CSE460: VLSI Design

Lecture 2

Review of digital logic design

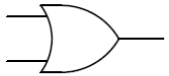
Background

- Logic gates (AND, OR, NOT, XOR, etc.)
- Boolean algebra
- Truth tables
- Logic functions
- Logic function synthesis by
  - Sum of Products (SOP)
  - Product of Sums (POS)
  - K-maps
- Logic blocks (MUX, DEMUX)
- Sequential elements (Latch, Flip-flop)

Logic gates

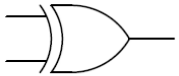
AND

A	B	Output
0	0	0
0	1	0
1	0	0
1	1	1



OR

A	B	Output
0	0	0
0	1	1
1	0	1
1	1	1



XOR

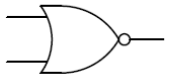
A	B	Output
0	0	0
0	1	1
1	0	1
1	1	0

NOT

Input	Output
0	1
1	0

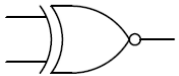
NAND

A	B	Output
0	0	1
0	1	1
1	0	1
1	1	0



NOR

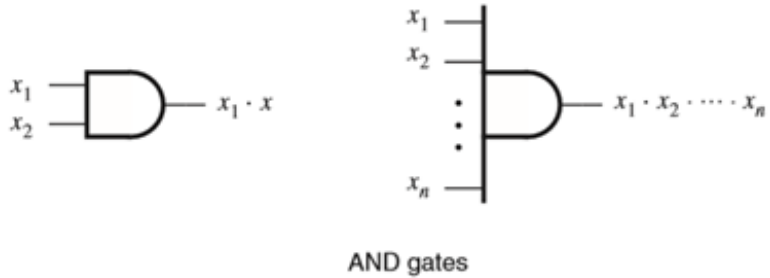
A	B	Output
0	0	1
0	1	0
1	0	0
1	1	0



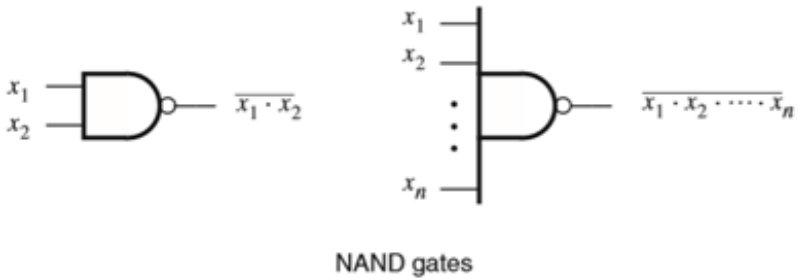
XNOR

A	B	Output
0	0	1
0	1	0
1	0	0
1	1	1

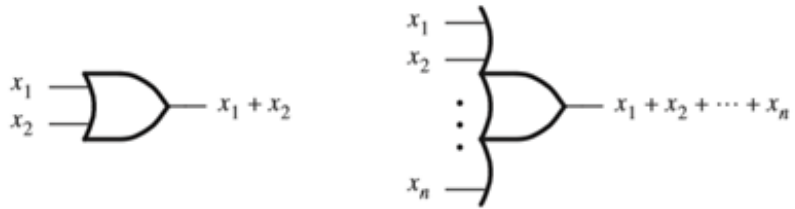
Generalized n-input logic gates



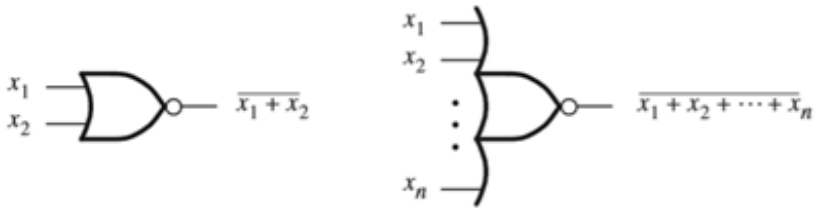
AND gates



NAND gates



OR gates



NOR gates

Axioms of Boolean Algebra

- 1a. $0 \cdot 0 = 0$
- 1b. $1 + 1 = 1$
- 2a. $1 \cdot 1 = 1$
- 2b. $0 + 0 = 0$
- 3a. $0 \cdot 1 = 1 \cdot 0 = 0$
- 3b. $1 + 0 = 0 + 1 = 1$
- 4a. If $x = 0$, then $x' = 1$
- 4b. If $x = 1$, then $x' = 0$

Boolean Algebra - Single Variable Theorems

- 5a. $x \cdot 0 = 0$
- 5b. $x + 1 = 1$
- 6a. $x \cdot 1 = x$
- 6b. $x + 0 = x$
- 7a. $x \cdot x = x$
- 7b. $x + x = x$
- 8a. $x \cdot x' = 0$
- 8b. $x + x' = 1$
- 9. $(x')' = x$

Boolean Algebra - Two Variable Properties

10a. $x \cdot y = y \cdot x$	Commutative
10b. $x + y = y + x$	
11a. $x \cdot (y \cdot z) = (x \cdot y) \cdot z$	Associative
11b. $x + (y + z) = (x + y) + z$	
12a. $x \cdot (y + z) = x \cdot y + x \cdot z$	Distributive
12b. $x + y \cdot z = (x + y) \cdot (x + z)$	
13a. $x + x \cdot y = x$	Absorption
13b. $x \cdot (x + y) = x$	
14a. $x \cdot y + x \cdot y' = x$	Combining
14b. $(x + y) \cdot (x + y') = x$	

Boolean Algebra - Two & Three Variable Properties

- **15a. (x · y)’ = x’ + y’** **DeMorgan’s theorem**
- **15b. (x + y)’ = x’ · y’**
- 16a. x + x’ · y = x + y
- 16b. x · (x’ + y) = x · y
- 17a. x · y + y · z + x’ · z = x · y + x’ · z Consensus
- 17b. (x + y) · (y + z) · (x’ + z) = (x + y) · (x’ + z)

Logic Function Synthesis - Three variable SOP & POS

- Function synthesis from truth table

Row number	x_1	x_2	x_3		Minterm	Maxterm
0	0	0	0		$m_0 = \bar{x}_1\bar{x}_2\bar{x}_3$	$M_0 = x_1 + x_2 + x_3$
1	0	0	1		$m_1 = \bar{x}_1\bar{x}_2x_3$	$M_1 = x_1 + x_2 + \bar{x}_3$
2	0	1	0		$m_2 = \bar{x}_1x_2\bar{x}_3$	$M_2 = x_1 + \bar{x}_2 + x_3$
3	0	1	1		$m_3 = \bar{x}_1x_2x_3$	$M_3 = x_1 + \bar{x}_2 + \bar{x}_3$
4	1	0	0		$m_4 = x_1\bar{x}_2\bar{x}_3$	$M_4 = \bar{x}_1 + x_2 + x_3$
5	1	0	1		$m_5 = x_1\bar{x}_2x_3$	$M_5 = \bar{x}_1 + x_2 + \bar{x}_3$
6	1	1	0		$m_6 = x_1x_2\bar{x}_3$	$M_6 = \bar{x}_1 + \bar{x}_2 + x_3$
7	1	1	1		$m_7 = x_1x_2x_3$	$M_7 = \bar{x}_1 + \bar{x}_2 + \bar{x}_3$

