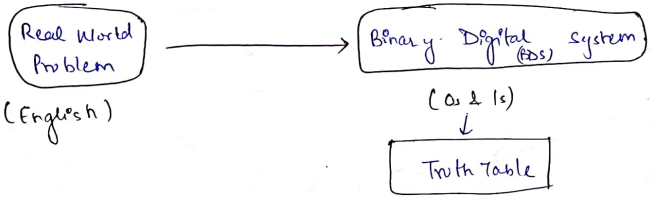
DLD

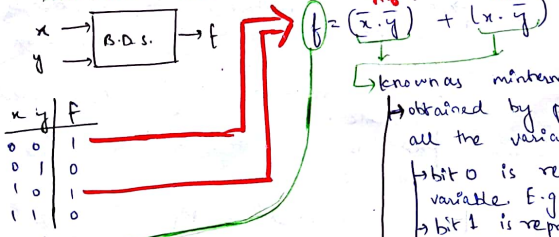
Chapter 1: Representation of Boolean functions:



→ A BDS is used to solve/implement real world problems. Moreover, whenever any real world problem has to be solved using a digital system, the initial necessary requirement is to first convert the problem from our familiar English language representation into the language understood by a BDS, i.e., the language of 0s and 1s.

→ Furthermore, a truth table is the name given to the most basic representation of a real world problem in the language of 0s and 1s.

Min terms approach of designing a BDS:



$f = (\bar{x} \cdot \bar{y}) + (x \cdot \bar{y})$ (m_0, m_2)

↳ known as minterms.

↳ obtained by performing AND operation of all the variables in such a way that:

↳ bit 0 is represented by complemented variable. E.g. $\bar{x}$

↳ bit 1 is represented by uncomplemented variable. E.g. x

↳ They are also represented as m_j, where "j" is the decimal equivalent of the binary input.

↳ The func" f" is said to be in the sum of minterms form.

→ The func" f" is also represented as $f = m_0 + m_2$

$f = \Sigma(0,2)$

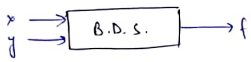
↳ implies the OR operation.

→ After simplifying the function, the function is said to be in the sum of products form [SOP form].

E.g. An integer number N ranges from 0 to 3. Find out if N is divisible by 2; then output = 1
or else, output = 0

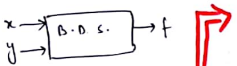
	x	y	f
0	0	0	1
1	0	1	0
2	1	0	1
3	1	1	0

0 1 2 3	0 1
$2^n : 4$	$2^n : 2$
$n \geq 2$	$n \geq 1$



NOTE
→ For simplifying a function, there are 2 types of simplification techniques:
→ Boolean algebra Simplification → Karnaugh Map Simplification

Max terms approach of designing a BDS:



x	y	f
0	0	1
0	1	0
1	0	1
1	1	0

$f = (x + \bar{y}) \cdot (\bar{x} + \bar{y})$ (M_1 · M_3)

→ known as maxterms
→ obtained by performing OR operation of all the variables of the function in such a way that:
→ bit 0 is represented by uncomplemented variable.
→ bit 1 is represented by complemented variable.
→ Also represented as M_j, where "j" is the decimal equivalent of the binary input.

→ The func" "f" is said to be in the product of maxterms form.
→ The func" "f" is also represented as:
$f = M_1 \cdot M_3$
$f = \pi(1,3)$
↳ represents AND operation.
→ After simplification, the func" "f" is said to be in Product of Sums (POS) form.

Variables, Literals and Terms:

E.g. $f(x,y) = \bar{x} \cdot \bar{y} + x \cdot \bar{y}$
Variables = 2
Terms = 2
Literals = 4 (No. of occurrence of variables)

Canonical Forms:

→ A function that is represented in either the sum of minterms or the product of maxterms form. is said to be in the canonical form.

* Conversion between canonical forms:

E.g.

x	y	f
0	0	1
0	1	0
1	0	1
1	1	0

$f = \Sigma(0,2)$
$= \pi(1,3)$

→ The steps of conversion are as follows:
① Interchange the symbol "Σ" and "π".
② Write the missing numbers of the function.

GATE-2015 | EE

E.g. $f(x,y,z) = \bar{x}y\bar{z} + x\bar{y}\bar{z} + xy\bar{z} + xyz$
$= \Sigma(2,4,6,7)$
$= \pi(0,1,3,5)$
$= (x+y+z)(x+y+\bar{z})(x+\bar{y}+\bar{z})(\bar{x}+y+\bar{z})$ (a)

Q Eg $f(x,y,z) = \Sigma(1,2,5,6,7)$
$= \pi(0,3,4)$
$= (x+y+z)(x+\bar{y}+\bar{z})(\bar{x}+y+z)$ (b)

Total Possible functions: $= 2^{2^n}$

x	y	f_0	f_1	f_2	.....	f_{15}
0	0	0	0	0		1
0	1	0	0	0		1
1	0	0	0	1		1
1	1	0	1	0		1

→ With 'n' boolean variables, a total of 2^{2^n} number of different Bodean functions can be formulated.

Eg

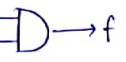
No. of variables	Total possible functions
2	$2^{2^2} = 16$
3	$2^{2^3} = 256$
4	$2^{2^4} =$ ~~16384~~ 65536 ≈ 64 K

Chapter-2 Logic Gates

① NOT / Inverter
② AND
③ OR
④ NAND
⑤ NOR
⑥ EXOR / XOR
⑦ EXNOR / XNOR

① ② ③ — Basic Logic Gates
④ ⑤ — Universal Logic Gates

AND Gate:



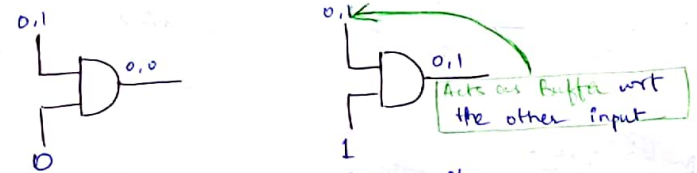
x	y	f
0	0	0
0	1	0
1	0	0
1	1	1

$f = xxy = xy$
$= x \wedge y$

• If there is no operator mentioned, then it is AND operation.

→ It obeys the commutative law. [$xy = yx$]
→ It obeys the associative law. [$x \cdot (y \cdot z) = (x \cdot y) \cdot z$]

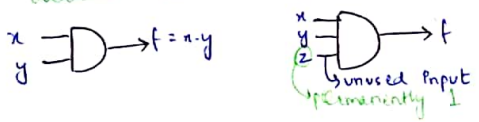
Control inputs of AND gate:



NOTE:

$f = x$
Buffer Gate

Unused Input of AND Gate:



$f = x \cdot y \cdot 1 = x \cdot y$ | $f = x \cdot y \cdot y = x \cdot y$

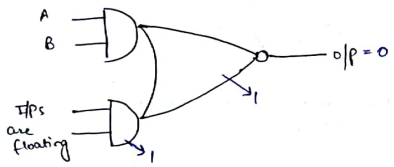
→ The unused input or inputs of an AND gate can be connected in one of the following ways:

① Logic 1 / Pull up method
② One of the used inputs
③ In case of TTL, it can be left unconnected (open/floating).

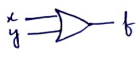
NOTE
→ In case of TTL, open/floating input acts as logic 1.

23/10/2017

TTL ADI



OR Gate:

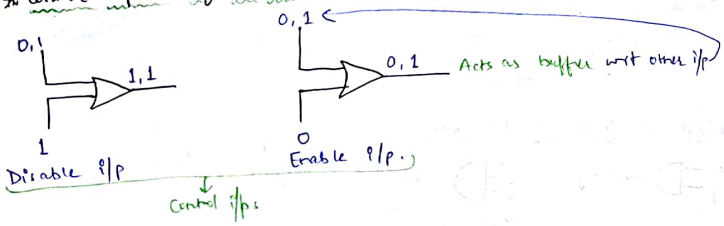


x	y	f
0	0	0
0	1	1
1	0	1
1	1	1

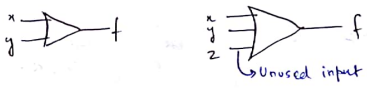
$f = x + y$
$= x \vee y$

→ Obeys the commutative law
→ Obeys the associative law

* Control inputs of OR Gate:



* Unused i/p of "OR" Gate:

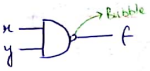


$f = x + y + 0$
$= x + y$

$f = x + y + y$
$= x + y$

→ Unused input/inputs of an "OR" gate can be connected in one of the following ways:
① Logic 0 (Pull down method) → Best method
② One of the used inputs.

NAND Gate:



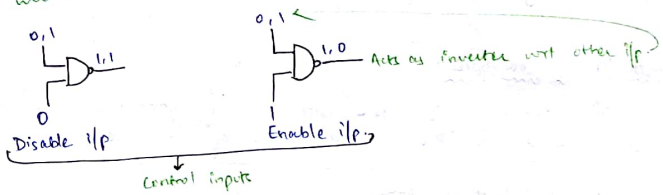
x	y	f
0	0	1
0	1	1
1	0	1
1	1	0

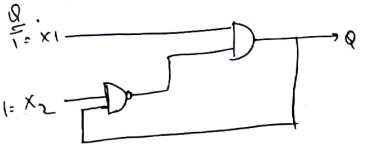
$f = \bar{x} + \bar{y}$
$= \overline{x \cdot y}$
$= x \uparrow y$

→ Obeys commutative law.
→ Does not obey associative law.
$(\overline{x \cdot y}) \cdot z \neq x \cdot (\overline{y \cdot z})$

NOTE: The universal logic gates (NAND & NOR) do not obey the associative law.

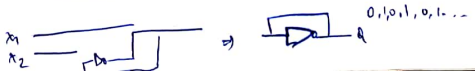
* Control inputs of NAND Gate:





$x_1 = 1$
$x_2 = 1$

ⓐ at logic 1
ⓑ at logic 0
ⓒ at its initial
ⓓ Unstable



* Unused input of NAND Gate:

→ The unused input(s) of a NAND Gate can be connected in exactly the same ways as the unused input of AND Gate.

NOR Gate : x, y → f

x	y	f
0	0	1
0	1	0
1	0	0
1	1	0

$f = \overline{x} \cdot \overline{y}$

$= \overline{x+y}$

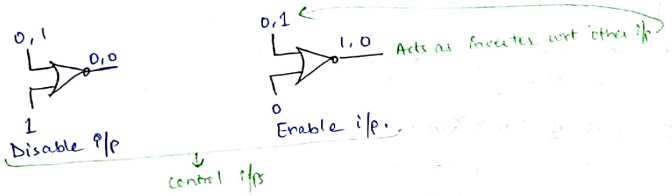
$= x \downarrow y$

→ obeys commutative law

→ Does NOT obey associative law.

$(\overline{x+y})+z \neq \overline{x+(\overline{y+z})}$

* Control i/ps for NOR Gate:



* Unused i/p of NOR Gate:

→ The unused input(s) of a NOR Gate can be connected in exactly the same ways as the unused input of OR gate.

EXOR Gate / XOR Gate : x, y → f

↳ Exclusive OR

x	y	f
0	0	0
0	1	1
1	0	1
1	1	0

$f = (\overline{x} \cdot y) + (x \cdot \overline{y})$

$= (x+y) \cdot (\overline{x}+\overline{y})$

$= x \oplus y$

→ f = 1 ; when $x \neq y$

= 0 ; when $x = y$

→ EXOR acts as an odd function.

* In an odd function, the toggle output is equal to logic 1 when odd number of inputs are equal to 1.

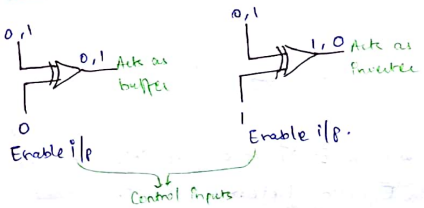
x	y	z	$x \oplus y$	$x \oplus y \oplus z$
0	0	0	0	0
0	0	1	0	1
0	1	0	1	1
0	1	1	1	0
1	0	0	1	1
1	0	1	1	0
1	1	0	0	0
1	1	1	0	1

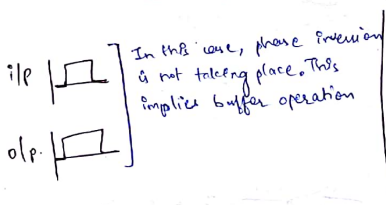
NOTE:
OR gate is also known as inclusive OR gate.

→ XOR gate obeys the commutative law.

→ XOR gate obeys the associative law.

* Control Inputs of XOR Gate:





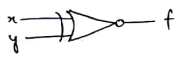
* $A \oplus A = 0$
$A \oplus \bar{A} = 1$
$A \oplus 0 = A$
$A \oplus 1 = \bar{A}$

If $A \oplus B = C$
then, $A \oplus C = B$
$B \oplus C = A$
$A \oplus B \oplus C = 0$
$C \oplus C = 0$

$A \oplus A \oplus A = A$
$A \oplus A \oplus A \oplus A = 0$

- $A \oplus A \oplus A \oplus A \ldots\ldots n$ times
  $= A$, when n is odd
  $= 0$, when n is even

EXNOR Gate / XNOR Gate :



x	y	f
0	0	1
0	1	0
1	0	0
1	1	1

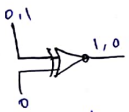
$f = \bar{x}\bar{y} + xy$
$= \overline{xy}\,(x+\bar{y}).(\bar{x}+y)$
$= x \odot y$

$f = 1$; when $x = y$
$= 0$; when $x \neq y$

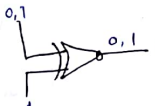
Because of this reason
- EXNOR logic is also known as Equivalence logic / Coincidence logic.
- EXNOR is used as a Equality Detector circuit.

→ Obeys the commutative law.
→ Obeys the associative law.

* Control i/ps of EXNOR gate:



Enable i/p



Enable i/p.

$A \odot A = 1$
$A \odot \bar{A} = 0$
$A \odot 0 = \bar{A}$
$A \odot 1 = A$

If $A \odot B = C$
then, $A \odot C = B$
$B \odot C = A$
$A \odot B \odot C = 1$

$A \odot A \odot A = A$
$A \odot A \odot A \odot A = 1$

- $A \odot A \odot A \odot A \ldots\ldots n$ times
  $= A$; when n is odd
  $= 1$; when n is even

$\bar{A} \oplus B = \bar{A} \cdot B + \bar{A} \cdot \bar{B}$
$= A \cdot B + \bar{A} \cdot \bar{B}$
$= A \odot B$

$A \odot B = \overline{A \oplus B}$
$= \bar{A} \oplus B$
$= A \oplus \bar{B}$

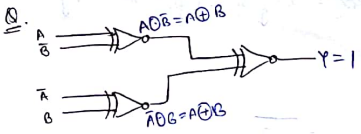
$A \oplus B = \overline{A \odot B}$
$= \bar{A} \odot B$
$= A \odot \bar{B}$

$A \odot B = \overline{A \oplus B}$
$A \odot B \odot C = \overline{A \oplus B} \odot C$
let $x = A \oplus B$ $y = C$
$= \bar{x} \odot y$
$= x \odot y$
$= A \oplus B \oplus C$

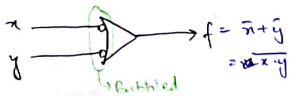
NOTE

→ In case of EXOR and EXNOR;
1. for even no. of variables, EXOR and EXNOR are complements of each other.
2. for odd no. of variables, EXOR and EXNOR are equal to each other.

Q.



Bubbled Logic:



→ NAND gate is equivalent to a Bubbled OR gate.

Similarly:

→ NOR gate = Bubbled AND gate
→ AND gate = Bubbled NOR gate
→ OR gate = Bubbled NAND gate

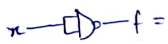
Universal Logic Gates:

→ With the help of only 1 or more NAND gates, or with the help of only 1 or more than 1 NOR gates, it is possible to implement any other logic gate.

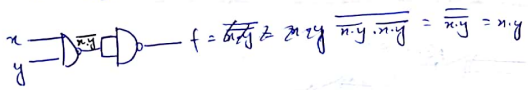
→ It is because of this reason NAND gate and NOR gates are known as universal logic gates.

* NAND Gate as a Universal logic gate:

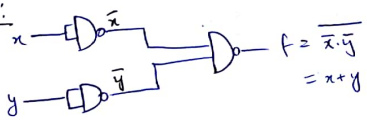
① NOT Gate:



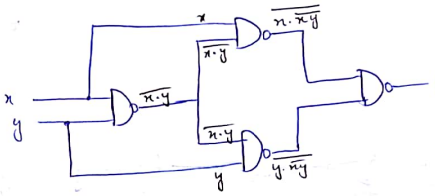
② AND Gate:



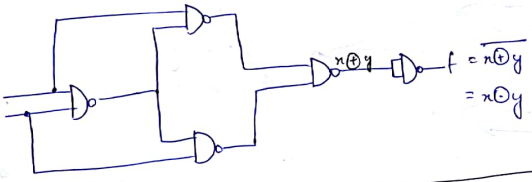
③ OR Gate:



④ EXOR Gate:

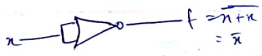


⑤ EXNOR Gate:

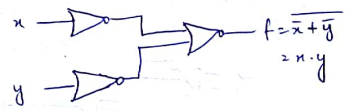


NOR Gate as a universal logic gate:

① NOT Gate:

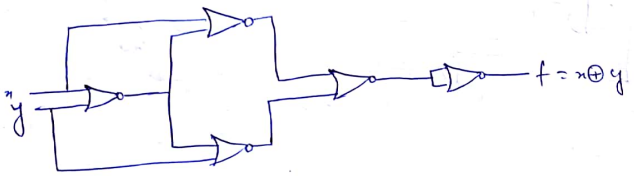


② AND Gate:

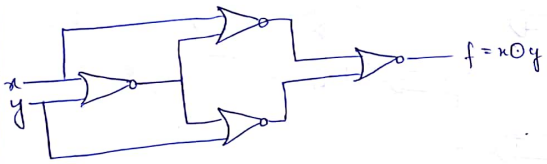


③ OR Gate:

④ EXOR Gate:



⑤ EXNOR Gate:



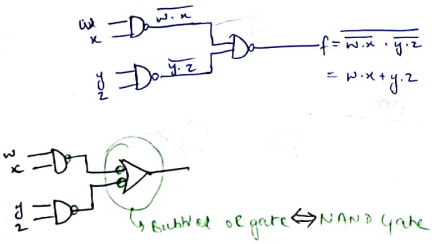
NOTE:
→ To implement the NOR gate from NAND gate and vice versa, a total of 4 logic gates are needed.

Q. What is the minimum number of:

2-input NAND gates needed to implement the boolean function

$f = w.x + y.z$? = $\overline{\overline{w.x} \cdot \overline{y.z}} \rightarrow \overline{\overline{w.x}} + \overline{\overline{y.z}} = w.x + y.z$

(a) 2
(b) 3
(c) 4
(d) 5



Bubbled OR gate ⇔ NAND Gate.

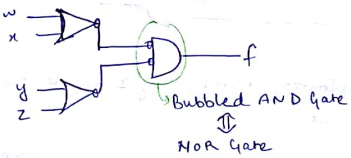
NOTE

1. The previous circuit is termed as a 2-level AND-OR implementation.
2. A 2-level ~~or~~ AND-OR implementation is equivalent to a 2-level NAND-NAND implementation.
3. A 2-level AND-OR implementation is obtained when a digital circuit is implemented from a SOP ~~expression~~ expression.

Q. What is the minimum no. of 2-input NOR gates needed to implement the boolean function:

$f = (w+x).(y+z)$

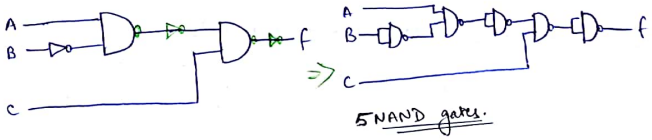
(a) 2
(b) 3
(c) 4
(d) 5



NOTE

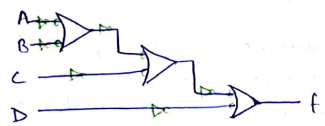
1. The above circuit is termed as a 2-level OR-AND implementation.
2. A 2-level OR-AND implementation is equivalent to a 2-level NOR-NOR implementation.
3. A 2-level OR-AND implementation is obtained when a digital circuit is implemented from a POS expression.

