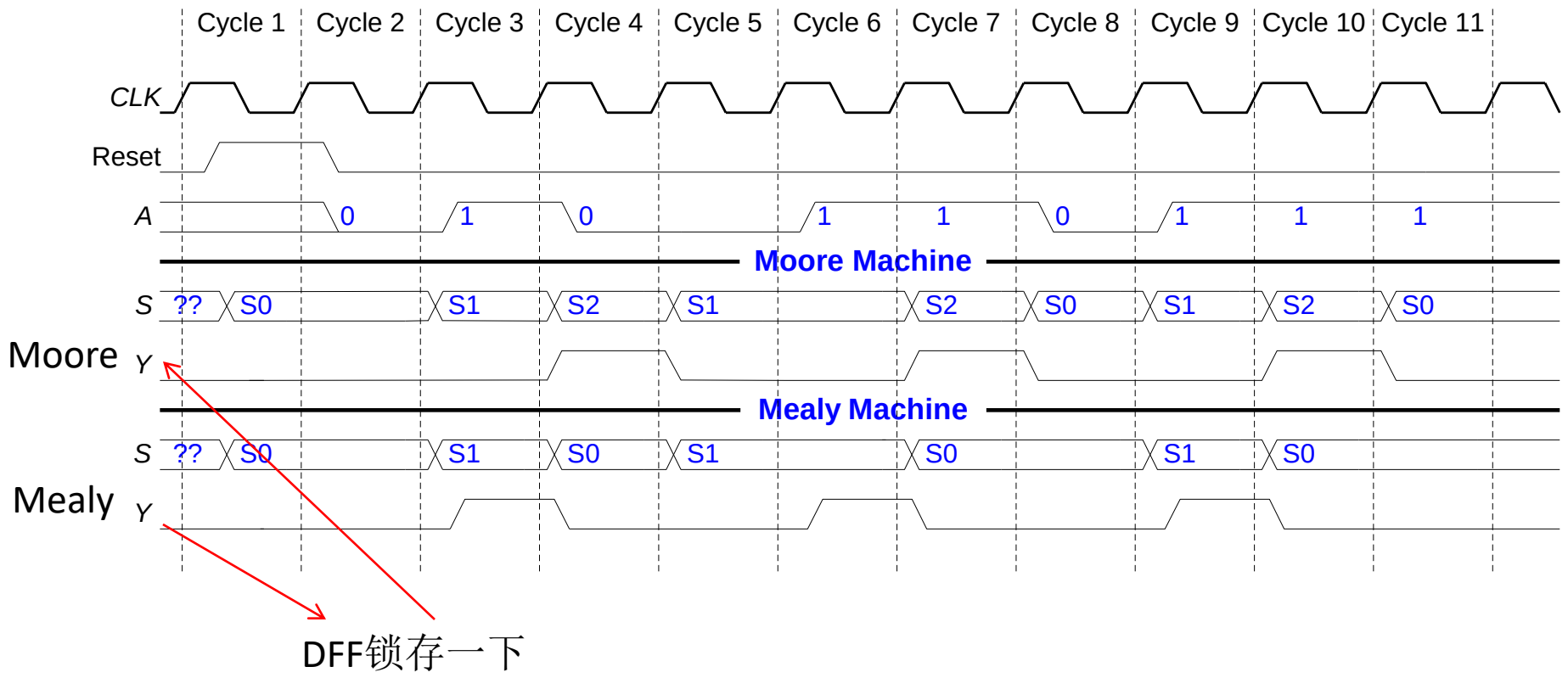
Mealy



$$S_0' = \overline{A}$$

$$Y = S_0 A$$

Moore & Mealy Timing Diagram



Moore & Mealy的systemverilog实现

Moore

```
module snail_moore(clk,Reset,A,S,Y);
input  clk,Reset,A;
output[1:0]  S;
output       Y;
parameter S0=2'b00,S1=2'b01,S2=2'b10;
reg[1:0]      current_state,next_state;
reg           Y;
wire[1:0]     S;
always_ff @(posedge clk or posedge Reset)
   if(Reset)  current_state <= S0;
   else  current_state <= next_state;
always_comb //@(current_state or A)
   case(current_state)
   S0: if(A) next_state = S0;
       else  next_state = S1;
   S1: if(A) next_state =  S2;
       else  next_state =  S1;
S2: if(A) next_state =  S0;
       else  next_state =  S1;
   default:  next_state = S0;
   endcase
always_comb //@(current_state)
   case(current_state)
   S0: Y = 1'b0;
   S1: Y = 1'b0;
   S2: Y = 1'b1;
   default: Y = 1'b0;
   endcase
assign S = current_state;
endmodule
```

Moore & Mealy的systemverilog实现

Mealy

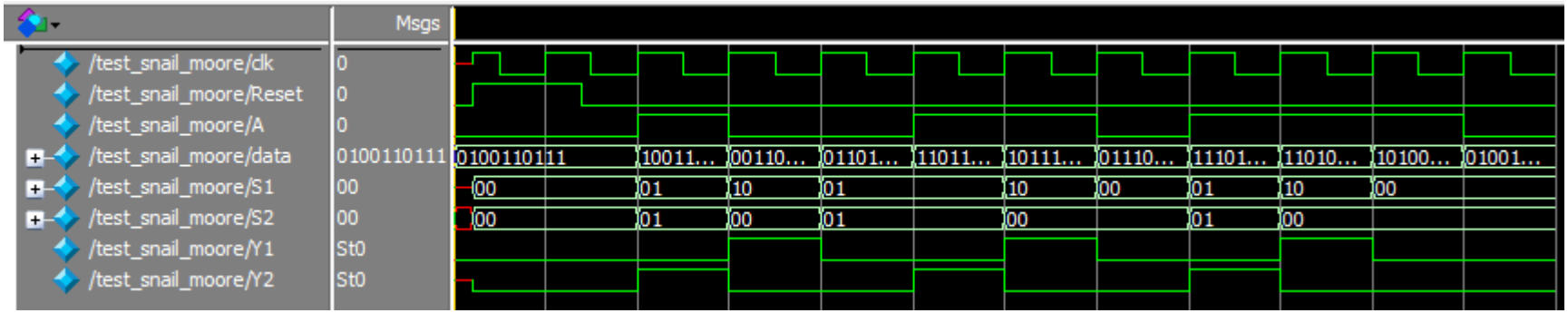
```
module snail_mealy(clk,Reset,A,S,Y);
input  clk,Reset,A;
output[1:0]  S;
output       Y;
parameter S0=1'b0,S1=1'b1;
reg          current_state,next_state;
reg          Y;
wire[1:0]    S;
always_ff @(posedge clk or posedge Reset)
   if(Reset)  current_state <= S0;
   else  current_state <= next_state;
always_comb //@(current_state or A)
   case(current_state)
   S0: if(A) begin next_state = S0; Y = 1'b0; end
       else  begin next_state = S1; Y = 1'b0; end
   S1: if(A) begin next_state = S0; Y = 1'b1; end
       else  begin next_state = S1; Y = 1'b0; end
 default:  next_state = S0;
   endcase
assign S = {1'b0,current_state};
//reg          Y1;
//always_ff @(posedge clk or posedge Reset)
//   if(Reset)  Y1 <= 1'b0;
//   else  Y1 <= Y;
endmodule
```

Moore & Mealy的systemverilog实现

测试模块

```
`timescale 1ns/10ps
module test_snail;
reg       clk,Reset,A;
reg[9:0]  data;
wire[1:0] S1,S2;
wire      Y1,Y2;
initial  begin
   Reset=1'b0; #1 Reset = 1'b1; clk =
1'b1;
   #6 Reset = 1'b0; end
initial  begin
   data = 10'b01_0011_0111;
   #60   $stop;   end
always #2.5 clk = ~clk;
always @(posedge  clk or posedge
Reset)
   if(Reset)
      data = data;
   else
      data = {data[8:0],data[9]};
assign A = data[9];

snail_moore  n1(clk,Reset,A,S1,Y1);
snail_mealy  n2(clk,Reset,A,S2,Y2);
endmodule
```

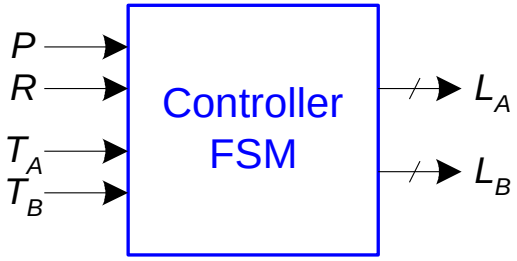


状态机分解

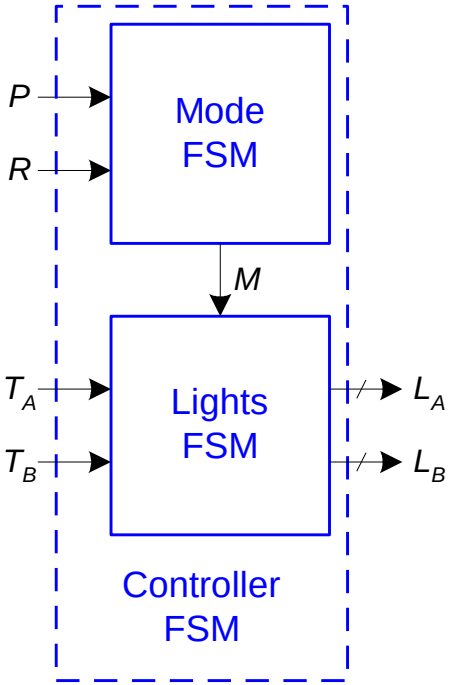
- 将复杂的状态机 分解为小的交互状态机
- 例子: 修改交通灯到游行模式.
  - 增加了两个输入: *P*, *R*
  - 当***P* = 1**, 进入游行模式
  - 当***R* = 1**, 离开游行模型

