BME2322 – Logic Design

The Instructors:

Dr. Görkem SERBES (C317)

gserbes@yildiz.edu.tr

https://avesis.yildiz.edu.tr/gserbes/

Lab Assistants:

Nihat AKKAN

nakkan@yildiz.edu.tr

https://avesis.yildiz.edu.tr/nakkan

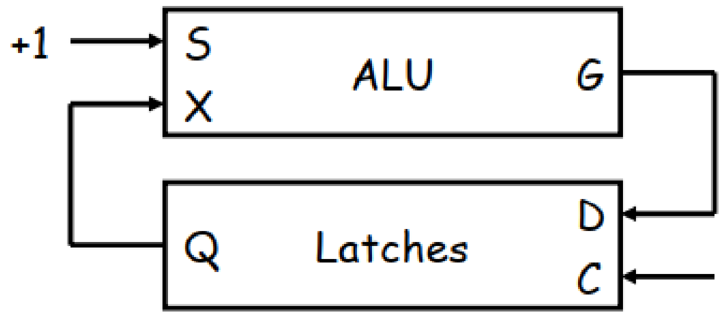
LECTURE 10

Flip-Flops

- So far, we have seen how latches can be used as memory in a circuit
- Latches inherit new problems:
  – We need to know when to enable a latch
  – We also need to quickly disable a latch
  – In other words, it's difficult to control the timing of latches in a large circuit
- We solve these problems with two new elements: clocks and flip-flops
  – Clocks tell us when to write to our memory
  – Flip-flops allow us to quickly write the memory at clearly defined times
  – Used together, we can create circuits without worrying about the memory timing

A real life problem

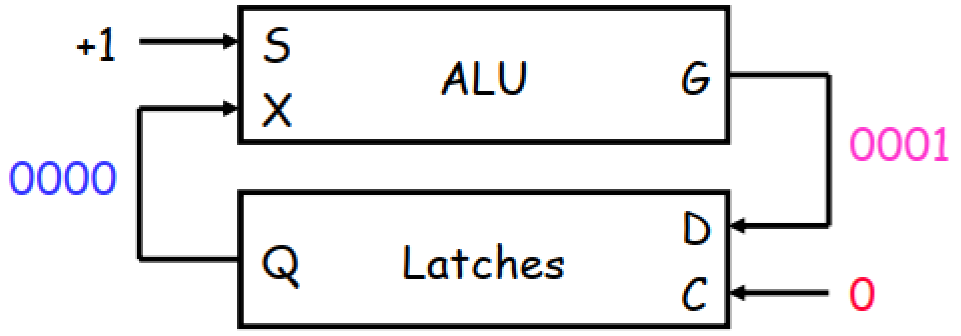
- We can connect some latches, acting as memory, to an ALU



- Let’s say these latches contain some value that we want to increment
  – The ALU should read the current latch value
  – It applies the “G = X + 1” operation
  – The incremented value is stored back into the latches
- At this point, we must stop the cycle, so the latch value doesn’t get incremented again by accident
- One convenient way to break the loop is to disable the latches

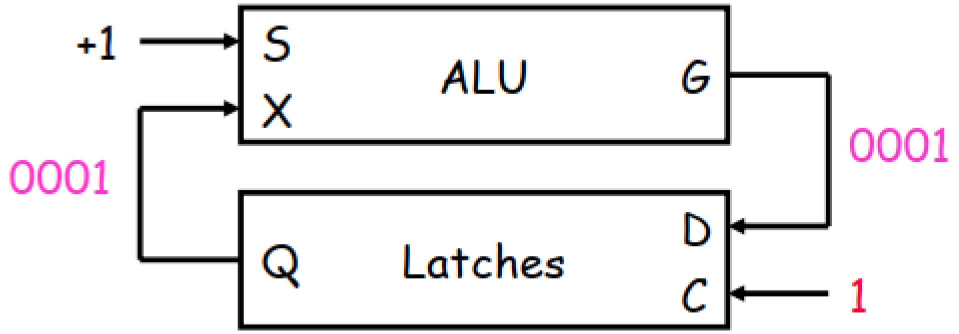
Timing issues in latches

- Our example used latches as memory for an ALU
  – Let’s say there are four latches initially storing 0000
  – We want to use an ALU to increment that value to 0001
- Normally the latches should be disabled, to prevent unwanted data from being accidentally stored
  – In our example, the ALU can read the current latch contents, 0000, and compute their increment, 0001
  – But the new value cannot be stored back while the latch is disabled

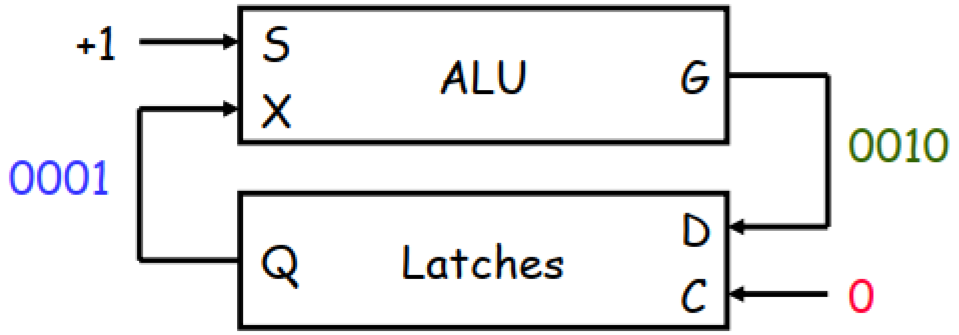


Timing issues in latches, cont.

- After the ALU has finished its increment operation, the latch can be enabled, and the updated value is stored.



- The latch must be quickly disabled again, before the ALU has a chance to read the new value 0001 and produce a new result 0010.



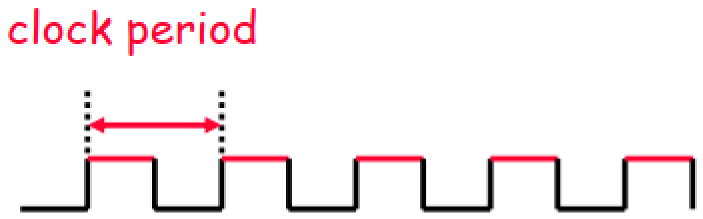
Two main problems

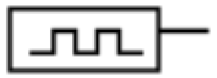
- To use latches correctly within a circuit, we must:
  – Keep the latches disabled until new values are ready to be stored
  – Enable the latches just long enough for the update to occur
- There are two main issues we need to address:
  – How do we know exactly when the new values are ready?
  We'll add another signal to our circuit. When this new signal becomes 1, the latches will know that the ALU computation has completed and data is ready to be stored

  – How can we enable and then quickly disable the latches?
  This can be done by combining latches together in a special way, to form what are called flip-flops

Clocks and timing control

- A clock is a special signal whose output continuously alternates between 0 and 1.

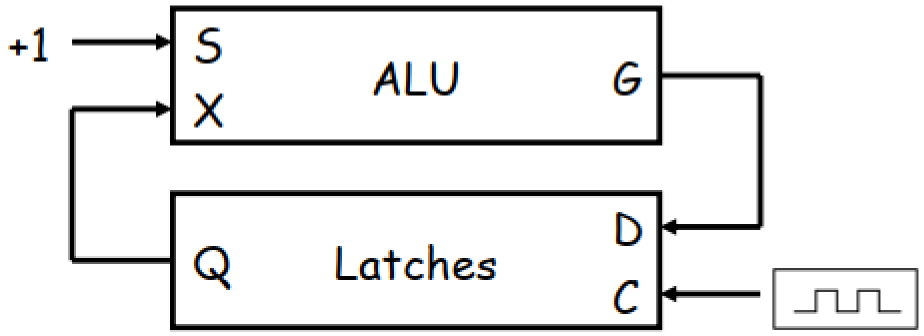




- The time it takes the clock to change from 1 to 0 and back to 1 is called the clock period, or clock cycle time
- The clock frequency is the inverse of the clock period. The unit of measurement for frequency is the hertz
- Clocks are often used to synchronize circuits
  – They generate a repeating, predictable pattern of 0s and 1s that can trigger certain events in a circuit, such as writing to a latch
  – If several circuits share a common clock signal, they can coordinate their actions with respect to one another
- This is like how humans use real clocks for synchronization.

Synchronization in our example

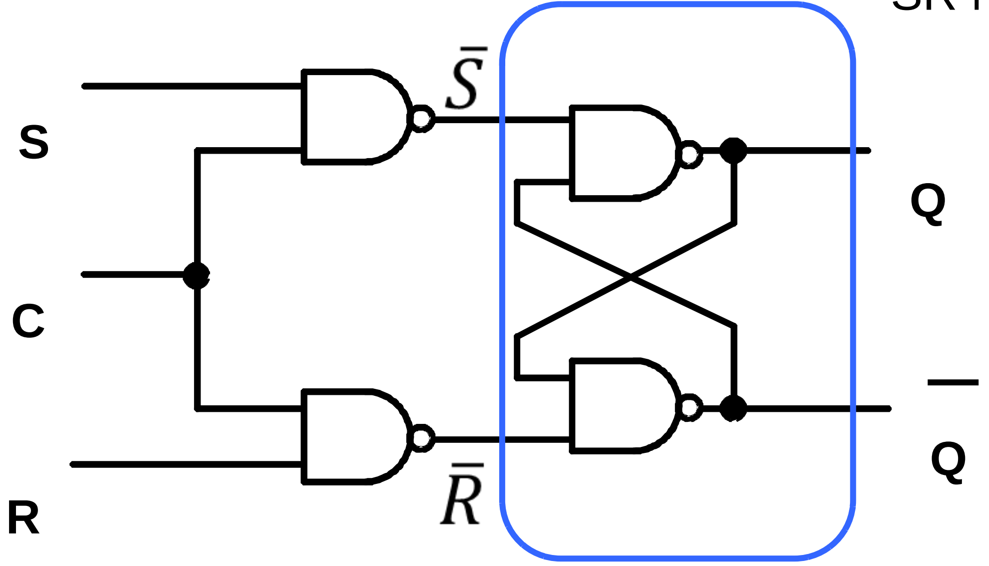
- We can use a clock to synchronize our latches with the ALU
  – The clock signal is connected to the latch control input C
  – The clock controls the latches. When it becomes 1, the latches will be enabled for writing



- The clock period must be set appropriately for the ALU
  – It should not be too short. Otherwise, the latches will start writing before the ALU operation has finished
  – The faster the ALU runs, the shorter the clock period can be
- The second issue was how to enable a latch for just an instant

SR flip-flop

- C is the control input, clock

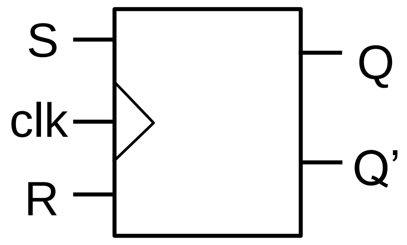


S'	R'	Q	Q'
0	0	Not Used	
0	1	1	0
1	0	0	1
1	1	Memory	

$$\bar{S} = \overline{(S.CLK)} = \bar{S} + \overline{CLK}$$
$$\bar{R} = \overline{(R.CLK)} = \bar{R} + \overline{CLK}$$

CLK	S	R	Q	Q'
0	X	X	Memory	
1	0	0	Memory	
1	0	1	0	1
1	1	0	1	0
1	1	1	Not Used	

Truth-table



Edge triggered flip-flop

Characteristic and excitation table for SR FF

Truth table:

Next state

CLK	S	R	Q_{n+1}
0	X	X	Q_n
1	0	0	Q_n
1	0	1	0
1	1	0	1
1	1	1	Not Used

Present state

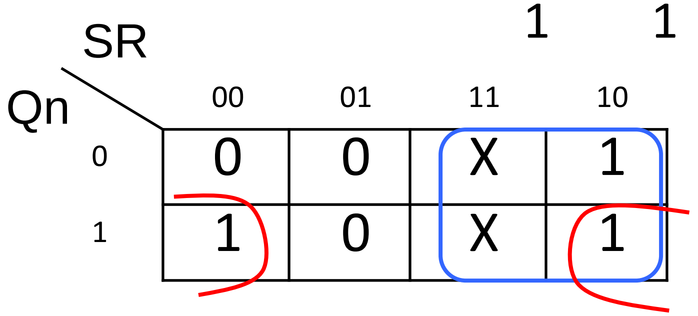
- An excitation table shows the minimum inputs that are necessary to generate a particular next state when the current state is known.
- The current state and next state are next to each other on the left side, the inputs needed to make that state change happen are shown on the right side.

Characteristic table (clk = 1)

Q_n	S	R	Q_{n+1}
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	X
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	X

Excitation table

Q_n	Q_{n+1}	S	R
0	0	0	X
0	1	1	0
1	0	0	1
1	1	X	0



Qn \ SR	00	01	11	10
0	0	0	X	1
1	1	0	X	1

$Q_{n+1}=S+Q_n.R'$

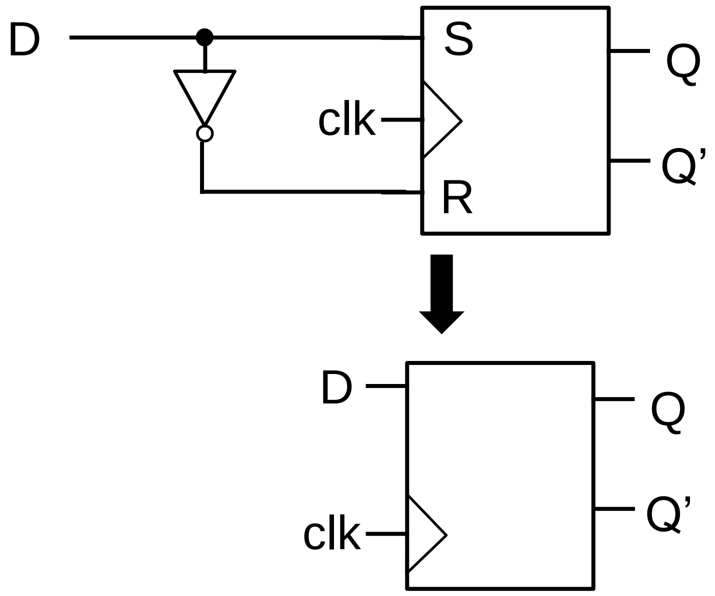
Characteristic equation

D type Flip-flop

Truth table for SR flip-flop

CLK	S	R	Q_{n+1}
0	X	X	Q_n
1	0	0	Q_n
1	0	1	0
1	1	0	1
1	1	1	Not Used

S and R are complement to each other



Truth table for D flip-flop

CLK	D	Q_{n+1}
0	X	Q_n
1	0	0
1	1	1

S = 1 and R = 1, S = 1 and R = 1 combinations are not possible

Characteristic table

Q_n	D	Q_{n+1}
0	0	0
0	1	1
1	0	0
1	1	1

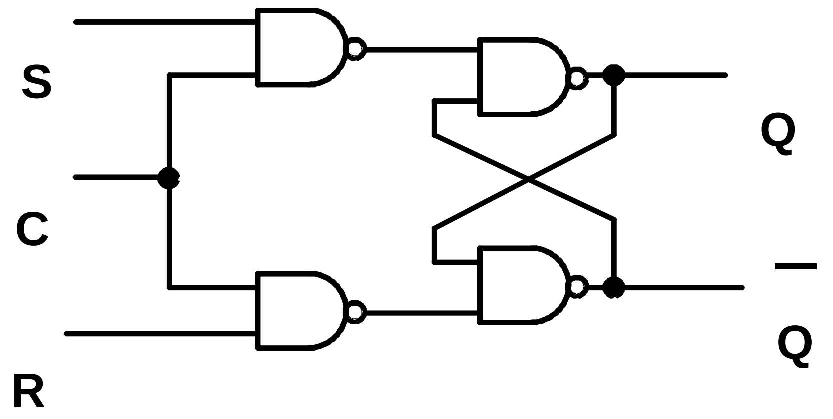
$Q_{n+1}=D$

Excitation table

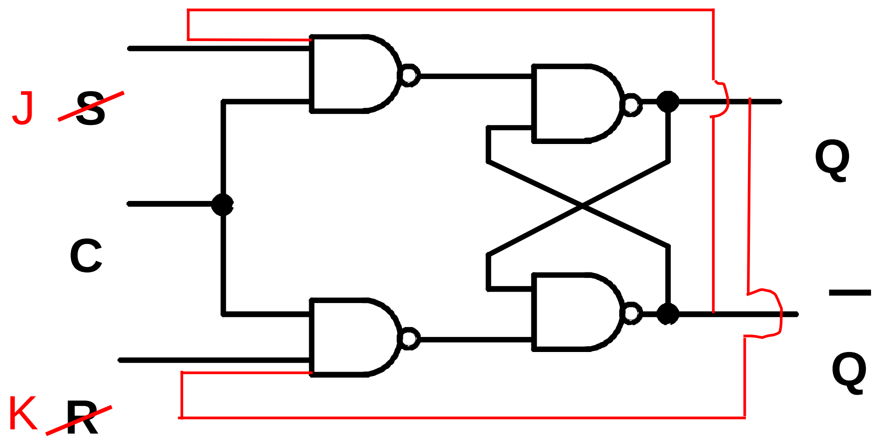
Q_n	Q_{n+1}	D
0	0	0
0	1	1
1	0	0
1	1	1

JK flip-flop

- The advantage of JK flip-flop: the last combination (S = 1, R = 1) is not used in SR flip-flop. With some modifications applied to SR flip-flop, this disadvantage can be overcome.



CLK	S	R	Q_{n+1}
0	X	X	Q_n (Memory)
1	0	0	Q_n (Memory)
1	0	1	0
1	1	0	1
1	1	1	Not Used



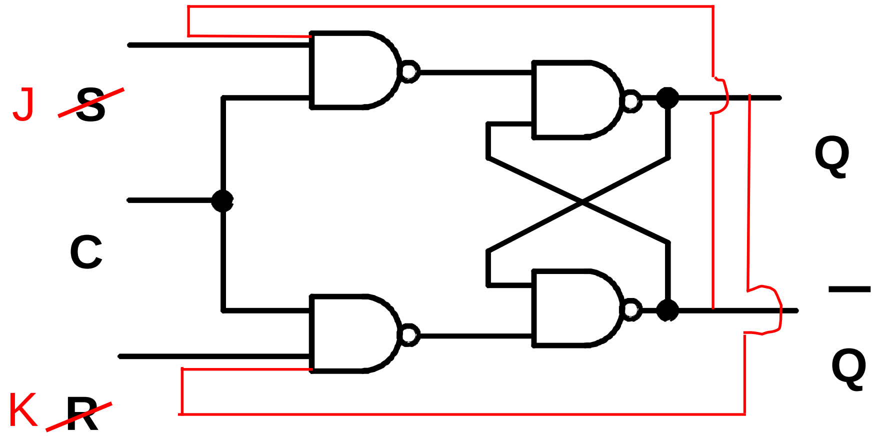
S'	R'	Q	Q'
0	0	Not Used	
0	1	1	0
1	0	0	1
1	1	Memory	

JK flip-flop cont.

- When CLK = 0, the output of first layer NANDs will be ‘1’ and we will have memory state.
- When CLK = 1, J = 1, K = 0 then Q = 1 and Q’ = 0
- When CLK = 1, J = 0, K = 1 then Q = 0 and Q’ = 1
- When CLK = 1, J = 1, K = 1, assume Q = 0 and Q’ = 1 then we will have

$$\begin{tikzcd}[row sep=small] & \text{Q = 0, 1, 0, 1, …} \\ \text{Racing condition} \arrow[ur] \arrow[dr] & \\ & \text{Q’ = 1, 0, 1, 0, …} \end{tikzcd}$$

$$Q_{n+1} = \overline{Q_n}$$

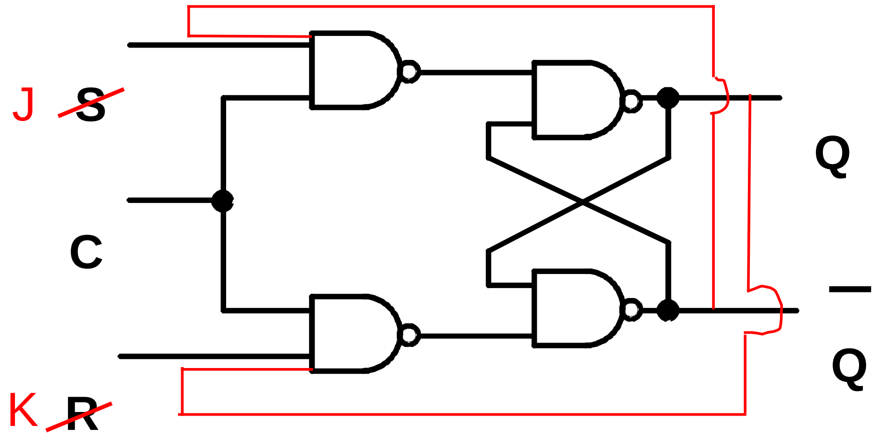


S'	R'	Q	Q'
0	0	Not Used	
0	1	1	0
1	0	0	1
1	1	Memory	

JK flip-flop cont.

- When CLK = 0, the output of first layer NANDs will be ‘1’ and we will have memory state.
- When CLK = 1, J = 1, K = 0 then Q = 1 and Q’ = 0
- When CLK = 1, J = 0, K = 1 then Q = 0 and Q’ = 1
- When CLK = 1, J = 1, K = 1, assume Q = 0 and Q’ = 1 then we will have

$$\text{Racing condition} \begin{cases} Q = 0, 1, 0, 1, \ldots \\ Q' = 1, 0, 1, 0, \ldots \end{cases} \qquad Q_{n+1} = \overline{Q_n}$$

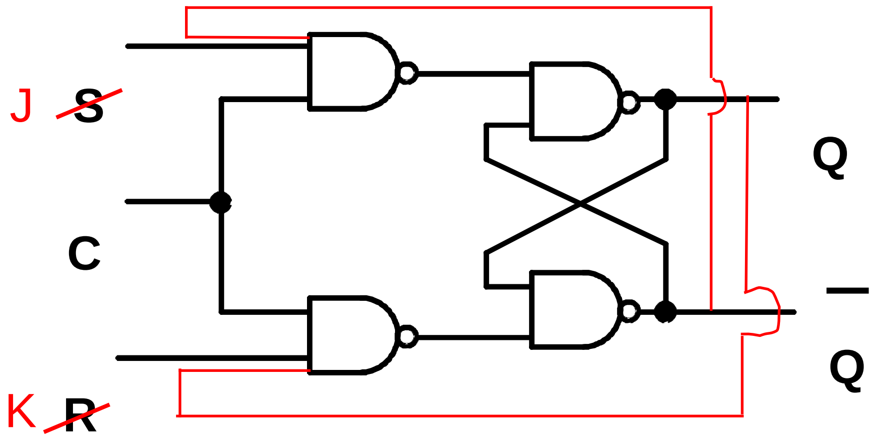


CLK	J	K	Q_{n+1}
0	X	X	Q_n
1	0	0	Q_n
1	0	1	0
1	1	0	1
1	1	1	$(Q_n)'$

Characteristic and excitation table for JK FF

Truth Table

CLK	J	K	Q_{n+1}
0	X	X	Q_n
1	0	0	Q_n
1	0	1	0
1	1	0	1
1	1	1	$(Q_n)'$



Characteristic table

Q_n	J	K	Q_{n+1}
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

Excitation table

Q_n	Q_{n+1}	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

Characteristic and excitation table for JK FF cont.

Characteristic table

Q_n	J	K	Q_{n+1}
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

$$\begin{array}{c|cccc} Q_n \backslash JK & 00 & 01 & 11 & 10 \\ \hline 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 \end{array}$$

$Q_{n+1}=Q_n.K'+(Q_n)'.J$

Characteristic equation

Excitation table

Q_n	Q_{n+1}	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

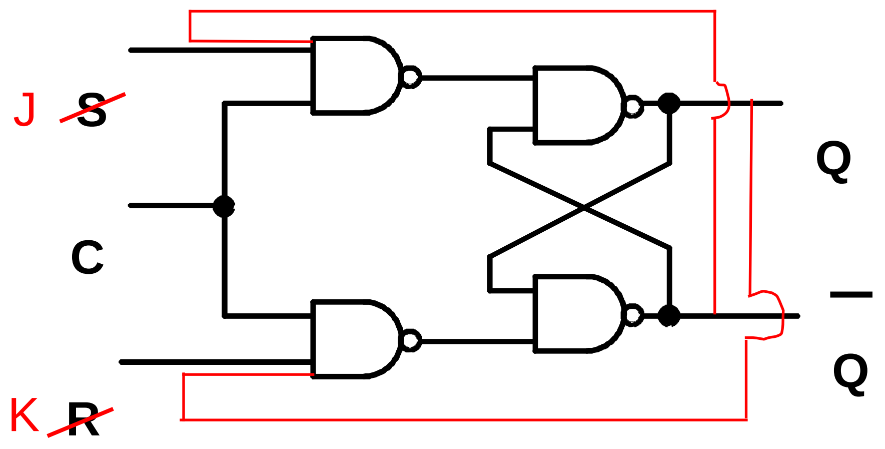
$$\begin{array}{c|cc} Q_n \backslash Q_{n+1} & 0 & 1 \\ \hline 0 & 0 & 1 \\ 1 & X & X \end{array}$$

$J = Q_{n+1}$

$$\begin{array}{c|cc} Q_n \backslash Q_{n+1} & 0 & 1 \\ \hline 0 & X & X \\ 1 & 1 & 0 \end{array}$$

$K = (Q_{n+1})'$

Race around condition



CLK	J	K	Q_{n+1}
0	X	X	Q_n
1	0	0	Q_n
1	0	1	0
1	1	0	1
1	1	1	$(Q_n)'$

- When CLK = 1, J = 1, K = 1, assume Q = 0 and Q’ = 1 then we will have

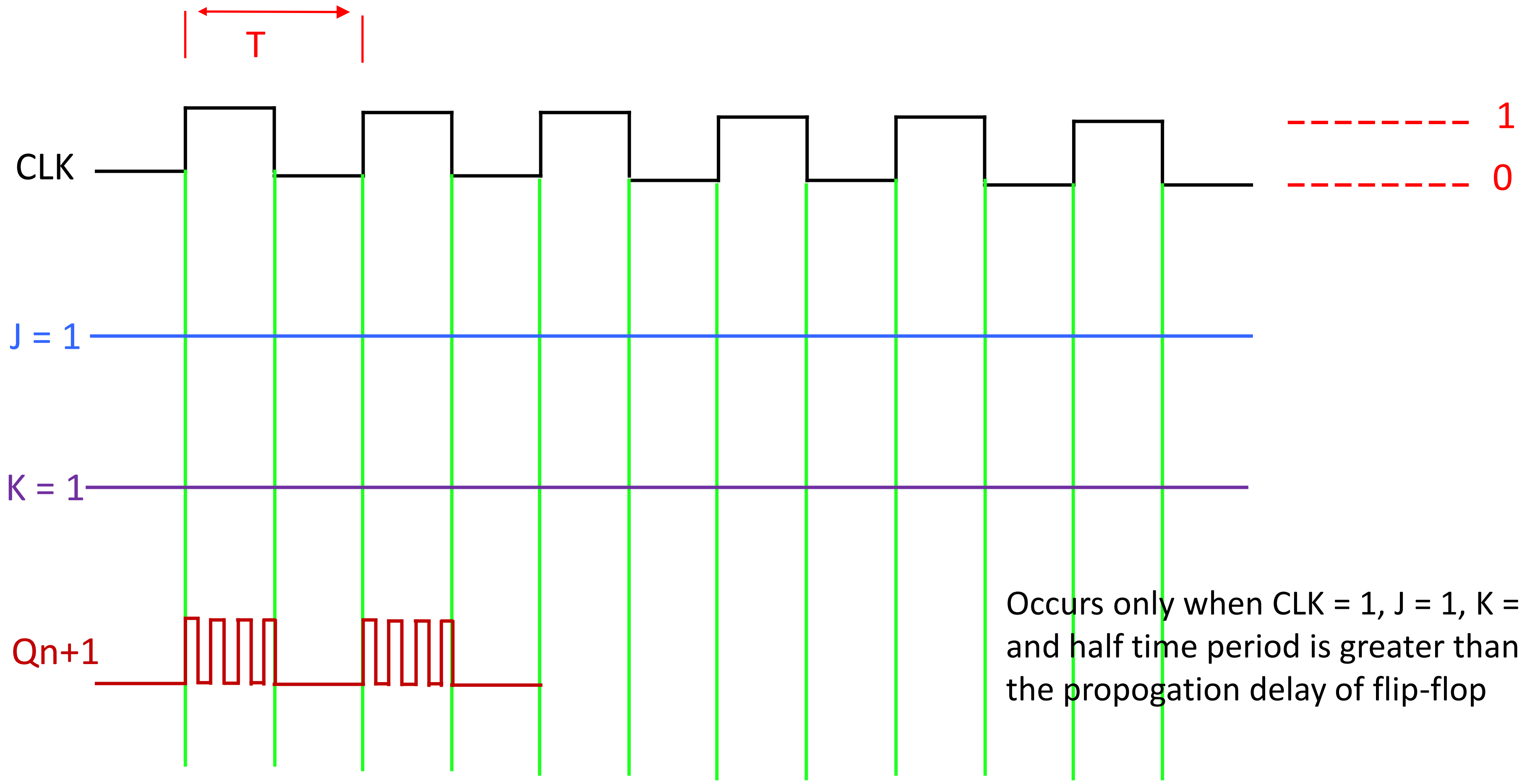
Racing condition → Q = 0, 1, 0, 1, …

Not controlled → Q’ = 1, 0, 1, 0, …

$$Q_{n+1} = \overline{Q_n}$$

- We need to convert this race-around (uncontrolled) condition to toggle (controlled) condition

Race around condition cont.

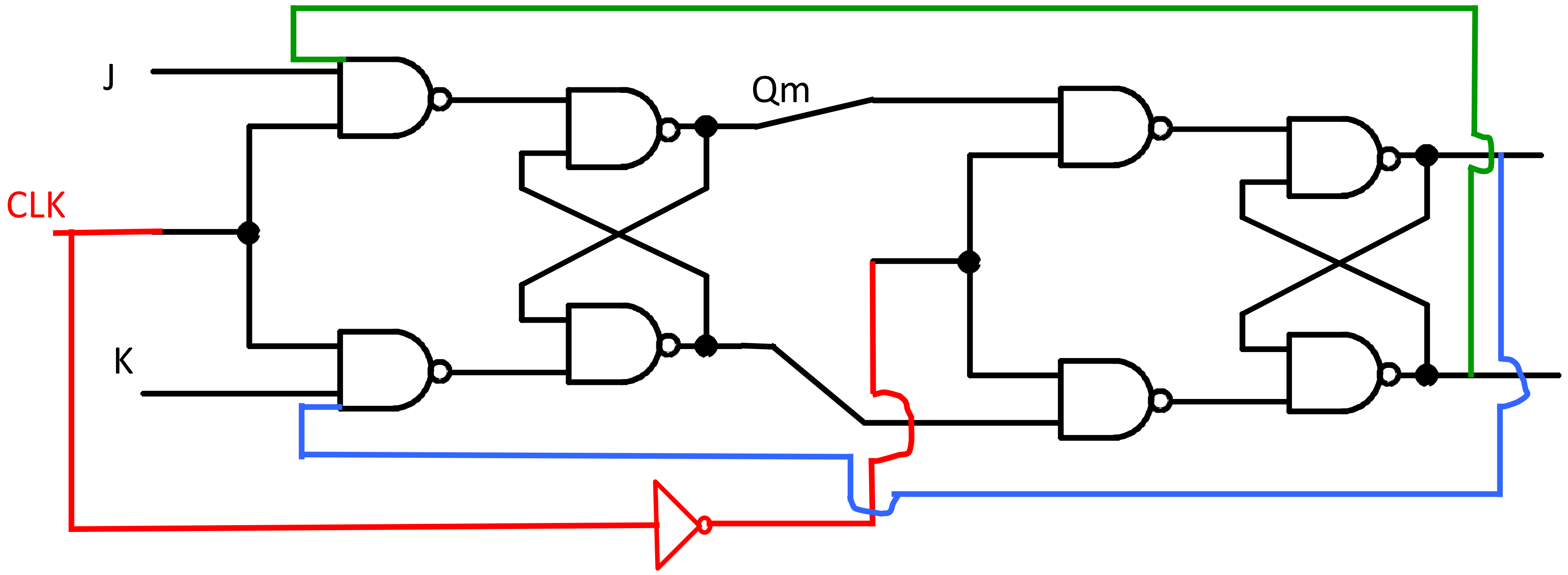


Occurs only when CLK = 1, J = 1, K = 1 and half time period is greater than the propogation delay of flip-flop

- Conditions to overcome racing:
  - T/2 < propagation delay of flip-flop
  - Edge triggering
  - Master-slave configuration

Master-Slave Operation of JK flip-flop

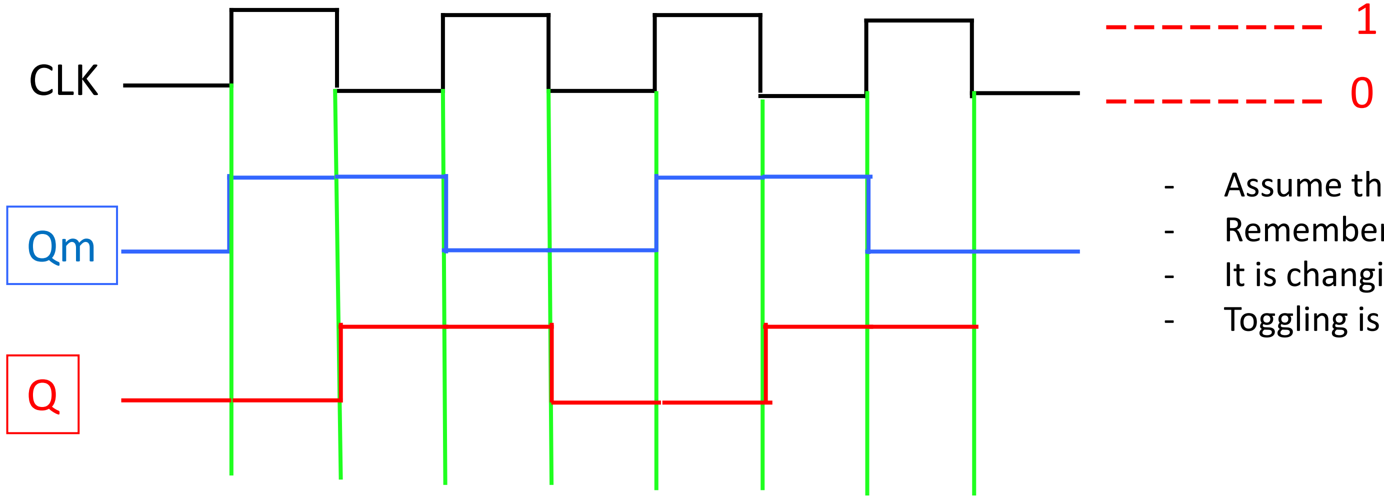
- Master-Slave operation is same as negative edge-triggering. (Equivalence to positive-edge triggering can be achieved by making changes in configuration.)
- We will add another stage to JK flip-flop



Master-Slave Operation of JK flip-flop cont

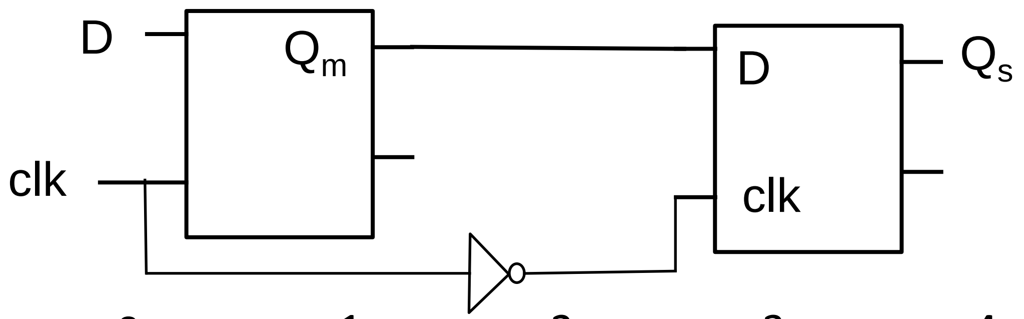
- When the clock is High, the master will be operational and the slave will work as a memory (keeps Q and Q' same)
- When the clock is Low, the master will be in memory state and the slave will be functional (changes Q and Q')
- Even we have feedback from slave to input side, the master will not be functional when feedback comes and this prevents racing
- In this configuration, the output changes once in a clock cycle. This is called as toggling.

when J = 1, K = 1

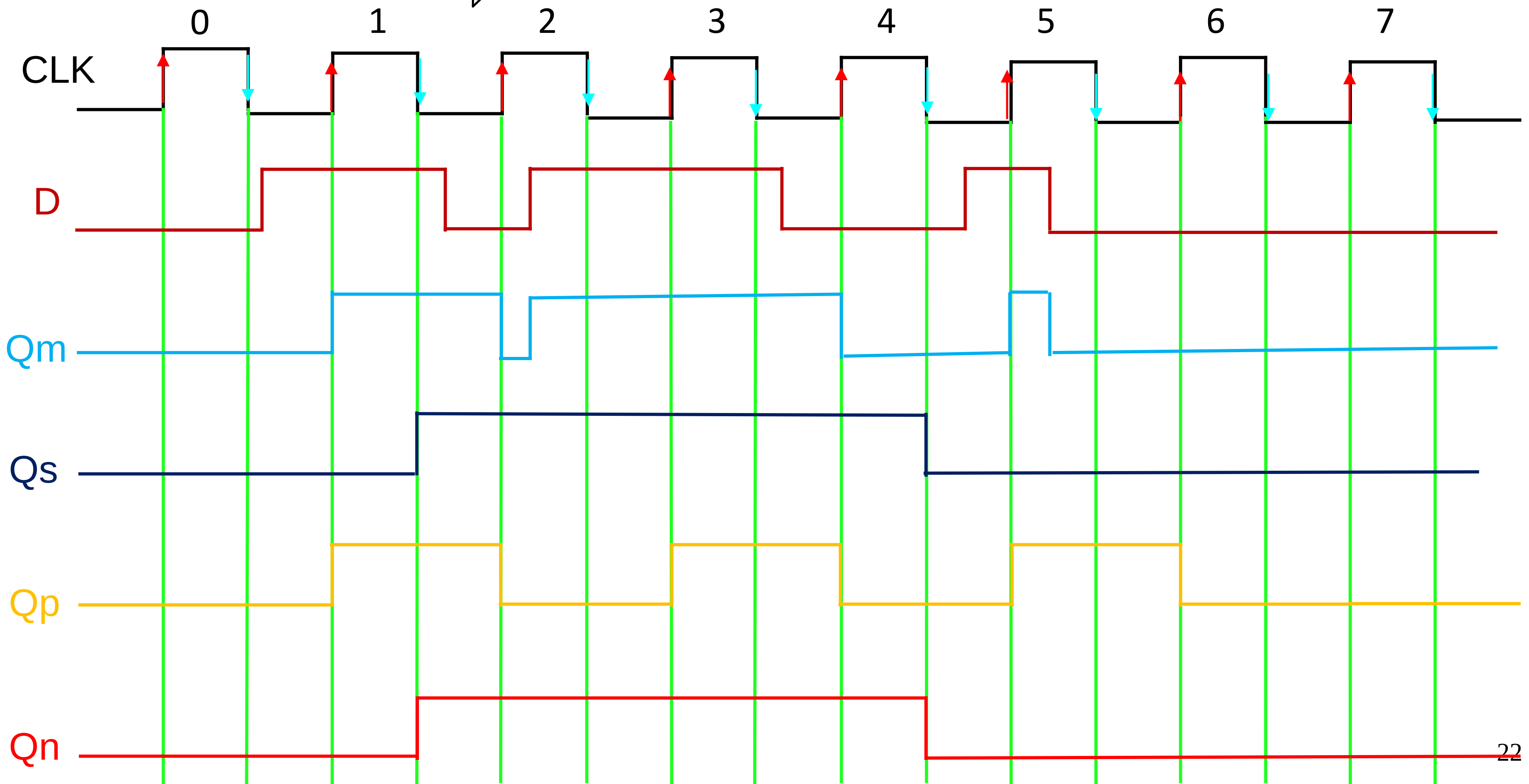


- Assume that Qm and Q is low previously
- Remember when clk = 0, the state is stored
- It is changing once in a cycle (toggling)
- Toggling is used in counters

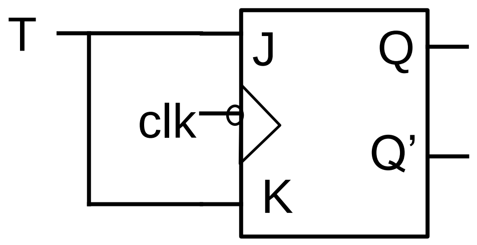
Master-Slave Operation of D flip-flop



CLK	D	Q_{n+1}
0	X	Q_n
1	0	0
1	1	1



Characteristic and excitation table for T FF



Truth Table

CLK	T	Q_{n+1}
0	X	Q_n
1	0	Q_n
1	1	$(Q_n)'$

$Q_n \rightarrow$ Memory

$(Q_n)' \rightarrow$ Toggling

Characteristic table

Q_n	T	Q_{n+1}
0	0	0
0	1	1
1	0	1
1	1	0

Odd 1's detector, XOR

$$Q_{n+1} = Q_n \oplus T$$

Characteristic equation

Excitation table

Q_n	Q_{n+1}	T
0	0	0
0	1	1
1	0	1
1	1	0