Analog & Digital Electronics

Course No: PH-218

Lec-31: Combinational Logic Modules

Course Instructor:

❖ **Dr. A. P. VAJPEYI**

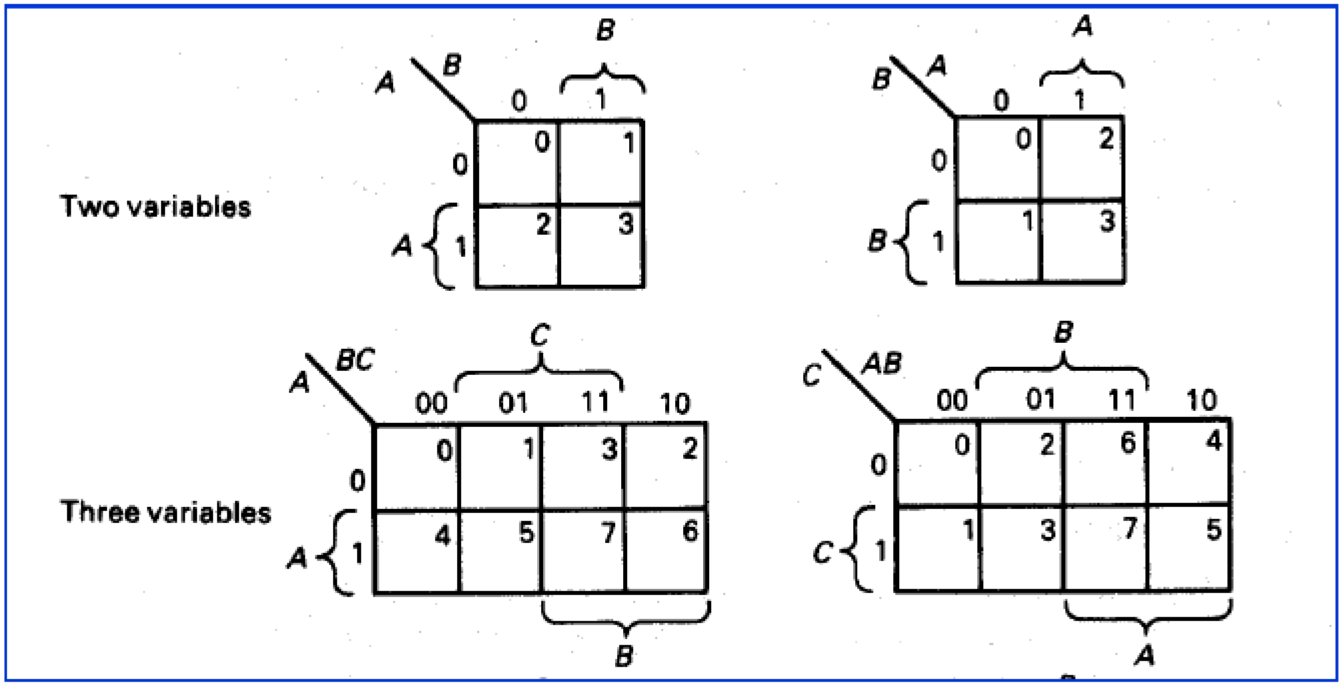
Department of Physics,
Indian Institute of Technology Guwahati, India

Karnaugh map minimization method

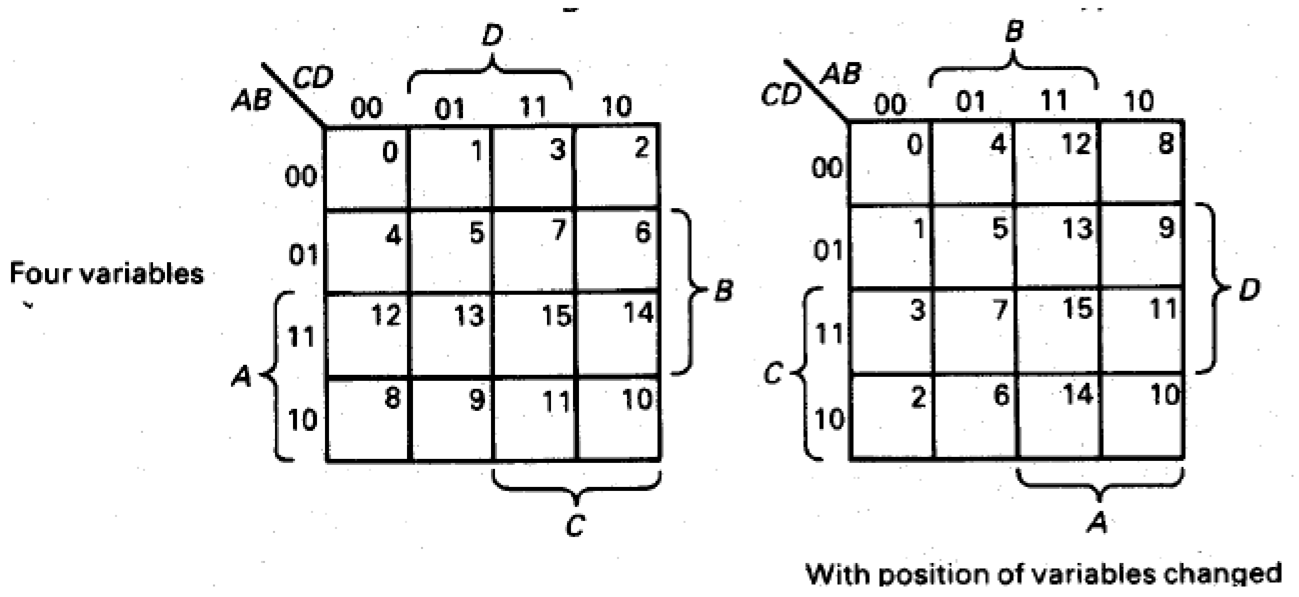
A Karnaugh map consists of a grid of squares, each square representing one canonical minterm combination of the variables or their inverse

– Arranged with squares representing minterms which differ by only one variable to be adjacent both vertically and horizontally.

