LECTURE 5: DIGITAL LOGIC CIRCUIT DESIGN

Prime Implicants

We must insure that all minterms of the function are covered using the minimum number of groups. Also redundant terms should be avoided. Sometimes there could be two or more expressions that satisfy the simplification criteria. Prime implicants and essential prime implicants help in organizing the procedure for simplification.

Prime Implicant

A product term obtained by combining the maximum possible number of adjacent squares in the map.

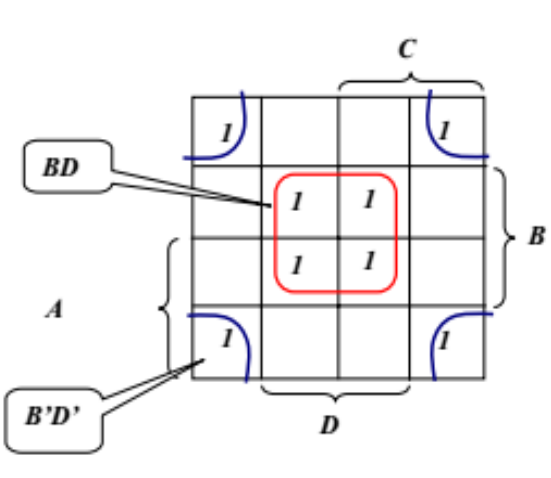
Essential Prime Implicant

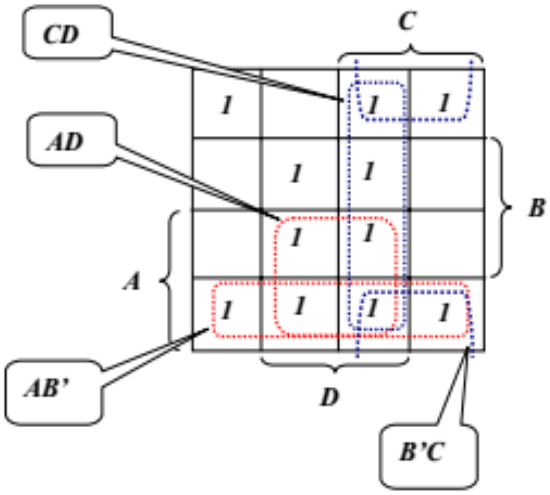
If a minterm is covered only by a prime implicant, then this prime implicant is called an essential prime implicant.

Example

Consider the function :

$$F(A,B,C,D) = \sum(0,2,3,5,7,8,9,10,11,13,15)$$





The two essential prime implicants are shown on the left and are:

BD and *B'D'*

There are also four prime implicants shown on the right. Only two of them can be used to cover the remaining three minterms 3, 9, and 11. These prime implicants are:

B'C and *CD*
Also, *AB'* and *AD*

The simplified function can take one of four different forms:

$$F = BD + B'D' + CD + AD$$

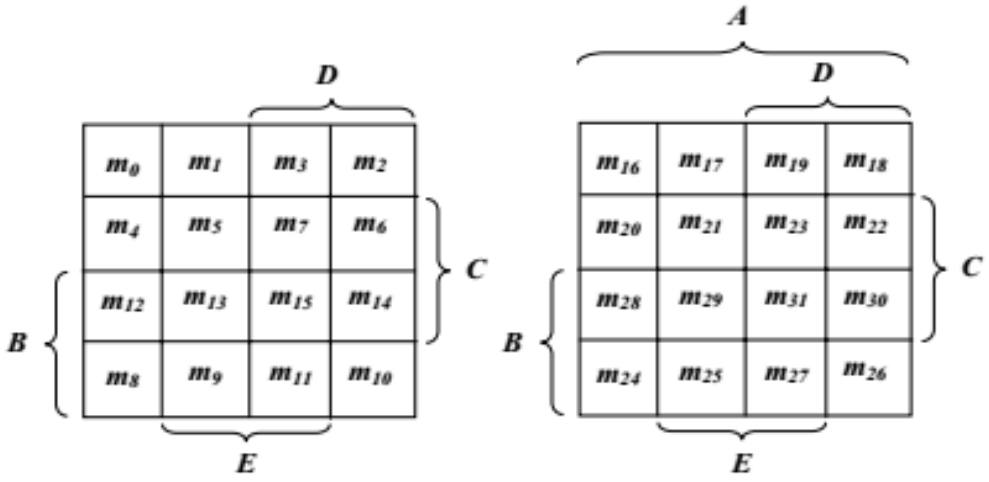
$$F = BD + B'D' + CD + AB'$$

$$F = BD + B'D' + B'C + AD$$

$$F = BD + B'D' + B'C + AB'$$

Five Variable Map

The map of five variables is shown below. It consists of two four variable maps.

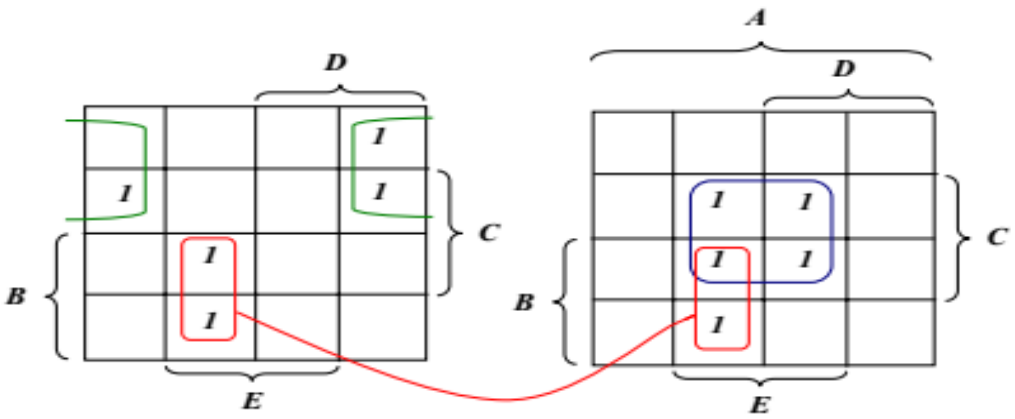


LECTURE 5: DIGITAL LOGIC CIRCUIT DESIGN

Example

Simplify the Boolean function

$$F(A,B,C,D,E)=\sum(0,2,4,6,9,13,21,23,25,29,31)$$



$$F(A,B,C,D,E)=A'B'E'+BD'E+ACE$$

Product of Sums (POS) Simplification

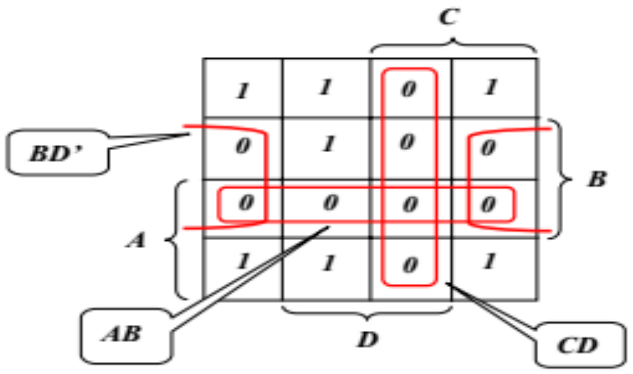
We combine the zeros of the function to obtain a simplified expression in a POS form. This follows from the fact that the zeros of the function are the ones of the complement of the function.

Example

Simplify the Boolean function

$$F(A,B,C,D)=\Sigma(0,1,2,5,8,9,10)$$

a. In SOP form b. In POS form.



LECTURE 5: DIGITAL LOGIC CIRCUIT DESIGN

Combining the ones of the function, we get:

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

The complement of the function can be expressed in SOP by combing the zeros of the function:

$$F' = AB + CD + BD'$$

We complement the last espression to get the function F in POS form,

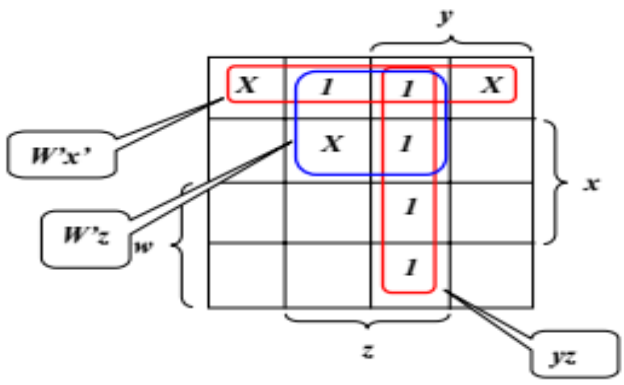
$$F = \left(AB + CD + BD'\right)' = (A' + B')(C' + D')(B' + D)$$

Don't care conditions

When we design combinational logic circuits, we sometimes encounter situations where combinations of the input variables will never occur. In such cases, we can assume that these conditions can take on the value 1 or 0, whichever goins to give us a simpler expression. These conditions are indicated by letter X or D.

Example

Simplify the function $F(w,x,y,z) = \Sigma(1,3,7,11,15)$, which has the don't care conditions $d(w,x,y,z) = \Sigma(0,2,5)$



$$F = w'x' + w'z + yz$$

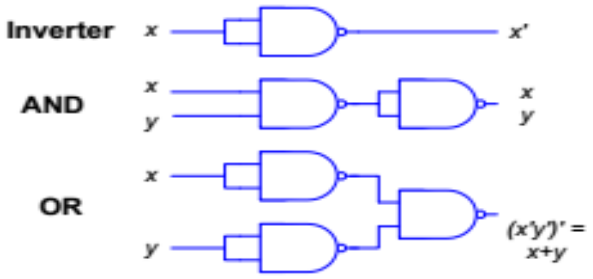
LECTURE 5: DIGITAL LOGIC CIRCUIT DESIGN

NAND and NOR Implementation

Digital circuits are frequently constructed with NAND or NOR gates rather than with AND or OR gates. NAND and NOR gates are easier to fabricate with electronic components and are the basic gates used in all IC digital logic families.

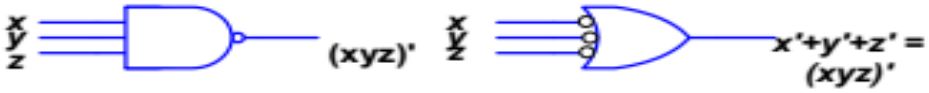
NAND Circuits

The basic AND, OR, and NOT gates can be implemented using NAND gates only.



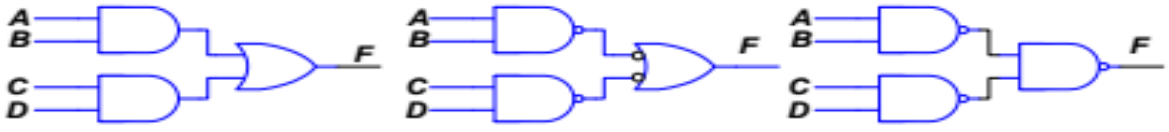
Logic Operations with NAND gates

Two equivalent graphic symbols for NAND gate are shown below:



Two Graphic Symbols for NAND gate

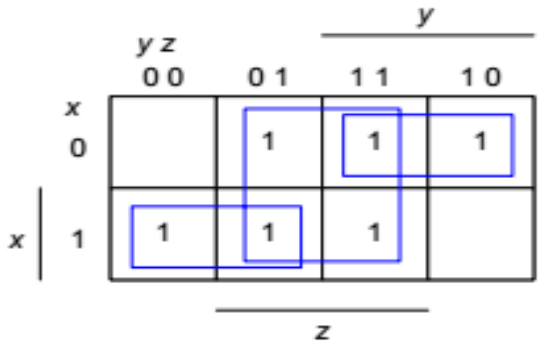
Example 1: **Implement F = AB + CD using NAND gates**

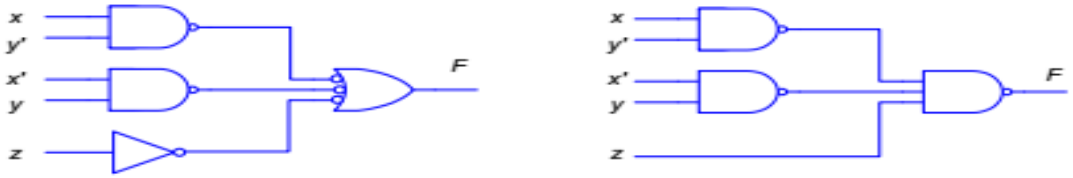


LECTURE 5: DIGITAL LOGIC CIRCUIT DESIGN

Example 2: Implement $F = \Sigma(1,2,3,4,5,7)$ using NAND gates

Simplification of the function gives F = *xy'+ x'y + z*

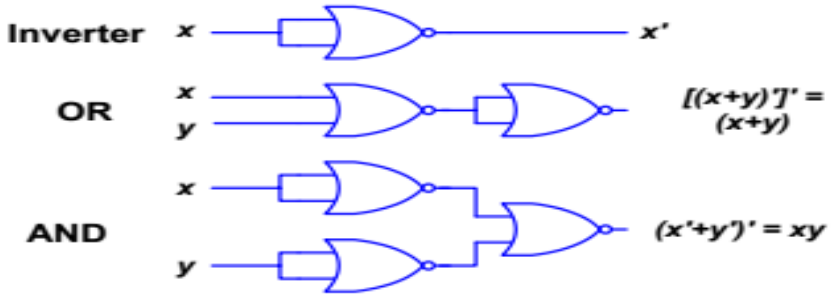




Two-level NAND implementation for F = xy'+ x'y + z

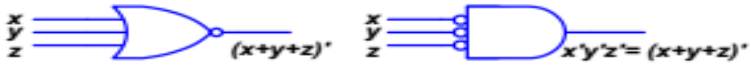
NOR Implementation

The basic AND, OR, and NOT gates can be implemented using NOR gates only:



Logic Operations with NOR gates

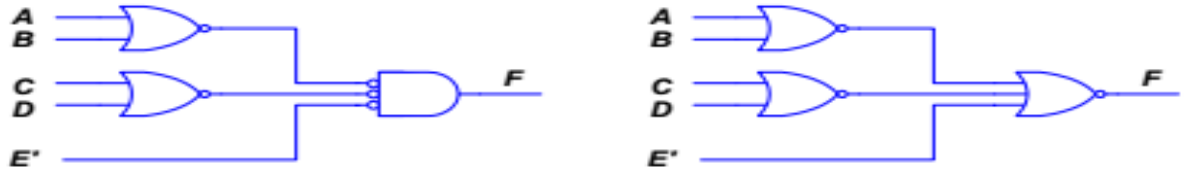
Two equivalent graphic symbols for the NOR gate, are shown below:



Two graphic symbols for NOR gate

LECTURE 5: DIGITAL LOGIC CIRCUIT DESIGN

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



OTHER TWO-LEVEL IMPLEMENTATIONS

Wired logic implements Boolean functions in other two-level forms. Examples are open collector TTL NAND gates and ECL NOR gates. The logic performed by the circuit in (a) is given by

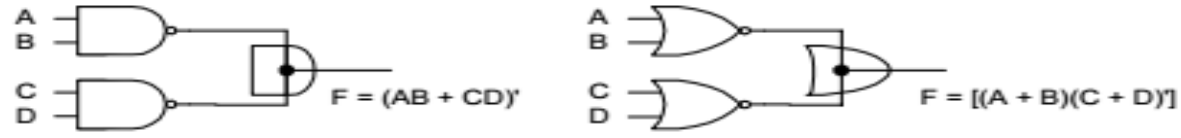
$$F = (AB)'.(CD)' = (AB + CD)'$$

and is called an AND-OR-INVERT function.

Similarly, the NOR output of ECL gates can be tied together to perform a wired-OR function. The logic performed by the circuit in (b) is given by

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

and is called an OR-AND-INVERT function.



