LOGIC DESIGN

Digital Fundamentals

Introduction

- *This lecture shall introduce basic digital fundaments as building blocks of bigger digital systems such as digital computer.*
- *We will continue with VHDL after we have covered lecture 4.*

Logic Gates

- *The basic building blocks of a digital computer or any other digital electronic system are:*
  - ***NOT** gate*
  - ***OR** gate*
  - ***AND** gate*
- *Other gates are derived from the three given above:*
- ***NOR** gate, **NAND** gate, exclusive **OR** gate, exclusive **NOR** gate*

Description of the Gates

- Each gate has a function that can be expressed as a symbol or in *Boolean algebra, or truth table.*
- This enables design or combinations of several gates to be expressed in simple terms or in an optimised manner.

Logic Gate representations

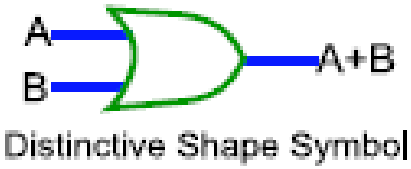
- Logic Symbols
- Algebraic expression
- Truth table
- Karnaugh maps

OR Function

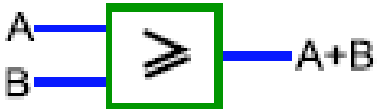
$A \quad OR \quad B = A + B$

The result is a 0 if A=0 OR B=0

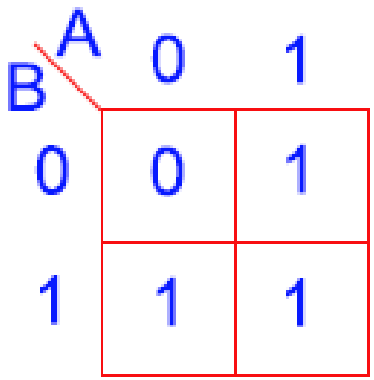
A	B	$A+B$
0	0	0
0	1	1
1	0	1
1	1	1



Distinctive Shape Symbol



Rectangular Outline



AND Function

A **AND** B = A*B

Truth Table

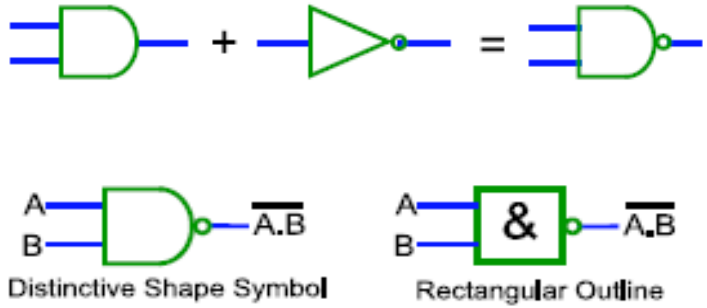
A	B	A*B
0	0	0
0	1	0
1	0	0
1	1	1

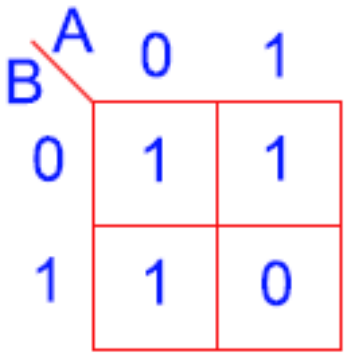
NAND function

$A \quad NAND \quad B = \overline{A.B}$

Output is HIGH if any of the input variables are LOW

A	B	$A.B$	$\overline{A.B}$
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0





NOR function

$A \quad NOR \quad B = \overline{A + B}$

Output is HIGH if ALL the inputs are LOW

A	B	$A+B$	$\overline{A+B}$
0	0	0	1
0	1	1	0
1	0	1	0
1	1	1	0

B \ A	0	1
0	1	0
1	0	0

XOR function

$$X = A \oplus B = A\overline{B} + \overline{A}B$$

Output is HIGH if inputs are different

A	B	X
0	0	0
0	1	1
1	0	1
1	1	0

De Morgan's Theorem

- *The complement of a sum is equal to the product of the complements*
- The complement of two or more variables ORed is the same as the complements of each individual variable ANDed.

$$\overline{A+B} = \overline{A}.\overline{B}$$

change operation from OR to AND, invert each operand individually, invert expression

De Morgan's Theorem

- *The complement of a product is equal to the sum of the complements*
- The complement of two or more variables ANDed is the same as the complements of each individual variable ORed.

$$\overline{A.B} = \overline{A} + \overline{B}$$

change operation from AND to OR, invert each operand individually, invert expression

Exercise1

use De Morgan's theorem to simplify the following expression

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

Exercise2

use De Morgan's theorem to simplify the following expression

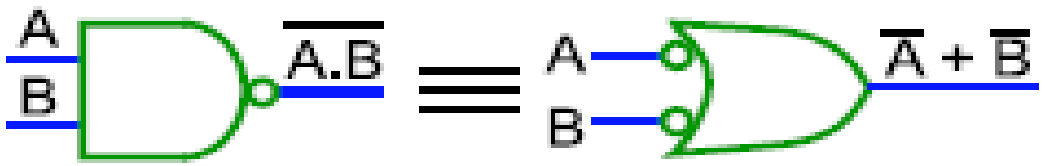
$$X = \overline{\overline{(\overline{A} + B)} + \overline{CD}}$$

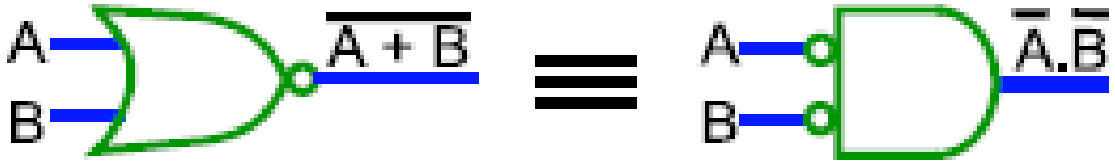
Exercise3

use De Morgan's theorem to simplify the following expression

$$X = \overline{\overline{A + B\overline{C}} + \overline{D(E + \overline{F})}}$$

Schematic Representation of De Morgan's Theorem



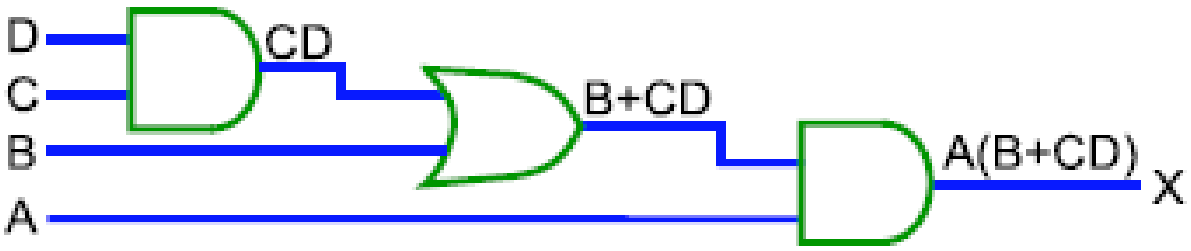


Boolean Expressions and Logic Gate Networks

- The form of a boolean expression may suggest a particular implementation using logic gates.

$$X = A(B + CD)$$

might be interpreted as indicating that C and D are ANDed to give CD, CD is ORed with B to give (B+CD), and (B+CD) is ANDed with A to give X = A(B+CD).

