CSE 140: Components and Design Techniques for Digital Systems

Lecture 7:
Sequential Networks

CK Cheng
Dept. of Computer Science and Engineering
University of California, San Diego

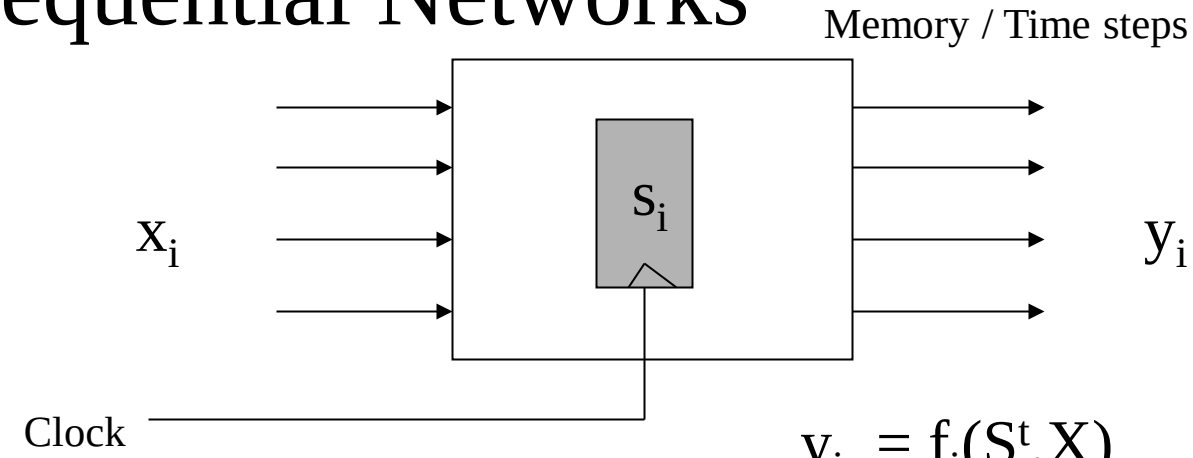
What is a sequential circuit?

"A circuit whose output depends on current inputs and past outputs"

"A circuit with memory"

Memory => Time

Part II. Sequential Networks



$y_i = f_i(S^t, X)$

$s_i^{t+1} = g_i(S^t, X)$

Memory: Flip flops

Specification: Finite State Machines

Implementation: Excitation Tables

Main Theme: Timing

Present time = t and next time = t+1

Timing constraints to separate the present and next times.

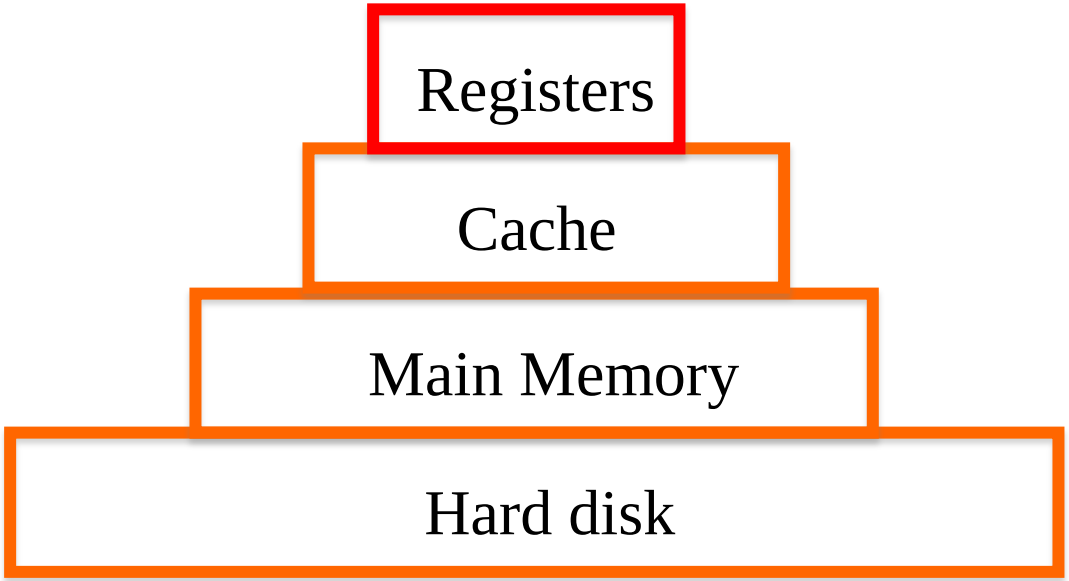
Sequential Networks

- Memory Components
  - Hierarchy of Memory
  - Basic Mechanism of Memory
  - Types of Flip-Flops
- Implementation
  - Finite State Machine

Memory Hierarchy

- What are registers made of?

Flip-Flops, Latches

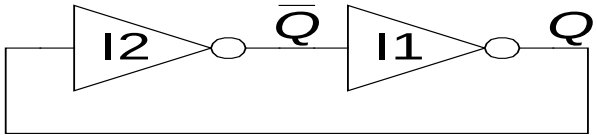


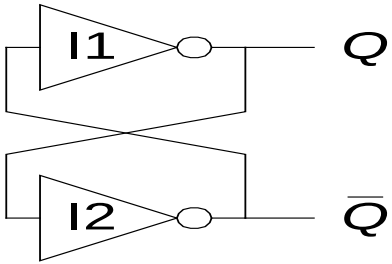
The usage of a sequential machine

iClicker Question:

A. Digital systems are implemented using sequential machines.
B. Only a small subset of digital systems can be implemented using sequential machines.
C. Sequential machines are too simple for complicated digital systems.

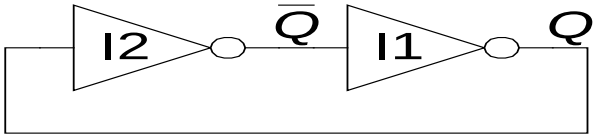
Fundamental Memory Mechanism

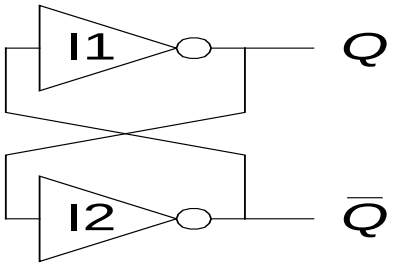




Memory Mechanism: Capacitive Load

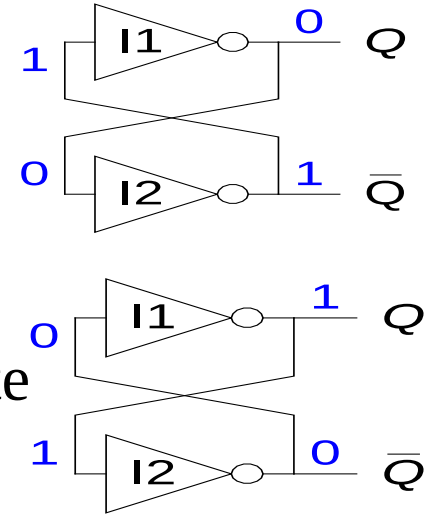
- Fundamental building block of sequential circuits
- Two outputs: Q, $\overline{Q}$
- There is a feedback loop!
  - In a typical combinational logic, there is no feedback loop.
- No inputs





Capacitive Loads

- Consider the two possible cases:
  - $Q = 0$: then $Q' = 1$ and $Q = 0$ (consistent)
  - $Q = 1$: then $Q' = 0$ and $Q = 1$ (consistent)
  - Bistable circuit stores 1 bit of state in the state variable, Q (or Q')
  - Hold the value due to capacitive charges and feedback loop strengthening



- But there are **no inputs to control the state**

iClicker

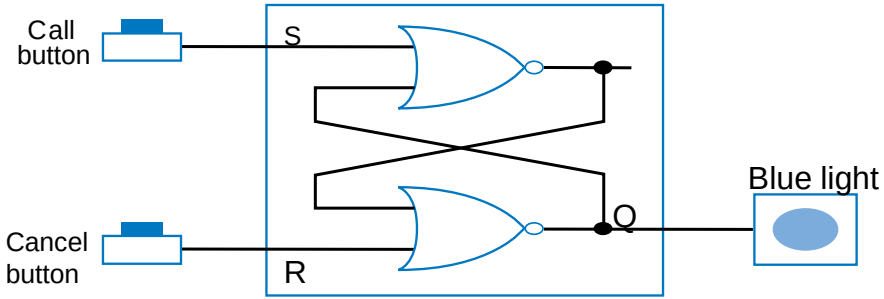
Q. Given a memory component made out of a loop of inverters, the number of inverters in the loop has to be

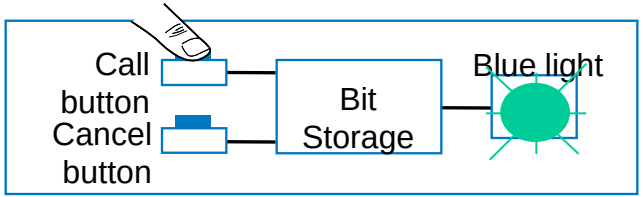
A. Even

B. Odd

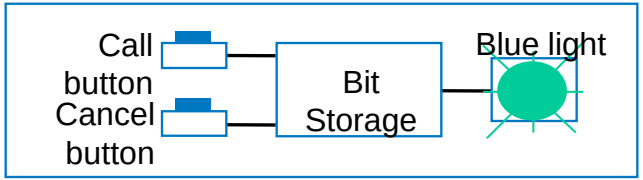
Flight attendant call button

- Flight attendant call button
  - Press call: light turns on
    - ***Stays on*** after button released
  - Press cancel: light turns off
  - Logic gate circuit to implement this?
- SR latch implementation
  - Call=1 : sets Q to 1 and keeps it at 1
  - Cancel=1 : resets Q to 0

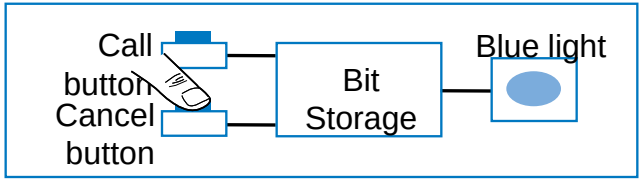




1. Call button pressed – light turns on



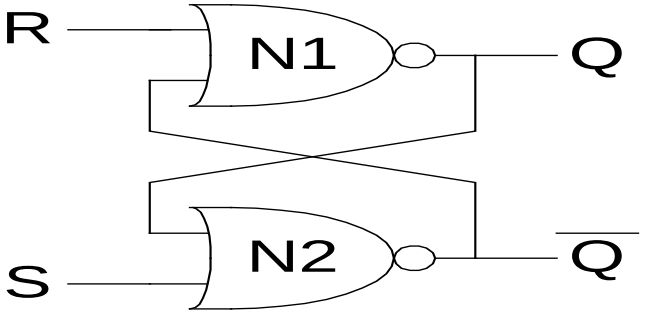
*2. Call button released – light **stays on***



3. Cancel button pressed – light turns off

SR (Set/Reset) Latch

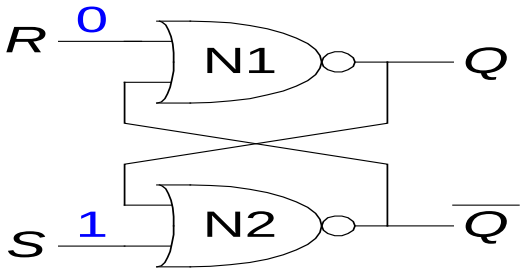
- SR Latch



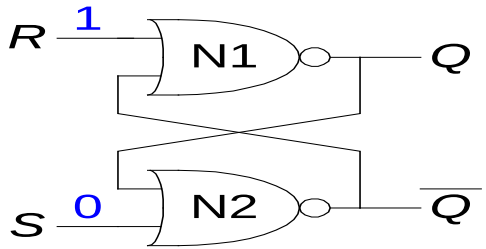
- Consider the four possible cases:
  - $S = 1, R = 0$
  - $S = 0, R = 1$
  - $S = 0, R = 0$
  - $S = 1, R = 1$

SR Latch Analysis

– $S = 1, R = 0$:

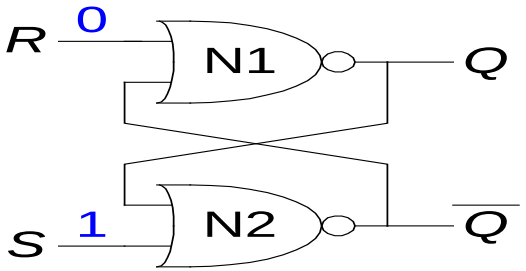


– $S = 0, R = 1$:

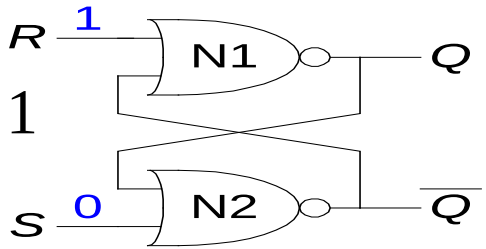


SR Latch Analysis

– $S = 1, R = 0$: then $Q = 1$ and $\overline{Q} = 0$

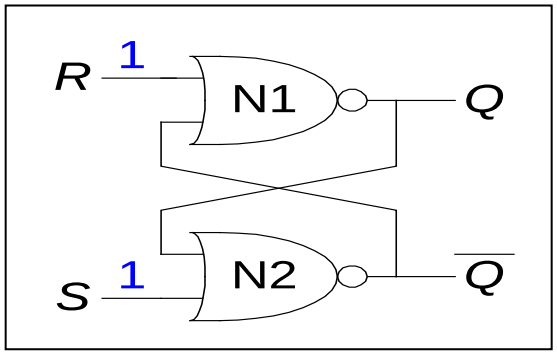


– $S = 0, R = 1$: then $Q = 0$ and $\overline{Q} = 1$



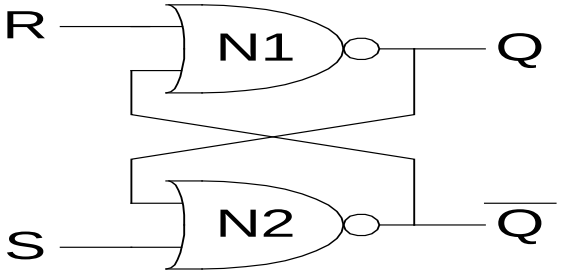
SR Latch Analysis

– $S = 1, R = 1$:



