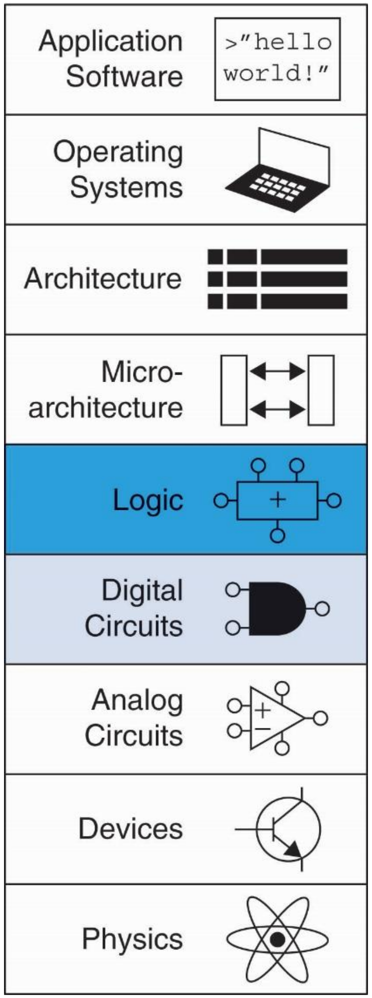
Digital Design & Computer Architecture

Sarah Harris & David Harris

Chapter 3: Sequential Logic Design

Chapter 3 :: Topics

- **State Elements**
  - **Bistable Circuit**
  - **SR Latch**
  - **D Latch**
  - **D Flip-Flop**
  - **Variations**
- **Synchronous Sequential Logic**
- **Finite State Machines**
  - **Moore**
  - **Mealy**
  - **Factored**
- **Timing of Sequential Logic**
  - **Clock Skew**
  - **Synchronization**
- **Parallelism**



State Elements

Introduction

- Outputs of sequential logic depend on current *and* prior input values – it has ***memory***.

Sequential Circuits

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

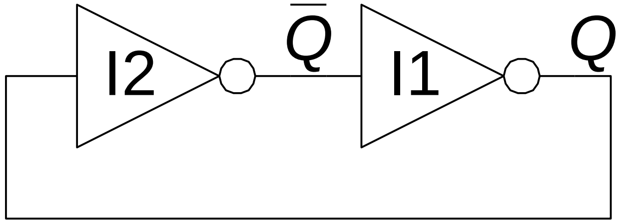
State Elements

- **State**: everything about the prior inputs to the circuit necessary to predict its future behavior
  - Usually just 1 bit, the last value captured
- State elements store state
  - Bistable circuit
  - SR Latch
  - D Latch
  - D Flip-flop

Bistable Circuit

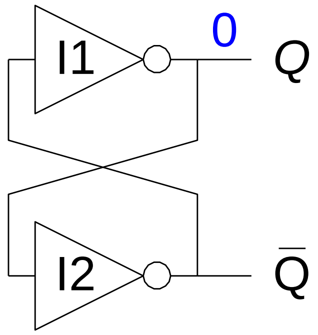
Bistable Circuit

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

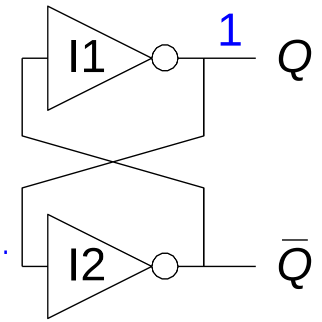


Bistable Circuit Analysis

- Consider the two possible cases:
  - ***Q* = 0:**



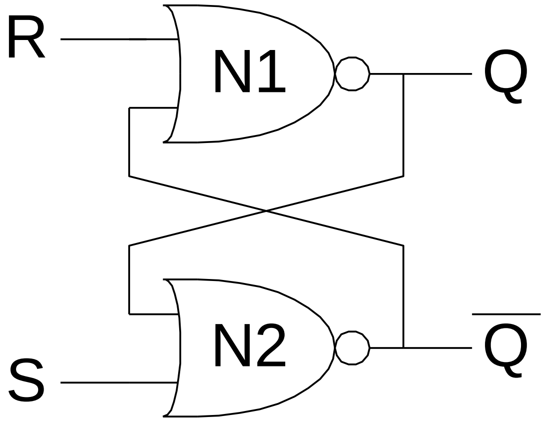
  - ***Q* = 1:**



SR Latch

SR (Set/Reset) Latch

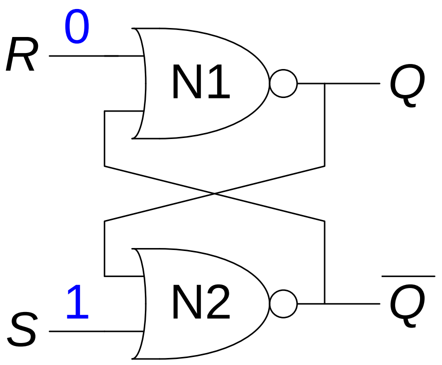
- **SR Latch**



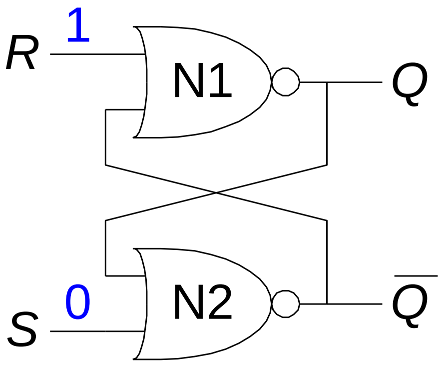
- Consider the four possible cases:

SR Latch Analysis

- **S = 1**, R = 0:



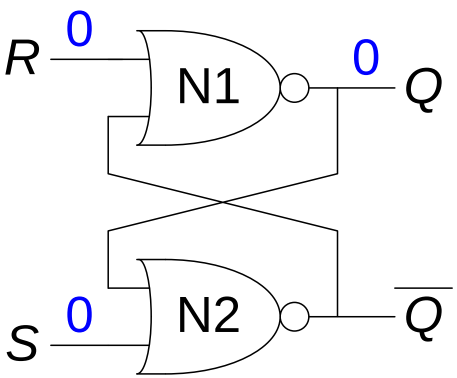
- S = 0, **R = 1:**



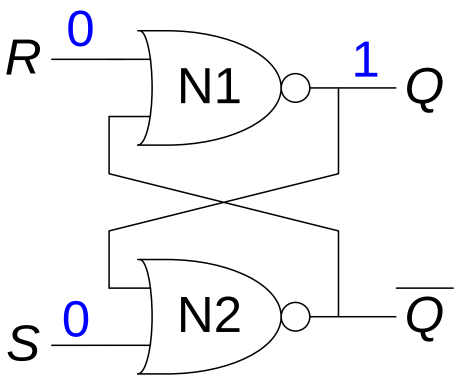
SR Latch Analysis

– ***S* = 0, *R* = 0:**

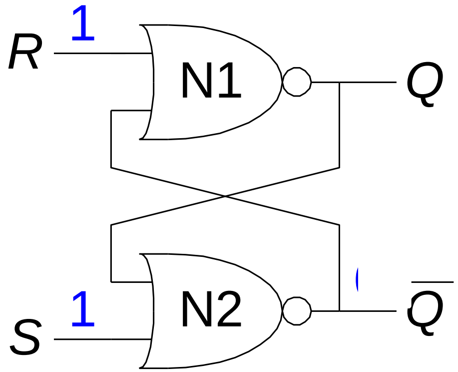
$Q_{prev} = 0$



$Q_{prev} = 1$



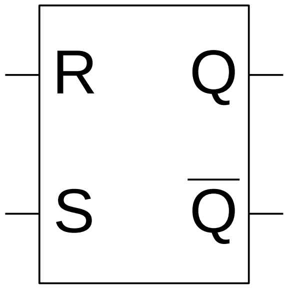
– ***S* = 1, *R* = 1:**



SR Latch

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

SR Latch Symbol



- **Must do something to avoid invalid state (when $\boldsymbol{S = R = 1}$)**

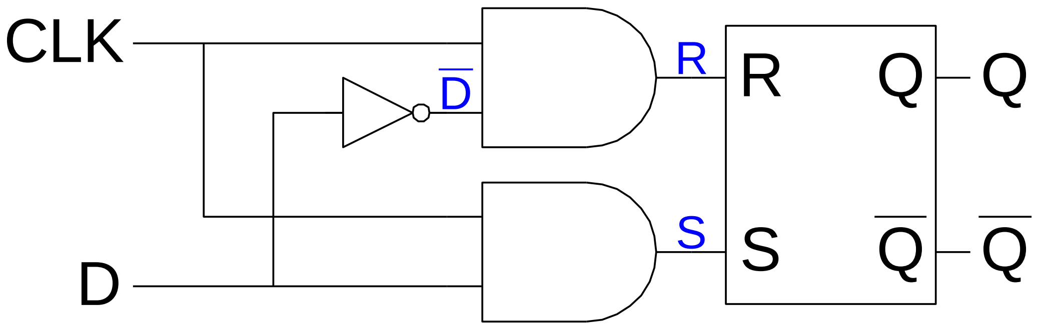
Chapter 3: Sequential Logic

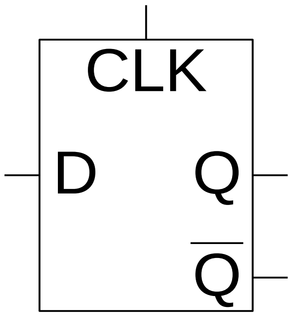
D Latch

D Latch

- **Two inputs:** *CLK*, *D*
  - ***CLK*:** controls *when* the output changes
  - ***D*** (the data input): controls *what* the output changes to

D Latch Internal Circuit





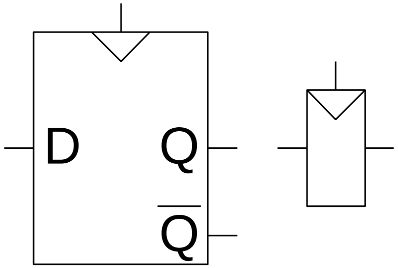
CLK	*D*	$\overline{D}$	*S*	*R*	*Q*	$\overline{Q}$
0	X					
1	0					
1	1					

D Flip-Flop

D Flip-Flop

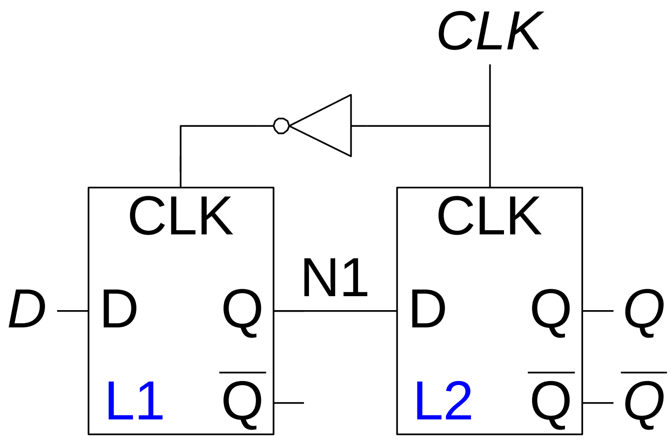
- **Inputs:** *CLK*, *D*

D Flip-Flop
Symbols



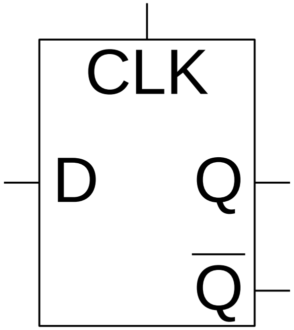
D Flip-Flop Internal Circuit

- **Two back-to-back D latches** (L1 and L2) controlled by complementary clocks
- When ***CLK* = 0**

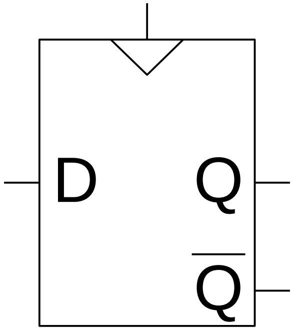


- When ***CLK* = 1**

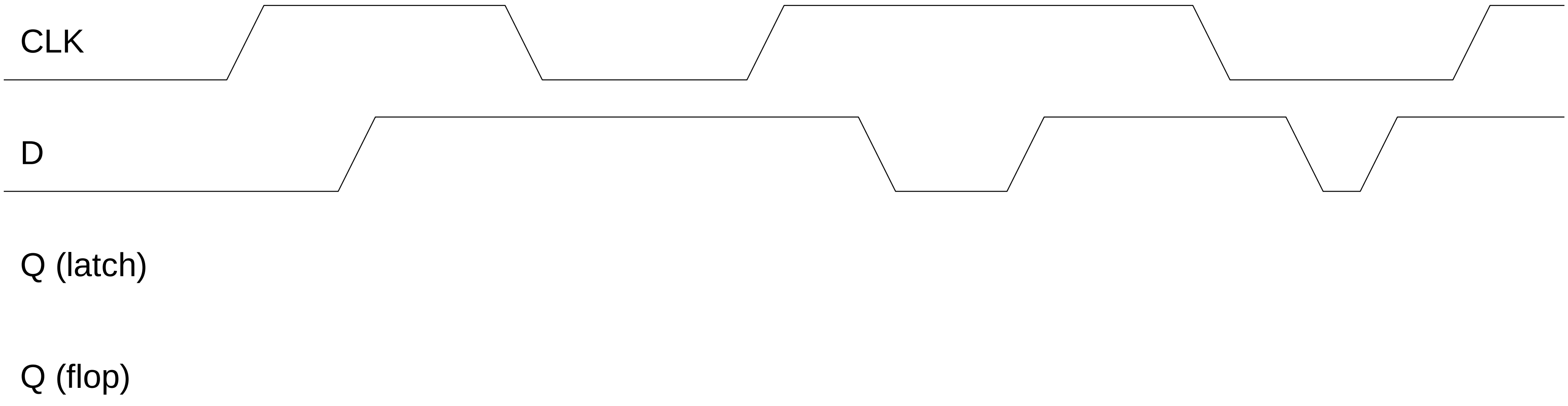
D Latch vs. D Flip-Flop



D Latch

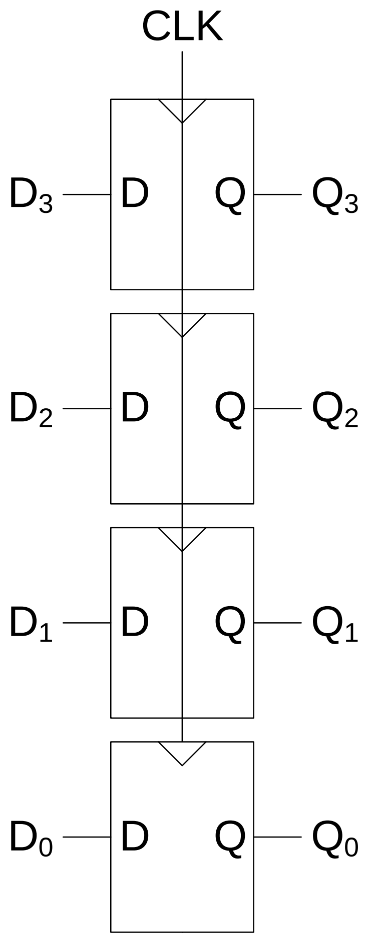


D Flip-flop

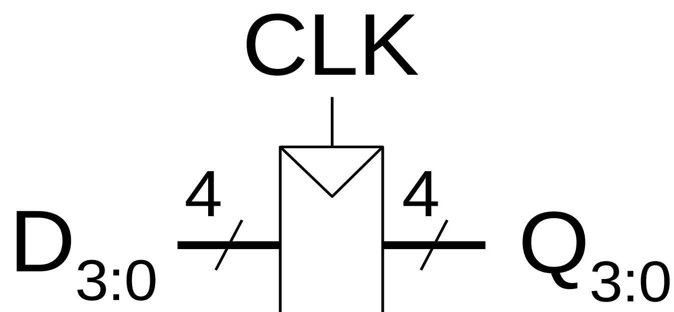


Variations on a Flop

Registers: One or More Flip-flops



4-bit Register

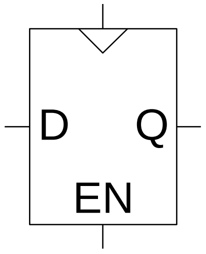


4-bit Register

Enabled Flip-Flops

- **Inputs:** *CLK*, *D*, *EN*
  - The enable input (*EN*) controls when new data (*D*) is stored
- **Function**
  - ***EN* = 1:**
  - ***EN* = 0:**

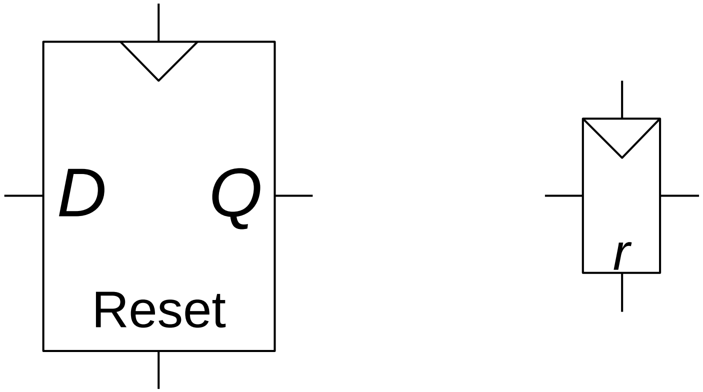
Symbol



Resettable Flip-Flops

- **Inputs:** *CLK*, *D*, *Reset*
- **Function:**
  - ***Reset* = 1:**
  - ***Reset* = 0:**

Symbols



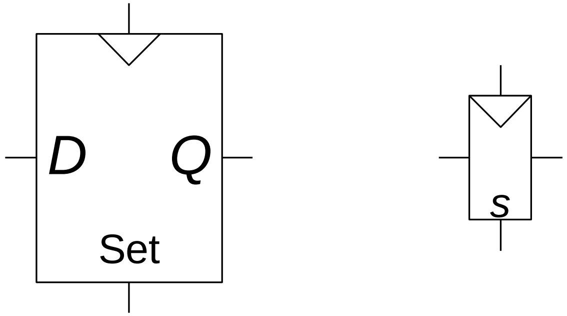
Resettable Flip-Flops

- Two types:
  - **Synchronous:**
  - **Asynchronous:**

Settable Flip-Flops

- **Inputs:** *CLK*, *D*, *Set*
- **Function:**
  - ***Set* = 1:**
  - ***Set* = 0:**

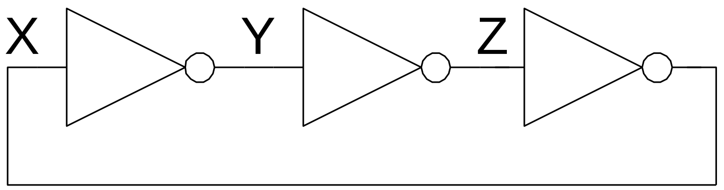
Symbols

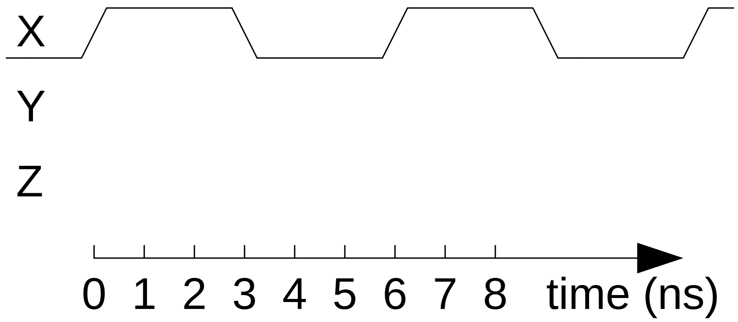


Synchronous Sequential Logic

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

Synchronous Sequential Logic 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**
  - Every **cyclic path** contains at least **one register**
- Two common synchronous sequential circuits
  - **Finite State Machines (FSMs)**
  - **Pipelines**

FSMs:

Finite State Machines

Finite State Machine (FSM)

- **Consists of:**
  - **State register**
  - **Combinational logic**

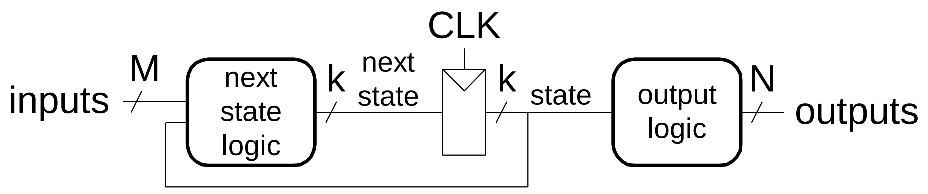
Finite State Machines (FSMs)