– Squares on one edge of the map are regarded as adjacent to those on the opposite edge.



Karnaugh map minimization method



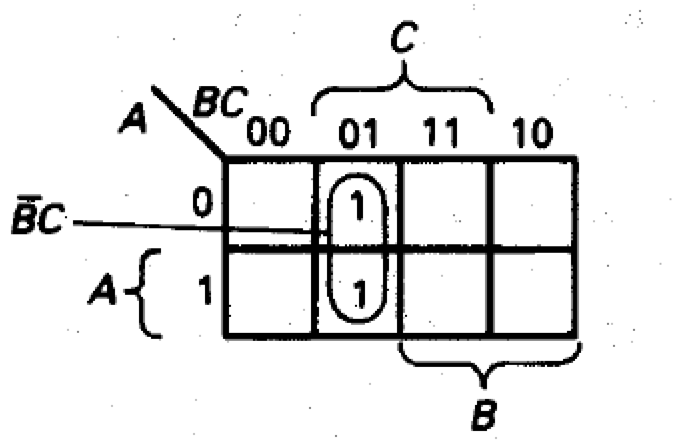
➢ The expression to be minimized should generally be in sum-of-product form.

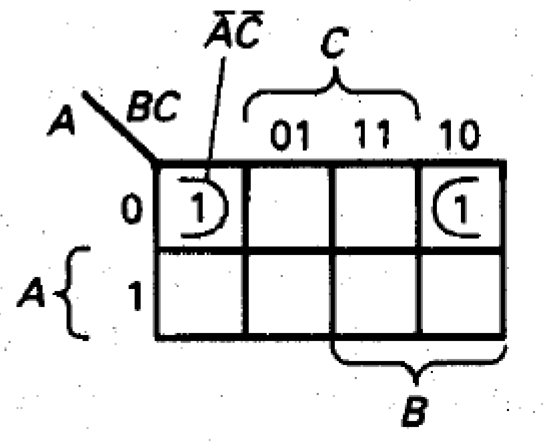
➢ The function is 'mapped' onto the Karnaugh map by marking a 1 in those squares corresponding to the terms in the expression to be simplified.

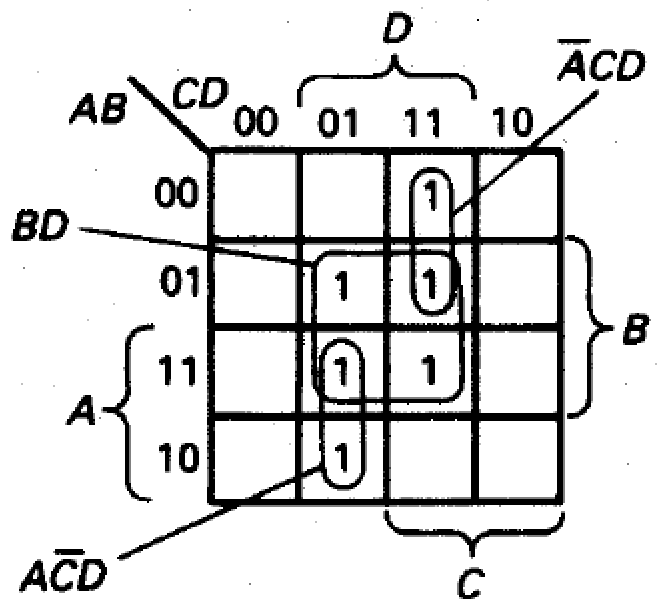
➢ If two or more pairs are also adjacent, these can also be combined using the same theorem.

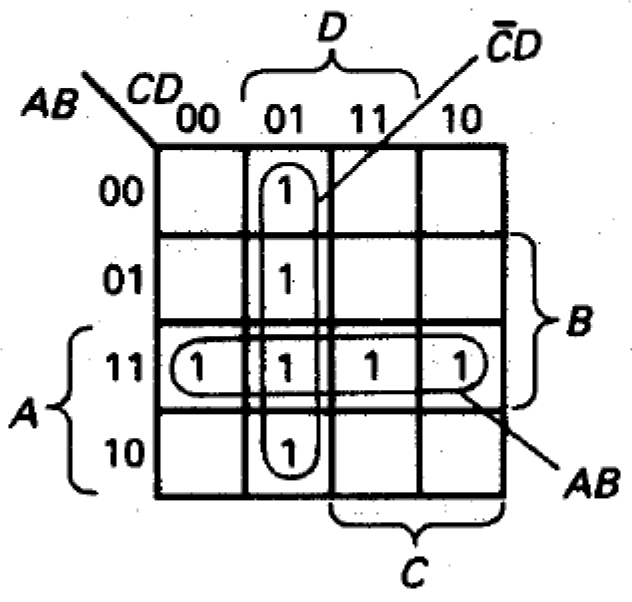
➢ The minimization procedure consists of recognizing multiple pairs in terms of 2, 4 or 8 cells.

