Lecture 5, Jan. 17, 2007

Product of Sums Simplification

Recall that when we simplify sum of products, we combine squares with a 1 in them. The squares with no 1 represent the complement of the function. If we put a "0" in the squares with no 1 in them and combine these squares to form F' and finally complement the F', we get F in the product of sums form.

<u>Example</u>: Simplify $F = \sum(0,1,2,5,8,9,10)$

in a) sum of products form

b) product of sums form

wx \ yz	00	01	11	10
00	1	1	0	1
01	0	1	0	0
11	0	0	0	0
10	1	1	0	1

$w'y'z$; $x'z'$; $x'y'$

a) $F = x'z' + x'y' + w'y'z$

wx \ yz	00	01	11	10
00	1	1	0	1
01	0	1	0	0
11	0	0	0	0
10	1	1	0	1

b)

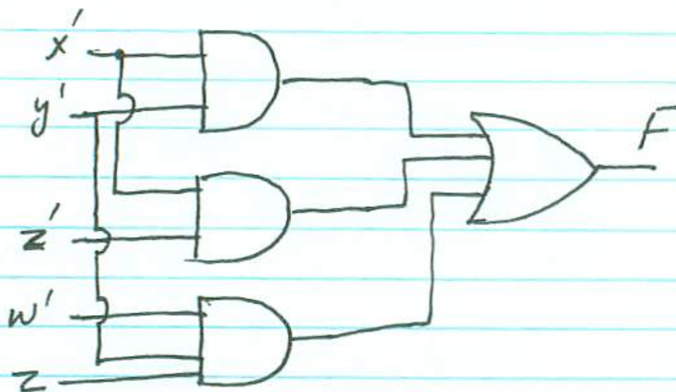
$$F' = wx + yz + xz'$$

and inverting F', we get

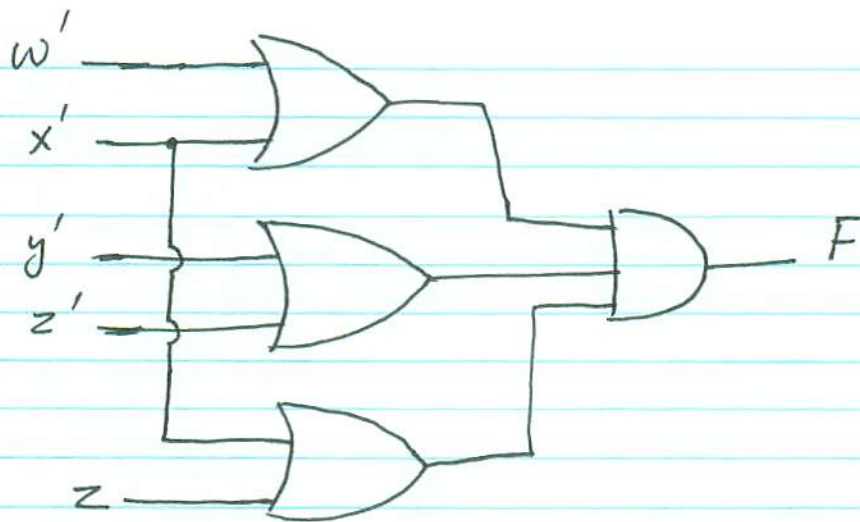
$$F = (w' + x')(y' + z')(x' + z)$$

The forms derived in parts (a) and (b) result in two implementations.

$F = x'y' + x'z' + w'y'z$ is implemented as:



$F = (w' + x')(y' + z')(x' + z)$ can be implemented as: (See next page).



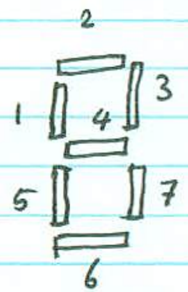
The above implementations are two examples of two-level implementation. One uses AND-OR implementation where AND is used in the first level and OR is used in the second level. The second implementation uses OR-AND implementation. Here OR is used in level one and in level two AND is used. We will see other two level implementations later.

Don't Care Conditions

Sometimes the value of the logic for a certain combination of variables either is not defined or not important for us. In such a case, we put a X instead of a 0 or a 1 in the square. These squares can be considered as

a 1 square or a zero square and combined with other squares of similar content when doing simplification. We consider a don't care a 1 or a 0 as is suitable for simplification.