带游行模式的FSM

未分解的FSM

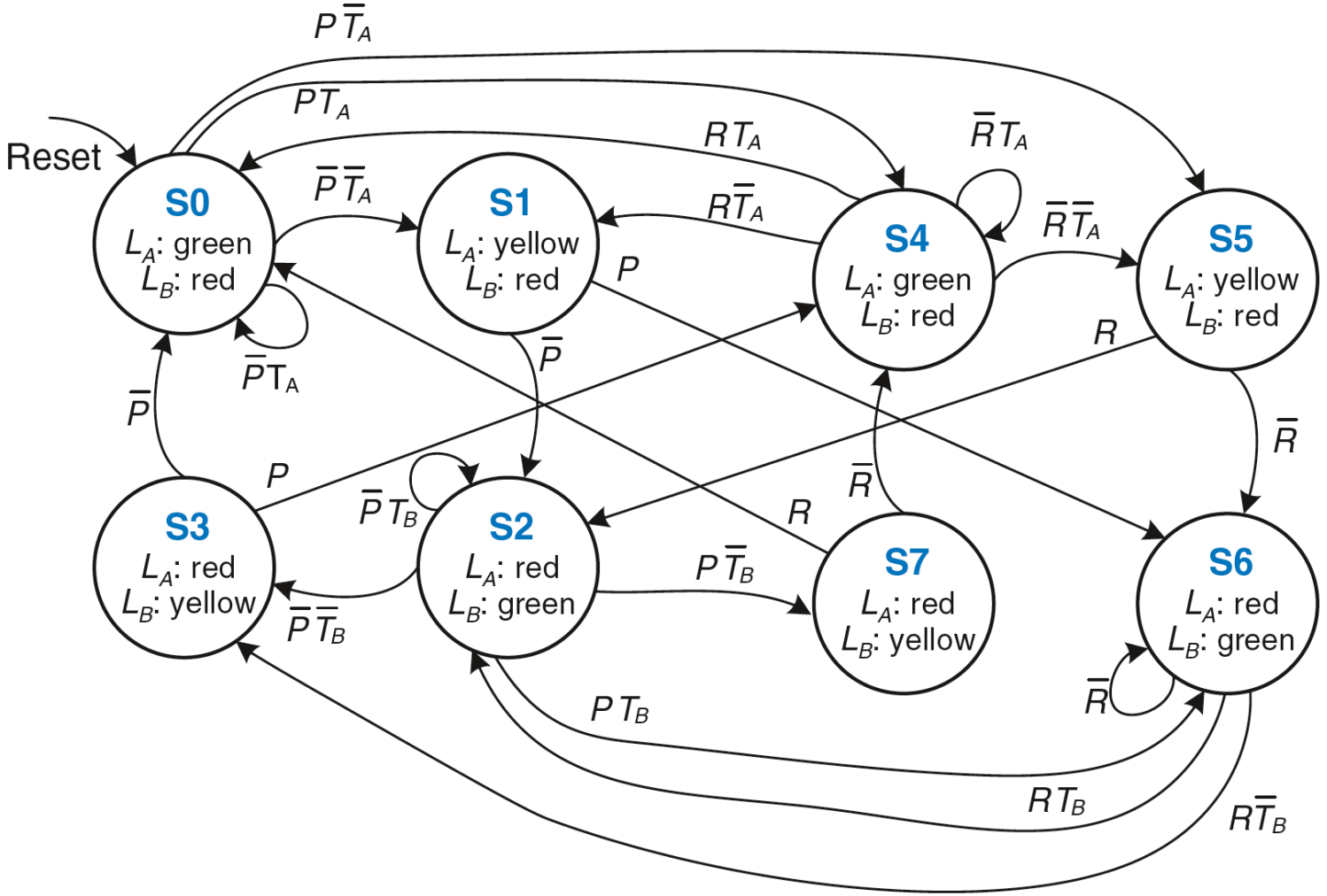


分解的FSM

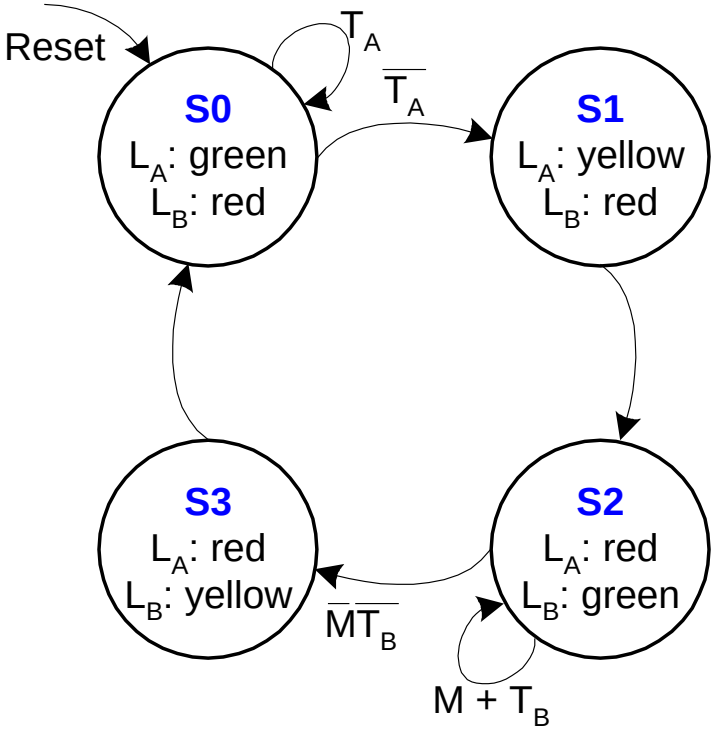


M为下一层状态机分解

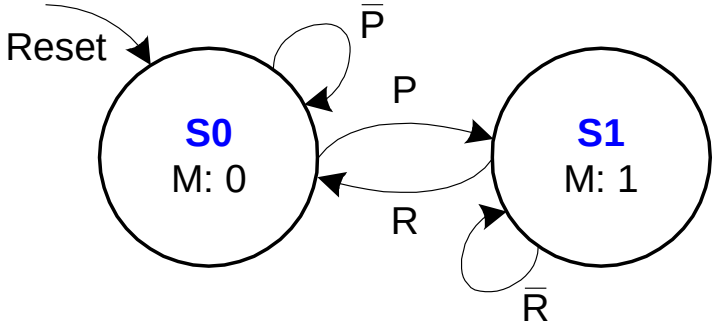
未分解的FSM


Reset
S0
L_A: green
L_B: red
S1
L_A: yellow
L_B: red
S2
L_A: red
L_B: green
S3
L_A: red
L_B: yellow
S4
L_A: green
L_B: red
S5
L_A: yellow
L_B: red
S6
L_A: red
L_B: green
S7
L_A: red
L_B: yellow
$P\overline{T}_A$
PT_A
RT_A
$\overline{R}T_A$
$\overline{P}\overline{T}_A$
$R\overline{T}_A$
$\overline{R}\overline{T}_A$
$\overline{P}T_A$
P
$\overline{P}$
R
$\overline{R}$
$\overline{P}T_B$
$P\overline{T}_B$
$\overline{P}\overline{T}_B$
PT_B
RT_B
$R\overline{T}_B$

分解的 FSM

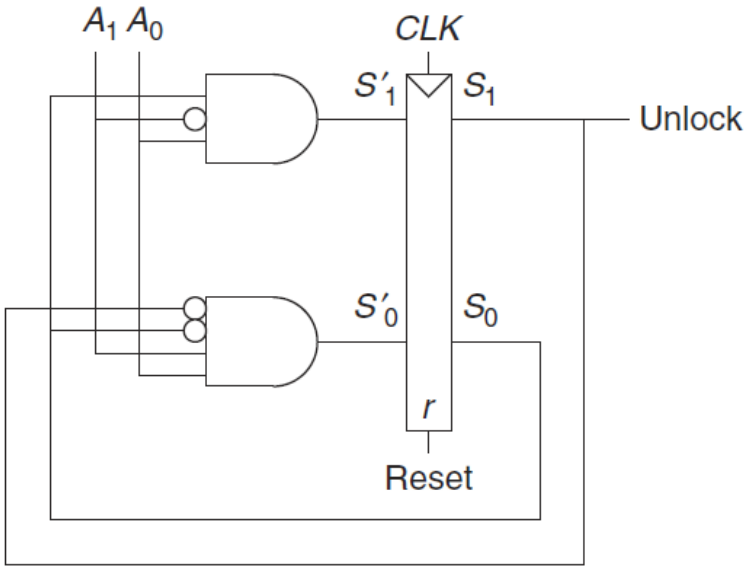


Lights FSM



Mode FSM

由电路图导出状态机



电路图

1、写出下一个状态、输出值的布尔表达式；

$$S'_1 = S_0 \overline{A_1} A_0$$
$$S'_0 = \overline{S_1}\ \overline{S_0} A_1 A_0$$
$$\text{Unlock} = S_1$$

2、根据当前状态和输入值列出下一个状态及输出值的真值表；

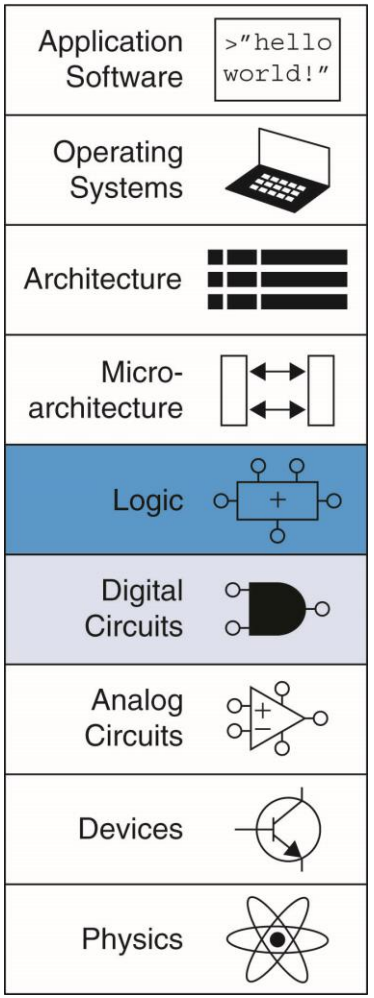
3、真值表简化后，画出状态图。

状态机设计小结

1. 确定inputs and outputs
2. 画状态图，确定状态个数及转换关系
3. 写出状态转换表
4. 选择状态编码
5. 对于Moore状态机:
   写出状态转换表、输出表
6. 对于Mealy 状态机:
   写出组合的状态转换表和输出表
7. 为下一个状态及输出写出布尔表达式
8. 画出电路图草图

第三章：时序逻辑设计

- 介绍
- 锁存器与触发器
- 同步逻辑设计
- 有限状态机
- 时序逻辑时序
- 并行

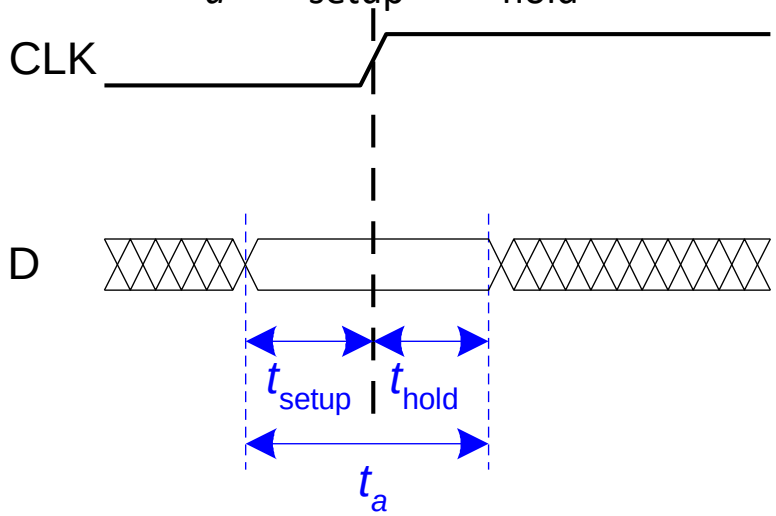


时序逻辑的时序

- 触发器在时钟沿采样数据D；
- 数据*D* 必须在采样前稳定；
- 真实的系统中，时钟不能同时到达所有的D触发器，存在时钟偏移（clock skew）

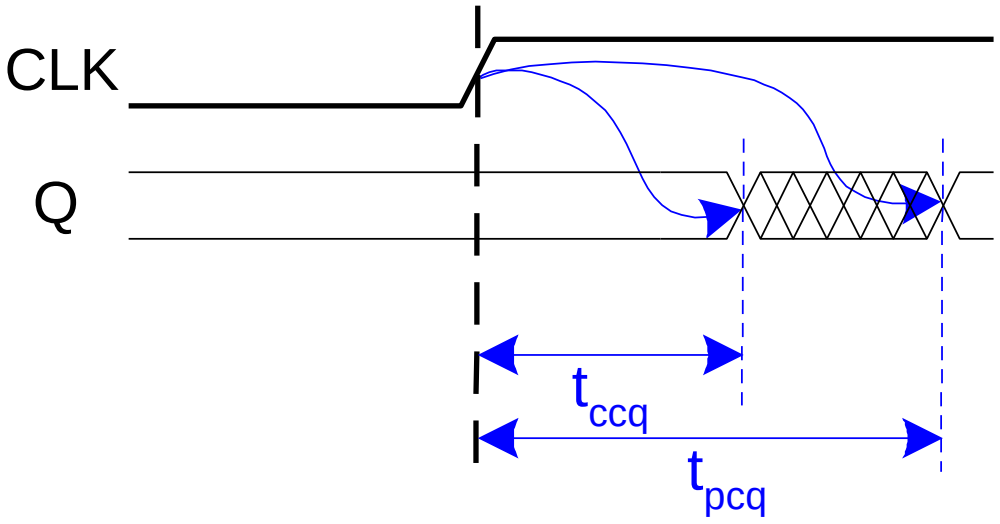
输入时序约束

- **建立时间:** t_{setup} = 时钟沿到来时数据提前保持稳定的时间 (i.e. not changing)
- **保持时间:** t_{hold} = 时钟沿到来后数据需要继续保持稳定的时间
- **孔径时间:** t_a = 建立和保持时间合在一起，为输入保持稳定状态的时间总和 ($t_a = t_{setup} + t_{hold}$)



