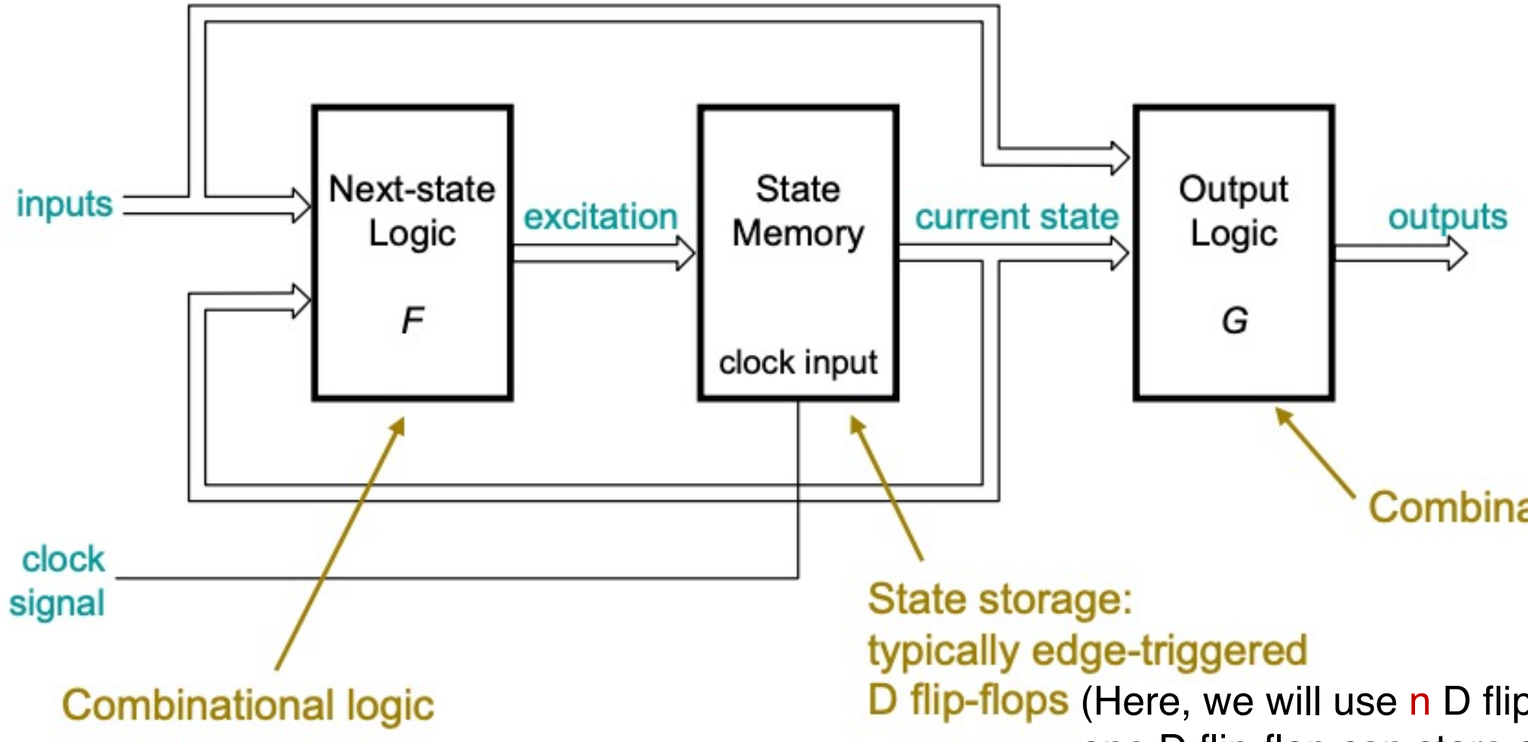
Canonical Form: Mealy and Moore Machines

Mealy FSM

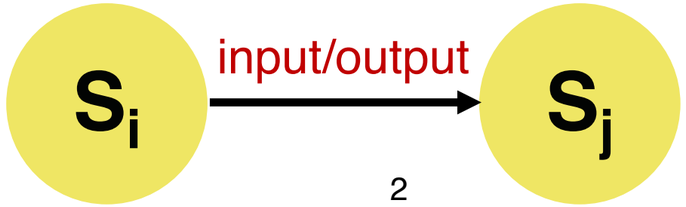
Mealy machine: output depends on state and current input

Next state = F (current state, input); Output = G (current state, input)



(Here, we will use n D flip-flops to store 2^n states as one D flip-flop can store one bit.)

State transition on Mealy machine:



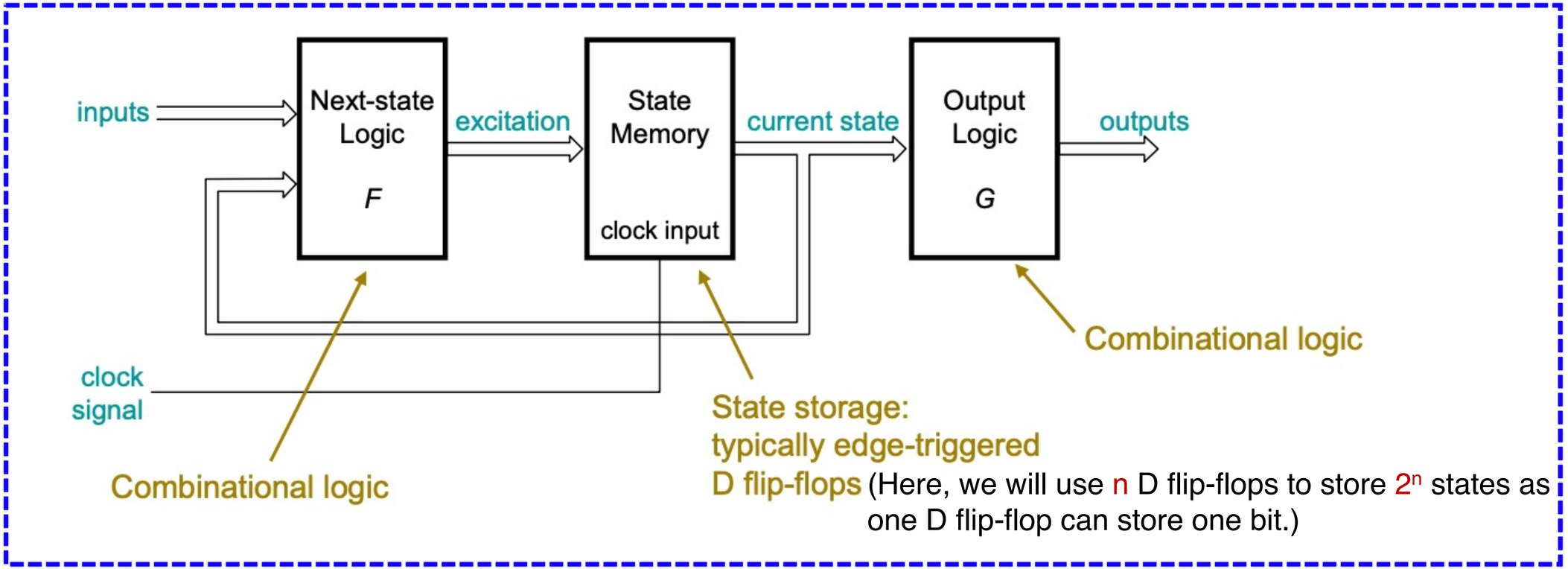
S_i and S_j are two states

Figure is adapted from Ivan Marsic @Rutgers University

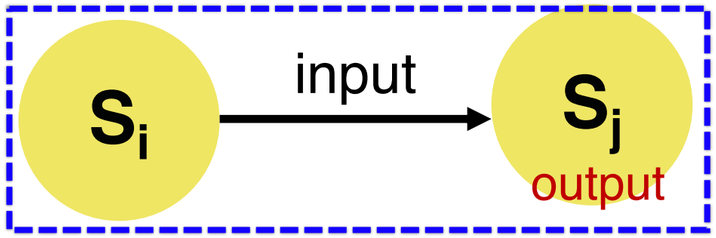
Moore FSM

Moore machine: output depends only on current state

Next state = F (current state, input); Output = G (current state)



State transition on Moore machine:



OR

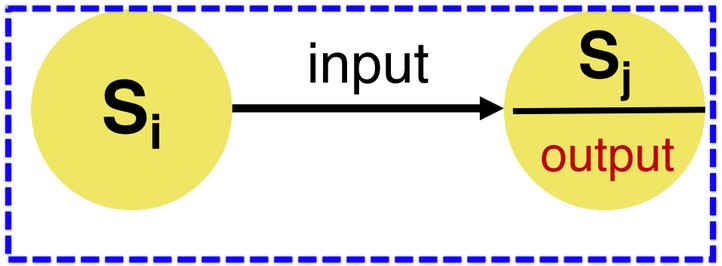
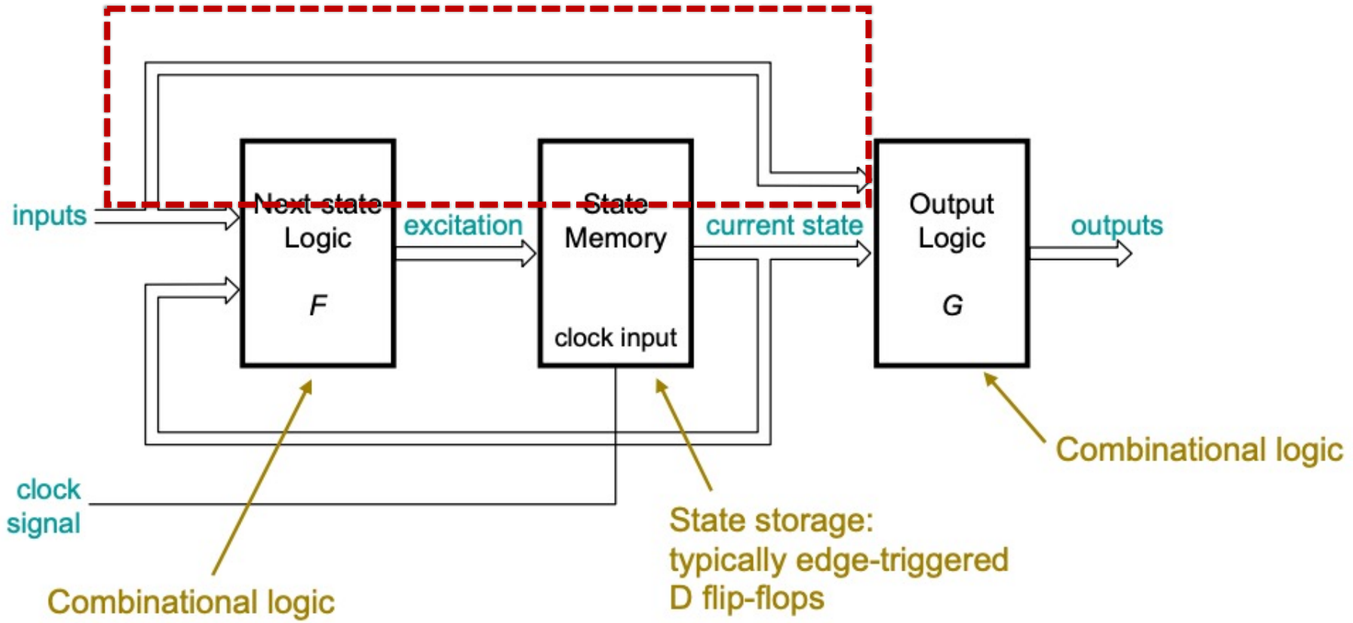


Figure is adapted from Ivan Marsic @Rutgers University

Mealy machine vs Moore machine

Mealy



Moore

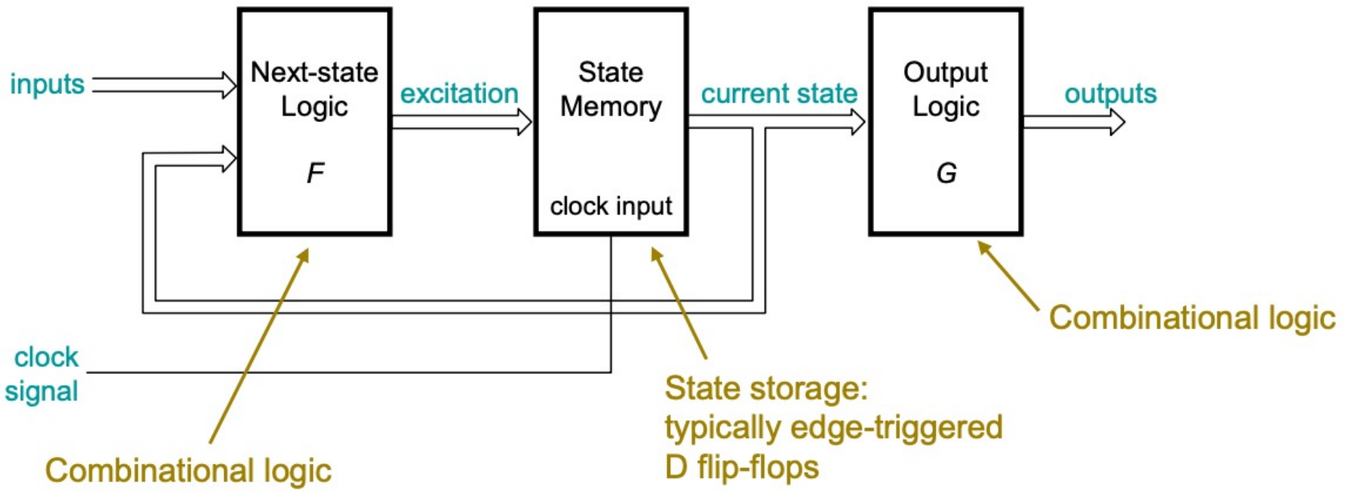
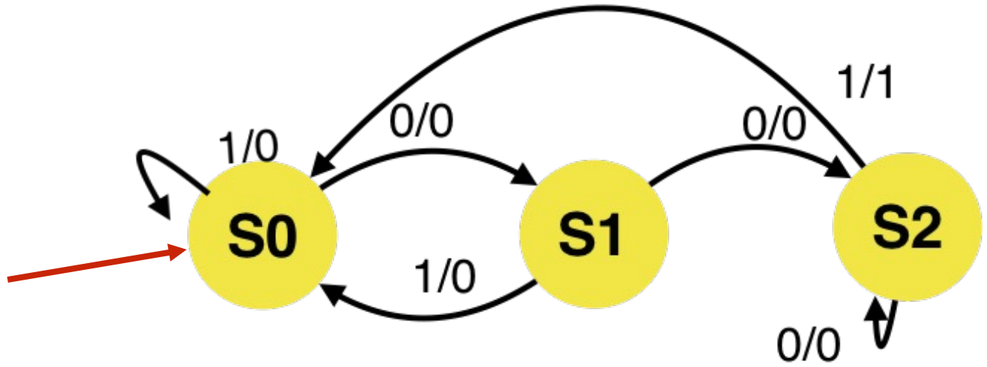


Figure is adapted from Ivan Marsic @Rutgers University

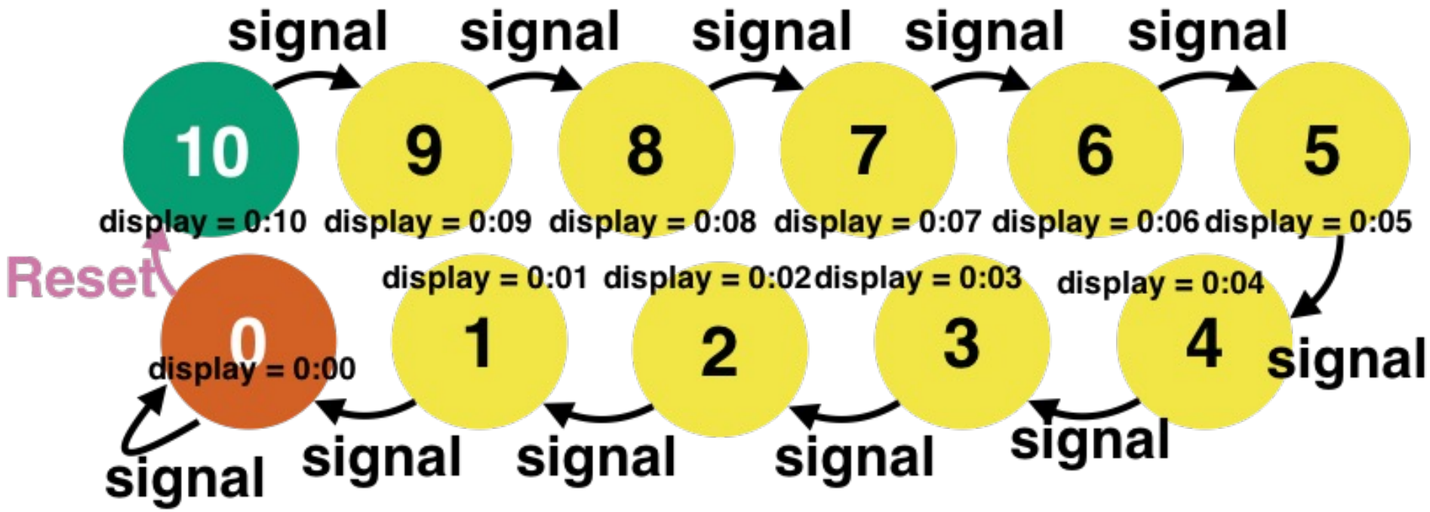
Initial state of a FSM



- An arrow points to the initial state, i.e., S0 is the initial state
- Not every FSM diagram specifies the initial state.
- Not every initial state is called S_0.

How to make FSM true?

Revisit the FSM for a timer



What do we need to physically implement the timer?

1) Logic to display the remaining time
   — use combinational logic, e.g., 7-segment display

2) Logic that uses the "*current state*" and "*new input(s)*" to transit to a "*new state*" and generate the "*output(s)*"
   — use combinational logic

3) Logic to keep track of the "current state"
   — need **memory**

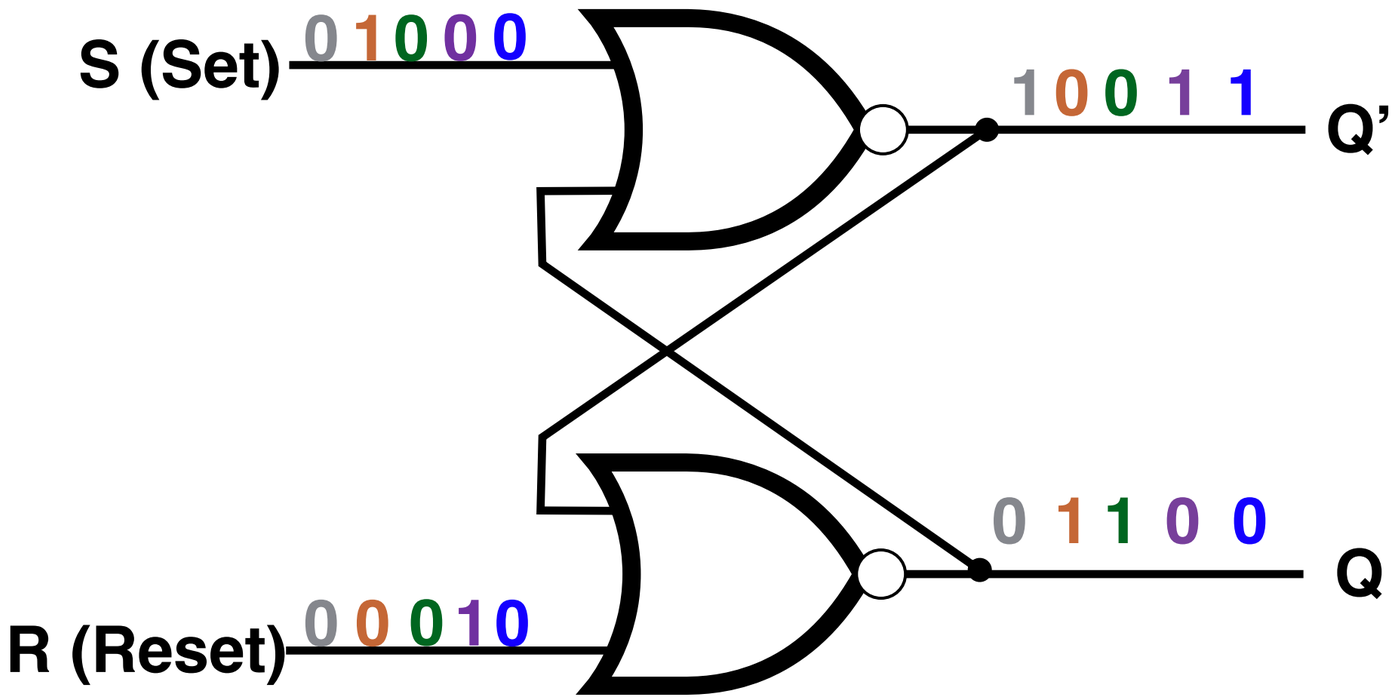
4) A control signal that helps us to transit from current state to next state state at the right time
   — need **clock**

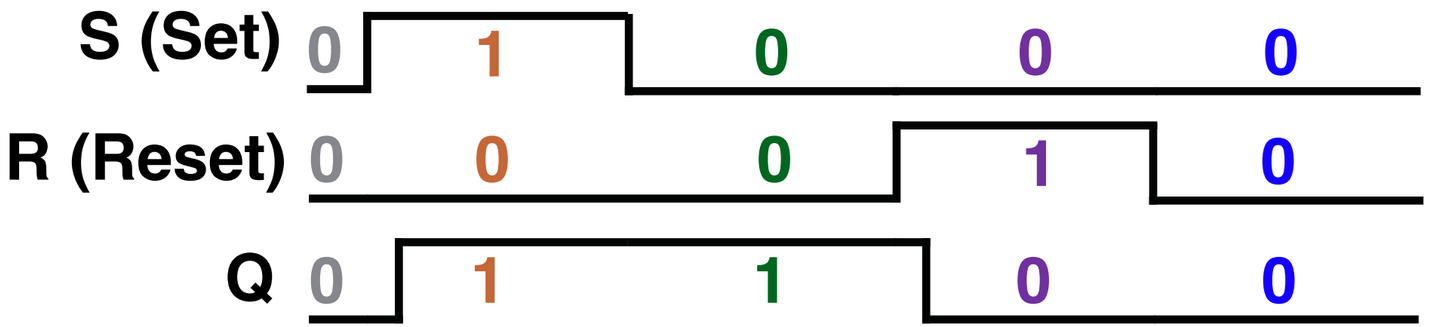
The basic form of memory

SR-Latch: the very basic “memory”

NOR gate

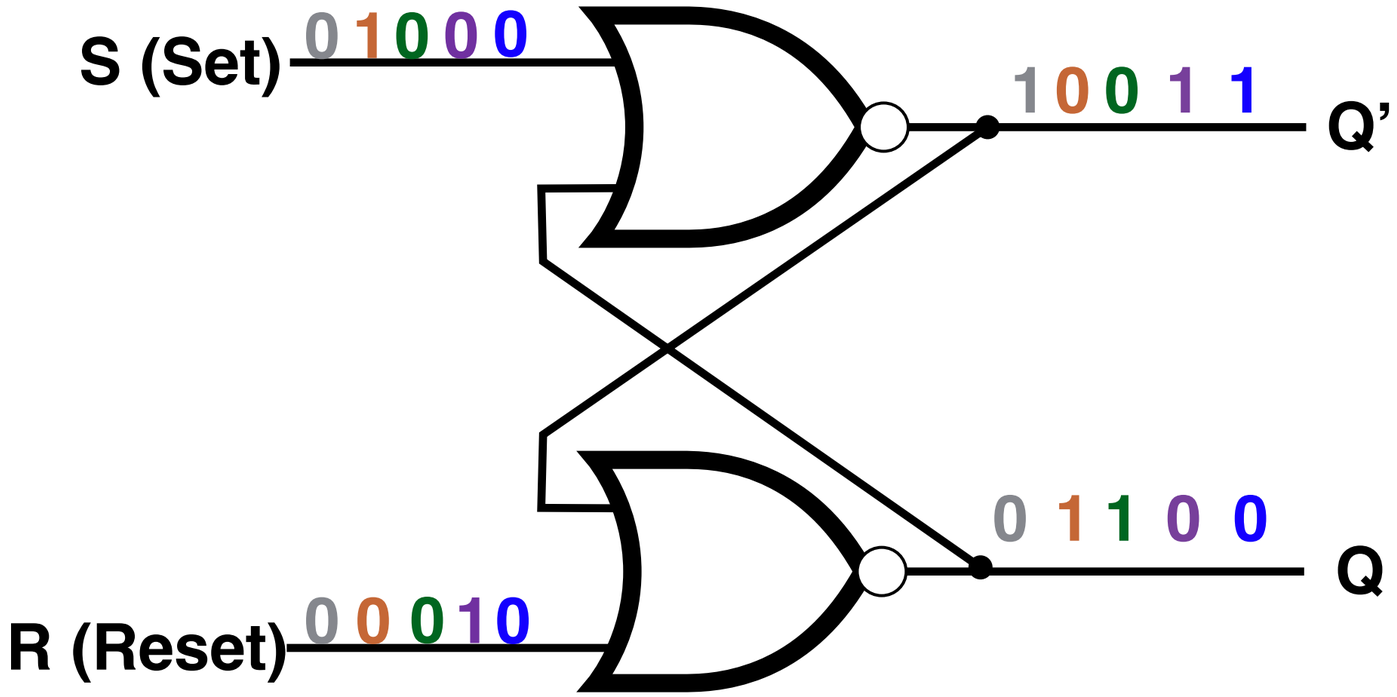
Input		Output
A	B	
0	0	**1**
0	1	**0**
1	0	**0**
1	1	**0**

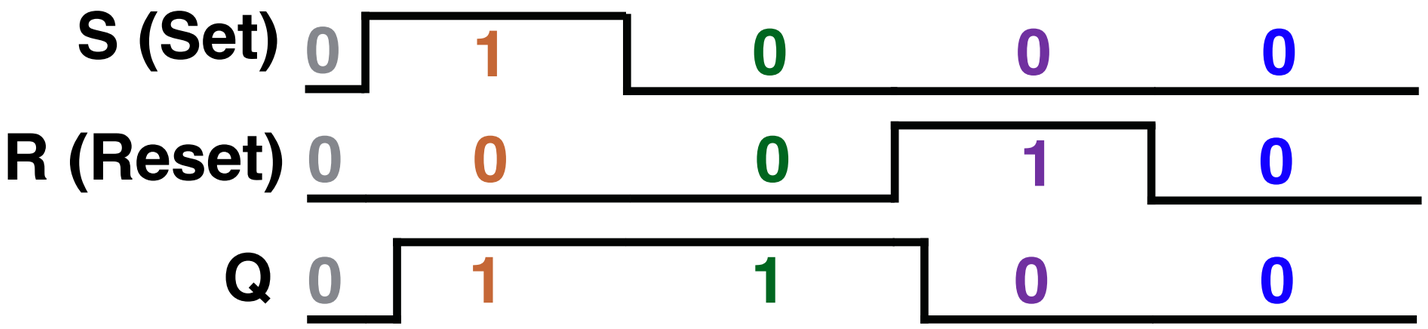




Timing diagram

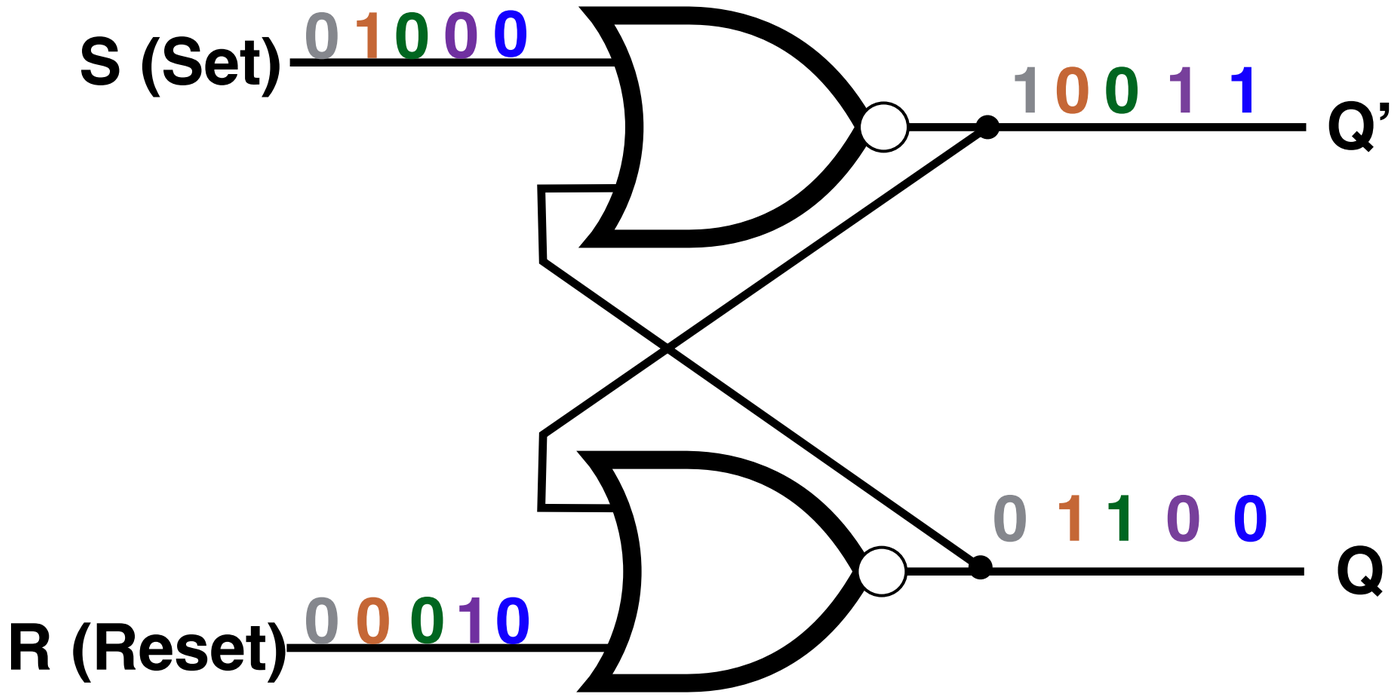
SR-Latch: the very basic “memory”

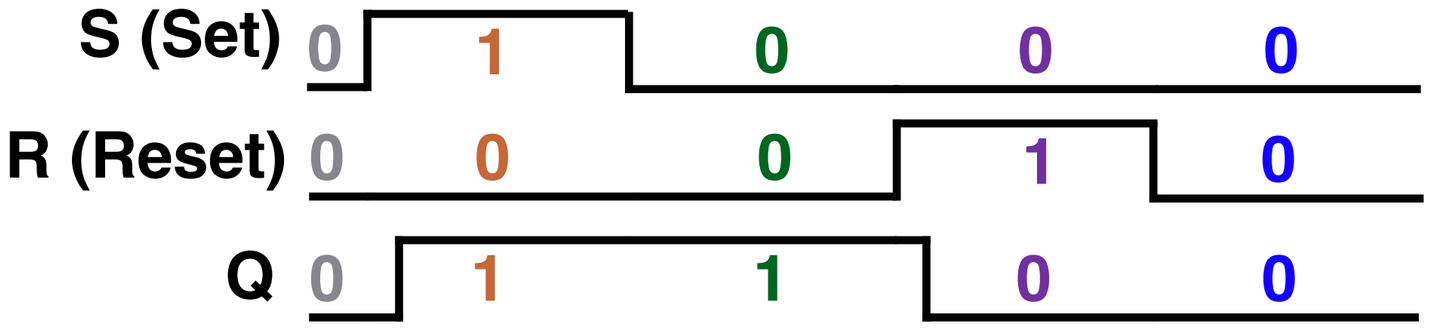




S	R	Q(t)	Q(t+1)
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	
1	1	1	

SR-Latch: the very basic “memory”



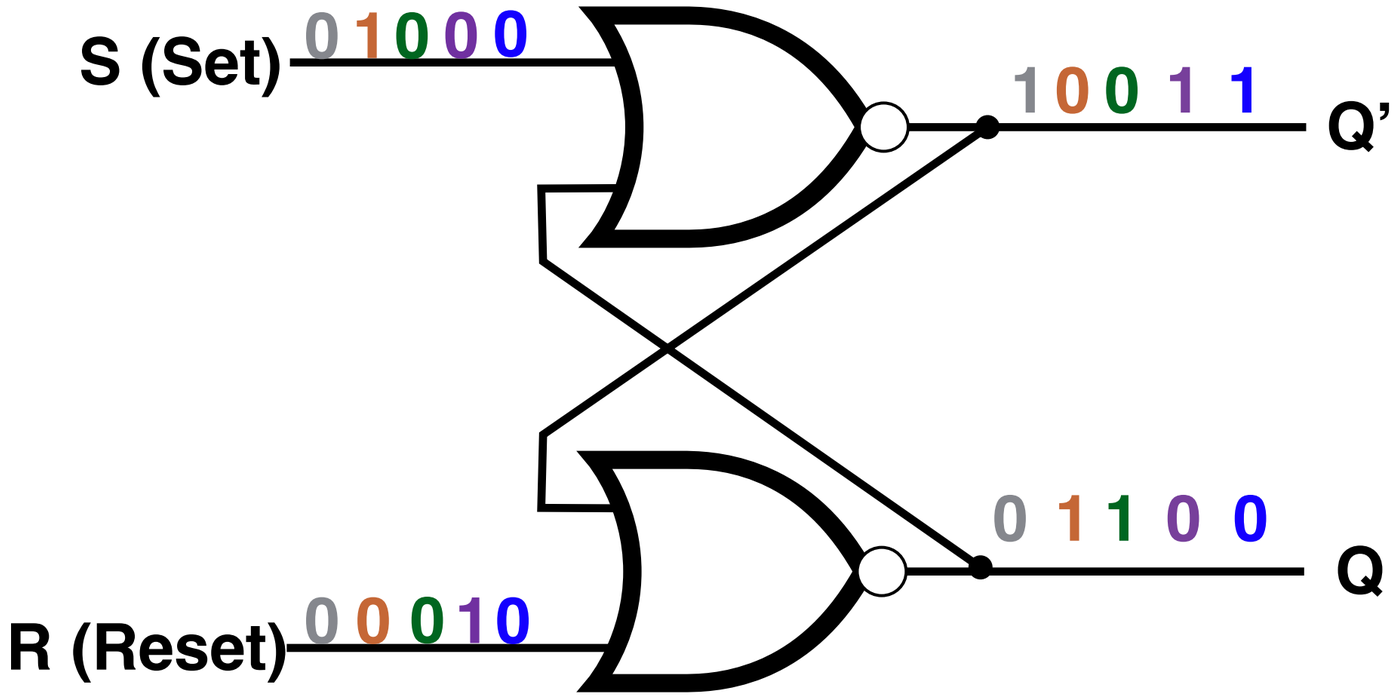


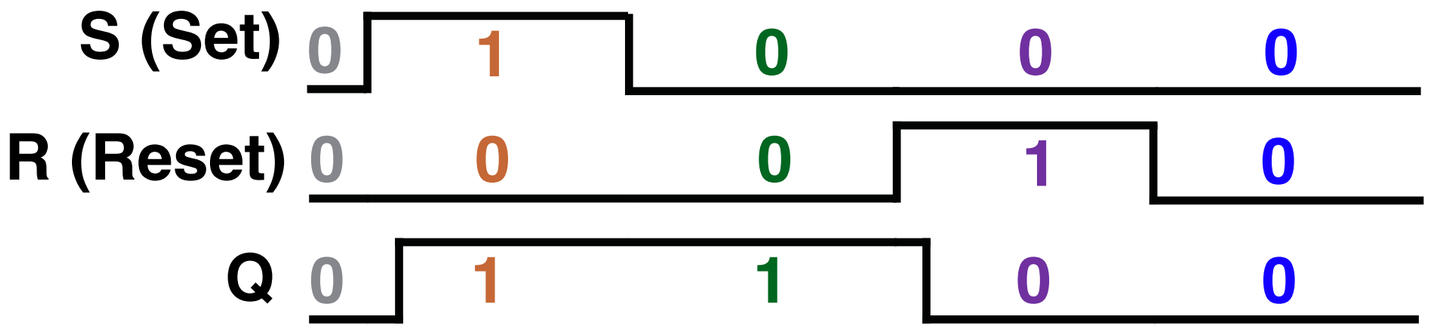
S	R	Q(t)	Q(t+1)
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	
1	1	1	

Q(t+1) = 1 when S=1, R=0, regardless of Q(t)

Set — Make the “stored bit 1”

SR-Latch: the very basic “memory”



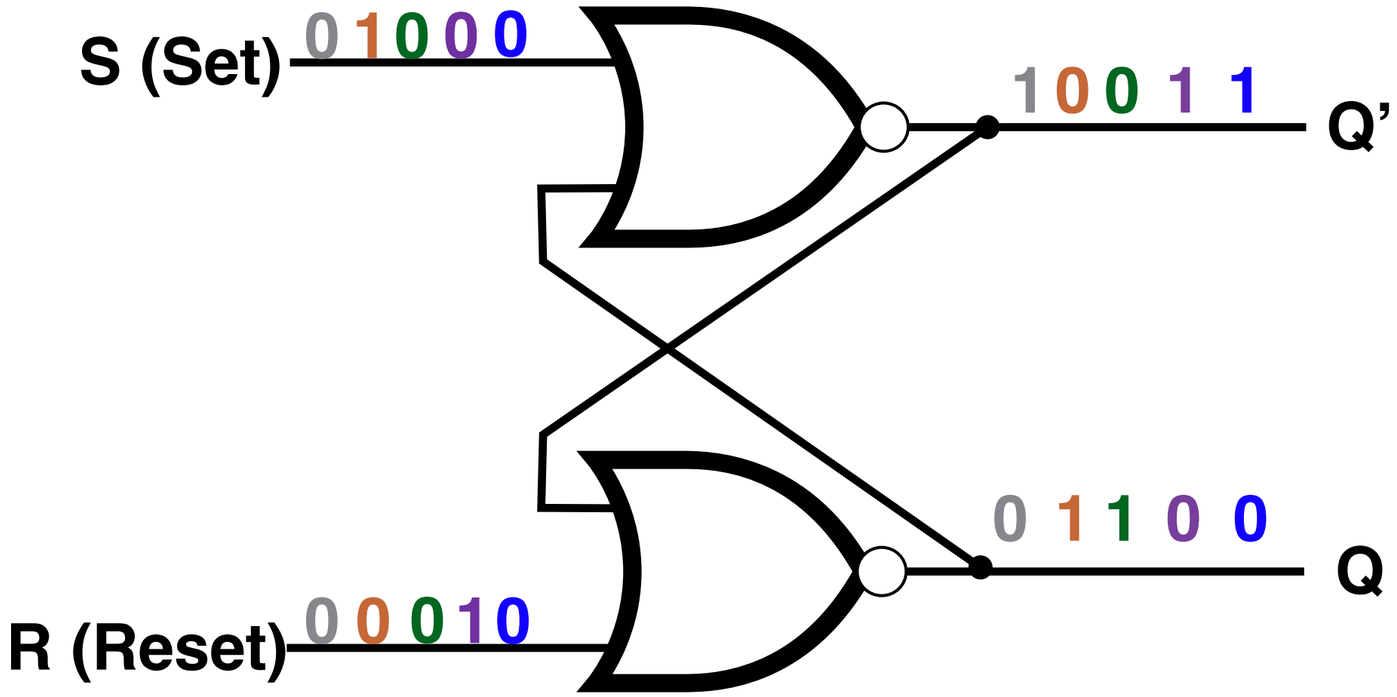


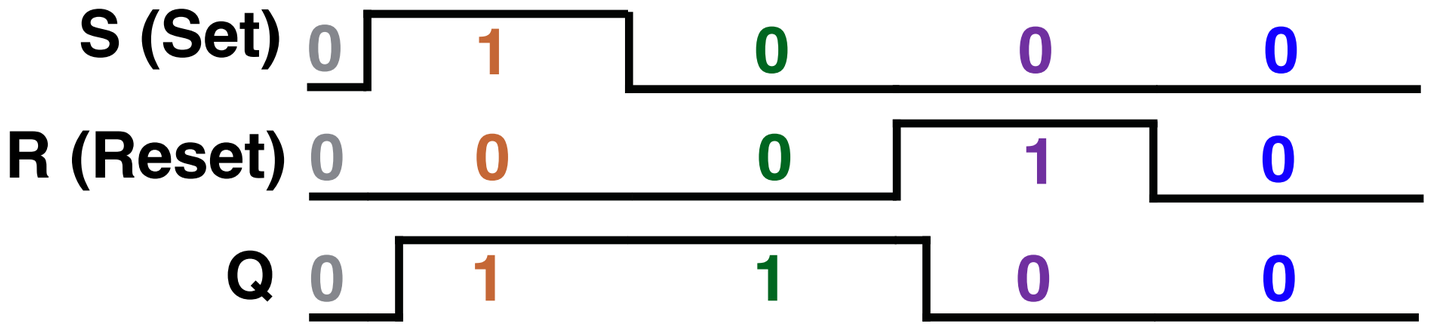
S	R	Q(t)	Q(t+1)
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	
1	1	1	

Q(t+1) = 0 when R=1, S=0, regardless of Q(t)

Reset — Make the “stored bit 0”

SR-Latch: the very basic “memory”



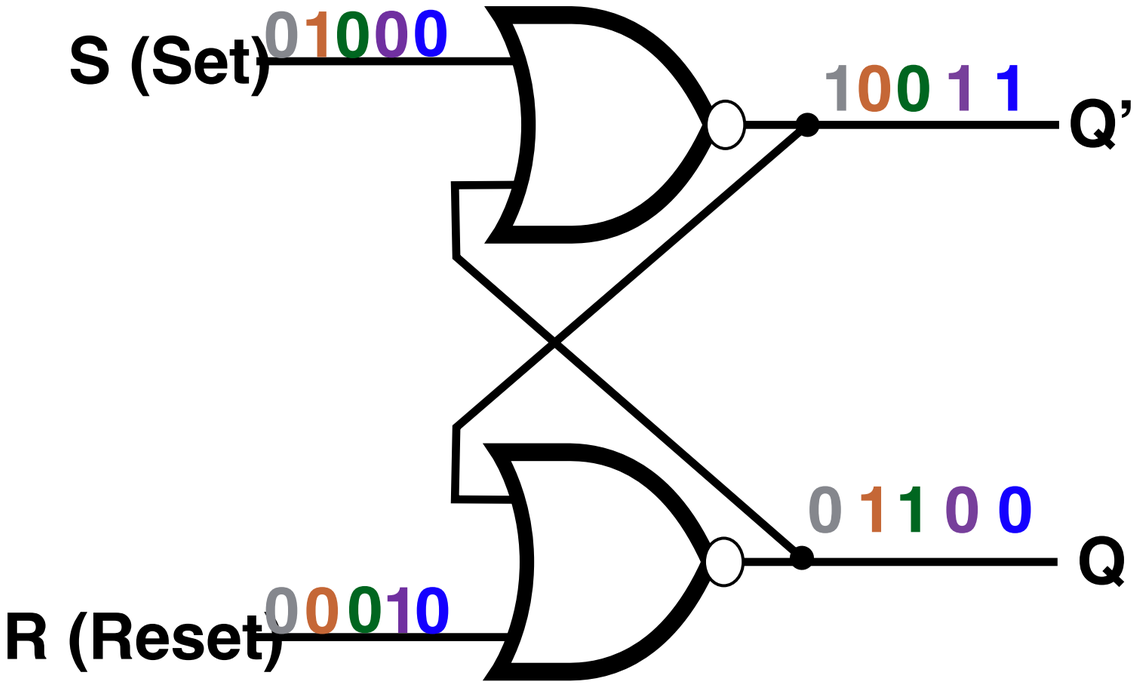


S	R	Q(t)	Q(t+1)
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	
1	1	1	

Q(t+1) = Q(t) when R=0, S=0

Hold — Store the previous value

SR-Latch: the very basic “memory”



S	R	Q(t)	Q(t+1)
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	
1	1	1	

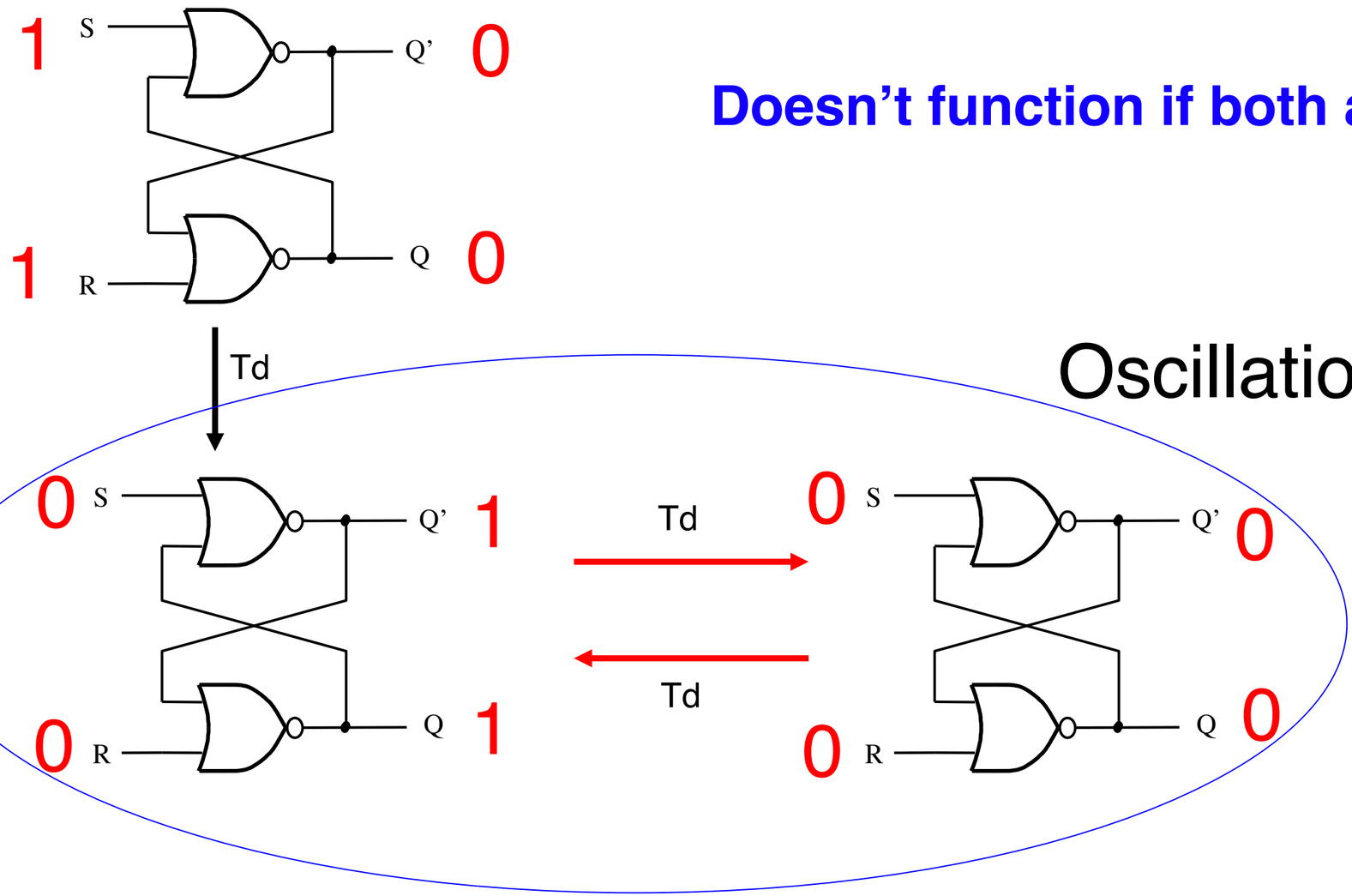
Set — Make the “stored bit 1”
Reset — Make the “stored bit 0”
Hold — Store the previous value

The circuit has memory!

What if both S & R are “1”?

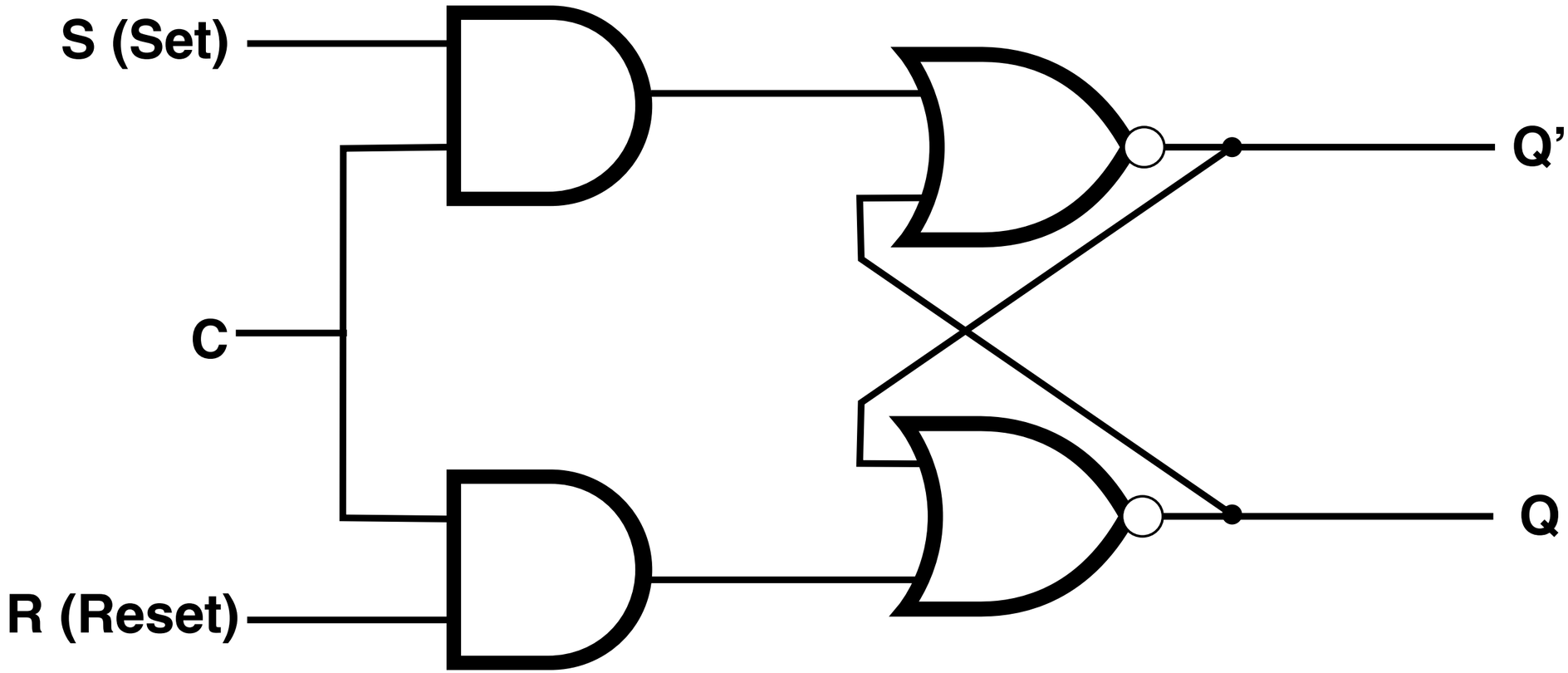
NOR gate

Input		Output
A	B	
0	0	**1**
0	1	**0**
1	0	**0**
1	1	**0**



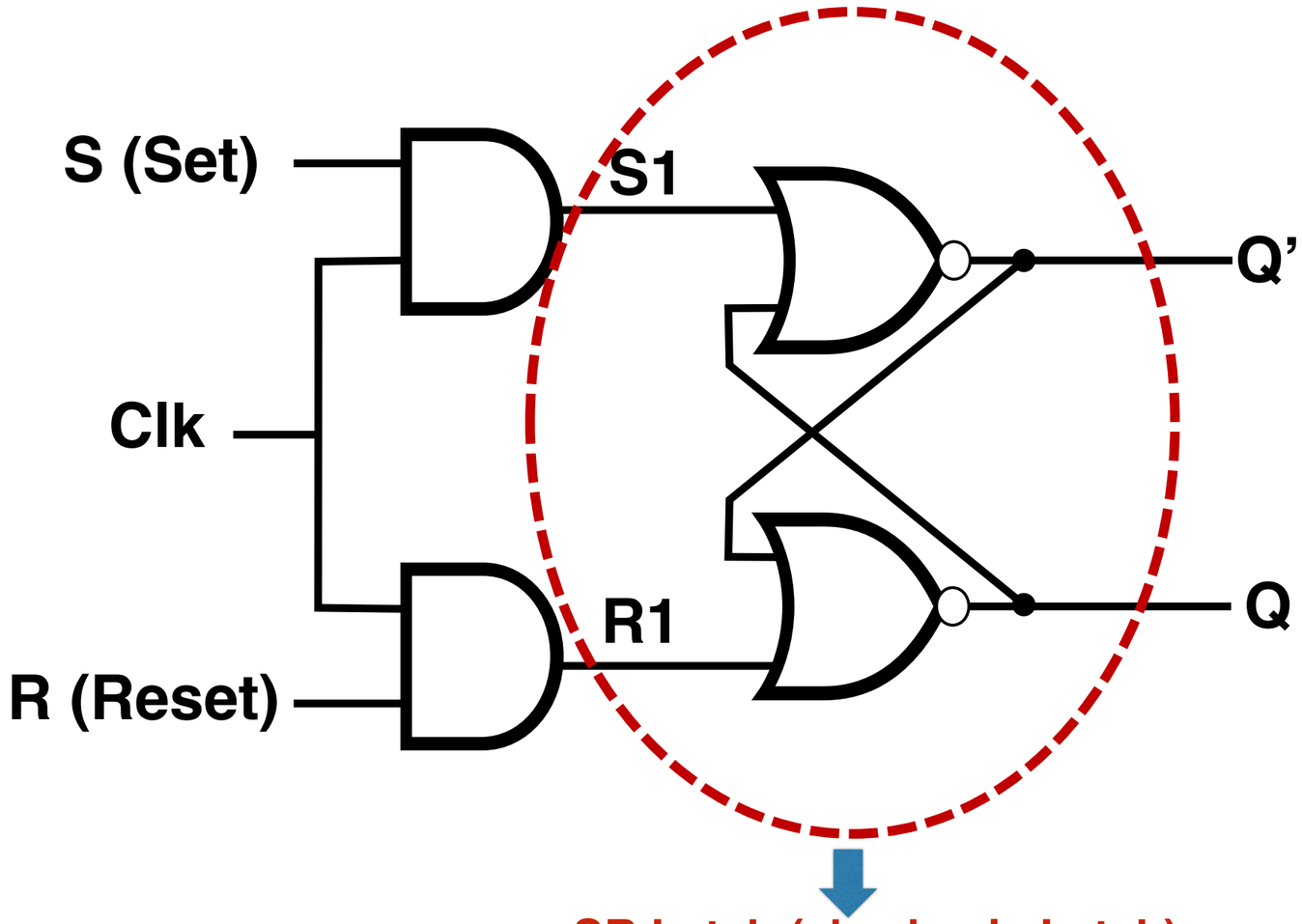
Doesn’t function if both are "1"!

Add an input! — Level-Sensitive SR Latch



- C is a signal aware of the **timing** of gates — **clock**

Gated SR Latches

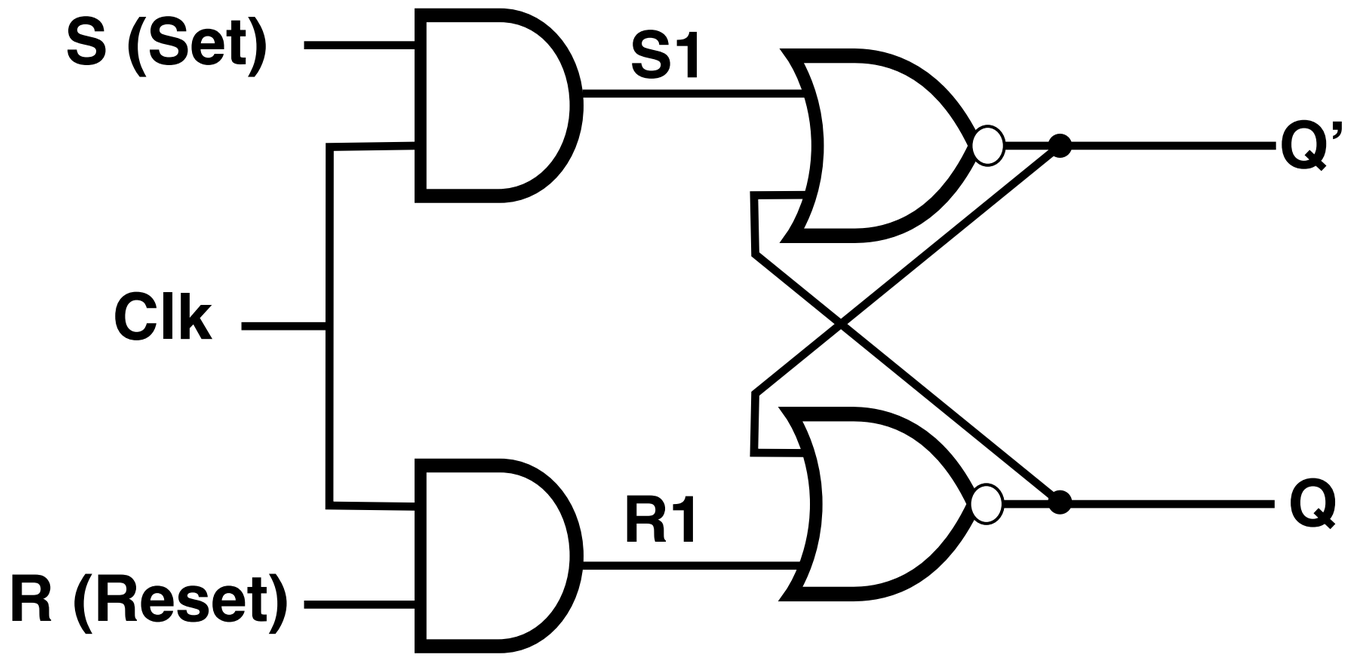


SR Latch (aka, basic Latch)

Clk	S	R	Q (t + 1)
0	x	x	Q(t)
1	0	0	Q(t)
1	0	1	0
1	1	0	1
1	1	1	x

Characteristic table

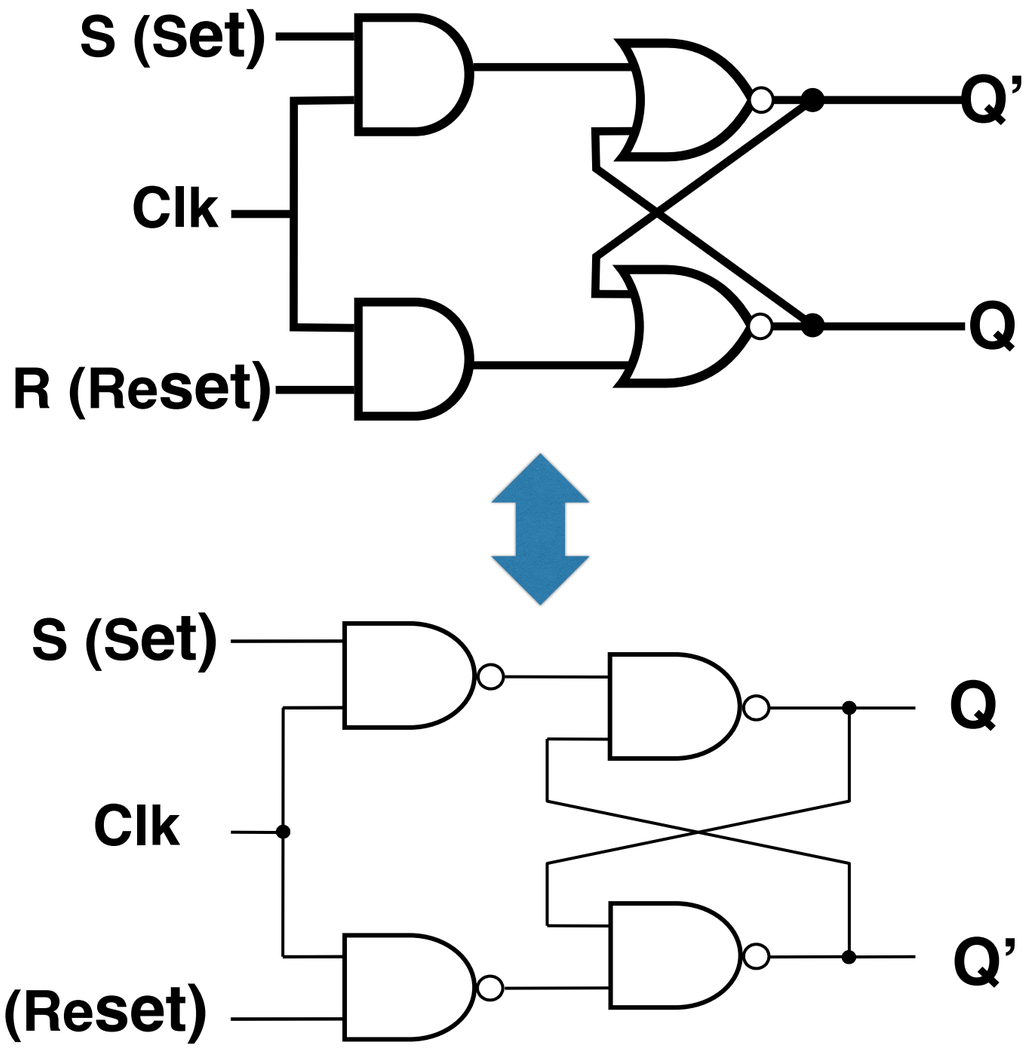
Gated SR Latches

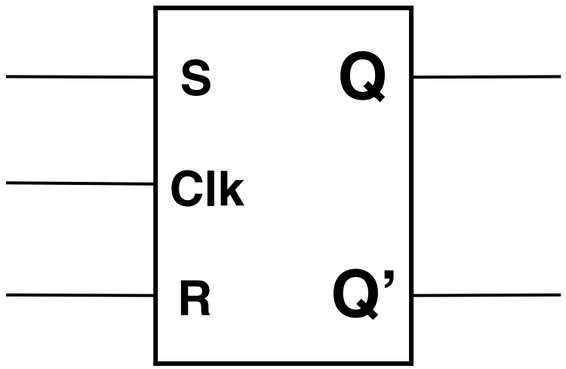


Clk	S	R	Q (t + 1)
0	x	x	Q(t)
1	0	0	Q(t)
1	0	1	0
1	1	0	1
1	1	1	x

When Clk = 1, Gated SR Latch = SR Latch

Two designs of the Gated SR Latches

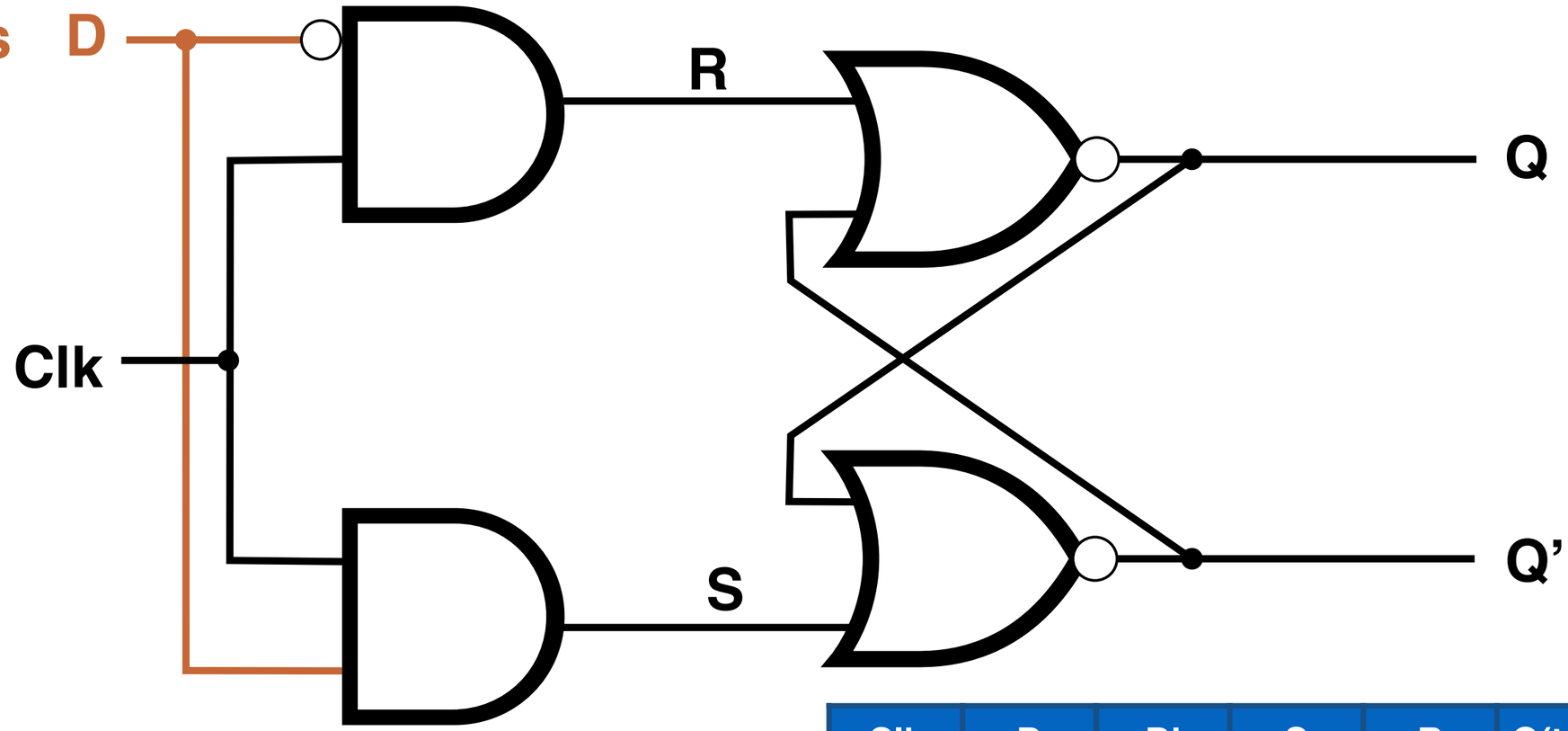




Graphical symbol

D-Latch

We will never get 1, 1 in this way



Clk	D	D'	S	R	Q(t+1)	Q(t+1)'
0	X	X'	0	0	Q(t)	Q(t)'
1	0	1	0	1	0	1
1	1	0	1	0	1	0

D-Latch

Clk	D	D’	S	R	Q(t+1)	Q(t+1)’
0	X	X’	0	0	Q(t)	Q(t)’
1	0	1	0	1	0	1
1	1	0	1	0	1	0

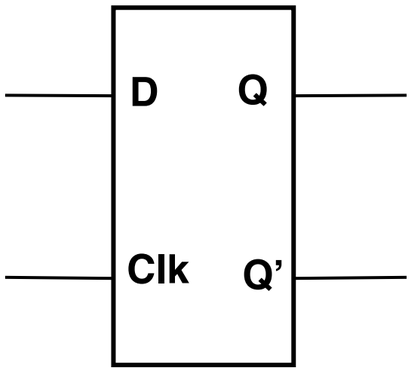
Note:

Q(t+1) = Q(t) when Clk=0

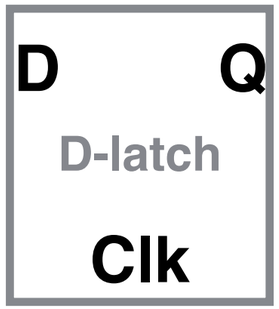
Q(t+1) = D when Clk=1

Clk	D	Q(t + 1)
0	x	Q(t)
1	0	0
1	1	1

Characteristic table



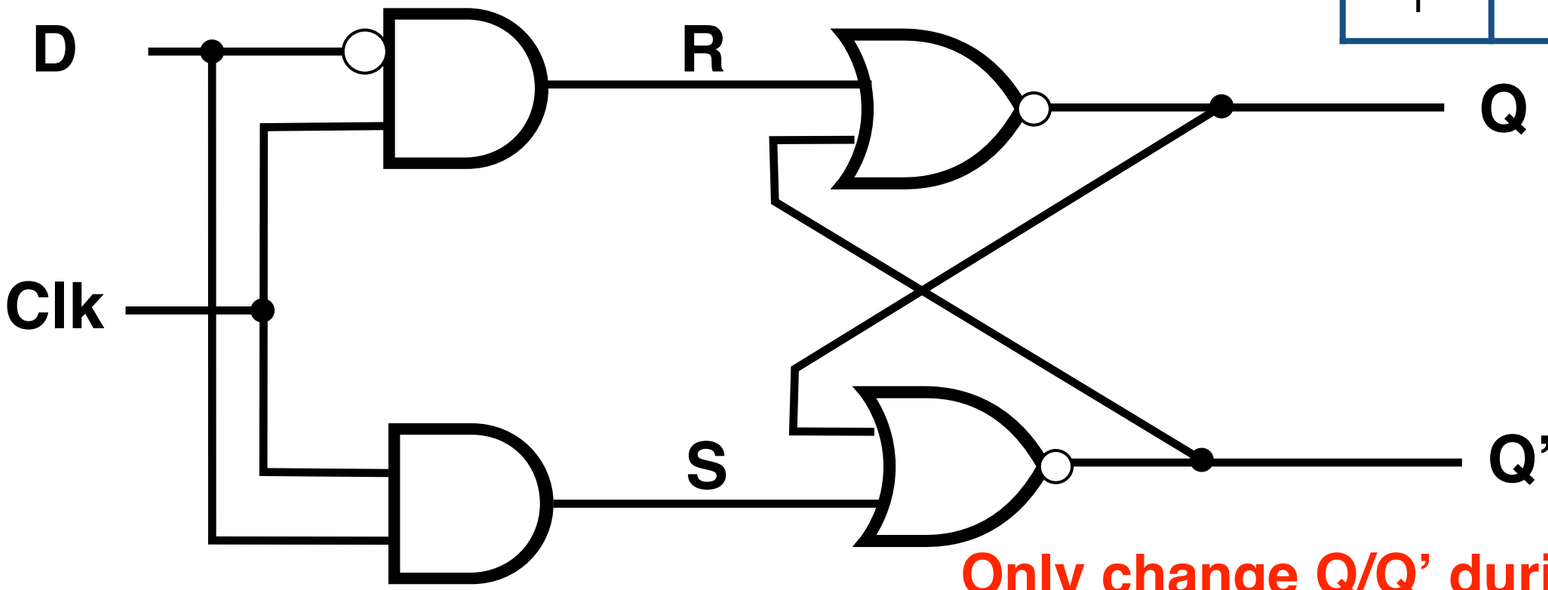
OR



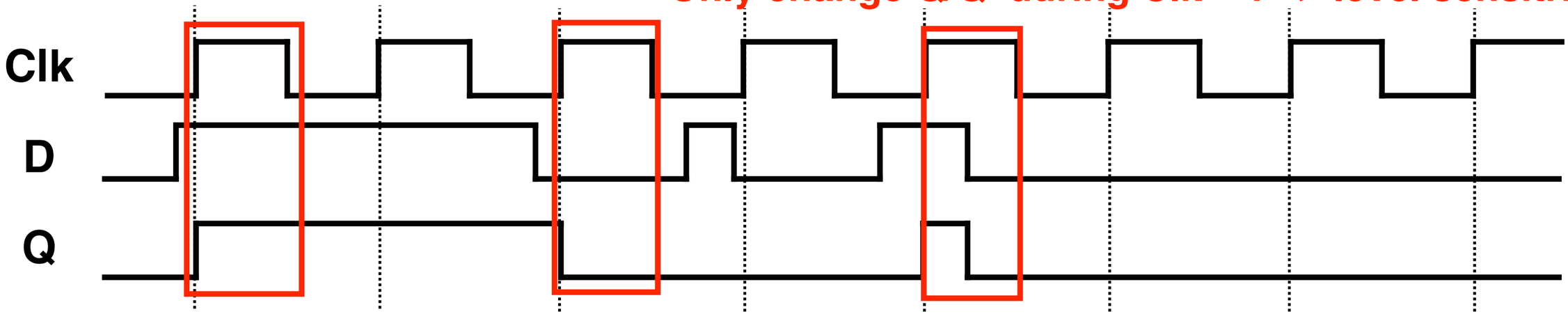
Graphical symbol

D-Latch

Clk	D	D’	S	R	Q(t+1)	Q(t+1)’
0	X	X’	0	0	Q(t)	Q(t)’
1	0	1	0	1	0	1
1	1	0	1	0	1	0



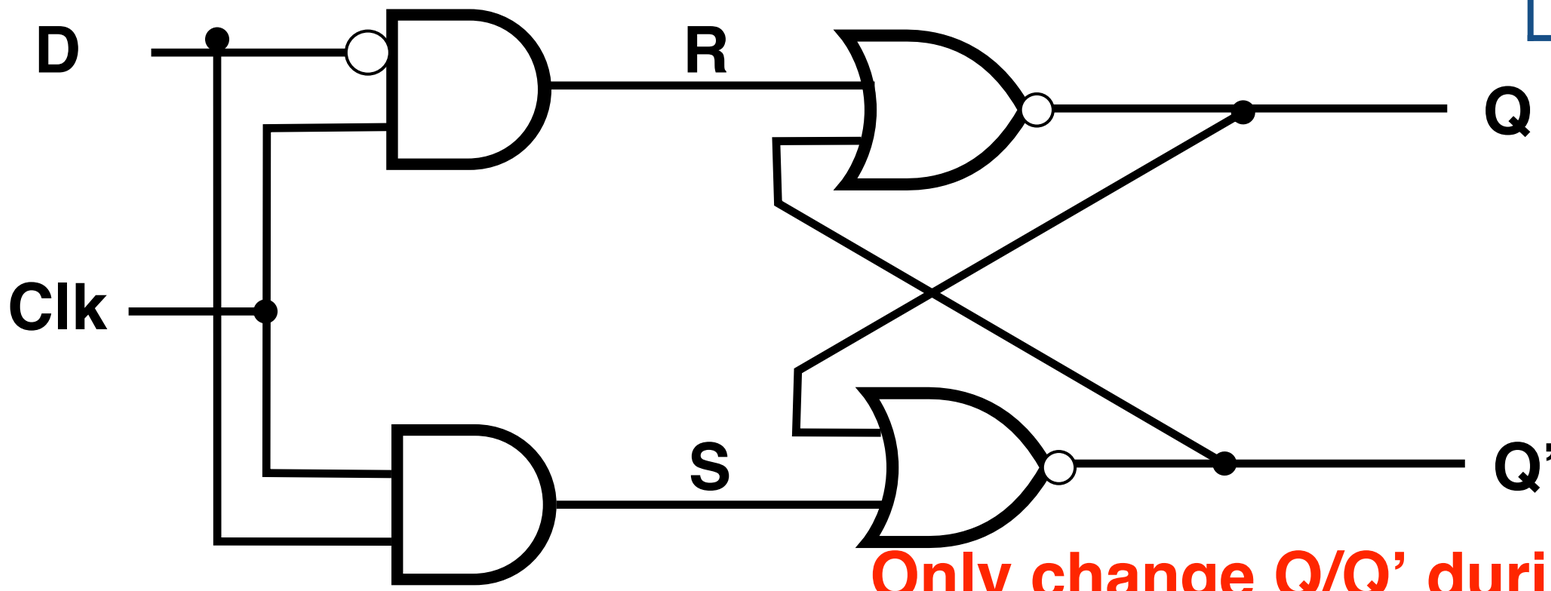
Only change Q/Q’ during Clk=“1”-> level sensitive



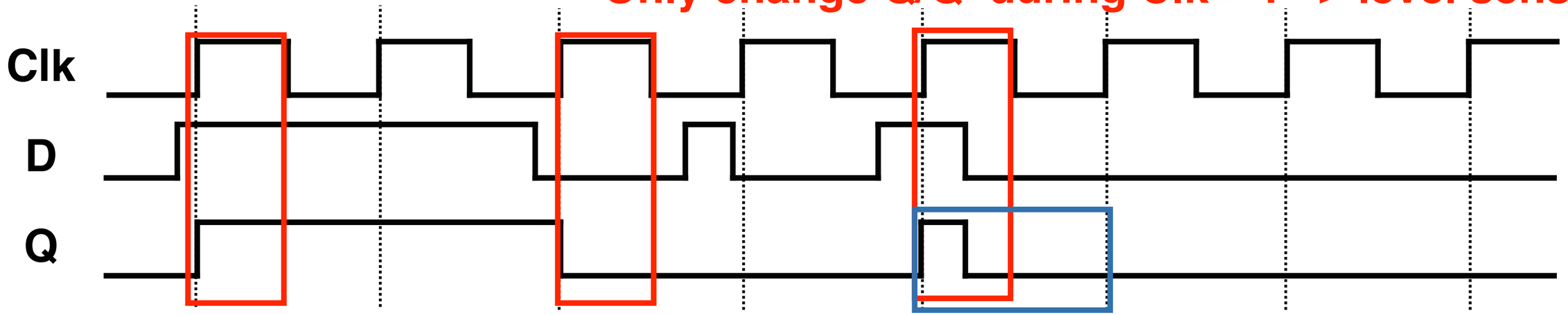
Timing diagram

D-Latch

CLK	D	D’	S	R	Q	Q’
0	X	X’	0	0	Qprev	Qprev’
1	0	1	0	1	0	1
1	1	0	1	0	1	0



Only change Q/Q’ during Clk=“1”-> level sensitive

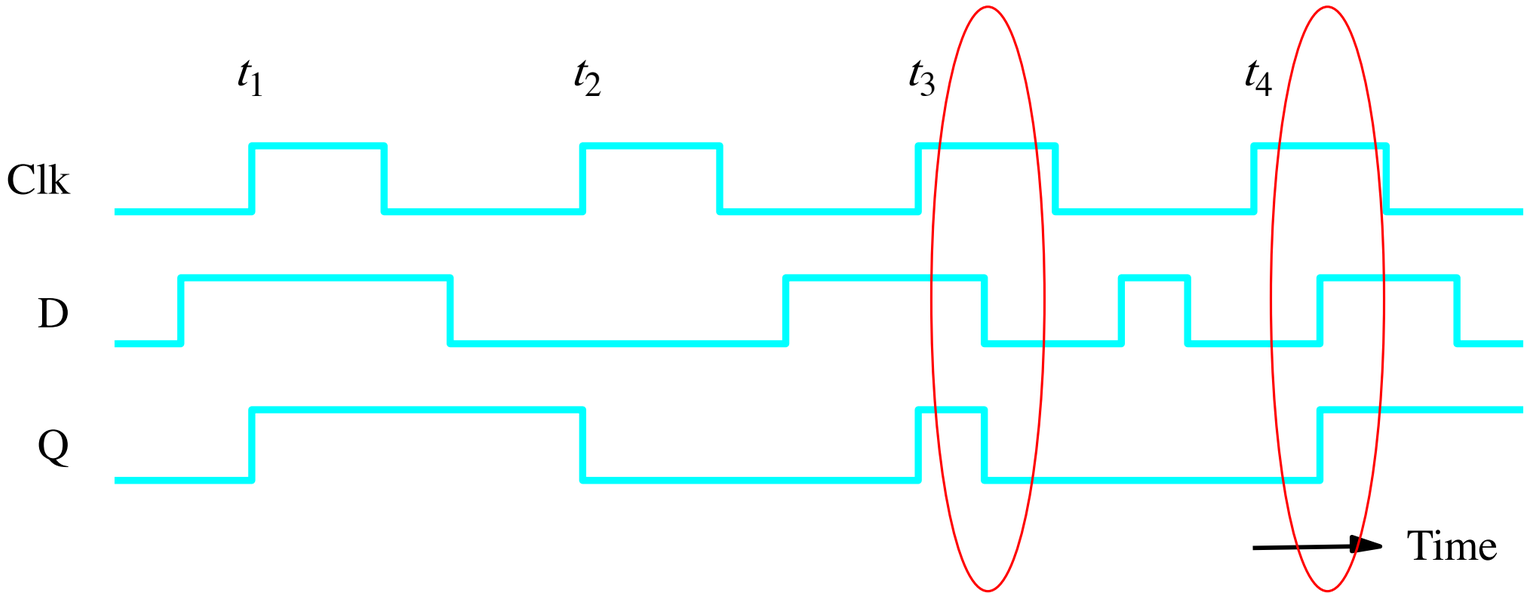


Timing diagram

Output doesn’t hold for the whole cycle

Latches vs Flip-flops

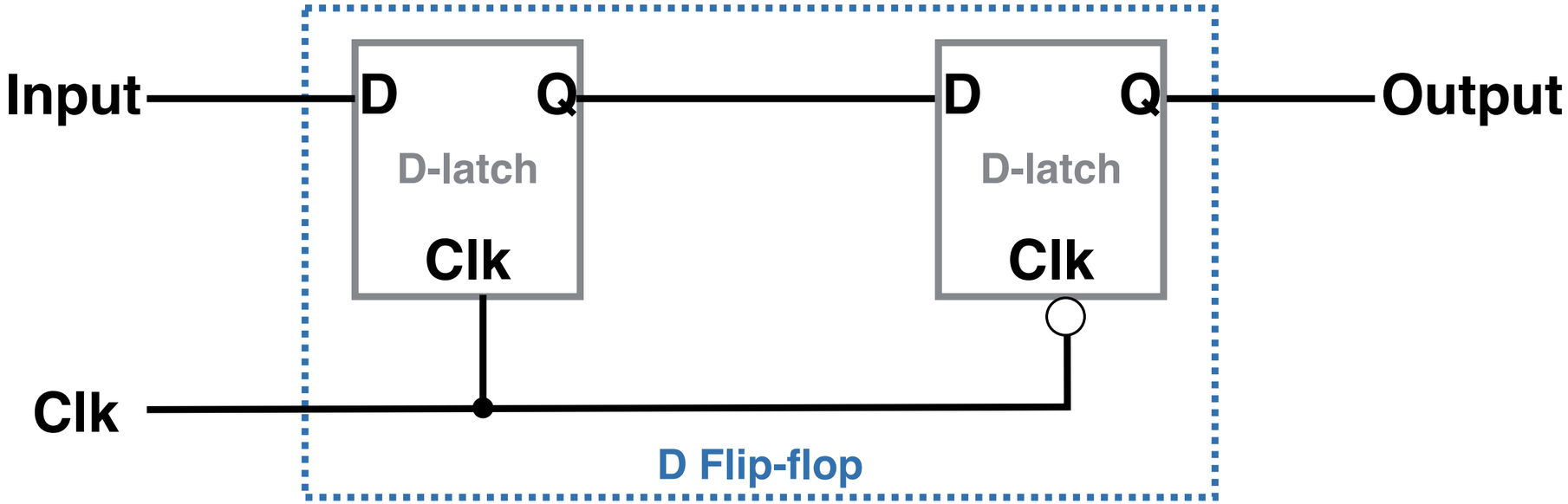
- When the Clk control signal is 1, the output of a latch may respond to all the changes of the inputs. So latches are level sensitive.

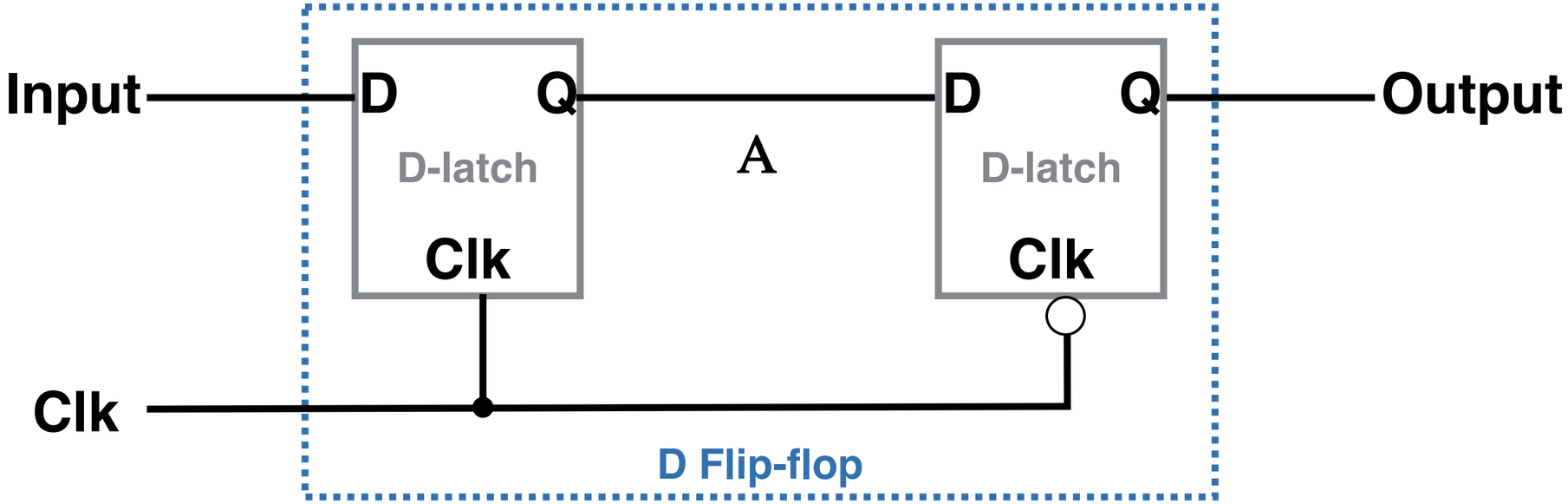


Timing diagram of D Latch

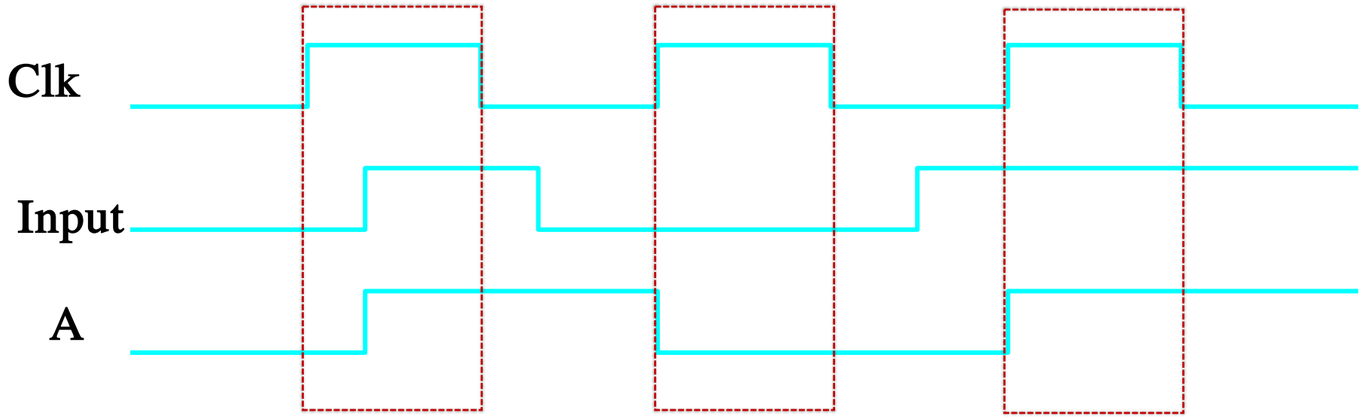
- With flip-flops, the output state can only be changed at the active edges of the Clk control signal. So flip-flops are edge triggered.

D flip-flop



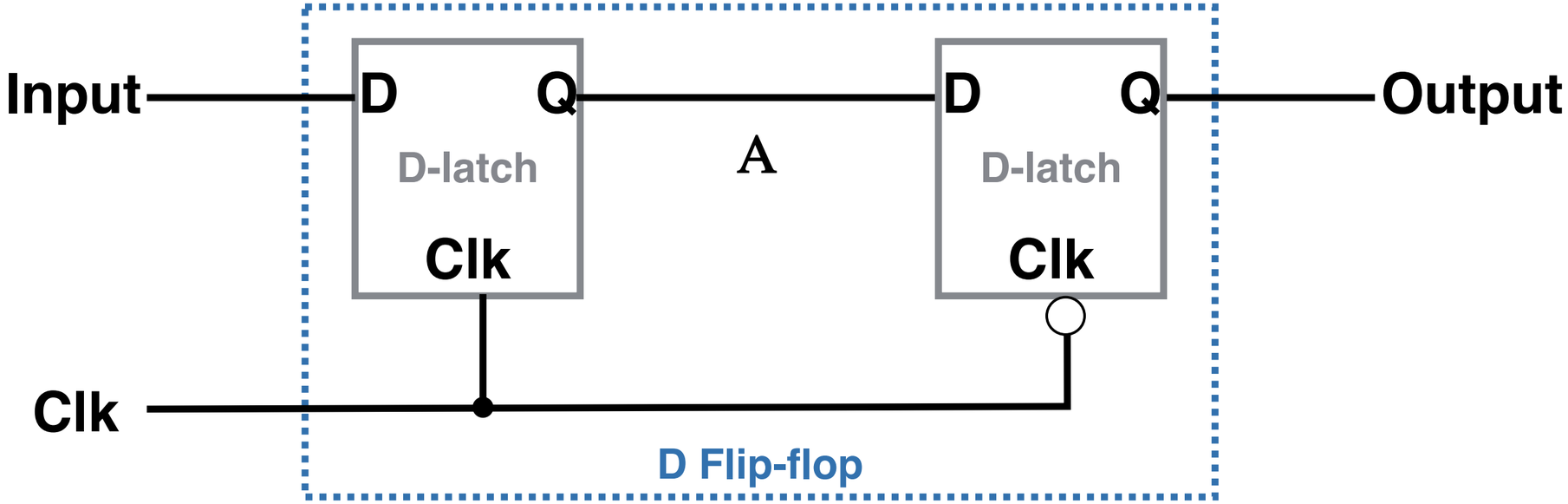


The first D-Latch

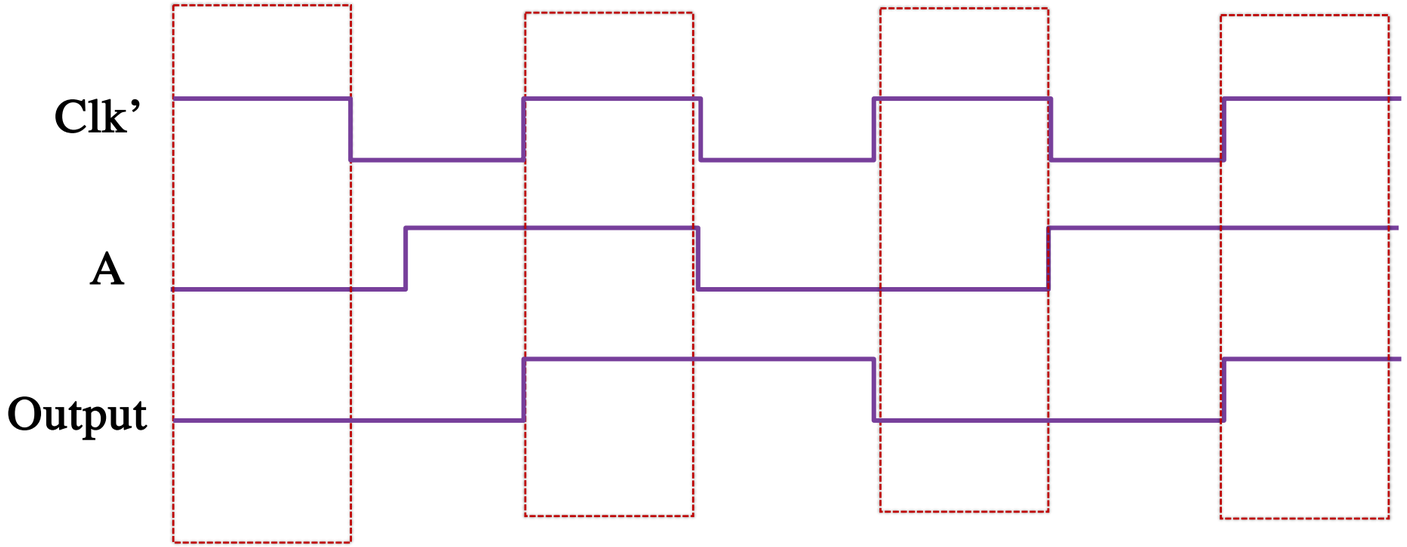


A = Input when Clk = 1

A = Previous A when Clk = 0

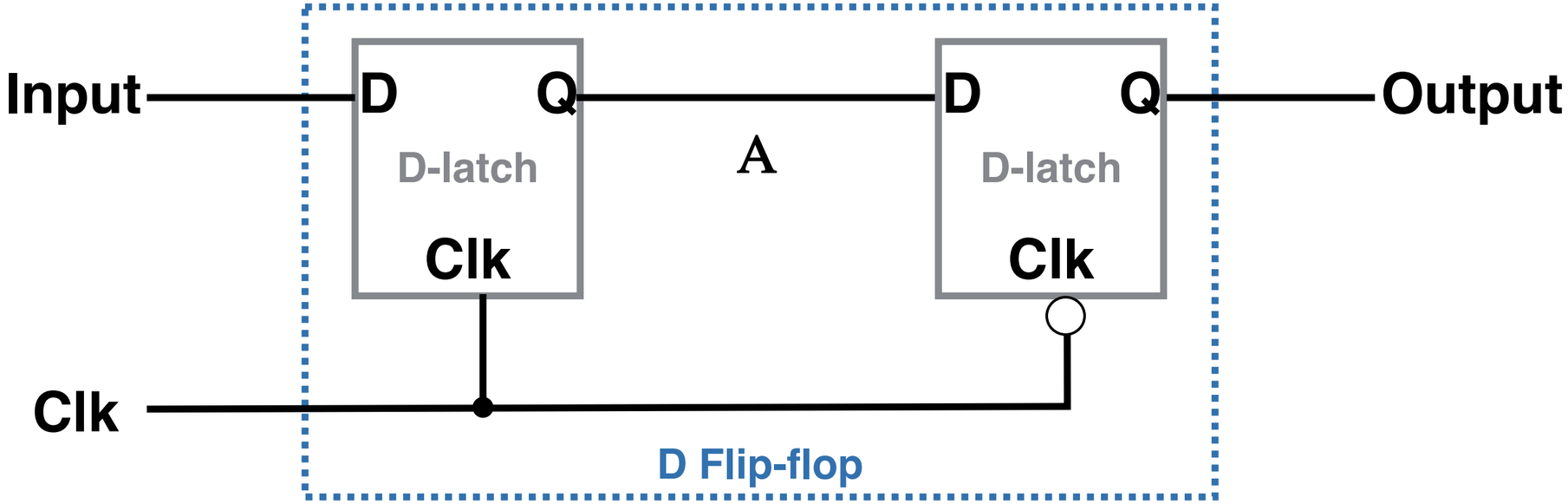


The second D-Latch

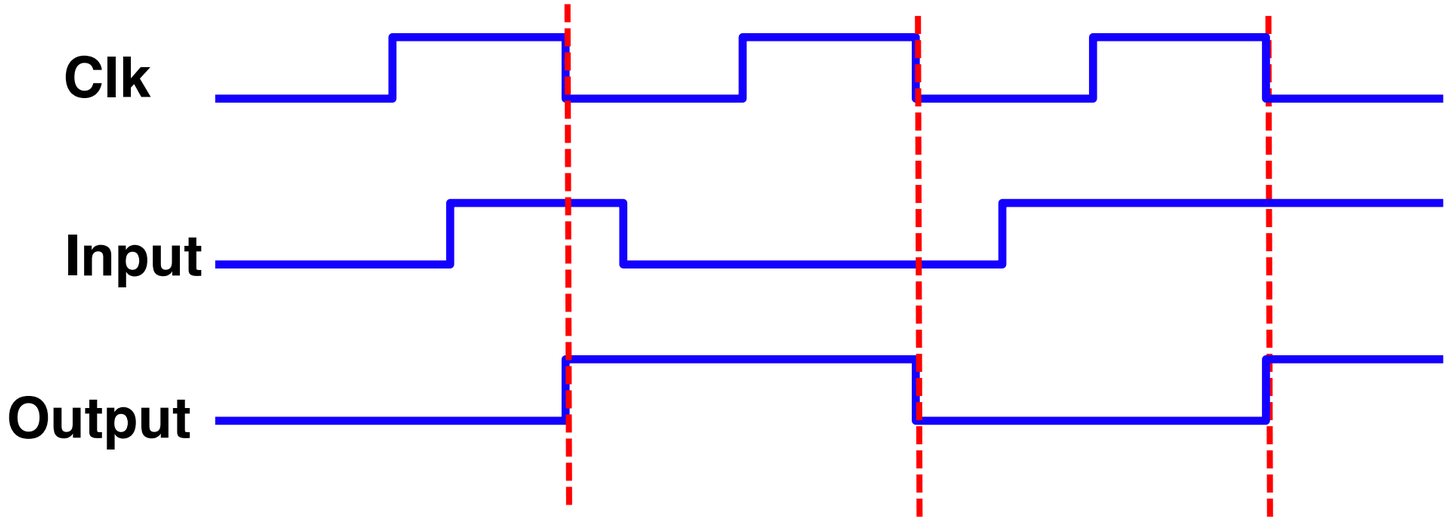


Output = A when Clk' = 1

Output = Previous Output when Clk' = 0



Putting the analysis of two D-Latches together

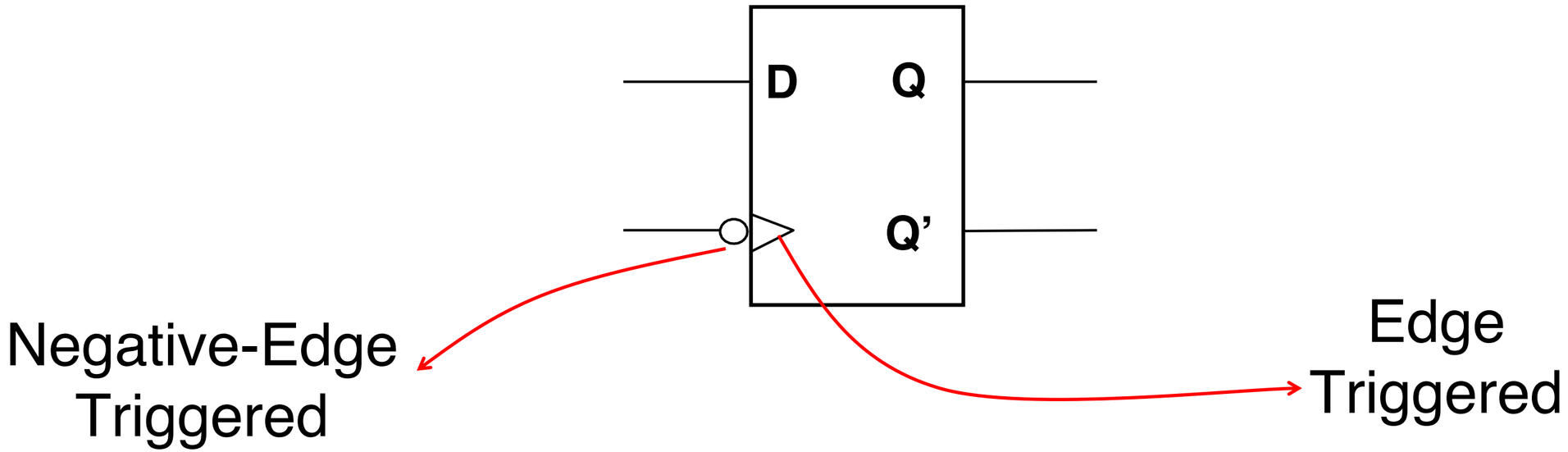


Timing diagram

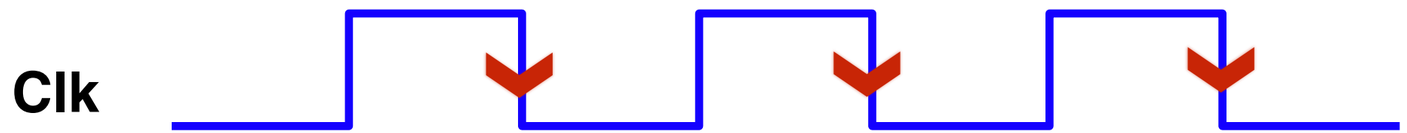
Output follows Input when Clock drops (Clk is changing from “1” to “0”).

D Flip-Flops

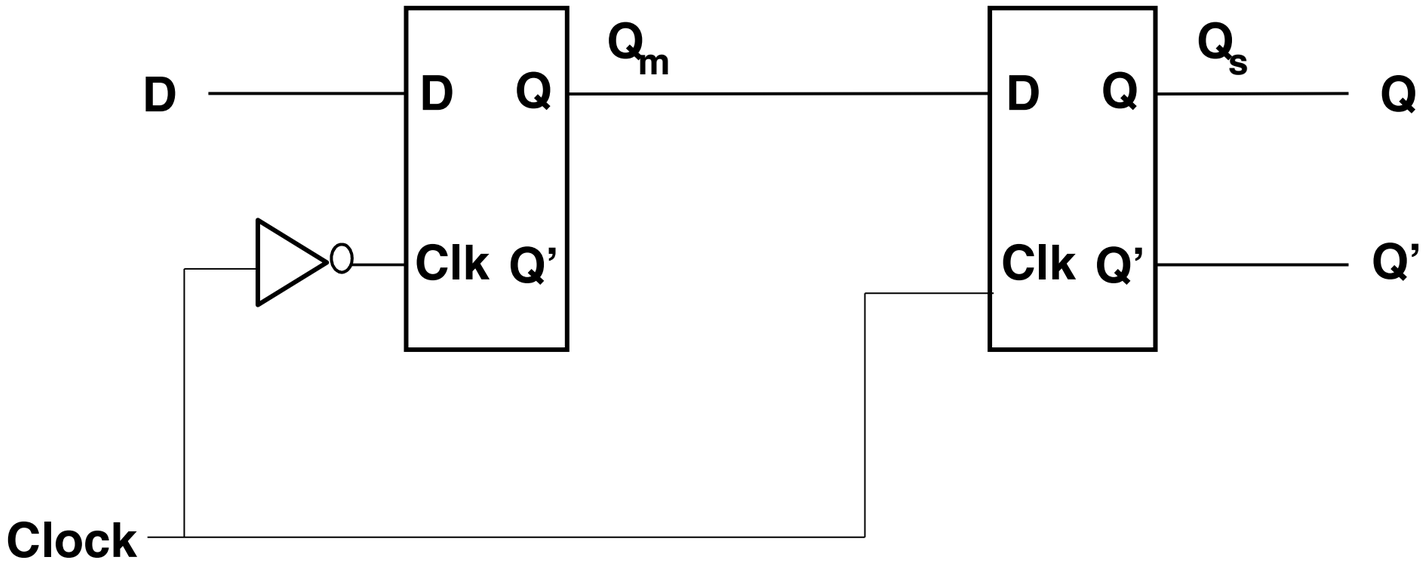
Graphical symbol

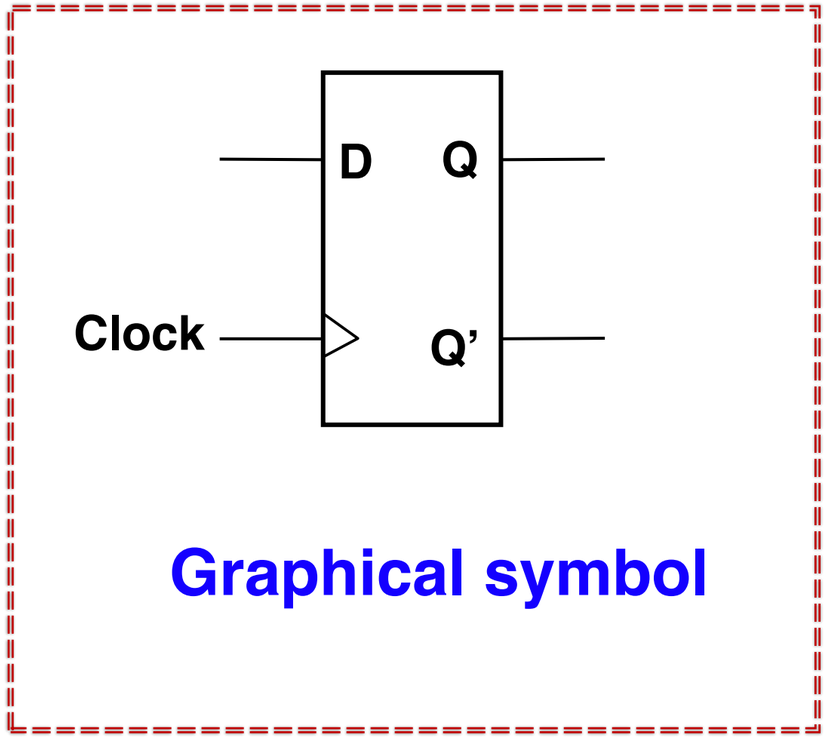


Clock signal's negative edges

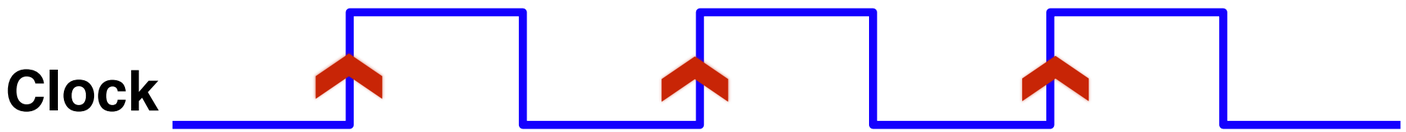


Positive-edge-triggered D flip-flop





Clock signal's positive edges



Output follows Input when Clock rises (Clock is changing from "0" to "1").

What do we need to physically implement the timer?

1) Logic to display the remaining time
   — use combinational logic, e.g., 7-segment display

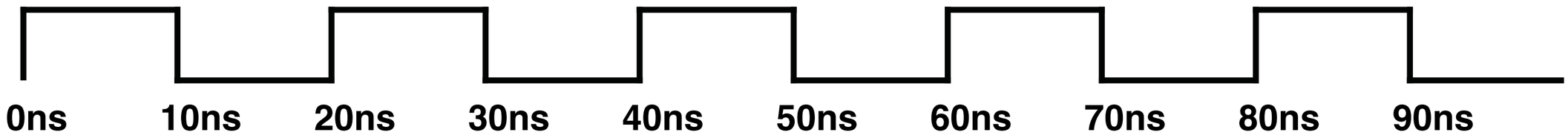
2) Logic that uses the "*current state*" and "*new input(s)*" to transit to a "*new state*" and generate the "*output(s)*"
   — use combinational logic

3) Logic to keep track of the "current state"
   — need **memory**

4) A control signal that helps us to transit from current state to next state state at the right time
   — need **clock**

The basic concept of “clock”

Clock signal



- Clock -- Pulsing signal for enabling latches; ticks like a clock
- The clock's period must be longer the critical path which is the longest path in the circuit
- Synchronous circuit: sequential circuit with a clock
- Clock period: time between pulse starts
  - Above signal: period = 20 ns
- Clock cycle: one such time interval
  - Above signal shows 5 clock cycles
- Clock duty cycle: time clock is high
  - 50% in this case
- Clock frequency: 1/period
  - Above : freq = 1 / 20ns = 50MHz;

Clock signal

- Regarding the above clock signal, please identify how many of the following statements are correct?
  - ① Clock period of 4ns with 250MHz frequency
  - ② Clock duty cycle 75%
  - ③ Clock period of 1ns with 1GHz frequency
  - ④ The above contains two complete clock cycles.
  - A. 0
  - B. 1
  - C. 2
  - D. 3
  - E. 4

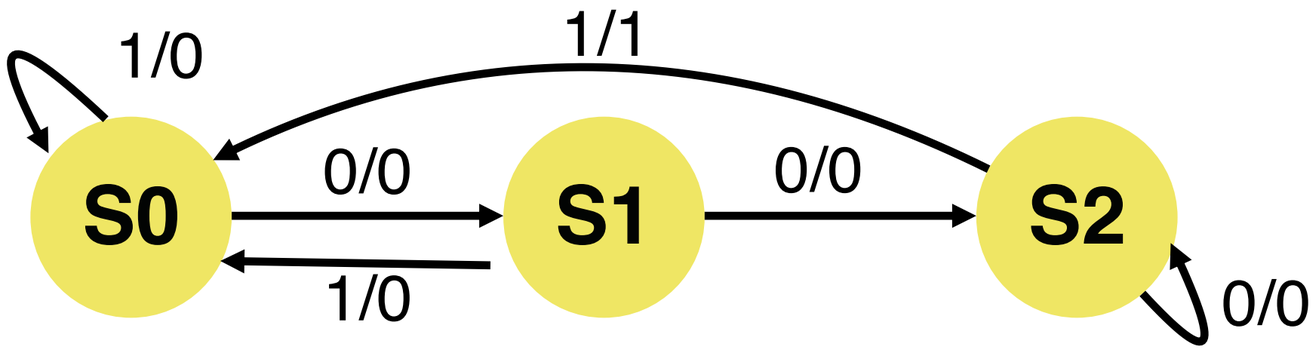
Clock signal

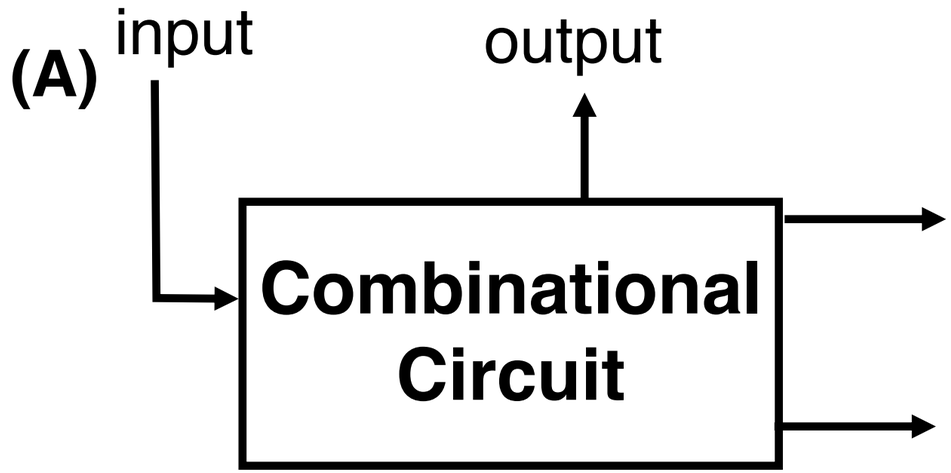
- Regarding the above clock signal, please identify how many of the following statements are correct?
  - ① Clock period of 4ns with 250MHz frequency
  - ② Clock duty cycle 75%
  - ③ Clock period of 1ns with 1GHz frequency
  - ④ The above contains two complete clock cycles.

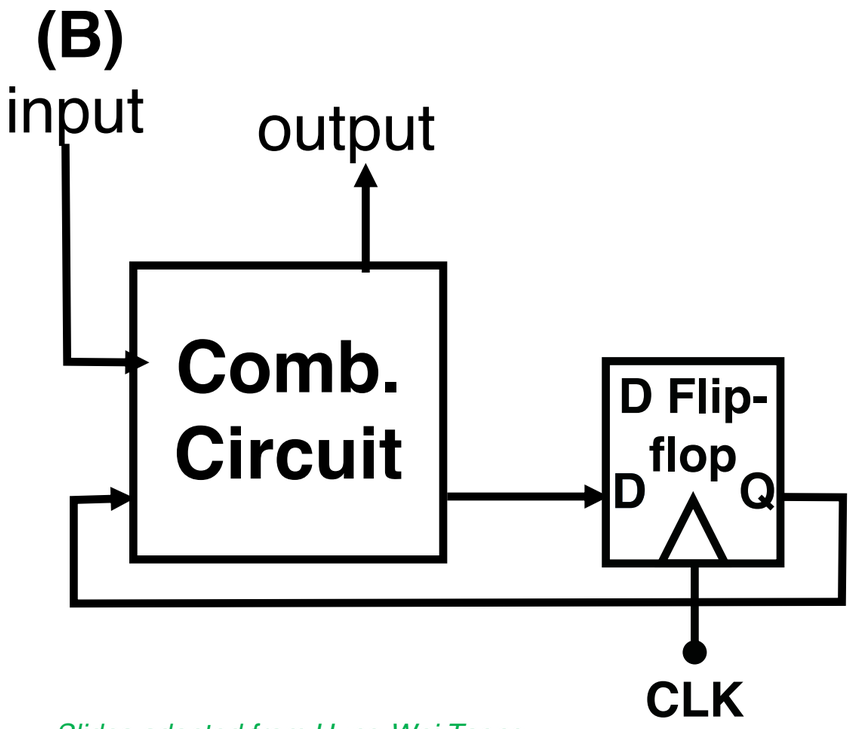
  A. 0
  B. 1
  C. 2
  D. 3
  E. 4

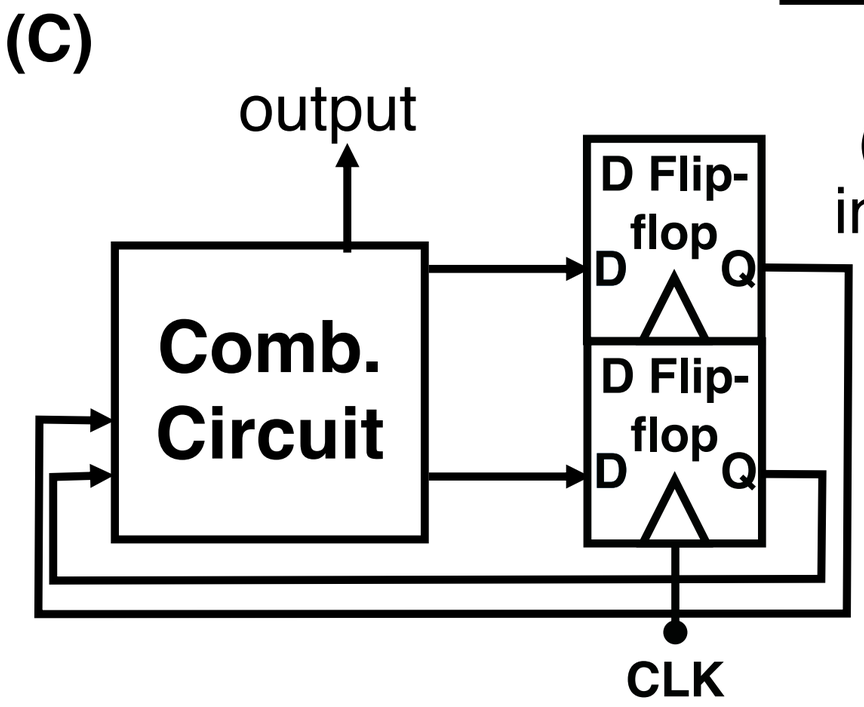
Let’s learn how to design sequential circuits!

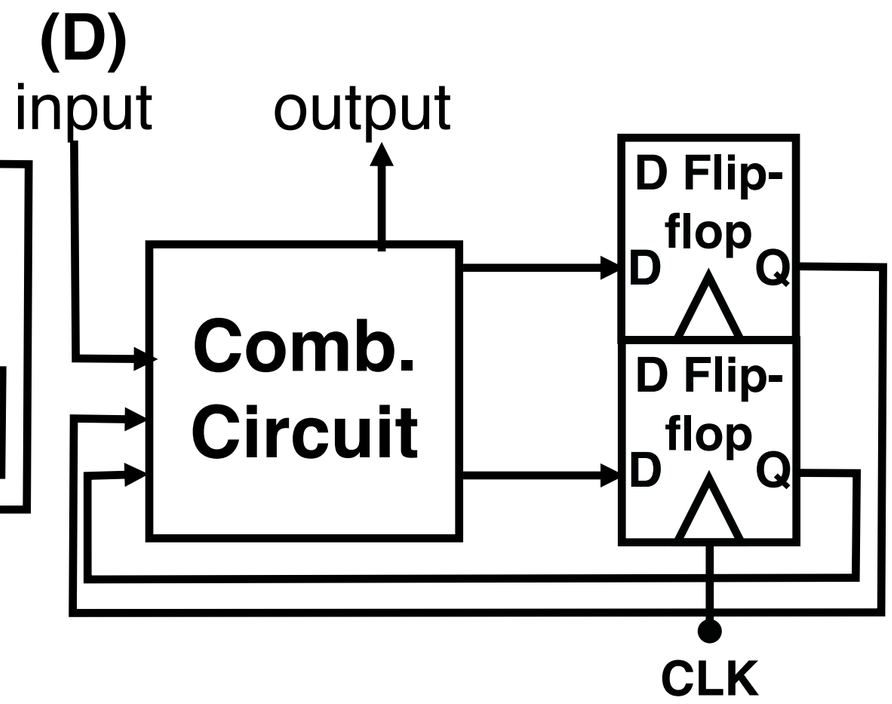
Which is the most likely circuit realization of Life on Mars





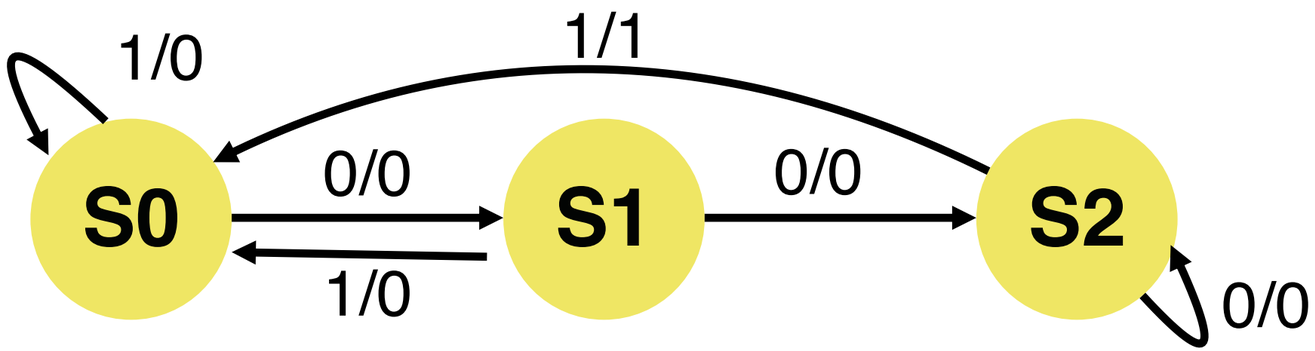




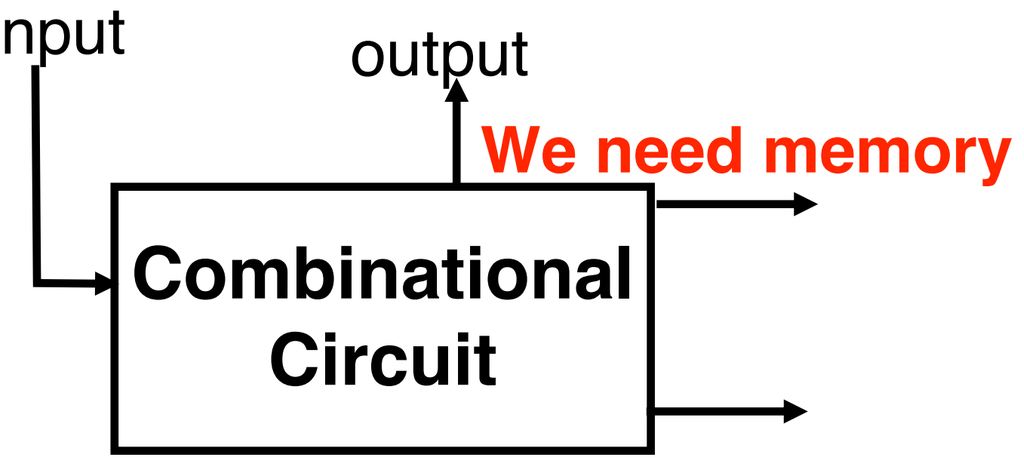


Slides adopted from Hung-Wei Tseng

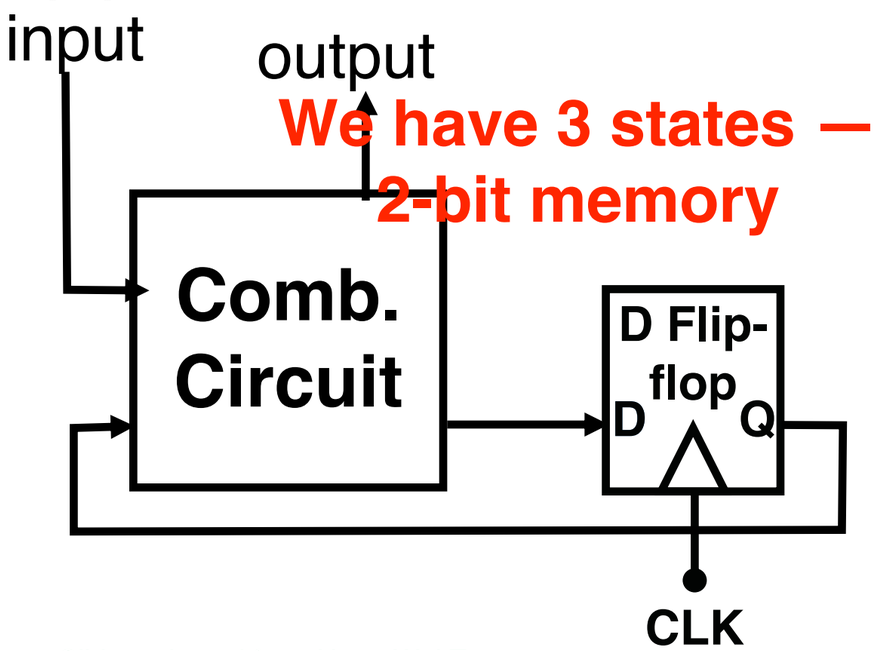
Which is the most likely circuit realization of Life on Mars



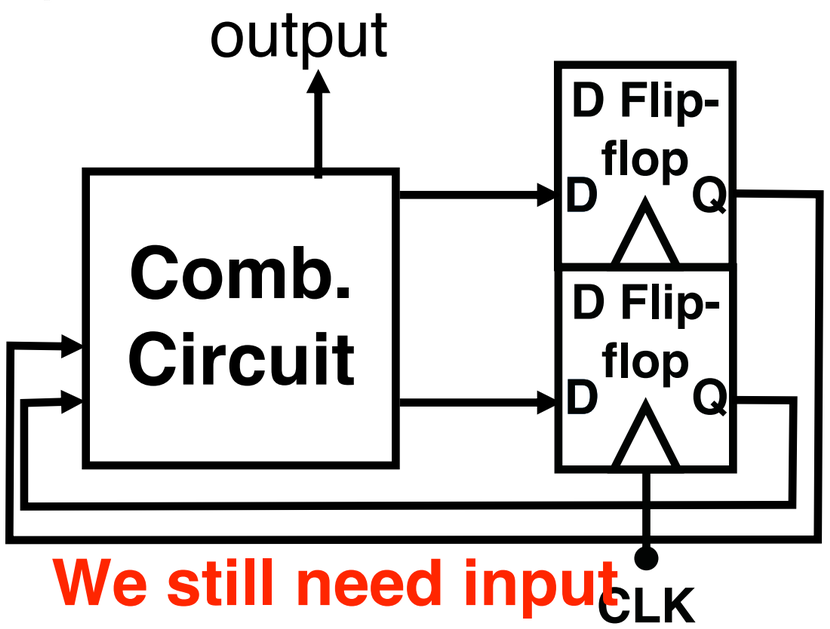
(A)



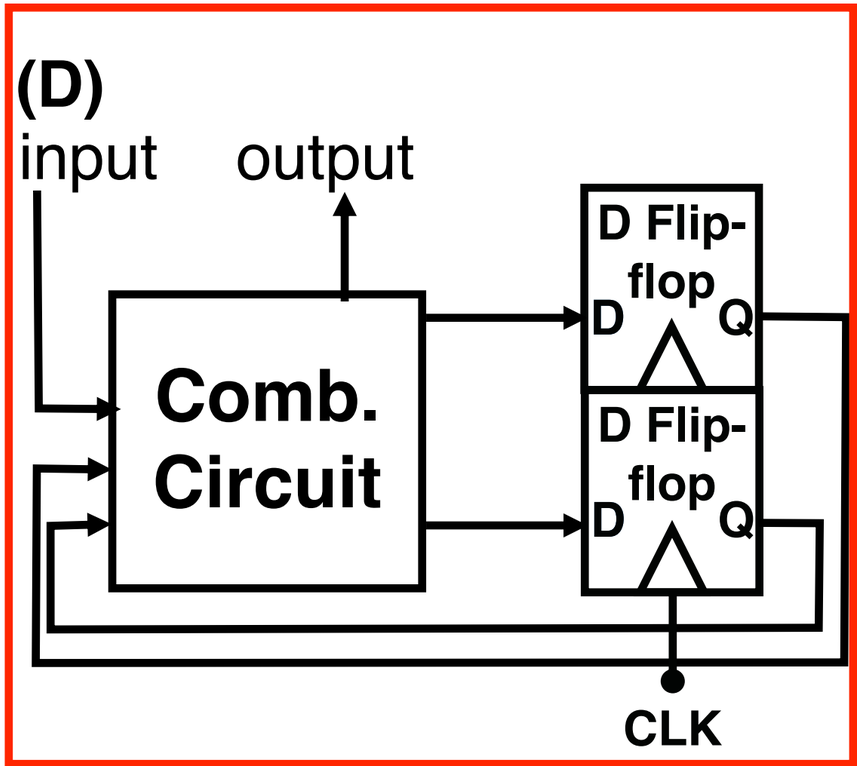
(B)



(C)



(D)



Slides adopted from Hung-Wei Tseng

Sequential Circuit Design Flow

Step 1: Create FSM (state diagram or state table)

Step 2: Reduce states if necessary (merging equivalent states)

Step 3: State assignment (use binary number to represent each state)

Step 4: Excitation table or Transition table (truth table for outputs and next state)

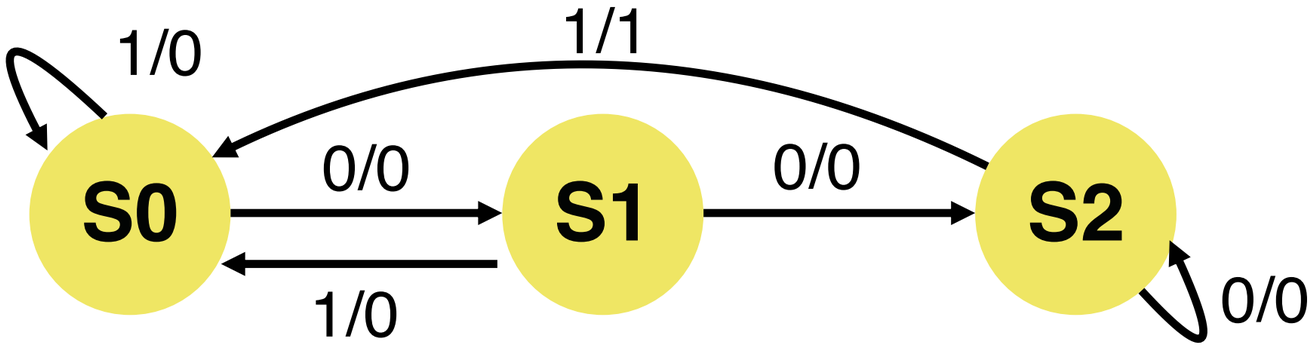
Step 5: Excitation equation and output equation (K-Map or Boolean algebra for equation simplification)

Step 6: Design the circuit using Flip-flops (based on Step 3) and combinational logic (based on Step 5)

Life on Mars

Step 1: Create FSM (state diagram or state table)

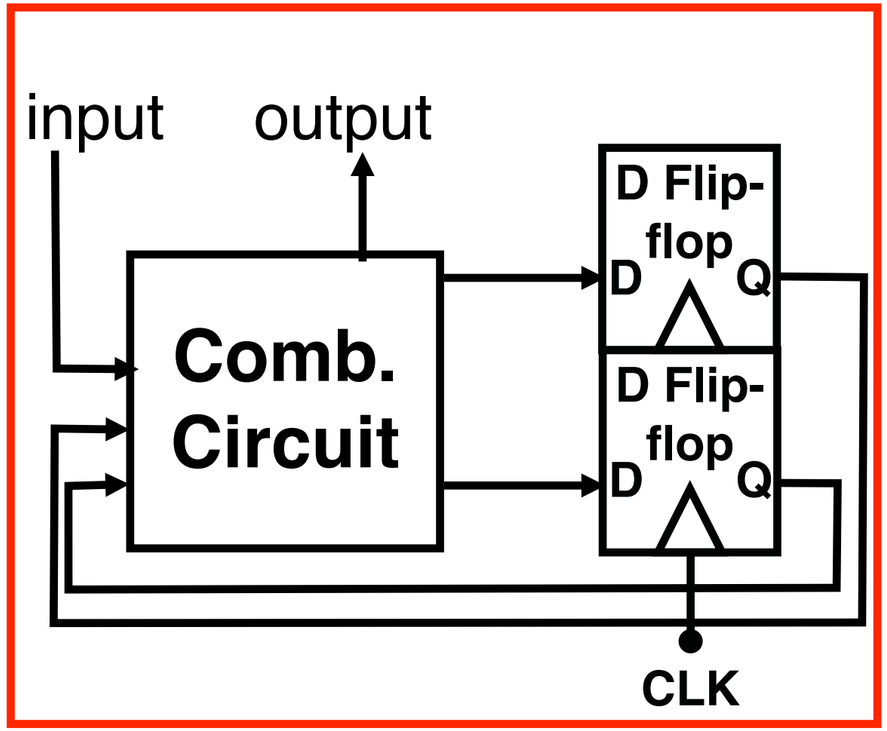
State Diagram



CurrentState	Next State, Output	
	Input	
	0	1
S0	S1, 0	S0, 0
S1	S2, 0	S0, 0
S2	S2, 0	S0, 1

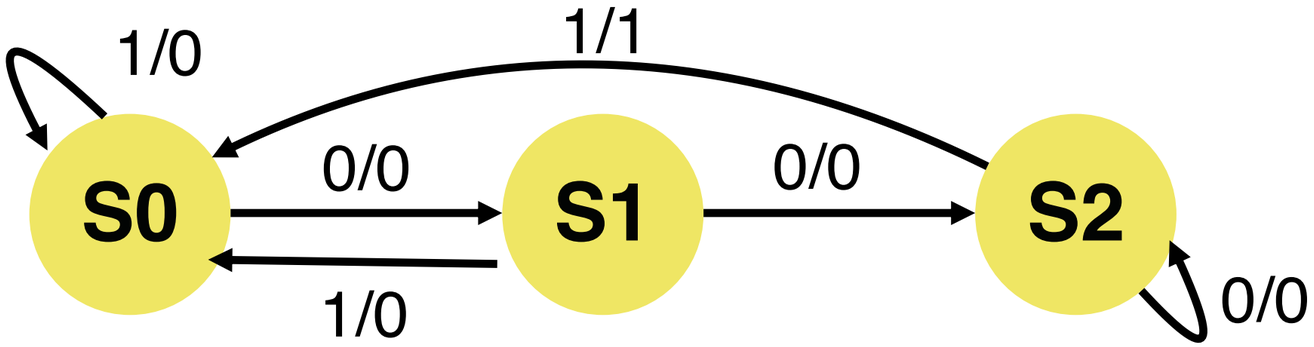
State Table

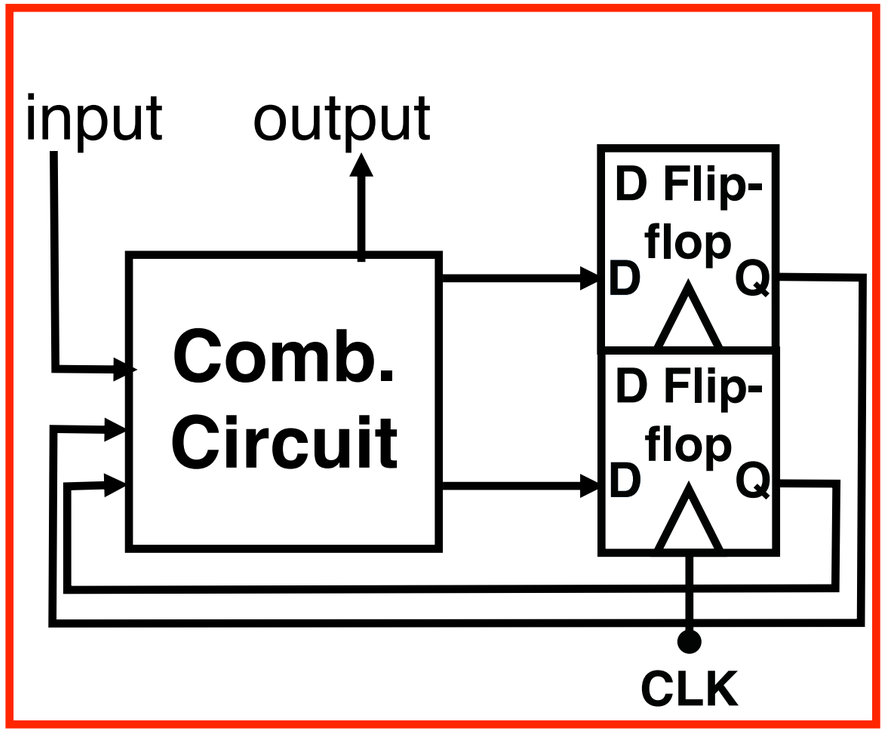
Overview of the circuit



Life on Mars

State Diagram





CurrentState	Next State, Output	
	Input	
	0	1
S0	S1, 0	S0, 0
S1	S2, 0	S0, 0
S2	S2, 0	S0, 1

State Table

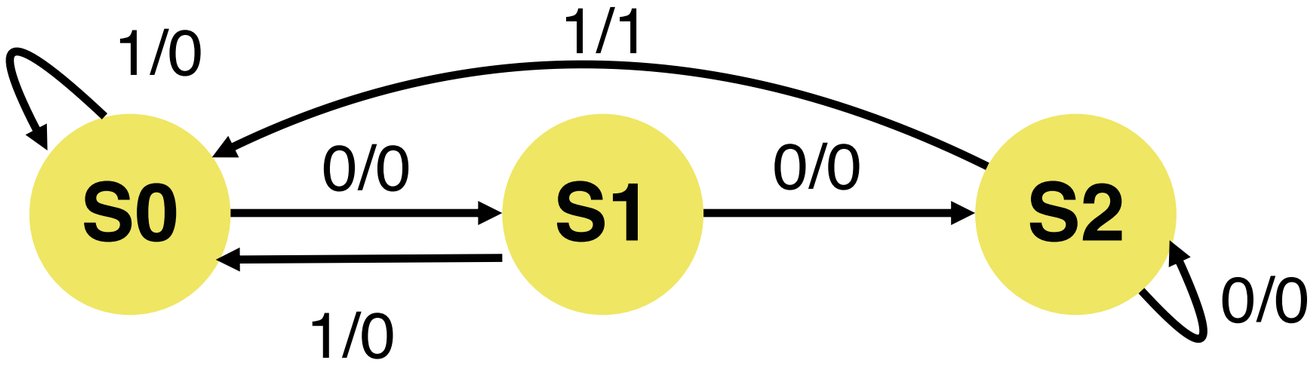
Step 3: State assignment (use binary number to represent each state)

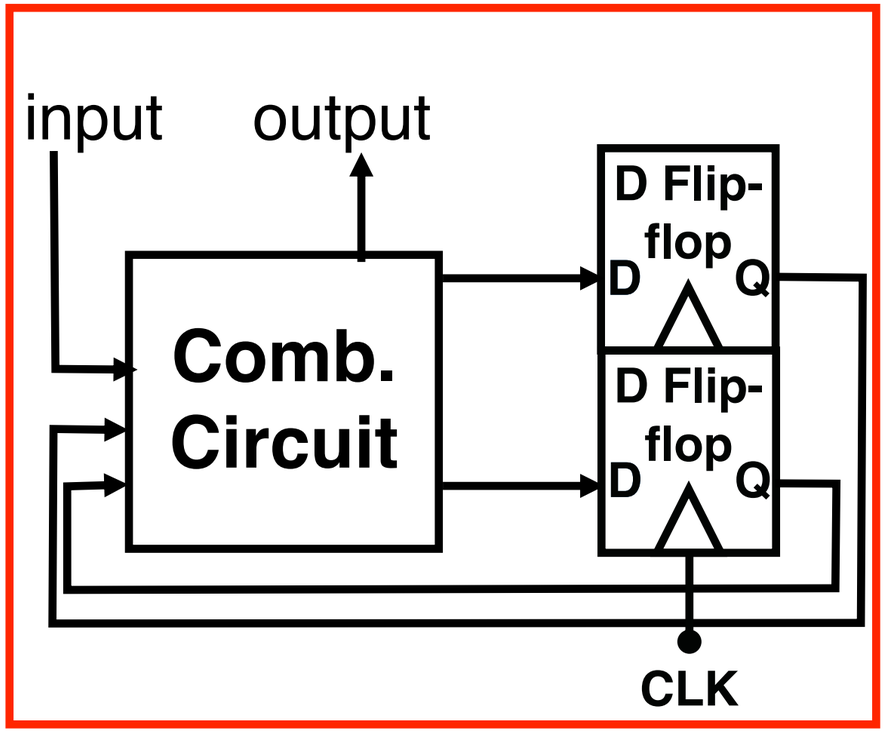
State Assignment

S0	00
S1	01
S2	10

Life on Mars

State Diagram





CurrentState	Next State, Output	
	Input	
	0	1
S0	S1, 0	S0, 0
S1	S2, 0	S0, 0
S2	S2, 0	S0, 1

State Table

State Assignment

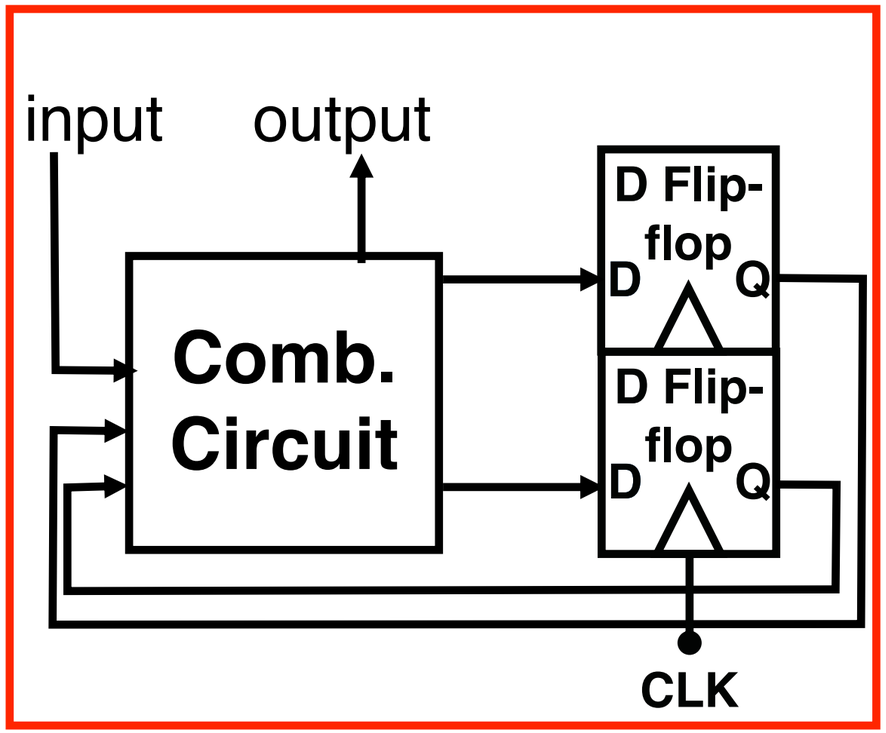
S0	00
S1	01
S2	10

State Truth Table

State\Input	0	1
00	01, 0	00, 0
01	10, 0	00, 0
10	10, 0	00, 1

Excitation Table

- Excitation table is basically the truth table describing the combinational circuit that provides inputs for the flip-flops in the sequential circuit. How many rows are there in the excitation table of Life on Mars?
  A. 2
  B. 3
  C. 4
  D. 8
  E. 32



Slides adopted from Hung-Wei Tseng

Excitation Table

- Excitation table is basically the truth table describing the combinational circuit that provides inputs for the flip-flops in the sequential circuit. How many rows are there in the excitation table of Life on Mars?

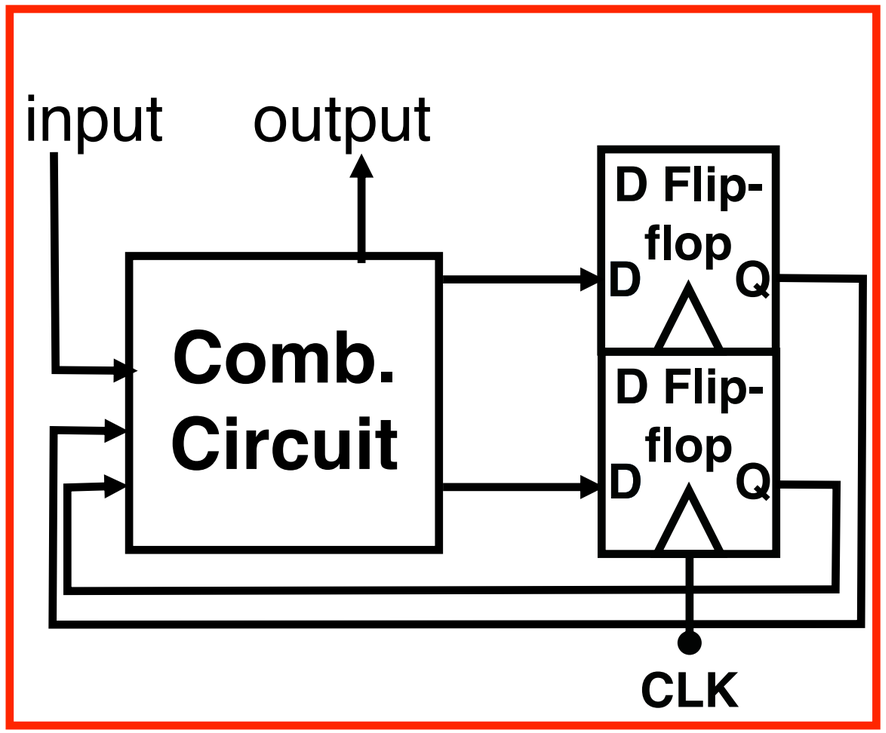
A. 2

B. 3

C. 4

D. 8

E. 32



Life on Mars

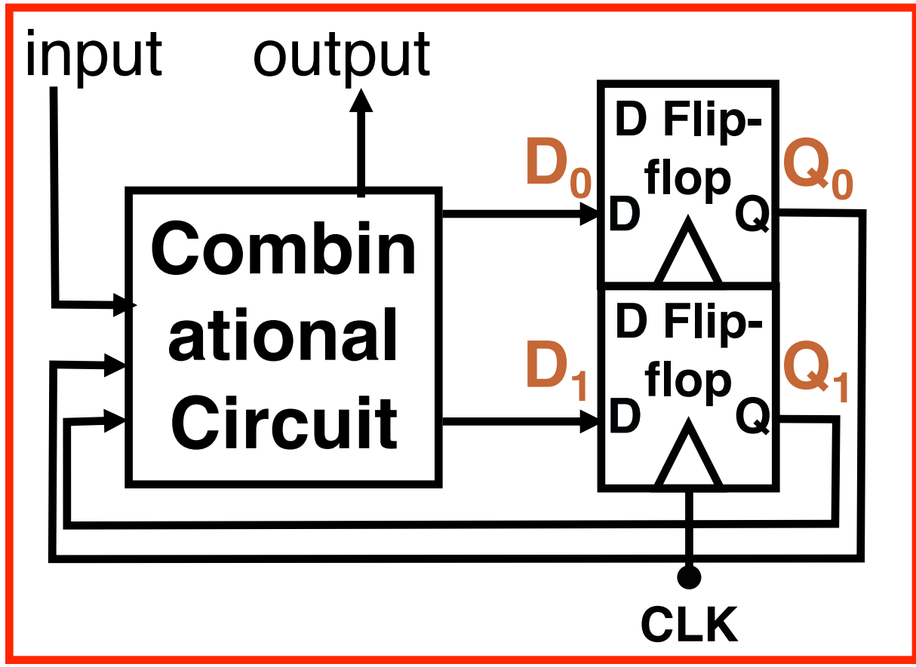
State Truth Table

State\Input	0	1
00	01, 0	00, 0
01	10, 0	00, 0
10	10, 0	00, 1

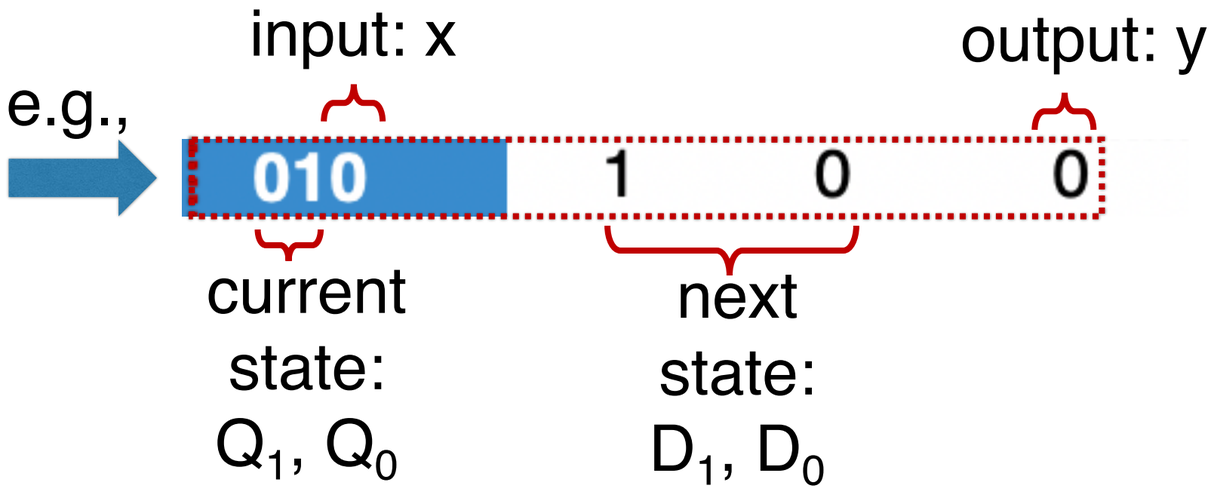
Excitation Table

NextStateOput StateInput	D_1	D_0	y
000	0	1	0
001	0	0	0
010	1	0	0
011	0	0	0
100	1	0	0
101	0	0	1
110	X	X	X
111	X	X	X

Step 4: Excitation table or Transition table (truth table for outputs and next state)



e.g.,



Life on Mars

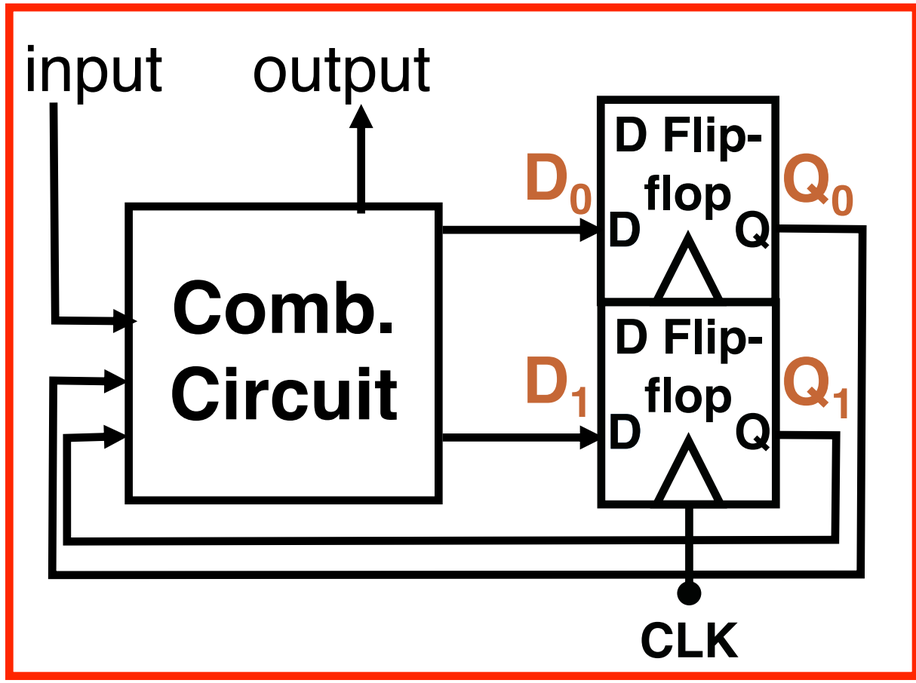
State Truth Table

State\Input	0	1
00	01, 0	00, 0
01	10, 0	00, 0
10	10, 0	00, 1

Excitation Table

NextStateOput StateInput	D_1	D_0	y
000	0	1	0
001	0	0	0
010	1	0	0
011	0	0	0
100	1	0	0
101	0	0	1
110	X	X	X
111	X	X	X

Step 5: Excitation equation and output equation (K-Map or Boolean algebra for equation simplification)



K-Map — D_1

	0,0	0,1	1,1	1,0
0	0	1	X	1
1	0	0	X	0

K-Map — D_0

	0,0	0,1	1,1	1,0
0	1	0	X	0
1	0	0	X	0

K-Map — y

	0,0	0,1	1,1	1,0
0	0	0	X	0
1	0	0	X	1

K-Map for D_1

Step 5: Excitation equation and output equation (K-Map or Boolean algebra for equation simplification)

Excitation Table

NextStateOput StateInput	D_1	D_0	y
000	0	1	0
001	0	0	0
010	1	0	0
011	0	0	0
100	1	0	0
101	0	0	1
110	X	X	X
111	X	X	X

x \ Q_1Q_0	0,0	0,1	1,1	1,0
0	0	1	X	1
1	0	0	X	0

Excitation equation: $D_1 = x'Q_0 + x'Q_1$

K-Map for D_0

Step 5: Excitation equation and output equation (K-Map or Boolean algebra for equation simplification)

Excitation Table

NextStateOput StateInput	D_1	D_0	y
000	0	1	0
001	0	0	0
010	1	0	0
011	0	0	0
100	1	0	0
101	0	0	1
110	X	X	X
111	X	X	X

x \ Q_1Q_0	0,0	0,1	1,1	1,0
0	1	0	X	0
1	0	0	X	0

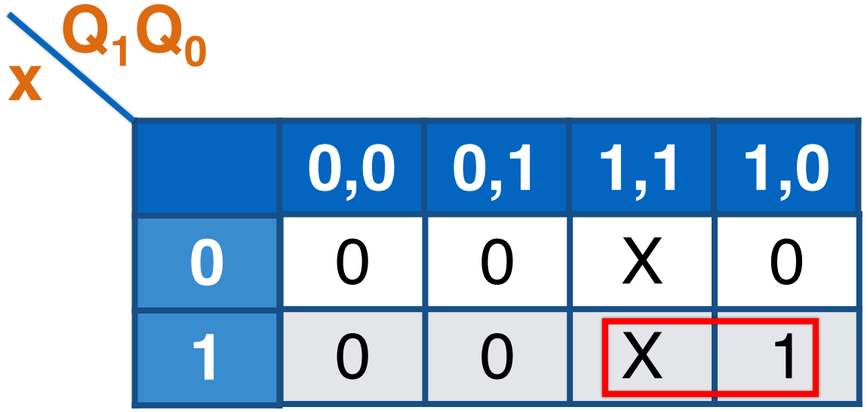
Excitation equation: $D_0 = Q_0'Q_1'x'$

K-Map for y

Step 5: Excitation equation and output equation (K-Map or Boolean algebra for equation simplification)

Excitation Table

NextStateOput StateInput	D_1	D_0	y
000	0	1	0
001	0	0	0
010	1	0	0
011	0	0	0
100	1	0	0
101	0	0	1
110	X	X	X
111	X	X	X



Output equation: $y = Q_1x$

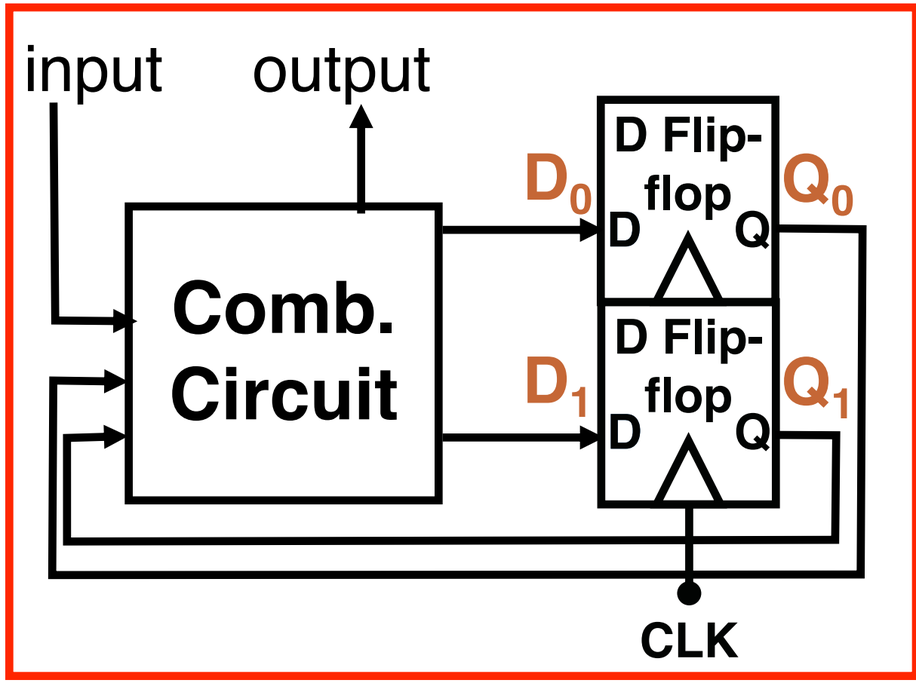
Summary

State Truth Table

State\Input	0	1
00	01, 0	00, 0
01	10, 0	00, 0
10	10, 0	00, 1

Excitation Table

NextStateOput StateInput	D_1	D_0	y
000	0	1	0
001	0	0	0
010	1	0	0
011	0	0	0
100	1	0	0
101	0	0	1
110	X	X	X
111	X	X	X



K-Map — D_1

	0,0	0,1	1,1	1,0
0	0	1	X	1
1	0	0	X	0

$D_1 = x'Q_0 + x'Q_1$

K-Map — D_0

	0,0	0,1	1,1	1,0
0	1	0	X	0
1	0	0	X	0

$D_0 = Q_0'Q_1'x'$

K-Map — y

	0,0	0,1	1,1	1,0
0	0	0	X	0
1	0	0	X	1

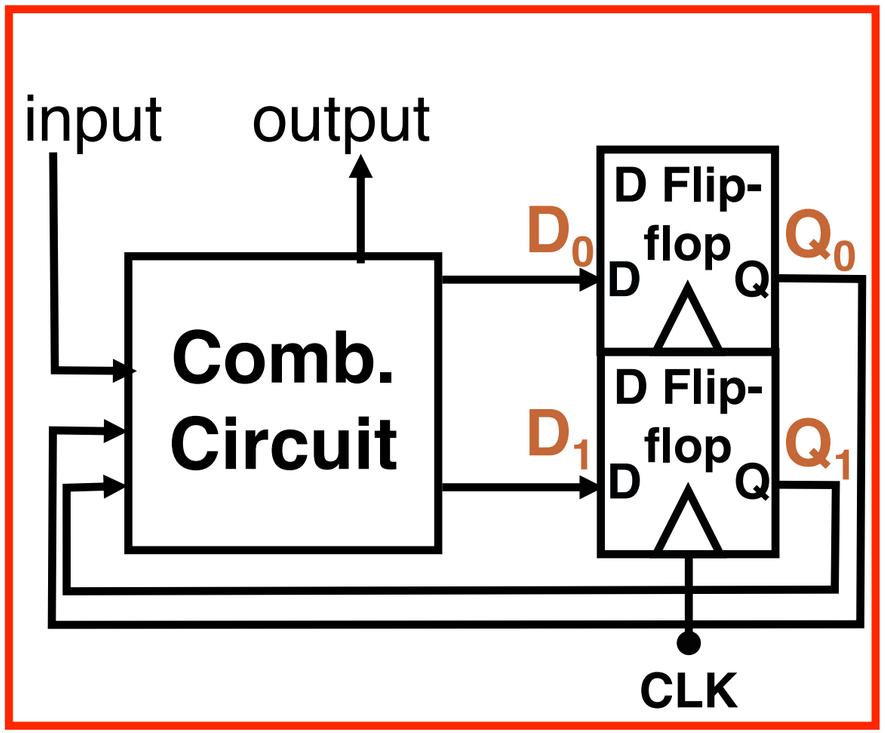
$y = Q_1x$

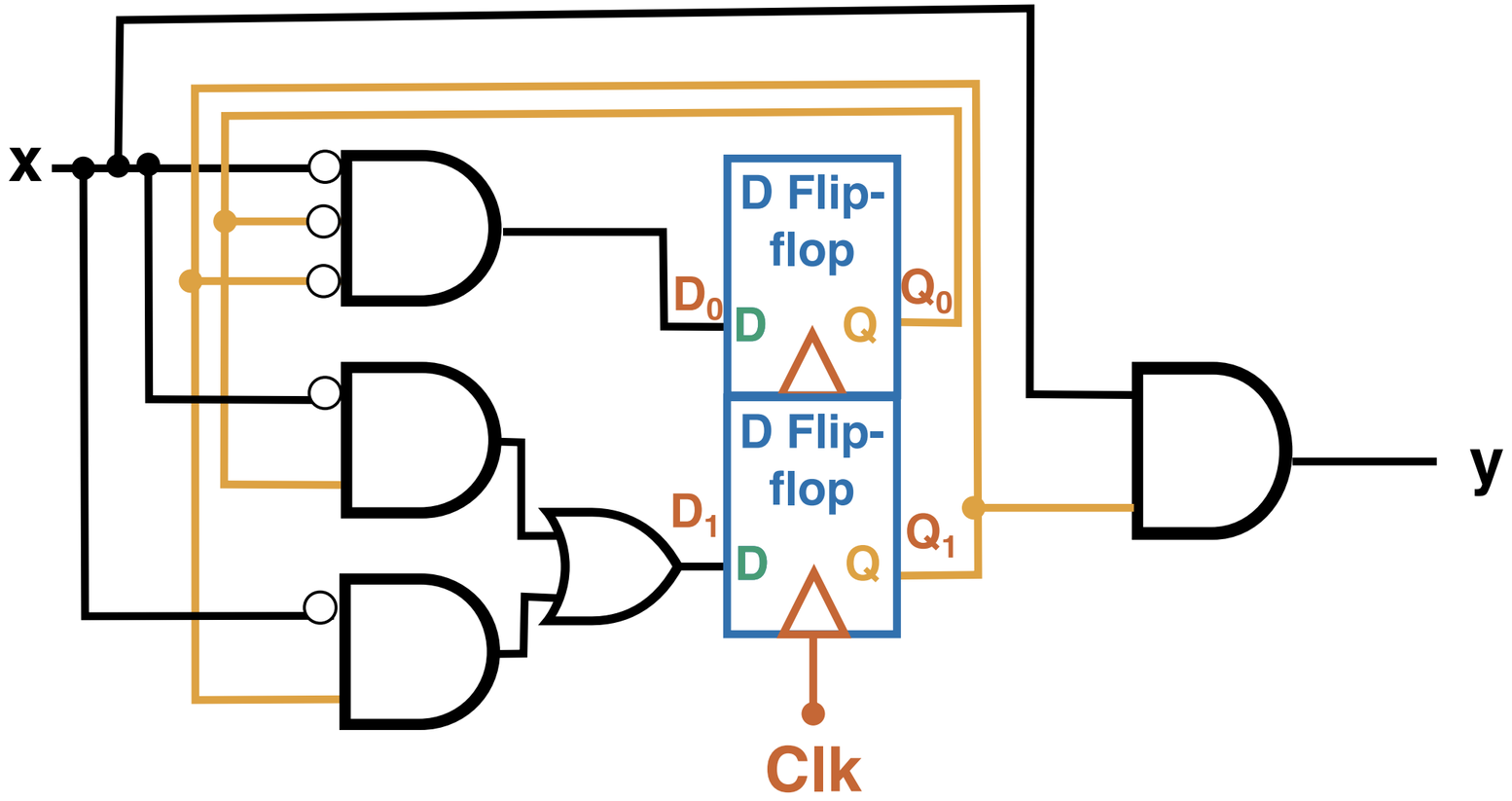
Circuit — Life on Mars

$D_1 = x'Q_0 + x'Q_1$

$D_0 = Q_0'Q_1'x'$

$y = Q_1x$





Step 6: Design the circuit using Flip-flops (based on Step 3) and combinational logic (based on Step 5)

Terminologies - state table & state truth table

(1) State Table

CurrentState	Next State, Output	
	Input	
	0	1
S0	S1, 0	S0, 0
S1	S2, 0	S0, 0
S2	S2, 0	S0, 1

Substitute state names with values

via state assignment

S0	00
S1	01
S2	10

(2) State Truth Table

State\Input	0	1
00	01, 0	00, 0
01	10, 0	00, 0
10	10, 0	00, 1

State table, continue

State Table example 1:

CurrentState	Next State, Output	
	Input	
	0	1
S0	S1, 0	S0, 0
S1	S2, 0	S0, 0
S2	S2, 0	S0, 1

State Table example 2:

	X Y				
S	00	01	10	11	Z1 Z2
A	A	E	B	B	10
B	B	B	D	D	10
C	C	G	A	A	10
D	D	D	C	C	00
E	F	F	F	F	11
F	B	B	B	B	10
G	H	H	H	H	11
H	D	D	D	D	11
	S*				

Q: which example represents a Moore machine?

S: current state
S*: next state
State names: A, B, C, D, E, F, G, H
Inputs: X and Y
Outputs: Z1 and Z2

Terminologies- excitation table & transition table

(3) Excitation Table

NextStateOput / StateInput	D_1	D_0	y
000	0	1	0
001	0	0	0
010	1	0	0
011	0	0	0
100	1	0	0
101	0	0	1
110	X	X	X
111	X	X	X

(4) Transition Table

	X Y				
Q2 Q1 Q0	00	01	10	11	Z1 Z2
000	000	100	001	001	10
001	001	001	011	011	10
010	010	110	000	000	10
011	011	011	010	010	00
100	101	101	101	101	11
101	001	001	001	001	10
110	111	111	111	111	11
111	011	011	011	011	11
	Q2* Q1* Q0*				

Note:

- Both excitation table and transition table contain truth table for outputs (y in (3) and Z1, Z2 in (4)).
- Excitation table contains the truth table for D1, D0 (inputs of D flip-flops).
- Transition table contains the truth table for next states (Q2*, Q1*, Q0* in (4)).

Terminology – excitation equation

(5) Excitation Equation: **Boolean algebra expression(s) for input signals of flip-flops**

Example: $D_1 = x'Q_0 + x'Q_1$

$D_0 = Q_0'Q_1'x'$

Note:
Input signals of flip-flops (D_1, D_0) are functions on current state (Q_1, Q_0) and input(s) (x)

Terminology – characteristic equation

(6) Characteristic Equation:

The functional behavior of a latch or flip-flop can be described formally by a characteristic equation that specifies the flip-flop's next state as a function of its current state and inputs.

E.g., Gated SR Latch

characteristic table

Clk	S	R	Q($t+1$)
0	x	x	Q(t) (no change)
1	0	0	Q(t) (no change)
1	0	1	0
1	1	0	1
1	1	1	x

characteristic equation

$Q^* = S + R'Q$

Alternatively,

$Q(t+1) = S + R'Q(t)$

Q^* = next state
Q = current state

Previously, we also used
$Q(t+1)$ = next state
$Q(t)$ = current state

Terminology – characteristic equation

(6) Characteristic Equation:

The functional behavior of a latch or flip-flop can be described formally by a characteristic equation that specifies the flip-flop's next state as a function of its current state and inputs.

Q* = next state
Q = current state

Device Type	Characteristic Equation
Gated SR Latch	**Q* = S + R'Q**
D Latch	**Q* = D**
D Flip-flop	**Q* = D**
J-K Flip-flop	**Q* = JQ' + K'Q**
T Flip-flop	**Q* = Q'**

Most applicable!
(our focus)

Not very applicable!
(out of the scope)

Terminology – transition equation

7) Transition equation : Boolean algebra expression(s) of next state as a function of current-state and inputs.

Example: $Q^*_1 = x'Q_0 + x'Q_1$

$Q^*_0 = Q_0'Q_1'x'$

Note:

- Substitute excitation equations into characteristic equations to obtain transition equations.
- Same as excitation equations when using D flip-flops.

Terminology – output equation

8) output equation : **Boolean algebra expression(s) of output(s)**

Example 1: $y = Q_1 x$

Example 2: $Z1 = Q2 + Q1' + Q0'$

$Z2 = Q2 \cdot Q1 + Q2 \cdot Q0'$

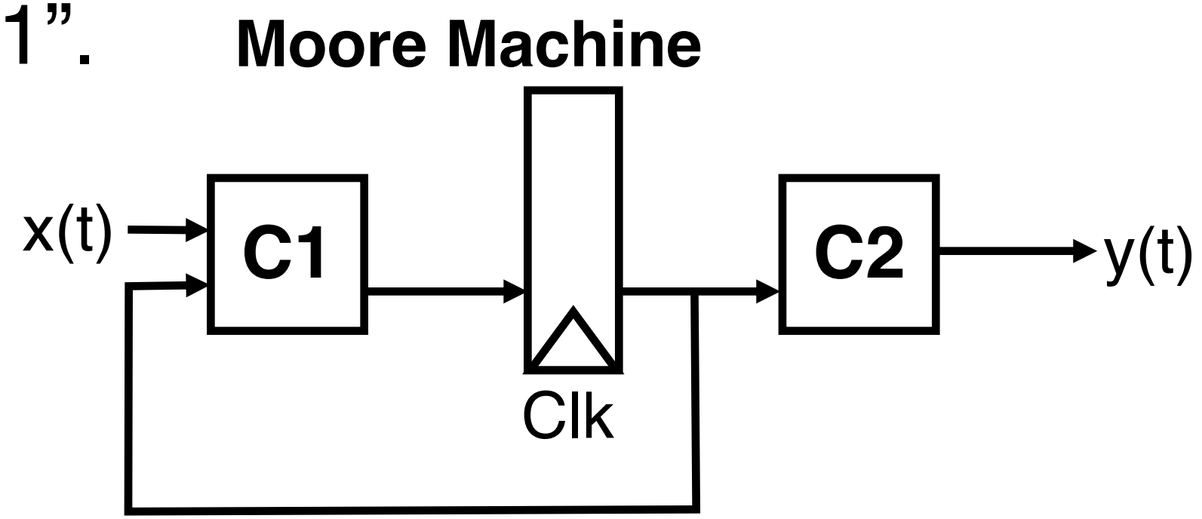
Note:

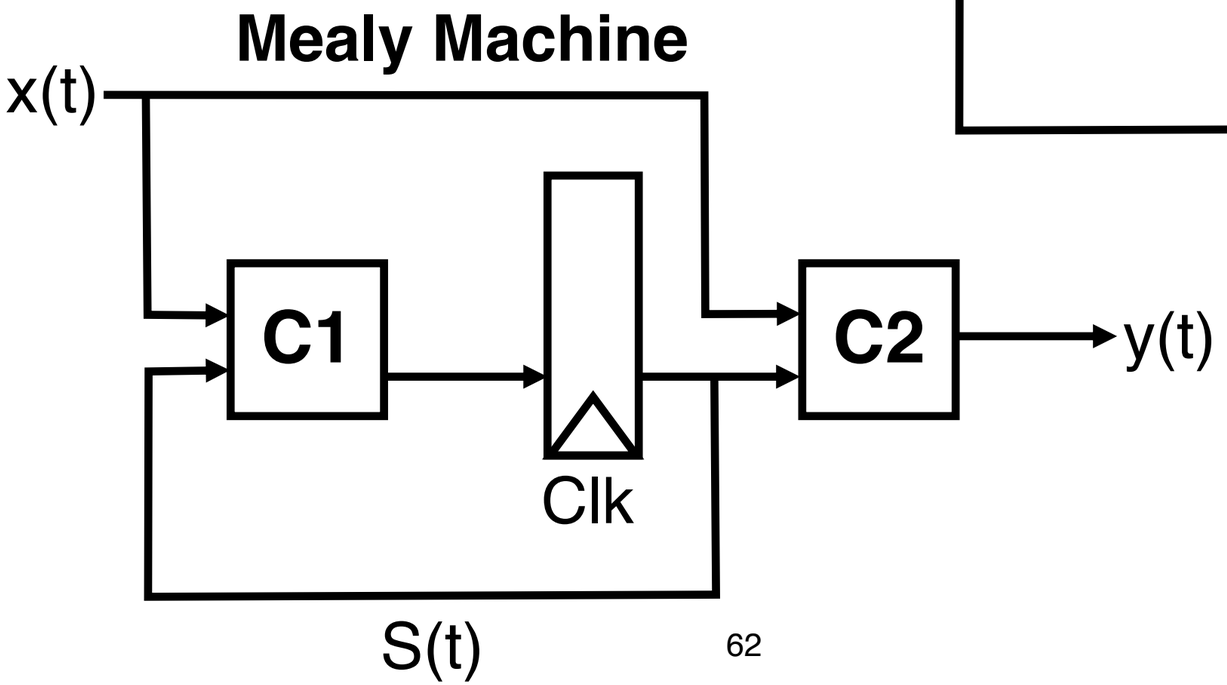
- For Mealy machine, a current output is a function of current state and input(s), e.g., Example 1
- For Moore machine, a current output is only a function of current state, e.g., Example 2

Converting Mealy to Moore

Moore or Mealy? — Life on Mars

- Which type of state machine can describe the “Life on Mars” pattern recognizer of “001”.
  A. Moore machine
  B. Mealy machine
  C. Both
  D. None

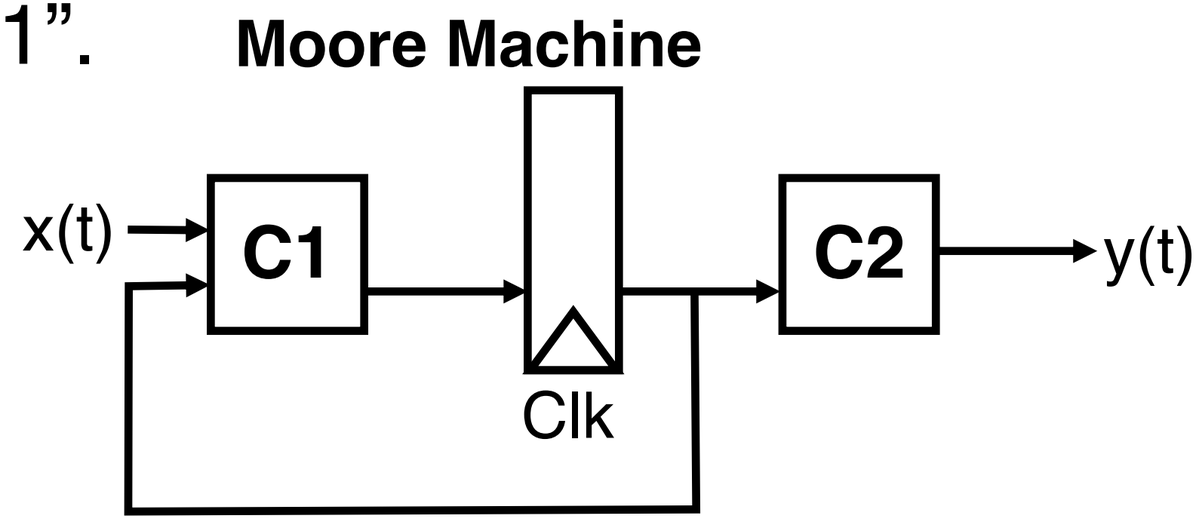


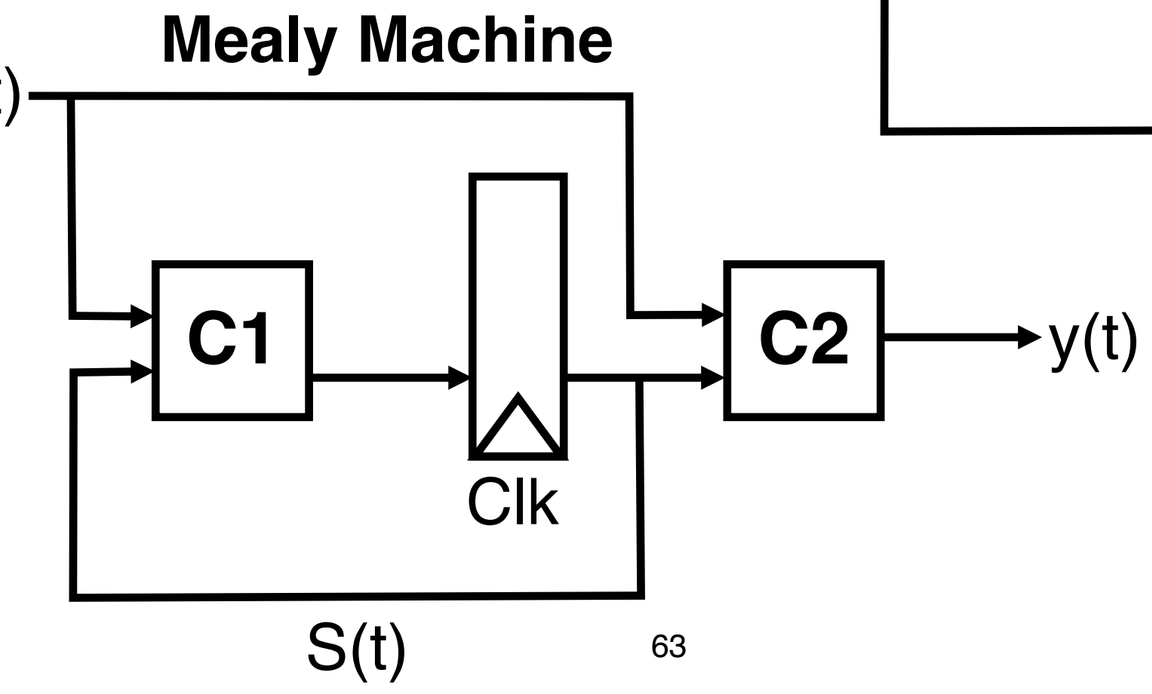


Slides adopted from Hung-Wei Tseng

Moore or Mealy? — Life on Mars

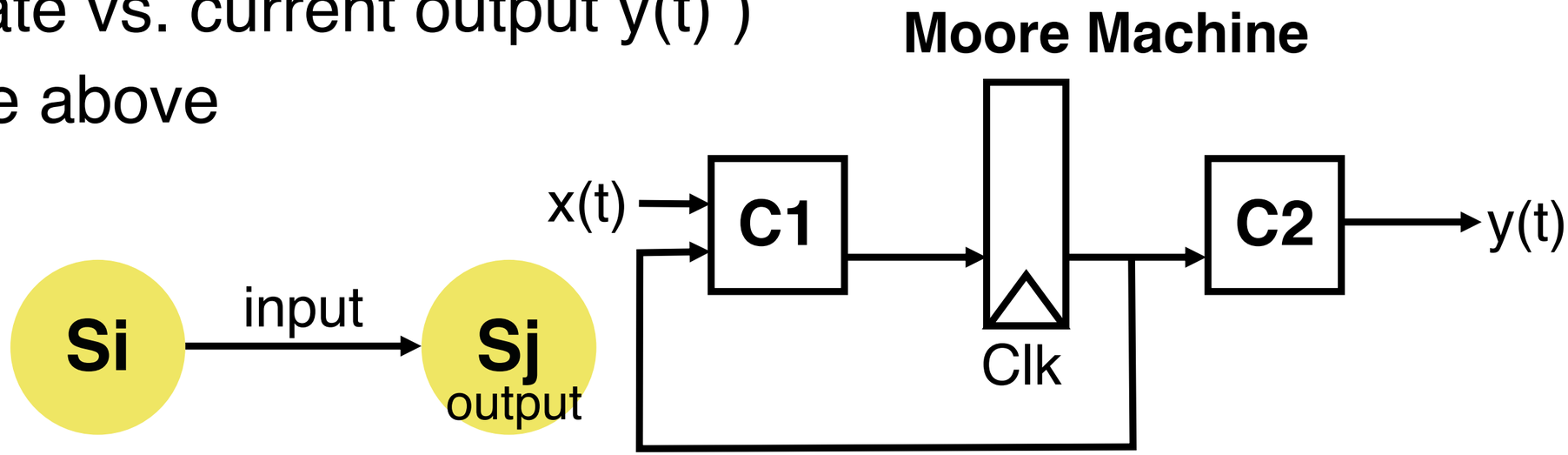
- Which type of state machine can describe the “Life on Mars” pattern recognizer of “001”.
  A. Moore machine
  B. Mealy machine
  C. Both
  D. None





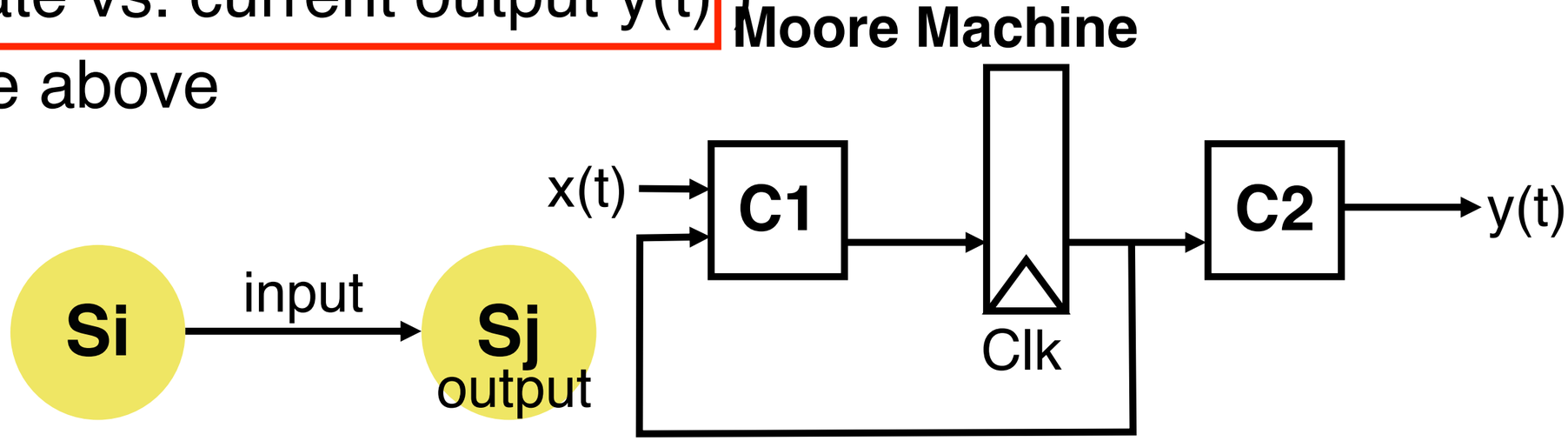
Moore machine for Life on Mars

- What does state table need to show to design controls of C2 to implement pattern recognizer “001”?
  A. (current input x(t), current state S(t) vs. next state S(t+1))
  B. (current input, current state vs. current output y(t))
  C. (current state vs. current output y(t) and next state)
  D. (current state vs. current output y(t))
  E. None of the above



Moore machine for Life on Mars

- What does state table need to show to design controls of C2 to implement pattern recognizer “001”?
  A. (current input x(t), current state S(t) vs. next state, S(t+1))
  B. (current input, current state vs. current output y(t))
  C. (current state vs. current output y(t) and next state)
  D. (current state vs. current output y(t))
  E. None of the above



Conversion from Mealy to Moore

① Identify distinct (Next State, y) pair
② Replace each distinct (Next State, y) pair with distinct new states
③ Insert rows of present state = new states
④ Append each present state with its output y

Mealy machine

CurrentState	Next State Input 0	Next State Input 1
S0	S1, 0	S0, 0
S1	S2, 0	S0, 0
S2	S2, 0	S0, 1

Conversion from Mealy to Moore

① **Identify distinct (Next State, y) pair**

② Replace each distinct (Next State, y) pair with distinct new states

③ Insert rows of present state = new states

④ Append each present state with its output y

CurrentState	Next State Input	
	0	1
S0	S1, 0	S0, 0
S1	S2, 0	S0, 0
S2	S2, 0	S0, 1

Distinct (Next State, y) pairs:

S0, 0
S1, 0
S2, 0
S0, 1

Conversion from Mealy to Moore

① Identify distinct (Next State, y) pair

② **Replace each distinct (Next State, y) pair with distinct new states**

③ Insert rows of present state = new states

④ Append each present state with its output y

CurrentState	Next State Input	
	0	1
S0	S1, 0	S0, 0
S1	S2, 0	S0, 0
S2	S2, 0	S0, 1

Distinct (Next State, y) pairs: → **Distinct new states:**

S0, 0 → S0

S1, 0 → S1

S2, 0 → S2

S0, 1 → S3

Conversion from Mealy to Moore

① Identify distinct (Next State, y) pair

② Replace each distinct (Next State, y) pair with distinct new states

③ Insert rows of present state = new states

④ Append each present state with its output y

CurrentState	Next State Input	
	0	1
S0	S1, 0	S0, 0
S1	S2, 0	S0, 0
S2	S2, 0	S0, 1

Distinct (Next State, y) pairs: → **Distinct new states:**

S0, 0 → S0

S1, 0 → S1

S2, 0 → S2

S0, 1 → S3

Conversion from Mealy to Moore

3: Insert rows of present state = new states

CurrentState	Next State Input	
	0	1
S0	S1, 0	S0, 0
S1	S2, 0	S0, 0
S2	S2, 0	S0, 1

Distinct (Next State, y) pairs:

S0, 0

S1, 0

S2, 0

S0, 1

Distinct new states:

S0

S1

S2

S3

CurrentState	Next State Input	
	0	1
S0	S1	S0
S1	S2	S0
S2	S2	S3
S3		

Conversion from Mealy to Moore

① Identify distinct (Next State, y) pair

② Replace each distinct (Next State, y) pair with distinct new states

③ Insert rows of present state = new states

④ **Append each present state with its output y**

Distinct (Next State, y) pairs: → **Distinct new states:**

S0, 0 → S0

S1, 0 → S1

S2, 0 → S2

S0, 1 → S3

CurrentState	Next State Input 0	Next State Input 1	Output
S0	S1	S0	0
S1	S2	S0	0
S2	S2	S3	0
S3			1

Conversion from Mealy to Moore

① Identify distinct (Next State, y) pair

② Replace each distinct (Next State, y) pair with distinct new states

③ Insert rows of present state = new states

④ **Append each present state with its output y**

Distinct (Next State, y) pairs:

S0, 0

S1, 0

S2, 0

S0, 1

Distinct new states:

S0

S1

S2

S3

CurrentState	Next State Input 0	Next State Input 1	Output
S0	S1	S0	0
S1	S2	S0	0
S2	S2	S3	0
S3			1

Conversion from Mealy to Moore

Distinct (Next State, y) pairs: **Distinct new states:**

S0, 0 → S0

S1, 0 → S1

S2, 0 → S2

S0, 1 → S3

CurrentState	Next State Input 0	Next State Input 1	Output
S0	S1	S0	0
S1	S2	S0	0
S2	S2	S3	0
S3	?	?	1

? For the given Moore machine, what are the next states with respect to present state S3

Conversion from Mealy to Moore

Distinct (Next State, y) pairs:

Distinct new states:

S0, 0 → S0

S1, 0 → S1

S2, 0 → S2

S0, 1 → S3

CurrentState	Next State Input 0	Next State Input 1	Output
S0	S1	S0	0
S1	S2	S0	0
S2	S2	S3	0
S3	S1	S0	1

CurrentState	Next State Input 0	Next State Input 1
S0	S1, 0	S0, 0
S1	S2, 0	S0, 0
S2	S2, 0	S0, 1

Summary

① Identify distinct (Next State, y) pair

② Replace each distinct (Next State, y) pair with distinct new states

③ Insert rows of present state = new states

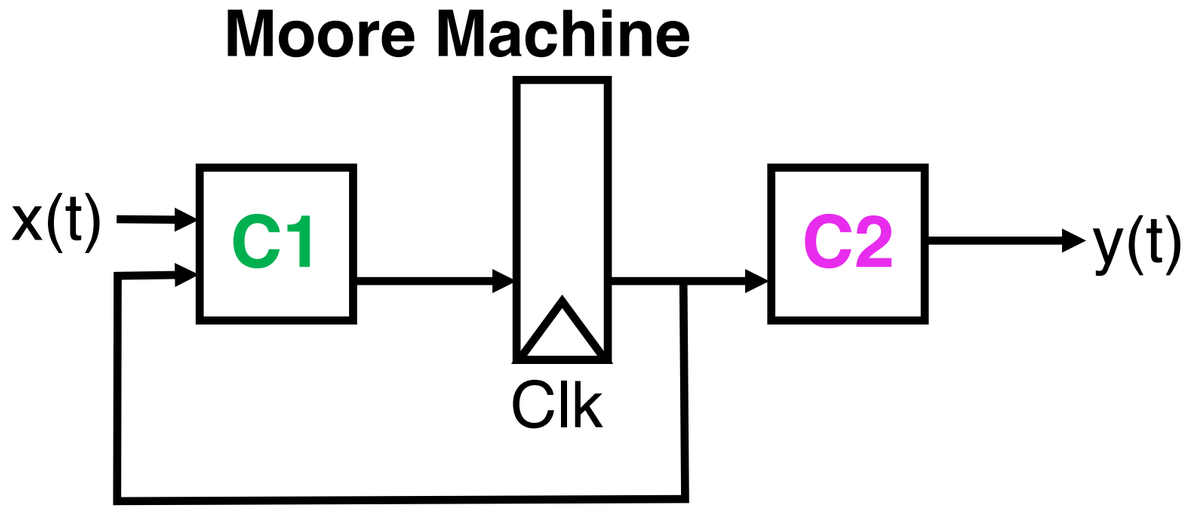
④ Append each present state with its output y

CurrentState	Next State / Input	
	0	1
S0	S1, 0	S0, 0
S1	S2, 0	S0, 0
S2	S2, 0	**S0, 1**

CurrentState	Next State / Input		Output
	0	1	
S0	S1	S0	0
S1	S2	S0	0
S2	S2	**S3**	0
S3	**S1**	**S0**	**1**

State tables for C1 and C2

Current State		Next State Input 0	Next State Input 1	Output
S0	00	S1	S0	0
S1	01	S2	S0	0
S2	10	S2	S3	0
S3	11	S1	S0	1



C1

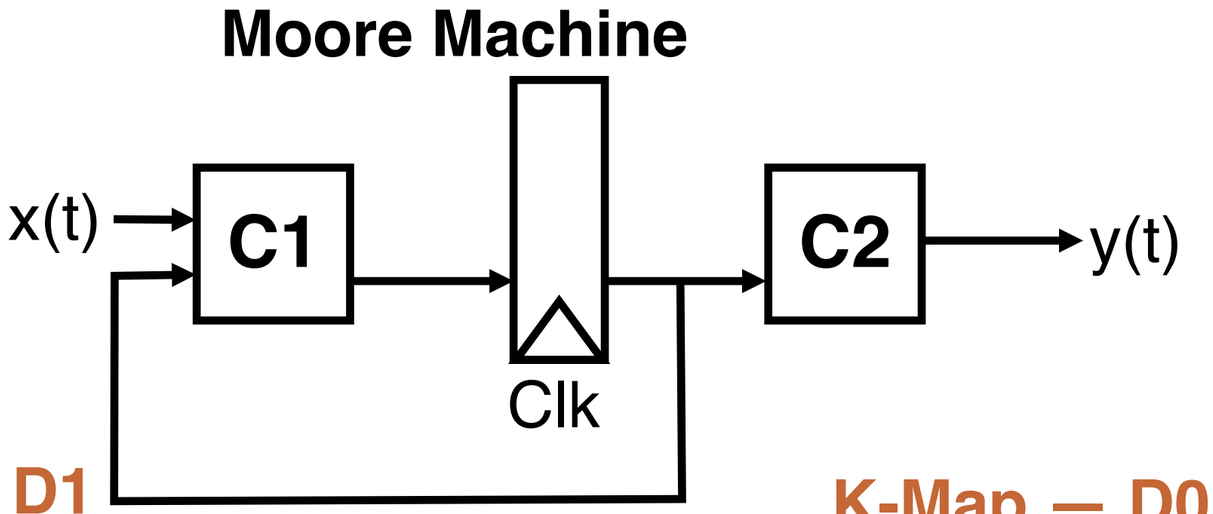
CurrentState-Input	D1	D0
000	0	1
001	0	0
010	1	0
011	0	0
100	1	0
101	1	1
110	0	1
111	0	0

C2

State	y
00	0
01	0
10	0
11	1

State tables for C1 and C2

Current State		Next State Input 0	Next State Input 1	Output
S0	00	S1	S0	0
S1	01	S2	S0	0
S2	10	S2	S3	0
S3	11	S1	S0	1



C1

CurrentState-Input	D1	D0
000	0	1
001	0	0
010	1	0
011	0	0
100	1	0
101	0	0
110	0	1
111	0	0

C2

State	y
00	0
01	0
10	0
11	1

K-Map — D1

	0,0	0,1	1,1	1,0
0	0	1	0	1
1	0	0	0	1

$D1 = x'Q_1'Q_0 + Q_1Q_0'$

K-Map — D0

	0,0	0,1	1,1	1,0
0	1	0	1	0
1	0	0	0	1

$D0 = Q_0'Q_1'x' + Q_0Q_1x' + Q_0'Q_1x'$

K-Map — y

	0	1
0	0	0
1	0	1

$y = Q_0Q_1$