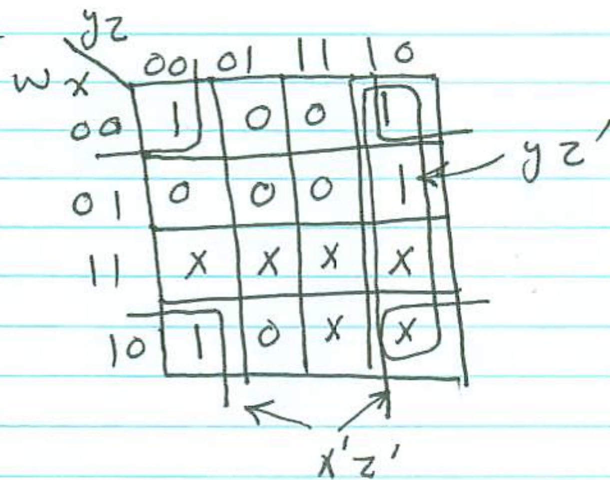
As an example consider a 7-segment encoder. Since a 7-segment LED only counts 0, 1, ⋯, 9 certain combinations of bits do not occur. So, we use don't care to represent the output of encoder for these combinations:



<u>Example</u>: Design a 7-Segment encoder.

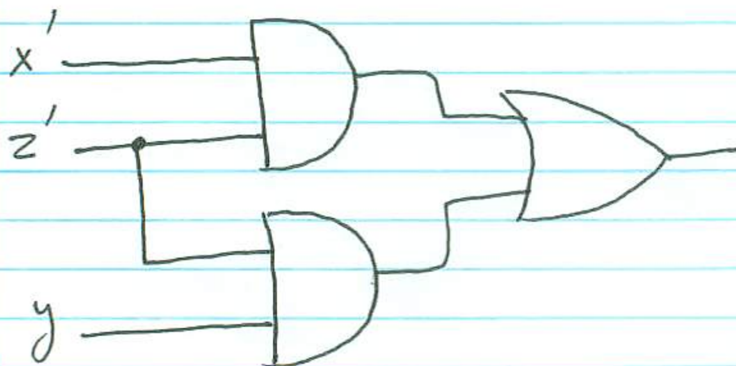
w	x	y	z	F_1	F_2	F_3	F_4	F_5	F_6	F_7
0	0	0	0	1	1	1	0	1	1	1
0	0	0	1	0	0	1	0	0	0	1
0	0	1	0	0	1	1	1	1	1	0
0	0	1	1	0	1	1	1	0	1	1
0	1	0	0	1	0	1	1	0	0	1
0	1	0	1	1	1	0	1	0	1	1
0	1	1	0	1	1	0	1	1	1	1
0	1	1	1	0	1	1	0	0	0	1
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	1	0	1	1
1	0	1	0	X	X	X	X	X	X	X
1	0	1	1	X	X	X	X	X	X	X
1	1	0	0	X	X	X	X	X	X	X
1	1	0	1	X	X	X	X	X	X	X
1	1	1	0	X	X	X	X	X	X	X
1	1	1	1	X	X	X	X	X	X	X

Now, let's implement one of these functions, say F_5. You may try to do the rest as an exercise



We consider don't cares in squares 1010 and 1110 as 1's and combine them with two other 1's to get the term yz'. We also combine x in 1010 with three other squares to get $x'z'$. The other x's will be considered as 0's. So, we have

$$F_5 = x'z' + yz'$$

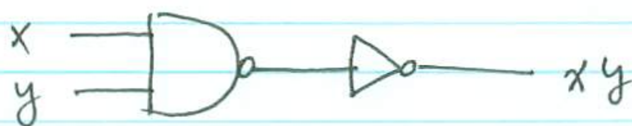


NAND and NOR implementations

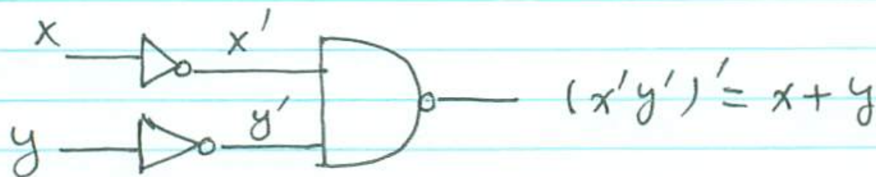
So far we have seen two-level implementations using AND-OR or OR-AND. Digital circuits are, however, implemented more frequently using NOR or NAND gates since these gates are easier to implement.