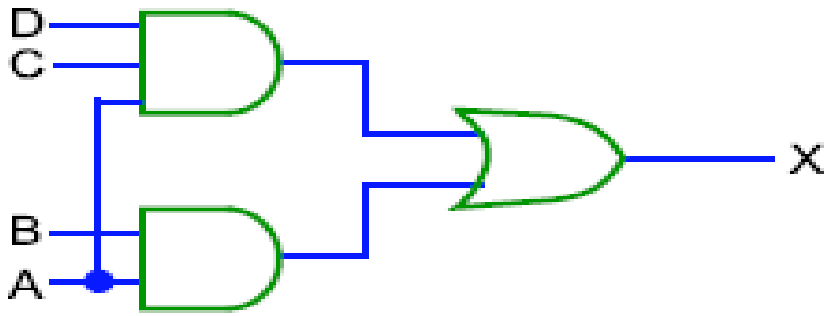


Boolean Expressions and Logic Gate Networks

Alternatively,

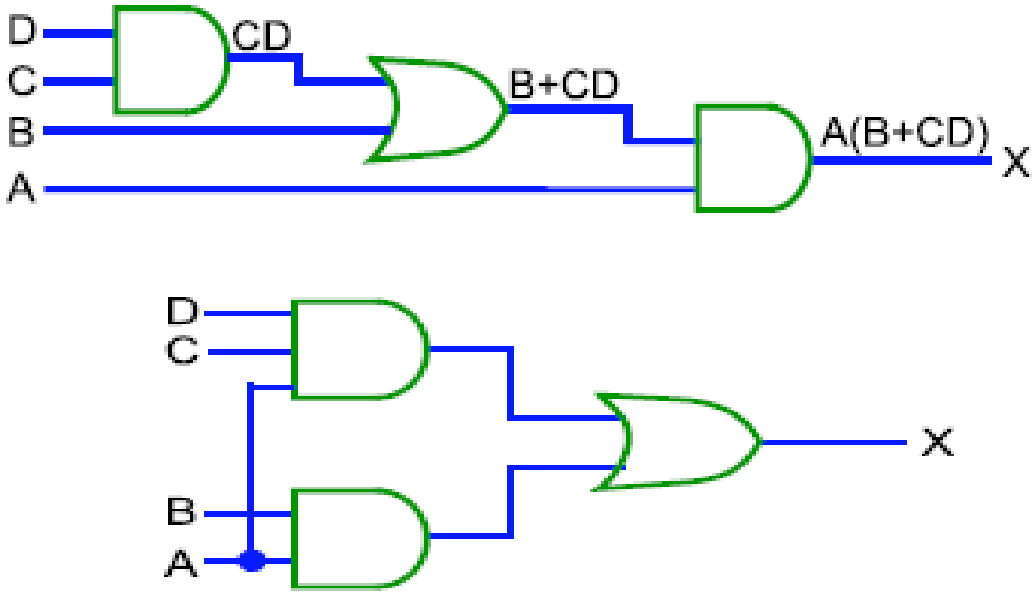
$$X = A(B + CD)$$
$$= AB + ACD$$

which might be interpreted as indicating the following circuit



Boolean Expressions and Logic Gate Networks

the two circuits are equivalent and one or other might be considered to have advantages, depending on desired performance and on component availability.



Common forms of Implementation

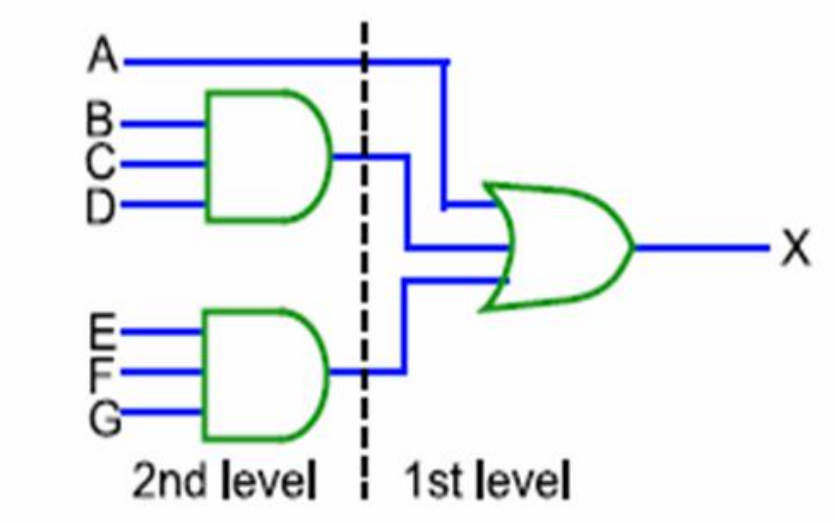
1. Sum of Products

$$A.B + C.D$$
$$A.\overline{B}.C + D.\overline{E}.F + A.E.G$$
$$A + B.C.D + E.F.G$$

A sum of products expression consists of a number of ANDed functions ORed together. Any logic expression can be converted into a sum of products form using the rules of boolean algebra.

Common forms of Implementation

A sum of products expression can be implemented as a two stage gate network. The maximum number of gates a signal must pass through from input to output is two.



Common forms of Implementation

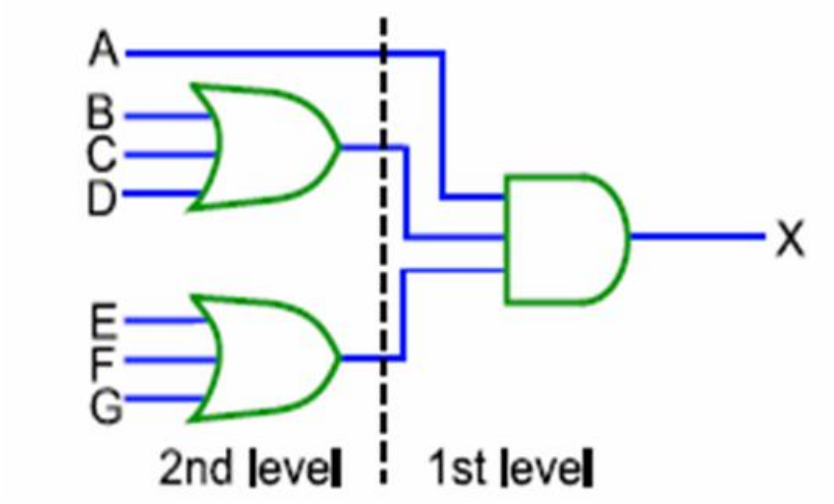
2. Products of Sums

$$(A + B) \,.\, (B + C) \,.\, D$$
$$A \,.\, (B + C + D) \,.\, (E + F + G)$$

A product of sums expression consists of a number of ORed functions ANDed together. Any logic expression can be converted into a product of sums form using the rules of boolean algebra.

Common forms of Implementation

A product of sums expression can be implemented as a two stage gate network. The maximum number of gates a signal must pass through from input to output is two.



Exercise4

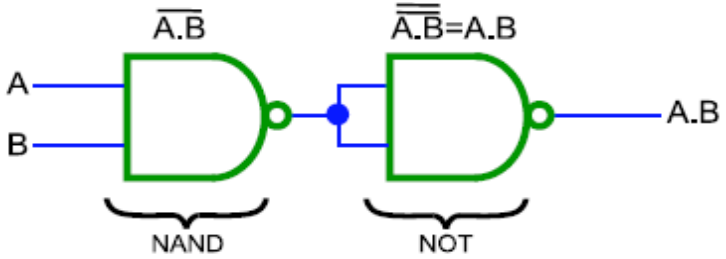
simplify

$$X = (A\overline{B}(C + BD) + \overline{A}\,\overline{B})C$$

Universal NAND Gate

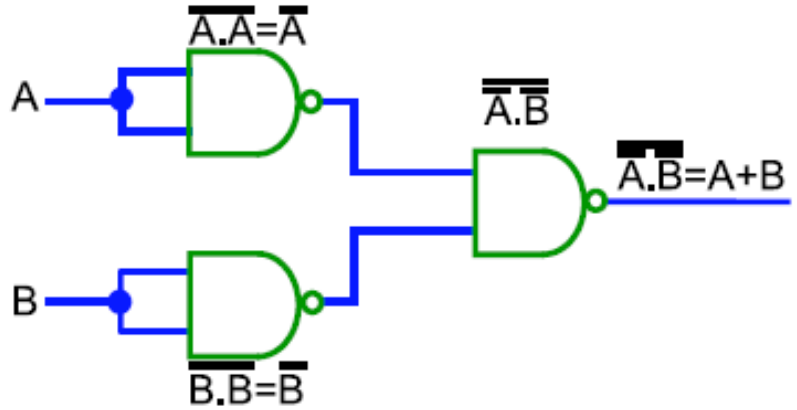
- Implement only with NAND gates. Any circuit can be made using only two-input NAND Gates (universal NAND)
- NOT Gate

- AND Gate

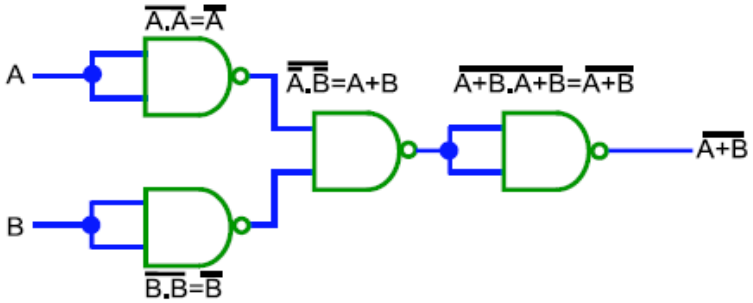


Universal NAND Gate

- OR Gate



- NOR Gate



Exercise5

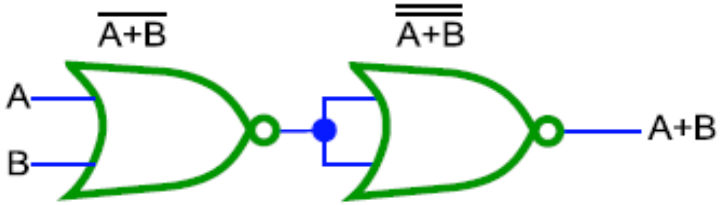
- Implement the expression given below using NAND gates

$$X = A.B + \overline{A}.C$$

Universal NOR Gate

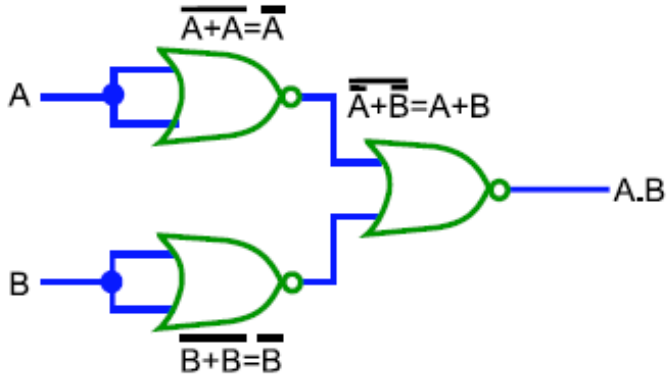
- Implement only with NOR gates. Any circuit can be made using only two-input NOR Gates (universal NOR)
- NOT Gate

- OR Gate

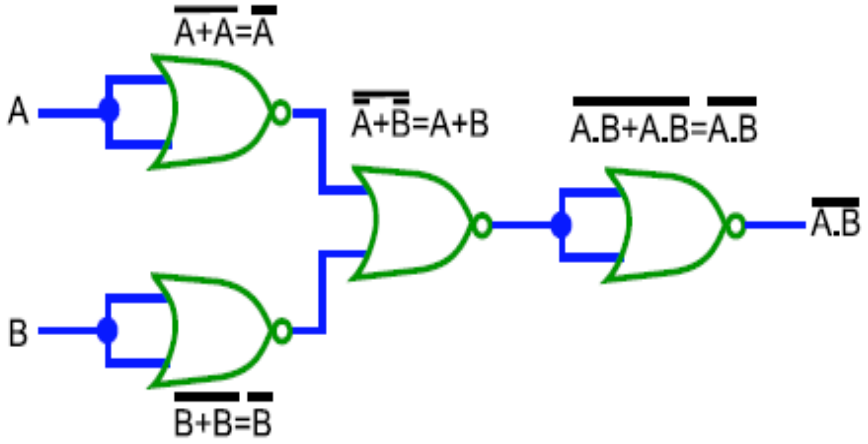


Universal NOR Gate

- AND Gate



- NAND Gate



Karnaugh Maps

- boolean algebra can be quite tricky to manipulate
- difficult to tell whether simplest expression has been found
- Karnaugh maps are a powerful and widely used graphical method of logic simplification

Karnaugh Maps

- can be used for expressions with up to six input variables
- Each cell in a Karnaugh map corresponds to one line in a truth table

The main stages to creating logic using a Karnaugh map are

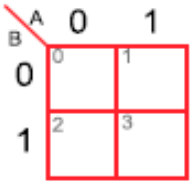
- draw the empty Karnaugh map (number of inputs define how the map should be drawn)
- fill in the desired outputs (1's and 0's) corresponding to each input. (from logic expression or truth table)
- form groups of adjacent cells containing 1's.
- find expressions for each group to give the entire logic expression

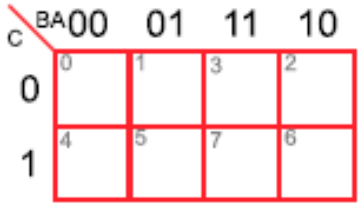
Creating karnaugh map

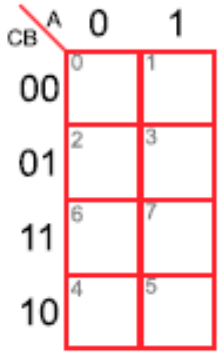
- For two variables A and B there are four possible States
  - The small numbers in the top left corner of each cell shows the Gray code for that cell
  - Each cell corresponds to one state from the function's truth table. Each of the cells corresponds to one minterm

  (is the Anding of one combination of all of the inputs of a function)

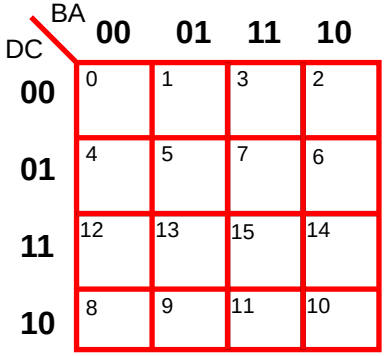
B	A	Output	Gray Code
0	0		$00_{Gray}=0_{10}$
0	1		$01_{Gray}=1_{10}$
1	0		$10_{Gray}=2_{10}$
1	1		$11_{Gray}=3_{10}$





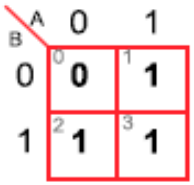


- For three Variables A, B and C there are eight possible states
- Four variables have sixteen cells
- Five and six variables, three dimensional maps are used

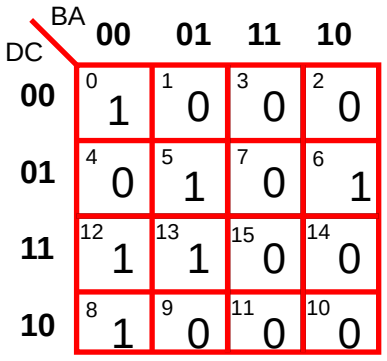


Examples – 1

- Fill out the karnaugh map given *G* = 1,2,3 with input variables *BA*.



- Fill out the karnaugh map given *F* = 0,5,6,8,12,13 with input variables *DCBA*



Exercise6

- Given the following logic expression create and fill out karnaugh map and give the simplified expression of G.

$$G = A\overline{B}\,\overline{C}\,\overline{D} + A\,B\,\overline{C}\,\overline{D} + \overline{A}\,\overline{B}\,C\,\overline{D} + A\,\overline{B}\,C\,\overline{D} + A\,B\,C\,\overline{D} + \overline{A}\,B\,C\,\overline{D} + \overline{A}\,B\,C\,D + \overline{A}\,\overline{B}\,\overline{C}\,D + \overline{A}\,B\,\overline{C}\,D$$

Exercise7

- Given the following truth table for the function G, fill in the cells of the karnaugh map and give the simplified expression of G.

A	B	C	D	G
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	1
1	1	1	1	0

Grouping Adjacent Cells in Karnaugh Maps

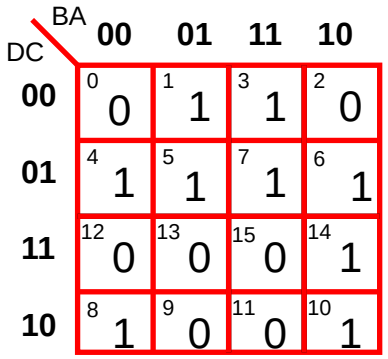
- Groups must contain 2^n adjacent cells, each containing a 1.
  - The groups should be made as large as possible, while remaining integer powers of two.
  - The groups have to be square or rectangular.
  - The larger the groups chosen, the simpler the function, as with each increase of power of 2 in the size of the group, the group becomes dependent on one less variable. So the minimal form is obtained with the largest possible groups that are integer power of 2.
- All cells containing a 1 must be included in at least one group.
- The groups may overlap, so one cell may be included in several groups, but any group that has all its elements included in other groups is not required.
- There may be several correct minimal forms for a given logic function, dependent upon the particular groupings that are chosen.
- The edges of a Karnaugh map are adjacent to each other because of the Gray code used for labelling the cells. This means that groups can join from the left side to the right side and from the top to the bottom.

Grouping Adjacent Cells in Karnaugh Maps

- Groups must contain adjacent cells each containing a 1.
- All cells containing a one must be included in at least one group
- The groups may overlap
- The edges of karnaugh maps are adjacent to each other because of the gray code used for labelling the cells

Exercise8

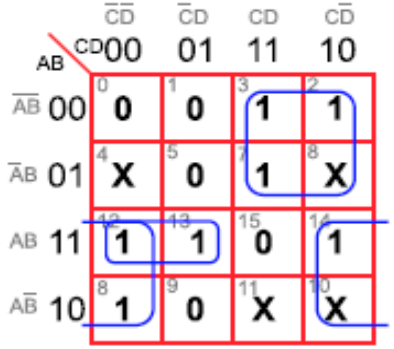
- Illustrate a possible set of groups, then calculate the boolean logic expression for each loop.



- Karnaugh Map

Don’t care conditions

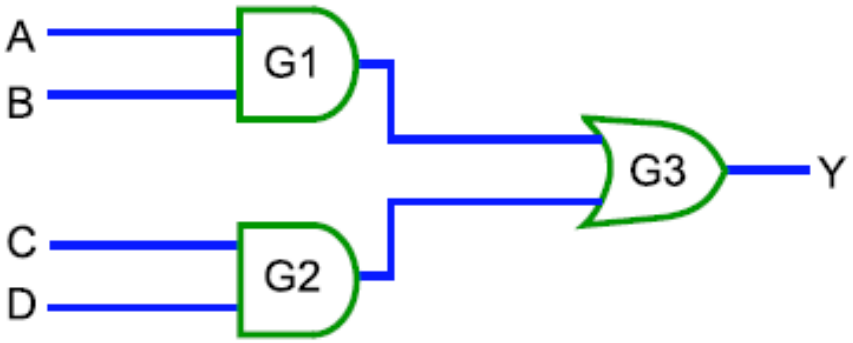
- When particular combination of inputs cannot occur. In 8421 BCD – 1100 cannot occur. They help simplify the minimal Boolean expression derived from the map.



A	B	C	D	P
0	0	0	0	0
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	X
0	1	0	1	0
0	1	1	0	X
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	X
1	0	1	1	X
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

Combinational Logic Circuits

- The output is at all times dependent on the combination of input levels.
- Analysis: the determination of the output of the circuit for all possible combinations of input values. Logic gates can be analysed considering the output of each gate.



- Output of Gate G1 = AB
- Output of Gate G2 = CD
- Output of Gate G3 = $AB + CD$

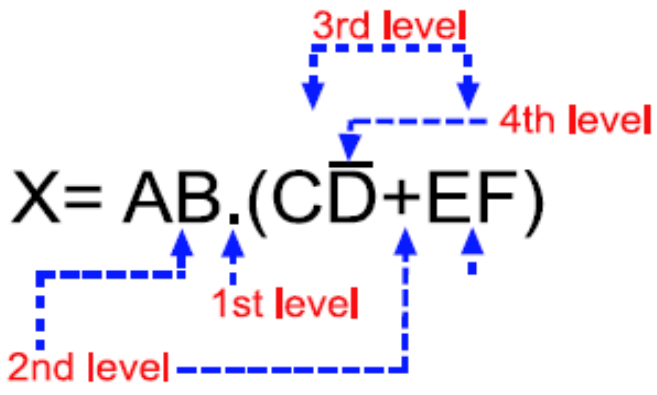
INPUTS				G1 Output	G2 Output	G3 Output
A	B	C	D	AB	CD	G1 + G2 $(AB + CD)$
0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	0	1	0	0	0	0
0	0	1	1	0	1	1
0	1	0	0	0	0	0
0	1	0	1	0	0	0
0	1	1	0	0	0	0
0	1	1	1	0	1	1
1	0	0	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	0	0	0
1	0	1	1	0	1	1
1	1	0	0	1	0	1
1	1	0	1	1	0	1
1	1	1	0	1	0	1
1	1	1	1	1	1	1

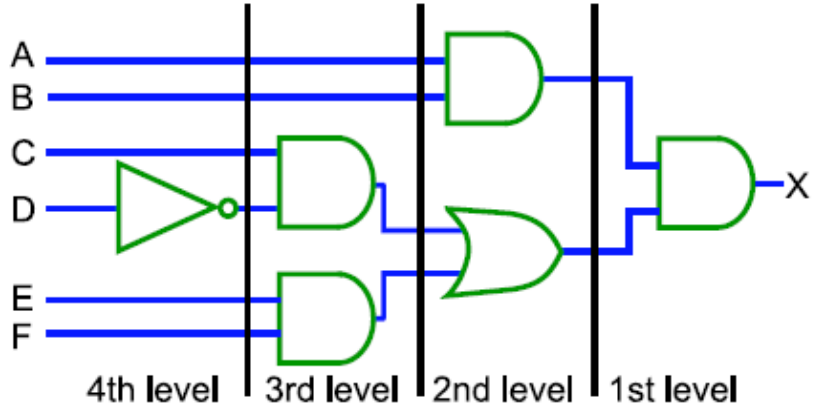
Example of Implementing Combinational logic circuits

- Consider the following function

$$X = AB.(CD + \overline{EF})$$

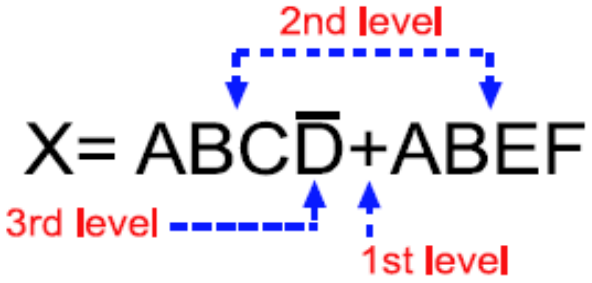
Assign the levels to this function and implement

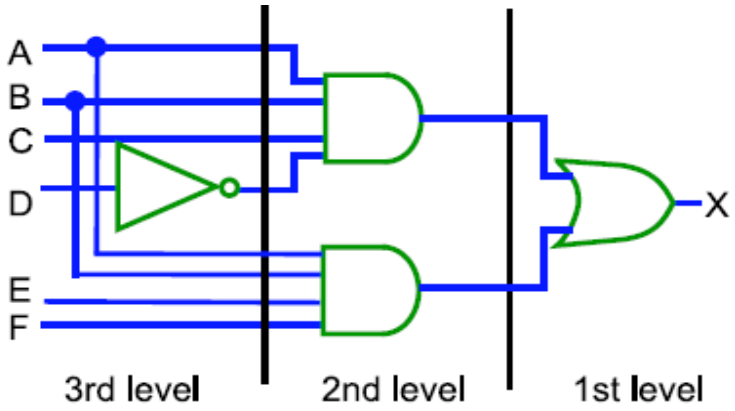




Example of Implementing Combinational logic circuits

- Let us multiply out the brackets, assign the levels to this function and implement.





Combinational Logics

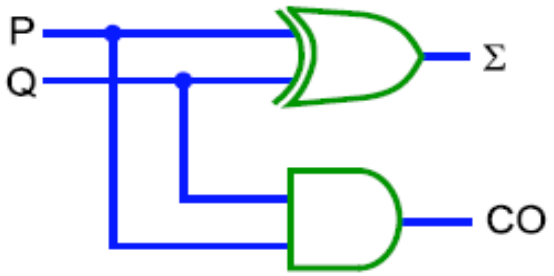
Some building blocks in more complex systems

- Adders
- Magnitude comparators
- Decoders
- Encoders
- Multiplexors

Half Adder

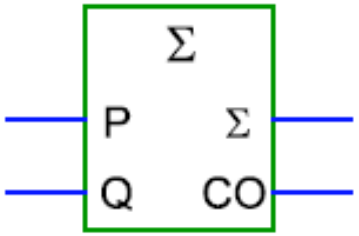
Rule 1	0 + 0 = 0
Rule 2	0 + 1 = 1
Rule 3	1 + 0 = 1
Rule 4	$1 + 1 = 10_2$*

INPUTS		OUTPUTS	
P	Q	CO	$\sum$
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0



$$CO = P.Q$$

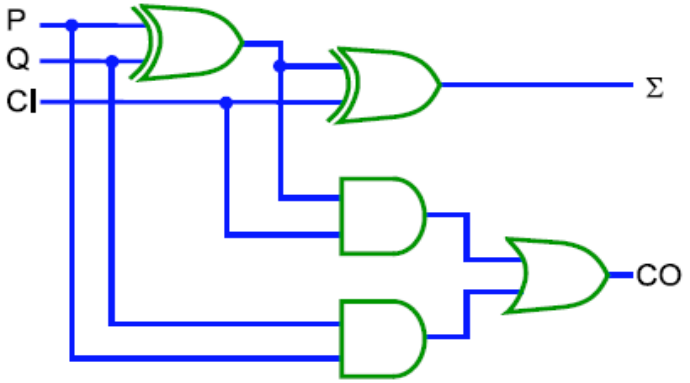
$$\sum = P \oplus Q = P.\overline{Q} + \overline{P}.Q$$



half adder because it doesn't have a carry in, allowing us to carry in from lower bits down in binary addition.

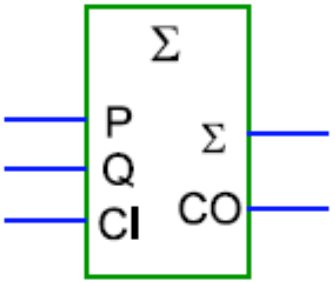
Full Adder

INPUTS			OUTPUTS	
P	Q	CI	CO	$\sum$
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



$$\sum = (P \oplus Q) \oplus CI$$

$$CO = P \,.\, Q + (P \oplus Q) \,.\, CI$$



The full adder can also be constructed using only NOT, AND and OR gates.

Full Adder – another implementation

PQ \ CI	0	1
00		1
01	1	
11		1
10	1	

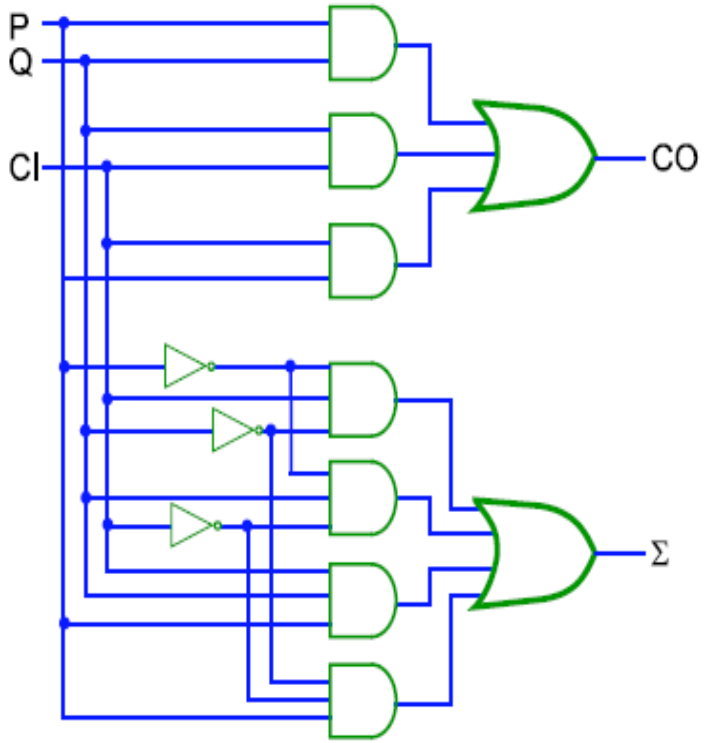
Σ

PQ \ CI	0	1
00		1
01		1
11	1	1
10		1

CO

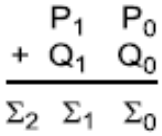
$$\sum = \overline{P}.\overline{Q}.CI + \overline{P}.Q.\overline{CI} + P.Q.CI + P.\overline{Q}.\overline{CI}$$

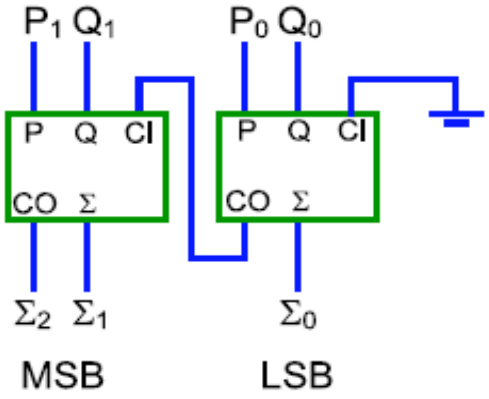
$$CO = P.Q + Q.CI + P.CI$$



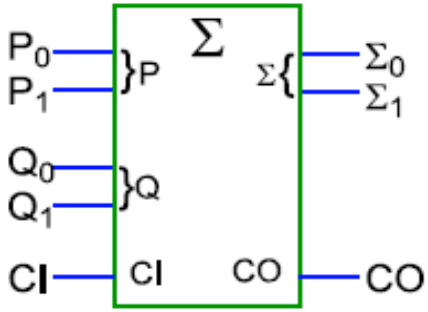
A single full adder can only add two 1-bit numbers and an input carry. To add binary numbers with more than one bit requires more adders

Parallel Binary Adder

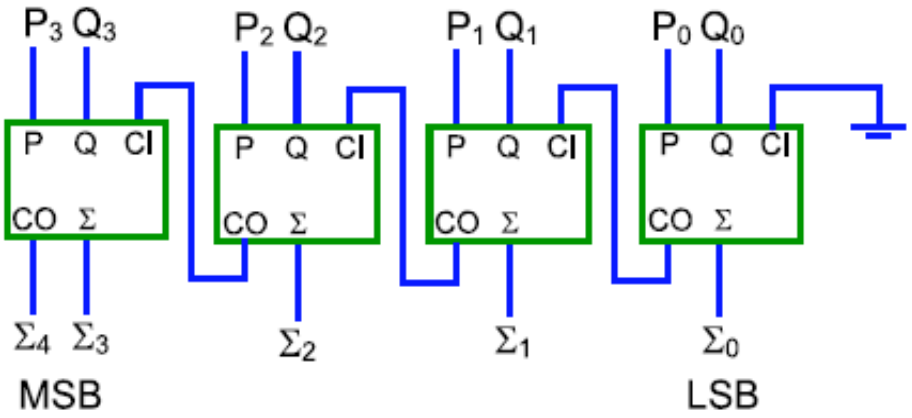




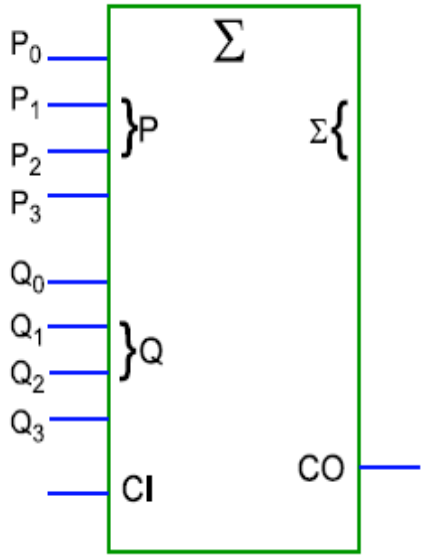
The output carry is effectively the MSB (Most Significant Bit) of the sum.



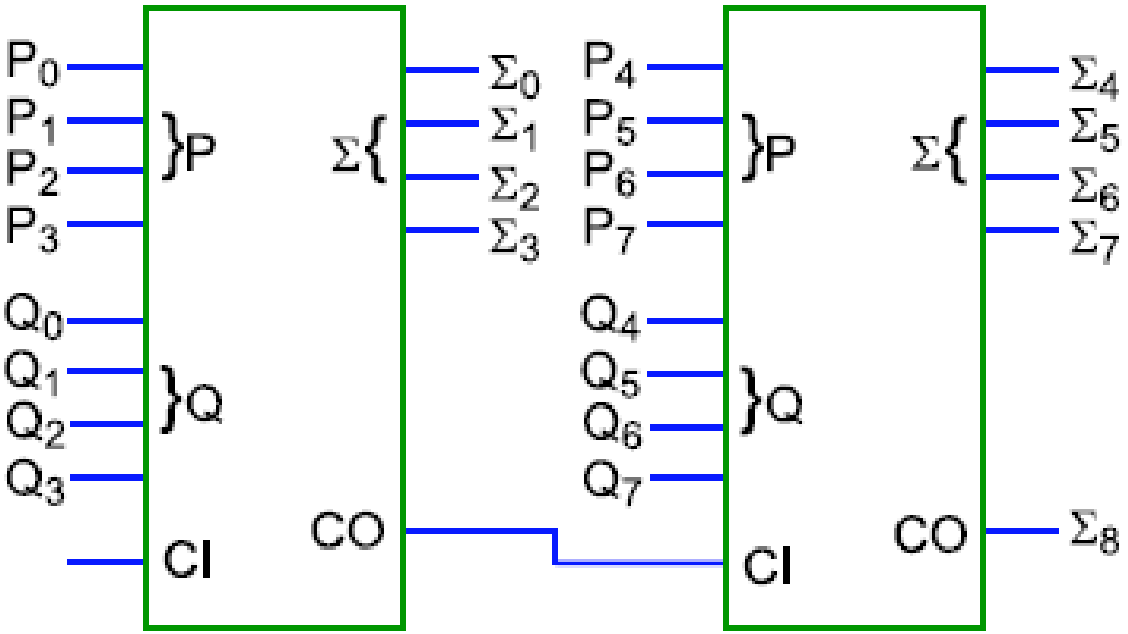
4-bit Parallel Adder



$$\begin{array}{r} P_3\ P_2\ P_1\ P_0 \\ +\ Q_3\ Q_2\ Q_1\ Q_0 \\ \hline \Sigma_4\ \Sigma_3\ \Sigma_2\ \Sigma_1\ \Sigma_0 \end{array}$$



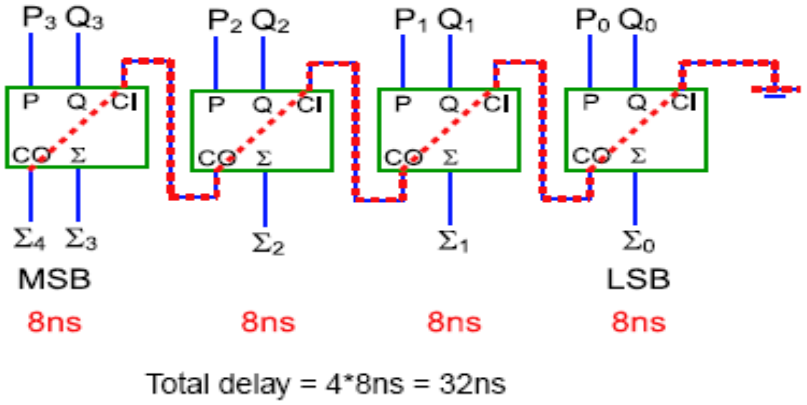
8-bit Parallel Adder



Ripple carry adder

Because the carry output from each full adder stage is connected to the carry input of the next stage. This means that the sum and carry of a stage can't be generated until the carry input is ready.

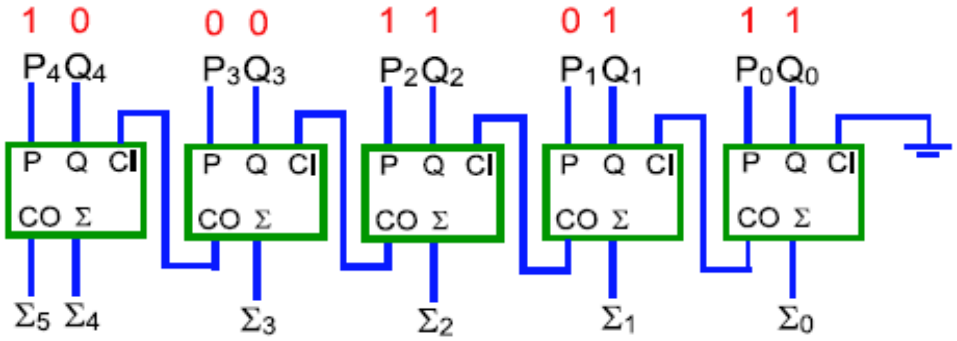
- results in a time delay in the addition
- slows down the operation of the circuit



Total delay = 4*8ns = 32ns

Example 1

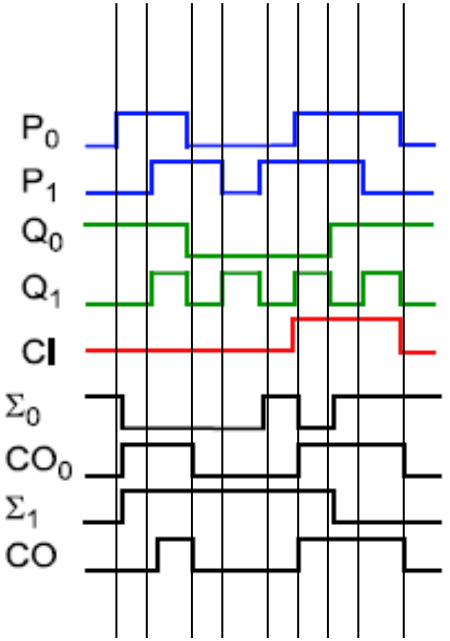
- Determine the sum by analysis of the logical operation of the circuit.



$$\begin{array}{r} 10101 \\ +00111 \\ \hline 11100 \end{array}$$

Example 2

- Determine the waveforms for the sum and carry outputs in relation to the inputs.



Example 3

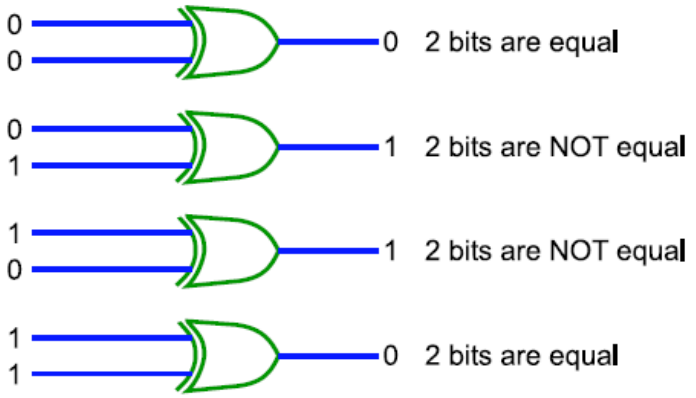
- Each of eight full adders in an 8-bit parallel ripple carry adder exhibits the following propagation delays.

P to $\sum$ and CO	40ns
Q to $\sum$ and CO	40ns
CI to $\sum$	35ns
CI to CO	25ns

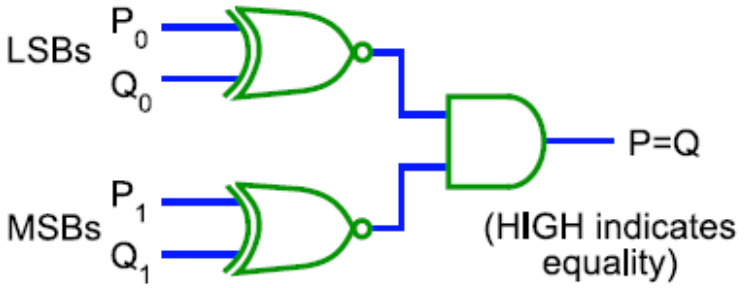
40+6*25+35

Magnitude Comparators

- Determine if 2 numbers are equal. The Exclusive OR (XOR) can be used as a basic comparator because its output is a 1if the 2 input bits are NOT equal, and the output is a 0 if the 2 input bits are equal

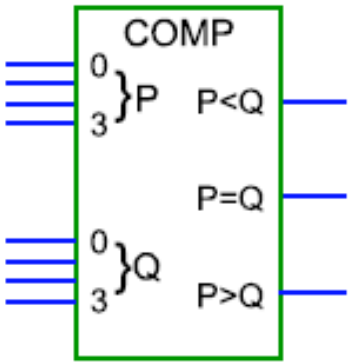


- To compare binary numbers with 2 bits in each number, then the least significant bit (LSB) and the most significant bit (MSB) must be compared.



Logic process of a multi-bit comparator.

- Integrated circuits have additional outputs to indicate if the number is larger or smaller or equal.
- To test if P > Q or if P < Q further circuitry is required



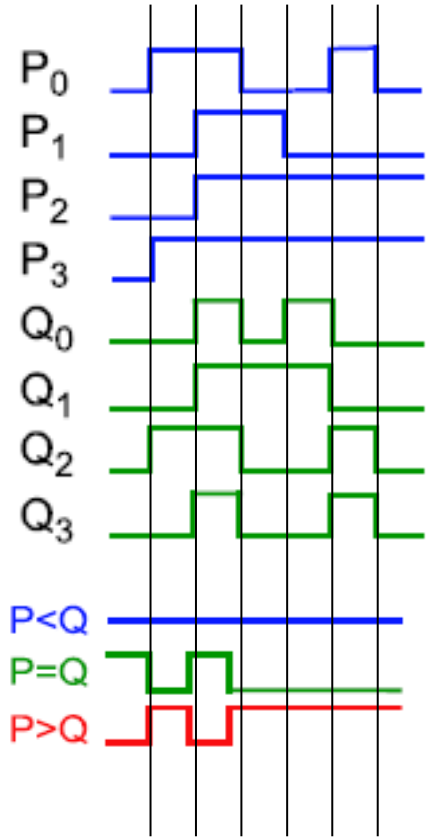
1. Check for inequality in each bit position starting with most significant bit
2. When inequality is found relationship is established.

Ex: P (P3P2P1P0) and Q (Q3Q2Q1Q0).

1. If P3 = 1 and Q3 = 0 then P > Q
2. If P3 = 0 and Q3 = 1 then P < Q
3. If P3 = Q3 = 1 then we need to look at next significant bit and this is repeated.

Example 4

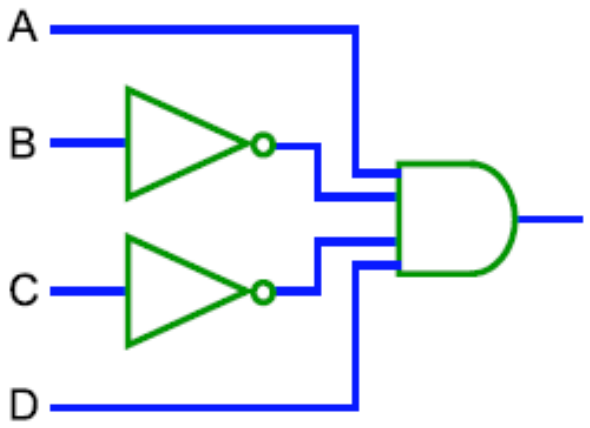
Plot each output waveform for the inputs shown in Figure 4.23. The outputs are active HIGH.



Decoders

- A decoder has *n* input lines and from 1 to 2^n output lines to indicate the presence of 1 or more *n* bit combinations.

This circuit will only produce a HIGH output when the code 1001 is present at its inputs.



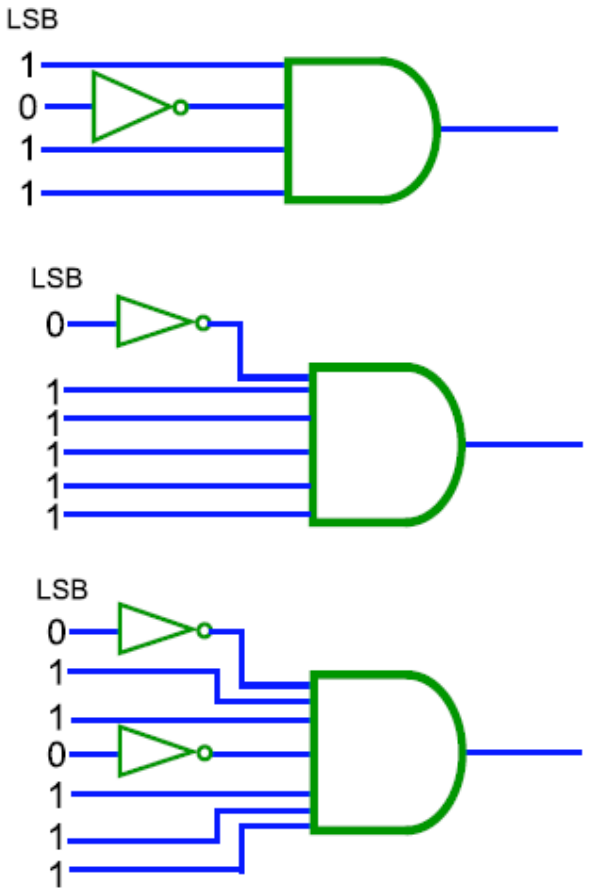
Example 5

- Show the decoding logic for each of the following codes

1101

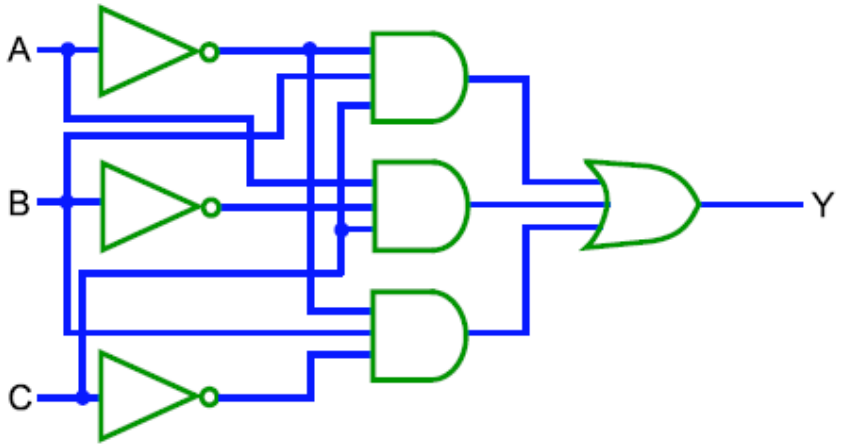
111110

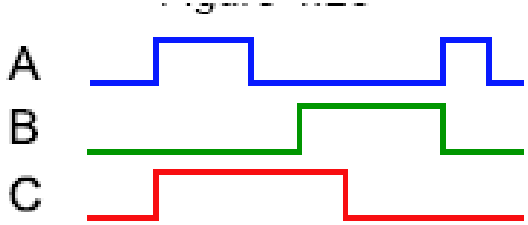
1110110



Example 6

- Sketch the output waveform in relation to the inputs.



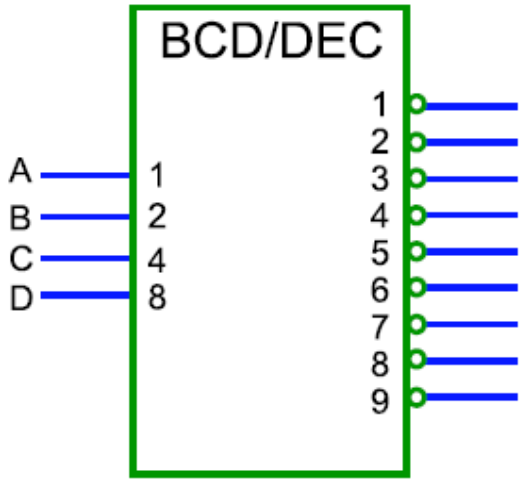


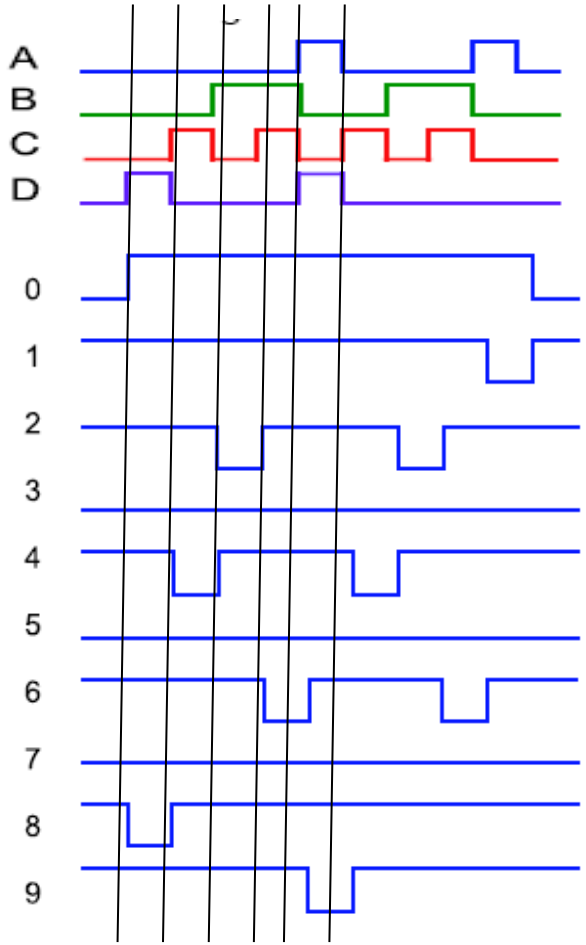
Binary Coded Decimal - BCD

Decimal	8421 BCD	5211 BCD	Aitken 2421 BCD
0	0000	0000	0000
1	0001	0001	0001
2	0010	0011	0010
3	0011	0101	0011
4	0100	0111	0100
5	0101	1000	0101
6	0110	1001	0110
7	0111	1011	0111
8	1000	1101	1110
9	1001	1111	1111

Convert the two decimal number 4710 and 9210 into BCD code.

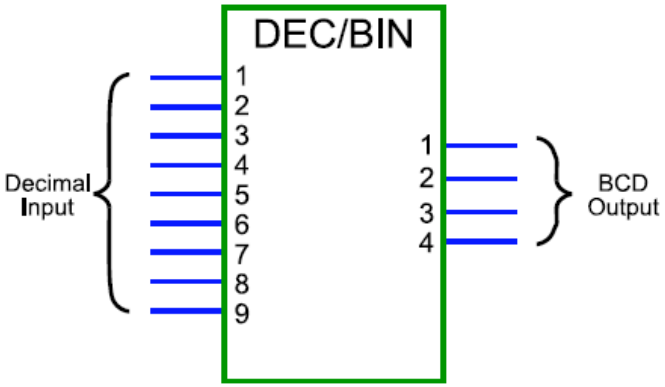
BCD/DEC



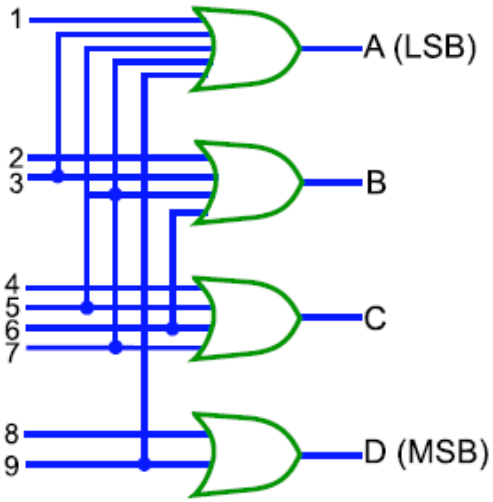


Encoders

- An encoder accepts an active level on one of its inputs representing a digit, for example a decimal number, and converts it to a coded output



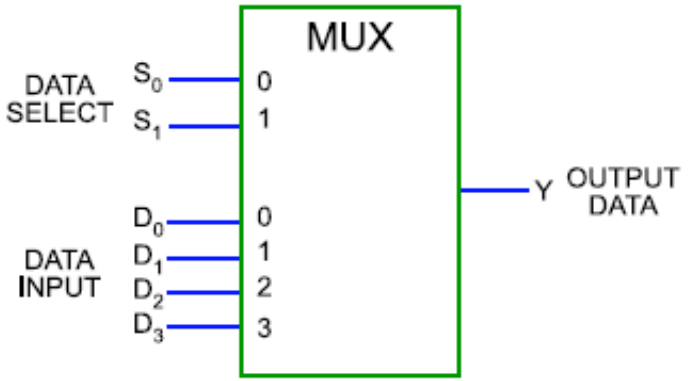
Input	BCD Code Output			
Decimal Digit	D	C	B	A
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1



Multiplexors (Data Selectors)

- A basic multiplexor has several input lines and only one output line. allows digital information from several sources to be routed along one line to a common destination

- For a four-input multiplexor we require 2 data select inputs in order to select an input line.

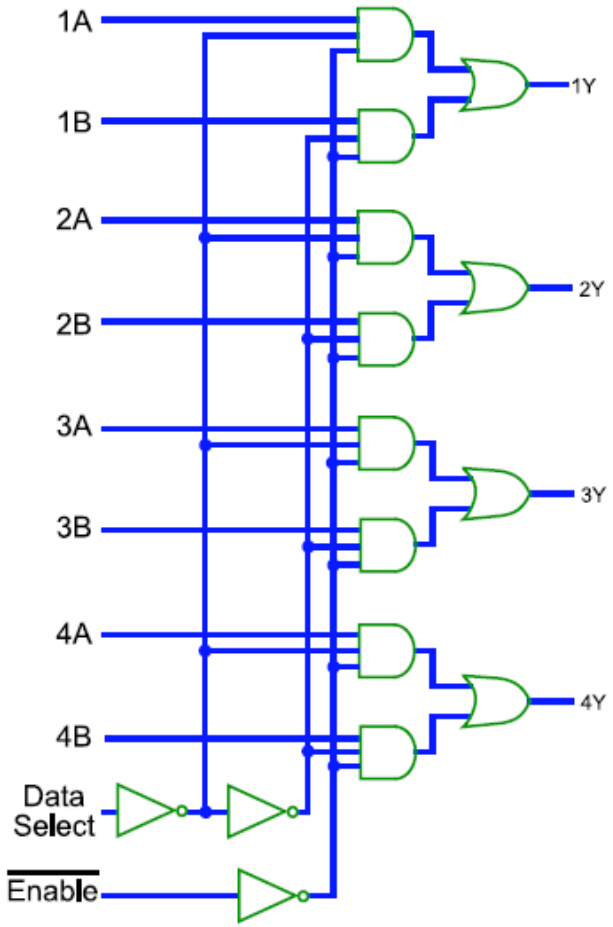


Data Select I/P		I/P connected to O/P
S_1	S_0	Y
0	0	D_0
0	1	D_1
1	0	D_2
1	1	D_3

$$Y = D_0 . \overline{S_1} . \overline{S_0} + D_1 . \overline{S_1} . S_0 + D_2 . S_1 . \overline{S_0} + D_3 . S_1 . S_0$$

The 74157 Quadruple 2-input MUX

It contains 4 separate 2-input multiplexors on a single chip.



The 74150 is a 16-input data MUX

- It has 16 data inputs and therefore requires 4 data select lines

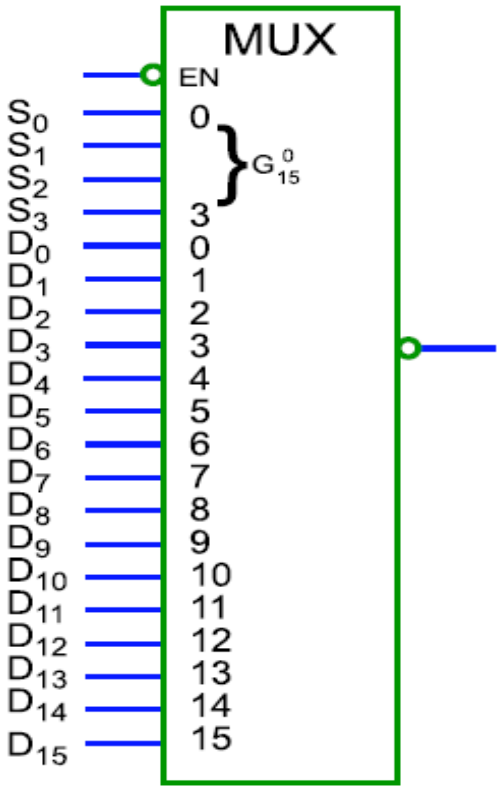
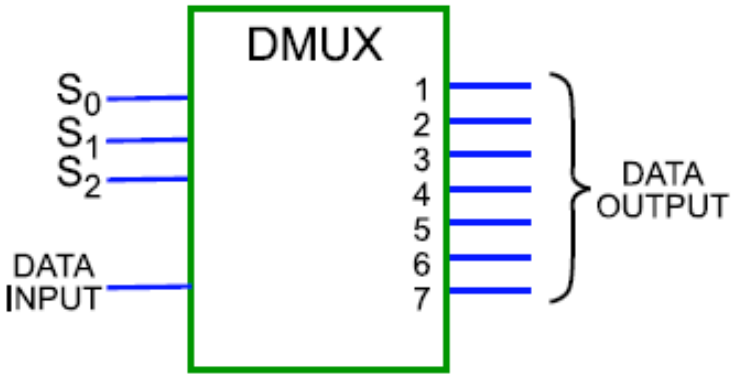


Figure 4.34: 16-Input Multiplexor

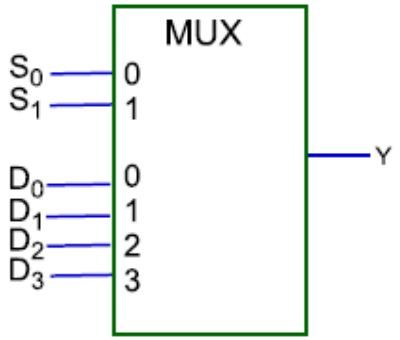
Demultiplexor

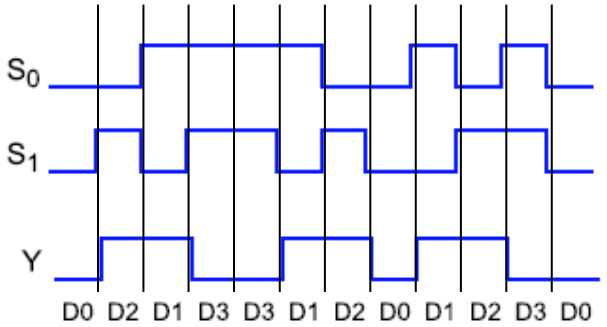
- It takes data from one line and distributes it over several lines. The selectors S0, S1 and S2 allow you to select which output line to transmit the input data on



Example 7

Determine the output waveform with data Inputs: D0 = 0, D1 = 1, D2 = 1 D3 = 0.





Example 8

- Determine the data output waveform.

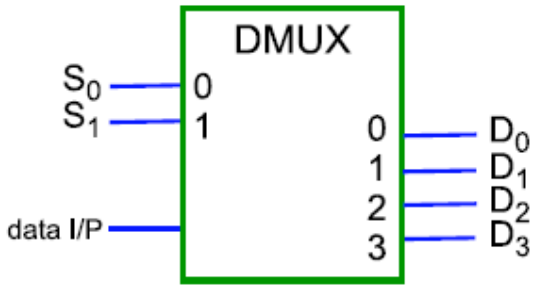


Figure 4.38

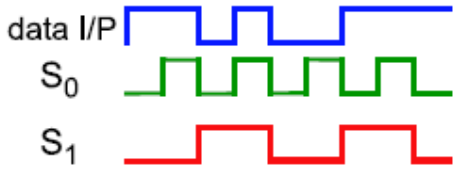


Figure 4.39

Assignment1

- Please do simple assignment1 consisting of exercises from exercise1 to exercise8 in the lecture notes and hand it in, next week on Monday in class.
- Regards

End