Lecture 6, Jan. 22, 2007

Other two level logic implementations

Let's take four basic logic gates AND, OR, NAND, NOR. There are 16 possible two level arrangements such as AND-AND, AND-OR, ... , NOR-NOR. Out of these 16 combinations 8 will degenerate to a single operations, e.g., AND-AND, OR-OR, AND-NAND, ... (you may verify this). These combinations are called degenerate.

The non-degenerate two level implementations include:

AND-OR	OR-AND
NAND-NAND	NOR-NOR
NOR-OR	NAND-AND
OR-NAND	AND-NOR

The ones on the first row, i.e., AND-OR and OR-AND are the basic sum-of-products and product-of-sums we discussed first.

We have also seen NAND-NAND and NOR-NOR before.

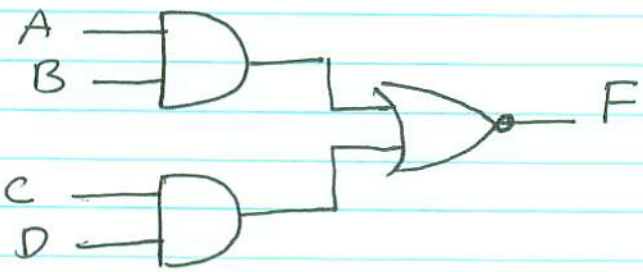
Now, let's discuss the other four combinations.

NAND-AND and AND-NOR implementation

(AND-OR-INVERT implementation)

The two combinations NAND-AND and AND-NOR are equivalent.

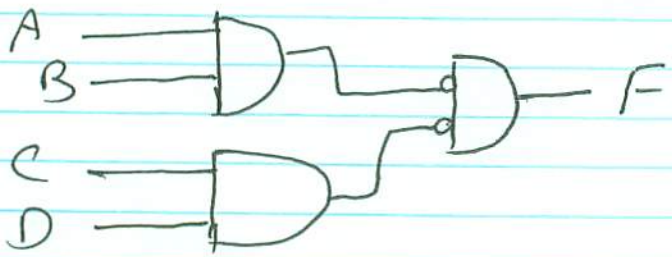
Take as an example:



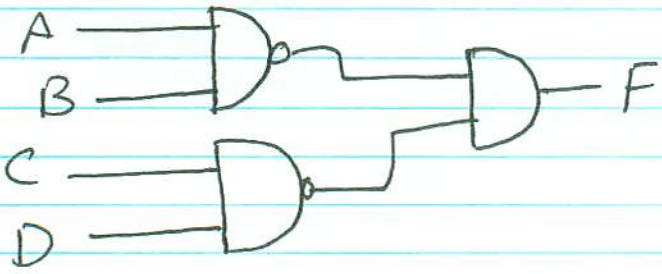
which is an AND-NOR combination implementing

$$F = (AB + CD)'$$

Note that this is equivalent to AND-OR-invert operation. Note also that, we can replace the NOR with the equivalent form to get:



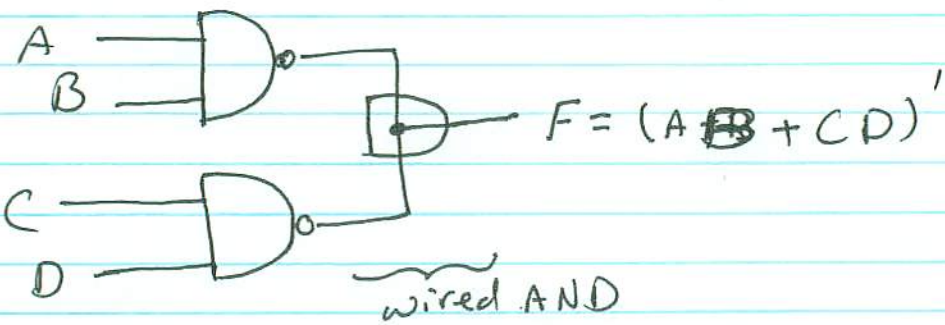
This can be drawan as:



which is a NAND-AND implementation

So: AND-NOR and NAND-AND combinations are equivalent and can be implemented using AND-OR-INVERT.