NAND Circuits

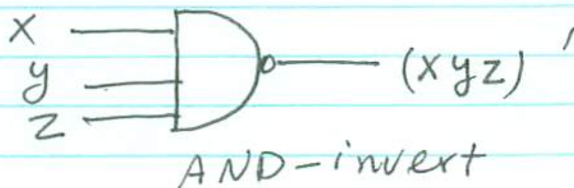
An AND gate can be implemented using a NAND and an inverter:



An OR gate can be implemented using two inverters and a NAND gate.

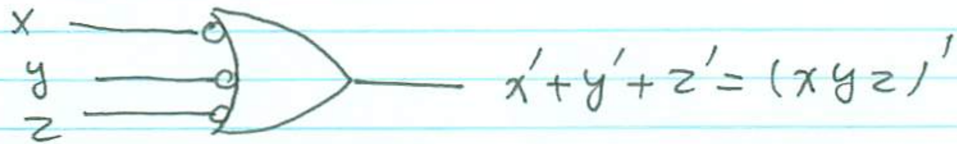


Note also that a NAND gate can either implemented by inverting the output of an AND:



AND-invert

or by inverting the variables and then feeding them to an OR gate

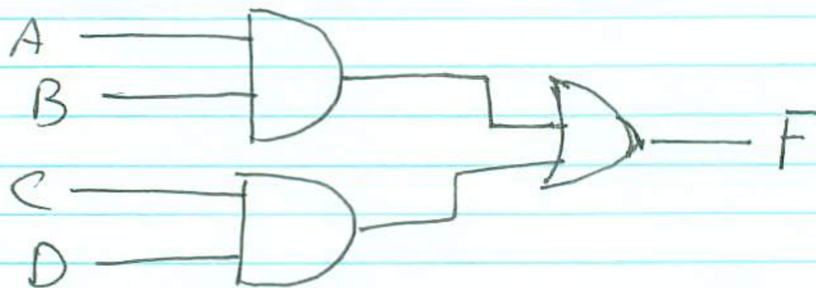


Invert-OR

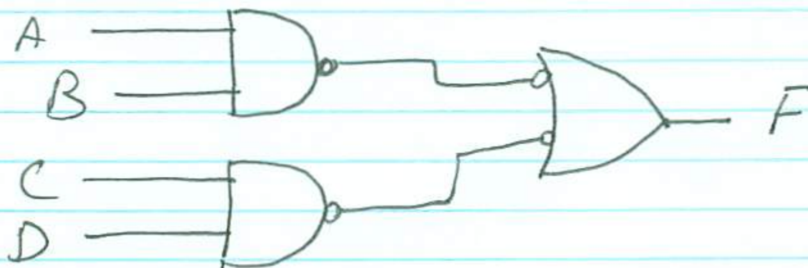
Now, let's implement

$$F = AB + CD$$

the AND-OR implementation is:



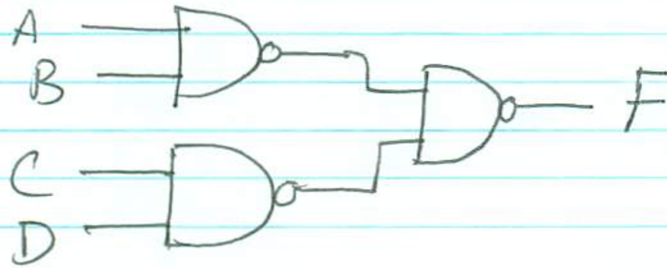
If we invert the output of the AND's and also invert the input of the OR, we will have the same circuit since $(x')' = x$



