**Theory:**

In [digital circuit](http://en.wikipedia.org/wiki/Digital_circuit) theory, combinational logic is a type of logic circuit whose output is a [pure function](http://en.wikipedia.org/wiki/Pure_function) of the present input only. This is in contrast to [sequential logic](http://en.wikipedia.org/wiki/Sequential_logic), in which the output depends not only on the present input but also on the history of the input. In other words, sequential logic has [memory](http://en.wikipedia.org/wiki/Computer_storage) while combinational logic does not.

Combinational logic circuits are constructed with basic logic gates, such as: OR, And, Nand, Nor …etc. The design procedure of a combinational circuit passes through these stages:

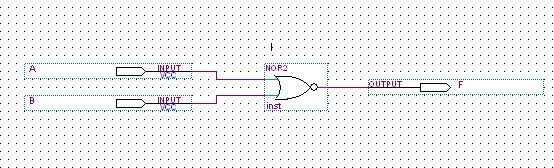
-first Writing the truth table of inputs and the corresponding output.

-secound Using the K-map method or the Q table method to minimize the expression of Maxterms or Minterms.

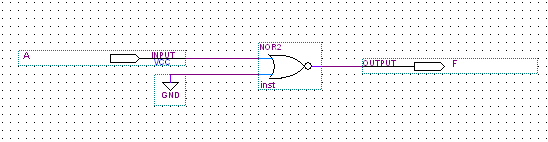
-and final Getting the final Boolean expression that we can implement.

**Procedure :**

1. **Nor- gate** :



* 1. –Nor gate as a NOT-gate :



block-c is used to build a Not- gate using the basic Nor gate

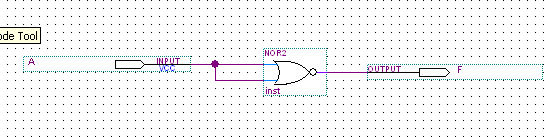
with input connected at the switch and output F2 ,that led to the following results :

When SW="0". F2= \_\_\_\_\_1\_\_\_\_\_\_\_\_\_\_

When SW="1", F2= \_\_\_\_\_0\_\_\_\_\_\_\_\_\_\_

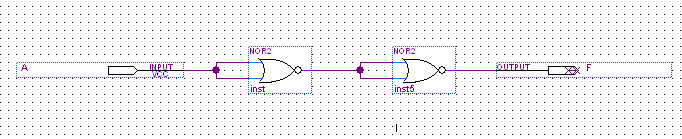
that verified the circuit acts like a Not-gate.

A second kind of connection can be used to implement a Not-gate:



This circuit is connected and the same results in the first part where found .

1.2 – Nor gates equivalent to buffer :

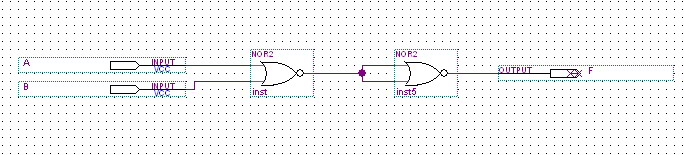
this circuit is connected with input at the switch and output at F2 ,this led to the following results :

When SW= "0", F4 =\_\_\_\_\_\_\_0\_\_\_\_\_\_

When SW= "1", F4 =\_\_\_\_\_\_\_1\_\_\_\_\_\_

that verified the circuit acts like a buffer.

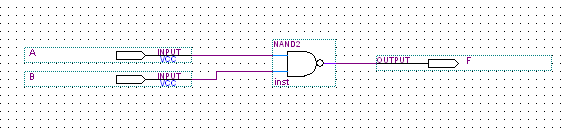
1.3- Nor-gates equivalent to OR-gate :



This circuit is connected and the following truth table is obtained :

|  |  |  |
| --- | --- | --- |
| **A** | **B** | **Output F** |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

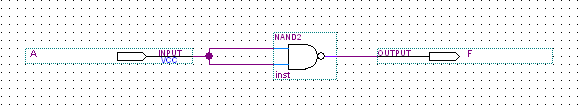
**2. Nand-gate :**



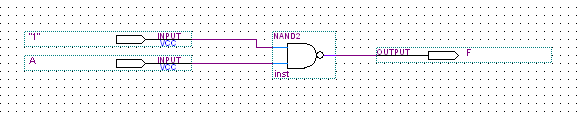
2.1- Nand-gate as a Not-gate :

If one of the two inputs of a Nand gate was connected to logic “1” then the gate will negate the other input (work as Not).Second way to do this is to connect the same input on the two pins of the Nand gate.

Both circuits are shown in Fig(1) and Fig(2):



Fig(1)



Fig(2)

block-c is used to build a Not- gate using the basic Nand- gate

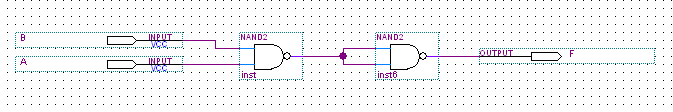
with input connected at the switch and output F2 ,which led to the following results :

When SW="0". F2= \_\_\_\_\_1\_\_\_\_\_\_\_\_\_\_

When SW="1", F2= \_\_\_\_\_0\_\_\_\_\_\_\_\_\_\_

that verified the circuit acts like a Not-gate.

2.2-Nand-gates equivalent to And-gate :

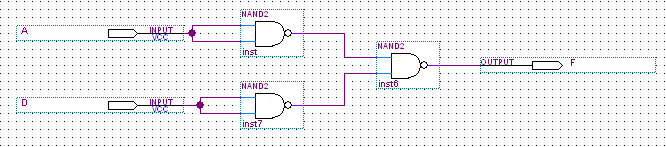


The inputs A and B is connected to a switches and the truth table is obtained :

|  |  |  |
| --- | --- | --- |
| **A** | **B** | **F4** |
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

2.3-Nand-gates equivalent to OR-gate :

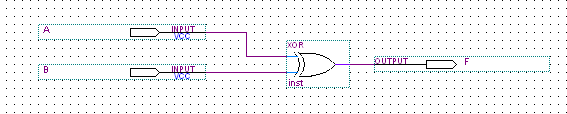
If we trace the following circuit, we see that from the first two gates we got A’ and B’ then we Nand them to get (A’.B’)’ as an output and by Demorganes theorem this term is equivalent to A+B so this composition of Nand’s works as an OR gate.



The circuit above is connected and atruth table is obtained :

|  |  |  |
| --- | --- | --- |
| **A** | **D** | **F4** |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

**3. Xor-gate :**

****

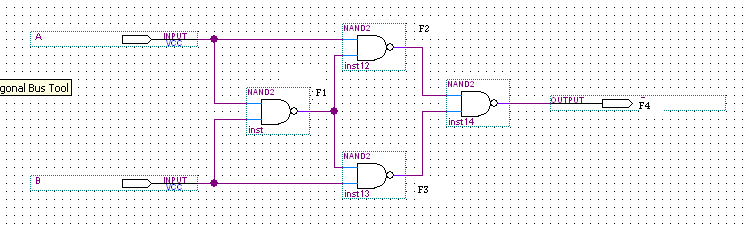
The Xor gate output is expressed by F=AB’+A’B

And this can be implemented using basic gates (And, Or, Not)

Or Nand /Nor gates can be used for this propose.

There is two ways to implement the Xor-gate:

3.1- Constructing XOR Gate with NAND Gates:

As shown in Fig(3) below : 

To explain how these gates acts like an Xor-gate consider Fig(4) below :



Fig(4)

it’s better to trace the inputs A,D.

First they are Nand together so F1= (A.D)’ ,then A is Nand with F1 and so is D.

Now A2= ((A.D)’.A)’ ,by Demorgan’s theorem it can be simplified to be A2=A.D +A’=(A+A’).(D+A’)

* A2=D+A’

On the other branch B2=(D+D’).(A+D’)

* B2=A+D’

Now,F4 = A2 Nand B2 =(( D+A’).(A+D’))’ ,by Demorgan’s

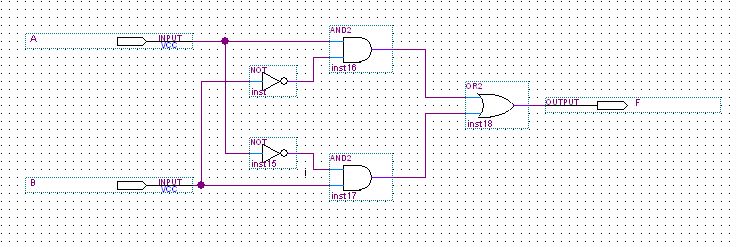
* F4= AD’+A’D = A Xor

And the truth table as shown below :

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **A** | **D** | **F4** | **F2** | **F3** | **F1** |
| 0 | 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 | 1 |
| 1 | 0 | 1 | 1 | 0 | 1 |
| 1 | 1 | 0 | 1 | 1 | 0 |

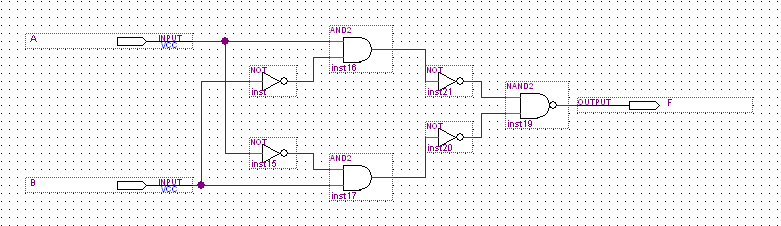
3.2- Constructing XOR Gate with basic Gates:

As shown in Fig(5) below :



Fig(5)

We can also replace the last OR-gate with NAND and tow NOT's as shown in Fig(6) below :



Fig(6)

this can be proved easily as shown :

Nand gate output = (A’.B’)’

= A+B ; by Demorgan’s law

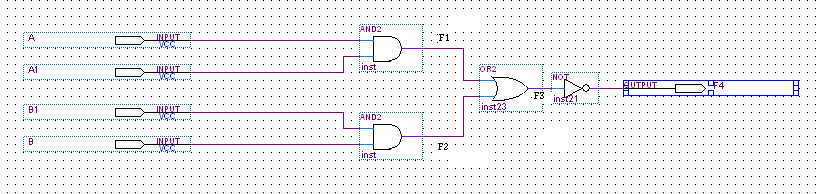
= OR- gate output .

And the truth table obtained as shown below :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| A | D | F1 | F2 | F3 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 0 |

**4. AOI gate**:

AND-OR-INVERTER (AOI) gates consist of two AND gates, one OR gate and one INVERTER (NOT) gate. The equivalent circuit as shown in Fig(7) :

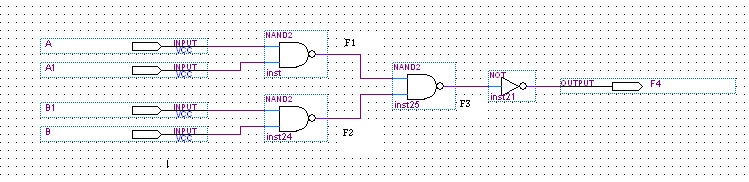


Fig(7)

The output F equals (AB+CD) ‘

Can also be written as (A’+B’) (C’+D’) .

And the actual circuit in Fig(8) :



Fig(8)

The truth table for AOI-circuit with active inputs A1 and A2 is obtained and listed below :

|  |  |  |  |
| --- | --- | --- | --- |
| **A** | **A1** | **F3** | **F4** |
| 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 |

The truth table for AOI-circuit with active inputs B1 and B2 is obtained and listed below :

|  |  |  |  |
| --- | --- | --- | --- |
| **B** | **B1** | **F3** | **F4** |
| 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 |

From Fig(8) above :

F3 = ((A.A1)’. (B.B1)’)’

= A.A1+B >>>> which is true .

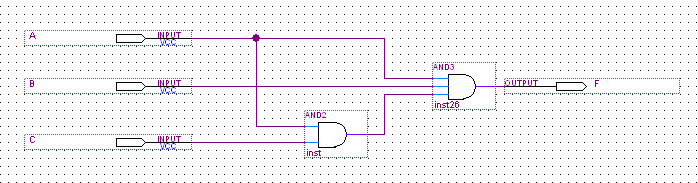
**Conclusion:**

The NAND and NOR gates are of great importance since any Boolean function can be expressed using NAND gates only or using NOR gates only, that’s why they are called the universal gates. this property of functional completeness has many advantages since NAND and NOR gates are less expensive and are constructed of fewer transistors compared with other gates.

**The logic problem :**

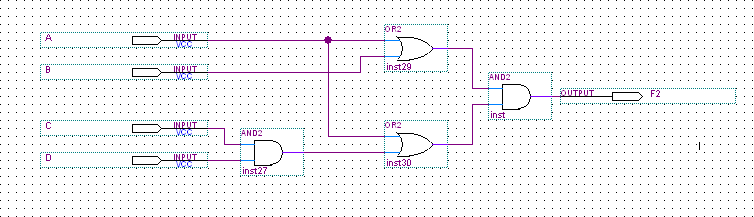
1. The logic diagram of the following Boolean equation using “AND” gates

F = AB (CA)

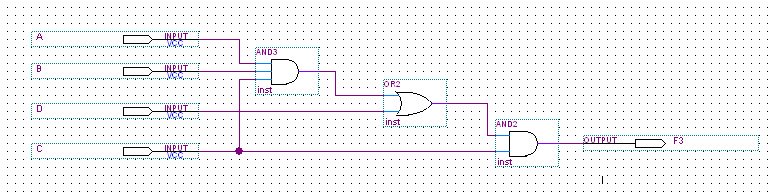


1. The logic Diagram of the following Boolean equations

F2 = (A+B) (CD+A)



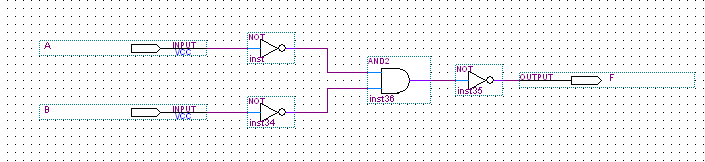
F3 = (ABC + D) C



1. The implementation of the OR gate using AND ,NOT gate.

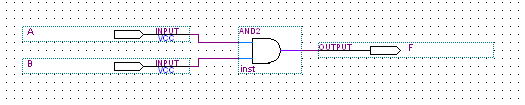
F = A+B = (A+B)’’

= ( A’. B’)’

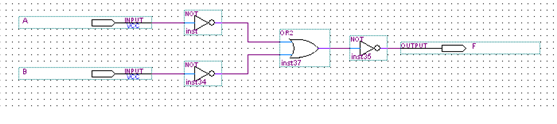


1. The implementation of AND gate using OR ,NOT gate .

F= A.B = (A.B)’’= (A’ + B’) ’



F = A . E



F=(A'+B')'

1. The equality operation F1=AB+A’B’ is the inverse of exclusive OR operation F2=AB’+A’B

F1’ = ( AB + A’B’)’

= ( AB )’. ( A’B’)’

= ( A’ + B’ ) . ( A + B

= A’A + A’B + B’A + B’B

= 0 + A’B + B’A + 0

= A’B + B’A = F2

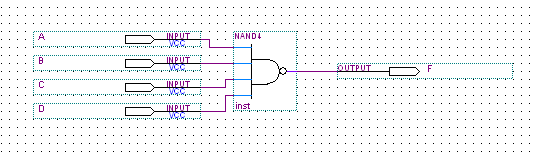
1. The implementation of four input NAND gate using two input NAND gates

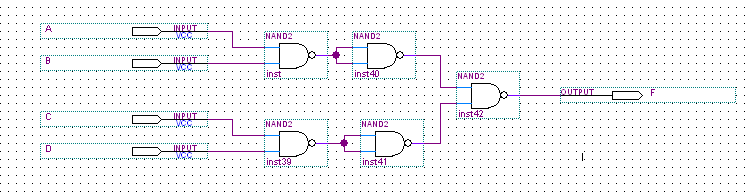
F = (ABCD)’ = (AB)’ + (CD)’

= ( (AB) + (CD)’ )’’

= ((AB)’’. (CD)’’)’

= NAND( ( NAND(A,B) )’, ( NAND(C,D))’)





1. Using the k- map to reduce the following boolean expression:

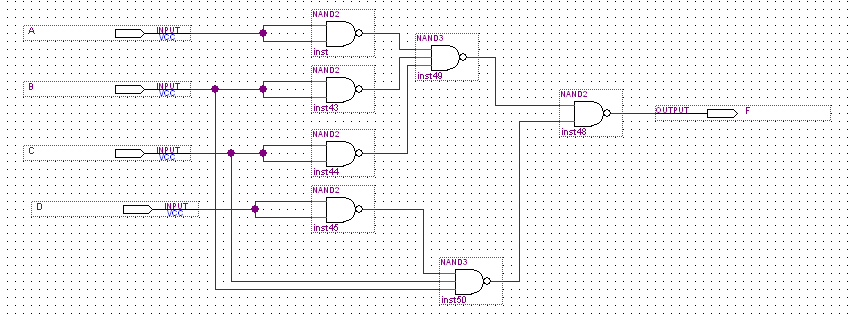
F1= A’BCD + ABCD’ + A’BCD’ + ABCD’ = A’B’C’ + BCD’

C

|  |  |  |  |
| --- | --- | --- | --- |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 |

A

D



F2 = A’B’C’D’ + AB’CD’ + A’B’CD’ + A’BC’D’ = A’C’D’ + B’CD’

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 |

A