- **A Minterm** m_i is a complete argument vector **(a,b,c,...,x)** for which a Boolean function f(a,b,c,...,x) delivers the value **'1'.** It has "**minimum**" **satisfiability.**
- **A Maxterm** M_i is a complete argument vector **(a,b,c,...,x)** for which a Boolean function f(a,b,c,...,x) delivers the value **'0'.** It has "**maximum**" **satisfiability.**

SOP: Sum of product

- ABC'+B'CD+ABCD+AD'
- **A**→1, **A'**→0

POS: Product of sum

- (A+B+C')(A+D')(B'+C+D)
- **A**→0, **A'**→1

Logic Function Synthesis - Three variable SOP & POS

Example: 01: Write down the Boolean expression in **SOP** form that describes this truth table

A	B	C	Q
0	0	0	0
0	1	0	1
1	0	0	1
1	1	0	0
0	0	1	1
0	1	1	0
1	0	1	0
1	1	1	1

$$\overline{A}.B.\overline{C} + A.\overline{B}.\overline{C} + \overline{A}.\overline{B}.C + A.B.C = Q$$

Example: 02: Write down the Boolean expression in **POS** form that describes this truth table

Sensor Inputs

A	B	C	Output
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Logic Function Synthesis - Three variable SOP & POS

Example: 01: Write down the Boolean expression in **POS** form that describes this truth table

A	B	C	Output	Output	
0	0	0	0	0	(A + B + C)
0	0	1	0	1	
0	1	0	0	1	
0	1	1	1	1	
1	0	0	0	1	
1	0	1	1	1	
1	1	0	1	1	
1	1	1	1	0	$(\overline{A} + \overline{B} + \overline{C})$

$$\text{Output} = (A + B + C)\,(\overline{A} + \overline{B} + \overline{C})$$

Logic Function Synthesis - 2/3/4 variable k-map

- Function synthesis using k-maps

x_1	x_2	
0	0	m_0
0	1	m_1
1	0	m_2
1	1	m_3

x_2 \ x_1	0	1
0	m_0	m_2
1	m_1	m_3

x_1	x_2	x_3	
0	0	0	m_0
0	0	1	m_1
0	1	0	m_2
0	1	1	m_3
1	0	0	m_4
1	0	1	m_5
1	1	0	m_6
1	1	1	m_7

x_3 \ x_1x_2	00	01	11	10
0	m_0	m_2	m_6	m_4
1	m_1	m_3	m_7	m_5

x_3x_4 \ x_1x_2	00	01	11	10
00	m_0	m_4	m_{12}	m_8
01	m_1	m_5	m_{13}	m_9
11	m_3	m_7	m_{15}	m_{11}
10	m_2	m_6	m_{14}	m_{10}

x_1 (top), x_4 (right), x_3 (left), x_2 (bottom)

Logic Function Synthesis - 2/3/4 variable k-map

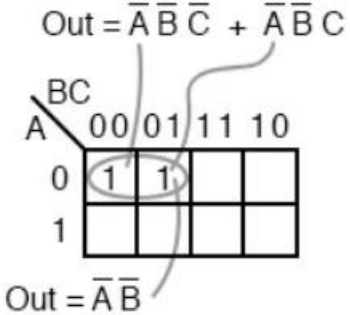
Function synthesis using k-maps

1. No zeros allowed.
2. No diagonals.
3. Only power of 2 number of cells in each group. (2^0=1, 2^1=2, 2^2=4, 2^3=8, etc.)
4. Groups should be as large as possible.
5. Every 1 must be in at least one group.
6. Overlapping allowed.
7. Wrap around allowed.
8. Fewest number of groups possible.

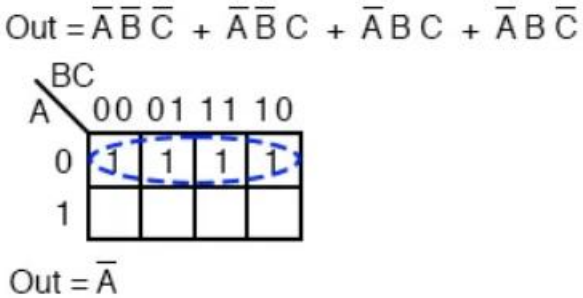
Visit: http://www.ee.surrey.ac.uk/Projects/Labview/minimisation/karrules.html

3 variable K-Map

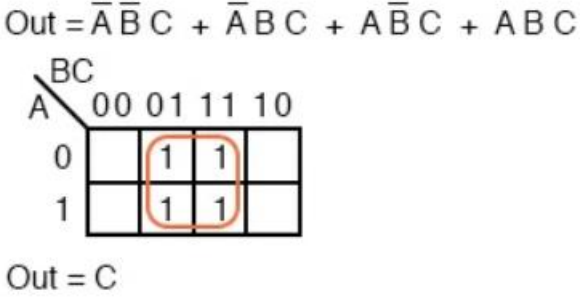
Example: 01



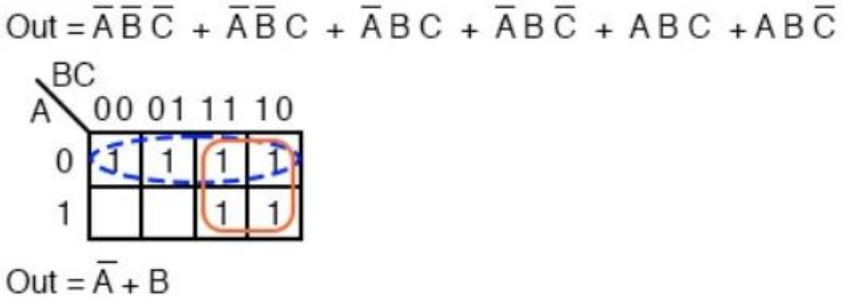
Example: 02



Example: 03



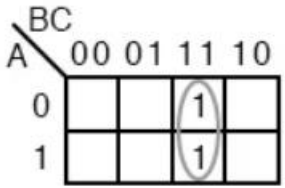
Example: 04



3 variable K-Map

Example: 05

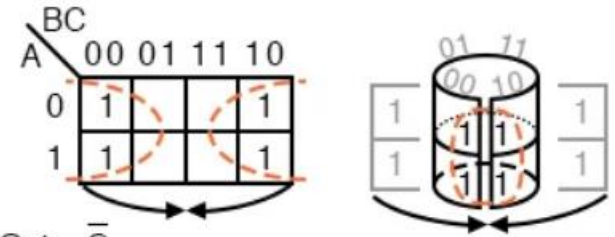
$Out = \overline{A}\,B\,C + A\,B\,C$



$Out = B\,C$

Example: 06

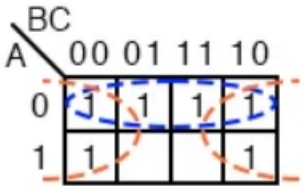
$Out = \overline{A}\,\overline{B}\,\overline{C} + A\,\overline{B}\,\overline{C} + \overline{A}\,B\,\overline{C} + A\,B\,\overline{C}$