- **Next state** determined by **current state** and **inputs**

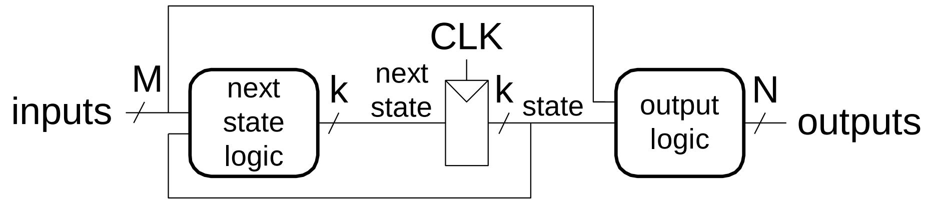
Finite State Machines (FSMs)

- **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 Design Procedure

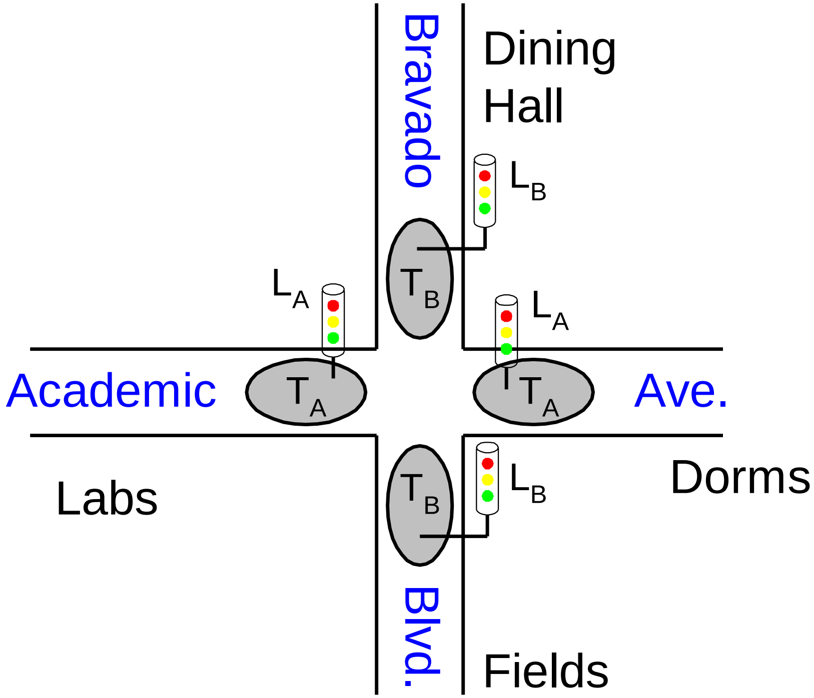
FSM Design Procedure

1. Identify **inputs** and **outputs**
2. Sketch **state transition diagram**
3. Write **state transition table** and **output table**
   - **Moore FSM:** write **separate** tables
   - **Mealy FSM:** write **combined** state transition and output table
4. Select **state encodings**
5. Rewrite state transition table and output table with state **encodings**
6. Write **Boolean equations** for next state and output logic
7. Sketch the circuit **schematic**

Moore FSM Example

FSM Example

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

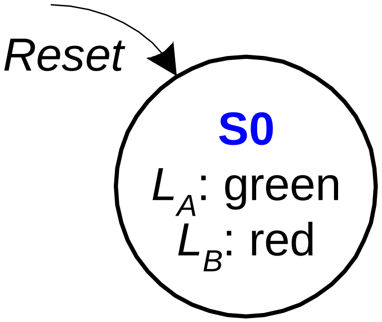


FSM Black Box

- **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

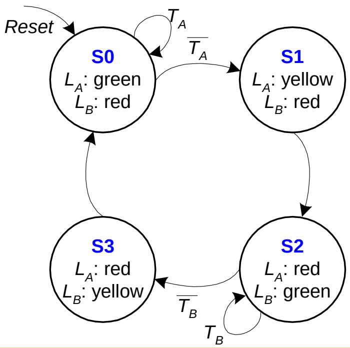


FSM State Transition Table

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	

S **:** Current State

S': Next State



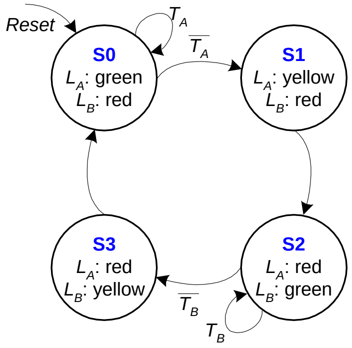
FSM Encoded State Transition Table

Current State		Inputs		Next State	
S_1	S_0	T_A	T_B	S'_1	S'_0
S0		0	X	S1	
S0		1	X	S0	
S1		X	X	S2	
S2		X	0	S3	
S2		X	1	S2	
S3		X	X	S0	

State	Encoding
S0	00
S1	01
S2	10
S3	11

FSM Output Table

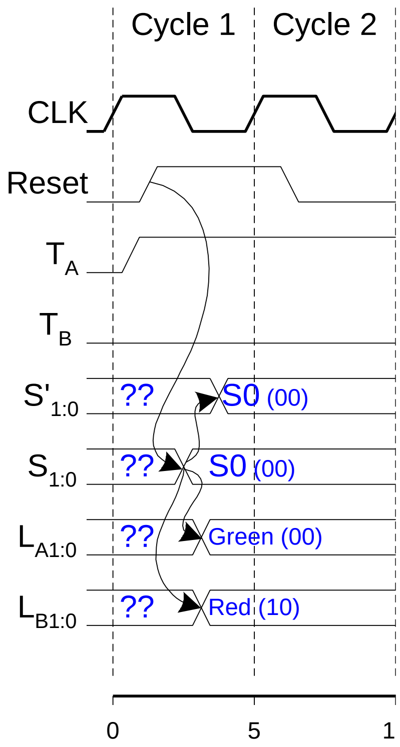
Current State		Outputs			
S_1	S_0	L_{A1}	L_{A0}	L_{B1}	L_{B0}
S0		green		red	
S1		yellow		red	
S2		red		green	
S3		red		yellow	

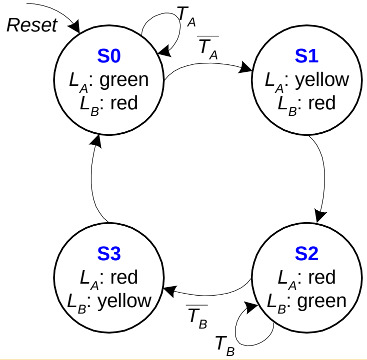


Output	Encoding
green	00
yellow	01
red	10

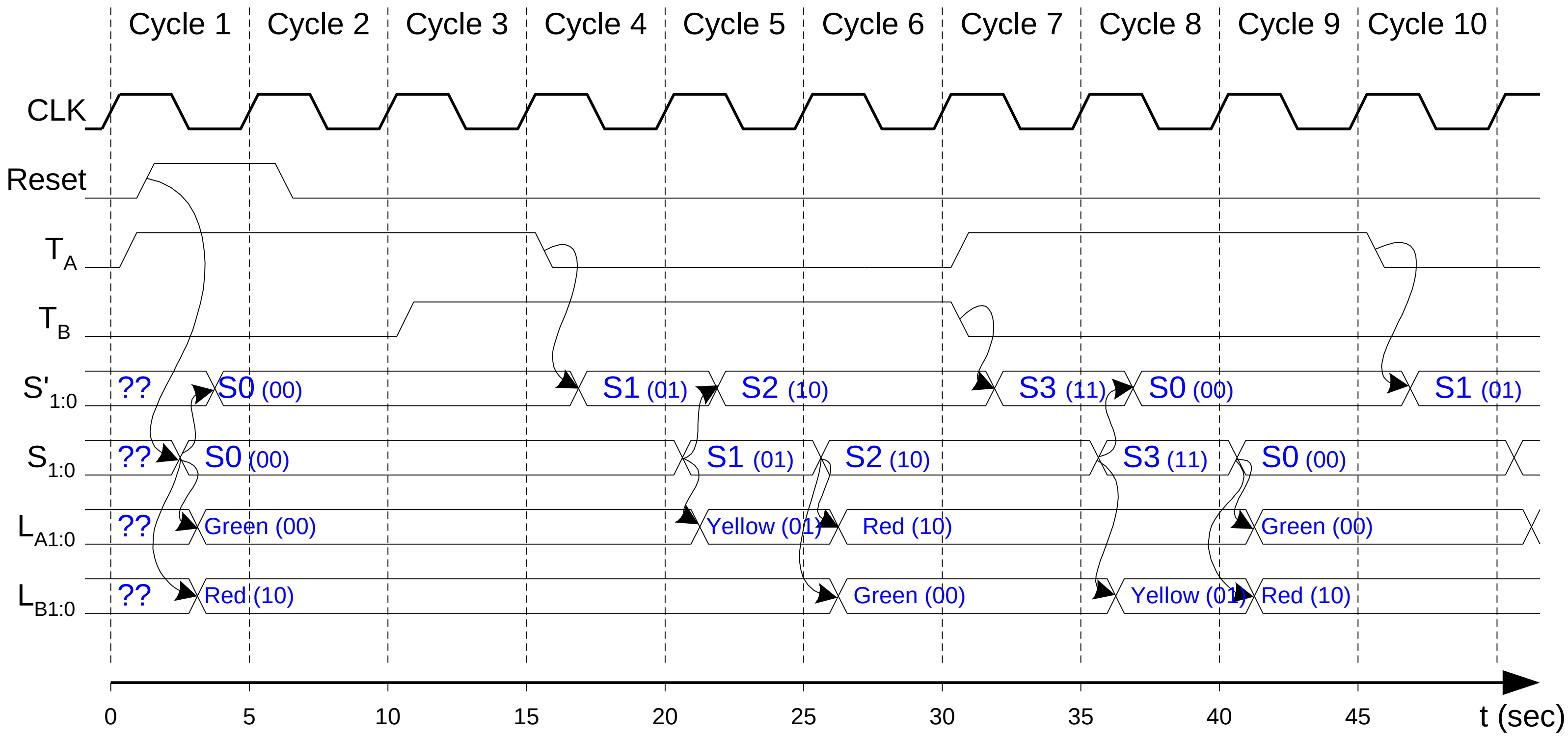
FSM Schematic

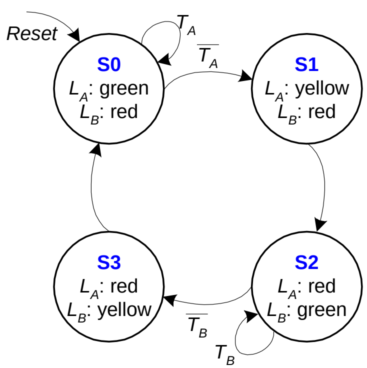
FSM Timing Diagram





FSM Timing Diagram





State Encodings

- **Binary** encoding:

