UNIT - II

Gate Level Minimization

Gate-level minimization is the design task of finding an optimal Gate-level implementation of the Boolean functions describing a digital circuit.

The Map Method : The Complexity of digital Logic gates that implement a boolean function is directly related to the Complexity of the algebric expression from which the function is implemented. The true table ~~expression~~ representation of a function is unique - when it is expressed algebrically it can appear in many different, but equivalent forms. The mapmethod Provide a simple, straight forward procedure for minimizing boolean functions. This method may be regarded as a pictorial form of a true table. The mapmethod is also known as the Karnaugh map (or) K-map. It was introduced by a telecome engineer Mouric Karnaugh at Bell Labs in 1953.

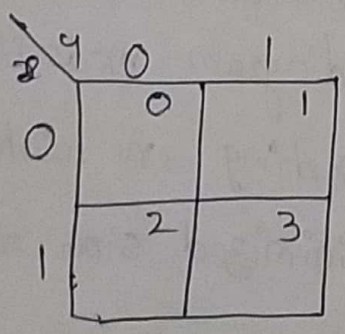
A K-map is a diagram made up of squares, which each square representing one minterm of the function that is to be minimized. since any boolean function can be expressed as a sum of minterms.

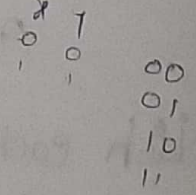
It follows that a Boolean function is recognized graphically in the map from the area enclosed by those squares whose minterms are included in the function. The map presents a visual diagrams of all possible ways a function may be expressed in standard form. By recognizing various patterns, the user can derive alternative algebric expressions for the same function from which the simplest can be selected.

The simplified expressions produced by the map are always in one of the two standard forms: sum of products or product of sums. It will be assumed that the simplest algebric expression is an algebric expression with a minimum number of terms and with the smallest possible number of literals in each term. This expression produces a circuit diagram with a minimum number of gates and the minimum number of inputs to each gate.

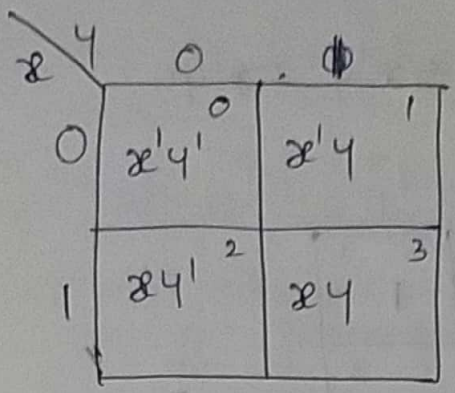
Two-variable K-map:

In the two-variable map there are four minterms for two variable hence the map consists of four squares one for each minterm.





The possible minterms with 2 variables (x and y) are x'y', x'y, xy' and xy.

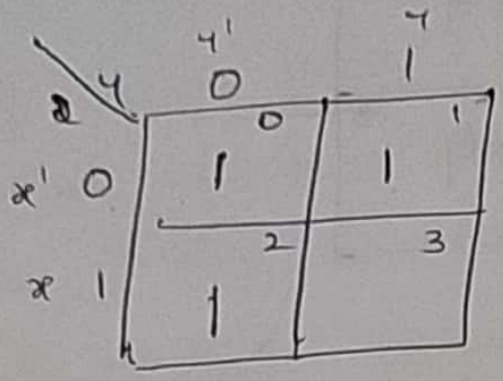


When we are simplifying a boolean function equation using karnaugh map, we represent the each cell of K-map containing the conjunction term with 1. After that we group the adjucent cells with possible sizes as 2 or 4. In case of large K-maps we can group the variables in large size like 8 or 16.

The group of variables should be in rectangular shape, that means the groups must be formed by combining adjusent cells either vertically or horizontally.

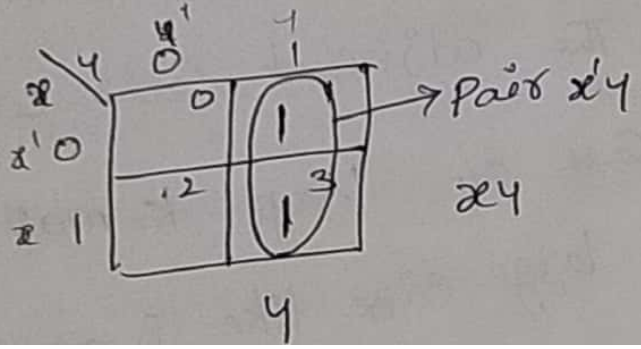
eg: simplify the given two variable Boolean equation by using K-map

$$F = xy + xy' + x'y'$$



$$F = x + y'$$

eg: $F(x,y) = \Sigma(1,3)$



Three Variable K-map:

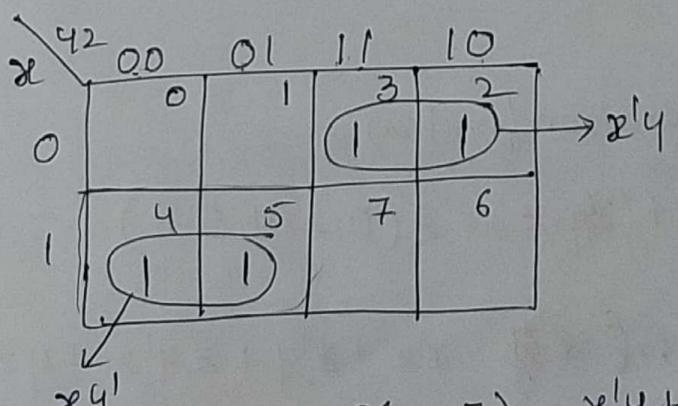
For a 3-variable Boolean function, there is a possibility of 8 output minterms. The

x \ yz	00	01	11	10
0	0 $x'y'z'$	1 $x'y'z$	3 $x'yz$	2 $x'yz'$
1	4 $xy'z'$	5 $xy'z$	7 xyz	6 xyz'

The map consist of eight squares. Note that the minterms are arranged not in the binary sequence but in a sequence similar to the Gray code. the characteristics of this sequence is that only one bit changes in value from one adjacent column to next.

→ simplify the boolean function

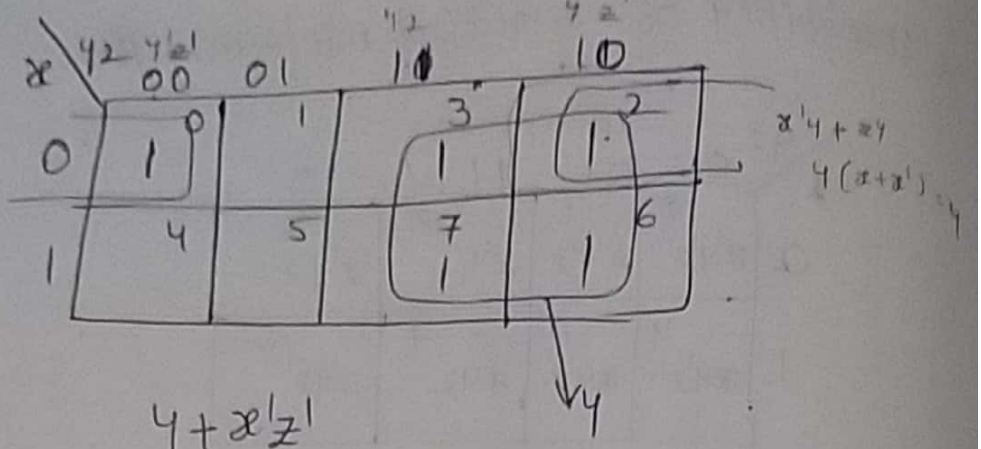
$$F(x,y,z) = \Sigma(2,3,4,5) = m_2 + m_3 + m_4 + m_5$$



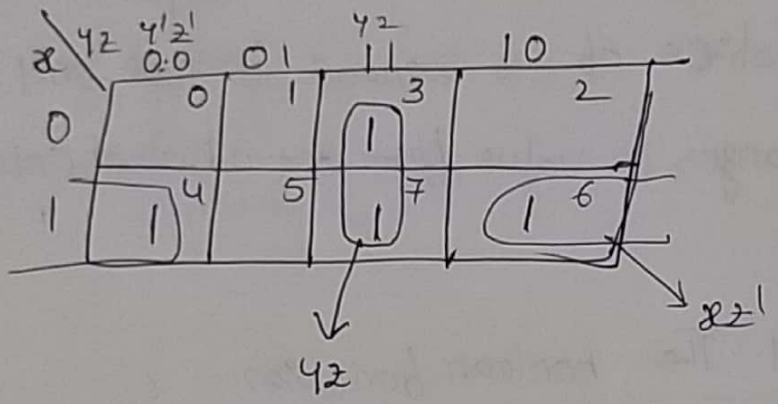
$$F(x,y,z) = x'y + xy'$$

eg: simplify the Boolean function

$$F(x,y,z) = \Sigma(0,2,3,6,7)$$



eg: $F(x,y,z) = \Sigma(3,4,6,7)$



$$F = x'yz + xz'$$

simplify the Boolean function

$$f(x,y,z) = \Sigma(0,2,4,5,6)$$

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

Four Variable K-map

There are 16 Possible minterms in case of a 4-variable Boolean function

wx \ yz	00	01	11	10
00	0 $w'x'y'z'$	1 $w'x'y'z$	3 $w'x'yz$	2 $w'x'yz'$
01	4 $w'xy'z'$	5 $w'xy'z$	7 $w'xyz$	6 $w'xyz'$
11	12 $wxy'z'$	13 $wxy'z$	15 $wxyz$	14 $wxyz'$
10	8 $wx'y'z'$	9 $wx'y'z$	11 $wx'yz$	10 $wx'yz'$

→ one square represent one minterm, giving a term with four literals

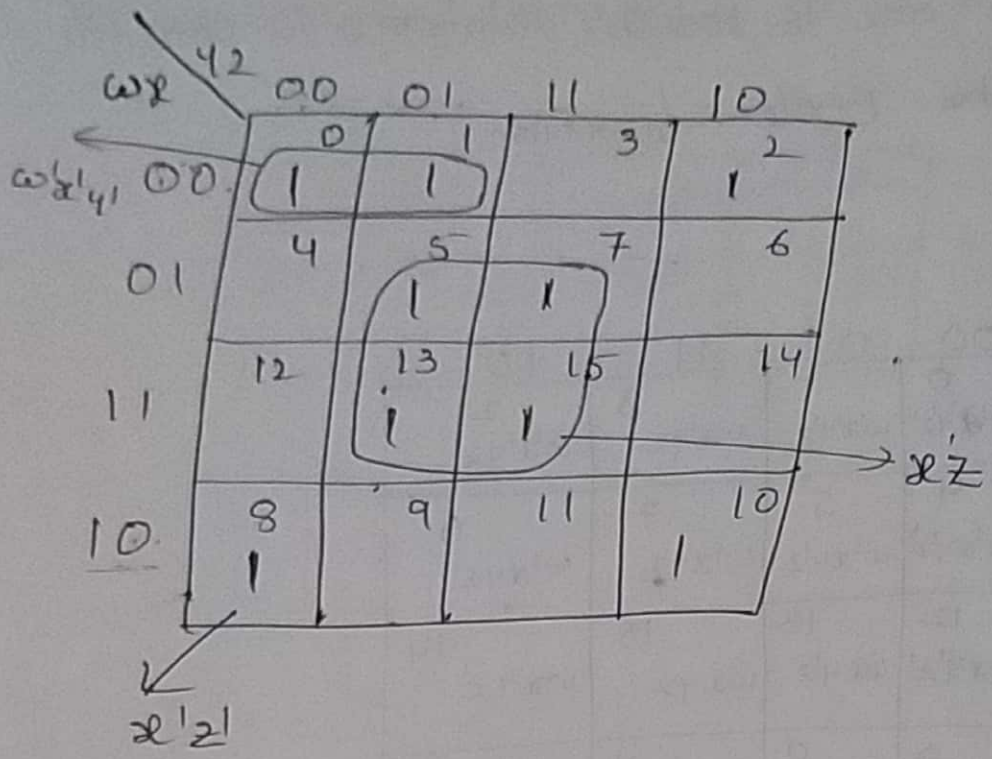
→ Two adjacent squares represent a term with Three literals

→ four adjacent squares represent a term with Two literals

→ Eight adjacent squares represent a term with one literal

→ sixteen adjacent squares produce a function that is always equal to 1.

