

Adder

Adder: An adder is a digital circuit that performs addition of binary numbers.

Half Adder: A Half adder is a combinational circuit that adds two single-bit binary numbers and produces two outputs:

- sum (s) - the result of addition
- carry (c) - the carryout to the next higher bit.

Input: A, B | output: sum = $A \oplus B$, carry = $A \cdot B$

Full-Adder: A Full Adder is a combinational circuit that adds three bits: two input bits and a carry-in from the previous stage. It produces:

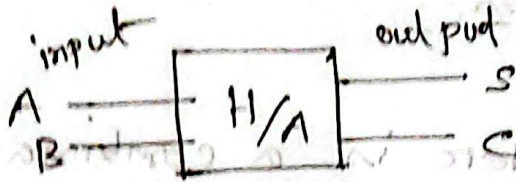
- sum (s) = the result of the bit-wise addition
- carry (c) = the carry-out to the next stage

$$\text{sum} = A \oplus B \oplus C_{in}$$

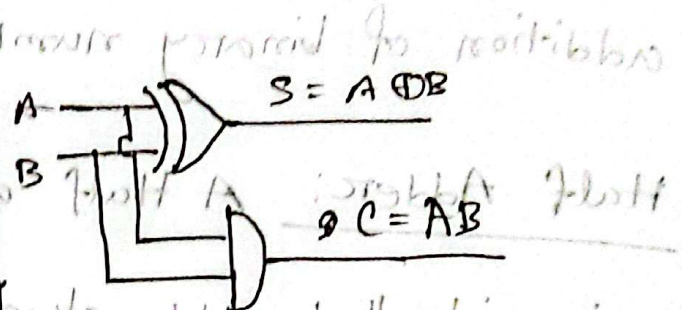
$$\text{carry} = A \cdot B + A \cdot C_{in} + B \cdot C_{in}$$

Half-Adder (XOR + AND) Gate

i) Half-Adder Symbol:



ii) Circuit:



iii) Truth Table:

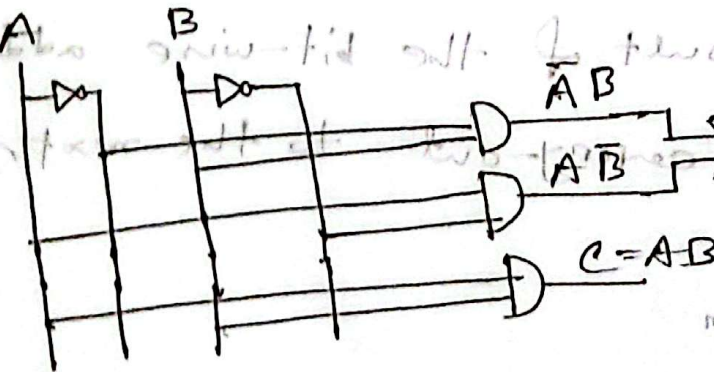
Input		Output	
A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

अवयव (मinterm) मध्ये.

$$S = \bar{A}B + A\bar{B}$$

$$C = AB$$

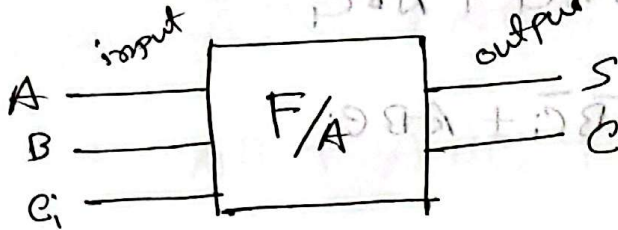
(अन्य गेटे वापरून)



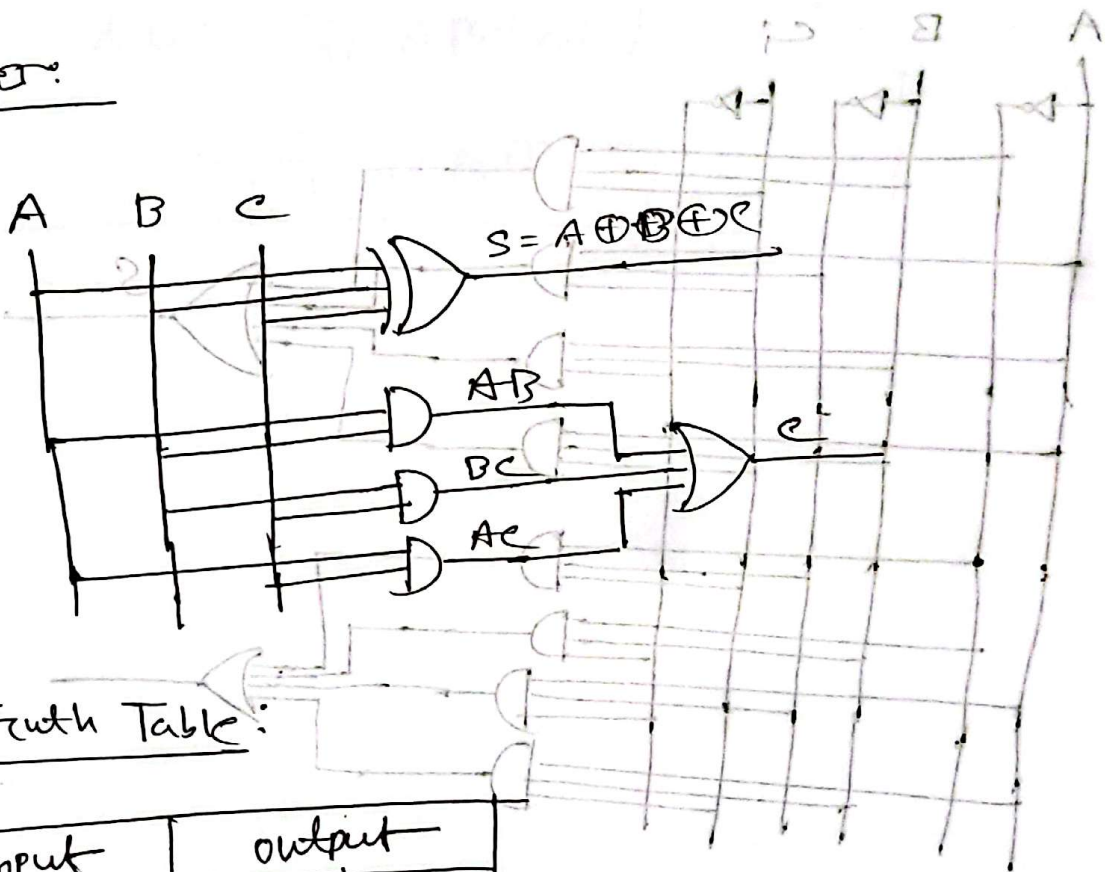
A = []

Full Adder

i) ସୂଚନା



ii) ଚିତ୍ର:



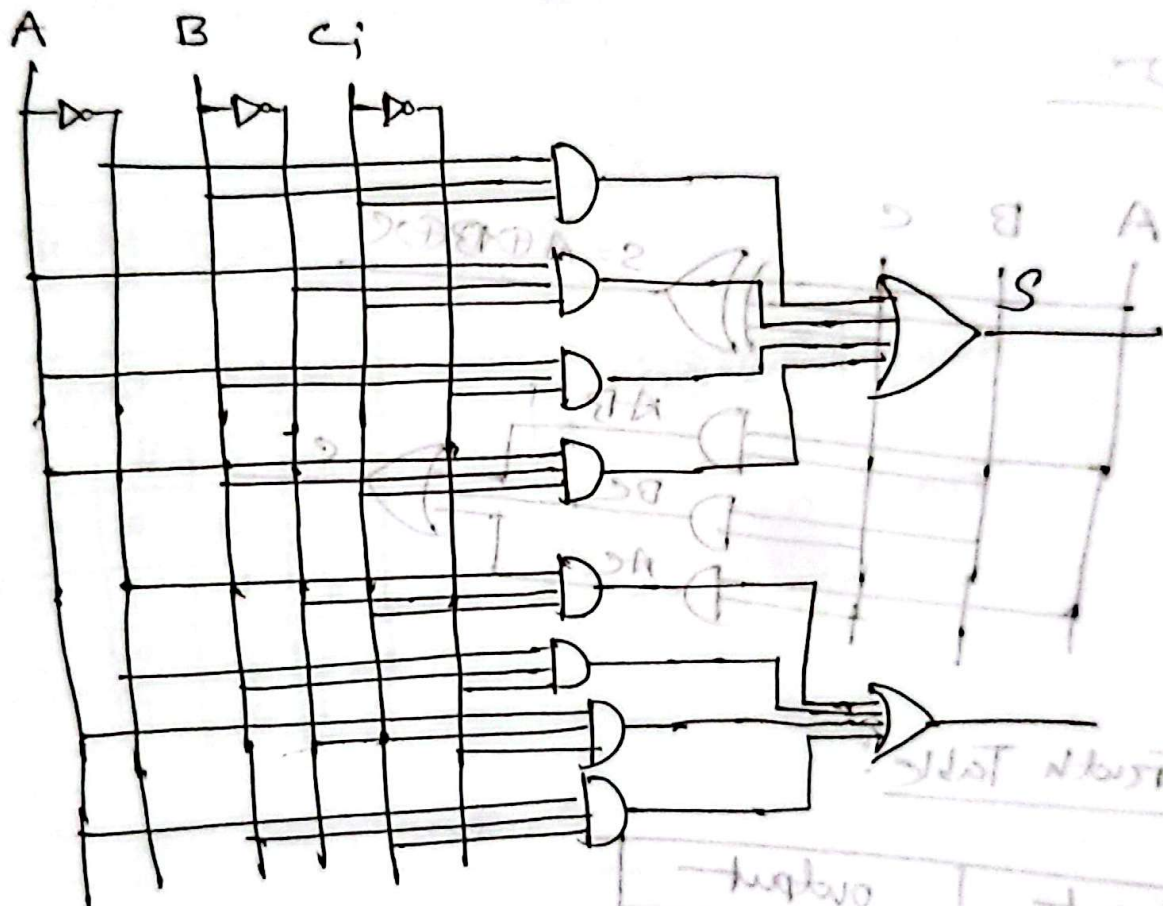
iii) Truth Table:

Input			output	
A	B	C	S	C
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

iv) लॉगिक चयन गेट्स का उपयोग

$$C_0 = \bar{A}B\bar{C}_i + A\bar{B}\bar{C}_i + A\bar{B}\bar{C}_i + AB\bar{C}_i$$

$$S = \bar{A}\bar{B}C_i + \bar{A}B\bar{C}_i + A\bar{B}\bar{C}_i + AB\bar{C}_i$$



$$\text{OR } C_1 = \bar{A}B\bar{C}_i + A\bar{B}\bar{C}_i + A\bar{B}\bar{C}_i + AB\bar{C}_i$$

$$= \bar{A}B\bar{C}_i + AB\bar{C}_i + A\bar{B}\bar{C}_i + AB\bar{C}_i + A\bar{B}\bar{C}_i + AB\bar{C}_i$$

$$= B\bar{C}_i(\bar{A} + A) + A\bar{C}_i(\bar{B} + B) + AB(\bar{C}_i + C_i)$$

$$= B\bar{C}_i + A\bar{C}_i + AB$$

$$\text{OR } C_1 = A \cdot B + C_i(A \oplus B)$$

Input	A	B	C _i	Output
1	0	0	0	0
2	0	0	1	0
3	0	1	0	0
4	0	1	1	1
5	1	0	0	0
6	1	0	1	1
7	1	1	0	1
8	1	1	1	1

$$C_o = \bar{A}B\bar{C}_i + A\bar{B}\bar{C}_i + AB\bar{C}_i + ABC_i$$

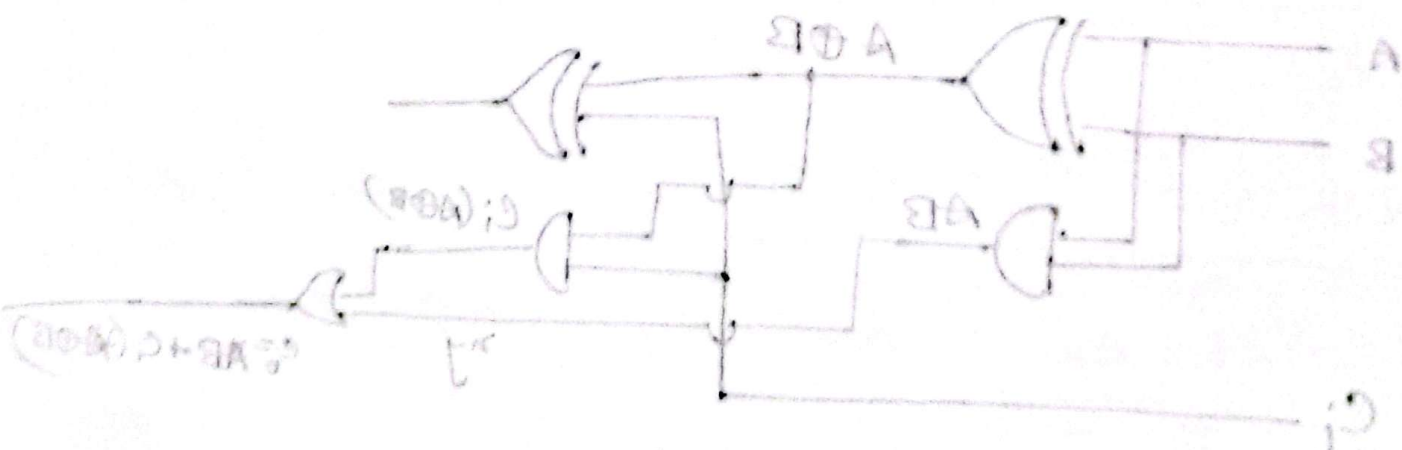
$$= \bar{A}B\bar{C}_i + A\bar{B}\bar{C}_i + AB(\bar{C}_i + C_i)$$

$$= AB + \bar{A}B\bar{C}_i + A\bar{B}\bar{C}_i$$

$$= AB + C_i(\bar{A}B + A\bar{B})$$

$$C_o = AB + C_i(A \oplus B)$$

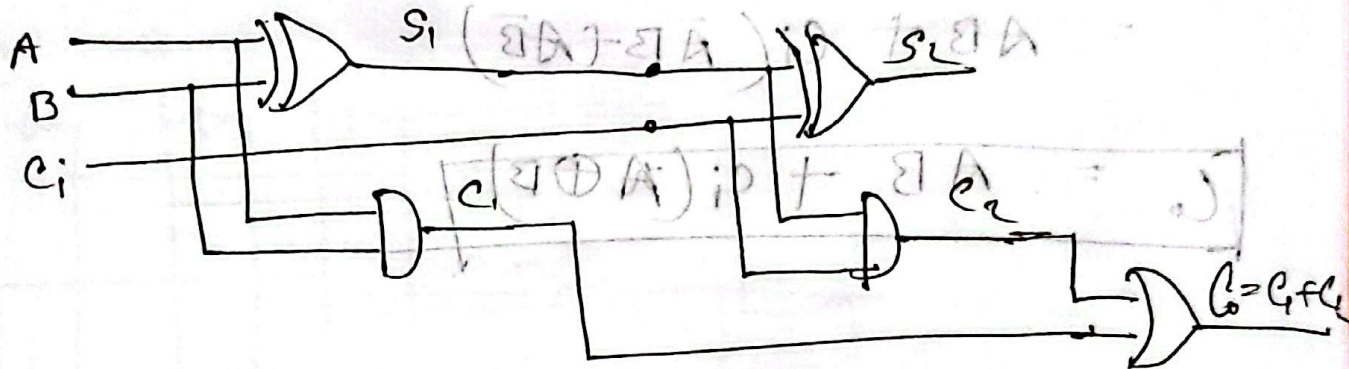
$$C_o = AB + C_i(A \oplus B)$$



* Half-Adder ବା Full-Adder:

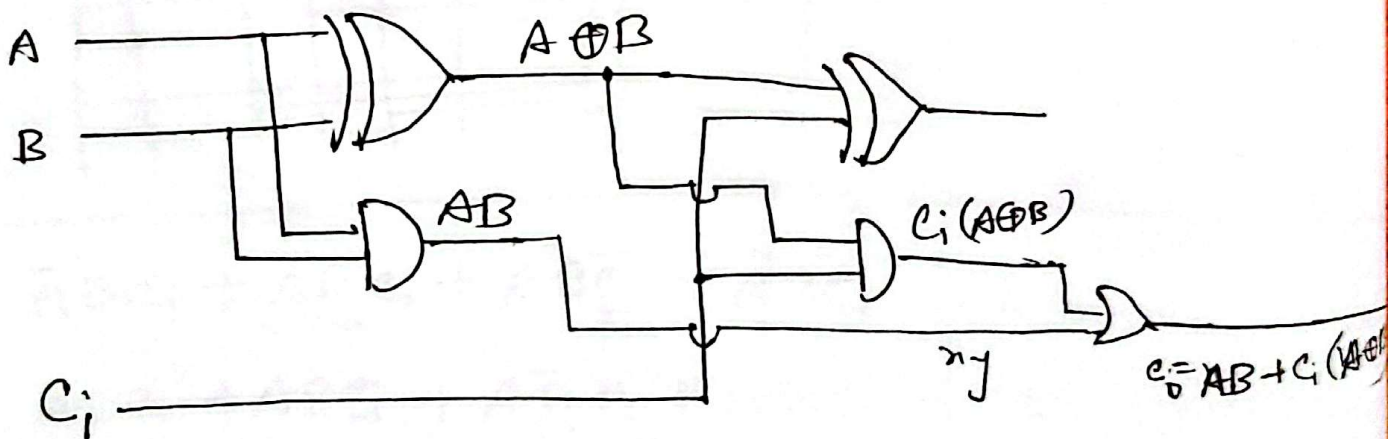
Circuit:

ଏହି Half Adder ଓ ଗାଣିତିକ ଓr gate ବ୍ୟବହାର କରାଯାଇଛି।

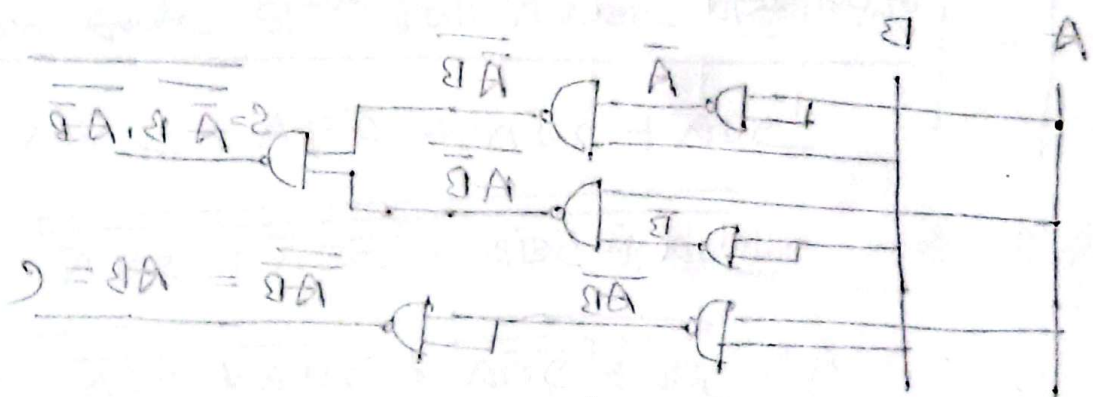
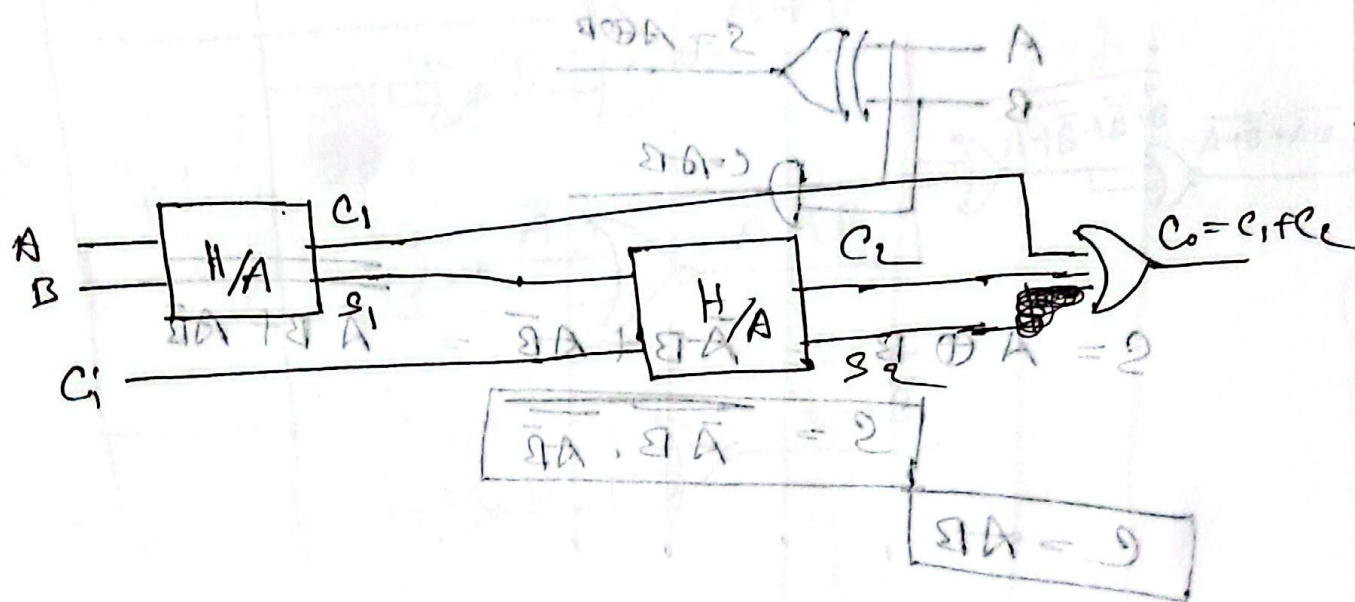


ଫର୍ମୁଲା:

$$C_0 = AB + C_1(A \oplus B)$$



ଉଦାହରଣ: ନିମ୍ନଲିଖିତ କାର୍ଯ୍ୟକାରୀ ସର୍କିଟ୍ ଗଠନ କରନ୍ତୁ



* ନିମ୍ନଲିଖିତ କାର୍ଯ୍ୟକାରୀ ସର୍କିଟ୍ ଗଠନ କରନ୍ତୁ

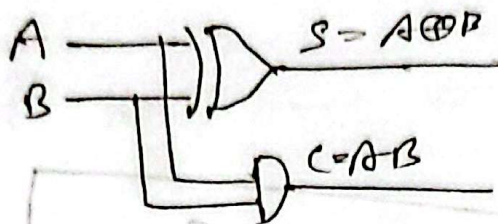
$$\overline{\overline{AB}} + \overline{\overline{AB}} = \overline{\overline{AB} + \overline{AB}} = \overline{AB + A\overline{B}} = 2$$

$$\overline{\overline{B} + \overline{A}} + \overline{\overline{B} + \overline{A}} =$$

$$\overline{\overline{B} + \overline{A}} + \overline{\overline{B} + \overline{A}} = 2$$

$$\overline{\overline{B} + \overline{A}} = \overline{\overline{AB}} = AB = 0$$

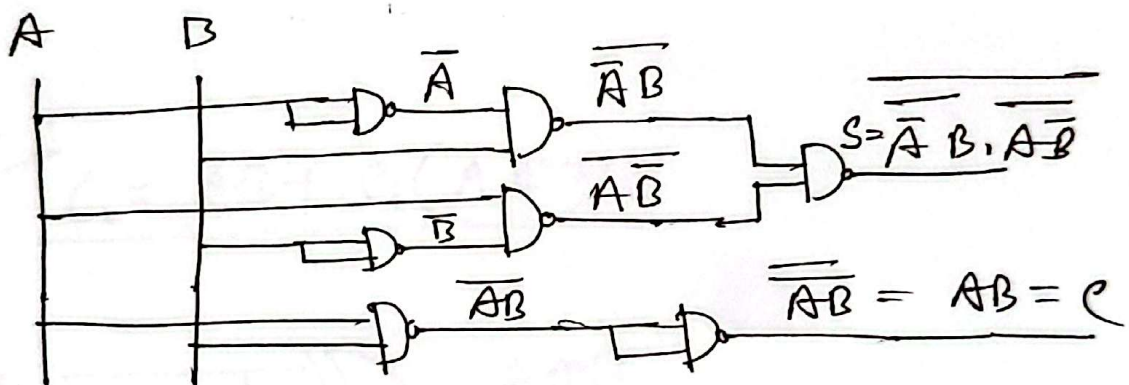
* Nand gate total Half-Adder circuit: Karnaugh



$$S = A \oplus B = \overline{A}B + A\overline{B} = \overline{A}B + A\overline{B}$$

$$S = \overline{A}B + A\overline{B}$$

$$C = AB$$



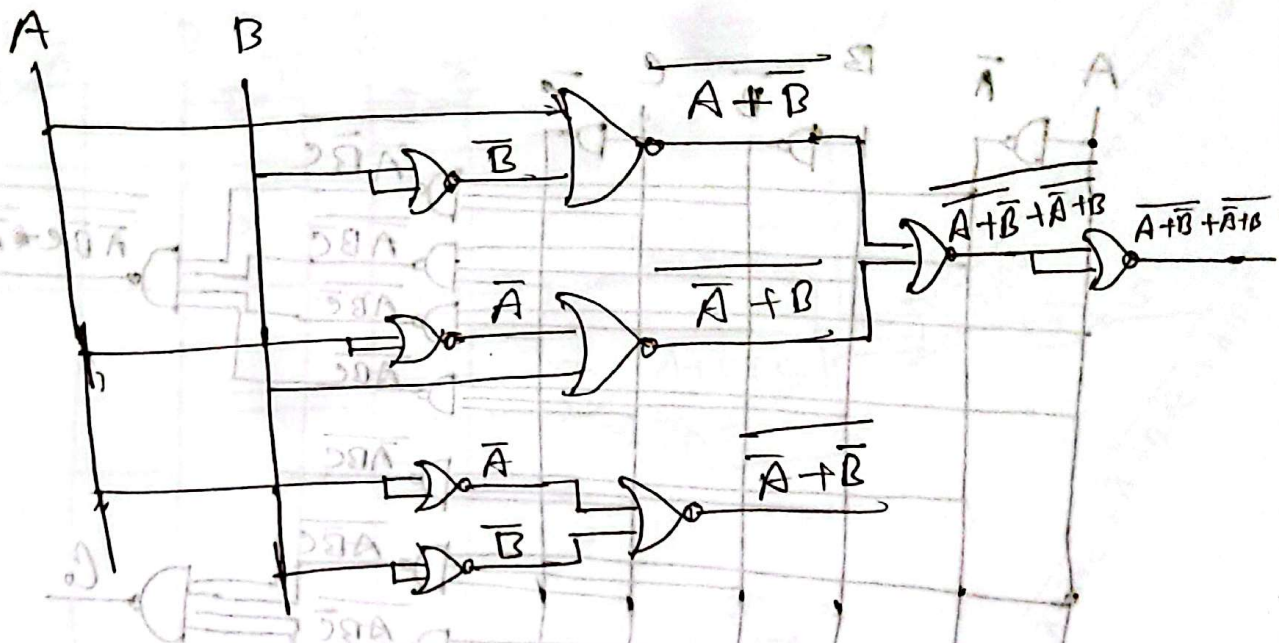
* NOR gate total Half-Adder:

$$S = \overline{A}B + A\overline{B} = \overline{\overline{\overline{A}B} + \overline{A\overline{B}}} = \overline{\overline{A}B + A\overline{B}}$$

$$= \overline{\overline{A} + \overline{B} + \overline{A} + \overline{B}}$$

$$S = \overline{A + \overline{B} + \overline{A} + B}$$

$$C = AB = \overline{\overline{AB}} = \overline{\overline{A} + \overline{B}}$$



* Nand gate for Full Adder

$$S = \bar{A}\bar{B}C + \bar{A}B\bar{C} + A\bar{B}\bar{C} + ABC$$

$$= \overline{\bar{A}\bar{B}C + \bar{A}B\bar{C} + A\bar{B}\bar{C} + ABC}$$

$$= \overline{\bar{A}\bar{B}C + \bar{A}B\bar{C} + A\bar{B}\bar{C} + ABC}$$

$$C = \bar{A}B\bar{C} + A\bar{B}\bar{C} + A\bar{B}C + ABC$$

$$= \overline{\bar{A}B\bar{C} + A\bar{B}\bar{C} + A\bar{B}C + ABC}$$

$$= \overline{\bar{A}B\bar{C} + A\bar{B}\bar{C} + A\bar{B}C + ABC}$$

$$= \bar{A}B\bar{C} + A\bar{B}\bar{C} + A\bar{B}C + ABC$$

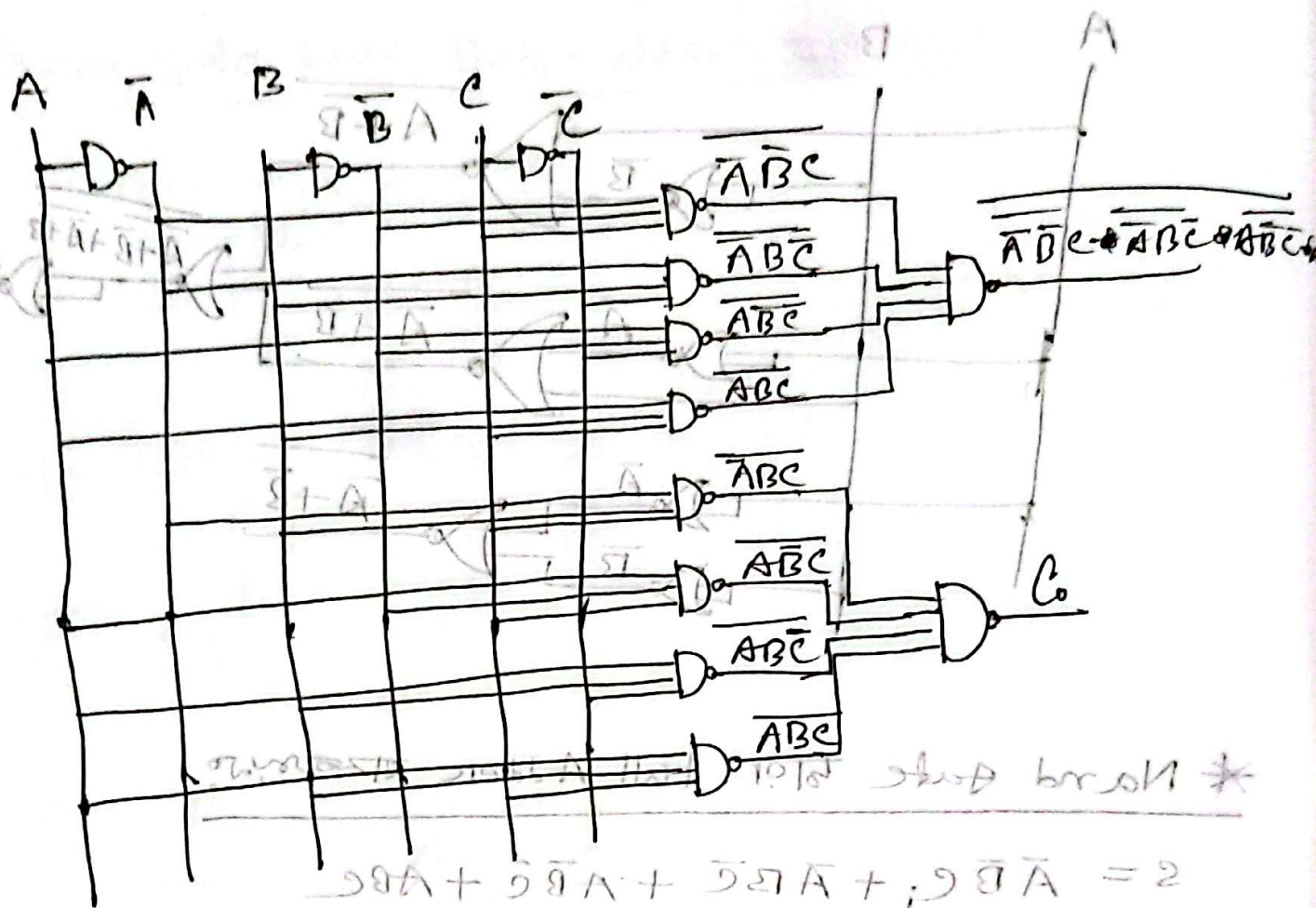
$$= \bar{A}B\bar{C} + A\bar{B}\bar{C} + A\bar{B}C + ABC$$

$$= \bar{A}B\bar{C} + A\bar{B}\bar{C} + A\bar{B}C + ABC$$

$$= \bar{A}B\bar{C} + A\bar{B}\bar{C} + A\bar{B}C + ABC$$

$$= \bar{A}B\bar{C} + A\bar{B}\bar{C} + A\bar{B}C + ABC$$

$$= \bar{A}B\bar{C} + A\bar{B}\bar{C} + A\bar{B}C + ABC$$



* NOR-gate का Full Adder गणित:

$$S = \bar{A}\bar{B}C + \bar{A}B\bar{C} + A\bar{B}\bar{C} + ABC$$

$$= \overline{\overline{\bar{A}\bar{B}C}} + \overline{\overline{\bar{A}B\bar{C}}} + \overline{\overline{A\bar{B}\bar{C}}} + \overline{\overline{ABC}}$$

$$= \overline{\bar{A} + \bar{B} + \bar{C}} + \overline{\bar{A} + \bar{B} + C} + \overline{\bar{A} + B + \bar{C}} + \overline{\bar{A} + B + C}$$

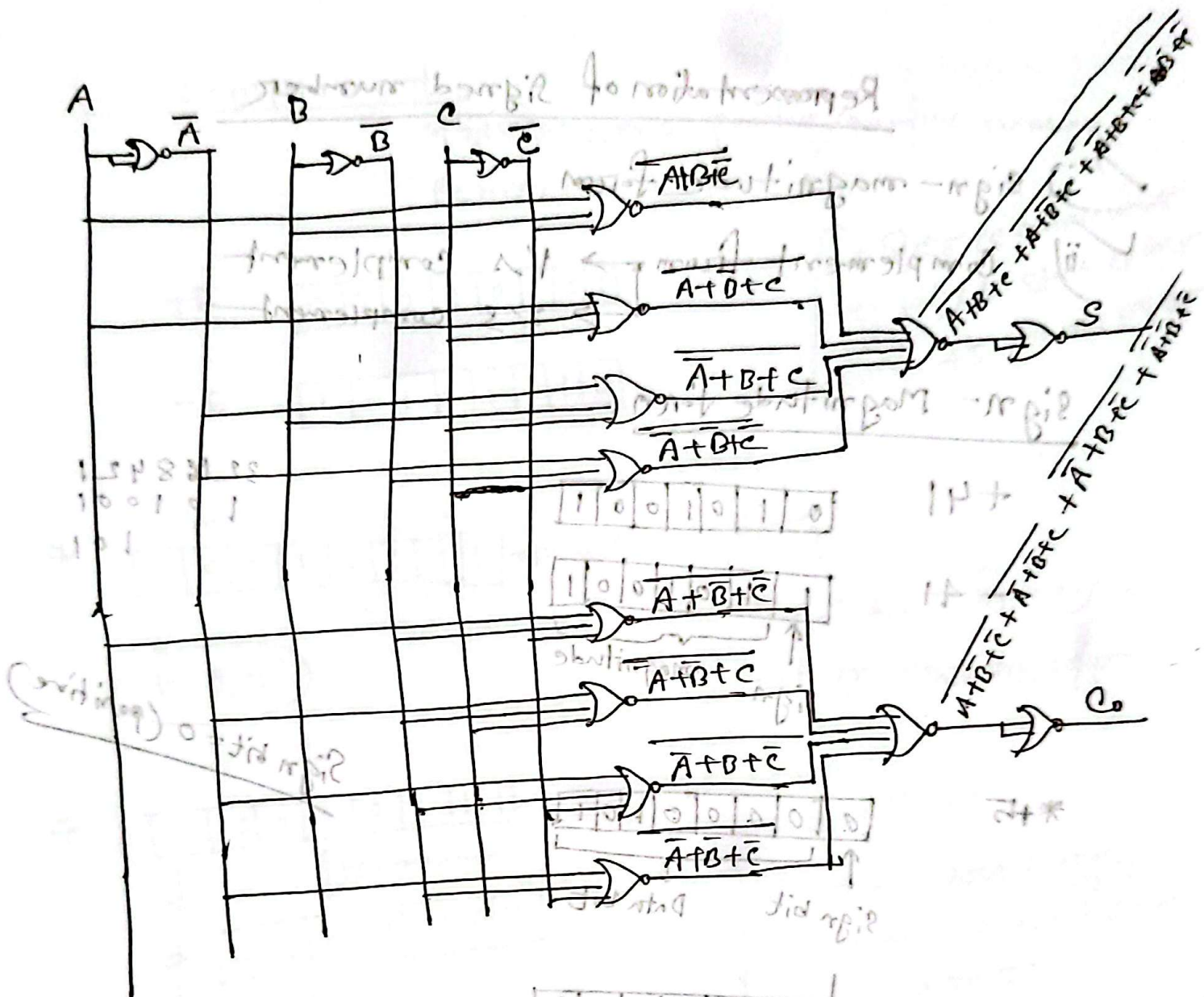
$$= \overline{A + B + \bar{C}} + \overline{A + B + C} + \overline{\bar{A} + B + C} + \overline{\bar{A} + B + \bar{C}}$$

$$C_c = \bar{A}BC + A\bar{B}C + AB\bar{C} + ABC$$

$$= \overline{\overline{\bar{A}BC}} + \overline{\overline{A\bar{B}C}} + \overline{\overline{AB\bar{C}}} + \overline{\overline{ABC}}$$

$$= \overline{\bar{A} + \bar{B} + \bar{C}} + \overline{\bar{A} + \bar{B} + C} + \overline{\bar{A} + B + \bar{C}} + \overline{\bar{A} + B + C}$$

$$= \overline{A + B + \bar{C}} + \overline{A + B + C} + \overline{\bar{A} + B + \bar{C}} + \overline{\bar{A} + B + C}$$



Truth table for the majority gate:

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

Truth table for the majority gate:

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

0 -
1 -

Two representations of

5-bit (00000) and 1-bit (00000)