(a) Wired-AND in open-collector TTL NAND gates
(AND-OR_INVERT)

(b) Wired-OR in ECL gates
(OR-AND-INVERT)

There are 16 possible arrangements of two level implementations. Eight are degenerate and will not be considered.

LECTURE 5: DIGITAL LOGIC CIRCUIT DESIGN

Nondegenerate Forms

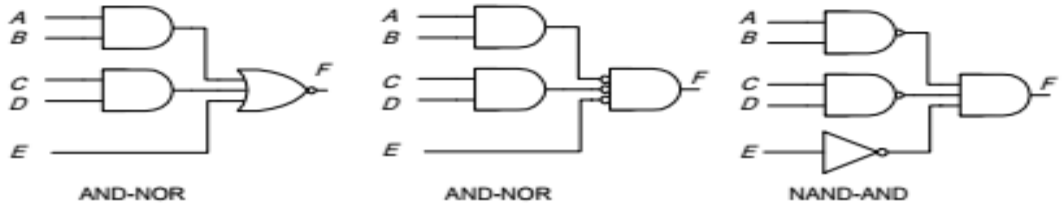
The eight nondegenerate forms are as follow:

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

The first gate listed in each of the forms constitutes a first level implementation. The second gate listed is a single gate placed in the second level. Note that any two forms listed in the same line are duals of each other. The first four forms listed above have been investigated previously. The remaining four forms are investigated next.

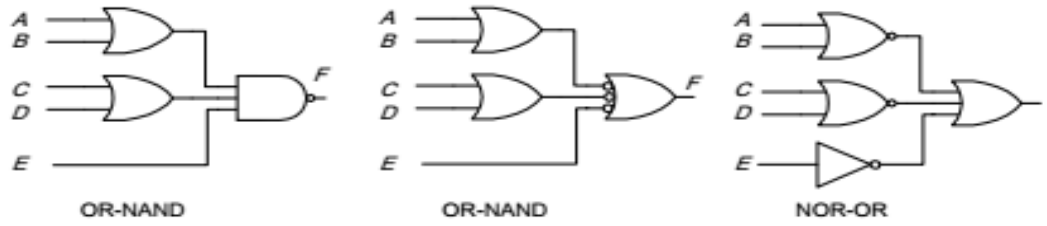
AND-OR-INVERT Implementation

The two forms NAND-AND and AND-NOR are equivalent forms and can be treated together. Both perform the AND-OR-INVERT function as shown in the circuit below which implements the function $F = (AB + CD + E)'$.



OR-AND-INVERT Implementation

The OR-NAND and NOR-OR forms perform the OR-AND-INVERT function. This is shown in the circuit below which implements $F = [(A + B)(C + D)E]'$.



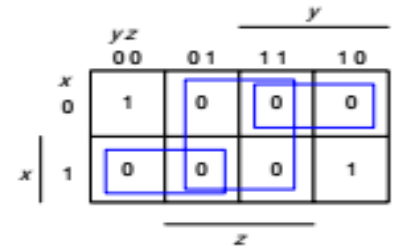
LECTURE 5: DIGITAL LOGIC CIRCUIT DESIGN

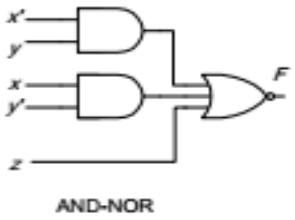
Example: Implement the function given by the map below with the four two-level forms discussed

Map simplification in SOP form

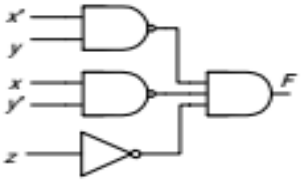
$F = xy'z' + xyz'$

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



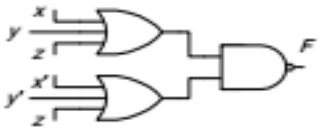


AND-NOR

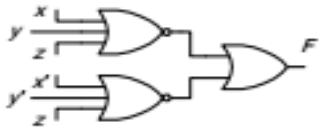


NAND-AND

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



OR-NAND



NOR-OR

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

above.