Two NAND gates implemented using open collector TTL when tied together, their outputs will be AND with each other. This is called a <u>wired</u> AND operation



The output is:

$$F = (AB)' \cdot (CD)' = (A'+B')(C'+D')$$

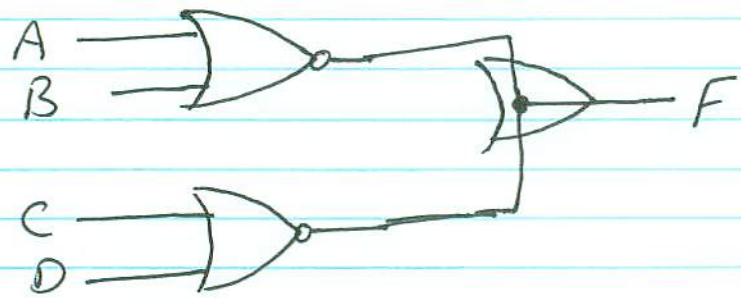
$$= (AB + CD)'$$

which is an AND-OR-INVERT form.

OR-NAND and NOR-OR Combination

(the OR-AND-INVERT implementation)

When two NOR gates implemented in ECL (Emitter Coupled Logic) are wired together, their outputs are OR'd (wired OR logic).

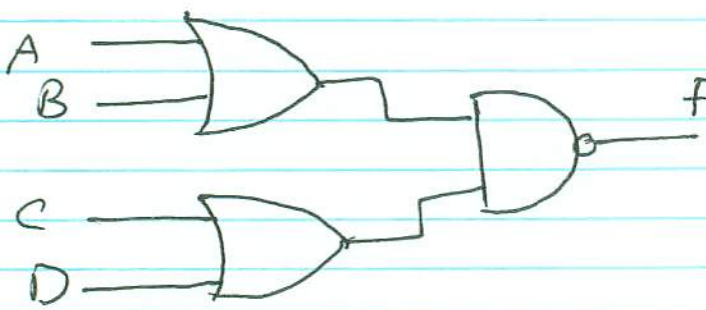


$$F = (A+B)' + (C+D)' = A'B' + C'D'$$

$$= [(A+B)(C+D)]'$$

This is an OR-AND-INVERT implementation.

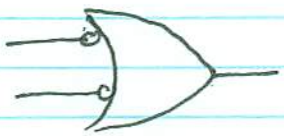
Now, Consider the following OR-NAND Combination:



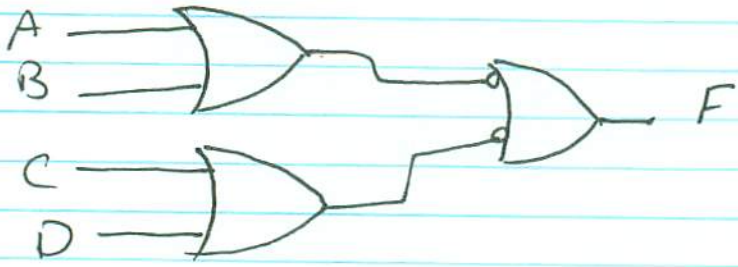
$F = [(A+B)(C+D)]'$

OR-AND-INVERT

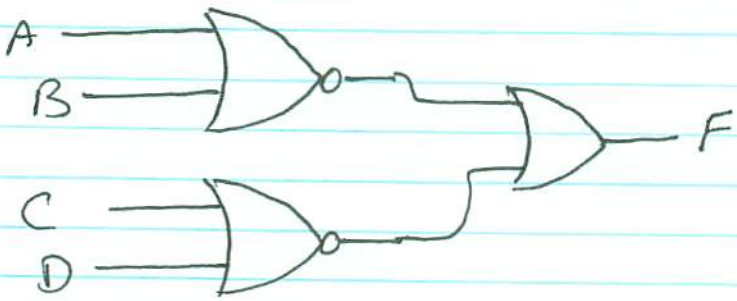
Replacing the NAND gate with its equivalent form



we get:



or equivalently:



which is a NOR-OR Combination.

So, OR-NAND and NOR-OR combinations are equivalent and can be implemented using OR-AND-INVERT operations.

