5. Implementation Using NAND / NOR Gates

5.1 Universal Gates

- NAND and NOR gates are often referred to as **universal gates** as they can produce the NOT, AND, OR, NAND and NOR operations.

- In order to understand how the NOT, AND and OR gates can be obtained, we need to compare their truth tables with those of the NAND and NOR gates.

- Recall that the truth tables of the NAND, NOR, AND and OR gates are as follows:

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

NAND

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

NOR

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

AND

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

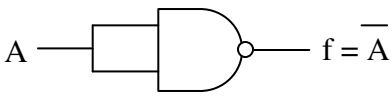
OR

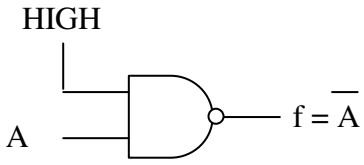
NOT gate using NAND …

- Examining the NAND truth table, it is clear that when both inputs are the same, the output is the complement of the inputs.

- Also, when one input is a logic '1' or HIGH, the output is then the complement of the other input.

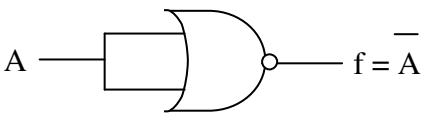
- Thus, 2 possible NAND gate configurations can be used to generate a NOT gate as follows:

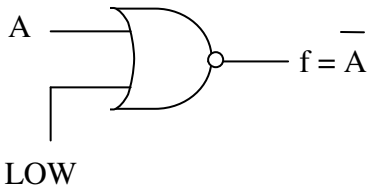




NOT gate using NOR …

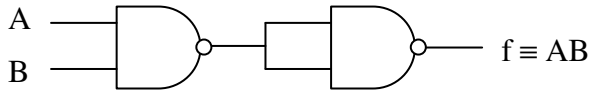
- By a similar analogy to above, there are also 2 possible NOR gate configurations can be used to generate a NOT gate as follows:





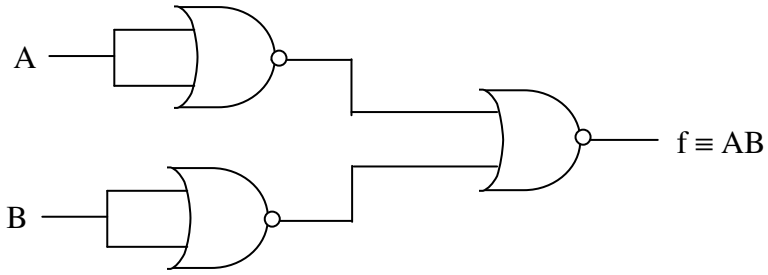
AND gate using NAND …

- Obtaining an AND gate using NAND is straightforward as we simply invert the output of the NAND gate, as follows:



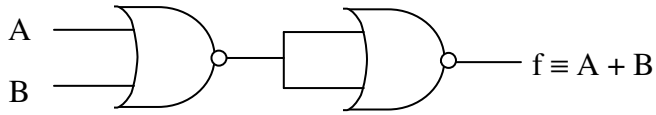
AND gate using NOR…

- By comparing the AND truth table and the NOR truth table we can derive the following NOR gate implementation for an AND operation:



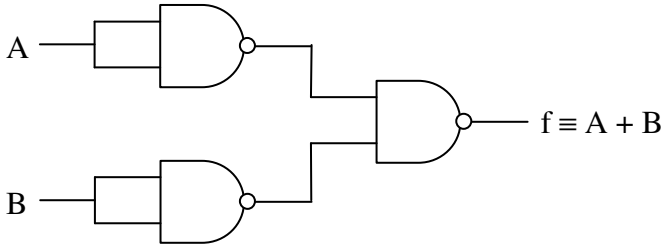
OR gate using NOR …

- Obtaining an OR gate using NOR is straightforward as we simply invert the output of the NOR gate, as follows:



OR gate using NAND …

- By comparing the OR truth table and the NAND truth table we can derive the following NAND gate implementation for the OR operation:



5.2 Implementing Logic using NAND / NOR gates

- In practice, the NAND and NOR gates are regarded as the natural primitive gates of integrated circuit technology, used to implement the simplest and fastest electronic circuits.

- Furthermore, as we have just seen, NAND and NOR gates are 'universal gates' and can be used to implement any other gate or any Boolean expression.

- As such, these gates are the mostly widely used gates in the implementation of actual logic circuits.

- In general, when we carry out logic minimisation (using either Boolean algebra or Karnaugh Maps), we express the final solution in sum-of-products (SOP) form.

- This consists of a collection of AND, OR and NOT gates.

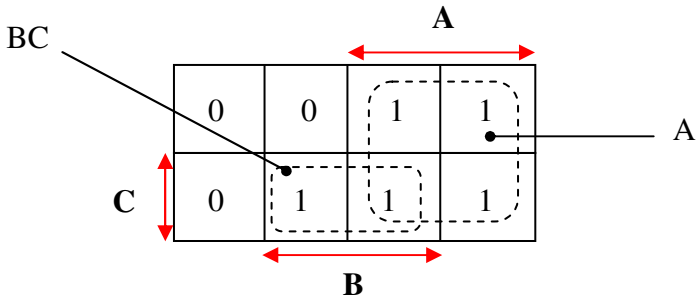
- We now want to implement such expressions using either NAND gates only or NOR gates only.