Right-shifting from A 8/16 and 2/16 are also possible (3)

Left 1/2, 2/2, 3/2, 4/2, 5/2, 6/2, 7/2, 8/2, 9/2, 10/2, 11/2, 12/2, 13/2, 14/2, 15/2

Representation of signed number

i) sign-magnitude form

ii) Complement form $\rightarrow$ 1's complement
 $\rightarrow$ 2's complement

sign-magnitude form

+41

0	1	0	1	0	0	1
---	---	---	---	---	---	---

-41

1	1	0	1	0	0	1
---	---	---	---	---	---	---

↑
sign magnitude

*+5

0	0	0	0	0	1	0	1
---	---	---	---	---	---	---	---

↑ ↓
sign bit Data bits

-5

1	0	0	0	0	1	0	1
---	---	---	---	---	---	---	---

↑ ↓
MSB (Most) LSB (least significant bit)

32	16	8	4	2	1	
1	0	1	0	0	1	
				1	0	1

Sign bit = 0 (positive)

Disadvantage:

Two representations of zero (00000000 and 10000000)

+0
-0

0	0	0	0	0	0	0	0
1							

sign-magnitude form is not used in computers because it is not efficient (2)

1's and 2's complement

1's, 2's complement of numbers (e.g. 12, 10, 20)

1's complement

ଅନୁକ୍ରମିକ (ସଂକଳନ) positive/negative number
 Remainder to store 20

+5 0 0 0 0 0 1 0 1

-5 1 1 1 1 1 0 1 0

5 ଥିବା 1's complement
 ଓ 10101010 ଥିବା 2's complement
 ଓ 5 ଥିବା 1's complement

* 0 0 0 1 0 1 1 1 ⇒ 23
 + 23

ସଂକଳନ 23 ଥିବା 8
 bit (n) ଥିବା ସଂକଳନ

* 1 0 0 1 0 1 1 1
 ↓ 1's complement
 0 1 1 0 1 0 0 0
 - 104

ଉଦାହରଣ:

(1) 23 ଥିବା (-)

(2) 23 ଥିବା positive
 1's/2's
 ସଂକଳନ ସଂକଳନ

ଫଳାଫଳ

(-) ସଂକଳନ ସଂକଳନ
 ସଂକଳନ

* 1 1 1 1 1 0 1 0
 ↓ 1's complement
 0 0 0 0 0 1 0 1
 - 5

0 1 1 1 0 0 0 0
 1 5 8

Disadvantage: Still has two representations of zero
 (00000000) and (11111111)

2's complement:

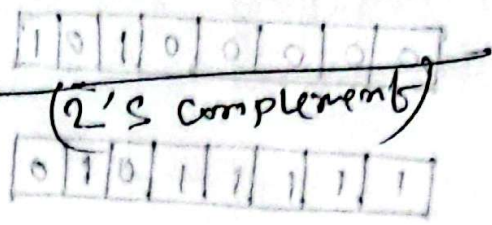
1001010

1's complement

0110101

+1

0110110



Register Store:

* +5

00000101

-5

11111011

2's complement



* -13 store:

1st: +13

00001101

-13

11110011

11110100

00010110

Not -

* 2's complement

Decimal no:

10000000

2's Equivalent

10000000

128 64 32 16 8 4 2 1

-128

positive 2's direct

no

* 00001110

+14

Still has two representations (11111111)

* + 0

0	1	0	0	0	0	0	0
---	---	---	---	---	---	---	---

 = (821)₁₀ - (121)₁₀ *

- 0

0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---

 11000001 = 121₁₀

00000000 $\rightarrow$ 1's complement

$$\begin{array}{r} 11111111 \\ + 1 \\ \hline 100000000 \end{array}$$
 8 bit

A dvantage: Only one representation of zero

- Arithmetic operations (addition/subtraction) are easier.

1's Complement গণিত (পূরক/বিপরীত)

$\rightarrow$ যদি end carry মূল 8 bit এর বরাবরী থাকে
 final ans. এর মূল 1 (যদি নাহলে 2)

Ex-1 : $+168 - 134 = 168 + (-134)$

$+168 = 10101000$

$+134 = 10000110$

$\therefore -134 = 01111001$

$\rightarrow$ 1's complement

$$\begin{array}{r} 10101000 \\ 01111001 \\ \hline 10010001 \\ + 1 \\ \hline 10010010 \end{array}$$

$$\therefore (10010010)_2 = + (34)_{10}$$

$$*(131) - (+168) = (131) + (-168)$$

$$+131 = 10000011$$

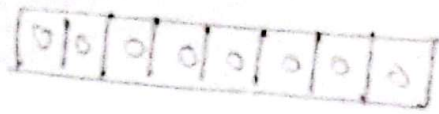
$$+168 = 10101000$$

$$-168 = 01010111$$

$$\begin{array}{r} 10000011 \\ 01010111 \\ \hline \end{array}$$

$$\begin{array}{r} 11011010 \\ 10010010 \\ \hline \end{array}$$

37



1's complement
2's complement
result
last bit 2:-1

1's complement

2's complement

$$Ex-1: +168 - 131 = 168 + (-131)$$

$$+168 = 10101000$$

$$+131 = 10000110$$

$$-131 = 01111001$$

$$\begin{array}{r} 10101000 \\ 10000110 \\ \hline \end{array}$$

$$10111101$$

$$10000110$$

$$+1$$

$$(10111101) = +131$$

2's Complement नक़ज़िफ़ (अस/असिअस)

* Ex-I

$(+5) + (+13)$

$+5 =$

00000101

$+13 =$

00001101

$18 = 00010010 = (+18)$

* E-II: $+25$ अस -12 असिअस

$\therefore +25 - (-12) = 25 + 12$

* E-III: $+8 + (-5)$

$+8 = 00001000$

$+5 = 00000101$

$-5 = 11111011$

2's Complement

$+8 = 00001000$

$-5 = 00000101$

$3 = 00001101$

$+8 = 00001000$

$-5 = 11111011$

$(+3) = 00001101$

* $(-8) + (+5)$

2 4 2 1
0 1 0 1

$15 + 8 = 11111000$

$10 + 8 = 00001000$

$-8 = 11111000$

$+5 = 0000101$

$-3 = 1111101$

$\rightarrow -(0000011) = -3$

result (+)
(-) 2's complement
2's complement

* Sign-Extension

A 4-bit signed number 1110 is to be sign-extended to 8 bits.

i) Perform the sign extension.

ii) Determine the decimal value of both the original and the extended numbers.

Answer:

Given the 4-bit number: 1110

Here, MSB = 1, so it's negative (in two's complement form)

$11000001 = -3$

To sign-extend from 4 bits to 8 bits,
we repeat the sign bit (1) on the left until it
becomes 8 bits.

Result: $1110 \rightarrow 4\text{-bits}$

8 bit: $\underline{1111} \underline{1110}$

ii) original 4-bit number: 1110

$\begin{array}{r} 8421 \\ 10 \end{array}$

Since MSB = 1 $\rightarrow$ negative

$$\therefore 1110 \xrightarrow{\text{2's complement}} (0010)_2 \rightarrow (-2)_{10}$$

Extended 8-bit number

11111110

MSB = 1 (-)

$$\underline{11111110} \xrightarrow{\text{2's complement}} (00000010)_2 \rightarrow (-2)_{10}$$

Final Ans:

i) 8-bit sign-extended number: 11111110

ii) Decimal value:

• Original 4-bit number = -2

• Extended 8-bit number = -2

Flip-Flop

- SR - Flip-Flop
- JK - " "
- D - " "
- T - " "
- Master-Slave " "

A FF is a basic digital memory circuit that can store one bit of data. It has two stable states and can be used to store binary 0 or 1.

SR Flip-Flop :

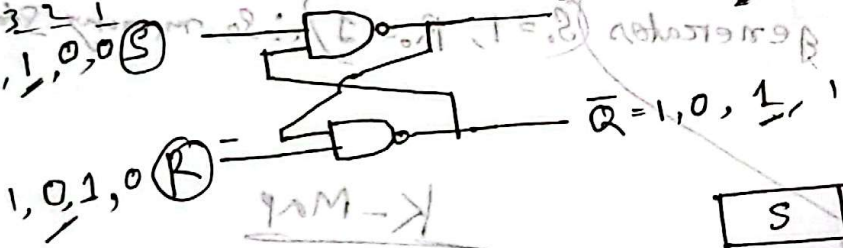
S-R Latch

NAND-Truth table

A	B	output
0	0	0
0	1	1
1	0	1
1	1	0

S	R	Q	Q̄
0	0	Forbidden	Invalid
0	1	1	0
1	0	0	1
1	1	Q	Q̄

Hold storage stage / unchanged



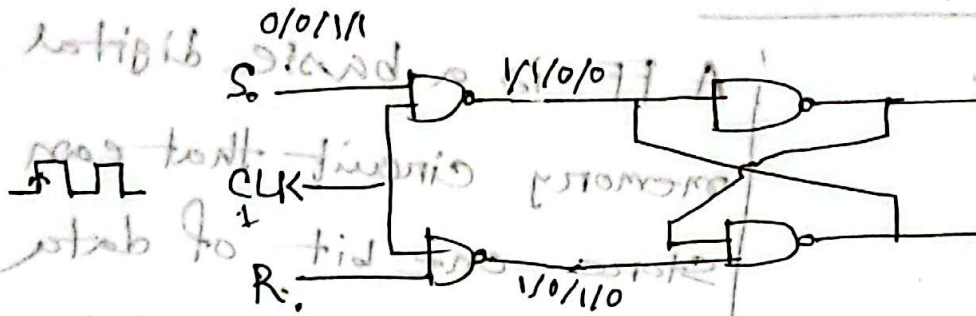
1	X	0	0
1	1	0	0

FF are building blocks of registers, counters and memory devices in digital electronics

$$\bar{Q} + 2 = 11 \text{ mod } 2$$

$$(J) (\bar{Q}) + 2 = (J) 11 \text{ mod } 2$$

SR Flipflop :



Truth Table

CLK	S	R	Q _n	Q _{n+1}
0	x	x	Memory	Q _n
1	0	0	Memory	Q _n
1	0	1	0	1
1	1	0	1	0
1	1	1	Invalid	

Unchanged/Hold

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	Invalid

If CLK = 0, it always generates (S = 1, R = 1) ∴ So, memory state

Characteristic

S	R	Q _n	Q _{n+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	0
1	1	1	0

K-Map

		Q _{n+1} / SR			
		00	01	11	10
Q _n	0	0	0	X	1
	1	1	0	X	1

$$Q_{n+1} = S + Q_n \bar{R}$$

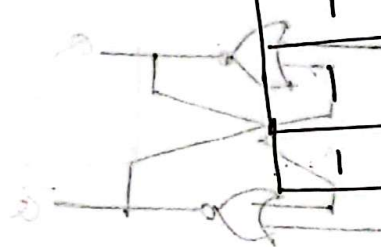
$$Q_{n+1}(t) = S + Q_n(\bar{R})(t)$$

Excitation Table (9011) 77 - Characteristic Table

Q_n	Q_{n+1}	S	R
0	0	0	x(0)
0	1	1	0
1	0	0	1
1	1	x(1)	0

4

Q_n	S	R	Q_{n+1}
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	x(1)
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	x(1)



Truth Table

Input	R	S	Output
0/0/0	x	x	0
0/0/1	0	0	1
0	1	0	1
1	0	1	1
1/1/1	1	1	1

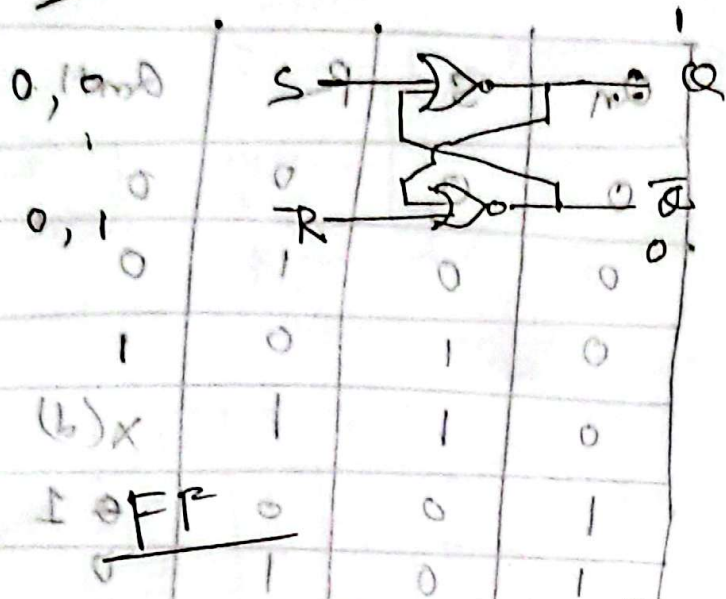
↔

$\bar{Q}_n$	Q_n	R	S	Output
1	0	x	x	0
1	0	0	0	1
1	1	0	0	1
0	1	0	1	1
0	1	1	1	1

Latch

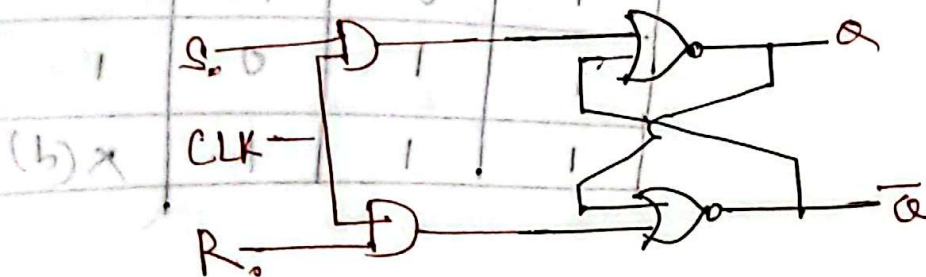
SR-FF (NOR)

SR-FF (NOR)



S	R	Q	Q-bar
0	0	Hold/mem	
0	1	0	1
1	0	1	0
1	1	Invalid	

FF



Truth Table

CLK	S	R	Q _n	Q-bar _n
0	X	X	memory	
1	0	0	memory	
1	0	1	0	1
1	1	0	1	0
1	1	1	Invalid	



CLK	S	R	Q _{n+1}
0	X	X	Q _n /hold/mem
1	0	0	Q _n /memory
1	0	1	0
1	1	0	1
1	1	1	Invalid

Characteristics Table:

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

Excitation Table

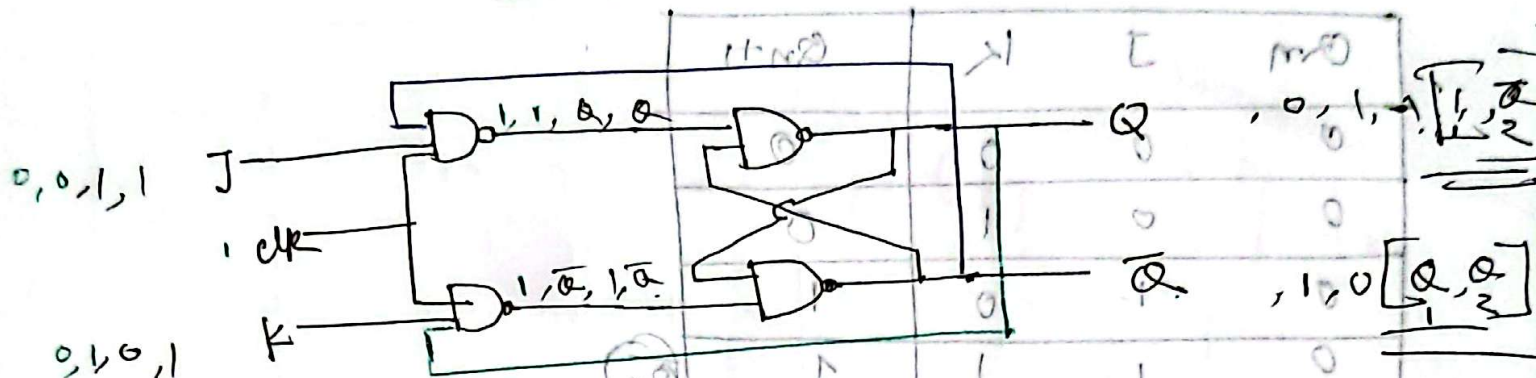
Q_n	Q_{n+1}	S	R
0	0	0	x
0	1	1	0
1	0	0	1
1	1	x	0

K-Map

$Q_{n+1} \backslash SR$	00	01	11	10
$Q_n = 0$	0 ⁰	0 ¹	x ³	1 ²
$Q_n = 1$	1 ⁴	0 ⁵	x ⁷	1 ⁶

$$Q_{n+1} = S + Q_n \cdot \overline{R}$$

J-K Flip Flop (Jack-Killy)



clk	J	K	Q _n	Q _{n+1}
0	x	x	0	0
1	0	0	0	0
1	0	1	0	0
1	1	0	0	1
1	1	1	0	1

$$J \cdot 1 \cdot \bar{Q} = \bar{Q}$$

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

i) (1,1) এর ক্ষেত্রে

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

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

কমপ্লিমেন্ট (Toggle)

Q	Q _n	Q _{n+1}
0	0	1
1	1	0

ii) $J \Rightarrow \bar{Q} \cdot \bar{Q} = 0 - 1 = 1$

এই 1 এর $\bar{Q}$ (Nand) = 0

কমপ্লিমেন্ট, এই 0 এর J এর Q (Nand)

$$\bar{Q} \cdot Q = Q$$

∴ একটি কমপ্লিমেন্ট (Toggle)

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

K-Map

JK	00	01	11	10
$\overline{Q_n}$ 0	0	0	1	1
Q_n 1	1	0	0	1

$$Q_{n+1} = \overline{Q_n} \cdot J + Q_n \cdot \overline{K}$$

For J

Q_{n+1}	0	1
$\overline{Q_n}$ 0	0	1
Q_n 1	X	X

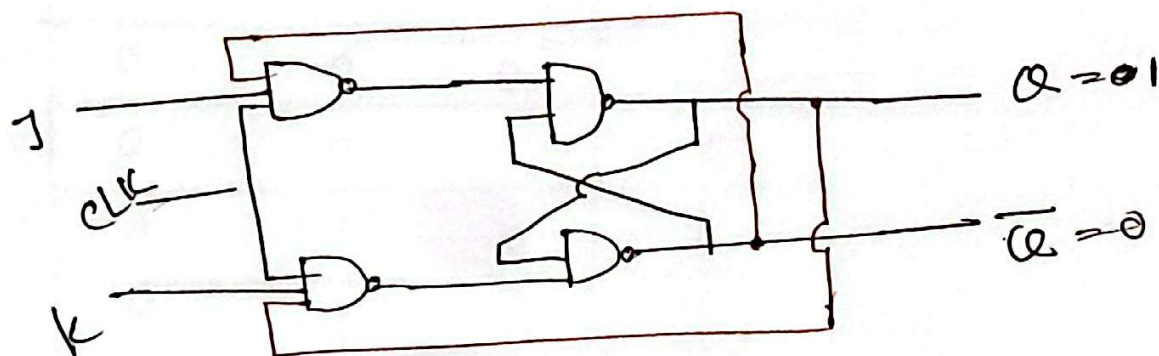
For K

Q_{n+1}	0	1
$\overline{Q_n}$ 0	X	X
Q_n 1	1	0

$$J = Q_{n+1}$$

$$K = Q_{n+1}$$

Race Around Condition



(1,1) case race

First $Q=1, \bar{Q}=0$

2nd Input

J, K, CLK = 1

CLK	J	K	Q_{n+1}	$\bar{Q}_{n+1}$
0	x	x	Q_n	$\bar{Q}_n$
1	0	0	Q_n	$\bar{Q}_n$
1	0	1	0	1
1	1	0	1	0
1	1	1	?	?

