UCCD1133

Introduction to Computer Organisation and Architecture

Chapter 3

Basic Concept of Logic

Disclaimer

- This slide may contain copyrighted material of which has not been specifically authorized by the copyright owner. The use of copyrighted materials are solely for educational purpose. If you wish to use this copyrighted material for other purposes, you must first obtain permission from the original copyright owner.

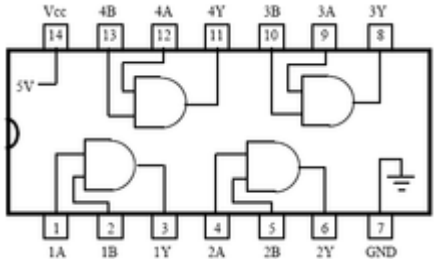
Chapter 3-3

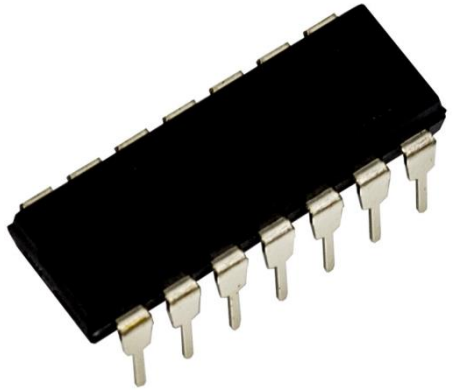
Basic Logic Gates

Outline

- Express the operation of
  - NOT,
  - OR,
  - AND,
  - NOR,
  - NAND,
  - XOR and
  - XNOR

  gates with Boolean algebra and truth table.
- Construct timing diagram showing the proper timing behaviour of inputs and outputs for various logic gates.
- Use logic gates in simple applications.

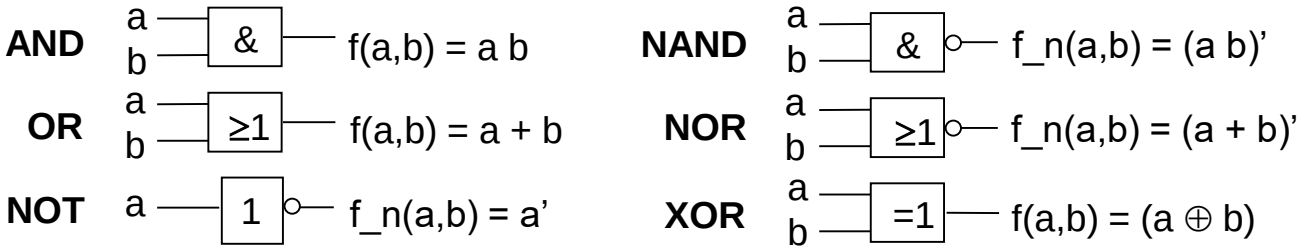




Logic Gates

- Logic gates are used to realize Boolean expression.
- They are the fundamental building blocks of all digital circuits.
- Three basic logic gates: AND, OR, NOT gates.
- Other common logic gates: NAND, NOR, XOR, XNOR gates.

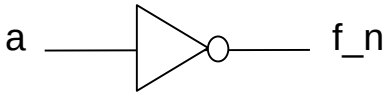
(a) Common logic gate symbols



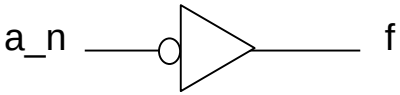
(b) Symbols defined in IEEE/ANSI Standard 91-84

NOT Gate (Inverter)

- Logic symbol of NOT gate:



or



Active-high input, active-low output

a	f_n = a'
0	1
1	0

Truth table

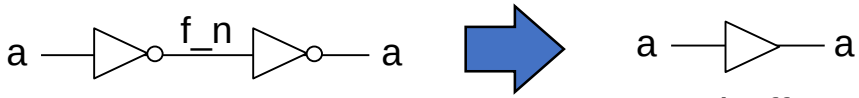
Active-low input, active-high output

a_n	f = a_n'
0	1
1	0

Truth table

- Logic equation
  - f_n = a'
- Logic equation
  - f = a_n'