$Out = \overline{C}$

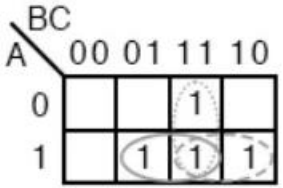
Example: 07

$Out = \overline{A}\,\overline{B}\,\overline{C} + \overline{A}\,\overline{B}\,C + \overline{A}\,B\,C + \overline{A}\,B\,\overline{C} + A\,\overline{B}\,\overline{C} + A\,B\,\overline{C}$



$Out = \overline{A} + \overline{C}$

Example: 08

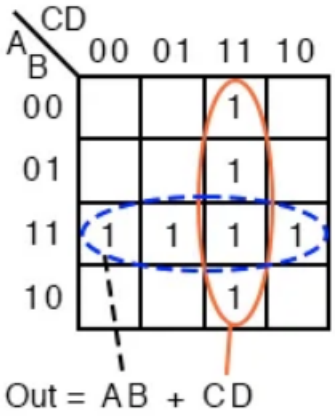


$Output = A\,B + B\,C + A\,C$

4 variable K-Map

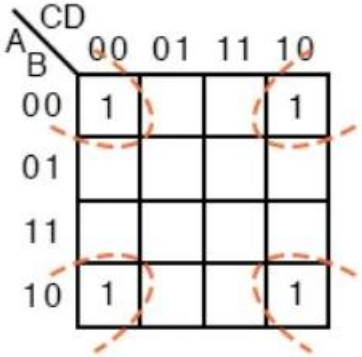
Example: 01

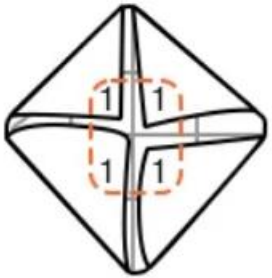
$$Out = \overline{A}\overline{B}CD + \overline{A}B\overline{C}D + ABCD + A\overline{B}CD + AB\overline{C}\overline{D} + AB\overline{C}D + ABC\overline{D}$$



Example: 02

$$Out = \overline{A}\overline{B}\overline{C}\overline{D} + \overline{A}\overline{B}C\overline{D} + A\overline{B}\overline{C}\overline{D} + A\overline{B}C\overline{D}$$

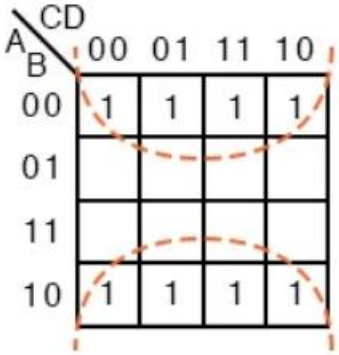




4 variable K-Map

Example: 03

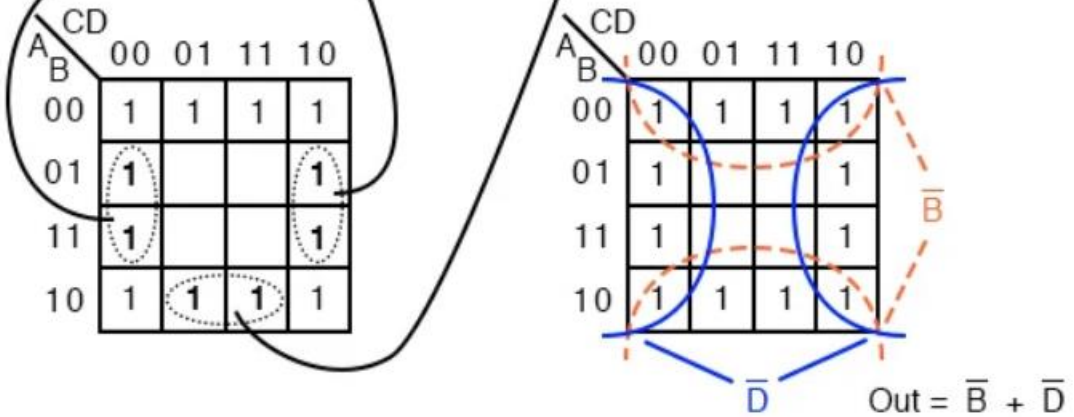
$$\text{Out} = \bar{A}\bar{B}\bar{C}\bar{D} + \bar{A}\bar{B}\bar{C}D + \bar{A}\bar{B}C\bar{D} + \bar{A}\bar{B}CD + A\bar{B}\bar{C}\bar{D} + A\bar{B}\bar{C}D + A\bar{B}C\bar{D} + A\bar{B}CD$$



$$\text{Out} = \bar{B}$$

Example: 04

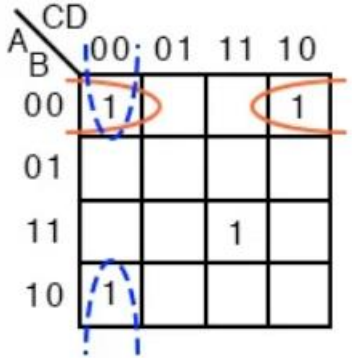
$$\text{Out} = \bar{A}\bar{B}\bar{C}\bar{D} + \bar{A}\bar{B}\bar{C}D + \bar{A}\bar{B}CD + \bar{A}\bar{B}C\bar{D} + B\bar{C}\bar{D} + BC\bar{D} + A\bar{B}\bar{C}\bar{D} + A\bar{B}D + A\bar{B}C\bar{D}$$



4 variable K-Map

Example: 05

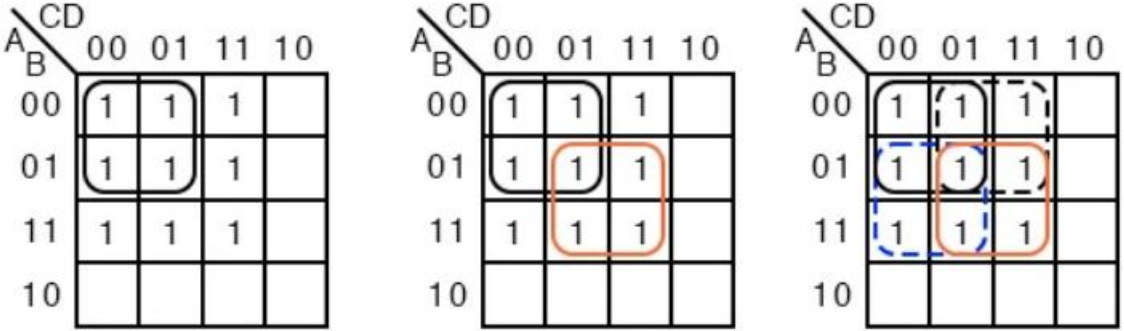
Out = $\overline{A}\,\overline{B}C\overline{D} + \overline{A}\,\overline{B}C\overline{D} + A\overline{B}\,\overline{C}\,\overline{D} + ABCD$



Out = $\overline{B}\,\overline{C}\,\overline{D} + \overline{A}\,\overline{B}\,\overline{D} + ABCD$

Example: 06

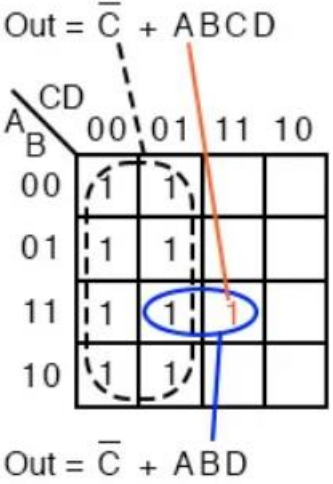
Out = $\overline{A}\,\overline{B}C\overline{D} + \overline{A}\,\overline{B}C\overline{D} + \overline{A}\,\overline{B}CD$
$+ \overline{A}B\overline{C}\,\overline{D} + \overline{A}B\overline{C}D + \overline{A}BCD$
$+ AB\overline{C}\,\overline{D} + AB\overline{C}D + ABCD$



Out = $\overline{A}\,\overline{C} + \overline{A}D + B\overline{C} + BD$

4 variable K-Map

Example: 07



Simplification by Boolean Algebra

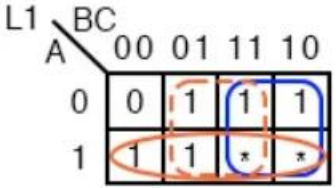
Out = $\overline{C}$ + ABCD

Applying rule A + $\overline{A}$B = A + B to the $\overline{C}$ + ABCD term

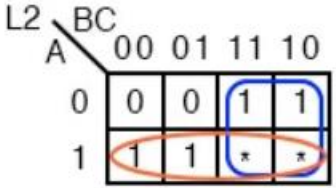
Out = $\overline{C}$ + ABD

Don’t Care Condition

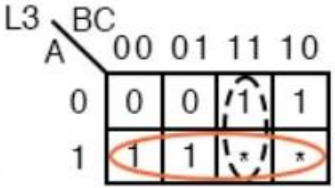
Example: 01



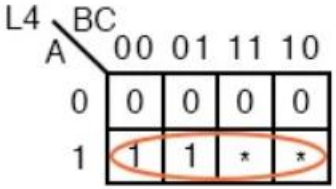
L1 = A + B + C



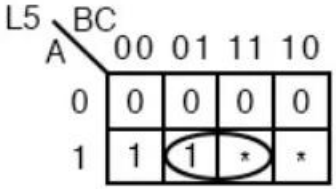
L2 = A + B



L3 = A + BC



L4 = A



L5 = AC

*** Don’t care cells may be used as either 1s or 0s, whichever is useful

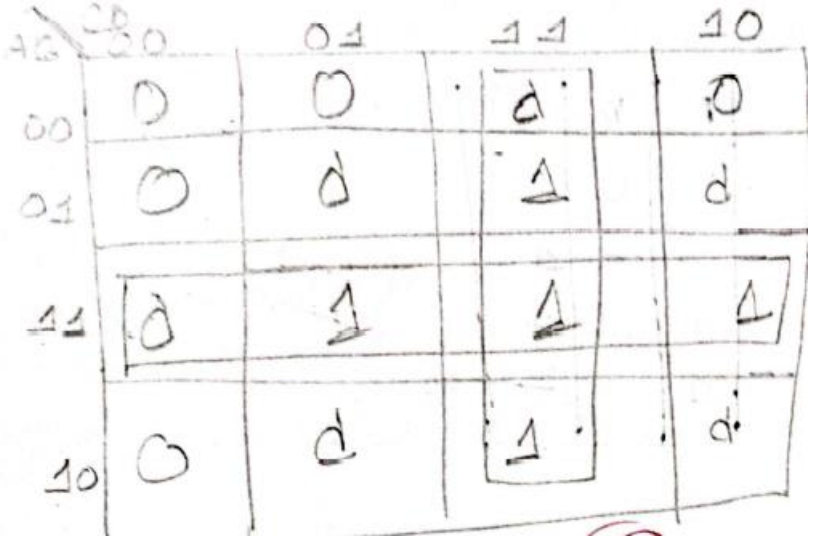
There are four sensors, A, B, C, and D that control the lights in a room. Each sensor transitions to a **HIGH** state when it detects an ambient light level exceeding a predefined threshold. The room's lights will turn **ON** when the majority of sensors are in the **LOW** state. If the number of sensors in the **HIGH** and **LOW** states is equal, the room's lighting condition remains **indeterminate** (i.e., a don't care condition). Answer questions (a-c) below based on this description.

(a) **[2 marks] Write** down the truth table for the system, describing all possible states of the sensors (A, B, C, D) and their corresponding output states.

(b) **[5 marks] Draw** the Karnaugh map (K-map) for the system based on the truth table in part (a). **Derive** the logical expression that governs the system from the K-map.

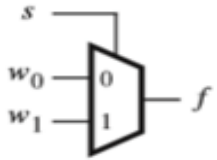
(c) **[5 marks] Draw** the CMOS logic circuit for the logical expression derived in part (b). **Specify** the number of MOSFETs required to implement your design.

A	B	C	D	Y	
0	0	0	0	0	→ OFF
0	0	0	1	0	→ OFF
0	0	1	0	0	→ OFF
0	0	1	1	d	→ don't care
0	1	0	0	0	→ OFF
0	1	0	1	d	→ don't care
0	1	1	0	d	→ don't care
0	1	1	1	1	→ ON
1	0	0	0	0	→ OFF
1	0	0	1	d	→ don't care
1	0	1	0	d	→ don't care
1	0	1	1	1	→ ON
1	1	0	0	d	→ don't care
1	1	0	1	1	→ ON
1	1	1	0	1	→ ON
1	1	1	1	1	→ ON



Multiplexer

- Multiple inputs, single output. Output is chosen by selector pin/s

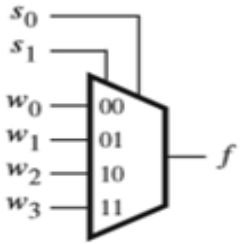


Graphical symbol

s	f
0	w_0
1	w_1

Truth table

2x1 Multiplexer



Graphical symbol

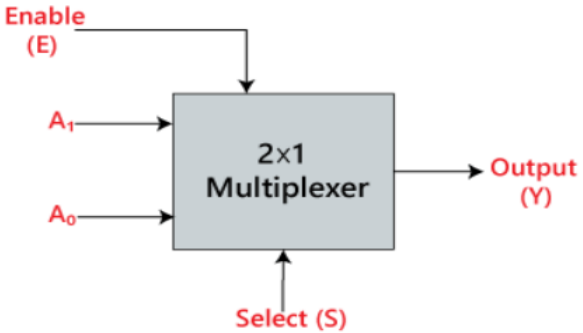
s_1	s_0	f
0	0	w_0
0	1	w_1
1	0	w_2
1	1	w_3

Truth table

4x1 Multiplexer

Multiplexer

Block Diagram:



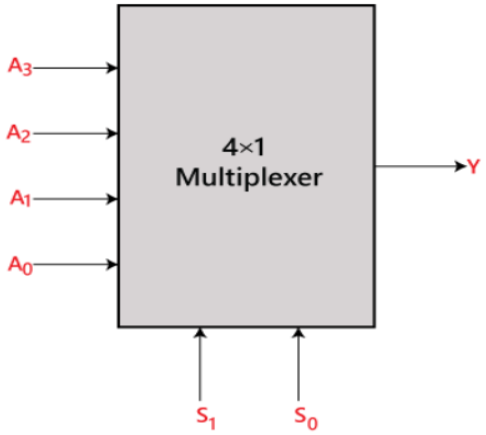
Truth Table:

INPUTS	Output
S_0	Y
0	A_0
1	A_1

The logical expression of the term Y is as follows:

$Y=S_0'.A_0+S_0.A_1$

Block Diagram:



Truth Table:

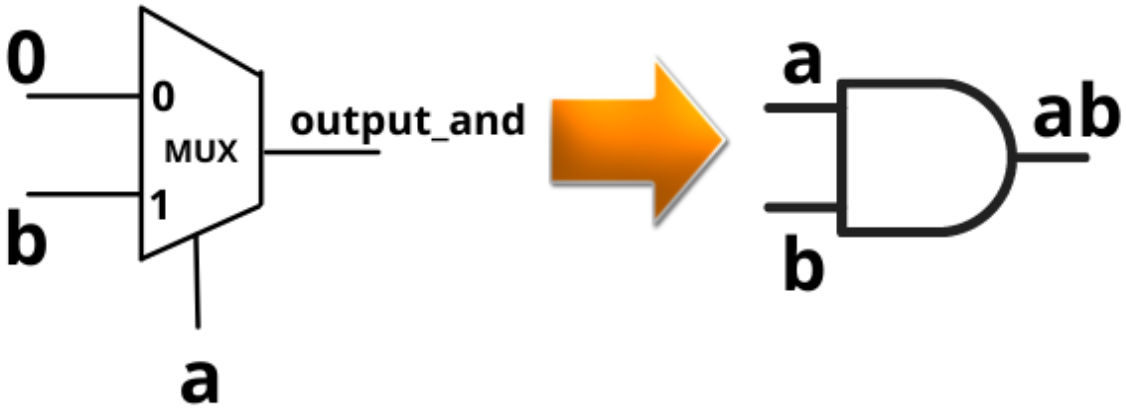
INPUTS		Output
S_1	S_0	Y
0	0	A_0
0	1	A_1
1	0	A_2
1	1	A_3

The logical expression of the term Y is as follows:

$Y=S_1' S_0' A_0+S_1' S_0 A_1+S_1 S_0' A_2+S_1 S_0 A_3$

Logic Gates using Multiplexer

And Gate Using 2:1 MUX



AND

A	B	Output
0	0	0
0	1	0
1	0	0
1	1	1

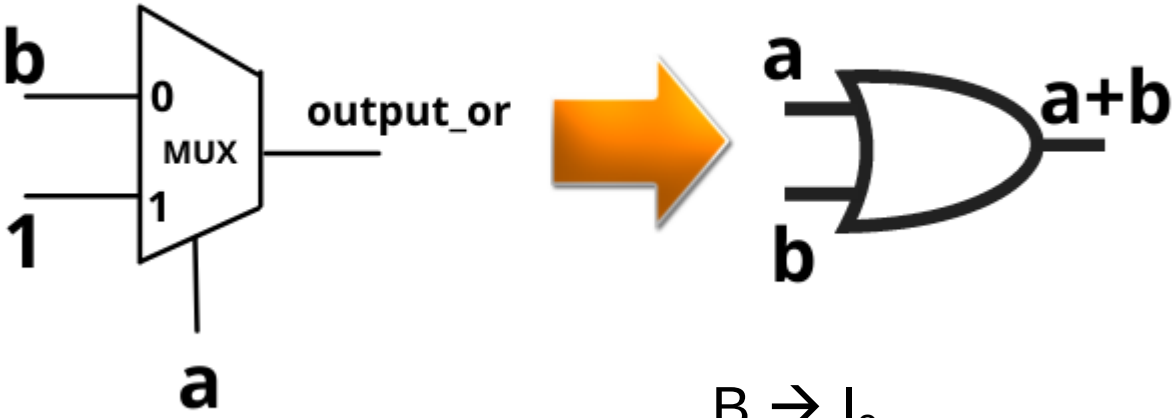
B → I_1

A → S

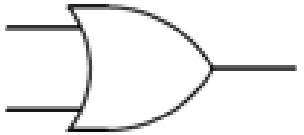
I_0 → 0 V

Logic Gates using Multiplexer

OR Gate Using 2:1 MUX



B → I_0
A → S
I_1 → 5 V



OR

A	B	Output
0	0	0
0	1	1
1	0	1
1	1	1

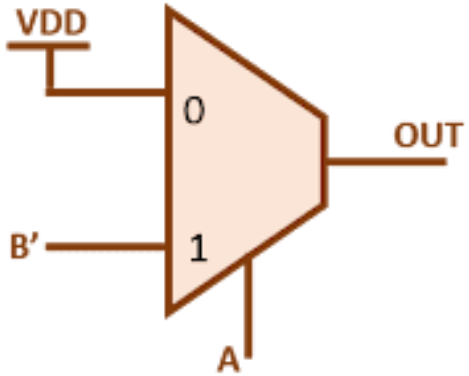
Logic Gates using Multiplexer

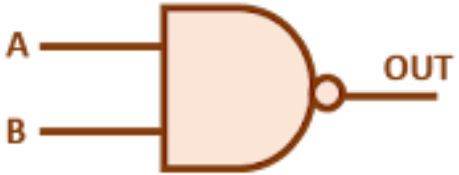
NAND Gate using MUX

A	B	OUT
0	0	1
0	1	1
1	0	1
1	1	0

OUT = 1 when A = 0

OUT = B' when A = 1





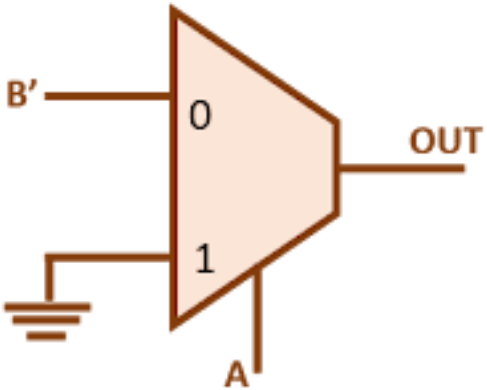
Logic Gates using Multiplexer

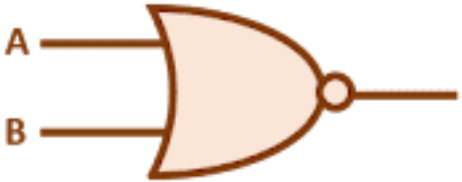
NOR Gate using MUX

A	B	OUT
0	0	1
0	1	0
1	0	0
1	1	0

OUT = B′ when A = 0

OUT = 0 when A = 1





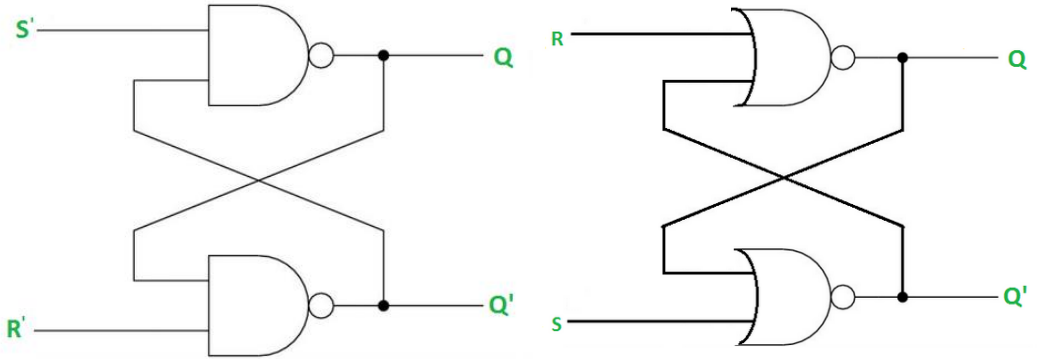
Difference between combinational and sequential circuits

Combinational Circuit	Sequential Circuit
This output is solely dependent on the current input.	This output is affected by both current and previous input.
The process is quick.	The process is slow.
It is intended to be simple.	When compared to combinational circuits, it is more efficiently designed.
There is no feedback from input to output.	A feedback path exists between input and output.
This is not dependent on time.	This is time-sensitive.
Basic building blocks: Logical gates	Basic building blocks: Flip-flops
Used for both arithmetic and boolean operations.	Mostly used for data storage.
Combinational circuits are incapable of storing any state.	Sequential circuits can store any state or retain previous states.
Combinational circuits do not require triggering because they lack a clock.	Sequential circuits require triggering because they are clock dependent.
These circuits lack a memory element.	Memory elements are used in these circuits.
It is simple to use and manage.	It is difficult to use and handle.

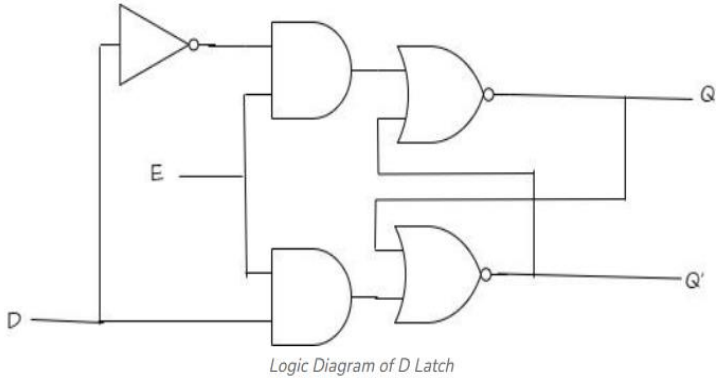
- **Combinational circuits** are built with logic gates such as AND, OR, NOT, NAND, and NOR. These logic gates serve as the foundation for combinational circuits.
- **Sequential circuits** are built with counters, shift registers, latches, etc.

Latch

- Latches are digital circuits that can store a single bit of information and hold its value until it is updated by new input signals. Latches are **asynchronous**.
- They are used in digital systems as temporary storage elements to store binary information.
- Latches can be implemented using various digital logic gates, such as AND, OR, NOT, NAND, and NOR gates.
- Different types of latches: SR (Set-Reset) Latches, Gated SR Latches, D Latches, Gated D Latches, JK Latches, T Laches.



SR latch



D latch

D Latch

- Level sensitive element
- A *positive level triggered* D latch
  - copies D to output Q, if Clock=1, else preserves the previous output
- A *negative level triggered* D latch
  - copies D to output Q, if Clock=0, else preserves the previous output

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

Characteristic table

D Flip-flop

Symbol

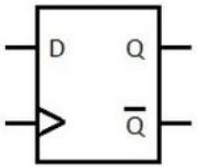
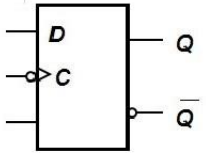


Table of truth:

clk	D	Q	$\overline{Q}$
0	0	Q	$\overline{Q}$
0	1	Q	$\overline{Q}$
1	0	0	1
1	1	1	0

Negative Edge triggering of D f/f

- Symbol

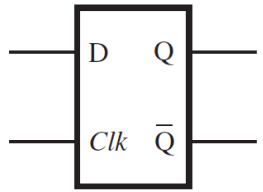


Truth table

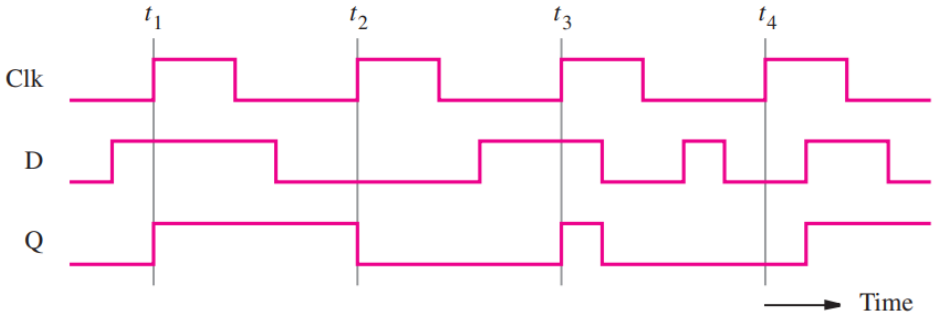
INPUTS		OUTPUTS		
CLK	D	Q	Q'	COMMENTS
0	0	0	1	NC
↑	1	0	1	NC
↓	0	0	1	RESET
↓	1	1	0	SET

D Latch

- Level sensitive element
- A *positive level triggered* D latch
  - copies D to output Q, if Clock=1, else preserves the previous output
- A *negative level triggered* D latch
  - copies D to output Q, if Clock=0, else preserves the previous output



Graphical symbol

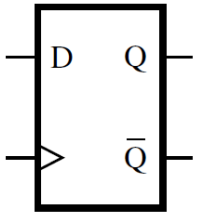


Clk	D	Q(t + 1)
0	x	Q(t)
1	0	0
1	1	1

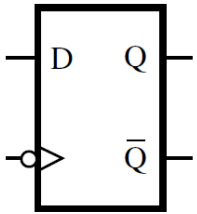
Characteristic table

D Flip-flop

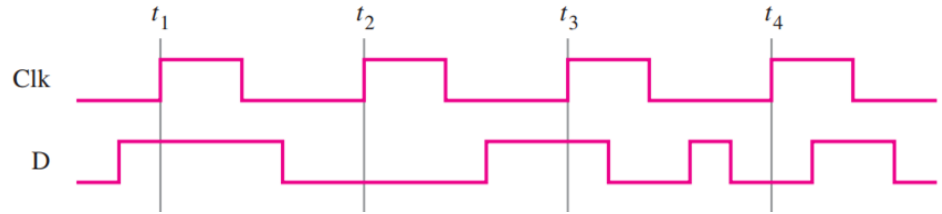
- Edge sensitive element
- **Flip-flops are synchronous** and operate based on clock signal



