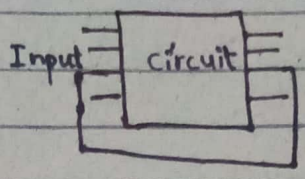
If any other grouping of implicants is not possible and there is only one F_1, then all the prime implicants are essential.

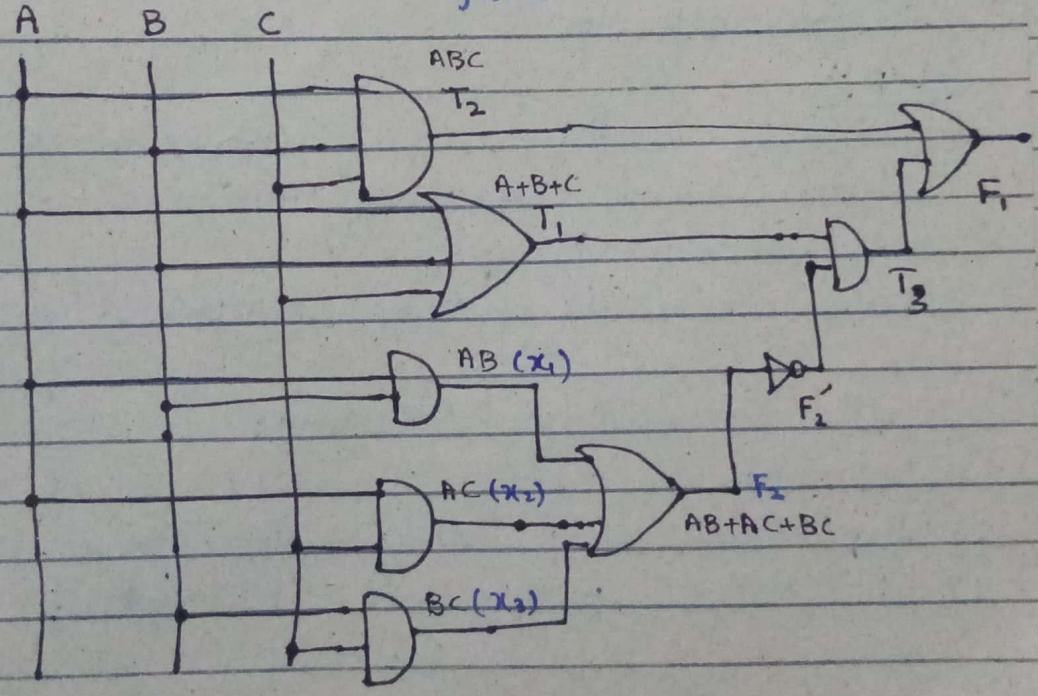
Combinational Circuits:

(other type of Circuit is **sequential Circuit**)



* Output is dependent on input and input is dependent on output.

Combinational Circuits: → circuit which generates output from inputs using gates



$F_1 = T_3 + T_2$

$F_1 = (T_1 \cdot F_2') + T_2$

$F_1 = (A+B+C) \cdot F_2' + T_2$

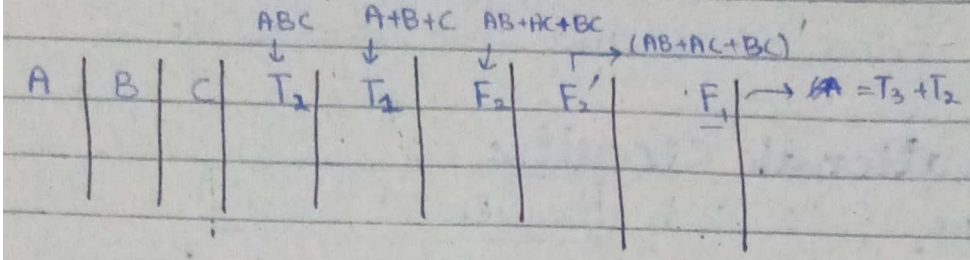
$F_1 = (A+B+C) \cdot (AB+AC+BC)' + T_2$

$F_1 = (A+B+C) \cdot (AB+AC+BC)' + ABC$

$F_2 = (AB+AC+BC)$

$F_2' = (AB+AC+BC)'$

Table can be derived direct by circuit or by Boolean expression.



Requirement → Circuit Design.

↳ See: Input

↳ Output

Step 1:

→ Input, output (Numbers).

Step 2:

→ Assign values (names) $x_1, x_2, x_3 \ldots$

Step 3:

→ Design Truth Table

Step 4:

→ Simplify output from truth table by writing terms against minterms and then simplify it.

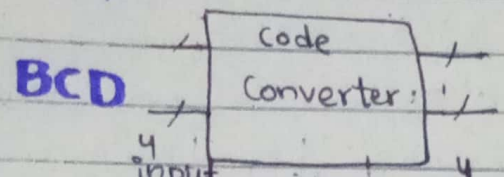
Draw Circuit from simplified output.