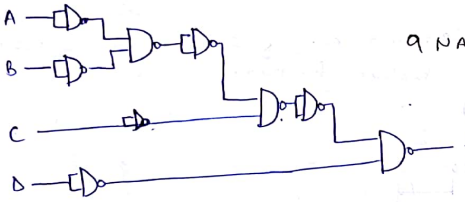
Q56. $f(A,B,C) = A\,\overline{B}\,C$ = ~~$\overline{A.\overline{B}} \rightarrow \overline{A} + \overline{\overline{B}.\overline{C}} \rightarrow \overline{A} + B + \overline{C} \rightarrow \overline{A}.\overline{B} + \overline{C} \rightarrow \overline{A.B} + \overline{\overline{A.B}.C}$~~



5 NAND gates.

Q.55. $f = A + B + C + D$

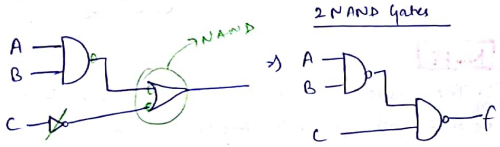




9 NAND Gates.

Q. What is the minimum number of 2-input NAND gates needed to implement:

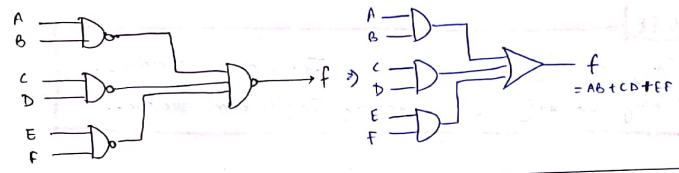
$F = AB + \bar{C}$



Q. AND-OR equivalent to

(a) NAND-NAND
(b) NAND-NOR
(c) NOR-NAND
(d) NOR-NOR

Q. The dp of the ckt shown below is:



NOT Gate / Inverter : x — f

x	f
0	1
1	0

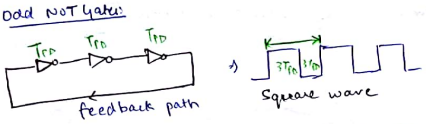
$f = \bar{x}$
$= x'$

Even NOT Gates:

→ The above circuit acts as the most basic memory element.

→ If even no. of NOT gates are connected in a feedback path, the circuit acts as a memory element/bistable multivibrator.

Odd NOT Gates:



$T = 3T_{PD} + 3T_{PD}$
$= 2 \cdot 3 T_{PD}$

→ In general, the time period of the square wave generated,

$T = 2 \times (N) T_{PD}$

→ odd no. of NOT gates connected in the feedback path.

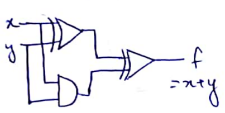
→ Frequency of square wave $= \frac{1}{T}$

→ If odd no. of NOT gates are connected in a feedback path, then the circuit acts as a square wave generator/astable multivibrator/Ring oscillator.

NOTE

- $\bar{x}\cdot y$ and $x\cdot\bar{y}$ are known as Inhibition functions.
- $\bar{x}+y$ and $x+\bar{y}$ are known as Implication functions.

Q.6. $f = x \oplus y \oplus xy$



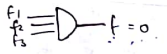
x	y	xy	x⊕y	x⊕y⊕xy
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	0	1

(6)

Q51. $f_1(A,B,C) = \Sigma(2,3,4)$

$f_2(A,B,C) = \pi(0,1,3,6,7)$

$= \Sigma(2,4,5)$



	A	B	C	f_1	f_2	f_3
0	0	0	0	0	0	
1	0	0	1	0	0	
2	0	1	0	1	1	1
3	0	1	1	1	1	0
4	1	0	0	1	1	1
5	1	0	1	0	1	
6	1	1	0	0	0	
7	1	1	1	0	0	

$f_1 = \bar{A}B\bar{C} + \bar{A}BC + AB\bar{C}$

$f_2 = \bar{A}B\bar{C} + A\bar{B}\bar{C} + A\bar{B}C$

$f_3 = \Sigma(0,1,3,5,6,7)$

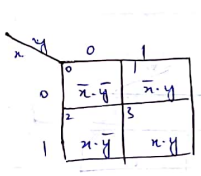
Total maximum no. of possible minterms = 6

f_3

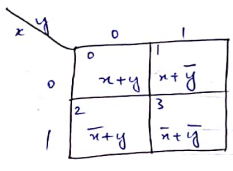
KARNAUGH MAP SIMPLIFICATION : Chapter-3

#2 variable k-map:

$f(x,y)$ — MSB, LSB



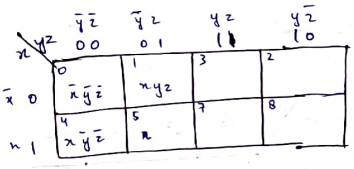
Minterm Map



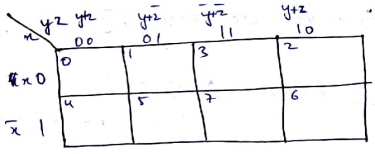
Maxterm Map

#3-variable k-map:

$f(x,y,z)$ — MSB, LSB



Minterm map



Maxterm Map

NOTE:

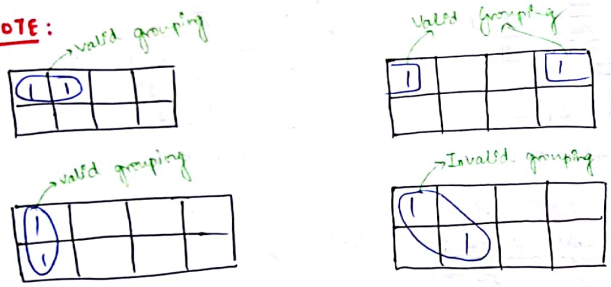
→ In a k-map, any 2 cells are considered to be adjacent, if their binary numbering differs by only 1-bit.

→ Furthermore, while performing simplification, any 2 cells that are adjacent to each other can be grouped together to obtain the simplified expression.

NOTE:

- In a k-map, the no. of cells that can be grouped is always a power of 2 value.
  E.g. 1, 2, 4, 8, 16 . . . . cells.

NOTE:



4-variable k-map : [24/10/2017] Point-(4).

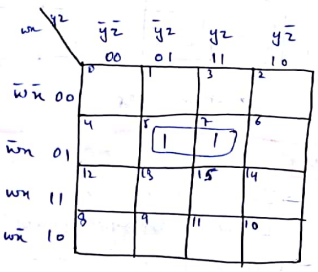
Eg $f(w,x,y,z) = \Sigma(5,7)$

$= \bar{w}\cdot x\cdot \bar{y}\cdot z + \bar{w}\cdot x\cdot y\cdot z$

$= \bar{w}x\bar{y}z + \bar{w}xyz$

$= \bar{w}xz$

$\bar{w}x\bar{y}z$
$\bar{w}xyz$
$\bar{w}xz$

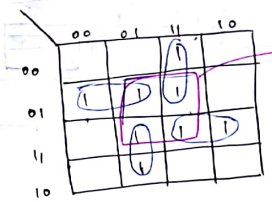


(No. of cells) Grouping	No. of variables eliminated
2 (Pair)	1
4 (Quad)	2
8 (Octet)	3
16	4
2^n	n

Priority of grouping:
Octet > Quad > Pair

* Remove redundant group(s).
  ↳ It is a group whose all the cells are covered by adjacent groups.

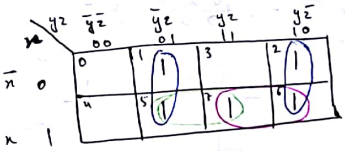
Q.31. $f = \bar{w}x\bar{y} + \bar{w}yz + w\bar{y}z + wxy$ (b)



This group is a redundant group. Therefore, it is eliminated.

NOTE
↳ In a k-map, while forming the groupings, the biggest possible groupings should be formed.
↳ In order to do so, if overlapping of 2 groups is required, then this overlapping is also allowed.

Ques What is the simplified expression for the boolean function:
$f(x,y,z) = \Sigma(1,2,5,6,7)$



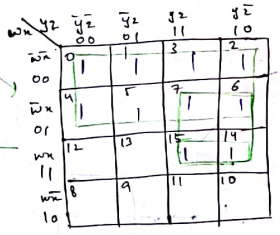
$f = \bar{y}z + y\bar{z} + xz$
$= \bar{y}z + y\bar{z} + xy$

NOTE
↳ In a k-map, sometimes, it may be possible to obtain more than 1 simplified expressions.

Semi-Simplified boolean function:

Eg. $f(w,x,y,z) = \bar{w}.\bar{y} + wxy + \bar{w}.y$

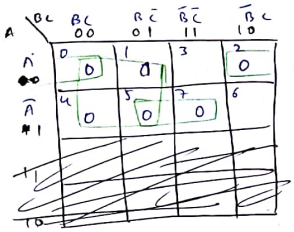
$= \bar{w} + xy$



$y(wx+\bar{w})$
$= y((w+\bar{w})(\bar{w}+x))$
$= y\bar{w} + xy + \bar{w}\bar{y} + \bar{w}(y+\bar{y}) + xy$
$= \bar{w} + xy$

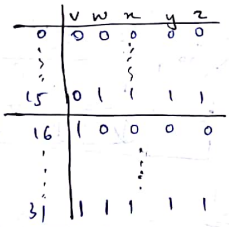
Eg. $f = \pi(0,1,2,4,5,7)$

$= \cancel{B + AC}$
$B.(A+C).(\bar{A}+\bar{C})$ (a)

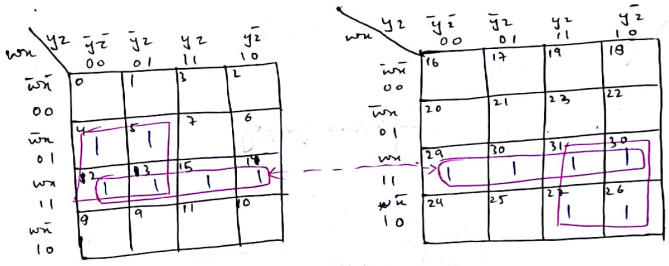


5-variable K-map [Point-5]

→ In a 5 variable K-map, a cell in the v=0 map is adjacent to the corresponding cell in the v=1 map.



Eg. $f(v,w,x,y,z) = \Sigma(4,5,12,13,14,15,26,27,28,29,30,31)$



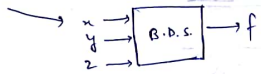
$f = wx + \bar{v}x\bar{y} + vwy$

Don't Care Conditions:

Eg.

	x	y	z	f
0	0	0	0	1
1	0	0	1	0
2	0	1	0	0
3	0	1	1	1
4	1	0	0	0
5	1	0	1	1
6	1	1	0	1-0-X
7	1	1	1	1-0-X

Real world Problem

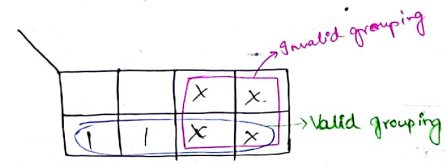


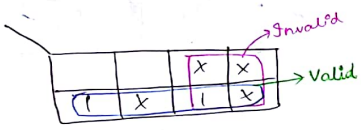
Don't Care

→ A function in which, depending upon the real world problem statement, the output is not specified for certain input combinations [i.e., unused input combinations] is known as an incompletely specified function.
→ While simplifying such a function on the K-map, for the unused inputs we assume that the output of the function is either logic 1 or logic 0.
→ The advantage of doing so is that, it may provide additional amount of simplification on the K-map.
→ These unused input combinations are known as don't care conditions.

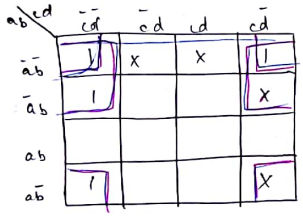
→ While simplifying a function on the k-map, it is not necessary to include the don't care conditions (~~DD~~DCCs) in a grouping.

E.g:





Q. What is the simplified expression.

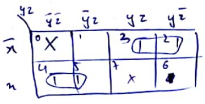


$f = \bar{b}\bar{d} + \bar{a}\bar{d}$

T18

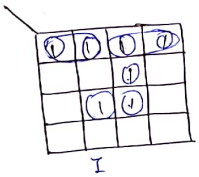
x	y	z	f
0	0	0	X
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	X

$f = \Sigma(2,3,4,5)$

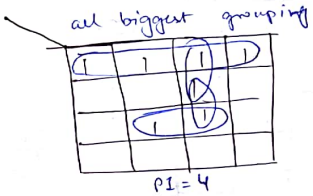


$f = x\bar{y} + \bar{x}y$

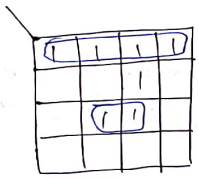
~~Prime~~ Implicants, Prime Implicants, Essential Prime Implicants, Non-essential prime implicants Point - 7 :



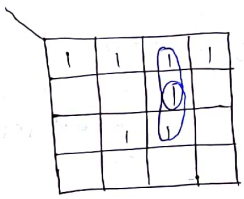
I



PI = 4



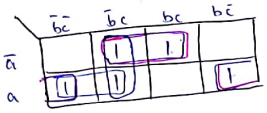
EPI = 2



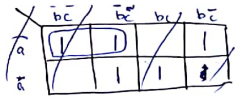
NPI = 2

NOTE
↳ In an n-variable boolean function, the maximum possible no. of prime implicants is equal to 2^{n-1}.

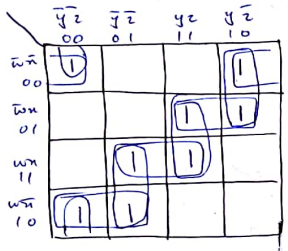
Q.42. $f(a,b,c) = \bar{a}c + a\bar{c} + \bar{b}c$



PI = 4 EPI = 2 = $\bar{a}c$ and $a\bar{c}$ (a)



Q.89. $f(w,x,y,z) = \Sigma(0,2,6,7,8,9,13,15)$



PI = 8

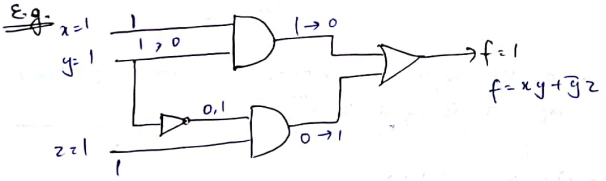
EPI = 0

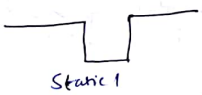
NPI = 8

Hazards: (Point - 9)

→ It is a problem that occurs in a digital circuit because of the finite (non-zero) propagation delay of logic gates in a digital circuit.

E.g.





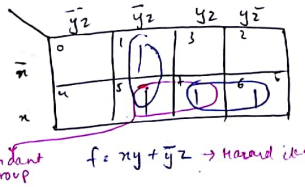
Static 1

Static 0

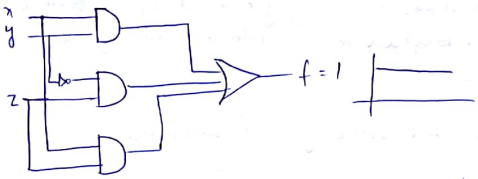
Elimination of Hazards:

$f = xy + \bar{y}z + xz$

(Hazard free ckt

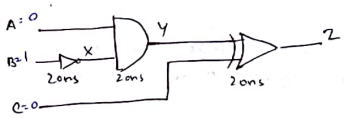


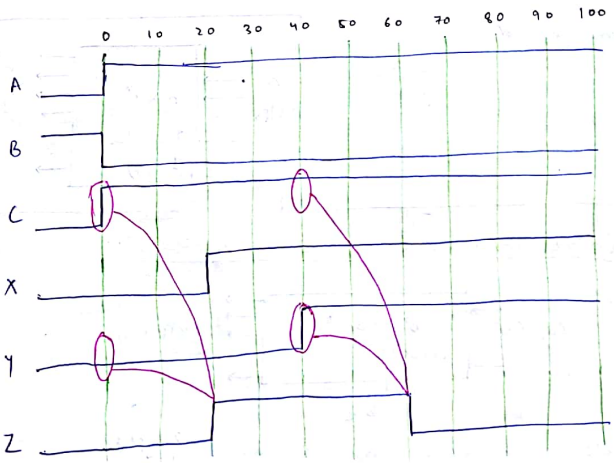
$f = xy + \bar{y}z$ → Hazard ckt



→ In a digital circuit, hazards are eliminated by including redundant groups on the k-map.

TIMING DIAGRAM:





→ While solving a problem, the propagation delay of the logic gates is not considered in the initial value of the waveforms.

↳ * while drawing the initial value of the waveforms.

CHAPTER-4 Combinational Circuits

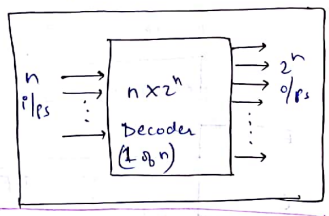
#General design procedure of a combinational circuit Point-10.

#Decoder :

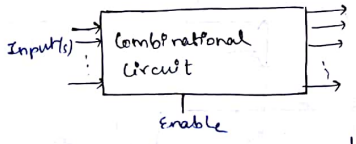
→ A decoder is a combinational circuit that converts binary coded information into familiar representations like octal, decimal, hexadecimal, etc.

→ A decoder has n inputs and $\leq 2^n$ outputs.

* Minterm generator.



Note → Enable input in a combinational circuit.



→ Enable input is an activating signal
→ It is an optional input.

2 types
→ Active High Enable (E)
→ Active Low Enable ($\bar{E}$)

Active High Enable

Active low Enable

2 x 4 Decoder with Active High Enable : [1 of 4 decoder]

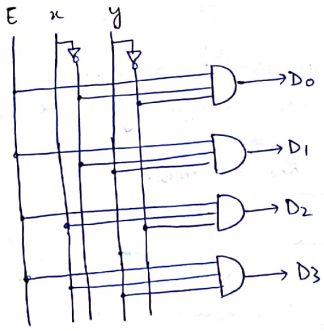
E	x	y	D_0	D_1	D_2	D_3
0	X	X	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1

$D_0 = E\,\bar{x}\,\bar{y}$

$D_1 = E\,\bar{x}\,y$

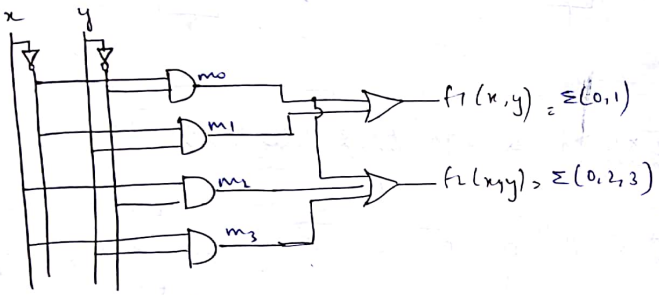
$D_2 = E\,x\,\bar{y}$

$D_3 = E\,x\,y$



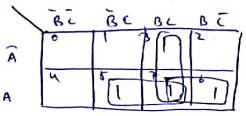
Boolean function implementation using a decoder:

Eg



Q13. $x = \Sigma(\cancel{0,5,7}, 3,5,6,7)$

$= AC + AB + BC$ (a)

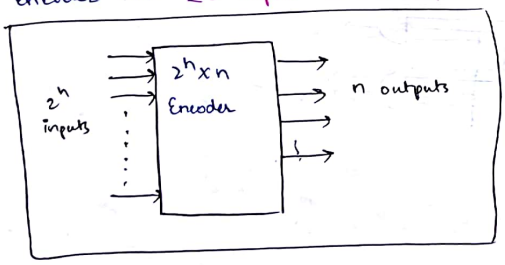


Encoder:

→ It performs the inverse operation of decoder.

→ It generates the binary code corresponding to the input value

→ An encoder has $\leq 2^n$ inputs and n outputs.

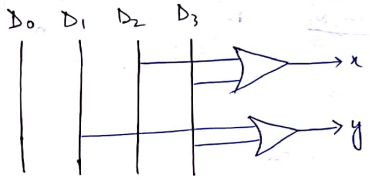


4X2 Encoder: (Basic Encoder)

D_0	D_1	D_2	D_3	x	y
1	0	0	0	0	0
0	1	0	0	0	1
0	0	1	0	1	0
0	0	0	1	1	1

$x = D_2 + D_3$

$y = D_1 + D_3$



4X2 Encoder: (Priority Encoder)

D_0	D_1	D_2	D_3	x	y
1	0	0	0	0	0
X	1	0	0	0	1
X	X	1	0	1	0
X	X	X	1	1	1

D_0	D_1	D_2	D_3	x	y
X	X	1	0	1	0
0	0	1	0	1	0
0	1	1	0	1	0
1	0	1	0	1	0
1	1	1	0	1	0

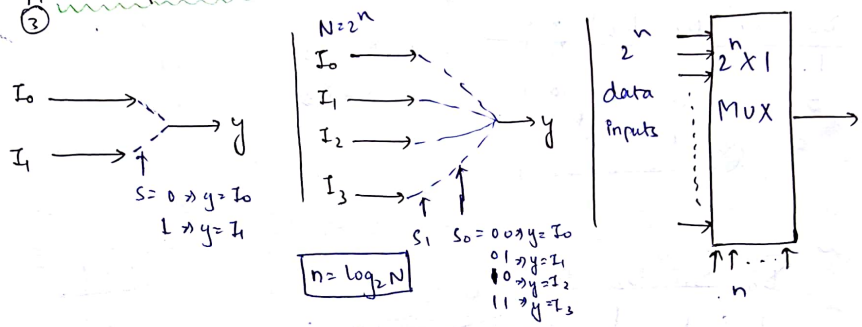
→ The i/p may either be logic 0 or logic 1.

$x = D_2 + D_3$

$y = D_1\bar{D}_2 + D_3$

Multiplexer :

③



→ MUX is also known as

- Data Selector
- Parallel to serial converter
- Many to one circuit
- Universal logic circuit

NOTE

↳ In general, in a truth table,

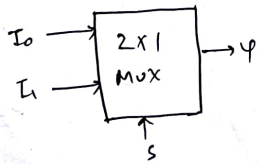
E.g.

x	y	f
0	0	1
0	1	0
1	0	1
1	1	0

$f = \bar{x}\cdot\bar{y} + x\bar{y}$

$= \bar{x}\cdot\bar{y}\cdot A + \bar{x}y\cdot B + x\bar{y}\cdot C + xy\cdot D$

2x1 MUX :



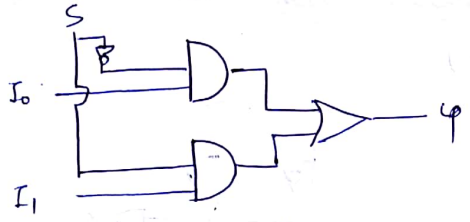
Block Diagram

S	φ
0	I0
1	I1

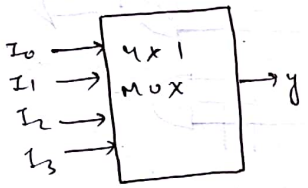
Truth Table (or) Function Table

$\varphi = \bar{S}\cdot I_0 + S\cdot I_1$

↳ Boolean Expression

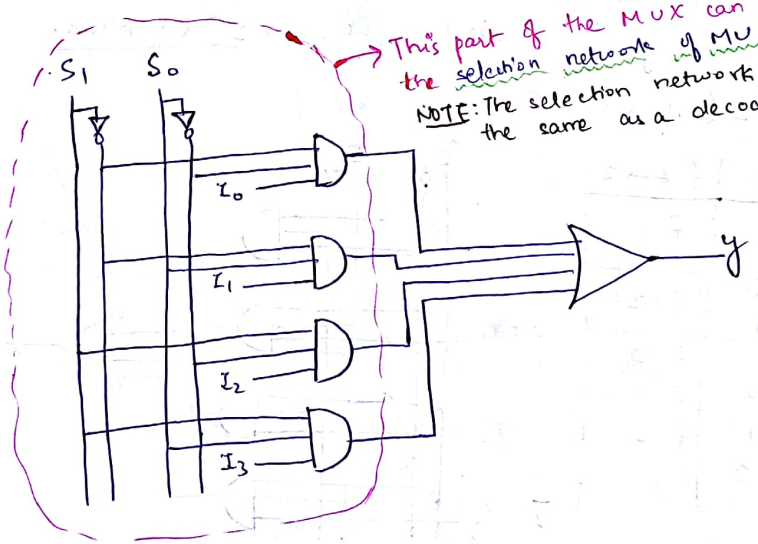


4x1 MUX :



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

$y = \bar{S_1}\bar{S_0}I_0 + \bar{S_1}S_0I_1 + S_1\bar{S_0}I_2 + S_1S_0I_3$



This part of the MUX can be termed as the selection network of MUX

NOTE: The selection network of a MUX is exactly the same as a decoder.

(a)

a
0
0
0
0
1
1
1
1

Q7.

E.g. f

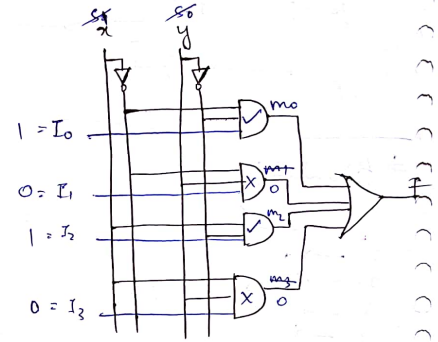
x	
0	0
1	0
2	0
3	0
4	1
5	1
6	1
7	1

Boolean function implementation using MUX:

Approach 1: E.g. $f(x,y) = \Sigma(0,2)$

$n \rightarrow 2^n \times 1$

x	y	f
0	0	1
0	1	0
1	0	1
1	1	0



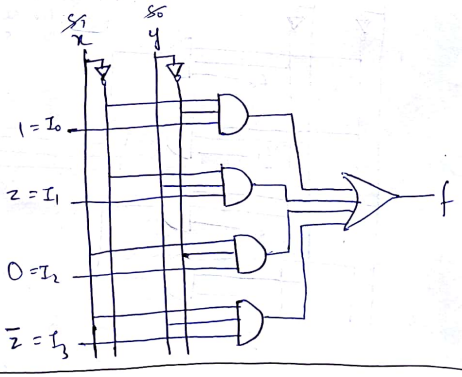
Point-14

Approach 2:

→ An n-variable boolean function can also be implemented using a MUX having only (n-1) select input, i.e., $2^{n-1} \times 1$ MUX.

E.g. $f(x,y,z) = \Sigma(0,1,3,6)$

x	y	z	f	f ⟷ z
0	0	0	1	f=1
0	0	1	1	
0	1	0	0	f=z
0	1	1	1	
1	0	0	0	f=0
1	0	1	0	
1	1	0	1	$f=\bar{z}$
1	1	1	0	



Limitation:

→ The limitation of the above approach is that, while finding the relation between the function 'f' and any 1 variable of the function, if the complemented term (E.g. $\bar{z}$) appears in the relation, then in addition to the MUX circuit, 1 NOT gate is also needed to implement the function.

(a)

a	b	c	f
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

	$\bar{b}\bar{c}$	$\bar{b}c$	bc	$b\bar{c}$
$\bar{a}$	1		1	
a	1	1	1	

$f = \bar{a}\bar{b}\bar{c} + a\bar{b}\bar{c} + a\bar{b}c + \bar{a}bc + abc$

Q7.

P	Q	R
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

$f = \bar{S}_1\bar{S}_0 I_0 + \bar{S}_1 S_0 I_1 + S_1\bar{S}_0 I_2 + S_1 S_0 I_3$

$= \bar{P}\bar{Q}R + \bar{P}Q\bar{R} + P\bar{Q}\bar{R} + PQR$

$= \bar{P}(\bar{Q}R + Q\bar{R}) + P(\bar{Q}\bar{R} + QR)$

$= \bar{P}(Q \oplus R) + P(Q \odot R)$

$= \bar{P}(Q \oplus R) + P(\overline{Q \oplus R})$

$= P \oplus Q \oplus R$

E.g. $f(x, y, z) = \Sigma(0, 1, 3, 6)$

x	y	z	f
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

	I_0	I_1	I_2	I_3
$\bar{z}$	0	2	4	6
z	1	3	5	7
$f \leftrightarrow z$	1	2	0	$\bar{z}$

	I_0	I_1	I_2	I_3
$\bar{y}$	0	1	4	5
y	2	3	6	7
$f \leftrightarrow y$	$\bar{y}$	1	y	0

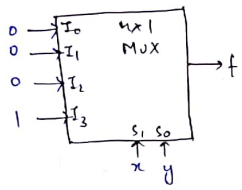
	I_0	I_1	I_2	I_3
$\bar{x}$	0	1	2	3
x	4	5	6	7
$f \leftrightarrow x$	$\bar{x}$	$\bar{x}$	x	$\bar{x}$

Implementation of standard boolean functions (logic gates) using MUX:

Approach 1 Implementation of logic gates using a MUX having n-select lines.

AND Gate:

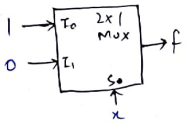
x	y	f
0	0	0
0	1	0
1	0	0
1	1	1



* Similarly, the OR, NAND, NOR, EX-OR, EX-NOR, logic gates can be implemented.

NOT Gate:

x	f
0	1
1	0



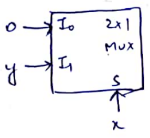
Approach 2 Implementation of logic gates using a MUX having (n-1) select lines.

* This approach is not applicable for NOT gate.

AND Gate:

x	y	f
0	0	0
0	1	0
1	0	0
1	1	1

	I_0	I_1
	0	2
y	1	3
f=y	0	y



$\Sigma(3)$

* **Point - 15**

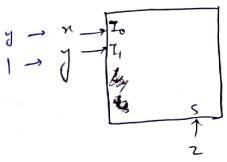
NOTE

↳ The uncomplemented variable (e.g. x) is also sometimes known as True form / Basic form.

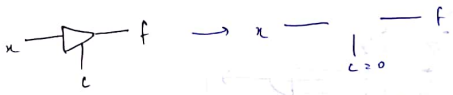
→ The complemented variable (Eg $\bar{x}$) is also known as false form.

Q11. $f = T + R$, $x + y$

- R to x
- 1 to y
- T to z

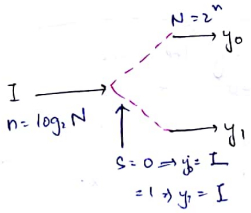


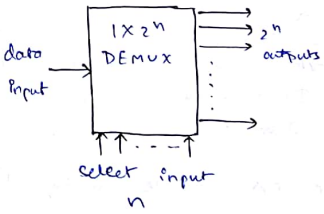
Tristate Buffer:



→ Generally, in a digital circuit, there are 2 states (0 & 1), However, sometimes, with the help of control & input, a third state can be introduced in the system. This is known as high impedance state (high Z state).

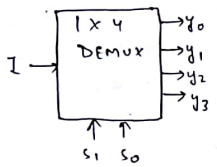
#(4) De Multiplexer (DEMUX)





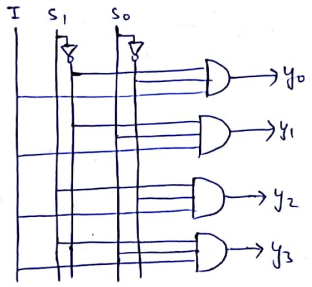
→ A DEMUX is also known as
- Data distributor
- Serial to parallel converter
- One to many circuit.

1X4 DEMUX:



S_1	S_0	y_0	y_1	y_2	y_3
0	0	I	0	0	0
0	1	0	I	0	0
1	0	0	0	I	0
1	1	0	0	0	I

$y_0 = \bar{S_1}\,\bar{S_0}\,I$
$y_1 = \bar{S_1}\,S_0\,I$
$y_2 = S_1\,\bar{S_0}\,I$
$y_3 = S_1\,S_0\,I$



NOTE
→ A decoder with enable input is exactly the same as a DEMUX. Therefore, the circuit is known as decoder-demultiplexer circuit.

* A $n \times 2^n$ decoder is equivalent to a 1×2^n DEMUX.

Implementation of larger size circuits (Decoder, MUX, DEMUX) from smaller size circuits:

AXB Decoder (to implement) → CXD Decoder (required)

1XB DEMUX → 1XD DEMUX

BX1 MUX → DX1 MUX

BX1 MUX → B|D = K_1, K_1|D = K_2, K_2|D = K_3, ..., K_{n-1}|D = K_n = 1

BX1 MUX $\xrightarrow{K_1 K_2 K_3 \cdots K_n}$ DX1 MUX

E.g 256X1 MUX → 4X1 MUX

$\frac{256}{4} = 64$
$\frac{64}{4} = 16$
$\frac{16}{4} = 4$
$\frac{4}{4} = 1$

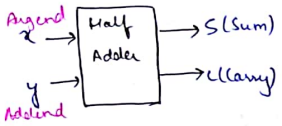
$64 + 16 + 4 + 1 = 85$

*Special Case:
BX1 MUX $\xrightarrow{B-1}$ 2X1 MUX

E.g BX1 MUX $\xrightarrow{B-1}$ 2X1 MUX ⇒ 7 [2X1 MUX] required

#⑤ Half-Adder:

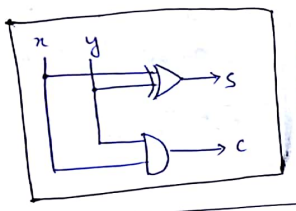
→ Adds 2-bits.



x	y	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

$S = x \oplus y$

$C = x \cdot y$



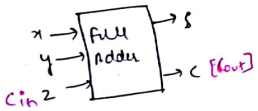
→ To implement a half adder:

(16) • No. of NAND gates needed = 5

(17) • No. of NOR gates needed = 5

#⑥ Full-Adder:

→ Adds 3 bits



	x	y	z	S	C
0	0	0	0	0	0
1	0	0	1	1	0
1	0	1	0	1	0
2	0	1	1	0	1
1	1	0	0	1	0
2	1	0	1	0	1
2	1	1	0	0	1
3	1	1	1	1	1

$S = \Sigma(1,2,4,7)$
$= \bar{x}\bar{y}z + \bar{x}y\bar{z} + x\bar{y}\bar{z} + xyz$
$= x \oplus y \oplus z$

$C = \Sigma(3,5,6,7)$
$= \bar{x}yz + x\bar{y}z + xy\bar{z} + xyz$

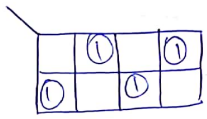
* The sum function acts as an odd function.

* The carry acts out as a majority function.

→ In a majority function, the o/p is logic 1, when majority number of inputs are logic 1.

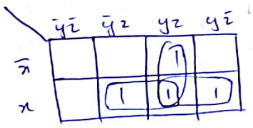
*K-map simplification of sum and carry:

Sum



* Simplification is not possible for the sum output.

Carry

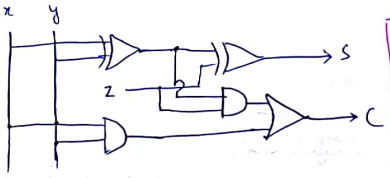


$C = xz + yz + xy$
$= xy + yz + zx$

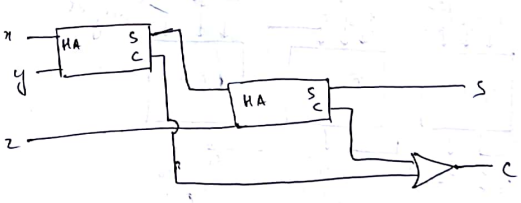
#Implementation of Full adder from two half adders:

$S(x,y,z) = \Sigma(1,2,4,7)$
$= \bar{x}\bar{y}z + \bar{x}y\bar{z} + x\bar{y}\bar{z} + xyz$
$= \bar{x}(\bar{y}z + y\bar{z}) + x(\bar{y}\bar{z} + yz)$
$= \bar{x}(y \oplus z) + x(y \odot z)$
$= \bar{x}(y \oplus z) + x(\overline{y \oplus z})$
$\bar{A} \cdot B + A \cdot \bar{B}$
$= x \oplus y \oplus z$

$C(x,y,z) = \Sigma(3,5,6,7)$
$= \bar{x}yz + x\bar{y}z + xy\bar{z} + xyz$
$= z(\bar{x}y + x\bar{y}) + xy(z + \bar{z})$
$= z(x \oplus y) + xy$



* One full adder is equivalent to 2 half adders + 1 OR Gate



NOTE

→ To implement a full adder:
- No. of NAND gates needed = 9
- No. of NOR gates needed = 9

(7) Parallel Adder (Ripple Carry Adder):

→ Adds 2 n-bit binary numbers.

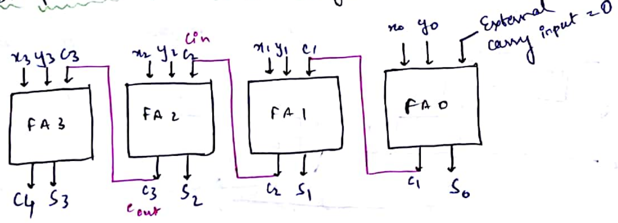
$$\begin{array}{ccccc} C_4 & \overset{C_3}{x_3} & \overset{C_2}{x_2} & \overset{C_1}{x_1} & x_0 \\ \downarrow & y_3 & y_2 & y_1 & y_0 \\ \hline C_4 & S_3 & S_2 & S_1 & S_0 \end{array}$$

* Requirements for 4-bit addition:

→ 3 FA and 1 HA
→ 4 FA
→ 7 HA and 3 OR gates

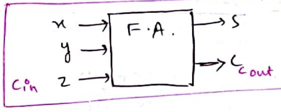
∴ In general, for n-bit addition
→ (n−1) FA and 1 HA
→ n FA
→ (2n−1) HA and (n−1) OR gates

* 4 bit parallel adder (Ripple carry adder):



→ In the above circuit, the carry bit propagates from the LSB to the MSB position in a ripple like manner.

→ It is because of this reason, this circuit is known as ripple carry adder.

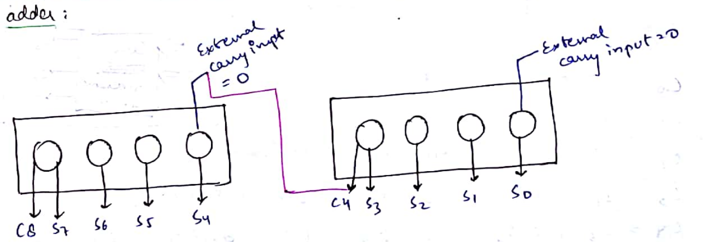


Cin → Carry coming from previous state
Cout → Carry going to the next stage.

~~Cin → Carry~~

Implementation of larger size parallel adder from smaller size parallel adders:

* Implementation of 8-bit parallel adder from 2 4-bit parallel adder:



Q 32 1 FA = 75 ns → S
C = 50 ns

4-bit parallel adder = ~~600 300 ns 50 ns~~

~~$T_s \Rightarrow \frac{1}{500} \times 10^{9} \Rightarrow 2 \times 10^6$~~

~~A-2~~ ~~75 + 50~~

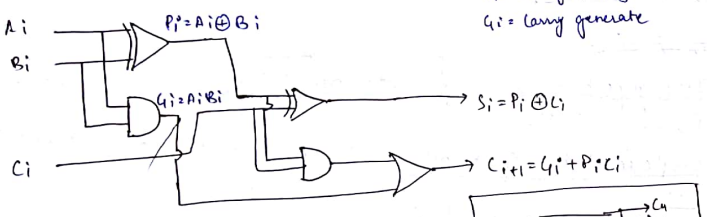
C_4	S_3	S_2	S_1	S_0
(200)	(225)	175	125	75

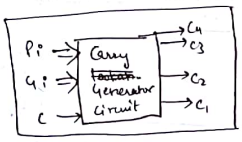
$$\frac{1000}{225} \times 10^6 = 4.45 \times 10^6$$

Parallel Adder: Carry Lookahead Adder:

* Disadvantage of Ripple carry adder: Point -1 8

P_i = Carry propagate
G_i = Carry generate





C_0 = external carry input

$C_1 = G_0 + P_0C_0$

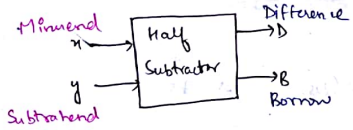
$C_2 = G_1 + P_1C_1 = G_1 + P_1G_0 + P_1P_0C_0$

$C_3 = G_2 + P_2C_2 = G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1P_0C_0$

$C_4 = G_3 + P_3C_3 = G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0 + P_3P_2P_1P_0C_0$

⑧ Half Subtractor:

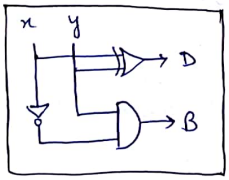
→ Subtracts 2 bits



x	y	D	B
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

$D = x \oplus y$

$B = \bar{x} \cdot y$



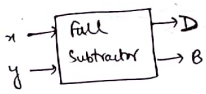
NOTE

→ To implement a half subtractor: no. of NAND gates needed = 5.

→ No. of NOR gates needed = 5

Full Subtractor:

→ Subtracts 3 bits



x	y	z	D	B
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

$D = \Sigma(1,2,4,7)$

$= \bar{x}\bar{y}z + \bar{x}y\bar{z} + x\bar{y}\bar{z} + xyz$

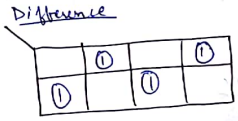
$B = \Sigma(1,2,3,7)$

$= \bar{x}\bar{y}z + \bar{x}y\bar{z} + \bar{x}yz + xyz = \bar{x}(y \oplus z) + yz$

* Difference acts as an odd function.