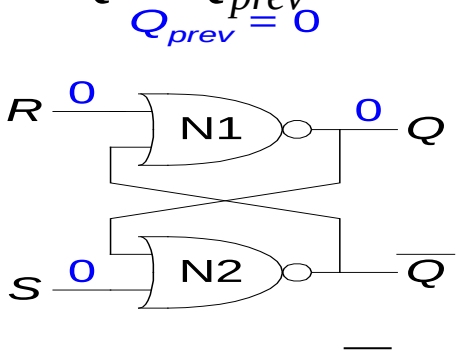
SR Latch Analysis

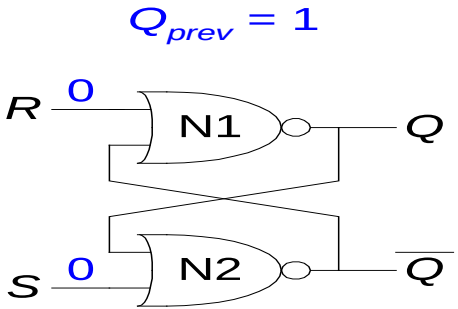
– $S = 0, R = 0$:



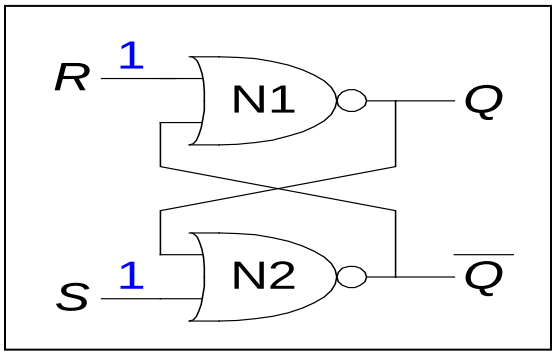
SR Latch Analysis

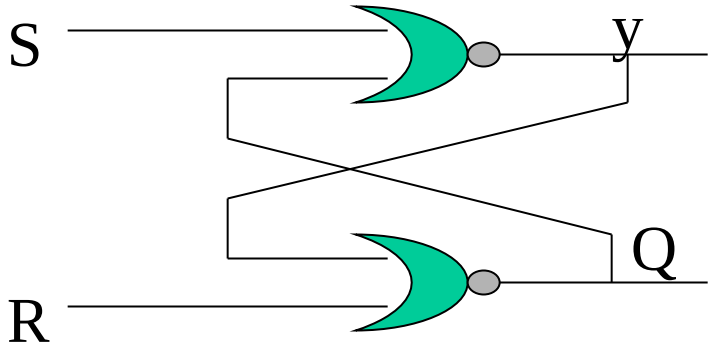
– $S = 0, R = 0$: then $Q = Q_{prev}$





– $S = 1, R = 1$: then $Q = 0$ and $Q = 0$



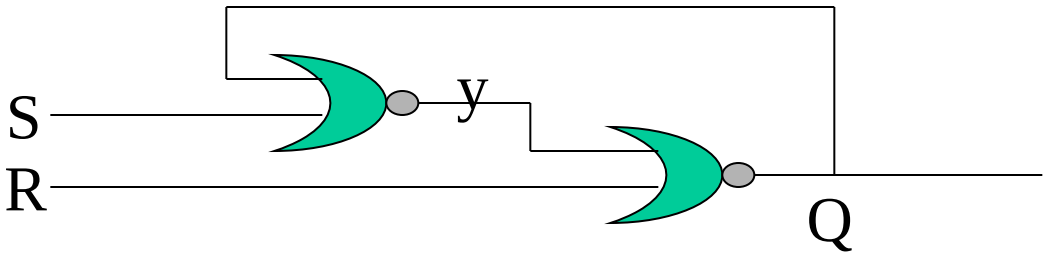


y = (S+Q)’

Q = (R+y)’

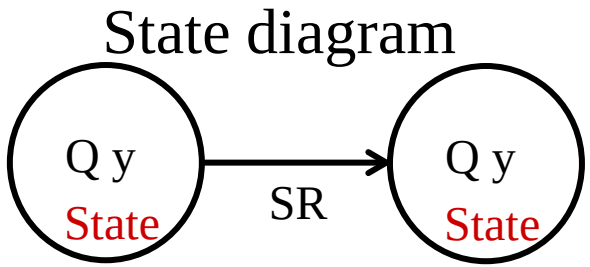
SR Latch

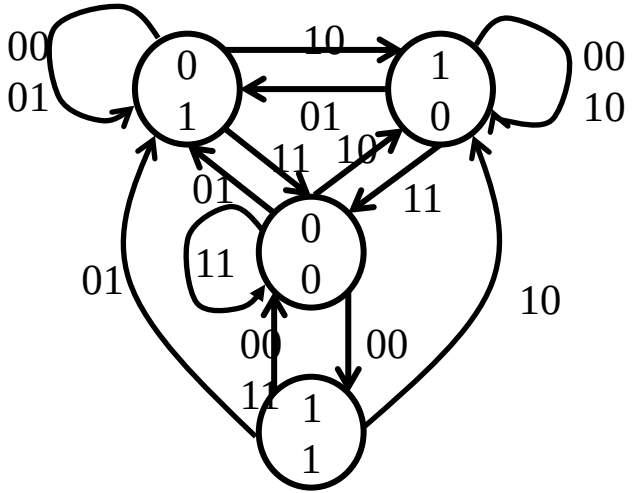
SR F-F (Set-Reset)



Inputs: S, R State: (Q, y)

Id	Q(t)	y(t)	S	R	$Q(t_1)$	$y(t_1)$	$Q(t_2)$	$y(t_2)$	$Q(t_3)$	$y(t_3)$
0	0	0	0	0	1	1	0	0	1	1
1	0	0	0	1	0	1	0	1	0	1
2	0	0	1	0	1	0	1	0	1	0
3	0	0	1	1	0	0	0	0	0	0
4	0	1	0	0	0	1	0	1	0	1
5	0	1	0	1	0	1	0	1	0	1
6	0	1	1	0	0	0	1	0	1	0
7	0	1	1	1	0	0	0	0	0	0
8	1	0	0	0	1	0	1	0	1	0
9	1	0	0	1	0	0	0	1	0	1
10	1	0	1	0	1	0	1	0	1	0
11	1	0	1	1	0	0	0	0	0	0
12	1	1	0	0	0	0	1	1	0	0
13	1	1	0	1	0	0	0	1	0	1
14	1	1	1	0	0	0	1	0	1	0
15	1	1	1	1	0	0	0	0	0	0





CASES:

SR=01, (Q,y) = (0,1)

SR=10, (Q,y) = (1,0)

SR=11, (Q,y) = (0,0)

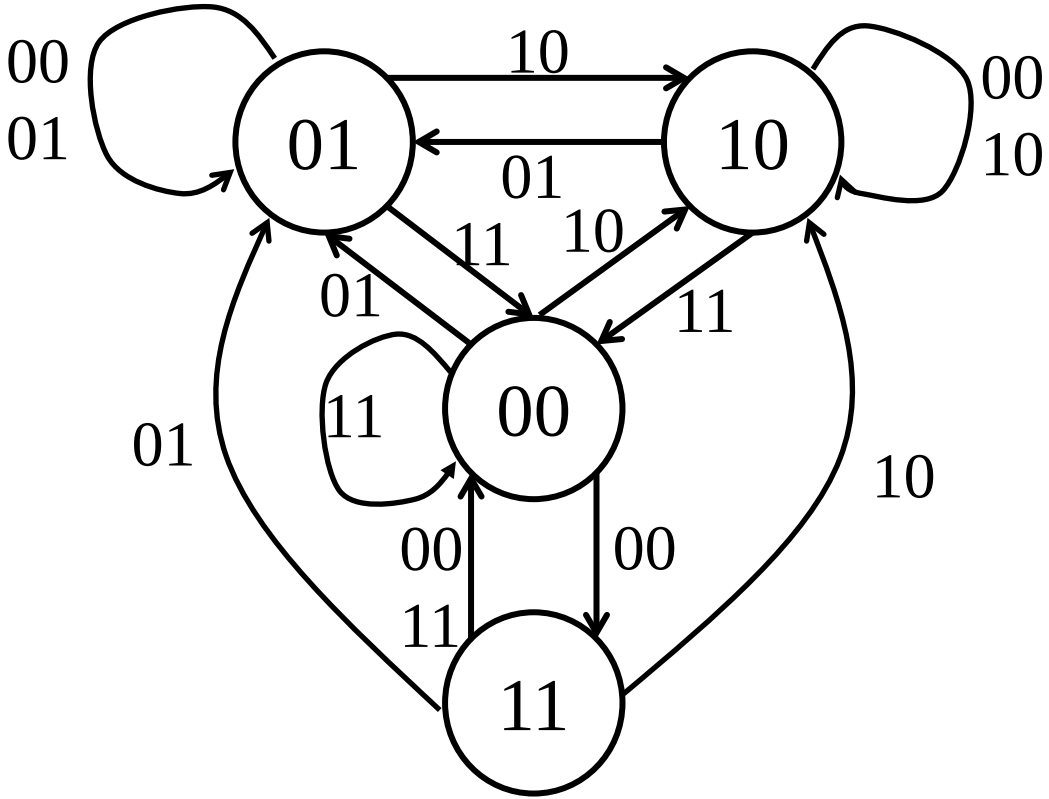
SR= 00 => if (Q,y) = (0,0) or (1,1), the output keeps changing

Q. To avoid the SR latch output from toggling or behaving in an undefined way which input combinations should be avoided:

A. (S, R) = (0, 0)

B. (S, R) = (1, 1)

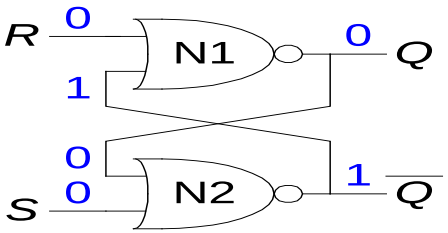
SR Latch



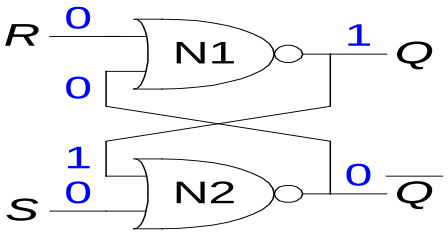
SR Latch Analysis

– $S = 0, R = 0$: then $Q = Q_{prev}$ and $\overline{Q} = \overline{Q_{prev}}$ (memory!)

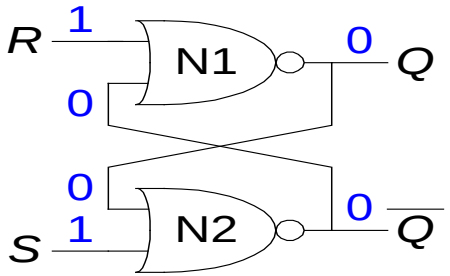
$Q_{prev} = 0$



$Q_{prev} = 1$



– $S = 1, R = 1$: then $Q = 0$ and $\overline{Q} = 0$ (invalid state: $Q \neq$ NOT $\overline{Q}$)



CASES

SR=01: (Q,y) = (0,1)

SR=10: (Q,y) = (1,0)

SR=11: (Q,y) = (0,0)

SR = 00: if (Q,y) = (0,0) or (1,1), the output keeps changing

Solutions: Avoid the two cases

1) SR = (0,0),

2) SR = (1,1).

State table

PS \ inputs SR	00	01	10	11
Q(t) 0	0	0	1	-
1	1	0	1	-

Q(t+1) NS (next state)

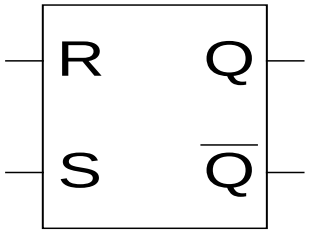
Characteristic Expression

Q(t+1) = S(t)+R’(t)Q(t)

SR Latch Symbol

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

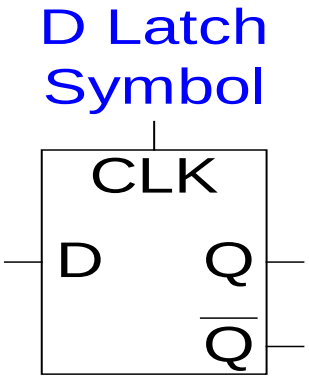
SR Latch Symbol



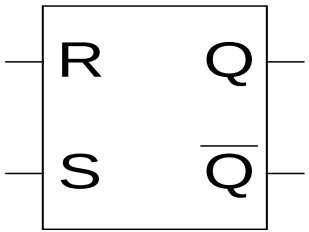
- **Must do something to avoid invalid state (when $S = R = 1$)**

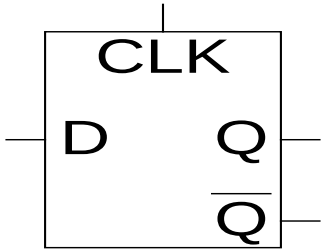
D Latch

- 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* (the latch is *transparent*)
  - When *CLK* = 0, *Q* holds its previous value (the latch is *opaque*)
- Avoids invalid case when $Q \neq \text{NOT } Q$

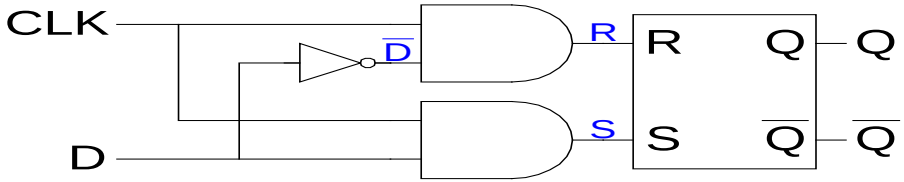


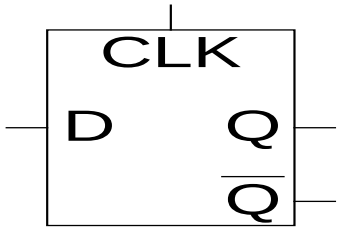
D Latch Internal Circuit





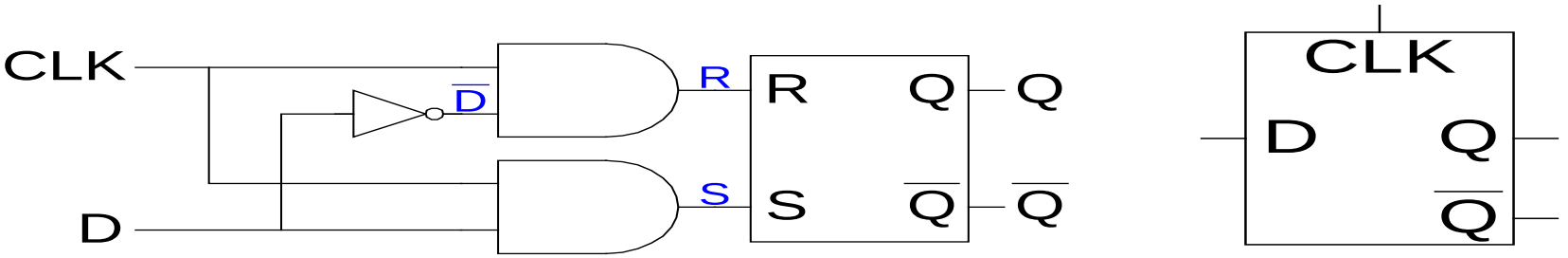
D Latch Internal Circuit





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

D Latch Internal Circuit

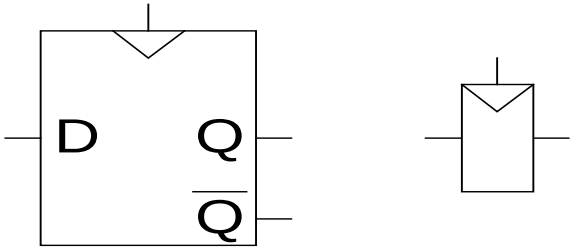


CLK	D	$\overline{D}$	S	R	Q	$\overline{Q}$
0	X	$\overline{X}$	0	0	Q_{prev}	$\overline{Q}_{prev}$
1	0	1	0	1	0	1
1	1	0	1	0	1	0

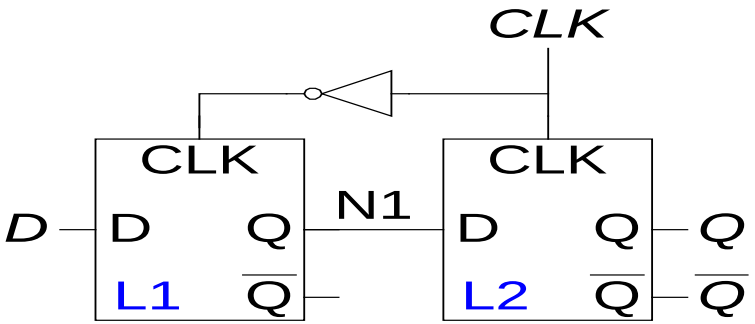
D Flip-Flop

- Two inputs: *CLK*, *D*
- *Function*
  - The flip-flop "samples" *D* on the 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 the rising edge of *CLK*
- A flip-flop is called an *edge-triggered* device because it is activated on the clock edge

D Flip-Flop Symbols

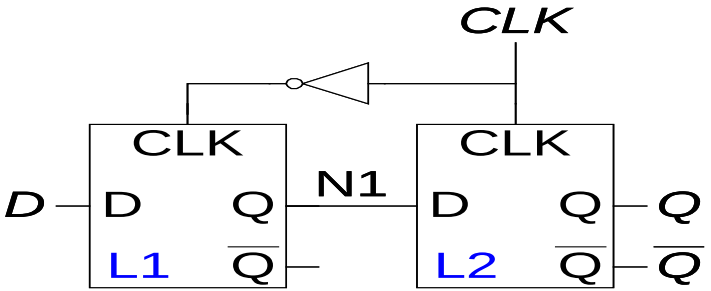


D Flip-Flop Internal Circuit

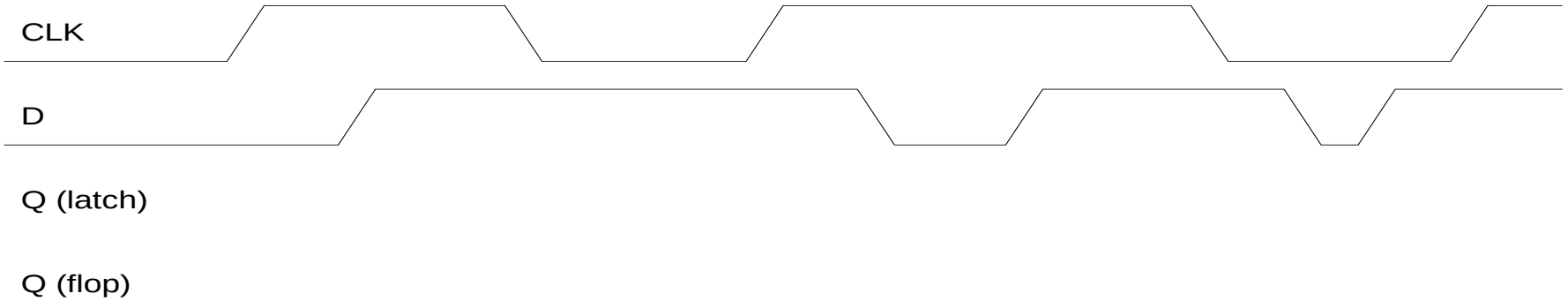


D Flip-Flop Internal Circuit

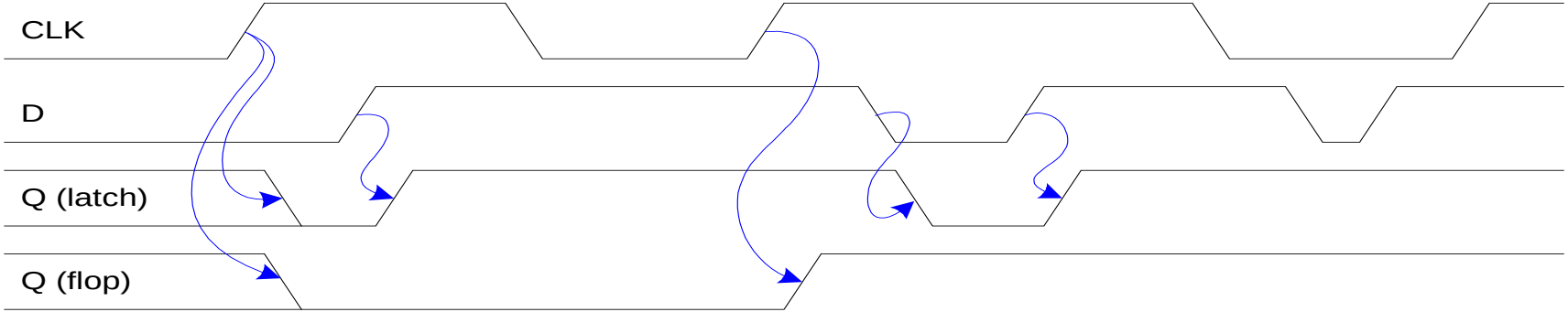
- Two back-to-back latches (L1 and L2) controlled by complementary clocks
- When *CLK* = 0
  - L1 is transparent, L2 is opaque
  - *D* passes through to N1
- When *CLK* = 1
  - L2 is transparent, L1 is opaque
  - N1 passes through to *Q*
- Thus, on the edge of the clock (when *CLK* rises from 0 → 1)
  - *D* passes through to *Q*



D Flip-Flop vs. D Latch

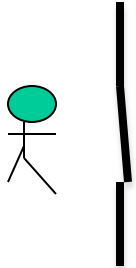


D Flip-Flop vs. D Latch

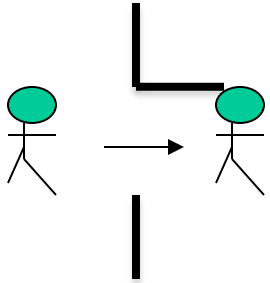


Latch and Flip-flop (two latches)

A latch can be considered as a door

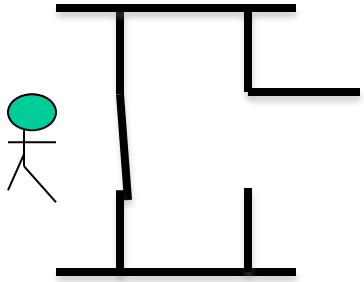


CLK = 0, door is shut

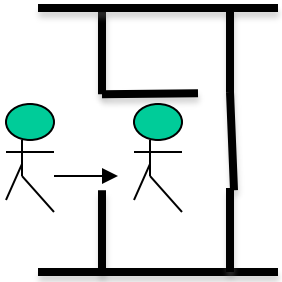


CLK = 1, door is unlocked

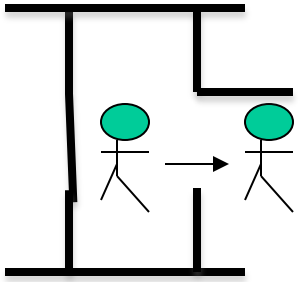
A flip-flop is a two door entrance



CLK = 1

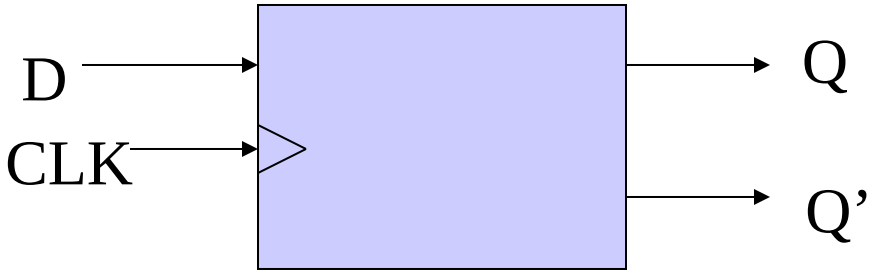


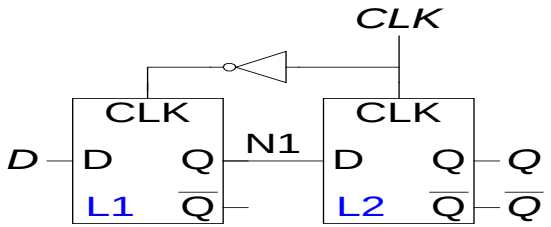
CLK = 0



CLK = 1

D Flip-Flop (Delay)





Id	D	Q(t)	Q(t+1)
0	0	0	0
1	0	1	0
2	1	0	1
3	1	1	1

State table

PS \ D	0	1
0	0	1
1	0	1

NS= Q(t+1)

Characteristic Expression: Q(t+1) = D(t)

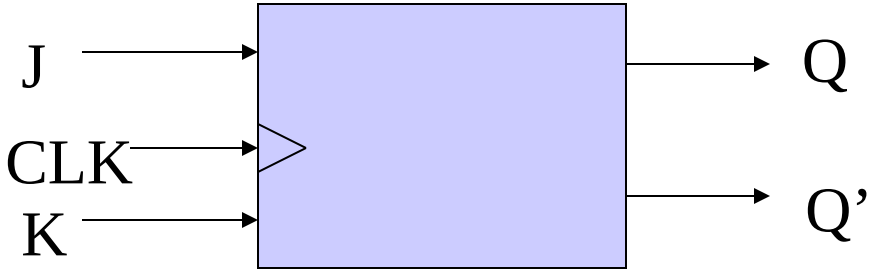
iClicker

Can D flip-flip serve as a memory component?

A. Yes

B. No

JK F-F

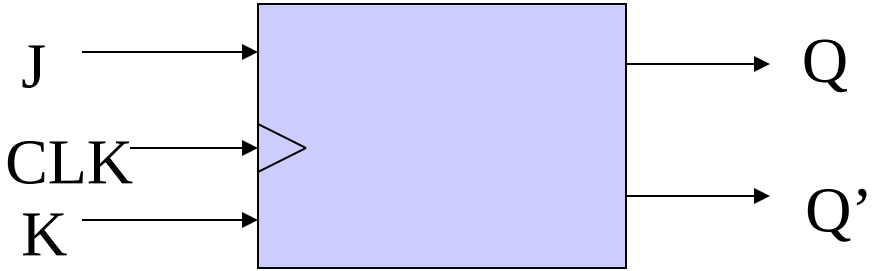


State table

PS \ JK	00	01	10	11
0	0	0	1	?
1	1	0	1	?

Q(t+1)

JK F-F



State table

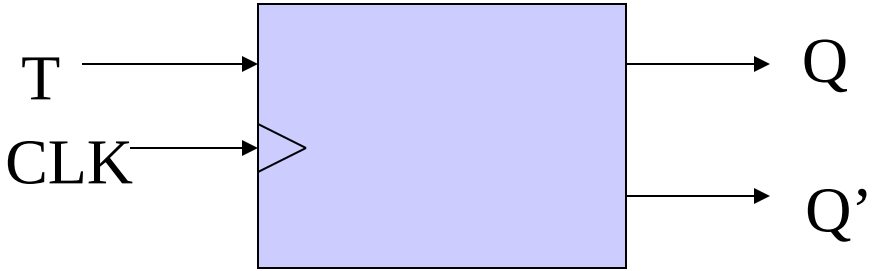
PS \ JK	00	01	10	11
0	0	0	1	1
1	1	0	1	0

Q(t+1)

Characteristic Expression

$Q(t+1) = Q(t)K'(t)+Q'(t)J(t)$

T Flip-Flop (Toggle)



State table

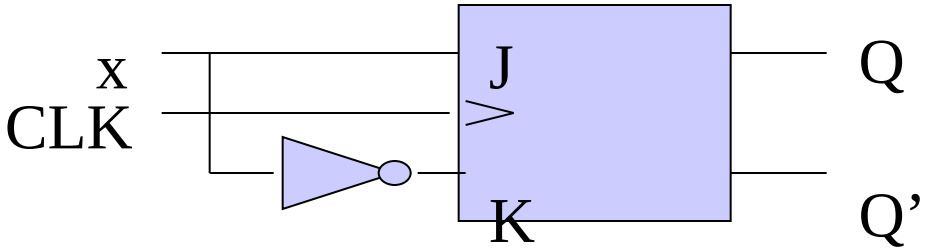
PS \ T	0	1
0	0	1
1	1	0

Q(t+1)

Characteristic Expression

$$Q(t+1) = Q'(t)T(t) + Q(t)T'(t)$$

Using a JK F-F to implement a D and T F-F

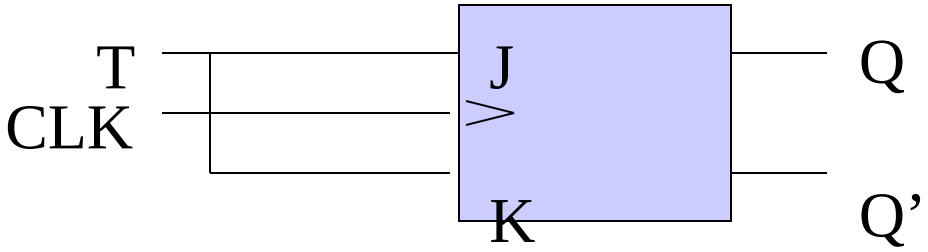


iClicker

What is the function of the above circuit?

A. D F-F
B. T F-F
C. None of the above

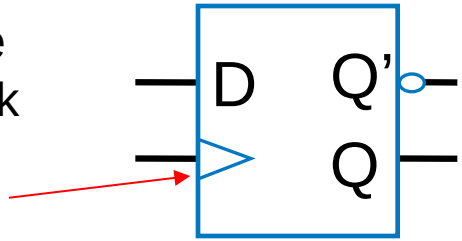
Using a JK F-F to implement a D and T F-F



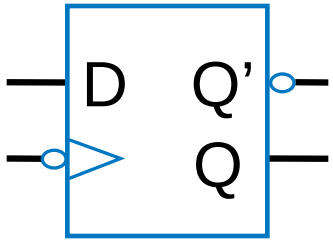
T flip flop

Rising vs. Falling Edge D Flip-Flop

The triangle means clock input, edge triggered

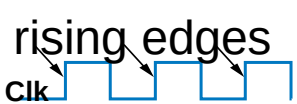


Symbol for rising-edge triggered D flip-flop



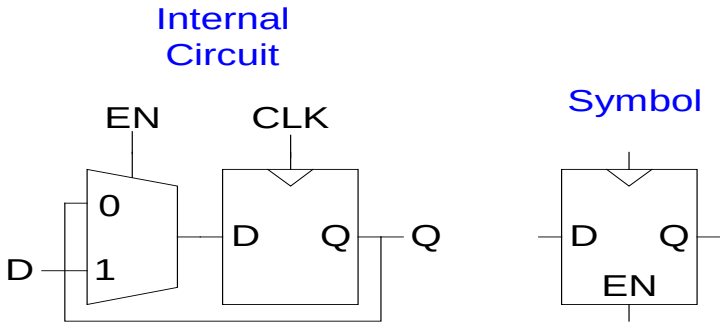
Internal design: Just invert servant clock rather than master

Symbol for falling-edge triggered D flip-flop



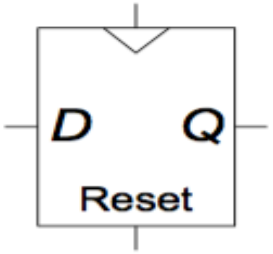
Enabled D-FFs

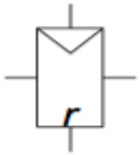
- **Inputs:** *CLK*, *D*, *EN*
  - The enable input (*EN*) controls when new data (*D*) is stored
- **Function**
  - ***EN* = 1:** *D* passes through to *Q* on the clock edge
  - ***EN* = 0:** the flip-flop retains its previous state



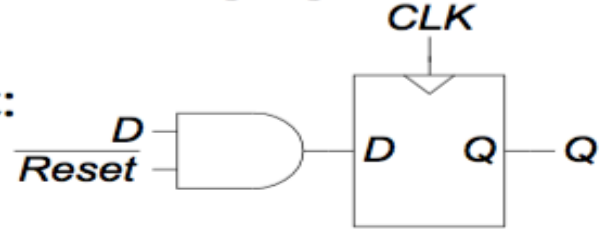
Resettable Flip-Flops

Symbols





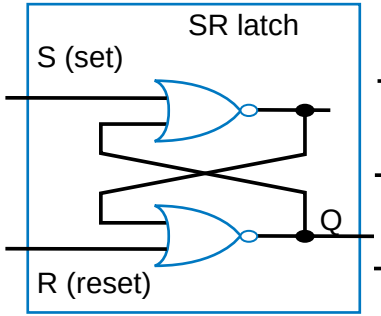
- **Inputs:** *CLK*, *D*, *Reset*
- **Function:**
  - ***Reset* = 1:** *Q* is forced to 0
  - ***Reset* = 0:** flip-flop behaves as ordinary D flip-flop
- Two types:
  - **Synchronous:** resets at the clock edge only
  - **Asynchronous:** resets immediately when *Reset* = 1
- Asynchronously resettable flip-flop requires changing the internal circuitry of the flip-flop
- Synchronously resettable flip-flop circuit:

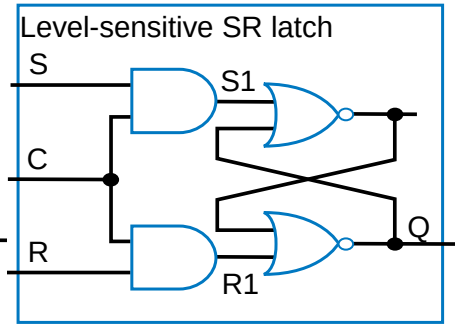


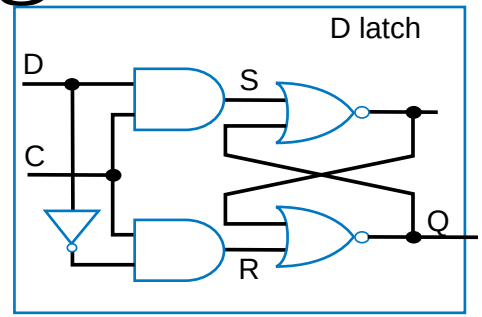
- There are also synch/asynch settable FFs

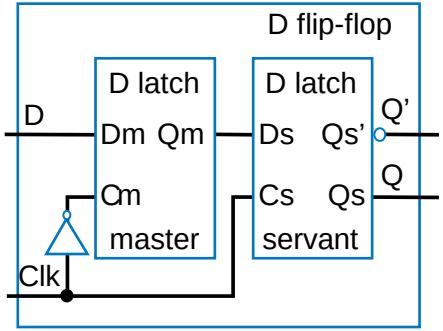
Sources: TSR, Katz, Boriello & Vahid

Bit Storage Overview









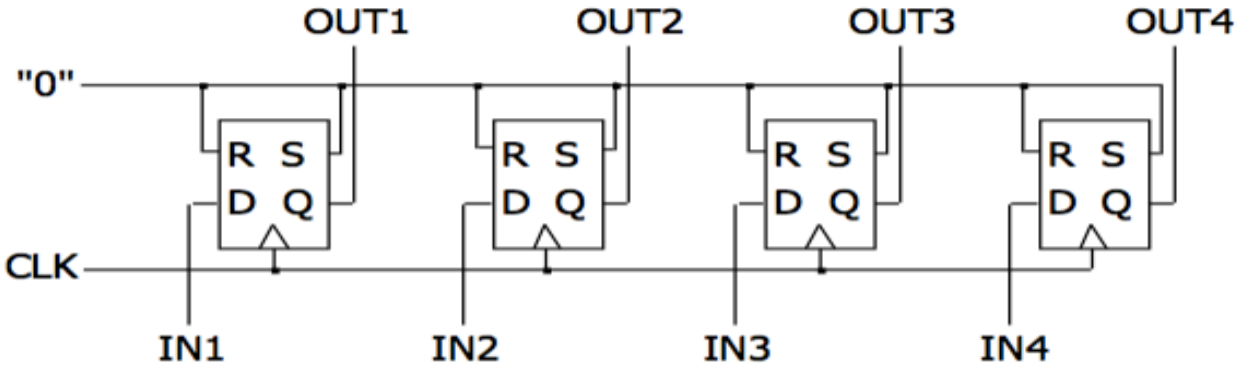
S=1 sets Q to 1, R=1 resets Q to 0. Problem: SR=11 yield undefined Q.

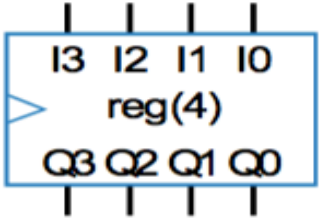
S and R only have effect when C=1. We can design outside circuit so SR=11 never happens when C=1. Problem: avoiding SR=11 can be a burden.

SR can't be 11 if D is stable before and while C=1, and will be 11 for only a brief glitch even if D changes while C=1.
*Transition may cross many levels of latches.

Only loads D value present at rising clock edge, so values can't propagate to other flip-flops during same clock cycle.
*Transition happens between two level of flip-flops.

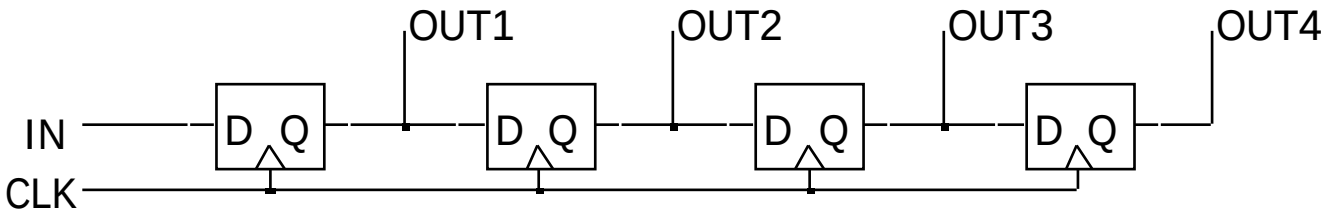
Building blocks with FFs: Basic Register





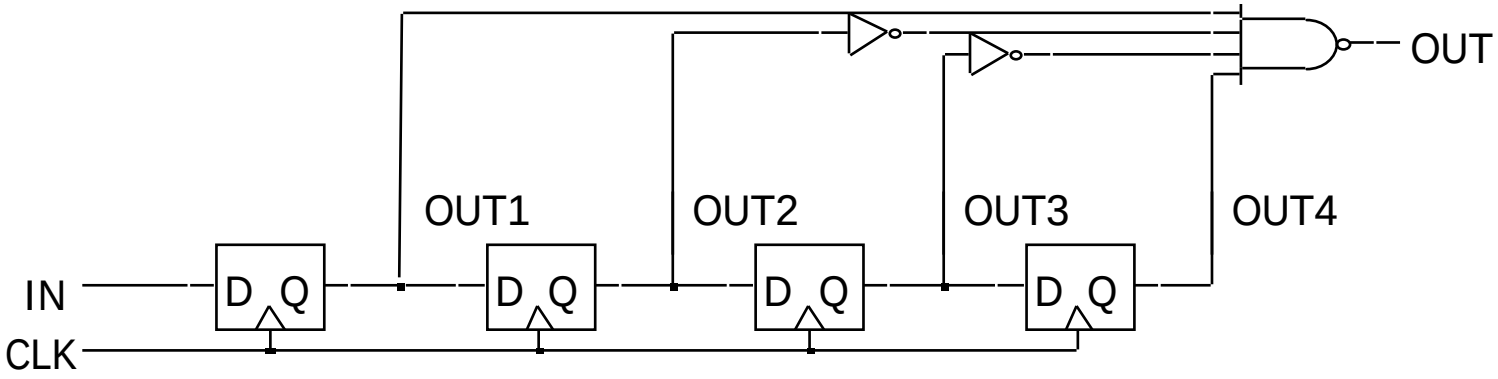
Shift register

- Holds & shifts samples of input



Pattern Recognizer

- Combinational function of input samples



Counters

- Sequences through a fixed set of patterns