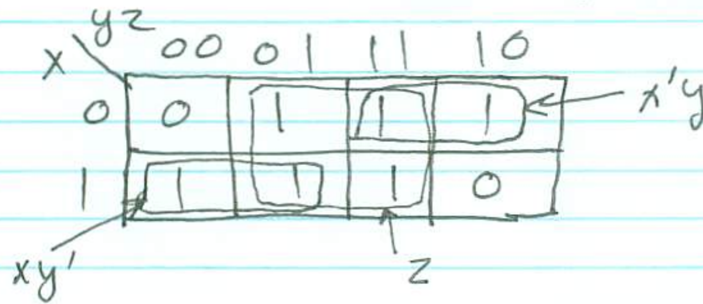
Note that Now we have two NAND's in the first level and another NAND, although

in invert-OR for in the second level.

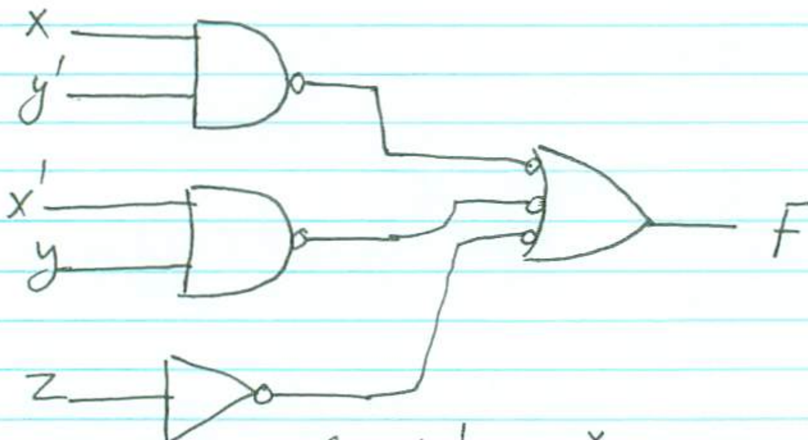
So, we have the following NAND-NAND implementation



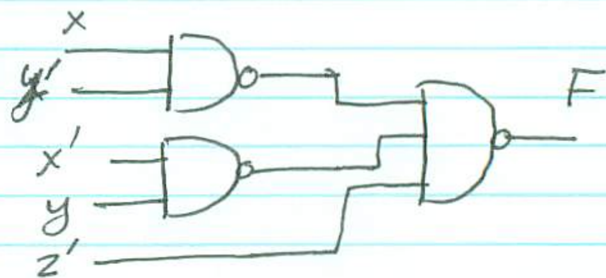
Example: Implement $F(x, y, z) = \sum(1,2,3,4,5,7)$ with NAND gates