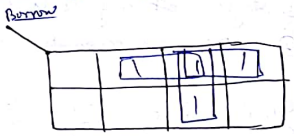
K-map simplification for difference and borrow:

Difference



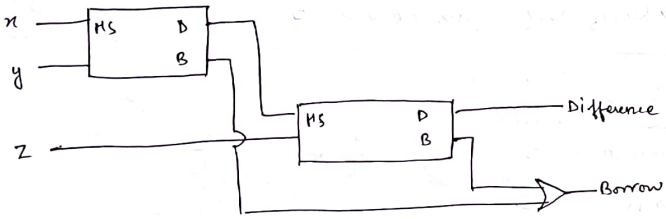
Simplification is not possible for difference output

Borrow



$B = \bar{x}y + yz + \bar{x}z$

Implementation of full subtractor from 2 half subtractors



NOTE
↳ To implement a full subtractor:
- No. of NAND gates needed = 9
- No. of NOR gates needed = 9

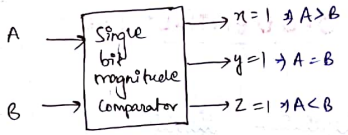
#⑨ Magnitude Comparator:

→ It determines the relative magnitude of 2 binary numbers.

→ It is of 2 types:
① Single bit magnitude comparator
② Multiple bit magnitude comparator

* Single Bit Magnitude Comparator:

→ The 2 inputs are of 1 bit each.



A B	x	y	z
0 0	0	1	0
0 1	0	0	1
1 0	1	0	0
1 1	0	1	0

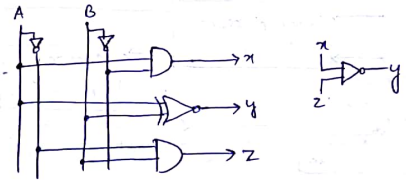
$x = A\bar{B}$

$y = A \odot B = AB + \bar{A}\bar{B}$

$z = \bar{A}B$

$y = \overline{A \oplus B}$

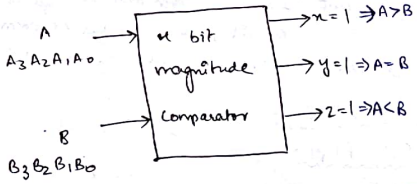
$= A\bar{B} + \bar{A}B$



Multiple Bit Magnitude Comparator:

26/12/2017

4-bit magnitude comparator:



$y_3 = A_3 \odot B_3$

$y_2 = A_2 \odot B_2$

$y_1 = A_1 \odot B_1$

$y_0 = A_0 \odot B_0$

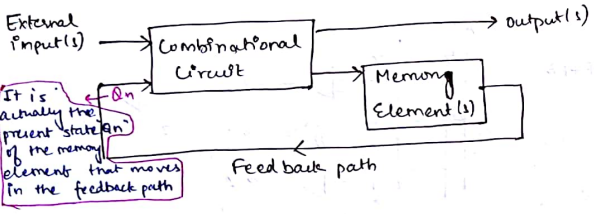
$y = y_3 y_2 y_1 y_0$

$x = A_3\bar{B_3} + y_3 . A_2\bar{B_2} + y_3y_2 . A_1\bar{B_1} + y_3y_2y_1 . A_0\bar{B_0}$

$z = \bar{A_3}B_3 + y_3 . \bar{A_2}B_2 + y_3y_2 . \bar{A_1}B_1 + y_3y_2y_1 . \bar{A_0}B_0$

Chapter-5 SEQUENTIAL LOGIC

Block Diagram of a sequential circuit:



It is actually the present state of the memory element that moves in the feedback path

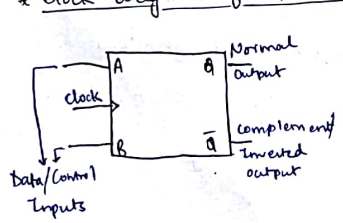
Types of sequential circuits:

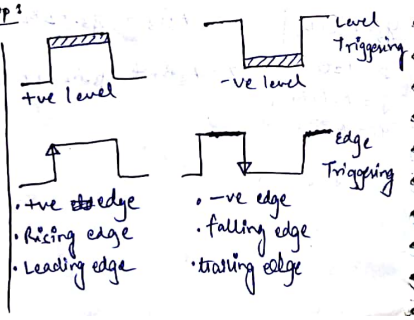
① **Synchronous Sequential Circuit**: In a ~~sequential~~ ~~circuit,~~ synchronous sequential circuit, all the memory elements are activated (triggered) simultaneously by a common activating signal known as a clock pulse.

② **Asynchronous sequential circuit**: In an asynchronous sequential circuit, the different memory elements are triggered at random instants of time independent of each other.

Memory Elements: Flip-Flops:

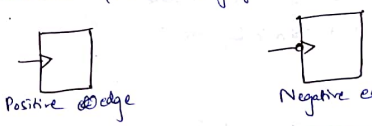
* Block diagram of a flip-flop:





→ Level triggering implies that the digital circuit is activated for a relatively significant amount of time.

→ Edge triggering implies that the digital circuit is activated for a negligible amount of time.



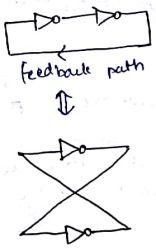
NOTE
↳ It is only under normal/valid operation that the Q and $\bar{Q}$ outputs of a memory element are complements of each other,
whereas $Q = \bar{Q} = 0$ ⇒ Invalid state
and $Q = \bar{Q} = 1$

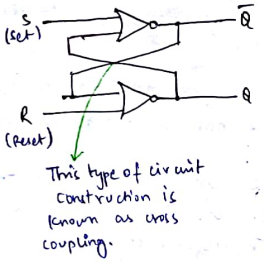
S-R Latch / SC Latch (Set, clear):

→ It consists of two cross-coupled NOR gates (↳NOR latch) or NAND gates (↳NAND latch).

NOTE
↳ The term latch, by default, refers to NAND latch.

NOR Latch: [$\bar{Q}$ infront of S; Q infront of R]





This type of circuit construction is known as cross coupling.

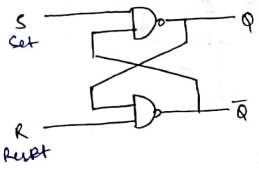
S	R	Q_{n+1}
0	0	Q_n (Hold / Latch)
0	1	0 (Reset)
1	0	1 (Set)
1	1	$Q = \bar{Q} = 0$ (Invalid state)

Function Table (or) Truth Table

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	Invalid state
1	1	1	

Characteristic Table

NAND Latch : [Q infront of S ; $\bar{Q}$ infront of R]



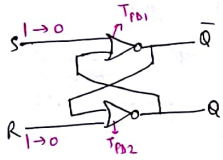
S	R	Q_{n+1}	
0	0	$Q=\bar{R}=1$	Invalid state
0	1	1	set state
1	0	0	reset/clear state
1	1	Q_n	hold state

→ A NAND latch is also known as $\bar{S}\bar{R}$ Latch / $\bar{S}\bar{C}$ Latch.
Because for set state → $\bar{S}$
reset state → $\bar{R}$

* Special Case : Hold state applied immediately after invalid state in a S-R latch [Applicable for both NOR & NAND latches]

→ Below, we understand this special case by considering the example of NOR latch.

S	R	Q	$\bar{Q}$
0	1	0	1
0	0	0	1
1	0	1	0
0	0	1	0
1	1	0	0
0	0	0	0



Practically, T_{PD1} can never be exactly equal to T_{PD2}.

Case 1

S	R	Q	$\bar{Q}$
1	1	0	0
↓			
0	1	0	1
↓			
0	0	0	1

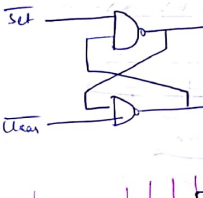
Case 2

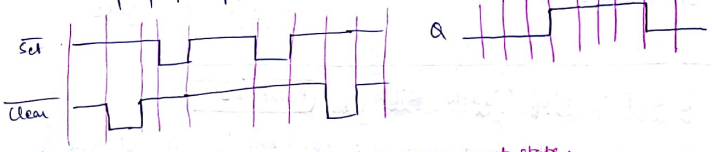
S	R	Q	$\bar{Q}$
1	1	0	0
↓			
1	0	1	0
↓			
0	0	1	0

→ In the above circuit, when the hold state is applied immediately after the invalid state, ideally, the o/p must be equal to "00".
→ In the very first place, we are not able to get this desired o/p.
→ Furthermore, we are also not able to determine the exact value of the output.
→ The only thing, that we can conclude is that either the output is "01" or it is "10". Hence, it is said that the latch enters into an Indeterminate state.

Q. T3 .

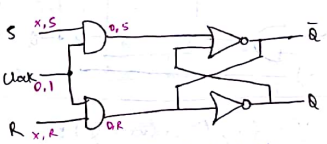
S	R	Q	$\bar{Q}$
1	1	0	1
1	0	0	1
0	1	0	1
0	0	1	1
1	1	0	1
1	0	0	1
0	1	0	1
0	0	1	1
1	1	0	1
1	0	0	1
0	1	0	1
0	0	1	1



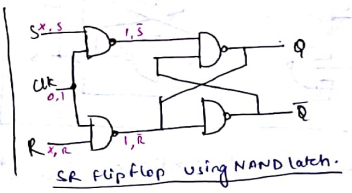


* If no state initial state not given, assume reset state.

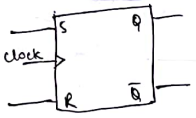
Clocked Latches : Flip Flop :



SR Flip Flop using NOR latch



SR Flip Flop using NAND latch.



Block diagram of SR Flip Flop

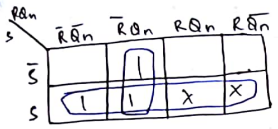
Clock	S	R	Q_{n+1}	
0	X	X	Q_n	
↑	0	0	Q_n	
↑	0	1	0	
↑	1	0	1	
↑	1	1		Invalid

Function Table

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	X
1	1	1	X

Characteristic Table

Since the input "SR = 1 1" drives the flip flop into invalid state, hence, practically "SR = 11" is never applied to a SR flip flop. Hence, it is treated as don't care conditions.



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

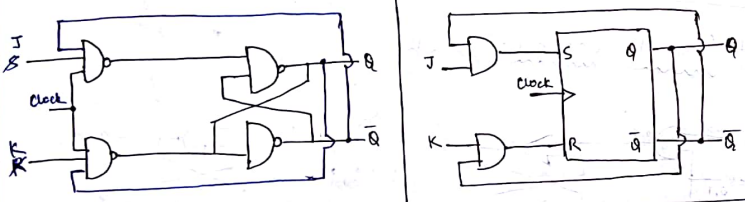
Characteristic Equation

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

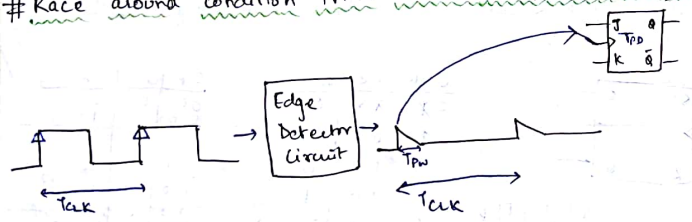
Excitation Table

JK Flip Flop (Jack Kilby): Point-22

→ The JK Flip Flop resolves the problem of invalid state of SR flip flop.



Race around condition in a JK flip flop (Disadvantage):



Let,
T_{PD} = Flip flop propagation delay
T_{PW} = clock pulse width
T_{CLK} = clock time period

→ During the race around condition, the output of the JK flip flop toggles more than once, whereas, ideally, it must toggle, only once.

→ It occurs when:

- J = K = 1 and
- $T_{PW} > T_{PD}$

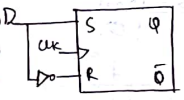
→ This problem is eliminated by ensuring the following condition:

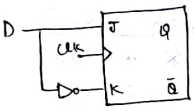
$T_{PW} < T_{PD} < T_{CLK}$

Master Slave → Point-23

D- Flip Flop (Data/Delay):

→ In a D flip flop, the output follows the input (only on the active edges of the clock pulse).





Clock	D	Q_{n+1}
↯	X	Q_n
↑	0	0
↑	1	1

Function Table

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

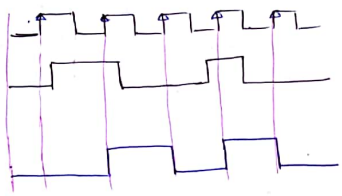
Characteristic Table

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

Excitation Table

Pg 64

Q25. Clock, Input A, Q



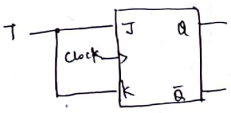
(b)

NOTE

↳ Generally, in a problem related to memory elements, the initial state of the memory elements is given in the problem.

→ However, if it is not given, then, we assume the initial state to be the reset state.

T- Flip Flop (Toggle):



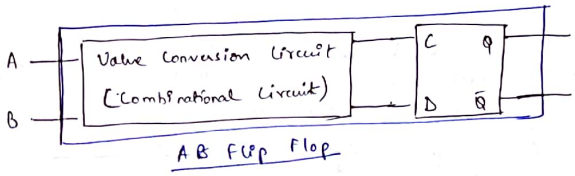
Clock	T	Qn+1
✱	X	Qn
↑	0	Qn
↑	1	Q̄n

T	Qn	Qn+1
0	0	0
0	1	1
1	0	1
1	1	0

Qn+1 = T ⊕ Qn

Qn	Qn+1	T
0	0	0
0	1	1
1	0	1
1	1	0

Conversion Between flip flops: Point 24



AB Flip Flop

A	B	Qn	Qn+1	C	D

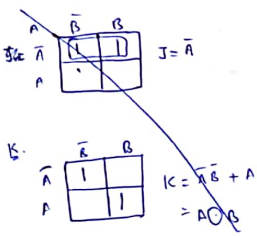
→ Excitation Table of CD

→ Characteristic table of AB flip flop

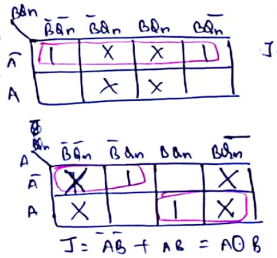
Pg 64 Q.7.

A	B	Qn	Qn+1	I	K
1	0	0	0	0	X
0	1	0	1	1	X
1	1	1	0	X	1
		1	1	X	0

A	B	Qn	Qn+1	J	K
0	0	0	1	1	X
0	0	1	0	X	1
0	1	0	1	1	X
0	1	1	1	X	0
1	0	0	0	0	X
1	0	1	1	X	0
1	1	0	0	0	X
1	1	1	0	X	1



(a)



Chapter - 6 Registers And Counters

→ A register is a group of flip flops.
→ A register consists of a series connection of D-flip flops or other flip flops converted into D-flip flops.
→ A register is always designed as a synchronous sequential circuit.

Types of registers: (Depending upon the method of giving the input and taking output in a register)

ⓐ SISO (Serial in serial out)
ⓑ SIPO (Serial in parallel out)
ⓒ PISO (Parallel in serial out)
ⓓ PIPO (Parallel in parallel out)

NOTE
→ Serial data is also known as Temporal code.
→ Parallel data is also known as Spatial code.

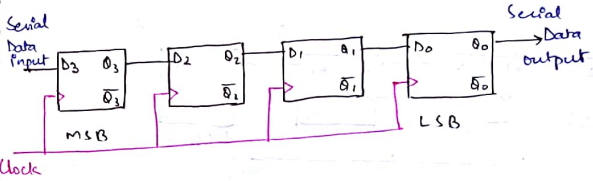
Clock Pulse requirements in an n-bit register:

→ for serial data
  → to input n bits n clock pulses.
  → to output n bits n-1 clock pulses.
→ for parallel data
  → to input n bits 1 clock pulse.
  → to output n bits 0 clock pulse.

26/10/2017

SISO Register:

* 4 bit right shift register:

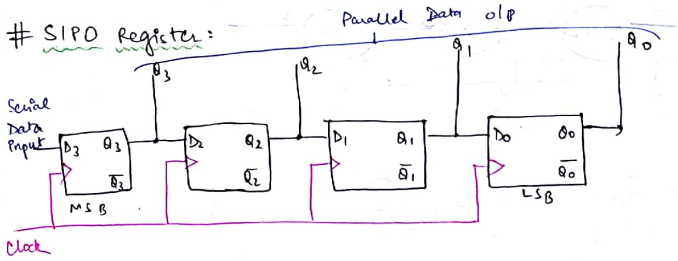


Clock	Q_3	Q_2	Q_1	Q_0
0	1	0	1	1
1	0	1	0	1
2	0	0	1	0
3	0	0	0	1
4	0	0	0	0

NOTE
→ A register in which the o/p of one flip flop is connected to the data input of an adjacent flip flop is known as a shift register.

→ It is of 2 types:
① Right shift register: In this, the data shifts from MSB to LSB.
② Left shift register: In this, the data shifts from LSB to MSB.

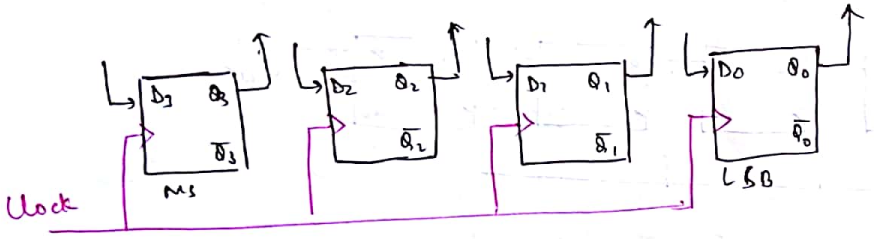
SIPO Register:



NOTE → Converts temporal code to spatial code.

PISO Register : Point 25

PIPO Register :



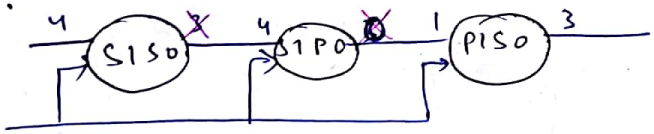
→ PIPO cannot act as shift register.
→ PIPO is the fastest register.

Q34. EE GATE 1992

Clock	Q_3	Q_2	Q_1	Q_0
0	0	1	1	0
1	1	0	1	1
2	0	1	0	1
3	1	0	1	0

(c)

Q39.



= 12 clock pulses.

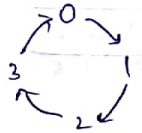
Counters :

→ A counter is a special type of register that goes through a predefined / predetermined sequence of states.

* Types of Counters → Point 26

→ In general, a counter can follow the sequence of states in 3 possible way :

(a) Ascending order : Eg



* Such a counter is known as up counter.

(b) Descending order : Eg

* Such a counter is known as down counter.

(c) Random order : Eg

* Such a counter is known as random counter.

MOD value / Modulo value of a counter :

→ In general, in a counter, If the total no. of flip flops = n and the total no. of used states = N

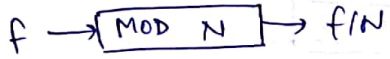
then, $N \leq 2^n$

- → MOD N counter
- → Modulo N counter
- → Divide by N counter
- → N:1 Scalar

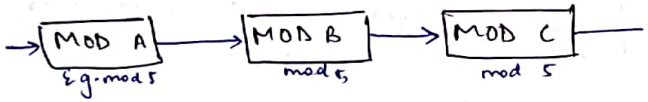
Counter Applications:

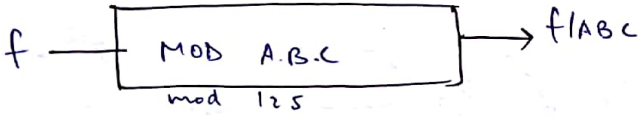
→ To count the no. of clock pulses.

→ As a frequency divider.



Cascading of counters:

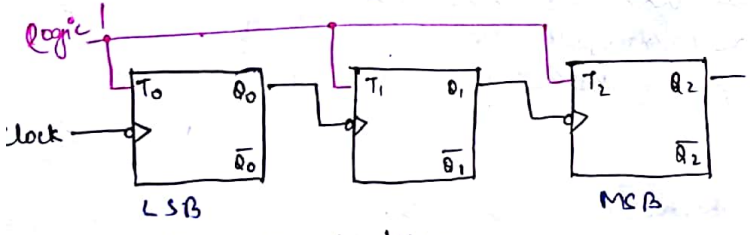




Asynchronous Counters / Ripple Counters:

Point-27

3-bit Up counter: * For down counter take +ve edge trigger.



State Transition Conditions:

→ Q_0 toggles on every clock pulse.

→ Q_1 toggles when Q_0 ~~goes~~ goes from 1 to 0.

→ Q_2 toggles when Q_1 goes from 1 to 0.

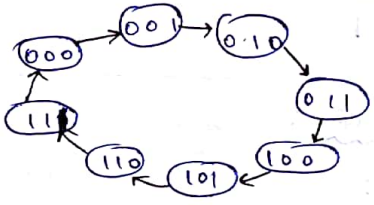
State Transition Table / State Table:

Clock	Q_2	Q_1	Q_0	
0	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	0
9	0	0	1	1
10	0	1	0	2

⇒ MOD 8 counter

⇒ Up counter

State Transition Diagram / State Diagram:

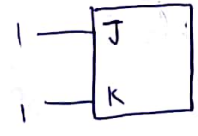


→ In general, the total number of ~~states in the~~ circles in the state diagram of a sequential circuit is equal to the total number of used states in the sequential circuit.

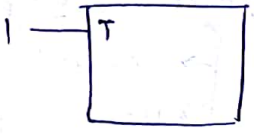
* Timing Diagram: Point-28

* Input frequency is clock.

* Output frequency is MSB flip flop.



is the same as

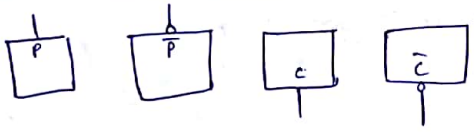


Possible combinations between flip flop output and clock triggering in a ripple counter:

Flip Flop Output		Clock Triggering		Counter Operation	
$\bar{Q}$	0	-ve	0	Down	0
$\bar{Q}$	0	+ve	1	Up	1
Q	1	-ve	0	Up	1
Q	1	+ve	1	Down	0

Implies Truth table of EXOR gate.

Preset & Clear inputs in a flip flop: Point 29



Decade ~~Counter~~ (Ripple) Counter:

→ It is also known as:
- Decimal counter
- BCD [Binary coded decimal] counter
- It's MOD 10 counter (or) divide by 10 counter (or) 10:1 scalar

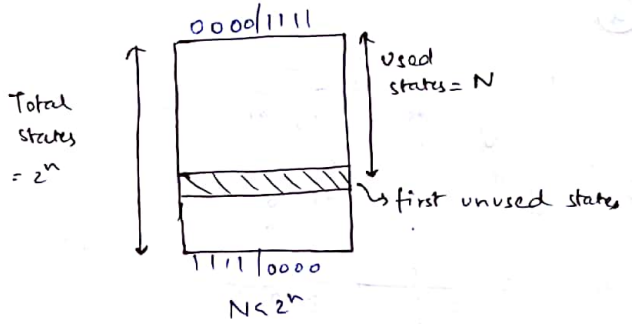
→ It counts a total of 10 states from 0 to 9.

Therefore, total no. of flips flops needed = n = 4

∴ total no. of possible states = 2^n = 16
- → used/valid states = N = 10
- → Unused/Invalid states = 6.

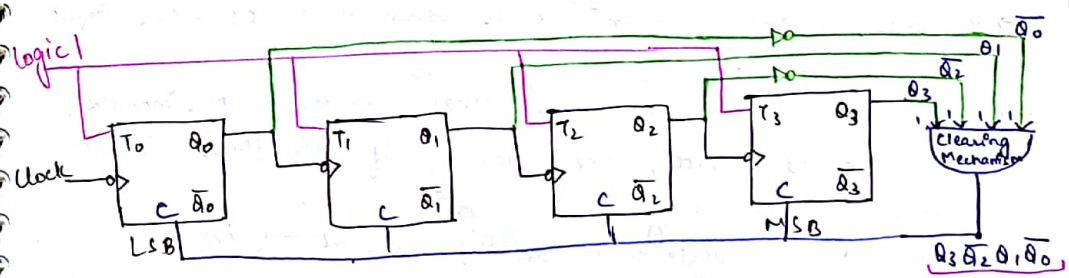
$n < 2^n$

NOTE
→ Underline idea in the design of a counter having unused states.



#Pos → The underline idea for an up counter is to clear all the flipa flops of the counter on reaching the first unused state.

→ The underline idea for a down counter is to preset all the flip flops of the counter on reaching the first unused state.



	Q_3	Q_2	Q_1	Q_0
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0

→ first unused states

→ In the above circuit, the clearing mechanism can also be drawn as follows:

NOTE → While ~~solved~~ solving a problem related to unused states in a counter, the method of solving a problem remains exactly the same for both asynchronous as well as synchronous counters.

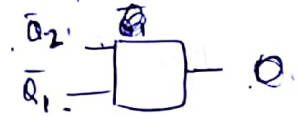
Pg 65.

Q.16.

Q_4	Q_3	Q_2	Q_1	Q_0
1	1	0	0	0

(a)

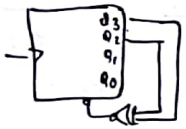
Q.21.



(c)

Q_2	Q_1	Q_0	$\bar{Q}_2$	$\bar{Q}_1$
0	0	0	1	1
0	0	1	1	1
0	1	0	1	0
0	1	1	1	0
1	0	0	0	1
1	0	1	0	1
1	0	0	0	0
1	1	1	0	0

Q.



Q_3	Q_2	Q_1	Q_0
0	0	0	0
0	0	0	1
0	0	1	0
0	0	1	1
0	1	0	0
0	1	0	1

Q. 37
EE Workbook

NOTE

↳ Generally, the preset and clear inputs are ~~asynchronous~~ asynchronous in nature. In this case, the presetting/clearing of the flip flops is performed immediately.

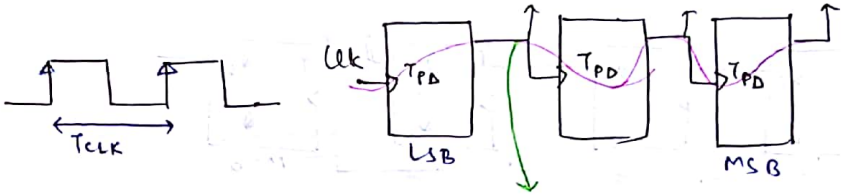
→ However, in the problem it is given that the clear input is Synchronous.

→ Hence, in this case, when the counter enters into the first unused state, the clearing of flip flops is not performed immediately.

→ Rather, it is performed on the arrival of the next clock pulse. As a result of this, the first unused state also effectively becomes one of the used states.

→ Hence, the counter acts as a mod-5 counter and not as a mod-4 counter.

Propagation Delay in a Ripple Counter: Point 30



Let,
n = Total no. of flip flops
T_{PD} = flip flop propagation delay
T_{CLK} = clock time period
f_{CLK} = clock frequency.

- It is a ripple. It is because of this reason an asynchronous counter is also known as ripple counter.
- In a ripple counter, for valid operation.

$$T_{CLK} \geq n \cdot T_{PD}$$

$$\Rightarrow f_{CLK} \leq \frac{1}{nT_{PD}}$$

NOTE

↳ In a ripple counter, to eliminate Decoding errors,

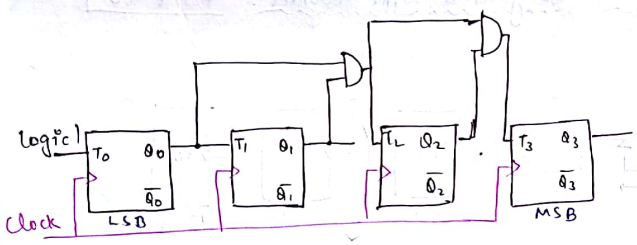
$$T_{CLK} \geq n \cdot T_{PD} + T_s \quad (T_s \rightarrow \text{Strobe Time})$$

$$\Rightarrow f_{CLK} \leq \frac{1}{nT_{PD} + T_s}$$

Synchronous Counter:

* Synchronous Up and Down Counter: Point 31

* 4-bit Up counter:



State Transition Conditions:

→ Q_0 toggles on every clock pulse.
→ Q_1 toggles when Q_0 equals to 1.
→ Q_2 toggles when $Q_1 \cdot Q_0 = 1$
→ Q_3 toggles when $Q_2 \cdot Q_1 \cdot Q_0 = 1$

State Transition Table | State Table:

Clock	Q_3	Q_2	Q_1	Q_0	
0	0	0	0	0	0
1	0	0	0	1	1
2	0	0	1	0	2
3	0	0	1	1	3
4	0	1	0	0	4
5	0	1	0	1	5
6	0	1	1	0	6
7	0	1	1	1	7
8	1	0	0	0	8
9	1	0	0	1	9
10	1	0	1	0	10
11	1	0	1	1	11
12	1	1	0	0	12
13	1	1	0	1	13
14	1	1	1	0	14
15	1	1	1	1	15
16	0	0	0	0	0
17	0	0	0	1	1
18	0	0	1	0	2

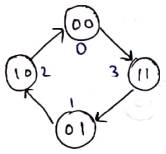
⇒ mod 16 counter
⇒ Up counter

Synchronous Random Counter: ☆

→ A synchronous random counter can be easily/directly designed with the help of any flip flop.
→ It is not necessary for the flip flop to be first converted into a T-flip flop.

* Design of synchronous random counter:

E.g. with the help of D-flip flops, design a synchronous random counter



State diagram

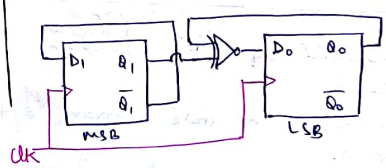
Total no. of states = N
∴ Total no. of flip flops needed = n

Present State		Next State		Flip Flop inputs	
Q_{1n}	Q_{0n}	Q_{1n+1}	Q_{0n+1}	D_1	D_0
0	0	1	1	1	1
0	1	1	0	1	0
1	0	0	0	0	0
1	1	0	1	0	1

State Table

$D_1 = \bar{Q}_{1n}\bar{Q}_{0n} + \bar{Q}_{1n}Q_{0n} = \bar{Q}_{1n}$

$D_0 = Q_{1n} \odot Q_{0n}$

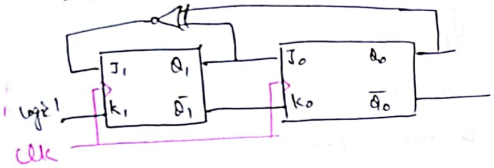


* Design of synchronous random counter having unused states:

Point 33

Analysis of synchronous random counter:

→ Obtain the count sequence of the counter circuit shown below:

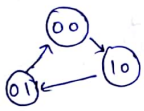


J	K	
0	0	Q_n
0	1	0
1	0	1
1	1	$\bar{Q}_n$

Present State		Next flip flop Inputs				Next State	
Q_1	Q_0	J_1	K_1	J_0	K_0	$Q_{1,n+1}$	$Q_{0,n+1}$
0	0	1	1	0	1	1	0
1	0	0	1	1	0	0	1
0	1	0	1	0	1	0	0
0	0						

State Table

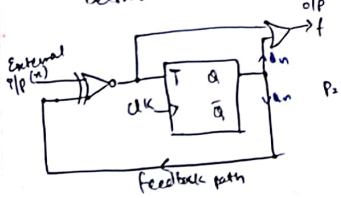
0 → 2 → 1



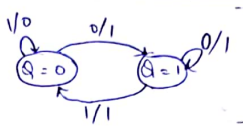
State Diagram

Construction of state diagram of a general sequential circuit:

E.g. Construct the state diagram of the sequential circuit shown below.

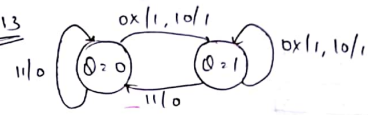


	Input factors		output factors	
	Present state	External Input	Next State	Output
$P_3 \oplus Q_0 x$	Q_n	x	Q	f
1	0	0	1	1
0	0	1	0	0
0	1	0	1	1
1	1	1	0	1



⇒ State Diagram

Q.13



P	Q_n	x	Q_{n+1}	f
0	0	0x, 10	1	1
0	0	11	0	0
1	1	0x, 10	1	1
1	1	11	0	0

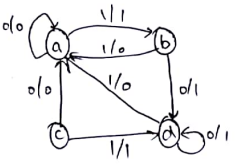
0	0	1
0	1	1
1	0	1
1	1	0

NAND (d)

28/10/2017:

State Reduction: Point 34

E.g. Reduce the state diagram shown below:



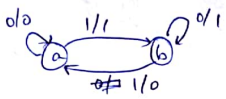
Present State	Next state		Output (f)	
	x=0	x=1	x=0	x=1
a	a	b	0	1
b	d	a	1	0
c	a	d	0	1
d	d	a	1	0

* States b and d are equivalent, ∴ state d is eliminated.

Present state	Next state		Output (f)	
	x=0	x=1	x=0	x=1
a	a	b	0	1
b	b	a	1	0
c	a	b	0	1

* a & c are equivalent.
∴ c is eliminated.

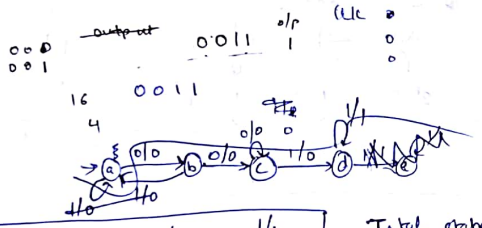
Present state	Next state x=0	Next state x=1	Output (f) x=0	Output (f) x=1
a	a	b	0	1
b	b	a	1	0

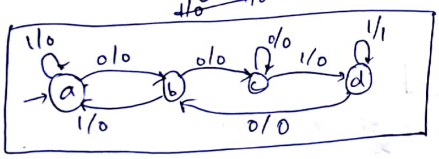


Shift Register Counter, Point 35

Pg 68 Q 40.

output 0011





Total states = 4
Flip flops = 2

$000^n111^n; n \geq 0$

CHAPTER - 7 BOOLEAN ALGEBRA Point 39

Chp 2. I7.

$Y = \bar{A}B + A(B+\bar{C}) + (A+\bar{B})C$

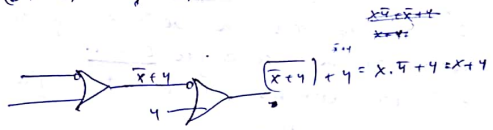
$= \bar{A}B + AB + A\bar{C} + AC + \bar{B}C$

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

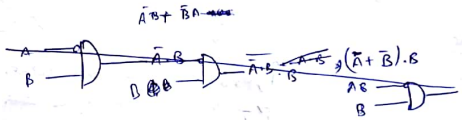
$= (B+\bar{B})(B+C) + A = A+B+C$

I8. $A.(B+CD).E.\bar{F}.G.\bar{H}$

Pg 69



I 10.



CHAPTER-8 NUMBER SYSTEMS

Radix/Base of a number system (r):

→ The radix of a number system defines the total no. of elements present in that number system.

→ The elements of any number system are derived from the decimal system and the English alphabets in such a way that:

* If "r" < 10, the elements of the number system will be the first "r" decimal digits.
* If "r = 10", it becomes the decimal system itself and the elements will be the digits 0 to 9.
* If "r > 10", the elements will be the digits 0 to 9 + the first (r-10) English letters.

Radix Point:

→ In the decimal numbers, 1140
- → known as Decimal Point.
- → In general, known as ~~decimal~~ radix point.
  - → for binary system (r=2), it is a binary point.
  - → for octal system (r=8), it is a octal point.
  - → for hexadecimal (r=16), it is a hexadecimal point.

Conversion of a no. from any radix (r) to Decimal system:

Point 39

Conversion of a decimal number to any radix "r":

① Conversion of integer part:

eg $(53)_{10}$

r	I	
2	Q	R
2	Q	R
2	Q	R
2	0	1

2	53	
2	26	1
2	13	0
2	6	1
2	3	0
2	1	1
	0	1

$(110101)_2$

eg $(422)_{10}$

6	422	
6	70	2
6	11	4
6	1	5
	0	1

$= (1542)_6$

② Conversion of fractional part:

f * r = I.f
f * r = I.f
f * r = I.f
⋮
until f = 0

eg $(0.625)_{10} = (?)_4 = (0.22)_4$

0.625 * 4 = 2.500
0.500 * 4 = 2.000

eg $(0.840)_{10} = (?)_2 = (0.110)_2$

0.84 * 2 = 1.68
0.68 * 2 = 1.36
0.36 * 2 = 0.72
0.72 * 2 = 1.44
0.44 * 2 = 0.88
0.88 * 2 = 1.76
0.76 * 2 = 1.52
0.52 * 2 = 1.04
0.04 * 2 = 0.08
0.08 * 2 = 0.16
0.16 * 2 = 0.32
0.32 * 2 = 0.64

Octal to binary conversion:

→ It is obtained by writing/expressing the individual octal digits in its 3 bit binary equivalent form.

eg. $(76.21)_8 = (111\ 110.\ 010\ 001)_2$

Hexadecimal to binary conversion:

→ It is obtained by expressing the individual hexadecimal digits in its binary form.

eg. $(DF.2C)_{16} = (1101\ 1111.\ 0010\ 1100)_2$

Binary to Octal:

E.g $(011\ 010\ .\ 110\ 010)_2 = (32.62)_8$

Binary to Hexadecimal:

E.g $(0110\ 1010\ .\ 1110\ 0110)_2 = (6A.E6)_{16}$

Q6. $4*4096 + 9*256 + 7*16 + 5$

$4*16^3 + 9*16^2 + 7*16^1 + 5*16^0 = (4975)_{16}$

0100 1001 0111 0101 ⇒ 8 1's. (a)

T5. $(123)_5 \Rightarrow 25 + 2*5 + 3*1$

⇒ 38 = 8 + xy

⇒ xy = 30

$(x8)_y$

x	y
1	30
2	15
3	10
5	6
6	5
10	3
15	2
30	1

2	30
5	15
3	3
	1

x < y

x	y
1	30
2	15
3	10
5	6

y ≥ 9

x	y
1	30
2	15
3	10

Q7. $(E3)_{16}$ = 3+

1110 0011

Octal:

0 1 2 3 4 5 6 7
10 11 12 13 14 15 16 17
20 21 22 23 24 25 26 27
⋮
77
100 ... 107
177
⋮
777
1000

Hexadecimal:

0 1 2 3 4 5 6 7 8 9 A B C D E F
10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F

(d)

```
 F
1 F 10
 2 0 0
- 1 1 D
  D E 3
```

(d)

Q8.

```
 A
 B A
- A B
 0
```

(b)

Q5. ~~$(3.5)_4$~~

```
  1
  2.3
+ 1.2
 10.1
```

0 1 2 3 10 11 12 13
20 21

(a)

Complements:

→ In general, in any number system, there are two types of complements:

① (r-1)'s complement ~~(Radix complement)~~ (Diminished radix complement)

② r's complement ~~(Diminished radix complement)~~ Radix Complement / True Complement

Binary	Octal	Decimal	Hexadecimal
1's	7's	9's	15's F's
2's	8's	10's	16's

→ In general in any radix 'r', the r's complement can be obtained by adding 1 to the (r-1)'s complement.

(r-1)'s Complement:

The (r-1)'s complement of a number N, of radix 'r' and having 'n' no. of digits is given by: $r^n - N - 1$.

NOTE:

→ As a shortcut, instead of applying the above formula, the (r-1)'s complement can be obtained by simply subtracting the individual digits of the given number from (r-1).

→ E.g. Decimal system (9's complement)

$$N = \begin{array}{cccccc} 9 & 9 & 9 & 9 & 9 & 9 \\ 6 & 0 & 9 & 8 & 3 & 0 \\ \hline 3 & 9 & 0 & 1 & 6 & 9 \end{array}$$

Special case:

* Binary system - 1's complement:

→ The 1's complement of a binary number can be obtained by simply changing all the 1's to 0's and 0's to 1's.

E.g. · N = 101100 ⟶ 010011

N = 1001 ⟶ 0110

r's complement:

→ The r's complement of a no. N, of radix 'r' and having 'n' no. of digits is given by, $r^n - N$; when $N \neq 0$.

0; when $N = 0$.

Point 40

E.g. $N = \begin{array}{cccccc} 9 & 9 & 9 & 9 & 10 & \\ 6 & 0 & 9 & 8 & 3 & 9 & 8 \\ \hline 3 & 9 & 0 & 1 & 7 & 0 & 0 \end{array}$ → 10's complement.

E.g. N = 101100 → 2's → 010100

N = 1011 → 2's → 0101

N = 1000 → 2's → 1000

N = 000 → 2's → 000

Subtraction of Positive/Unsigned Numbers using complements:

$$\begin{array}{r} A \\ -B \\ \hline D \end{array} \longrightarrow \begin{array}{r} A \\ +X \\ \hline D \end{array} \rightarrow r\text{'s complement of } B$$

~~A - B~~ D = A − B

= A + X

= A + [r's compl. of B]

OR

= A + [(r-1)'s comp. of B + 1]

Signed Binary number:

→ Unsigned Numbers

5

→ They are always +ve numbers by default.

→ The only exception is the digit 0.

→ Signed Number

+5, −5

→ They are both +ve as well as −ve numbers.

* Sign bit: In case of signed binary numbers, the MSB of the number is used to represent the sign of the number.

→ This bit is termed as the sign bit. Furthermore, as a standard notation:

① Sign bit = 0 ⇒ the binary number is +ve.

② Sign bit = 1 ⇒ the binary number is −ve.

NOTE → The above allocation always remains the same irrespective of the signed binary number representation used.

Signed Binary Number Representations:

- for +ve no. → Signed Magnitude System
- for -ve no. → Signed Complement System
  - 1's complement
  - 2's complement

Signed magnitude system of representation:

E.g: +5 → 0 101 (0 = sign bit, 101 = value bits)

−5 → 1 101 (1 = sign bit, 101 = value bits)

→ In the signed magnitude system, the value bits are simply the binary equivalent of the decimal number to be represented.

Signed complement system representation of a -ve number:

E.g. −5 → start with the corresponding +ve no. → +5 → first find the signed mag. system representation of +5 → 0101

To find the signed 1's complement system representation of −5:

0101 → simply find the 1's comp. of 0101 → 1010

To find the S2'sCS repr. of −5:

0101 → simply find the 2's comp. of 0101 → 1011

Q.12. 0100008 → 101111 (b)

Q.32. 10011 → 2's → 01101 → +13 → 1 1101 = −13

Q.12. −17 → +17 → SMS → 0 10001 → 2's comp. → 01111 (b)

Q.32. 10011

Signed complement system representation of a positive number:

E.g	SMS	S1'sC.S	S2'sC.S
+5	0 101	0 101	0101

→ A positive no. is represented in the S.C.S. in exactly the same manner as it is represented in the S.M.S.

Q. Two nos. x & y are represented in the signed 1's complement system as x "00101" & y "10100". Find their decimal values.

00101 → +5

↳ sign bit is '0'.
∴ it is already in SMS, so we can directly find its decimal value.
∴ x = +5

10100 → 01011 → 01100 → +11 → 11011 = −11

↳ sign bit is '1'.
∴ To find the decimal value, we need to find its complement once again
∴ y = −11

Bit Expansion / Extension in Signed Complement Systems:

→ To extend/increase the number of bits in a number represented in the SC.S., simply rewrite the sign bit of the number repeatedly.

E.g. +ve: 0101 → 0 0 0101

−ve: 1101 → 1 1 1 1101

: Q.18. (c) Q.25 EEWB → (c)

Ranges of different number representations:

→ Unsigned → 0 to $2^n - 1$

→ Signed Magnitude System → $-(2^{n-1}-1)$ to $(2^{n-1}-1)$

→ Signed 1's complement System → $-(2^{n-1}-1)$ to $(2^{n-1}-1)$

→ Signed 2's complement System → $-(2^{n-1})$ to $(2^{n-1}-1)$

unsigned x y z
0 0 0
0 0 1
0 1 0
0 1 1
1 0 0
1 0 1
1 1 0
1 1 1

signed x y z
0 0 0
0 0 1
0 1 0
0 1 1
1 0 0
1 0 1
1 1 0
1 1 1

* Eg. 3 bit.

	+0	−0
SMS	000	100
1CS	000	111
2CS	000	

* The disadvantage of the SMS and 1CS is that they have 2 different representations for the digit "0", i.e., +0 & −0, where This is considered as a disadvantage because in our real world, we have only one representation for the digit "0".

Addition of signed binary number: Point 42

Eg +11 + 4 = +15

0 1 0 1 1 + 0 0 1 0 0 = 0 1 1 1 1

+11 + 4 = +7

0 1 0 1 1 + 1 1 0 0 = (1) 0 0 1 1 1 (discard carry)

−11 − 4 = −15

1 0 1 0 1 + 1 1 0 0 = (1) 1 0 0 0 1 → 2's → 0 1 1 1 → +15

−11 + 4 = −7

1 0 1 0 1 + 0 0 1 0 0 = 1 1 0 0 1 → 2's → 0 0 1 1 1 → +7

1 1 1 1 1 → −7

Detection of overflow in signed 2's complement system: Point 43

E.g. C_4 C_3 C_2 C_1

$0 \leftarrow x_3$ x_2 x_1 x_0

$0 \leftarrow y_3$ y_2 y_1 y_0

$1 \leftarrow s_3$ s_2 s_1 s_0

→ $\bar{x}_3 \bar{y}_3 s_3 + x_3 y_3 \bar{s}_3$

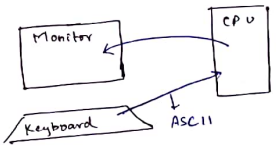
→ $C_4 \oplus C_3$

means overflow.

Complement of a fractional number: Point 44

Chapter - 9 Binary Codes Point 45

ASCII Codes:



Q21. $(843)_{10}$ BCD (1000 0100 0111) (a)

22. b XS-3 (d)

23. (d)

24. 0 → 101 e.t.c.

4 → 111110 (d)

T14. B_3 B_2 B_1 B_0

C_3 C_2

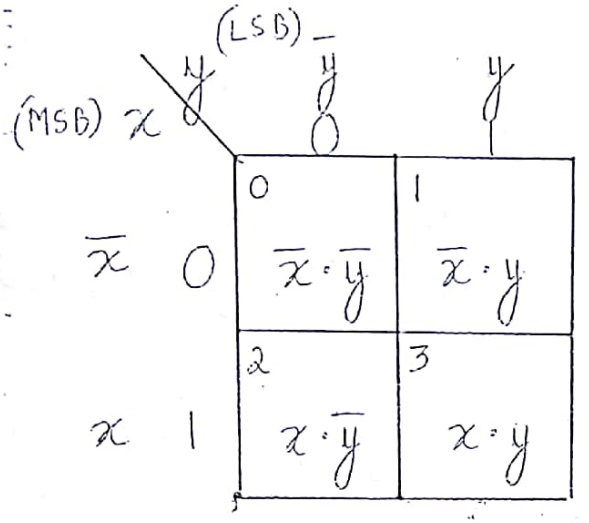
0101 → 0110

0110

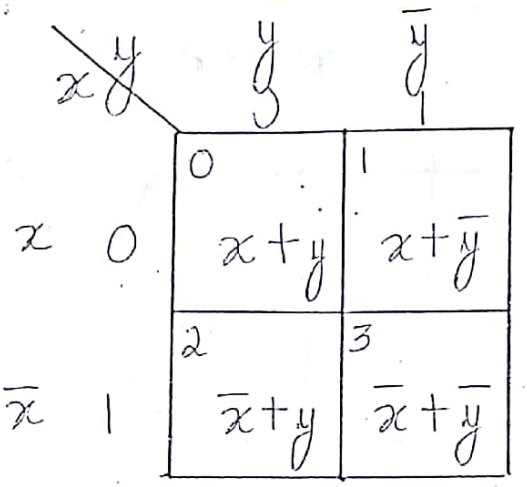
(d) Gray to binary

Kshitij Srivastava (CS)

2 variable K-Map

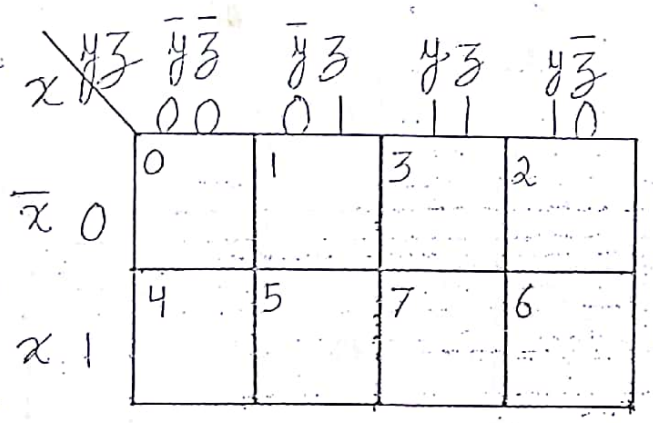


Minterms Map.

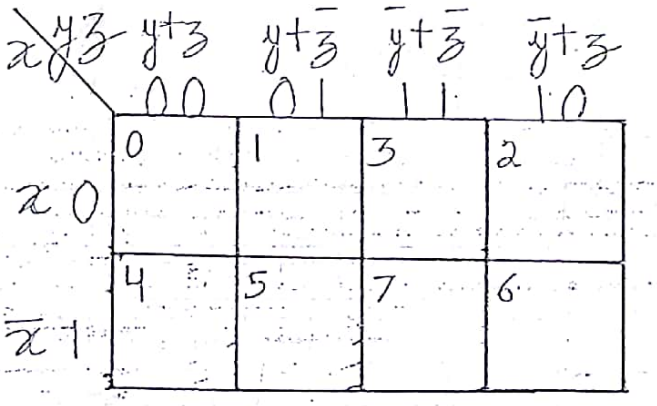


Maxterms Map.

3 variable K-Map



Minterms Map.



Maxterms Map.

4 variable K-Map.

wx \ yz	$\bar{y}\bar{z}$ 00	$\bar{y}z$ 01	yz 11	$y\bar{z}$ 10
$\bar{w}\bar{x}$ 00	0	1	3	2
$\bar{w}x$ 01	4	5	7	6
wx 11	12	13	15	14
$w\bar{x}$ 10	8	9	11	10

Minterms Map.

wx \ yz	$y+z$ 00	$y+\bar{z}$ 01	$\bar{y}+\bar{z}$ 11	$\bar{y}+z$ 10
$w+x$ 00	0	1	3	2
$w+\bar{x}$ 01	4	5	7	6
$\bar{w}+\bar{x}$ 11	12	13	15	14
$\bar{w}+x$ 10	8	9	11	10

Maxterms Map.

5 variable K-Map

$v = 0$ map

wx \ yz	$\bar{y}\bar{z}$ 00	$\bar{y}z$ 01	yz 11	$y\bar{z}$ 10
$\bar{w}\bar{x}$ 00	0	1	3	2
$\bar{w}x$ 01	4	5	7	6
wx 11	12	13	15	14
$w\bar{x}$ 10	8	9	11	10

$v = 1$ map.

wx \ yz	$\bar{y}\bar{z}$ 00	$\bar{y}z$ 01	yz 11	$y\bar{z}$ 10
$\bar{w}\bar{x}$ 00	16	17	19	18
$\bar{w}x$ 01	20	21	23	22
wx 11	28	29	31	30
$w\bar{x}$ 10	24	25	27	26

Minterms Map.

5 variable K-Map.

$v = 0$ map.

wx \ yz	$\bar{y}\bar{z}$	$\bar{y}z$	yz	$y\bar{z}$
$\bar{w}\bar{x}$	0	1	3	2
$\bar{w}x$	4	5	7	6
wx	12	13	15	14
$w\bar{x}$	8	9	11	10

$v = 1$ map.

wx \ yz	$\bar{y}\bar{z}$	$\bar{y}z$	yz	$y\bar{z}$
$\bar{w}\bar{x}$	16	17	19	18
$\bar{w}x$	20	21	23	22
wx	28	29	31	30
$w\bar{x}$	24	25	27	26

Implicants (I), Prime Implicants (PI), Essential Prime Implicants (EPI), Non-Essential Prime Implicants (NPI)

wx \ yz	$\bar{y}\bar{z}$	$\bar{y}z$	yz	$y\bar{z}$
$\bar{w}\bar{x}$				
$\bar{w}x$				
wx				
$w\bar{x}$				

wx \ yz	$\bar{y}\bar{z}$	$\bar{y}z$	yz	$y\bar{z}$
$\bar{w}\bar{x}$				
$\bar{w}x$				
wx				
$w\bar{x}$				

wx \ yz	$\bar{y}\bar{z}$	$\bar{y}z$	yz	$y\bar{z}$
$\bar{w}\bar{x}$				
$\bar{w}x$				
wx				
$w\bar{x}$				

wx \ yz	$\bar{y}\bar{z}$	$\bar{y}z$	yz	$y\bar{z}$
$\bar{w}\bar{x}$				
$\bar{w}x$				
wx				
$w\bar{x}$				

the product term obtained by considering any possible grouping on a K-Map is known as an Implicant (I).

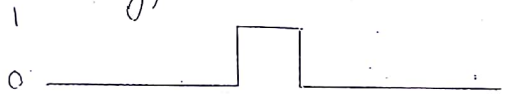
→ the product terms obtained by considering all the biggest possible groupings on a K-Map are known as Prime Implicants (PI).

→ If a PI happens to be the only biggest possible grouping for a particular minterm, then that PI is known as Essential Prime Implicant (EPI).

→ PI minus EPI gives the Non-Essential Prime Implicants. (NPI).

→ If y goes from 1 to 0, the output (f) must remain at logic 1, but it momentarily goes to 0. this is known as Static-1 Hazard. It occurs when the circuit is implemented as a 2-level AND-OR implementation.

Similarly,



If the output must remain at logic 0, but it momentarily goes to logic 1, it is known as Static-0 Hazard. It occurs when the circuit is implemented as a 2-level OR-AND implementation.

General Design Procedure of a Combinational Circuit

→ Identify the number of inputs and outputs.
→ Construct the truth table.
→ Obtain the Boolean expression(s) of the output(s) using either the Minterms approach or the Maxterms approach.
→ Simplify the Boolean expression(s) of the output(s) using either K-Map simplification or Boolean Algebra simplification.
→ Implement the digital circuit using logic gates.

→ Note: In a decoder, at any instant, only one output is at logic 1 (active output) and all other outputs are at logic 0.
∴ an $n \times 2^n$ decoder is also known as 1 of 2^n decoder.

→ Note: A decoder can also be constructed using NAND gates. In that case, the active output is logic 0 and all other outputs are logic 1. ~~Such~~ It is known as a Decoder with Active Low Outputs.

Boolean function implementation using Decoder

→ An $n \times 2^n$ decoder generates all the 2^n possible minterms for n input variables.
→ Any Boolean function can be represented in Sum of Minterms form.
→ ∴ any Boolean function can be implemented with a decoder by Summing / ORing the desired minterms.

---

(13)

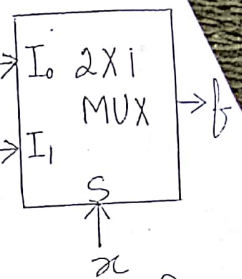
Limitation of Basic Encoder: In a Basic Encoder, at any instant, only one input must be logic 1 and all other inputs must be logic 0; or else, the circuit produces invalid output. this problem is resolved in a Priority Encoder. In a Priority Encoder, at any instant, if more than one inputs are at logic 1, then the encoded output corresponds to the input having the highest decimal value.

(14)

* → To implement an n variable Boolean function, a MUX having n select inputs is needed, that is, a 2^n X1 MUX.

→ Apply,
- → the variables of the function on the select inputs.
- → the truth table outputs of the function on the data inputs.

EXNOR	EXOR	NOR	NAND	OR	AND
$\bar{y}$	y	$\bar{y}$	1	y	0
y	$\bar{y}$	0	$\bar{y}$	1	y

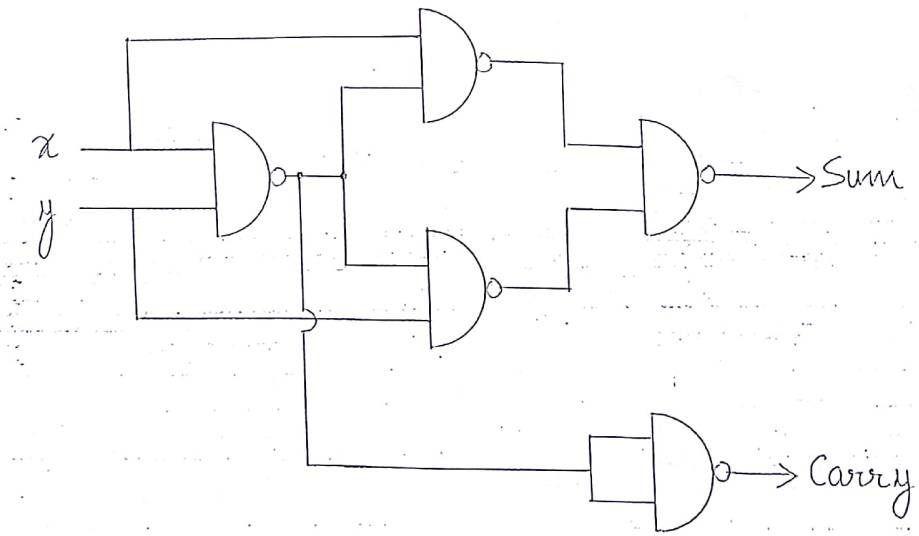


Note: To obtain the NAND, NOR, EXOR, EXNOR gates, two 2X1 MUXes are needed. the first 2X1 MUX is used as a NOT gate to obtain $\bar{y}$ from y.

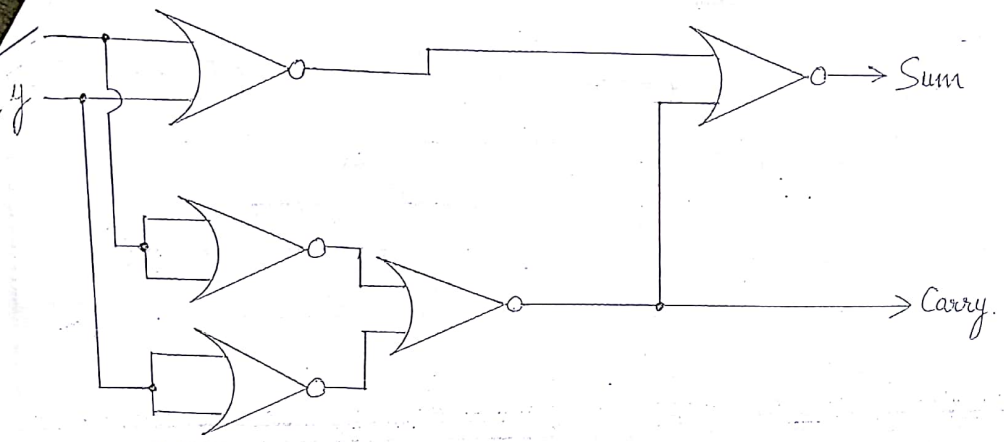
---

(16)

Implementation of Half Adder using NAND gates.



plementation of Half Adder using NOR gates.



Disadvantage of Ripple Carry Adder: In a Ripple Carry Adder, C_{i+1} depends upon C_i, and is generated after C_i passes through an AND gate followed by an OR gate; that is, after a propagation delay of 2 gate levels. ∴ the propagation delay in an n-bit Ripple Carry Adder is 2n gate levels. this problem is resolved in a Carry Lookahead Adder where C_{i+1} is directly generated from the Pi and Gi terms.

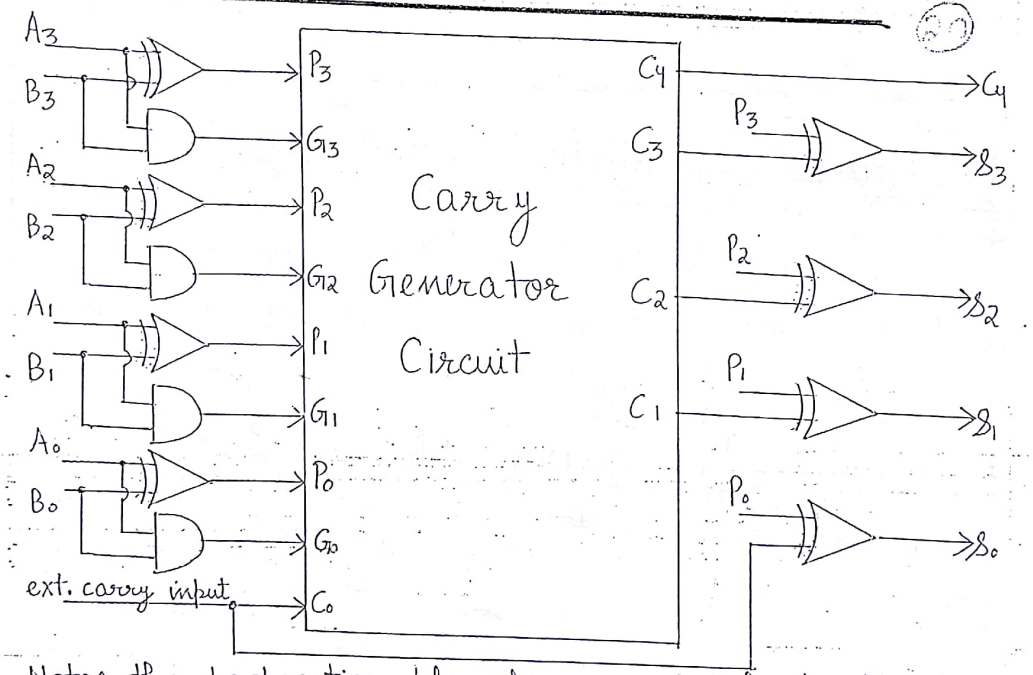
→ the carry ~~ge~~ lookahead logic increases speed of operation but the tradeoff is a highly complex circuit.

→ the carry generator circuit is a 2-level AND-OR implementation.

→ In an n-bit carry generator circuit,

→ No. of AND gates = $\frac{n(n+1)}{2}$

→ No. of OR gates = n



Note: the propagation delay of a carry lookahead adder is always a constant factor, and is independent of the number of bits in the adder.

SEQUENTIAL

The binary data stored in the memory elements is known as the State of the Memory Elements.

→ Previous State → Q_{n-1}

→ Present State → Q_n

→ Next State → Q_{n+1}

→ the Output(s) of a sequential circuit depends upon,

→ Present State ⇒ Moore Sequential Circuit

OR

→ Present State + External Input(s) ⇒ Mealy Sequential Circuit

→ the Next State of a sequential circuit depends upon,

→ Present State and/or External Input(s).

Note: the Moore and Mealy models of sequential circuits are commonly known as Finite State Machines (FSM).

→ Moore FSM

→ Mealy FSM

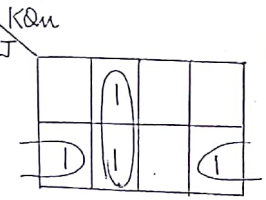
Clock	J	K	Q_{n+1}
✳	X	X	Q_n
↑	0	0	Q_n
↑	0	1	0
↑	1	0	1
↑	1	1	$\overline{Q_n}$

⇓ Toggle state.

Function Table

J	K	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	1
1	1	1	0

Characteristic Table



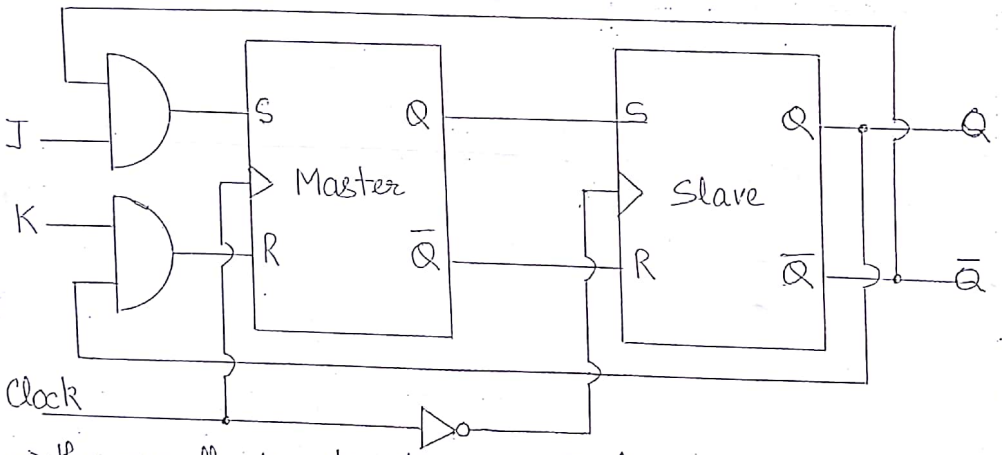
$Q_{n+1} = J \cdot \overline{Q_n} + \overline{K} \cdot Q_n$

Characteristic Equation

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

Excitation Table.

Master Slave JK flip flop (MS JK FF)



→ the overall circuit acts as a single flip flop & stores 1 bit.
→ At any instant, only one flip flop is triggered.
→ the problem of race around condition is not present in a Master Slave JK flip flop.
→ the function tables of Master Slave JK flip flop and JK flip flop are the same.

Standard flip flop conversions.

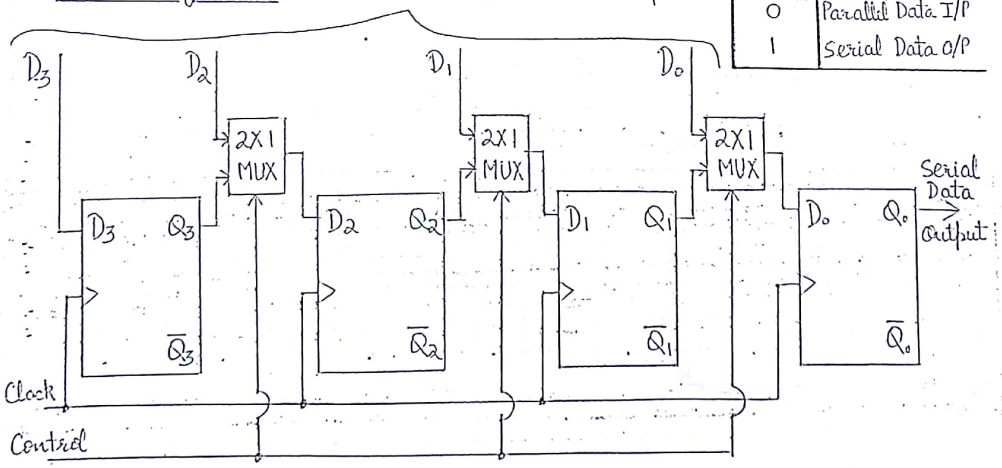
from ⇒ / main ↓	S	R	J	K	D	T
SR	—	—	S	R	$S + \bar{R} \cdot Q_n$	$S \cdot \bar{Q}_n + R \cdot Q_n$
JK	$J \cdot \bar{Q}_n$	$K \cdot Q_n$	—	—	$J \cdot \bar{Q}_n + \bar{K} \cdot Q_n$	$J \cdot \bar{Q}_n + K \cdot Q_n$
D	D	$\bar{D}$	D	$\bar{D}$	—	$D \oplus Q_n$
T	$T \cdot \bar{Q}_n$	$T \cdot Q_n$	T	T	$T \oplus Q_n$	—

Note: To obtain any flip flop from the D flip flop, simply apply the characteristic equation of that flip flop on the D input.

PISO Register

Parallel Data Input

Control	Operation
0	Parallel Data I/P
1	Serial Data O/P



Note: PISO converts Spatial Code into Temporal Code.

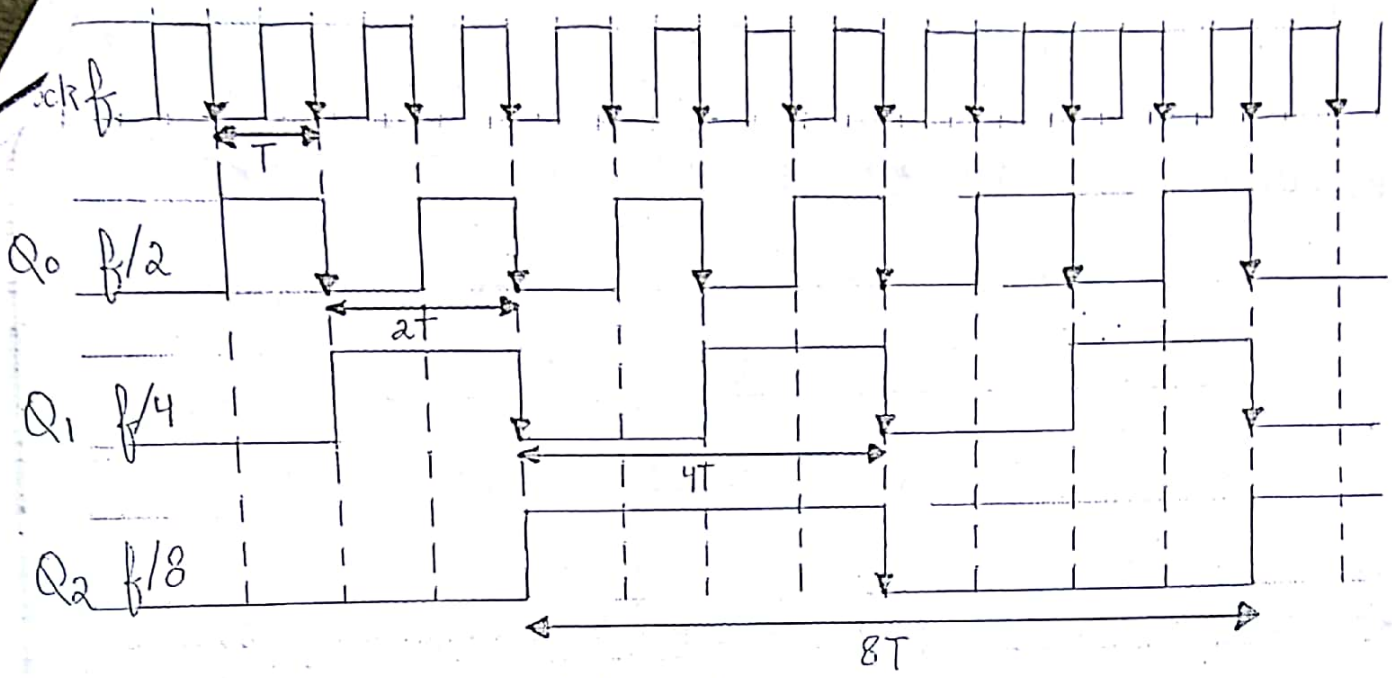
Types of Counters.

Synchronous Counter	Asynchronous / Ripple Counter
→ All the flip flops are triggered by a common clock pulse.	the flip flops are triggered either by clock pulse or by the outputs of adjacent flip flops.
→ Faster	Slower.
→ More complex circuit	→ Less complex circuit.
→ Up, Down and Random counters can be designed.	Only Up and Down counters can be designed.
→ Decoding Errors are not present.	Decoding Errors are present.

General Constructional Details of Ripple Counter.

→ It consists of a series connection of T flip flops (or other flip flops converted into T flip flops).
→ All the T flip flops operate in Toggle mode (⇒ T=1).
→ the clock pulse is applied to only one flip flop. this flip flop represents the LSB of the count sequence.
→ the output of each flip flop is connected to the clock input of the adjacent flip flop.

ming Diagrams.



Note: $f \rightarrow f/2 \rightarrow f/4 \rightarrow f/8$

$\Rightarrow$ frequency scaling by a factor of 2 takes place in a ripple counter as we move from LSB flip flop to MSB flip flop.

Preset and Clear Inputs in a Flip Flop.

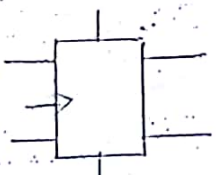
Preset ⟸ Sets the flip flop.

Preset ⟹ 2 types.
- Direct Set
- Asynchronous Set

→ Active High Preset (P)

→ Active Low Preset ($\bar{P}$)

Irrespective of whatever is applied on the data inputs and the clock input of the flip flop.



Clear ⟹ 2 types
- Direct Reset
- Asynchronous Reset ⟸ Resets the flip flop

→ Active High Clear (C)

→ Active Low Clear ($\bar{C}$)

→ In a ripple counter, the outputs of the flip flops are generated one bit at a time, starting from the LSB flip flop. this is because of the cascaded nature of flip flop interconnection. In a synchronous counter, the outputs of all the flip flops are generated simultaneously.
∴ ripple counters are slower than synchronous counters.

General Constructional Details of Synchronous Up and Down Counters.

→ It consists of a series connection of T flip flops (or other flip flops converted into T flip flops).
→ the T flip flop in the LSB position operates in Toggle mode (⇒ T=1).
→ A flip flop in a higher significant position toggles when the outputs of all lower significant flip flops are equal to 1.

Note: In any synchronous counter (Up, Down, Random), the count sequence is independent of +ve or −ve edge triggering.

Note: To obtain Down Counter, simply connect ie complement output ($\bar{Q}$) as the input to the next stages.

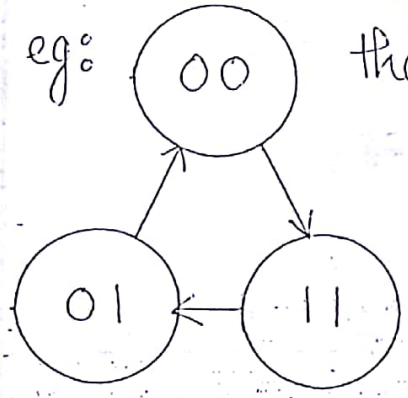
Note: there are two types of Synchronous Up and Down Counters:

→ Series Type Counter: the input of an AND gate is connected to the output of the previous AND gate.

→ Parallel Type Counter: the inputs of all the AND gates are directly connected to the flip flop outputs.

Special Case: Design of Synchronous Random Counter having unused states.

eg:



Unused State = 10

there are 2 methods to deal with unused states

Method 1: Assign any used state as next state for unused state

eg: Assign 00 as next state

PS	NS	FF I/P's
0 0		
0 1		
1 0	0 0	
1 1		

Method 2: Treat them as Don't Care Conditions

PS	NS	FF I/P's
0 0		
0 1		
1 0	– –	X X
1 1		

Note: By default, Method 2 is considered.

State Reduction: It is the process of ~~Keeping~~ reducing the total no. of states in a sequential circuit, while keeping the input-output relationship unchanged.

the steps of state reduction are as follows:

→ Construct the state table.

→ Find out if two states are equivalent states. If so, then eliminate any one of them.

→ Repeat the above step until there are no more equivalent states.

Note: 2 present states are considered to be equivalent states if they produce exactly same next state and same output.

Shift-Register Counter

→ It consists of a series connection of D flip flops (or other flip flops converted into D flip flops.)

→ the output of the last flip flop is connected back to the input of the first flip flop.

→ 2 types.

→ Ring Counter → Johnson Counter

g Counter → Standard example of synchronous random counter.

Also known as,

→ Straight Ring Counter → One-Hot Counter

→ Overbeck Counter

→ An n-bit counter counts n states.

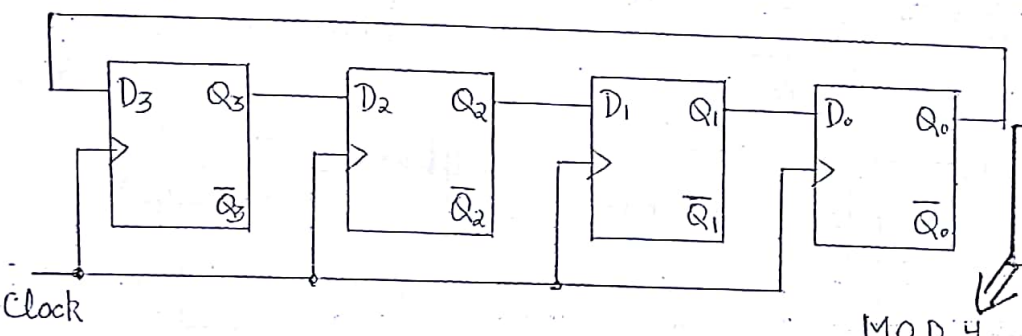
⇒ Used states = N = n.

→ If the input frequency is f, then the output frequency of all the flip flops in the counter is = f/N = f/n.

→ At any instant, only one flip flop is Set and all other flip flops are Cleared. (∵ also known as One-Hot Counter).

→ the Hamming Distance of the used states is 2.

→ Decoding Circuit is not needed



MOD 4 Counter

Clock	Q3	Q2	Q1	Q0
0	1	0	0	0
1	0	1	0	0
2	0	0	1	0
3	0	0	0	1
4	1	0	0	0
5	0	1	0	0
6	0	0	1	0
7	0	0	0	1
8	1	0	0	0
9	0	1	0	0

Initially Preset ↑ Initially Cleared ↓

Johnson Counter

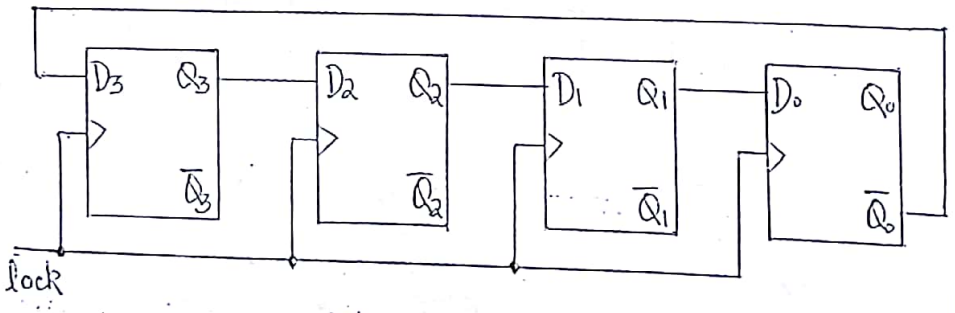
→ Also known as,
- → Twisted Ring Counter
- → Switch Tail Ring Counter
- → Walking Ring Counter
- → Creeping Ring Counter
- → Moebius / Mobius Counter

→ An n-bit counter counts 2n states
- ⇒ Used States = N = 2n.

→ If the input frequency is f, then the output frequency of all the flip flops in the counter = f/N = f/2n.

→ the Hamming Distance of the used states is 1.

→ Decoding Circuit is needed. this circuit consists of a total of 2n logic gates. these logic gates are either 2-input AND gates or 2-input NOR gates.



MOD 8 Counter

Clock	Q3	Q2	Q1	Q0
0	0	0	0	0
1	1	0	0	0
2	1	1	0	0
3	1	1	1	0
4	1	1	1	1
5	0	1	1	1
6	0	0	1	1
7	0	0	0	1
8	0	0	0	0
9	1	0	0	0

Boolean Algebra.

Introduced by George Boole in 1854

⇒ Basic operations of Boolean Algebra

→ NOT
→ AND
→ OR

∴ NOT gate, AND gate and OR gate are known as Basic Logic Gates.

→ Operator Precedence
→ Brackets.
→ NOT
→ AND
→ OR

Note:

→ AND operation is also known as Logical Conjunction

∴ Conjunctive form of Boolean expression ⇒ POS expression

→ OR operation is also known as Logical Disjunction/Alternation

∴ Disjunctive form of Boolean expression ⇒ SOP expression.

Basic theorems of Boolean Algebra.

Identity	Null Element/ Annulment	Idempotent	Complement
→ $A \cdot 1 = A$ → $A + 0 = A$	→ $A \cdot 0 = 0$ → $A + 1 = 1$	→ $A \cdot A = A$ → $A + A = A$	→ $A \cdot \bar{A} = 0$ → $A + \bar{A} = 1$
Involution / Double Negation	**Absorption / Covering / Redundancy**	**De Morgan's**	**Distribution**
→ $\bar{\bar{A}} = A$	→ $A + A \cdot B = A$ → $A \cdot (A + B) = A$	→ $\overline{A \cdot B \cdot C} = \bar{A} + \bar{B} + \bar{C}$ → $\overline{A + B + C} = \bar{A} \cdot \bar{B} \cdot \bar{C}$	→ $A \cdot (B + C) = A \cdot B + A \cdot C$ → $A + B \cdot C = (A + B) \cdot (A + C)$

Consensus theorem

It is applicable on a Boolean expression when,

→ 3 variables are present (eg: A, B, C)

→ each variable is used 2 times (⇒ 6 literals)

→ only 1 literal is complemented (eg: $\bar{A}$)

then the term which contains neither A nor $\bar{A}$ is known as Consensus Term. this term is eliminated

eg: $AB + BC + \bar{A}C = AB + \bar{A}C$

BC ⇓ Consensus Term

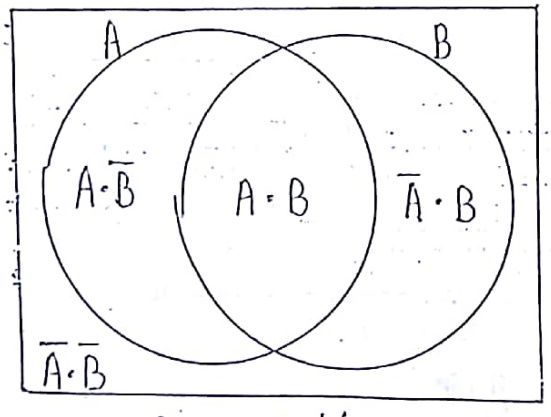
the Consensus theorem is also applicable to POS expressions

eg: $(A+\bar{B})\cdot(B+C)\cdot(A+C) = (A+\bar{B})(B+C)$

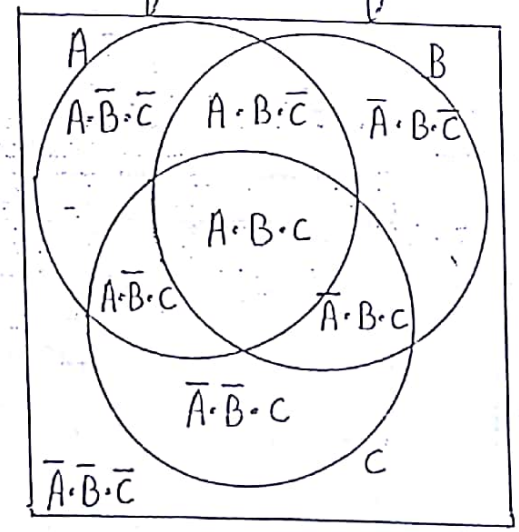
$(A+C)$ ⇓ Consensus Term

Venn Diagram

It is a graphical approach of Boolean function representation

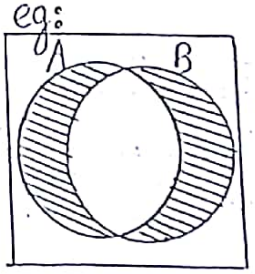


2 variable.



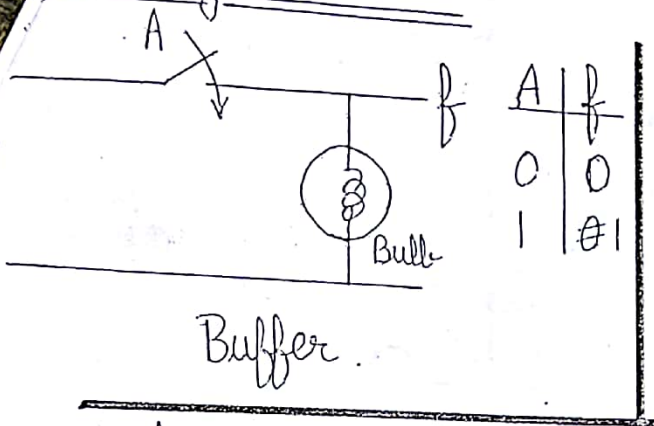
3 variable.

eg:



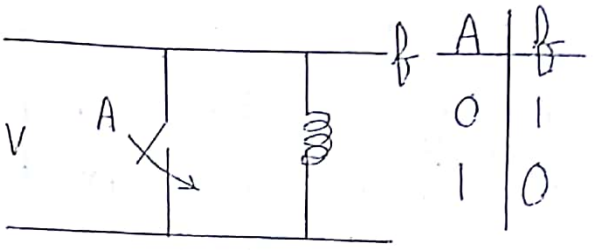
$f = A\cdot\bar{B} + \bar{A}\cdot B$

tching Circuits.



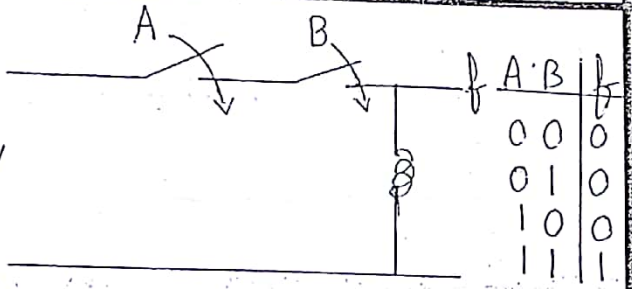
A	f
0	0
1	1

Buffer.



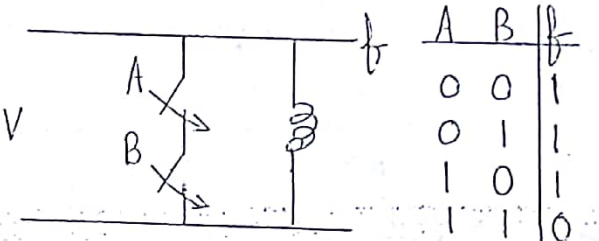
A	f
0	1
1	0

NOT gate



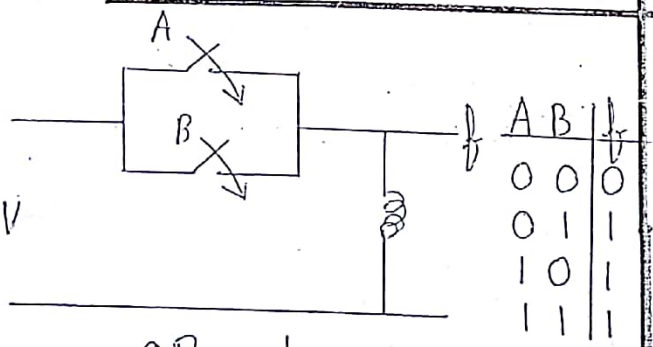
A	B	f
0	0	0
0	1	0
1	0	0
1	1	1

AND gate



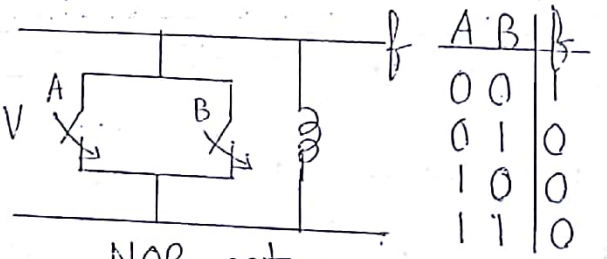
A	B	f
0	0	1
0	1	1
1	0	1
1	1	0

NAND gate.



A	B	f
0	0	0
0	1	1
1	0	1
1	1	1

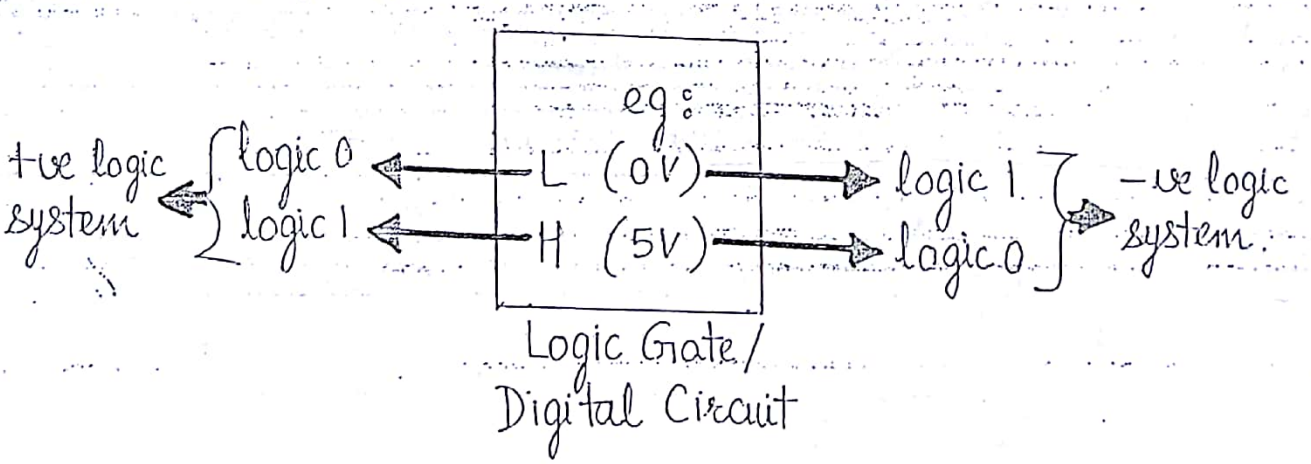
OR gate



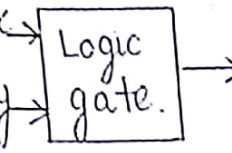
A	B	f
0	0	1
0	1	0
1	0	0
1	1	0

NOR gate

Positive Logic and Negative Logic Systems.



eg:



x	y	f
L	L	L
L	H	L
H	L	L
H	H	H

Assuming +ve logic

x	y	f
0	0	0
0	1	0
1	0	0
1	1	1

AND gate

Assuming -ve logic

x	y	f
1	1	1
1	0	1
0	1	1
0	0	0

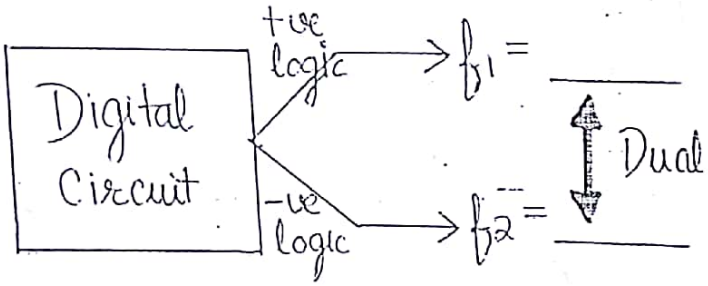
OR gate

⇒ +ve logic AND gate is equivalent to -ve logic OR gate.

Similarly,

+ve logic	-ve logic
AND	OR
OR	AND
NAND	NOR
NOR	NAND
EXOR	EXNOR
EXNOR	EXOR
NOT	NOT

Duality Principle / Dual.



Steps to find Dual:
→ Interchange the AND and OR operations.
→ Interchange the 0's and 1's.

eg: $A \cdot B \cdot \bar{C} + \bar{A} \cdot B \cdot C + A \cdot \bar{B} \cdot C$

↓ Dual

$(A+B+\bar{C}) \cdot (\bar{A}+B+C) \cdot (A+\bar{B}+C)$

$A + 1 = 1$

↓ Dual

$A \cdot 0 = 0$

→ Finding the dual of an expression two times gives back the original expression.

→ Self Dual Expressions. (Not very imp).

→ In self dual expressions, finding the dual only once gives back the original expression.

→ With n variables, a total of $2^{2^{n-1}}$ self dual expressions can be formulated.

Functionally Complete Boolean Operator

A Boolean operator (or a set of Boolean operators) is said to be functionally complete if all possible Boolean functions can be implemented from it.

eg: {NAND}
{NOR}
{AND, NOT}
{OR, NOT}

Number Systems:

Conversion of a no. from any radix (r) to Decimal System

$\rightarrow (2206)_{10} = 2000 + 200 + 0 + 6$
$= (2 \times 10^3) + (2 \times 10^2) + (0 \times 10^1) + (6 \times 10^0)$

∴ in general, a decimal no $a_n \ldots a_1 a_0 . a_{-1} \ldots a_{-m}$ is a shorthand of,
$(a_n \times 10^n) + \ldots (a_1 \times 10^1) + (a_0 \times 10^0) + (a_{-1} \times 10^{-1}) + \ldots (a_{-m} \times 10^{-m})$

∴ in general, for any radix r,
$(a_n \times r^n) + \ldots (a_1 \times r^1) + (a_0 \times r^0) + (a_{-1} \times r^{-1}) + \ldots (a_{-m} \times r^{-m})$

The unique property of the power series expansion of a number (in any radix r) is that, upon solving back, it always gives the decimal equivalent of the no.

eg: $(1101)_2 = (1 \times 2^3) + (1 \times 2^2) + (0 \times 2^1) + (1 \times 2^0)$
$= 8 + 4 + 0 + 1$
$= (13)_{10}$

(40)

Note: the r's complement of a no. in any radix can be obtained by applying the following 3 steps on the no.

→ Leave all the least significant 0's unchanged.
→ Subtract the first, least significant, non-zero digit from r.
→ Subtract all higher significant digits from (r-1).

(41)

Special Case: Binary System (2's complement)

→ Leave all the least significant 0's unchanged.
→ Leave ~~all~~ the first least significant 1 unchanged.
→ Change all higher significant 0's to 1's and 1's to 0's.

(42)

Addition of Signed Binary Numbers.

the steps of adding 2 signed binary numbers are as follows:

→ the +ve operands are represented in Signed Magnitude System.
→ the -ve operands are represented in Signed 2's Complement System.
→ the 2 operands, including the Sign Bit, are added.
→ A carry, if generated, is discarded.
→ the result, if +ve (Sign Bit = 0) is automatically in the Signed Magnitude System.
→ the result, if -ve (Sign Bit = 1) is automatically in the Signed 2's Complement System.

tection of Overflow in Signed 2's Complement system

C_4	C_3	C_2	C_1
	x_3	x_2	x_1 x_0
	y_3	y_2	y_1 y_0
	S_3	S_2	S_1 S_0

the following Boolean expressions detect the occurence of overflow when 2 binary numbers are added in the Signed 2's Comp. system:

→ $\bar{x}_3 \cdot \bar{y}_3 \cdot S_3 + x_3 \cdot y_3 \cdot \bar{S}_3$

→ $C_4 \oplus C_3$

Complement of a fractional no

$(r-1)$ $(r-1)$ $(r-1)$. $(r-1)$ $(r-1)$

− (_ _ _ . _ _) → $(r-1)$'s comp

\+ (_ _ _ . _ 1) → r's comp.

→ $(r-1)$'s complement : Subtract the individual digits of the number (both integer and fractional parts) from $(r-1)$.

→ r's complement : First find the $(r-1)$'s complement. then add 1 to the rightmost digit of the fractional part of the $(r-1)$'s complement.

Binary Codes.

BCD (Binary Coded Decimal)

eg: 7 9 3
↓ ↓ ↓
0111 1001 0011

→ It is obtained by representing the individual digits of a decimal number in its 4 bit binary equivalent form.

→ It is a 4 bit code.
- → Used / Valid Codes : 0000 to 1001
- → Unused / Invalid Codes : 1010 to 1111

→ It is a Weighted Code. the weights of the bit positions are 8421. ∴ BCD code is also known as 8421 BCD code / 8421 code

Note: In a weighted code, each bit position of the code is assigned a decimal number (known as weight); and the decimal value represented by a particular binary sequence in a weighted code is obtained by adding the weights of all those bit positions where a 1 is present in the binary sequence.

eg: 1 1 0 0 ⇒ 6
8 4 2 1
weights.

BCD Addition

→ Valid BCD codes are from 0000 to 1001 ∴ if the sum of 2 BCD numbers is >1001, then it is considered Invalid Sum.

→ (Invalid Sum) + $(0110)_2$ gives the (Valid Sum)

eg:
5 0101
+3 0011
8 1000 ⇨ Valid Sum

9 1001
+4 0100
13 1101 ⇨ Invalid Sum
0110
0001 0011 ⇨ Valid Sum
1 3

2421 code

It is a 4 bit code. The weights of the bit positions are 2-4-2-1.

→ It is a Weighted Code.

→ Some decimal numbers can be coded in 2 ways.

eg: 5 = 0101
= 1011

→ It is a Self-Complementing Code.

eg: 2206 → 1000 0010 0000 1100

9's comp ↑ ↓ 1's comp.

7793 ← 0111 1101 1111 0011

Note: In a self-complementing code, if a decimal number D is represented by a binary code B, then the 1's complement of B directly represents the 9's complement of D.

Note: the following codes are weighted as well as self-complementing:

→ 2421 → 3321 → 4311 → ~~5211~~ → 5211 → 4221

Note: In codes that are weighted + self-complementing the sum of weights is always 9.

Excess-3 Code/Excess-3 BCD Code

eg: 7 9 3
+3 +3 +3
10 12 6
↙ ↓ ↘
1010 1100 0110

It is obtained by first incrementing the individual digits of a decimal number by 3, and then representing them in their 4 bit binary equivalent form.

→ It is a 4 bit code.
  → Used/Valid Codes. : 0011 to 1100
  → Unused/Invalid Codes : (0000 to 0010) & (1101 to 1111)
→ It is an Unweighted Code.
→ It is a Self-Complementing Code.

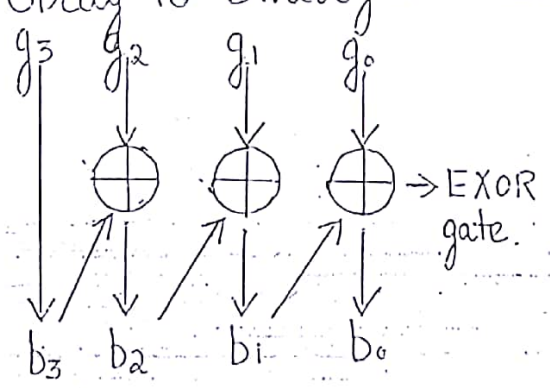
If a code is weighted, it is not necessarily self complementary & vice versa.

Gray Code / Reflected Binary Code

→ Also known as,
  → Minimum Error Code.
  → Cyclic Permutation Code
→ It is an Unweighted code.
→ It is a Unit Distance code.

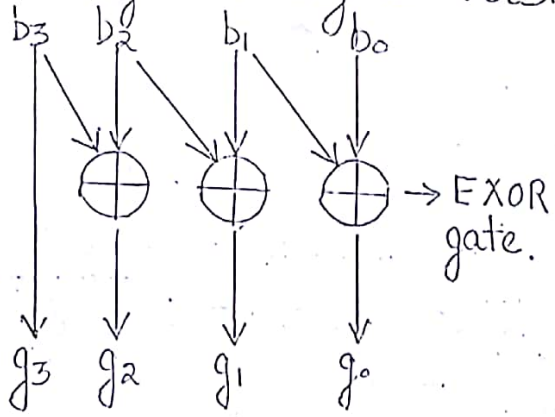
Note: In a Unit Distance code, the binary code corresponding to successive decimal numbers differs by only 1 bit.

Gray to Binary Conversion



eg: G = 110101
⇒ B = 100110

Binary to Gray Conversion



eg: B = 100101
⇒ G = 110111

Note: the MSB of a gray code and its corresponding binary code is always the same.

Error Detecting Code: Parity Bit

→ Parity Bit is an additional bit added to the message bits at the MSB position.

→ It is set to either logic 0 or logic 1 depending upon the no. of 1's in the message bits.

→ 2 types.
- → Even Parity.
- → Odd Parity.

Even Parity: Total no. of 1's in parity bit + message bits is an even no.

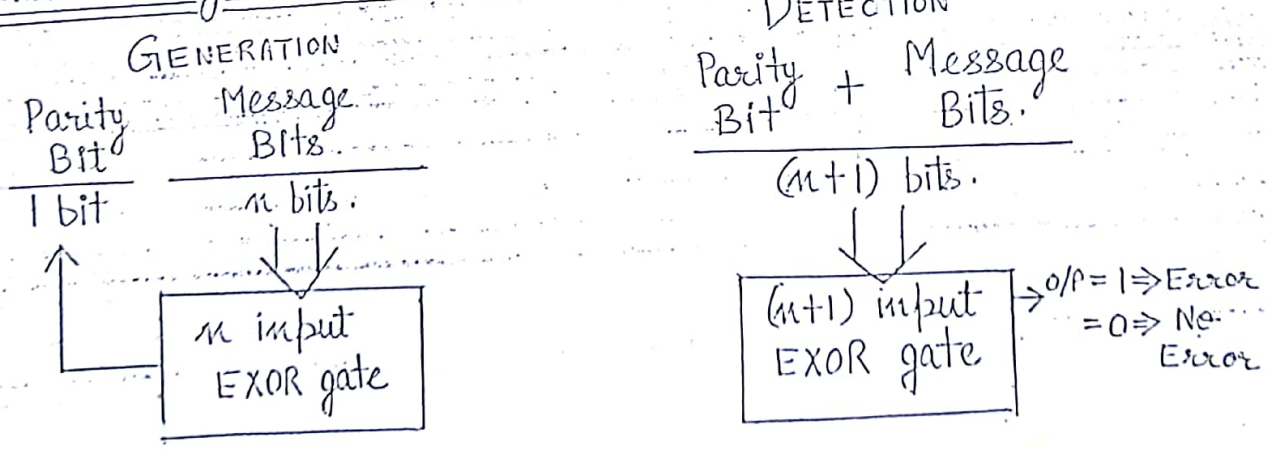
eg: 1101 → 1 1101
1001 → 0 1001
↳ Parity Bit

Odd Parity: Total no. of 1's in parity bit + message bits is an odd no.

eg: 1101 → 0 1101
1001 → 1 1001
↳ Parity Bit

Note: Parity Bit (both even and odd parity) can detect only odd no of errors.

Even Parity: Generation and Detection



ASCII (American Standard Code for Information Interc...

→ Alphanumeric Code.
→ 7 bit code ⇒ 128 characters.
  - → Graphical characters. (94)
  - → Control characters. (34) (Arrow, Caps lock, etc.)

Extended ASCII

→ 8 bit code

---

(46)

→ Memory Size = 2^k X n bits.
→ the n data lines transfer data to/from one complete word at a time.
→ the binary address is applied on the k address lines. An internal k X 2^k decoder selects one of the 2^k words.
→ Read / $\overline{\text{Write}}$ = 0 ⇒ Data Write operation.
= 1 ⇒ Data Read operation.

---

(47)

Types of RAM

SRAM (Static RAM)	DRAM (Dynamic RAM)
→ Data is stored in flip flops.	→ Data is stored in the form of charge on capacitor.
→ Periodic Recharging is not required. ∴ faster in operation (Advantage)	→ Periodic Recharging (Refreshing) is required to restore the charge on capacitor ∴ slower in operation (Disadv)
→ Implemented using both BJT and MOS transistors.	→ Implemented using only MOS transistors.