Karnaugh map minimization method









Examples: Karnaugh map minimization method

Example: Simplify the Boolean function X = *A'B+A'B'C'+ ABC'+AB'C'*

Solution: First write the Boolean function interms of canonical minterm

A'B = A'B(C+C') = A'BC+A'BC'

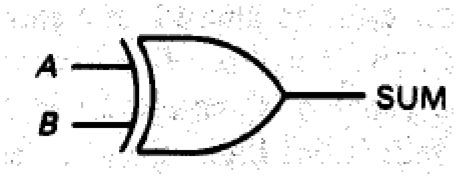
X = A'BC+A'BC' *+ A'B'C'+ ABC'+AB'C'*

Terms	C'	C
A'B'	1	
A'B	1	1
AB	1	
AB'	1	

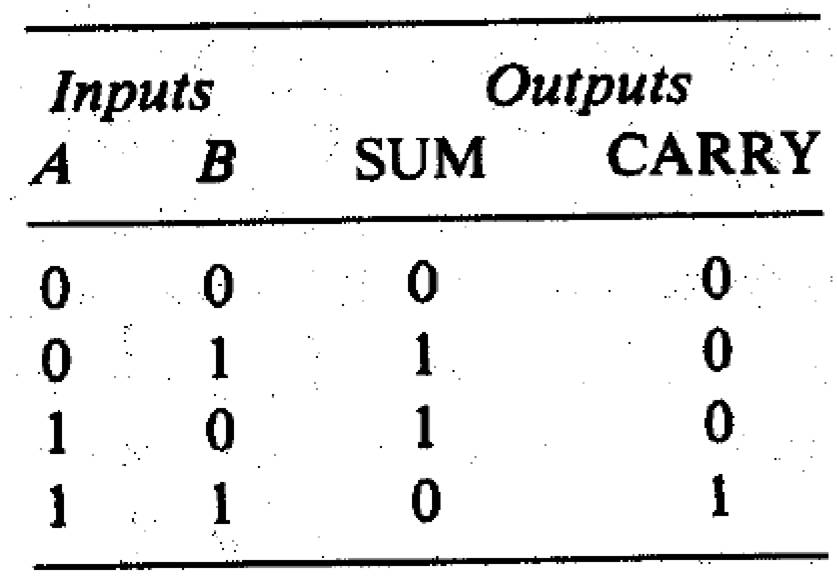
X = C'+A'B

Binary Adders: Half Adder Circuit

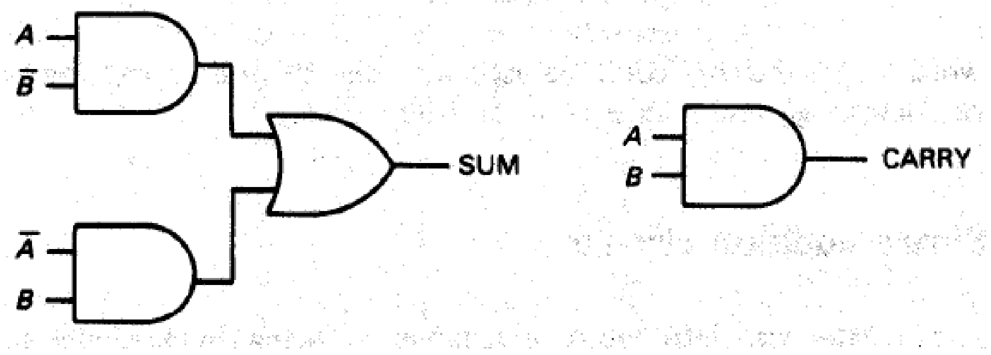
Half-adder: A logic circuit for the addition of two one bit numbers is referred to as an half-adder. The circuit accepts two inputs, A and B, and generates two outputs, SUM and CARRY according to the table.



□□ Half adder truth table



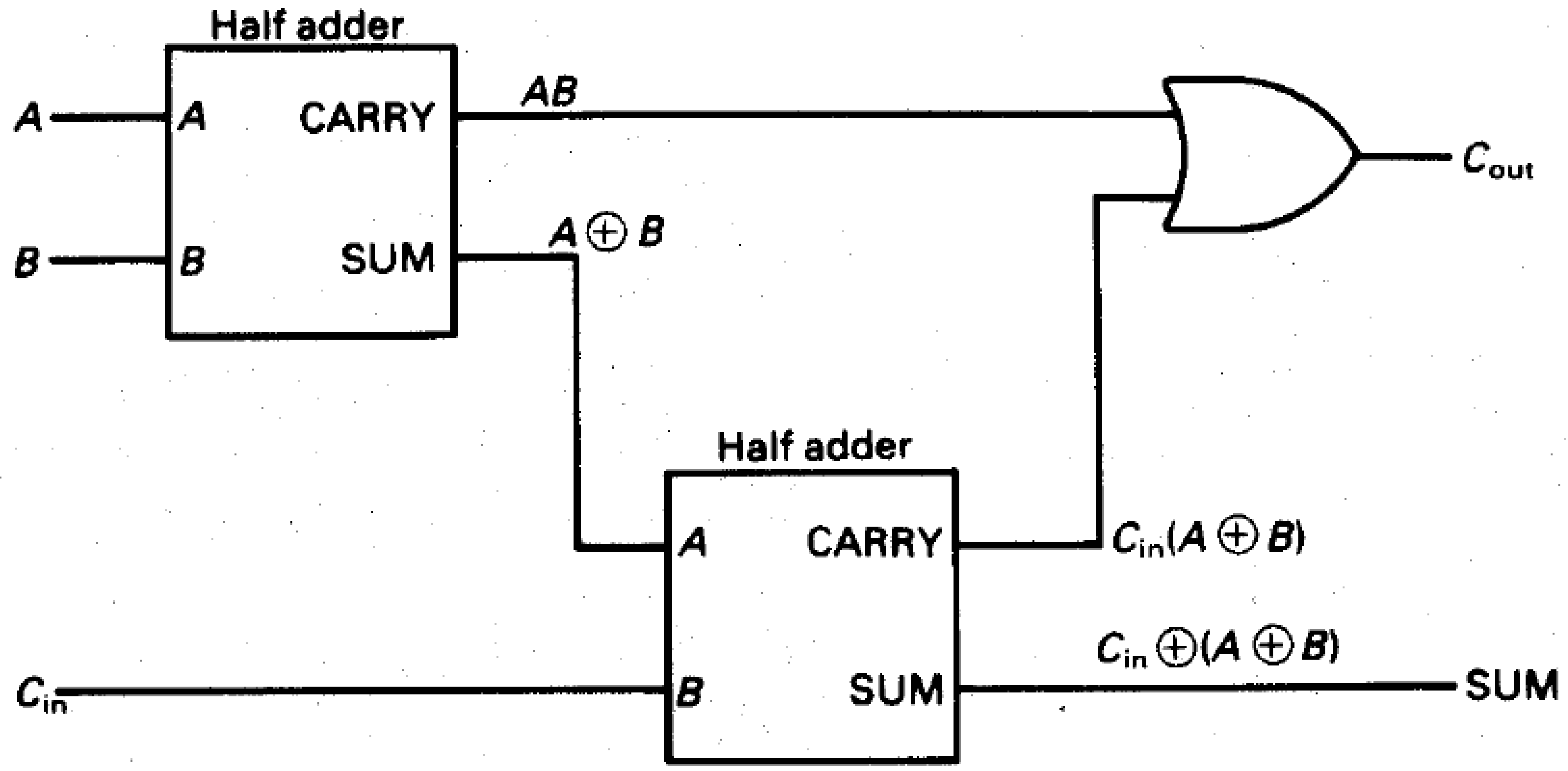
Inputs		*Outputs*	
A	*B*	SUM	CARRY
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



Sum =A B’+A’ B
Carry = A B

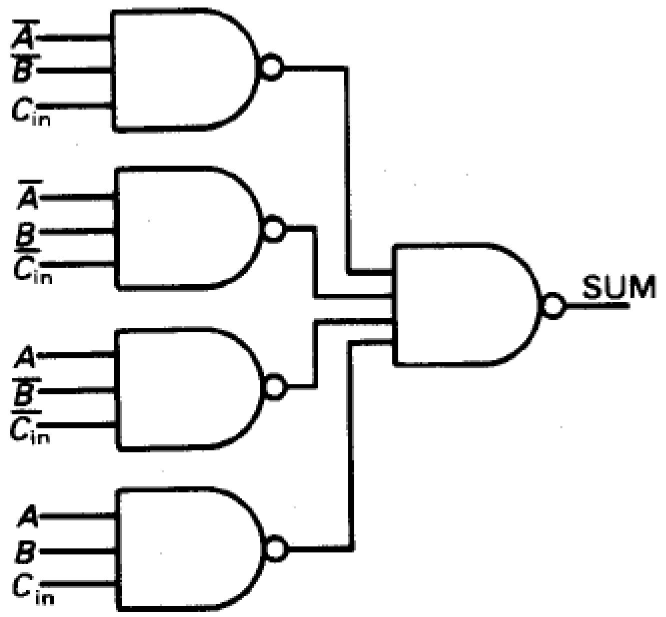
Binary Adders: Full Adder Circuit

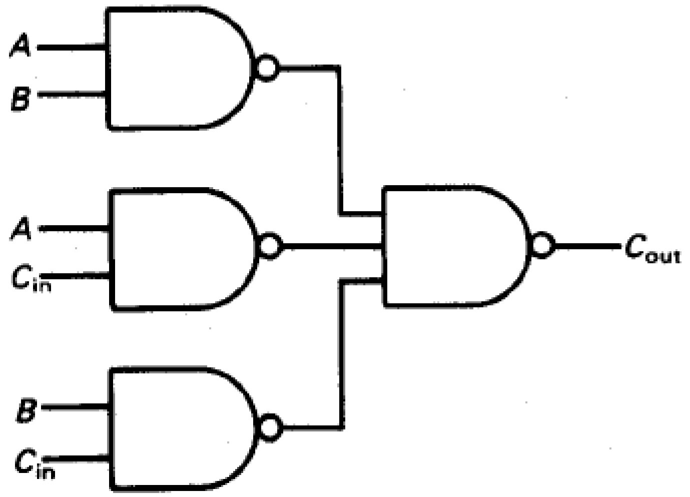
Full adder: adds two binary digits A and B together with a ‘carry-in’ from a previous addition. A full adder has three inputs, *A, B and ‘CARRY-IN’* (*Cin*), *and two outputs, SUM and* ‘CARRY OUT’ (C_{out})



$$Sum = C_{in} \oplus (A \oplus B) \qquad C_{out} = C_{in}(A \oplus B) + AB$$

Binary Adders: Full Adder Circuit





$Sum = A' B' C_{in} + A' B C_{in} + A B' C_{in} + A B C_{in}$

$\mathbf{C_{out}} = A' B C_{in} + A B' C_{in} + A B C'_{in} + A B C_{in}$

$Sum = C_{in} \oplus (A \oplus B)$

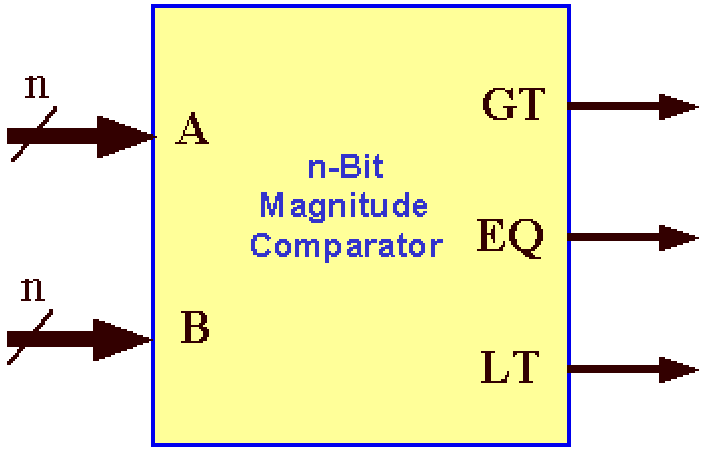
$C_{out} = C_{in}(A \oplus B) + AB$

Inputs			*Outputs*	
A	B	C_{in}	SUM	C_{out}
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

Magnitude Comparator

A magnitude comparator is a combinational circuit that compares two numbers A & B to determine whether: A > B, or A = B, or A < B

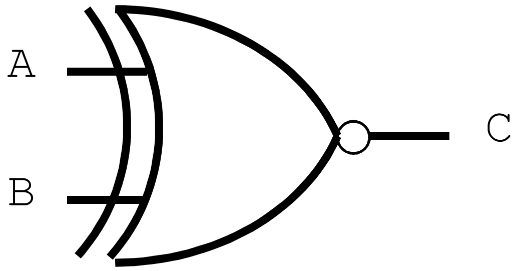
Consider two numbers, *A and B, with four* digits each; $A = A_3 A_2 A_1 A_0$ and $B = B_3 B_2 B_1 B_0$



An exclusive NOR gate is the easiest way to compare the equality of the bits.

X-NOR

A	B	C
0	0	1
0	1	0
1	0	0
1	1	1



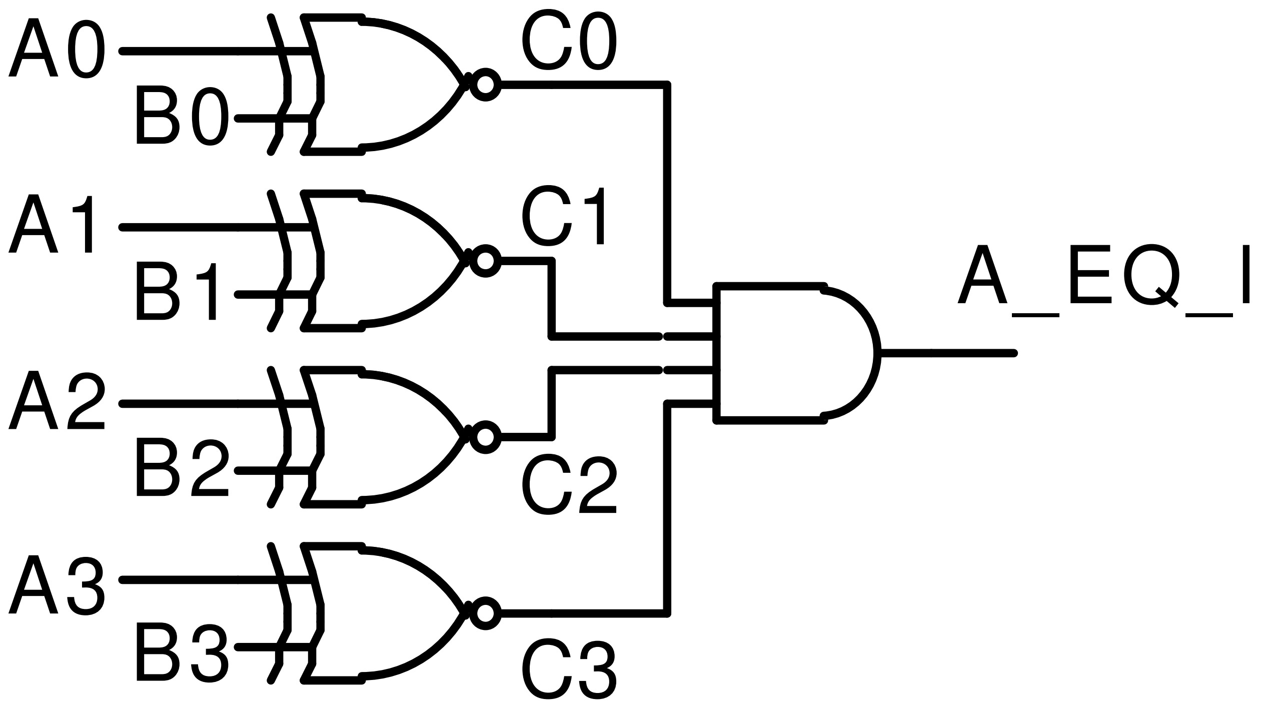
Inputs: First *n-bit number A*
Second *n-bit number B*
Outputs: 3 output signals (GT, EQ, LT), where:
1. GT = 1 IFF A > B
2. EQ = 1 IFF A = B
3. LT = 1 IFF A < B

Note: ***Exactly One of these 3 outputs equals 1, while the other 2 outputs are 0`s***

Equality Comparator

The equality relation of each pair of bits can be expressed logically with an equivalence function: $x_i = A_i B_i + A_i' B_i'$; *′ i = 0,1, 2, 3*

where $x_i = 1$ only if the pair of bits in position i are equal, i.e., if both are 1's or both are 0's

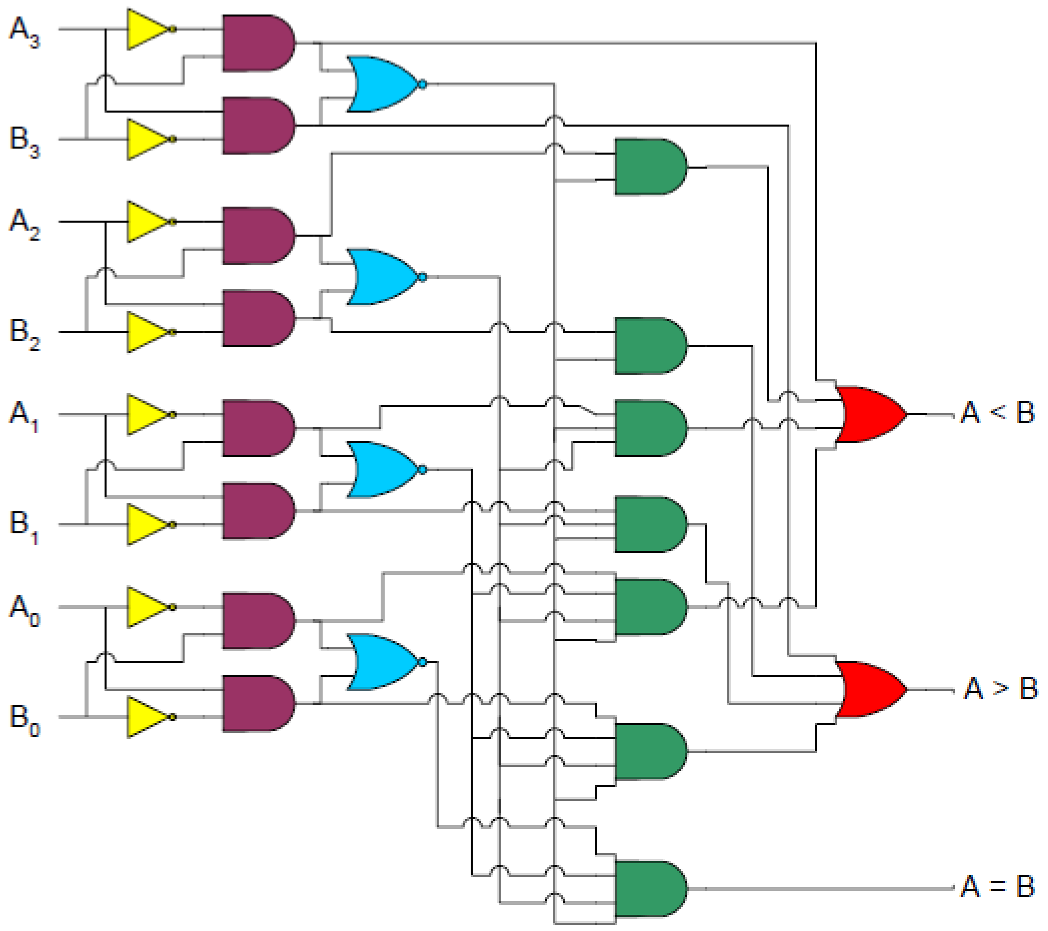


4-Bit Equality Comparator

Magnitude Comparator

- To determine if *A is greater than or less than B, we inspect the relative magnitudes of pairs* of significant digits starting from the most significant position.

- The sequential comparison can be expressed logically by the following two Boolean functions



GT = A3 B'3 + X3A2 B'2 + X3 X2A1 B'1 + X3X2X1A0B'0

LT = A'3 B3 + X3A'2 B2 + X3 X2A'1 B1 + X3X2X1A'0B0

Magnitude Comparator

Design of the GT output (A > B) 4-bit magnitude comparator

If A3 > B3, then A > B (GT=1) irrespective of the relative values of the other bits of A & B. Consider, for example, A = 1000 and B = 0111 where A > B.
This can be stated as GT=1 if **A3 B'3 =1**

If A3 = B3 (X3 = 1), we compare the next significant pair of bits (A2 & B2).
If A2 > B2 then A > B (GT=1) irrespective of the relative values of the other bits of A & B. Consider, for example, A = 0100 and B = 0011 where A > B. This can be stated as GT=1 if **X3A2 B'2 =1**