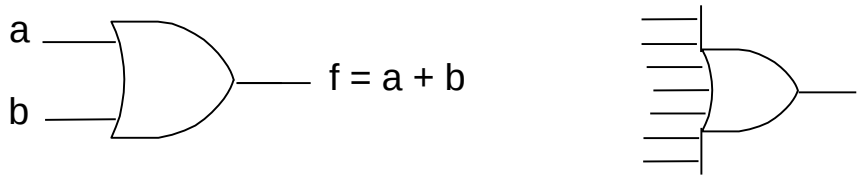
- A NOT operation on a logic variable *A* is denoted as $\bar{A}$ or *A*' (NOT A).
- The bubble indicates an active-low
- An absence of a bubble indicates otherwise
- Functionality
  - Invert the signal level – implementing NOT function.
  - To create a buffer



A buffer

OR Gate

- Logic symbol of OR gate:



2-input OR gate

Multiple input OR gate

a	b	f = a + b
0	0	0
0	1	1
1	0	1
1	1	1

2-input OR truth table

Deriving logic function from the truth table,

f = a' b + a b' + a b

= a' b + a (b' + b)

= a + a' b

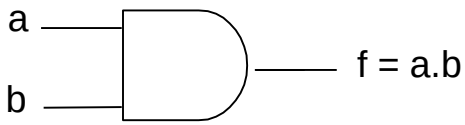
= a + b

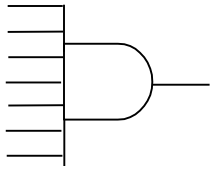
- Logic equation
  - f = a + b

- The OR function is represented by a plus sign + .
- From the truth table, OR function can be described as:
  - If any one of the input (a **or** b **or** c **or** d **or** …) is asserted, then output will be asserted. If all inputs are de-asserted, then output will be de-asserted.
  - For a 2-input OR gate, output f is HIGH when either input a or b is HIGH, or when a and b are HIGH.

AND Gate

- Logic symbol of AND gate:





2-input AND gate Multiple input AND gate

a	b	f = a b
0	0	0
0	1	0
1	0	0
1	1	1

Deriving logic function from truth table,
f = a b

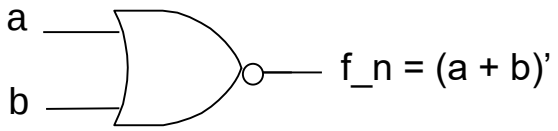
2-input AND truth table

- Logic equation
  - f = a b

- The AND function of a and b is represented by a dot as a•b, or simply ab.
- From truth table, AND function can be described as
  - If all inputs (a **and** b **and** c **and** d **and** …) are asserted, then output will be asserted; if any one input is de-asserted, then output will be de-asserted.
  - Output f is HIGH only when inputs a and b are HIGH.

NOR Gate

- Logic symbol of NOR gate:



2-input NOR gate

a	b	f_n = (a+b)'
0	0	1
0	1	0
1	0	0
1	1	0

Deriving logic function from the truth table,

f_n = a' b'
= (a' b')''
= (a + b)'

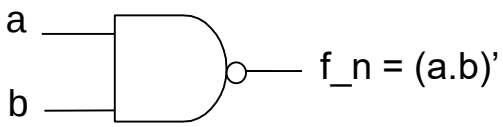
2-input NOR truth table

  - Logic equation
    - f_n = (a + b)'

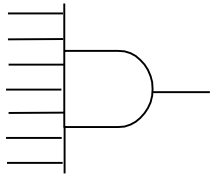
- NOR function is derived from OR
  - Has exactly the OR gate behaviour except that the output is active-low.
  - The output f_n is HIGH only when both inputs a and b are LOW.

NAND Gate

- Logic symbol of NAND gate:



2-input NAND gate



Multiple input AND gate

a	b	f_n = (a b)'
0	0	1
0	1	1
1	0	1
1	1	0

2-input NAND truth table

Deriving logic function from the truth table,

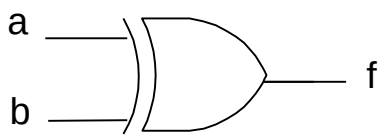
f_n = a' b' + a' b + a b'
= a' + b'
= (a' + b')''
= (a' + b')''
= (a b)'

- Logic equation
  - f_n = (a b)'

- NAND function is derived from AND
  - Has exactly the AND behaviour except that the output is active-low.
  - Output f_n is HIGH when either inputs a or b is LOW, or when a and b are LOW.

Exclusive-OR (XOR) Gate

- Logic symbol of XOR gate:



2-input XOR gate

a	b	f = a ⊕ b
0	0	0
0	1	1
1	0	1
1	1	0

2-input XOR truth table

- Logic equation
  - f = a ⊕ b
  - Deriving logic function from truth table:

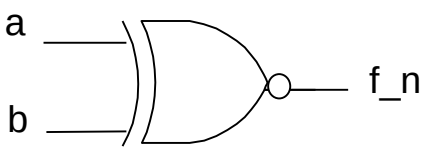
    SOP form: f = a' b + a b'

- For practical purposes, useful to recognize the pattern of both POS and SOP forms for XOR.