- A *positive edge triggered* D flip-flop
  - Sets Q=D at all positive edges (rising edges) of the clock, retains the old value of Q otherwise
- A *negative edge triggered* D flip-flop
  - Sets Q=D at all negative edges (falling edges) of the clock, retains the old

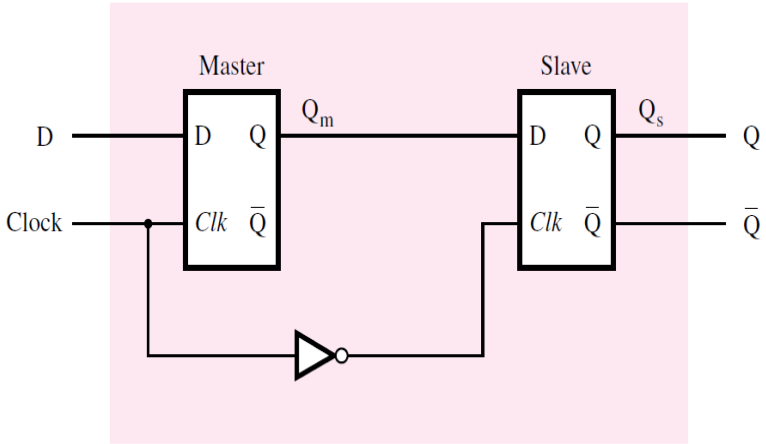


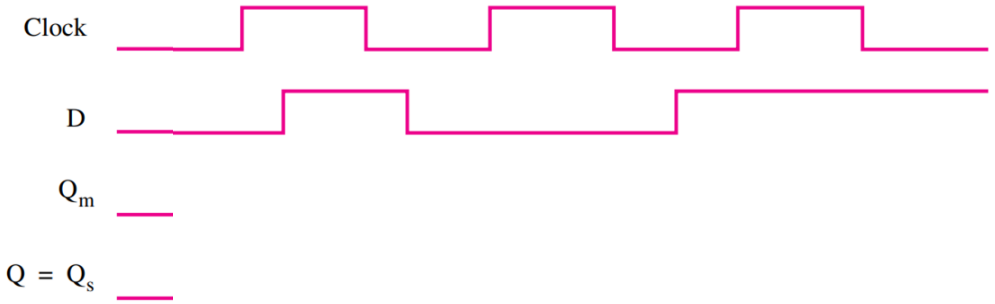
Graphical symbol

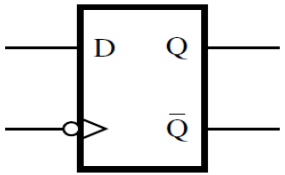


Building D Flip-flops using D Latches

- By cascading a positive level triggered D latch and a negative level triggered D latch we can build a negative edge triggered D flip-flop

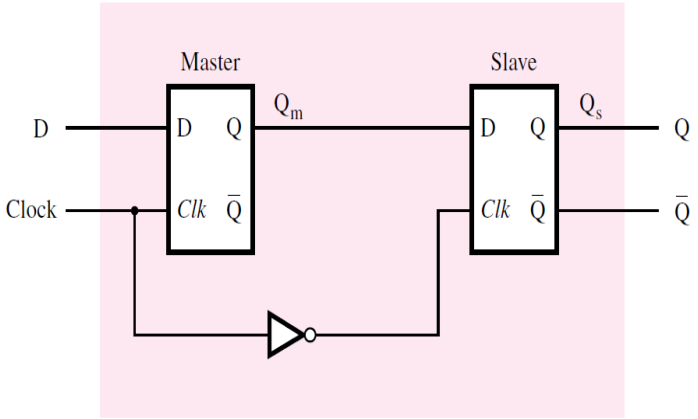


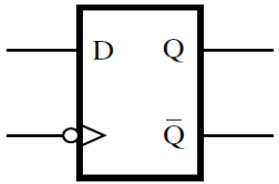


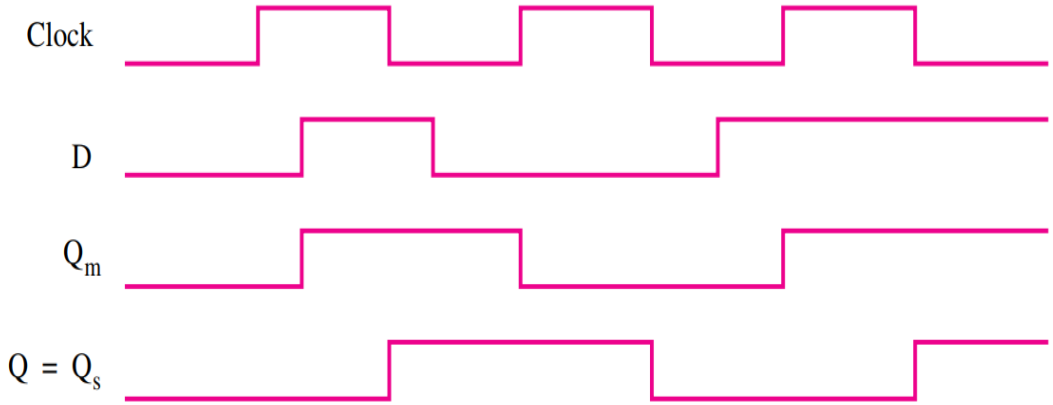


Building D Flip-flops using D Latches

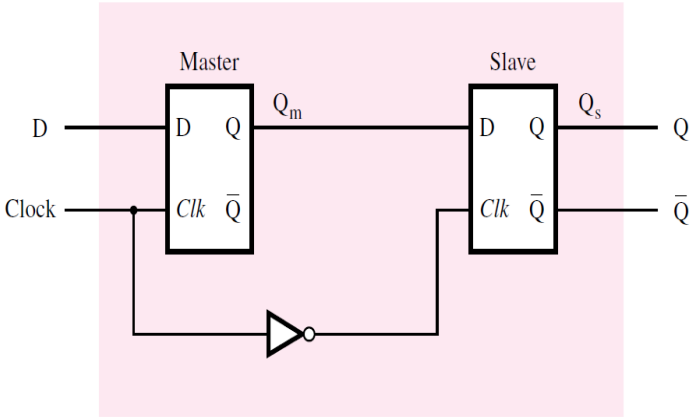
- By cascading a positive level triggered D latch and a negative level triggered D latch we can build a negative edge triggered D flip-flop

