Chapter 3

***Digital Design and Computer Architecture*, 2nd Edition**

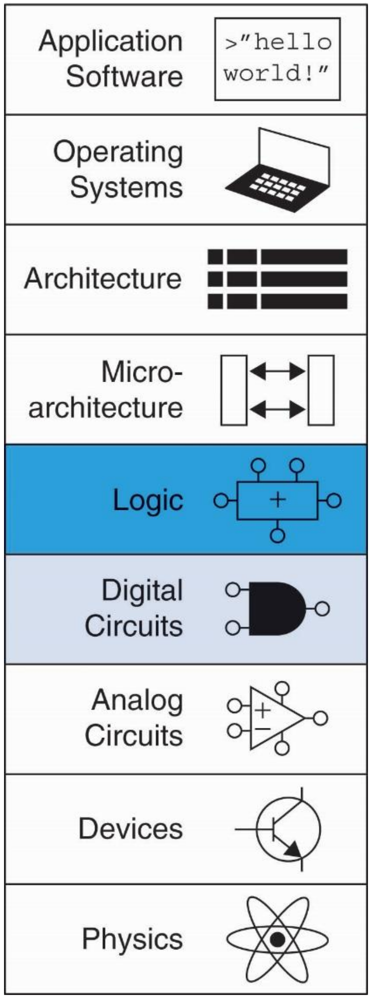
David Money Harris and Sarah L. Harris

ELSEVIER

SEQUENTIAL LOGIC DESIGN

Chapter 3 :: Topics

- **Introduction**
- **Latches and Flip-Flops**
- **Synchronous Logic Design**
- **Finite State Machines**
- **Timing of Sequential Logic**
- **Parallelism**



ELSEVIER

Introduction

- Outputs of sequential logic depend on current *and* prior input values – it has ***memory***.
- Some definitions:
  - **State:** all the information about a circuit necessary to explain its future behavior
  - **Latches and flip-flops:** state elements that store one bit of state
  - **Synchronous sequential circuits:** combinational logic followed by a bank of flip-flops

Sequential Circuits

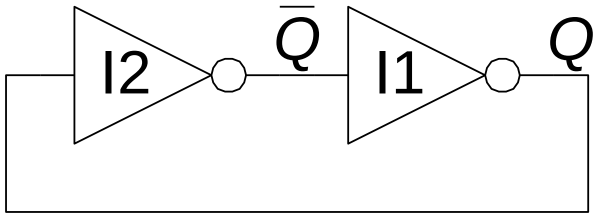
- Give sequence to events
- Have memory (short-term)
- Use feedback from output to input to store information

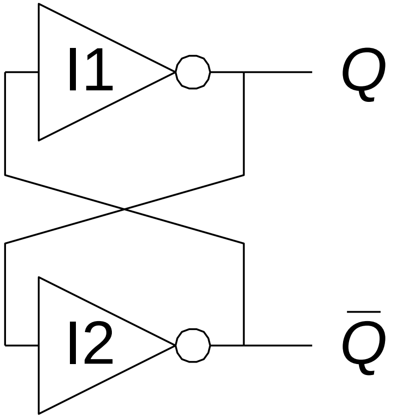
State Elements

- The state of a circuit influences its future behavior
- State elements store state
  - Bistable circuit
  - SR Latch
  - D Latch
  - D Flip-flop

Bistable Circuit

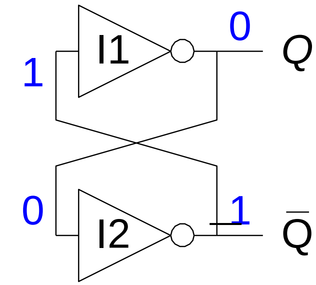
- Fundamental building block of other state elements
- Two outputs: Q, $\overline{Q}$
- No inputs



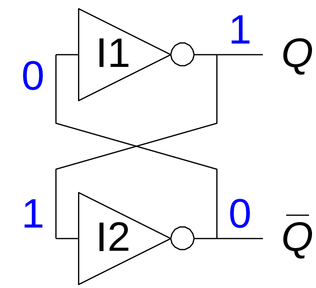


Bistable Circuit Analysis

- Consider the two possible cases:
  - **$Q = 0$:**
    then $\overline{Q} = 1$, $Q = 0$ (consistent)



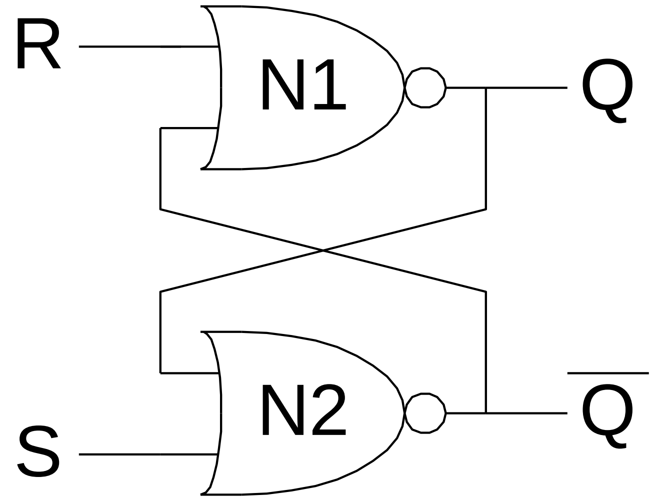
  - **$Q = 1$:**
    then $\overline{Q} = 0$, $Q = 1$ (consistent)



- Stores 1 bit of state in the state variable, Q (or $\overline{Q}$)
- But there are **no inputs to control the state**

SR (Set/Reset) Latch

- SR Latch

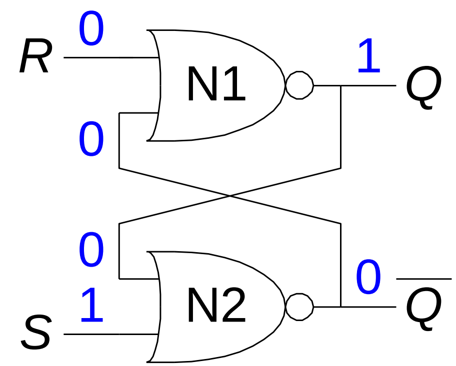


- Consider the four possible cases:
  - $S = 1, R = 0$
  - $S = 0, R = 1$
  - $S = 0, R = 0$
  - $S = 1, R = 1$

SR Latch Analysis

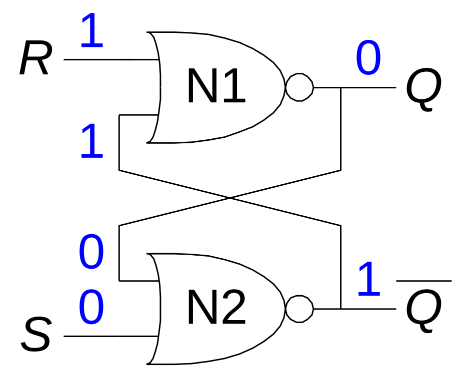
– ***S* = 1, *R* = 0:**

then $Q = 1$ and $\overline{Q} = 0$



– ***S* = 0, *R* = 1:**

then $\overline{Q} = 1$ and $Q = 0$

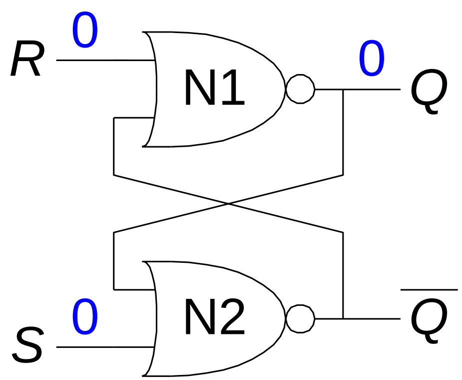


SR Latch Analysis

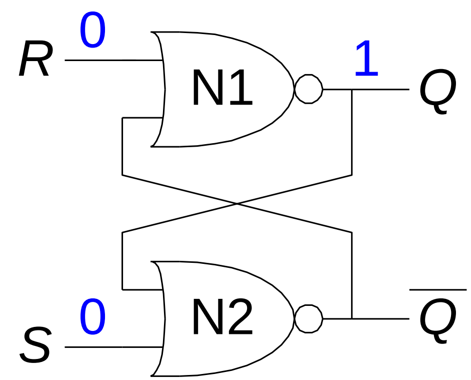
– ***S* = 0, *R* = 0:**

then $Q = Q_{prev}$

$Q_{prev} = 0$

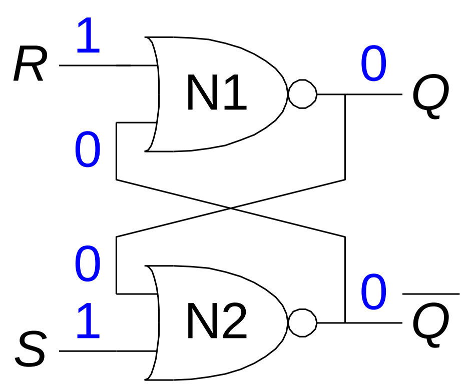


$Q_{prev} = 1$



– ***S* = 1, *R* = 1:**

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



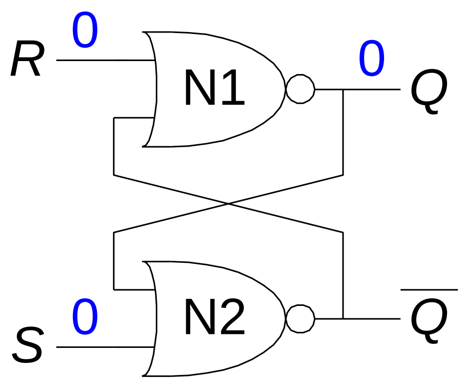
SR Latch Analysis

– $S = 0, R = 0$:

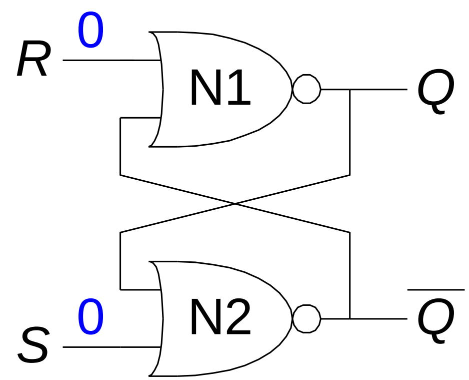
then $Q = Q_{prev}$

– **Memory!**

$Q_{prev} = 0$



$Q_{prev} = 1$

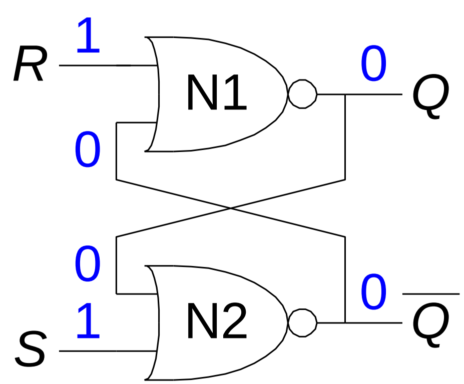


– $S = 1, R = 1$:

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

– **Invalid State**

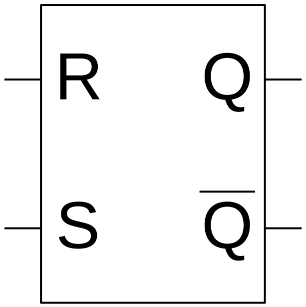
$\overline{Q} \neq \text{NOT } Q$



SR Latch Symbol

- SR stands for Set/Reset Latch
  - Stores one bit of state (Q)
- Control what value is being stored with S, R inputs
  - **Set:** Make the output 1
    ($S = 1$, $R = 0$, $Q = \mathbf{1}$)
  - **Reset:** Make the output 0
    ($S = 0$, $R = 1$, $Q = \mathbf{0}$)

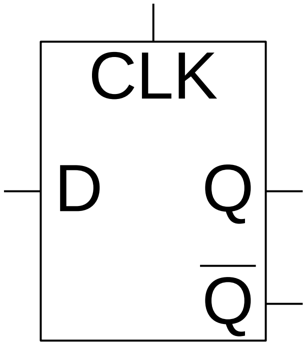
SR Latch Symbol



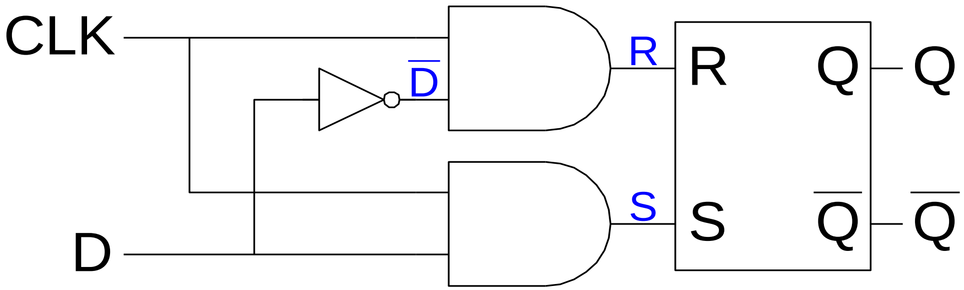
D Latch

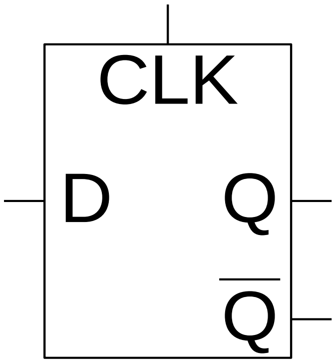
- Two inputs: *CLK*, *D*
  - ***CLK*: controls *when* the output changes**
  - ***D*** (the data input): controls *what* the output changes to
- Function
  - When ***CLK* = 1**, *D* passes through to *Q* (*transparent*)
  - When ***CLK* = 0**, *Q* holds its previous value (*opaque*)
- Avoids invalid case when $Q \neq \text{NOT } \overline{Q}$

D Latch Symbol



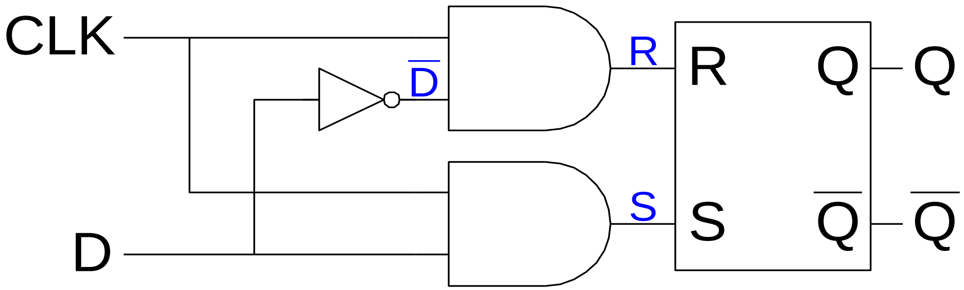
D Latch Internal Circuit

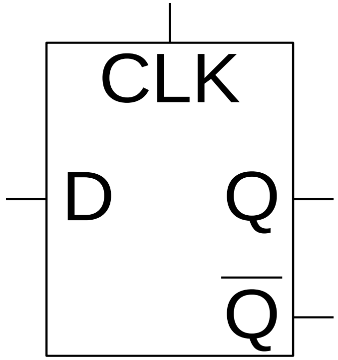




CLK	D	$\overline{D}$	S	R	Q	$\overline{Q}$
0	X					
1	0					
1	1					

D Latch Internal Circuit



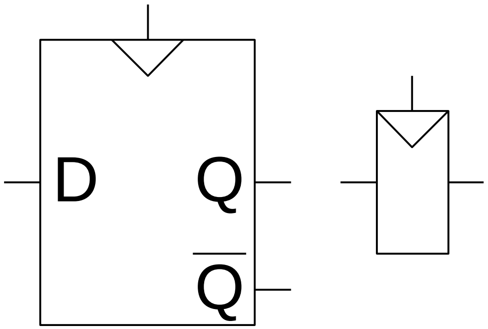


CLK	D	$\overline{D}$	S	R	Q	$\overline{Q}$
0	X	$\overline{X}$	0	0	Q_{prev}	$\overline{Q}_{prev}$
1	0	1	0	1	0	1
1	1	0	1	0	1	0

D Flip-Flop

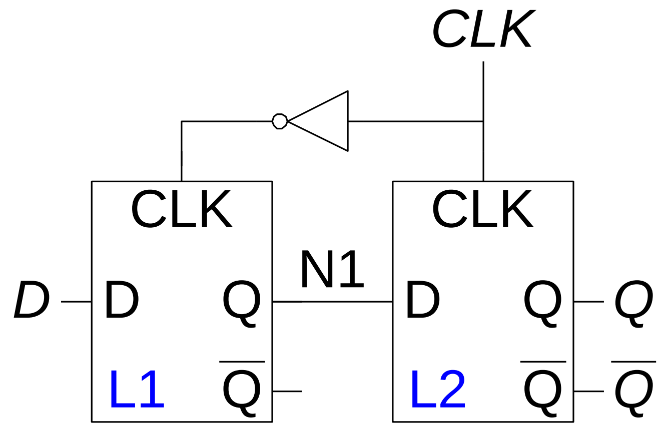
- **Inputs:** *CLK*, *D*
- **Function**
  - Samples *D* on rising edge of *CLK*
    - When *CLK* rises from 0 to 1, *D* passes through to *Q*
    - Otherwise, *Q* holds its previous value
  - *Q* changes only on rising edge of *CLK*
- Called *edge-triggered*
- Activated on the clock edge

D Flip-Flop Symbols



D Flip-Flop Internal Circuit

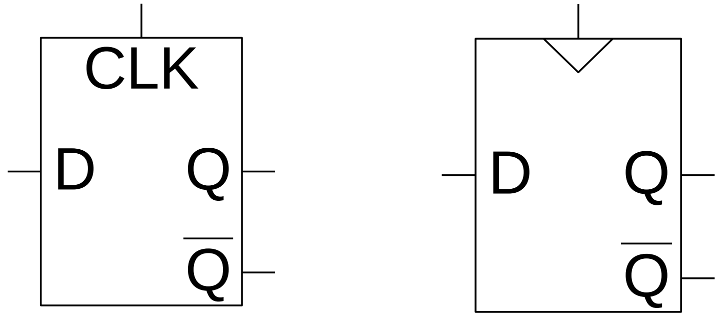
- Two back-to-back latches (L1 and L2) controlled by complementary clocks
- When ***CLK* = 0**
  - L1 is transparent
  - L2 is opaque
  - *D* passes through to N1
- When ***CLK* = 1**
  - L2 is transparent
  - L1 is opaque
  - N1 passes through to *Q*

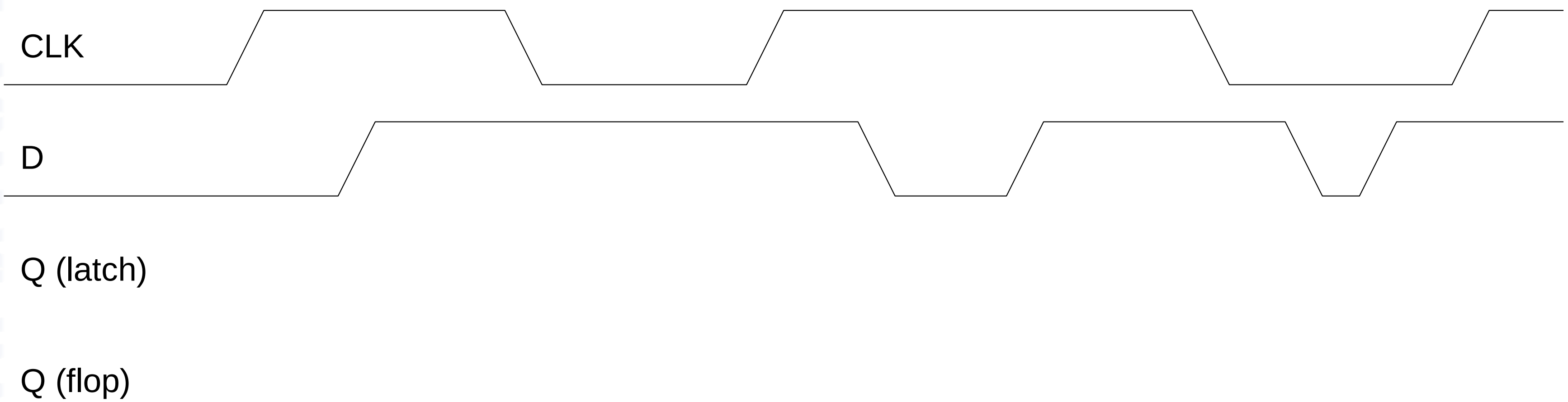


- Thus, on the edge of the clock (when ***CLK* rises from 0 → 1**)
  - *D* passes through to *Q*

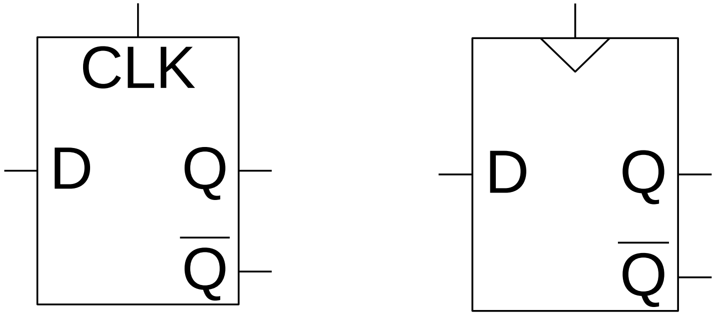
ELSEVIER

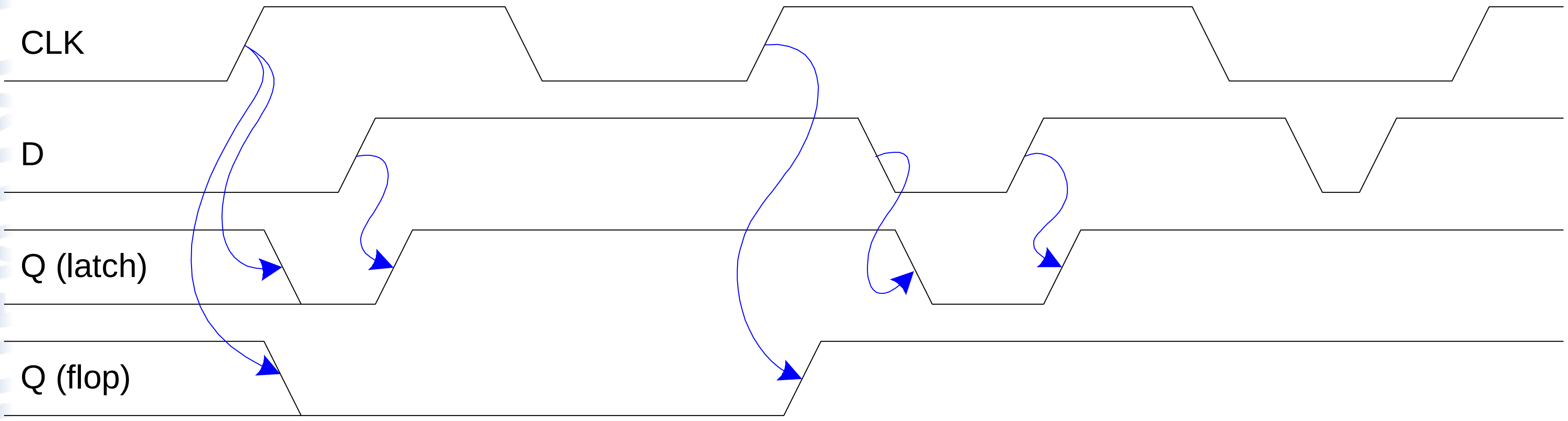
D Latch vs. D Flip-Flop



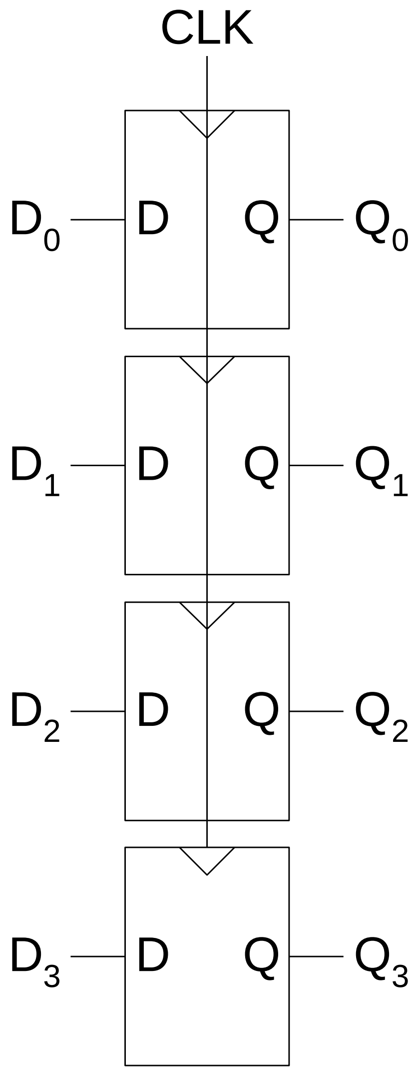


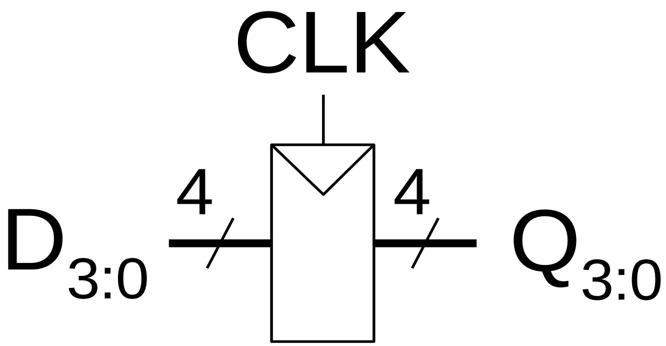
D Latch vs. D Flip-Flop





Registers

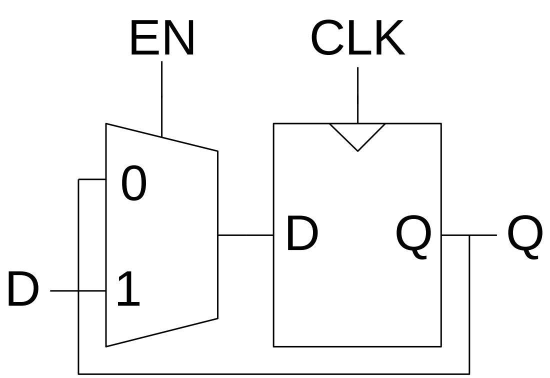




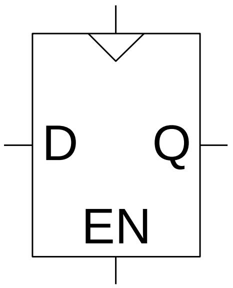
Enabled Flip-Flops

- **Inputs:** *CLK*, *D*, *EN*
  - The enable input (*EN*) controls when new data (*D*) is stored
- **Function**
  - ***EN* = 1:** *D* passes through to *Q* on the clock edge
  - ***EN* = 0:** the flip-flop retains its previous state

Internal Circuit



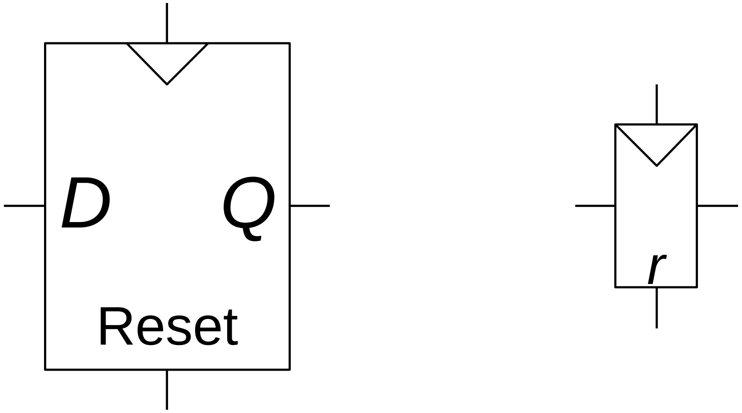
Symbol



Resettable Flip-Flops

- **Inputs:** *CLK*, *D*, *Reset*
- **Function:**
  - ***Reset* = 1:** *Q* is forced to 0
  - ***Reset* = 0:** flip-flop behaves as ordinary D flip-flop

Symbols

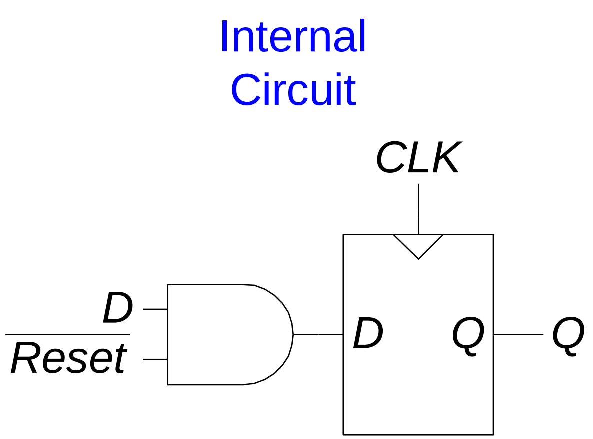


Resettable Flip-Flops

- Two types:
  - **Synchronous:** resets at the clock edge only
  - **Asynchronous:** resets immediately when *Reset* = 1
- Asynchronously resettable flip-flop requires changing the internal circuitry of the flip-flop
- Synchronously resettable flip-flop?

Resettable Flip-Flops

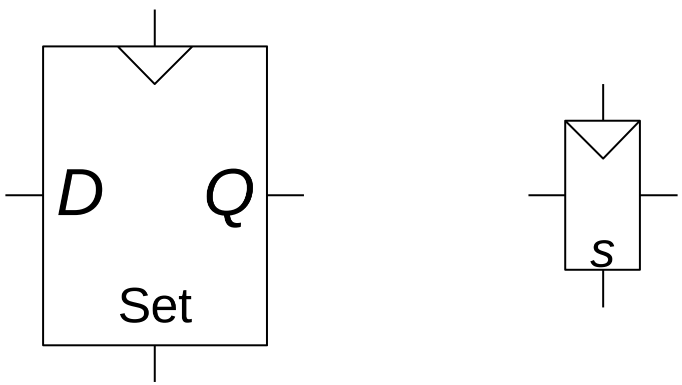
- Two types:
  - **Synchronous:** resets at the clock edge only
  - **Asynchronous:** resets immediately when *Reset* = 1
- Asynchronously resettable flip-flop requires changing the internal circuitry of the flip-flop
- Synchronously resettable flip-flop?



Settable Flip-Flops

- **Inputs:** *CLK*, *D*, *Set*
- **Function:**
  - ***Set* = 1:** *Q* is set to 1
  - ***Set* = 0:** the flip-flop behaves as ordinary D flip-flop

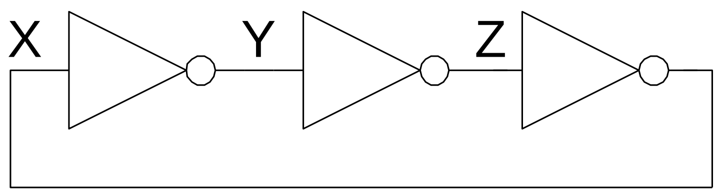
Symbols

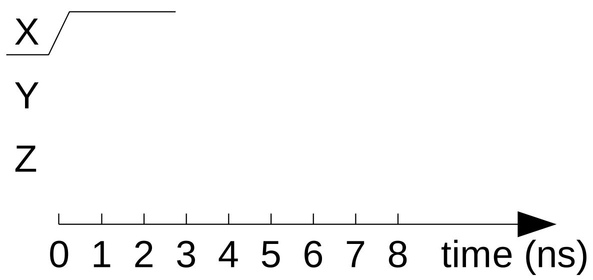


SEQUENTIAL LOGIC DESIGN

Sequential Logic

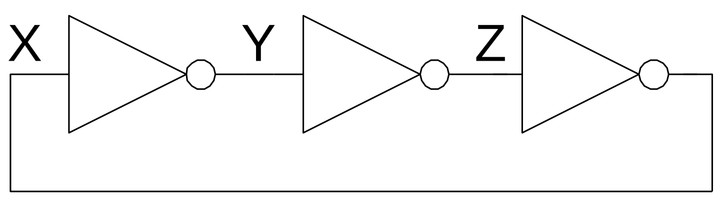
- Sequential circuits: all circuits that aren't combinational
- A problematic circuit:

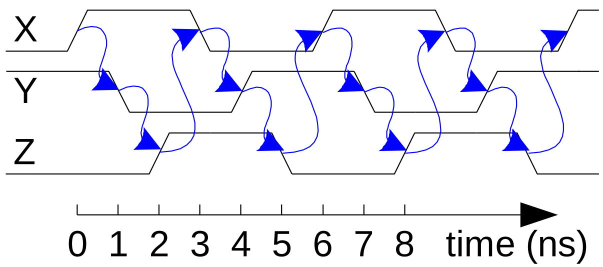




Sequential Logic

- Sequential circuits: all circuits that aren’t combinational
- A problematic circuit:



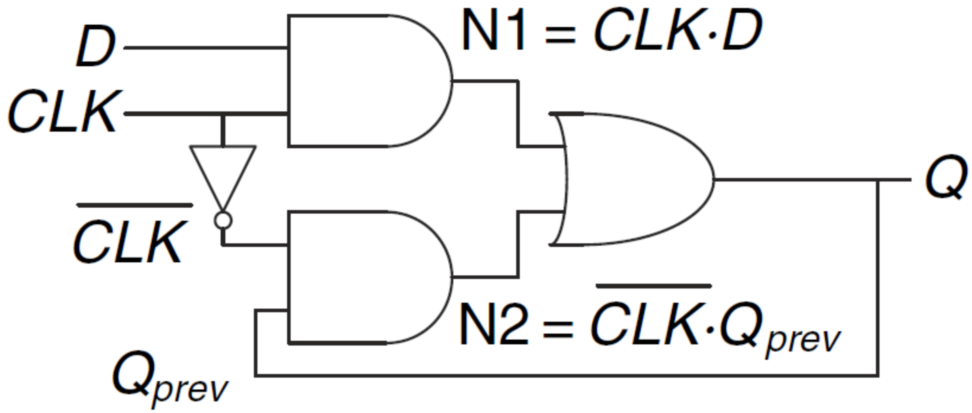


- No inputs and 1-3 outputs
- Astable circuit, oscillates
- Period depends on inverter delay
- It has a ***cyclic path***: output fed back to input

An Alternative D Latch Implementation
Will it always work correctly?

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

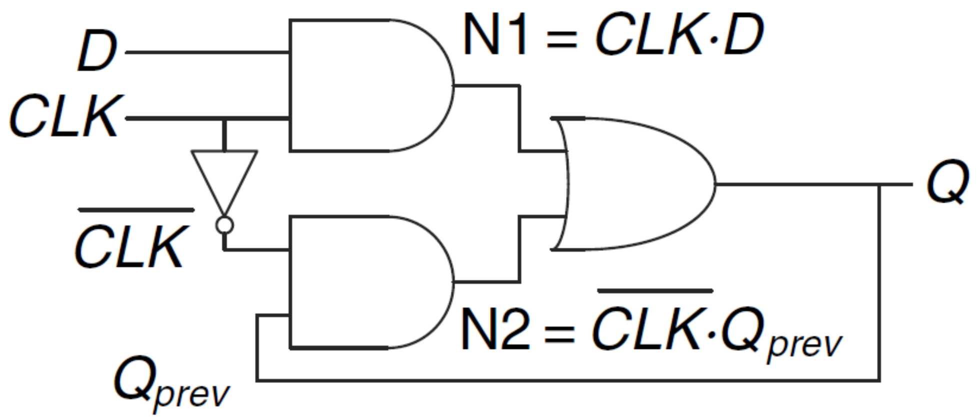
$$Q = CLK \cdot D + \overline{CLK} \cdot Q_{prev}$$

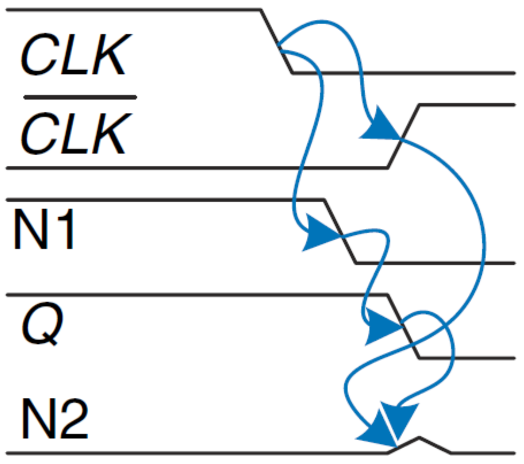


Hint: Assume the inverter delay is very large!

An Alternative D Latch Implementation
Will it always work correctly?

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





Race condition: Assume Q=1 when CLK=1. Then CLK is switched to 0. Assume INV delay is much larger than the delay of AND/OR gates. Then Q will become 0 and will stay zero, but latch should have kept the previous value when CLK=0. Incorrect functionality!

Asynchronous Circuit Design

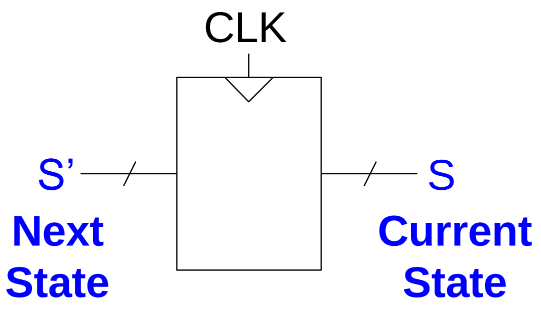
- Outputs directly fed back to inputs.
- Potential race conditions!
  - Behavior of the circuit depends on which path is fastest
- Seemingly identical circuits but with different gate delays may lead to different functionalities.
- Circuit may only work at certain temperatures or voltages.
- Solution: Break cycles by inserting registers -> Synchronous Design

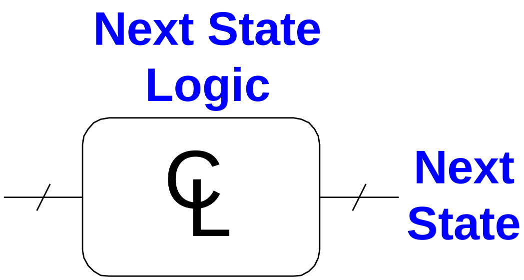
Synchronous Sequential Logic Design

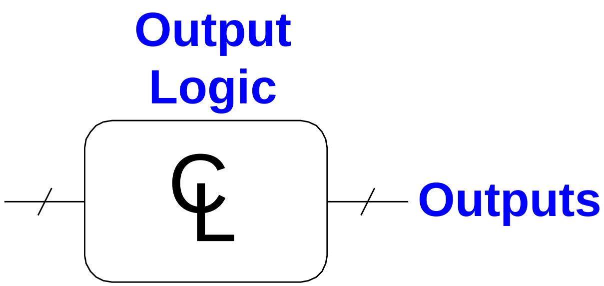
- Breaks cyclic paths by **inserting registers**
- Registers contain **state** of the system
- State changes at clock edge: system **synchronized** to the clock
- **Rules** of synchronous sequential circuit composition:
  - Every circuit element is either a register or a combinational circuit
  - At least one circuit element is a register
  - All registers receive the same clock signal
  - Every cyclic path contains at least one register
- Two common synchronous sequential circuits
  - Finite State Machines (FSMs)
  - Pipelines

Finite State Machine (FSM)

- Consists of:
  - **State register**
    - Stores current state
    - Loads next state at clock edge
  - **Combinational logic**
    - Computes the next state
    - Computes the outputs



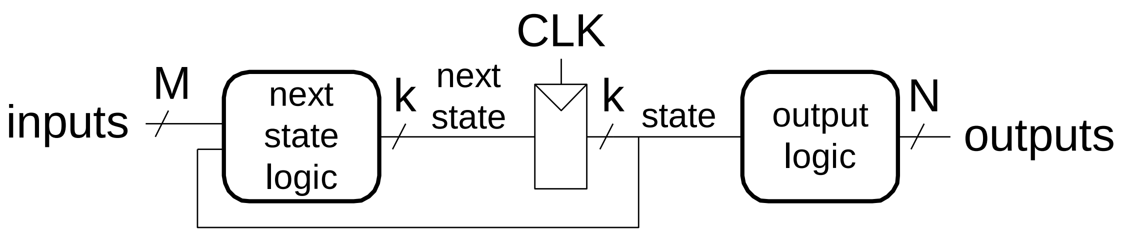




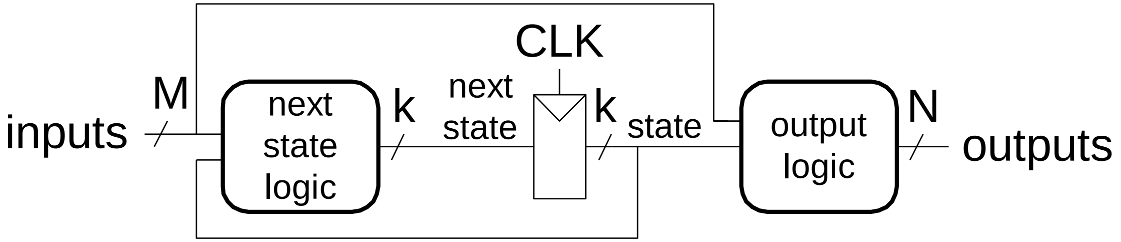
Finite State Machines (FSMs)

- Next state determined by current state and inputs
- Two types of finite state machines differ in output logic:
  - **Moore FSM:** outputs depend only on current state
  - **Mealy FSM:** outputs depend on current state *and* inputs

Moore FSM

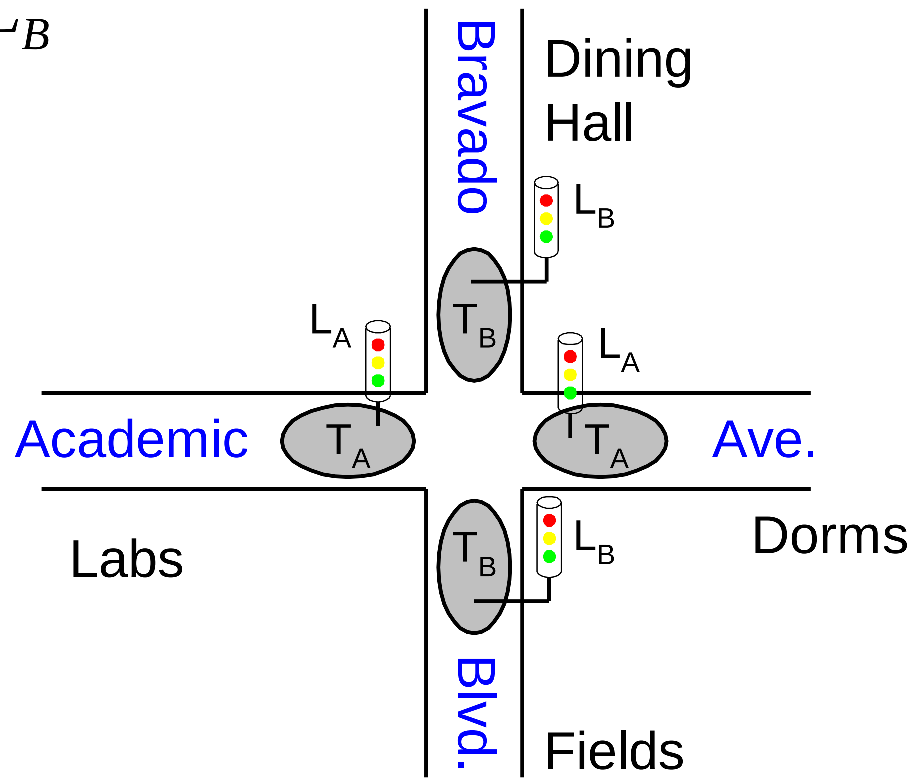


Mealy FSM



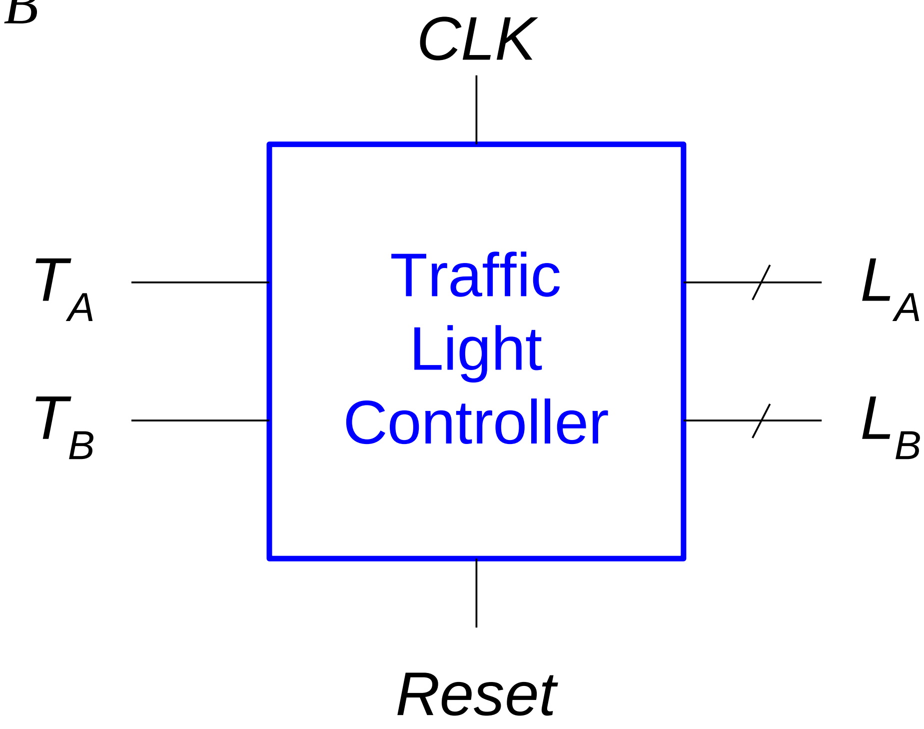
FSM Example

- Traffic light controller
  - Traffic sensors: T_A, T_B (TRUE when there's traffic)
  - Lights: L_A, L_B



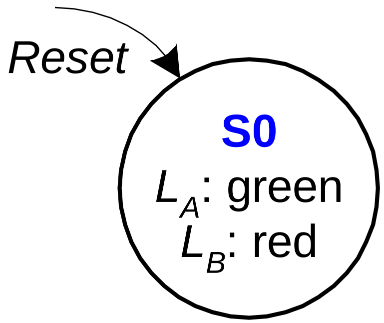
FSM Black Box

- Inputs: CLK, $Reset$, T_A, T_B
- Outputs: L_A, L_B



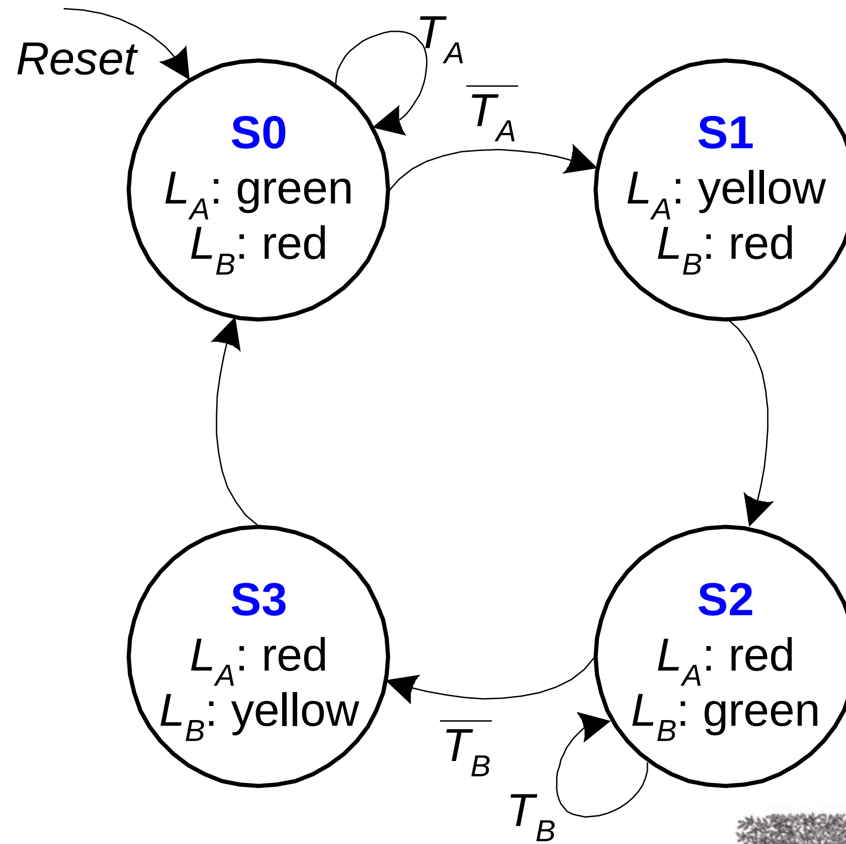
FSM State Transition Diagram

- **Moore FSM:** outputs labeled in each state
- **States:** Circles
- **Transitions:** Arcs



FSM State Transition Diagram

- **Moore FSM:** outputs labeled in each state
- **States:** Circles
- **Transitions:** Arcs



FSM State Transition Table

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

SEQUENTIAL LOGIC DESIGN

ELSEVIER

FSM State Transition Table

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

FSM Encoded State Transition Table

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		

State	Encoding
S0	00
S1	01
S2	10
S3	11

FSM Encoded State Transition Table

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	Encoding
S0	00
S1	01
S2	10
S3	11

$$S'_1 = S_1 \oplus S_0$$

$$S'_0 = \overline{S_1}\,\overline{S_0}\,\overline{T_A} + S_1\overline{S_0}\,\overline{T_B}$$

FSM Output Table

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

Output	Encoding
green	00
yellow	01
red	10

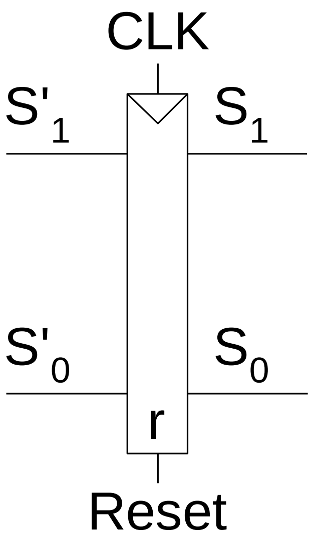
FSM Output Table

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

Output	Encoding
green	00
yellow	01
red	10

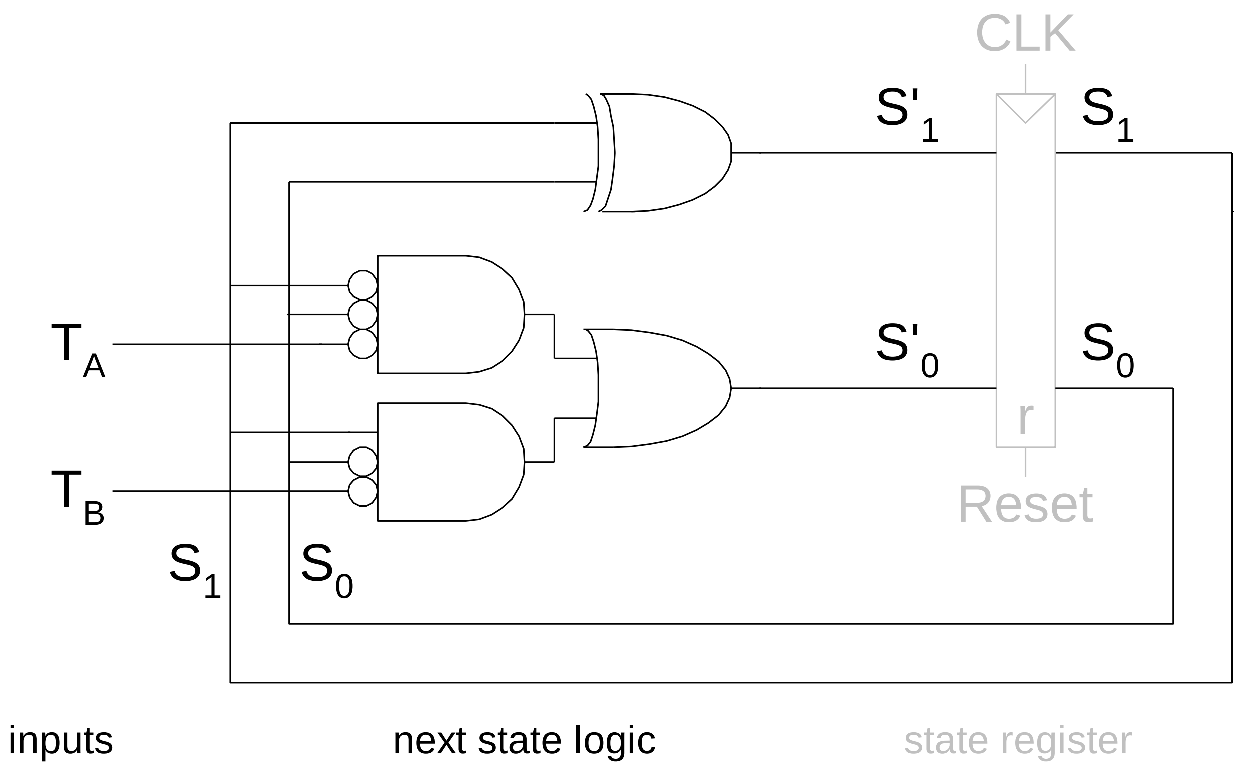
$$L_{A1} = S_1$$
$$L_{A0} = \overline{S_1}S_0$$
$$L_{B1} = \overline{S_1}$$
$$L_{B0} = S_1S_0$$

FSM Schematic: State Register



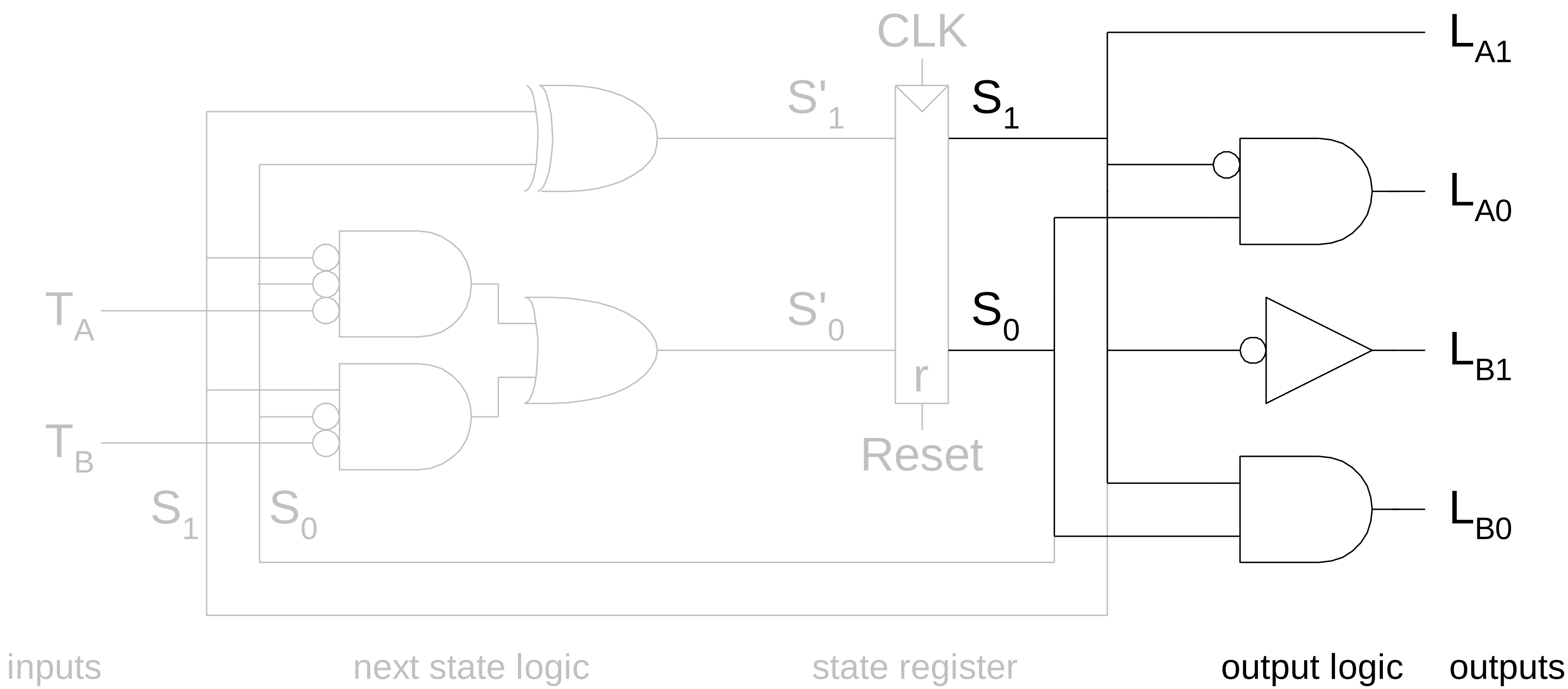
state register

FSM Schematic: Next State Logic



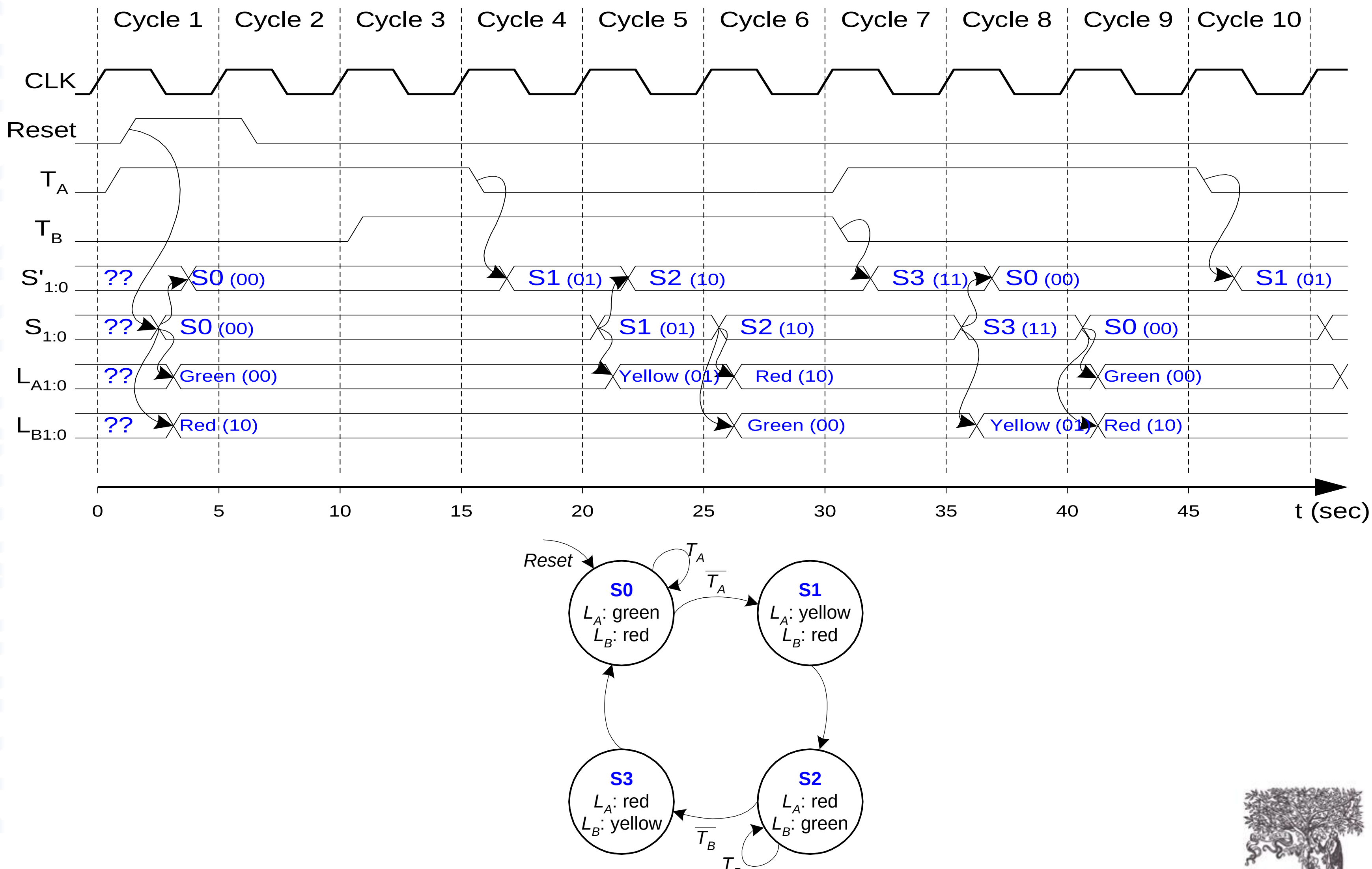
SEQUENTIAL LOGIC DESIGN

FSM Schematic: Output Logic



ELSEVIER

FSM Timing Diagram

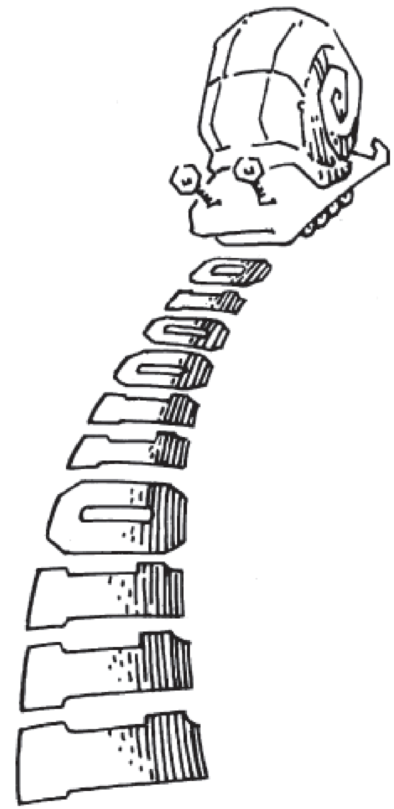


FSM State Encoding

- **Binary** encoding:
  - i.e., for four states, 00, 01, 10, 11
- **One-hot** encoding
  - One state bit per state
  - Only one state bit HIGH at once
  - i.e., for 4 states, 0001, 0010, 0100, 1000
  - Requires more flip-flops
  - Often next state and output logic is simpler

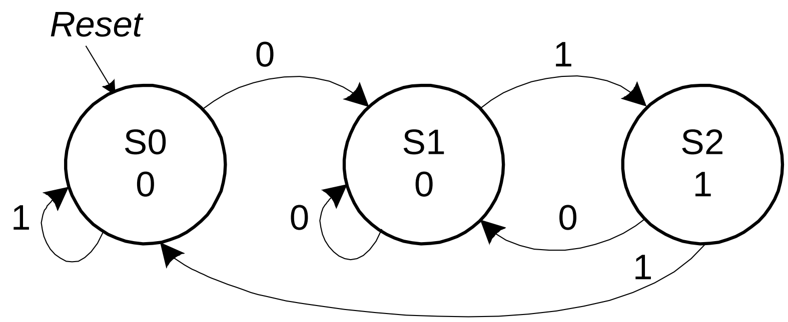
Moore vs. Mealy FSM

- Alyssa P. Hacker has a snail that crawls down a paper tape with 1's and 0's on it. The snail smiles whenever the last two digits it has crawled over are 01. Design Moore and Mealy FSMs of the snail's brain.

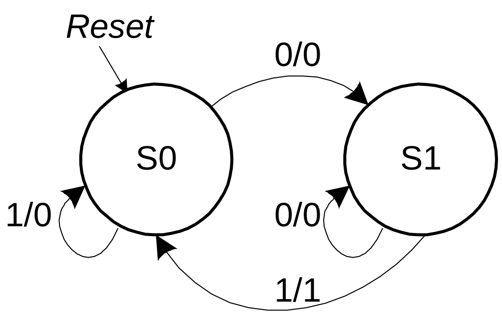


State Transition Diagrams

Moore FSM



Mealy FSM



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

State	Encoding
S0	00
S1	01
S2	10

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

Moore FSM Output Table

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

Moore FSM Output Table

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

$$Y = S_1$$

Mealy FSM State Transition & Output Table

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

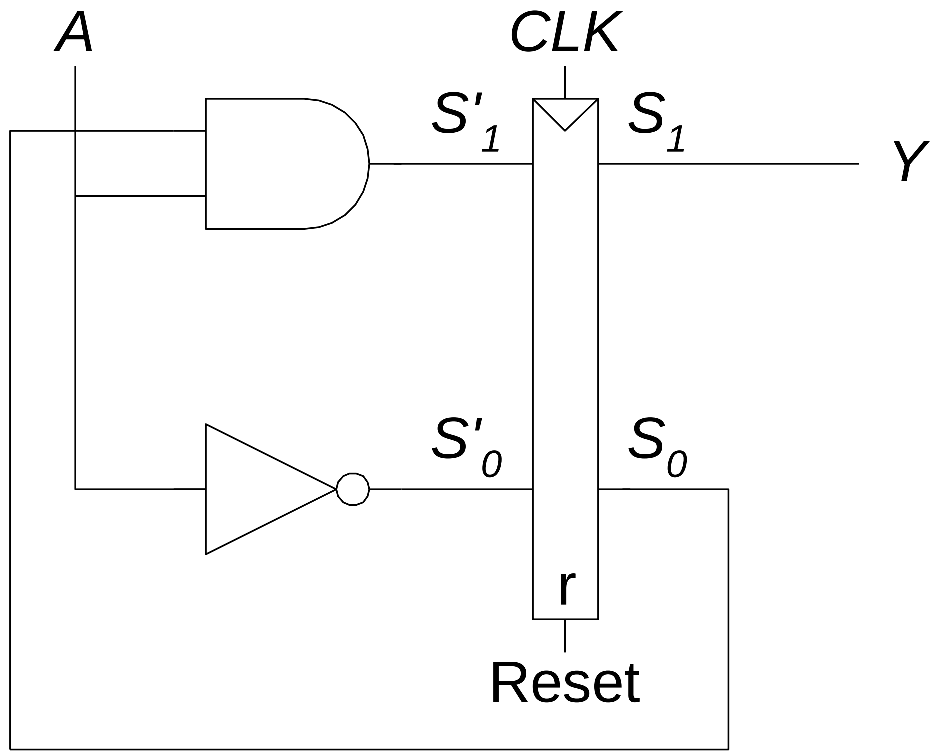
State	Encoding
S0	00
S1	01

Mealy FSM State Transition & Output Table

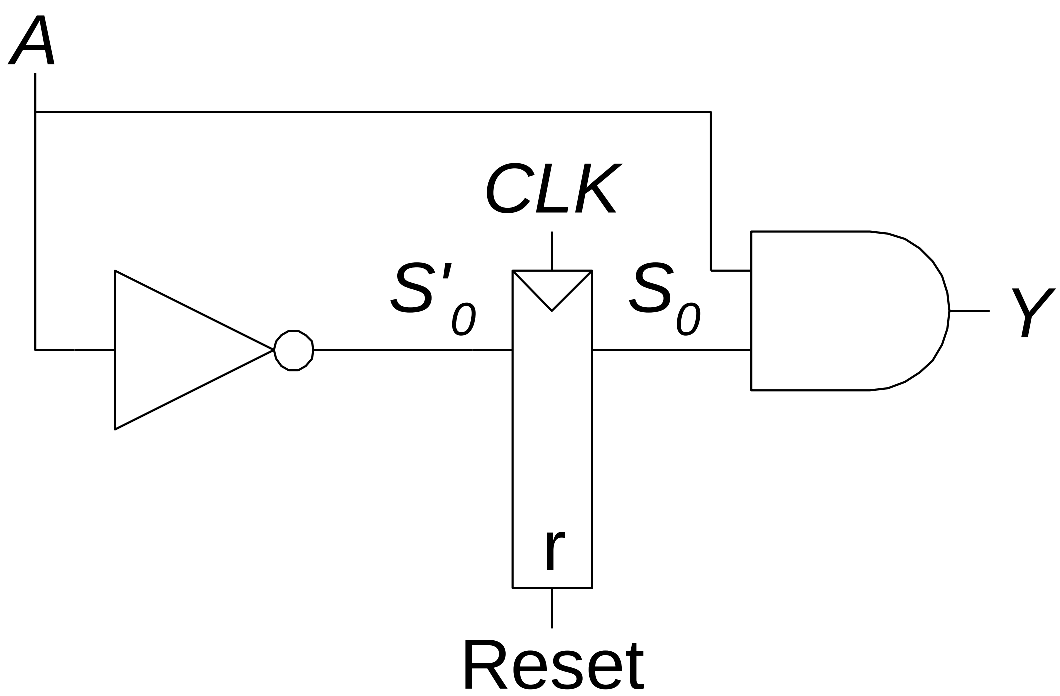
Current State	Input	Next State	Output
S_0	A	S'_0	Y
0	0	1	0
0	1	0	0
1	0	1	0
1	1	0	1

State	Encoding
S0	00
S1	01

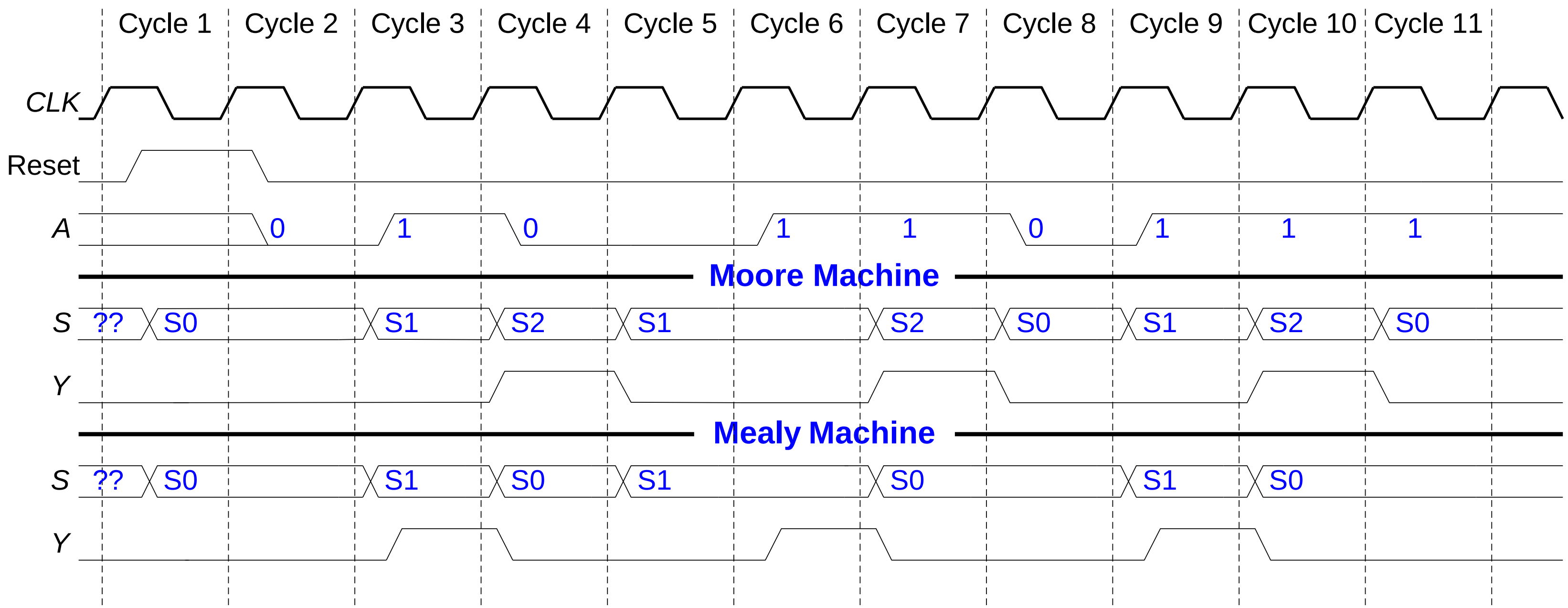
Moore FSM Schematic



Mealy FSM Schematic



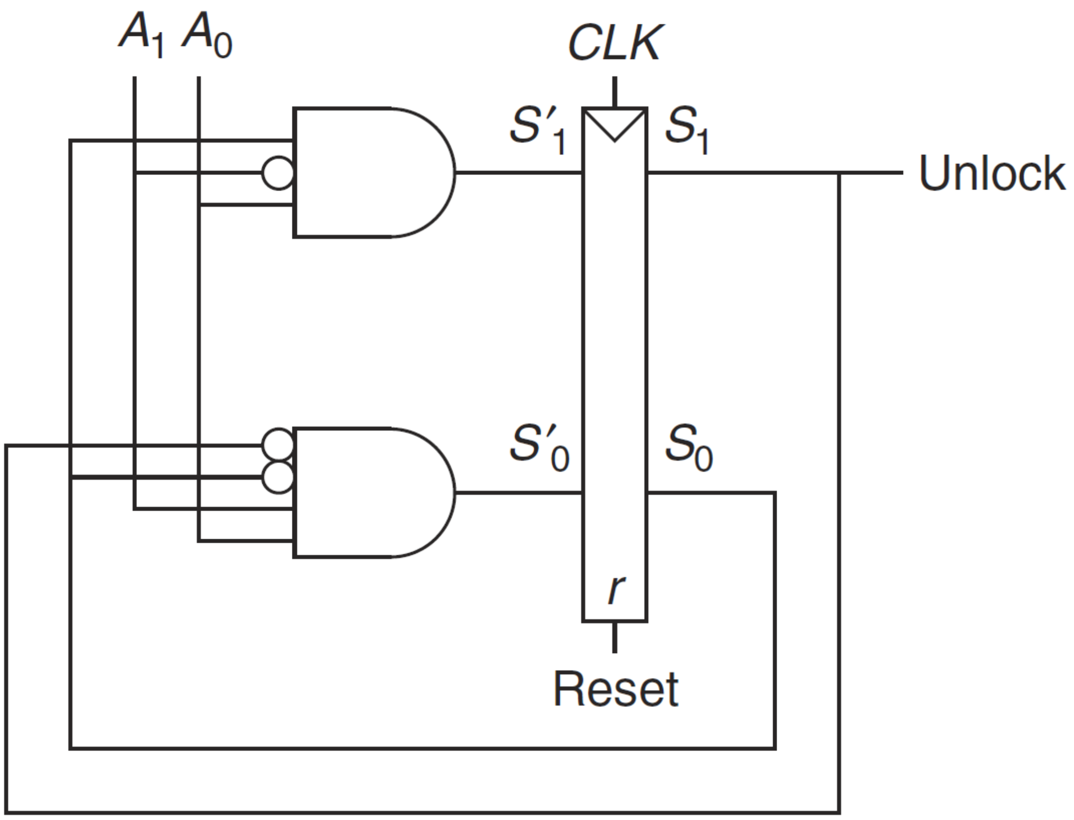
Moore & Mealy Timing Diagram



FSM Design Procedure

- Identify inputs and outputs
- Sketch state transition diagram
- Write state transition table
- Select state encodings
- For Moore machine:
  - Rewrite state transition table with state encodings
  - Write output table
- For a Mealy machine:
  - Rewrite combined state transition and output table with state encodings
- Write Boolean equations for next state and output logic
- Sketch the circuit schematic

Deriving an FSM from its Circuit



$$S'_1 = S_0 \overline{A_1} A_0$$

$$S'_0 = \overline{S_1}\,\overline{S_0} A_1 A_0$$

$$Unlock = S_1$$

What does this circuit do?

Next State Table

Current State S_1	Current State S_0	Input A_1	Input A_0	Next State S'_1	Next State S'_0
0	0	0	0	0	0
0	0	0	1	0	0
0	0	1	0	0	0
0	0	1	1	0	1
0	1	0	0	0	0
0	1	0	1	1	0
0	1	1	0	0	0
0	1	1	1	0	0
1	0	0	0	0	0
1	0	0	1	0	0
1	0	1	0	0	0
1	0	1	1	0	0
1	1	0	0	0	0
1	1	0	1	1	0
1	1	1	0	0	0
1	1	1	1	0	0

Observation 1: Next state is never 11

Observation 2: Next state after 10 is 00

Reduced Next State Table

Current State S_1	Current State S_0	Input A_1	Input A_0	Next State S'_1	Next State S'_0
0	0	0	0	0	0
0	0	0	1	0	0
0	0	1	0	0	0
0	0	1	1	0	1
0	1	0	0	0	0
0	1	0	1	1	0
0	1	1	0	0	0
0	1	1	1	0	0
1	0	X	X	0	0

Symbolic Next State Table

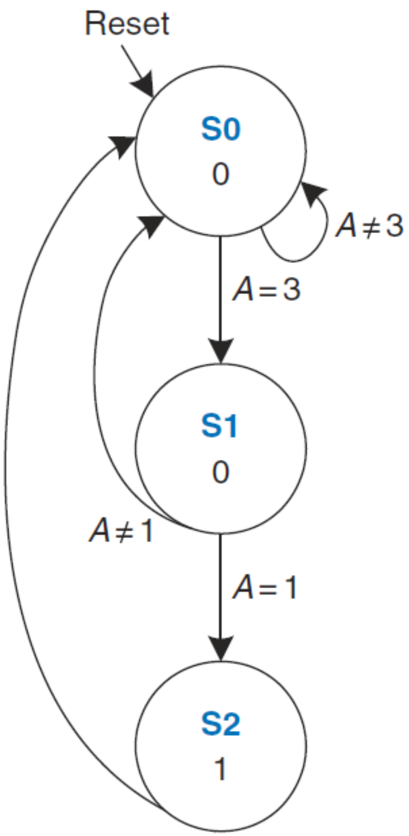
Current State S	Input A	Next State S'
S0	0	S0
S0	1	S0
S0	2	S0
S0	3	S1
S1	0	S0
S1	1	S2
S1	2	S0
S1	3	S0
S2	X	S0

Output Table

Current State S_1	S_0	Output *Unlock*
0	0	0
0	1	0
1	0	1

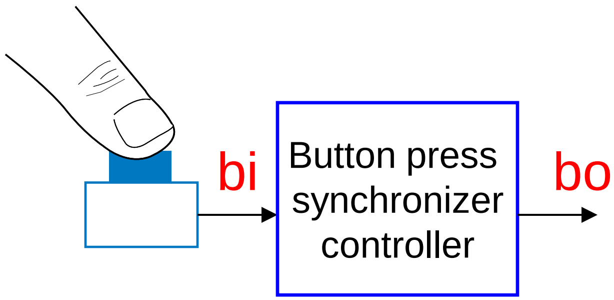
Current State S	Output *Unlock*
S0	0
S1	0
S2	1

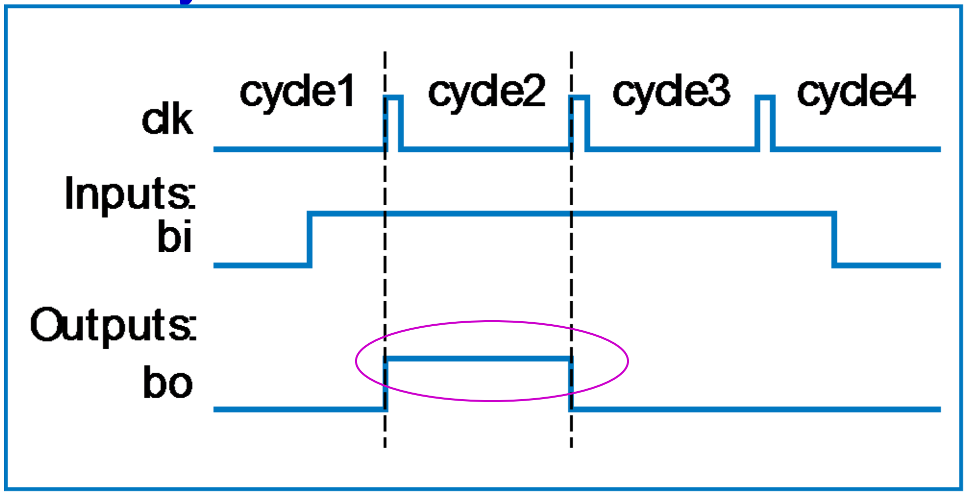
FSM and Functionality



Detects input sequence of 3 followed by 1, and outputs 1 (e.g. unlocks a door), and then restarts in the next cycle (e.g. locks the door again).

Controller Example: Button Press Synchronizer

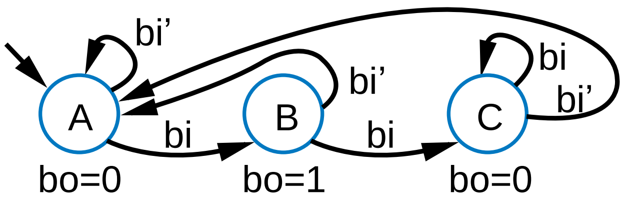




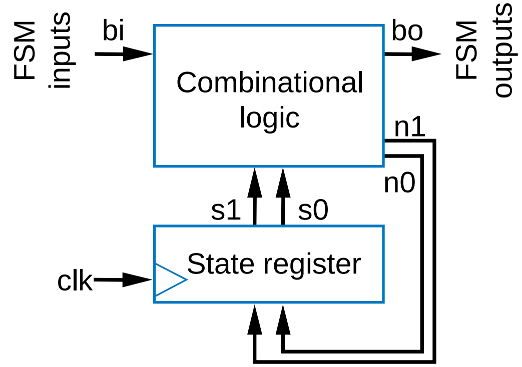
- Want simple sequential circuit that converts button press to single cycle duration, regardless of length of time that button was actually pressed

Controller Example: Button Press Synchronizer (cont)

FSM inputs: bi; FSM outputs: bo

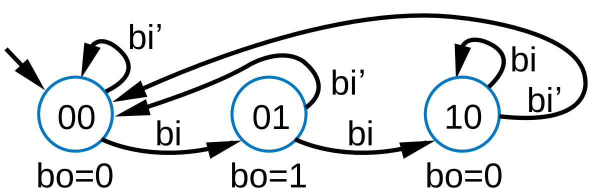


Step 1: Capture FSM



Step 2A: Set up architecture

FSM inputs: bi; FSM outputs: bo



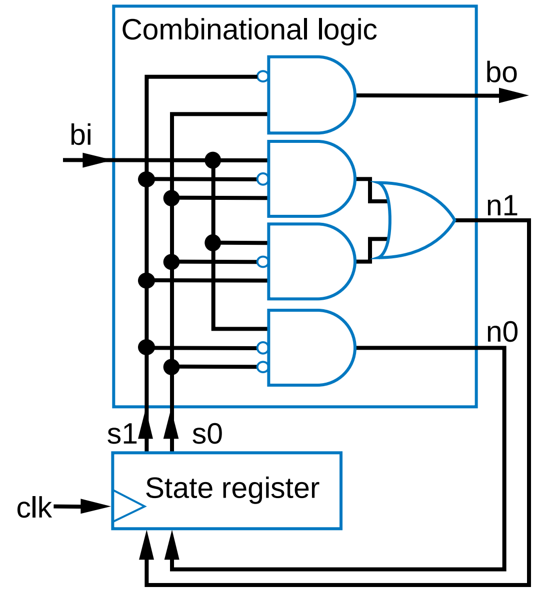
Step 2B: Encode states

Combinational logic

	Inputs			Outputs		
	s1	s0	bi	n1	n0	bo
A	0	0	0	0	0	0
	0	0	1	0	1	0
B	0	1	0	0	0	1
	0	1	1	1	0	1
C	1	0	0	0	0	0
	1	0	1	1	0	0
unused	1	1	0	0	0	0
	1	1	1	0	0	0

Step 2C: Fill in truth table

n1 = s1's0bi + s1s0'bi
n0 = s1's0'bi
bo = s1's0bi' + s1's0bi = s1s0



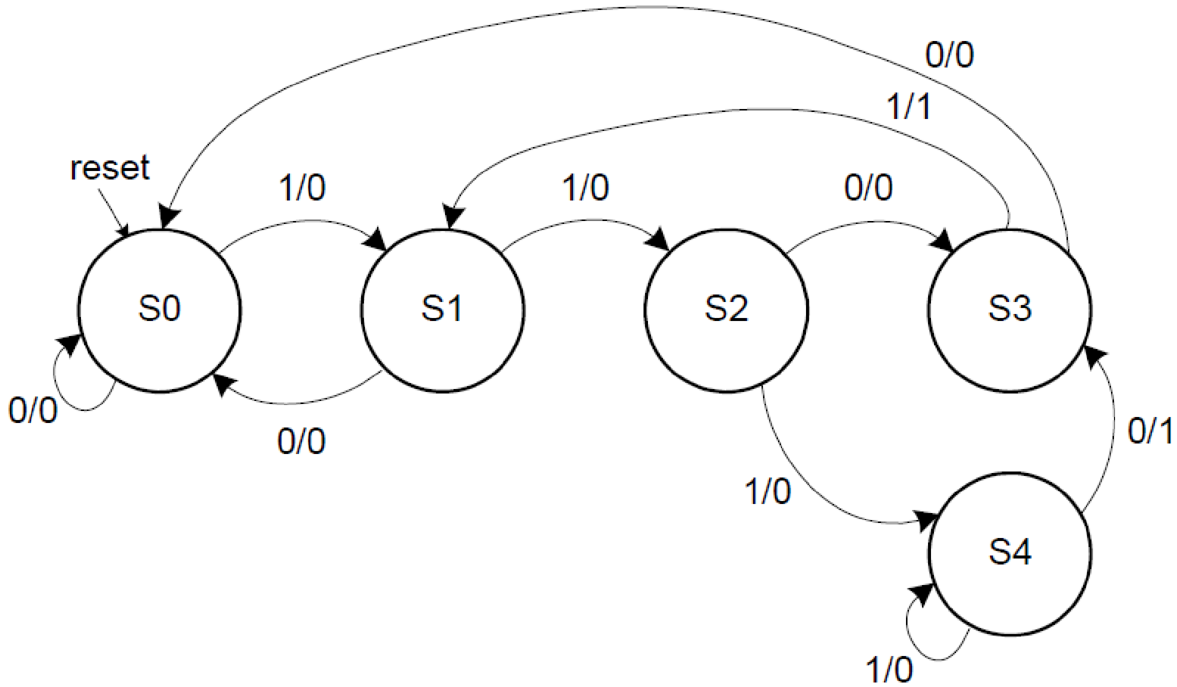
Step 2D: Implement combinational logic

How about Mealy FSM?