Circuit Correctness Checking:

(a) Manually. → (0,1)

(b) By comparing input values from truth table with circuit's output

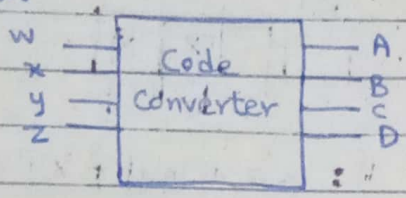
Step1:



bcz BCD is in 4 bits

outputs bcz excess-3 is also in 4 bits

Step#2:



Step 3: Truth Table

	BCD				Excess-3			
	w	x	y	z	A	B	C	D
0	0	0	0	0	0	0	1	1
1	0	0	0	1	0	1	0	0
2	0	0	1	0	0	1	0	1
3	0	0	1	1	0	1	1	0
4	0	1	0	0	0	1	1	1
5	0	1	0	1	1	0	0	0
6	0	1	1	0	1	0	0	1
7	0	1	1	1	1	0	1	0
8	1	0	0	0	1	0	1	1
9	1	0	0	1	1	1	0	0

Min-Terms against A:

$A(w,x,y,z) = \Sigma(5,6,7,8,9)$

Don't care functions:

$d(w,x,y,z) = \Sigma(10,11,12,13,14,15)$

B:

$B(w,x,y,z) = \Sigma(1,2,3,4,9)$

C:

$C(w,x,y,z) = \Sigma(0,3,4,7,8)$

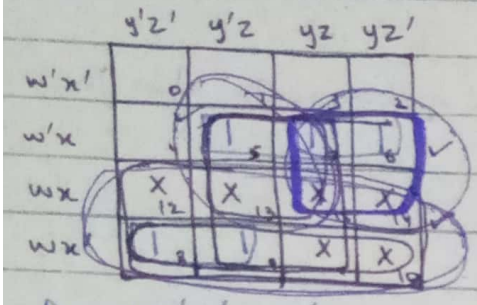
D:

$D(w,x,y,z) = \Sigma(0,2,4,6,8)$

Step 4:

Simplify by K-Map → (to reduce the no. of literals) then implement circuit.

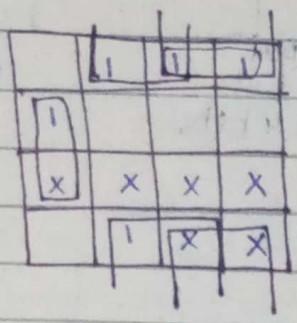
A:



~~A = wx'y' + w'x'z + w'xy~~

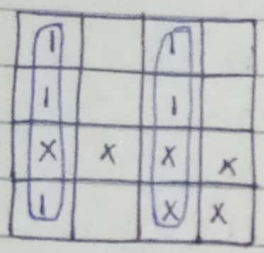
A = xz + xy + w

B:



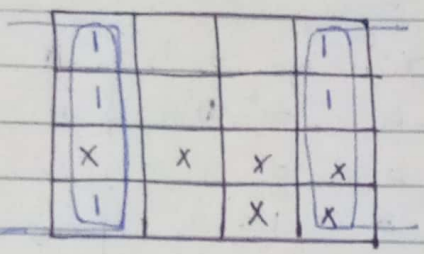
B: x'z + x'y + xy'z'

~~D:~~ C:



C: y'z' + yz

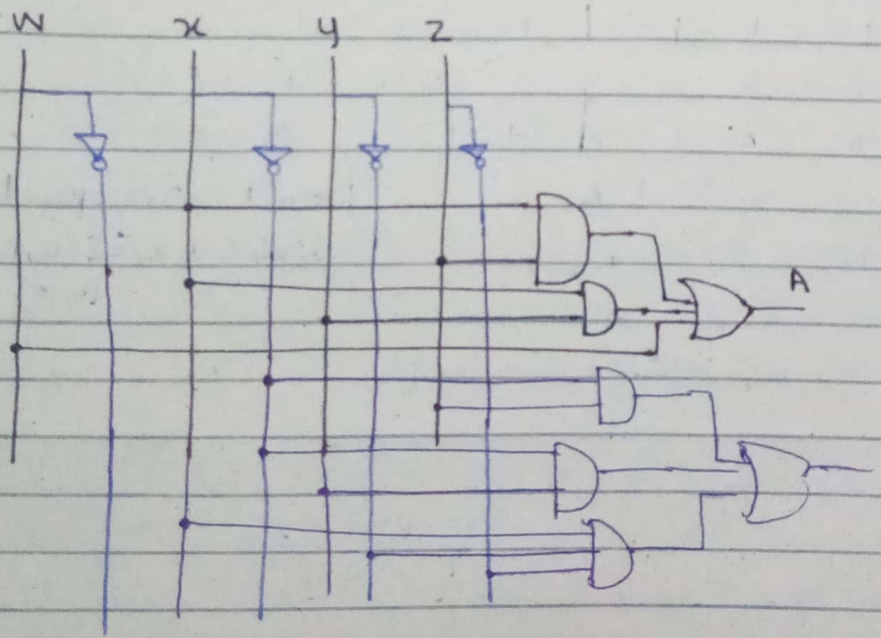
D:



D = ~~xy'z' + yz'~~

D = z'

Designing Circuit:



```
  1          100          100   (3)
& 1        & 000        & 001
  0          000          000
```

```
scanf("%d", &x)
```

```
while(1)
```

"Mostly codings contain 6 don't care combinations."