		Q	Next Q, $\bar{Q}$	
			Q	$\bar{Q}$
1st	1x1x1	1	1	0
	1x1x0	0	0	1
	1x1x1	0	0	1
2nd	1x1x0	0	1	0
	1x1x1	1	0	1
	1x1x0	1	0	1
3rd	1x1x0	1	0	1
	1x1x1	0	0	1
	1x1x1	0	0	1

In a JK ff, a race around condition occurs when the flip flop output toggles continuously within a single clock pulse, leading to an unpredictable final state.

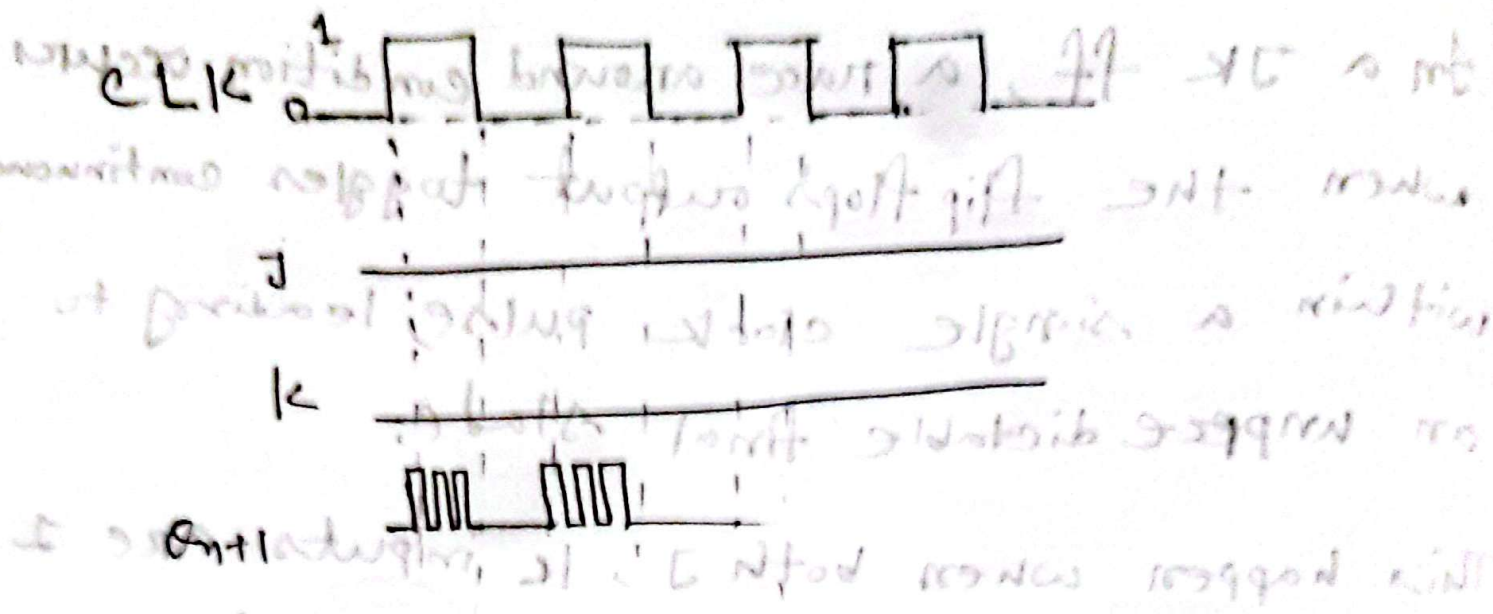
This happens when both J, K inputs are 1 and the clock pulse is held high for a duration longer than the flip-flop's propagation delay.

Ex: 1 suppose initially $Q = 0$

If $J = 1$, $K = 1$ and $CLK = 1$ (for long time)

- Q toggles $\rightarrow$ becomes 1
- Then toggles again $\rightarrow$ becomes 0
- Again $\rightarrow$ 1
- Again $\rightarrow$ 0
- This continues rapidly...

We don't know what the output will be at the end of the clock pulse. That's the problem.



Race Around condition

Avoid condition

- i) Level Triggered
- ii) $J=1, K=1$ & $CLK=1$
- iii) $\frac{T}{2} >$ propagation Delay

- i) Edge Trigger
- ii) propagation delay $> \frac{T}{2}$
- iii) Master-Slave FF

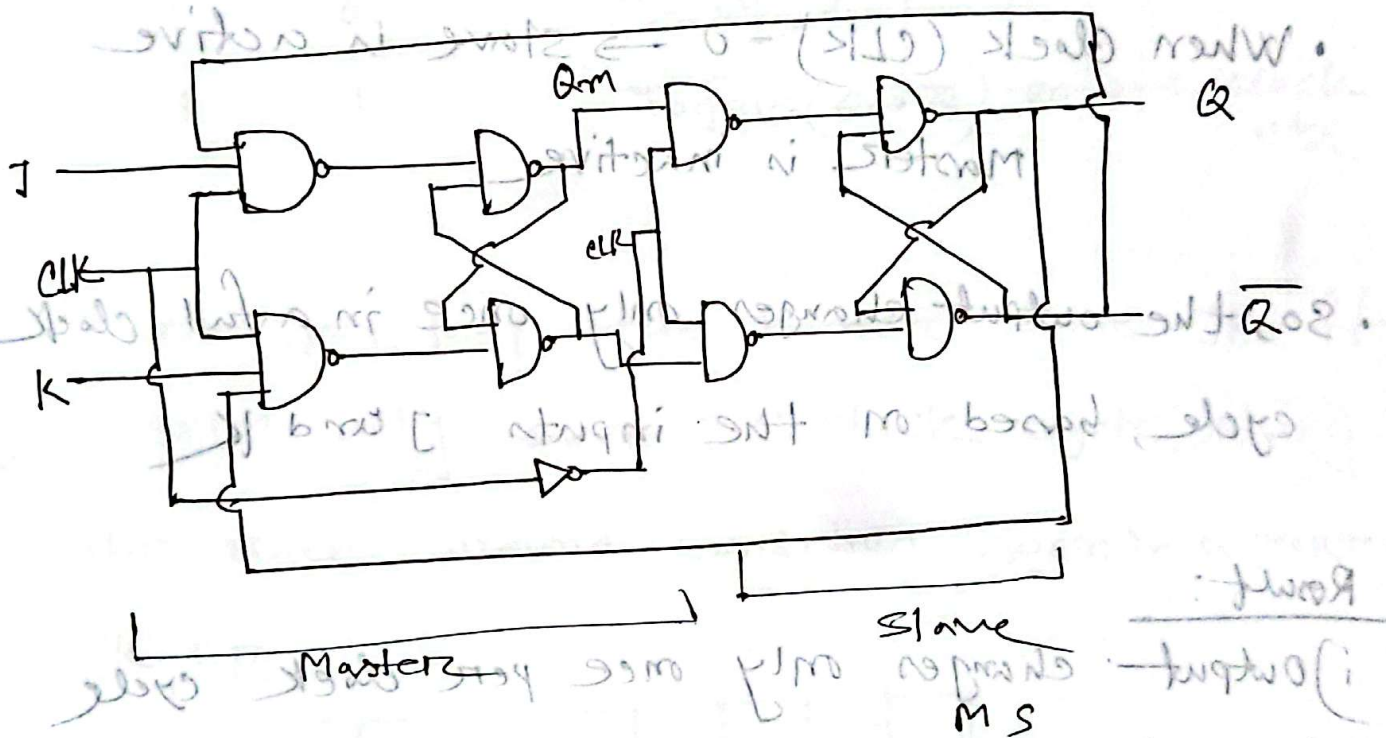
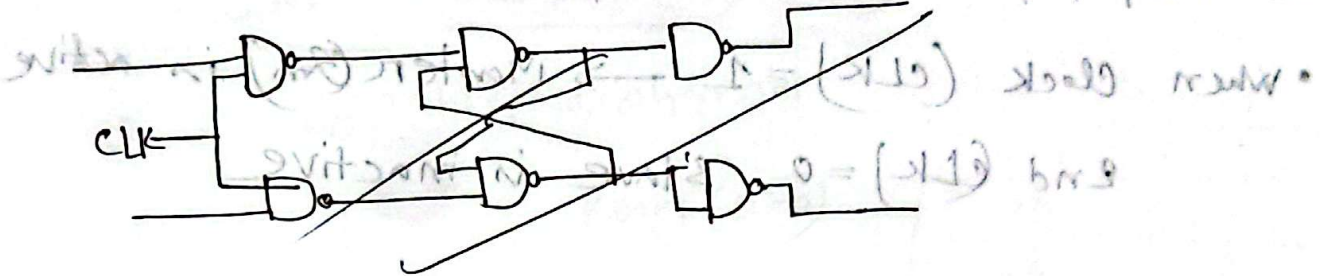
(unit and not) $J=1$ and $CLK=1$ (for long time)

0 toggles $\rightarrow$ becomes 1
 1 toggles again $\rightarrow$ becomes 0
 0 $\rightarrow$ 1
 1 $\rightarrow$ 0
 This continues rapidly.

We don't know what the output will be at the end of the clock pulse that's the problem.

Master slave FF

The flip-flop works in two phases:



When, $J=1, K=1, CLK=1$

CLK	J	K	Q_{n+1} ($\overline{Q}_{n+1}$)
1	0	0	Q_n ($\overline{Q}_n$)
1	0	1	0 (1)
1	1	0	1 (0)
1	1	1	$\overline{Q}_n$ (Q_n)

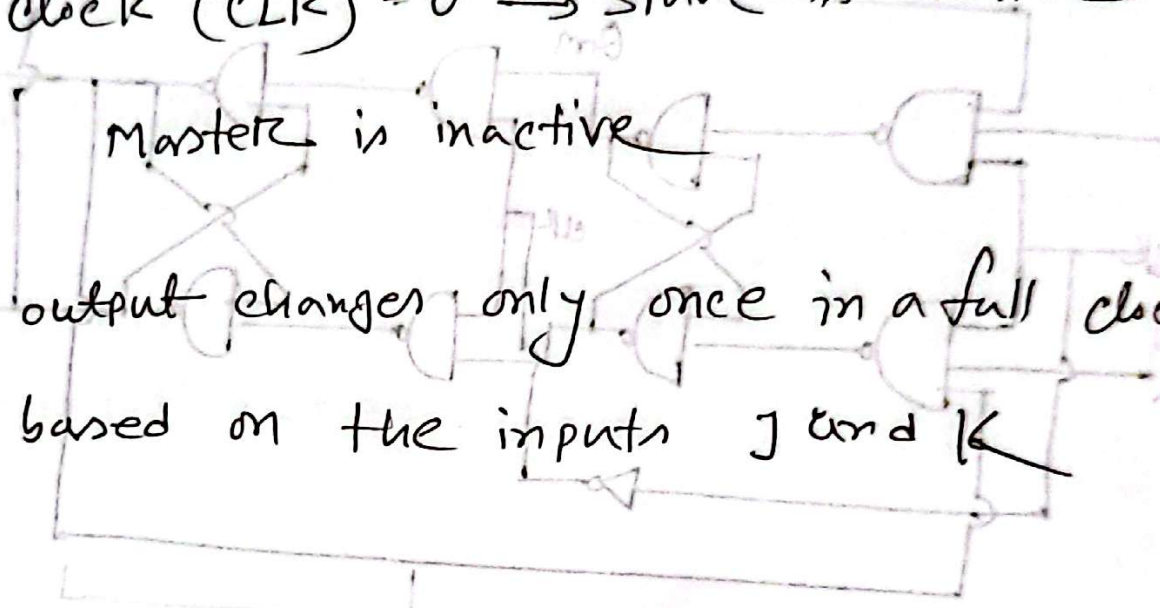
Working Principle:

The flip-flop works in two phases:

- When clock (CLK) = 1 $\rightarrow$ Master (M) is active
and (CLK) = 0, slave is inactive

- When clock (CLK) = 0 $\rightarrow$ slave is active
Master is inactive

- So, the output changes only once in a full clock cycle, based on the inputs J and K



Result:

- Output changes only once per clock cycle
- No rapid toggling
- Race around is solved

Advantages:

- Removes race around condition
- More stable output
- Used in Synchronous circuits

Operation Table

D FF (DQM)

D (Data or Delay) FF in words for states J and k

J k output (Q)

0 0 No change

0 1 Reset (Q=0)

1 0 Set (Q=1)

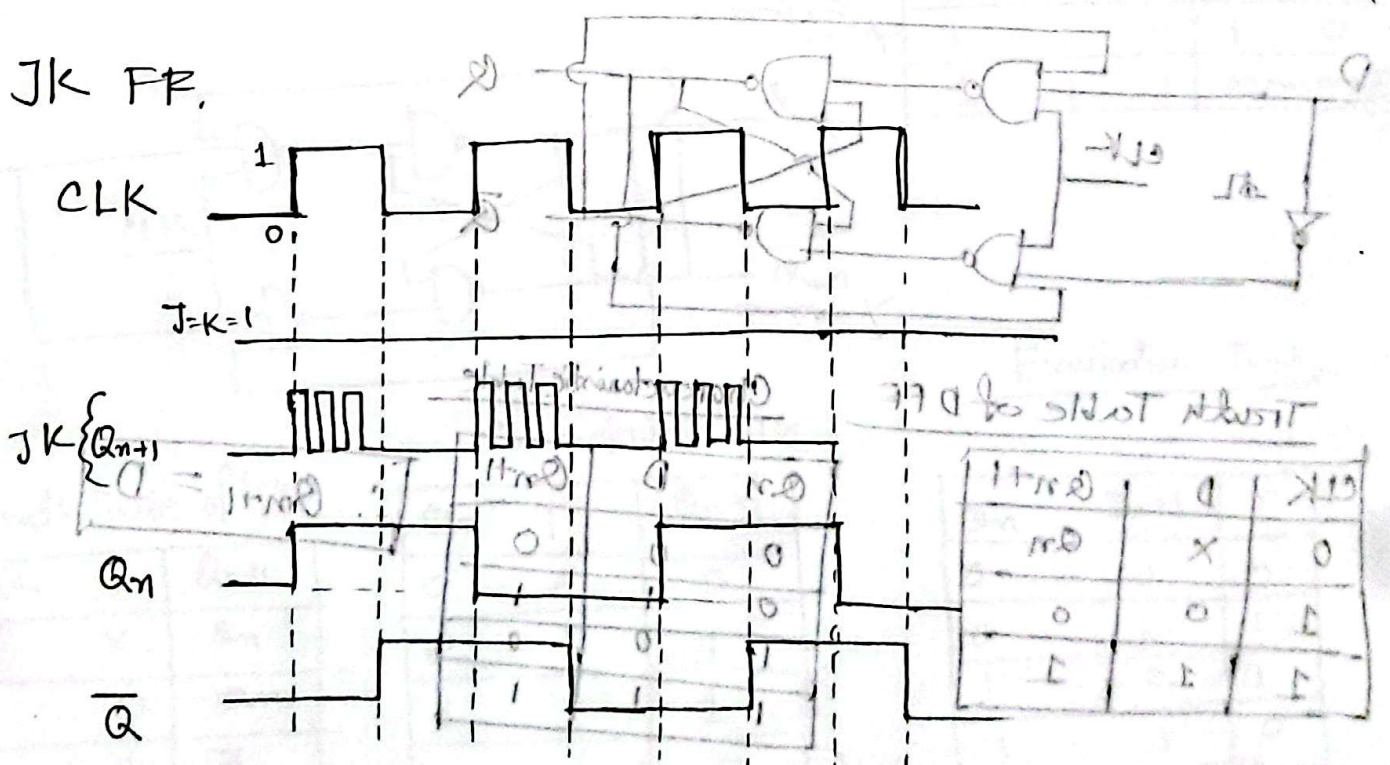
1 1 Toggle (Q $\leftrightarrow$ Q) on one clock edge

SR Truth Table:

SR	Q	Q'
0 0	0	1
0 1	0	1
1 0	1	0
1 1	1	0

When J=1, K=1, the Master-Slave JK FF toggles only once per clock cycle, solving the race around condition seen in a normal JK FF.

the race around condition seen in a normal JK FF.



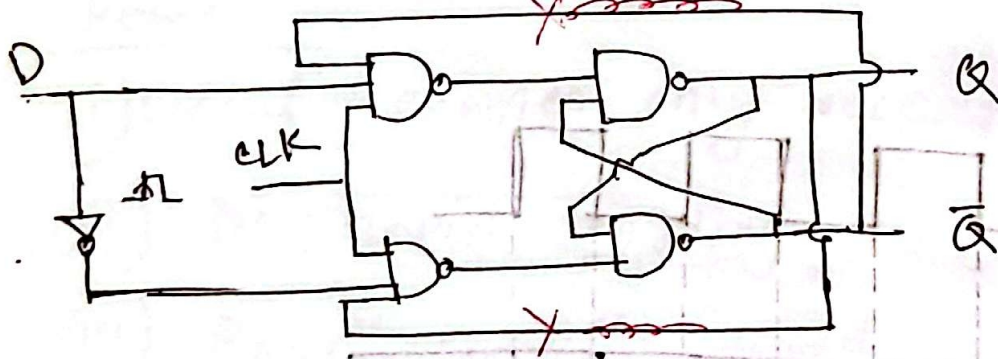
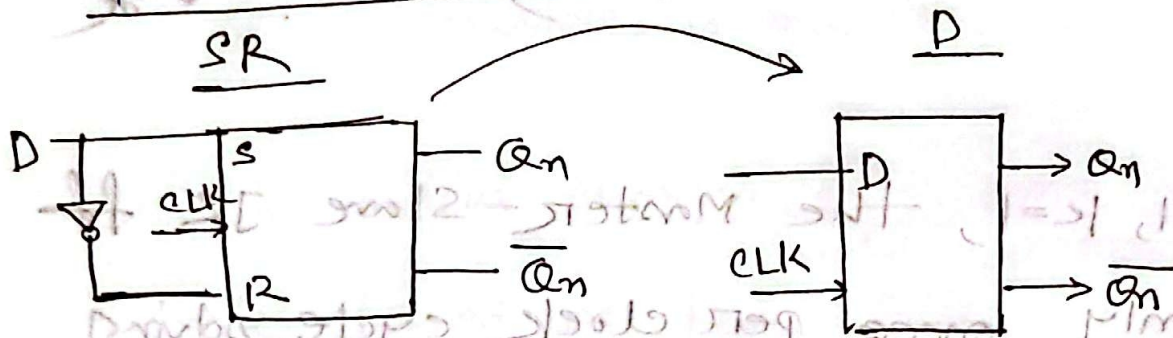
D FF (Delay)

D (Data or Delay FF) is used to store 1-bit data

SR Truth Table:

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	Invalid

SR FF $\rightarrow$ D FF



Truth Table of D FF

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

Characteristic Table

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

$$\therefore Q_{n+1} = D$$

Excitation Table:

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

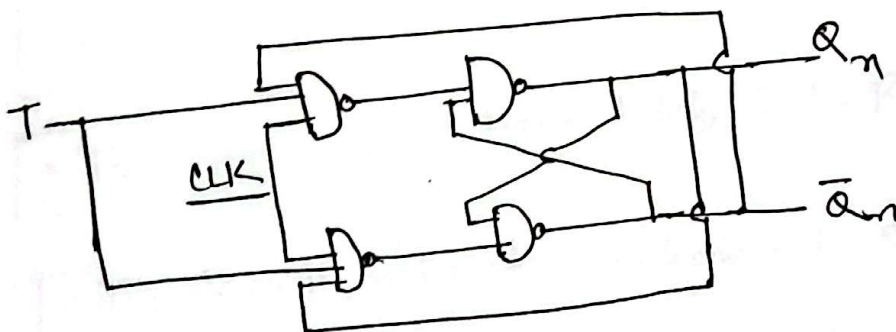
$$0 \cdot 1 + 1 \cdot 0 + 1 \cdot 1 \cdot 0 = 1$$

$$[1 \cdot 0 + 0] =$$

T (Toggle) FF

JK FF को T ऑन 20

जबकि जब Toggle input मिले



TT of JK

CLK	J	K	Q_n	$\bar{Q}_n$
0	x	x	Memory(Q_n)	
1	0	0	Memory	
1	0	1	0	1
1	1	0	1	0
1	1	1	Memory(Q_n)	

Excitation Table

Truth Table of TFF

CLK	T	Q_{n+1}
0	x	Q_n
1	0	Q_n
1	1	$\bar{Q}_n$

Characteristic Table

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

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

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

C.T @ K-Map

Q_n	$\overline{T}_0$	T_1
$Q_n = 0$	0	1
$Q_n = 1$	1	0

$$Q_{n+1} = \overline{Q}_n \cdot T + Q_n \cdot \overline{T}$$

$$= Q_n \oplus T$$

E.T @

Q_n	Q_{n+1}	$\overline{Q}_{n+1}$	Q_{n+1}
$Q_n = 0$	0	1	0
$Q_n = 1$	1	0	1

$$T = Q_n \cdot Q_{n+1} + Q_{n+1} \cdot \overline{Q}_n$$

$$= [Q_n \oplus Q_{n+1}]$$

FF (SR) T

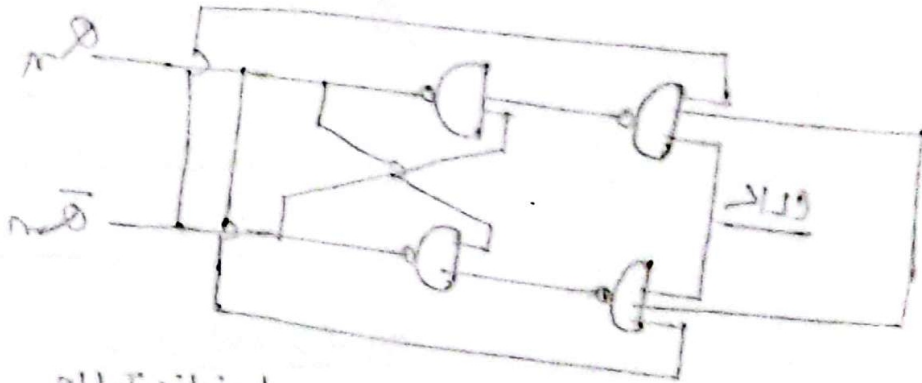
JK FF

Q_n	$\overline{Q}_n$	J	K	Q_{n+1}
0	1	0	0	0
1	0	0	1	0
0	1	1	0	1
1	0	1	1	0



JK FF T (SR) FF

Logic diagram of JK flip-flop



Excitation Table

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

Characteristic Table

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

Truth Table of DFF

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

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

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

* How a Nand latch circuit can be used to store a bit?

A Nand latch is a basic memory element formed using two cross-coupled NAND gates. It is also known as an SR (Set-Reset) latch. This circuit can store one bit (0 or 1) by maintaining a stable output state. It has two inputs: S (Set) and R (Reset) and two outputs: Q and $\bar{Q}$.

Explanation:

• When $S=0, R=1$, the output $Q=1$ $\Rightarrow$ The latch stores 1 (It means low input is applied to S and high is applied to R).

• To store 0 in Nand latch, $S=1, R=0$ (It means high input is applied to S and low to R).

• To keep output unchanged $S=1, R=1$ (It means high input is applied to S and R) Holds in previous state.

• When $S=0, R=0$

• It keeps the input value until the inputs change. So, it works like a 1-bit memory cell.

* What is the limitation of SR flip-flop?
Explain how the limitation can be resolved.

Limitation of SR FF:

- i) The major limitation occurs when $S=1, R=1$ at the same time.
- ii) In this case, both outputs Q and $\bar{Q}$ become 0 which is logically incorrect, because Q and $\bar{Q}$ are supposed to be complements.
- iii) This condition is known as the invalid state or indeterminate state / Forbidden state.
- iv) It creates uncertainty in the output and can lead to unstable operation in circuits.

Can be Resolved:

The limitation can be resolved by using a modified version of the SR FF called the JK FF.

- i) The JK FF has two inputs, J, K , similar to SR.
- ii) The JK FF addresses this issue by introducing feedback.
When $J=1$ and $K=1$ (both are high), the JK FF

toggle (changes from 0 to 1 or 1 to 0) instead of entering an invalid state

toggle (changes from 0 to 1 or 1 to 0), instead of going into an invalid state.

ii) Thus, JK FF removes the invalid state of SR FF

This makes the JK FF more versatile and reliable in applications where simultaneous set and reset conditions are possible.

* What is the limitation of JK FF? Draw a circuit diagram and timing diagram and its truth table which will eliminate its limitation.

The JK FF is an improvement over the SR FF. It eliminates the invalid state by allowing toggling when $J=1$, $K=1$.

Limitation:

- When $J=1$, $K=1$, the output toggles.
- If the clock pulse is high for a long time, the output toggles continuously (multiple times).
- This condition is known as the Race Around Condition.

iii) It creates unpredictable output and may lead to logic errors.

Race & Propagation delay

Solution:

To eliminate this problem, we use a Master-slave JK FF.

How it works:

i) The circuit uses two JK FF in series

- Master stage: triggered by the positive edge of the clock.

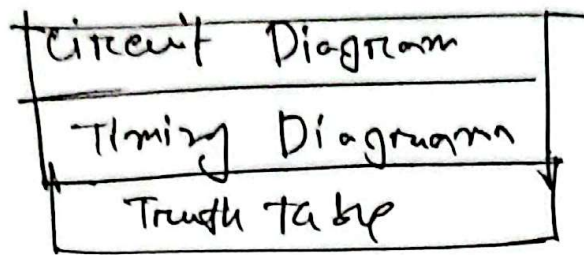
- Slave stage: triggered by the negative edge (inverted clock)

ii) When the clock is High: master is active slave

is inactive.

iii) When the clock is Low: master is inactive slave transfers the output.

iv) This ensures that output changes only once per clock pulse, avoiding the race condition.

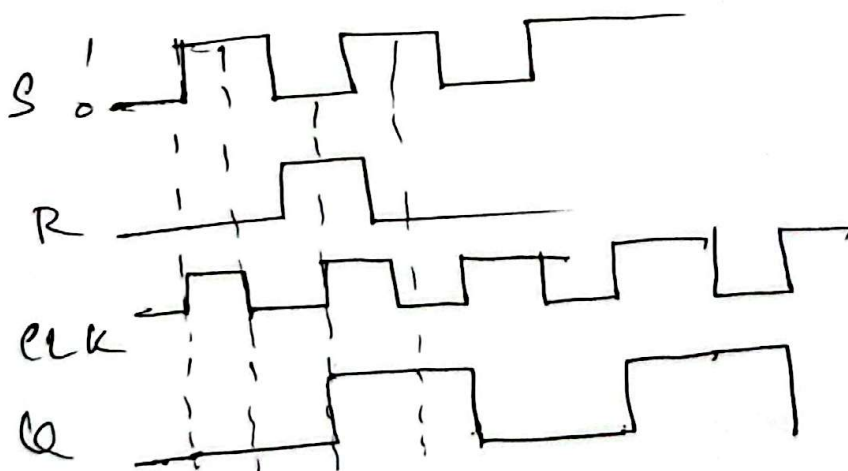


* In JK FF, if $J=K=1$ and if $clk=1$ for a long period of time, then Q output will toggle as long as clk is high, which makes the output of the flip-flop unstable or uncertain. This problem is called race around condition in J-K FF.

* Explain the synchronous/clocked S-R FF with the wave forms.



is a type of SR FF where the output change only when the clock signal is active



* Design and Explain the working principle of an

8 bit parallel adder.

An 8-bit parallel adder is a digital circuit that adds two 8-bit binary numbers and produces an 8 bit sum and a carry-out (C_o).

It is built using full adders, which can add three 1-bit binary numbers: Two input and a carry-in.

or, An 8-bit parallel adder adds two 8-bit binary numbers simultaneously using 8 full adder circuits.

Input: i) The adder takes two 8-bit binary numbers as input. These are usually called A and B.

ii) Each bit of A is labeled: $A_0, A_1, A_2, \dots, A_7$

iii) " " " B " " : $B_0, B_1, B_2, \dots, B_7$

Working Principle

i) The LSBs A_0 and B_0 are added with an initial carry-in ($C_i = 0$) using

Addition Process:

- i) Each pair of bits (A_i and B_i) is added together.
- ii) Along with that, a carry (C_{i-1}) from the previous bit is also added

The carry (C_{i-1}) = 0 for the LSB

- iii) This process is done using a circuit called a Full Adder, which takes three inputs

A_i, B_i, C_{i-1}

The Full Adder gives two outputs:

Sum

Carry (which goes to the next bit position)

Output:

The final result is an 8-bit sum,

$$S_i = S_7, S_6, S_5, S_4, S_3, S_2, S_1, S_0$$

There may also be a final carry-out from the last Full Adder, which can be used if needed:

Working Principle:

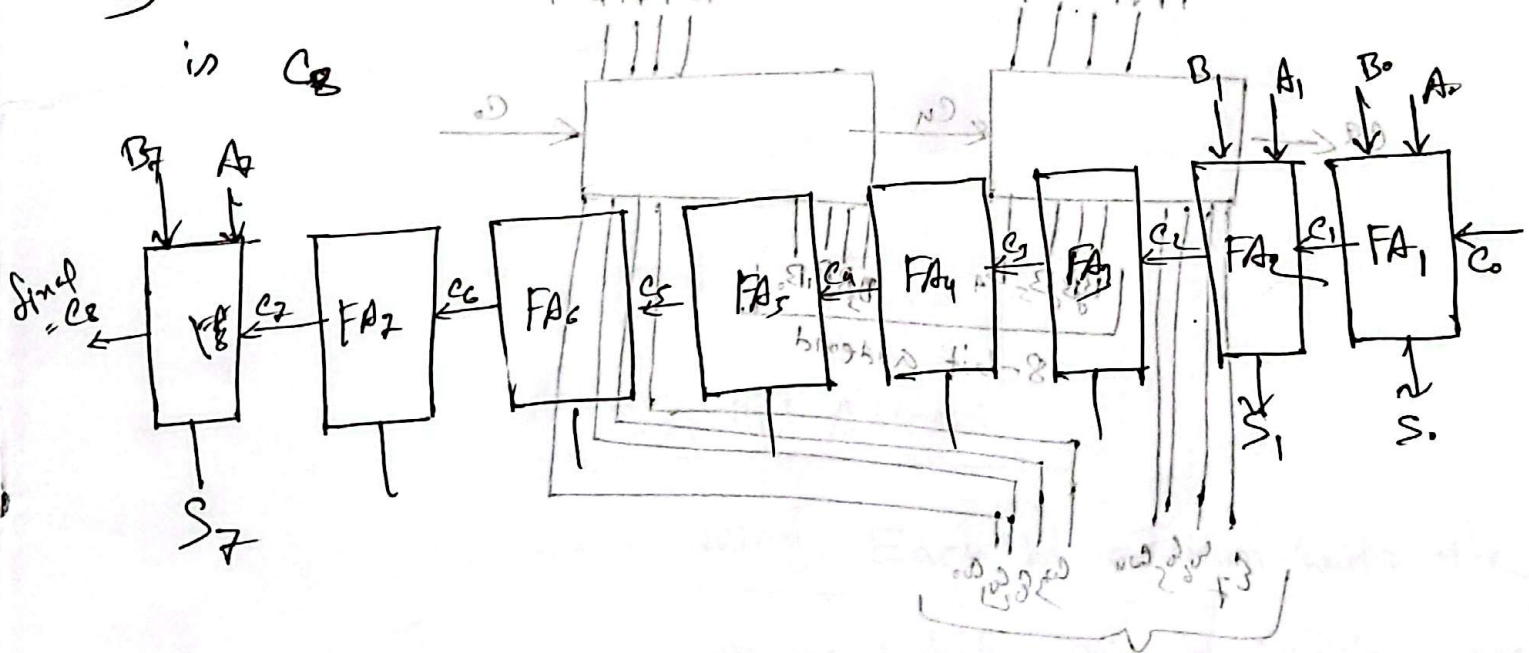
i) The LSBs A_0 and B_0 are added with an initial carry-in ($C=0$) using FA.

ii) The sum S_0 is generated and a carry C_1 is forwarded to FA1.

iii) FA1 adds A_1, B_1, C_1 to generate S_1 and C_2 .

iv) This ripple-carry addition continues up to FA₇.

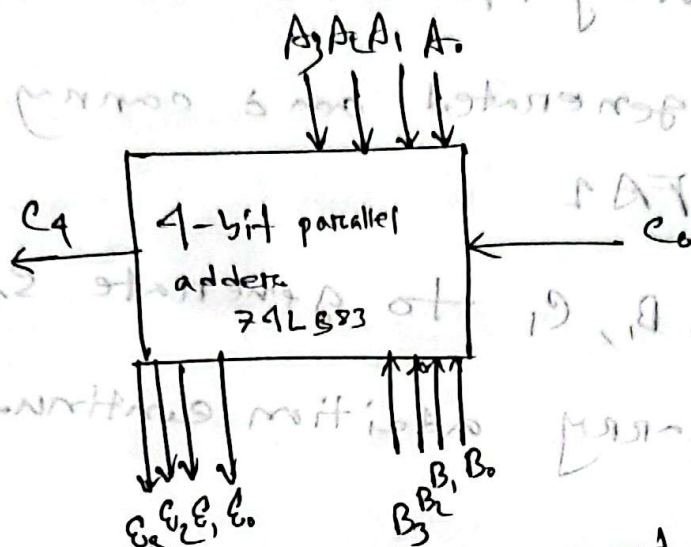
v) The final sum is ($S_0 - S_7$) and final carry-out is C_8 .



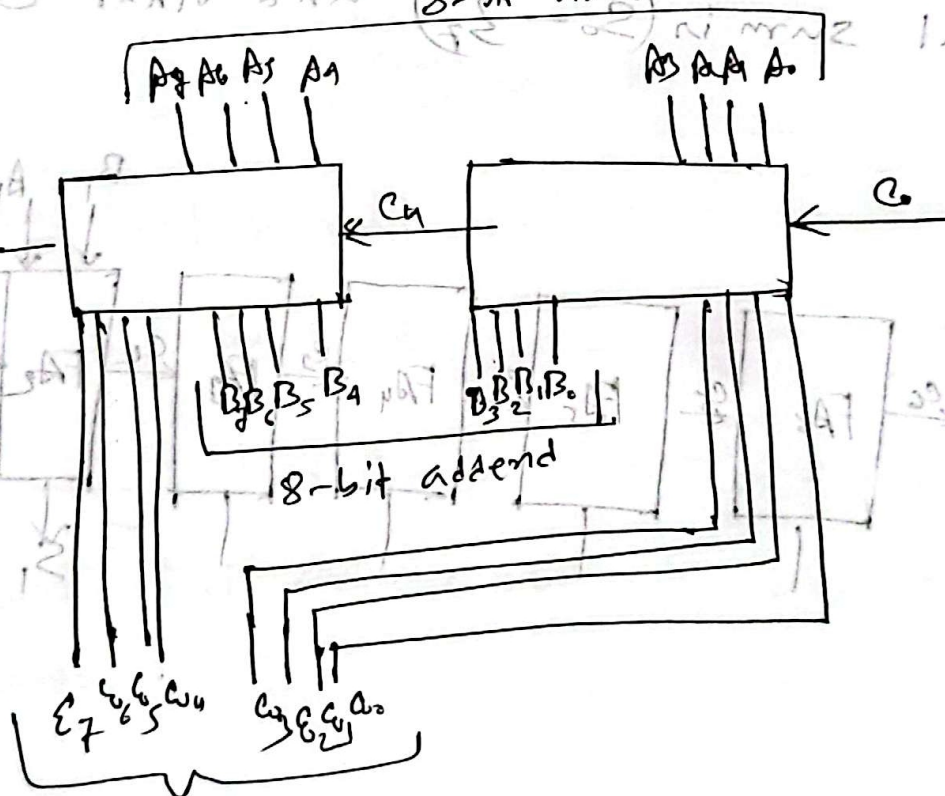
* Using 74LS83 IC's draw a 4/8/12/16

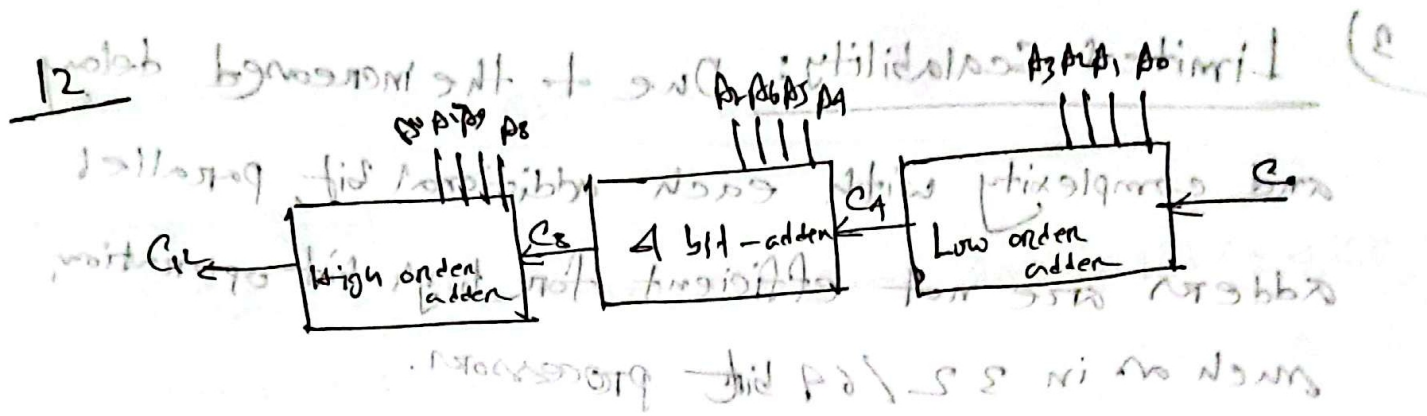
parallel adder

4



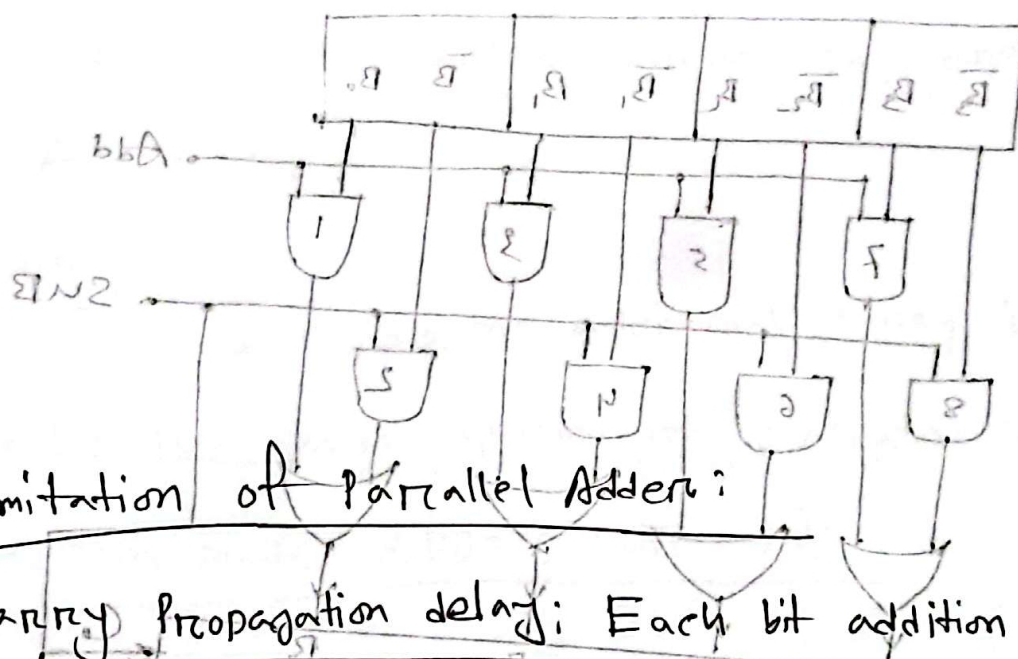
8-bit addend





* Draw a parallel adder/subtractor using 2's complement system and explain its operation.

16



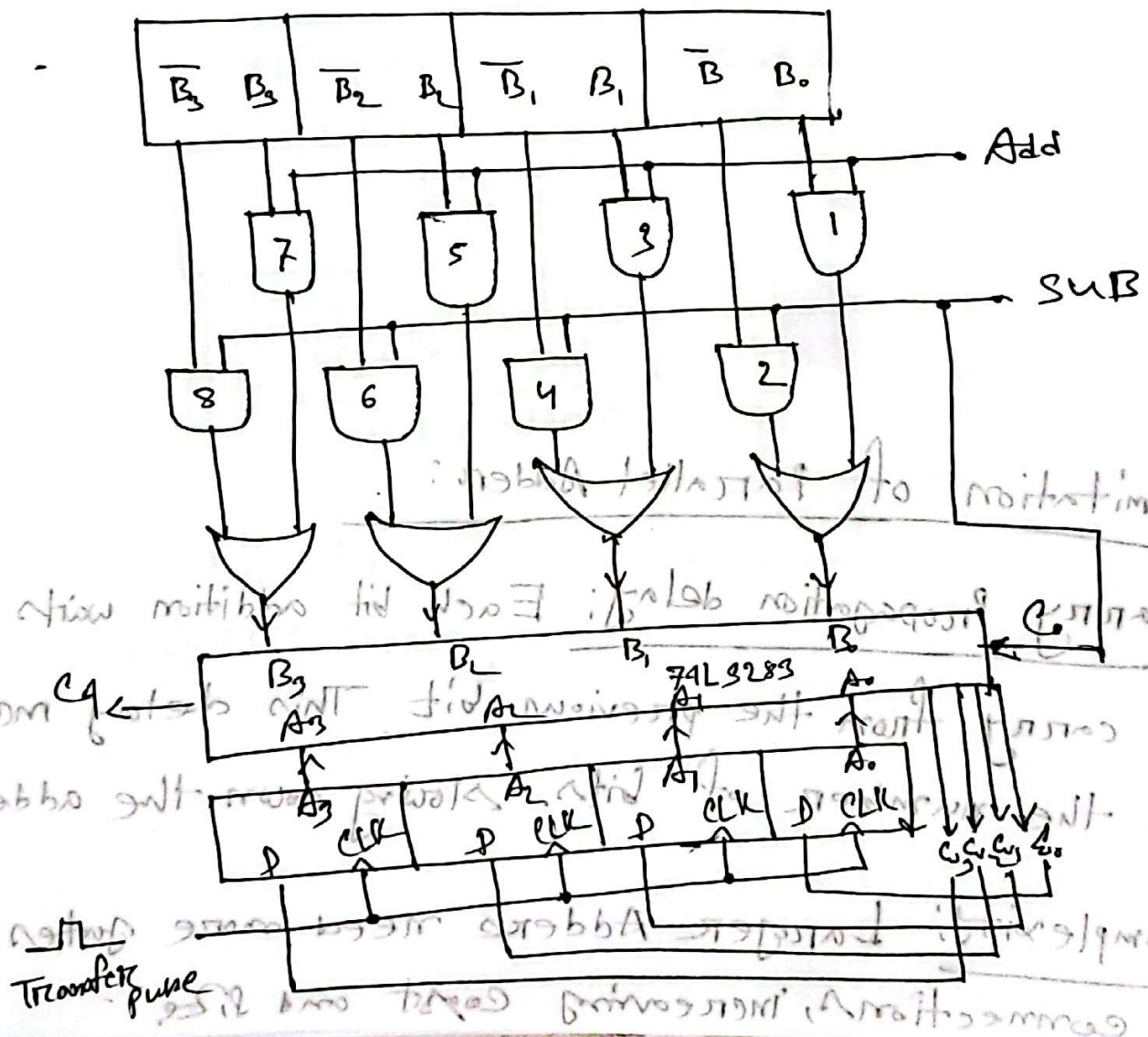
* Limitation of Parallel Adder:

- i) Carry Propagation delay: Each bit addition waits for the carry from the previous bit. This delay increases with the number of bits, slowing down the adder.
- ii) Complexity: Larger Adders need more gates and connections, increasing cost and size.

3)

Limited Scalability: Due to the increased delay and complexity with each additional bit, parallel adders are not efficient for high-bit operations, such as in 32/64 bit processors.

* Draw a parallel adder/subtractor using 2's complement system and explain its operation.



Principal

Addition Mode ($ADD=1, SUB=0$):

i) Control Signal:

i) $SUB=0$: This turns OFF some AND gates (2, 4, 6, 8) their input is 0.

ii) $ADD=1$: This turns ON the other AND gates.

(1, 3, 5, 7) → Passes B_0 to B_3 values.

operation:

The B register values (B_0 to B_3) pass through OR gates to the adder, where they are added to A_0 to A_3 .

Carry-In:

$C_0=0$, So, no additional carry is added.

output: The sum appears at outputs S_0 to S_3 .

Subtraction Mode ($ADD=0, SUB=1$):

Control signal:

- $ADD=0$: Turn OFF (1, 3, 5, 7)
- $SUB=1$: Turn ON (2, 4, 6, 8) → Passes inverted B_0 to B_3 values.

operation:

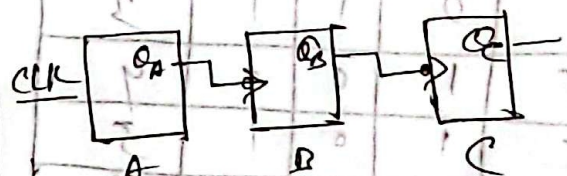
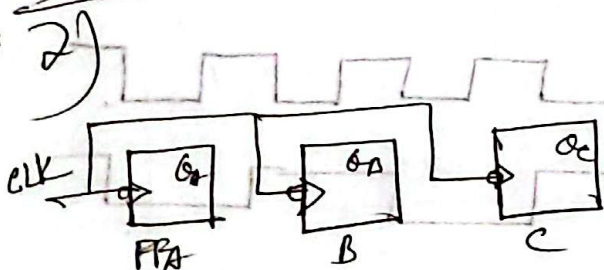
- i) The B values are inverted (1's complement) and passed to the adder
- ii) At the same time, $C_0 = 1$ is added $\rightarrow$ this makes it's 2's complements of B
- iii) So the circuit performs $A + (2's \text{ complement of } B) = A - B$

Output: The result $A - B$ is shown at outputs S_0 to S_3 .

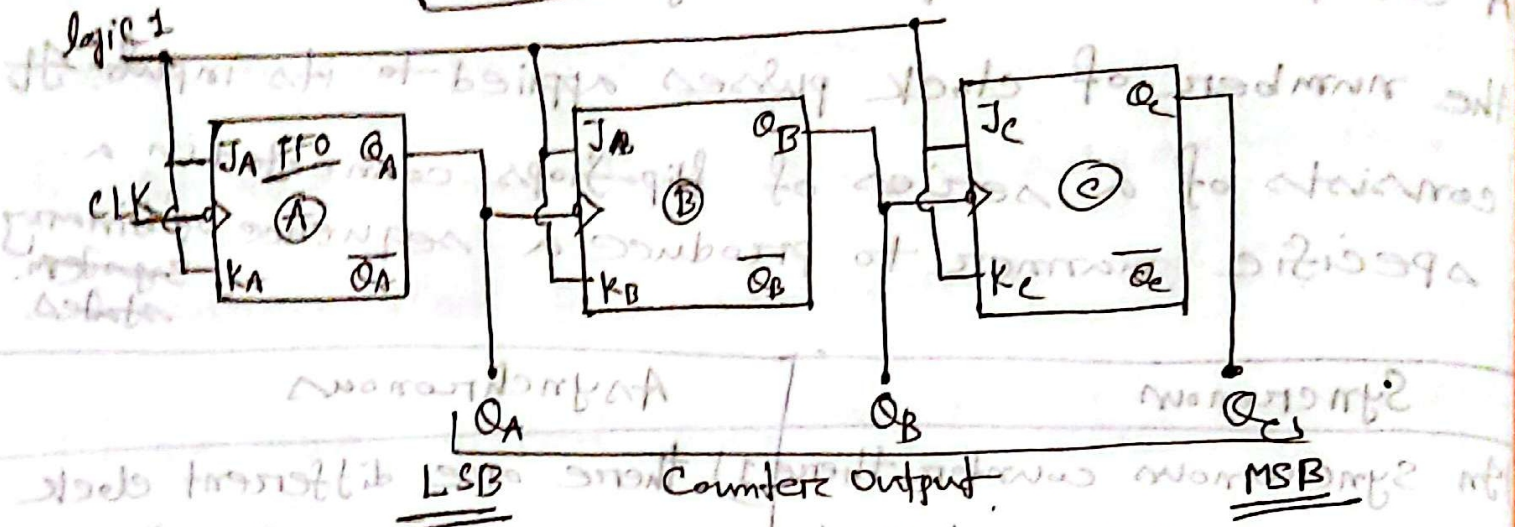
Counter

A counter is a sequential digital circuit that counts the number of clock pulses applied to its input. It consists of a series of flip-flops connected in a specific manner to produce a sequence of binary states.

Synchronous	Asynchronous
1) In synchronous counters, there are same clock input signal with flip-flops used to produce output	1) There are different clock signals used to produce the output
2) ... Operation is faster	2) ... operation is slower
3) Also known as parallel counter	3) ... Serial counter
4) Design of synchronous is complex	4) ... simple
5) Propagation delay is less	5) ... high
6) Ex: Ring, Johnson counter etc	6) Ex: Ripple up/down counter etc



* 3-bit Asynchronous UP Counter falling edge (-) ↓ Toggle operation (D)

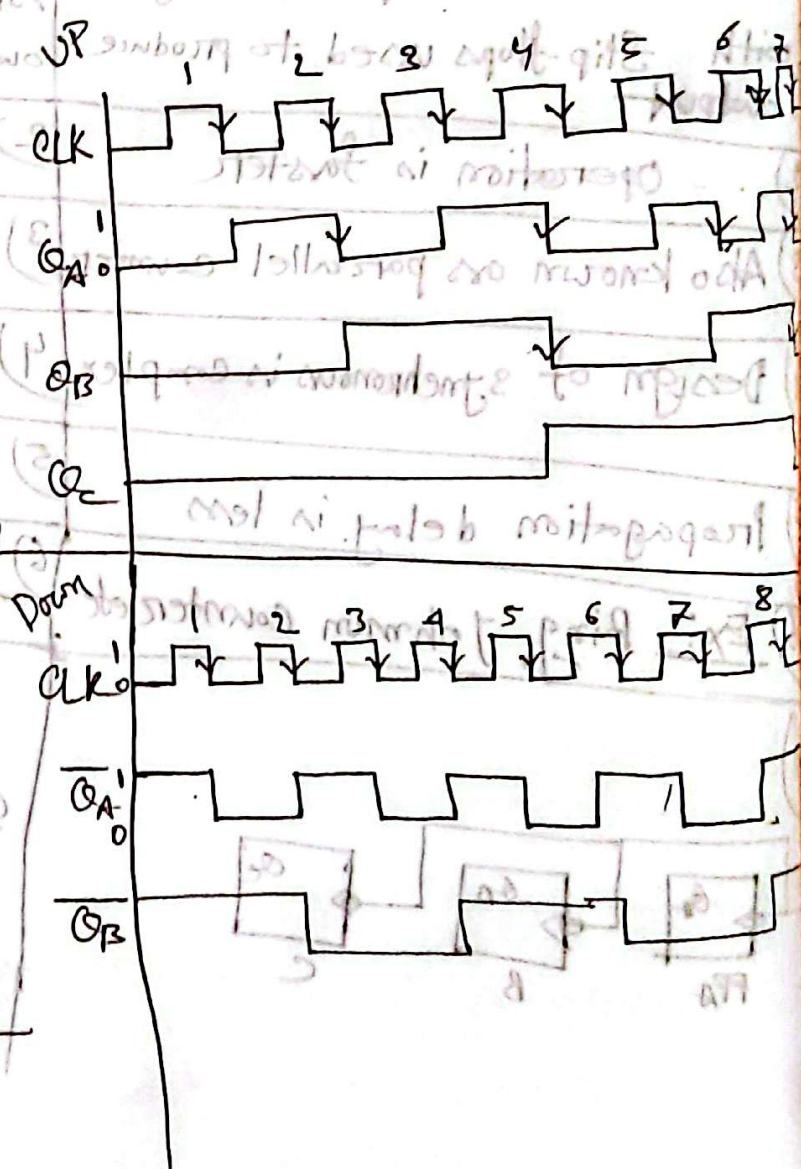


$2^3 = 8$

counter = $(8-1) = 7$

State = 8

	CLK	Q _C	Q _B	Q _A	Decimal
0	1	0	0	0	0
1	2	0	0	1	1
2	3	0	1	0	2
3	4	0	1	1	3
4	5	1	0	0	4
5	6	1	0	1	5
6	7	1	1	0	6
7	8	1	1	1	7
8	9	0	0	0	0



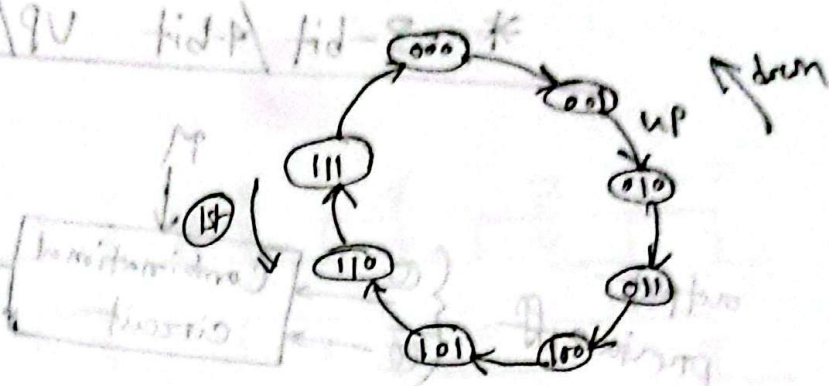
State diagram:

3-bit up counter

$$n=3$$

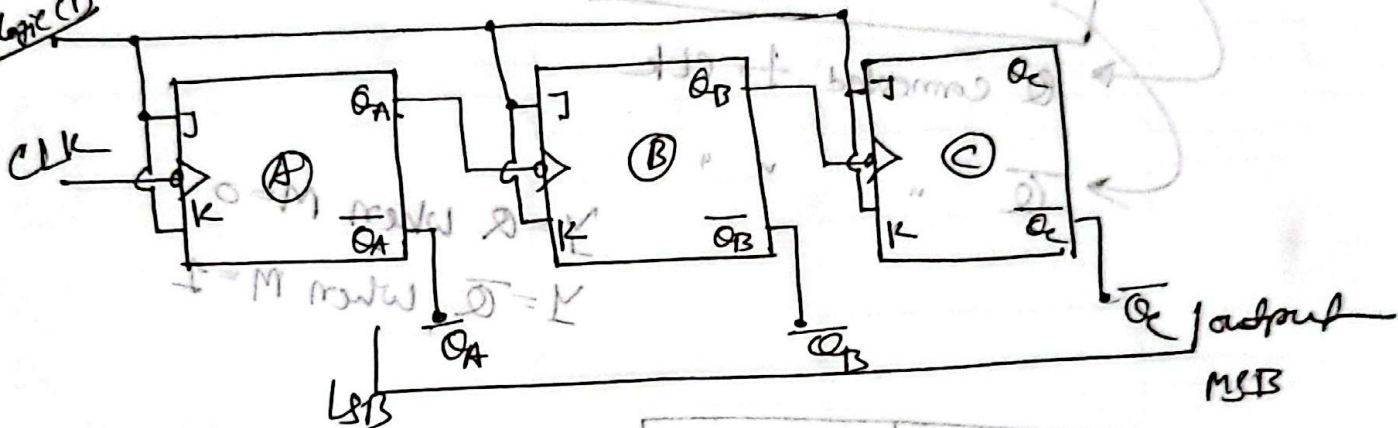
$$\text{no. of state} = 2^n = 8$$

$$\text{no. of count} = 2^n - 1 = 7(111)$$



* Down Counter: (3)

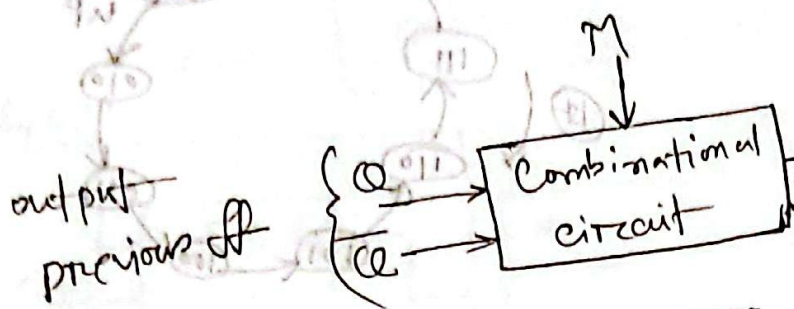
Logic (1)



	UP			DOWN			
CLK	QC	QB	QA	QC	QB	QA	Decimal
Initially	0	0	0	1	1	1	7
1st (A)	0	0	1	1	1	0	6
2nd (A)	0	1	0	1	0	1	5
3rd	0	1	1	1	0	0	4
4th	1	0	0	0	1	1	3
5th	1	0	1	0	1	0	2
6th	1	1	0	0	0	1	1
7th	1	1	1	0	0	0	0

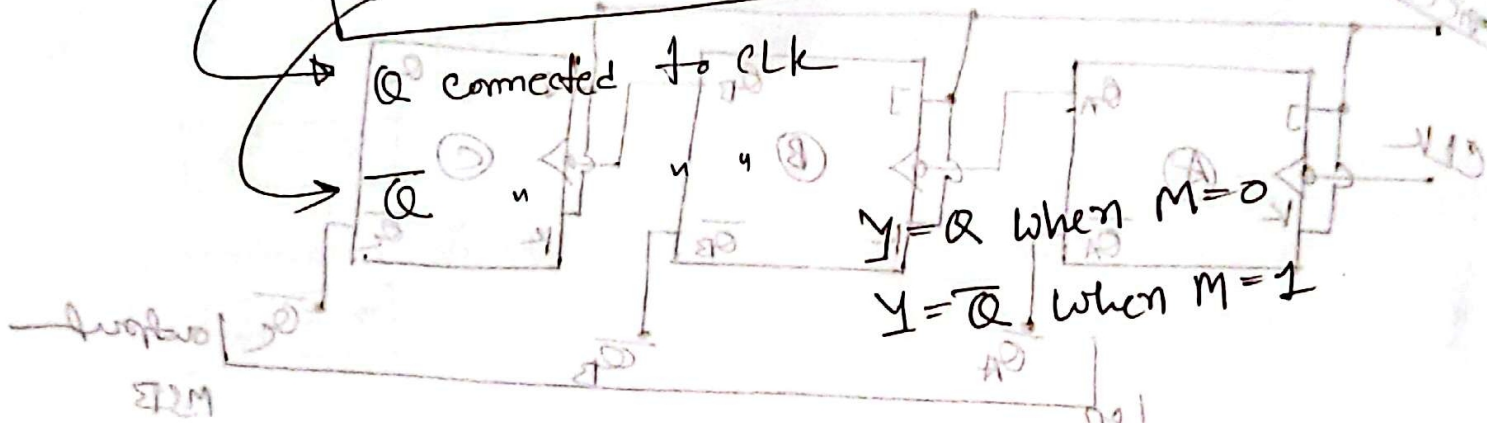
UP/DOWN COUNTER
 $X = \overline{M}A + M\overline{A}$

* 3-bit / 4-bit UP/DOWN Counters

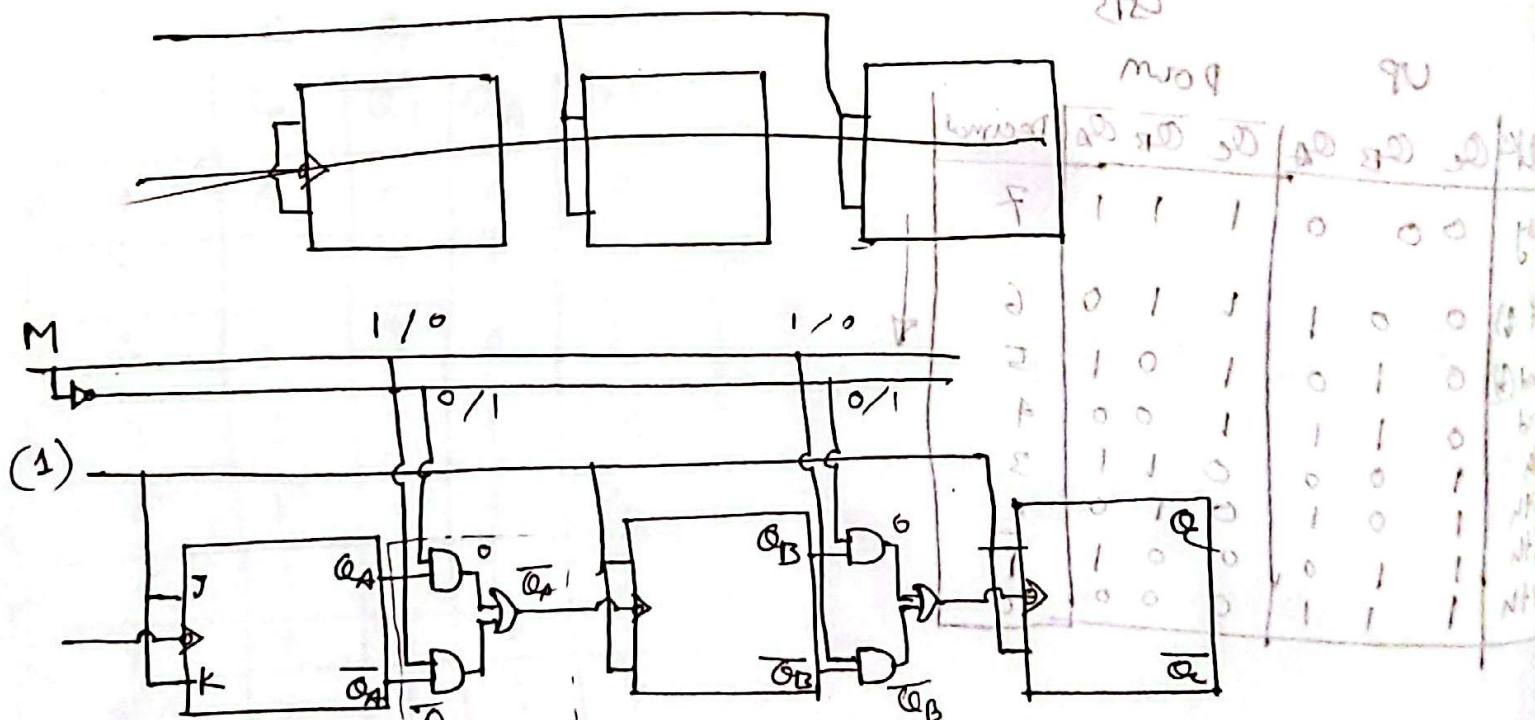


M	Q	Q̄	Y
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

M = 0; UP counting
M = 1; Down counting



$Y = Q$ when $M = 0$
 $Y = \bar{Q}$ when $M = 1$



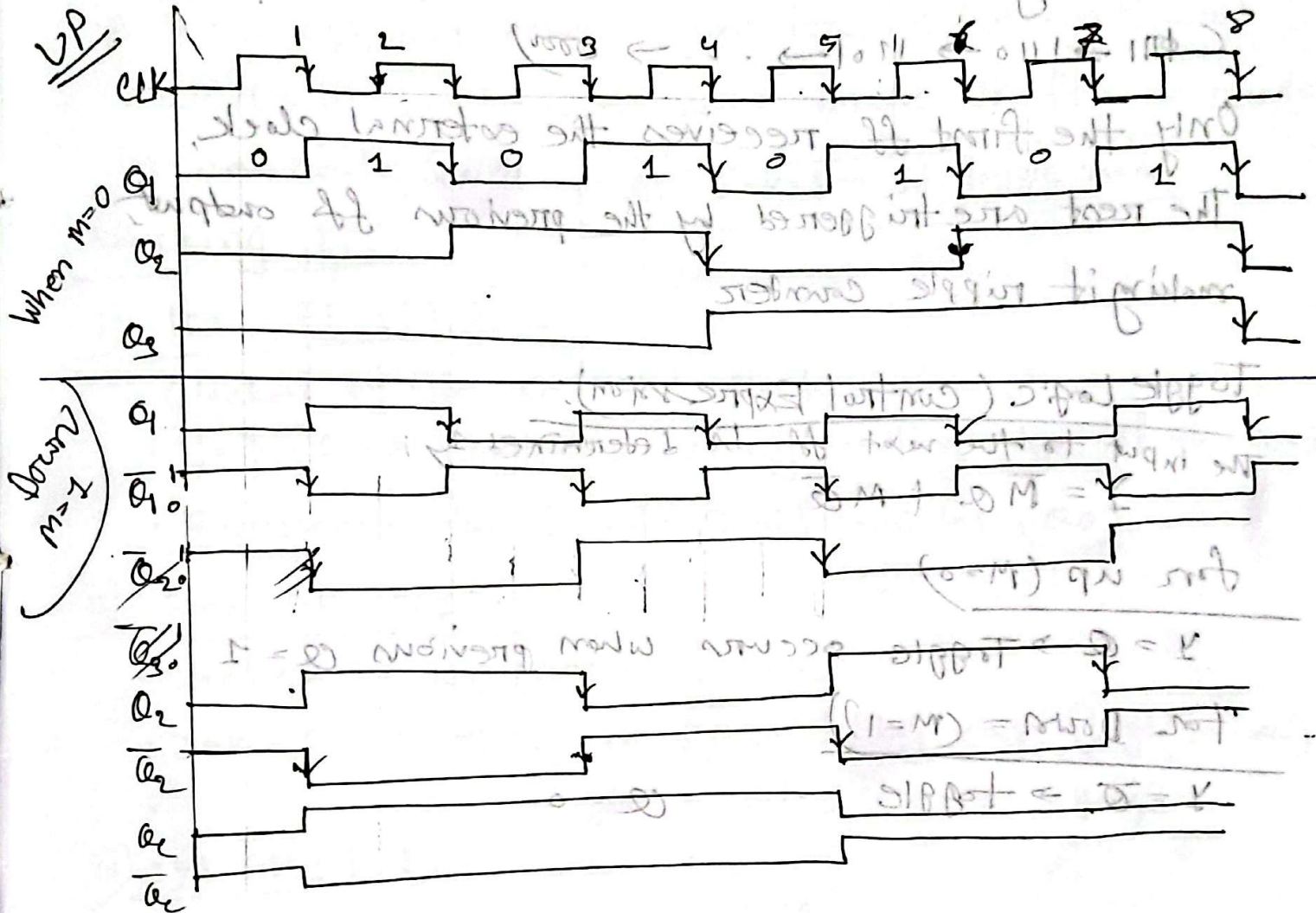
$Y = \bar{M}Q + M\bar{Q}$

UP/DOWN COUNTER

Timing diagram

• if $M = 1 \rightarrow$ Down counting:

The binary value decreases with every clock pulse.



Draw:

Here, ϕ JK ff. (used in toggle mode)

Logic gate AND, OR, NOT

MODE control input M

clock input.

Working

• if $M = 0 \rightarrow$ up counting:

The binary value increases with every clock pulse
 (0000 $\rightarrow$ 0001 $\rightarrow$ 0010 $\rightarrow$... $\rightarrow$ 1111)

• If $M=1 \rightarrow$ Down counting:

The binary value decreases with each clock pulse

($1111 \rightarrow 1110 \rightarrow 1101 \rightarrow \dots \rightarrow 0000$)

Only the first ff receives the external clock,

The rest are triggered by the previous ff output, making it ripple counter

Toggle Logic (Control Expression):

The input to the next ff is determined by:

$$Y = \overline{M}Q + M\overline{Q}$$

for up ($M=0$)

$Y = Q \rightarrow$ Toggle occurs when previous $Q = 1$

For Down ($M=1$)

$Y = \overline{Q} \rightarrow$ toggle $Q = 0$

(When output is 1111) - If it is 1111, then

logic says AND OR NOT

M will be 1 and 0

clock input

if $M=0 \rightarrow$ up counting

The binary value increases with every clock pulse
($0000 \rightarrow 0001 \rightarrow 0010 \rightarrow \dots \rightarrow 1111$)

* 3-bit Asynchronous UP Counter:

A 3-bit asynchronous up counter in binary from 000 to 111 (0 to 7). It uses three JK FFs connected in cascade/series.

All $J = K = 1 \rightarrow$ so toggle mode

Working PP:

i) 1st FF (FF0):

- Clocked by the external clock
- $J = K = 1 \rightarrow$ toggles on every falling edge of the clock
- Represents the Least Significant bit (Q0)

ii) 2nd FF (FF1):

- Clocked by Q0 (output of FF0)
- $J = K = 1 \rightarrow$ toggles on Q0's falling edge (high to low)
- toggles on falling edge of Q0/QA

iii) 3rd FF (FF2):

- Clocked by Q1 (output of FF1)
- $J = K = 1 \rightarrow$ toggles on Q1's falling edge.

From left to right

This is done with a reset logic circuit.

When Q2 = 1, Q1 = 1, Q0 = 1 $\rightarrow$ reset

Decimal	Binary	Q2	Q1	Q0
0	000	0	0	0
1	001	0	0	1
2	010	0	1	0
3	011	0	1	1
4	100	1	0	0
5	101	1	0	1
6	110	1	1	0
7	111	1	1	1

Mod 14: UP

A Mod-14 asynchronous up counter counts from 0000 to 1101 (0 - 13) - a total of 14 states, and then resets to 0000, so, it's a modulus-14 counter.

To count up 14, we need a 4-bit AC. (Since $2^4 = 16$)

Uses 4 JK ff (FF0, FF1, FF2, FF3)

All ff have $J=K=1 \rightarrow$ toggle mode

Reset Logic

$\overline{Q_3} \cdot Q_2 \cdot Q_1 \cdot \overline{Q_0}$

$Q_3 \cdot Q_2 \cdot Q_1$

Clocking:

FF0 is clocked by the external clock

FF1 " " " by Q_0

FF2 " " " by Q_1

FF3 " " " by Q_2

MOD Control:

• Normally, 4 ff count from 0 to 15

• To make it Mod-14, we detect state 1110 (14) and reset all ff at that moment

• This is done using a reset logic circuit

When $Q_3=1, Q_2=1, Q_1=1, Q_0=0 \rightarrow$ active reset

Truth Table

Clock	Q_3	Q_2	Q_1	Q_0	Binary	Decimal
13	1	1	0	1	1101	13
14 Reset	1	1	1	0	1110	Reset to 0000

MOD-14 DOWN: AS ;

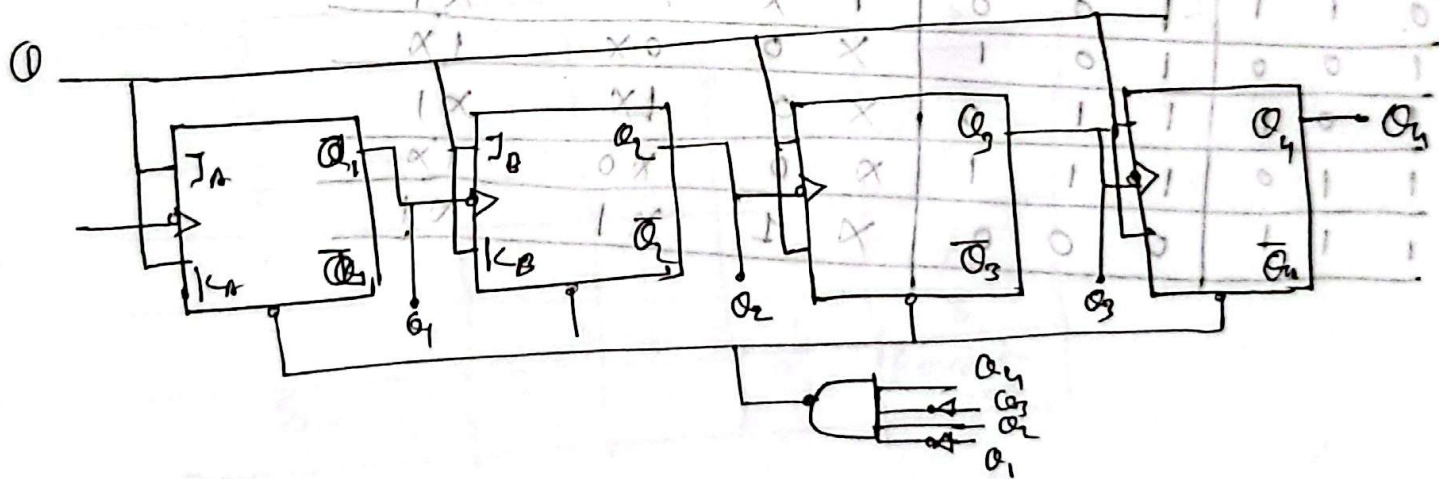
When the output of the counter reaches 0000 (0), the counter starts from 1101 (13). This ensures that the counter counts backward from 13 down to 0 repeatedly, giving a MOD-14 counting sequence.

0	✓	0	0	0
1	✓	1	1	0
1	1	✓	0	1
0	0	✓	1	1

* Decade (MOD-10) :

Reset logic: connect Q_1 AND Q_3 to an AND gate

The AND gate gives logic 1, triggering asynchronous Clear (CLR) input to all TFF, when output = 1010 ($Q_3=1$, $Q_1=1$)



Synchronous Counter

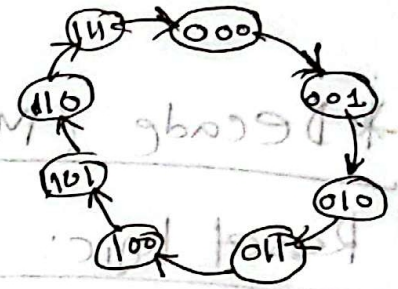
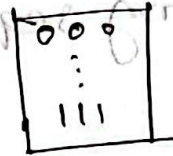
* 3-bit UPD counter

Excitation table of FF:

P.S.	N.S		J	K	T
Q_n	Q_{n+1}				
0	0		0	X	0
0	1		1	X	1
1	0		X	1	1
1	1		X	0	0

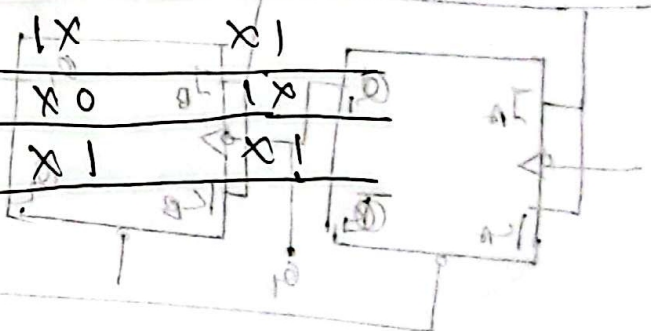
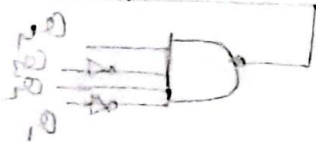
state of $= 2^3 = 8$

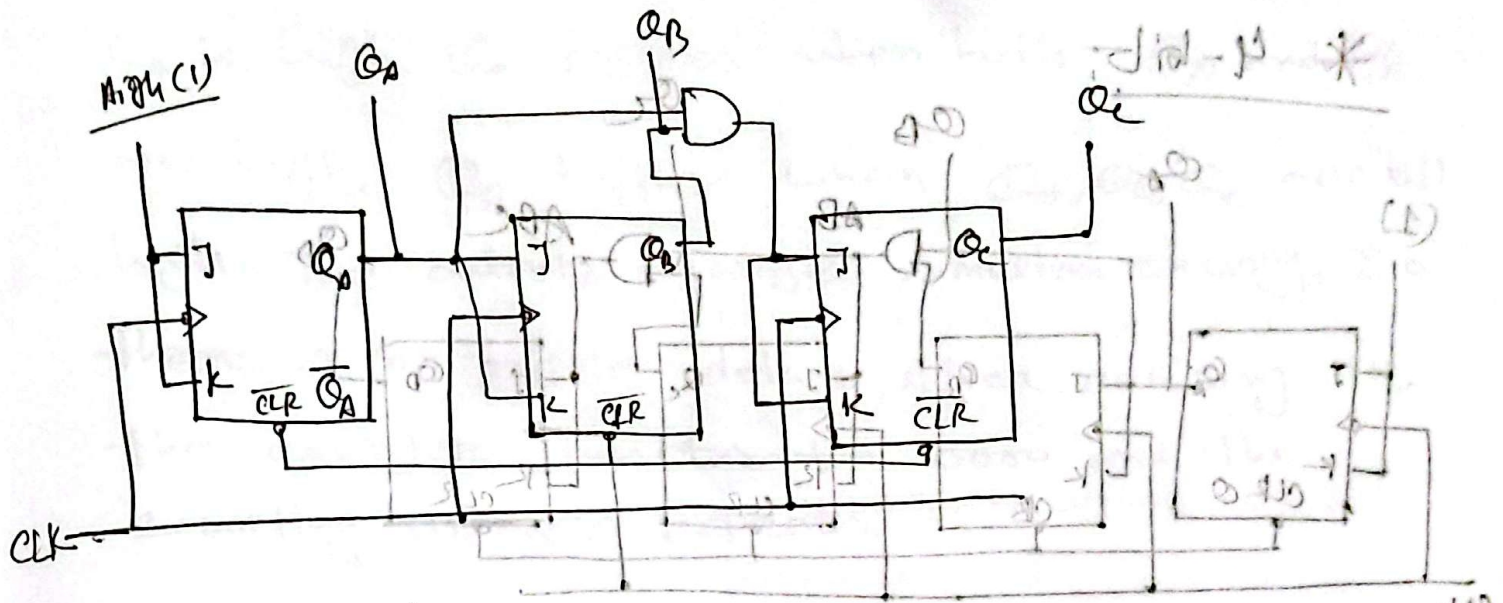
Maximum count $= 8 - 1 = 7$



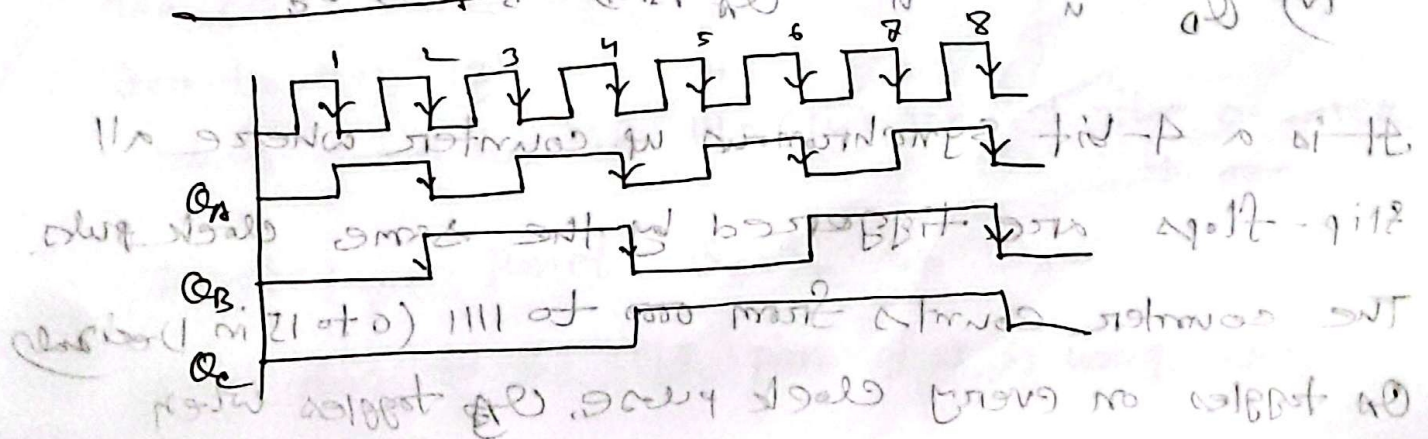
Circuit Excitation table:

P.S			N.S			J_c	K_c	J_B	K_B	J_A	K_A
Q_c	Q_B	Q_A	Q_c^+	Q_B^+	Q_A^+						
0	0	0	0	0	1	0	X	0	X	X	X
0	0	1	0	1	0	0	X	1	X	X	1
0	1	0	0	1	1	0	X	X	0	1	X
0	1	1	1	0	0	1	X	X	1	X	1
1	0	0	1	0	1	X	0	0	X	1	X
1	0	1	1	1	0	X	0	1	X	X	1
1	1	0	1	1	1	X	0	X	0	1	X
1	1	1	0	0	0	X	1	X	1	X	1

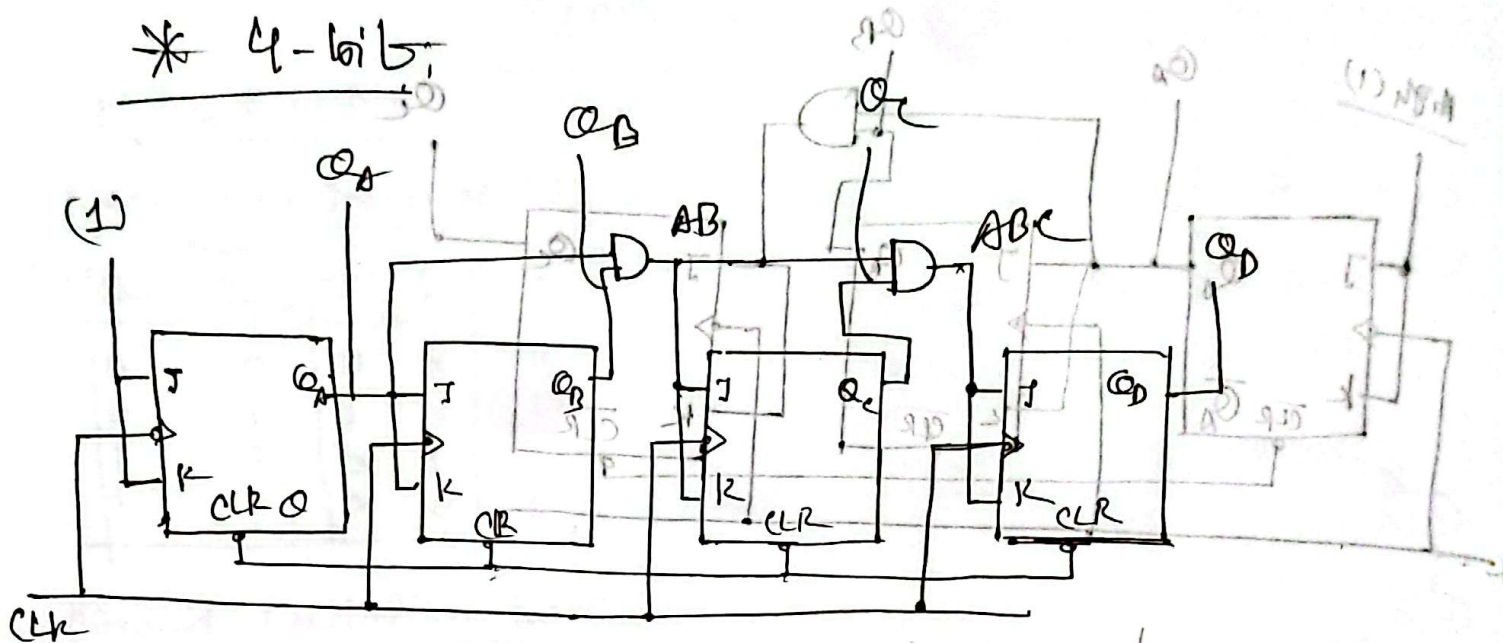




Clock Input	Output			
count	QC	QB	QA	Decimal
Initial	0	0	0	0
1	0	0	1	1
2	0	1	0	2
3	0	1	1	3
4	1	0	0	4
5	1	0	1	5
6	1	1	0	6
7	1	1	1	7
8	0	0	0	Reset to 0



* 4-bit



• All ff get the same clock input (CLK) $\rightarrow$ it's make it synchronous.

• connect $J=K=1$

• Use AND gate to drive the control logic for each ff toggle condition,

i) Q_A toggles with every clock pulse,

ii) Q_B " when $Q_A = 1$

iii) Q_C " $Q_A \text{ AND } Q_B = 1$

iv) Q_D " " $Q_A \text{ AND } Q_B \text{ AND } Q_C = 1$

It is a 4-bit synchronous up counter where all flip-flops are triggered by the same clock pulse. The counter counts from 0000 to 1111 (0 to 15 in decimal). Q_A toggles on every clock pulse, Q_B toggles when

Q_A is high, Q_C toggles when both Q_A and Q_B are high, Q_D toggles when Q_A, Q_B, Q_C are all high. The outputs change simultaneously, so there is no ripple delay. After reaching 1111, the counter resets to 0000 and the counting process repeats.

* MOD-14 (VP-synchronous):

- Use 4 JK FF: Q_A (LSB), Q_B, Q_C, Q_D (MSB)
- All FF get the same clock input (CLK) $\rightarrow$ this makes it synch. ~
- Connect $J=K=1$ for all flip-flops (so each ff toggles

when enabled)

- Use AND gates to build control logic for toggling;

Q_A toggles with -

clock

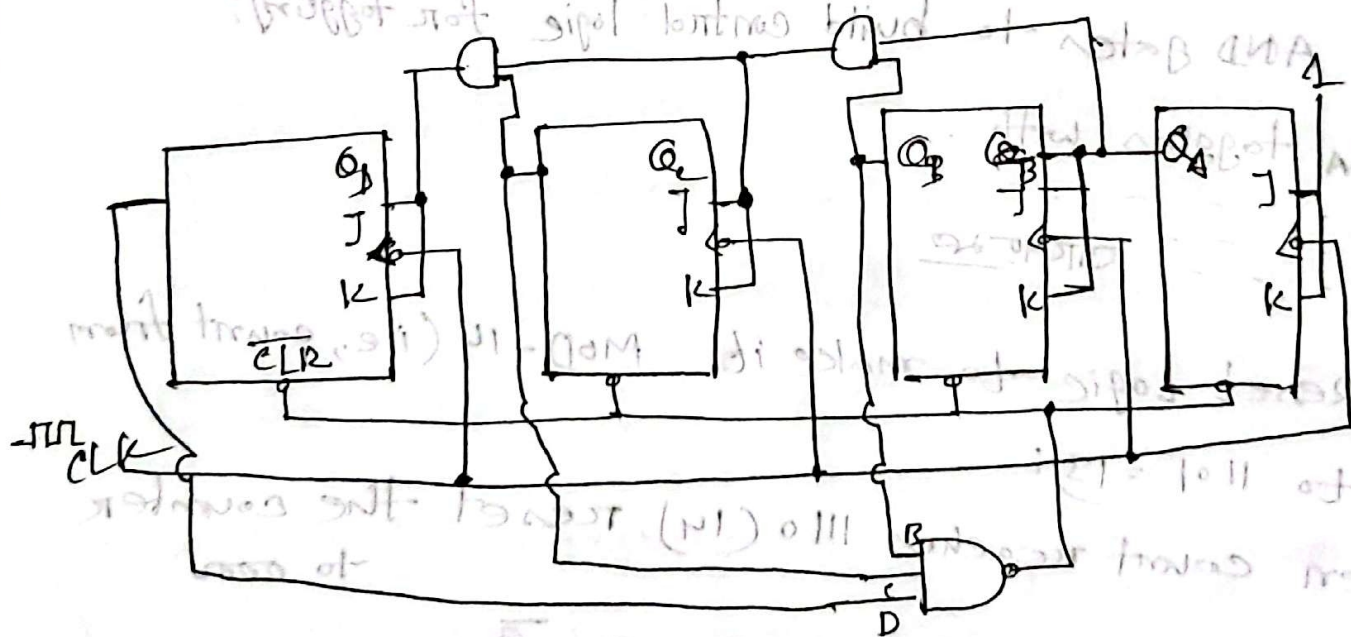
Add Reset logic to make it MOD-14 (i.e., count from 0000 to 1101 = 13);

When count reaches 1110 (14), reset the counter to 0000

Reset logic: $\text{Reset} = Q_D \cdot Q_C \cdot Q_B \cdot \bar{Q}_A$

Apply this to the CLR pins of all ff using gates.

It is a 4-bit SU-C designed to count from 0 to (MOD-14). All ff are triggered by the same clock signal. Q_A toggles when Q_B is high every clock pulse. Q_B is toggled when Q_A is high, Q_C is " " both Q_A, Q_B is high, and Q_D toggles when Q_A, Q_B, Q_C are all high. The counter progresses in binary from 0000 to 1101. When the state reaches (110) (14 in Decimal), the reset logic activates and clears all ff to return the counter to 0000. Because all ff are clocked simultaneously, the operation is fast and free from ripple delays.

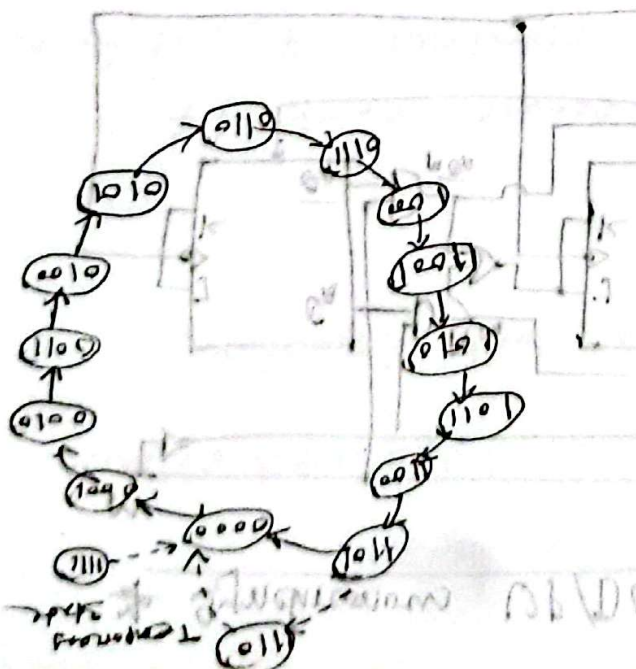


of how the output is used to
 control the counter.

Reset logic: $Q_3 \cdot Q_2 \cdot Q_1 \cdot Q_0$ (1111) turns off the counter.

14 unique states in reverse order, making it a mod-15 counter. The counter back to 1101 (13) reset logic to clear the counter back to 1101 (13). The next state becomes 110 (4), which is decoded by the next state (1101 to 0000) 13 to 0. After reaching 0, the counter goes back to 1101 (13).

Next	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Q ₃	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Q ₂	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1
Q ₁	0	1	1	0	1	0	0	1	0	1	0	0	1	0	0
Q ₀	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0



- Fir down: Inverted outputs are used.
- This design eliminates ripple delay and allows fast, reliable counting in both directions.

Async Option

→ It is called asynchronous because each ff is triggered by the output of the previous ff, not by a common clock

Presetable counter:

Ex:

Let's say the parallel inputs are set on

$P_2=1, P_1=0, P_0=1$ (binary 101)

while PL is High, these inputs have no effect, and the counter performs normal counting on each clock pulse

Now suppose PL is pulsed Low, when the counter is at 010 ($Q_2=0, Q_1=1, Q_0=0$). ^{the ff will load the parallel input value 101 anyway} This low signal at PL will cause:

- CLR (clear) input of Q_1 to go Low
- PRE (preset) inputs of Q_2 and Q_0 to go Low

As a result, the counter will immediately jump to the value 101, regardless of the current clock state. The count will hold at 101 as long as PL remains Low.

once PL returns High, the counter resumes normal counting from 101, increasing with each clock pulse

101 $\rightarrow$ 110 $\rightarrow$ 111 $\rightarrow$ 000 ...

~~The~~ The counter holds at 101 until PL goes High, then resumes counting from 101 on every clock pulse.

A Presetable counter is a counter that can be initialized to a specific binary value before it begins counting. It is done using parallel data inputs and a control signal called parallel Load (PL)

In this figure, the counter has:

- 3 inputs for normal up-counting

- Parallel data inputs: P_2, P_1, P_0
- FF: Q_2, Q_1, Q_0
- Control signal: PL (active Low)
- clock input (CLK)

How it works:

- The binary value to be loaded is applied to P_2, P_1, P_0 .
- A Low pulse is given to the PL input.
- This causes an asynchronous load; the value from P_2, P_1, P_0 are directly transferred to Q_2, Q_1, Q_0 ignoring the clock.
- While $PL=0$, the counter stops normal operation and holds the loaded value, because each FF receives an asynchronous counter input.
- When PL returns High, the counter resumes normal counting from the loaded value, based on clock pulse.

Ex: If $P_2=1, P_1=0, P_0=1$ (binary 101), and PL is pulsed Low, the counter loads 101. After PL goes High, it will count:

101 $\rightarrow$ 110 $\rightarrow$ 111 $\rightarrow$ 000 ...

Note: Asyn. presetting works without waiting for the clock.

Some counters use synchro presetting where loading happens on the clock edge.

IC: Asy: 74ALS190, 74HC191
Sy: 74ALS161, 74HC168

* Principle of oop in real life?

i) Encapsulation: Binding data & methods together and restricting direct access to data. hides them from outside

R.L.E. A medical capsule hides the medicine inside - only the outer shell is visible

A mobile phone hides internal circuit, we use buttons to operate it.

ii) Inheritance: one class inherits properties of another

Ex: A child inherits features from parents.

Private variable + public getter/setter

iii) Polymorphism: One action behaves differently in different situations.

In java: class B extends A

Ex:

A person can act as a teacher in school, parent at home and driver on road - same person, different role.

In java: method overloading, overriding

iv) Abstraction is the process of hiding complex internal details and showing only essential features.

R.L.E: You can drive a car without knowing how the engine works.

Java: Achieved using abstract class and interfaces

* Difference Object class:

Class	Object
i) Blueprint or template	i) Real world instance of a class
ii) Doesn't occupy memory	ii) occupies memory
iii) Defines properties and behaviours	iii) Uses those properties and behaviours
iv) Class is a datatype class car	iv) Object is an instance of class car mycar = new car();

Abstraction

Encapsulation

i) Hides internal details, show only essential

i) Binds data & methods together, hides actual data

ii) Solves the problem in design level

ii) Solves the problem in the implementation level

Ex

Ex: Private String name

Inheritance

Polymorphism

i)
ii) Uses extends keyword

i)

ii) Uses method overloading or overriding

iii) Ex: class Dog extends Animal

Ex:

area (int)

and area (double)

methods

Eat (method) behaves differently in Dog and cat classes.

Dynamic Binding

i) Method to call is decided at runtime

ii) Supports runtime polymorphism

```
Animal a = new Dog();
```

```
Dog d; a.Sound();
```

Message passing

i) object communicate by calling each other's methods

ii) Help in object collaboration

```
dog.bark();
```

sending message to dog object

*dog d = new dog();
d.bark();*

Polymorphism

i) Same data is used together with different actual data

ii) When the program is compiled, the compiler does not know which method to call

Ex: Private String name

Collaboration

i) Hidden internal details shown

only one method

ii) solves the problem in design

level

Ex

Polymorphism

i)

ii) When method overloading

Ex: Over (int)

and Over (double)

method

Ex: Over (int)

and Over (double)

Abstraction

i) When extends keyword

ii) Ex: class Dog extends Animal