FSM Example

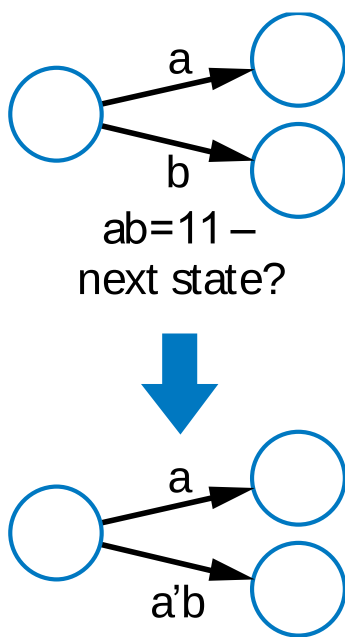
- Example 3.25
  - Daughter snail smiles whenever she slides over 1101 or the 1110 pattern
  - Draw state diagram and respective circuit

FSM Example (cont’d)

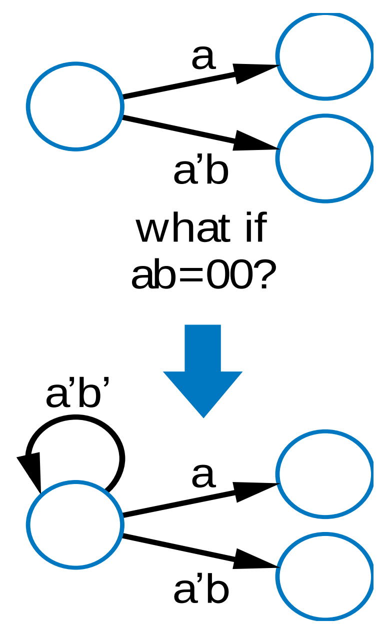


Common Mistakes when Capturing FSMs

- Non-exclusive transitions

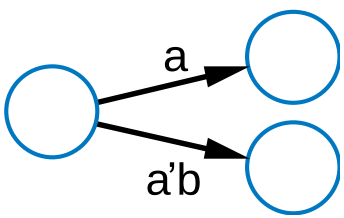


- Incomplete transitions



Verifying Correct Transition Properties

- Can verify using Boolean algebra
  - Only one condition true: AND of each condition pair (for transitions leaving a state) should equal 0 → proves pair can never simultaneously be true
  - One condition true: OR of all conditions of transitions leaving a state) should equal 1 → proves at least one condition must be true
  - Example



Answer:

a * a'b
= (a * a') * b
= 0 * b
= 0
OK!

a + a'b
= a*(1+b) + a'b
= a + ab + a'b
= a + (a+a')b
= a + b
Fails! Might not be 1 (i.e., a=0, b=0)

Q: For shown transitions, prove whether:
* Only one condition true (AND of each pair is always 0)
* One condition true (OR of all transitions is always 1)

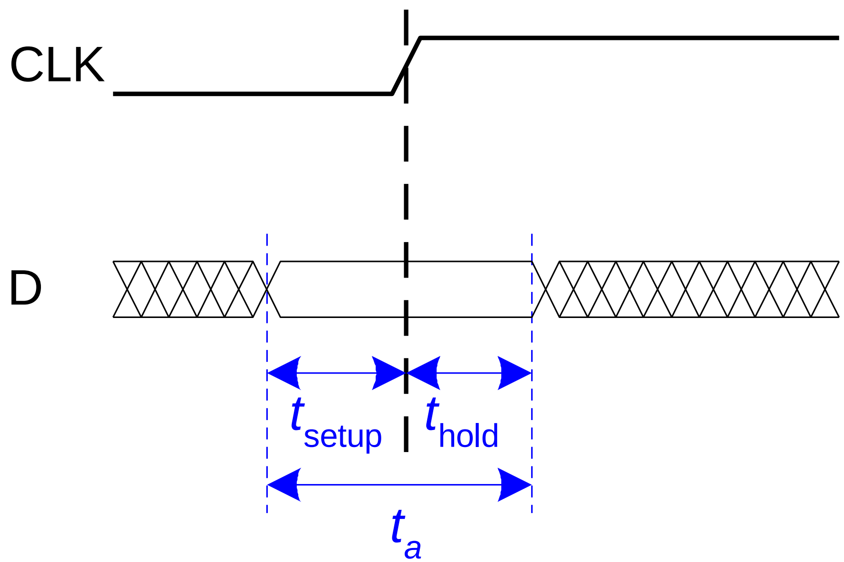
SEQUENTIAL LOGIC DESIGN

Timing

- Flip-flop samples *D* at clock edge
- *D* must be stable when sampled
- Similar to a photograph, *D* must be stable around clock edge
- If not, metastability can occur

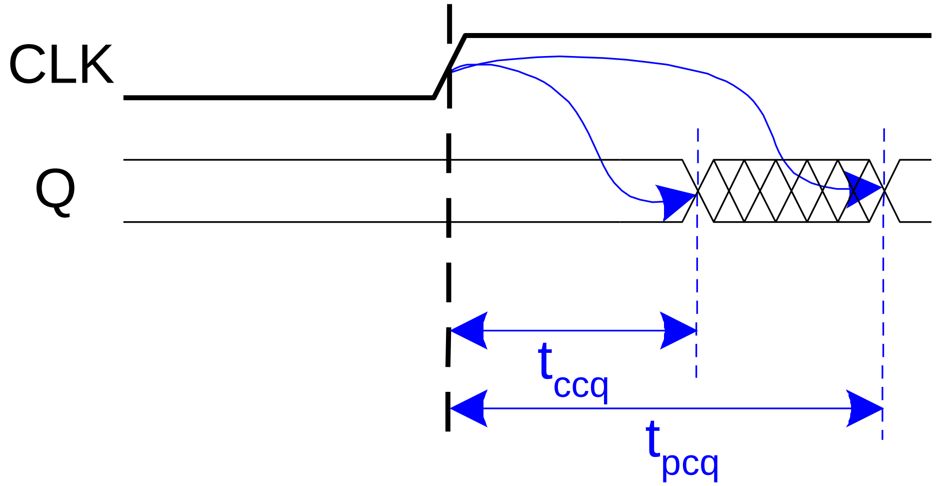
Input Timing Constraints

- **Setup time:** t_{setup} = time *before* clock edge data must be stable (i.e. not changing)
- **Hold time:** t_{hold} = time *after* clock edge data must be stable
- **Aperture time:** t_a = time *around* clock edge data must be stable ($t_a = t_{setup} + t_{hold}$)



Output Timing Constraints

- **Propagation delay:** t_{pcq} = time after clock edge that the output *Q* is guaranteed to be stable (i.e., to stop changing)
- **Contamination delay:** t_{ccq} = time after clock edge that *Q* might be unstable (i.e., start changing)



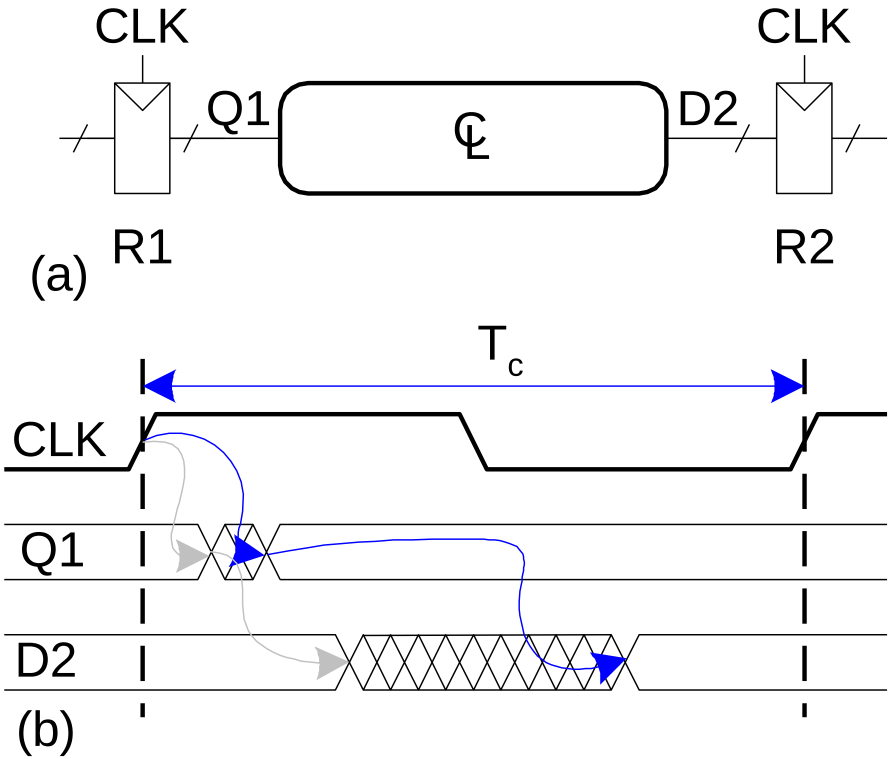
ELSEVIER

Dynamic Discipline

- Synchronous sequential circuit inputs must be stable during aperture (setup and hold) time around clock edge
- Specifically, inputs must be stable
  - at least t_{setup} before the clock edge
  - at least until t_{hold} after the clock edge

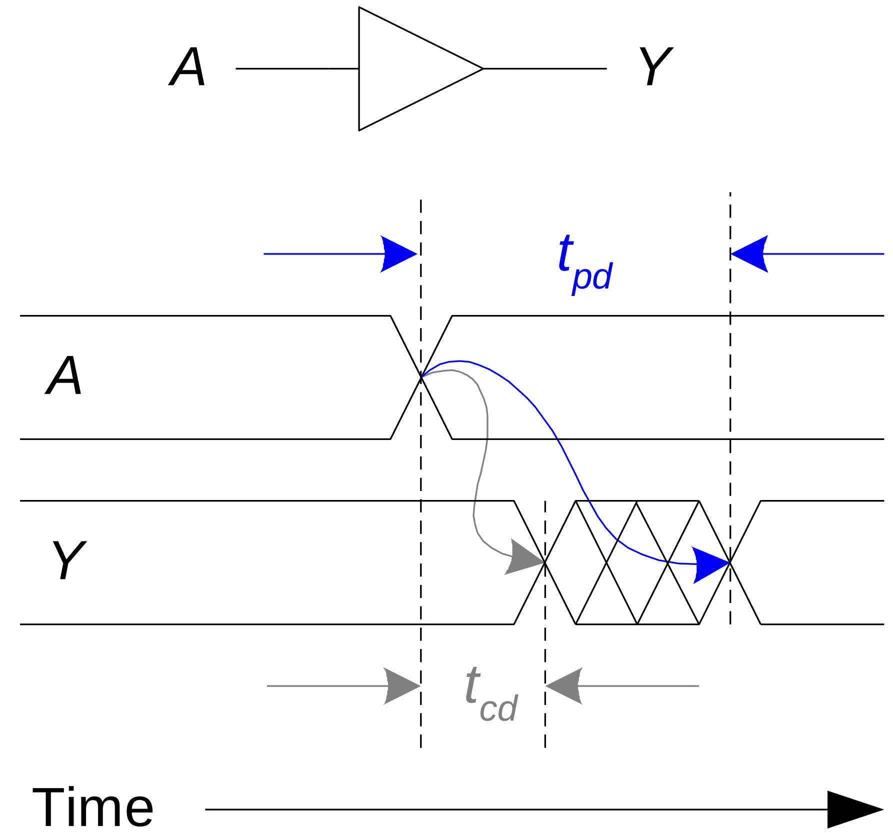
Dynamic Discipline

- The delay between registers has a **minimum** and **maximum** delay, dependent on the delays of the circuit elements



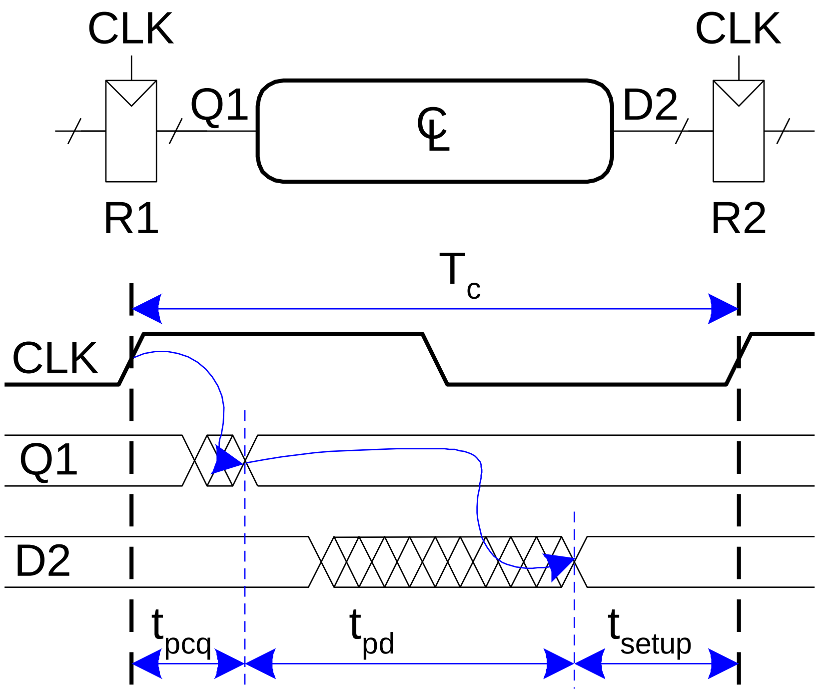
Reminder: Propagation & Contamination Delays of Combinational Logic

- **Propagation delay:** t_{pd} = max delay from input to output
- **Contamination delay:** t_{cd} = min delay from input to output



Setup Time Constraint

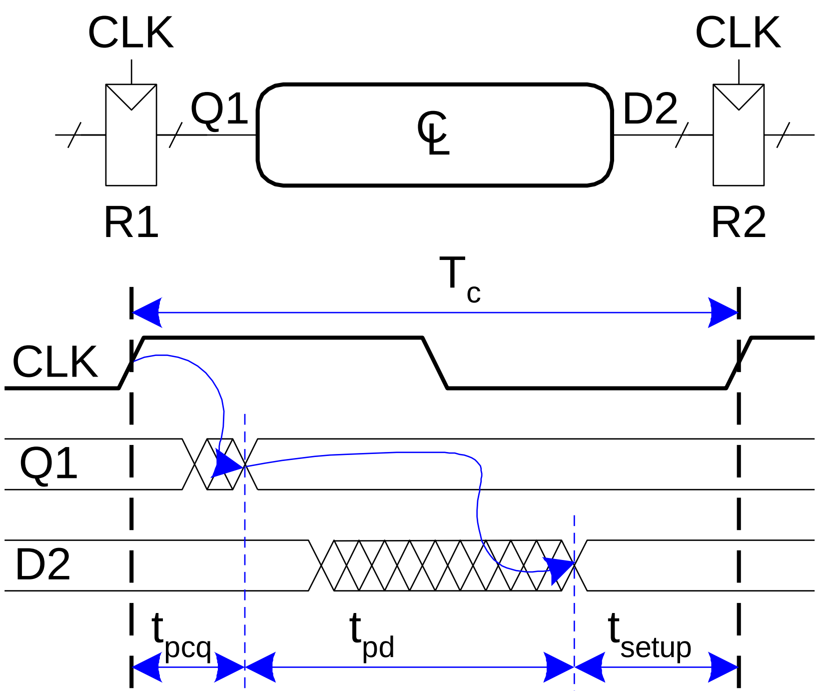
- Depends on the **maximum** delay from register R1 through combinational logic to R2
- The input to register R2 must be stable at least t_{setup} before clock edge



$T_c \geq$

Setup Time Constraint

- Depends on the **maximum** delay from register R1 through combinational logic to R2
- The input to register R2 must be stable at least t_{setup} before clock edge

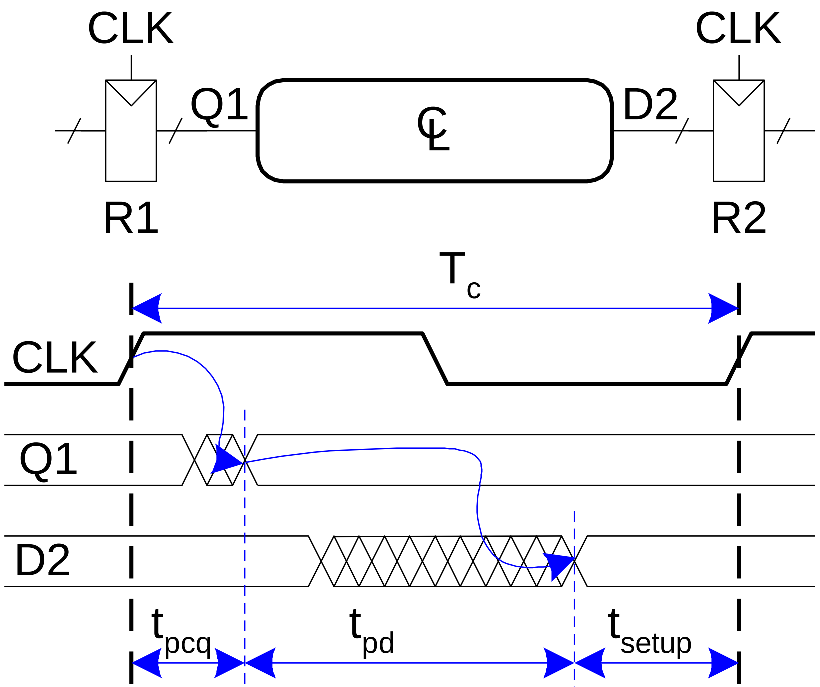


$$T_c \geq t_{pcq} + t_{pd} + t_{setup}$$

$$t_{pd} \leq$$

Setup Time Constraint

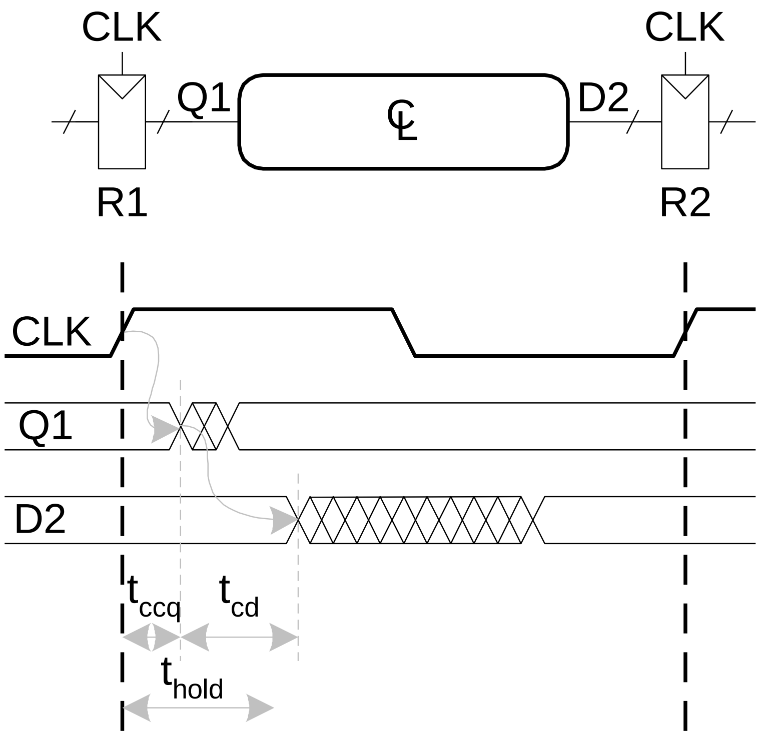
- Depends on the **maximum** delay from register R1 through combinational logic to R2
- The input to register R2 must be stable at least t_{setup} before clock edge



$$T_c \geq t_{pcq} + t_{pd} + t_{setup}$$
$$t_{pd} \leq T_c - (t_{pcq} + t_{setup})$$

Hold Time Constraint

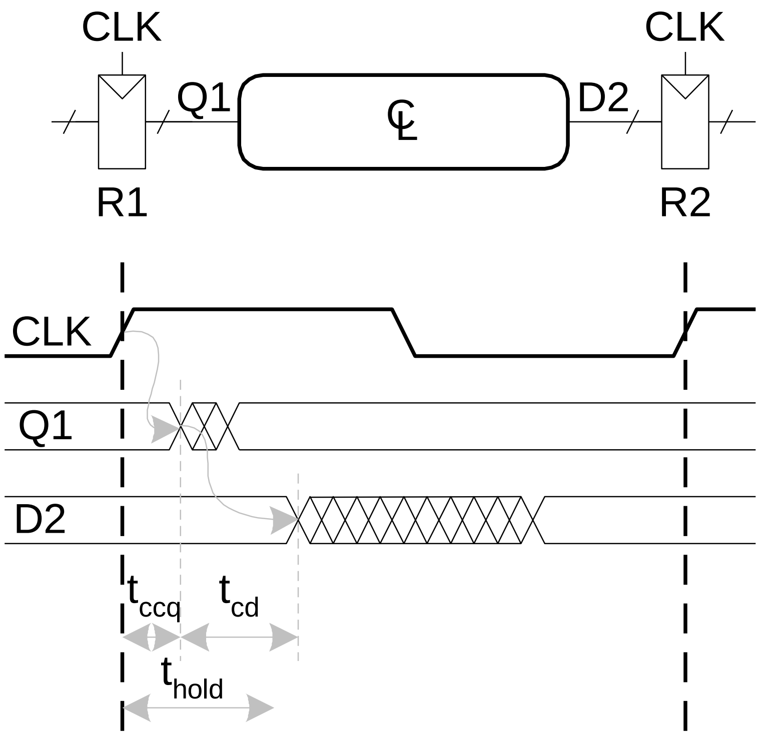
- Depends on the **minimum** delay from register R1 through the combinational logic to R2
- The input to register R2 must be stable for at least t_{hold} after the clock edge



$t_{hold} <$

Hold Time Constraint

- Depends on the **minimum** delay from register R1 through the combinational logic to R2
- The input to register R2 must be stable for at least t_{hold} after the clock edge

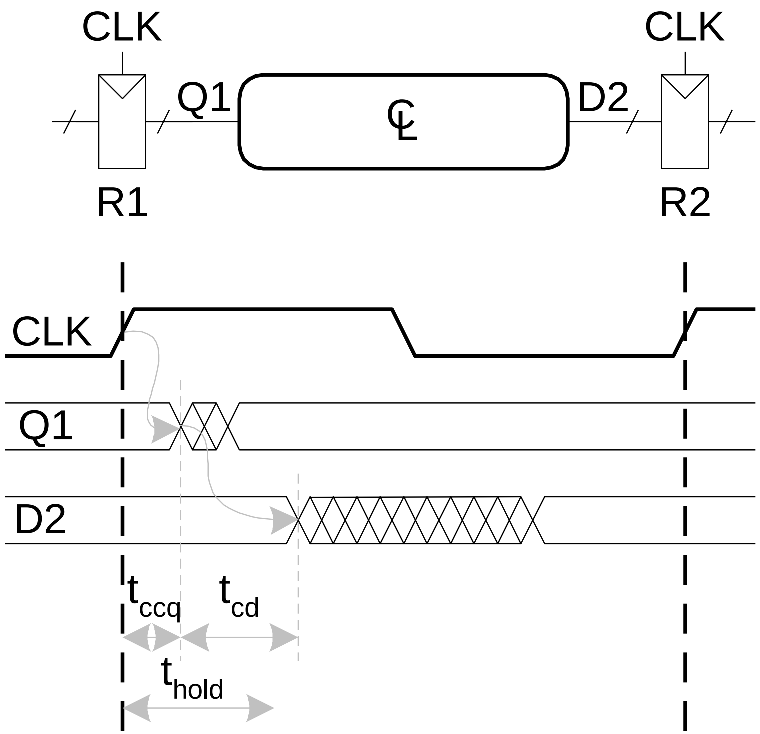


$t_{hold} < t_{ccq} + t_{cd}$

$t_{cd} >$

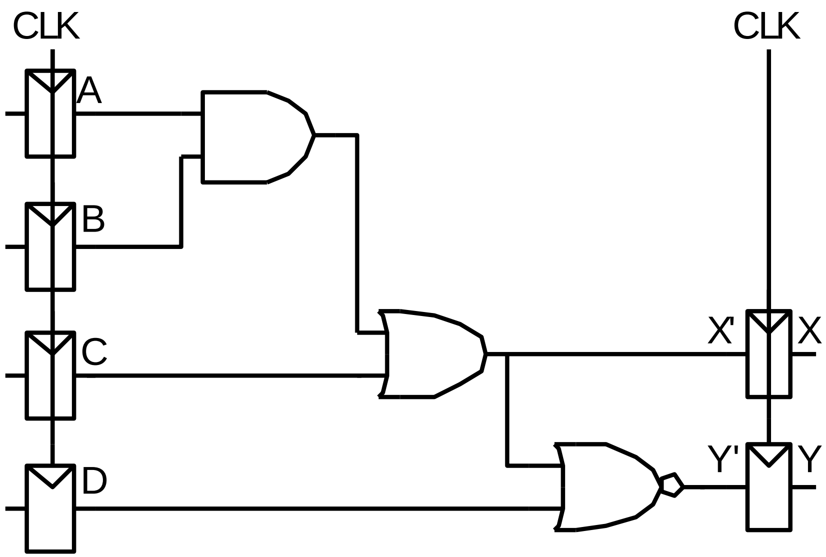
Hold Time Constraint

- Depends on the **minimum** delay from register R1 through the combinational logic to R2
- The input to register R2 must be stable for at least t_{hold} after the clock edge



$$t_{hold} < t_{ccq} + t_{cd}$$
$$t_{cd} > t_{hold} - t_{ccq}$$

Timing Analysis



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} =

t_{cd} =

Setup time constraint:

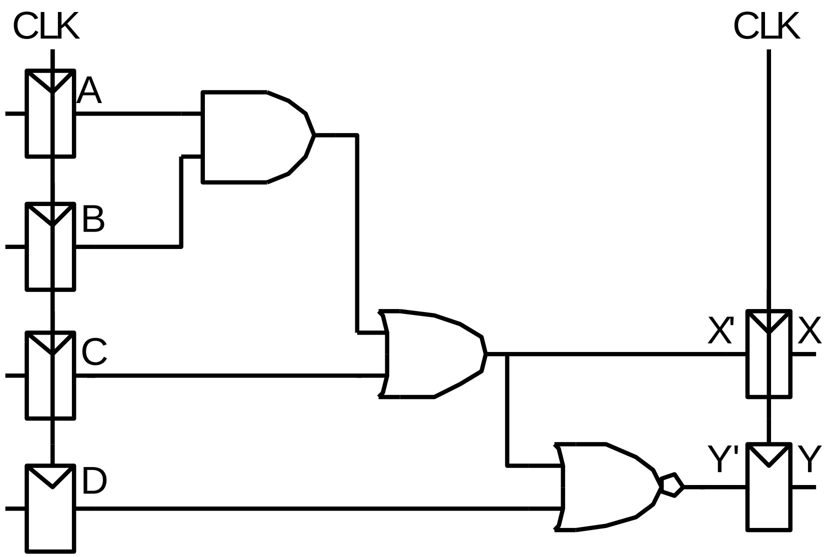
$T_c \geq$

f_c =

Hold time constraint:

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

Timing Analysis



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:

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

$f_c = 1/T_c$ = 4.65 GHz

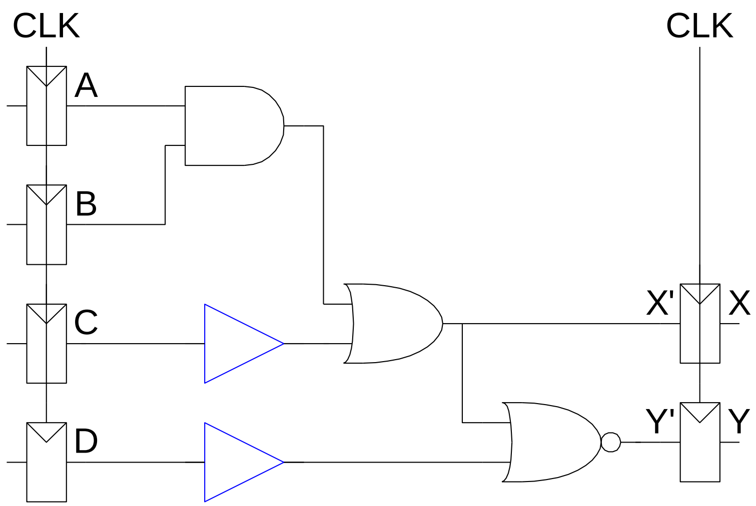
Hold time constraint:

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

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

Timing Analysis

Add buffers to the short paths:



t_{pd} =

t_{cd} =

Setup time constraint:

$T_c \geq$

f_c =

Timing Characteristics

t_{ccq} = 30 ps

t_{pcq} = 50 ps

t_{setup} = 60 ps

t_{hold} = 70 ps

per gate

t_{pd} = 35 ps

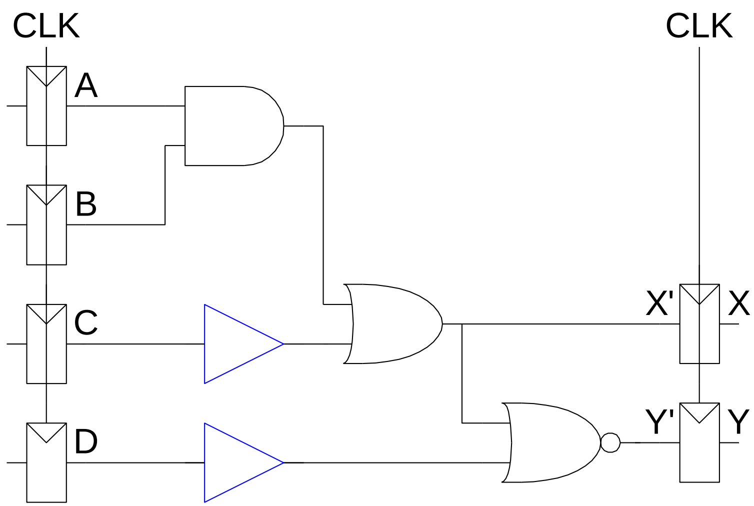
t_{cd} = 25 ps

Hold time constraint:

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

Timing Analysis

Add buffers to the short paths:



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

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

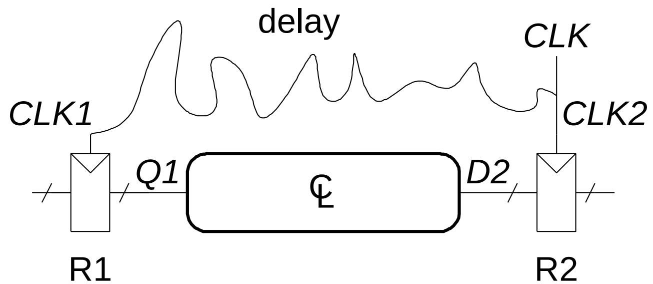
Hold time constraint:

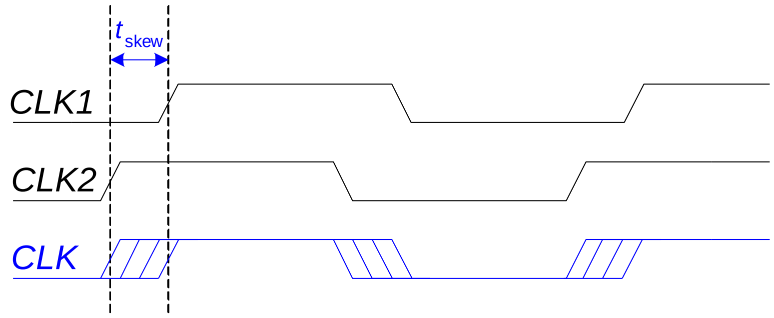
$t_{ccq} + t_{cd} > t_{hold}$?

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

Clock Skew

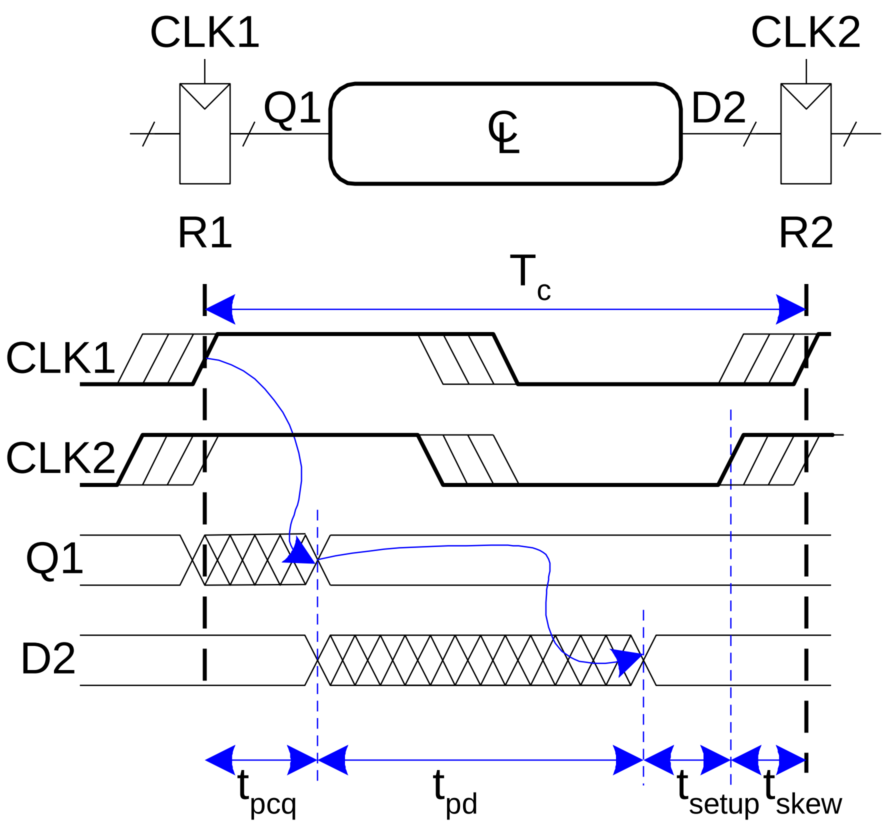
- The clock doesn't arrive at all registers at same time
- **Skew:** difference between two clock edges
- Perform **worst case analysis** to guarantee dynamic discipline is not violated for any register – many registers in a system!





Setup Time Constraint with Skew

- In the worst case, CLK2 is earlier than CLK1

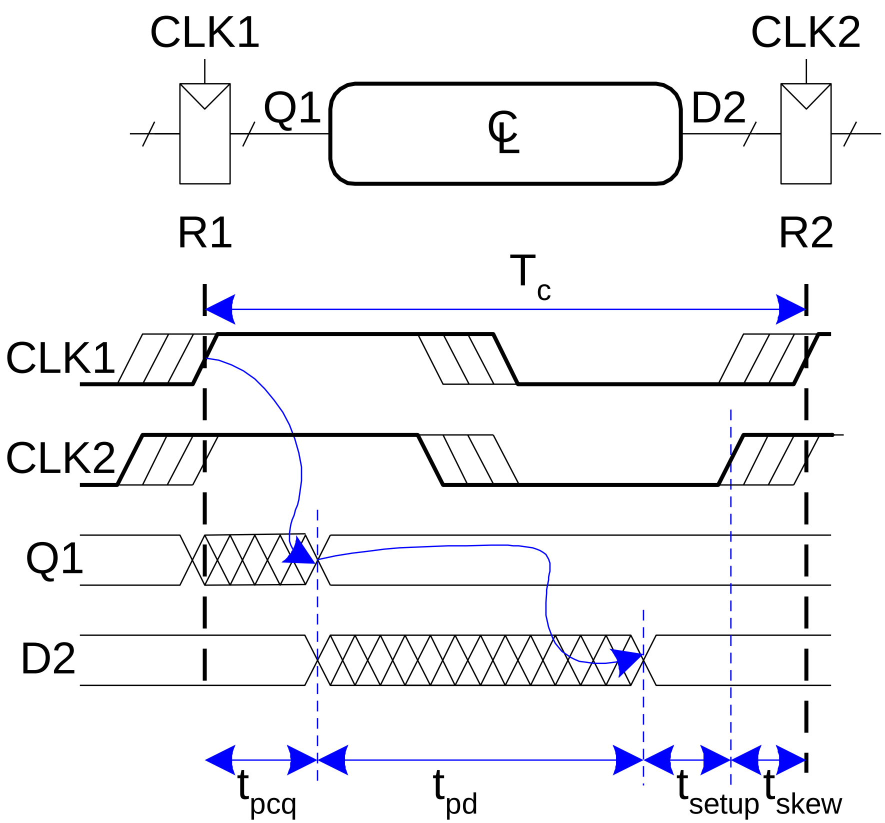


$T_c \geq$

ELSEVIER

Setup Time Constraint with Skew

- In the worst case, CLK2 is earlier than CLK1

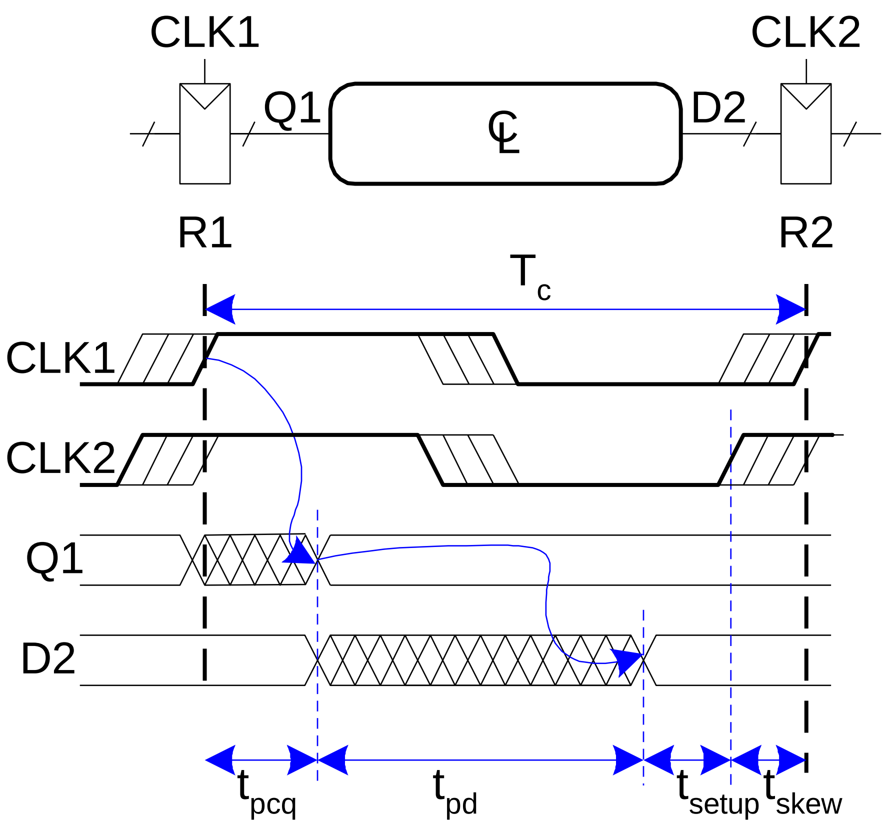


$T_c \geq t_{pcq} + t_{pd} + t_{setup} + t_{skew}$

$t_{pd} \leq$

Setup Time Constraint with Skew

- In the worst case, CLK2 is earlier than CLK1

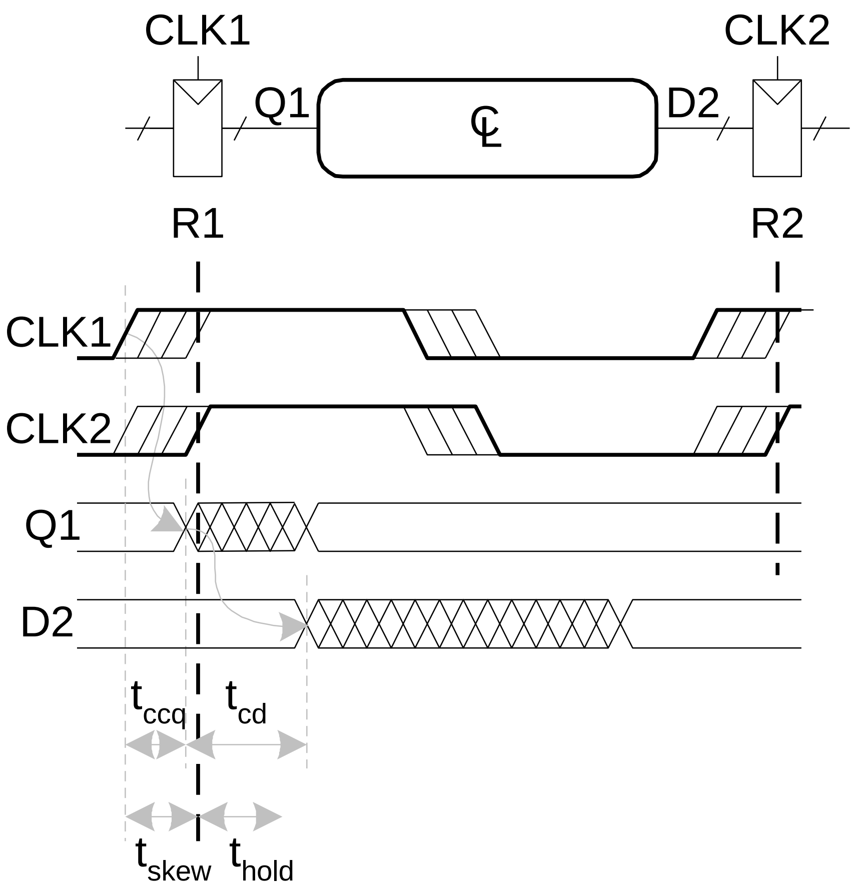


$$T_c \geq t_{pcq} + t_{pd} + t_{setup} + t_{skew}$$

$$t_{pd} \leq T_c - (t_{pcq} + t_{setup} + t_{skew})$$

Hold Time Constraint with Skew

- In the worst case, CLK2 is later than CLK1

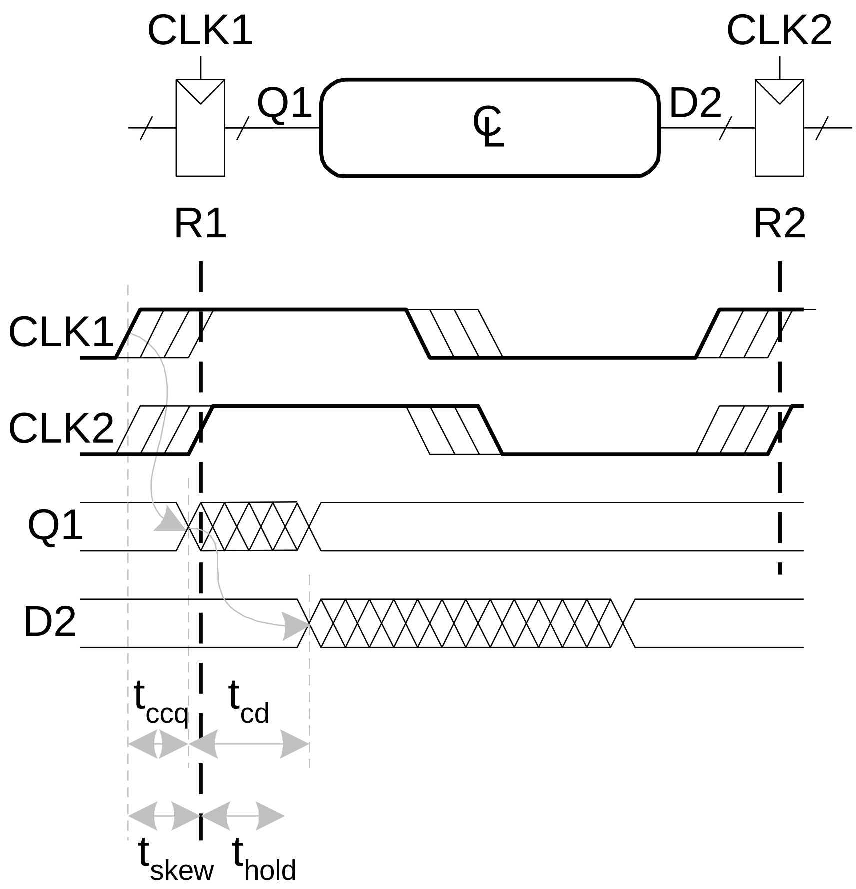


$t_{ccq} + t_{cd} >$

ELSEVIER

Hold Time Constraint with Skew

- In the worst case, CLK2 is later than CLK1



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

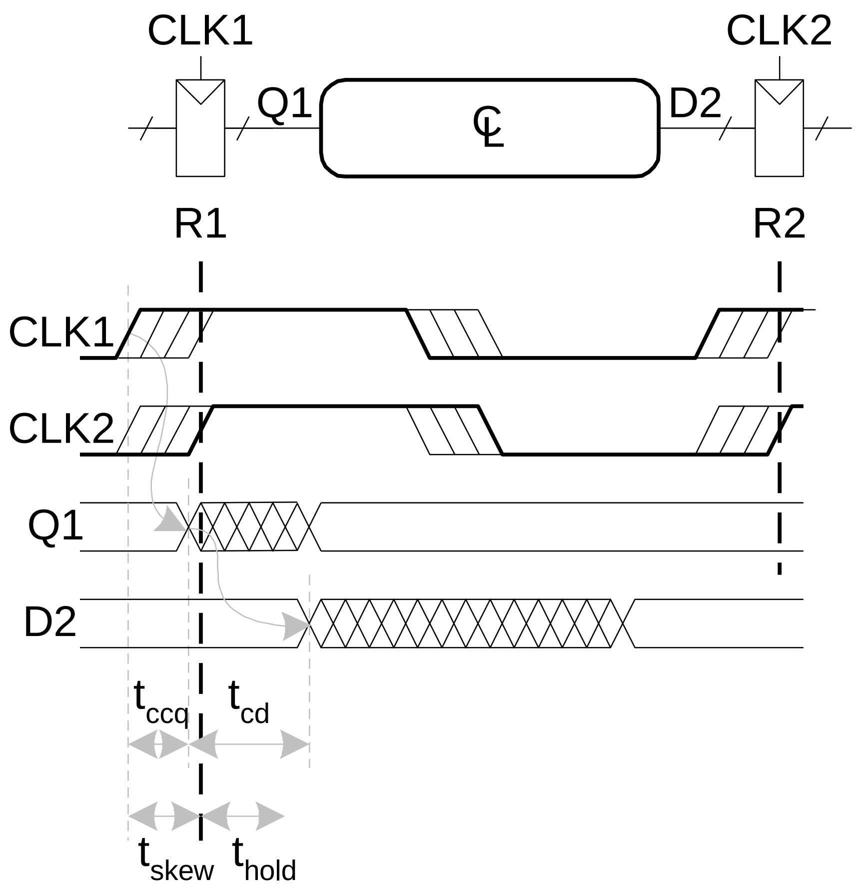
$t_{cd} >$

SEQUENTIAL LOGIC DESIGN

ELSEVIER

Hold Time Constraint with Skew

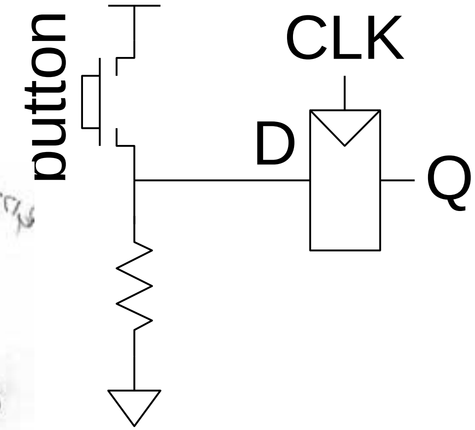
- In the worst case, CLK2 is later than CLK1

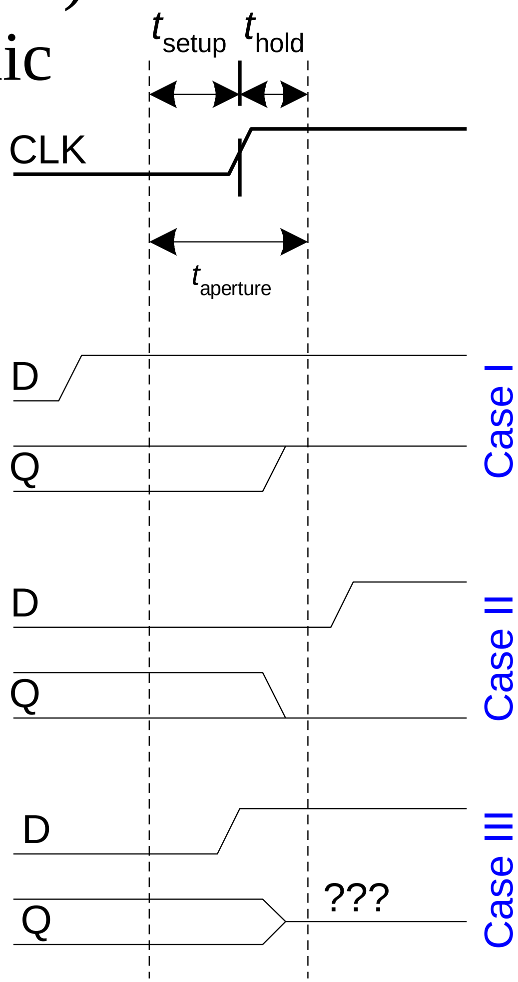


$$t_{ccq} + t_{cd} > t_{\mathrm{hold}} + t_{\mathrm{skew}}$$
$$t_{cd} > t_{\mathrm{hold}} + t_{\mathrm{skew}} - t_{ccq}$$

Violating the Dynamic Discipline

- Asynchronous (for example, user) inputs might violate the dynamic discipline





Metastability

- **Bistable devices:** two stable states, and a metastable state between them
- **Flip-flop:** two stable states (1 and 0) and one metastable state
- If flip-flop lands in metastable state, could stay there for an undetermined amount of time