Design a code converter from 84-2-1 to BCD.

8 W	4 x	-2 y	-1 z		8 A	4 B	2 C	1 D	(BCD)
0	0	0	0	m_0	0	0	0	0	
0	1	1	1	m_7	0	0	0	1	
0	1	1	0	m_6	0	0	1	0	
0	1	0	1	m_5	0	0	1	1	
0	1	0	0	m_4	0	1	0	0	
1	0	1	1	m_{11}	0	1	0	1	
1	0	1	0	m_{10}	0	1	1	0	
1	0	0	1	m_9	0	1	1	1	
1	0	0	0	m_8	1	0	0	0	
1	1	1	1	m_{15}	1	0	0	1	

Min-terms against A:

$A = \Sigma(8, 15)$

Against B:

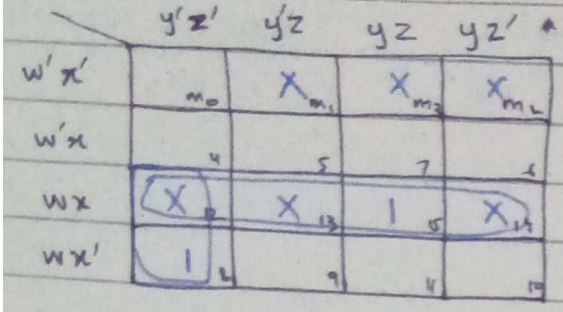
$B = \Sigma(4, 11, 10, 9)$

$C = \Sigma(6, 5, 10, 9)$

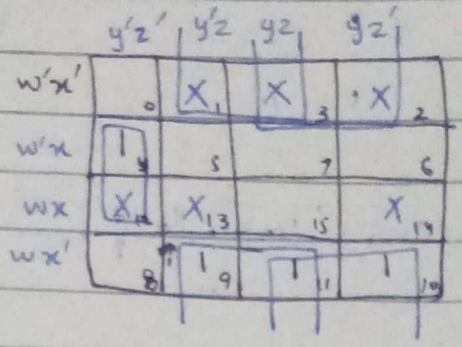
$D = \Sigma(7, 3, 11, 9, 15)$

$d = \Sigma(1, 2, 3, 12, 13, 14)$

A:



B:

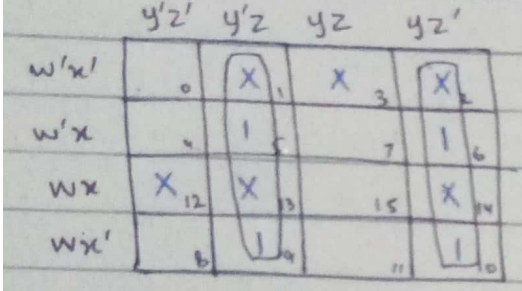


$A = wy'z' + wx$

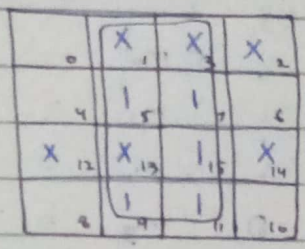
A'

$B = x'z + x'y + xy'z'$

C:



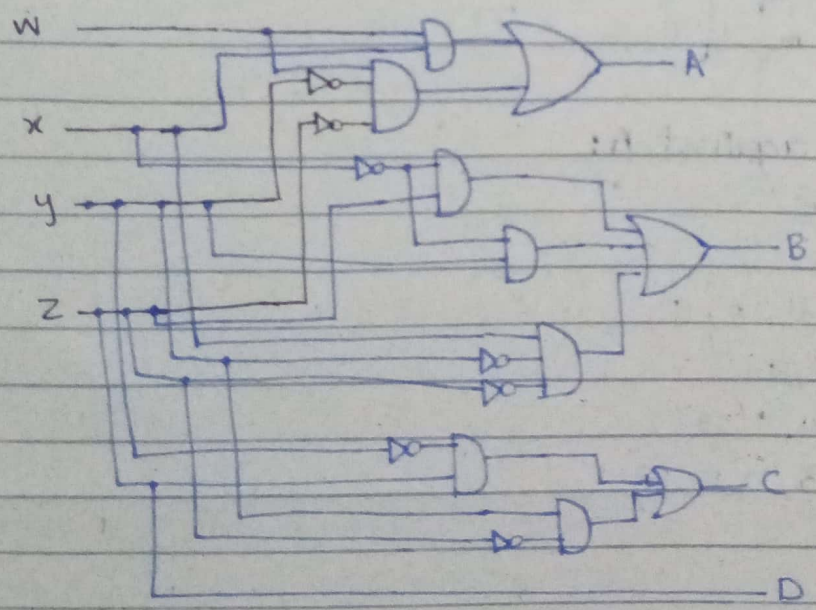
D:



$C = y'z + yz'$

$D = z$

Circuit Diagram:



Design a Half Adder Circuit.

Binary Adder:

(i) This circuit is called half adder bcz it has 2 inputs and 2 outputs

x	y	S (x+y)	C (carry)
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

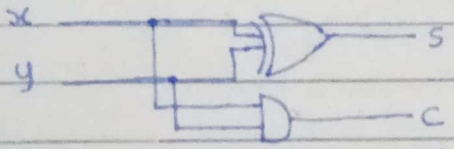
```
  1
+ 1
-----
 ① 0
 ↑
carry
```

For S output and C output, make min-terms by K-map:

$S = x'.y + x.y' = x \oplus y$

$C = x.y$

Half-Adder Circuit:



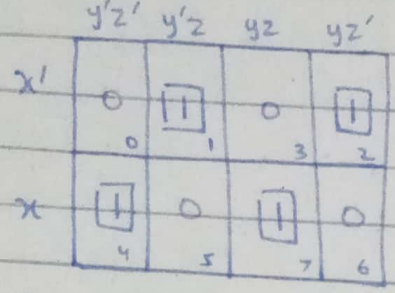
(ii) Full Adder Circuit:

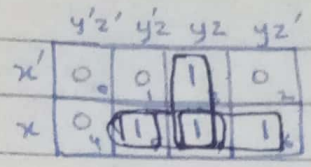
3 inputs → 2 outputs (that's why it is called full-adder Circuit)

x	y	z		S (x+y+z)	C (carry)
0	0	0	m_0	0	0
0	0	1	m_1	1	0
0	1	0	m_2	1	0
0	1	1	m_3	0	1
1	0	0	m_4	1	0
1	0	1	m_5	0	1
1	1	0	m_6	0	1
1	1	1	m_7	1	1

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

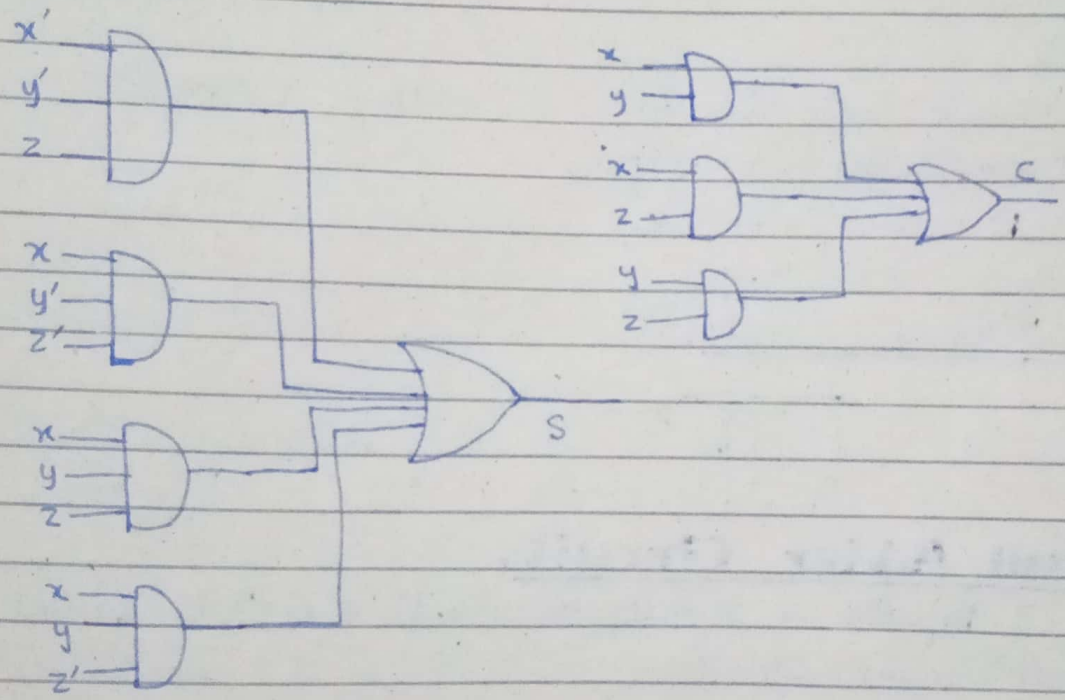
$C = \Sigma(3,5,6,7)$



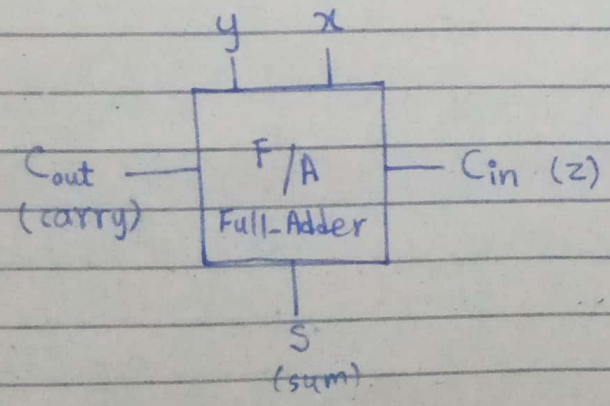


$C = yz + xz + xy$

$S = x'y'z + xy'z' + xyz + xyz'$

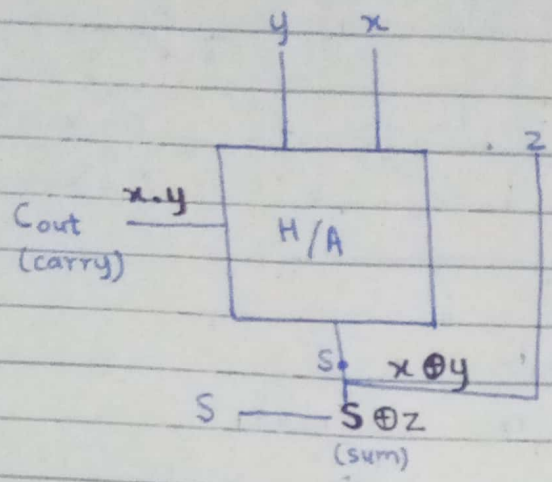


Black Box Representation of full-Adder Circuit:

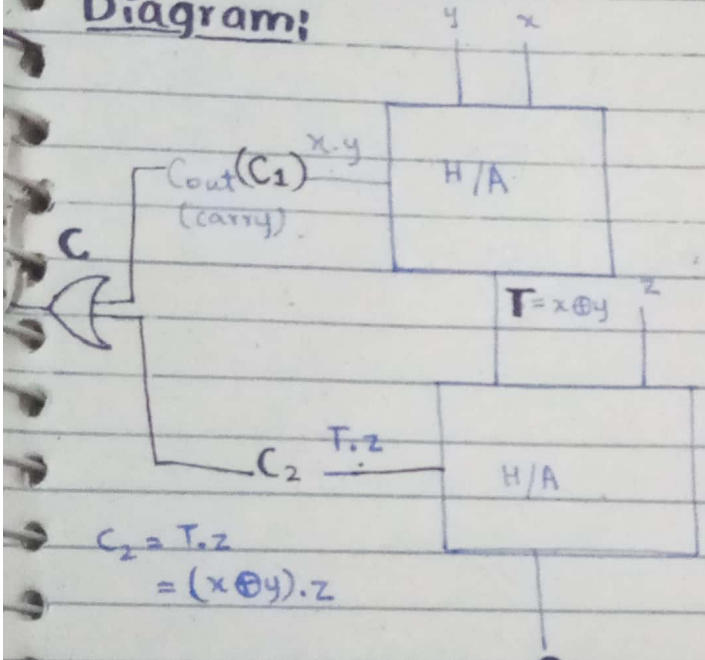


Working of full Adder Using half-Adder:

(2 half-Adders will be used, first for 2 inputs, and H/A for S and z (as input)



Block-Diagram:



$C_2 = T.z$
$= (x \oplus y).z$

0+0+1
↓
S+1
↓
S

$C = C_1 + C_2$
$C = Tz + x.y$
$C = (x \oplus y)z + x.y$

Why OR?
to generate 1 as C only when $C_1=1$ and $C_2=1$ or $C_1=0$ and $C_2=1$ or $C_1=1$ and $C_2=0$

$S = (T \oplus z) = \boxed{(x \oplus y \oplus z)}$
(T)

$S = x'y'z + x'yz' + xyz + xy'z'$

$= x'y'z + xy'z' + x'yz + xyz$

$= z'(x'y + xy') + z(xy + x'y')$

(x'y + xy' : XOR ; xy + x'y' : XNOR ; XNOR = (XOR)')

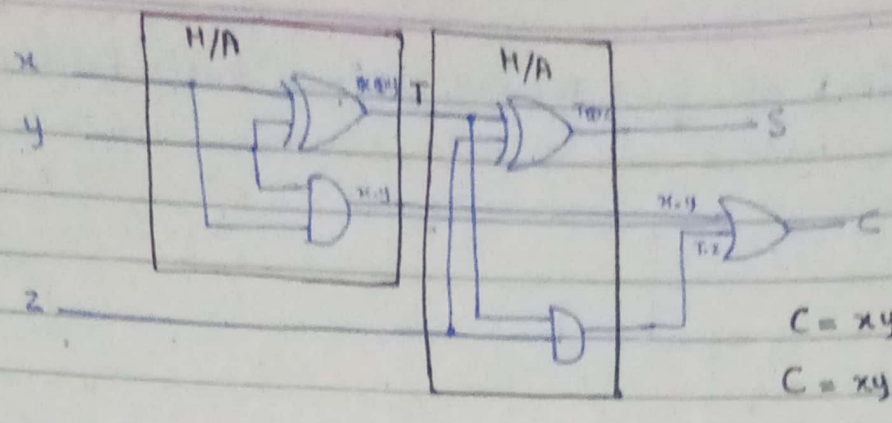
$= z'(x'y + xy') + z(x'y + xy)'$

($x'y + xy' = T = x \oplus y$)

$= z'T + zT'$

$= z \oplus T = \boxed{z \oplus x \oplus y}$ Proved

Circuit Diagram:



Half Subtractor:

2 inputs → 2 outputs

① Truth Table:

x	y		Difference (x-y) D	Borrow in B_{in}
0	0	m_0	0	0
0	1	m_1	1	1
1	0	m_2	1	0
1	1	m_3	0	0

0 − 0 = 0

B_{in} ① → 0

0 − 1 = 1 ①

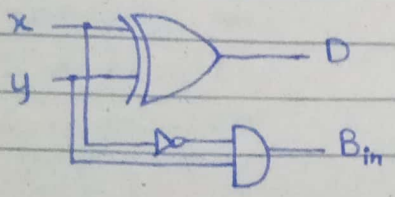
diff.

② Simplification:

$D = x'y + y'x = x \oplus y$ (XOR)

$B_{in} = x'y$

③ Circuit Diagram:



Full Subtractor:

3 inputs generally but here 3 inputs are x, y, Bout (Borrow Out).

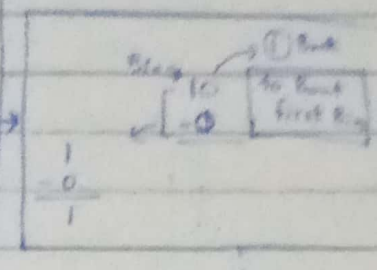
1 gives borrow which is called Bout.

y: 1 0 ← 0 takes borrow which is called B_{in}

− 0 1

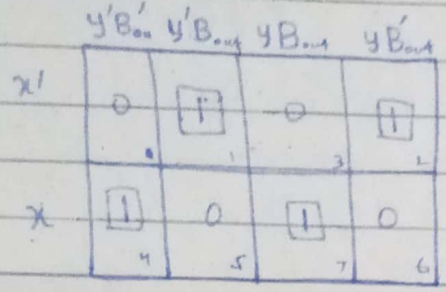
1. Truth Table

x	y	Bout		D	Bin
0	0	0	m_0	0	0
0	0	1	m_1	1	1
0	1	0	m_2	1	1
0	1	1	m_3	0	1
1	0	0	m_4	1	0
1	0	1	m_5	0	0
1	1	0	m_6	0	0
1	1	1	m_7	1	1

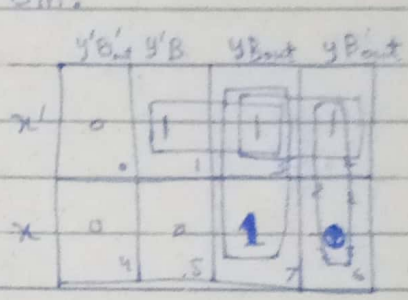


2. Simplification:

D:



Bin:



$D = x'y'B + x'yB' + x'yB + xyB$

$B_{in} = x'B_{in} + yB_{in} + x'y$

↓

For that combination, there is no grouping in k-map.

Then the combination is either exclusive NOR or exclusive OR.

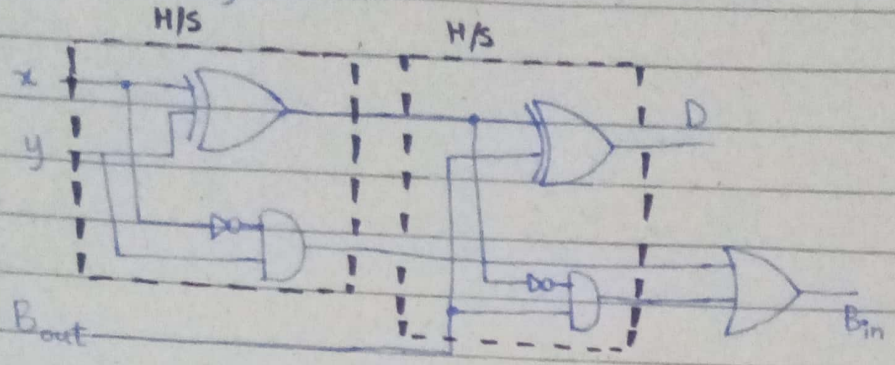
↓

also filling of K-maps is in alternative boxes, even in 4 inputs/4 variables' K-map.

$D = x'y'B + x'yB' + x'yB + xyB$

$D = x \oplus y \oplus B$

3. Circuit Diagram: 2 H/S Combine to give the working of F/S



Binary Adder: → Full adder ~~adders~~ adds only 1 bits of 2 inputs.

4 bit - Binary Adder

A ① 1 1 0 1

B 0 1 0 0

① ← 0 0 0 1

carry

Bits of A = $A_3 A_2 A_1 A_0$ ← C_0

Bits of B = $B_3 B_2 B_1 B_0$

C_4 ← $S_3 S_2 S_1 S_0$

Max. number in 4 bits = 15

A + B = 4+4 bits

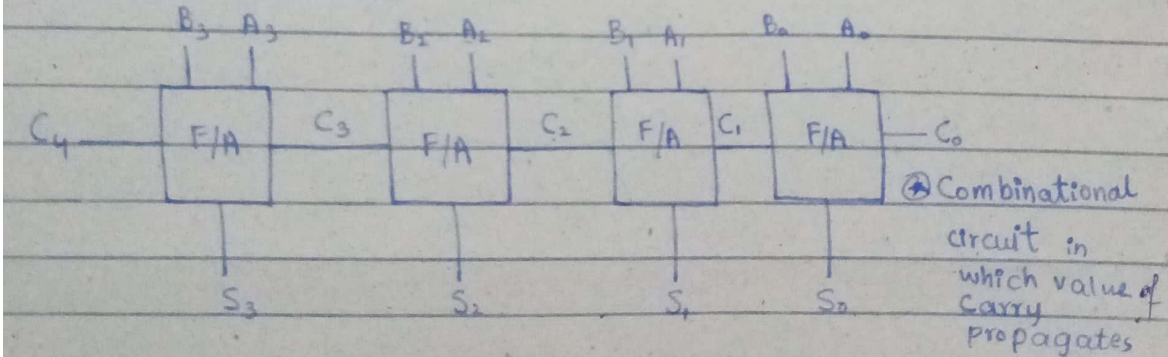
= 15 + 15 = 30

Binary adder is Solved by taking 4 full adders: why 4? bcz input numbers (A and B) are 4 bit numbers.

Bits of output = 5 = $S_3 S_2 S_1 S_0$ and C_4

If inputs were 8 bit number, then 8 full adders will be used.

Circuit Diagram:



* Combinational circuit in which value of carry propagates

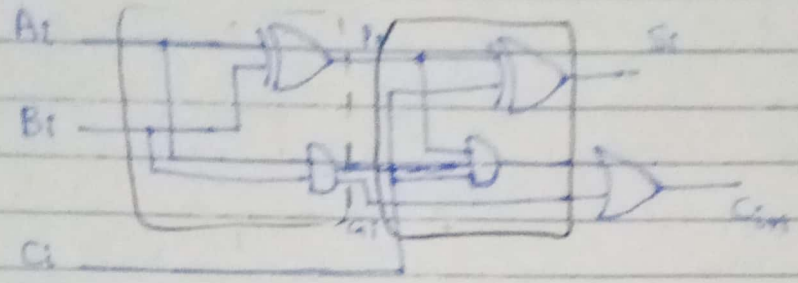
* C_1 is carry out of A_0, B_0 and carry in of A_1, B_1. Same goes for C_2 and C_3.

C_4 is final carry out. C_0 is carry in of A_0, B_0. →

General form to understand Sum and C_{out} / Cout:

A_i + C_i
+ B_i
S_i
(C_{i+1})
C_{out}

* This circuit is a combinational circuit which don't produce value of next Carry and Sum until the carry from previous F/A unit propagate into the next F/A.



$P_i = A_i \oplus B_i$

$G_i = A_i \cdot B_i$

$S_i = P_i \oplus C_i$

$C_{i+1} = P_i C_i + G_i$

P_i can be computed at same time bcz all 4-bits have diff. A_i and B_i. Same goes with G_i

Each full adder will produce output (S_i) at the ~~Binary~~ same time if we can compute value of Carry (C_i) and don't let F/A to compute it. We can calculate C_i by above 4 equations. This will make **parrallel binary adder.**