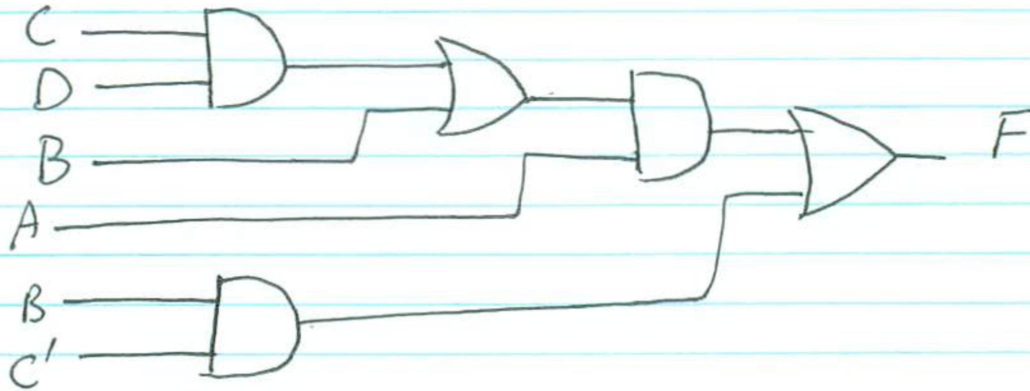
$$F = xy' + x'y + z$$



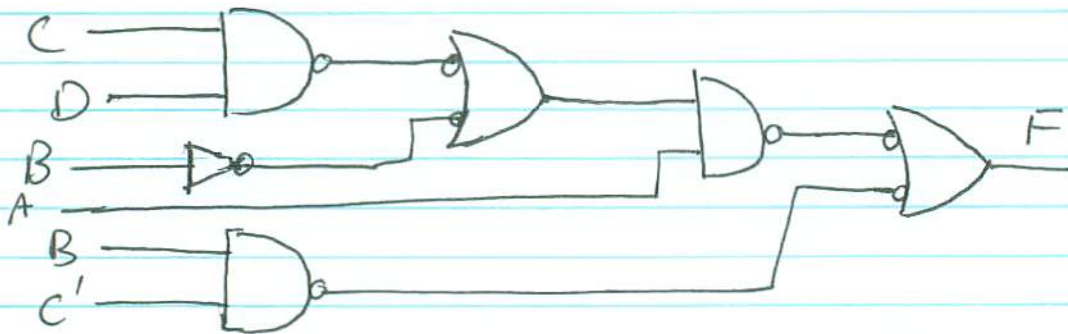
or equivalently



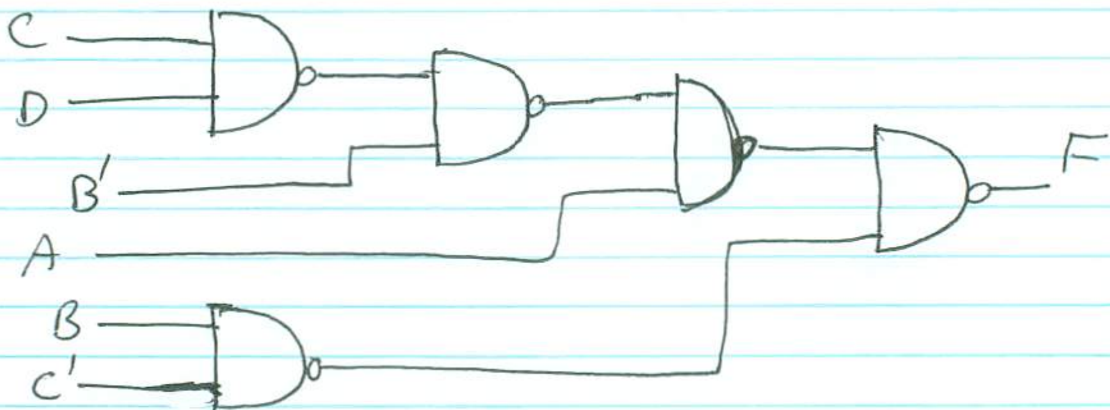
Example: Implement $F = A(CD+B) + BC'$ using NAND gates.



Turn this into:



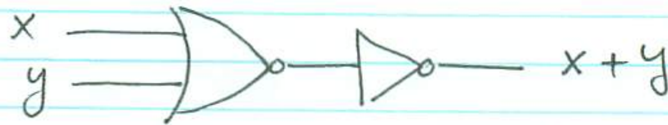
and Finally:



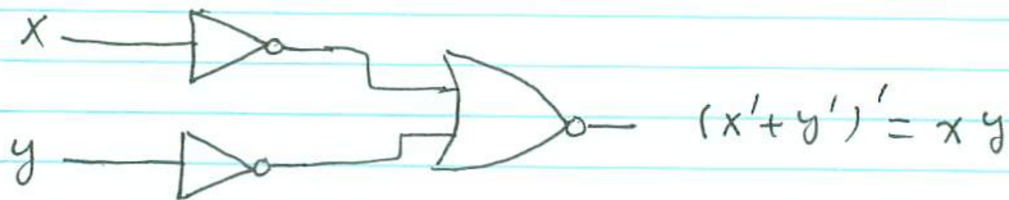
NAND-NAND implementation

NOR implementation

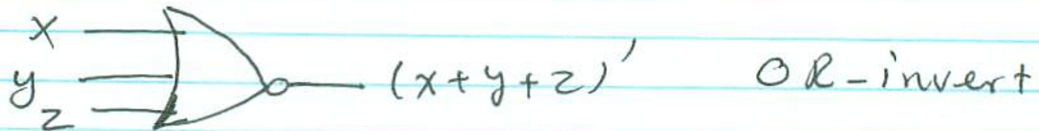
The implementation of an OR gate using NOR is



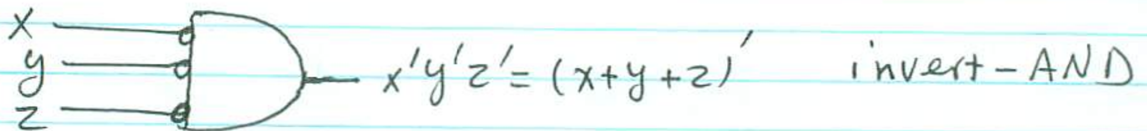
an AND gate can be implemented as



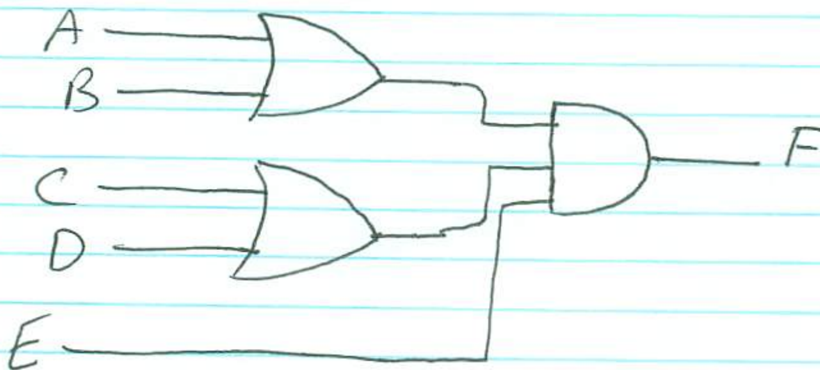
A NOR gate can be represented as



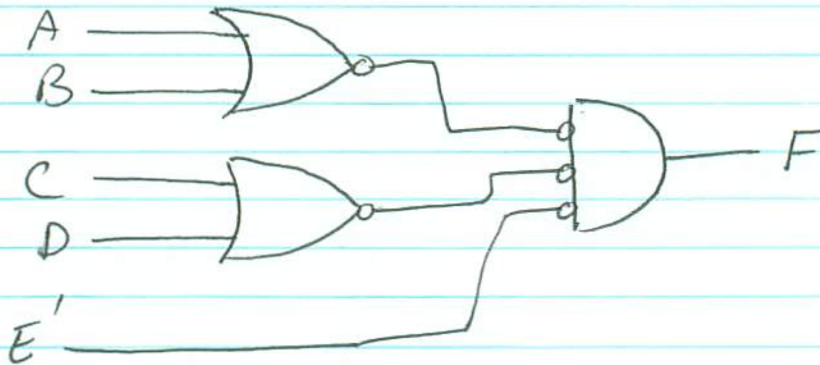
or



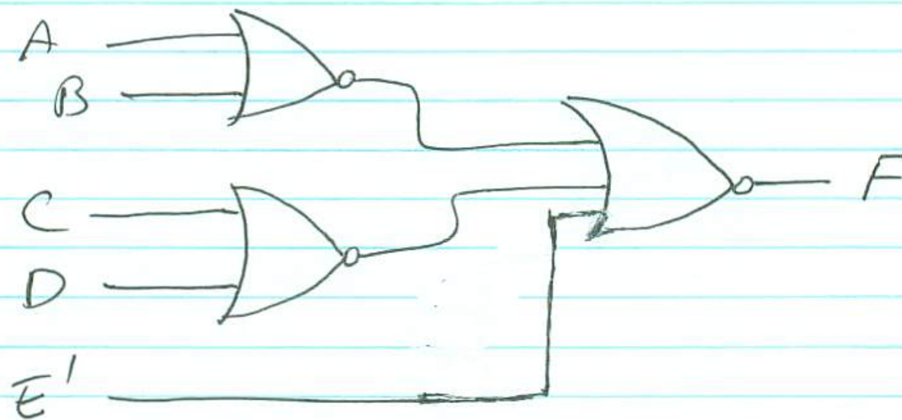
Example: Implement $F = (A+B)(C+D)E$ using NOR gates.



We invert the output of OR's and the inputs of the AND gate:



We can replace the 2nd. level NOR with the more familiar symbol to get:



NOR-NOR implementation.