- From truth table, XOR function can be described as
  - Odd number of inputs asserted, then output will be asserted.
  - Otherwise (even number), output will be de-asserted.

Exclusive-NOR (XNOR) Gate

- Logic symbol of XNOR gate:



2-input XNOR gate

a	b	f_n = (a ⊕ b)'
0	0	1
0	1	0
1	0	0
1	1	1

2-input XNOR truth table

- Logic equation
  - f_n = (a ⊕ b)' = a ⊙ b

  - Deriving logic function from truth table,

    SOP form: f_n = a' b' + a b

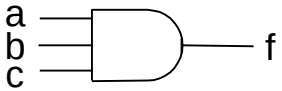
- XNOR function is derived from XOR
  - Has exactly the XOR behaviour except that the output is active-low.

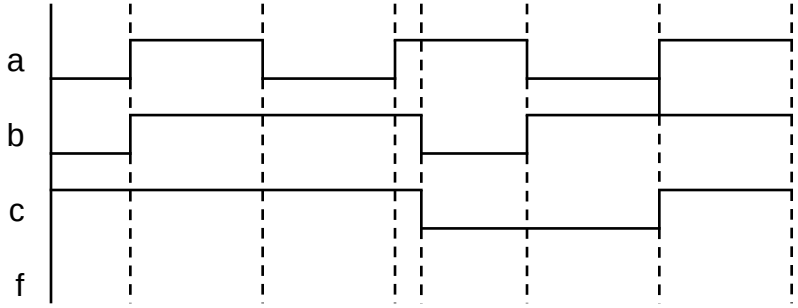
Timing Diagram Representation of a Logic Function

- Timing diagram is useful:
  - To relate output to inputs in terms of *functional behaviour* and *timing behaviour* of the circuit.
  - As a requirement to design circuits.
  - To analyse the output of the circuits in response to the different combination of input signals - testing.

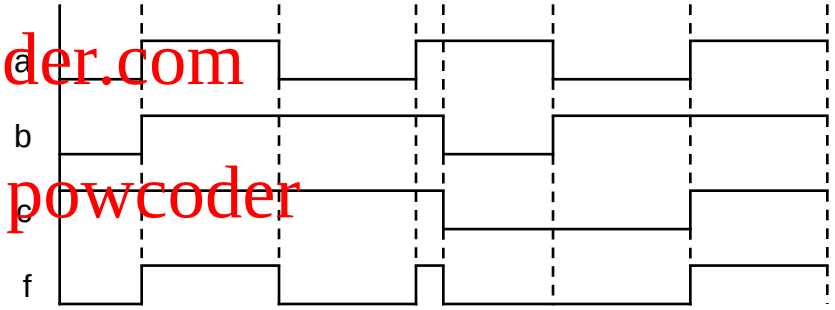
- Example 1

  Sketch the output waveform at f for the 3-input AND gate shown below, with the given a, b and c input signals.



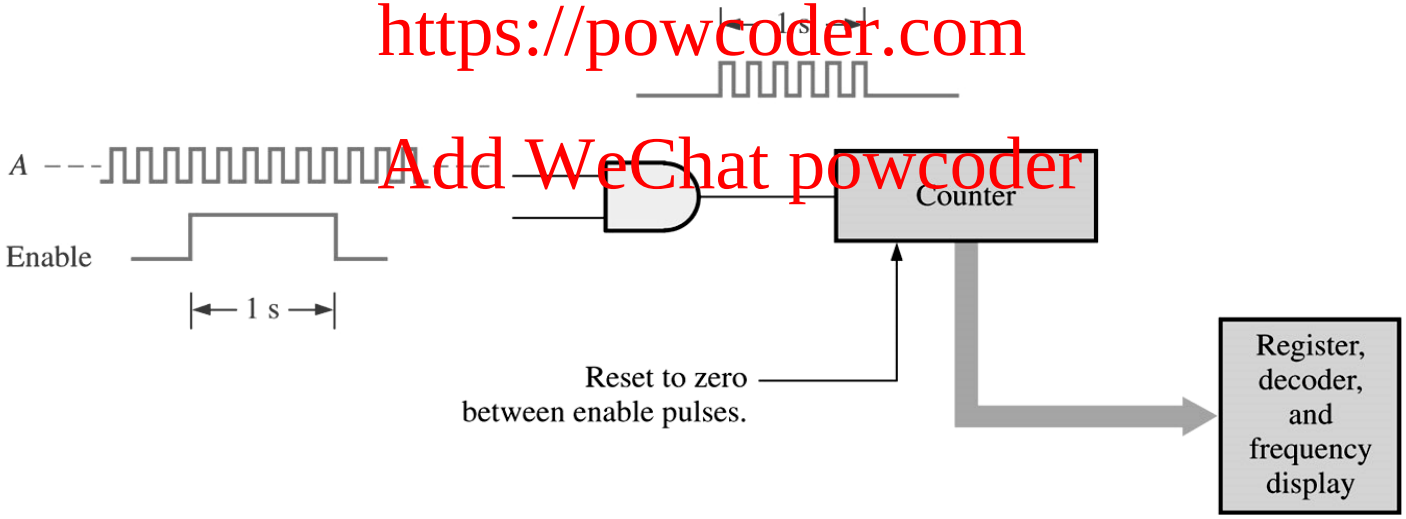


- Solution 1



Basic Logic Gates Applications: Enable and Disable Functions

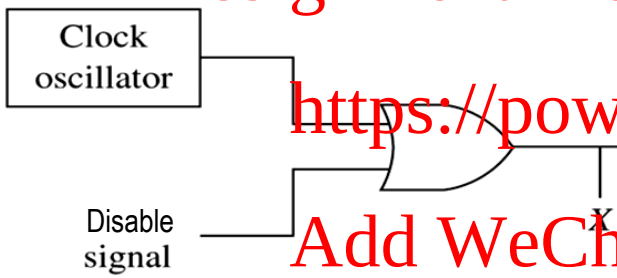
- Example 1
  - A circuit is to measure the frequency of waveform A.
  - Frequency => rate of occurrence i.e. how many occurrences per second.
  - The AND gate is enabled for 1 s for waveform A pulses to pass through and counted by the counter.
  - Then the AND gate is disabled by de-asserting enable signal.
  - No pulses to the counter
    - Signal A is blocked / disabled by the AND gate.
  - The counter counts the number of pulses per second and produces a binary output that goes to a decoding and display circuit to produce readout of the frequency.

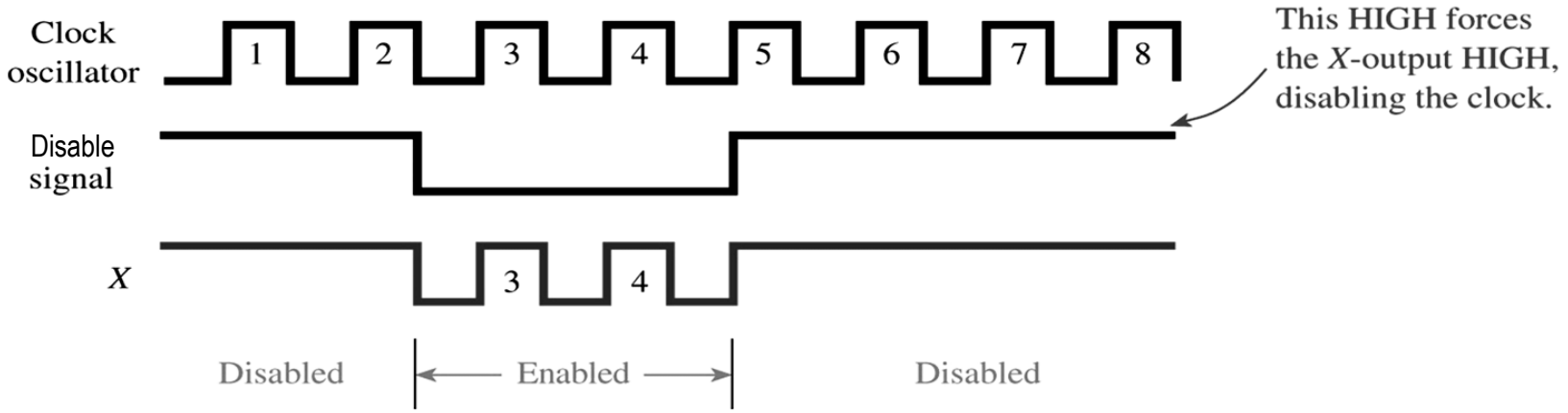


The enable pulse is repeated at certain intervals for new count. If the frequency of signal A changes, a new value will be displayed. Between enable pulses, the counter is reset to zero to ready for a new count.

Basic Logic Gates Applications: Enable and Disable Functions

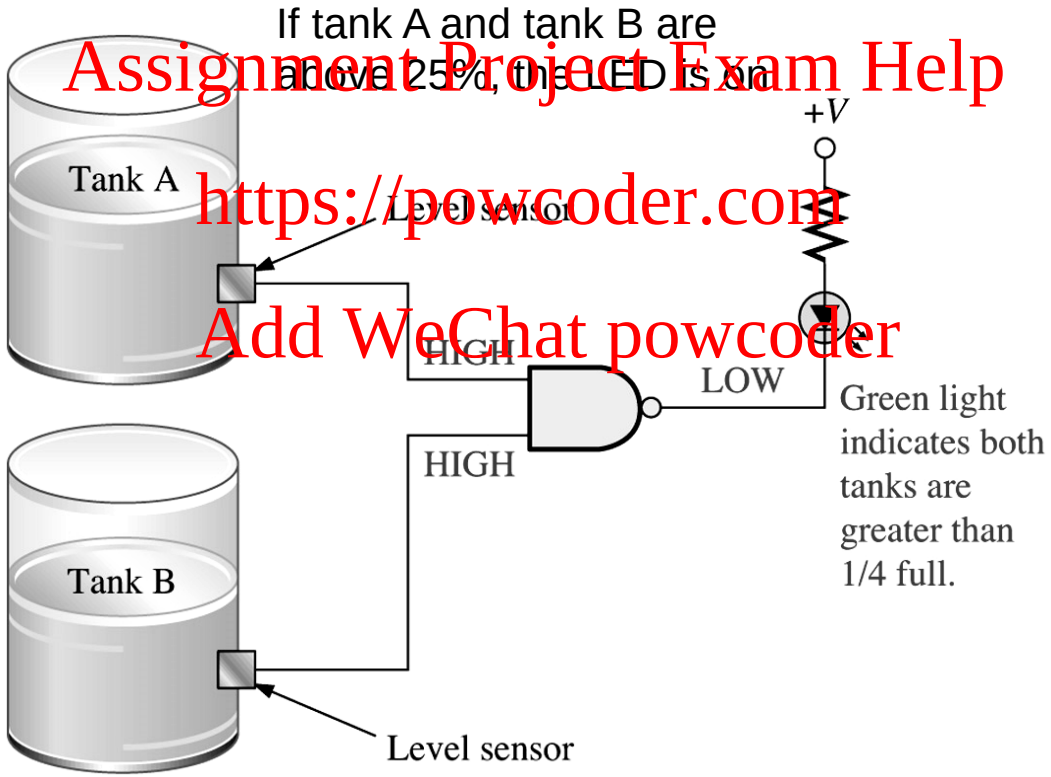
- Example 2
  - A 2-input OR gate is used to enable or disable a clock signal.
  - Instead of using an enable signal name, it is more appropriate to use the disable signal name.
  - When the disable signal is asserted, the OR output stays HIGH – no clock signal appears on the OR output.
  - When the disable signal is de-asserted, the clock signal appears on the OR output.





Basic Logic Gates Applications: Enable and Disable Functions

- Example 3
  - A manufacturing plant uses two tanks to store liquid chemicals. Each tank has a sensor that detects the chemical level drops to 25% of full – the sensors produce a HIGH when the water level is above 25% and LOW otherwise. When both tanks are more than 25% full, a green LED will light. Show how a NAND gate can be used to implement the function.

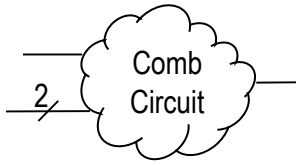


Chapter 3-4

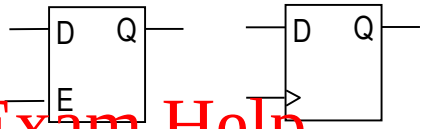
Combinational circuits Analysis and Synthesis

Digital Circuit Structure Classification

- Generally, digital circuits are classified into:
  - Combinational circuit.
    - Asynchronous (non-clocked).



  - Memory element circuit.
    - Asynchronous (non-clocked) – latch circuits.
    - Synchronous (clocked) – flip-flop circuits.



Combinational Circuit

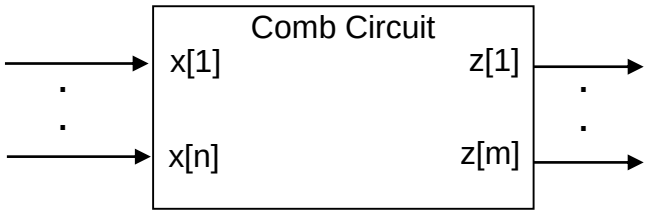
- Definition:
  - Combinational circuit - the output is a function of only the present input.

- The functional behaviour of a combinational circuit can be described/ represented/ modelled using:
  - Boolean algebra.
    - z[m] = fm(x[1], …, x[n]), where m = 1 to k
  - Truth table.
  - Timing diagram.
  - Block diagram and schematic diagram.
  - HDL – Hardware Description Language

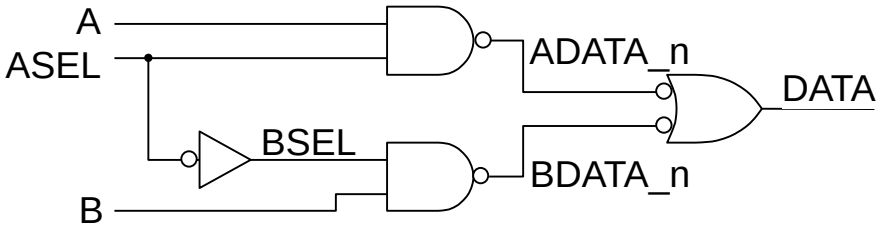
- The timing behaviour can be modelled using the timing diagram.

- Basically combinational circuits - to convert one form of data into another form to suit the requirement of the receiving circuit blocks.

- Example



Block diagram of a combinational circuit.

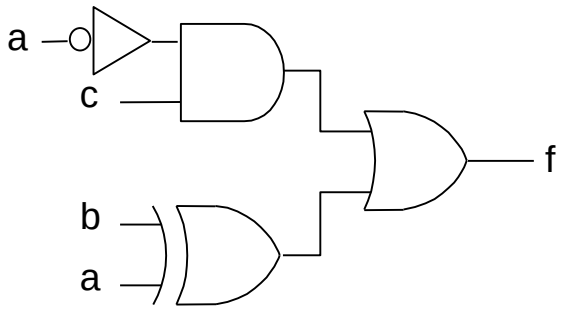


Combinational Circuit Analysis

- Combination circuit analysis enable us to determine the functions that the circuits implement.
- Analysis is useful to:
  - Determine the behavior of the circuit
    - Logic equations, truth tables, timing diagrams and schematic are used to describe the behavior of the logic circuit.
  - Verify the correctness of the circuit (as compare to the circuit specification).
  - Assist in converting the circuit to a different form.

- Example 1

  Describe the behaviour of the logic circuit in the form of truth table.



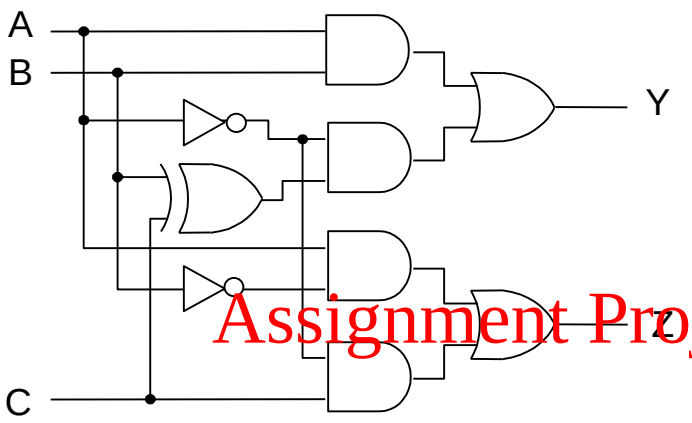
Solution 1

Convert the schematic into the truth table

Input	Intermediate Output	Intermediate Output	Output
a b c	**a'c**	**a ⊕ b**	**f = (a ⊕ b) + a'c**
0 0 0	0	0	0
0 0 1	1	0	1
0 1 0	0	1	1
0 1 1	1	1	1
1 0 0	0	1	1
1 0 1	0	1	1
1 1 0	0	0	0
1 1 1	0	0	0

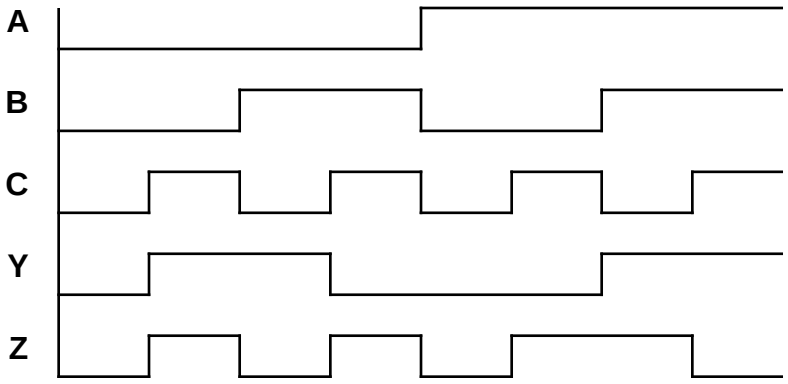
Combinational Circuit Analysis

- Example 2



Given the circuit topology, it is stimulated with a sequence of inputs to obtain the output signals Y and Z as shown in the timing diagram. Since there are 3 inputs, A, B and C, we can use all the 2^3 combinations of input values.

Find the truth table and the minterm lists for Y and Z.



Solution 2

Obtain the truth table then the minterm lists.

Inputs			Outputs	
A	B	C	Y	Z
0	0	0	0	0
0	0	1	1	1
0	1	0	1	0
0	1	1	0	1
1	0	0	0	0
1	0	1	0	1
1	1	0	1	1
1	1	1	1	0

$Y = A'B'C + A'BC' + ABC' + ABC$

$= \sum m(1, 2, 6, 7)$

$Z = A'B'C + A'BC + AB'C + ABC'$

$= \sum m(1, 3, 5, 6)$

Combinational Circuit Synthesis

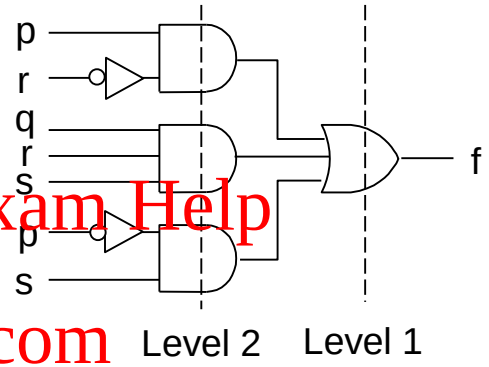
- Combination circuit synthesis:
  - from a model (algebra, truth table, timing diagram) becomes schematic (circuit topology/ structure information)
  - Realisation of the circuit

- From the **circuit topology**, logic gate levels are classified into:
  - Two-level logic. ✓
  - Multilevel logic.

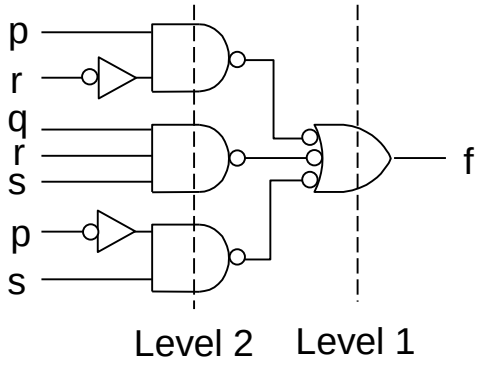
Two-level Logic

- In a two level logic, input signals pass through 2 levels of logic to reach the output.
  - For now, ignore inverters since very small delay
- The two-level logic circuit has two natural forms
  - SOP
    - AND-OR structure
    - All-NAND structure

- For reading, analysing and modelling work, digital circuits are expressed in terms of AND and OR gates.



- For actual implementation on silicon (to prepare for physical design), use NAND and NOR.
  - Faster than AND and OR



Two-level Logic: AND-OR and All-NAND circuit structures

- SOP natural form is AND-OR structure
- Getting from AND-OR to All-NAND structure
  - Method 1: gate manipulation f
  - Method 2: algebra manipulation

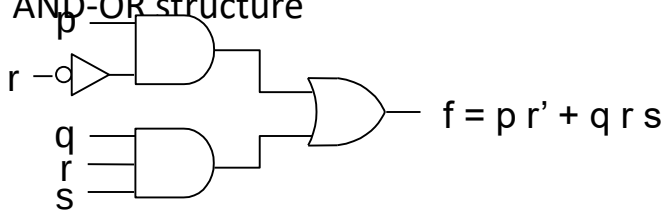
- Example 1

  Implement the function f = pr' + qrs

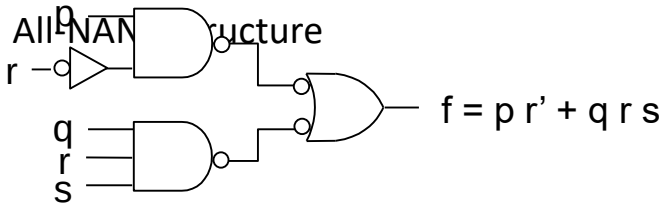
  using All-NAND structure

- Solution (Method 1)

AND-OR structure



All-NAND structure



- Solution (Method 2)

f = p r' + q r s

= (p r' + q r s)'' (DeMorgan's theorem)

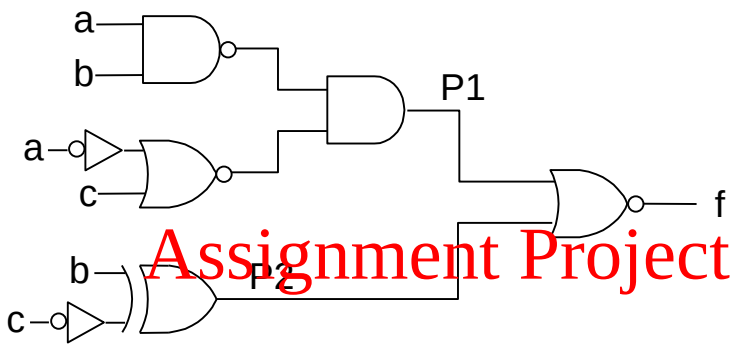
= ((p r')'(q r s)')' (NAND form)

After we have obtained the NAND form, draw the circuit

Combinational Circuit Synthesis

- Example 2

  Find a simplified logic network for the logic circuit shown below.



Solution 2

Convert the schematic into Boolean algebra. Then minimise the logic expression.

If the circuit is complex, can look for intermediate logic expression first. Example.

P1 = (ab)'(a' + c)'

P2 = b ⊕ c'

f = (P1 + P2)'

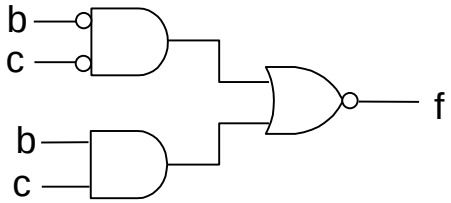
f = ((a b)'(a' + c)' + (b ⊕ c'))'

f = ((a b)'(a' + c)' + (b c + b' c'))'

f = ((a' + b')(a c') + (b c + b' c'))'

f = (a b' c' + (b c + b' c'))'

f = (b' c' + b c)'



2 levels, two 2-input NOR, one 2-input AND, 4 literals

Combinational Circuit Synthesis

- Example 3

  Implement f(X, Y, Z) = ∑ m(0, 3, 4, 5, 7) in NAND logic.

  Solution 3

  f = ∑ m(0, 3, 4, 5, 7)

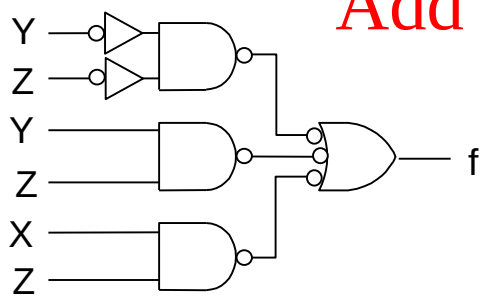
  = m0 + m3 + m4 + m5 + m7

  = X' Y' Z' + X' Y Z + X Y' Z' + X Y' Z + X Y Z

  = Y' Z' + Y Z + X Z

  = (Y' Z' + Y Z + X Z)'' or (Y' Z')'' + (Y Z)'' + (X Z)''

  = ((Y' Z')' (Y Z)' (X Z)')'



Combinational Circuit Synthesis

- Example 4

  The input to a logic circuit consists of four signals A, B, C and D. These inputs represent 4-bit binary number where A is the MSB and D is the LSB. Use algebraic method to design a 2-level logic circuit such that the output is high only when the binary number input is less than 7. Design a circuit using only NAND gates.

Solution 4

Using NAND gates => SOP:

f(A,B,C,D)

= ∑m(0,1,2,3,4,5,6)

= A'B'C'D' + A'B'C'D + A'B'CD' + A'B'CD + A'BC'D' + A'BC'D + A'BCD'

= A'B' + A'C' +A'D'