- De Morgan's theorems is used to achieve the conversion between AND and OR and NAND and NOR. This is best illustrated using examples.

- Consider the implementation of the following function using a quad 2 input **NAND** Integrated Circuit (IC) (i.e. the IC contains four 2-input NAND gates):

$$f_{(A,B,C)} = \overline{A}BC + A\overline{B}\overline{C} + A\overline{B}C + AB\overline{C} + ABC$$

- Firstly, we need to minimise the function. This was done previously in ***Ex. 4.2***, but we will provide the solution here again for the sake of convenience:



Hence: **f = A + BC**

- This is an AND-OR solution. Now, we want to convert this to a NAND only solution.

- Using De Morgan's theorem, we know that:

$$\overline{X + Y} = \overline{X}.\overline{Y} \qquad \text{or} \qquad X + Y = \overline{\overline{X}.\overline{Y}}$$

- Hence:

$$f = A + BC = \overline{\overline{A}.\overline{BC}}$$

- This is now a NAND-NAND solution and can be implemented as follows:

- Note that we have 3 NAND gates in total (the given IC had 4 available).

- Also note that one of the NAND gates is used to implement the NOT gate.

- Now, consider the implementation of the **same function** using a quad 2 input **NOR** Integrated Circuit (IC) (i.e. the IC contains four 2-input NOR gates):

- This time, we want a NOR only solution.

- Using De Morgan's theorem, we know that:

$$\overline{X.Y} = \overline{X} + \overline{Y} \qquad \text{or} \qquad X.Y = \overline{\overline{X} + \overline{Y}}$$

- Thus:

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

- This is currently a NOR-OR format.

- As we have seen already, the OR can be implemented using a NOR gate followed by an inverter, i.e.:

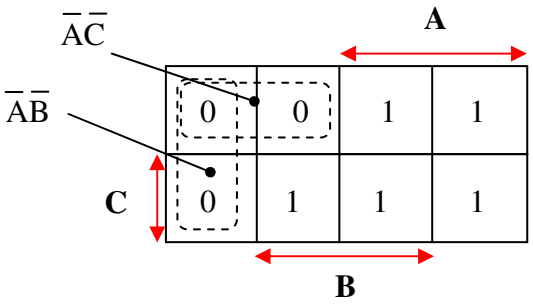
$$X + Y = \overline{\overline{X + Y}}$$

- Hence, the NOR-NOR solution is: $f = A + \overline{\overline{(B} + \overline{C)}}$

- This particular solution requires a total of 5 NOR gates for implementation purposes (the IC only contains 4 NOR gates!).

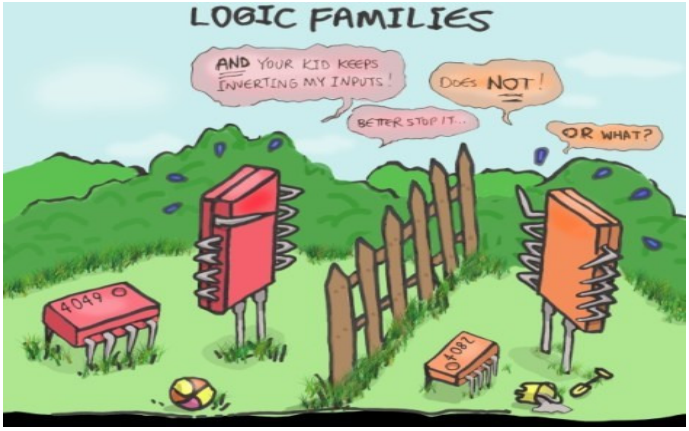
- However, this is **not a minimal** NOR solution.

- Let's revisit the Karnaugh Map, **group the zeros instead** and see what we get:



- This gives: $\overline{f} = \overline{A}\,\overline{B} + \overline{A}\,\overline{C}$
- Using De Morgan's rule, we get: $\overline{f} = \overline{(A + B)} + \overline{(A + C)}$
- Inverting both sides: $f = \overline{\overline{(A + B)} + \overline{(A + C)}}$
- Once again, we have a NOR-NOR solution, but this time we only require 3 gates as follows:

- In general, if we are looking for a NAND gate solution then we group the 1's in a Karnaugh Map.
- In general, if we are looking for a NOR gate solution then we group the 0's, as we have already seen.
- However, this does not ensure a minimal solution in either case.



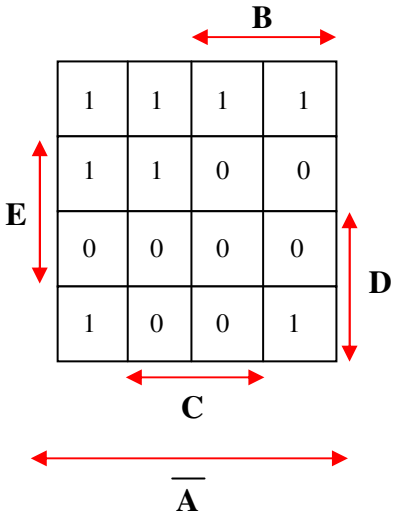
- If we require **a minimal solution** (either NAND or NOR), we need to **examine grouping both the 1's and the 0's in a Karnaugh Map** to see which one leads to the best solution.
- The next exercise will illustrate this concept.

