Digital Design &
Computer Architecture

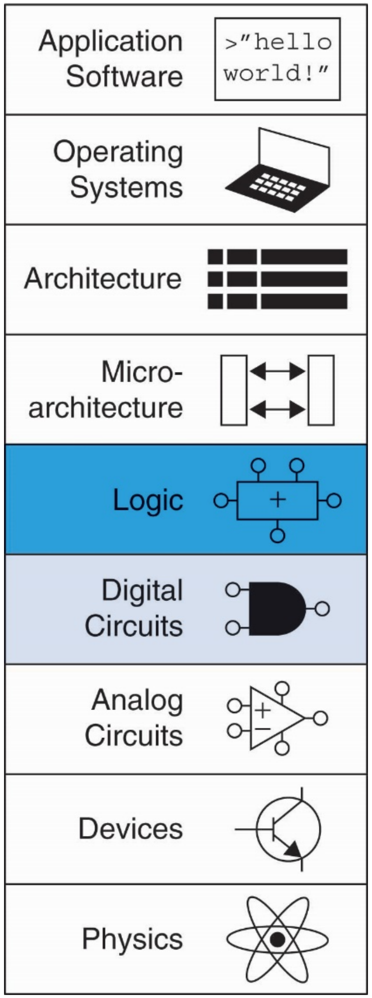
Sarah Harris & David Harris

Chapter 3: Sequential Logic Design

Modified by Younghwan Yoo, 2023

Chapter 3 :: Topics

- **State Elements**
  - **Bistable Circuit**
  - **SR Latch**
  - **D Latch**
  - **D Flip-Flop**
  - **Variations**
- **Synchronous Sequential Logic**
- **Finite State Machines**
  - **Moore**
  - **Mealy**
- **Metastability**
- **Parallelism**



Chapter 3: Sequential Logic

State Elements

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:** sequential circuits using flip-flops sharing a common clock

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

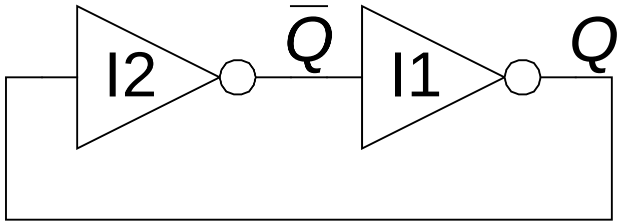
Chapter 3: Sequential Logic

Bistable Circuit

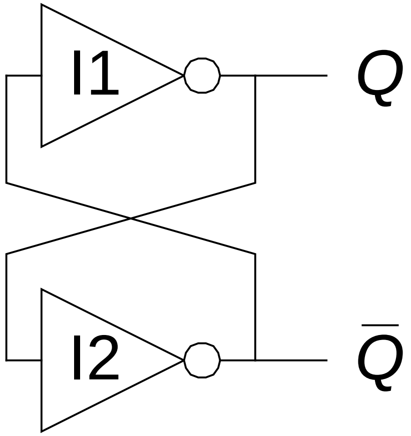
Bistable Circuit

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

Same circuit!



Back-to-back inverters

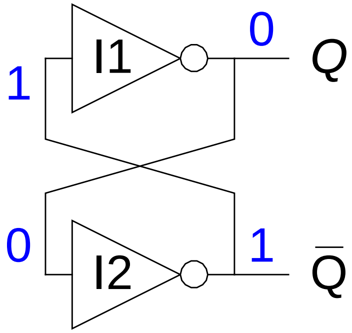


Cross-coupled inverters

Bistable Circuit Analysis

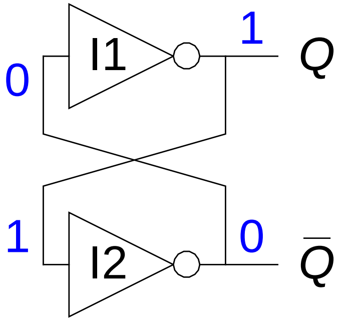
- Consider the two possible cases:
  - ***Q* = 0:**

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



  - ***Q* = 1:**

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



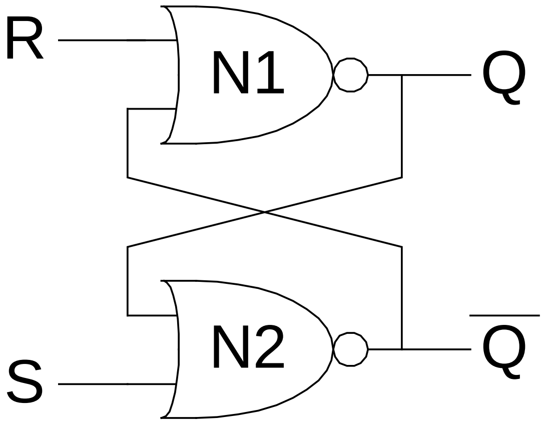
- Stores 1 bit of state in the state variable, Q (or $\overline{Q}$)
- But there are **no inputs to control the state**

Chapter 3: Sequential Logic

SR Latch

SR (Set/Reset) Latch

- **SR Latch**



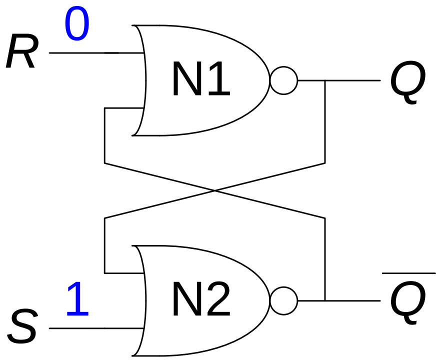
- Consider the four possible cases:
  - ***S* = 1, *R* = 0**
  - ***S* = 0, *R* = 1**
  - ***S* = 0, *R* = 0**
  - ***S* = 1, *R* = 1**

SR Latch Analysis

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

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

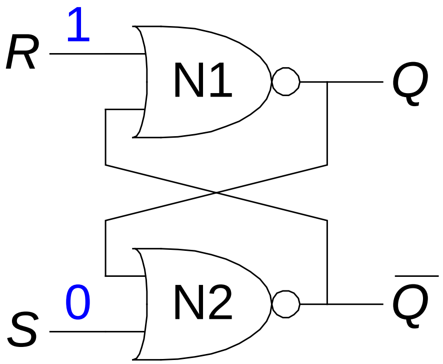
  ***Set* the output**



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

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

  ***Reset* the output**



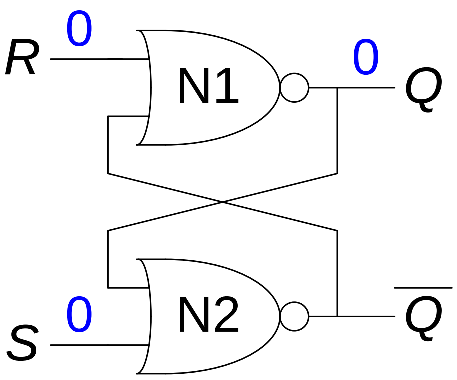
SR Latch Analysis

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

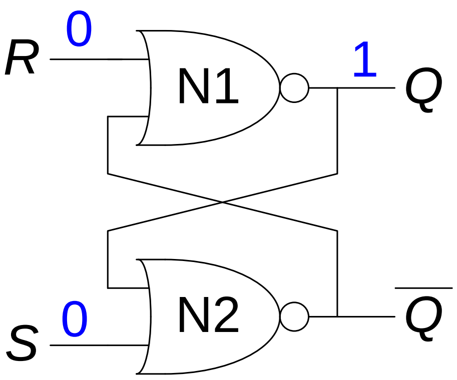
  then $Q = Q_{prev}$

  Memory!

$Q_{prev} = 0$



$Q_{prev} = 1$

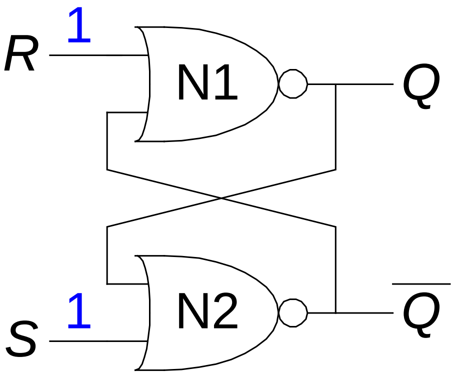


- ***S* = 1, *R* = 1:**

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

  Invalid State

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



SR Latch

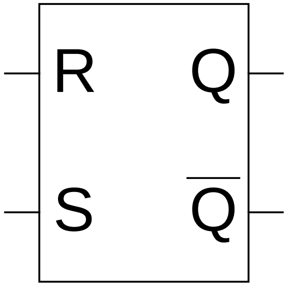
- **SR** stands for **S**et/**R**eset latch
  - Stores one bit of state (*Q*)
- Control what value is being stored with *S*, *R* inputs
  - **Set:** make the output 1

    $\boldsymbol{S = 1}$, $R = 0$, $Q = \mathbf{1}$
  - **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

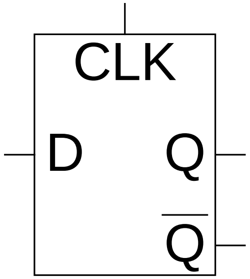
D Latch

D Latch

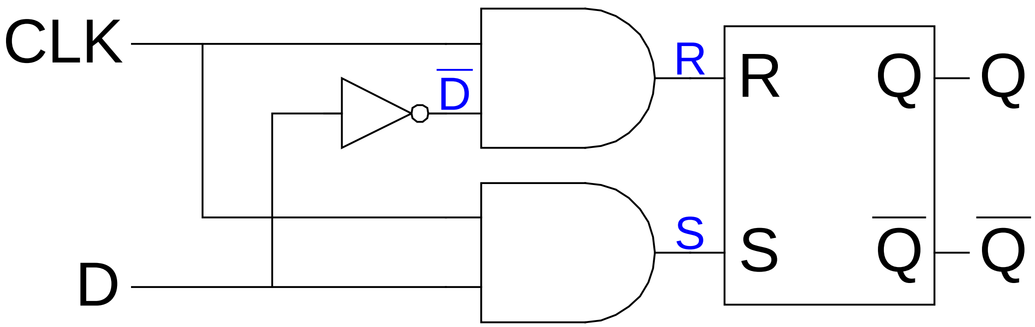
- **Two inputs:** *CLK*, *D*
  - ***CLK***: controls *when* the output changes
  - ***D*** (the data input): controls *what* the output changes to
- **Function**
  - When ***CLK* = 1**,
    D passes through to *Q* (*transparent*)
  - When ***CLK* = 0**,
    Q holds its previous value (*opaque*)
- **Avoids invalid case where**

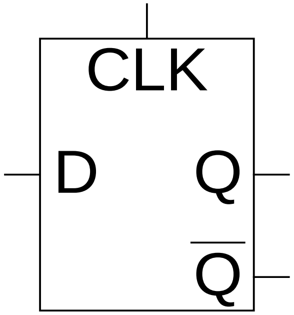
$$Q \neq \text{NOT } \overline{Q}$$

D Latch Symbol



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

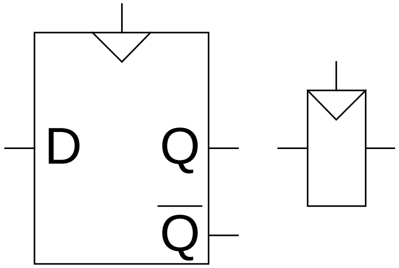
Chapter 3: Sequential Logic

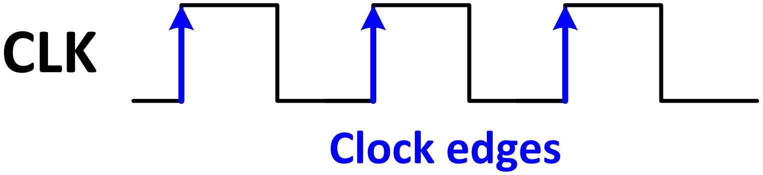
D Flip-Flop

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*

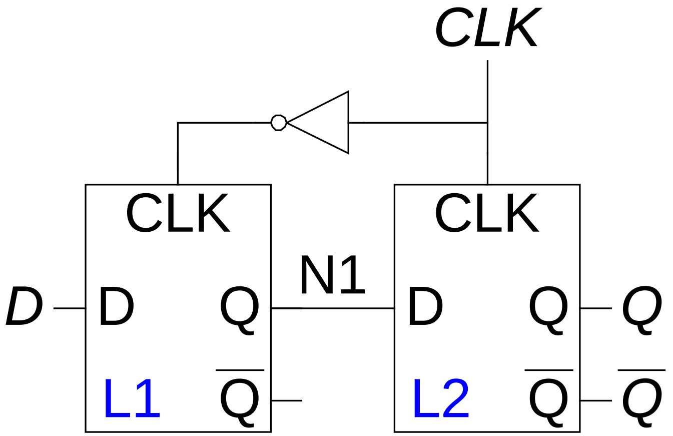
D Flip-Flop Symbols





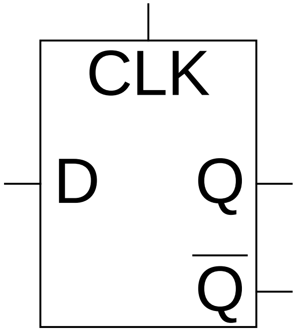
D Flip-Flop Internal Circuit

- **Two back-to-back D 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***

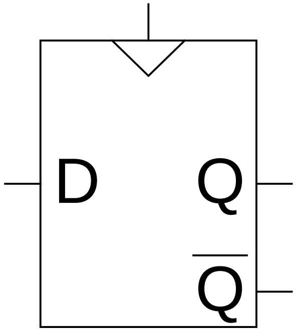


D Latch vs. D Flip-Flop

Level-sensitive, transparent when CLK=1

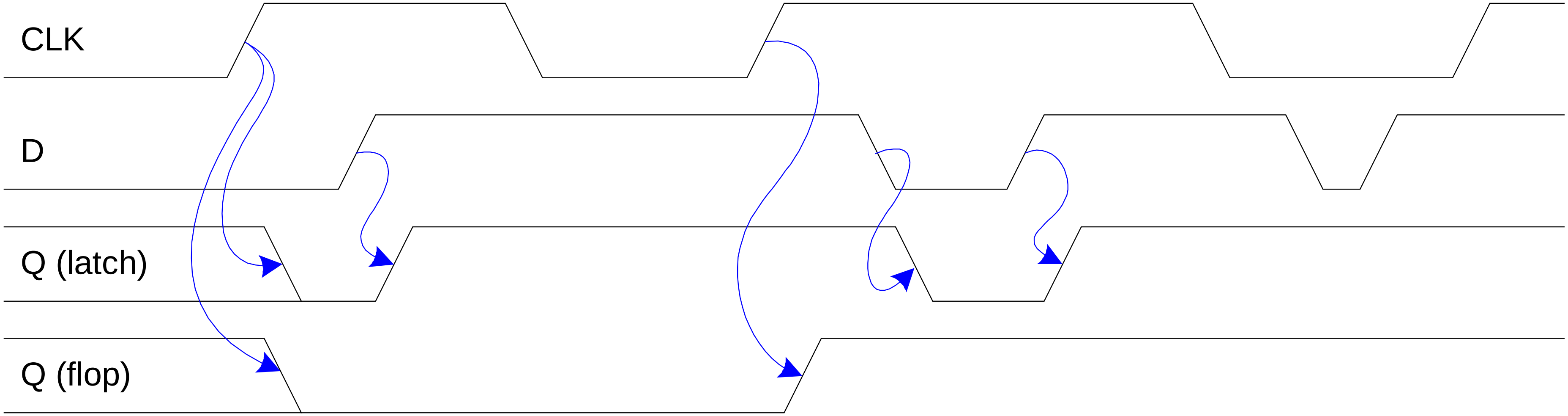


D Latch



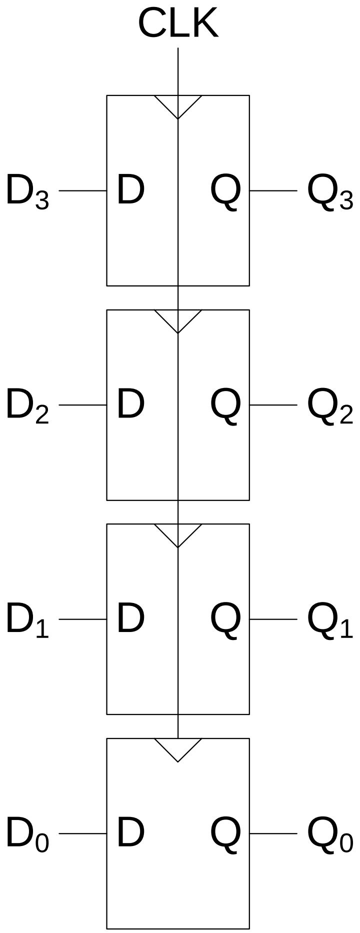
D Flip-flop

Edge-triggered, copies D to Q on the rising edge

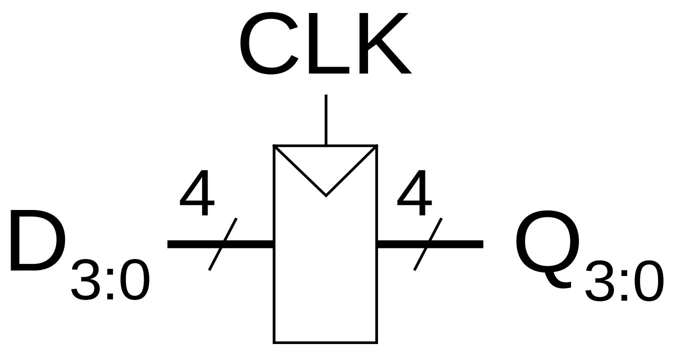


Variations on a Flop

Registers: One or More Flip-flops



4-bit Register



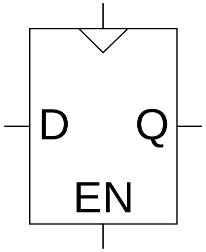
4-bit Register

Two ways to draw a register

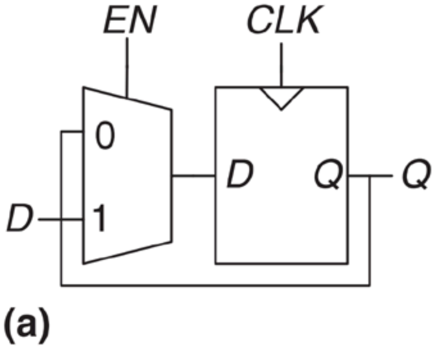
Enabled Flip-Flops

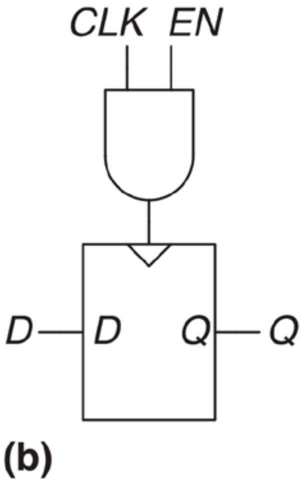
- **Inputs:** *CLK*, *D*, *EN*
  - The enable input (*EN*) controls when new data (*D*) is stored
- **Function**
  - ***EN* = 1:** *D* passes through to *Q* on the clock edge
  - ***EN* = 0:** the flip-flop retains its previous state

Symbol



Internal circuits

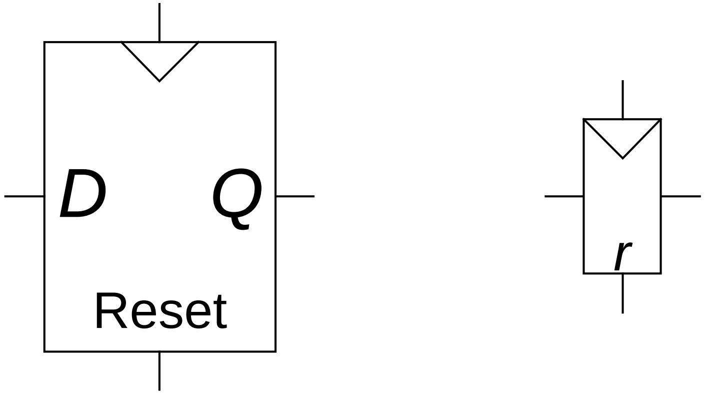




Resettable Flip-Flops

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

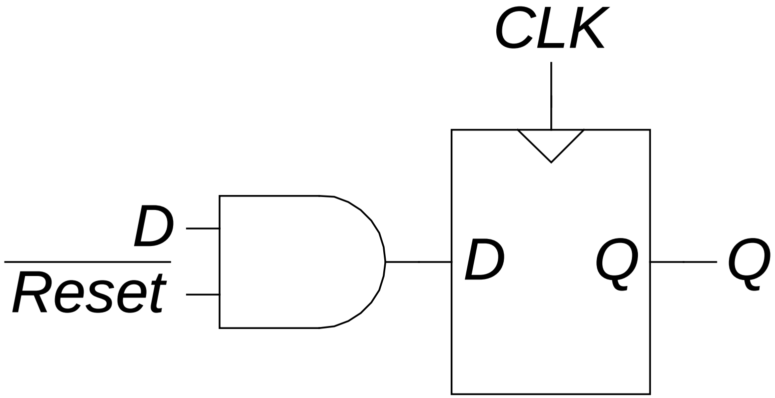
Symbols



Resettable Flip-Flops

- Two types:
  - **Synchronous:** resets at the clock edge only
  - **Asynchronous:** resets immediately when *Reset* = 1

Internal Circuit

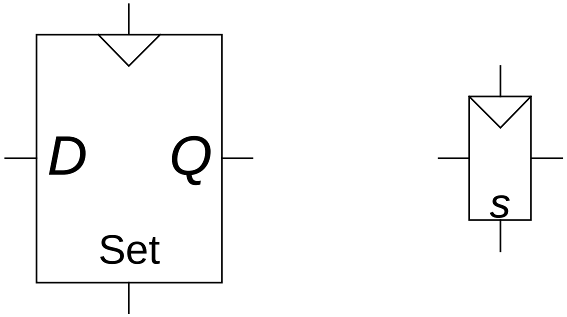


active low signal

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

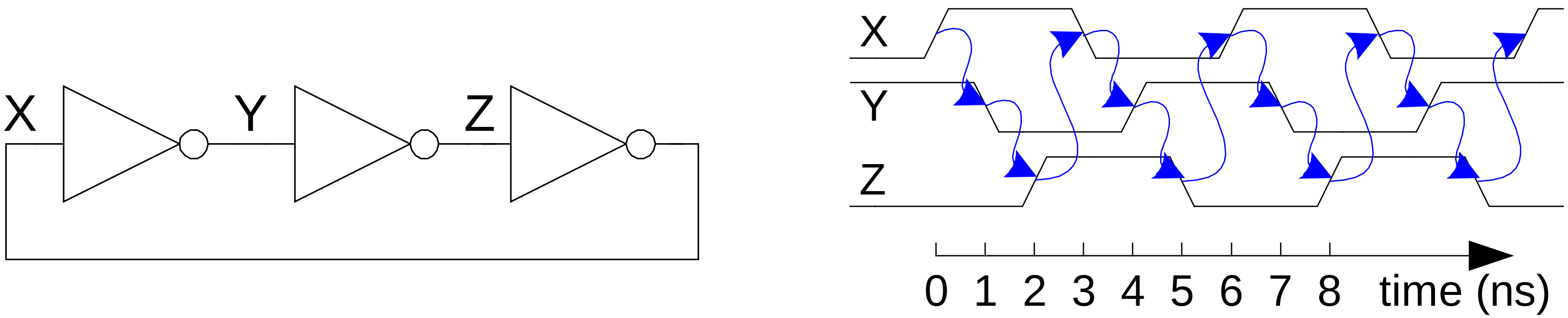


Chapter 3: Sequential Logic

Synchronous Sequential Logic

Sequential Logic

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



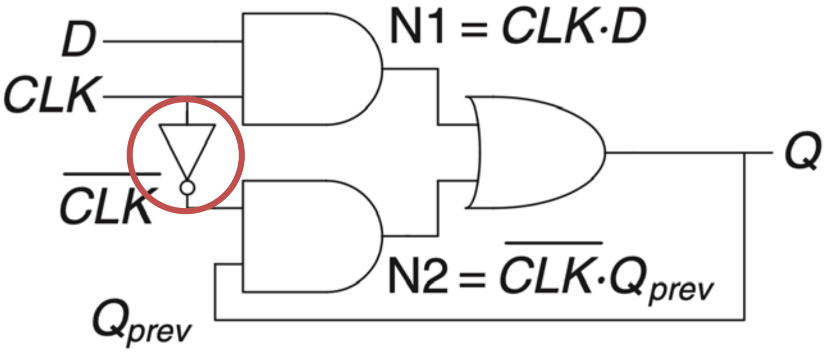
  - **No inputs** and 1-3 outputs
  - *Astable* (or *unstable*) circuit, **oscillates**
  - Period depends on inverter delay (1 ns)
  - It has a ***cyclic path***: output fed back to input

Race Condition

- An improved (?) D latch

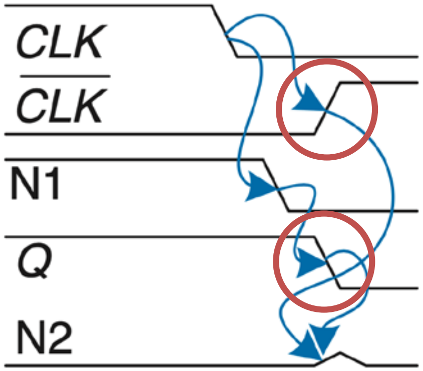
CLK	D	Q_{prev}	Q
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

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



- Suppose D=CLK=1, then Q=1
- CLK → 0, then N1 → 0, N2 → 1 expected
- If the delay of the NOT gate is long, Q → 0 while $\overline{\mathrm{CLK}}$ is still 0, then Q_{prev} → 0 and Q stuck at 0 forever

Race condition



Synchronous Sequential Logic Design

- Cyclic paths can have unstable behavior
- Breaks cyclic paths by **inserting registers**, transforming the circuit into a collection of combinational logic and registers
  - Registers contain state of the system
  - State changes only 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**

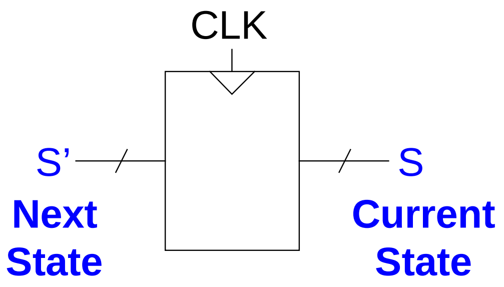
Chapter 3: Sequential Logic

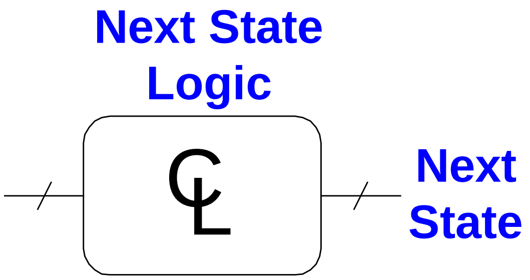
FSMs:

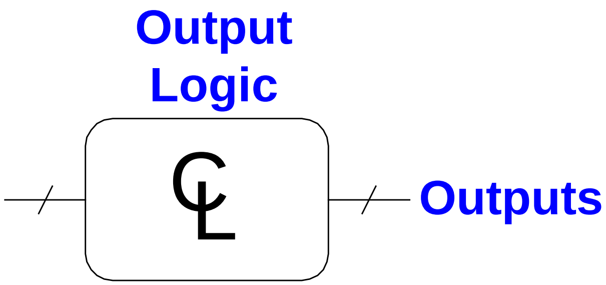
Finite State Machines

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





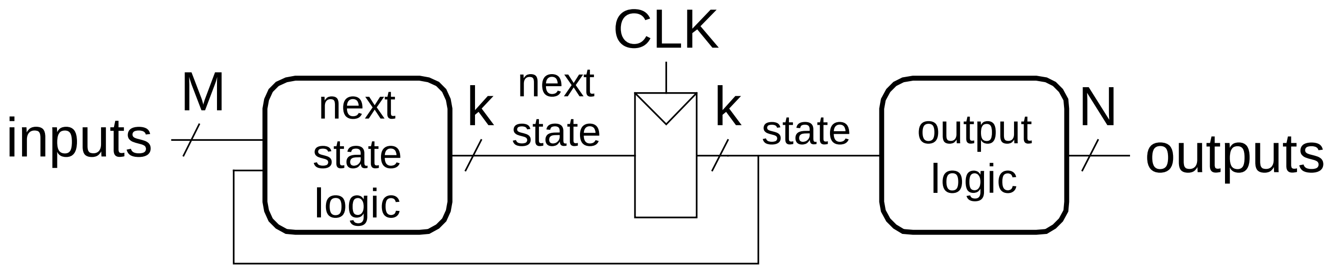


Finite State Machines (FSMs)

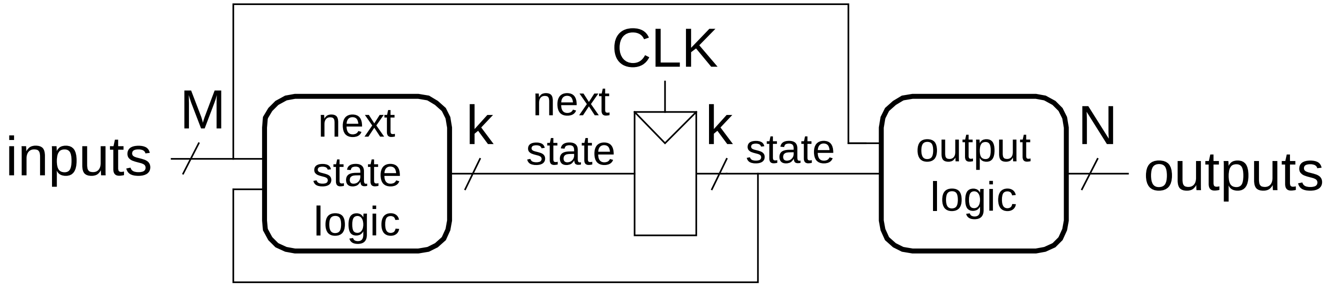
- **Next state** determined by **current state** and **inputs**
- Two types of finite state machines differ in **output** logic:
  - **Moore FSM: outputs** depend **only** on **current state**
  - **Mealy FSM: outputs** depend on **current state *and* inputs**

Finite State Machines (FSMs)

Moore FSM



Mealy FSM



FSM Design Procedure

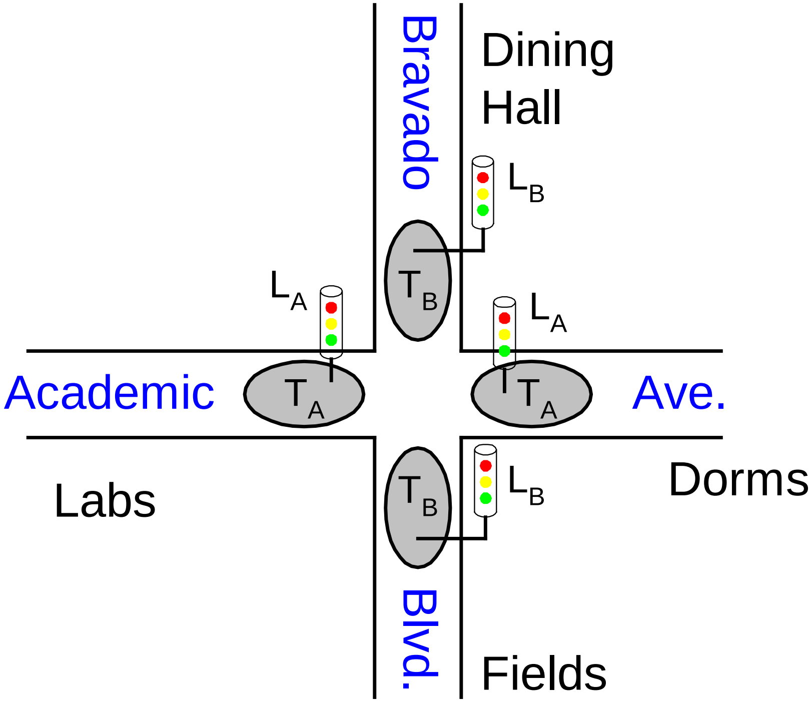
1. Identify **inputs** and **outputs**
2. Sketch **state transition diagram**
3. Write **state transition table** and **output table**
   - **Moore FSM:** write **separate** tables
   - **Mealy FSM:** write **combined** state transition and output table
4. Select **state encodings** represented in binary forms
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**

Chapter 3: Sequential Logic

Moore FSM Example

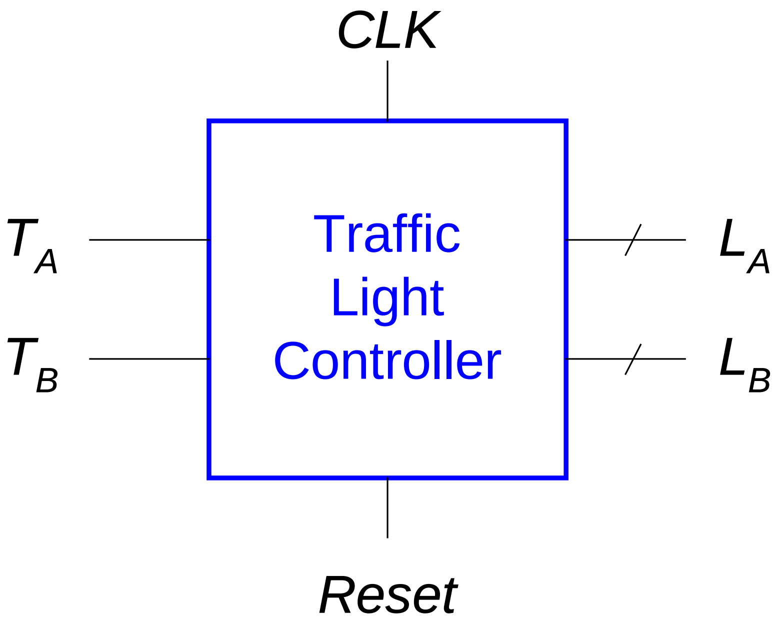
FSM Example

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



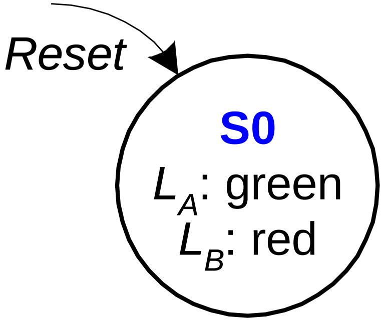
FSM Black Box

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



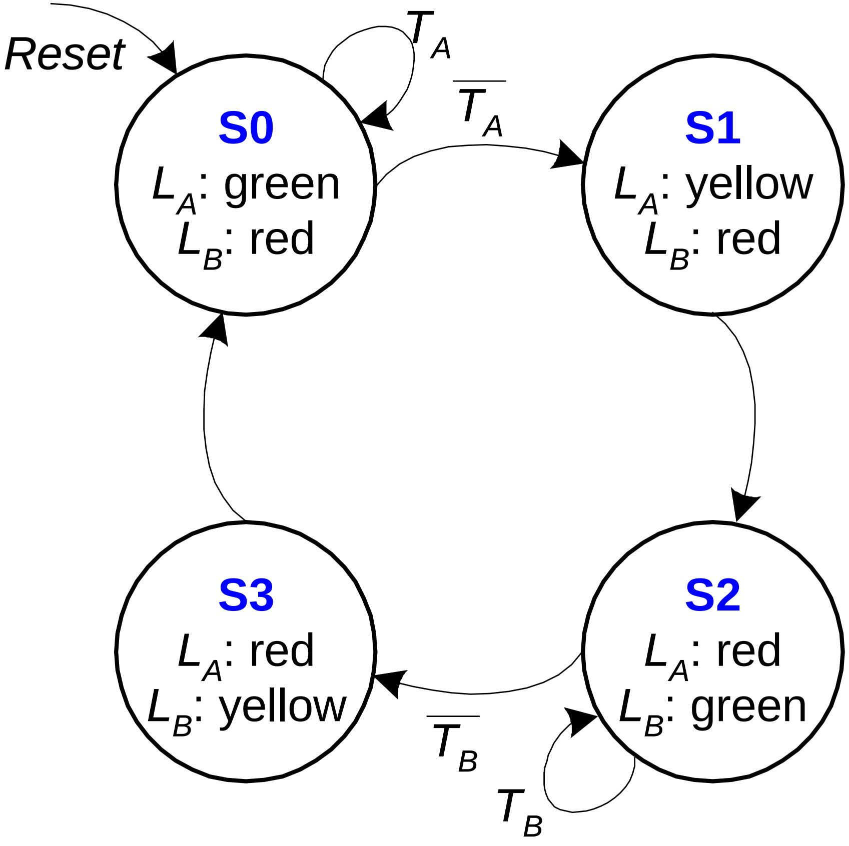
FSM State Transition Diagram

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



FSM State Transition Diagram

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

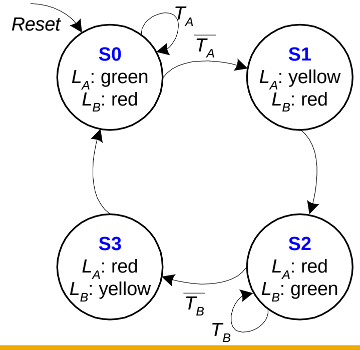


FSM State Transition Table

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
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

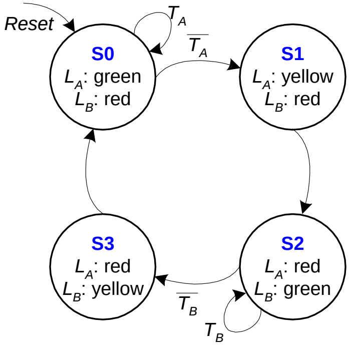
$$S'_1 = \overline{S_1}S_0 + S_1\overline{S_0}\,\overline{T_B} + S_1\overline{S_0}T_B = \overline{S_1}S_0 + S_1\overline{S_0} = S_1 \oplus S_0$$

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

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	



$$L_{A1} = S_1\overline{S_0} + S_1S_0 = S_1$$

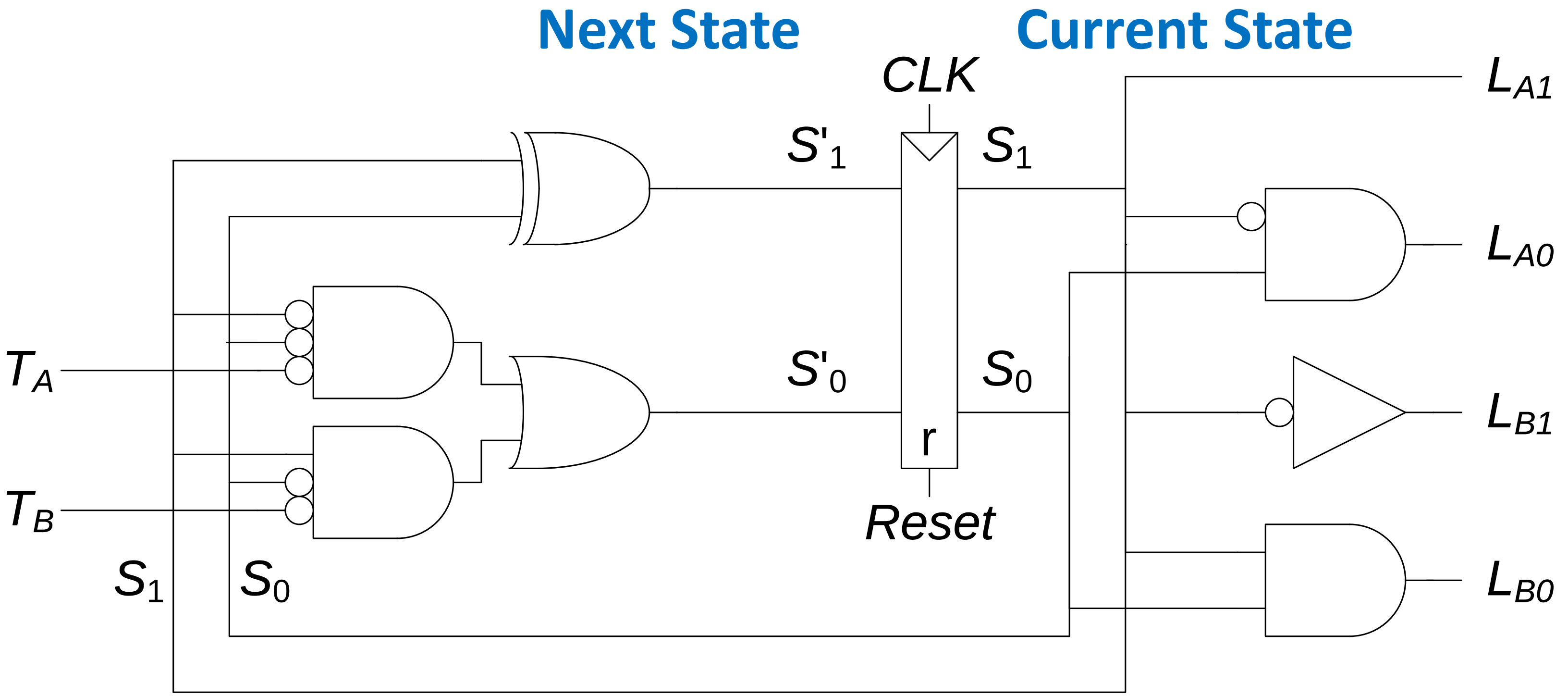
$$L_{A0} = \overline{S_1}S_0$$

$$L_{B1} = \overline{S_1}\,\overline{S_0} + \overline{S_1}S_0 = \overline{S_1}$$

$$L_{B0} = S_1S_0$$

Output	Encoding
green	00
yellow	01
red	10

FSM Schematic



Next State Logic

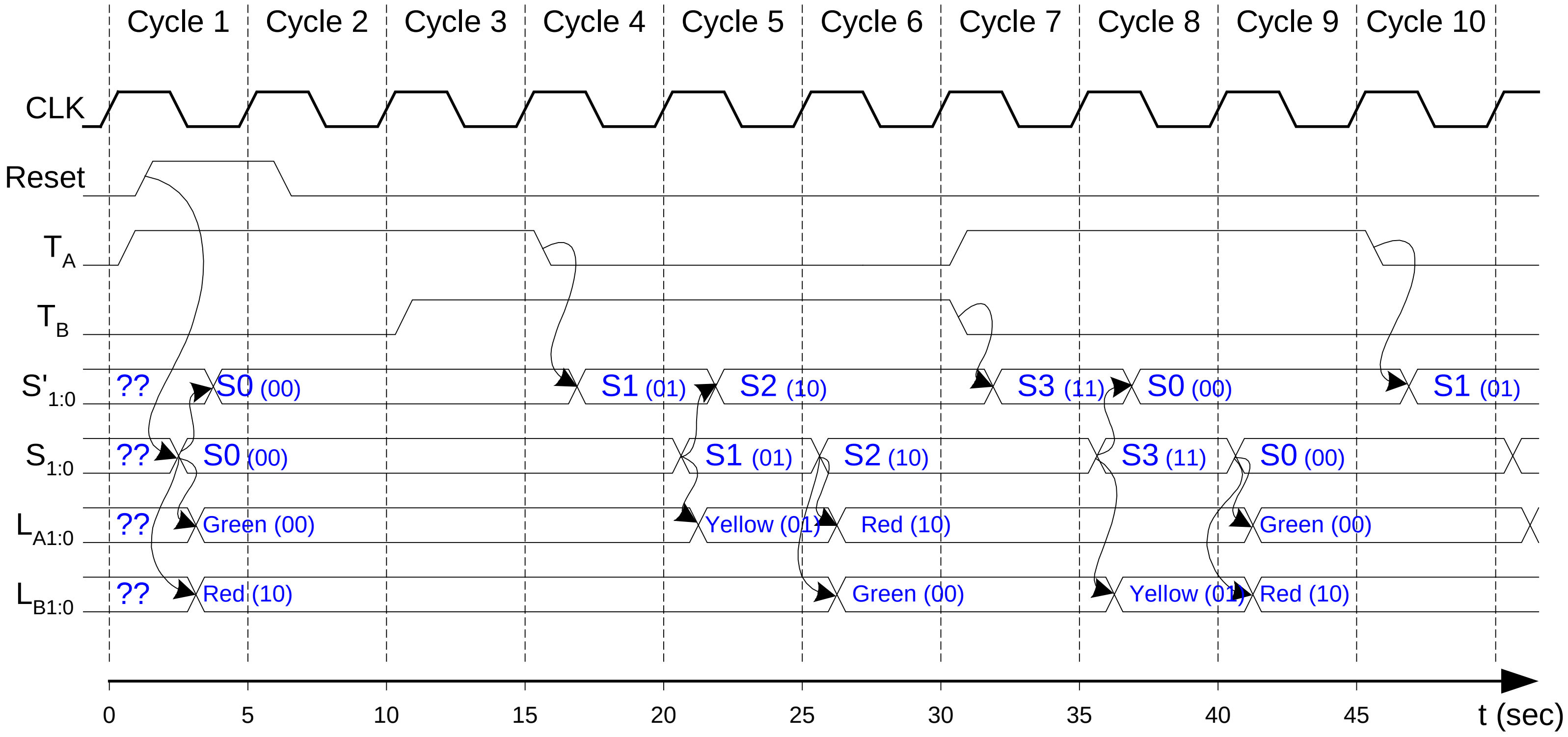
State Register

Output Logic

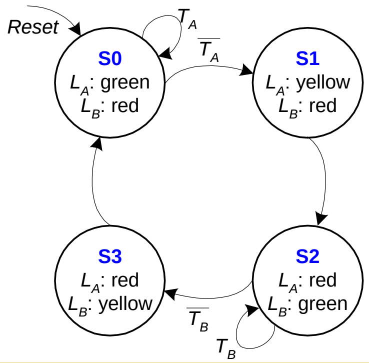
$$S'_1 = S_1 \oplus S_0$$
$$S'_0 = \overline{S_1}\,\overline{S_0}\,\overline{T_A} + S_1\overline{S_0}\,\overline{T_B}$$

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

FSM Timing Diagram



Cycle 1
Cycle 2
Cycle 3
Cycle 4
Cycle 5
Cycle 6
Cycle 7
Cycle 8
Cycle 9
Cycle 10
CLK
Reset
T_A
T_B
$S'_{1:0}$
?? S0 (00) S1 (01) S2 (10) S3 (11) S0 (00) S1 (01)
$S_{1:0}$
?? S0 (00) S1 (01) S2 (10) S3 (11) S0 (00)
$L_{A1:0}$
?? Green (00) Yellow (01) Red (10) Green (00)
$L_{B1:0}$
?? Red (10) Green (00) Yellow (01) Red (10)
0
5
10
15
20
25
30
35
40
45
t (sec)



State Encodings

- **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

1-Hot State Encoding Example

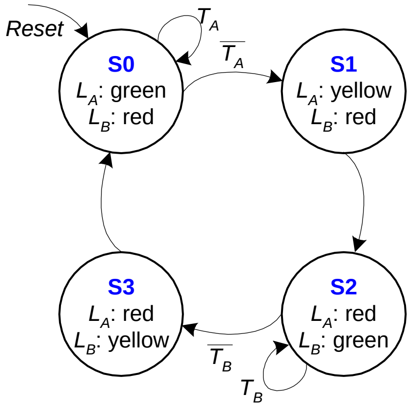
Current State				Inputs		Next State			
S_3	S_2	S_1	S_0	T_A	T_B	S'_3	S'_2	S'_1	S'_0
S0				0	X	S1			
S0				1	X	S0			
S1				X	X	S2			
S2				X	0	S3			
S2				X	1	S2			
S3				X	X	S0			

$$S'_3 = S_2 \overline{T_B}$$

$$S'_2 = S_1 + S_2 T_B$$

$$S'_1 = S_0 \overline{T_A}$$

$$S'_0 = S_0 T_A + S_3$$



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

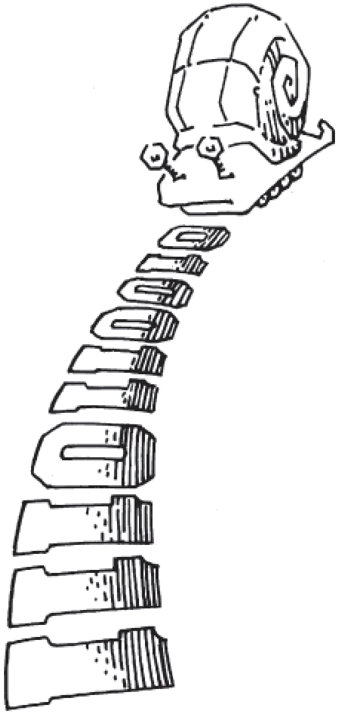
Chapter 3: Sequential Logic

Mealy FSM Example

Moore vs. Mealy FSMs

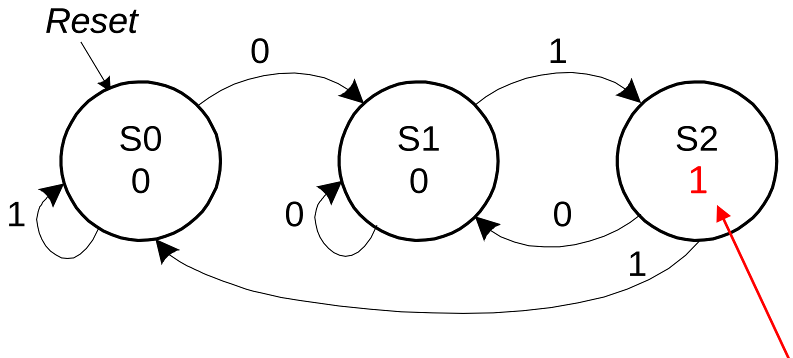
Alyssa P. Hacker has a snail that crawls down a paper tape with 1's and 0's on it. The snail smiles whenever the last two digits it has crawled over are 01. Design **Moore** and **Mealy** FSMs to compute when the snail should smile.

- Moore FSM: **outputs** depend **only** on **current state**
- Mealy FSM: **outputs** depend on **current state *and* inputs**

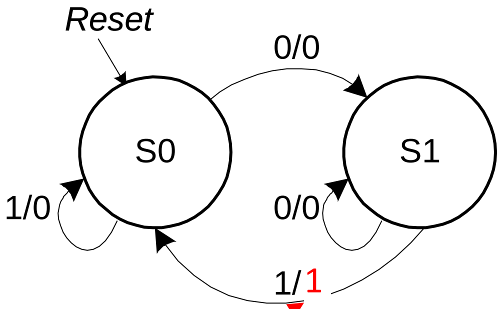


State Transition Diagrams

Moore FSM



Mealy FSM



Output:
"The snail smiles."

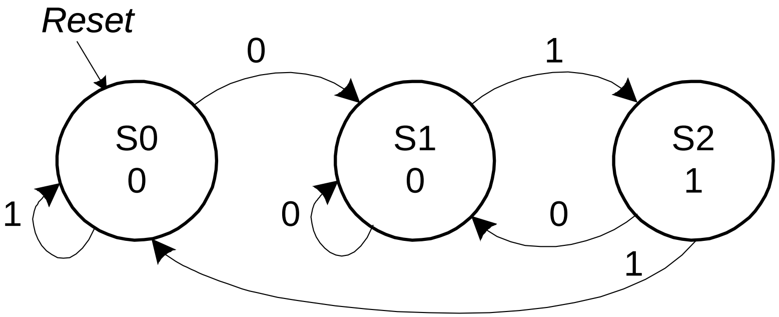
Moore FSM State Transition Table

Current State		Inputs	Next State	
S_1	S_0	A	S'_1	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

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

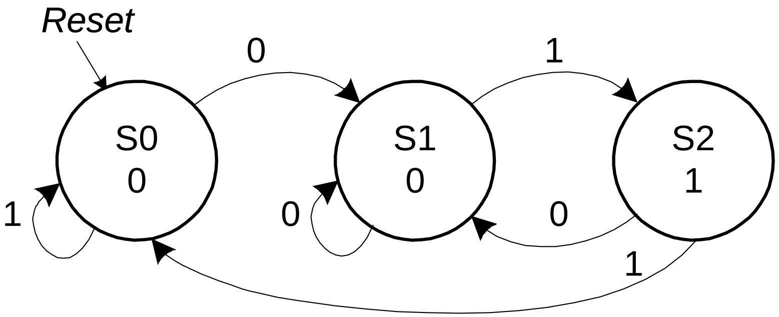
$$S_0' = \overline{A}$$



Moore FSM Output Table

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

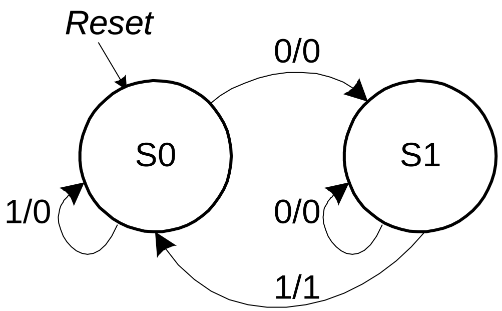
State	Encoding
S0	00
S1	01
S2	10



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

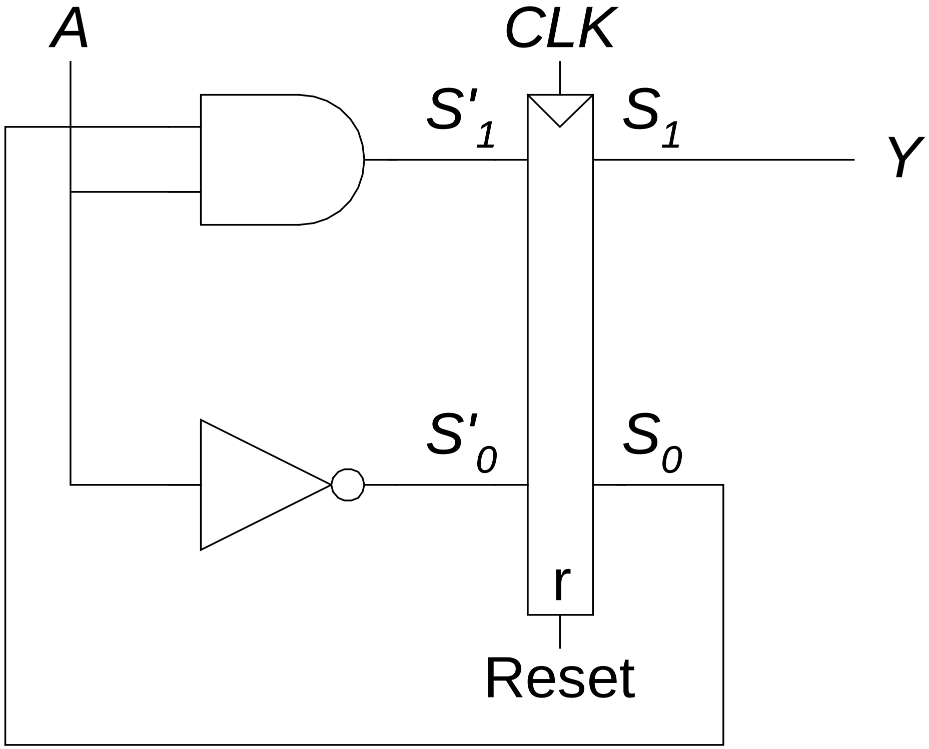
Next State Equations

$S_1' = S_0 A$

$S_0' = \overline{A}$

Output Equation

$Y = S_1$



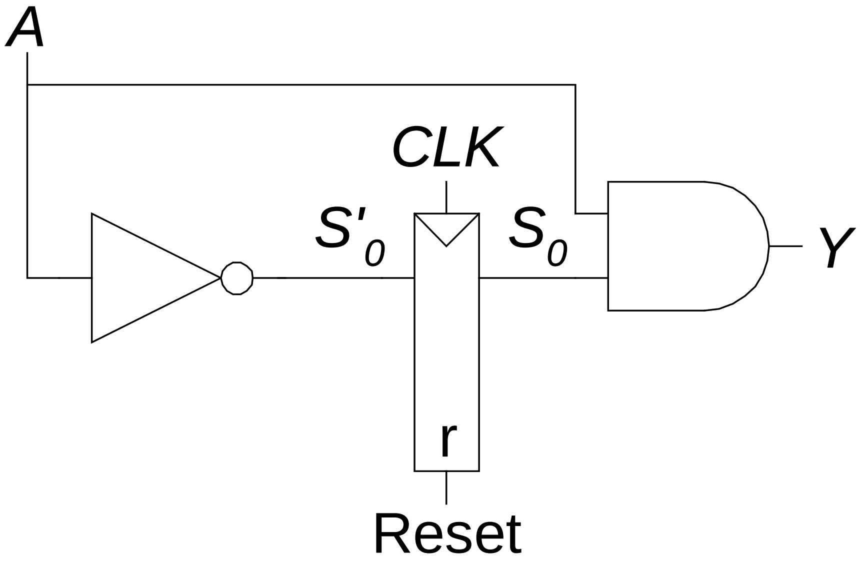
Mealy FSM Schematic

Next State Equation

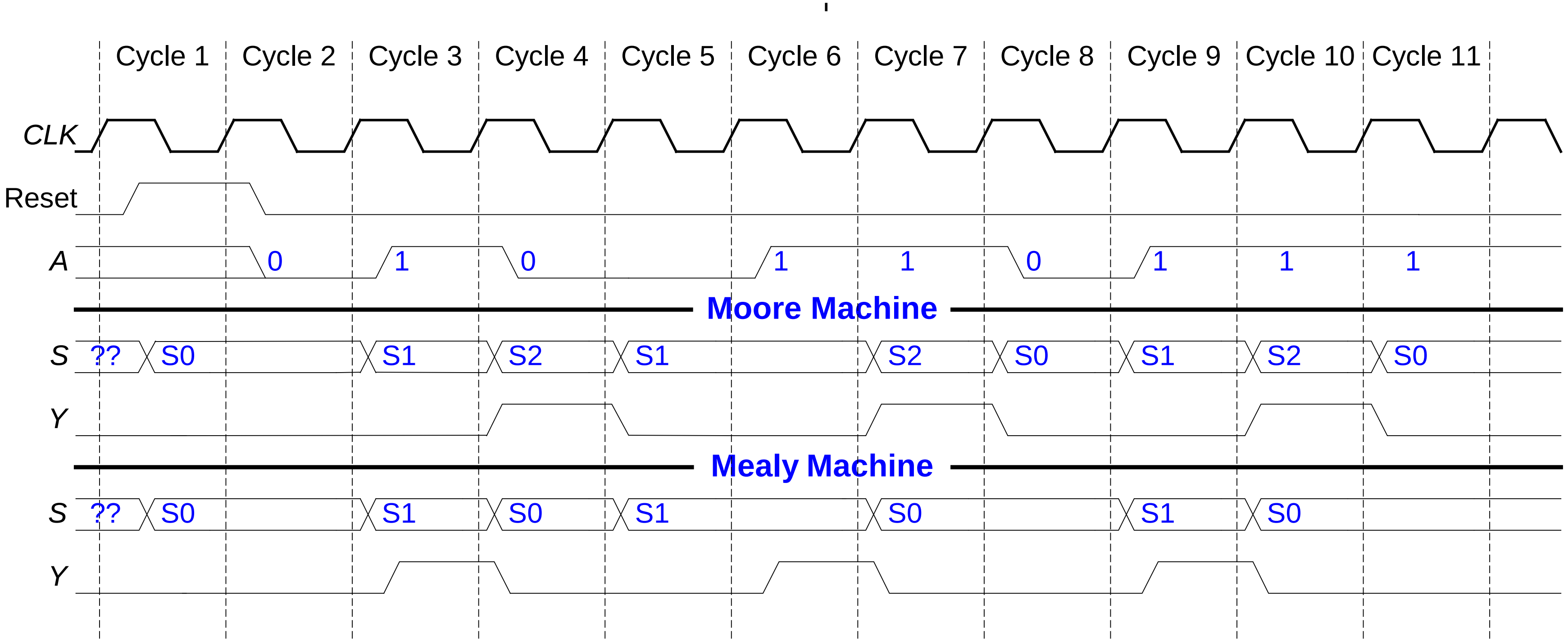
$S_0' = \overline{A}$

Output Equation

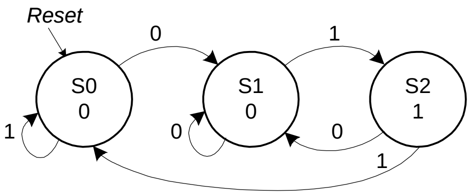
$Y = S_0 A$



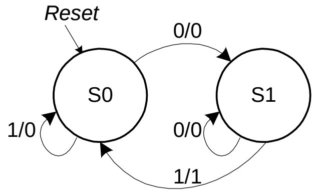
Moore and Mealy Timing Diagram



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

Chapter 3: Sequential Logic

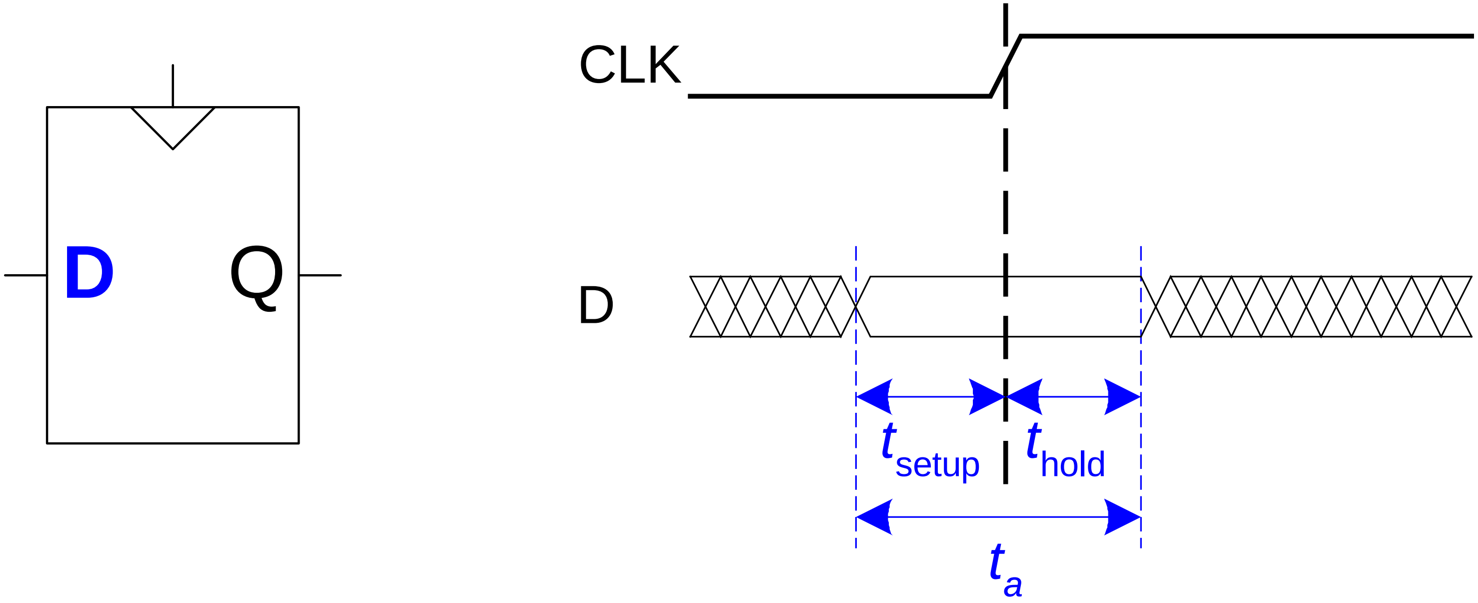
Timing

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
  - If an object moves while taking a picture, the image will be blurred.

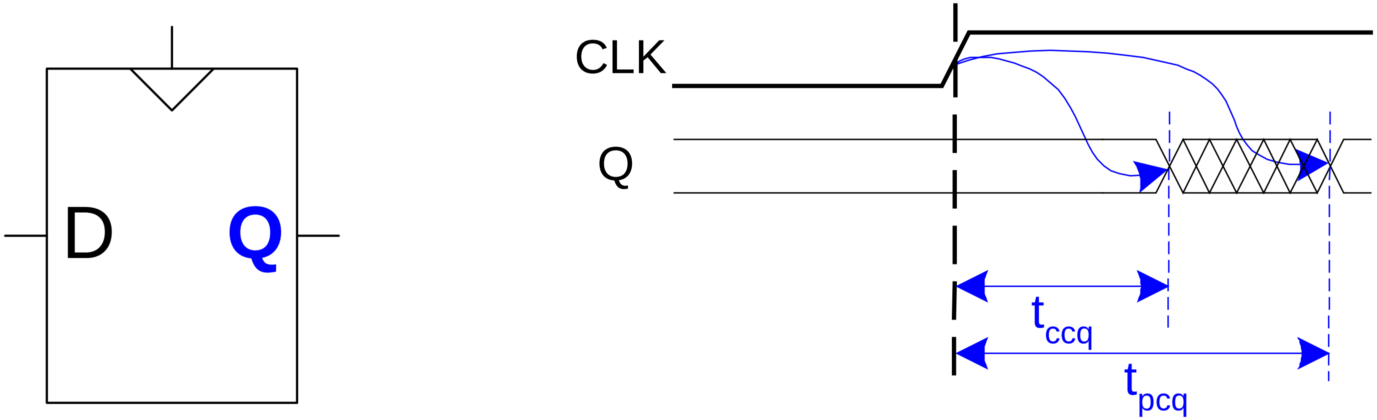
Input Timing Constraints

- **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

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

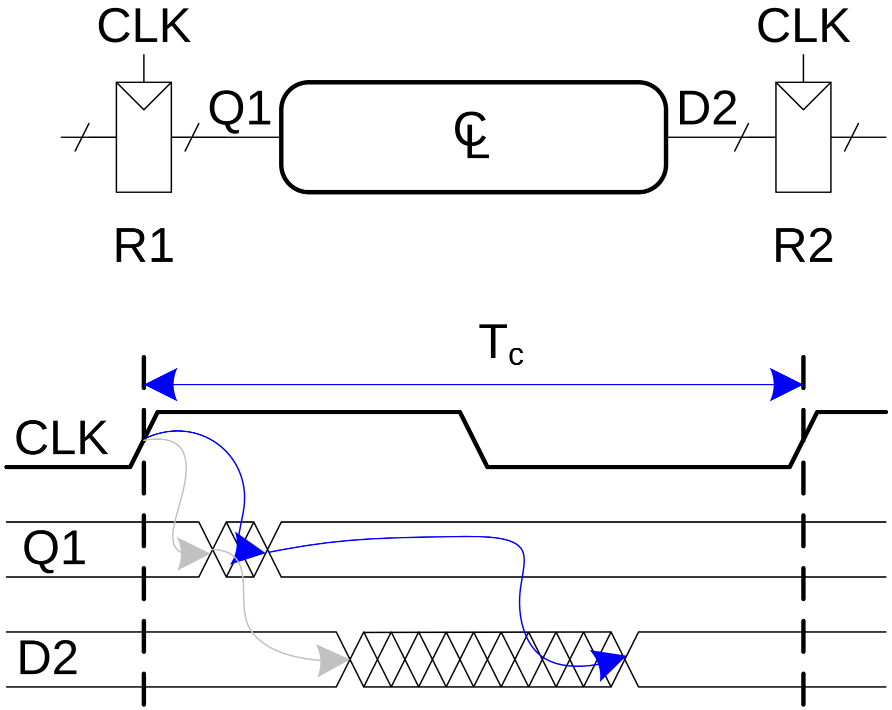


Dynamic Discipline

- Synchronous sequential circuit inputs must be **stable during aperture** (setup and hold) time around clock edge
- Specifically, inputs must be stable
  - at least $\boldsymbol{t}_{\mathbf{setup}}$ before the clock edge
  - at least until $\boldsymbol{t}_{\mathbf{hold}}$ after the clock edge

Dynamic Discipline

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

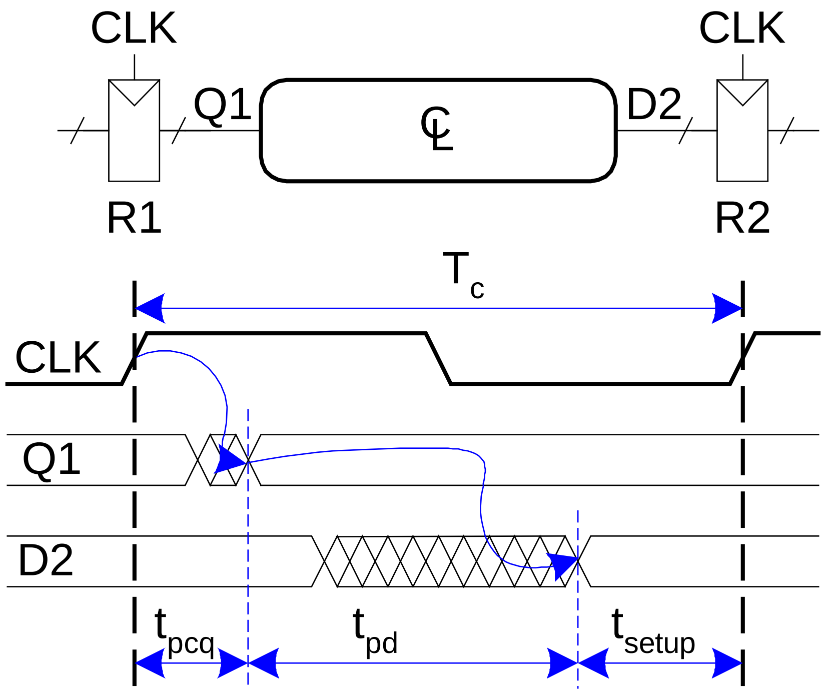


Setup Time Constraint

- 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

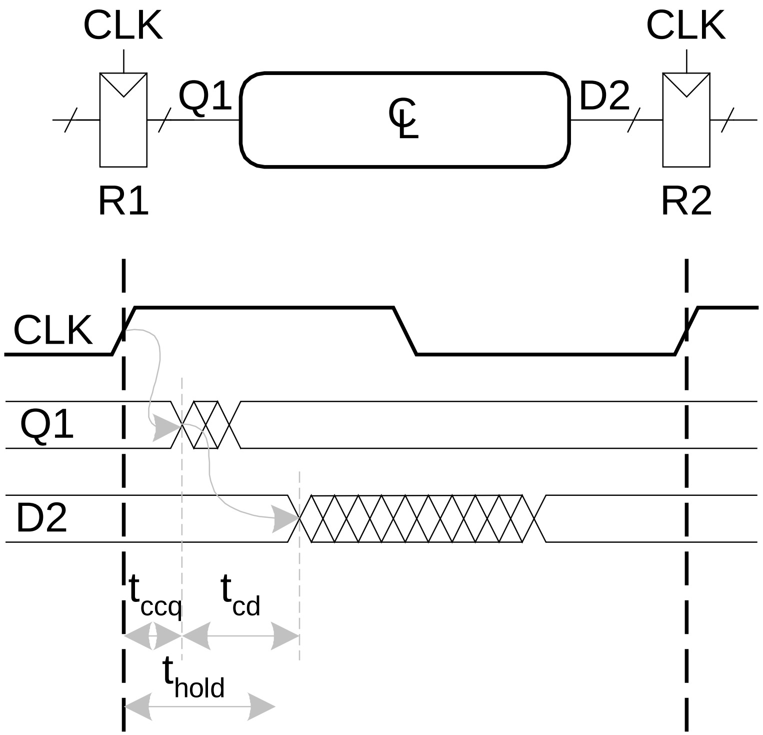


$$T_c \geq t_{pcq} + t_{pd} + t_{setup}$$
$$t_{pd} \leq T_c - (t_{pcq} + t_{setup})$$

$(t_{pcq} + t_{setup})$: *sequencing overhead*

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



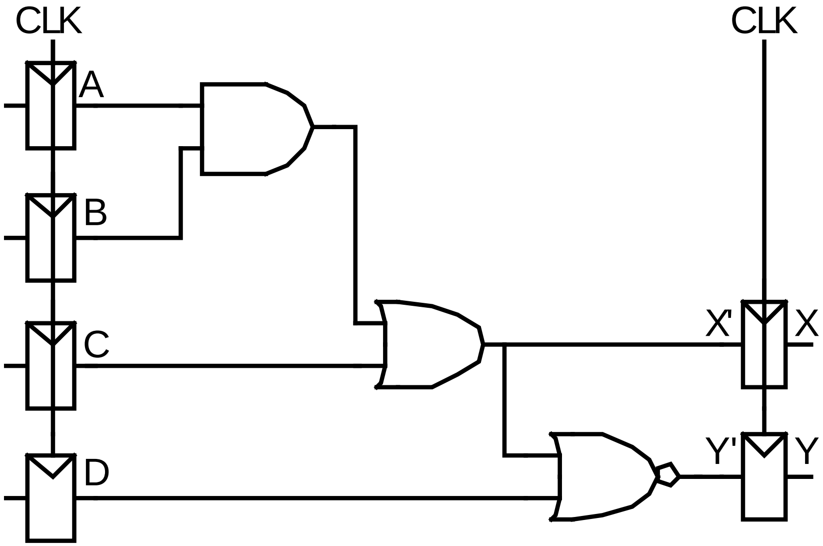
$$t_{hold} < t_{ccq} + t_{cd}$$

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

Timing Analysis

- Calculate **both constraints**:
  - **Setup time** constraint (a.k.a. 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

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

t_{cd} = 25 ps

Setup time constraint:

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

$f_c = 1/T_c$ = 4.65 GHz

Hold time constraint:

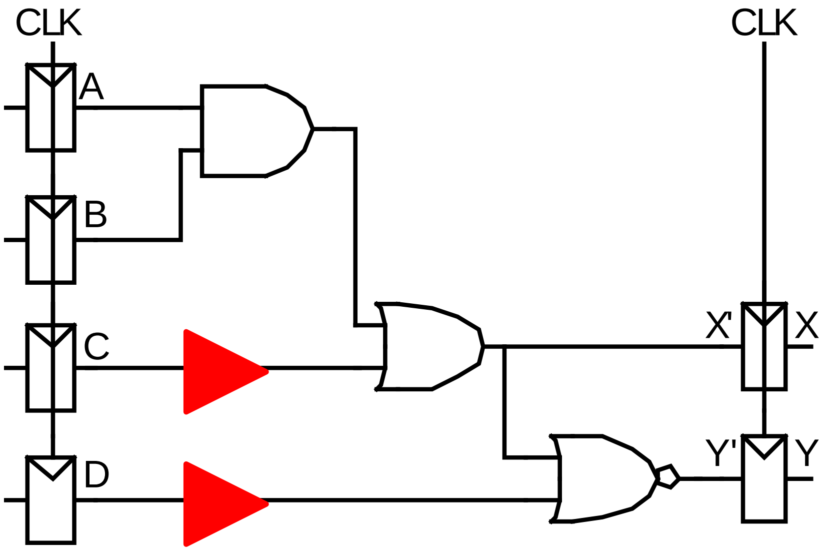
$t_{\text{ccq}} + t_{cd} > t_{\text{hold}}$?

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

Won't run reliably at any frequency

Timing Analysis Example

Add buffers on short paths



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} = **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:

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

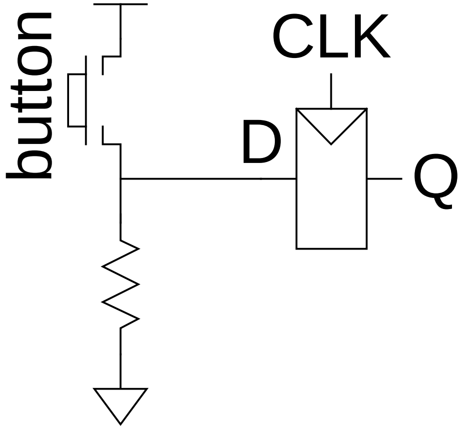
(30 + **50**) ps > 70 ps ? **Yes!**

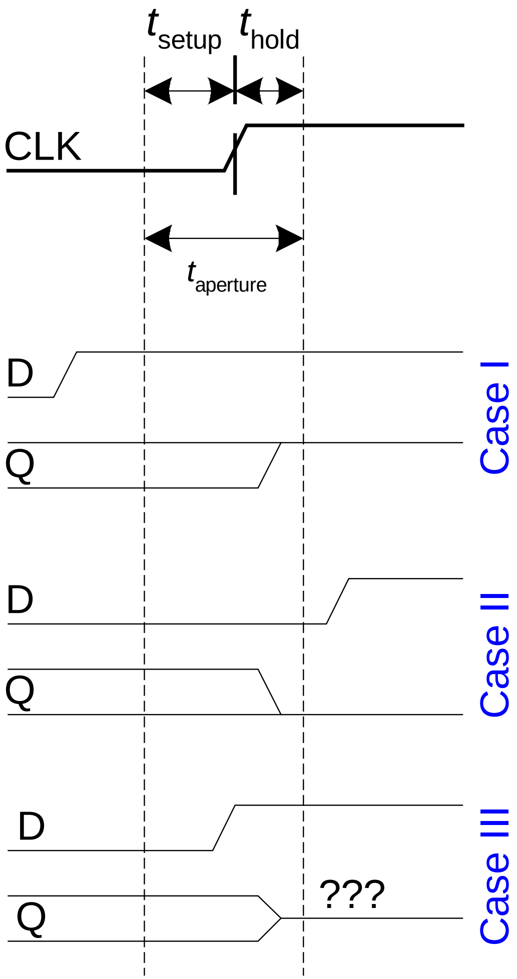
Chapter 3: Sequential Logic

Metastability

Violating the Dynamic Discipline

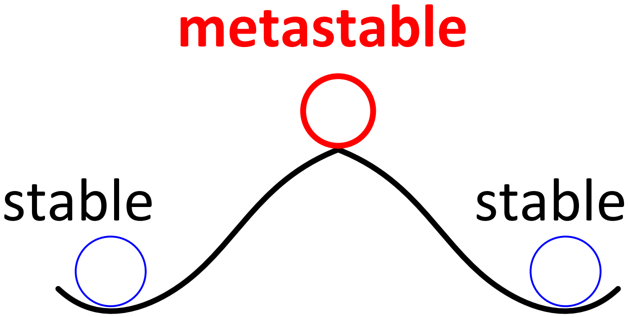
Asynchronous (for example, user) **inputs** might violate the dynamic discipline





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



Chapter 3: Sequential Logic

Parallelism

Parallelism

- **Two types of parallelism:**
  - Spatial parallelism
    - duplicate hardware performs multiple tasks at once
  - Temporal parallelism
    - task is broken into multiple stages
    - also called pipelining
    - for example, an assembly line

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?

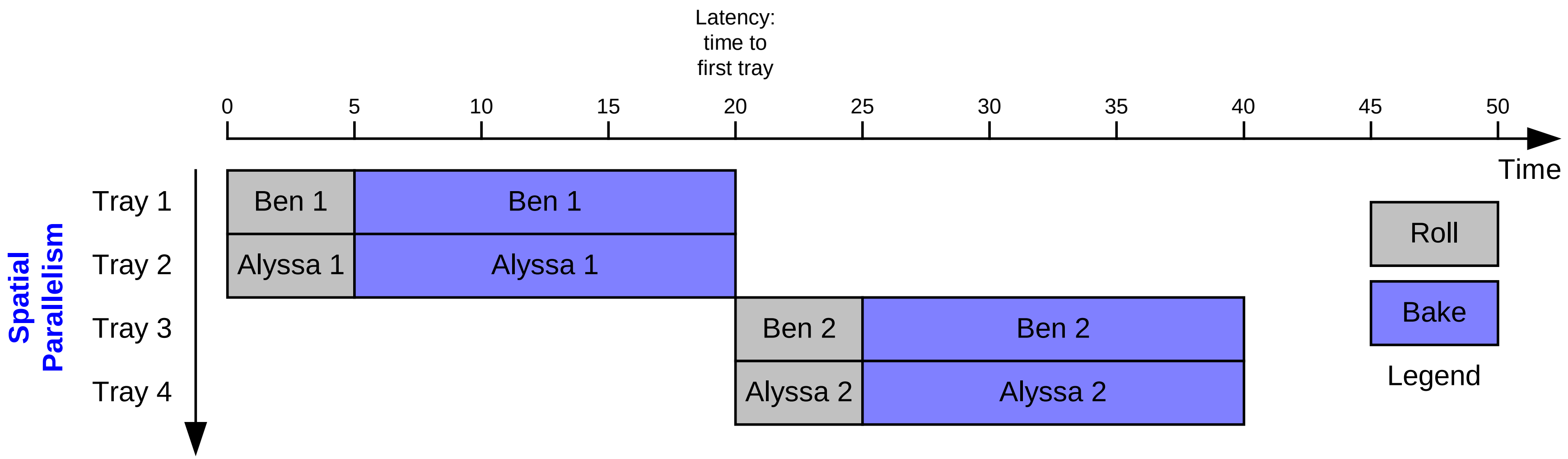
Latency = 5 + 15 = 20 minutes = **1/3 hour**

Throughput = 1 tray/ 1/3 hour = **3 trays/hour**

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.

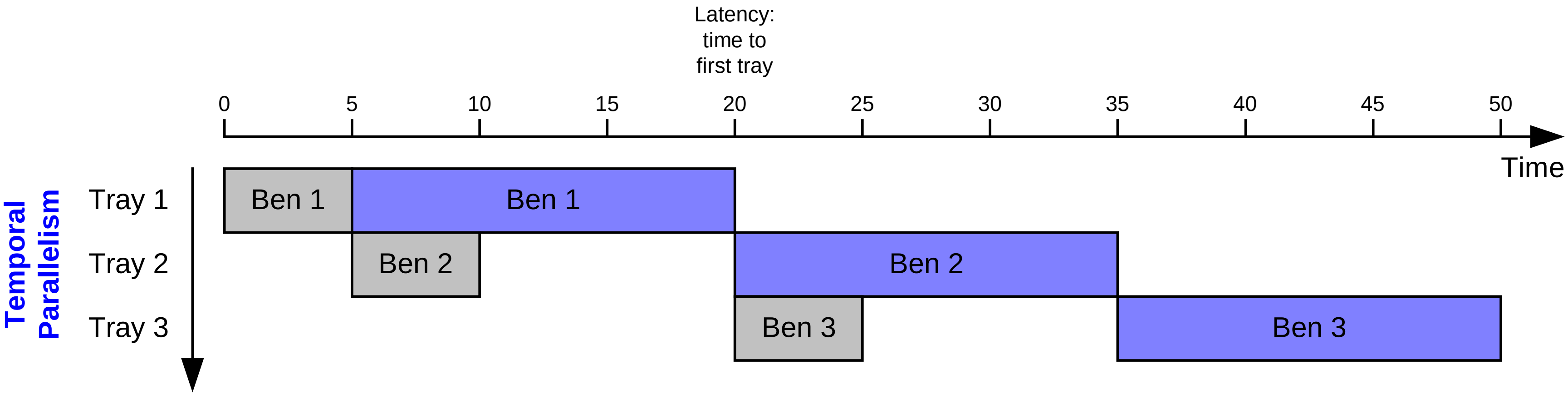
Spatial Parallelism



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

Throughput = 2 trays/ 1/3 hour = **6 trays/hour**

Temporal Parallelism



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

Throughput = 1 trays/ 1/4 hour = **4 trays/hour**

Using both techniques, the throughput would be **8 trays/hour**

About these Notes

Digital Design and Computer Architecture Lecture Notes