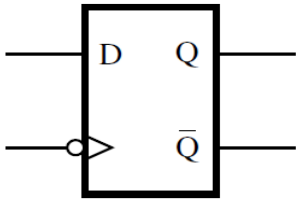




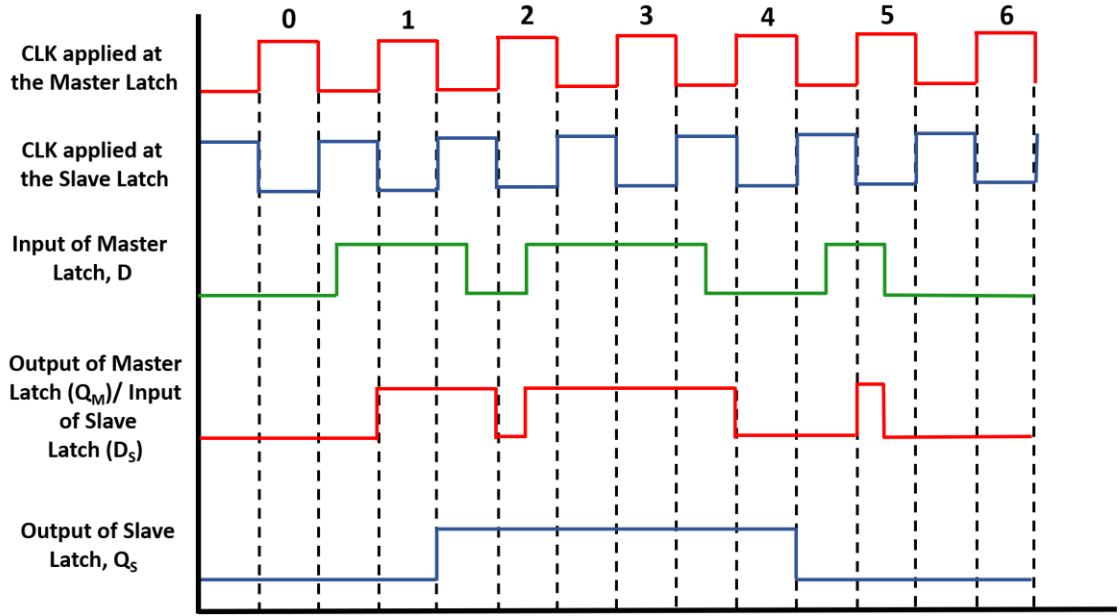


Building D Flip-flops using D Latches





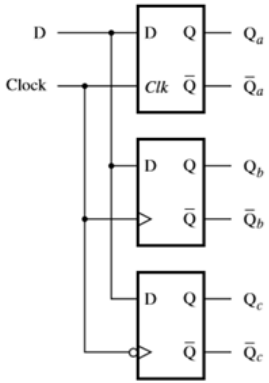
D Flip-flop



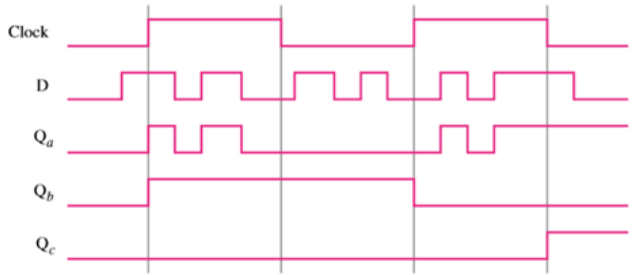
- **The output of the master-slave D flip-flop is similar to the output of a negative edge-triggered D flip-flo (FF)**

Level triggered vs. Edge triggered

- In level triggered elements
  - output is affected by the clock levels *(high/low)*
- In edge triggered elements
  - output is affected by the clock edges (positive edge/negative edge) (rising edge/falling edge)



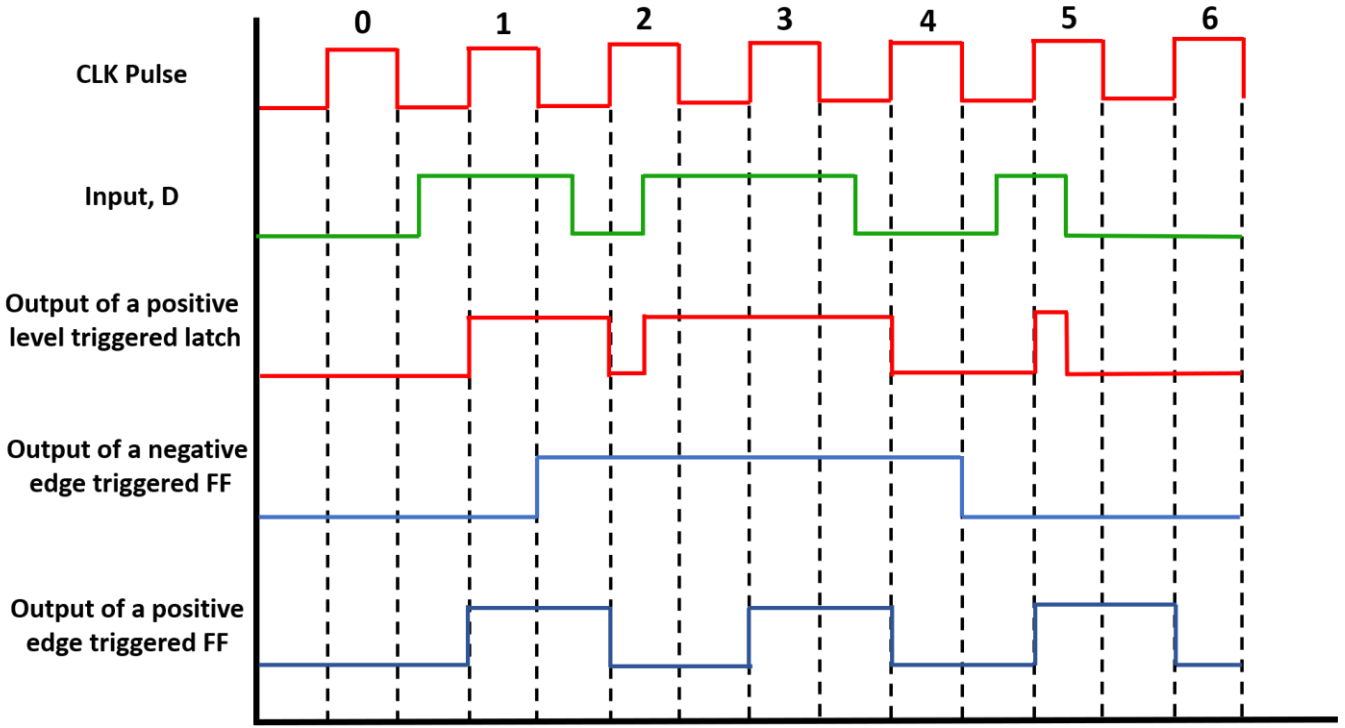
Circuit



Timing diagram

Level triggered vs. Edge triggered

- In level triggered elements
  - output is affected by the clock levels *(high/low)*
- In edge triggered elements
  - output is affected by the clock edges (positive edge/negative edge) (rising edge/falling edge)



Slide references

1. https://instrumentationtools.com/logic-gates/
2. Stephen Brown & Zvonko Vranesic - Fundamentals of Digital Logic with Verilog Design

Thank you!