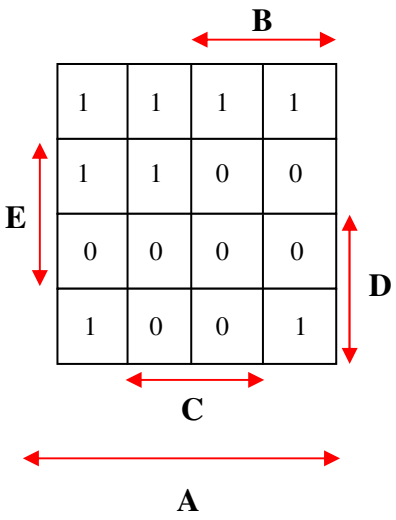
- ***Ex. 5.1 Obtain a minimal NAND and a minimal NOR implementation for the following function:***

$$f_{(A,B,C,D,E)} = \Sigma(0, 1, 2, 4, 5, 8, 10, 12, 16, 17, 18, 20, 21, 24, 26, 28)$$

given that the complemented input variables (i) are available and (ii) are NOT available.

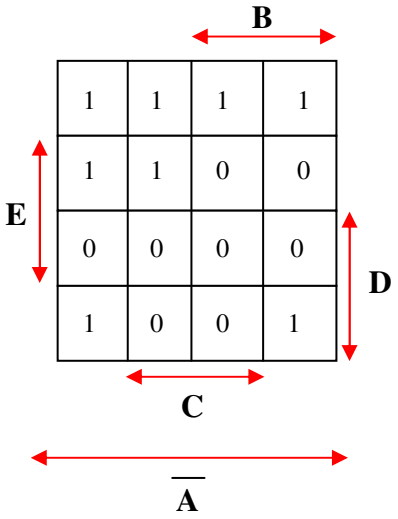
Grouping 1's …

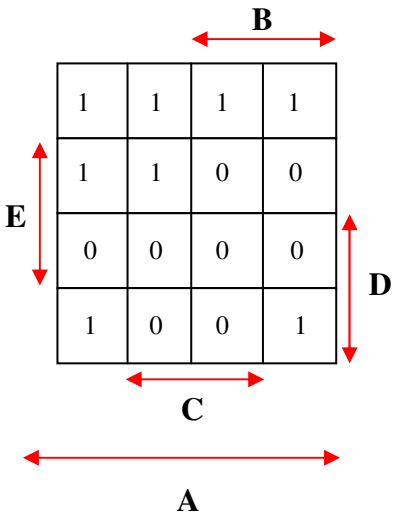




This gives:

Grouping 0's …





This gives:

NAND solutions:

#1:

Applying De Morgan's gives:

This requires 4 NAND gates for (i) and requires 8 NAND gates for (ii).

#2:

Applying De Morgan's gives:

This requires 5 NAND gates for (i) and also requires 5 NAND gates for (ii).

Hence the **minimal NAND solution for (i)** is:

And the **minimal NAND solution for (ii)** is:

NOR solutions:

#1:

Applying De Morgan's gives:

We now need to convert from OR to NOR:

This requires 5 NOR gates for (i) and also requires 5 NOR gates for (ii).

#2:

Applying De Morgan's gives:

This requires 4 NOR gates for (i) and requires 8 NOR gates for (ii)

Hence the **minimal NOR solution for (i)** is:

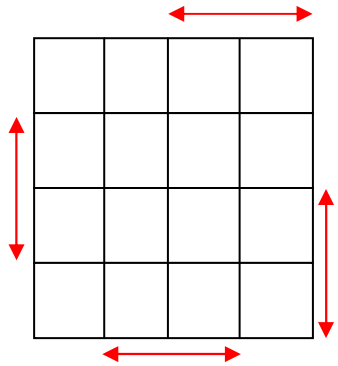
And the **minimal NOR solution for (ii)** is:

- ***Ex. 5.2 The output of a circuit containing 4 inputs is high when any three of the inputs are high, otherwise the output is low.***
  (i) Derive a minimal sum-of products function for this circuit.
  (ii) Implement this function using no more than 5 NAND gates. Draw out your final circuit.

  (i) *We could fill out a truth table and then transfer this to a Karnaugh Map to obtain the solution.*

  Alternatively we can write down all valid combinations as follows:

  Put this expression into a Karnaugh Map and simplify as follows:



  Hence:

  (ii) *We are not required to get a minimal solution in this case. Hence, we should group the 1's first for a NAND solution. If this doesn't meet the criteria, then and only then do we consider grouping the 0's in the Karnaugh Map.*

  Grouping 1's gives us:

  Applying De Morgan's:

  This requires 5 NAND gates which satisfies the given criteria.

  The final circuit is: