Introduction to Electrical Engineering

Course Code: EE 103

Department: Electrical Engineering

Instructor Name: B. G. Fernandes

E-mail id: bgf@ee.iitb.ac.in

Review : K- MAP

- In K-Map, squares are labelled so that horizontally adjacent squares differ only in one variable.

Eg: Upper left-hand square in the 4 – variable map is $\bar{A}\,\bar{B}\bar{C}\,\bar{D}$, while the square immediately to its right is $\bar{A}\,\bar{B}\bar{C}D$.
Similarly vertically adjacent squares differ only in one variable.
Square directly below $\bar{A}\,\bar{B}\bar{C}\,\bar{D}$ is $\bar{A}B\bar{C}\,\bar{D}$.

CD / AB	00	01	11	10
00				
01				
11				
10				

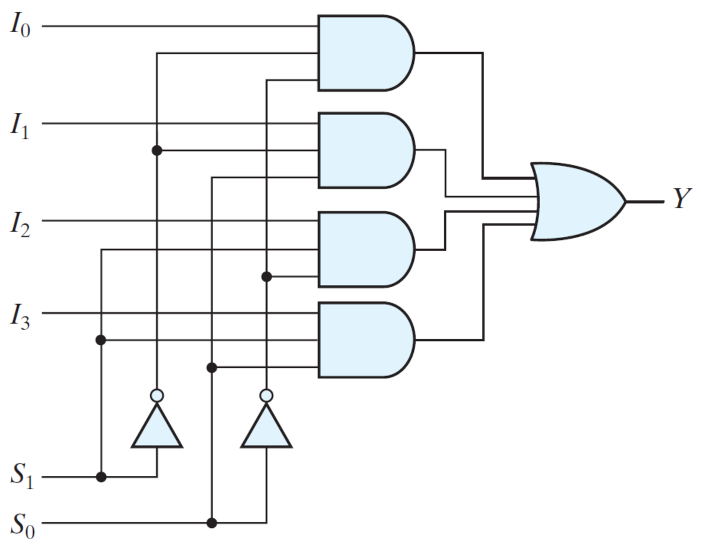
- Note that each square in the top row is considered to be adjacent to a corresponding square in the bottom row.

Eg: $\bar{A}\,\bar{B}CD$ square in the top row is adjacent to $A\bar{B}CD$ square in the bottom row.
Therefore In order for vertically and horizontally adjacent squares to differ in only one variable it must be
$\bar{A}\bar{B}$ $\bar{A}B$ AB $A\bar{B}$ and $\bar{C}\bar{D}$ $\bar{C}D$ CD $C\bar{D}$

	$\bar{C}\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$
$\bar{A}\bar{B}$				
$\bar{A}B$				
AB				
$A\bar{B}$				

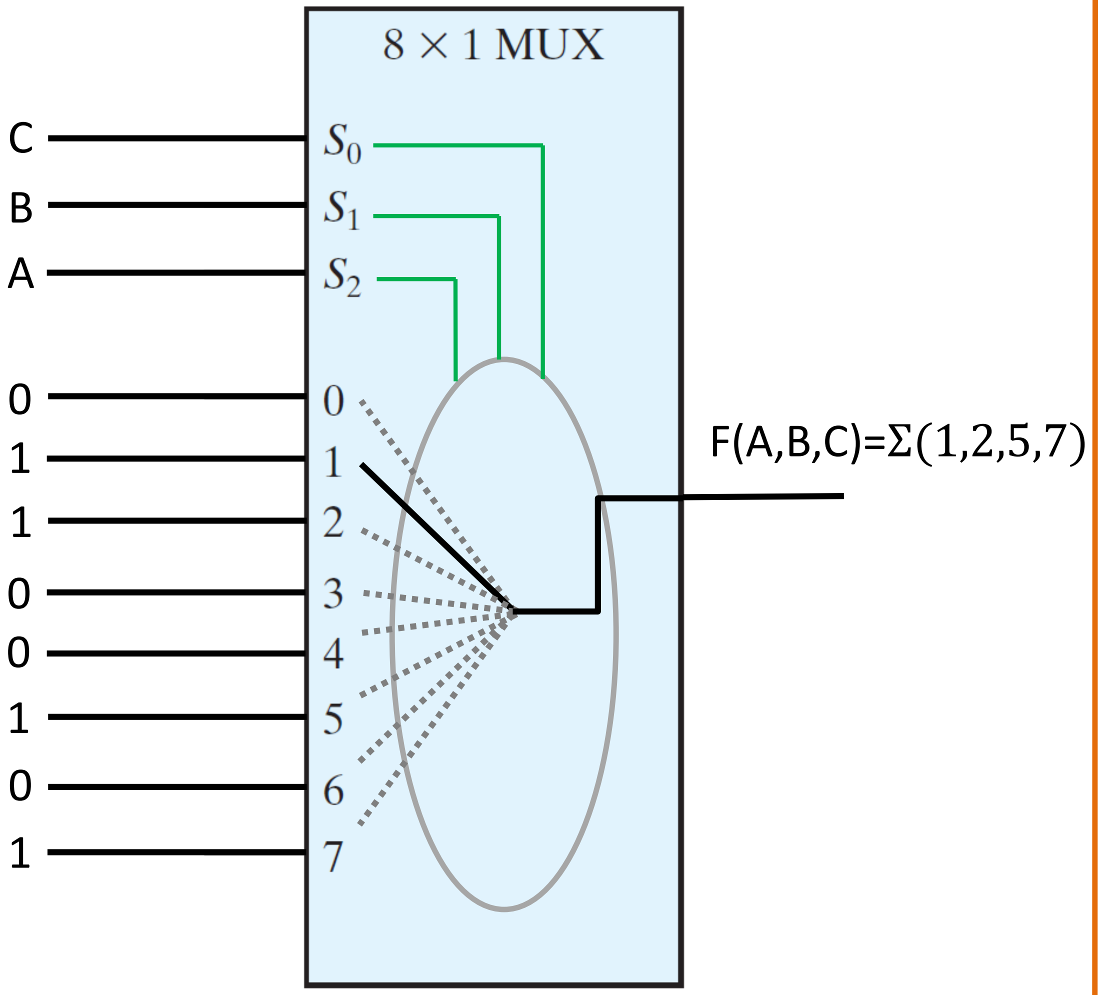
Review : MULTIPLEXER

- Logic circuit accepts several data inputs. But allows only one of them at a time to get through the o/p.
- Routing of the desired input to the o/p is done by 'select' inputs or 'address' inputs.



S_1	S_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

Four-to-one-line multiplexer



Implementation of Boolean expressions using Multiplexers (MUX)

- In a practical circuit, the number of ICs should be minimized; K-map solution requires a combination of gates, with differing number of inputs - which in most cases will not result in the minimum number of ICs.

Another Solution:

- A 4-to-1 MUX can directly implement the Truth Table of a TWO variable function using its TWO selection inputs and Input lines.
- Similarly, an 8-to-1 MUX can directly implement the Truth Table of a THREE variable function using its THREE selection inputs;
- A 16-to-1 MUX can implement the Truth table of a FOUR variable function using its FOUR selection inputs.
- It is more efficient to implement a Boolean function of *n* variables using a MUX that has *(n-1)* selection inputs.

Multiplexer: Example 1

There are 3 input variables and we are using two of them to get a output regardless of the value of third variable.

Implement using 4-to-1 multiplexer: $F(x, y, z) = \sum(1, 2, 6, 7)$

Address lines: x, y

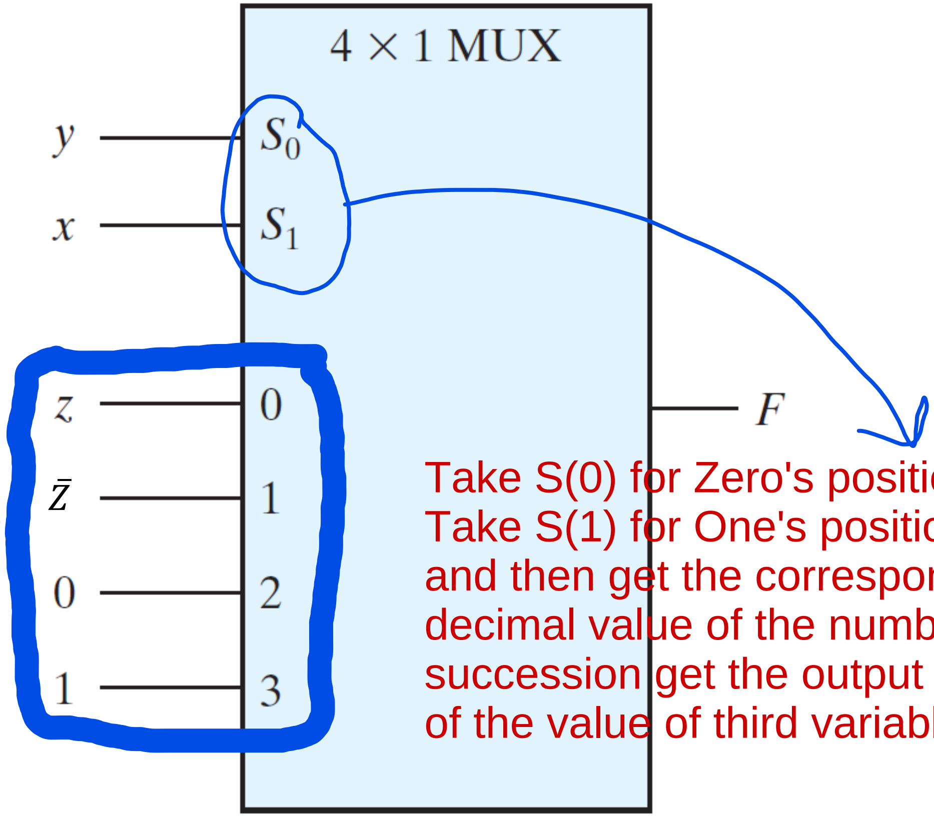
x	y	z	F	
0	0	0	0	$F = z$
0	0	1	1	
0	1	0	1	$F = \bar{z}$
0	1	1	0	
1	0	0	0	$F = 0$
1	0	1	0	
1	1	0	1	$F = 1$
1	1	1	1	

Truth table

Logic :

If xy = 0, F = z
If xy = 1, F = z bar
If xy = 2, F = 0
If xy = 3, F =1

Might have to remember the output corresponding to particular decimal number.



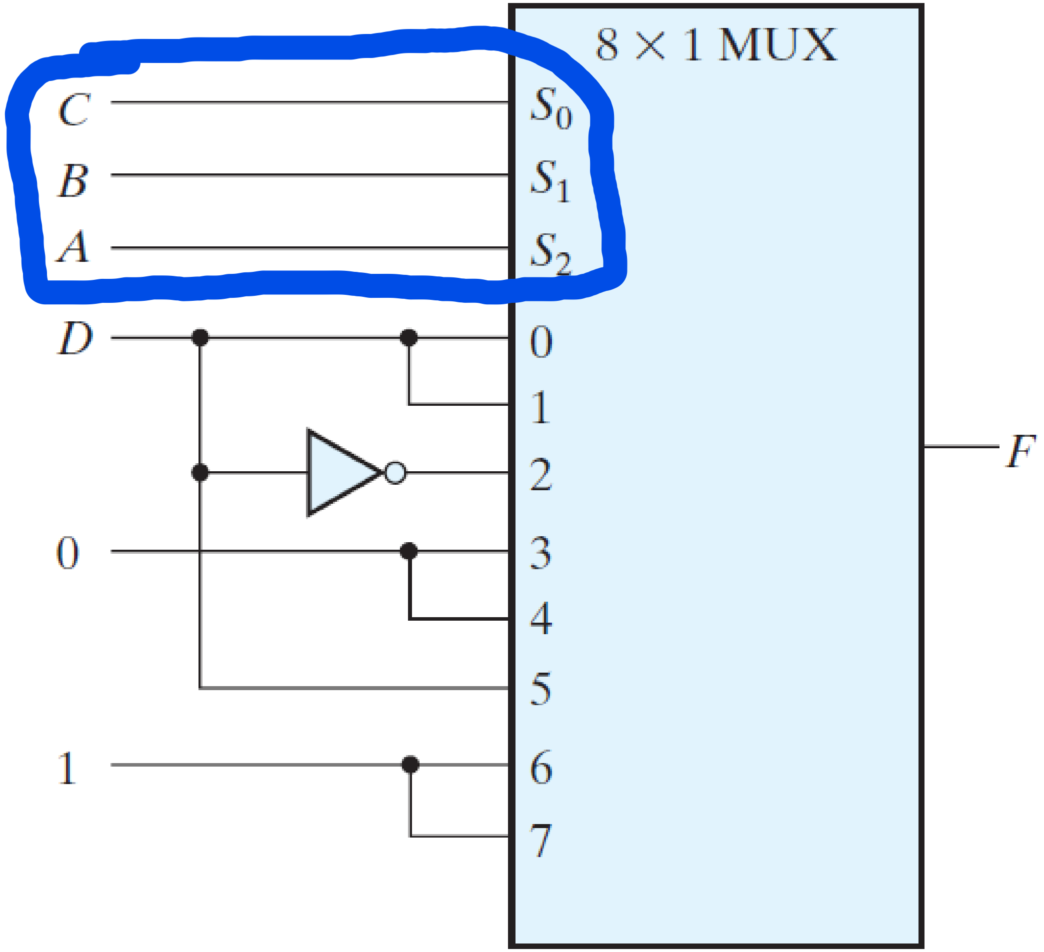
Take S(0) for Zero's position
Take S(1) for One's position
and then get the corresponding decimal value of the number and in succession get the output irrelevant of the value of third variable.

Multiplexer implementation

Multiplexer: Example 2

Implement using 8-to-1 multiplexer: $F(A,B,C,D) = \sum(1,3,4,11,12,13,14,15)$

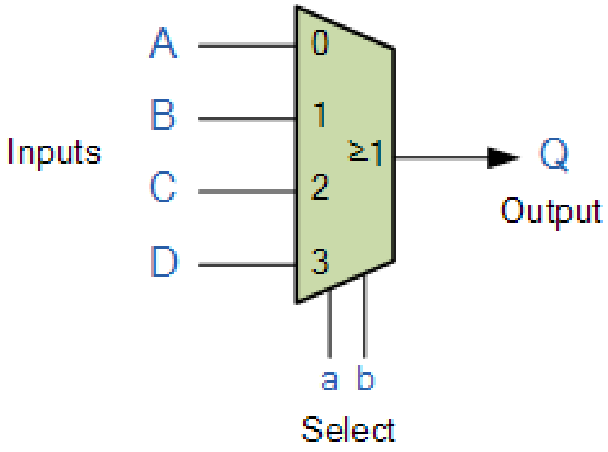
$\underbrace{A \quad B \quad C}_{\text{Address lines}}$			D		F
0	0	0	0	0	$F = D$
0	0	0	1	1	
0	0	1	0	0	$F = D$
0	0	1	1	1	
0	1	0	0	1	$F = \overline{D}$
0	1	0	1	0	
0	1	1	0	0	$F = 0$
0	1	1	1	0	
1	0	0	0	0	$F = 0$
1	0	0	1	0	
1	0	1	0	0	$F = D$
1	0	1	1	1	
1	1	0	0	1	$F = 1$
1	1	0	1	1	
1	1	1	0	1	$F = 1$
1	1	1	1	1	



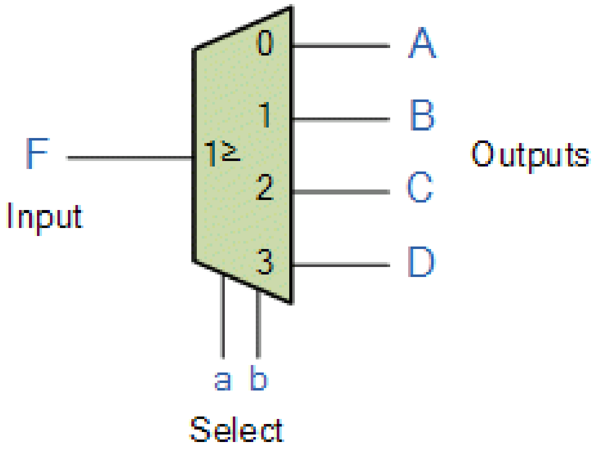
Multiplexer vs Demultiplexer

Multiplexer: takes several inputs and transmits one of them to the output.

Demultiplexer (DeMUX): takes single input and distributes it over several outputs.



4:1 MUX

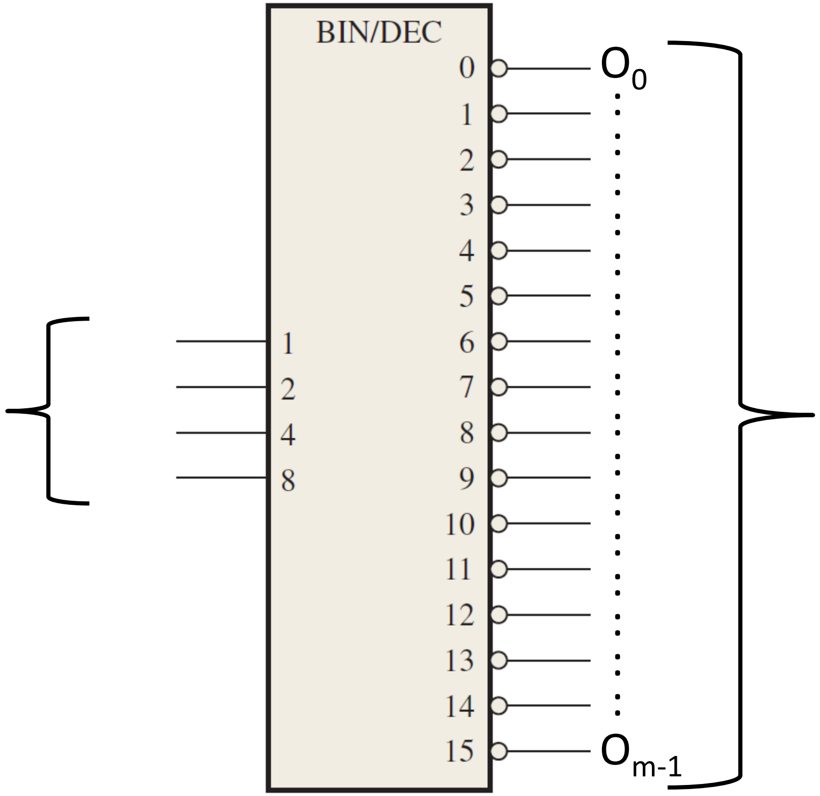


1:4 DeMUX

Decoder

A logic circuit that converts an N-bit binary input code into M output lines
Each output line is activated for only one of the possible combinations of Inputs

N-Inputs



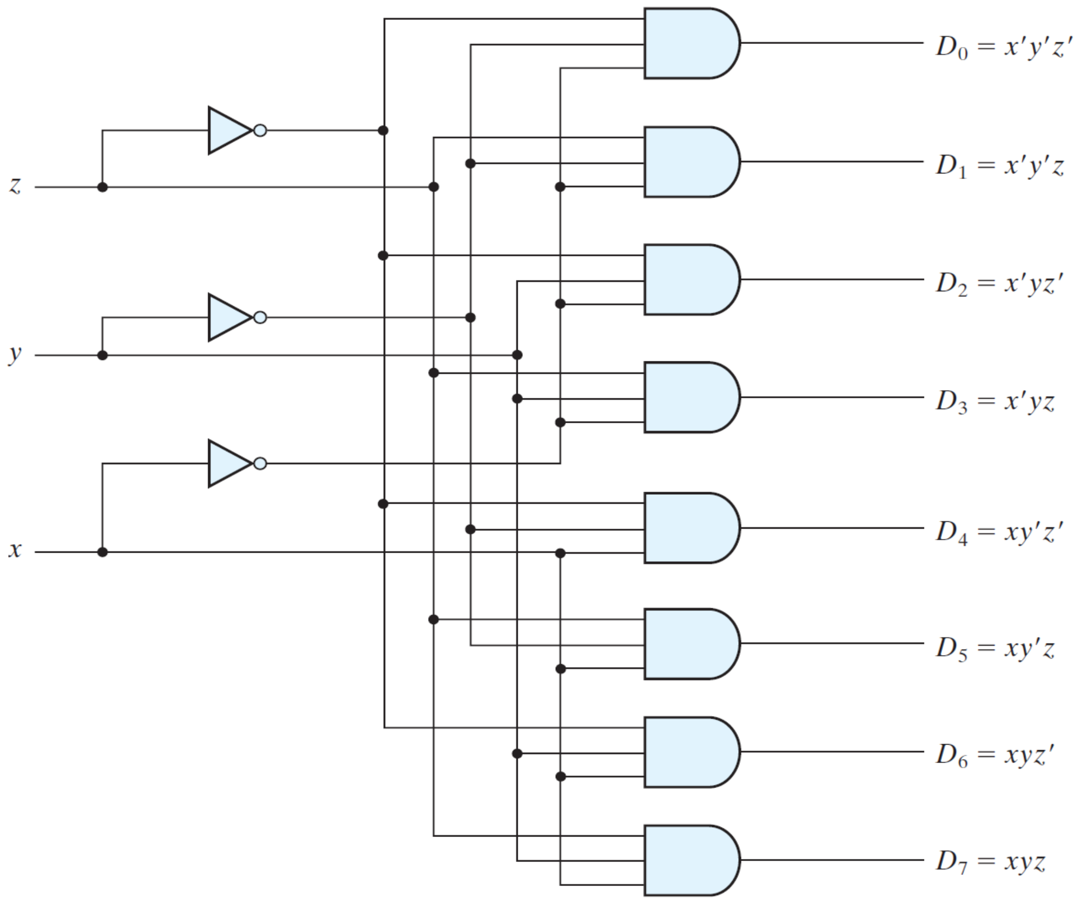
M-Outputs

Only one of the output is
High for each input condition.

$M \leq 2^N$

3 to 8 line Decoder → 3 input and 8 output lines or

1-of-8 Decoder → Only 1 of the 8 outputs is activated



Inputs			Outputs							
x	y	z	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
0	0	0	1							
0	0	1		1						
0	1	0			1					
0	1	1				1				
1	0	0					1			
1	0	1						1		
1	1	0							1	
1	1	1								1

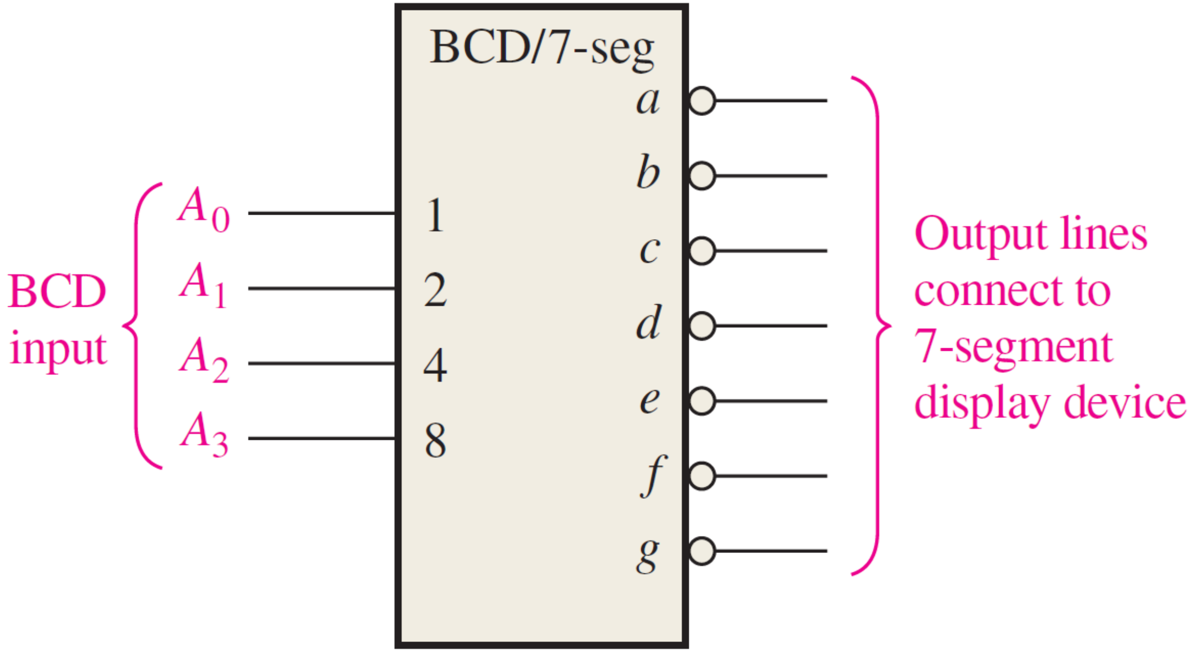
IC - 74LS138 is a popular 3-to8 decoder

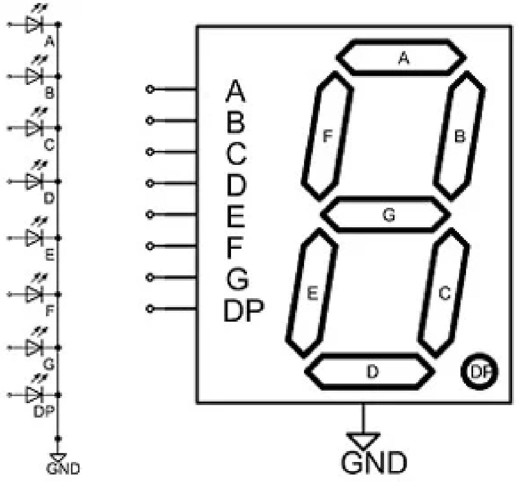
BCD to 7 Segment Decoder

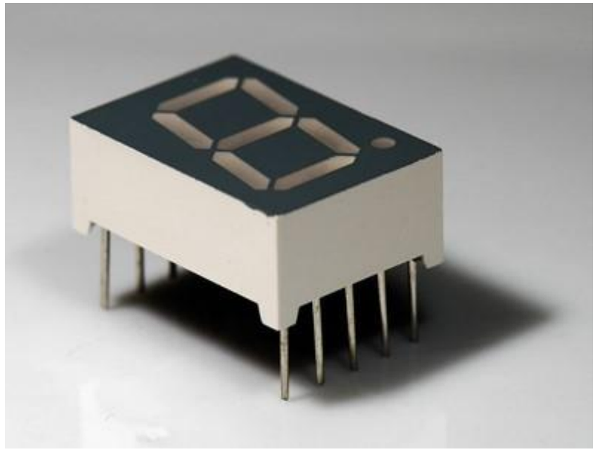
Binary Coded Decimal

4 bit BCD input.

According to the input it provides the output that pass current through the appropriate segments







Sequential Circuits

- Combinatorial circuits: Output at any instant of time are dependent on the input levels present at that time.

  (Eg: Gates, MUX, Decoder)

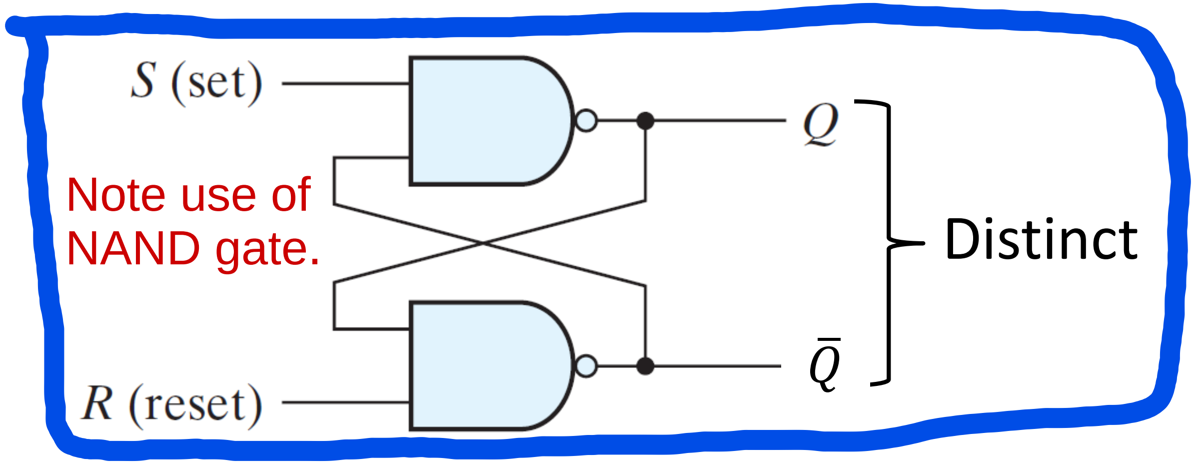
  Prior input conditions have no effect on the present output

  ⇒ No memory

- Sequential circuits: Output depends on the present set of inputs and <u>also</u> on the past output (Eg: Latches, Flip-flops, Digital Clocks)

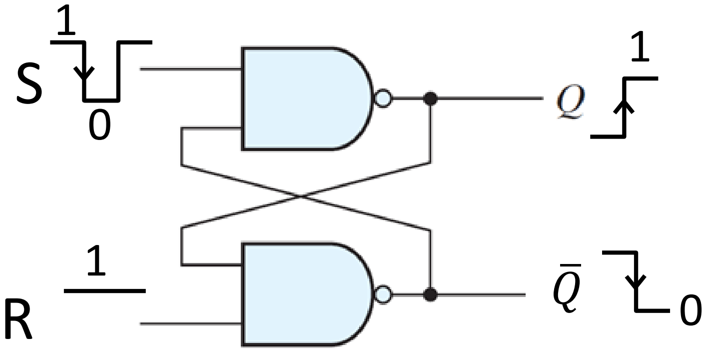
S-R Latch

An electronic device that changes its output immediately on the basis of the applied input. One can use it to store either 0 or 1 at a specified time

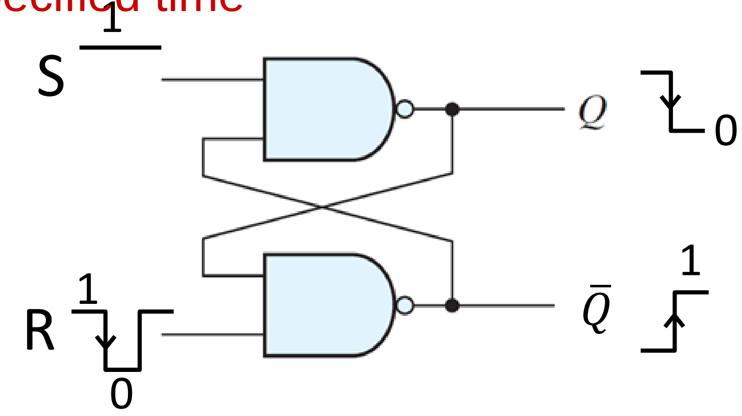


Normal operation S=R=1; Assume $Q = 0 \ \therefore \bar{Q} = 1$

($Q = 1$; $\bar{Q} = 0$ is also possible)



If $Q = 1$; $\bar{Q} = 0$,No change in output



Instead if $Q = 1$, $\bar{Q} = 0$

Q Becomes 0 and $\bar{Q} = 1$

Making S=R=0 is not allowed

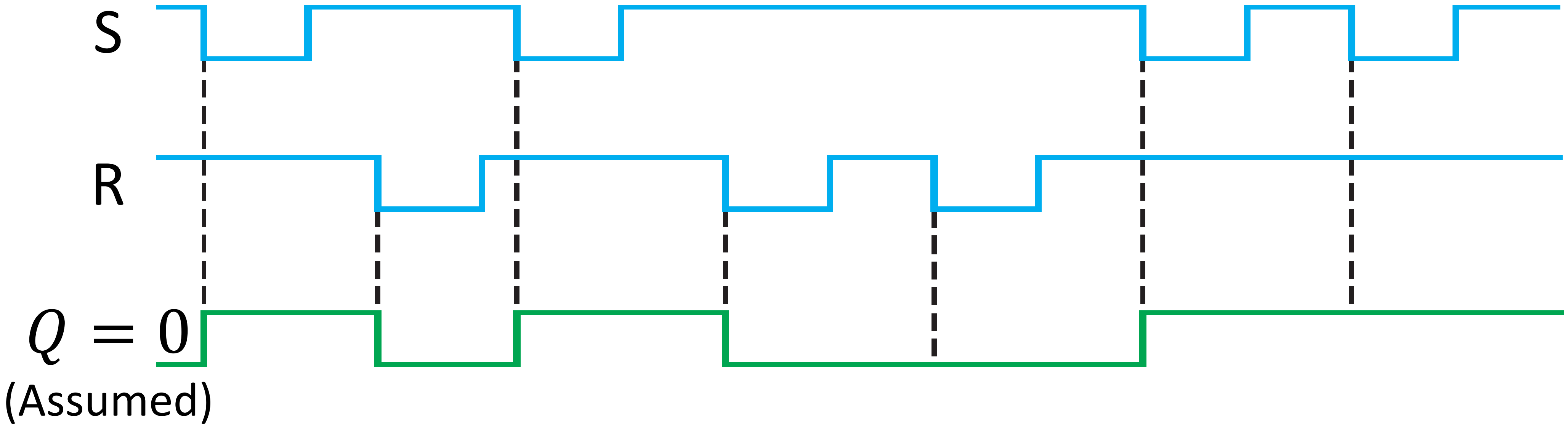
$\Rightarrow Q = \bar{Q} = 1$

Truth table of S-R Latch:

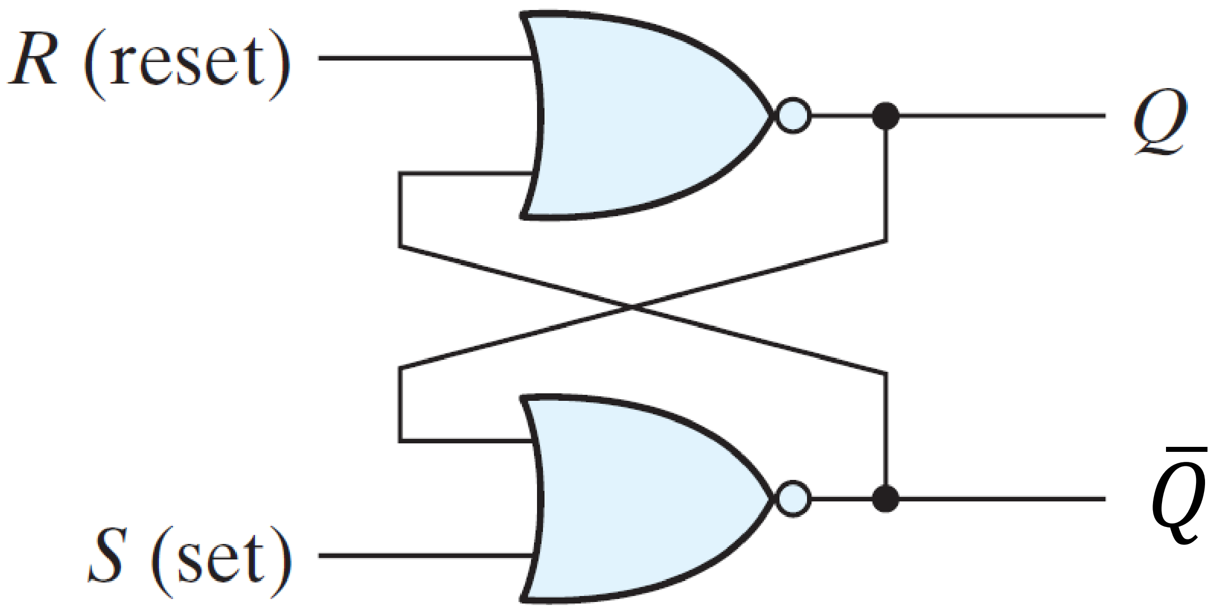
S	R	output
1	1	No change (Hold)
0	1	Q =1
1	0	Q =0
0	0	invalid

The output remains the same as of the previous one.

The output changes only when the value of one of the variables changes from 1 to 0.



Instead of NAND latch we can have NOR latch also.
Replace NAND gate by NOR gate



Truth table of NOR latch

S	R	output
0	0	No change (Hold)
1	0	Q =1
0	1	Q =0
1	1	invalid

Flip-Flop:

⇒ Made up of an assembly of logic gates.

⇒ Logic gates have no storage capacity.

⇒ Can be connected together in ways that permit information to be stored.

⇒ Bistable multivibrator latch

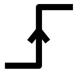
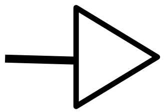
Digital systems can operate either asynchronously or synchronously.

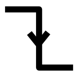
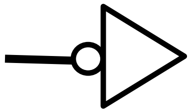
Asynchronous ⇒ Output can change any time. No common clock; output of previous flip flop is used as new clock; slower; more errors; aka Serial counter; easy to use.

⇒ Difficult to troubleshoot.

Synchronous Same clock is applied to each flip flop; faster; less errors; aka Parallel counter; hard to use.

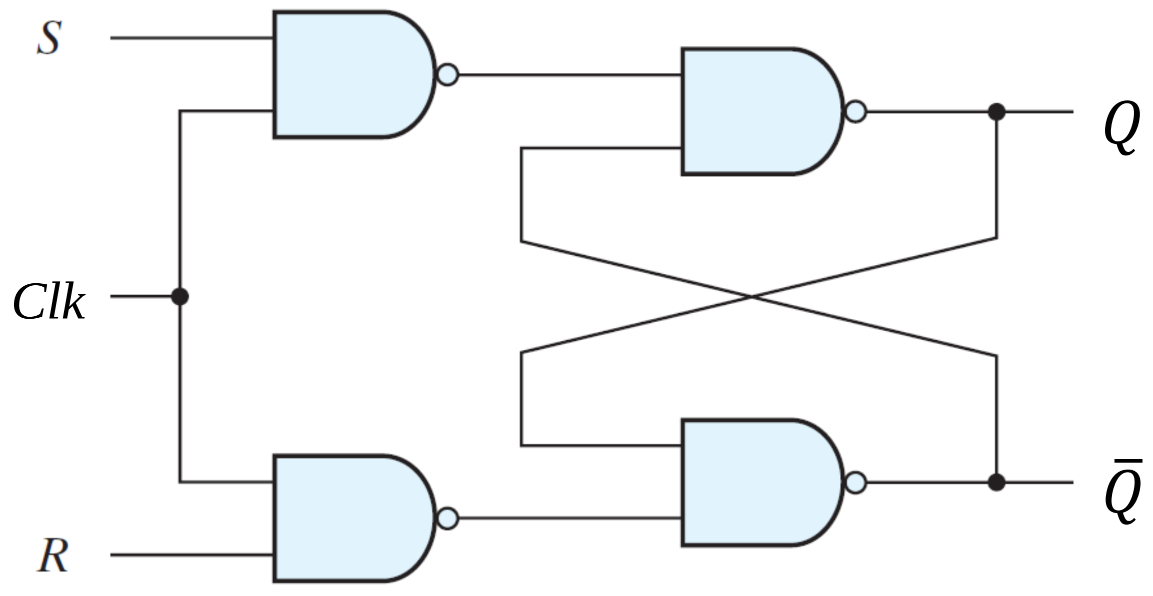
The exact time at which output can change is determined by a signal "clock"

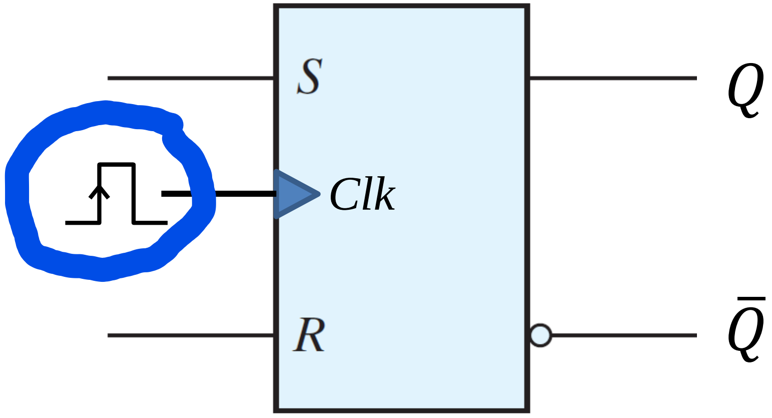
PET ⇒ Positive Edge Trigger  symbol 

NET ⇒ Negative Edge Trigger  symbol 

Clocked S-R Flip-Flop

S - R Latch with a clock.





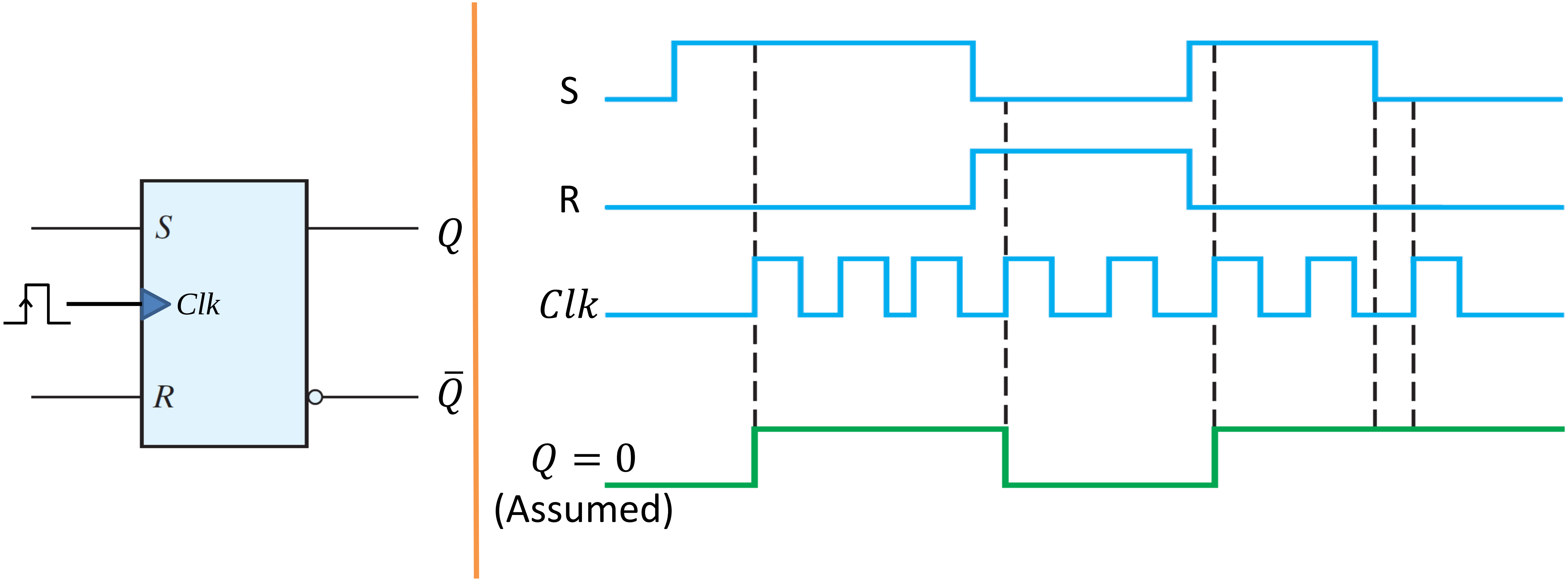
Truth table

= That of NOR Latch

S	R	Clk	Q
0	0	↑	Q(Hold)
1	0	↑	1
0	1	↑	0
1	1	↑	invalid

Depending upon the inputs,
output will change only at the ↑ of the clock

Clocked S-R Flip-Flop



Instead of PET ( ), if NET was used ( ) output changes at