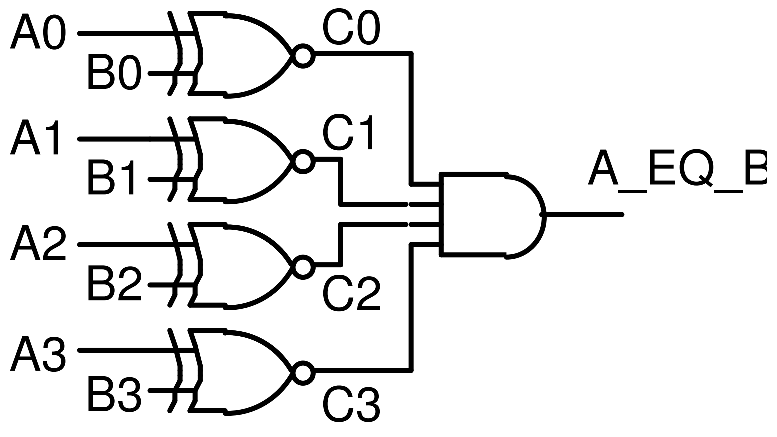
If A3 = B3 (X3 = 1) and A2 = B2 (X2 = 1), we compare the next significant pair of bits (A1& B1). If A1 > B1 then A > B (GT=1) irrespective of the relative values of the remaining bits A0 & B0. Consider, for example, A = 0010 and B = 0001 where A > B. This can be stated as GT=1 if **X3 X2A1 B1/ =1**

If A3 = B3 (X3 = 1) and A2 = B2 (X2 = 1) and A1 = B1 (X1 = 1), we compare the next pair of bits (A0 & B0). If A0 > B0 then A > B (GT=1). This can be stated as GT=1 if **X3X2X1A0B0/=1**

To summarize, GT =1 (A > B) IFF:
GT = A3 B'3 + X3A2 B'2 + X3 X2A1 B'1 + X3X2X1A0B'0

Magnitude Comparator

Find A_GT_B



If A = 1001 and
B = 0111
is A > B?
Why?

Because A3 > B3
i.e. A3 & !B3 = 1

Therefore, one term in the logic equation for A_GT_B is A3 & !B3

A_GT_B = A3 & !B3 + …..

If A = 1101 and B = 1011; is A > B? Why?

Because A3 = B3 and A2 > B2
i.e. C3 = 1 and A2 & !B2 = 1

Therefore, the next term in the logic equation for A_GT_B is C3 & A2 & !B2

A_GT_B = A3 & !B3 + C3 & A2 & !B2 +…..

If A = 1010 and B = 1001; is A > B? Why?

Because A3 = B3 and A2 = B2 and A1 > B1
i.e. C3 = 1 and C2 = 1 and A1 & !B1 = 1

A_GT_B = A3 & !B3 + C3 & A2 & !B2
+ C3 & C2 & A1 & !B1 + …..

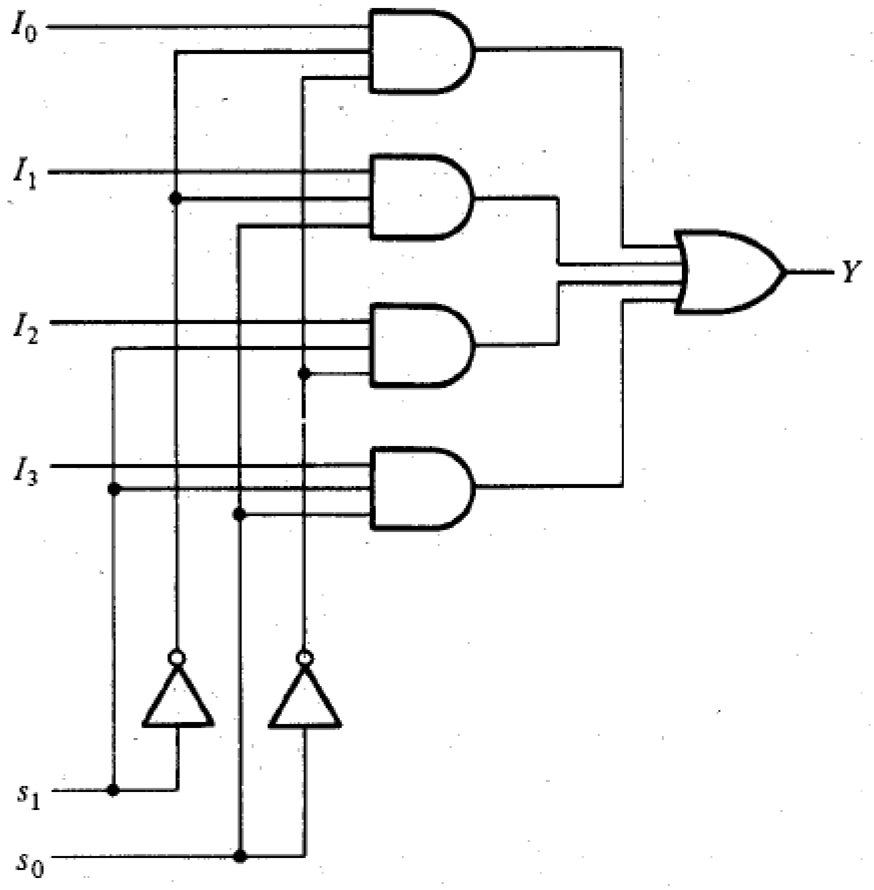
If A = 1011 and B = 1010; is A > B? Why?

Because A3 = B3 and A2 = B2 and A1 = B1 and A0 > B0 i.e. C3 = 1 and C2 = 1 and C1 = 1 and A0 & !B0 = 1

Multiplexers / Data Selectors

A *data selector or multiplexer is a logic circuit which* allows one of several data inputs to be selected and fed to a single output.

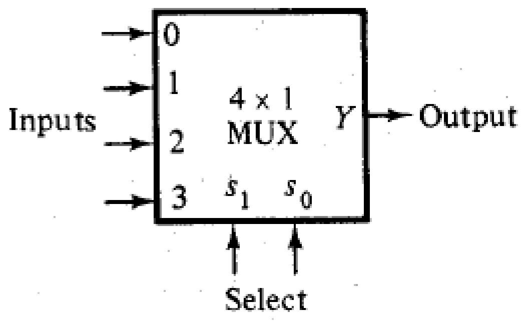
By applying control signals, any one of the input can be steer to the output.



(a) Logic diagram

s_1	s_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

(b) Function table



(c) Block diagram

Multiplexers / Data Selectors

A general procedure for implementing any Boolean function of *n variables with a 2^{n-1} to 1 multiplexer*

– Express the function in its sum of minterms
–List the inputs of the multiplexer and under them list all the minterms in two rows
– In the first row, *A is complemented*
– In the second row, *A is uncomplemented*
– Circle all the minterms of the function
» If the two minterms in a column are not circled, apply 0 to the corresponding multiplexer input
» If the two minterms are circled, apply 1 to the corresponding multiplexer input
» If the bottom minterm is circled and the top is not circled, apply *A to the corresponding* multiplexer input
» If the top minterm is circled and the bottom is not circled, apply *A' to the corresponding* multiplexer input

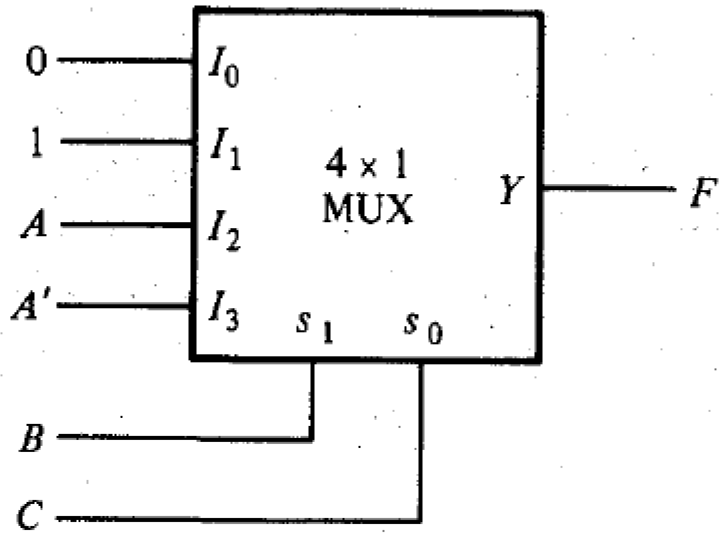
For a n:1 multiplexer, number of control signals m is given by: $n = 2^m$

Multiplexers / Data Selectors

Implement the function of three variables: *F(A,B,C) =Σ(1, 3, 5, 6)* with a 4-to-1 multiplexer

Minterm	A	B	C	F
0	0	0	0	0
1	0	0	1	1
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
7	1	1	1	0

(b) Truth table

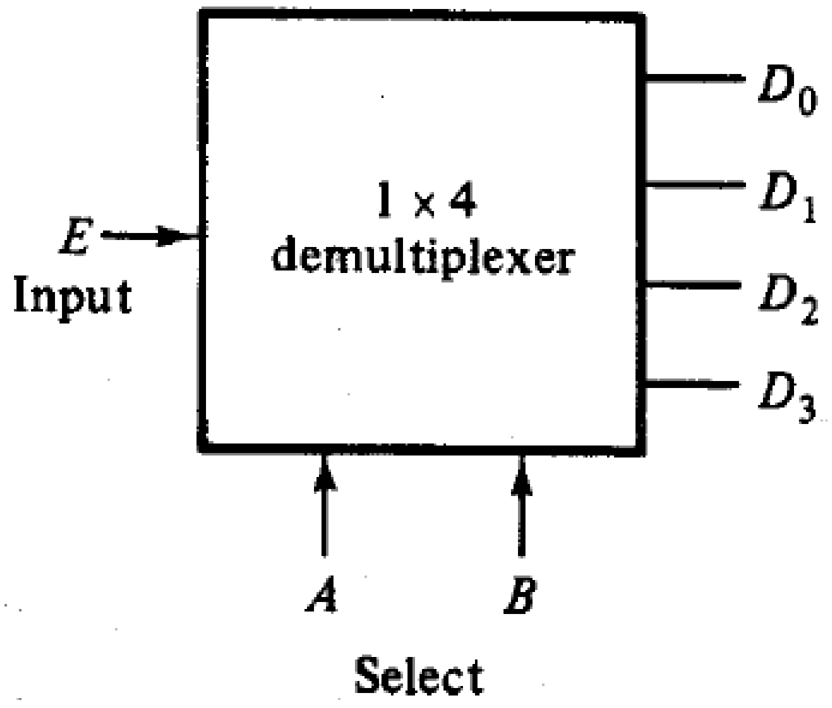


(a) Multiplexer implementation

	I_0	I_1	I_2	I_3
A'	0	①	2	③
A	4	⑤	⑥	7
	0	1	A	A'

Demultiplexer

A *demultiplexer is a circuit that receives information* on a single line and transmits this information on one of 2^n *possible output lines*



E
Input
1 × 4
demultiplexer
D_0
D_1
D_2
D_3
A
B
Select