- **One-hot** encoding

1-Hot State Encoding Example

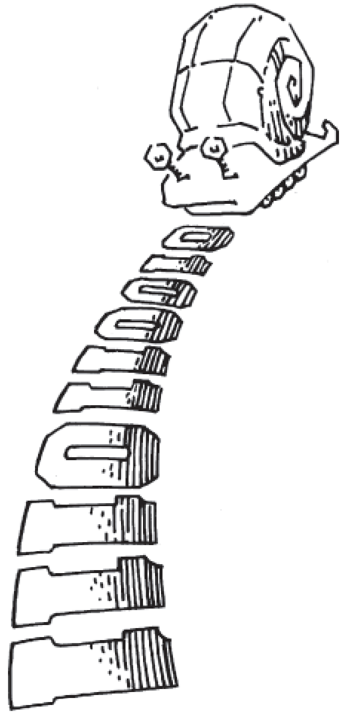
Current State				Inputs		Next State			
S_3	S_2	S_1	S_0	T_A	T_B	S'_3	S'_2	S'_1	S'_0
S0				0	X	S1			
S0				1	X	S0			
S1				X	X	S2			
S2				X	0	S3			
S2				X	1	S2			
S3				X	X	S0			

State	1-Hot Encoding
S0	0001
S1	0010
S2	0100
S3	1000

Mealy FSM Example

Moore vs. Mealy FSMs

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

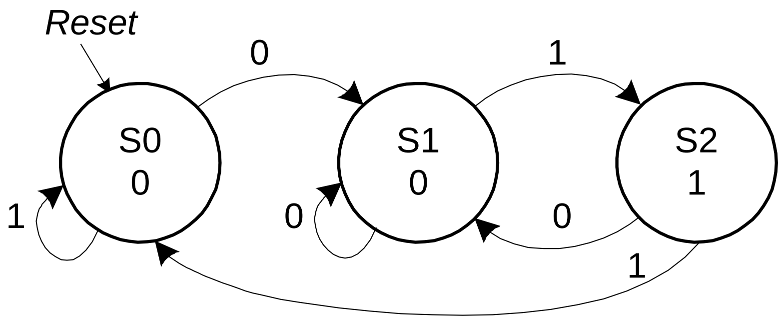
Moore FSM

Mealy FSM

Moore FSM State Transition Table

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

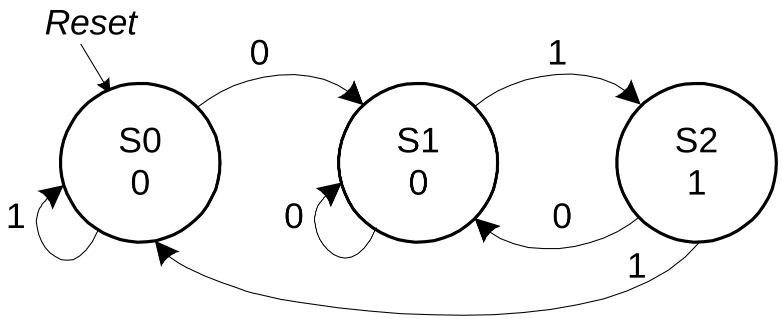
State	Encoding
S0	00
S1	01
S2	10



Moore FSM Output Table

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

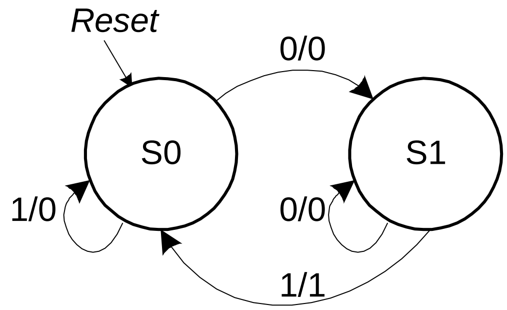
State	Encoding
S0	00
S1	01
S2	10



Mealy 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	0
S1	1



Moore FSM Schematic

Next State Equations

$S_1' = S_0A$

$S_0' = \overline{A}$

Output Equation

$Y = S_1$

Mealy FSM Schematic

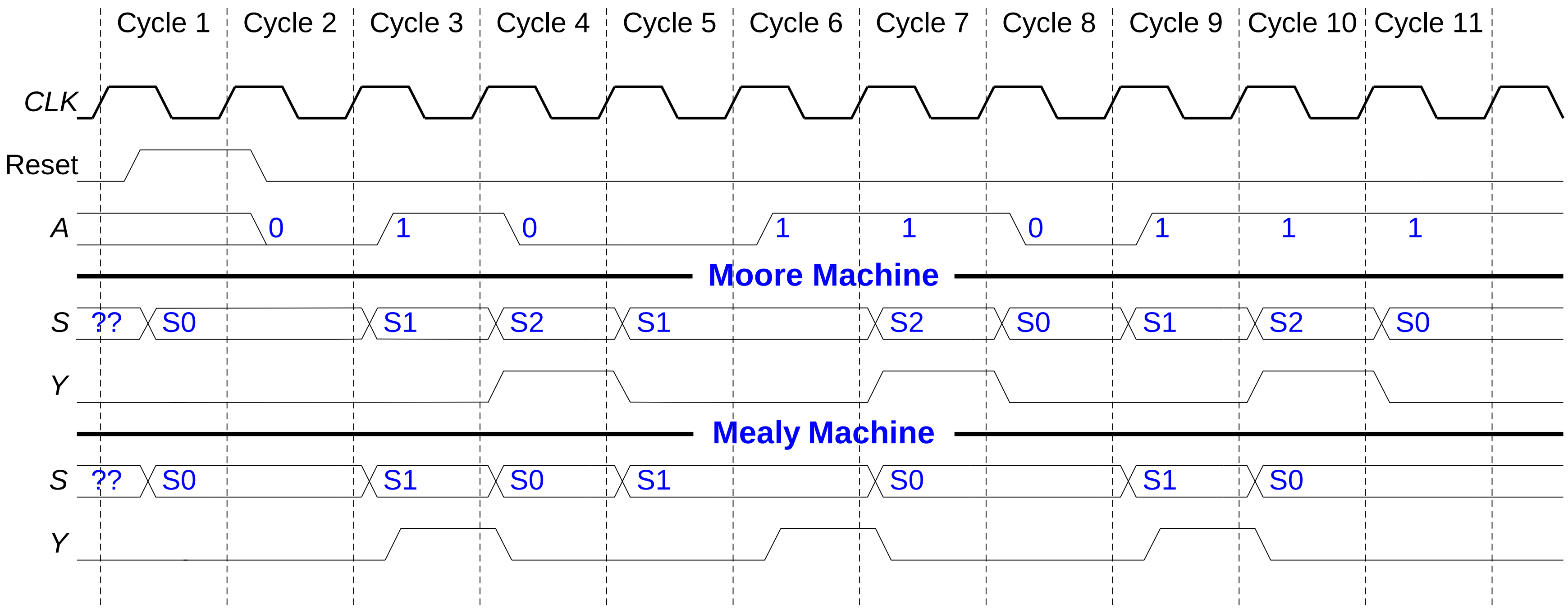
Next State Equation

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

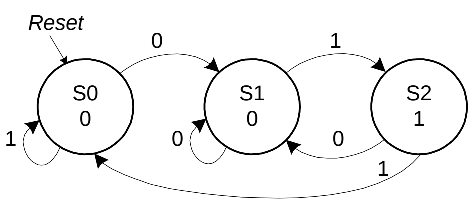
Output Equation

$$Y = S_0 A$$

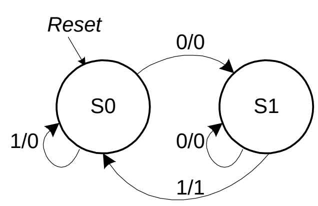
Moore and Mealy Timing Diagram



Moore FSM



Mealy FSM



Mealy FSM: asserts Y **immediately** when input pattern 01 is detected

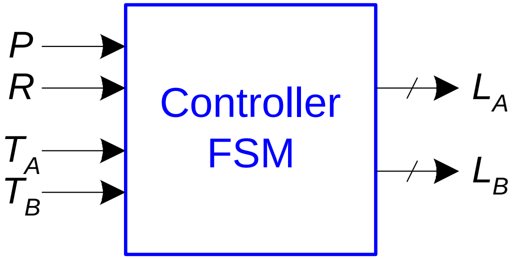
Moore FSM: asserts Y one cycle **after** input pattern 01 is detected

Factored FSMs

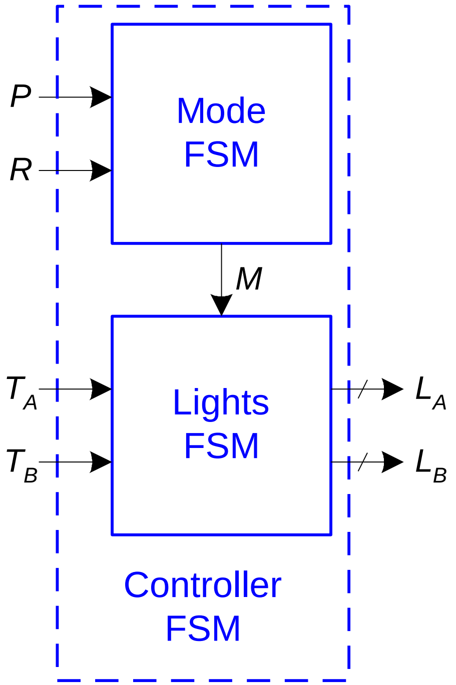
Factoring FSMs

- Break **complex FSMs** into **smaller interacting FSMs**
- **Example:** Modify traffic light controller to have Parade Mode.
  - Two more inputs: ***P***, ***R***
  - When ***P* = 1**, enter Parade Mode & Bravado Blvd light stays green
  - When ***R* = 1**, leave Parade Mode

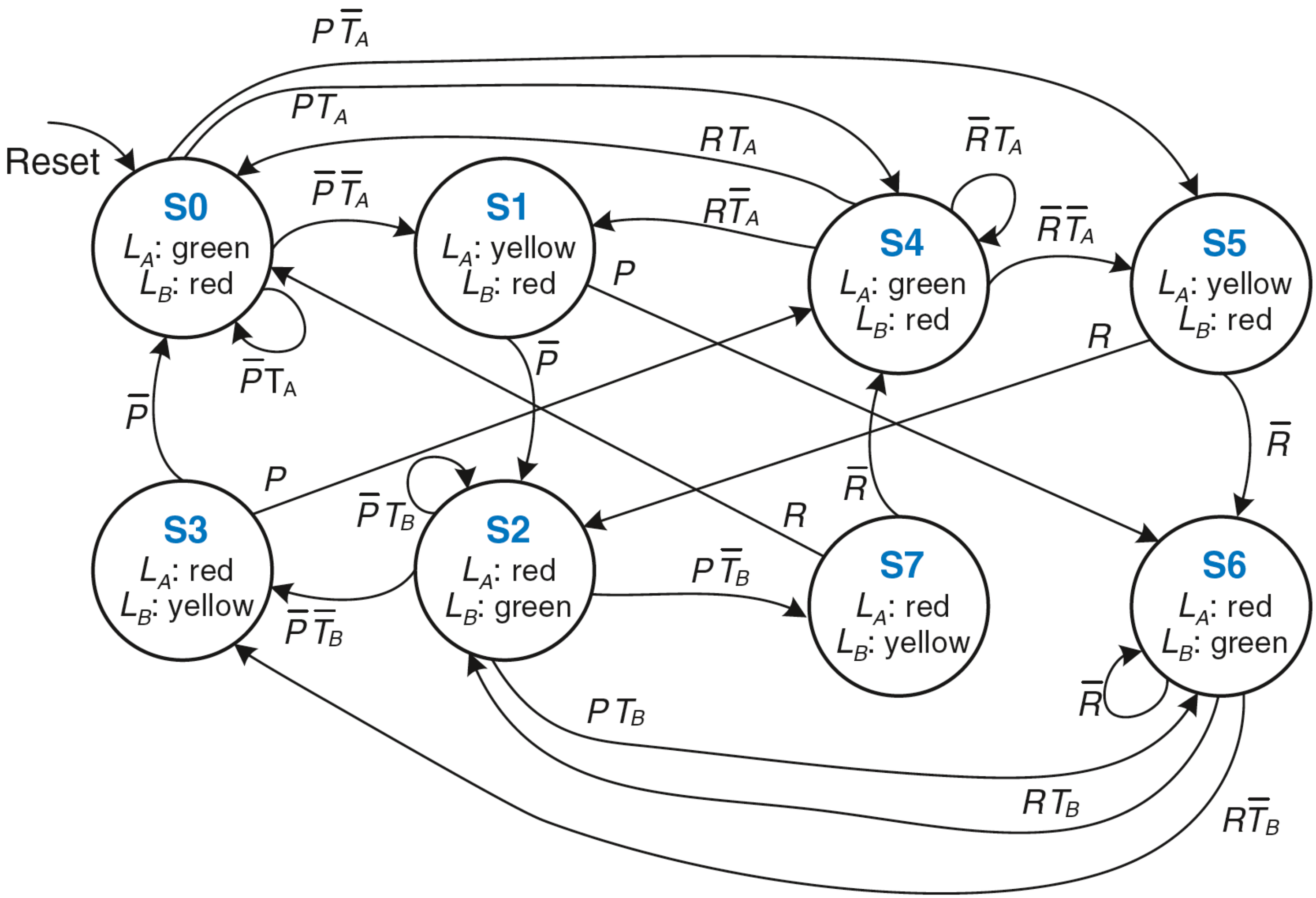
Unfactored FSM



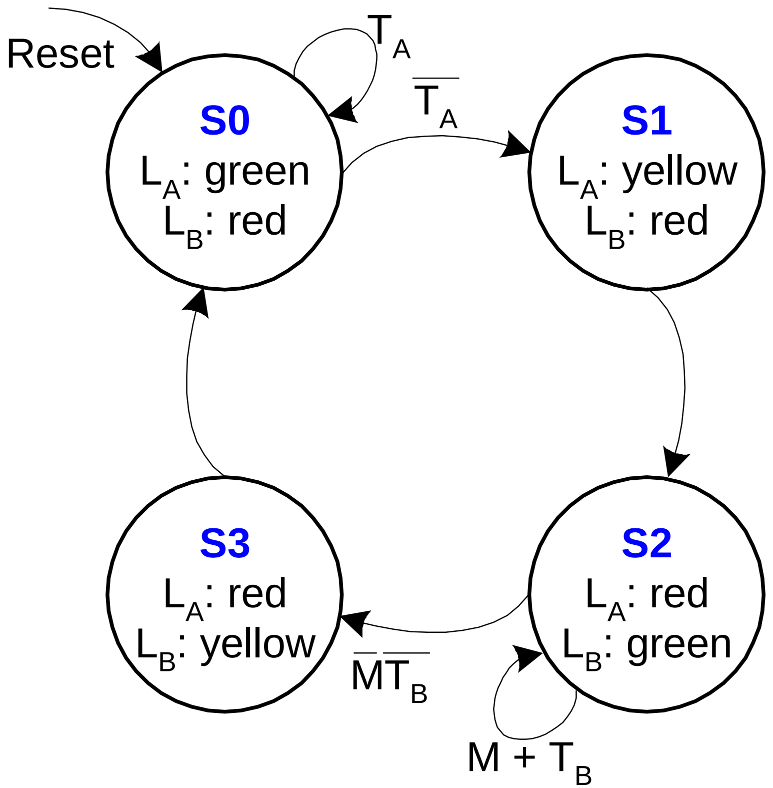
Factored FSM



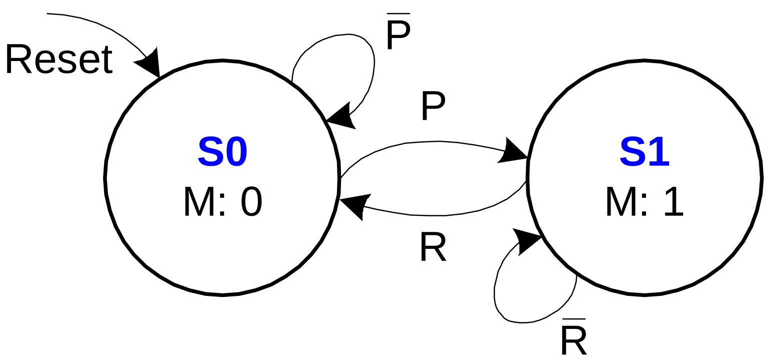
Unfactored FSM



Factored FSM



Lights FSM



Mode FSM

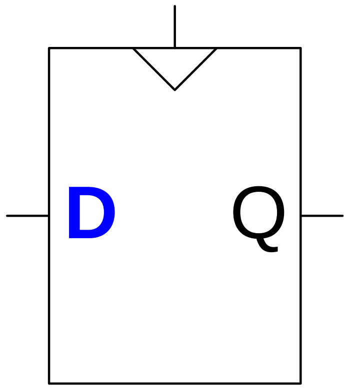
Timing

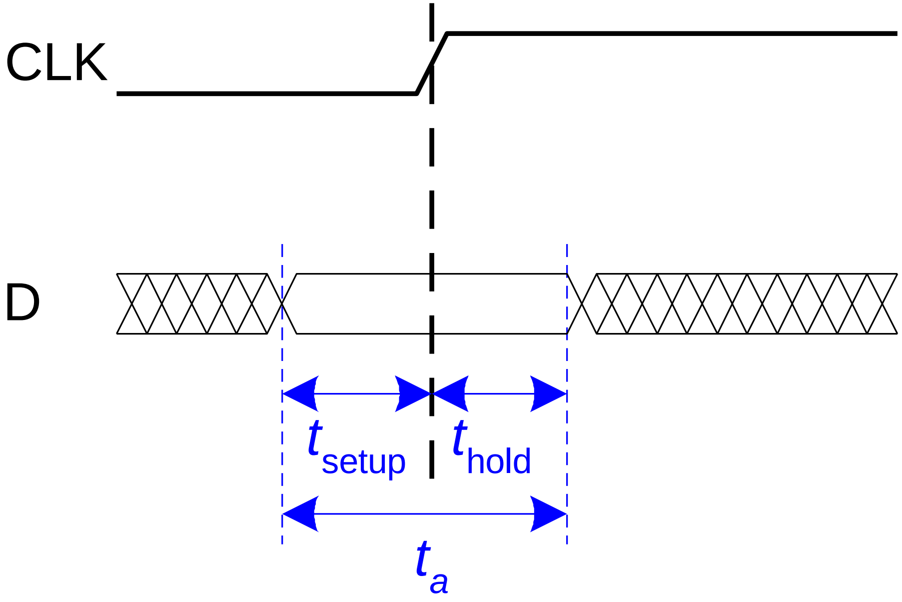
Timing

- Flip-flop samples *D* at clock edge
- ***D* must be stable when sampled**
- Similar to a photograph, *D* must be stable around clock edge

Input Timing Constraints

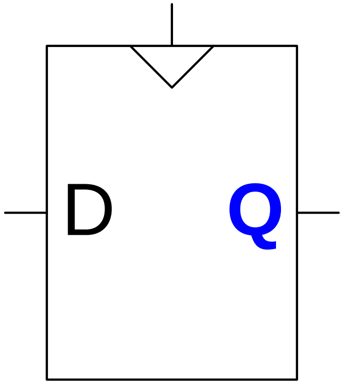
- **Setup time:** t_{setup} = time *before* clock edge data must be stable (i.e. not changing)
- **Hold time:** t_{hold} = time *after* clock edge data must be stable
- **Aperture time:** t_a = time *around* clock edge data must be stable ($t_a = t_{setup} + t_{hold}$)





Output Timing Constraints

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

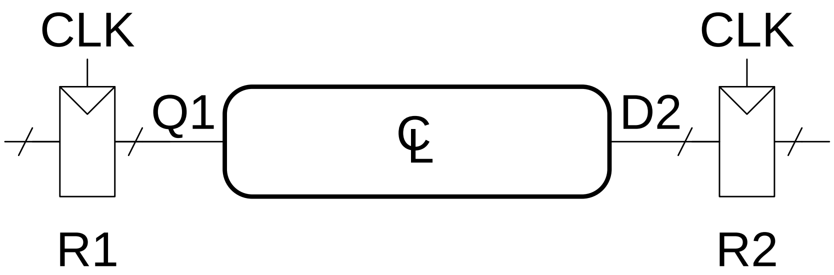


Dynamic Discipline

- 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

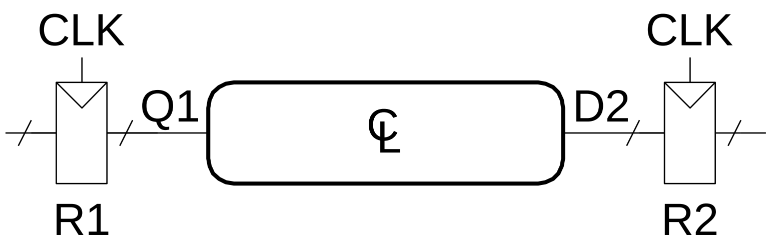
Dynamic Discipline

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



Setup Time Constraint

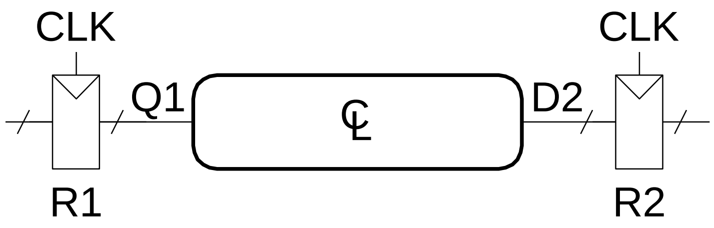
- Depends on the **maximum** delay from register R1 through combinational logic to R2
- The input to register R2 must be stable at least t_{setup} before clock edge



Also called:
Cycle Time Constraint

Hold Time Constraint

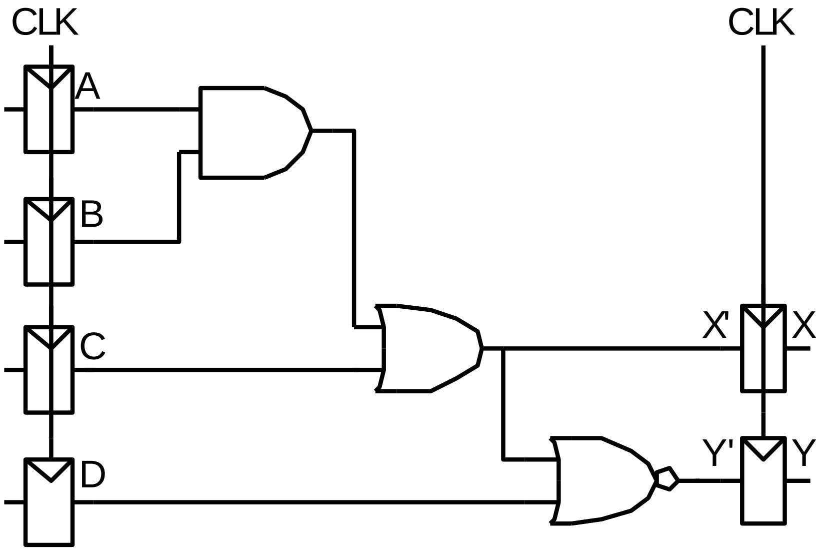
- Depends on the **minimum** delay from register R1 through the combinational logic to R2
- The input to register R2 must be stable for at least t_{hold} after the clock edge



Timing Analysis

- Calculate **both constraints**:
  - **Setup time** constraint (aka cycle time constraint)
  - **Hold time** constraint
- If the hold time constraint isn't met, the circuit **won't work reliably at any frequency**

Timing Analysis Example



Timing Characteristics

Flip-Flops:

t_{ccq} = 30 ps

t_{pcq} = 50 ps

t_{setup} = 60 ps

t_{hold} = 70 ps

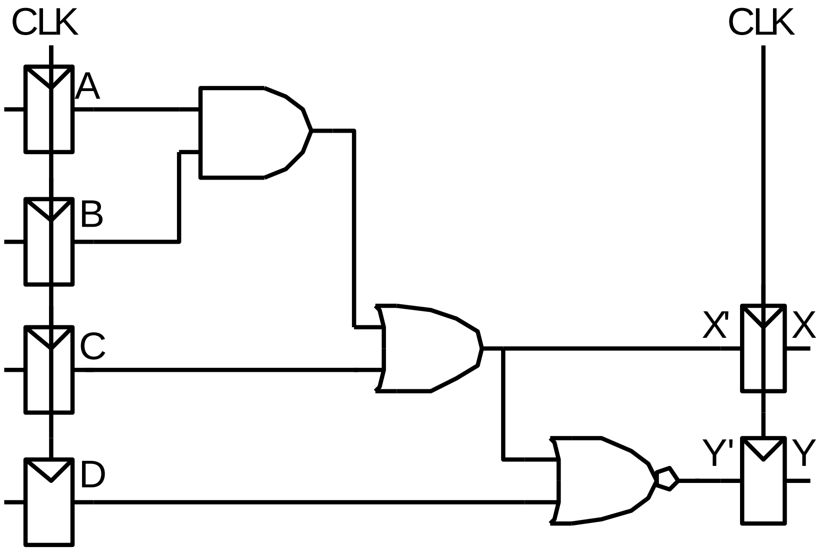
Logic delays: per gate

t_{pd} = 35 ps

t_{cd} = 25 ps

Timing Analysis Example

How to fix?



Timing Characteristics

Flip-Flops

t_{ccq} = 30 ps

t_{pcq} = 50 ps

t_{setup} = 60 ps

t_{hold} = 70 ps

Logic delays: per gate

t_{pd} = 35 ps

t_{cd} = 25 ps

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

t_{cd} = 25 ps

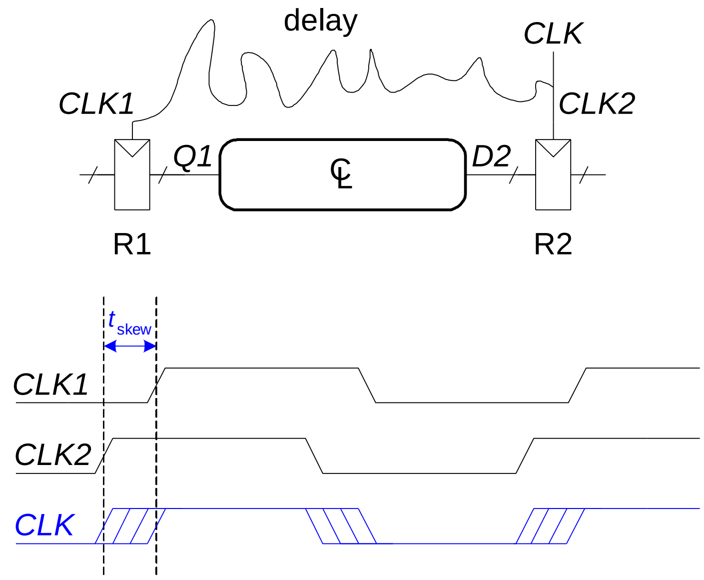
Setup time constraint:

Hold time constraint:

Clock Skew

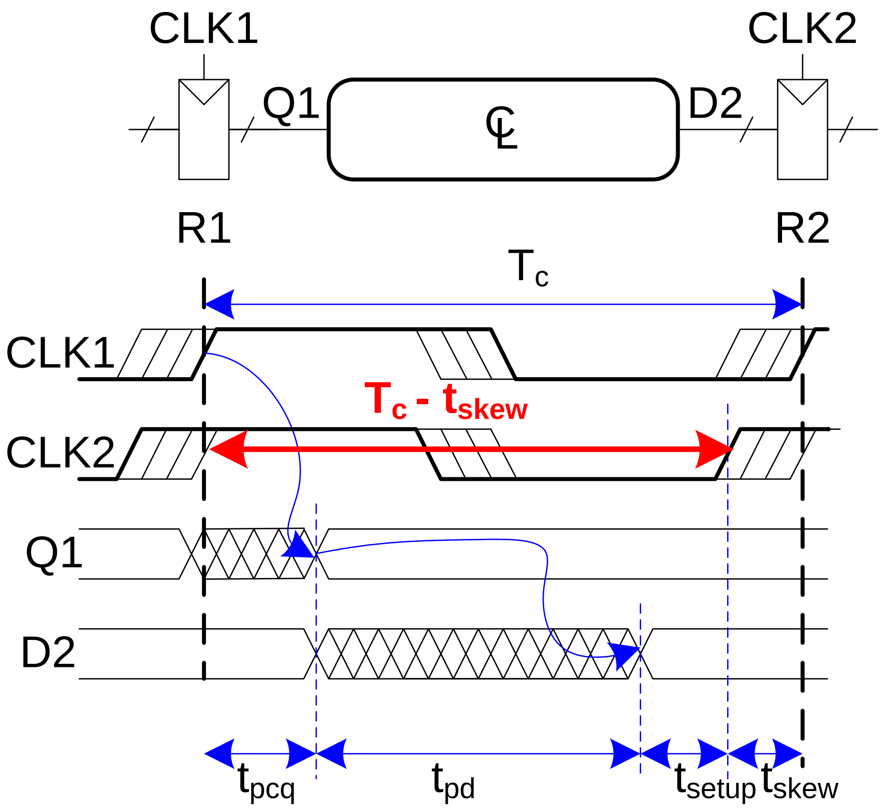
Clock Skew

- 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!



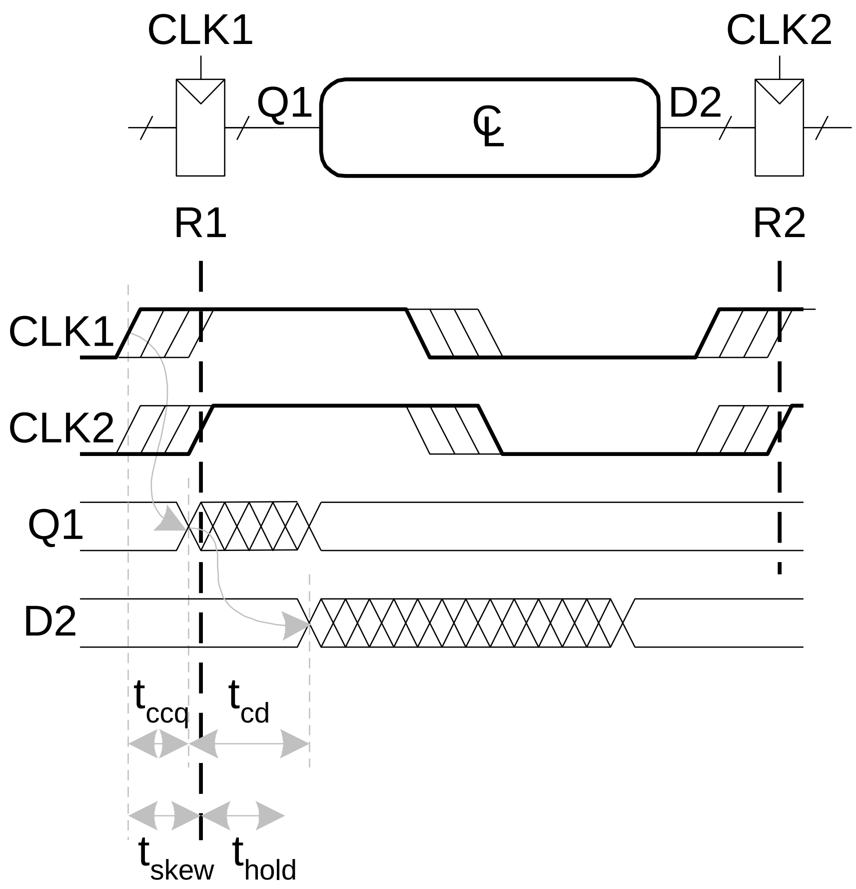
Setup Time Constraint with Skew

- In the worst case, CLK2 is earlier than CLK1



Hold Time Constraint with Skew

- In the worst case, CLK2 is later than CLK1

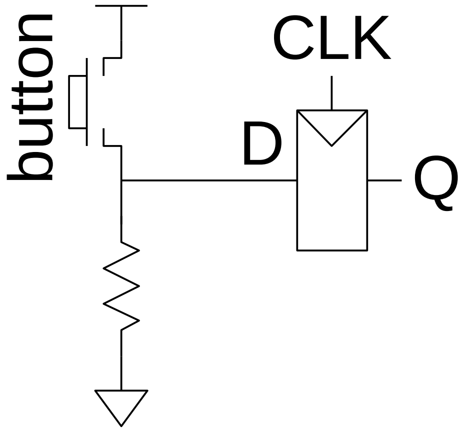


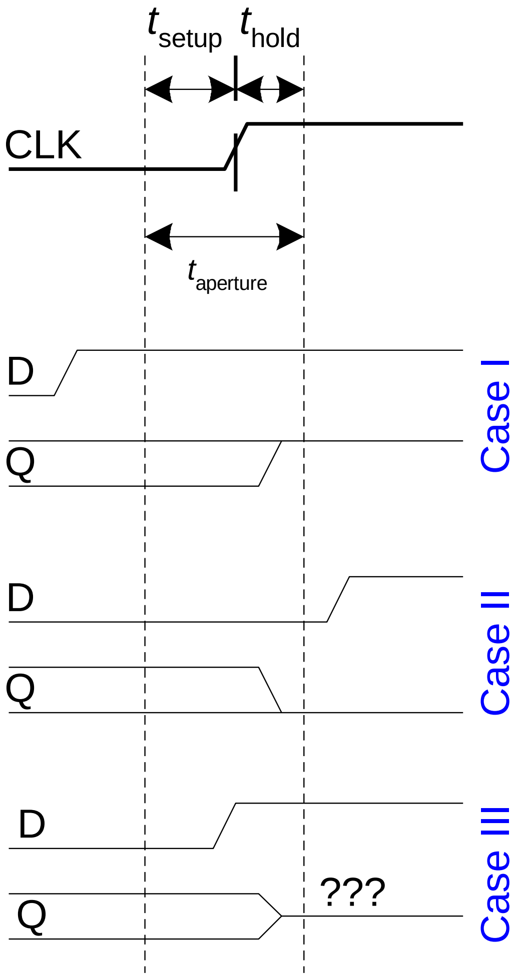
Chapter 3: Sequential Logic

Synchronization

Violating the Dynamic Discipline

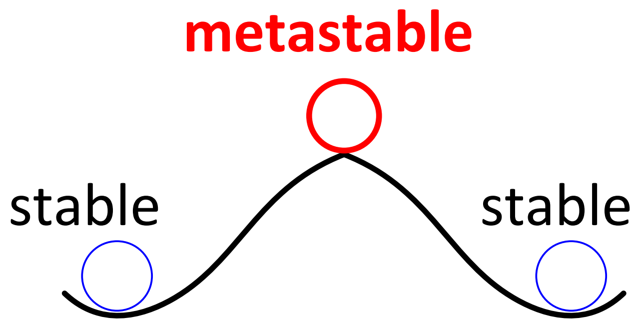
Asynchronous (for example, user) **inputs** might violate the dynamic discipline





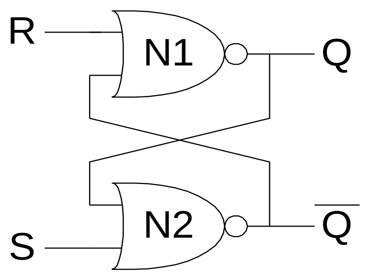
Metastability

- **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



Flip-Flop Internals

- Flip-flop has **feedback**: if *Q* is somewhere between 1 and 0, cross-coupled gates drive output to either rail (1 or 0)



- **Metastable signal:** if it hasn't resolved to 1 or 0
- If flip-flop input changes at random time, **probability that output *Q* is metastable** after waiting some time, *t*:

$$P(t_{res} > t) = (T_0/T_c)\, e^{-t/\tau}$$

t_{res} : time to resolve to 1 or 0

T_0, τ : properties of the circuit

Metastability

- **Intuitively:**

  T_0/T_c: probability input changes at a bad time (during aperture time)

$$P(t_{res} > t) = (T_0/T_c)\, e^{-t/\tau}$$

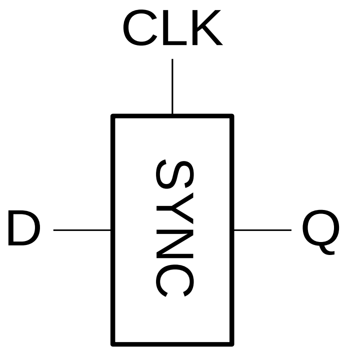
  τ: time constant for how fast flip-flop moves away from metastability

$$P(t_{res} > t) = (T_0/T_c)\, e^{-t/\tau}$$

- If flip-flop samples metastable input, if you wait long enough (t), the output will have resolved to 1 or 0 with high probability.

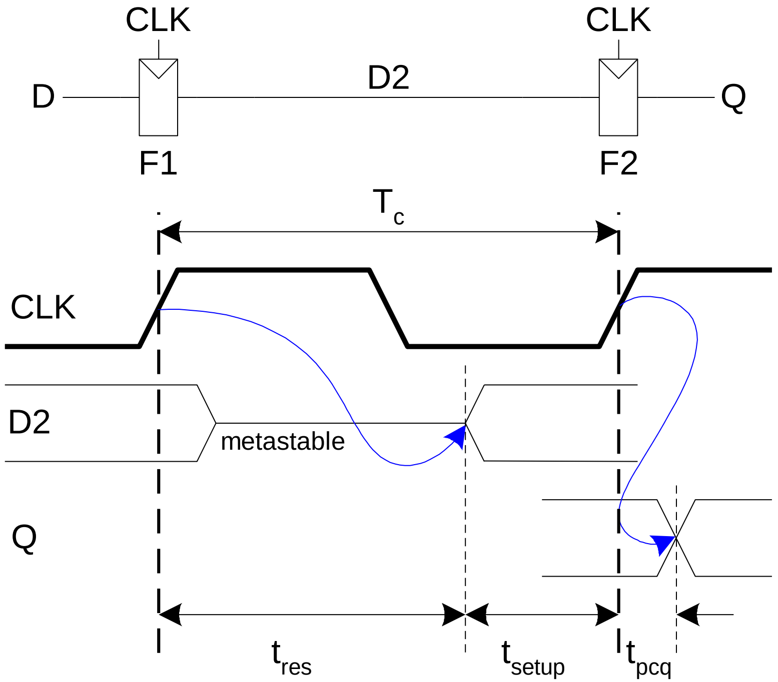
Synchronizers

- **Asynchronous inputs are inevitable** (user interfaces, systems with different clocks interacting, etc.)
- **Synchronizer goal:** make the **probability of failure** (the output *Q* still being metastable) **low**
- Synchronizer cannot make the probability of failure 0



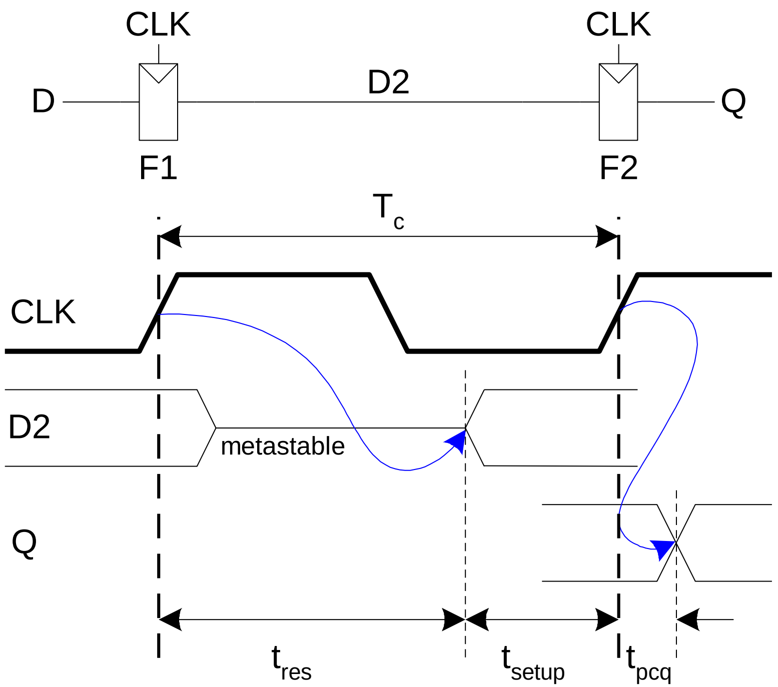
Synchronizer Internals

- **Synchronizer:** built with two back-to-back flip-flops
- Suppose D is transitioning when sampled by F1
- Internal signal D2 has (T_c - t_{setup}) time to resolve to 1 or 0



Synchronizer Internals

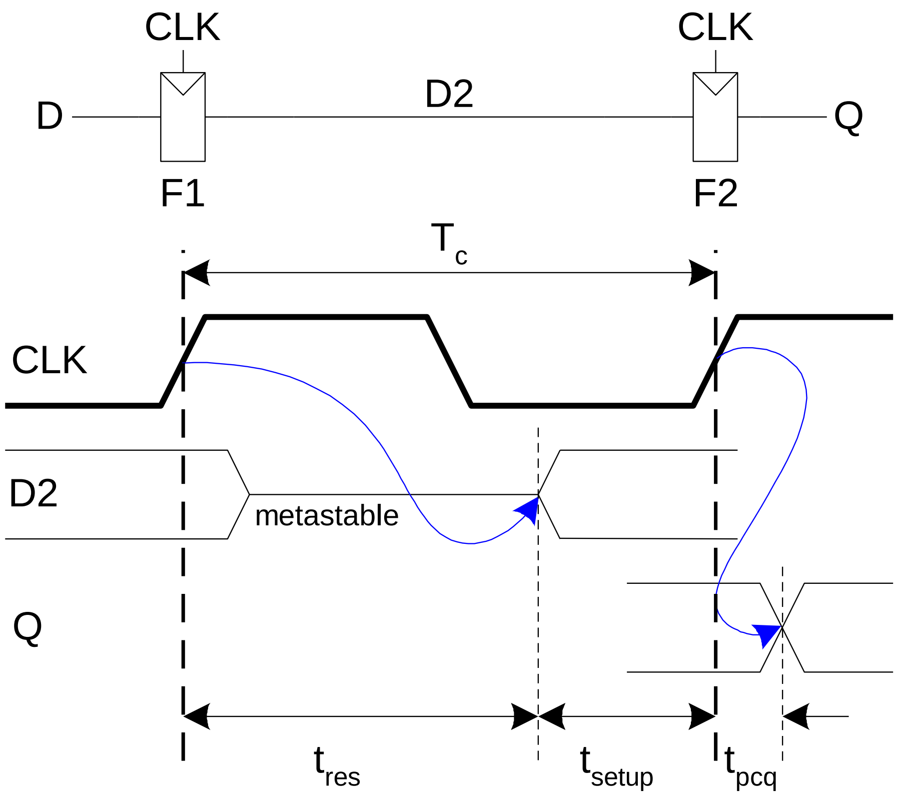
- **Synchronizer:** built with two back-to-back flip-flops
- Suppose D is transitioning when sampled by F1
- Internal signal D2 has (T_c - t_{setup}) time to resolve to 1 or 0



Synchronizer Probability of Failure

For each sample, probability of failure is:

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



Synchronizer Mean Time Between Failure

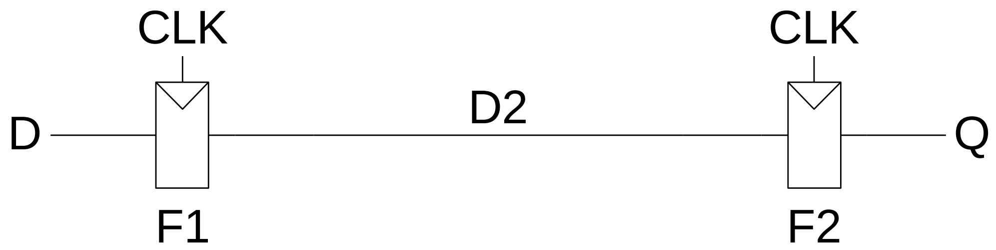
- If asynchronous input changes once per second, probability of failure per second is P(failure).
- If input changes N times per second, probability of failure per second is:

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

- Synchronizer fails, on average, 1/[P(failure)/second]
- Called ***mean time between failures***, MTBF:

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

Example Synchronizer



- **Suppose:** $T_c = 1/500\text{ MHz} = 2\text{ ns}$ $\quad \tau = 200\text{ ps}$
  $T_0 = 150\text{ ps}$ $\quad t_{setup} = 100\text{ ps}$
  $N = 10$ events per second
- What is the **probability of failure**? **MTBF**?

Parallelism

Parallelism

- **Two types of parallelism:**
  - **Spatial parallelism**
  - **Temporal parallelism**

Parallelism

- **Token:** Group of inputs processed to produce group of outputs
- **Latency:** Time for one token to pass from start to end
- **Throughput:** Number of tokens produced per unit time

Parallelism increases throughput

Parallelism Example

- Ben Bitdiddle bakes cookies to celebrate traffic light controller installation
  - **5 minutes** to roll cookies
  - **15 minutes** to bake
- What is the **latency** and **throughput** without parallelism?

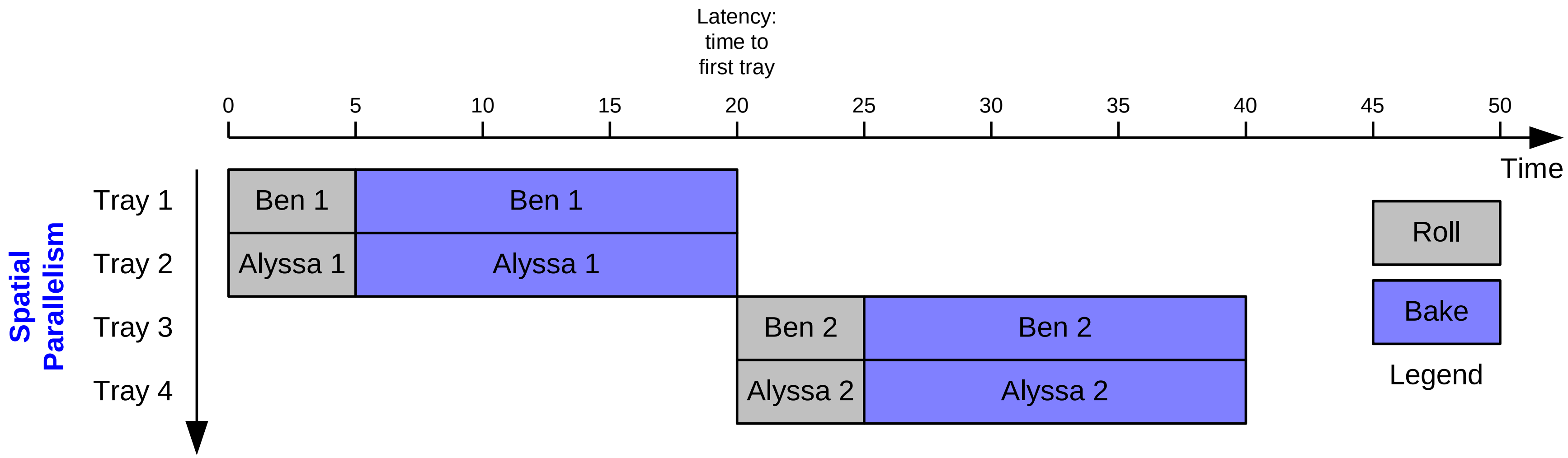
Latency =

Throughput =

Parallelism Example

- What is the latency and throughput if Ben uses parallelism?
  - **Spatial parallelism:** Ben asks Allysa P. Hacker to help, using her own oven
  - **Temporal parallelism:**
    - **two stages:** rolling and baking
    - He uses two trays
    - While first batch is baking, he rolls the second batch, etc.

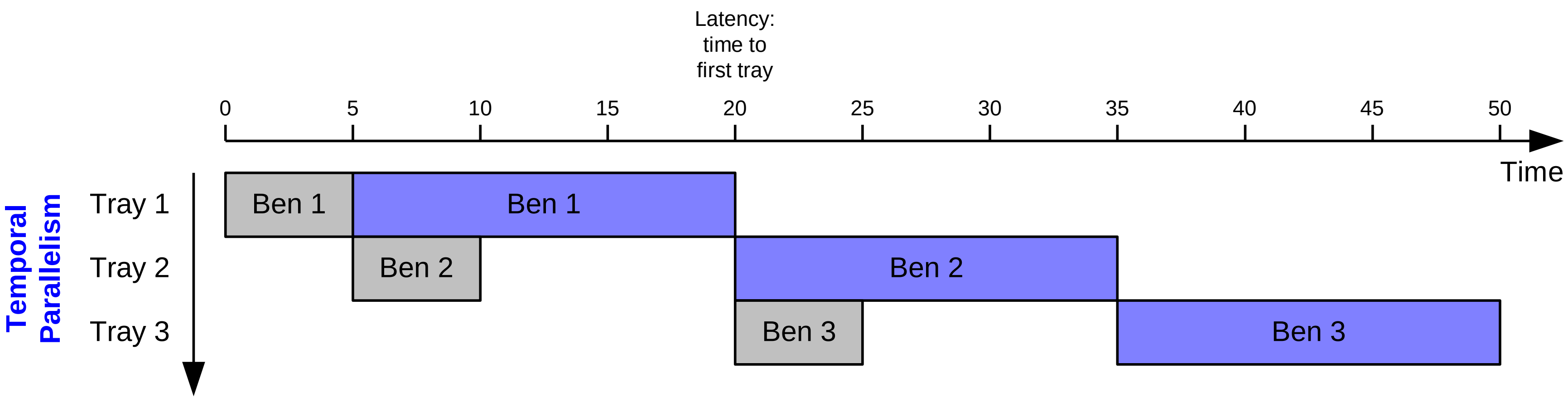
Spatial Parallelism



Latency =

Throughput =

Temporal Parallelism



Latency =

Throughput =

About these Notes

Digital Design and Computer Architecture Lecture Notes

© 2021 Sarah Harris and David Harris

These notes may be used and modified for educational and/or non-commercial purposes so long as the source is attributed.