$f(w,x,y,z) = \Sigma(0,1,2,5,7,8,10,13,15)$



$xz + x'z' + w'x'y'$

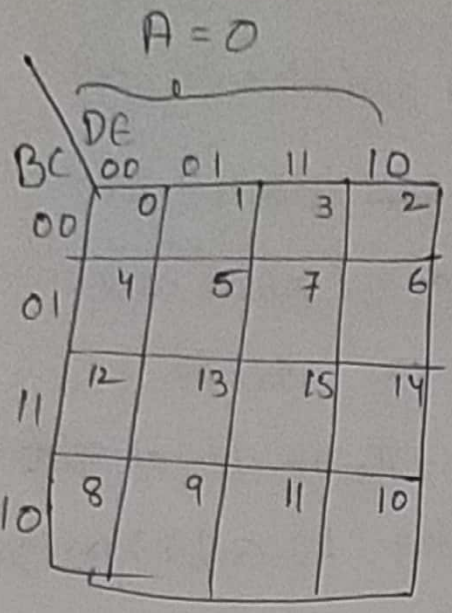
$F(w,x,y,z) = \Sigma(0,1,2,4,5,6,8,9,12,13,14)$

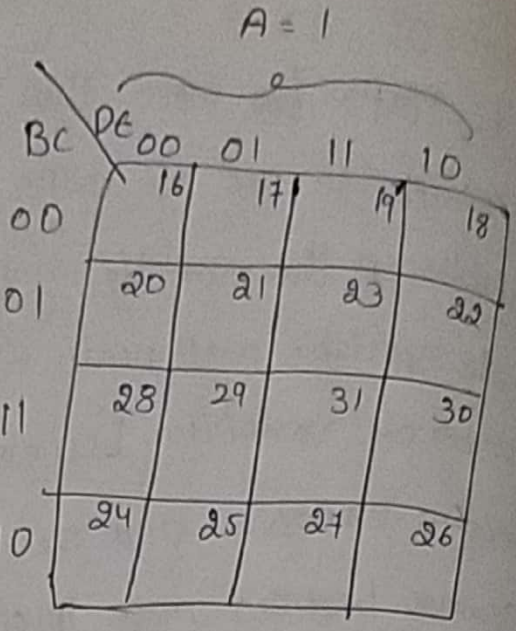
$F = A'B'C' + B'CD' + \cancel{A'BCD'} + AB'C'$

5 Variable K-Map:

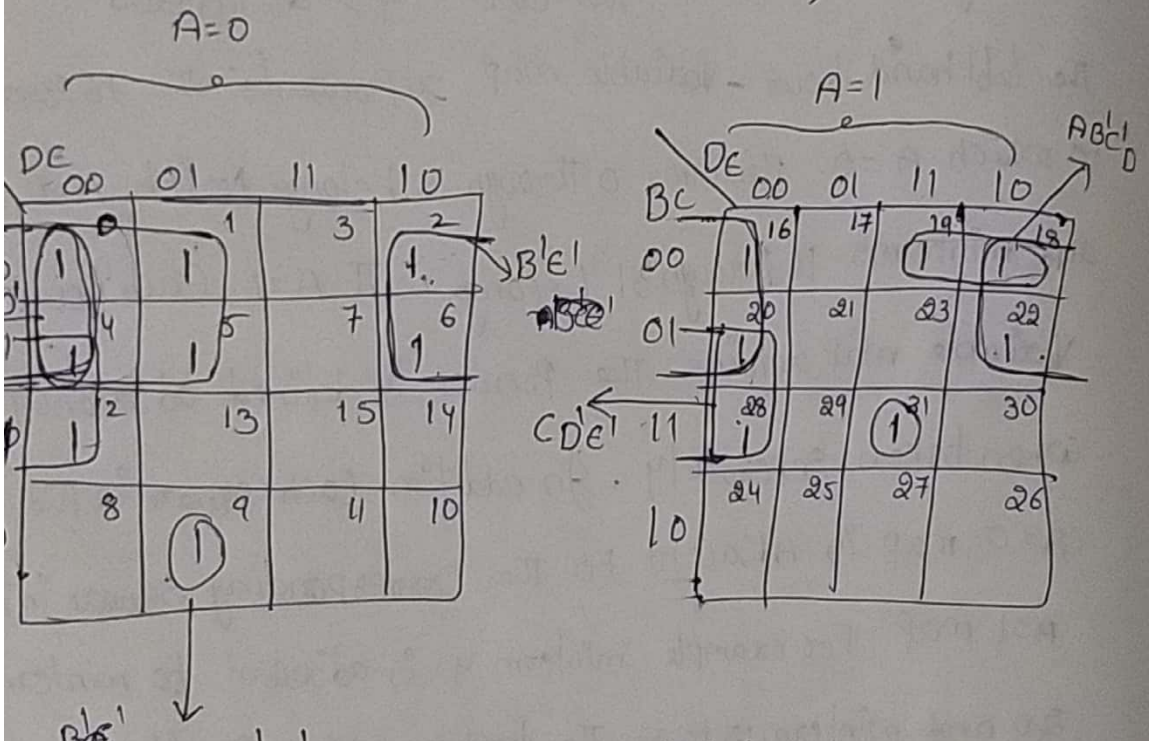
Maps for more than four variables are not as simple to use as maps for four or fewer variables. A five variable map needs 32 square and a six-variable map needs 64 square. When the number of variables become large, the number of squares become excessive and the geometry for combining adjacent squares becomes more involved.

The five variable map consists of 2 four-variable maps with variables A, B, C, D, and E. Variable A distinguishes between the two maps as indicated. The left hand four-variable map represents the 16 squares in which A=0. Minterms 0 through 15 belongs with A=0 and minterms 16 through 31 belongs with A=1. Each four variable map retains the previously defined adjacency when taken seperately. In addition each square in the A=0 map is adjacent to the corresponding square in the A=1 map. For example minterm 4 is adjacent to minterm 20 and minterm 15 to 31. The best way to visualize this new rule for adjacent square is to consider the two half maps as being one on top of the other. Any two squares that fall one over the other are consider adjacent.





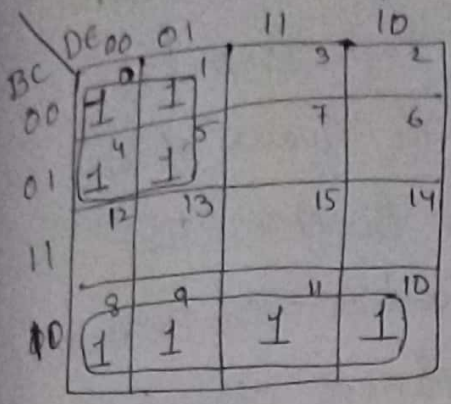
eg $F(A,B,C,D,E) = \Sigma(0,1,2,4,5,6,9,12,16,18,19,20,22,28,31)$



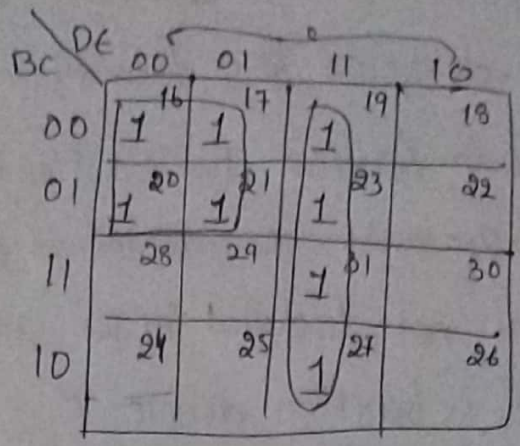
$A'BC'D'E + ABCDE + A'B'D' + B'E' + AB'C'D + CD'E'$

→ $f = \Sigma(0,1,4,5,8,9,10,11,16,17,19,20,21,23,27,31)$

A=0



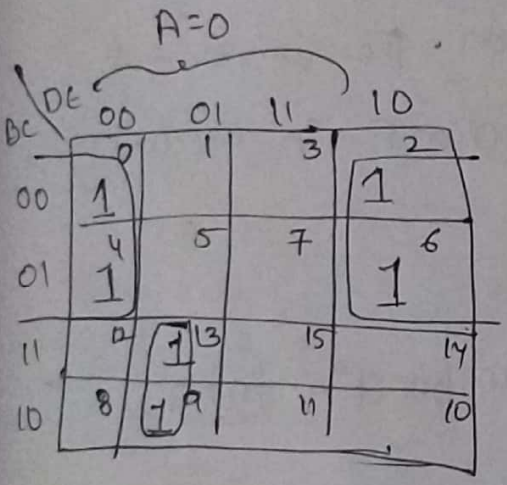
A=1



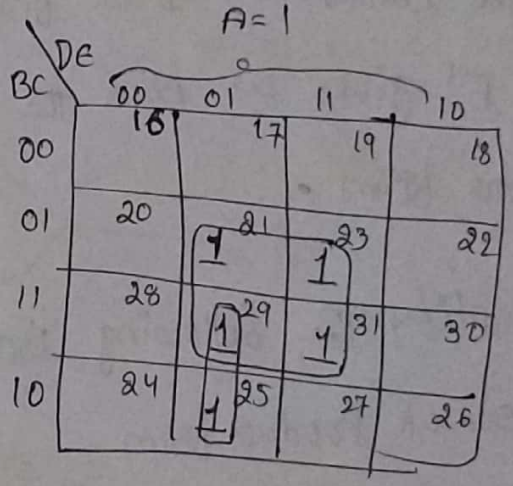
$f = B'D' + A'BC' + ADE$

→ $F(A,B,C,D,E) = \Sigma(0,2,4,6,9,13,21,23,25,29,31)$

A=0



A=1



$A'B'E' + A'BD'E + ABE + ABD'E$

$A'B'E' + BD'E(A+A') + ACE$

$A'B'E' + BD'E + ACE$

Product of Sums simplification:

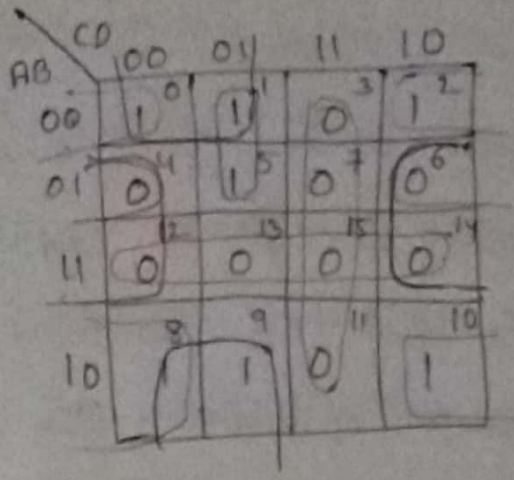
The Procedure for obtaining a minimized function in Product-of-Sums form follows from the basic properties of Boolean function. The 1's placed in the squares of the map represents the minterms of the function. The minterms not included in the standard sum of Products form of a function denote the complement of a function is represented in the map by the squares not marked by 1's. If we marked the empty squares by 0's and combine them into valid adjacent squares. we obtain a simplified Sum-of-Products expression of the complement of the function. The complement of F' gives us back the function F in Product of sums form.

→ simplify the following Boolean function into

a) sum of Product form

b) Product of sum form

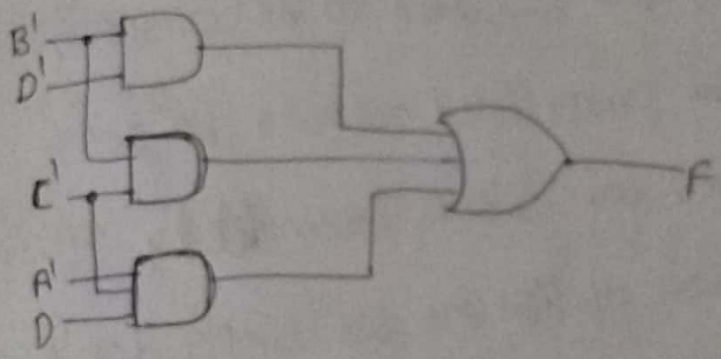
$$F(A,B,C,D) = \Sigma(0,1,2,5,8,9,10)$$



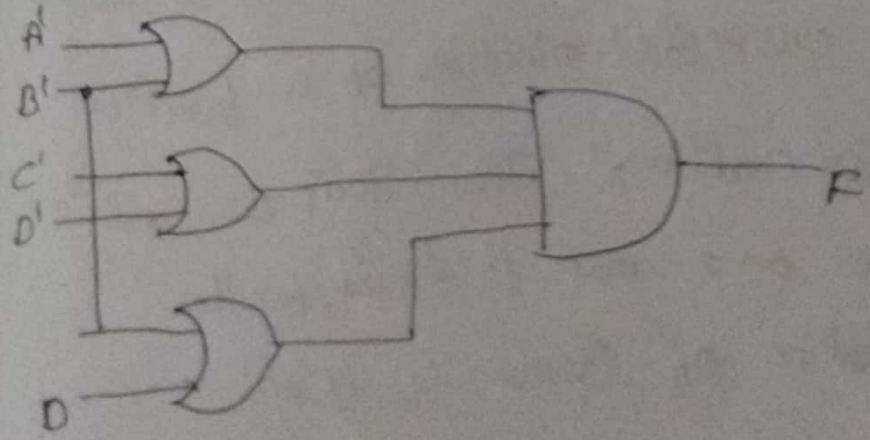
$F_1 = B'C' + B'D' + A'C'D$ (Sum of Products)

$F_1' = AB + CD + A'BD' + BD'$

$F_2 = (A'+B')(C'+D')(B'+D)$



(a) $F_1 = B'D' + B'C' + A'C'D$



Don't Care Conditions:

The logical sum of the minterms associated with a Boolean function specifies the conditions under which the function is equal to 1. The function is equal to zero for the rest of minterms. The pair of conditions assumes that all the combinations of the values for the variables of the function are valid. In some applications the function is not specified for certain combinations of the variables.

The four bit binary code for the decimal digits has six combinations ~~of the~~ variables that are not used. Consequently are considered to be unspecified. Functions that have unspecified outputs for some input combinations are called incompletely specified functions. In most applications we simply don't care what value is assumed by the function for the unspecified minterms. For this reason, it is customary to call the unspecified minterms of a function "don't care conditions". These don't care conditions can be used on a map to provide further simplification of Boolean function.

A don't Care Conditions minterm cannot be marked with a 1 in the map

A don't Care minterm is a Combination of variables who's logical value is not specified. such a minterm cannot be marked with a 1 in the map because it would required that the function always be a 1 for such a Combination. Putting a 0 on the square requires the function to be 0'.

To distinguish the don't Care Condition from 1's and 0's an 'x' is used. an x inside a square in the map indicates that we don't Care whether the value of 0 or 1 is assigned to F for the particular minterm.

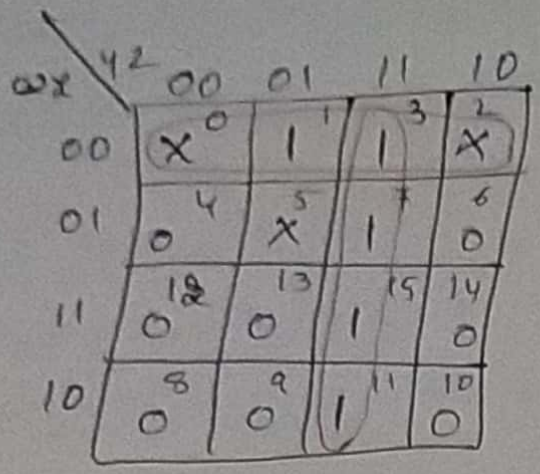
simplify the boolean function

$$F(w,x,y,z) = \Sigma(1,3,7,11,15)$$

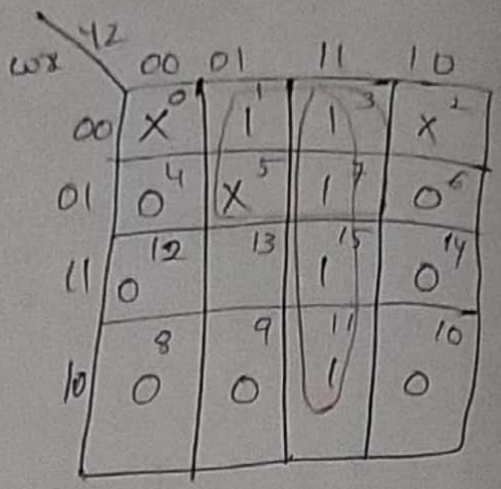
which has don't Care Conditions

$$d(w,x,y,z) = \Sigma(0,2,5)$$

Don't Care conditions:



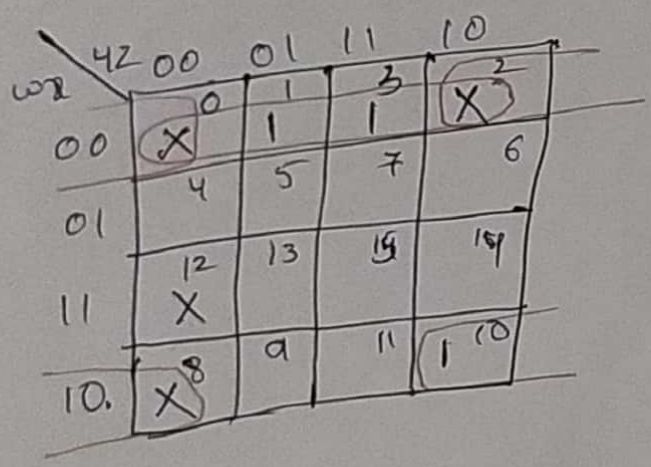
$F = yz + w'x'$



$F = yz + w'z$

→ simplify the boolean function

$F(w,x,y,z) = \Sigma(1,3,10) + \Sigma_d(0,2,8,12)$

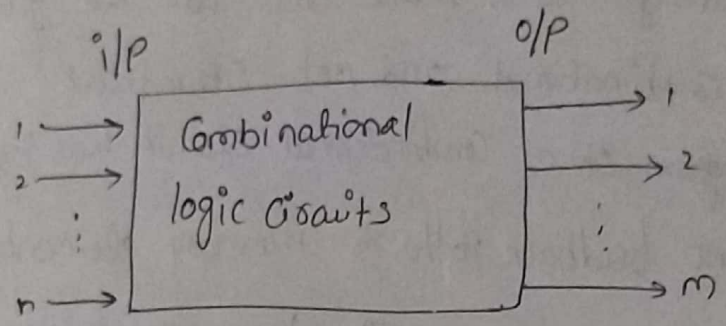


$w'x' + x'z'$

Combinational Circuits :-

A combinational circuit consists of an interconnection of logic gates. Combinational logic gates react to the values of the signal at their inputs and produce the value of the output signal, transforming binary information from the given input data to a required output data.

The circuits do not make use of any memory or storage device. Output is depends upon combination of input variables.



It can have n number of inputs and m number of outputs.

eg: (i) adders & subtractors
(ii) Decoders
3) MUX
code converters
Comparators
ROM
PLA (Programmable logic Array)
PAL (Programmable array logic)

In many applications, the source and destination are storage registers. If the registers are included with the combinational gates, then the total circuit must be considered to be a sequential circuit

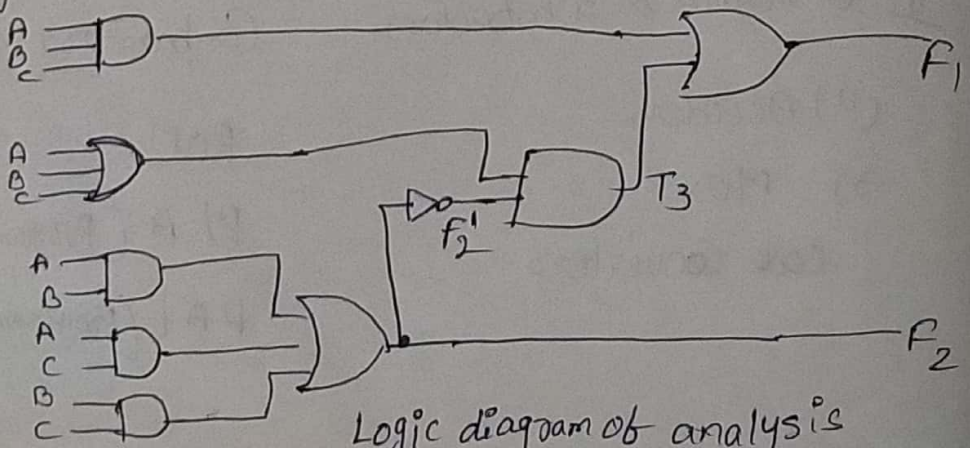
Analysis Procedure:

The analysis of a combinational circuits require that we determine the function that the circuit implements. This task start with a given logic diagram and culminates with a set of Boolean functions, a true table or an explanation of the circuit operation.

In the analysis is to make sure that the given circuit is combinational and not sequential

The diagram of a combinational circuit has logic gates with no feedback paths or memory elements

A feedback path is a connection from the output of one gate to the input to the second gate whose output forms part of the input to the first gates



Logic diagram of analysis

The analysis of the combinational circuit illustrates the proposed procedure. We note that the circuit has three binary inputs A, B and C and two binary outputs F_1 and F_2. The outputs of various gates are labeled with intermediate symbols. The outputs of gates that are a function only of input variables are T_1 and T_2. Output F_2 can easily derived from the input variables. The boolean function for these three outputs are

$$F_2 = AB + AC + BC$$
$$T_1 = A + B + C$$
$$T_2 = ABC$$

Next we consider output of gates that are a function of already defined symbols.

$$T_3 = F_2' T_1$$
$$F_1 = T_3 + T_2$$

To obtain F_1 as a function of A, B and C we form a series of substitutions.

$$F_1 = T_3 + T_2 = F_2' T_1 + ABC = (AB + AC + BC)'(A + B + C) + ABC$$
$$= (A' + B')(A' + C')(B' + C')(A + B + C) + ABC$$
$$= (A' + B'C')(AB' + AC' + BC' + B'C) + ABC \quad \text{(Theorem 5)}$$
$$= A'BC' + A'B'C + AB'C' + ABC$$

If we want to pursue the investigation and determine the information transformation task achieved by this circuit, we can draw the circuit from the derived Boolean expressions and try to recognize a familiar operation.

The derivation of the truth table for a circuit is a straightforward process once the output Boolean functions are known. To obtain the truth table directly from the logical diagram without going through the derivation of the Boolean function.

A	B	C	F_2	F_2'	T_1	T_2	T_3	F_1
0	0	0	0	1	0	0	0	0
0	0	1	0	1	1	0	1	1
0	1	0	0	1	1	0	1	1
0	1	1	1	0	1	0	0	0
1	0	0	0	1	1	0	1	1
1	0	1	1	0	1	0	0	0
1	1	0	1	0	1	0	0	0
1	1	1	1	0	1	1	0	1

Design Procedure :

The Design combinational Circuit starts from the specification of the design objective and culminates in a logic circuit diagram or a set of Boolean functions from which the logic diagram can be obtained the procedure involves the following steps :

1. From the specifications of the circuit, determine the required number of inputs and outputs and assign a symbol to each
2. Derive the tooth table that defines the required relationship between inputs and outputs.
3. Obtain the simplified Boolean function for each output as a function of input variables.
4. Draw the logic diagram and verify the correctness of the design.

A tooth table for a combinational circuit consists of input columns & output columns. The input columns are obtained from the 2^n binary numbers for the n input variables. The binary values for the outputs are determined from the stated specifications. The output ~~specification~~ functions specified in the true table give the exact defination of the combinational circuit. It is important that the verbal specifications be interpreted correctly in the true table as they are often incomplete and any wrong interpretation may result in an incorrect true table.

The output binary functions listed in the truth table are simplified by any available method, such as algebraic manipulation, the map method or a computer based simplification program. There is a variety of simplified expression from which to choose. A practical design must consider such constraints as the number of gates, number of inputs to a gates propagation time of the signal through the gates, number of interconnections, limitations of the driving capability of each gate and various other criteria that must be taken into consideration when design integrated circuits. The importance of each constraint is dictated by the particular application, it is difficult to make a general statement about what constitutes an acceptable implementation.

Design steps for a Combinational Circuit:

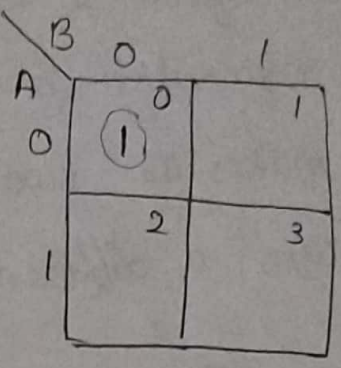
→ Observe the problem defination
→ Determine the required input & output variables
→ assign letters (or) symbols to the input variables
→ make truth table that defines required relationship
→ Determine the simplified Boolean Expression using K-map
→ Draw logic diagram

eg: Design a combinational circuit with two inputs which produce output as logic 0 when any one input is one.

→ i/p - A, B o/p - Y

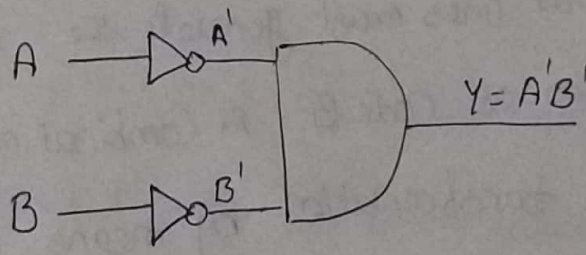
A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0

K-MAP



double quad octet

$Y = A'B'$ (Boolean expression)



The availability of a large variety of codes for the same discrete elements of information results in the use of different codes by different digital systems. It is sometimes necessary to use the output of one system as the input to another.

A conversion circuit must be inserted between the two system if each uses different codes for the same information. a code converter is a circuit that makes the two systems compatible even though each uses a different binary code.

To convert from binary code A to binary code B, the input lines must supply the bit combination of elements as specified by code A and the output lines must generate the corresponding bit combination of code B. A combinational circuit performs this transformation by means of logic gates.

The design procedure will be illustrated by example that convert BCD to the excess-3 code for the decimal digits.

Input BCD				Output Excess-3 Code			
A	B	C	D	w	x	y	z
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0

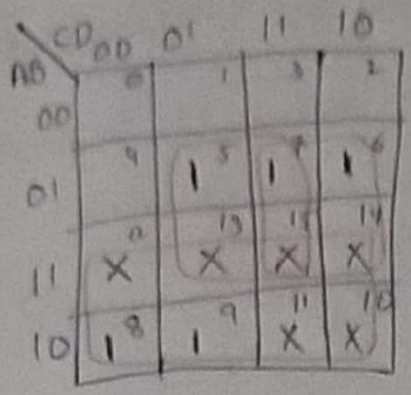
BCD inputs are A, B, C, D and outputs of Excess-3 is w, x, y, z. The four binary variables have 16 bit combinations but only 10 are listed in the true table. The six bit combinations not listed for the input variables are don't care combinations. These values have no meaning in BCD and we assume that they will never occur in actual operation of the circuit.

The maps are plotted to obtain simplified Boolean functions for the outputs. Each one of the four maps represents one of the four o/p of the circuits as a function of the four input variable. The 1's marked inside the squares are obtain from the minterms that make the output equal to 1. The 1's are obtained from the ~~minterms~~ truth table by going over the output columns one at a time.

The column under output z has five 1's. Therefore the map for z has five 1's. There for the map for z has five 1's each being in a square corresponding to the minterm that makes z equal to 1. The six don't care minterms 10 through 15 are marked with an x.

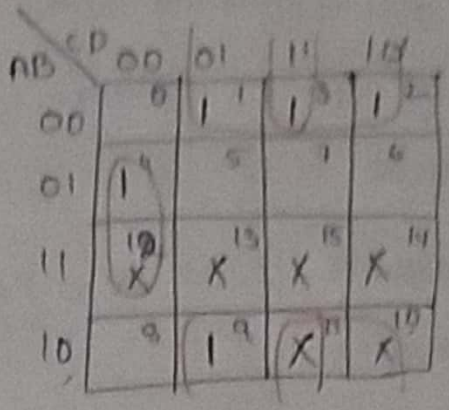
A two level logic diagram for each output may be obtained directly from the Boolean expression derived from the maps.

w



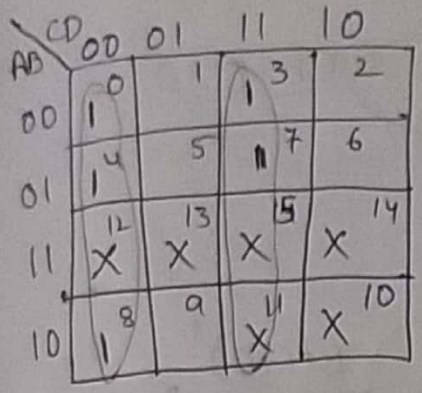
$w = A + BC + BD$

x

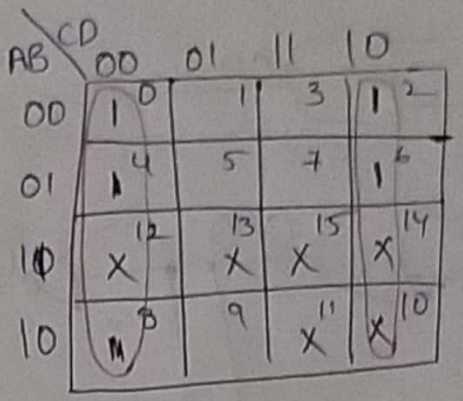


$x = B'C + B'D + BC'D'$

y



$y = C'D' + CD$



$z = D'$

$w = A + B(C+D)$

$x = B'(C+D) + BC'D'$

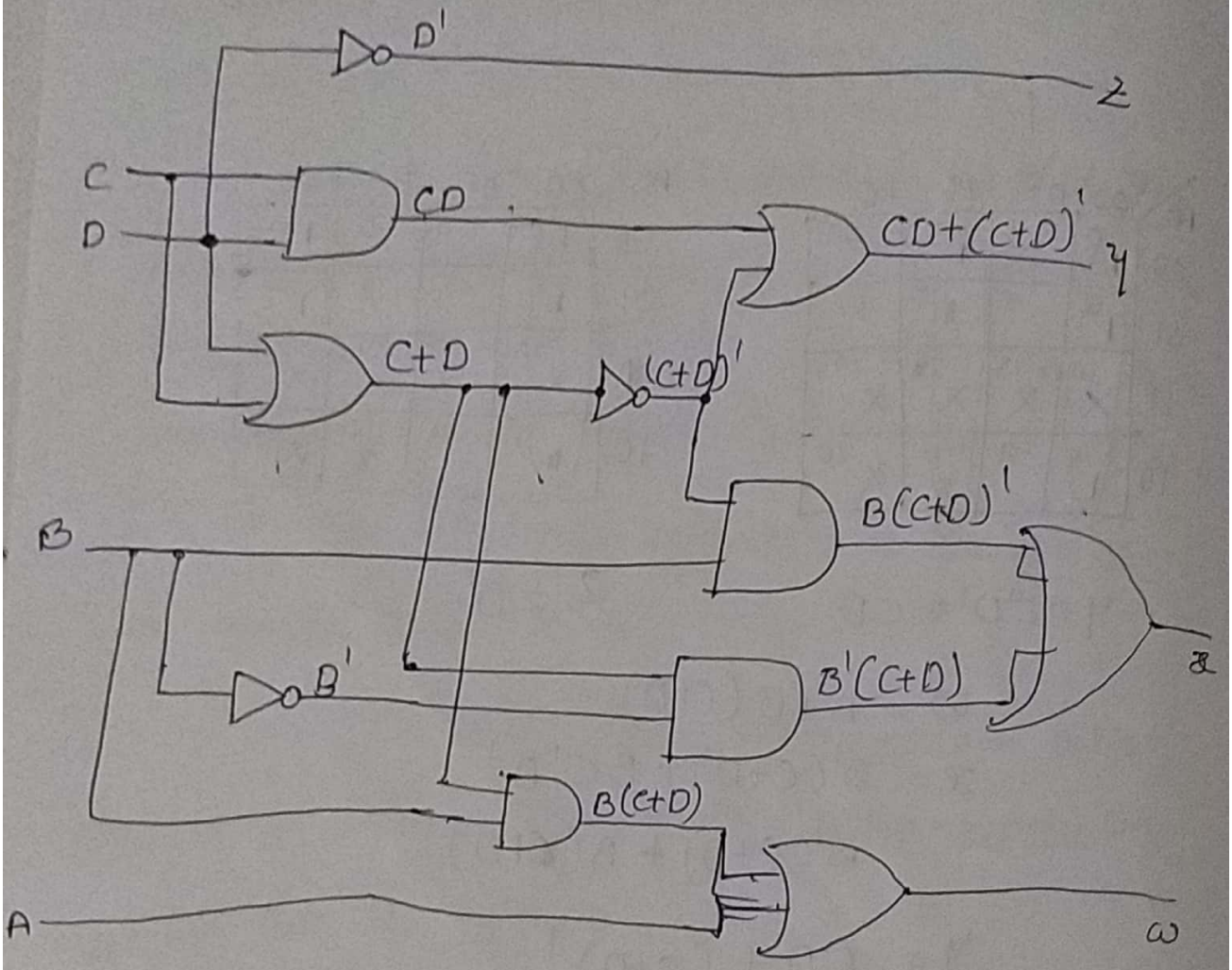
$= B'(C+D) + B(C+D)'$

$y = CD + (C+D)'$

$z = D'$

The logic diagram that implements these expressions that the OR gate whose output is C+D has been used to implement partially each of three o/p's

Not counting input inverters, the implementation in sum of products form requires seven AND gates and 3 OR gates. The implementation require four AND gates, 4 OR gates and one inverter.



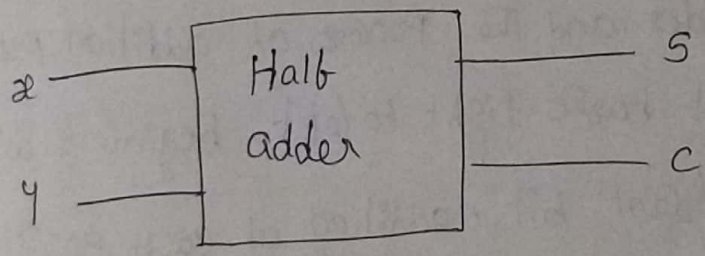
Binary Adder-Subtractor

Digital Computers perform a variety of information-processing tasks. Among the functions encountered are the various arithmatic operations. The most basic arithmatic operations is the addition of two binary digits. This simple addition consists of four possible elementary operation: 0+0=0 0+1=1 1+0=1 and 1+1=10. The first three operations produce a sum of one digit but when both augend and addend bits are equal to 1. The binary sum consist of two digits. The higher significant of this result is called Carry. When augend and addend numbers contain more significant digits, the Carry obtained from the addition of the two bits added to the next higher order pair of significant bits. A Combinational Circuit that performs the addition of two bits is called halfadder one that perform the addition of three bits is full adder.

A binary adder-subtractor is a combinational circuit that performs the arithmatic operations of addition and subtraction with binary number.

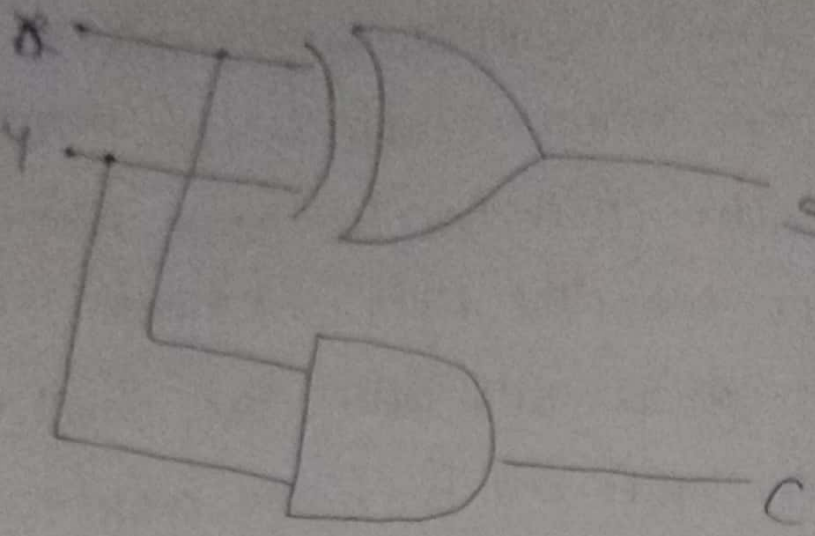
Half adder :-

An adder is a digital logic circuit in electronics that implements addition of numbers The half adder circuit has two inputs A and B which add two input digits and generate a Carry and sum. In the half adder the circuit need two binary inputs and two binary outputs The input variable designate the augend and addend bits The output variables produce sum and Carry we assign symbols x and y to the two inputs and S(for sum) and C (for Carry) to the outputs



Block diagram of Half adder.

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



Half adder $S = xy' + x'y = x \oplus y$

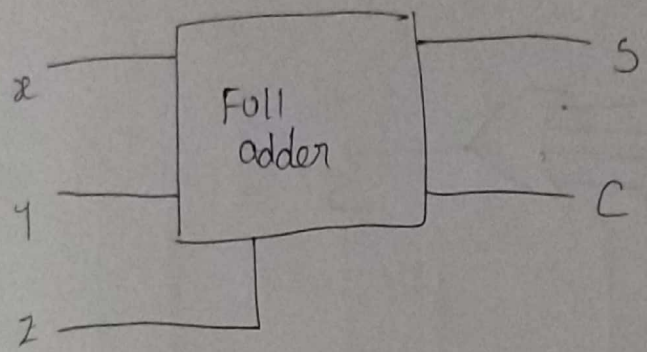
$C = x \cdot y$

Full adder :

Addition of n-bit binary nombers requires the use of a full adder, and the process of addition proceeds on a bit-by-bit basis right to left begining with the least significant bit, addition at each position adds not only the respective bits of the words but must also consider a possible carry bit from addition at the previous position.

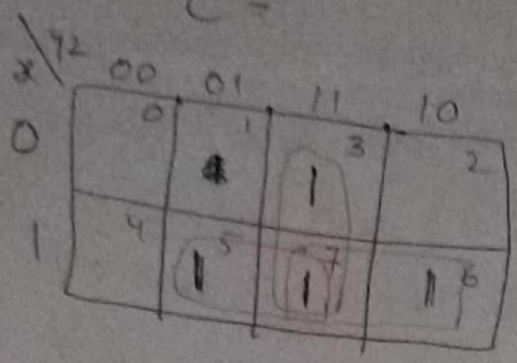
A full adder is a combinational circuit that forms the arithomatic sum of three bits. It consists of three inputs and two outputs. Two of the input variables, denoted by x and y represents the two significant bits to be added.

The third input z represents the carry from the previous lower significant position. Two outputs are necessary because the arithmetic sum of three binary digits ranges in value from 0 to 3, and binary representation of 2 or 3 needs two bits. The two outputs designated by the symbols S for sum and C for carry. The binary variable S gives the value of the least significant bit of the sum. The binary variable C gives the output carry formed by adding the input carry and the bits of the words.



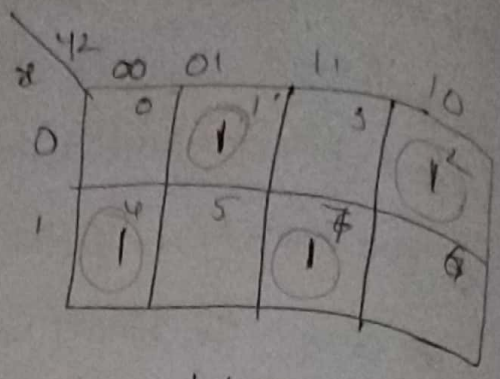
x	y	z	S	C
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

C =



$C = xz + xy + yz$

S

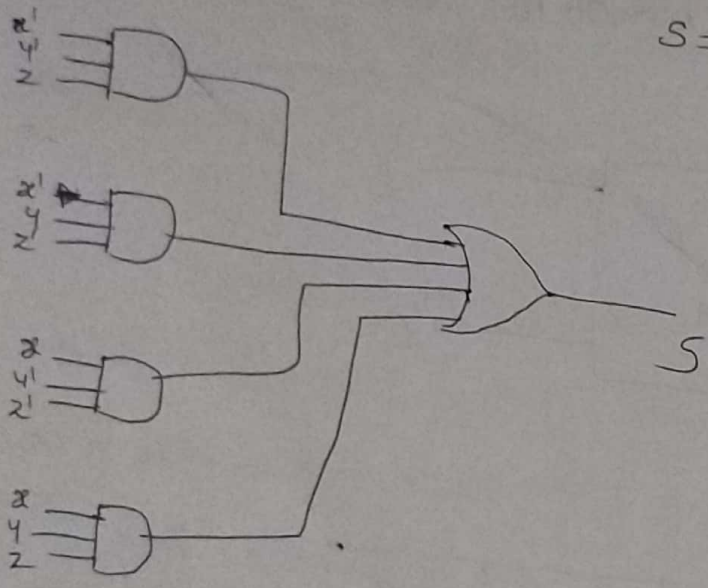


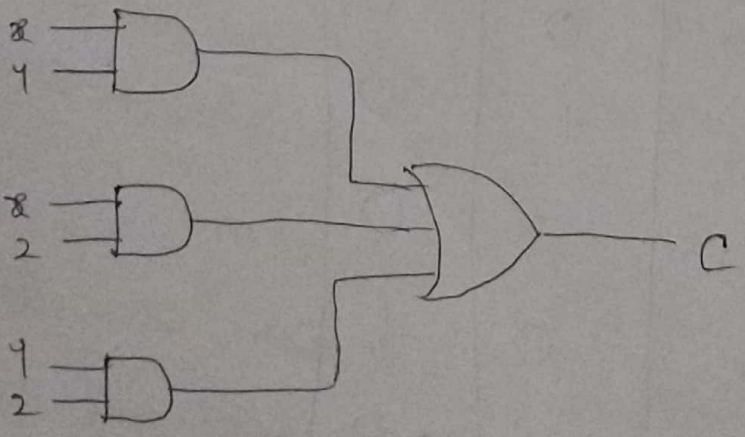
$S = x'y'z + x'yz' + xy'z'$

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

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

$S =$

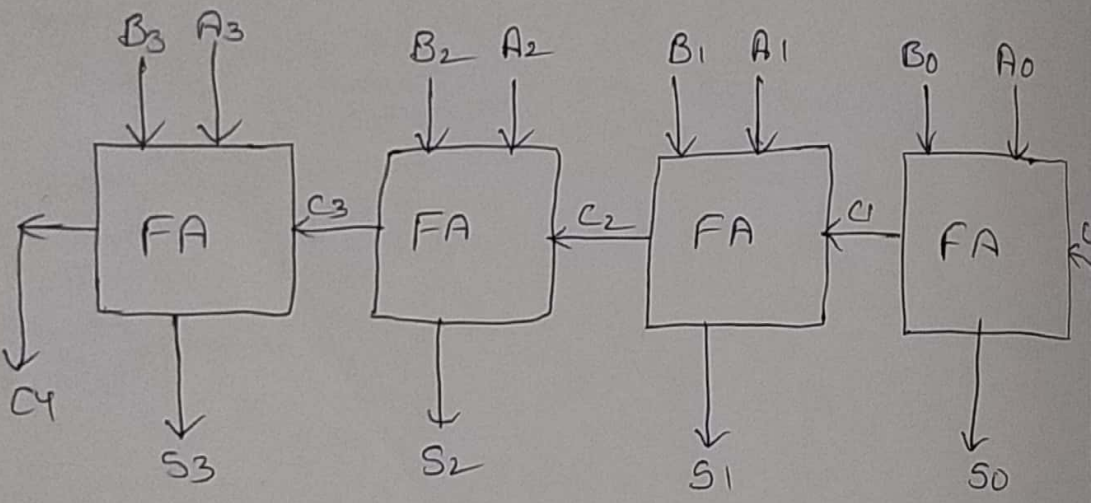




Binary Adder :-

A binary adder is a digital circuit that produces the arithmatic sum of two binary numbers. It can be constructed with full adders connected in cascade with the output carry from each full adder connected to the input carry of the next full adder in the chain.

Addition of n-bit numbers requires a chain of n full adders or a chain of one half adder and n-1 full adders.



Four full adder circuits to provide a four bit ripple carry adder. The augend bits of A and the addend bits of B are designated by subscript number from right to left with subscript 0 denoting the least significant bit. The carrys are connected in a chain through the full adders. The input carry to the adder is Co

and it ripples through the full adders to the output Carry C_4. The S outputs generate the required sum bits. An n-bit adder requires n full adders with each output Carry connected to the input Carry of the next higher order full adder.

$A = 1011$

$B = 0011$

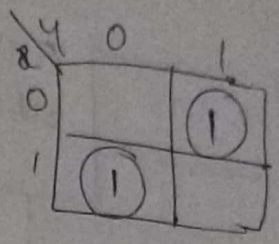
$C_i = 0110$

0	1	1	0	$-C_i$
1	0	1	1	$-A_0$
0	0	1	1	$-B_0$
1	1	1	0	$-S_0$
0	0	1	1	$-C_{i+1}$

Half subtractor :-

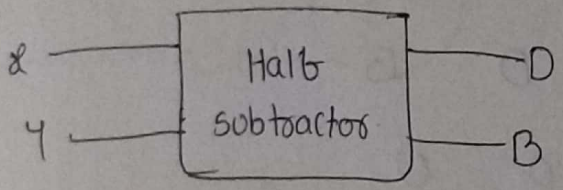
A Half subtractor and it has two inputs and two outputs. The two input x and y from the minuend and the subtrahend. D is the difference output and B is the borrow output.