输出时序参数

- **传播延迟: t_{pcq}** = 时钟沿之后时钟到Q输出全部改变结束、然后保持稳定的时间 (i.e., to stop changing)
  (Propagation clock after q)
- **最小延迟: t_{ccq}** = 时钟沿之后时钟到Q最先开始变化的时间 (i.e., start changing)　　(Contanimnation clock after q)

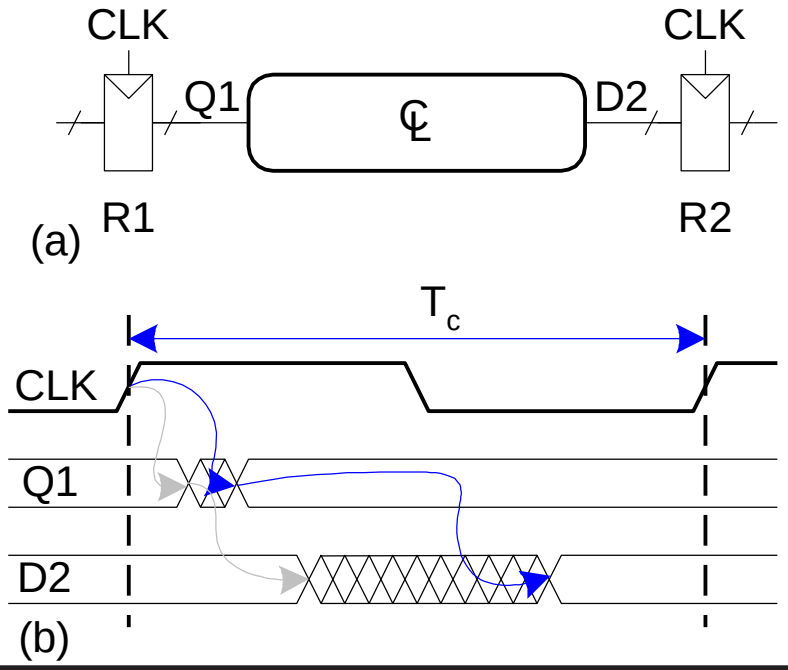


动态约束

- 同步时序电路输入必须在时钟沿附近的孔径时间内保持稳定（建立时间t_{setup}与保持时间t_{hold}）
- 特别地, 输入必须稳定
  - at least t_{setup} before the clock edge
  - at least until t_{hold} after the clock edge

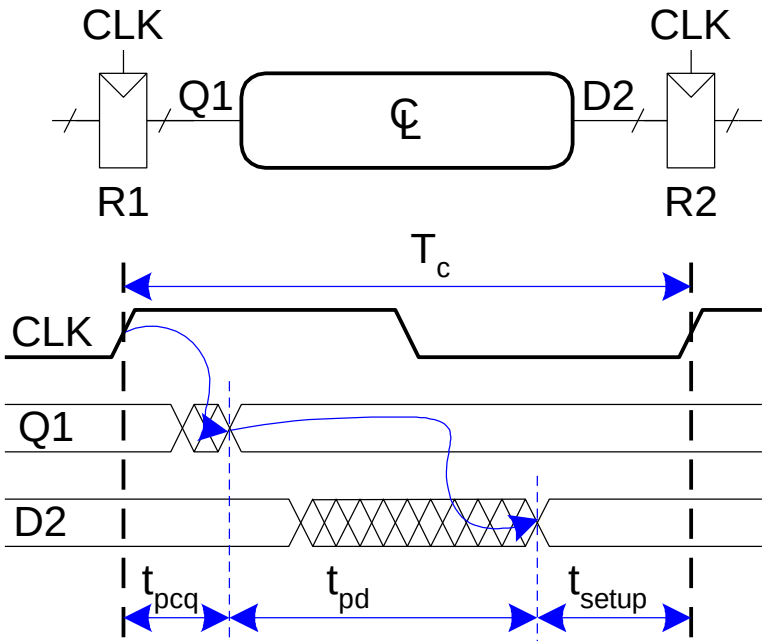
动态约束

- 系统速度多快？时钟频率 评估
- 寄存器之间的延迟有最大和最小值, **最大最小值受制于电路元件本身延迟**



建立时间约束

- 从寄存器R1通过组合逻辑到寄存器R2的最大延迟
- 到寄存器R2的输入必须在时钟沿之前t_{setup} 时间稳定
- (t_{pd}:组合逻辑的传播延迟)



$$T_c \geq$$

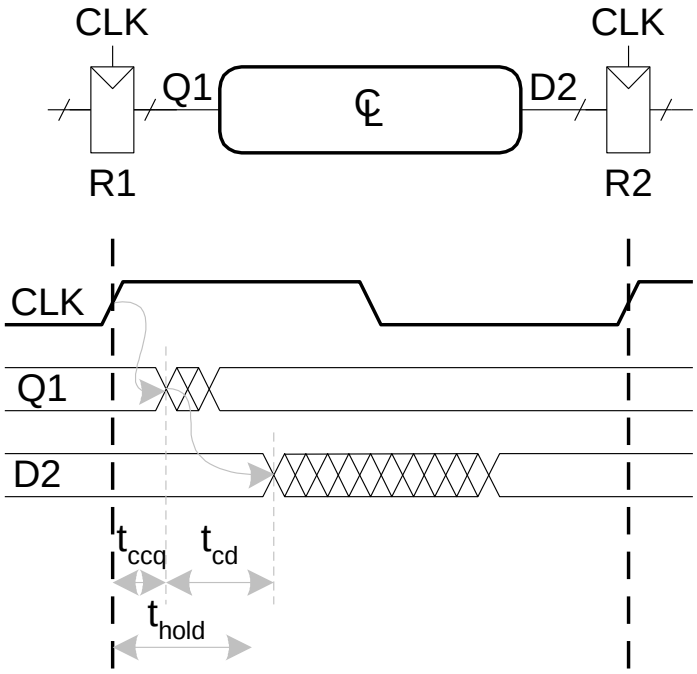
$$t_{pd} \leq$$

建立时间约束/最大延迟约束

$(t_{pcq} + t_{setup})$: 时序开销

保持时间约束

- 从寄存器R1通过组合逻辑到寄存器R2的最小延迟
- 到寄存器R2的输入必须在时钟沿之后至少t_{hold}时间保持稳定

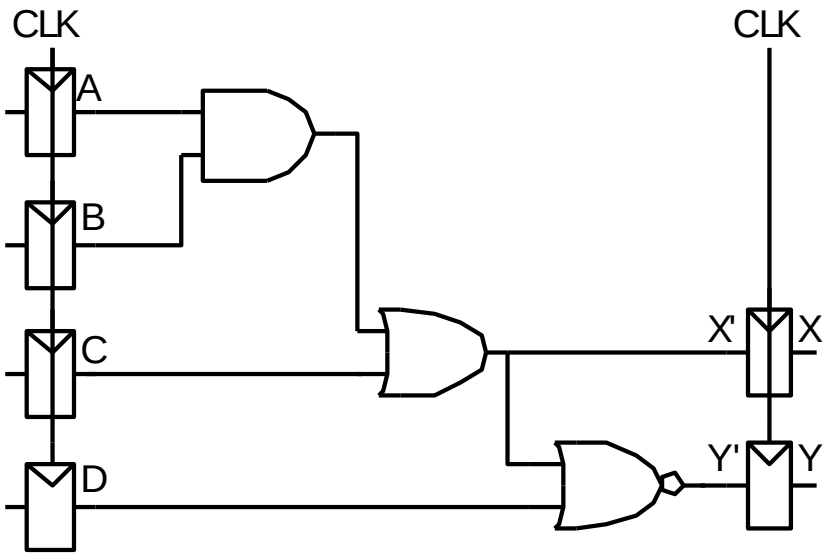


$$t_{hold} <$$
$$t_{cd} >$$

最小延迟约束

时序分析1

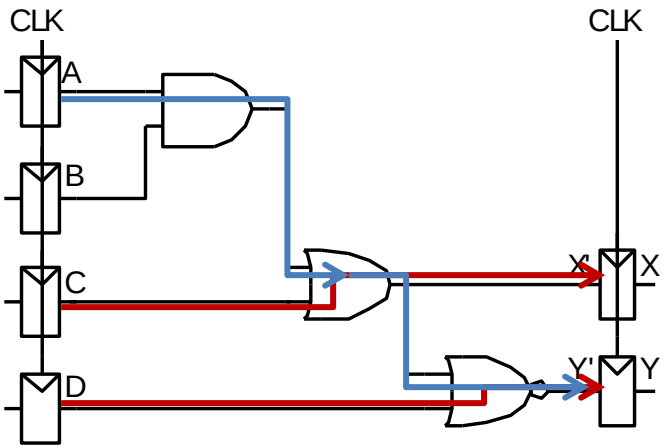
小结：时序电路中的建立时间和保持时间约束了D触发器之间组合逻辑的最大延迟和最小延迟。其中最大延迟约束受制于其关键路径上的组合逻辑。

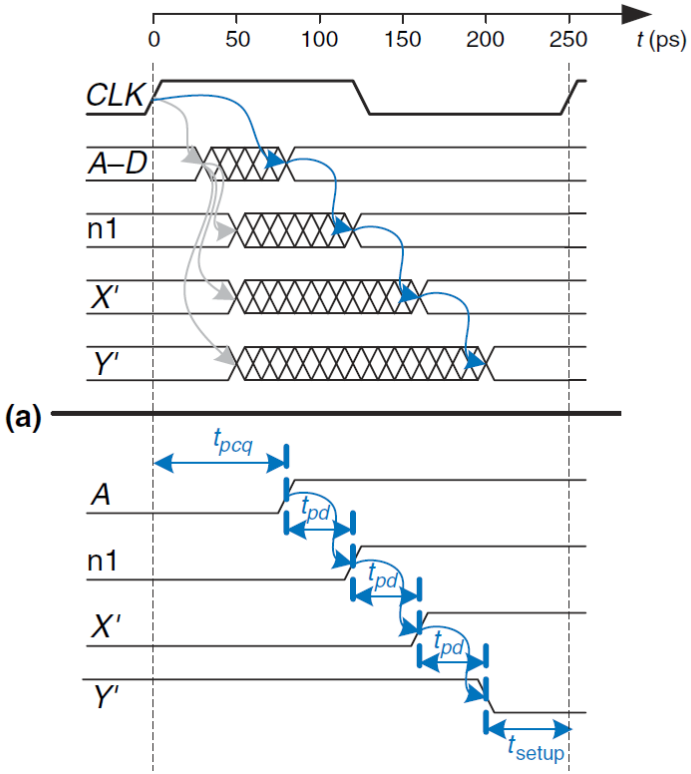


Timing Characteristics

t_{ccq} = 30 ps
t_{pcq} = 50 ps
t_{setup} = 60 ps
t_{hold} = 70 ps

per gate
t_{pd} = 35 ps
t_{cd} = 25 ps





t_{pd} = 3 x 35 ps = 105 ps

t_{cd} = 25 ps

Setup time constraint（关键路径）：

$T_c \geq (50 + 105 + 60)$ ps = 215 ps

$f_c = 1/T_c$ = 4.65 GHz

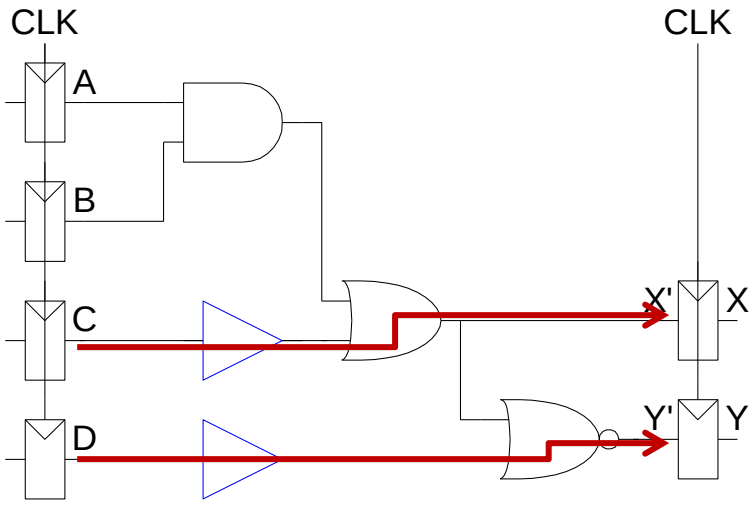
Hold time constraint（最小延迟路径）：

$t_{ccq} + t_{cd} > t_{hold}$?

(30 + 25) ps > 70 ps ? **No!**

时序分析2

Add buffers to the short paths:



Timing Characteristics

t_{ccq} = 30 ps
t_{pcq} = 50 ps
t_{setup} = 60 ps
t_{hold} = 70 ps

per gate:
t_{pd} = 35 ps
t_{cd} = 25 ps

t_{pd} = 3 x 35 ps = 105 ps

t_{cd} = 2 x 25 ps = 50 ps

Setup time constraint:

$T_c \geq (50 + 105 + 60)$ ps = 215 ps

$f_c = 1/T_c$ = 4.65 GHz

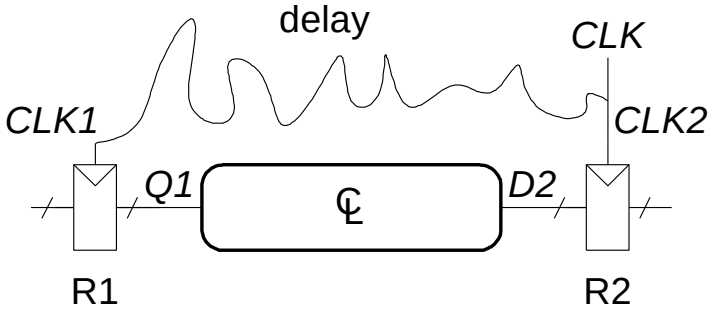
Hold time constraint:

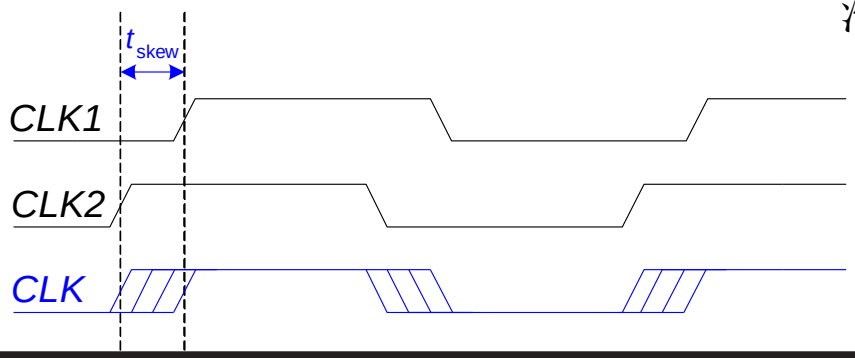
$t_{ccq} + t_{cd} > t_{hold}$?

(30 + 50) ps > 70 ps ? **Yes!**

时钟偏移

- 时钟并不是在同一时间到达所有寄存器的
- **偏移**t_{skew}：不同时钟沿的延迟
- 使用最坏情况分析



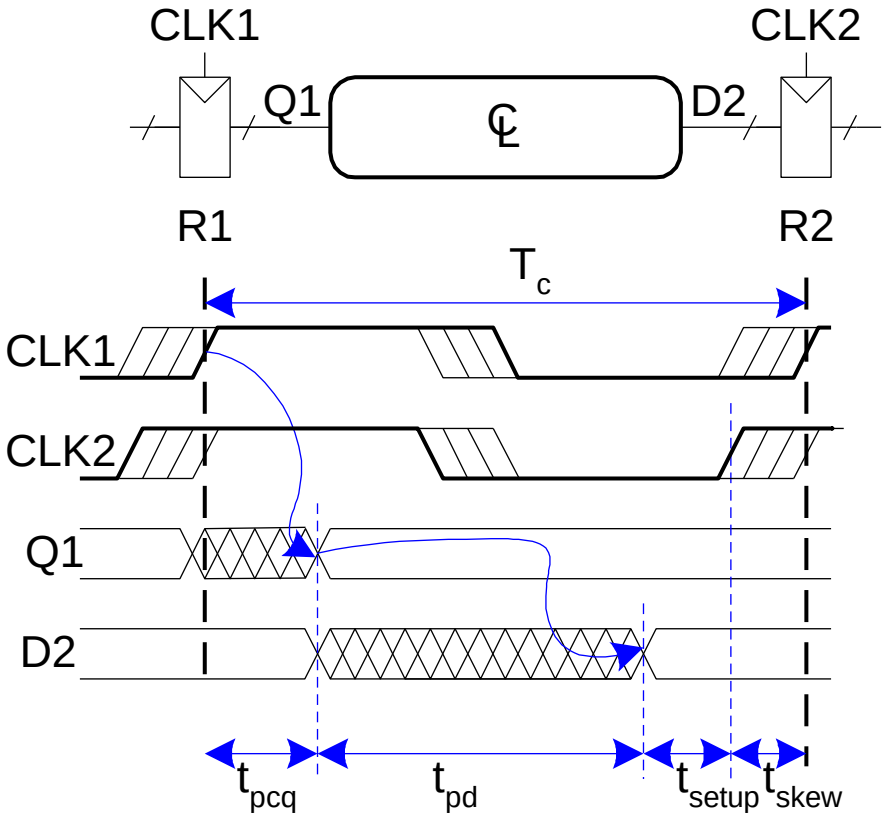


对于建立时间来说，由于时钟偏移，时钟可能提前到达。

对于保持时间来说，由于时钟偏移，时钟可能滞后到达。

带时钟偏移的建立时间分析

- 最坏情况, CLK2 is earlier than CLK1



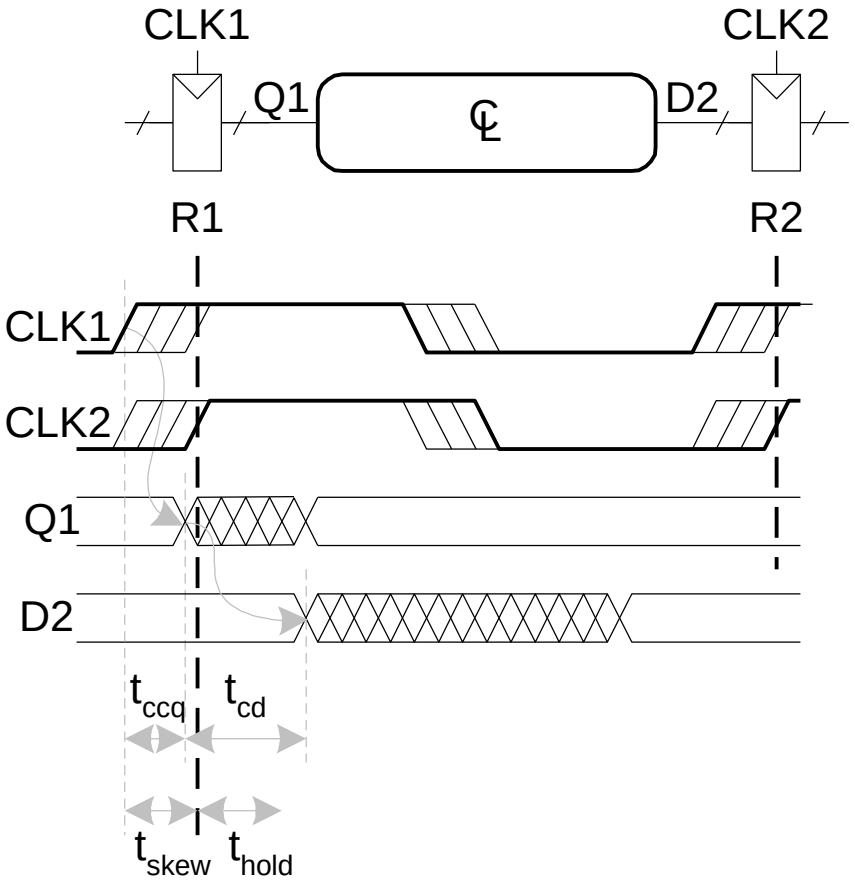
对于建立时间来说，由于时钟偏移，时钟可能提前到达。

$T_c \geq$

$t_{pd} \leq$

带时钟偏移的保持时间分析

- In the worst case, CLK2 is later than CLK1

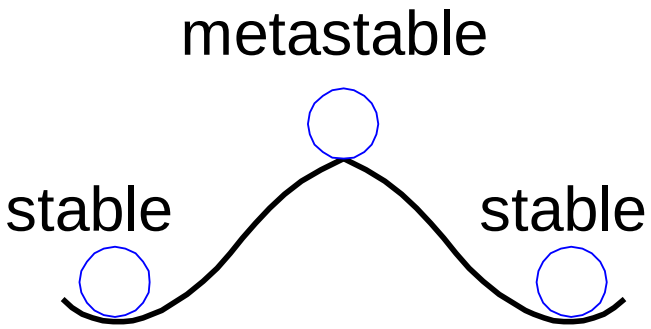


对于保持时间来说，由于时钟偏移，时钟可能滞后到达。

$t_{cd} >$

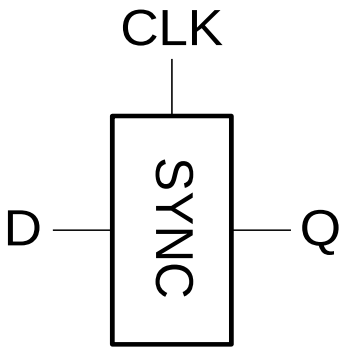
亚稳态

- **双稳态器件，两个稳定状态**, 亚稳态处于两者之间（孔径时间内输入数据可能变化导致）
- **触发器:** two stable states (1 and 0) and one metastable state（每一个双稳态的设备在两个稳定状态之间都存在一个亚稳态）



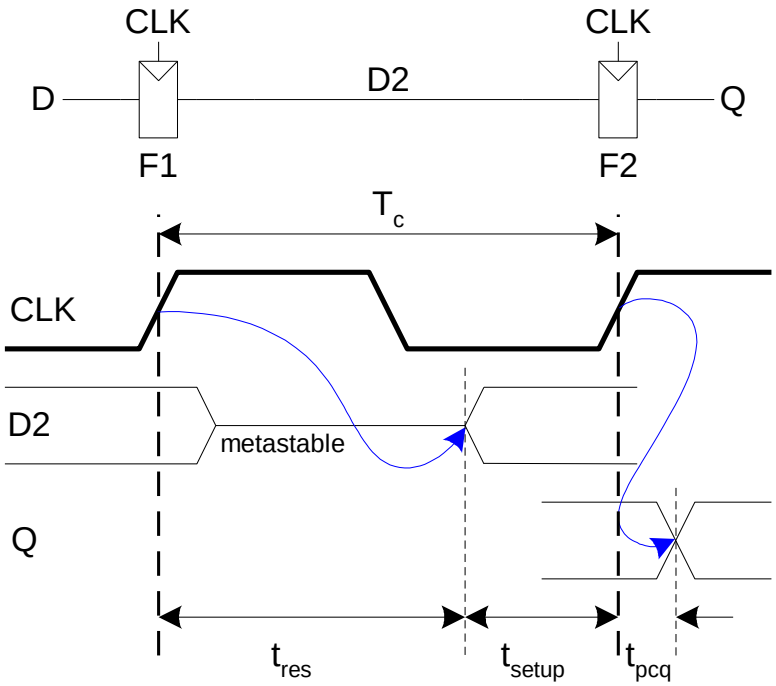
同步器

- **异步输入不可避免** (user interfaces, systems with different clocks interacting, etc.)
- **同步的目标:** make the probability of failure (the output *Q* still being metastable) low；所有的异步输入都需要经过同步器（synchronizer）处理。
- 同步器也不能使得亚稳态的概率降为0



同步器内部结构

- 同步器: 两个背靠背触发器组成
- 当D被F1采样时，数据就传输进来
- 内部信号 D2 has (T_c - t_{setup}) time to resolve to 1 or 0，只要T_c足够大



并行

- **系统速度可用延迟Latency和任务吞吐量Throughput来衡量；通过并行处理可提高吞吐量。**
- **两类并行:**
  - 空间并行
    - 硬件资源复制
  - 时间并行
    - 任务被打破成多个阶段
    - 流水线pipelining
    - 例如组装线

并行例子1

- 做蛋糕
- 5 minutes to 做
- 15 minutes to 烤
- 没有并行时的延迟及吞吐量?

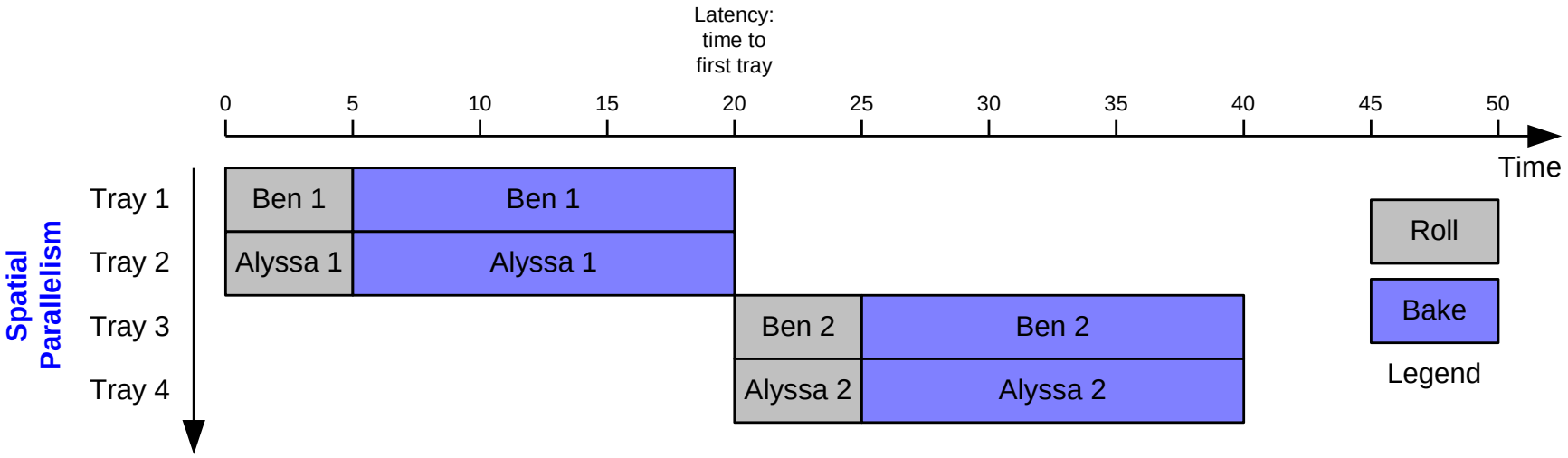
Latency = 5 + 15 = 20 minutes = **1/3 hour**

Throughpı **ur**

并行例子1

- 使用并行后的吞吐量?
  - 空间并行: Ben asks Allysa P. Hacker to help, using her own oven
  - 时间并行:
    - 两端: 做 and 烤
    - 使用两个盘子
    - 第一个在烤时，另一个在做

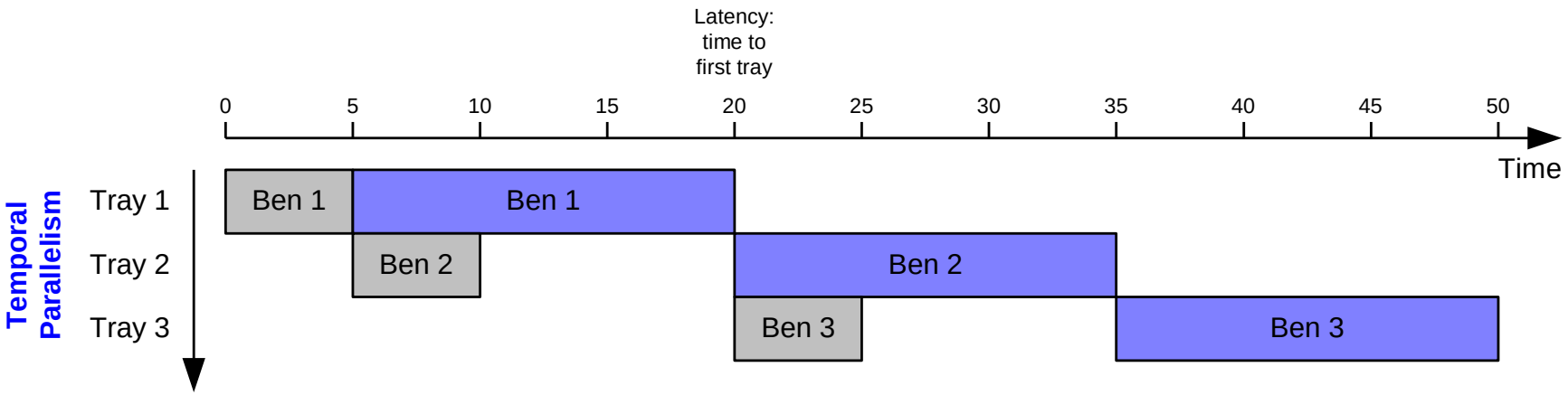
并行例子1-空间并行性



Latency =

Throughput =

并行例子1-时间并行



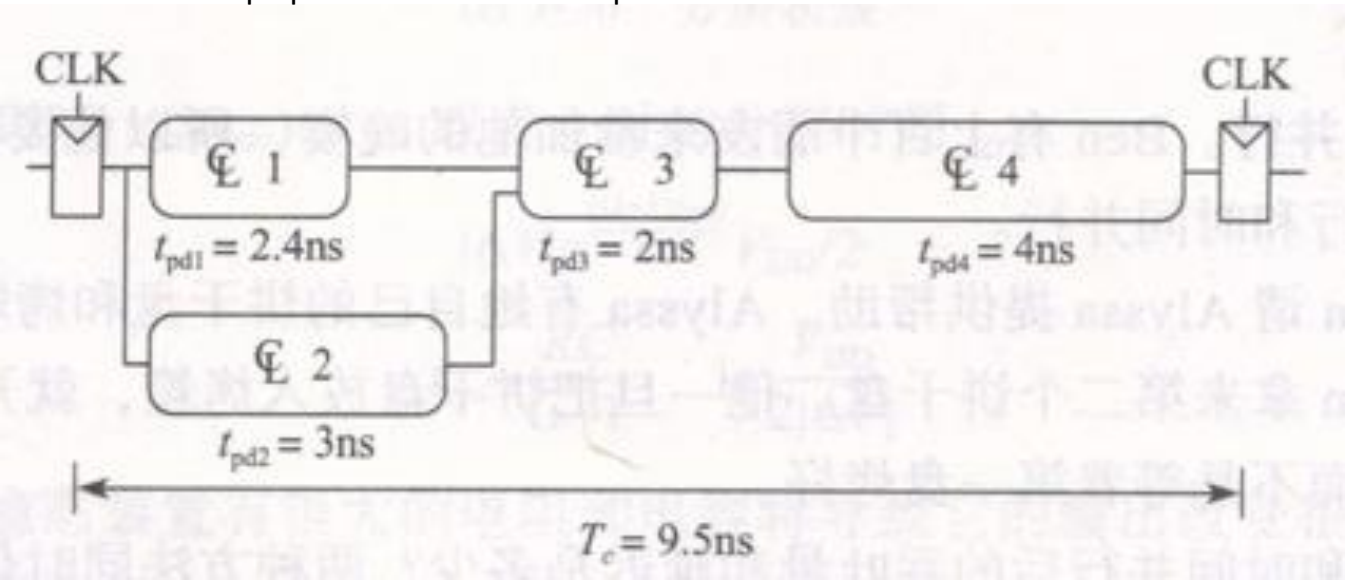
Latency =

Throughput =

同时使用两种计算, 吞吐量： **8 trays/hour**

并行例子2

已知D触发器的 t_{pcq}为0.3ns， t_{setup}为0.2ns，求下图电路的延迟和吞吐量？

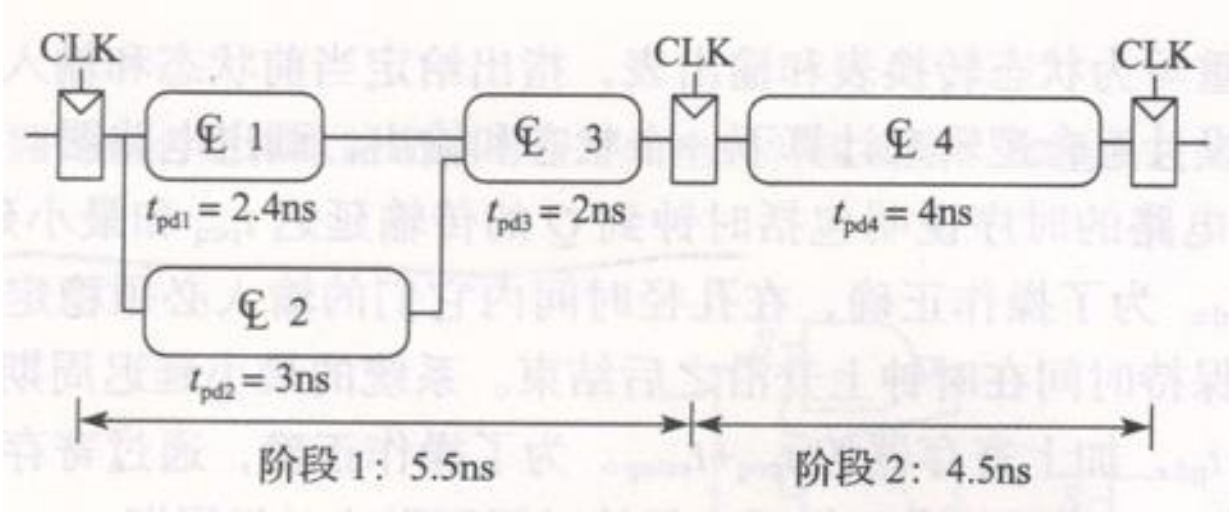


Latency = 0.3+3+2+4+0.2 = 9.5ns

Throughput = 1/9.5ns = 105MHz

并行例子2

已知D触发器的 t_{pcq}为0.3ns， t_{setup}为0.2ns，求下图采用二级流水线电路的延迟和吞吐量？



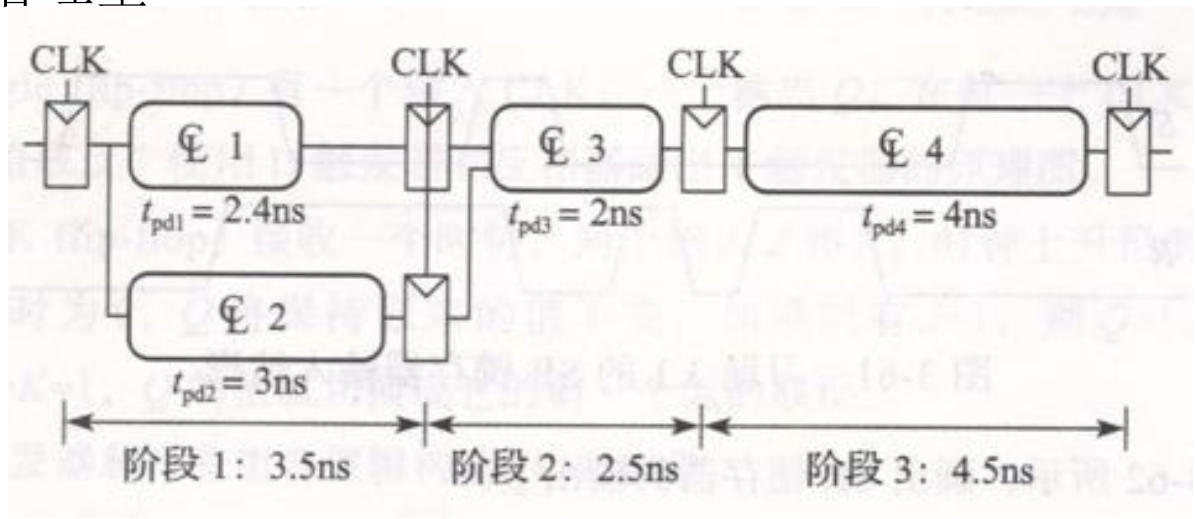
第一级延迟0.3+3+2+0.2= 5.5ns 第二级延迟0.3+4+0.2= 4.5ns

Latency = 2个时钟周期= 5.5 *2 = 11ns

Throughput = 1/5.5ns = 182MHz

并行例子2

已知D触发器的 t_{pcq}为0.3ns， t_{setup}为0.2ns，求下图采用三级流水线电路的延迟和吞吐量？



第一级延迟0.3+3+0.2= 3.5ns 第二级延迟0.3+2+0.2= 2.5ns

第三级延迟0.3+4+0.2= 4.5ns

Latency = 3个时钟周期= 4.5 *3 = 13.5ns

Throughput = 1/4.5ns = 222MHz

总结

1、时序电路记忆了先前的输入信息及状态，其输出跟之前状态和输入有关；

2、触发器是时序电路的重要元件，有复位和使能信号；

3、有限状态机FSM是设计时序电路重要手段，需掌握状态转换图、状态编码、状态转换表、输出表及电路图草图；

4、时序逻辑电路的时序分析：时钟到Q端的传输延迟t_{pcq}、最小延迟t_{ccq}，建立时间t_{setup}、保持时间t_{hold}，最小延迟周期$T_c = t_{pcq}$ + 中间逻辑块传输延迟$t_{pd} + t_{setup}$。

5、并行方式可以提高系统的吞吐量。

作业题

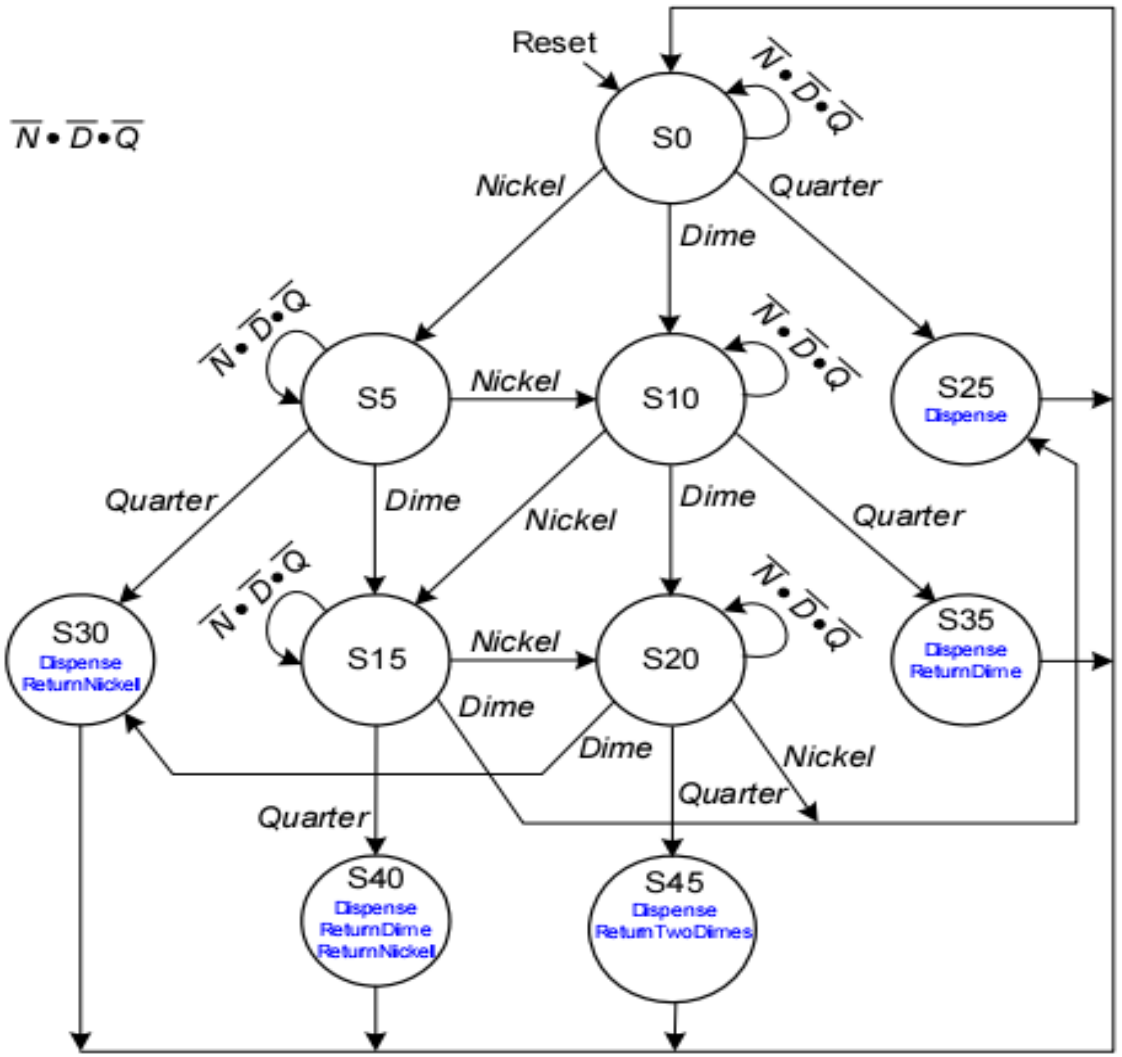
课后题：3.26，3.31， 3.33

以上3道为本章典型的题需要书面做。

其它课后题大家自己边复习本章内容边练习。熟能生巧，大家抽时间练习。

作业题3.26

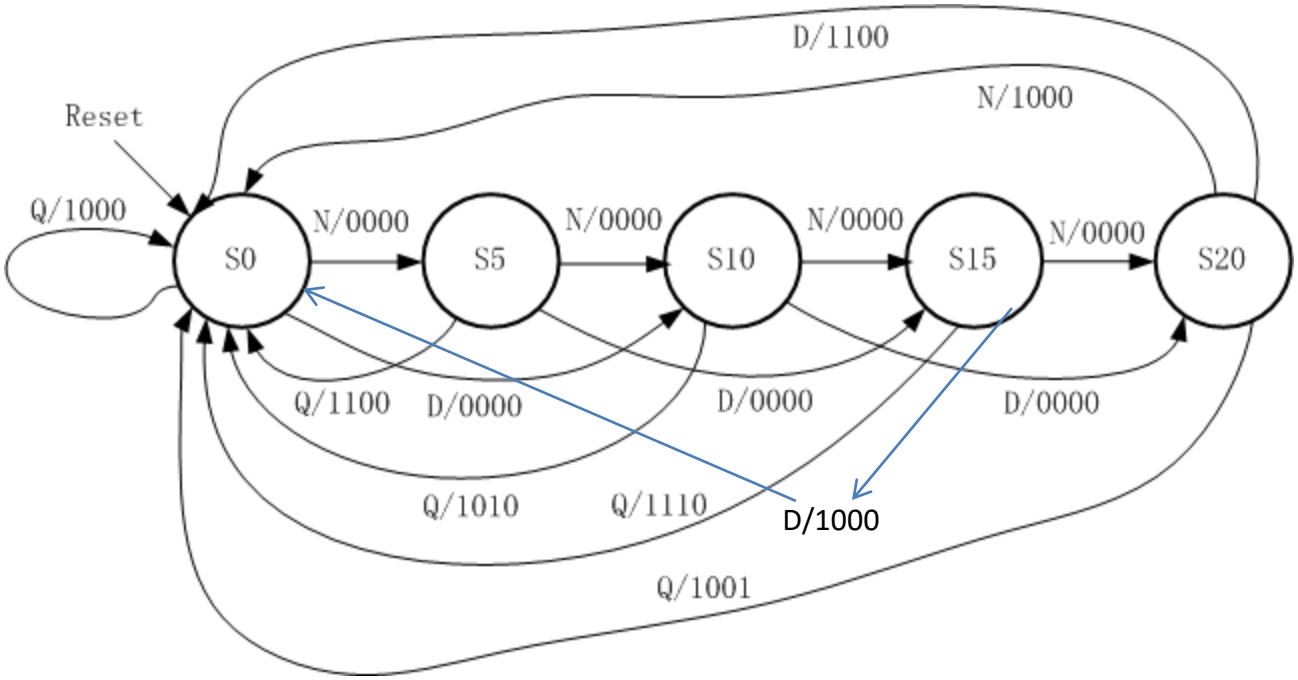
Moore型状态机



作业题3.26

输出 {Dispense, ReturnNickel, ReturnDime, ReturnTwoDime} = {D, RN, RD, RTD}

Mealy型状态机



注意：

1、case( ) /endcase中default状态应指定。

2、要求3.26分别用Moore型、Mealy型状态机SystemVerilog编程实现，并用同一个Testbench进行测试。看结果是否相同？

作业题3.26

SystemVerilog

```
module ex4_36(input  logic clk, reset, n, d, q,
              output logic dispense,
                           return5, return10,
                           return2_10);
  typedef enum logic [3:0] {S0  = 4'b0000,
                            S5  = 4'b0001,
                            S10 = 4'b0010,
                            S25 = 4'b0011,
                            S30 = 4'b0100,
                            S15 = 4'b0101,
                            S20 = 4'b0110,
                            S35 = 4'b0111,
                            S40 = 4'b1000,
                            S45 = 4'b1001}
  statetype;
  statetype [3:0] state, nextstate;

  // State Register
  always_ff @(posedge clk, posedge reset)
    if (reset) state <= S0;
    else       state <= nextstate;

  // Next State Logic
  always_comb
    case (state)
      S0:       if (n) nextstate = S5;
           else if (d) nextstate = S10;
           else if (q) nextstate = S25;
           else        nextstate = S0;
      S5:       if (n) nextstate = S10;
           else if (d) nextstate = S15;
           else if (q) nextstate = S30;
           else        nextstate = S5;
      S10:      if (n) nextstate = S15;
           else if (d) nextstate = S20;
           else if (q) nextstate = S35;
           else        nextstate = S10;
      S25:             nextstate = S0;
      S30:             nextstate = S0;
      S15:      if (n) nextstate = S20;
           else if (d) nextstate = S25;
           else if (q) nextstate = S40;
           else        nextstate = S15;
      S20:      if (n) nextstate = S25;
           else if (d) nextstate = S30;
           else if (q) nextstate = S45;
           else        nextstate = S20;
      S35:             nextstate = S0;
      S40:             nextstate = S0;
      S45:             nextstate = S0;
      default:         nextstate = S0;
    endcase

  // Output Logic
  assign dispense    = (state == S25) |
                       (state == S30) |
                       (state == S35) |
                       (state == S40) |
                       (state == S45);
  assign return5     = (state == S30) |
                       (state == S40);
  assign return10    = (state == S35) |
                       (state == S40);
  assign return2_10  = (state == S45);
endmodule
```

作业题3.31

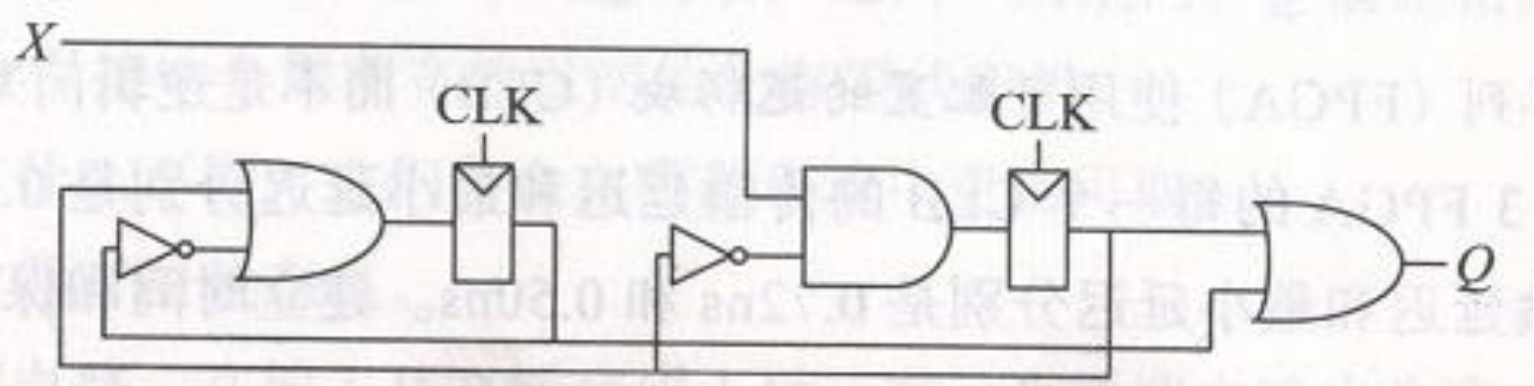


图 3-72　有限状态机原理图

分析上图，写出状态转换和输出表，画出状态转换图，编写Systemverilog代码，课后编写Testbench测试代码。

作业题3.31

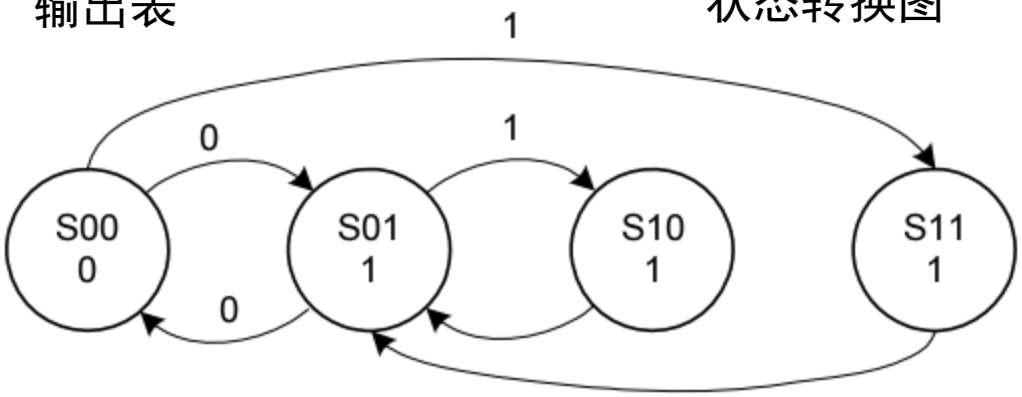
current state		input	next state	
s_1	s_0	x	s'_1	s'_0
0	0	0	0	1
0	0	1	1	1
0	1	0	0	0
0	1	1	1	0
1	X	X	0	1

状态转换表

current state		output
s_1	s_0	q
0	0	0
0	1	1
1	X	1

输出表

状态转换图



作业题3.33

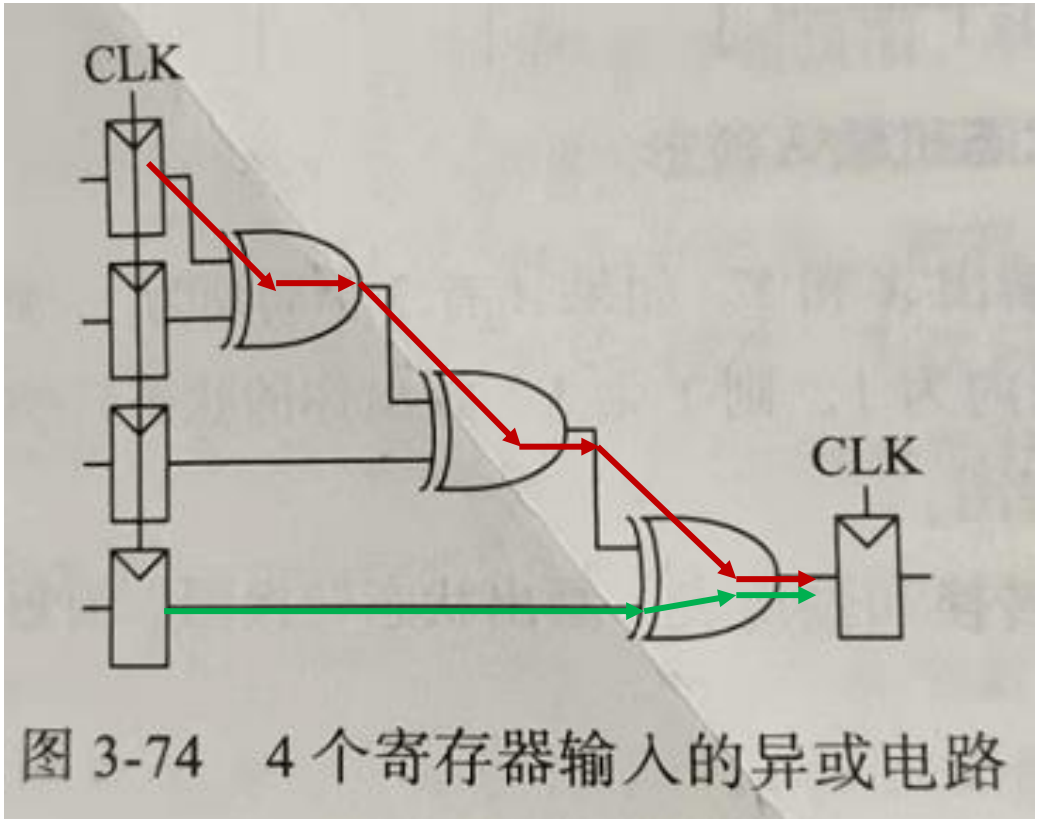


图 3-74　4 个寄存器输入的异或电路

xor tpd=100ps
　　tcd=55ps
tpcq=70ps
tccq=50ps
DFF tsetup=60ps
　　thold=20ps

作业题3.33

(a)

$t_{pd} = 3t_{pd\_XOR}$

$= 3 \times 100$ ps

$=$ **300 ps**

Next, we calculate the cycle time:

$T_c \geq t_{pcq} + t_{pd} + t_{setup}$

$\geq [70 + 300 + 60]$ ps

$= 430$ ps

$f = 1 / 430$ ps $=$ **2.33 GHz**

(b)

$T_c \geq t_{pcq} + t_{pd} + t_{setup} + t_{skew}$

Thus,

$t_{skew} \leq T_c - (t_{pcq} + t_{pd} + t_{setup})$, where $T_c = 1 / 2$ GHz $= 500$ ps

$\leq [500 - 430]$ ps $=$ **70 ps**

(c) First, we calculate the contamination delay through the combinational logic:

$t_{cd} = t_{cd\_XOR}$

$= 55$ ps

$t_{ccq} + t_{cd} > t_{hold} + t_{skew}$

Thus,

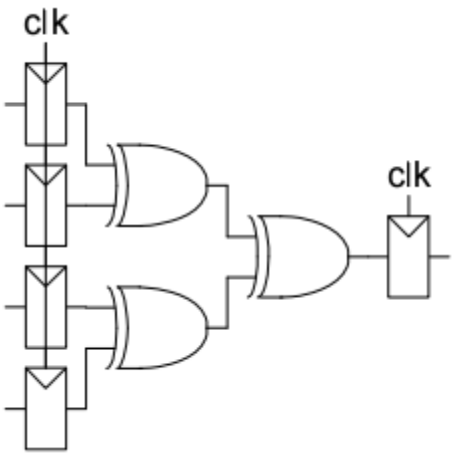
$t_{skew} < (t_{ccq} + t_{cd}) - t_{hold}$

$< (50 + 55) - 20$

$<$ **85 ps**

作业题3.33

(d)



First, we calculate the propagation and contamination delays through the combinational logic:

$$t_{pd} = 2t_{pd\_XOR}$$
$$= 2 \times 100 \text{ ps}$$
$$= \mathbf{200 \text{ ps}}$$
$$t_{cd} = 2t_{cd\_XOR}$$
$$= 2 \times 55 \text{ ps}$$
$$= \mathbf{110 \text{ ps}}$$

Next, we calculate the cycle time:

$$T_c \geq t_{pcq} + t_{pd} + t_{\text{setup}}$$
$$\geq [70 + 200 + 60] \text{ ps}$$
$$= 330 \text{ ps}$$
$$f = 1 / 330 \text{ ps} = \mathbf{3.03 \text{ GHz}}$$

$$t_{\text{skew}} < (t_{ccq} + t_{cd}) - t_{\text{hold}}$$
$$< (50 + 110) - 20$$
$$< \mathbf{140 \text{ ps}}$$

本章结束

Testvectors 文件

- 针对3.26输入激励 File: `example.tv`
- contains vectors of 输入NDQ_输出DRnRdRtd

```
100_
001
010
010
001
100
100
010
010
100
```

* 参考学习知识点

晶体管级锁存器和触发器设计

计算触发器的晶体管数量

一个D触发器需要多少个晶体管：

已知：一个与非门：4个晶体管

一个或非门：4个晶体管

一个非门：2个晶体管

一个与门：6个晶体管

求：一个SR锁存器需要多少个晶体管？

一个D锁存器需要多少个晶体管？

一个D触发器需要多少个晶体管？

晶体管级锁存器和触发器设计

计算触发器的晶体管数量

一个D触发器需要多少个晶体管：

已知：　一个与非门：4个晶体管

　　一个或非门：4个晶体管

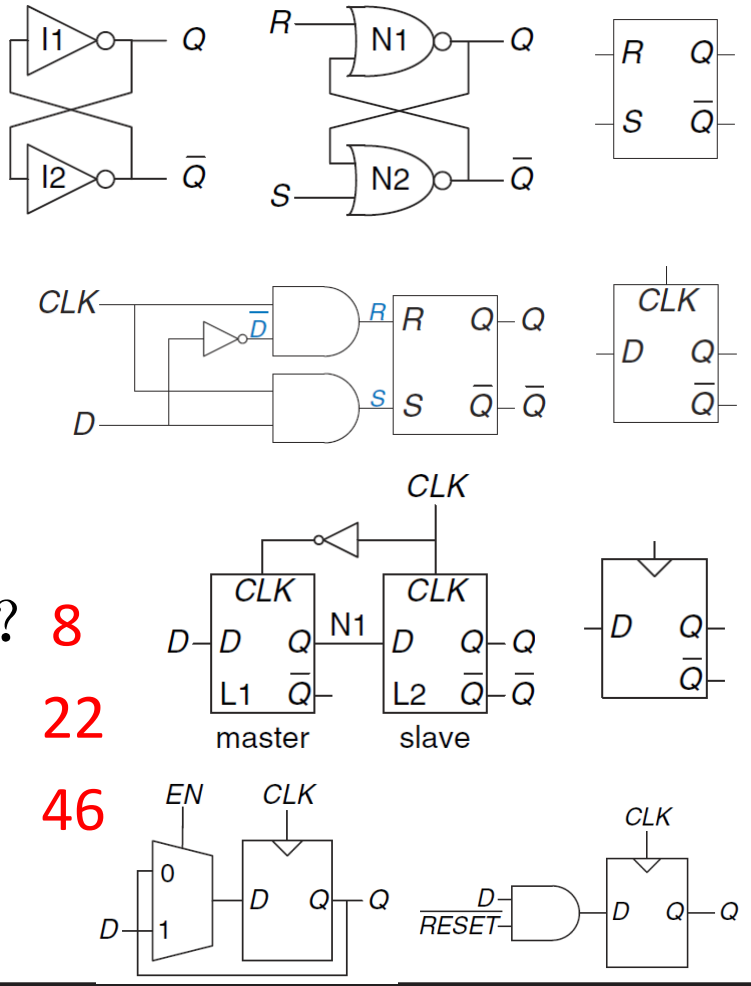
　　一个非门：　2个晶体管

　　一个与门：　6个晶体管

求：一个SR锁存器需要多少个晶体管？ 8

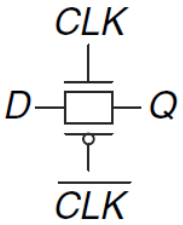
　　一个D锁存器需要多少个晶体管？ 22

　　一个D触发器需要多少个晶体管？ 46

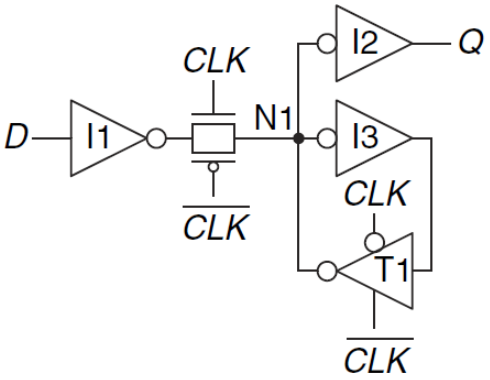


晶体管级锁存器和触发器设计

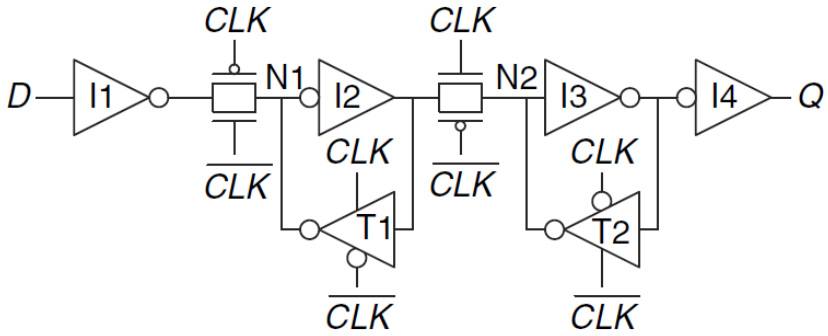
- 逻辑门构造锁存器和触发器需要大量的晶体管
- 锁存器的基本作用是透明或不透明，类似一个开关
- 利用传输门来减少晶体管的数量



(a)



(b)



D触发器20个晶体管

反相器I1,I2作为输入输出缓冲器，

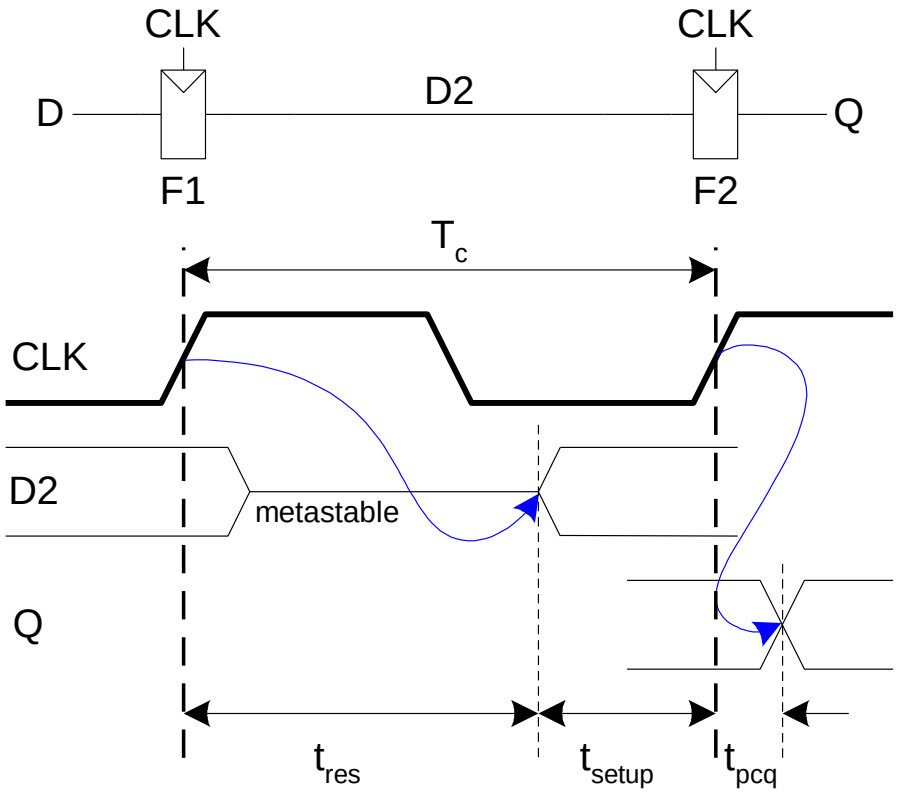
反相器I3和三态缓冲器T1，给N1提供反馈，是N1成为固定节点，

当CLK=0，T1驱动N1保持在有效逻辑。

同步器失效概率

如同步器输出Q仍为亚稳态，导致同步器失效的概率:

$$P(failure) = (T_0/T_c)\, e^{-(T_c - t_{setup})/\tau}$$



同步器的失效平均时间

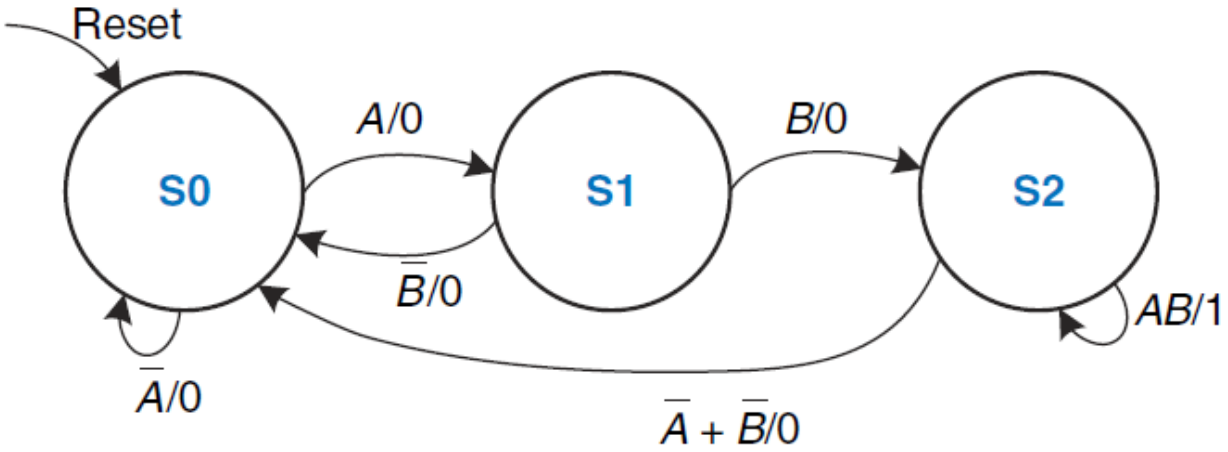
- 如果同步器的输入每秒改变一次，每秒失败的可能性为 *P*(failure).
- 如果输入改变N次，每秒失败的可能性：

$$P(\text{failure})/\text{second} = (NT_0/T_c)\, e^{-(T_c - t_{setup})/\tau}$$

- 同步器失效平均时间： 1/[*P*(failure)/second]
- Called ***Mean Time Between Failures***, MTBF:

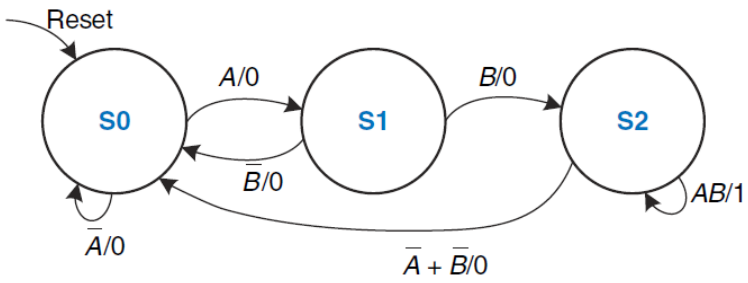
$$\text{MTBF} = 1/[P(\text{failure})/\text{second}] = (T_c/NT_0)\, e^{(T_c - t_{setup})/\tau}$$

FSM4例子



1：说明状态机的功能
2：FSM的状态转换表、输出表 （二进制码）
3：写成下一个状态和输出的布尔表达式
4：画出这个状态机的原理图

FSM4状态表及布尔表达式



功能：A B 为True，输出Q

编码表

state	encoding $s_{1:0}$
S0	00
S1	01
S2	10

状态与输出表

current state		inputs		next state		output
s_1	s_0	a	b	s'_1	s'_0	q
0	0	0	X	0	0	0
0	0	1	X	0	1	0
0	1	X	0	0	0	0
0	1	X	1	1	0	0
1	0	1	1	1	0	1
1	0	0	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	0	0	0

布尔表达式

$$S'_1 = \overline{S_1} S_0 B + S_1 A B$$

$$S'_0 = \overline{S_1}\,\overline{S_0} A$$

$$Q' = S_1 A B$$

FSM4原理图

$$S'_1 = \overline{S_1} S_0 B + S_1 A B$$

$$S'_0 = \overline{S_1}\,\overline{S_0} A$$

$$Q' = S_1 A B$$

原理图