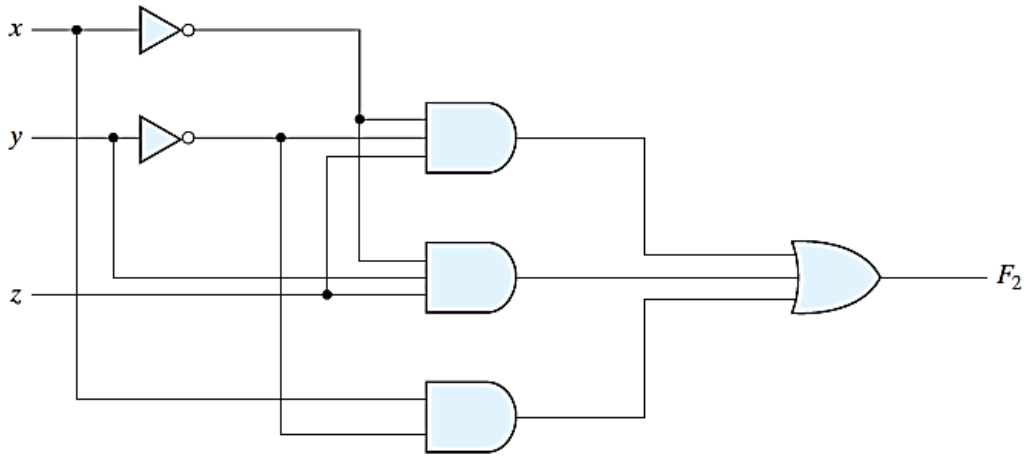
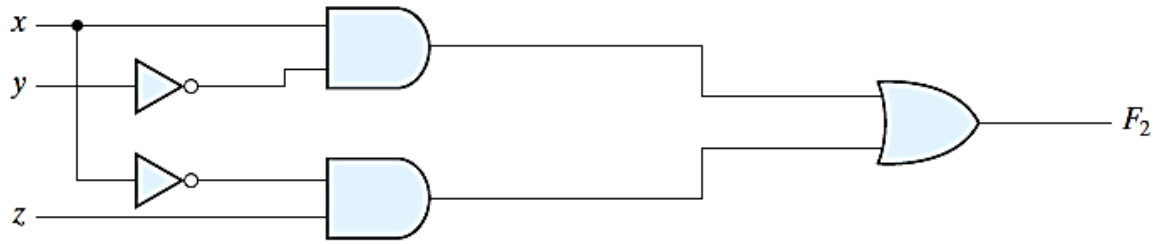
Ex) Simplify and draw the Boolean function $F_2 = x'y'z + x'yz + xy'$



(a) $F_2 = x'y'z + x'yz + xy'$

$$F_2 = x'y'z + x'yz + xy'$$
$$= x'z(y' + y) + xy'$$
$$= x'z + xy'$$

The function is reduced to only two terms and can be implemented with



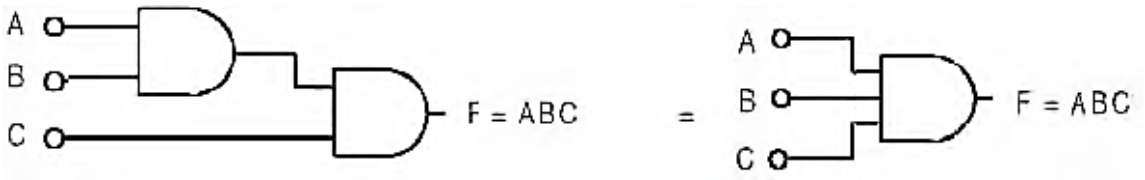
(b) $F_2 = xy' + x'z$

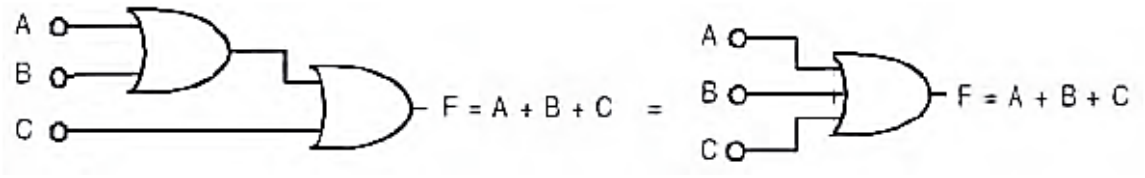
Ex) Simplify and draw the Boolean function $F = (\overline{A} + B)(A + B + C)\overline{C}$

Extension to Multiple Inputs

A gate can be extended to have multiple inputs **if** its binary operation is **commutative** and **associative**. AND & OR gates are both commutative and associative.

For the AND function, AB = BA -commutative
and
(AB)C = A(BC) = ABC. -associative
For the OR function, A + B = B + A -commutative
and
(A + B) + C = A + (B + C). -associative





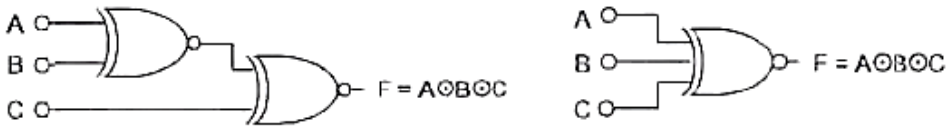
The NAND and NOR are commutative, but **not associative**. So these functions cannot be extended to multiple input.

For NAND function, $(AB)' = (BA)'$. -commutative
But, $((AB)'C)' \neq (A(BC)')'$. -does not follow associative property.

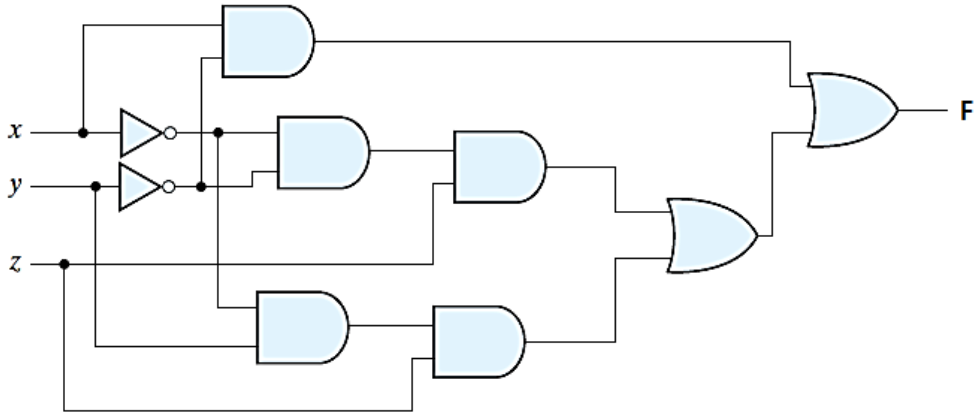
As $((AB)'\ C)' = (AB) + C'$ and
$(A(BC)')' = A' + BC$.

for NOR function, $((A + B)' + C)' \neq (A + (B + C)')'$.
As, $((A + B)' + C)' = (A + B)\ C' = AC' + BC'$.
And $(A + (B + C)')' = A'(B + C) = A'B + A'C$.

The Exclusive-OR gates and equivalence gates both possess commutative and associative properties.



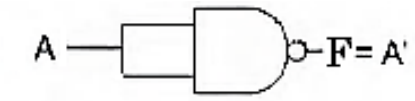
The function $F_2 = x'y'z + x'yz + xy'$ can be exteneded as follow:



Universal Gates

NAND and **NOR** gates are called **Universal Gates** because any type of gates or logic functions can be implemented by using these gates. The advantage of using the universal gates for implementation of logic functions is that it reduces the number of varieties of gates.

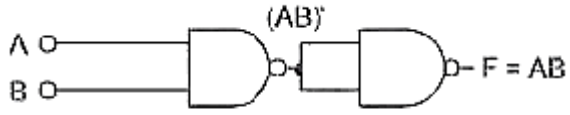
Implementing **NOT** gate using **NAND** and **NOR** gates



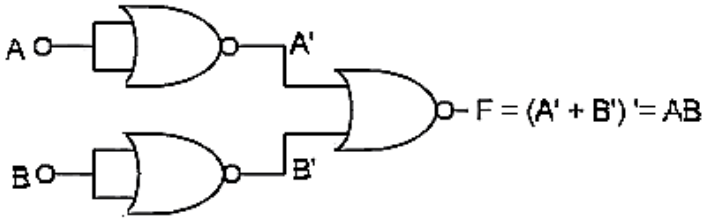
NOT function: F = A′

NOT function: F = A′

Implementing **AND** gate using **NAND** and **NOR** gates

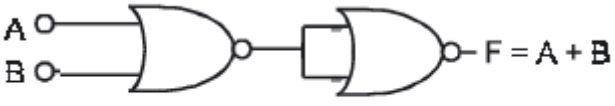


AND function: F = AB

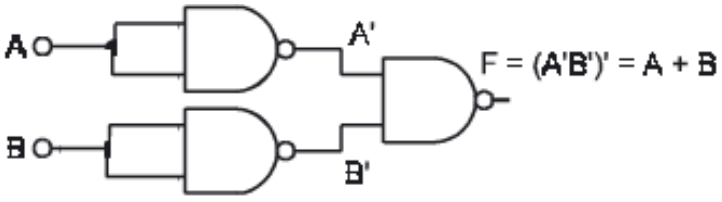


AND function: F = AB

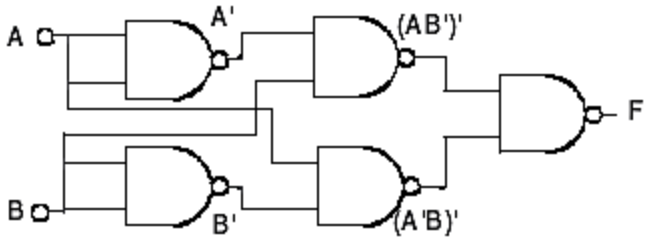
Implementing **OR** gate using **NAND** and **NOR** gates



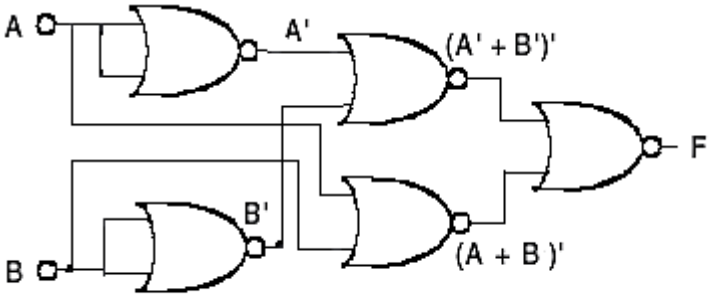
OR function: F = A + B



Implementing **XOR** gate using **NAND** and **NOR** gates



Ex-OR function: F= ((AB′)′(A′B)′)′ =AB′ + A′B



Ex–OR function: F = [(A′ + B′)′ + (A + B)′]′ = AB′ + A′B

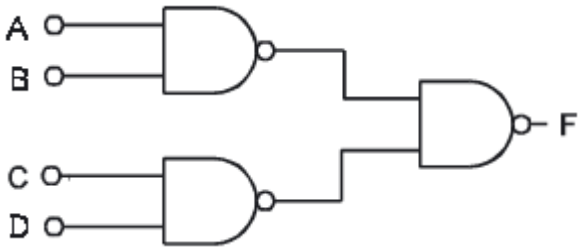
H.W: Implement XNOR using **NAND** and **NOR** gates only.

Implementation of Logic Functions by NAND Gates

To achieve the realization of logic functions by NAND gates, the first step is to express the function in NAND form Using Boolean algebra and Demorgan's theorems as follow:

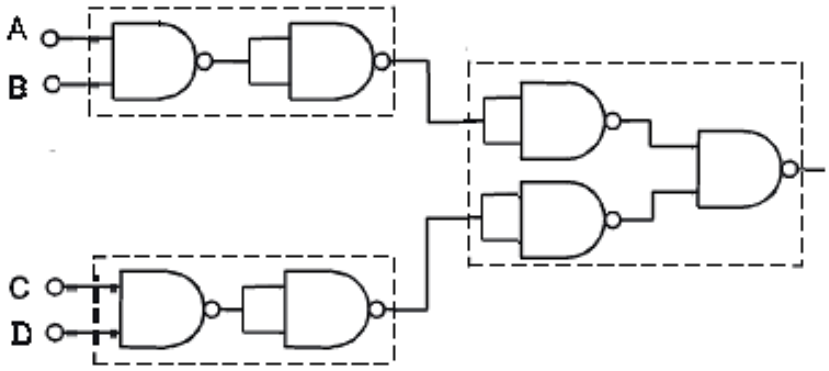
Ex) considering the logic expression F = AB + CD.

$$
\begin{aligned}
F &= AB + CD \\
&= ((AB + CD)')' \\
&= ((AB)'\,(CD)')'
\end{aligned}
$$

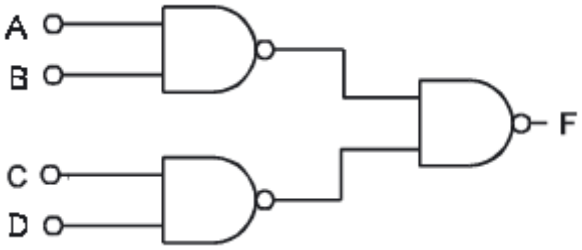


Or we can use the logic diagram of AND, OR, and NOT Gates constructed using NAND gates then remove any two cascaded inverters from the diagram:

$F = AB + CD$



After removing every two cascaded inverters



Ex) Realize the following function by NAND gates only:

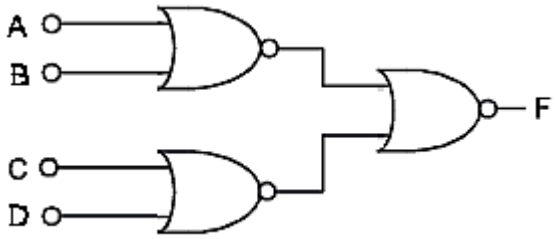
F = B (A + CD) + AC

Realization of Logic Functions by NOR Gates

To achieve the realization of logic functions by NOR gates, the first step is to express the function in NOR form Using Boolean algebra and Demorgan's theorems as follow:

Ex) considering the logic expression $F = (A + B)(C + D)$

$$
\begin{aligned}
F &= (A + B)(C + D) \\
&= (((A + B)(C + D))')' \\
&= ((A + B)' + (C + D)')'
\end{aligned}
$$



Similarly, we can use the logic diagram of AND, OR, and NOT Gates constructed using NOR gates then remove any two cascaded inverters from the diagram