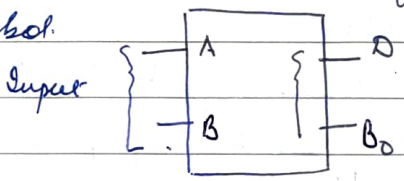
SUBTRACTORS

Half Subtractor

- It has two bits / binary digits as input and produces two bits / binary digits as output
- The output bits are
  (a) a sub bit
  (b) a borrow bit

→ used for subtraction of two bits

Logical Symbol.



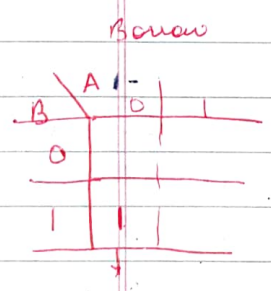
Truth Table.

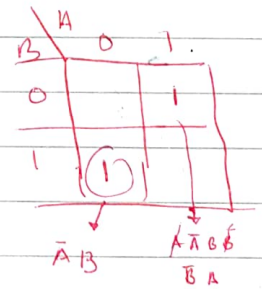
Input		Output		
A	B	Borrow	Sub	
0	0	0	0	0 - 0 = 0
0	1	1	1	0 - 1 = we need to borrow
1	0	0	1	1 - 0 = 1
1	1	0	0	1 - 1 = 0

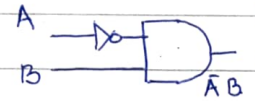
Remember: $A - B \neq B - A$

↑ (B) ↑ (S)

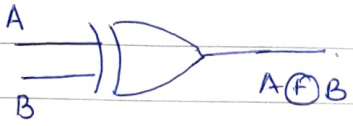
$\bar{A}B$ $A \oplus B$



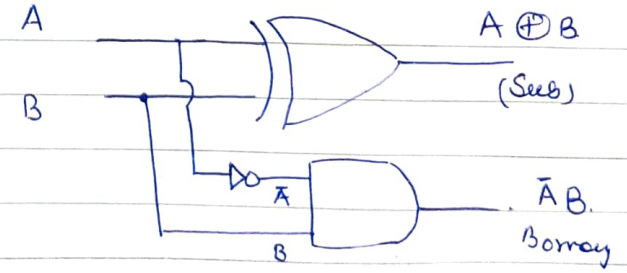




This can even be implemented using exclusive OR gate



Circuit



Subtra

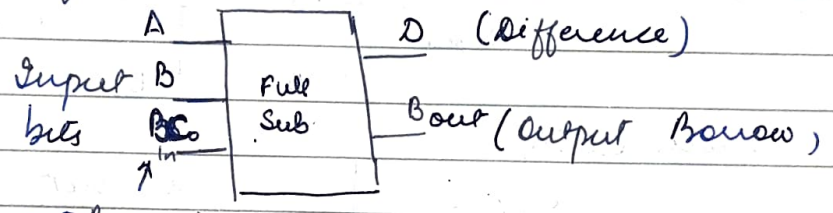
Date
Page

Full Subtractor - It has three inputs (two input bits and one borrow bit)

→ It generates two outputs a difference bit and a output borrow.

The full subtractor differs from a half subtractor as it has a borrow input

Logic Symbol.



Truth Table

	Inputs A	B	B_{in}	Outputs D	B_o	
0	0	0	0	0	0	0-0=0-0=0
1	0	0	1	1	1	0-0=0-1=1 (1 as borrow)
2	0	1	0	1	1	0-1=-1
3	0	1	1	0	1	0-1=-1 so -1=0
4	1	0	0	1	0	1-0=1-0=1
5	1	0	1	0	0	1-0=1-1=0
6	1	1	0	0	0	1-1=0
7	1	1	1	1	1	1-1=0-1 (2-1=1)

Minimize the function using k-map

Use k-map D

✡ learn.

A \ B Bin	00	01	11	10
0	0	1	0	1
1	1	0	1	0

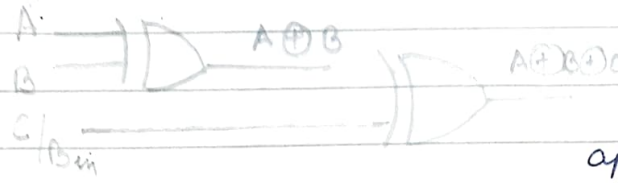
→ no groups possible
→ Check board Configu

Result

$D = A \oplus B \oplus C$

$A \oplus B \oplus C$

appears in full adder even.



Use k-map for B_0

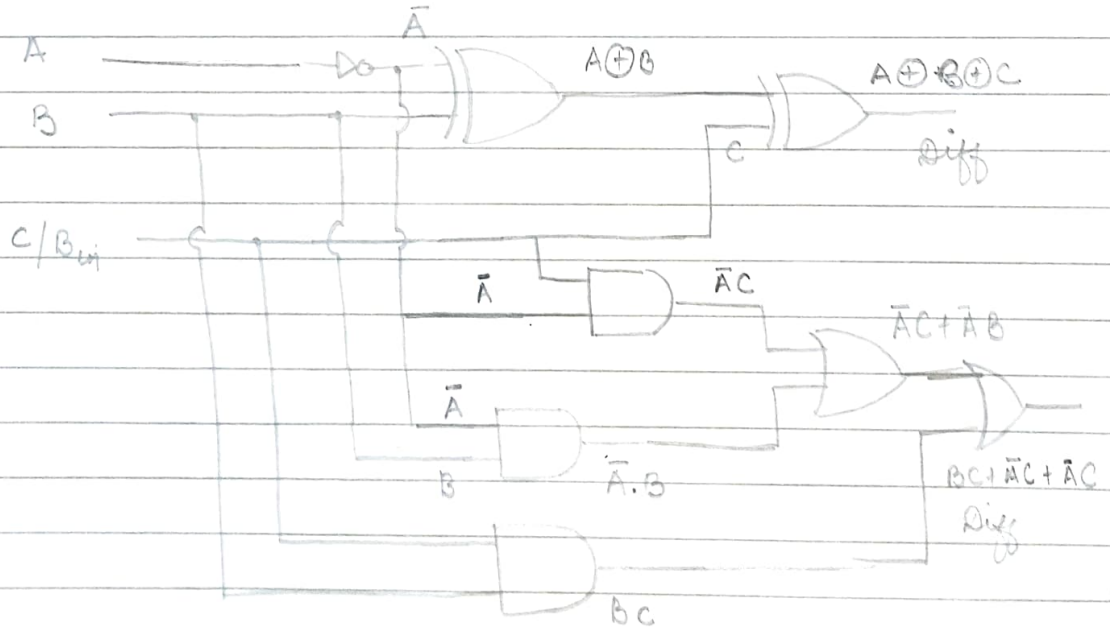
A \ B Bin	00	01	11	10
0	0	1	1	1
1	0	0	1	0

$\bar{A}C$

$\bar{A}B$.

BC

$B_0 = BC + \bar{A}C + \bar{A}B$.

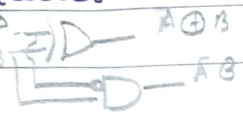


A full subtractor can be implemented using two half subtractor

The boolean expression of half subtractor is

Difference (D_h) = $A \oplus B$ — (1)

Borrow B_h = $\bar{A}B$ — (2)



The boolean expression of full subtractor is

Difference (D) = $A \oplus B \oplus C$

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

Consider the borrow expression

$= \bar{A}B + \bar{A}C + BC \quad (B + \bar{B} = 1)$

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

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

$= \bar{A}B(1 + C) + \bar{A}\bar{B}C + BC \quad (\because 1 + C = 1)$

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

$= \bar{A}B + \bar{A}\bar{B}C + BC(A + \bar{A}) \quad \because (A + \bar{A} = 1)$

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

$= \bar{A}B(1 + C) + C(\bar{A}\bar{B} + AB) \quad (1 + C = 1)$

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

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

Substitute (1) and 2 in the above

$= B_h + C$

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

Error

Input	Output
0 0	1
0 1	0
1 0	0
1 1	1

Error is inverse of Ex-OR

A

$A \oplus A = 0$ — $A \odot A = 1$

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

$A \oplus 0 = A$ Buffer — $A \odot 0 = \bar{A}$ Inverter

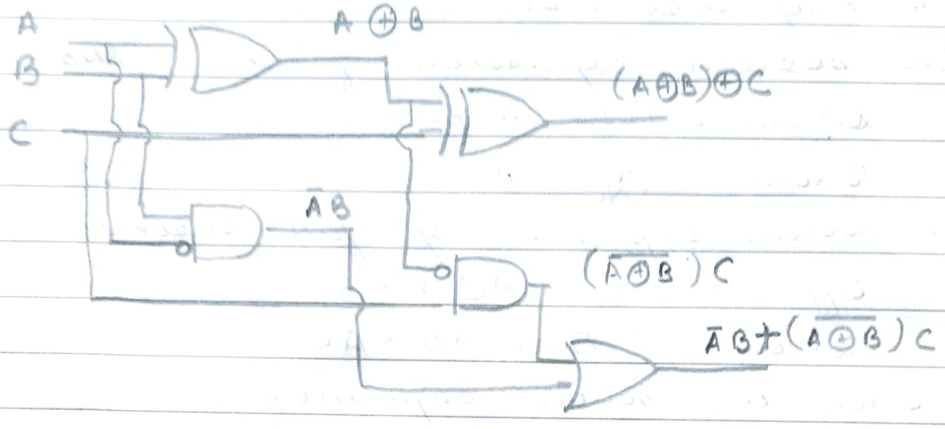
$A \oplus 1 = \bar{A}$ Inverter — $A \odot 1 = A$ Buffer

$A \oplus A \oplus A \ldots$ n times — $A \odot A \odot A \odot A \ldots$ n times

If n is Even = 0 — If n is Even = 1

If n is Odd = 1 — If n is Odd = 0

Full adder using two half adder :



literature uses Cin or CI
Cout or CO
Σ or Sum

ADDERS

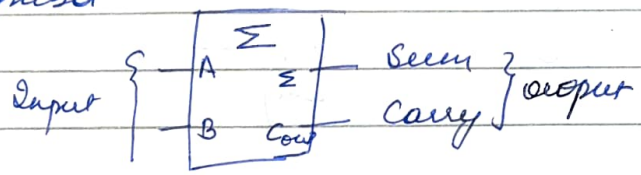
Binary addition

0 + 0 = 0	0 · 0 = 0
0 + 1 = 1	0 · 1 = 0
1 + 0 = 1	1 · 0 = 0
1 + 1 = 10	1 · 1 = 1

(2 binary digits)

Half adder → It has two bit/input and produces two bit/output (2 binary digits)
→ The output bits are (a) a sum bit
(b) a carry bit

- Logical symbol

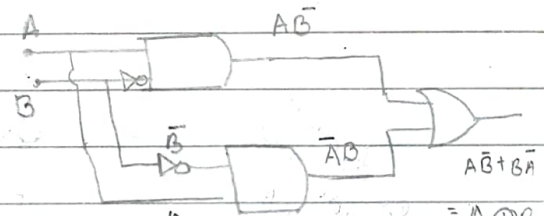


- Truth Table

Input A	Input B	Output Cout (MSB)	Output Σ (LSB)
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

using minterms
Σ = 1 for
= $\bar{A}B + A\bar{B}$
and Cout = A.B.

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



or

Use exclusive OR gate

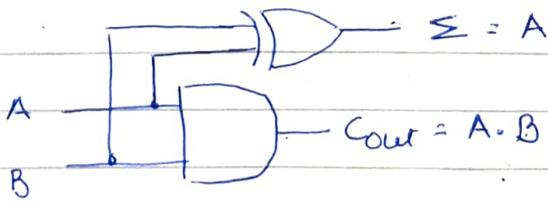
→ Cout - it is the AND of the input variables

$C_{out} = A.B$ —(1)

→ Sum output (Σ) it is 1 only if input variables (A and B) are not equal.

$\Sigma = A \oplus B$ —(2)

- Logic Diagram



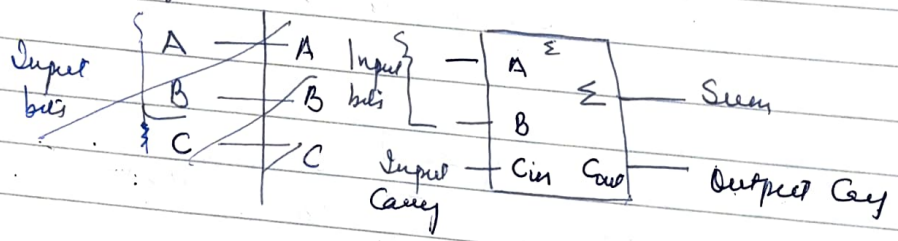
half adder

Adders

→ **Full Adder.** → It has two input bits and an input carry bit
→ It generates a sum output and an output carry.

The full adder is different from half adder as it accepts an input carry

- Logic symbol



- Truth table

Input			Output	
A	B	Cin	Cout	Σ
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Tip: Do decimal addition and write BCD

10
1
11

Cout
(0 + 0) 0
(0 + 0) 0
(0 + 1) 1
(0 + 1) 1
(0 + 1) 1
(0 + 1) 1
(1 + 10)
(1 + 10)

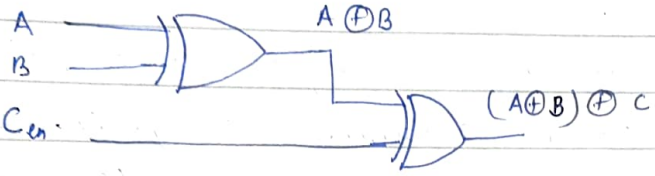
Logic $\Sigma = (A \oplus B) \oplus C_{in}$ — (3)

$C_{out} \quad AB + (A \oplus B) C_{in}$ — (4)

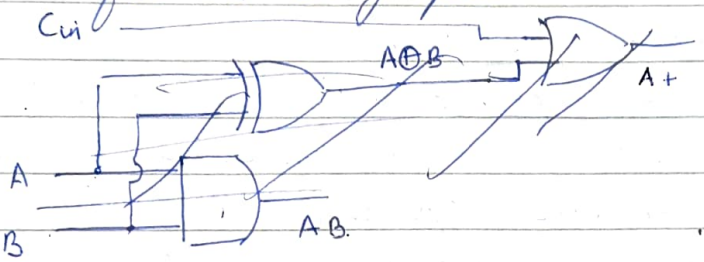
✡ Imp.

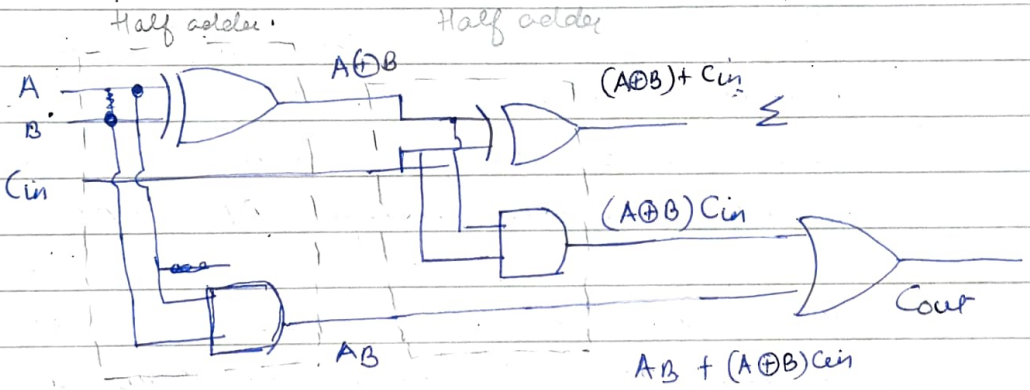
Logic circuit

If we need to add 3 bits the logic ckt would be

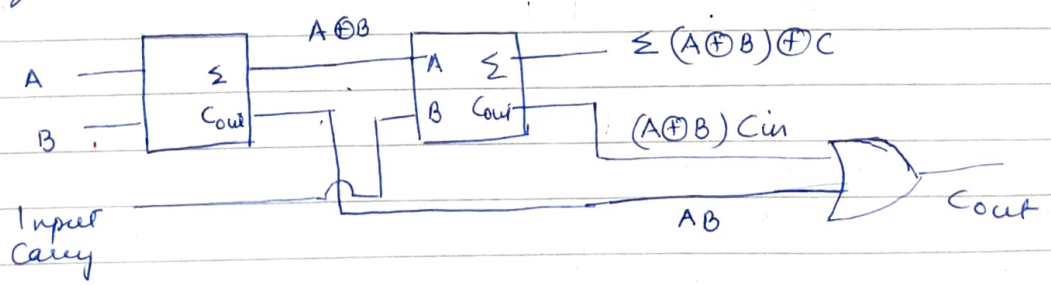


now for the carry operation

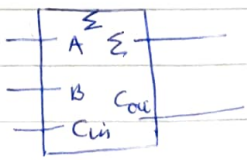




It is observed that two half adders are connected to give the sum and ORed to give the output



Full adder logic Symbol



Three input

two output

Parallel Binary Adders

- ~~When~~ Two or more full adders are connected to form parallel binary adder.

- ALU (Arithematic logic unit) is used for performing addition. When addition is performed by processor on two numbers simultaneously/ at the same time, the two numbers are called operands.
The source operand is the number that is added to an existing number called the destination operand. The sum of the source operand and the destination operand is stored in an ALU register such as the accumulator. In a processor the addition of integer number or floating-point numbers using ADD or FADD instruction

from ALU register done in processor

When we need to add two binary numbers a full adder is required for each bit in a number.

2 bit number needs 2 full adders
4 bit " " 4 full adder and so on.

Eg

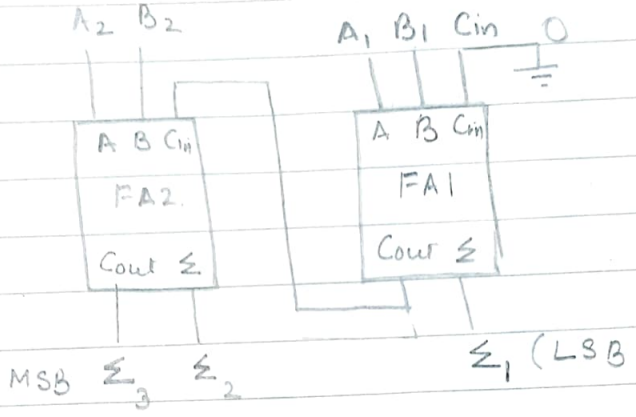
```
    1 <- Carry bit from right column
    1 1
    0 1
  -----
  1 0 0
```

Carry bit from second column becomes a sum bit

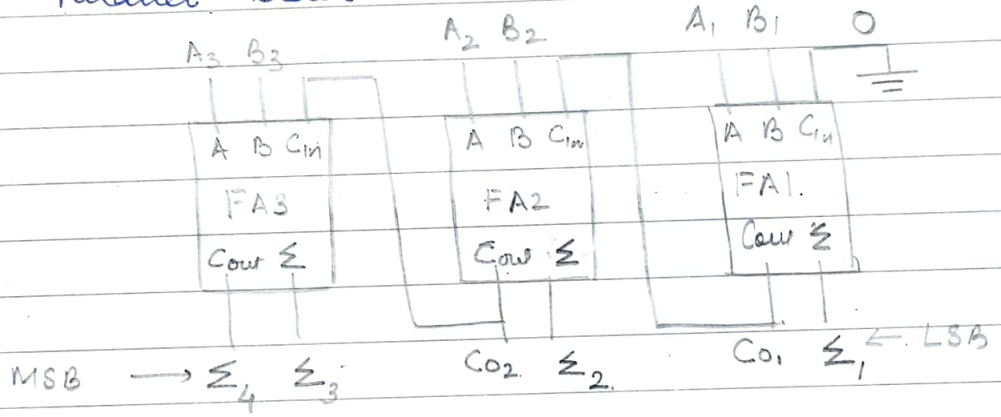
The general expression

```
        Column 2   Column 1
           ↓          ↓
          A2         A1
          B2         B1
        ------------------
  MSB -> S3  S2  S1 <- LSB
```

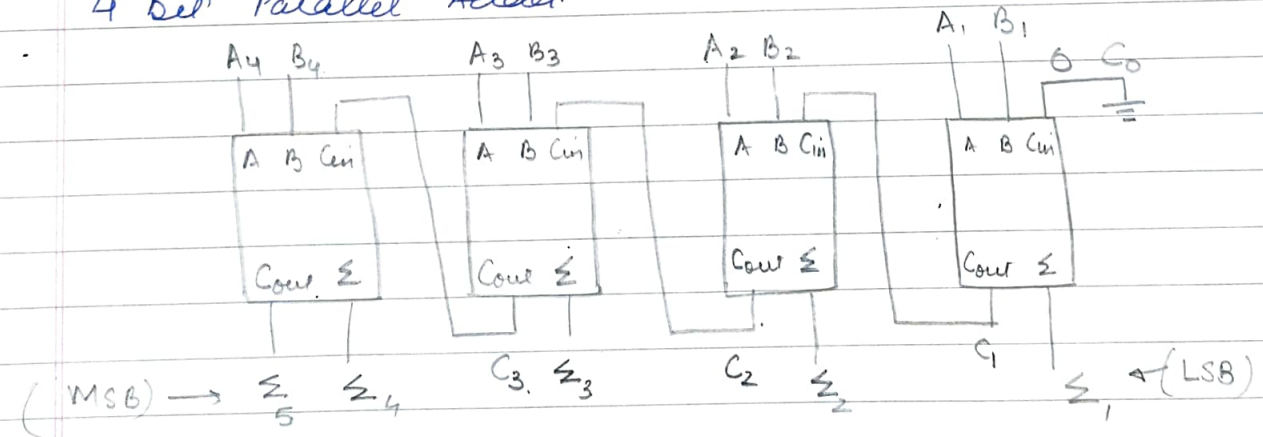
2. bit Parallel adder (start from RHS)



3 bit Parallel Adder



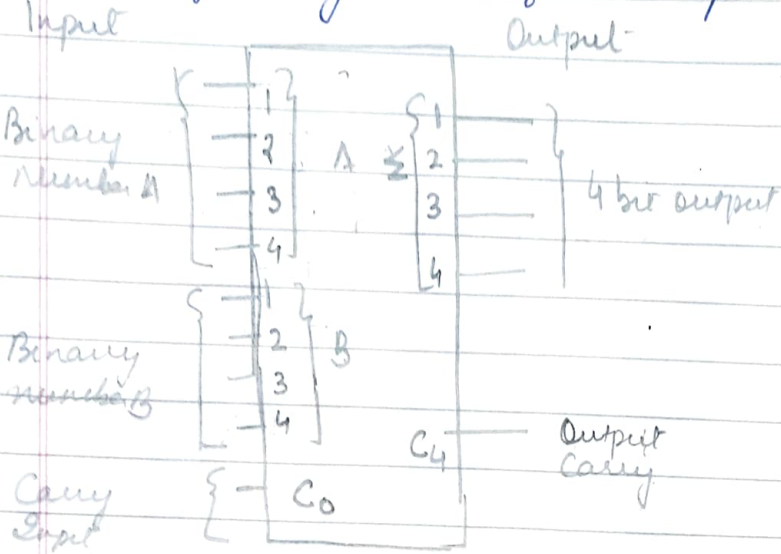
4 bit Parallel Adder.



C_1, C_2, C_3 → internal carriers.

The carrier adder are of two type in parallel adders. (i) ripple carry adder
(ii) Carry look adder.

• Logic symbol of 4-bit parallel adder



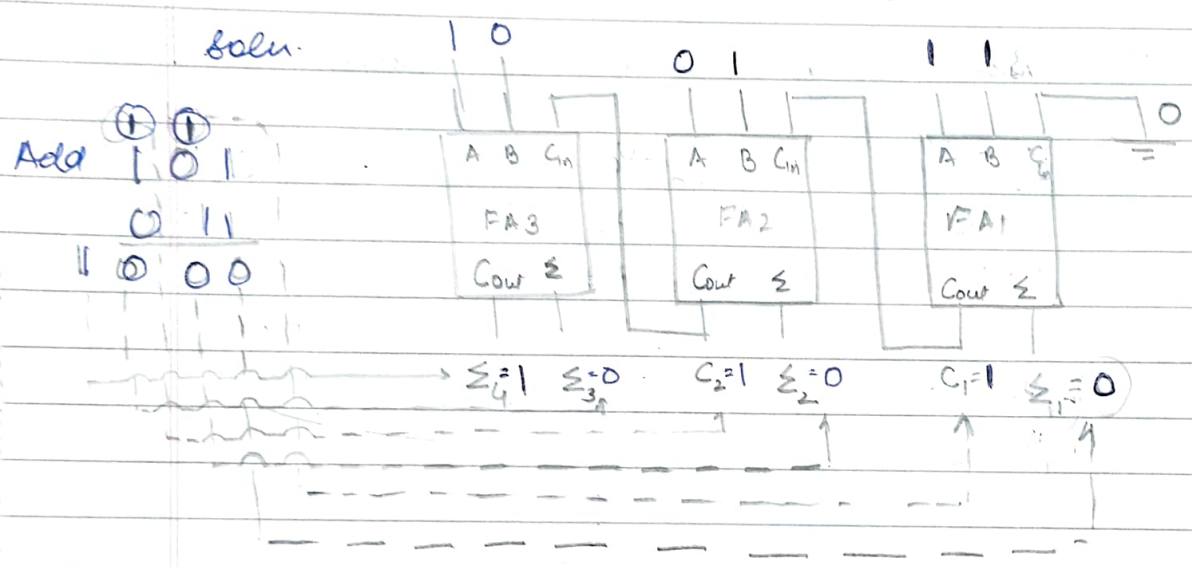
• Truth table for each stage 4-bit parallel adder.

C_{n-1}	A	B	C_n	Σ
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

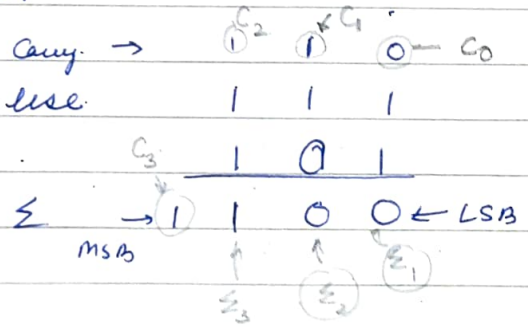
Example 1 Determine the sum generated by the 3-bit parallel adder ~~shown~~ using a figure. Indicate the value when the (a) binary numbers ~~xxxx~~ 101 and 011 are added.

you need to use the groundnet.

Soln.

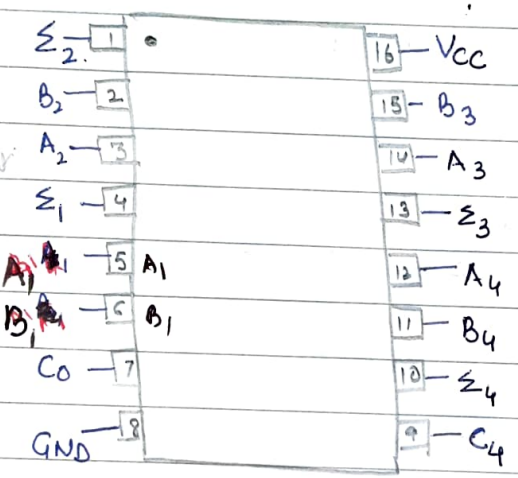


When the ~~xxx~~ of the binary number are 111 and 101

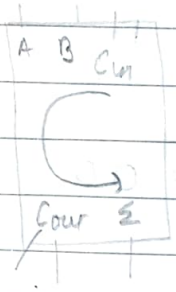


→ Implementation of 4 Bit Parallel Adder

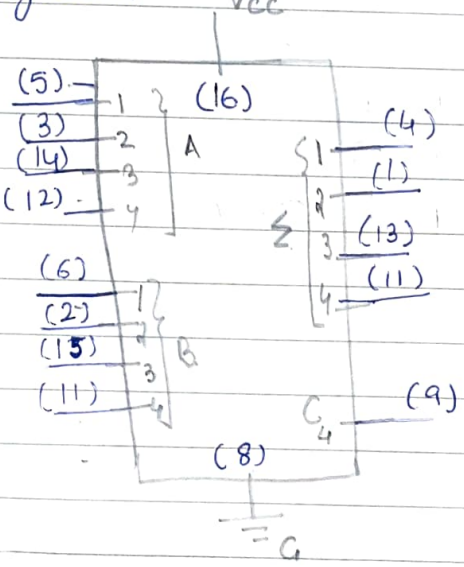
- A fixed function device 74HC 283 and 74LS284 are 4-bit parallel adders which have identical pin configuration
- The pin configuration is given below



important from implement point of view



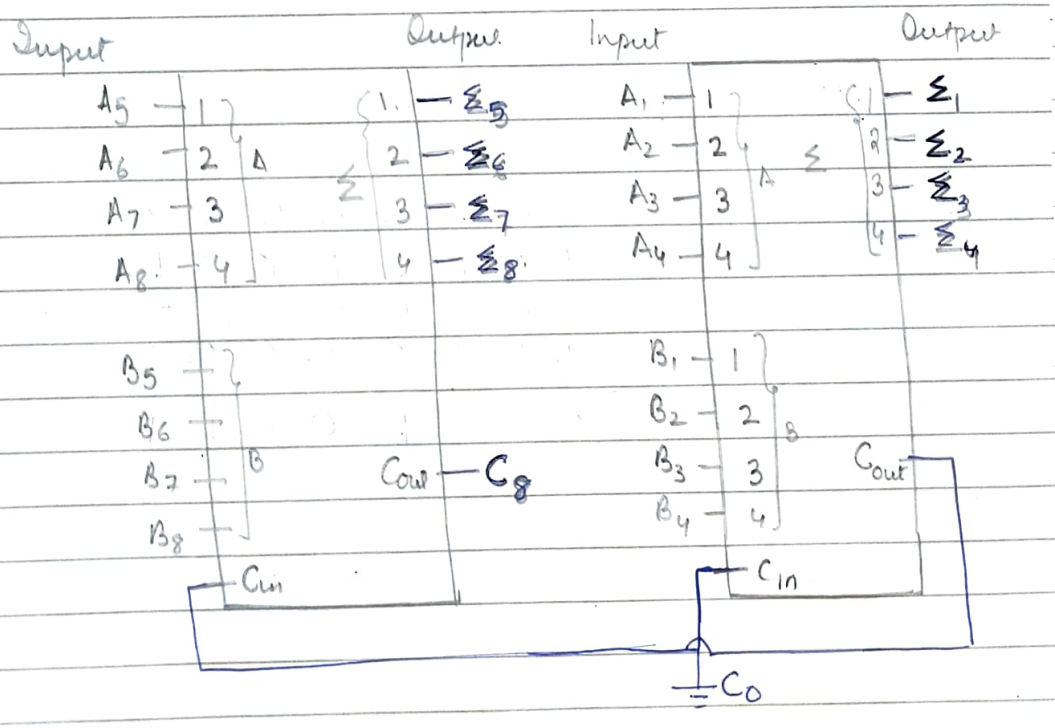
- Logic Symbol



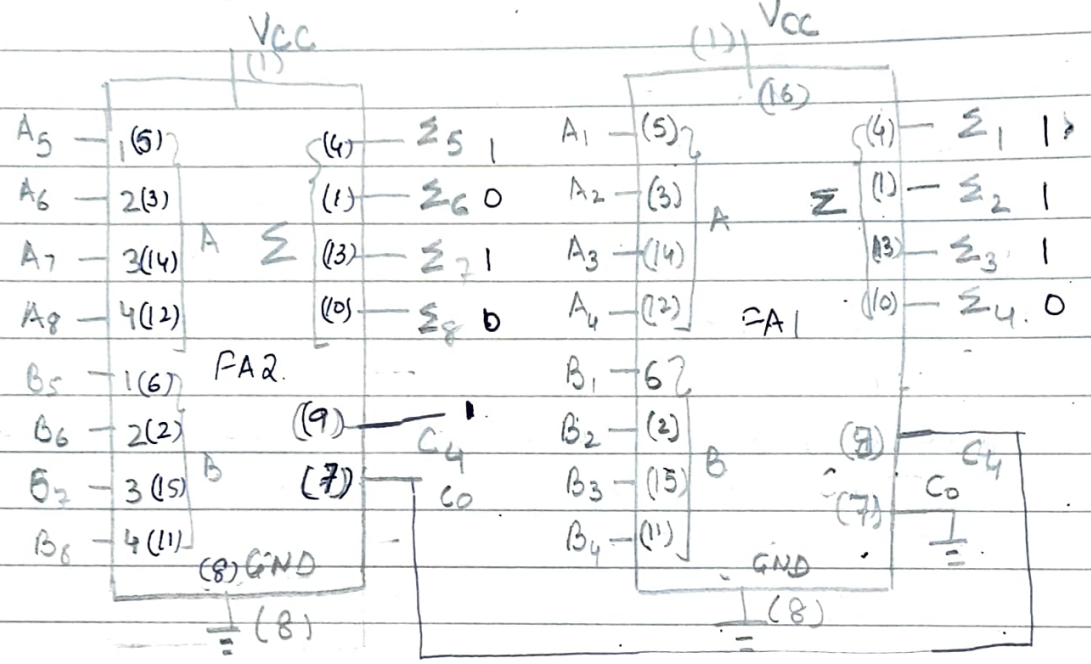
Adder Expansion

- To get a 8-bit parallel adder two 4-bit parallel are connected in cascade.
- Adders are expanded by cascading
- The input carry (C_0) is connected to ground since there is no carry for LSB.
- The carry of the lower order adder is connected to the carry input of the higher order adder. This is the cascading of adders.

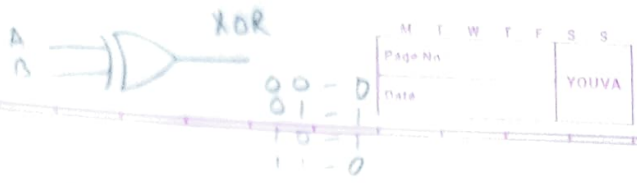
The figure shows the cascading of 4-bit adders to give an 8-bit adder.



Example Implement 8-bit parallel adder using 74HC283. Also indicate the output of the following 8-bit input numbers ~~A8 A7 A6~~ A1 10111001 and 10011110



A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1
1	0	1	1	1	0	0	1
1	0	0	1	1	1	1	0
B_8	B_7	B_6	B_5	B_4	B_3	B_2	B_1
1	0	1	0	1	0	1	1



A full adder can be implemented using two half adders

The boolean exp of half adder is

Sum = $A \oplus B$

Carry = AB

The boolean exp of full adder is

Sum = $A \oplus B \oplus C$

Carry = $AB + AC + BC$

Carry = $AB + AC + BC(A + \bar{A})$ $(\because A + \bar{A} = 1)$

$= AB + AC + BCA + BC\bar{A}$

$= AB(1 + C) + AC + \bar{A}BC$ $\because 1 + C = 1$

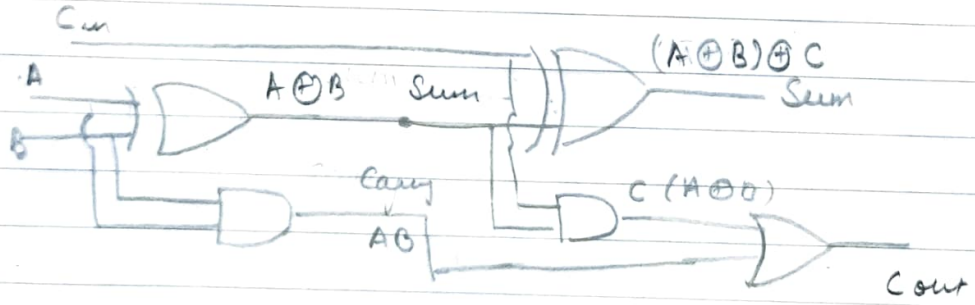
$= AB + AC + \bar{A}BC$

$= AB + AC(B + \bar{B}) + \bar{A}BC$ $\because B + \bar{B} = 1$

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

$= AB(1 + C) + C(A\bar{B} + \bar{A}B)$ $\because 1 + C = 1$

$= AB + C(A \oplus B)$



BCD Adder or 8421 Adder

for 0 - 15 in 4 bit value.

Decimal	8	4	2	1
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
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1

invalid

$BCD_1 + BCD_2$ → output > 9 is invalid
use 0-9
for >9 to 15

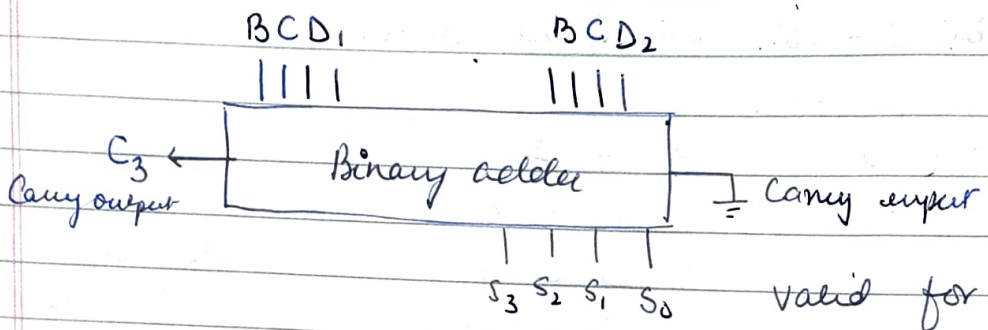
To convert invalid BCD to valid value we add binary 6 to it

Eg 10 10 — 10

$+\ 0\ 1\ 1\ 0$ add 6 to it

$0\ 0\ 0\ 1\ 0\ 0\ 0\ 0$

1 0



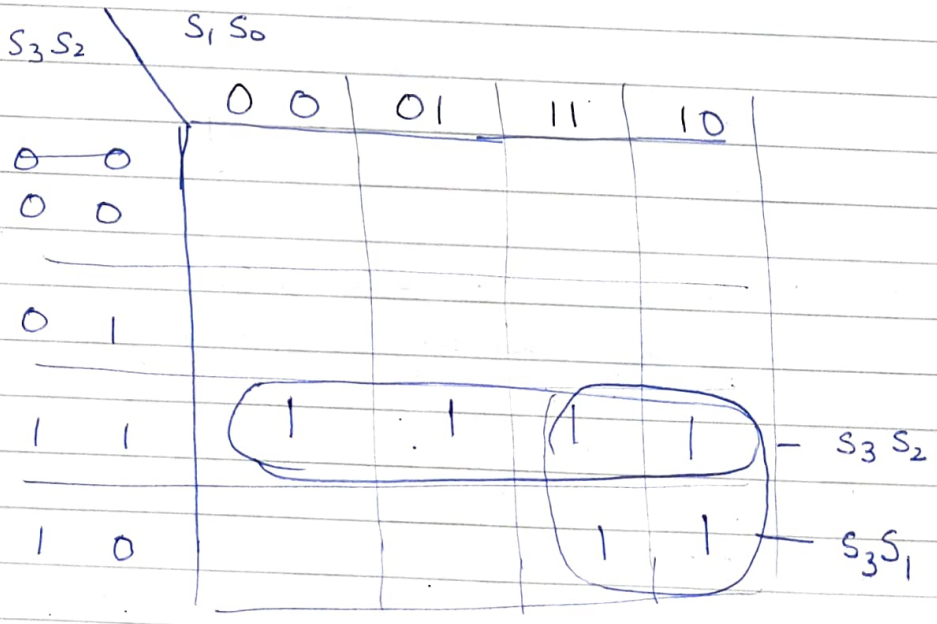
valid for 0 to 9

for all invalid patterns

	S_3	S_2	S_1	S_0	f
10	1	0	1	0	1
11	1	0	1	1	1
12	1	1	0	0	1
13	1	1	0	1	1
14	1	1	1	0	1
15	1	1	1	1	1

invalid bit

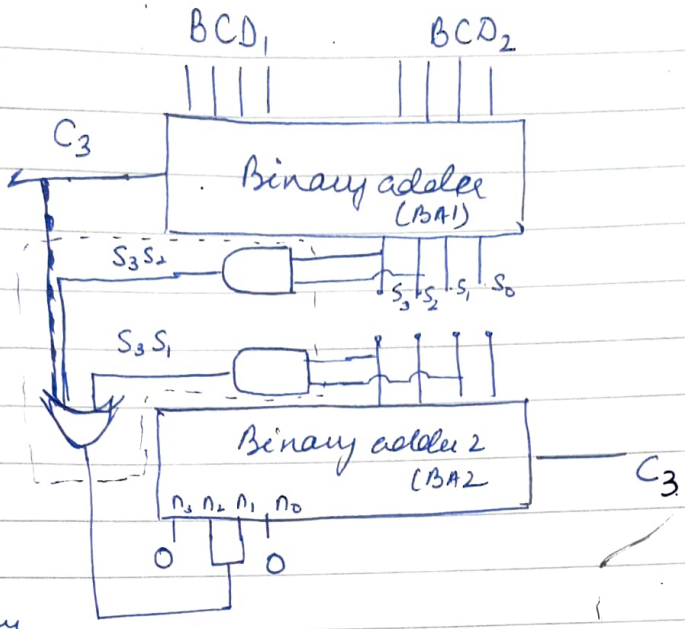
we generate 1 using k-map



$f = S_3S_2 + S_3S_1$

Ckt

Logic



$f = S_3S_2 + S_3S_1$

$f_1 = f + C_3$

If at any point of line invalid result is obtained after adding two BCD. then (6) 0110 is added by the circuit. If valid result is obtained from BA1 then the circuit generates 0 output and 0000 is added to the result obtained from BA1.

The carry function needs to be taken care of so the function f carry (C_3) needs to be added

$$f_1 = f + C_3$$

The carry of BA1 is added to the ckt .... The Carry (C_3) generated by BA2 is stored in a seperate register

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

✱ Invalid number of inputs.

BCD 16 patterns — BCD 16 patterns = 16×16 possible input

10 valid patterns — 10 valid patterns 10×10 valid inputs

$= 256 - 100 = 156$ invalid inputs possible.