Review$_9$

- □ Multiplexers (System Verilog model)
- □ Decoders (System Verilog Implementation)
- □ Logic Using Decoders
- □ Timing
- □ Propagation & Contamination Delay
- □ Critical (Long) & Short Paths
- □ Glitches

1st Midterm

14/11/2018

17:30

SEQUENTIAL LOGIC DESIGN

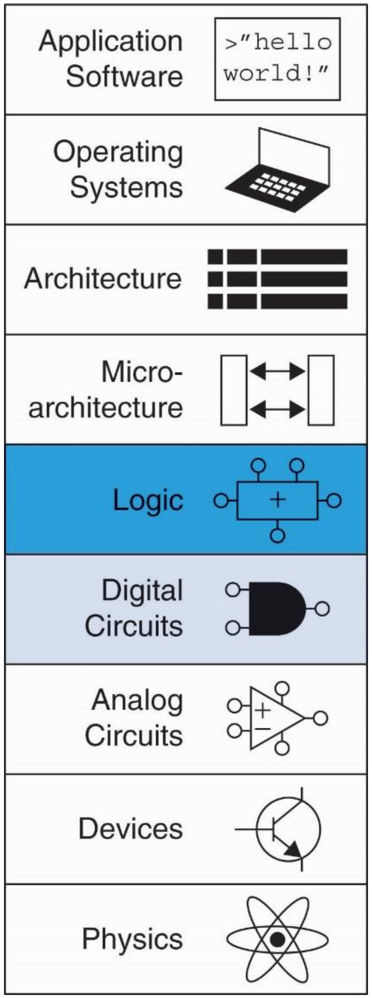
Chapter 3

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

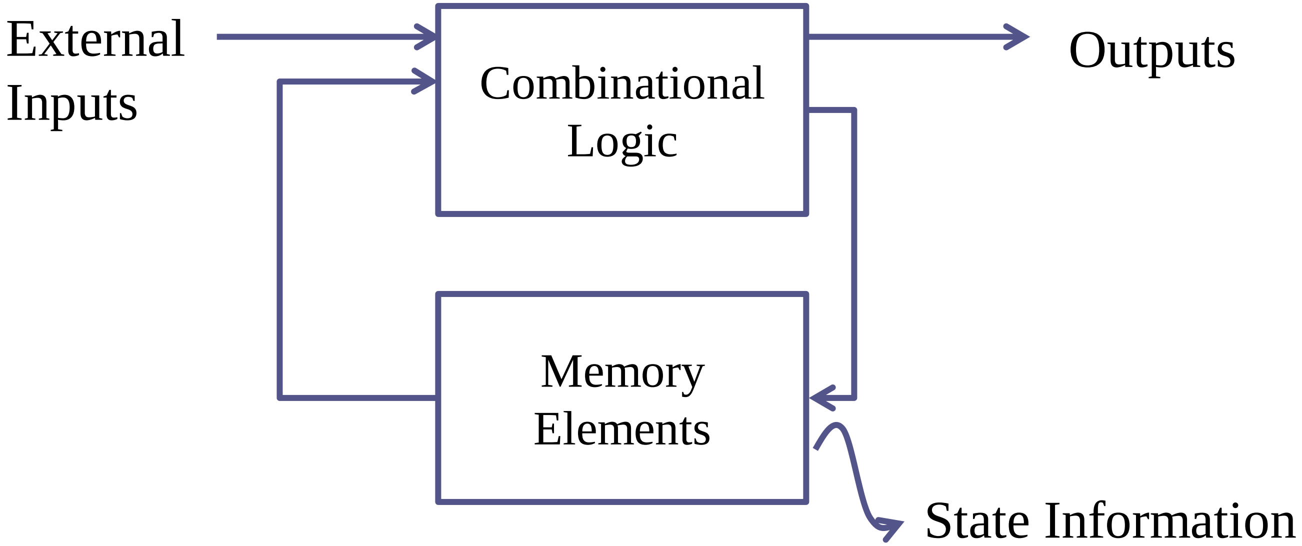
David Money Harris and Sarah L. Harris

Chapter 3 :: Topics

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

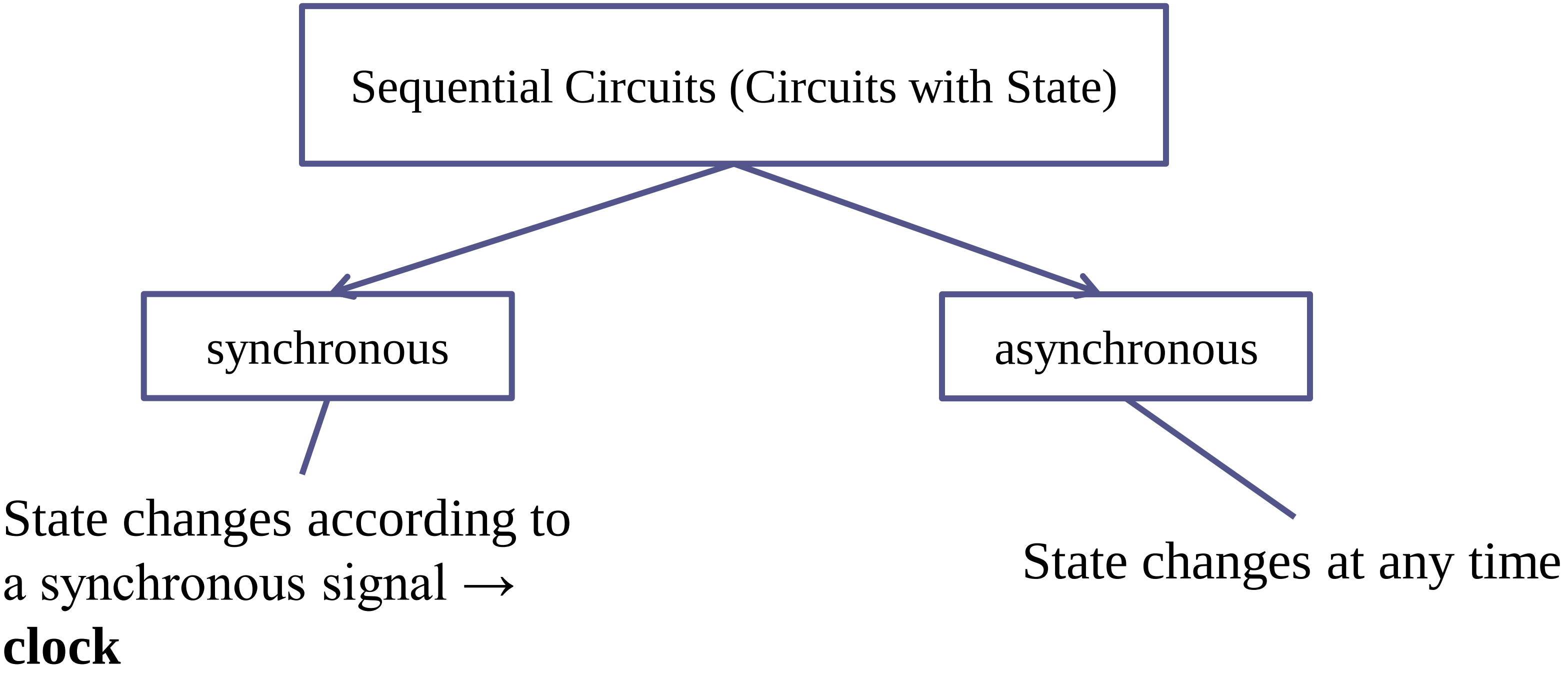


Introduction



- **State**: Binary information stored in the **memory element** at any given time. State of a squential circuit is a set of bits that contain the information about the **past** necessary to explain the **future behaviour**.
- Present state+external inputs→ outputs+next state
- **Q** is commonly used for the **output** of sequential logic

Introduction



Sequential Circuits (Circuits with State)

synchronous

asynchronous

State changes according to a synchronous signal → **clock**

State changes at any time

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

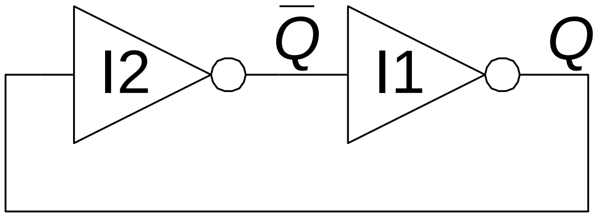
feedback

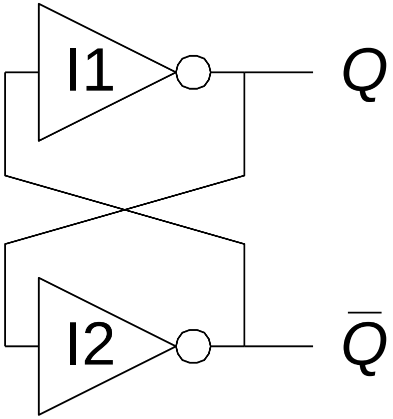
State Elements

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

Bistable Circuit

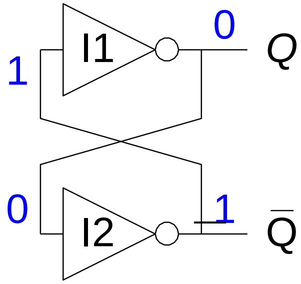
- Fundamental building block of other state elements
- Two outputs: Q, $\overline{Q}$
- No inputs



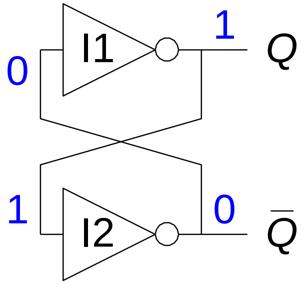


Bistable Circuit Analysis

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

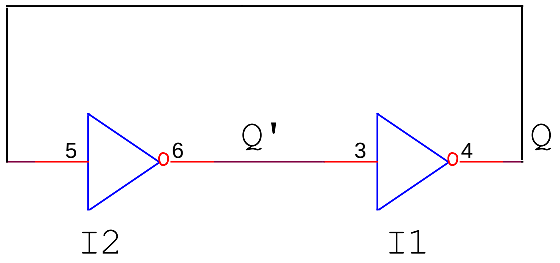


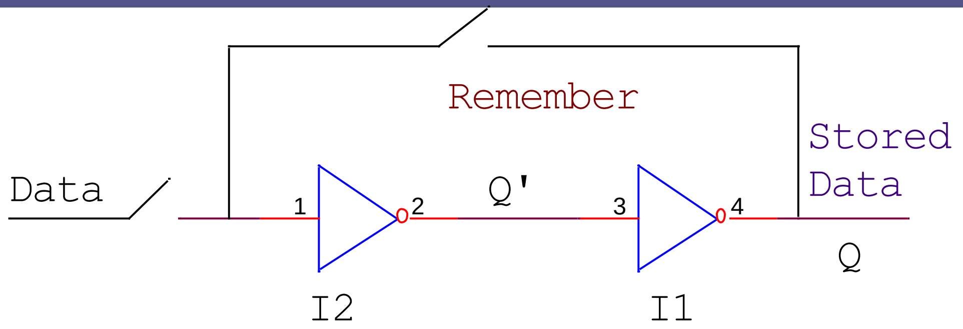
  - ***Q* = 1:**
    then $\overline{Q} = 0$, $Q = 1$ (consistent)



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

Bistable Circuit Analysis





- □ Fundemental building block of memory. Bistable element with two stable states.
- □ will hold value as long as it has power applied
- □ selectively break feedback path
- □ load new value into memory cell
- □ Since inveters have two stable states the circuit is said to e bistable
- □ Bistable circuit stores one bit and remains as long as power applied
- □ Bistable circuits with cross-couple inverters are not practical since user has no inputs to control its state
- □ A bistable circuit stores one bit data Q=0 or Q=1 will remain until power-off

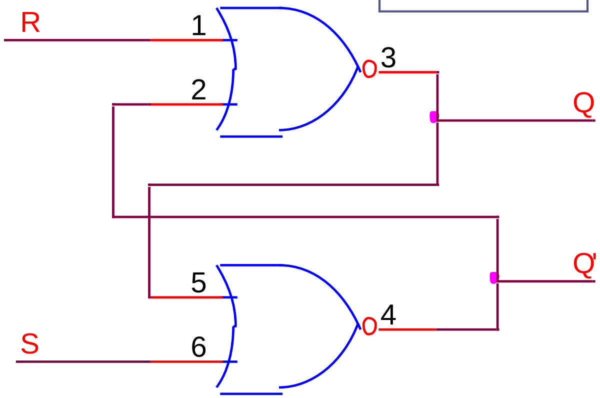
SR (Set/Reset) Latch

- Two cross-coupled NOR gates used for SR latch
- Two Inputs **S** (Set) – **R** (Reset)
- Outputs Q and its complement Q’
- Output is controlled by S and R inputs

SR - NOR GATES!!

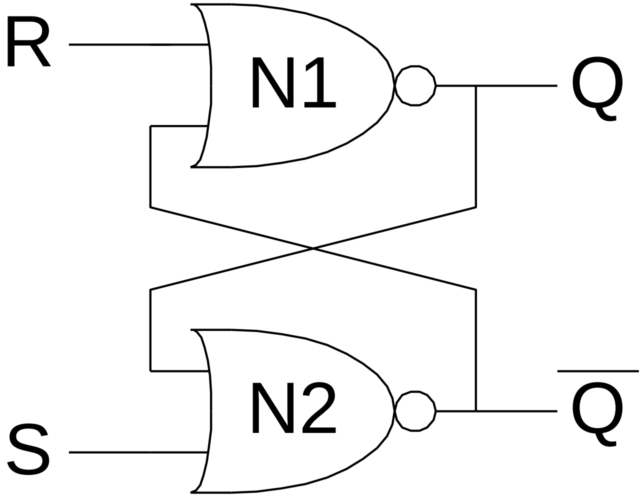
A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0

- Case I: R=1, S=0 >> Q=0, Q’=1
- Case II: R=0, S=1 >> Q=1, Q’=0
- Case III: R=1, S=1 >> Q=0, Q’=0
- Case IVa: R=0, S=0, Q=0 >> Q’=1, Q=0
- Case IVb: R=0, S=0, Q=1 >> Q’=0, Q=1



SR (Set/Reset) Latch

- SR Latch

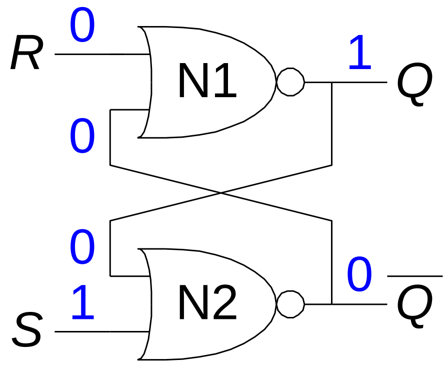


- Consider the four possible cases:
  - $S = 1, R = 0$
  - $S = 0, R = 1$
  - $S = 0, R = 0$
  - $S = 1, R = 1$

SR Latch Analysis

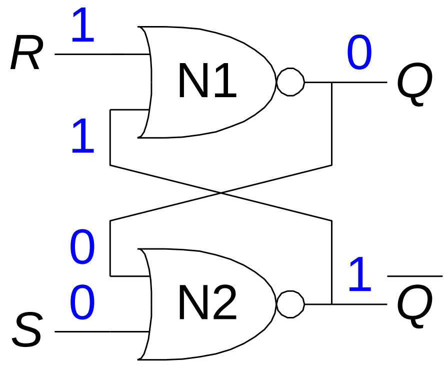
– ***S* = 1, *R* = 0:**

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



– ***S* = 0, *R* = 1:**

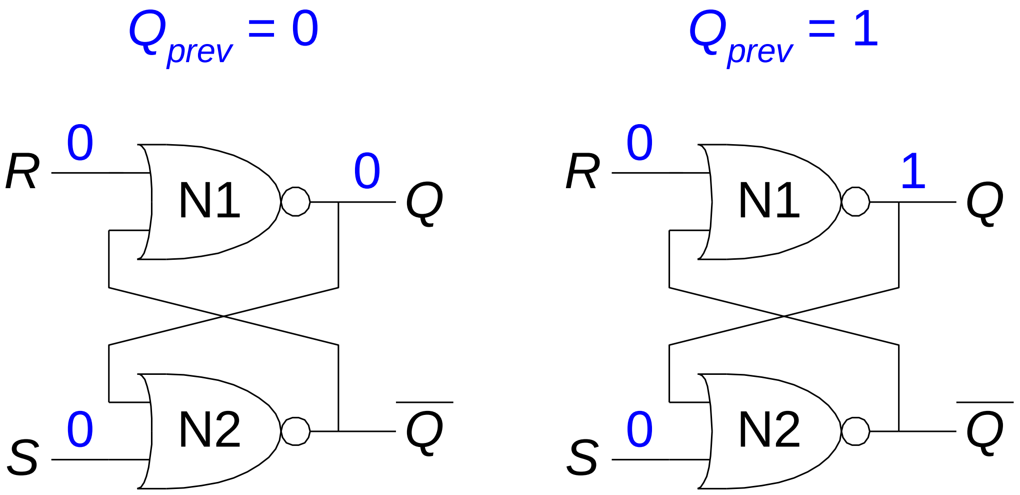
then $\overline{Q} = 1$ and $Q = 0$



SR Latch Analysis

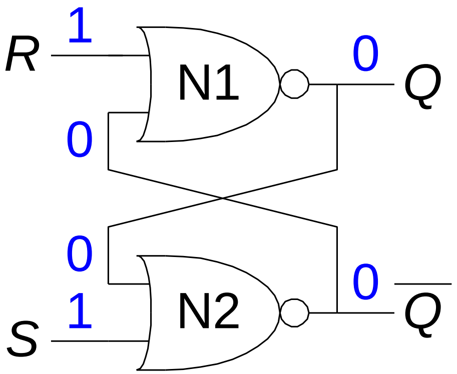
- ***S* = 0, *R* = 0:**

then $Q = Q_{prev}$



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

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

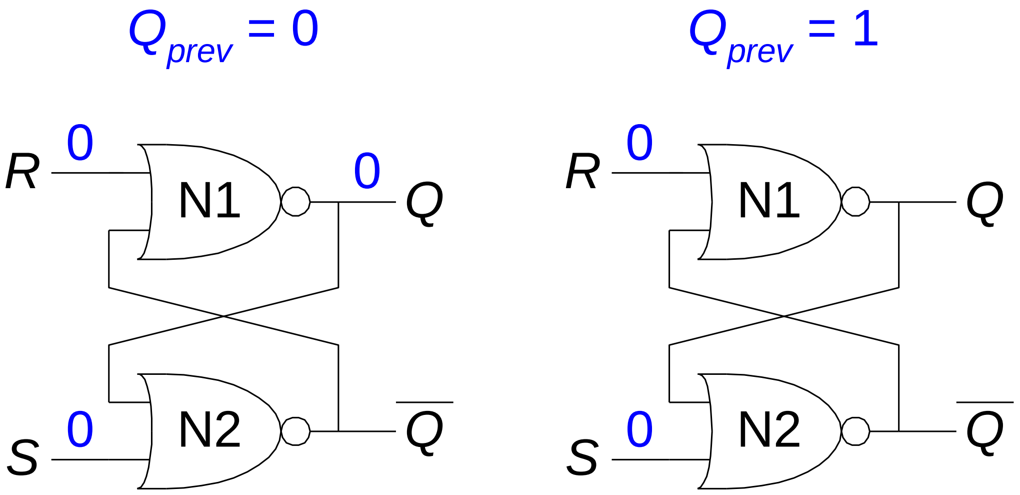


SR Latch Analysis

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

  then $Q = Q_{prev}$

- **Memory!**

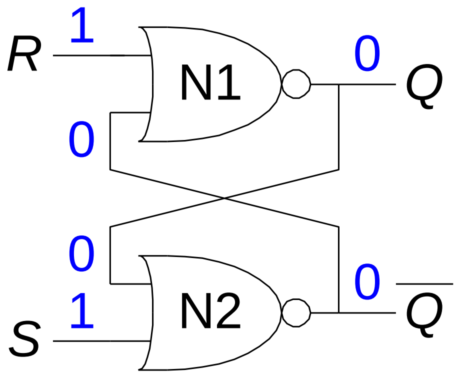


- ***S* = 1, *R* = 1:**

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

- **Invalid State**

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



ELSEVIER

SR Latch Analysis

S	R	Q (State)	
0	0	No change	
0	1	Reset	Q=0;Q'=1
1	0	Set	Q=1;Q'=0
1	1	NA	

S	R	Q_{t+1}
0	0	Q_t (Previous State)
0	1	Reset
1	0	Set
1	1	NA

- similar to inverter pair, with capability to force output to 0 (reset=1) or 1 (set=1)
- To set the output assert S (S=1)
- When R is asserted output becomes FALSE
- When both inputs are FALSE output remembers its previous state.
- When both inputs are TRUE latch will be SET and RESET at the same time (impossible). Hence both outputs are FALSE.

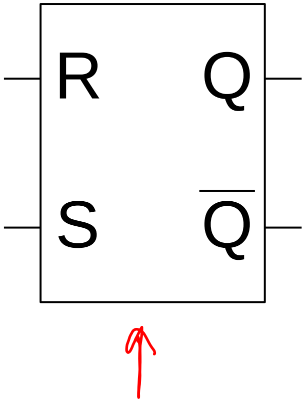
SR Latch Symbol

- 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 = **1**)
  - **Reset:** Make the output 0

    (S = 0, R = 1, Q = **0**)

SR Latch Symbol

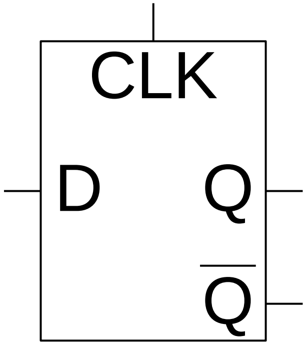


D Latch

- 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

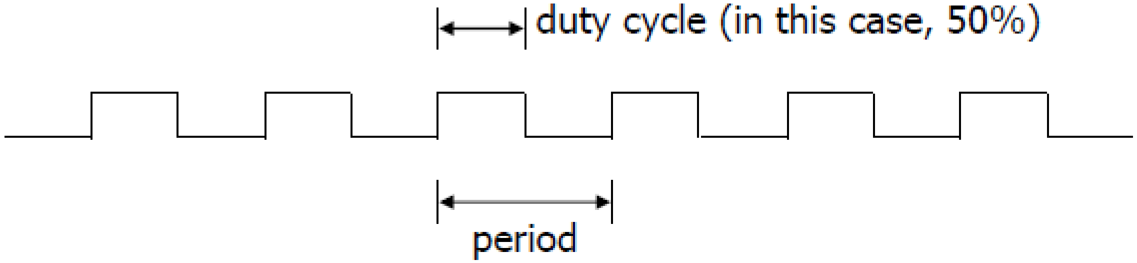
$Q \neq \text{NOT } \overline{Q}$

D Latch Symbol

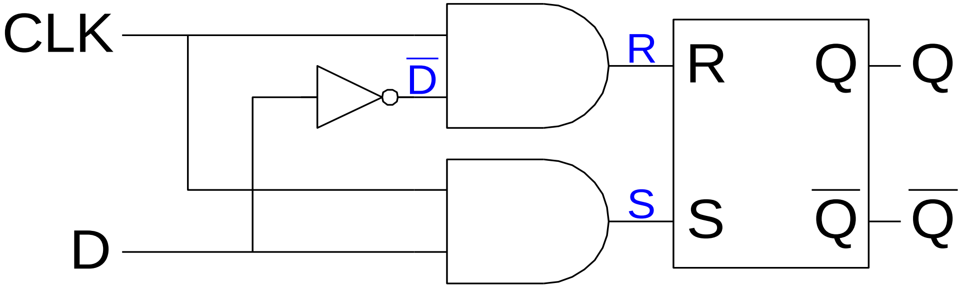


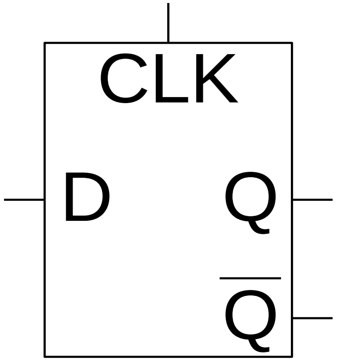
Clocks

- Used to keep time
  - wait long enough for inputs to settle
  - then allow to have effect on value stored
- Clocks are regular periodic signals
  - **period** (time between ticks)
  - **duty-cycle** (time clock is high between ticks - expressed as % of period)



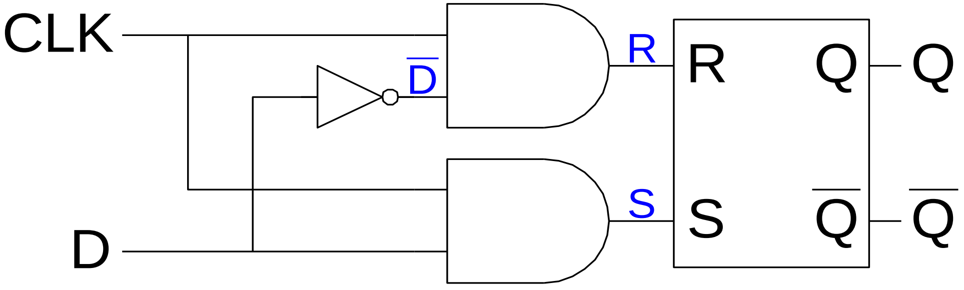
D Latch Internal Circuit

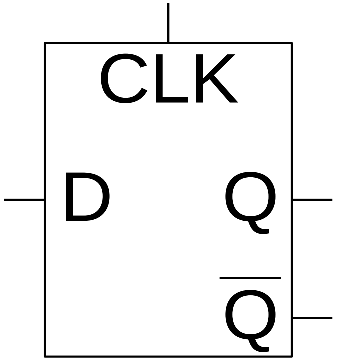




CLK	D	$\overline{D}$	S	R	Q	$\overline{Q}$
0	X	X	0	0	Prev	Prev
1	0	1	0	1	0	1
1	1	0	1	0	1	0

D Latch Internal Circuit



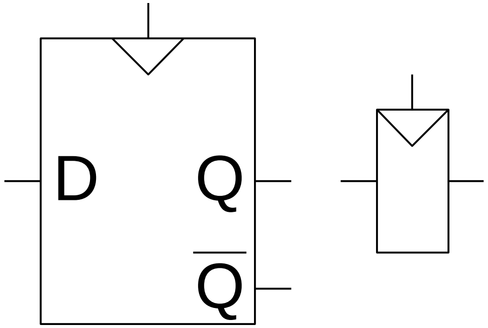


CLK	*D*	$\overline{D}$	*S*	*R*	*Q*	$\overline{Q}$
0	X	$\overline{X}$	0	0	Q_{prev}	$\overline{Q}_{prev}$
1	0	1	0	1	0	1
1	1	0	1	0	1	0

D Flip-Flop

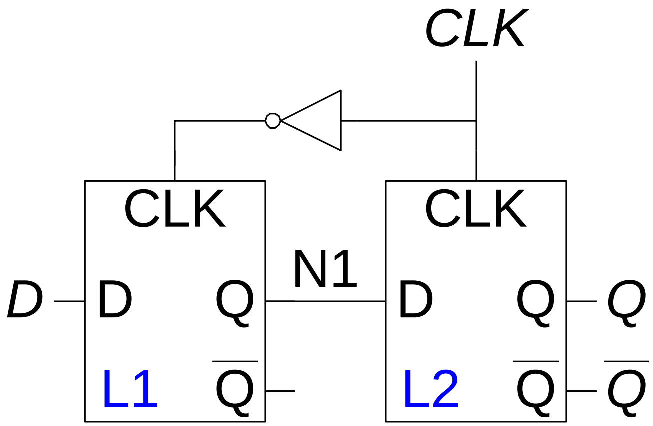
- **Inputs:** *CLK*, *D*
- **Function**
  - Samples *D* on rising edge of *CLK*
    - When *CLK* rises from 0 to 1, *D* passes through to *Q*
    - Otherwise, *Q* holds its previous value
  - *Q* changes only on rising edge of *CLK*
- Called *edge-triggered*
- Activated on the clock edge

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*



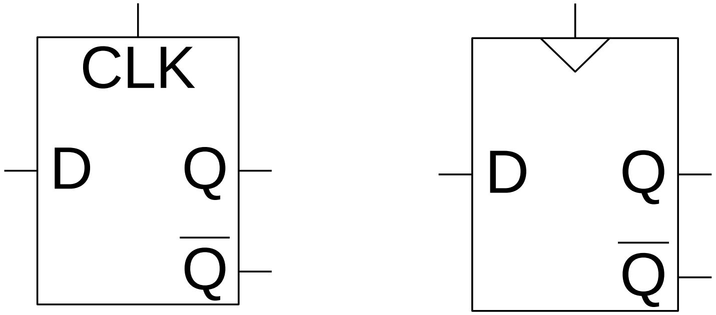
- Thus, on the edge of the clock (when ***CLK* rises from 0 → 1**)
  - *D* passes through to *Q*

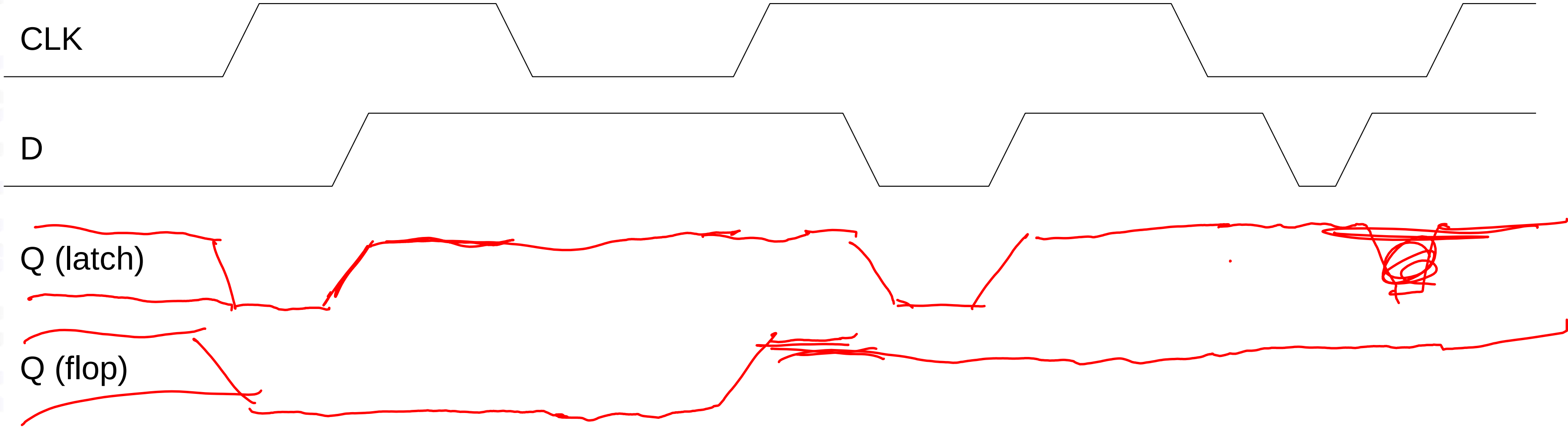
D Latch vs. D Flip-Flop (FF)

- D **FF** is also called master-slave Flip-Flop.
- D **FF** is edge triggered FF, triangle on the symbol denotes edge triggered input CLK.

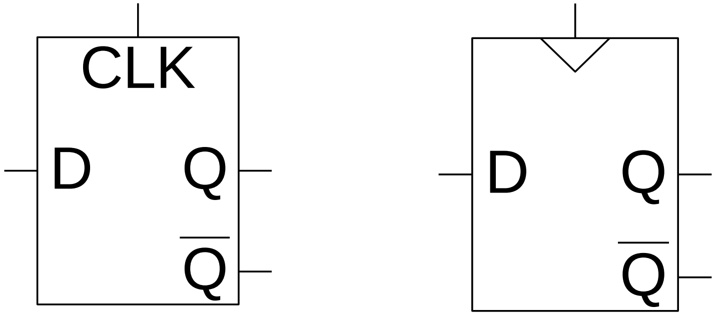
Latches are building blocks of sequential circuits which can be built from logic Gates.	Flip Flops are building blocks of sequential circuits which can be built from latches. ✓
Latch continuously checks its inputs and changes its output correspondingly	Flip Flop continuously checks its inputs and changes its output correspondingly only determined by clock signal ✓
Latch is based on clock(enable) input.	Flip Flop Works with clock pulses ✓
The latch's clock (enable) input is level sensitive, meaning the latch's output changes on the level (high or low) of the EN input.	The flip-flop's CLOCK input is edge sensitive, meaning the flip-flop's output changes on the edge (rising or falling) of the CLOCK input ✓

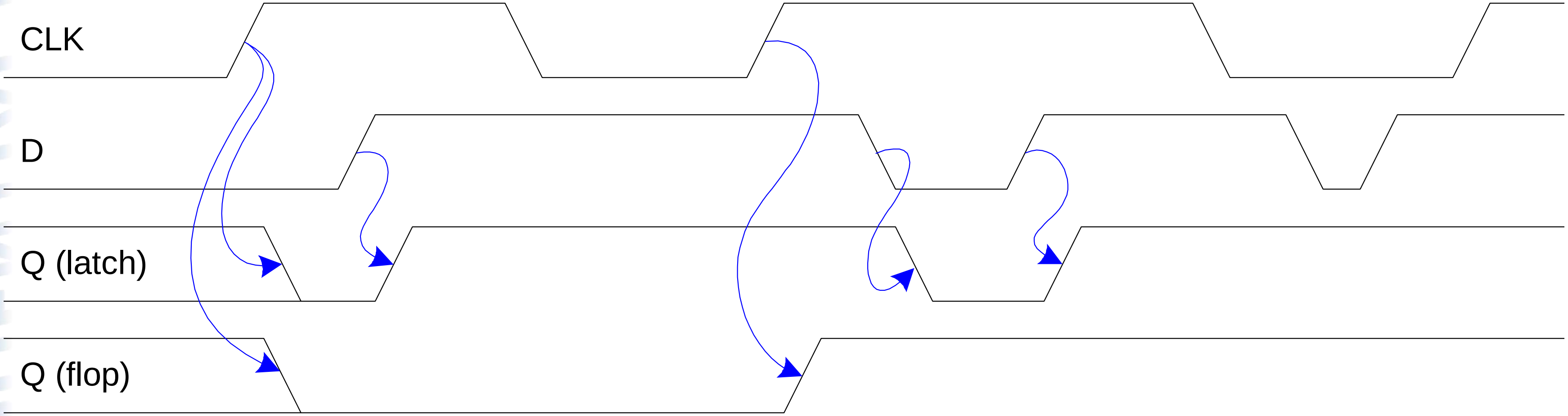
D Latch vs. D Flip-Flop





D Latch vs. D Flip-Flop

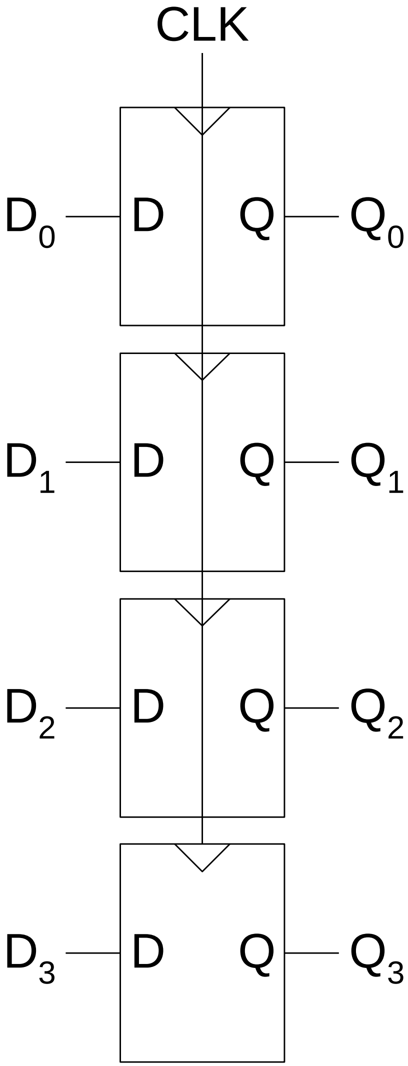


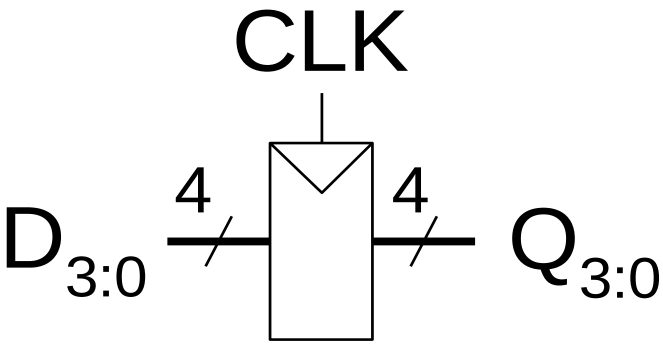


D Latch vs. D Flip-Flop

- Registers can be built by using flip-flops.
- N bit register can be formed by using N flip-flops
- All bits of the register will be updated with a **shared** clock signal.

Registers



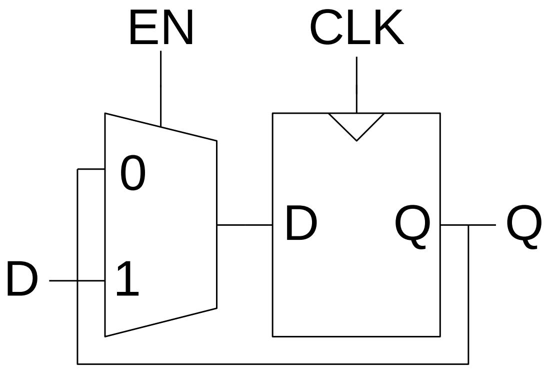


ELSEVIER

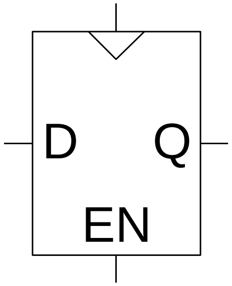
Enabled Flip-Flops

- **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

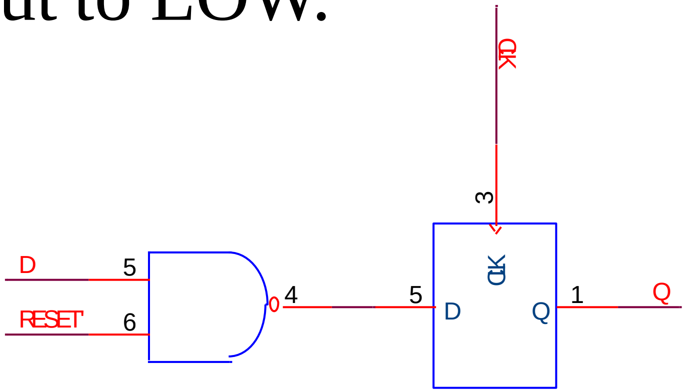


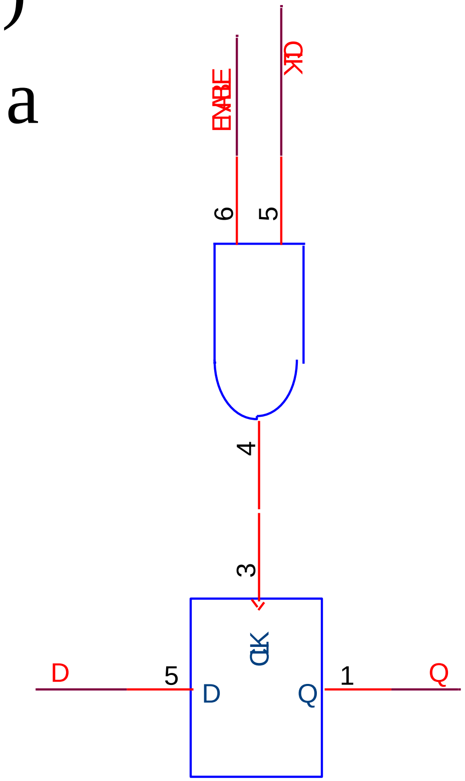
Symbol



Enabled-Resettable Flip-Flops

- □ Enabled FF has another input (EN)
- □ FF can be enabled by either using a Mux or an AND gate
- □ When reset is low FF resets the output to LOW.

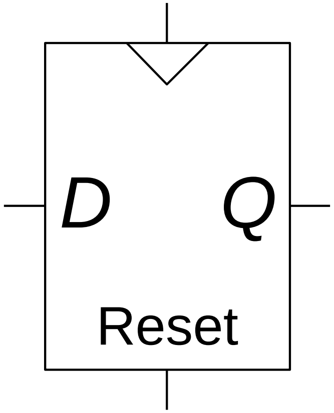


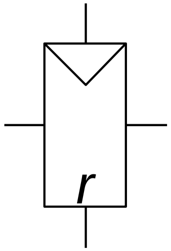


Resettable Flip-Flops

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

Symbols





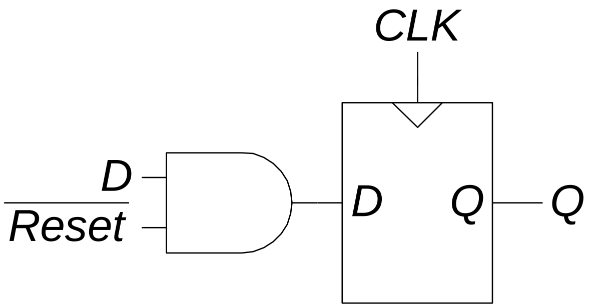
Resettable Flip-Flops

- 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?

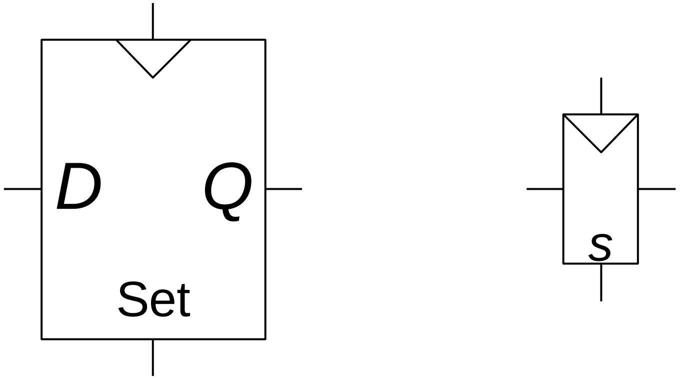
Internal Circuit



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



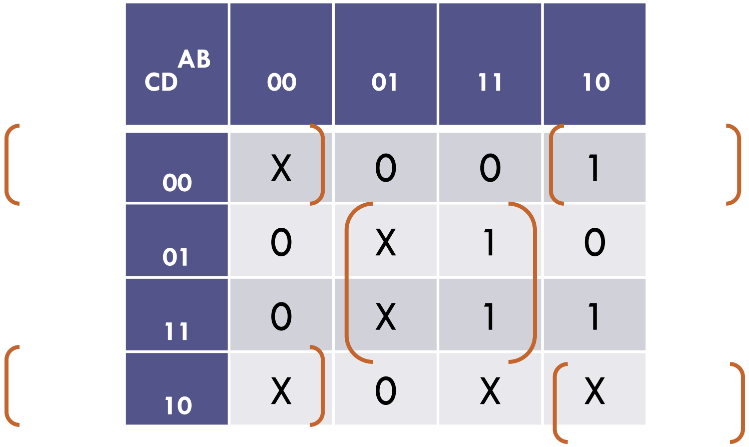
Review$_{10}$

- Sequential Circuits
- Bistable Circuit
- SR (Set/Reset) Latch
- D Latch
- D Flip-Flop
- Registers
- Enabled Flip-Flops
- Resettable/Settable Flip-Flops

Review$_{10}$

- Find a minimal Boolean equation for the function given with the below truth table. Remember to take advantage of the don’t care entries.

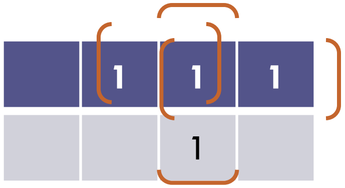
A	B	C	D	Y
0	0	0	0	X
0	0	0	1	0
0	0	1	0	X
0	0	1	1	0
0	1	0	0	0
0	1	0	1	X
0	1	1	0	0
0	1	1	1	X
1	0	0	0	1
1	0	0	1	0
1	0	1	0	X
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	X
1	1	1	1	1



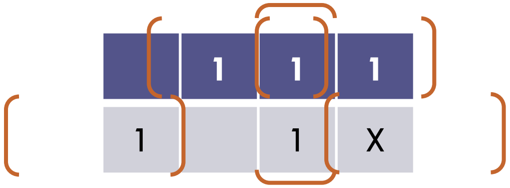
CD \ AB	00	01	11	10
00	X	0	0	1
01	0	X	1	0
11	0	X	1	1
10	X	0	X	X

Review$_{10}$

- Given the k-map for function F(A,B,C), Find a minimal Boolean equation for the function



	1	1	1
		1	



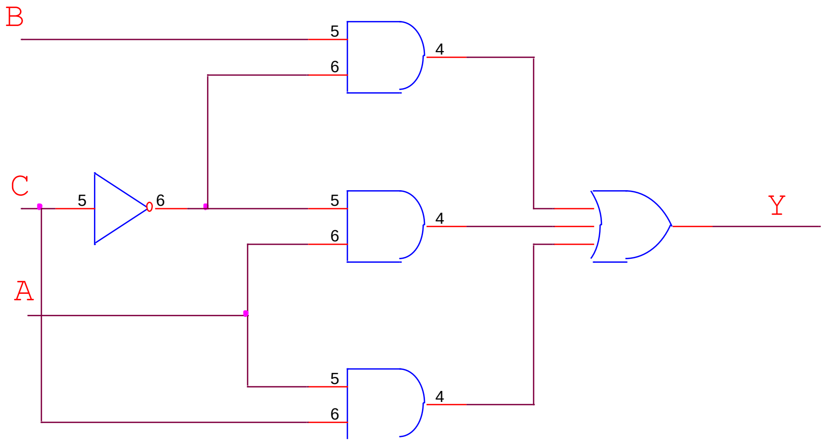
	1	1	1
1		1	X

$$F(A,B,C) = \sum(2,4,7,0,3)$$

- Can we implement this function by using 4:1 mux.

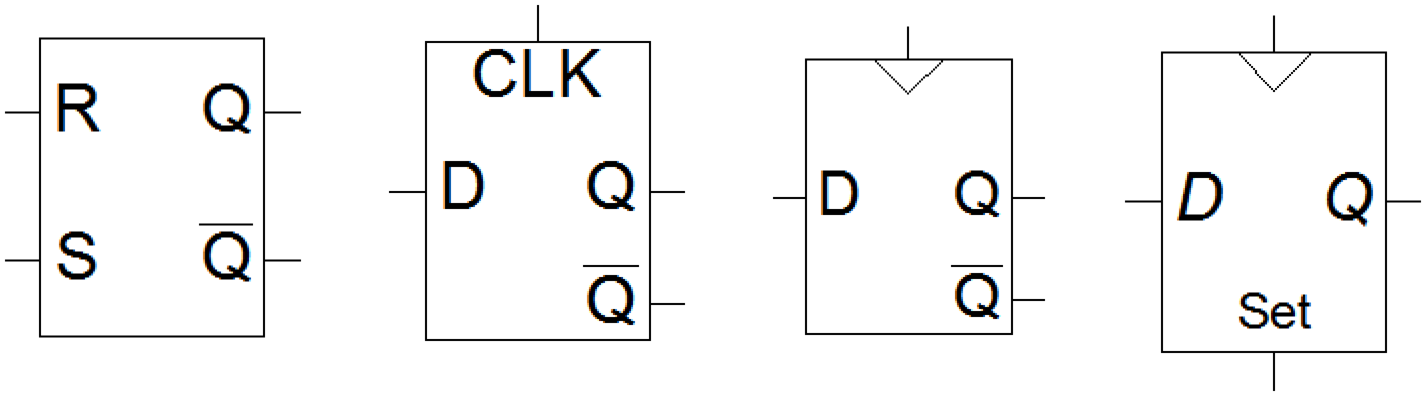
Review_{10}

- □ a. Write the Boolean equation for Y (using the schematic below). Update the following circuit by using only 2-input NAND gates (as many as needed).
- □ b. Implement this logic function using only a single 4:1 multiplexer



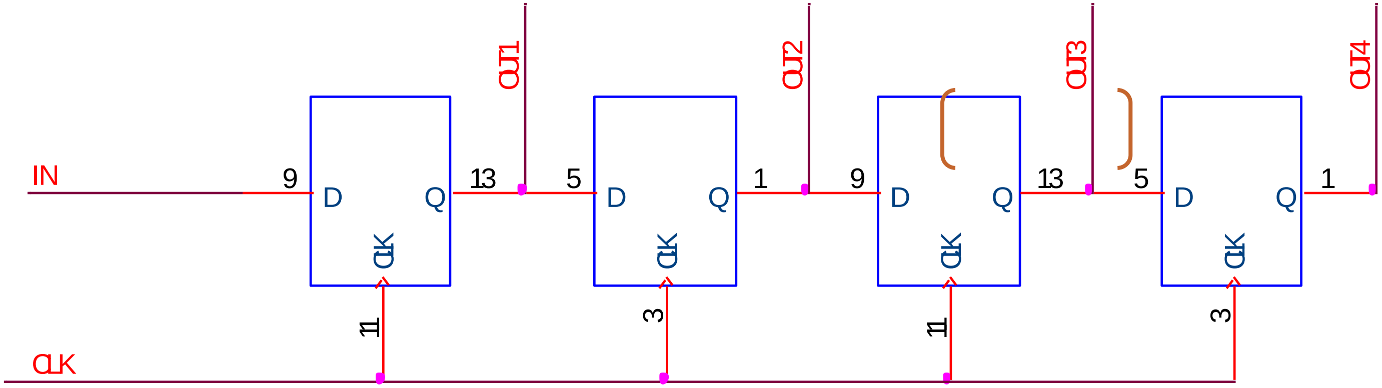
Review$_{10}$

- Main building blocks



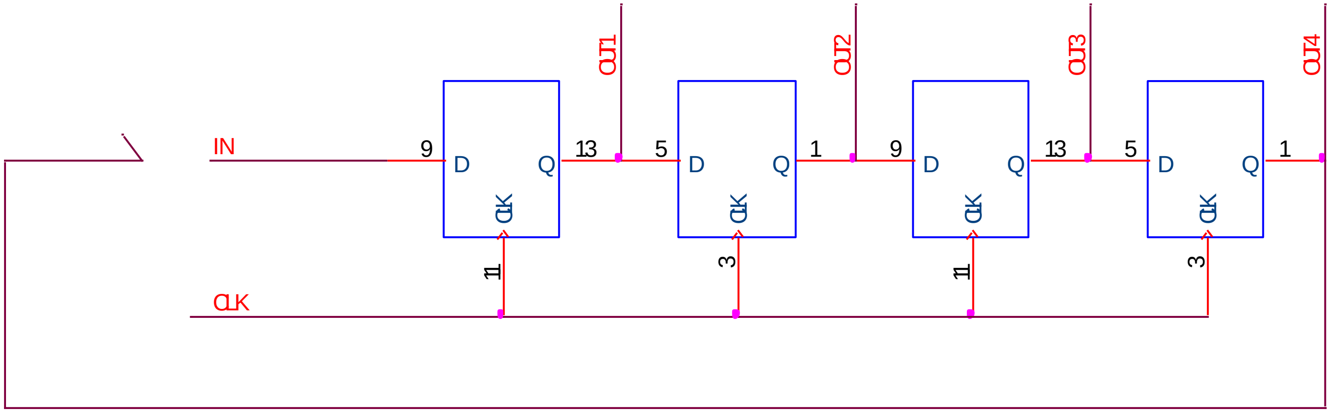
Review$_{10}$

- Shift Register
  - Holds samples of input
    - store last 4 input values in sequence
    - 4-bit shift register



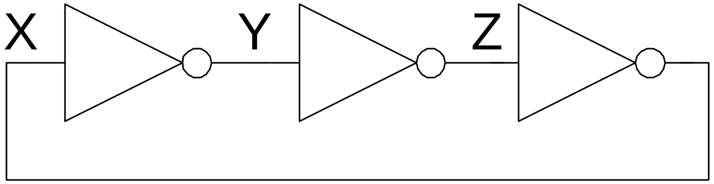
Review$_{10}$

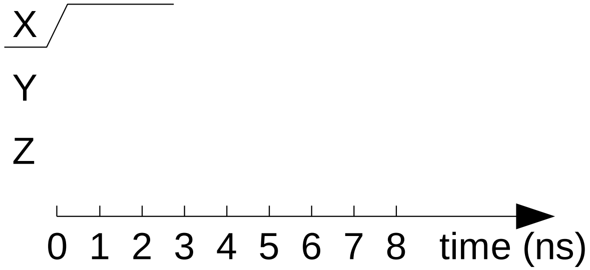
- Counter
  - Sequences through a fixed set of patterns
    - in this example, 1000, 0100, 0010, 0001
    - if one of the patterns is its initial state (by loading or set/reset)



Sequential Logic

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

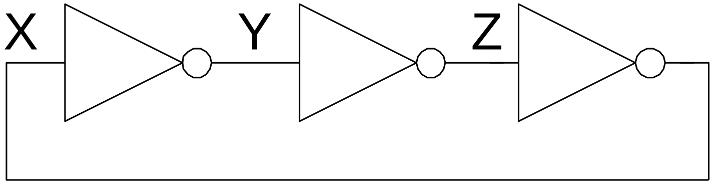


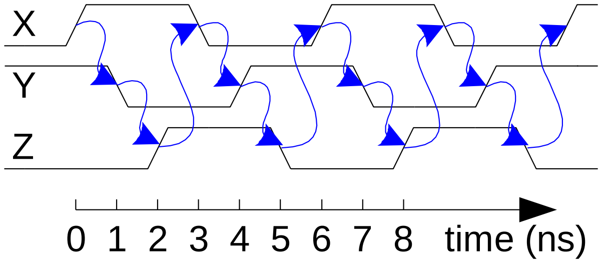


- If a circuit is not combinational then it is called sequential circuit.
- This circuit is called unstable or astable circuit
- Suppose X=0 initially, each inverter has 1nsec delay.

Sequential Logic

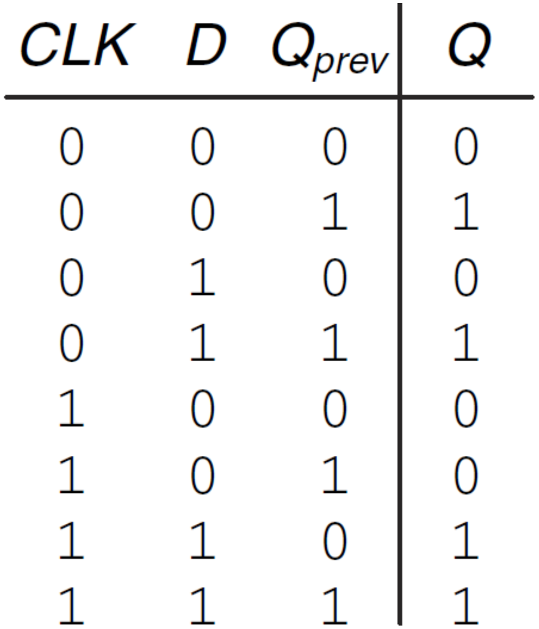
- Sequential circuits: all circuits that aren’t combinational
- A problematic circuit:





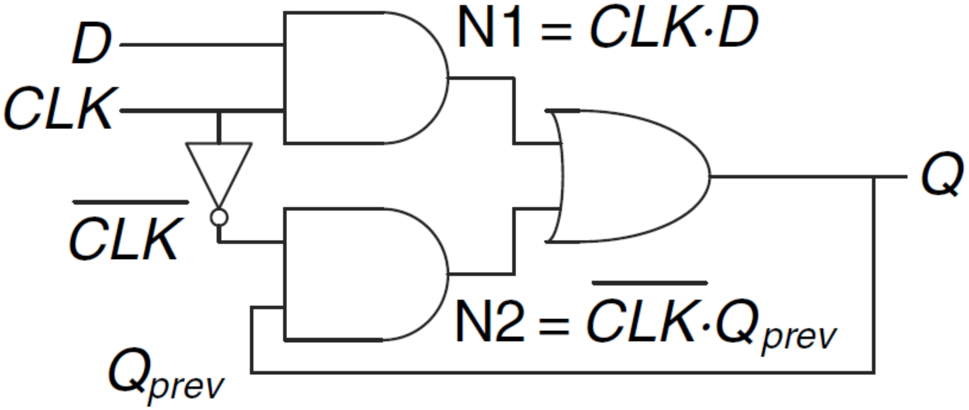
- No inputs and 1-3 outputs
- Astable circuit, oscillates
- Period depends on inverter delay
- It has a ***cyclic path***: output fed back to input

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

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

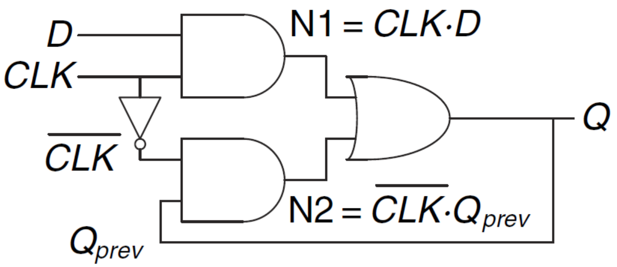


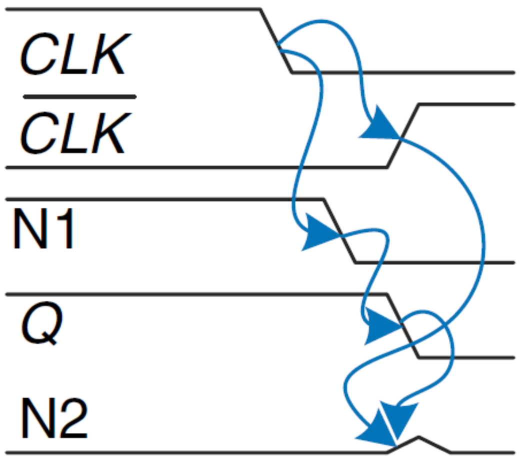
Hint: Assume the inverter delay is very large!

Suppose CLK=D=1 initially so Q=1, When CLK signal falls to 0

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!

N1 and Q may fall before CLK’ rises, then N2 will never rise and Q becomes stuck

Asynchronous Circuit Design

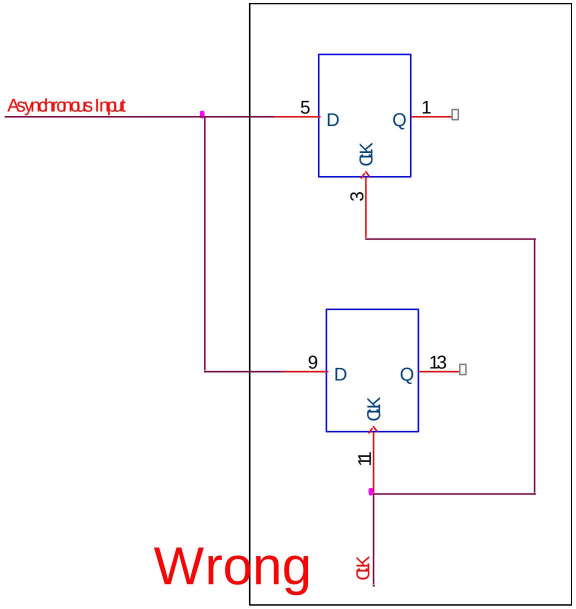
- **Outputs** directly fed **back** to inputs.
- Potential **race** conditions!
  - Behavior of the circuit depends on which path is **fastest**
- Seemingly identical circuits but with **different gate delays** may lead to different functionalities.
- Circuit may only work at certain **temperatures**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

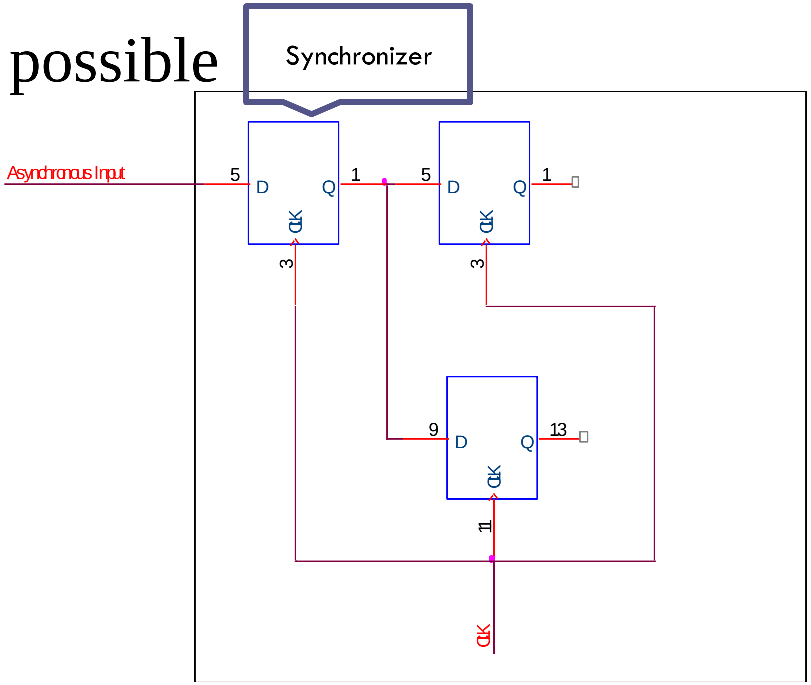
Synchronous Sequential Circuits

- Registers keep the state of the system which changes **only** at the **clock** edge.
- Inputs+Outputs+Function+Timing defines the circuit, sequential circuits:
  - Has a **finite** set of **states**
  - Synchronous Sequential Circuit has a **clock** input
  - With **rising edge** of clock **state changes** (current state and next state definitions will be used)

Synchronous Sequential Circuits

- Handling asynchronous inputs
  - Never allow asynchronous inputs to fan-out to more than one flip-flop
  - synchronize as soon as possible



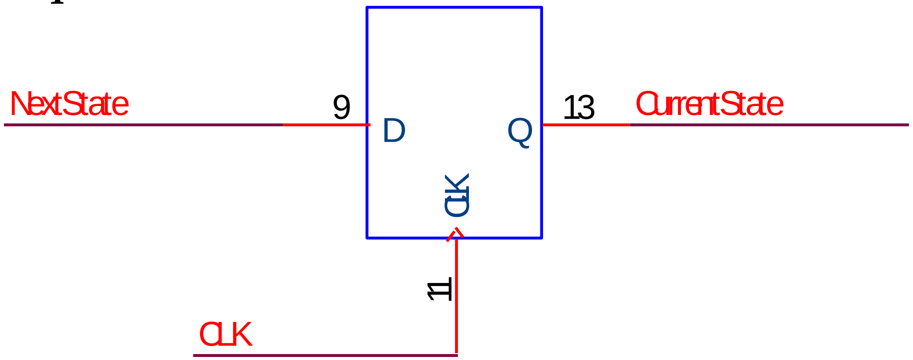


Synchronous Sequential Logic Design

- 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

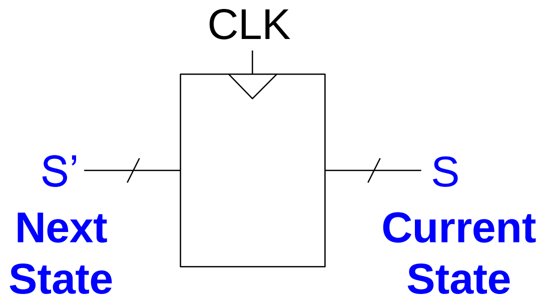
Synchronous Sequential Circuits

- Sequential circuits that does not satisfy these rules are called **asynchronous**
- Flip-Flop is a synchronous sequential circuit.
  - Input, D
  - Clock, CLK
  - Output, Q
  - States, 0 and 1

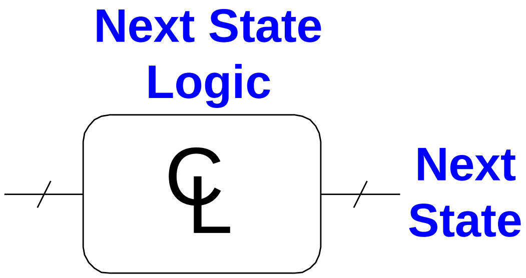


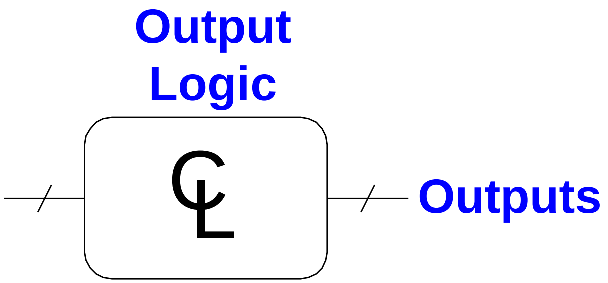
- **Current state** will be expressed by the letter **S**
- **Next state** will be expressed by the letter **S'** (Prime indicates next state, **not inversion**)

Finite State Machine (FSM)

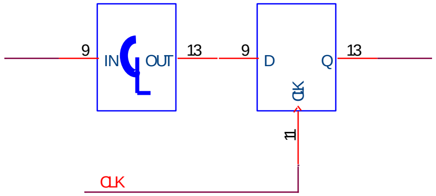


- Consists of:
  - – **State register**
    - Stores current state
    - Loads next state at clock edge
  - – **Combinational logic**
    - Computes the next state
    - Computes the outputs (it has no register)

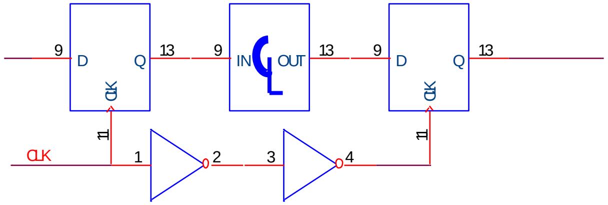




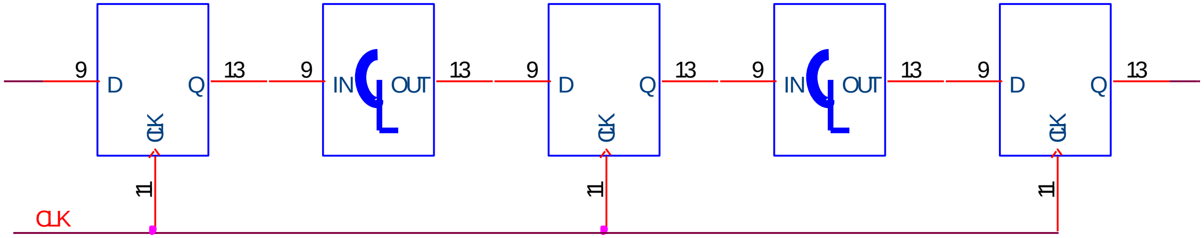
Finite State Machine (FSM)



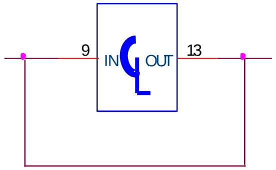
Sequential circuit
with no feedback



Not a Synchronous Sequential circuit



Synchronous Sequential circuit in the form of a pipeline



Asynchronous since no register on cyclic path

Finite State Machine (FSM)

- Asynchronous design is more general than synhronous design
- Asynchronous circuits are necessary when communicating between systems with different clocks, or when receiving inputs at arbitrary times.
- Synhronous circuits are **easier** to design and use than asynchronous circuits
- Synchronous sequential circuits can be drawn in the forms of Finite State Machines
- A circuit with k registers can have 2^k unique states.
- Finite State Machine (FSM) has
  - M **inputs**
  - N **outputs**
  - K bits of **state**
  - **Clock** Signal

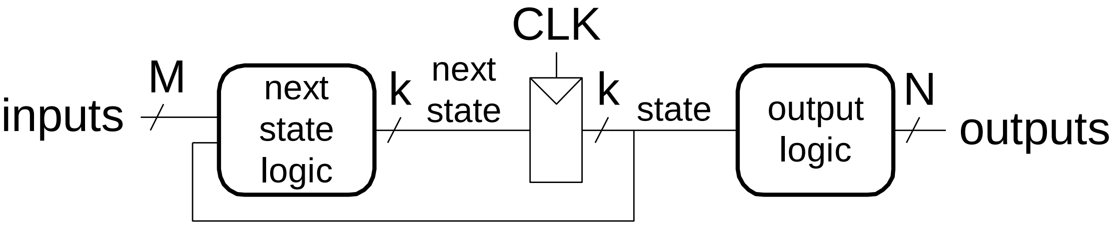
Finite State Machine (FSM)

- FSM consists combinational logic blocks,
  - **Next state** logic (combinational logic block-1)
  - **Output** logic (combinational logic block-2)
  - state **register**
- On each **clock edge** FSM advances **next state**
- FSM provide a systematic way to design synchronous sequential circuit.

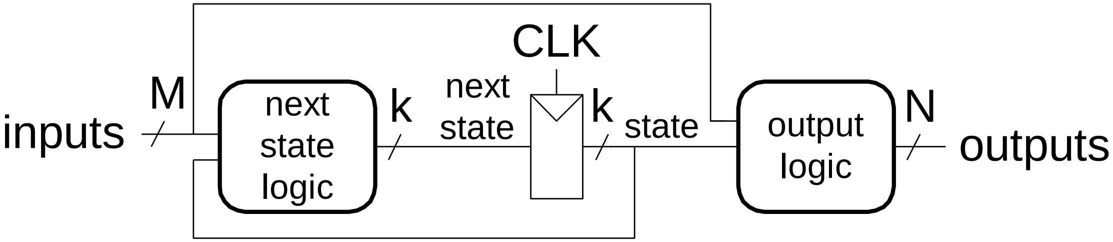
Finite State Machines (FSMs)

- 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



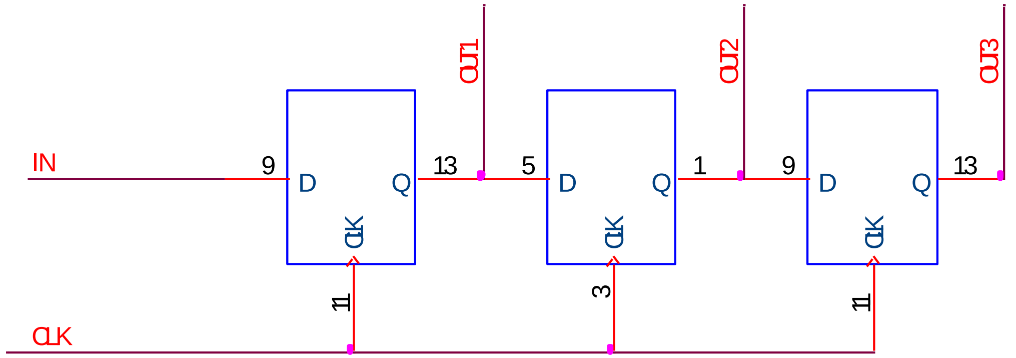
Mealy FSM

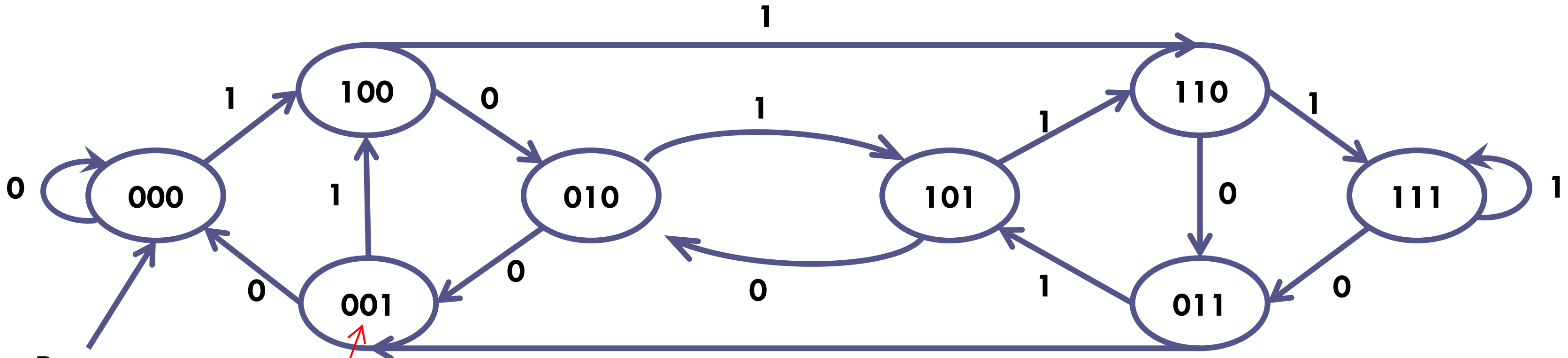


Finite State Machine (FSM)

- **States:** determined by possible values in sequential storage elements
- **Transitions:** change of state
- **Clock:** controls when state can change by controlling storage elements
- Sequential logic
  - **sequences** through a series of states
  - based on sequence of values on input signals
  - clock **period** defines elements of **sequence**

State Transition Diagrams





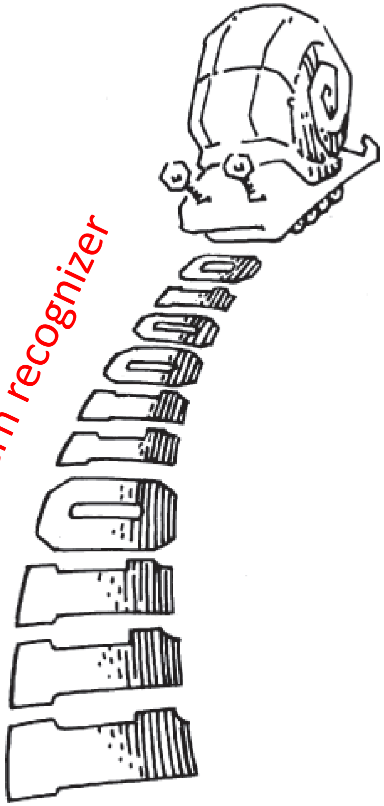
Reset

Outputs

Inputs

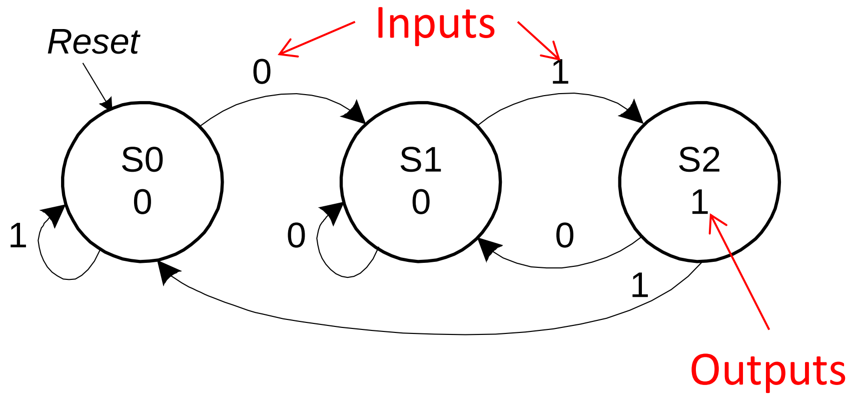
Moore vs. Mealy FSM

- 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.

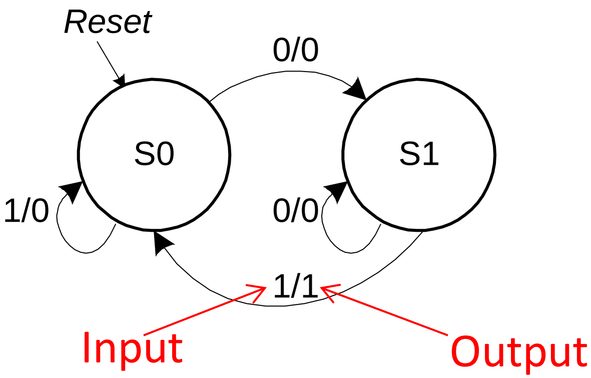


State Transition Diagrams

Moore FSM

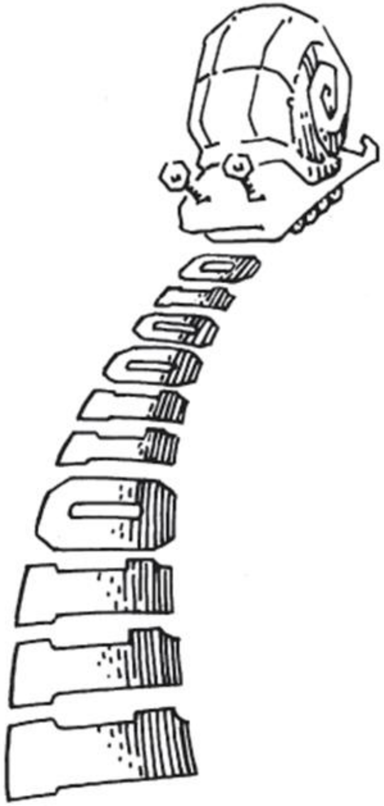


Mealy FSM

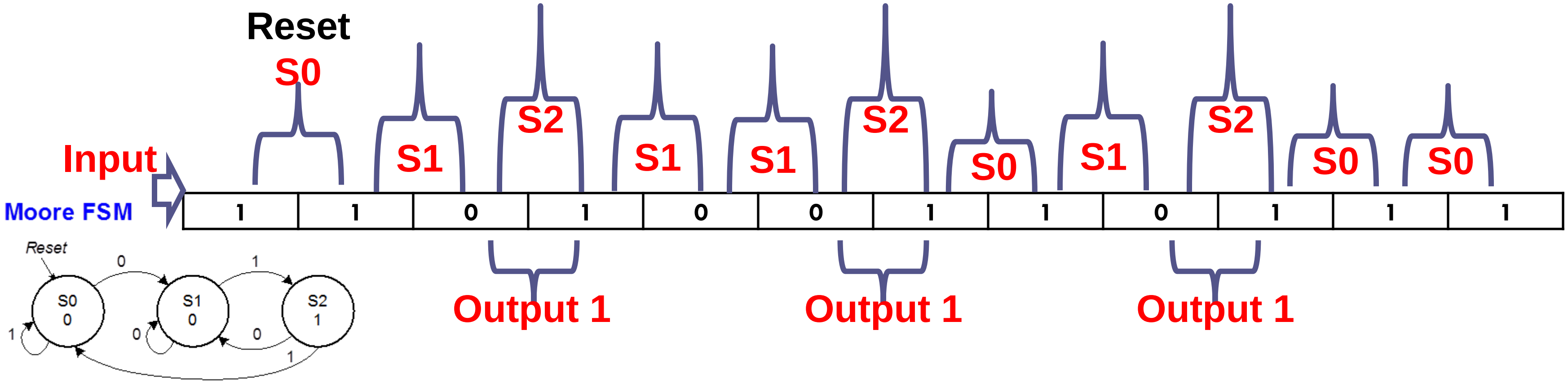


Mealy FSM: arcs indicate input/output

Path
0
1
0
0
1
1
0
1
1
1

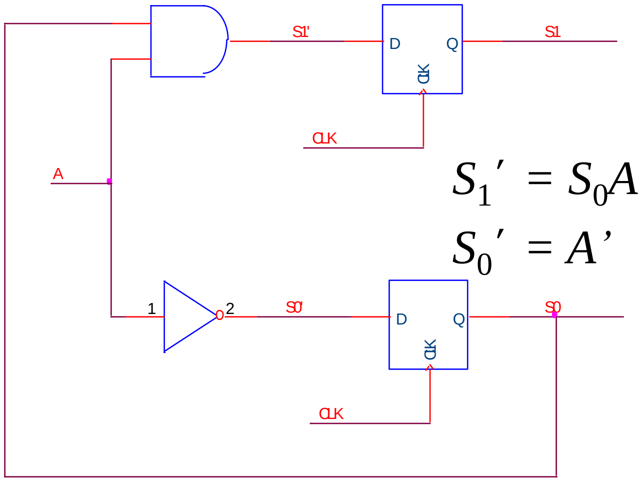


State Transition Diagrams



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

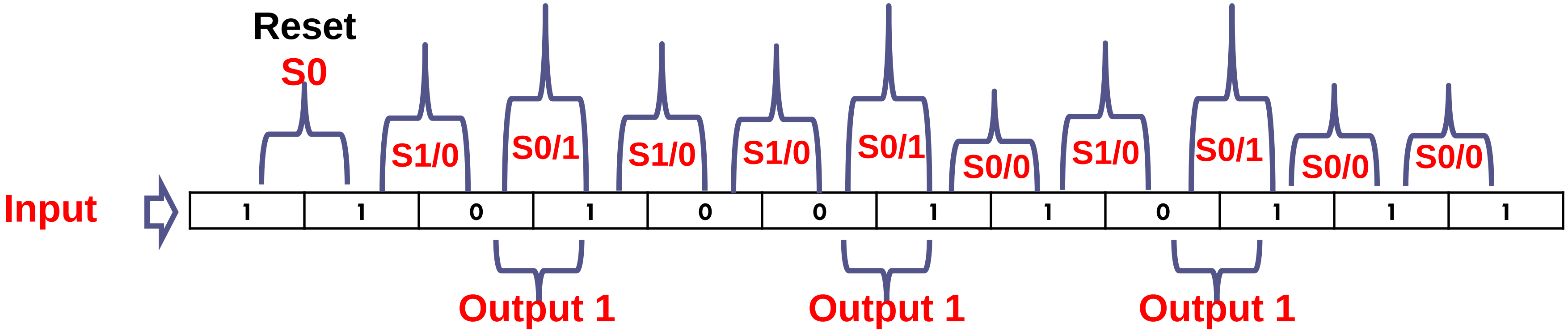
State transition table



$$S_1' = S_0A$$

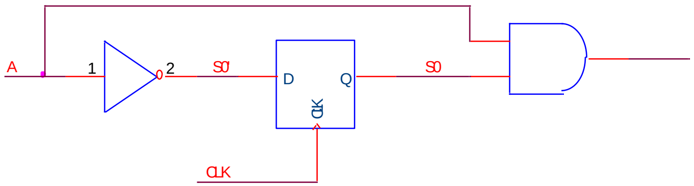
$$S_0' = A'$$

State Transition Diagrams

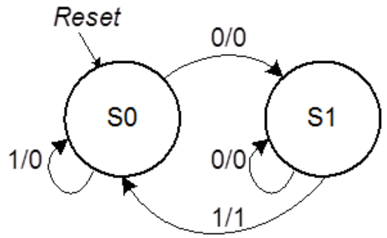


Current State	Input	Next State	Output
S_0	A	S'_0	Y
0	0	1	0
0	1	0	0
1	0	1	0
1	1	0	1

State transition table



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 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

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

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

$$Y = S_1$$

Review$_{11}$

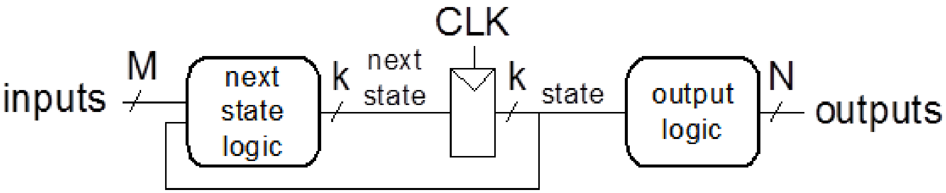
- □ Synchronous Sequential Circuits
  - Every circuit element is either a register or a combinational circuit
  - At least one circuit element is a register
  - All registers receive the same clock signal
  - Every cyclic path contains at least one register
- □ Two common synchronous sequential circuits
  - Finite State Machines (FSMs)
  - Pipelines
- □ Finite State Machine (FSM)
- □ State Transition Diagrams

1st Midterm

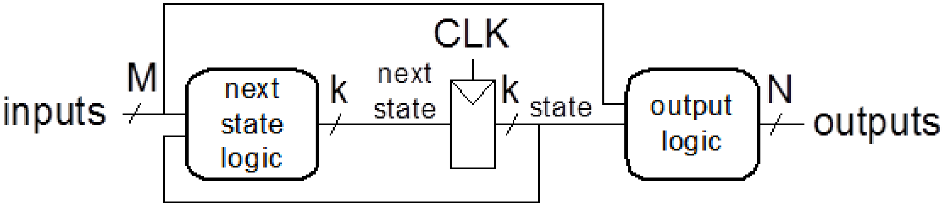
14/11/2018

17:30

Moore FSM



Mealy FSM

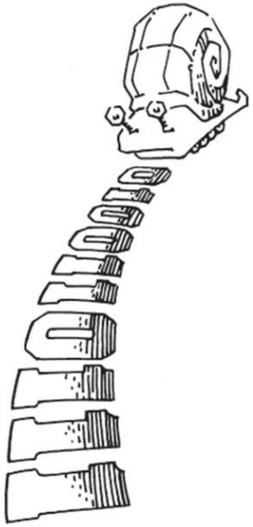


Review_{11}

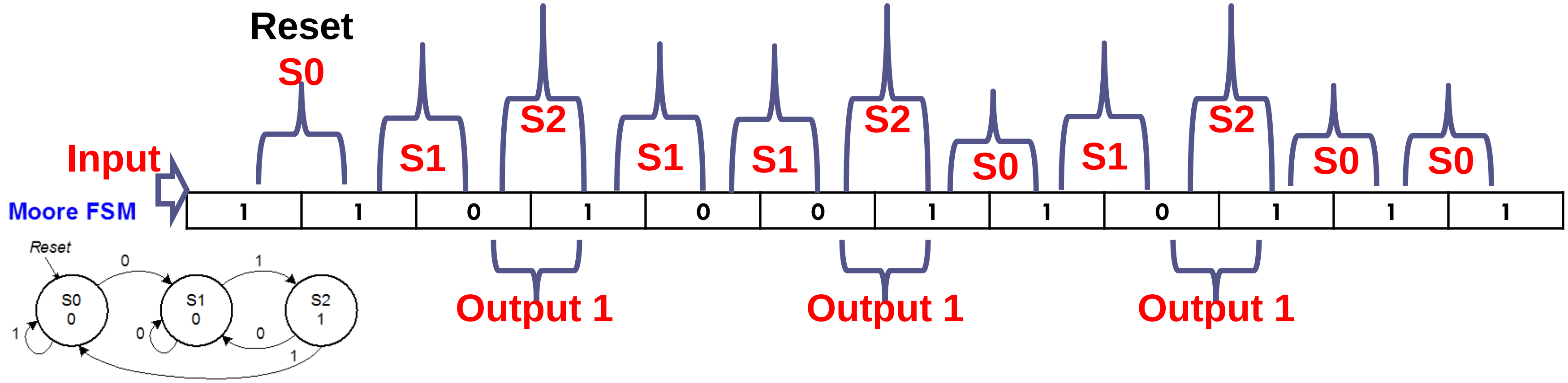
- Identify the inputs and outputs
- Sketch the state transition diagram
- Write state transition table
- Write output table
  - Write combined state transition table + output table (for mealy FSM)
- Select state encoding
- Write boolean equations for next state and output logic
- Sketch the circuit schematic

Review_{11}

- 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

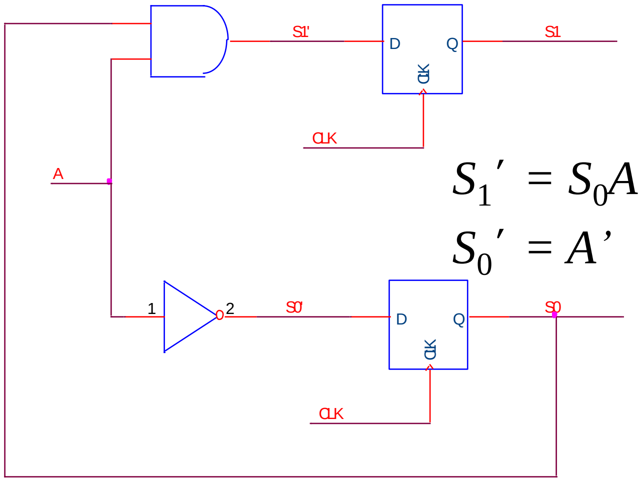


State Transition Diagrams



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

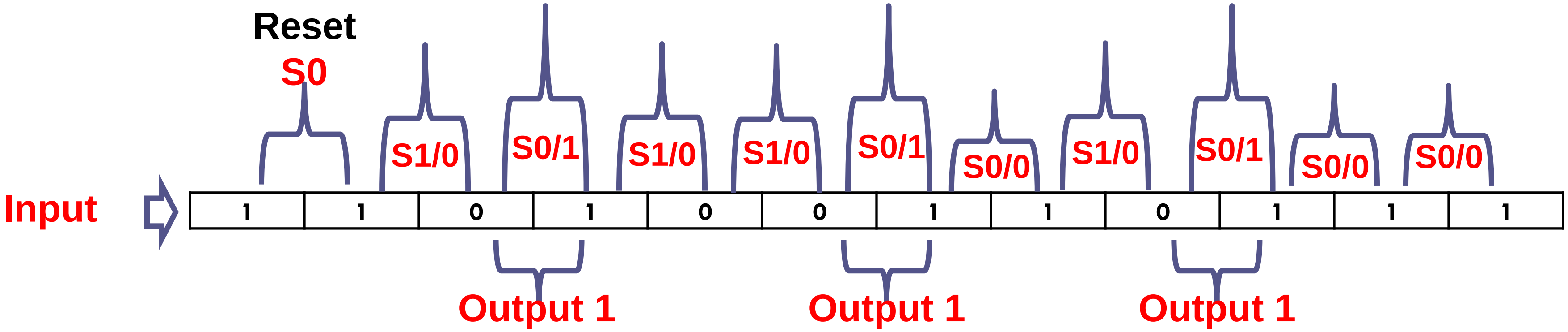
State transition table



$$S_1' = S_0 A$$

$$S_0' = A'$$

State Transition Diagrams



Reset

S0

S1/0 S0/1 S1/0 S1/0 S0/1 S0/0 S1/0 S0/1 S0/0 S0/0

Input

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

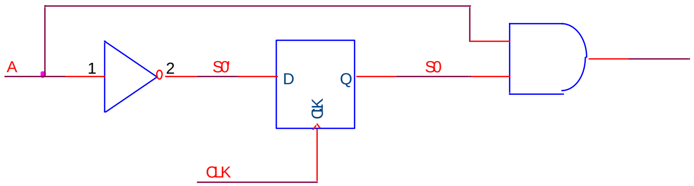
Output 1

Output 1

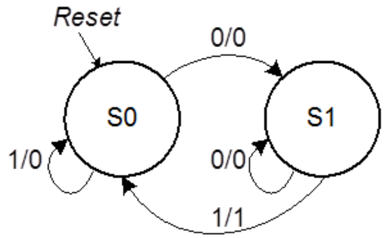
Output 1

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 transition table



Mealy FSM



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		

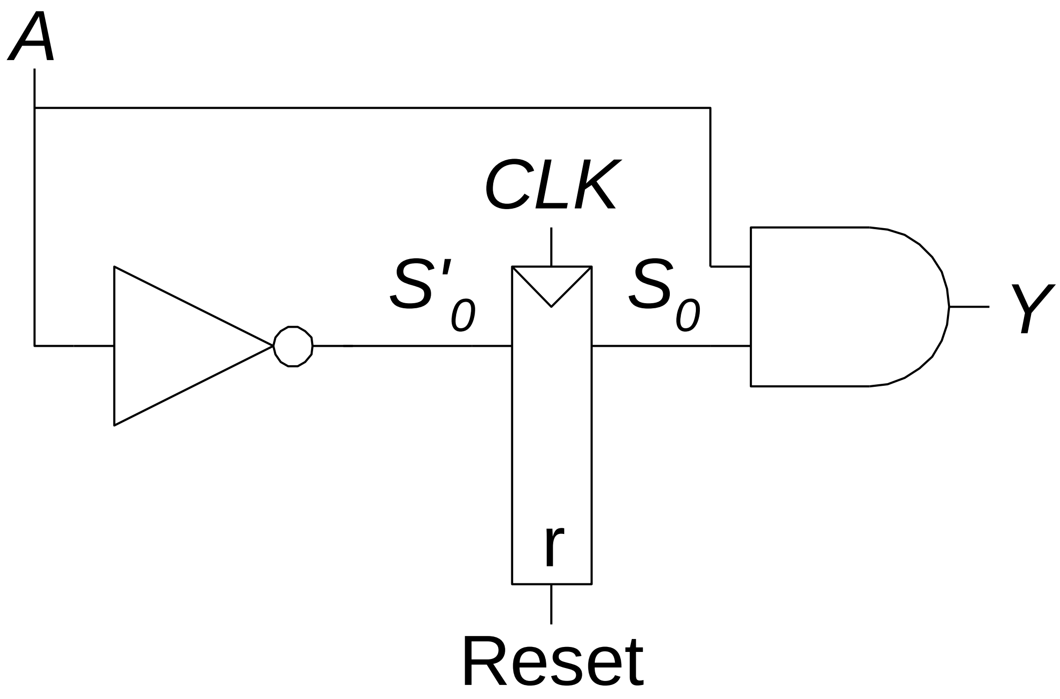
State	Encoding
S0	00
S1	01

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

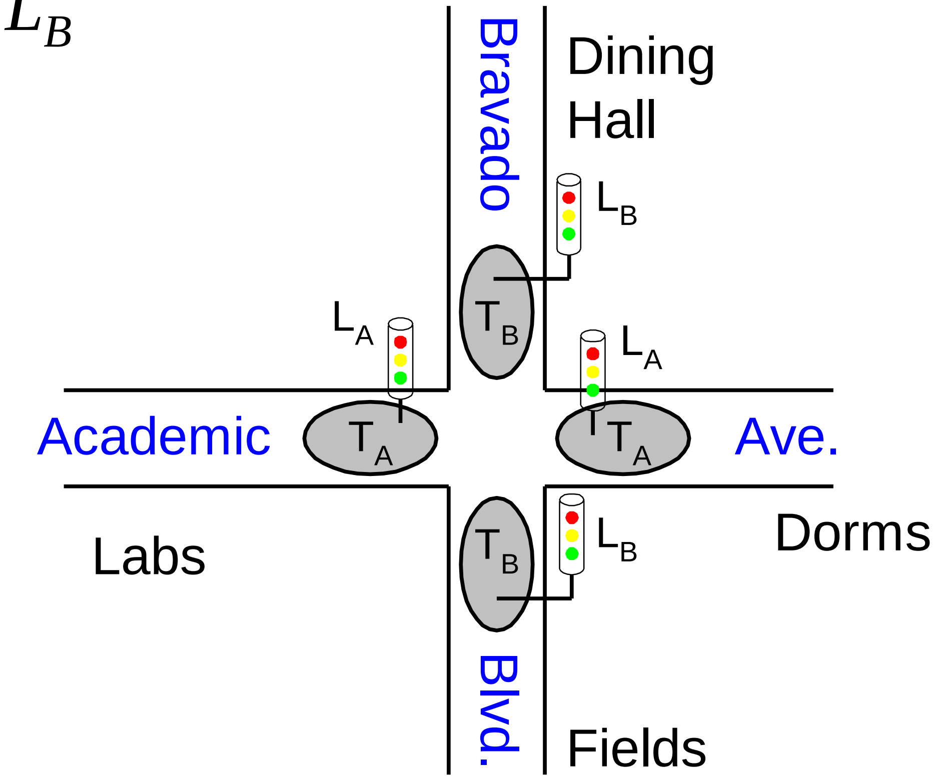
State	Encoding
S0	00
S1	01

Mealy FSM Schematic



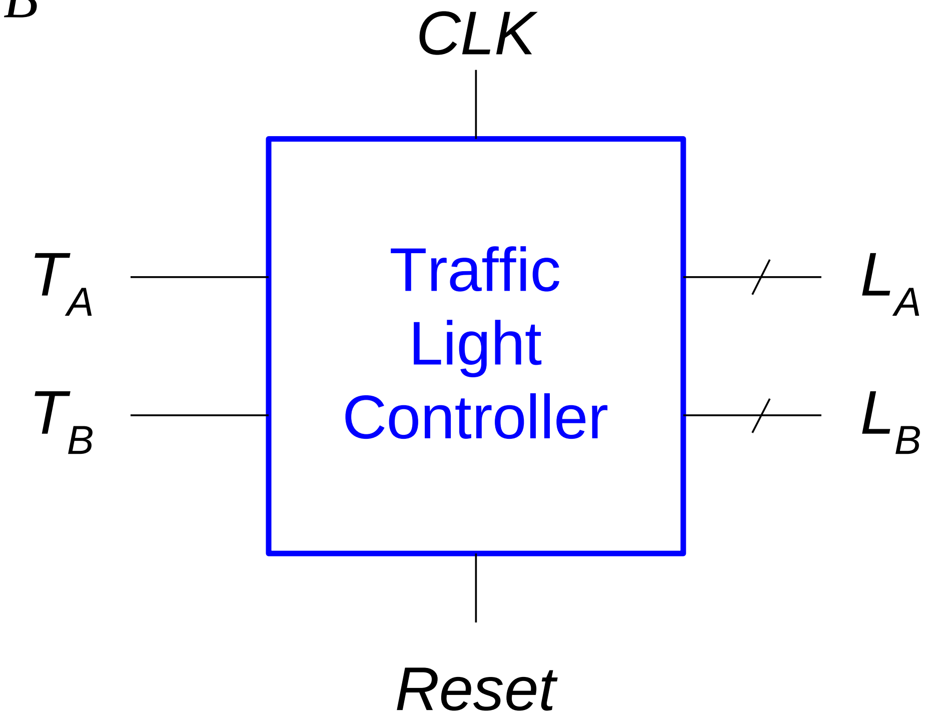
FSM Example

- 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

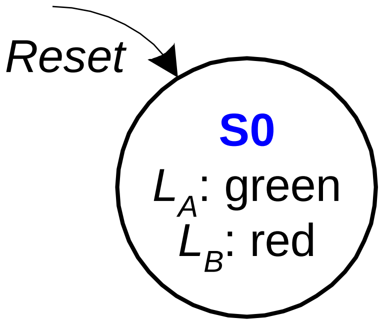


FSM State Transition Diagram

- Circles represent states
- Arcs represent state transitions
- If a state has a **single** arc leaving it that transition always occurs at the next CLK, **regardless** of the **input**
- If a state has multiple arcs they are labeled with the input that **triggers** the transition.
- State transittion table uses **X (don't care)** whenever the next state **does not depend on an input**.
- **Prime** indicates **next state**
- State transition red-yellow-green-red...

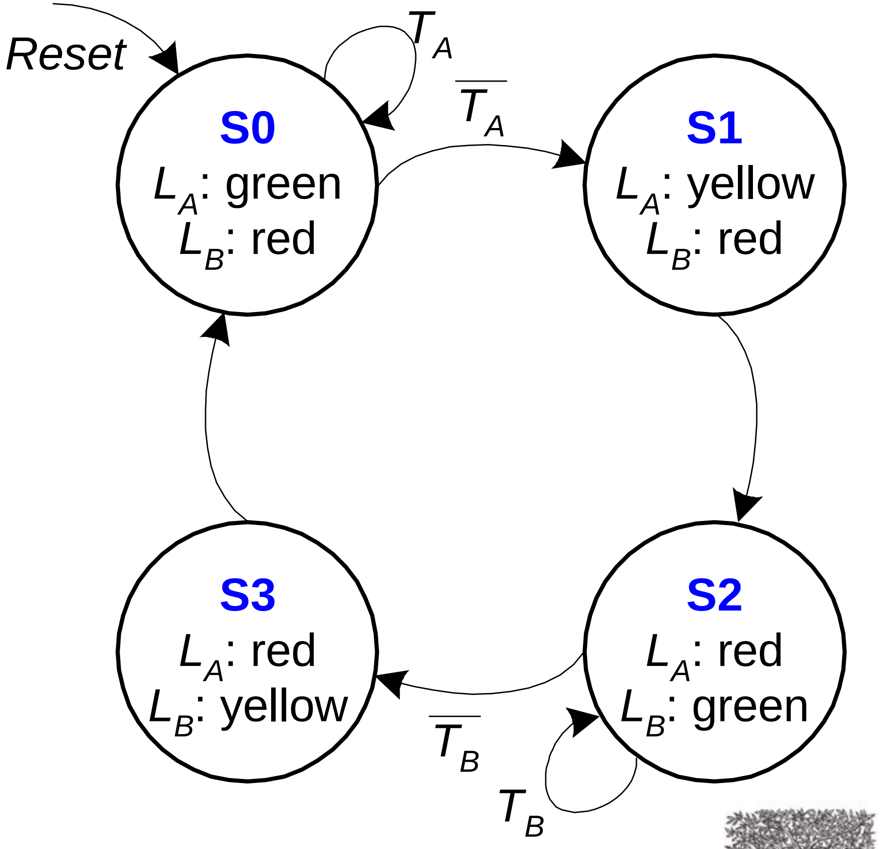
FSM State Transition Diagram

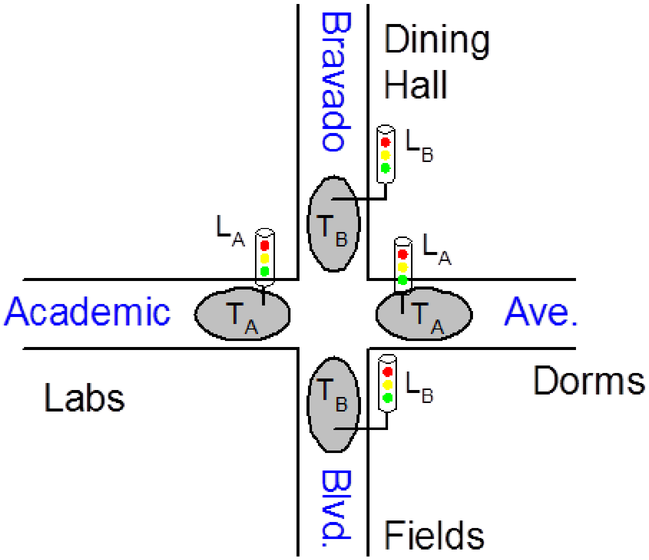
- **Moore FSM:** outputs labeled in each state
- **States:** Circles
- **Transitions:** Arcs



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	

ELSEVIER

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 State Transition Table

Current State	Inputs		Next State
S	T_A	T_B	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	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

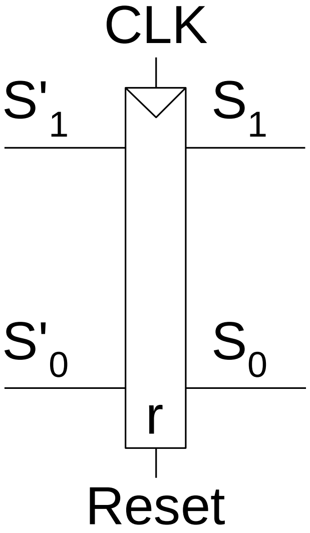
FSM Output Table

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

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

FSM Schematic: State Register



state register

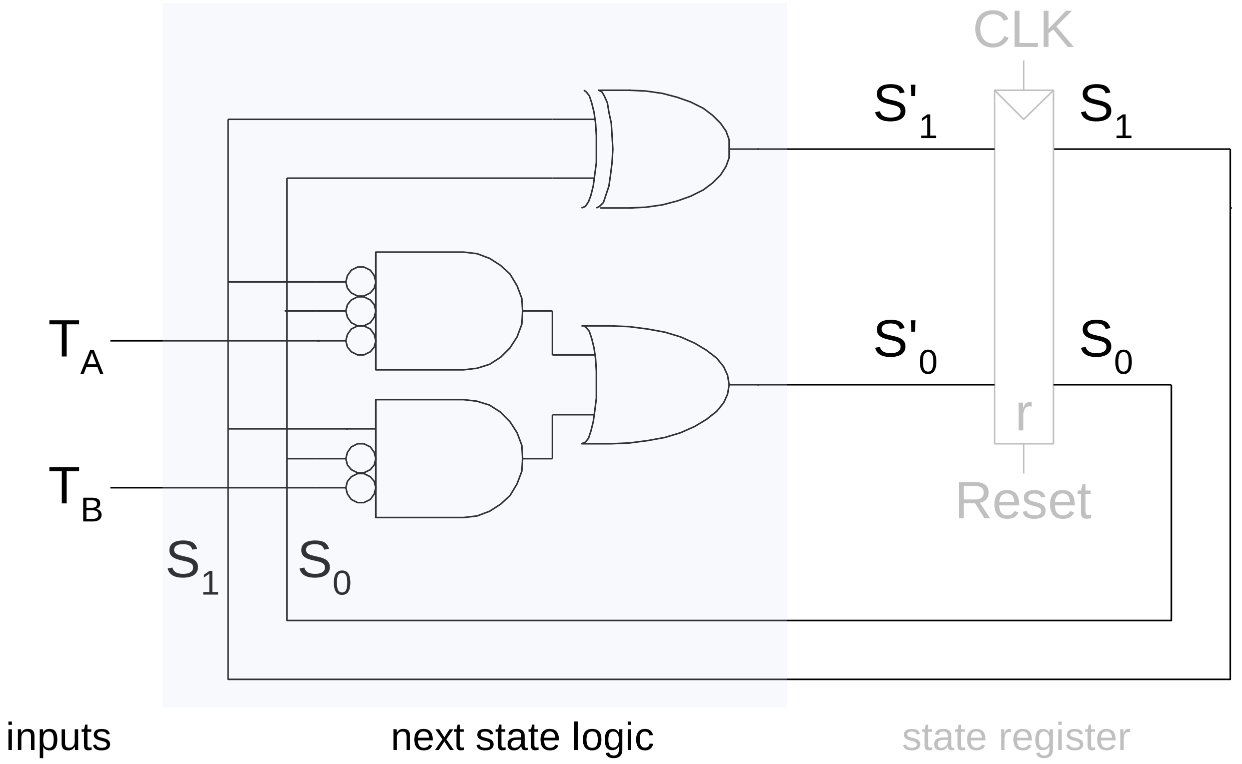
Next State Logic

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

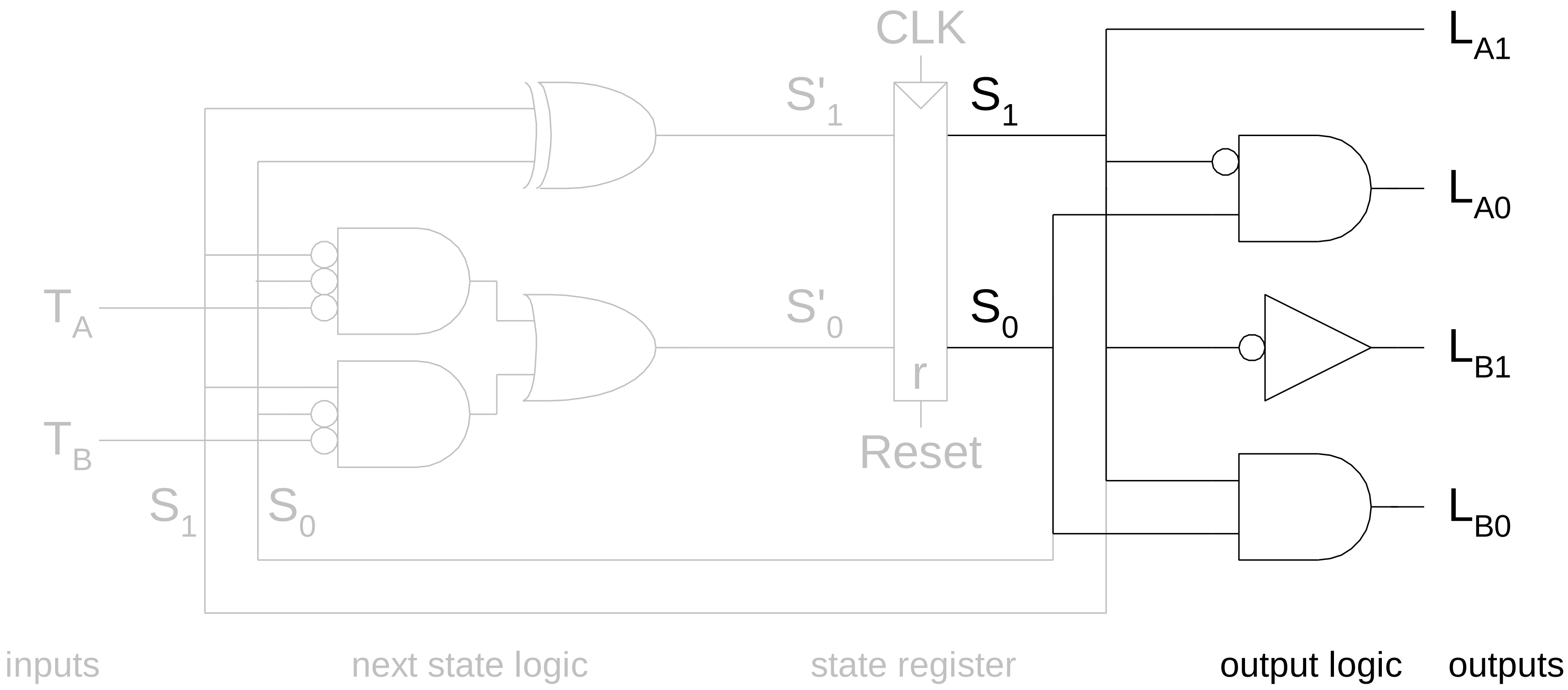
Output Logic

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

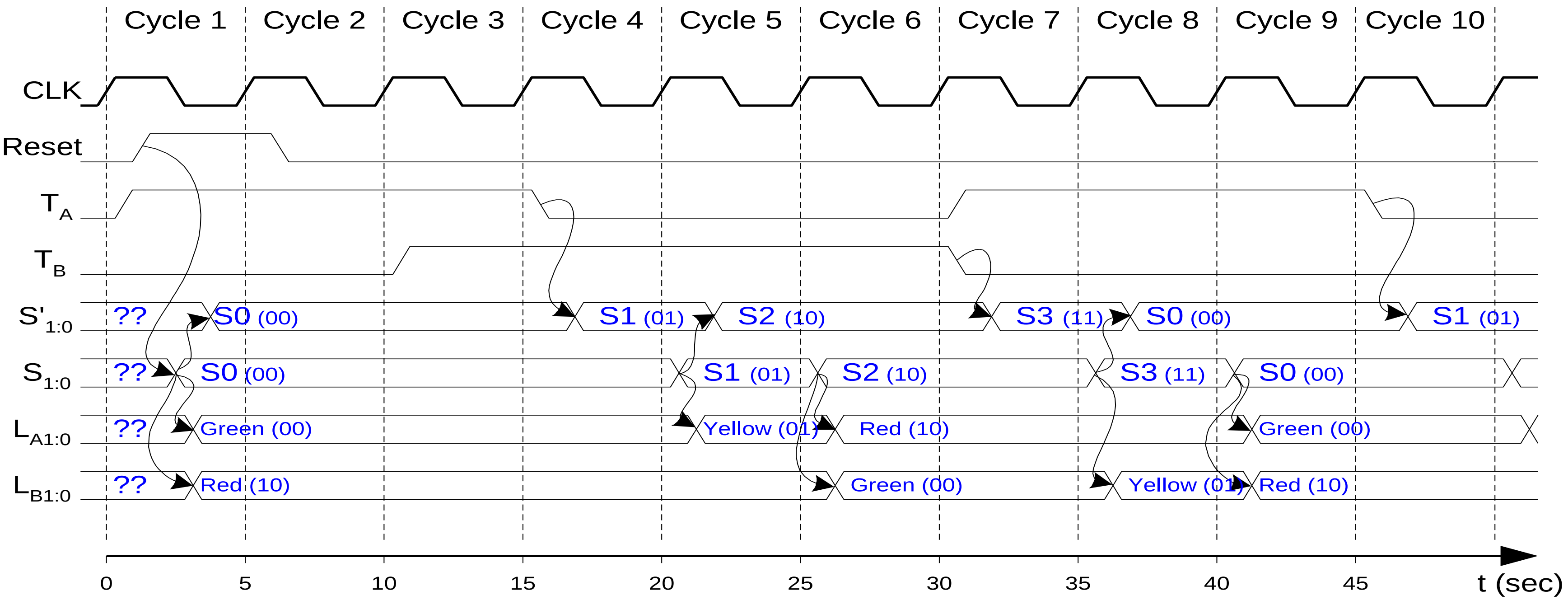
FSM Schematic: Next State Logic

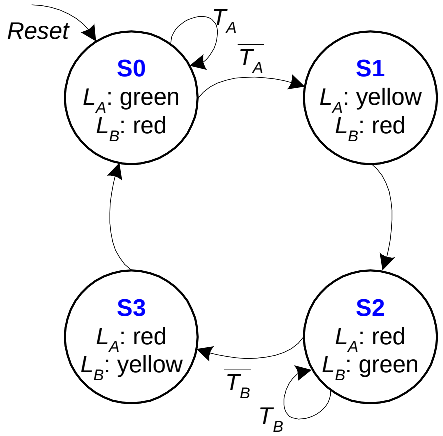


FSM Schematic: Output Logic



FSM Timing Diagram



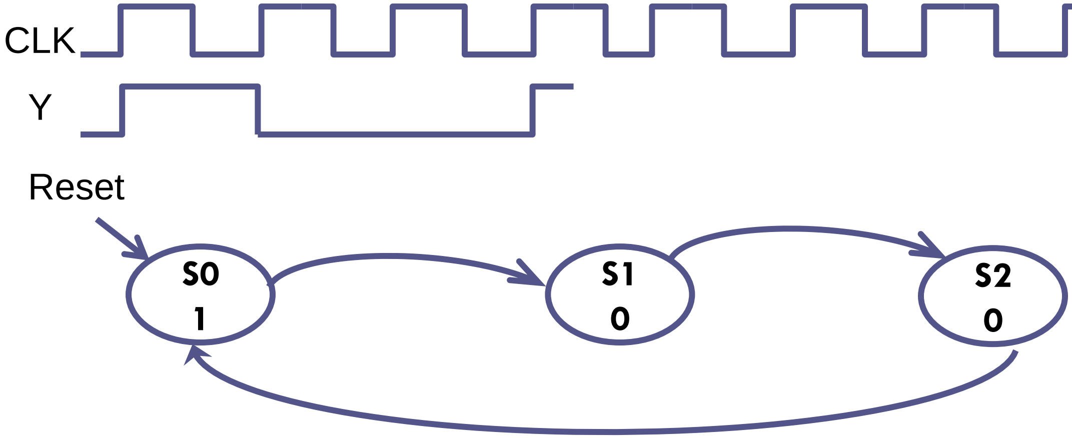


FSM State Encoding

- **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

FSM State Encoding

- □ Ex: Design a divide-by-3 counter using FSM. (divide by 3 counter has one output and no input except CLK signal, output is HIGH for one clock cycle out of every 3.



State	Binary Encoding		One Hot Encoding		
	S1	S0	S2	S1	S0
S0	0	0	0	0	1
S1	0	1	0	1	0
S2	1	0	1	0	0

FSM State Encoding

State	Current State		Next State		Output
	S1	S0	S1’	S0’	
S0	0	0	0	1	1
S1	0	1	1	0	0
S2	1	0	0	0	0

Binary encoding

S1’=$\overline{S1}\ S0$

S0’=$\overline{S1}\ \overline{S0}$

Y=$\overline{S1}\ \overline{S0}$

State	Current State			Next State			Output
	S2	S1	S0	S2	S1	S0	
S0	0	0	1	0	1	0	1
S1	0	1	0	1	0	0	0
S2	1	0	0	0	0	1	0

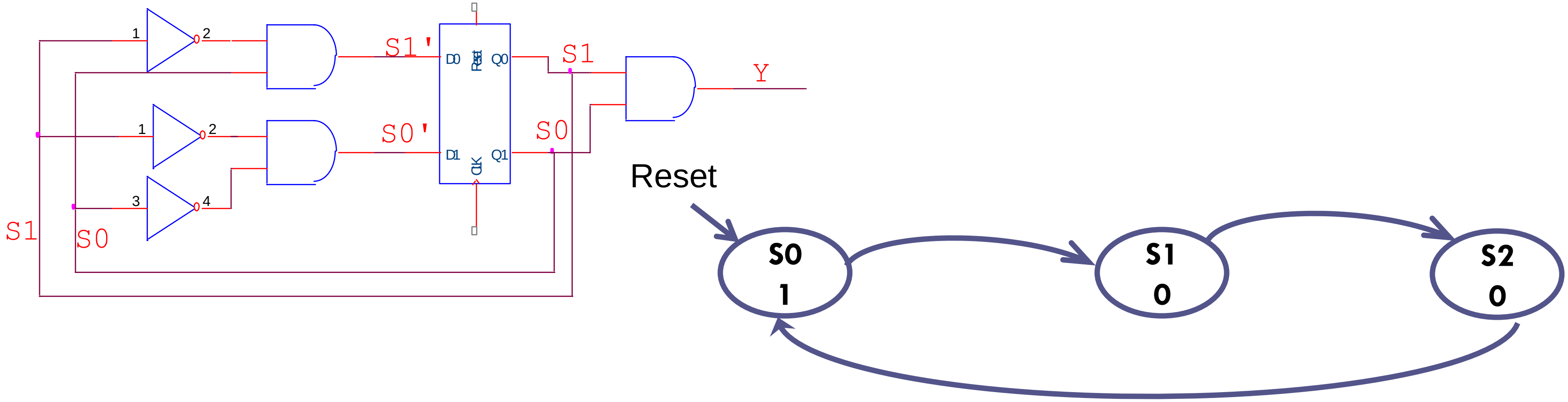
One-hot encoding

S2’=S1

S1’=S0

S0’=$S2$

Y=S0



FSM State Encoding

State	Current State			Next State			Output
	S2	S1	S0	S2	S1	S0	
S0	0	0	1	0	1	0	1
S1	0	1	0	1	0	0	0
S2	1	0	0	0	0	1	0

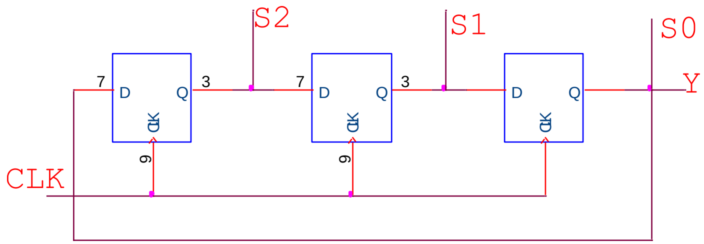
One-hot encoding

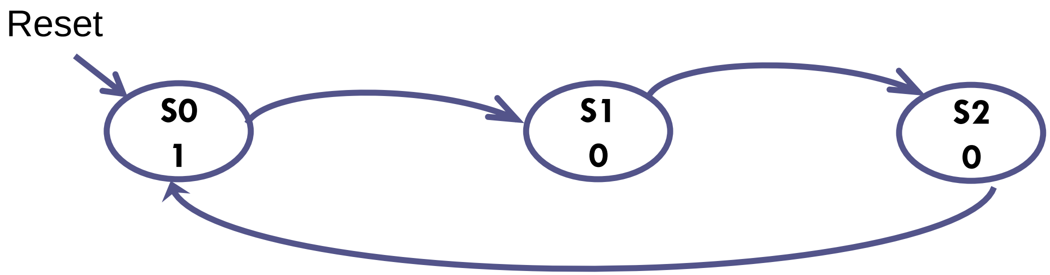
S2’=S1

S1’=S0

S0’=S2

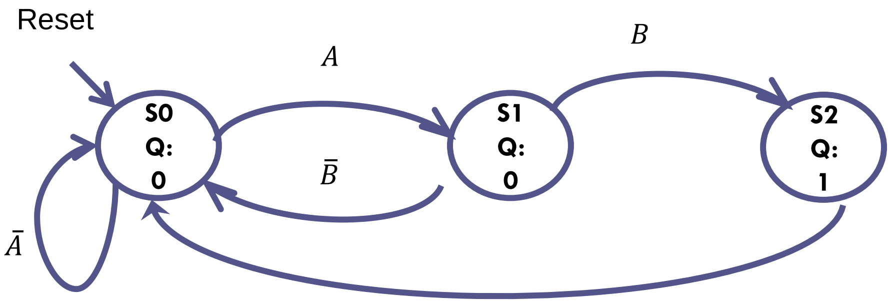
Y=S0





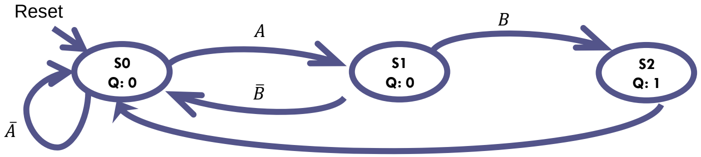
FSM Example

- Describe the function of the following FSM. Using binary state encoding write state transition table and the output table. Write the boolean equations for next state and output logic. Sketch a circuit schematic for the FSM.



FSM Example

- **Step 1: Observations**
  - FSM inputs: A, B
  - FSM output: Q
  - FSM States: S0, S1, S2
  - FSM is a moore machine since state transitions depends only on the inputs.



State Encoding	Binary Encoding	
	S1	S0
S0	0	0
S1	0	1
S2	1	0

Step 2: State transition table

State	Current State		Inputs		Next State	
	S1	S0	A	B	S1’	S0’
S0	0	0	0	X	0	0
S0	0	0	1	X	0	1
S1	0	1	X	0	1	0
S1	0	1	X	1	1	1
S2	1	0	X	X	0	0

Step 3: Output table

State	Current State		Output
	S1	S0	Q
S0	0	0	0
S1	0	1	0
S2	1	0	1

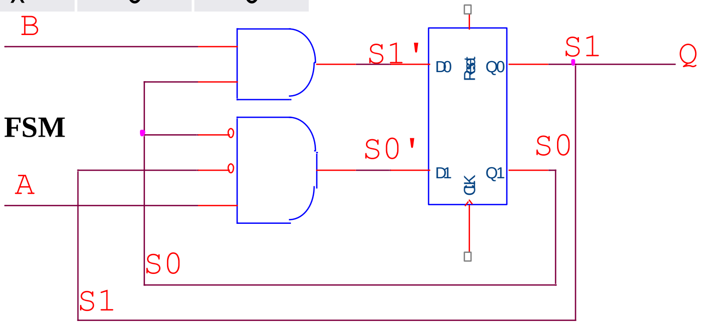
Step 4: Next state – output logic functions

S1’=S0B

S0’=$\overline{S1}\,\overline{S0}$A

Q = S1

Step 5: Circuit schematic for FSM



FSM Example

Exercise 3.23 Describe in words what the state machine in Figure 3.70 does. Using binary state encodings, complete a state transition table and output table for the FSM. Write Boolean equations for the next state and output and sketch a schematic of the FSM.

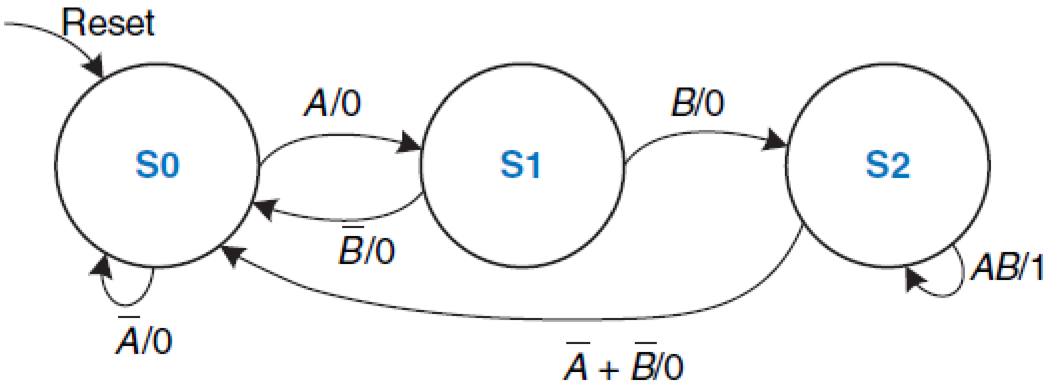
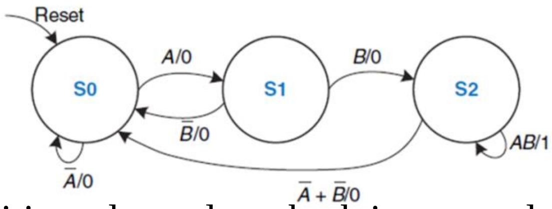


Figure 3.70 State transition diagram

FSM Example

- **Step 1: Observations**
  - FSM inputs: A, B
  - FSM output: Q
  - FSM States: S0, S1, S2
  - FSM is a mealy machine since state transitions depends on both inputs and output



Step 2: State transition table

State	Binary Encoding	
	S1	S0
S0	0	0
S1	0	1
S2	1	0

State	Current State		Inputs		Next State		Output
	S1	S0	A	B	S1'	S0'	0
S0	0	0	0	X	0	0	0
S0	0	0	1	X	0	1	0
S1	0	1	X	0	0	0	0
S1	0	1	X	1	1	0	0
S2	1	0	0	0	0	0	0
S2	1	0	0	1	0	0	0
S2	1	0	1	0	0	0	0
S2	1	0	1	1	1	0	1

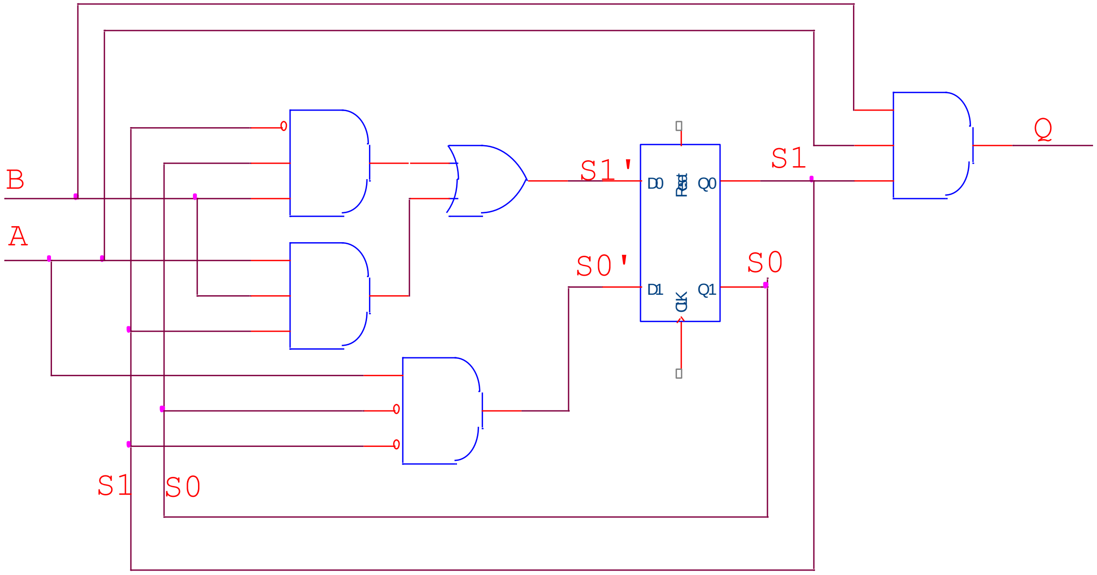
FSM Example

Step 4: Next state – output logic functions

$S1' = \overline{S1}\,S0\,B + S1AB$

$S0' = \overline{S1}\;\overline{S0}\,A$

$Q = S1AB$



Step 5: Circuit schematic for FSM

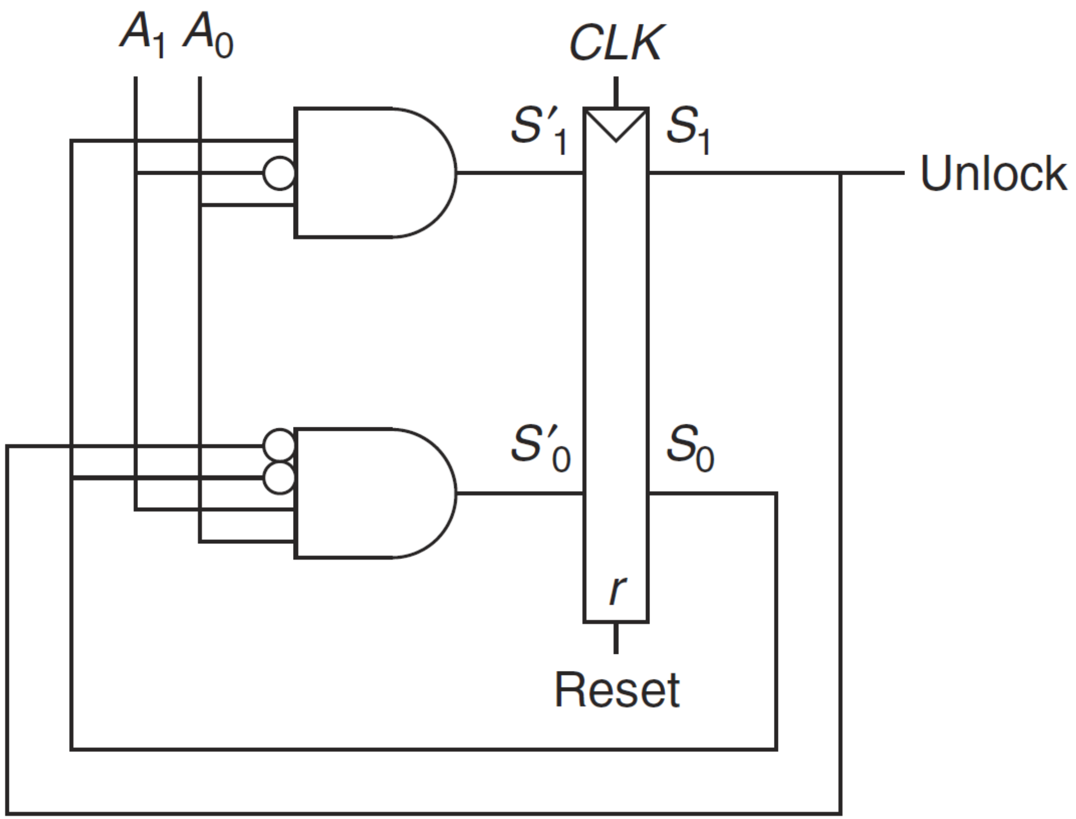
FSM Design Procedure

- 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 a FSM from its Circuit

- **Reverse** process of the FSM design
- Write **next state** and **output equations**
- Create next state and output **tables**
- **Reduce** the next state table (omit unreachable states)
- Assign a **name** for each reachable state
- Rewrite next state and output tables with **state names**
- Draw **state transition** diagram
- State the function of FSM in words

Deriving an FSM from its Circuit



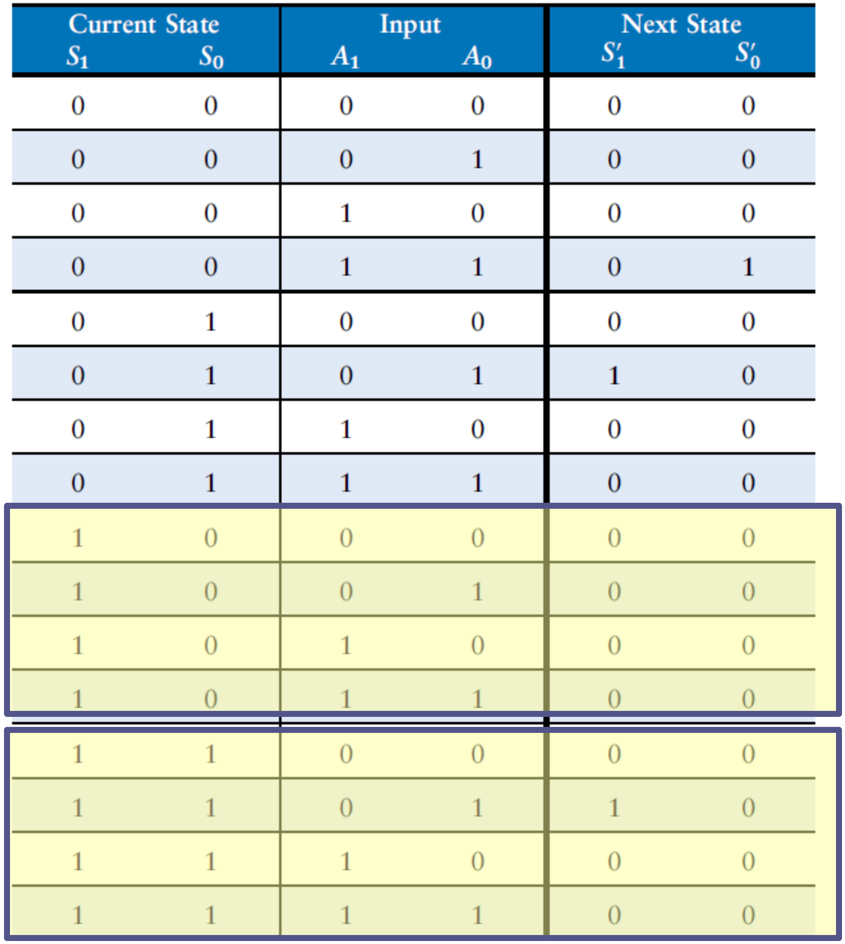
$$S'_1 = S_0 \overline{A_1} A_0$$

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

$$Unlock = S_1$$

What does this circuit do?

Next State Table

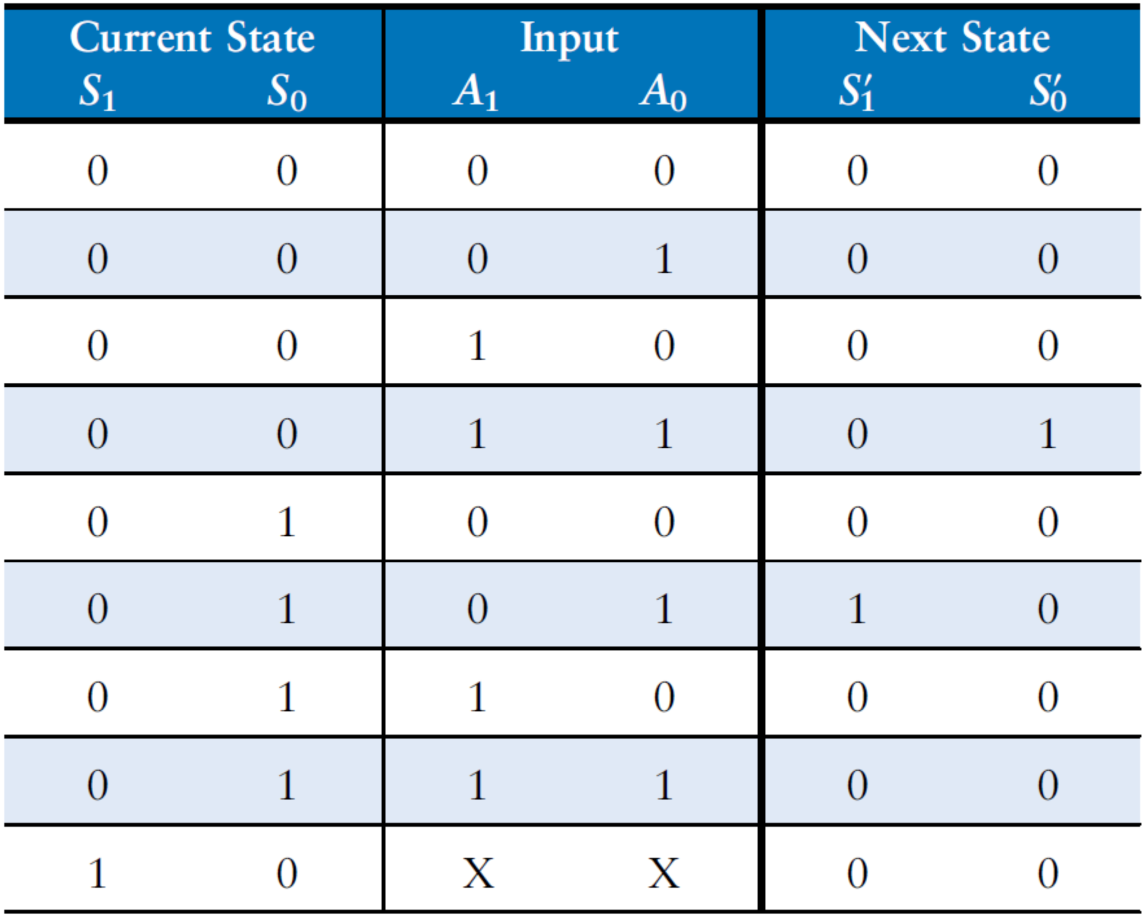


Current State		Input		Next State	
S_1	S_0	A_1	A_0	S'_1	S'_0
0	0	0	0	0	0
0	0	0	1	0	0
0	0	1	0	0	0
0	0	1	1	0	1
0	1	0	0	0	0
0	1	0	1	1	0
0	1	1	0	0	0
0	1	1	1	0	0
1	0	0	0	0	0
1	0	0	1	0	0
1	0	1	0	0	0
1	0	1	1	0	0
1	1	0	0	0	0
1	1	0	1	1	0
1	1	1	0	0	0
1	1	1	1	0	0

Observation 1: Next state is never 11

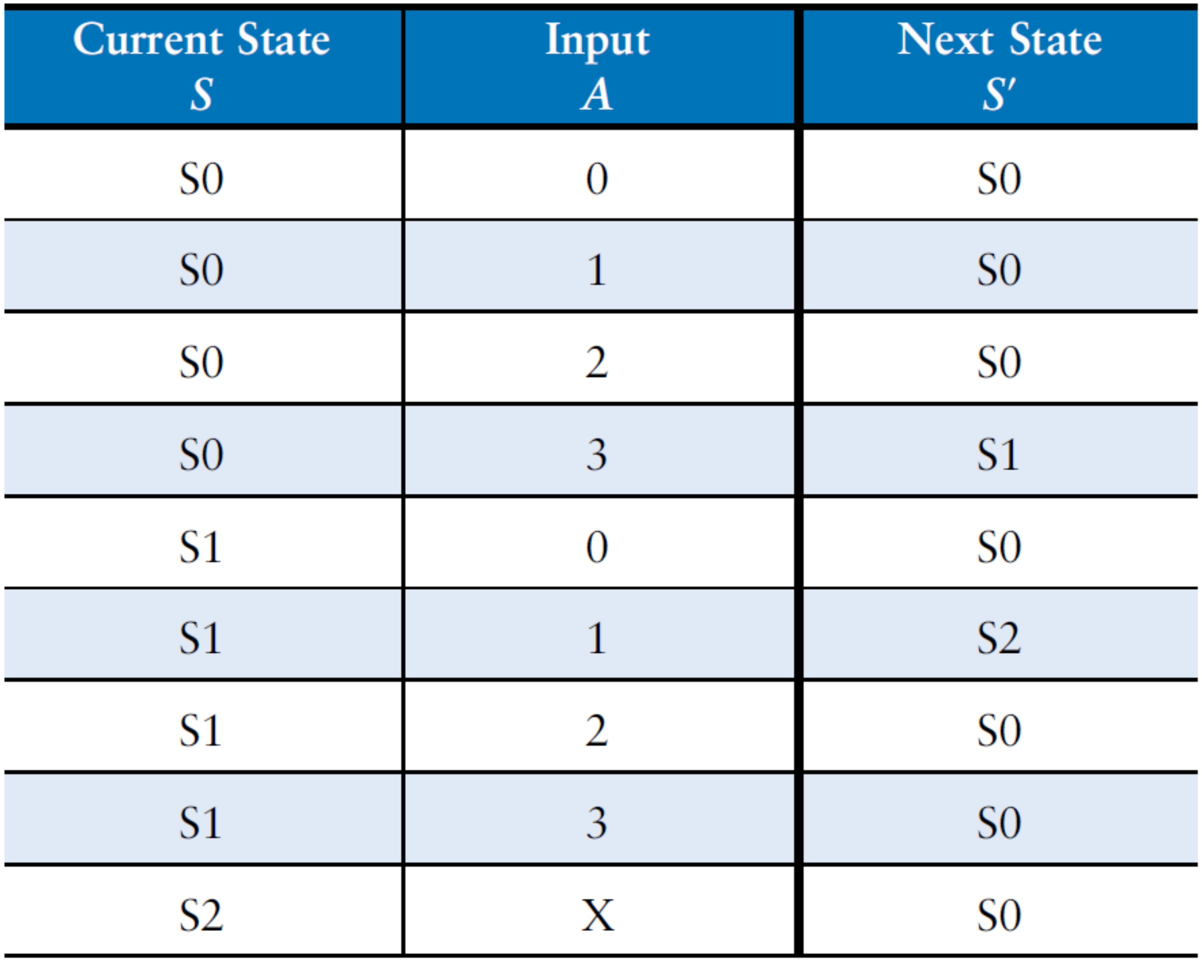
Observation 2: Next state after 10 is 00

Reduced Next State Table



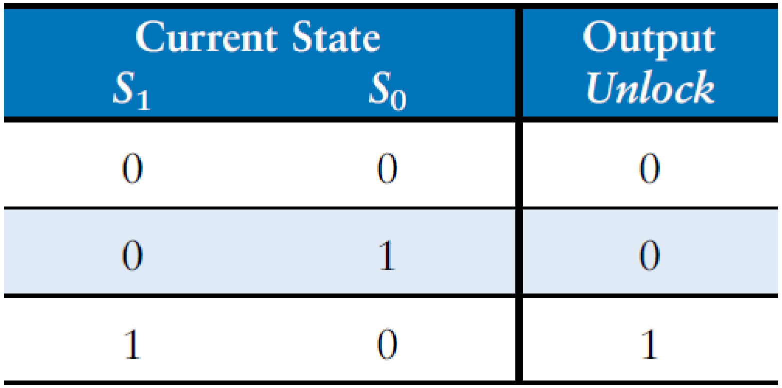
Current State		Input		Next State	
S_1	S_0	A_1	A_0	S'_1	S'_0
0	0	0	0	0	0
0	0	0	1	0	0
0	0	1	0	0	0
0	0	1	1	0	1
0	1	0	0	0	0
0	1	0	1	1	0
0	1	1	0	0	0
0	1	1	1	0	0
1	0	X	X	0	0

Symbolic Next State Table

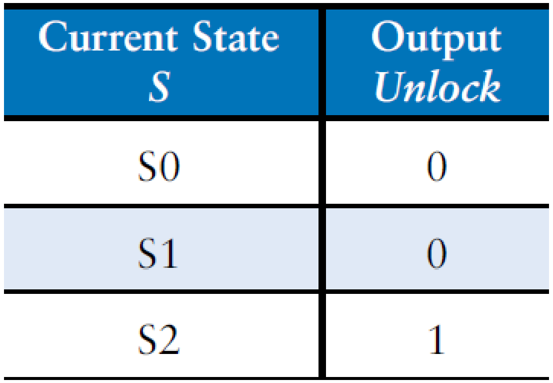


Current State *S*	Input *A*	Next State *S′*
S0	0	S0
S0	1	S0
S0	2	S0
S0	3	S1
S1	0	S0
S1	1	S2
S1	2	S0
S1	3	S0
S2	X	S0

Output Table

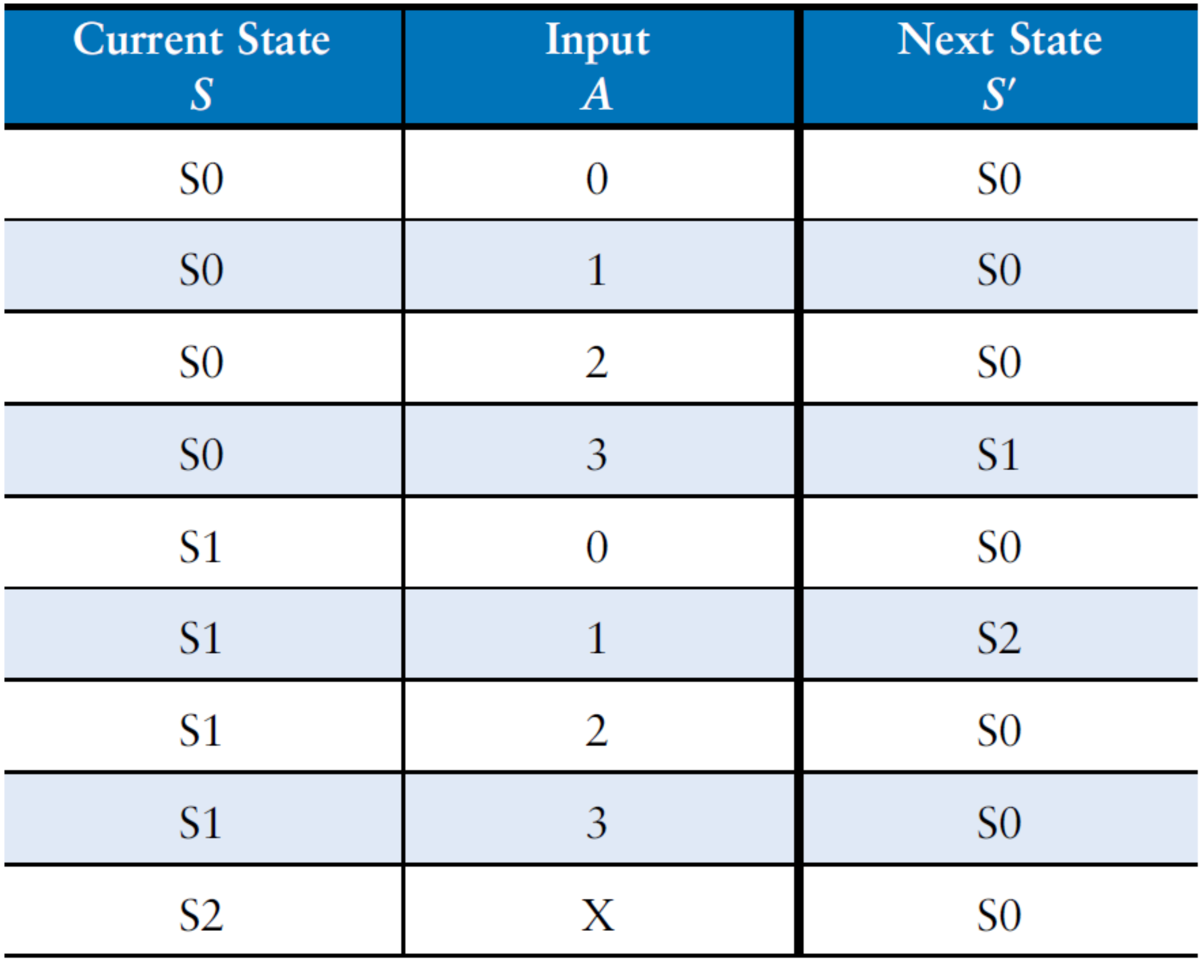


Current State		Output
S_1	S_0	*Unlock*
0	0	0
0	1	0
1	0	1



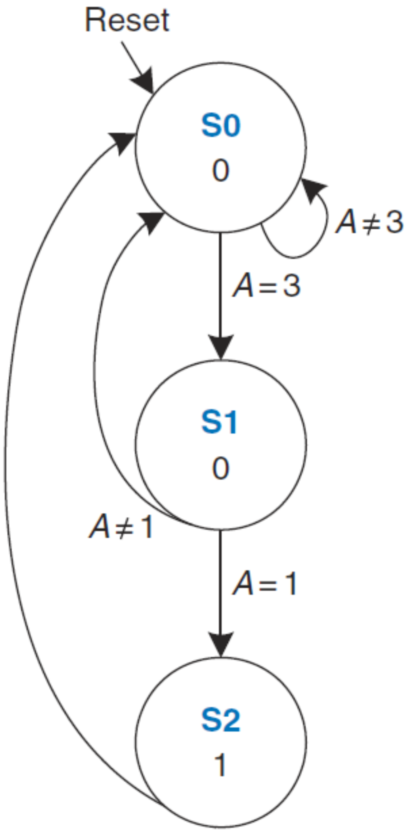
Current State	Output
S	*Unlock*
S0	0
S1	0
S2	1

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

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).

Review$_{11}$

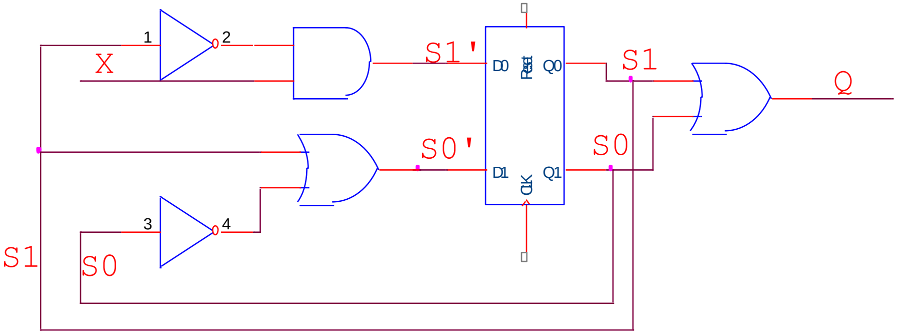
- Moore FSM Schematic
- Moore & Mealy Timing Diagram
- FSM State Transition Diagram
- FSM State Transition Table
- FSM Output Table
- FSM Schematic
- FSM State Encoding
- FSM Design Procedure
- Deriving an FSM from its Circuit

FSM summary

- Identify the inputs and outputs
- Sketch the state transition diagram
- Write state transition table
- Write output table
  - Write combined state transition table + output table (for mealy FSM)
- Select state encoding
- Write boolean equations for next state and output logic
- Sketch the circuit schematic

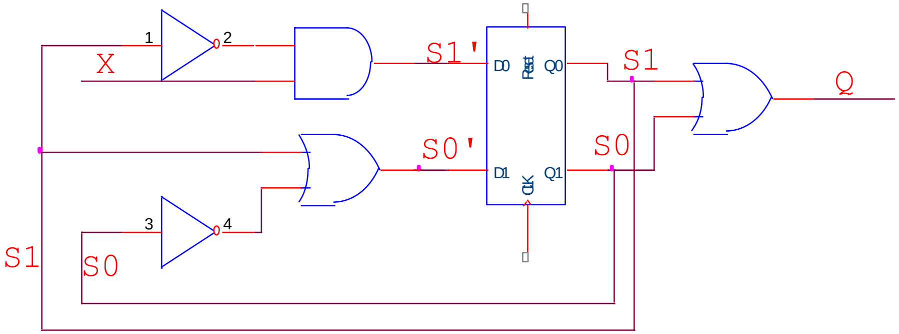
FSM Example

- Sketch the state transition diagram for the given circuit.



FSM Example

- **Step 1: Observations**
  - FSM inputs: X
  - FSM output: Q
  - FSM States: S0, S1
  - FSM is a moore machine



Step 2: Next state – output logic functions

S1'=$\overline{S1}X$

S0'=$S1 + \overline{S0}$

Q = S1+S0

State	Binary Encoding	
	S1	S0
S0	0	0
S1	0	1
S2	1	0

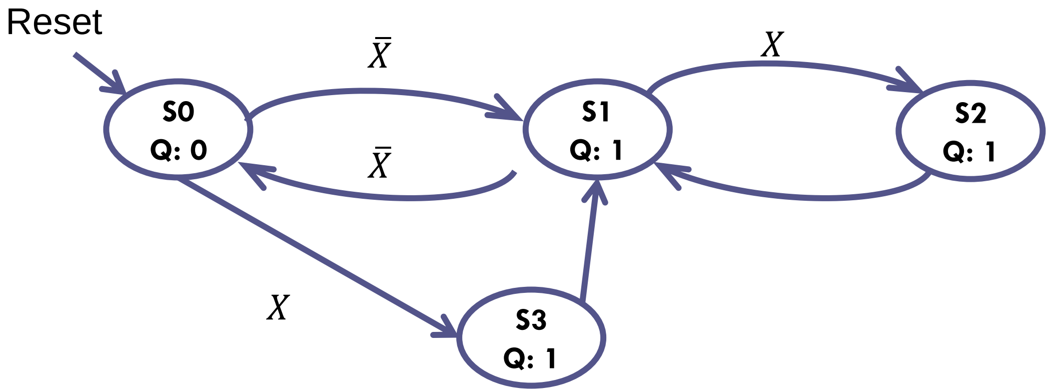
Step 3: State transition table

State	Current State		Input	Next State	
	S1	S0	X	S1'	S0'
S0	0	0	0	0	1
S0	0	0	1	1	1
S1	0	1	0	0	0
S1	0	1	1	1	0
S2	1	0	X	0	1
S2	1	1	X	0	1

State	Output
	Q
S0	0
S1	1
S2	1
S3	1

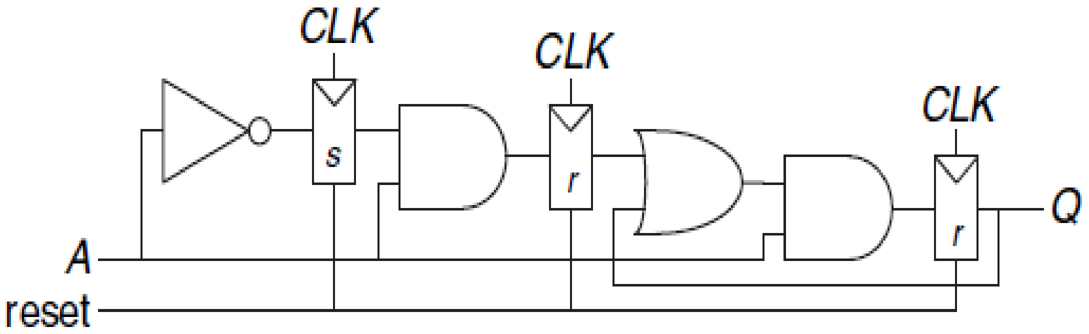
FSM Example

Step 4: State transition diagram



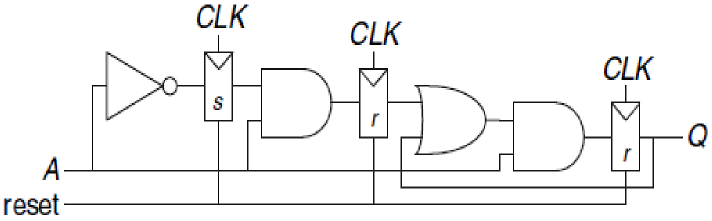
FSM Example

Exercise 3.32 Repeat Exercise 3.31 for the FSM shown in Figure 3.73. Recall that the *s* and *r* register inputs indicate set and reset, respectively.



FSM Example

- **Step 1: Observations**
  - FSM inputs: A
  - FSM output: Q
  - FSM States: S0, S1, S2
  - FSM is a moore machine



Step 2: Next state – output logic functions

S1'=$S0A$

S0'=$\overline{A}$

S2'=(S1+S2)A

Q = S2

State	OneHot Encoding		
	S2	S1	S0
S0	0	0	1
S1	0	1	0
S2	1	0	0

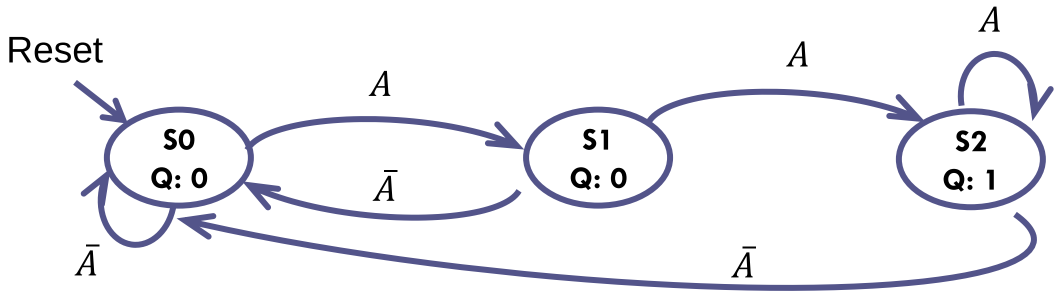
Step 3: State transition table

State	Current State			Input	Next State		
	S2	S1	S0	A	S2'	S1'	S0'
S0	0	0	1	0	0	0	1
S0	0	0	1	1	0	1	0
S1	0	1	0	0	0	0	1
S1	0	1	0	1	1	0	0
S2	1	0	0	0	0	0	1
S2	1	0	0	1	1	0	0

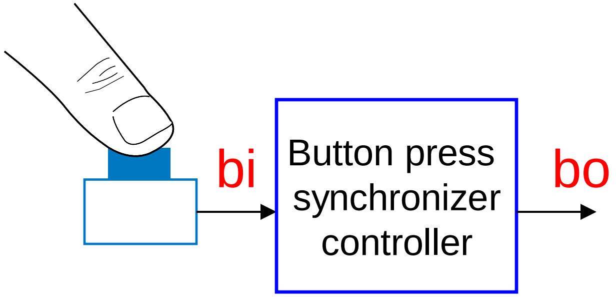
State	Output
	Q
S0	0
S1	1
S2	1

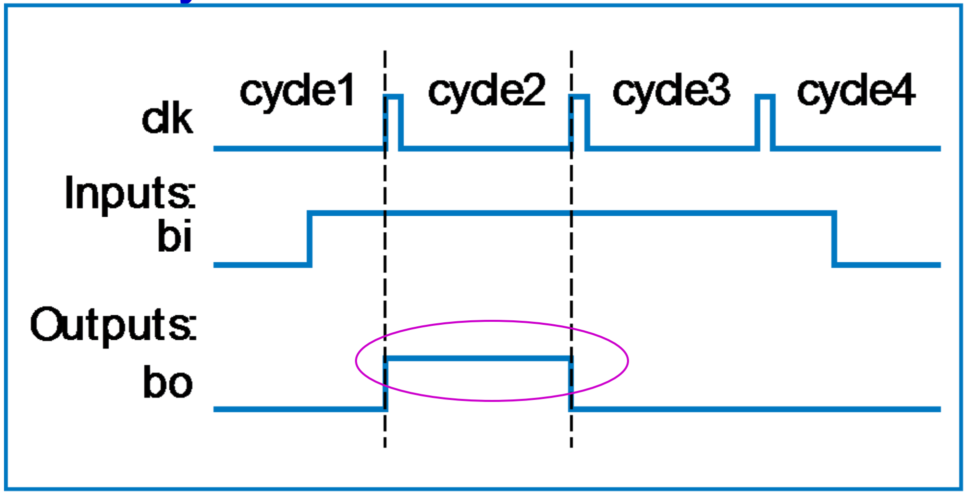
FSM Example

Step 4: State transition diagram



Controller Example:
Button Press Synchronizer

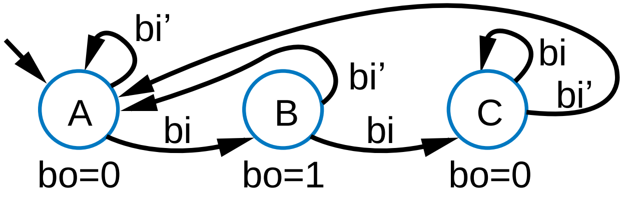




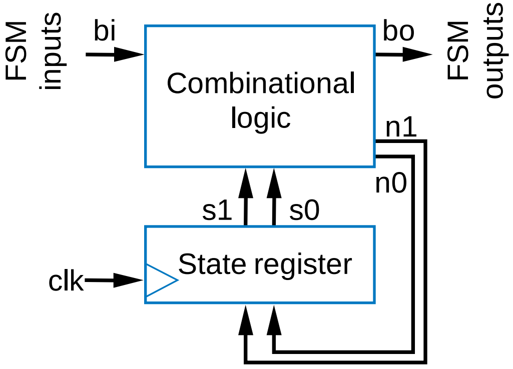
- Want simple sequential circuit that converts button press to single cycle duration, regardless of length of time that button was actually pressed

Controller Example: Button Press Synchronizer (cont)

FSM inputs: bi; FSM outputs: bo

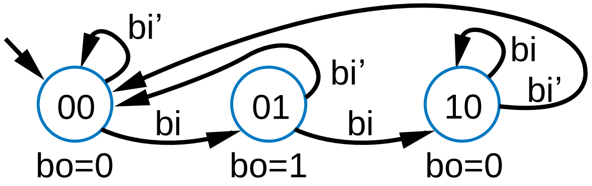


Step 1: Capture FSM



Step 2A: Set up architecture

FSM inputs: bi; FSM outputs: bo



Step 2B: Encode states

Combinational logic

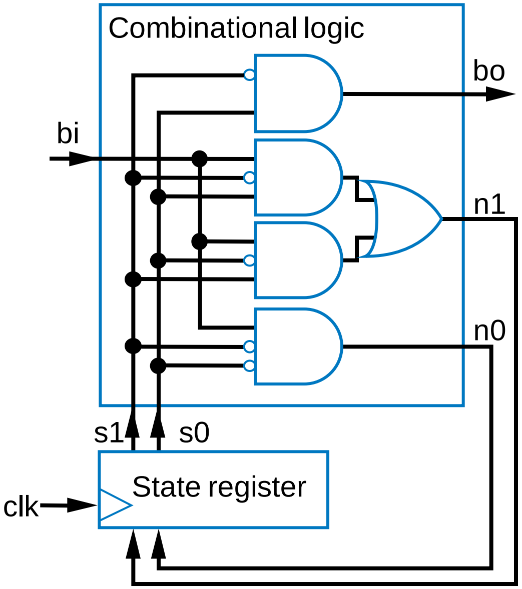
	Inputs			Outputs		
	s1	s0	bi	n1	n0	bo
A	0	0	0	0	0	0
	0	0	1	0	1	0
B	0	1	0	0	0	1
	0	1	1	1	0	1
C	1	0	0	0	0	0
	1	0	1	1	0	0
unused	1	1	0	0	0	0
	1	1	1	0	0	0

Step 2C: Fill in truth table

n1 = s1’s0bi + s1s0’bi

n0 = s1’s0’bi

bo = s1’s0bi’ + s1’s0bi = s1s0

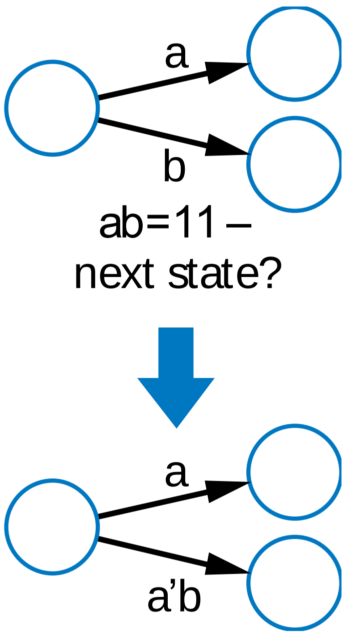


Step 2D: Implement combinational logic

How about Mealy FSM?

Common Mistakes when Capturing FSMs

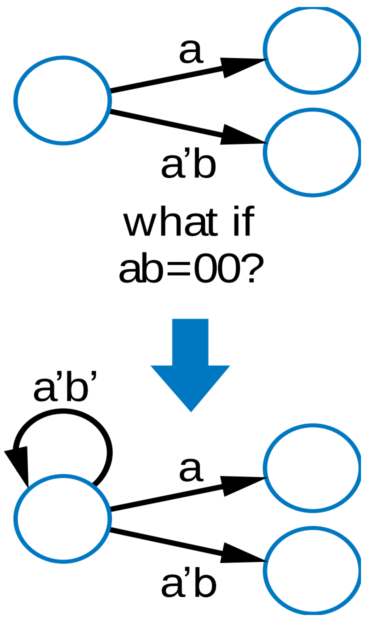
- Non-exclusive transitions



ab=11 – next state?

- Incomplete transitions

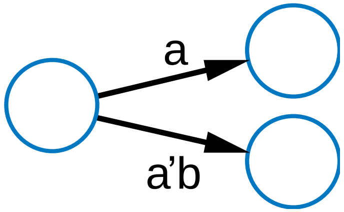
a



what if ab=00?

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 + 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:

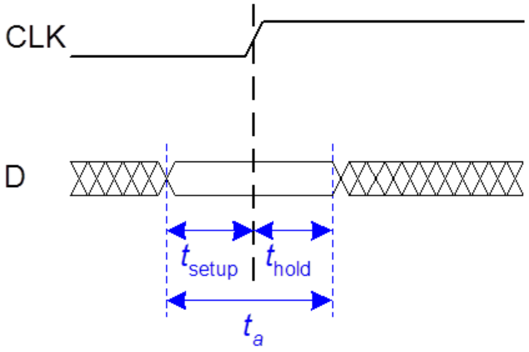
* Only one condition true (AND of each pair is always 0)

* One condition true (OR of all transitions is always 1)

Timing

- □ A flip-flop copies the input to output on rising CLK edge (Sampling Data on Clock edge)
- □ Dynamic discipline satisifed when D is stable and clock rises
- □ In order to obtain a well defined output input must be stable before the clock edge (sequential element has an aperture time)
- □ Aperture time is defined by setup time and hold time (just before the clock edge and just after the clock edge)
- □ The clock period must be long enough for all signals to settle, this limits the speed of the system (clock frequency)
- □ Aperture time is the total time for which

the inputs must remain stable

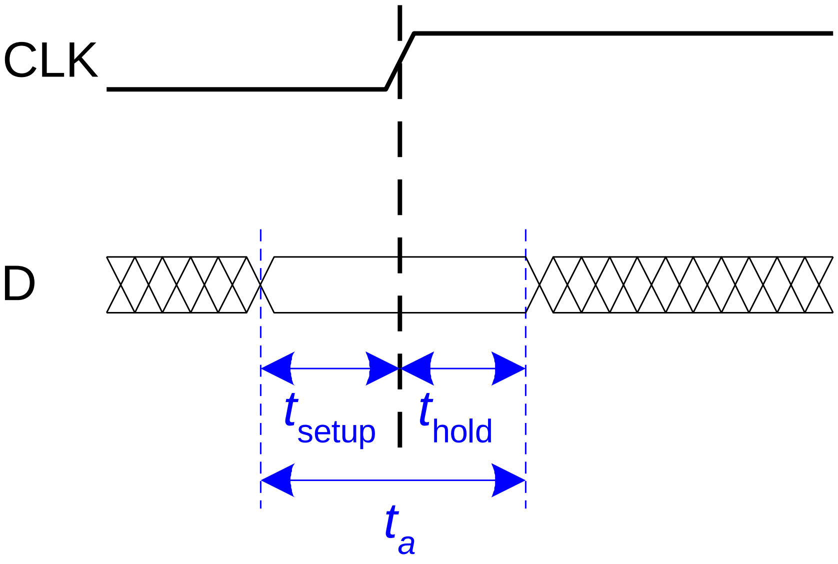


Timing

- 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

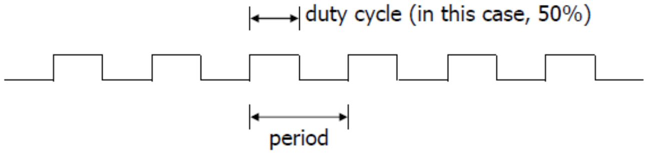
Input Timing Constraints

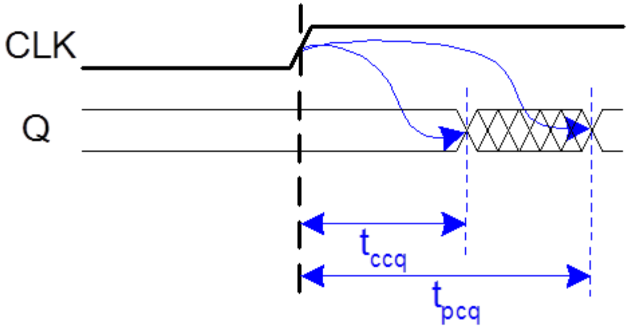
- **Setup time:** t_{setup} = time *before* clock edge data must be stable (i.e. not changing)
- **Hold time:** t_{hold} = time *after* clock edge data must be stable
- **Aperture time:** t_a = time *around* clock edge data must be stable ($t_a = t_{setup} + t_{hold}$)



Timing

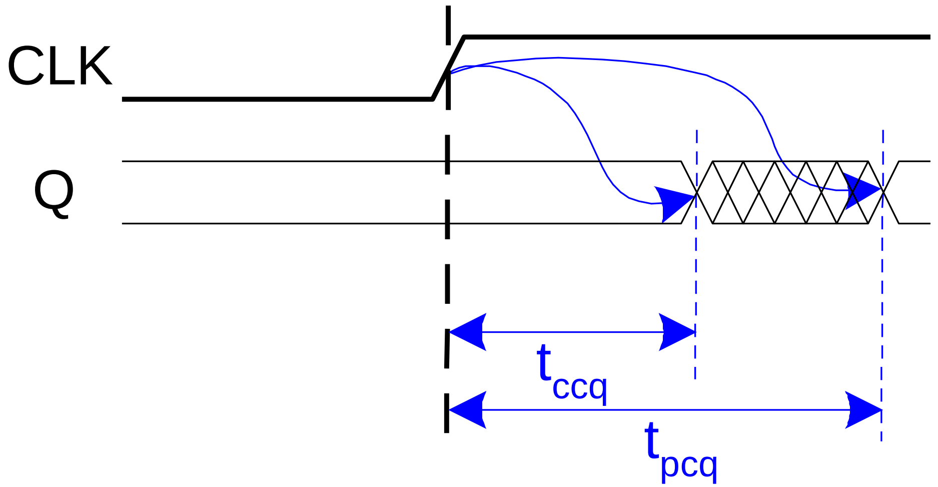
- When clock rises and Data sampling initiated the outputs may start to change (contamination delay-t_{ccQ}), must settle after a delay (propogation delay - t_{pcQ}).
- **Contamination delay: minimum** time from when an input changes until any output starts to change its value.
- **Propogation delay: maximum** time from when an input changes until the outputs reach their final value.
- Clock period (T_c) is the time between rising edges of a periodic signal. Frequency is measured in Hertz (Hz) and $f=1/T_c$. 1MHz = 10^6 Hz





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)

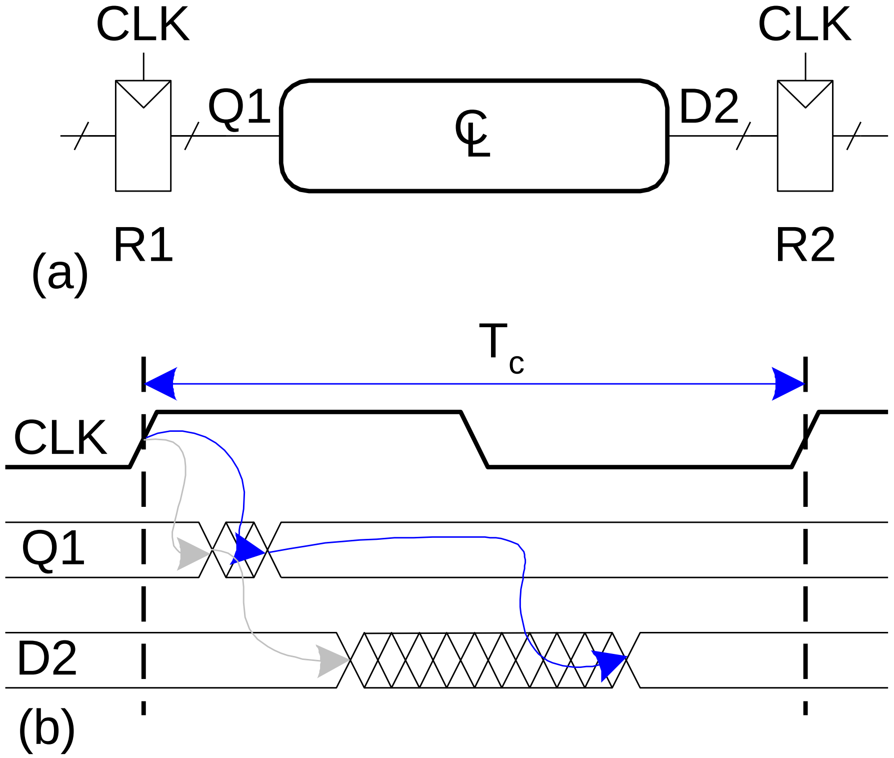


Dynamic Discipline

- 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

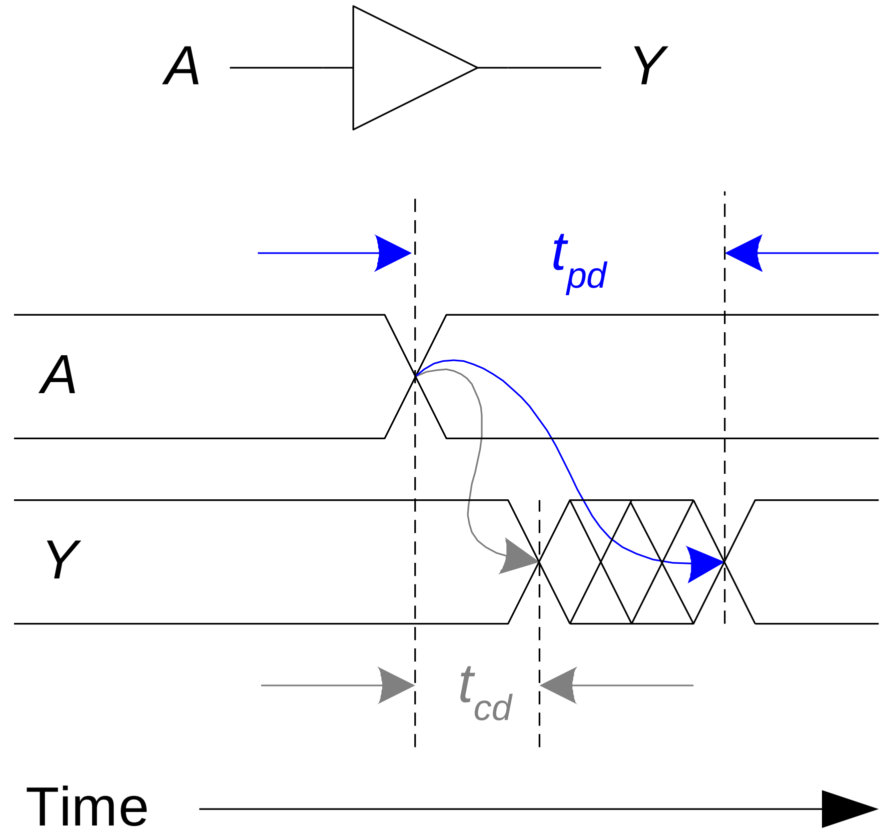
Dynamic Discipline

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



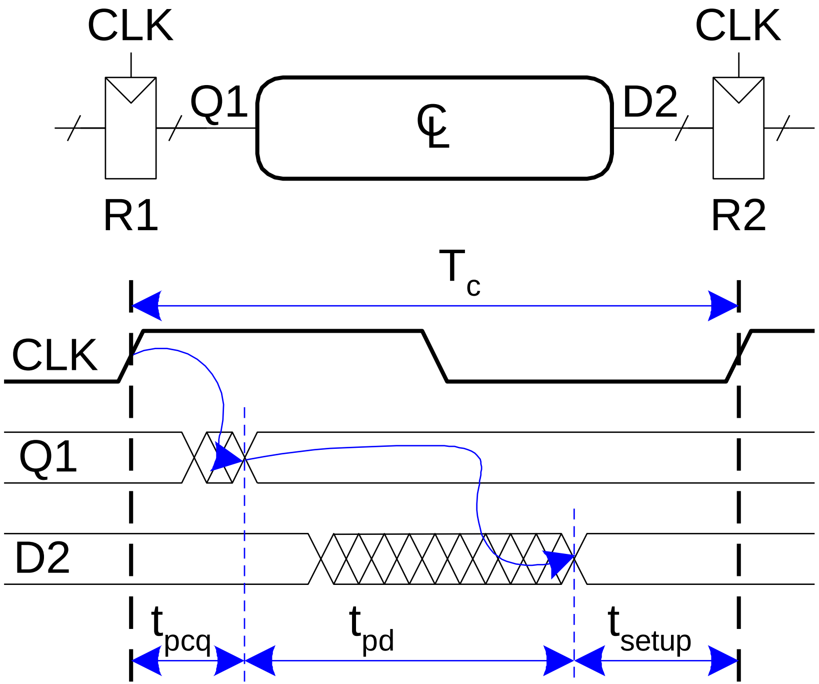
Reminder: Propagation & Contamination Delays of Combinational Logic

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



Setup Time Constraint

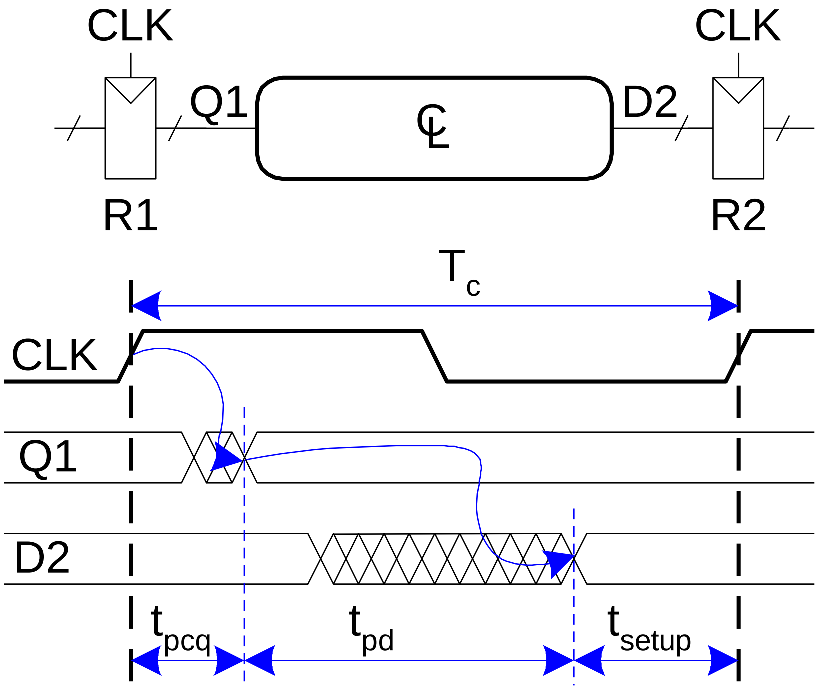
- Depends on the **maximum** delay from register R1 through combinational logic to R2
- The input to register R2 must be stable at least t_{setup} before clock edge



$T_c \geq$

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



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

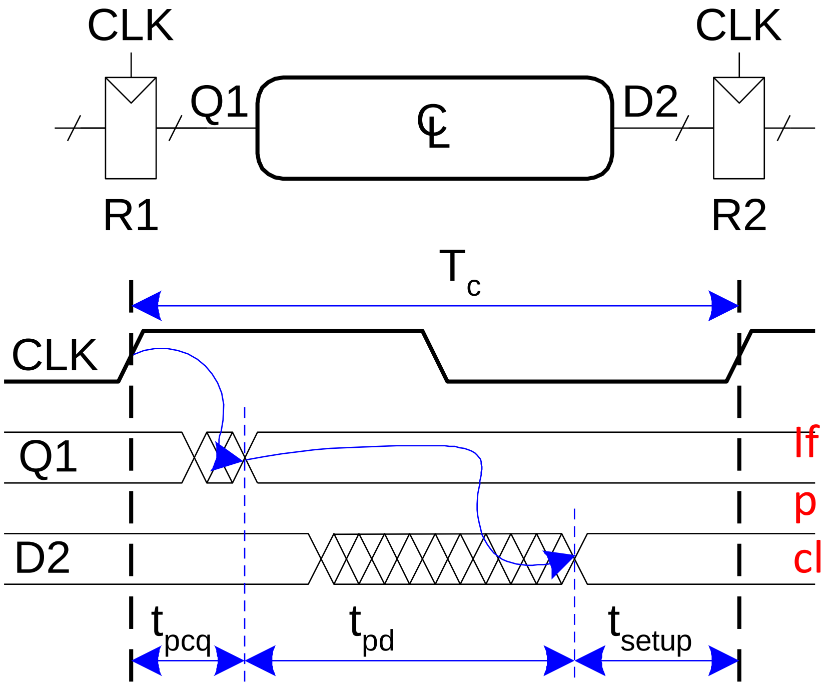
$$t_{pd} \leq$$

t_{pcq} and t_{setup} are specified by manufacturer

Sequencing overhead=(t_{pcq}+t_{setup})

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

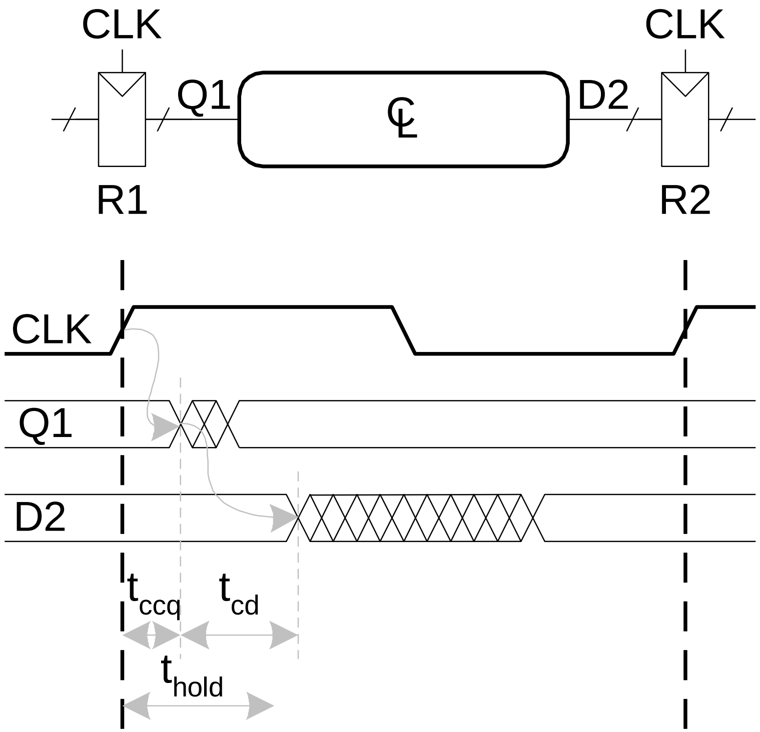


$$T_c \geq t_{pcq} + t_{pd} + t_{setup}$$
$$t_{pd} \leq T_c - (t_{pcq} + t_{setup})$$

If t_{pd} large in that case there may be problems in that case increase the clock perid (decrease clock frequency)

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

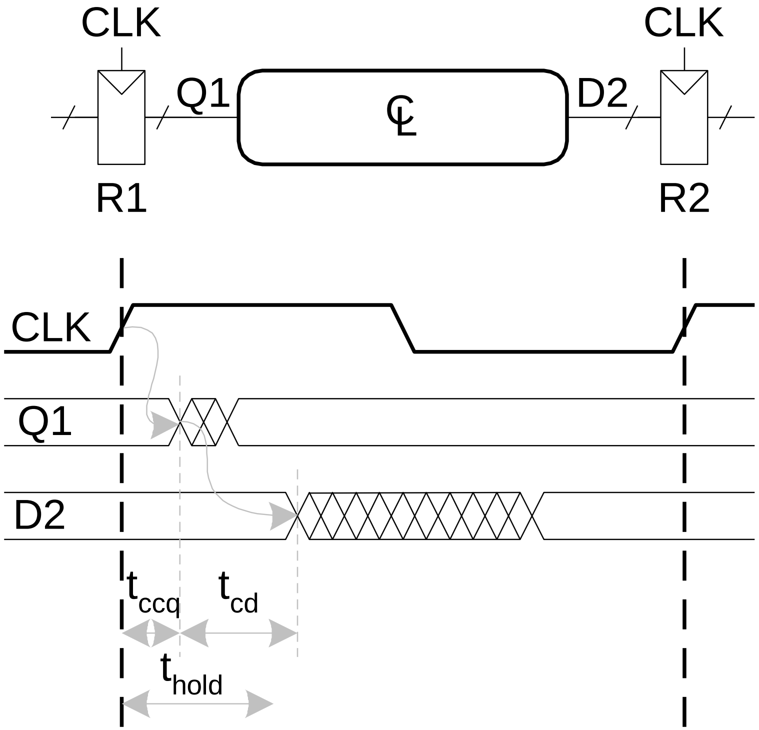


$t_{hold} <$

D2 must not change until sometime t_{hold} after the rising edge of the clock.

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

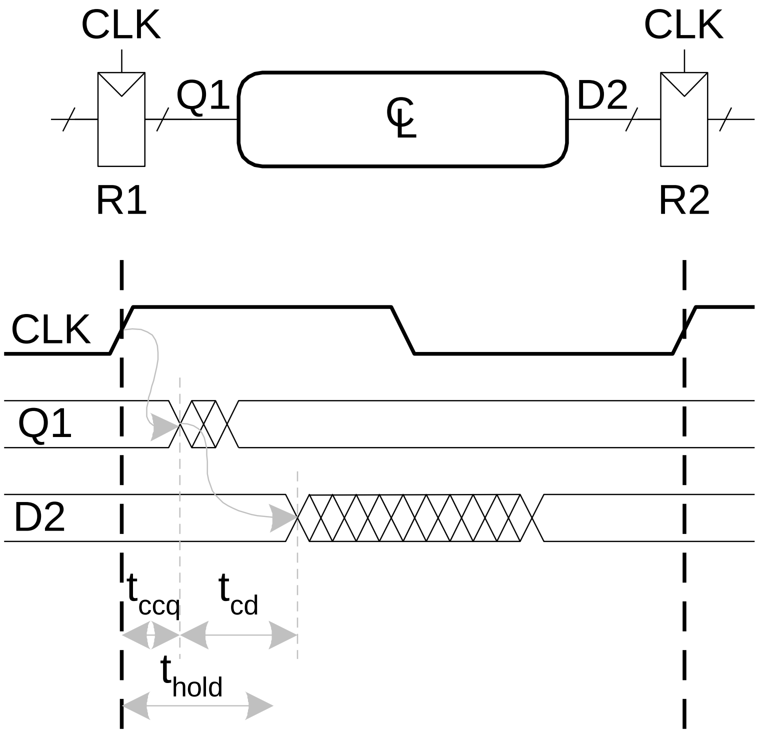


$$t_{hold} < t_{ccq} + t_{cd}$$

$$t_{cd} >$$

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



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

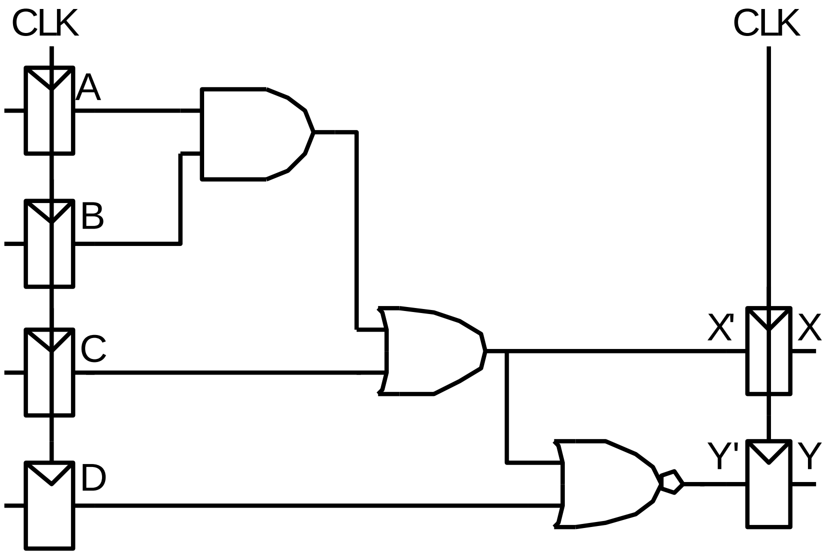
$$t_{cd} > t_{hold} - t_{ccq}$$

t_{ccq} and t_{cd} are specified by manufacturer
Minimum delay constraint

Timing

- If there is no combinational logic between flip-flops (back to back) then tcd=0 and $t_{hold} \le t_{ccq}$
- A reliable flip-flop must have a hold time shorter than its contamination delay.
- Hold time constraint is critical, in case violated then increase the contamination delay through the combinational logic. (requires redesigning the circuit)
- Setup time violations can be fixed by decreasing the propagation delay or increasing the clock period.

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:

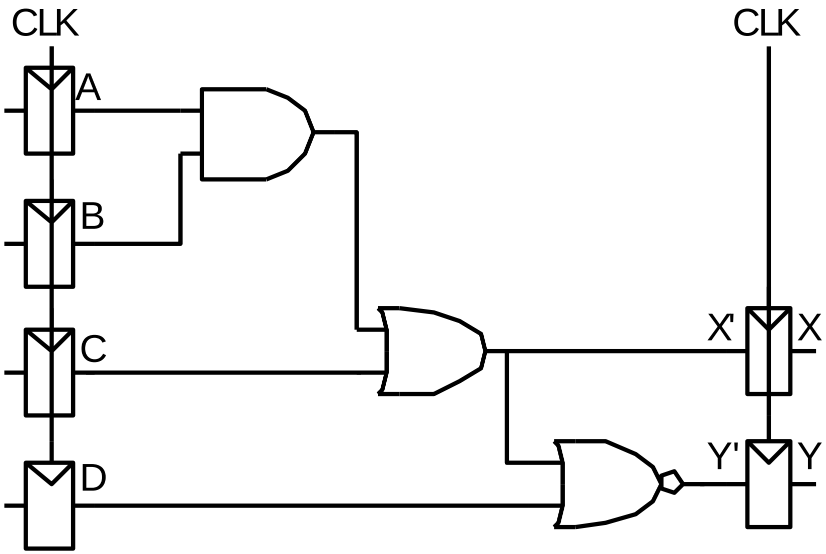
$T_c \geq$

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

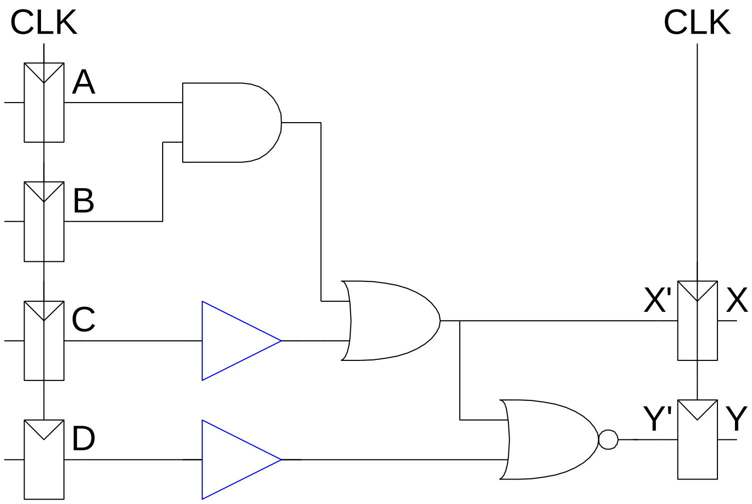
Hold time constraint:

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

(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

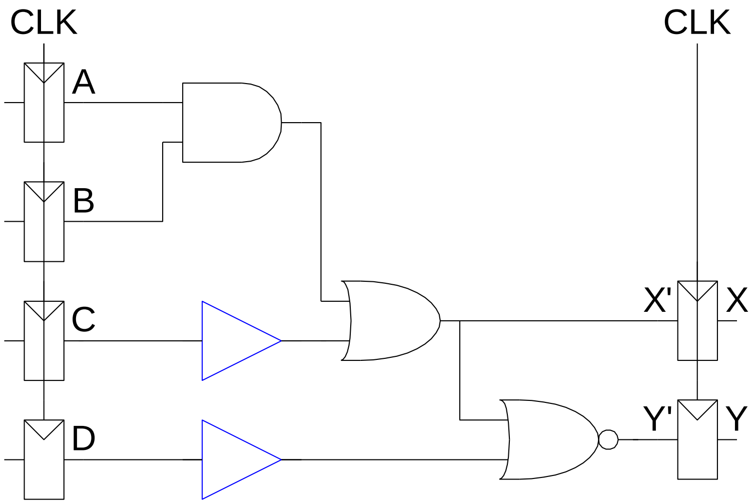
Hold time constraint:

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

SEQUENTIAL LOGIC DESIGN

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!**

ELSEVIER

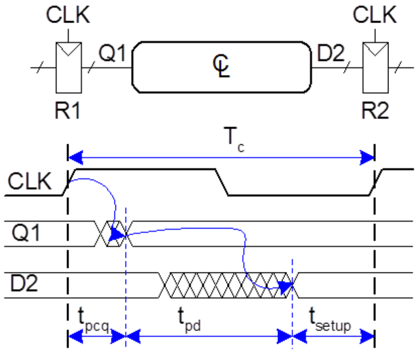
Review$_{12}$

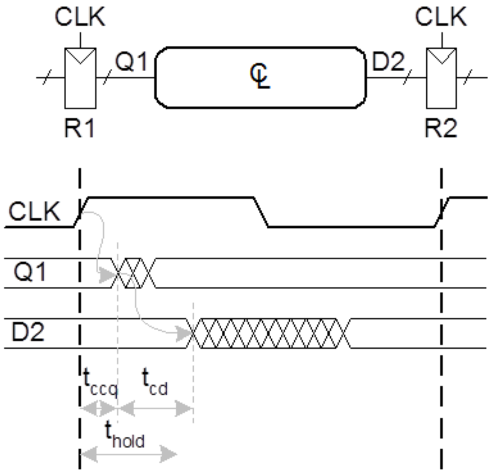
- FSM example
- Common Mistakes when Capturing FSMs
- Timing
- Setup Time Constraint

  $(T_c \geq t_{pcq} + t_{pd} + t_{setup})$
- Hold Time Constraint

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

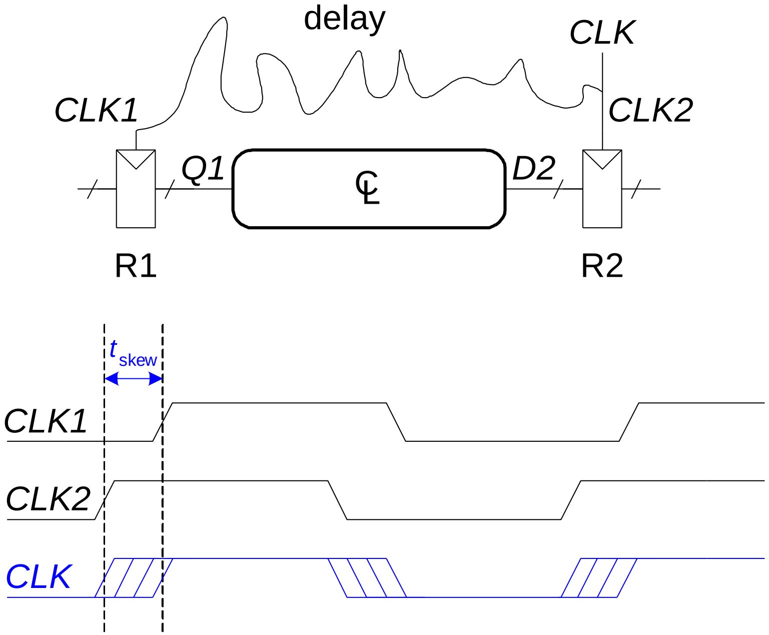
(D2, must not change until some time, t_{hold}, after the rising edge of the clock.)





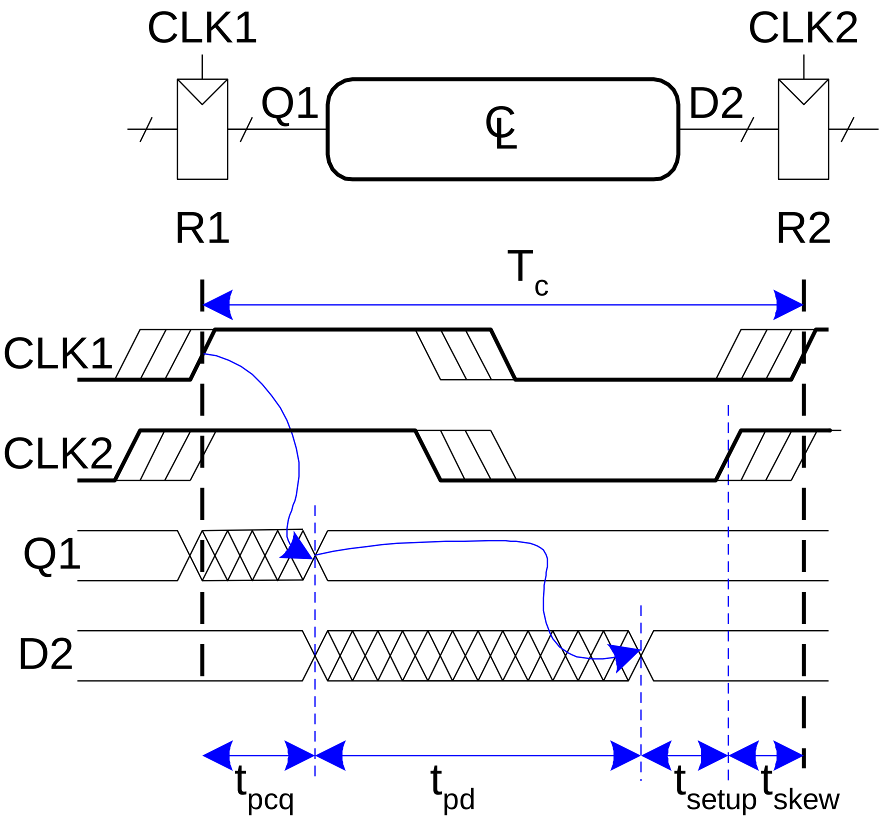
Clock Skew

- 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

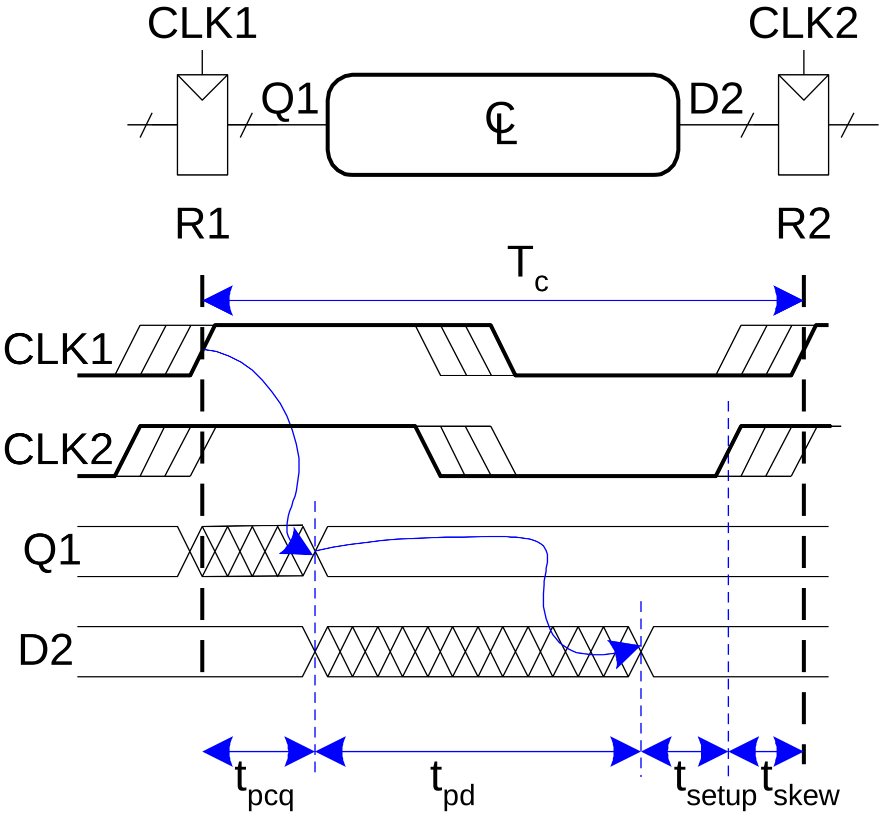
- In the worst case, CLK2 is earlier than CLK1



$T_c \geq$

Setup Time Constraint with Skew

- In the worst case, CLK2 is earlier than CLK1

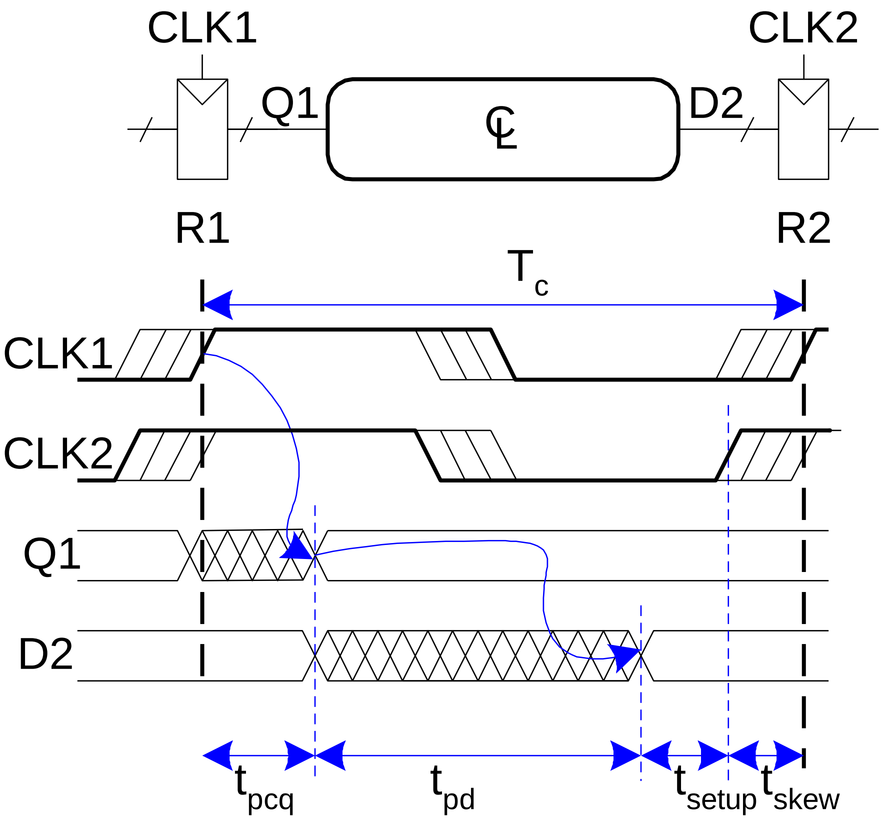


$$T_c \geq t_{pcq} + t_{pd} + t_{setup} + t_{skew}$$

$$t_{pd} \leq$$

Setup Time Constraint with Skew

- In the worst case, CLK2 is earlier than CLK1

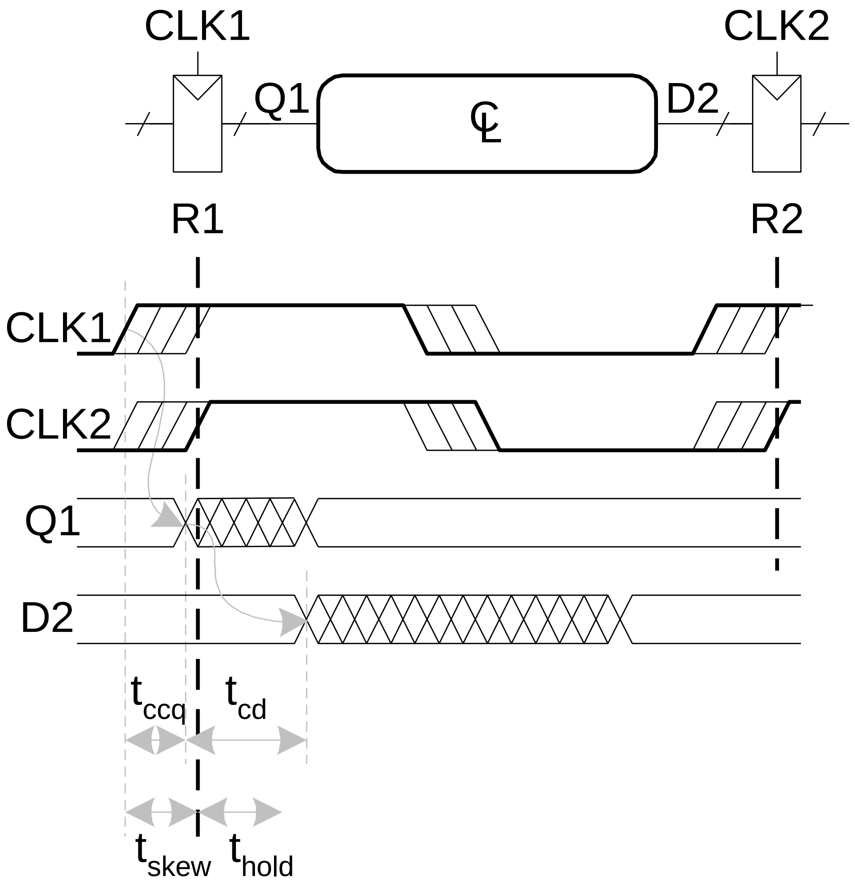


$$T_c \geq t_{pcq} + t_{pd} + t_{setup} + t_{skew}$$

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

Hold Time Constraint with Skew

- In the worst case, CLK2 is later than CLK1

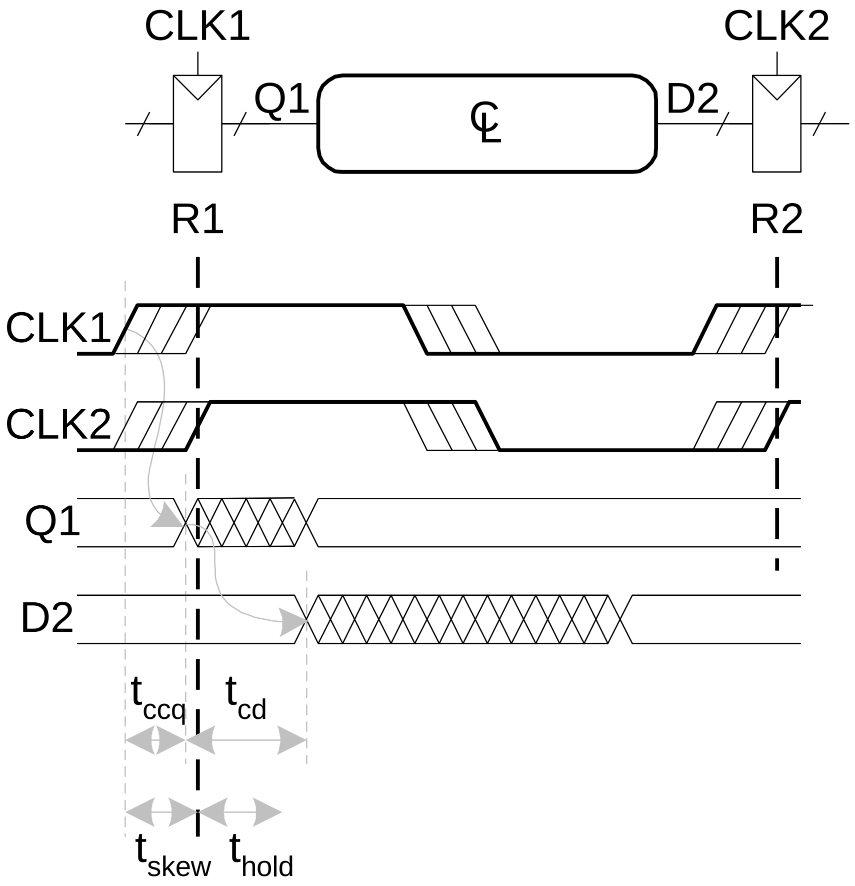


$t_{ccq} + t_{cd} >$

ELSEVIER

Hold Time Constraint with Skew

- In the worst case, CLK2 is later than CLK1

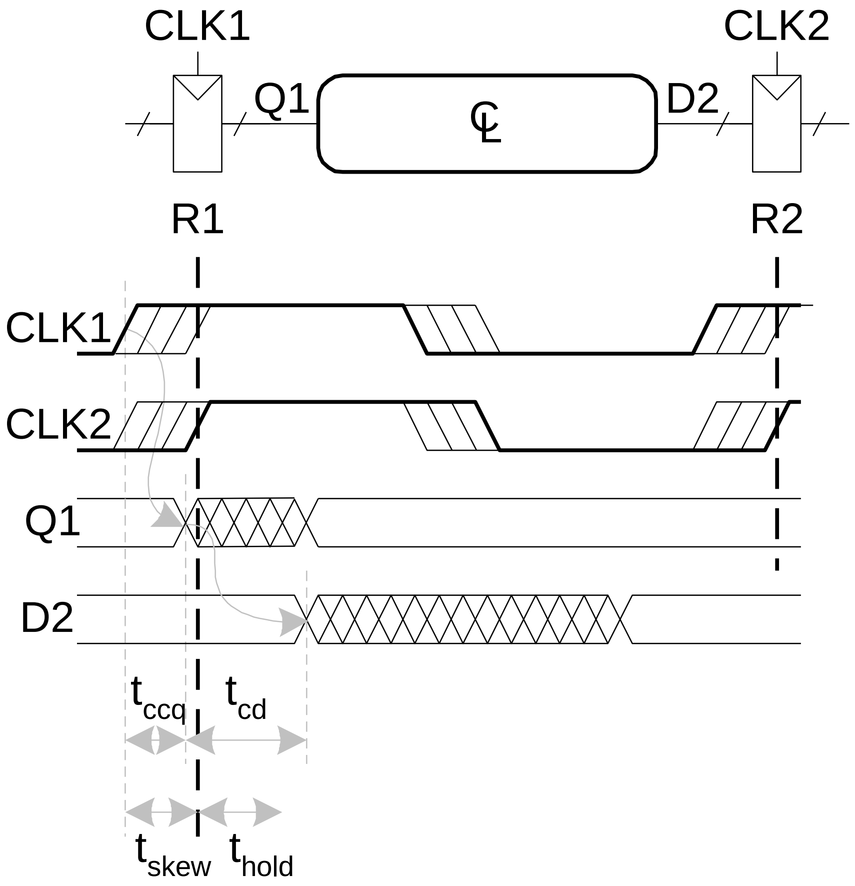


$$t_{ccq} + t_{cd} > t_{\text{hold}} + t_{\text{skew}}$$

$$t_{cd} >$$

Hold Time Constraint with Skew

- In the worst case, CLK2 is later than CLK1



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

Exercise 3.34 You are designing an adder for the blindingly fast 2-bit RePentium Processor. The adder is built from two full adders such that the carry out of the first adder is the carry in to the second adder, as shown in Figure 3.75. Your adder has input and output registers and must complete the addition in one clock cycle. Each full adder has the following propagation delays: 20 ps from C_{in} to C_{out} or to *Sum* (S), 25 ps from A or B to C_{out}, and 30 ps from A or B to S. The adder has a contamination delay of 15 ps from C_{in} to either output and 22 ps from A or B to either output. Each flip-flop has a setup time of 30 ps, a hold time of 10 ps, a clock-to-Q propagation delay of 35 ps, and a clock-to-Q contamination delay of 21 ps.

(a) If there is no clock skew, what is the maximum operating frequency of the circuit?

(b) How much clock skew can the circuit tolerate if it must operate at 8 GHz?

(c) How much clock skew can the circuit tolerate before it might experience a hold time violation?

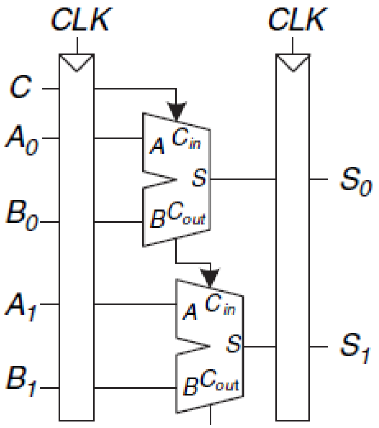
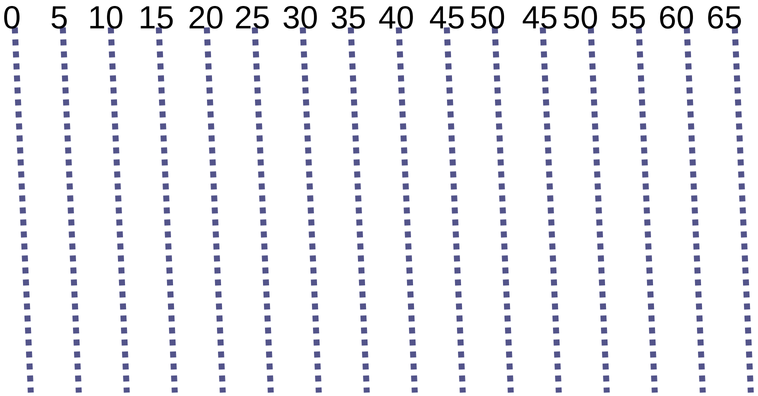


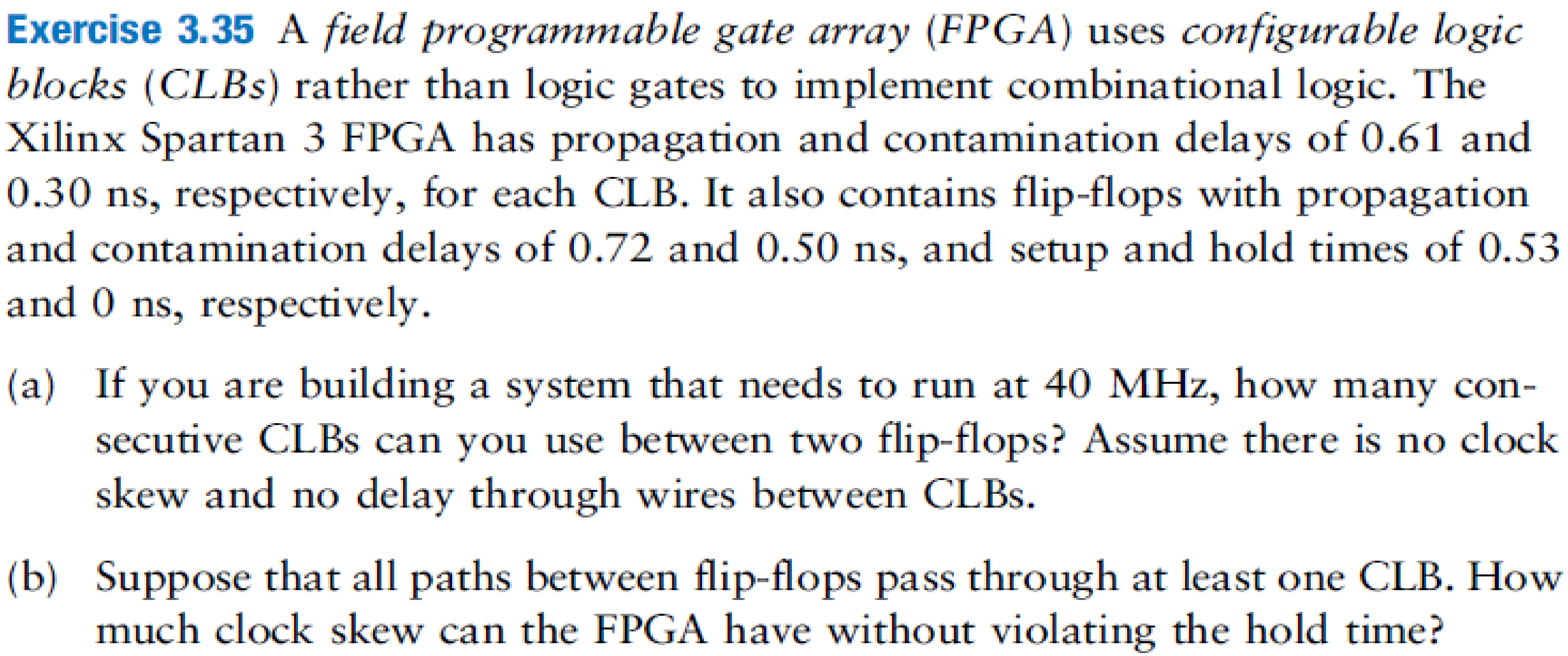
Figure 3.75 **2-bit adder schematic**

Example

Each full adder has the following propagation
delays: 20 ps from Cin to Cout or to Sum (S), 25 ps from A or B to Cout, and 30 ps from A or B to S. The adder has a contamination delay of 15 ps from Cin to either output and 22 ps from A or B to either output. Each flip-flop has a setup time of 30 ps, a hold time of 10 ps, a clock-to-Q propagation delay of 35 ps, and a clock-to-Q contamination delay of 21 ps.

(a) 9.09 GHz
(b) 15 ps
(c) 26 ps





Exercise 3.35 A *field programmable gate array* (*FPGA*) uses *configurable logic blocks* (*CLBs*) rather than logic gates to implement combinational logic. The Xilinx Spartan 3 FPGA has propagation and contamination delays of 0.61 and 0.30 ns, respectively, for each CLB. It also contains flip-flops with propagation and contamination delays of 0.72 and 0.50 ns, and setup and hold times of 0.53 and 0 ns, respectively.

(a) If you are building a system that needs to run at 40 MHz, how many consecutive CLBs can you use between two flip-flops? Assume there is no clock skew and no delay through wires between CLBs.

(b) Suppose that all paths between flip-flops pass through at least one CLB. How much clock skew can the FPGA have without violating the hold time?

Example

- (a) Tc = 1 / 40 MHz = 25 ns

$Tc > t_{pcq} + Nt_{CLB} + t_{setup}$

25 ns > [0.72 + N(0.61) + 0.53] ns

N < 38.9

N = 38

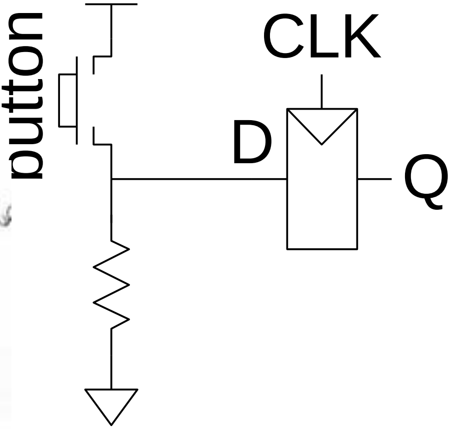
- (b) $t_{skew} < (t_{ccq} + t_{cd\_CLB}) - t_{hold}$

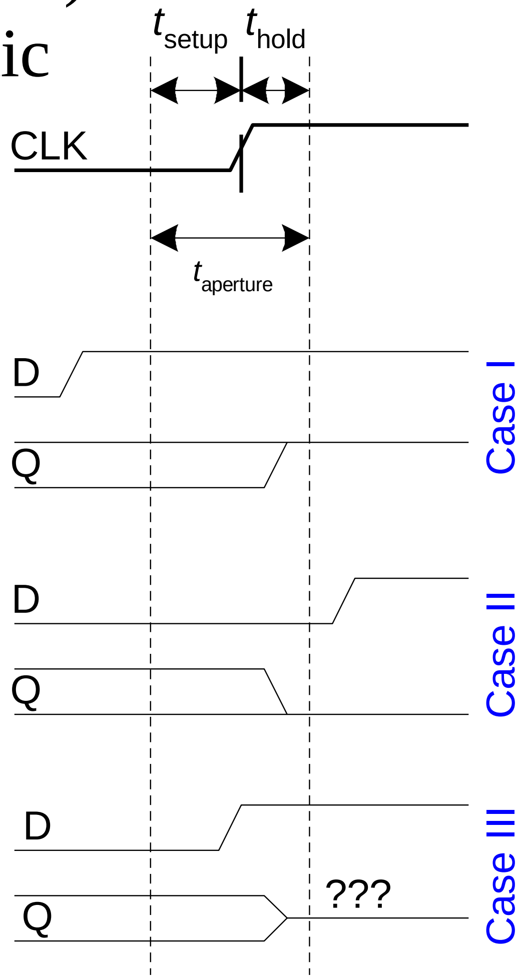
< [(0.5 + 0.3) - 0] ns

< 0.8 ns = 800 ps

Violating the Dynamic Discipline

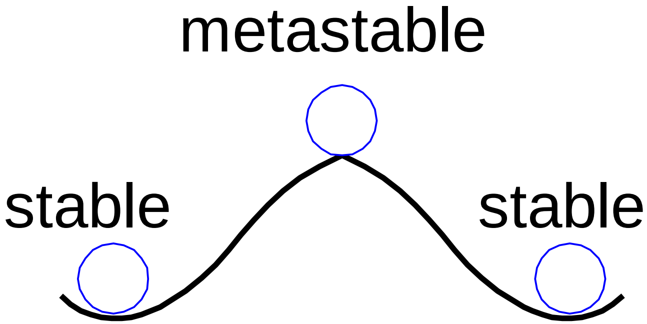
- Asynchronous (for example, user) inputs might violate the dynamic discipline





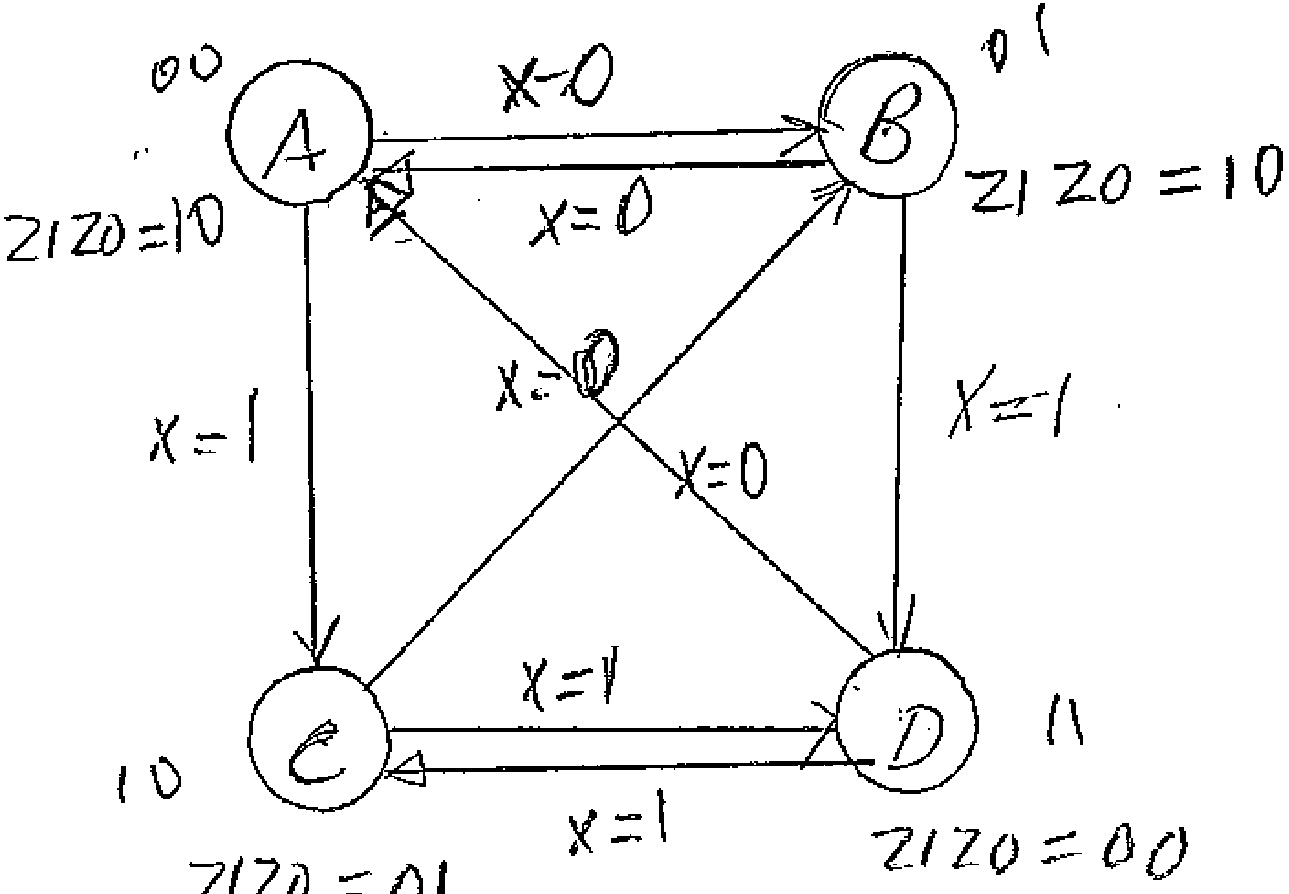
Metastability

- **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



Question 3

The state diagram of a FSM is given below. Design the controller for this FSM using only two 4x1 multiplexers and two 2x1 multiplexers.


00
A
Z1Z0=10
X=0
01
B
Z1 Z0 = 10
X=0
X=0
X=0
X=1
X=1
X=1
X=1
10
C
Z1Z0 = 01
11
D
Z1Z0 = 00