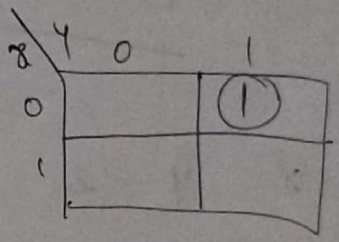
x	y	D	B
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0



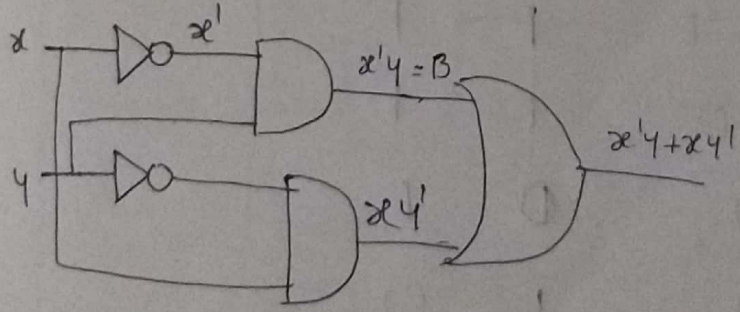
$xy' + x'y$

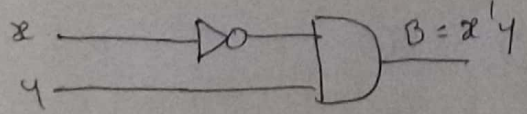
$D = x \oplus y$

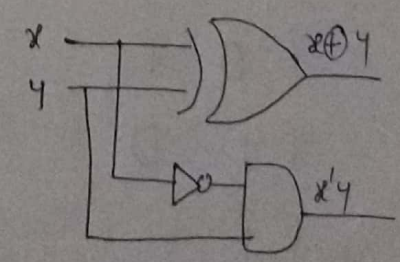




$B = x'y$





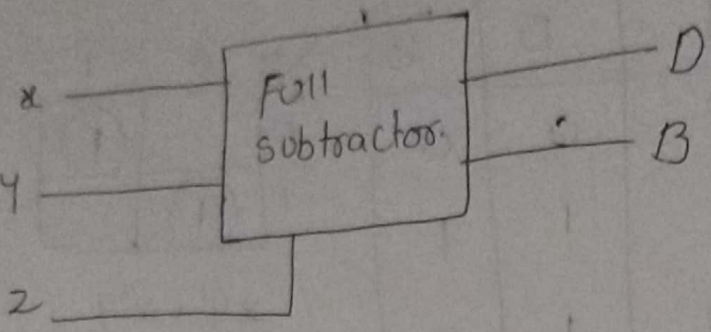


$D = x \oplus y$

$B = x'y$

Full subtractor

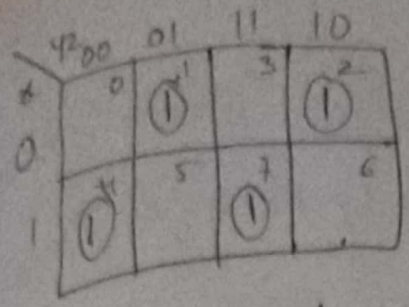
A full subtractor has three inputs and two outputs x, y, and z are the inputs to be subtracted in which z represents borrow from the next stage. D and B are the outputs



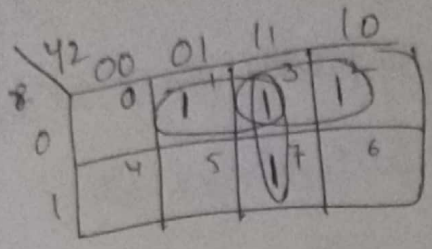
Block Diagram of Full subtractor

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

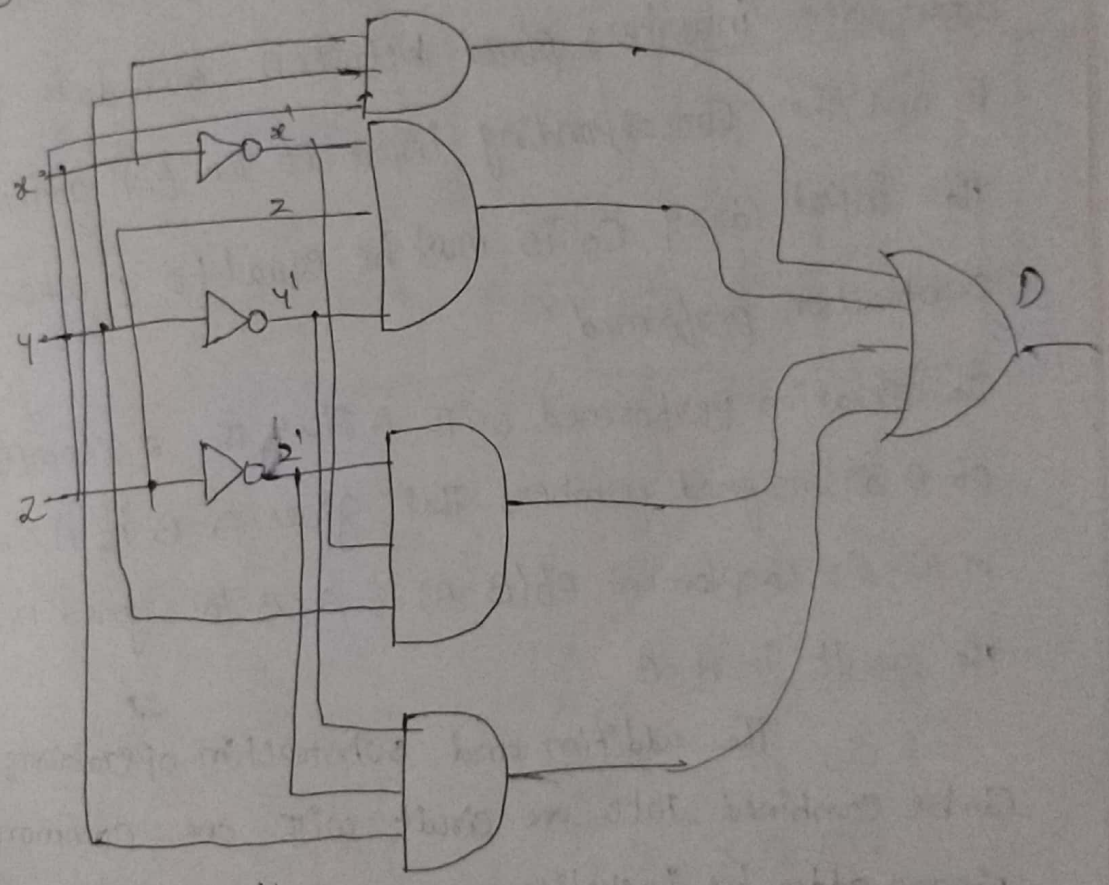
D

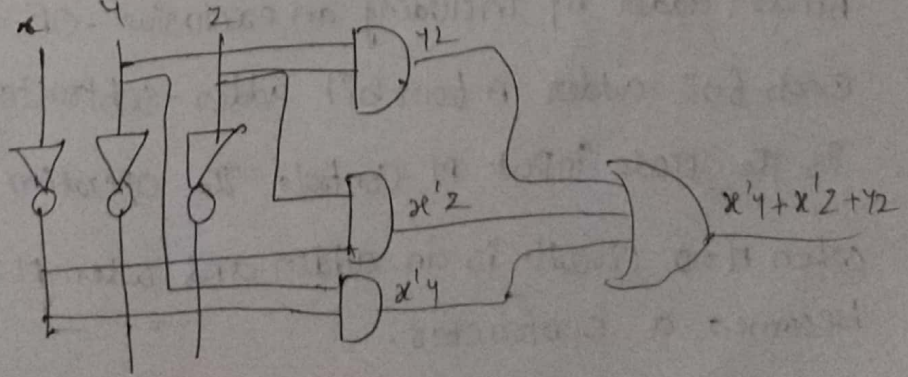


$D = x'y'z + x'yz' + xy'z' + xyz$



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





Binary Subtraction

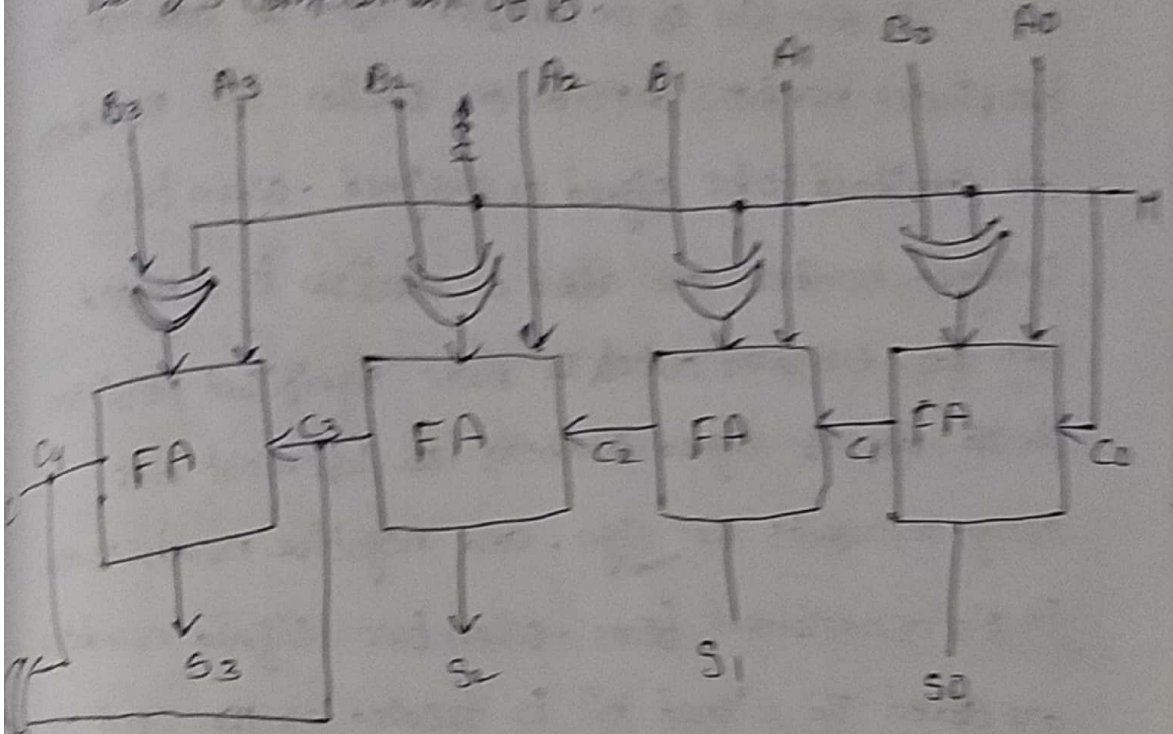
The subtraction of unsigned binary numbers can be done most conveniently by means of Complements The subtraction A-B can be done by taking the 2's Complement of B and adding it to A The 2's Complement can be obtained by taking the 1's Complement and adding 1 to least significant pair of bits

The Circuit for subtracting A-B Consists of an adder with inverters placed between each data input B and the Corresponding input of the full adder. The input Carry Co is must be equal to 1 when subtraction performed.

The operation performed with A Plus the 2's Complement of B for unsigned numbers. That gives A-B if A ≥ B or the 2's Complement of (B-A) if A < B for signed numbers The result is A-B

The addition and subtraction operations Can be Combined into one circuit with one Common binary adder by including an exclusive-OR gate with each full adder A four bit adder-subtractor circuit is the Mode input M Controls the operation when M=0 Circuit is an adder and when M=1 the Circuit becomes a subtractor.

Each Ex-OR gate receives input M and one of the inputs of B. When M=0 we have B⊕0=B. The full adders receives the value of B. The input Carry is 0 and the circuit performs A+B. when M=1, we have B⊕1=B' and Co=1. The B inputs are all complemented and a 1 is added through the input Carry. The circuit performs the operation A plus the 2's complement of B.



A=6 0110
B=3 0011
1001

A=6 0110
B=-3 1101
+3 x0011
0011

Overflow

When two numbers with n digits each are added and the sum is a number occupying n+1 digits, we say that an overflow occurred. This is true for binary or decimal numbers, signed or unsigned when the addition is performed in digital computers because the number of bits that hold the number is finite and a result that contain n+1 bits cannot be accommodated by an n-bit word.

The detection of an overflow after the addition of two binary numbers depends on whether the numbers are considered to be signed or unsigned. When two unsigned numbers are added an overflow is detected from the end carry out of the most significant position. In the case of signed numbers, the left most bit always represents the sign, and negative numbers are in 2's-complement form. When two signed numbers are added the signed bit is treated as the part of the number and the end carry does not indicate overflow.

An overflow cannot occur after an addition if one number is positive and the other is negative.

eg: Two signed binary numbers, +70 and +80 are stored in two eight-bit registers. The range of numbers that each register can accommodate is from binary +127 to binary -128 since the sum of the two numbers is +150, it exceeds the capacity of an eight bit register. This is also true for -70 and -80.

Carries 0 1

+70 01000110 : -70 10111010

+80 01010000 : -80 10110000

+150 10010110 : -150 01101010

1 0

70
2|35-0
2|17-1
2|8-1
2|4-0
2|2-0
1-0

An overflow condition can be detected by observing the carry into the sign bit position and the carry out of the sign bit position. If these two carrys are not equal an overflow has occured.

The binary adder subtractor circuit with outputs C and V. If the two binary numbers are considered to be unsigned then the C bit detects a carry after addition or a barrow after subtraction. If the numbers are signed then the V bit detects an overflow. If V=0 after addition or subtraction then no overflow occured. The n-bit result correct. If V=1 then the result of the operation contains n+1 bits but only the right most n bits of the numbers fit in the space available. So an overflow occured.

Decimal adder:

Computers that perform arithmatic operations directly in the decimal number system represents decimal numbers in binary coded form. A decimal adder requires a minimum of nine inputs and five outputs since four bits are required to code each decimal digit and the circuit must have an input and output carry. There is a wide variety of possible decimal adder circuits depending upon the code used to represent the decimal digits.

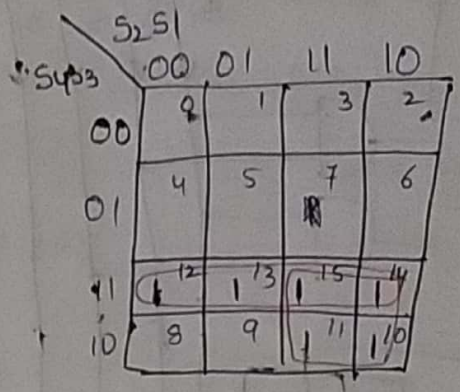
BCD Adder:

BCD adder a 4-bit binary adder that is capable of adding two-4bit words having a BCD format. The result of the addition is a BCD-format 4-bit output word, representing the decimal sum of the addend and augend and a carry that is generated if this sum exceeds a decimal value of 9.

BCD is a system of writing numerals that assignes a four-digit binary code to each digit 0 through 9 in a decimal numeral. The four-bit BCD code for the any particular single base-10 digits is its ~~bina~~ representation in binary notation.

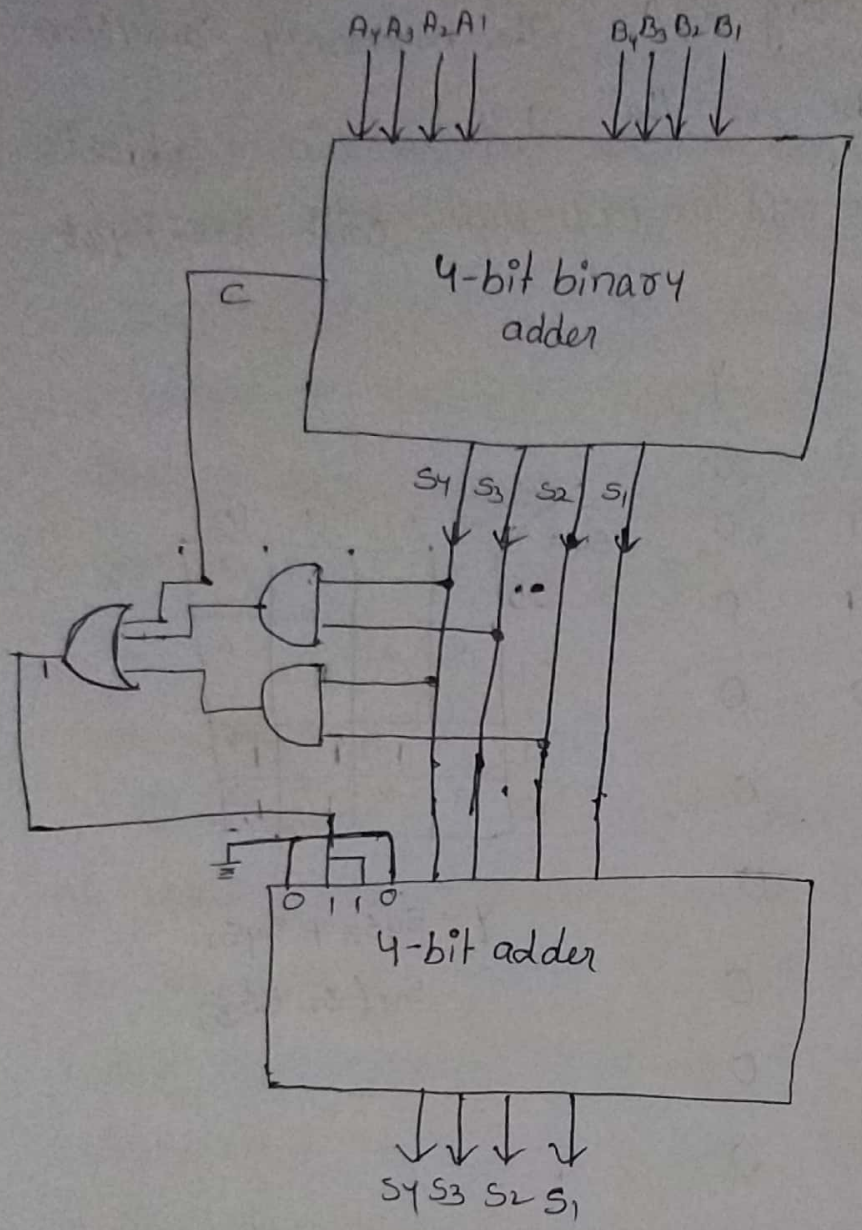
A BCD 1-digit adder is a circuit that adds two BCD digits in parallel and also produces the sum digit in BCD along with the necessary correction logic. It can be seen that a 4-bit binary adder is used initially to add two BCD digits with carry input.

S_4	S_3	S_2	S_1	Y
0	0	0	0	0
0	0	0	1	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1



$Y = S_4S_3 + S_4S_2$

$= S_4(S_2 + S_3)$

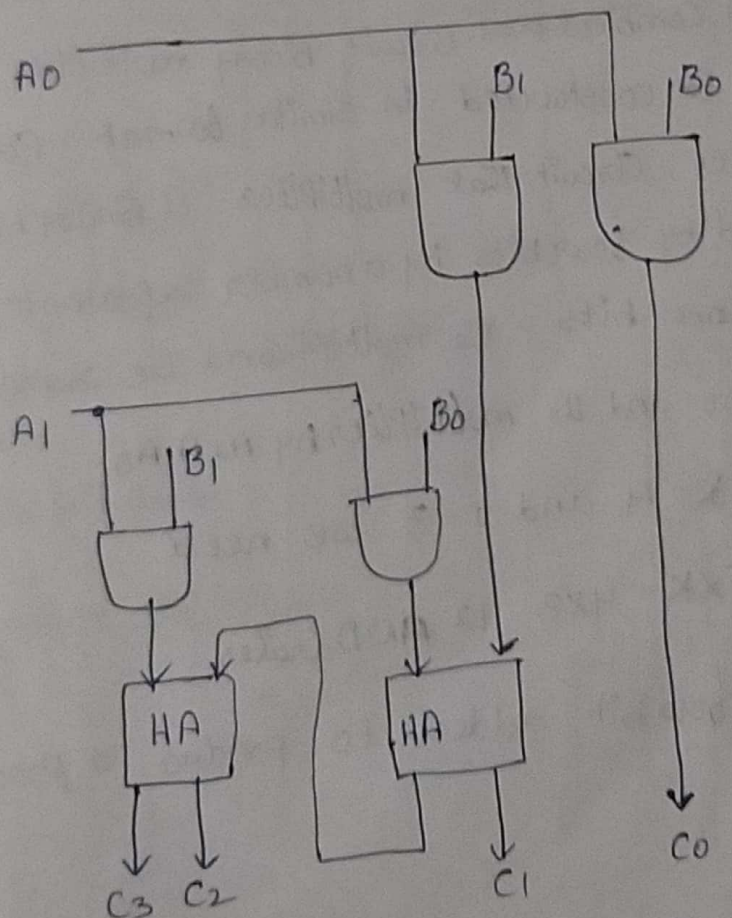


Block Diagram of a BCD Adder

Binary Multipliers

A Binary Multiplier is a Combinational logic circuit or digital device used for multiplying two binary numbers. The two numbers are more specifically known as multiplicand and multiplier and the result is known as a Product.

Multiplication of Binary numbers is performed in the same way as multiplication of Decimal numbers. The multiplicand is multiplied by each bit of the multiplier starting from least significant bit.



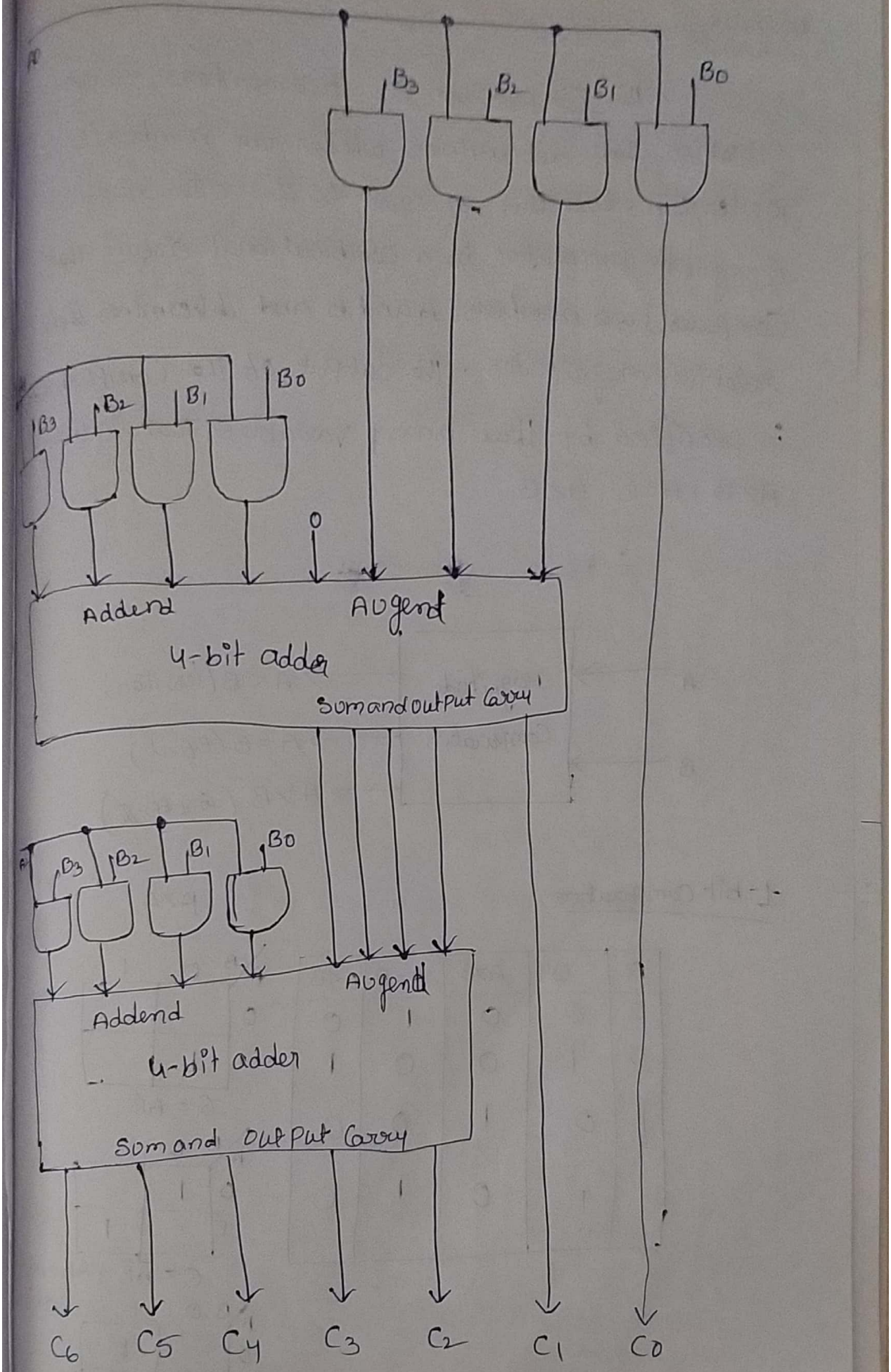
for example multiplication of two bit numbers the multiplicand bits are B_1 and B_0 multiplier bits are A_1 and A_0 and the product is $C_3C_2C_1C_0$. The first partial product formed by multiplying B_1B_0 by A_0. The multiplication of two bits such as A_0 and B_0 produces a 1 if both bits are 1. other wise it produces 0' this is identical to an AND operation. The second Partial product formed by multiplying B_1B_0 with A_1 and shifting one position to the left. the two partial products are added with two half adder circuits. for J multiplier bits K multiplicand bits we need (J x K) AND gates and (J-1) K-bit adders to produce product of (J+K) bit

A Combinational circuit binary multiplier with more bits can constructed in similar format. Consider a multiplier circuit that multiplies a binary number represented by. four bits, by a number represented by three bits. the multiplicand be represented by $B_3B_2B_1B_0$ and the multiplier by $A_2A_1A_0$.

since K = 4 and J = 3 we need

$$J \times K = 4 \times 3 = 12 \text{ AND Gates}$$

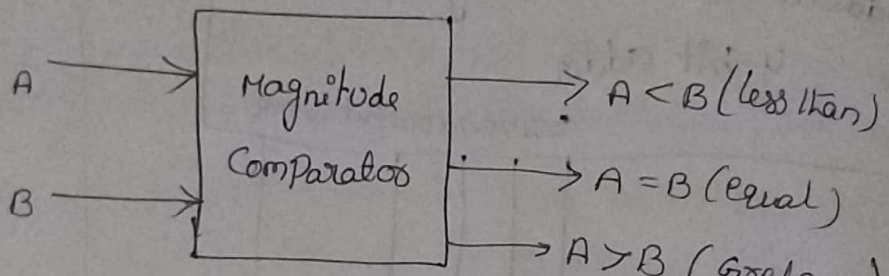
and two four bit adders. to produce a product of 7 bits



4-bit by Three bit binary multipliers

Magnitude Comparator

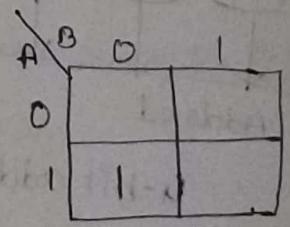
The Comparison of two numbers is an operation that determines whether one number is greater than, less than or equal to the other number. A magnitude Comparator is a combinational circuit that Compares two numbers A and B and determines their relative magnitudes. The output of the Comparison is specified by three binary variables that indicates A>B, A=B, A<B.



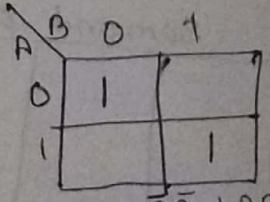
1-bit comparator :-

A	B	A>B	A=B	A<B
0	0	0	1	0
0	1	0	0	1
1	0	1	0	0
1	1	0	1	0

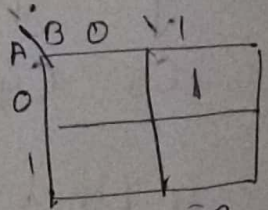
A>B



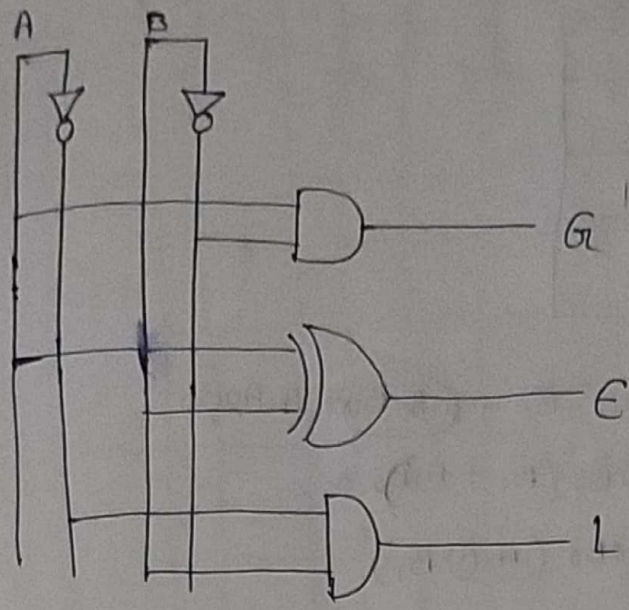
$G = A\bar{B}$



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



$L = \bar{A}B$



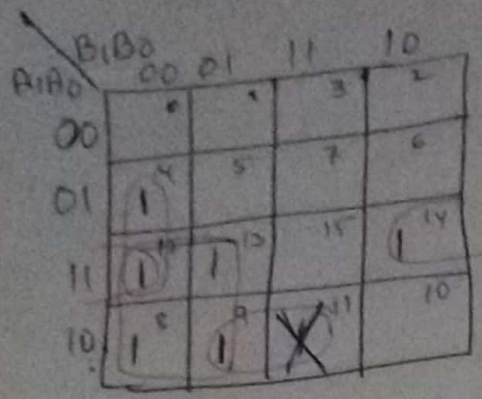
2 bit Comparator :-

$A = A_1 A_0$

$B = B_1 B_0$

A_1	A_0	B_1	B_0	$A > B$	$A = B$	$A < B$
0	0	0	0	0	1	0
0	0	0	1	0	0	1
0	0	1	0	0	0	1
0	0	1	1	0	0	1
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	1
1	0	0	0	1	0	0
1	0	0	1	1	0	0
1	0	1	0	0	0	0
1	0	1	1	0	0	1
1	1	0	0	1	0	0
1	1	0	1	1	0	0
1	1	1	0	1	0	0

A>B

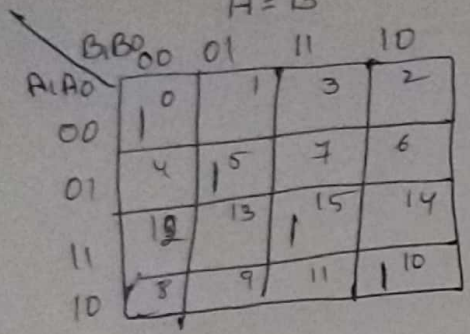


$G = A_1\bar{B}_1 + \bar{A}_0\bar{B}_1\bar{B}_0 + A_1\bar{B}_0B_0 + A_1A_0\bar{B}_0$

$= A_1\bar{B}_1 + A_0\bar{B}_0(\bar{B}_1 + A_1\bar{B}_0)$

$= A_1\bar{B}_1 + A_0\bar{B}_0(A_1 \odot B_1)$

A=B



$E = \bar{A}_1\bar{A}_0\bar{B}_1\bar{B}_0 + \bar{A}_1A_0\bar{B}_1B_0 + A_1A_0B_1B_0 + A_1\bar{A}_0B_1\bar{B}_0$

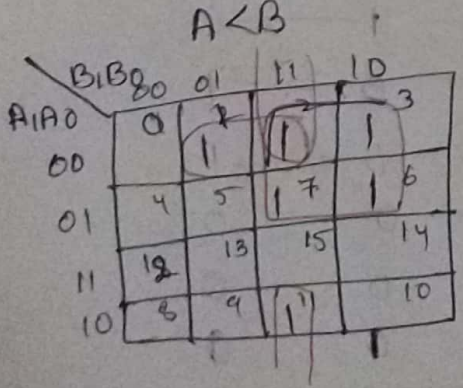
$= (A_1 \odot B_1)(A_0 \odot B_0)$

$= \bar{A}_1\bar{B}_1(\bar{A}_0\bar{B}_0 + A_0B_0) + A_1B_1(A_0B_0 + \bar{A}_0\bar{B}_0)$

$= (\bar{A}_0\bar{B}_0 + A_0B_0)(A_1B_1 + \bar{A}_1\bar{B}_1)$

$= (A_0 \odot B_0)(A_1 \odot B_1)$

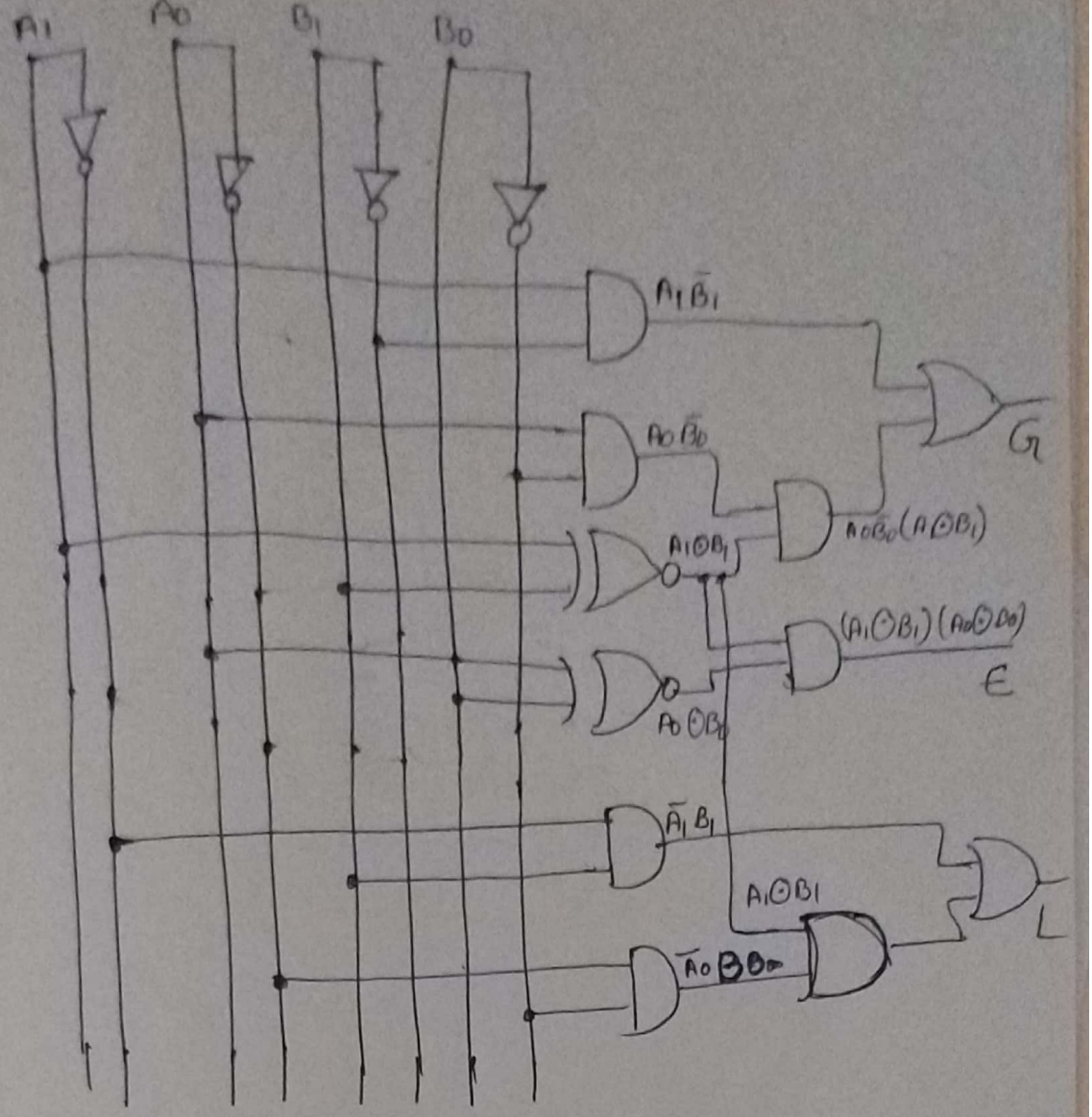
A<B



$L = \bar{A}_1B_1 + \bar{A}_1\bar{A}_0B_0 + \bar{A}_0B_1B_0$

$L = \bar{A}_1B_1 + \bar{A}_0B_0(\bar{A}_1 + B_1)$

$\bar{A}_1B_1 + \bar{A}_0B_0(A_1 \odot B_1)$



4-bit Comparator :-

$$G = A_3\bar{B}_3 + A_2\bar{B}_2(A_3 \odot B_3) + A_1\bar{B}_1(A_3 \odot B_3)(A_2 \odot B_2) + A_0\bar{B}_0(A_3 \odot B_3)(A_2 \odot B_2)(A_1 \odot B_1)$$

$$E = (A_3 \odot B_3)(A_2 \odot B_2)(A_1 \odot B_1)(A_0 \odot B_0)$$

$$L = \bar{A}_3 B_3 + \bar{A}_2 B_2(A_3 \odot B_3) + \bar{A}_1 B_1(A_3 \odot B_3)(A_2 \odot B_2) + \bar{A}_0 B_0(A_3 \odot B_3)(A_2 \odot B_2)(A_1 \odot B_1)$$