C_0 = input carry

$C_1 = P_0 C_0 + G_0$

$C_1 = (A_0 \oplus B_0) C_0 + A_0 \cdot B_0$

$C_2 = P_1 C_1 + G_1 = (A_1 \oplus B_1) C_1 + A_1 B_1$

$C_2 = P_1 (P_0 C_0 + G_0) + G_1$

$C_3 = P_2 C_2 + G_2 = (A_2 \oplus B_2) C_2 + G_2$

$C_3 = P_2 (P_1 C_1 + G_1) + G_2$

$C_3 = P_2 (P_1 (P_0 C_0 + G_0) + G_1) + G_2 = P_2 P_1 P_0 C_0 + P_2 P_1 G_0 + P_2 G_1 + G_2$

$C_4 = \underline{P_3 C_3} + G_3 = (A_3 \oplus B_3) C_3 + G_3$

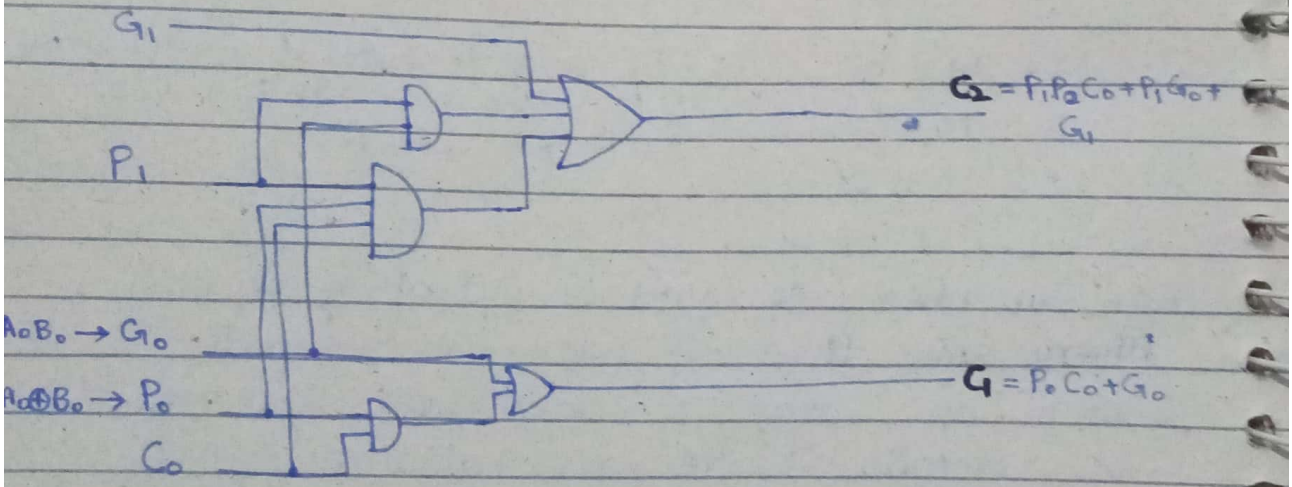
$= P_3(P_2 \underline{C_2} + G_2) + G_3$

$= P_3(P_2(P_1 \underline{C_1} + G_1) + G_2) + G_3$

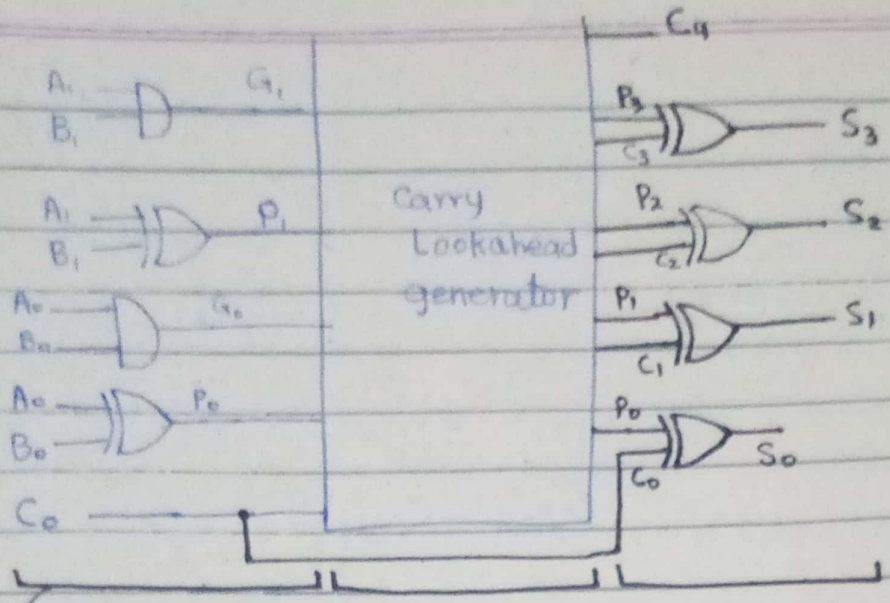
$= P_3(P_2(P_1(P_0 C_0 + G_0) + G_1) + G_2) + G_3$

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

* Carry Lookahead Generator (generates all carries in parallel way at the same time)



* It gives output in milliseconds.



→ No time delay

1st level 2nd level 3rd level due to AND/OR 4th level

$P_i = A_i \oplus B_i$

$G_i = A_i B_i$

$S_i = P_i \oplus C_i$

$C_{i+1} = P_i C_i + G_i$

Draw carry lookahead generater for 2 bits/3bits?