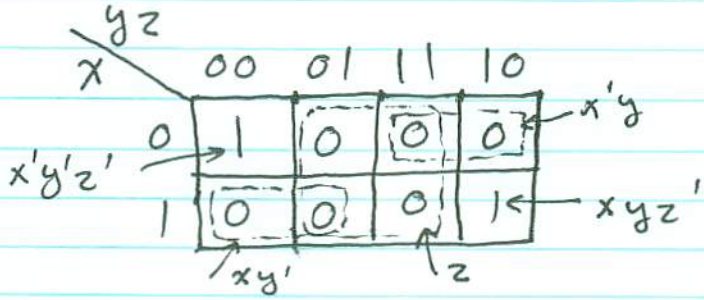
Implementing a function using AND-OR-INVERT or OR-AND-INVERT:

Note that we know how to implement a function (given by a truth table or a K-map) using either sum-of-products (AND-OR) or product-of-sums (OR-AND).

To implement a function using AND-OR-INVERT

we need just to implement its Complement (F') using AND-OR and then the inversion gives F. Similarly, for an OR-AND-INVERT, we implement F' using product of sums.

Example: Implement the function



using OR-NAND and NOR-OR combination. (also AND-NOR and NAND-AND)

We have:

1) $F = x'y'z' + xyz' \Rightarrow F' = (x+y+z)(x'+y'+z)$

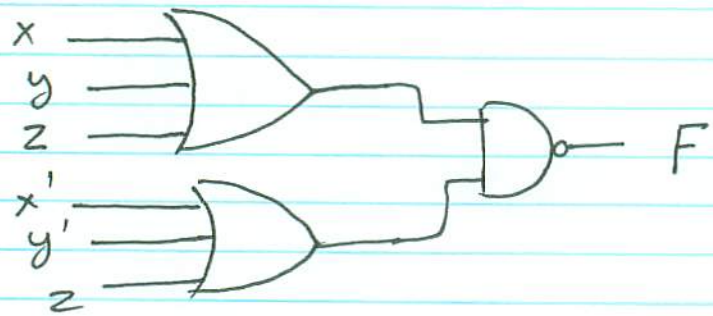
and

2) $F' = z + xy' + x'y \Rightarrow F = (z + xy' + x'y)'$

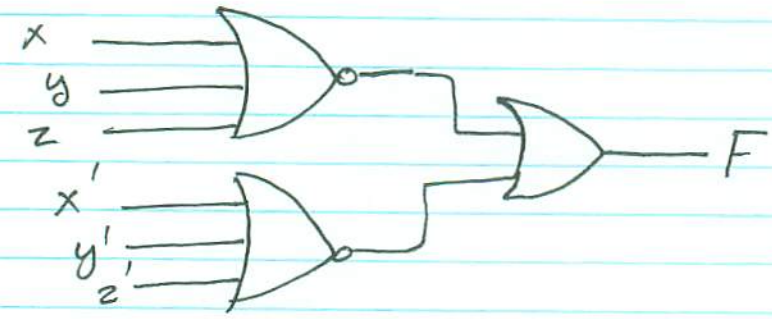
The first one give us:

$$F = [(x+y+z)(x'+y'+z)]'$$

which is an OR-AND-INVERT or OR-NAND.



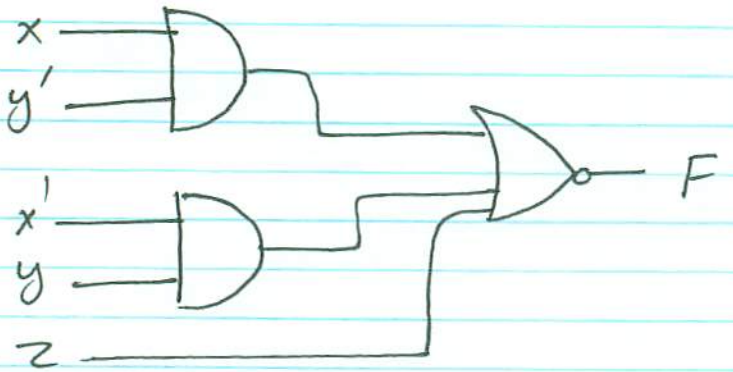
This can be easily changed into NOR-OR form



The second expression

$$F = (z + xy' + x'y)'$$

Can be implemented as



which is an AND-OR-INVERT or AND-NOR

This can be easily transformed to NAND-AND: