

**Sara Sameeh**

**#1130501**

**ENCS 211**

**Instructor: Abdulsalam Sayyad**

**Section #2**

**Experiments 1 & 2**

**\*The aims of experiment:**

\*To get used to different logic gate such as: AND, OR, NOT, NAND, NOR, XOR, and their implement.

\*To conclude that NAND is universal gate by construct the other gates using NAND gate only.

\*To become familiar with truth table for many circuit and write their Boolean functions, then using k-map to minimize the functions as much as possible.

\*By using basic gates construct AOI circuits, and become familiar with breadboard.

\* To get used to comparator and construct it with basic gates and IC.

\* To understand the concept of half – full adder and subtractor, and construct those using basic gates and IC.

\* \*To complete the previous aims we used:

1. KL-22001 Basic Electricity Circuit Lab

2. KL-26001 Combinational Logic Circuit Experiment Module (1)

3. Breadboard

4. IC 7400 (2-input NAND)

5. KL-26002 Combinational Logic Circuit Experiment Module (2).

6. KL-26005 Combinational Logic Circuit Experiment Module (5).

**\* Theory:**

There are two type of logic circuit, combinational or sequential, combinational circuit constructed using basic logic gate, this type of circuit depend on the current input only, it doesn’t depend on the previous output or input (feedback), the basic component of any circuit is the input, gates and output, input and output may take two values only 1 or 0, about the gates there are many gates like AND, OR, NOT … etc. and the universal gate like NAND , NOR , they called universal because we can construct any gate using them.

Beside to different types of gates that using to construct circuits, there is also another circuit, for example comparator, half –full adder, half – full subtractor circuits.

\*Comparator circuits: this circuit need at least two input to compare between them, there are three possibility for the output, A=B, A<B, A>B.

\*half – full adder: these combinational circuits do the basic arithmetic operation (addition), if this circuit add two bit then its half adder , if it start to add three bit then its full adder .

\*half – full subtractor: computers can’t do subtraction operation, to solve this problem we using 2’s complement , to do it there is Two steps are required . First, the subtrahend is inverted to 1’s complement, i.e. a “1” to a “0” and a “0” to a “1”. Secondly, a “1” is added to the least significant bit of the subtrahend in 1’s complement, half subtractor perform subtraction without take previous subtraction output in consideration, but full subtractor use the output from the previous subtraction.

**\* Procedure:**

**\*NOR gate circuit:**

****

By Using NOR gate we implement NOT gate as following:

A3 connected to sw0, A4 to sw1 as input, F2 is output.

We notice that:

When sw1 = 0, F2 = 1

When sw1 = 1, F2 = 0

So the circuit acts as NOT gate

Also:

A3 = A4, A3 connected to sw0 as input, F2 is output

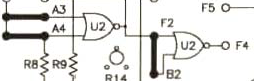
We notice that:

When sw0 = 0, F2 = 1

When sw0 = 1, F2 = 0

So the circuit acts as NOT gate

By using NOR gates, we construct Buffer as following:

A3 connected to sw0, F4 is output

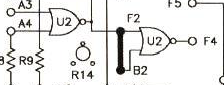
We notice that:

When sw0 = 0, F4 = 0

When sw0 = 1, F4 = 1

So the circuit acts as buffer

By using NOR gates, we implement OR gate as following:

A3 connected to sw0, A4 connected to sw1 as inputs, F4 is output

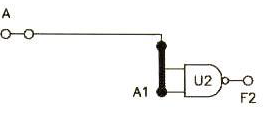
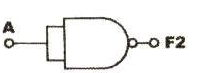
|  |  |  |
| --- | --- | --- |
| A3(sw0) | A4(sw1) | F4(output) |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

So the circuit acts like OR gate.

**\*NAND gate circuit:**

****

By using NAND gate we implements NOT gate as following:



A connected to sw1 as input, F2 is the output

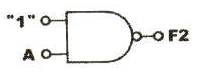
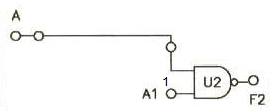
We notice that:

When sw1 = 0, F2 = 1

When sw1 = 1, F2 = 0

So the circuit acts as NOT gate

Also:

A1 is connected to +5v, A connected to sw1 as inputs, F2 is the output

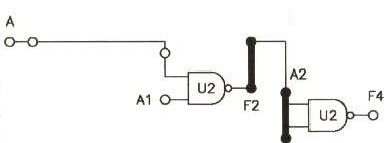
We notice that:

When sw1 = 0, F2 = 1

When sw1 = 1, F2 = 0

So the circuit acts as NOT gate

By using NANDs gates we implements AND gate as following:



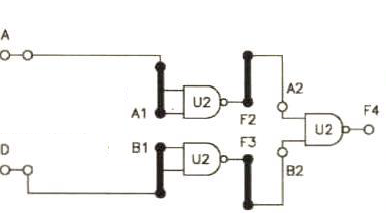
A is connected to sw1, A1 s connected to sw2 as output, F4 is the output

The result is:

|  |  |  |
| --- | --- | --- |
| Sw1(A) | Sw2(A1) | F4(output) |
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

So the circuit acts as AND gate.

By using NAND gates we construct OR gate as following:

 ****

A is connected to sw1, D is connected to sw2, F4 is the output.

We notice that:

|  |  |  |
| --- | --- | --- |
| Sw2(D) | Sw1(A) | Output(F4) |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

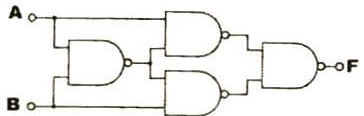
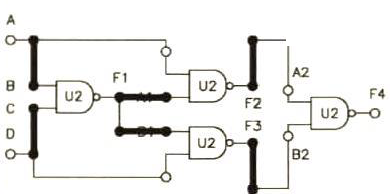
So the circuit acts as OR gate.

**\*XOR gate circuit:**

****

We can construct XOR gate using NOT, AND, OR, NOR or NAND gates.

By using NAND gates we construct XOR gate as following:

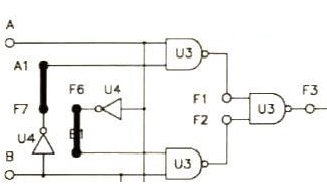
 

A is connected to sw1, D is connected to sw2 as inputs, F4 is the output.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Sw2(D) | Sw1(A) | F1 | F2 | F3 | F4(output) |
| 0 | 0 | 1 | 1 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 | 1 |
| 1 | 0 | 1 | 1 | 0 | 1 |
| 1 | 1 | 0 | 1 | 1 | 0 |

The circuit acts as XOR gate.

Using basic gates we construct XOR gate as following:

A is connected to sw1, B is connected to sw2 as inputs, F3 is the output

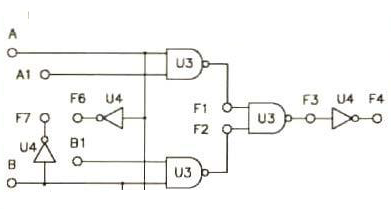
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Sw2(B) | Sw1(A) | F1 | F2 | F3(output) |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 0 |

So the circuit acts as XOR gate.

**\*AOI Gate Circuits**



The AOI consists of two AND gates, one OR gate, and one NOT gate.



A is connected to sw0, A1 to sw1, B to sw2, B1 to sw3, F3, f4 is output

B.B1 = 0

|  |  |  |  |
| --- | --- | --- | --- |
| Sw1(A1) | Sw0(A) | F3 | F4 |
| 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 |

F3 acts as AND gate when B.B1 =0.

F3 doesn’t as AND gate when B.B10.

When A.A1 =0

|  |  |  |  |
| --- | --- | --- | --- |
| Sw3(B1) | Sw2(B) | F3 | F4 |
| 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 |

F3 acts as AND gate when A.A1 =0.

F3 doesn’t as AND gate when A.A10.

F3 implements the function F3 = A.A1 + B.B1

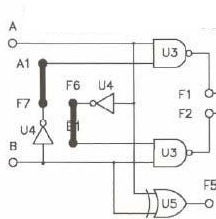
**\*Breadboard:**

****

**We check all the NAND gates on the breadboard, if we put one of the input 1 the NAND gate will acts as NOT gate , also if we connect the two input together the NAND gate will acts as NOT gate.**

**\*Comparator Circuits:**

Using basic logic gate we constructed comparator as following:

** **

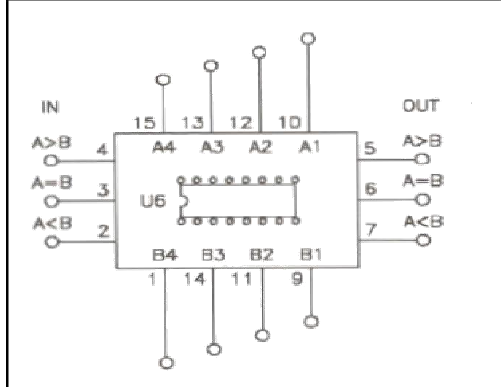
A connected to sw1, B connected to sw2 as inputs, F1, F2, and F5 the outputs.

\* Inputs active high.

The result was:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| B(sw2) | A(sw1) |  | F1 | F2 | F3 |
| 0 | 0 | A=B | 1 | 1 | 0 |
| 0 | 1 | A>B | 0 | 1 | 1 |
| 1 | 0 | A<B | 1 | 0 | 1 |
| 1 | 1 | A=B | 1 | 1 | 0 |

Using TTLC IC we constructed comparator as following:

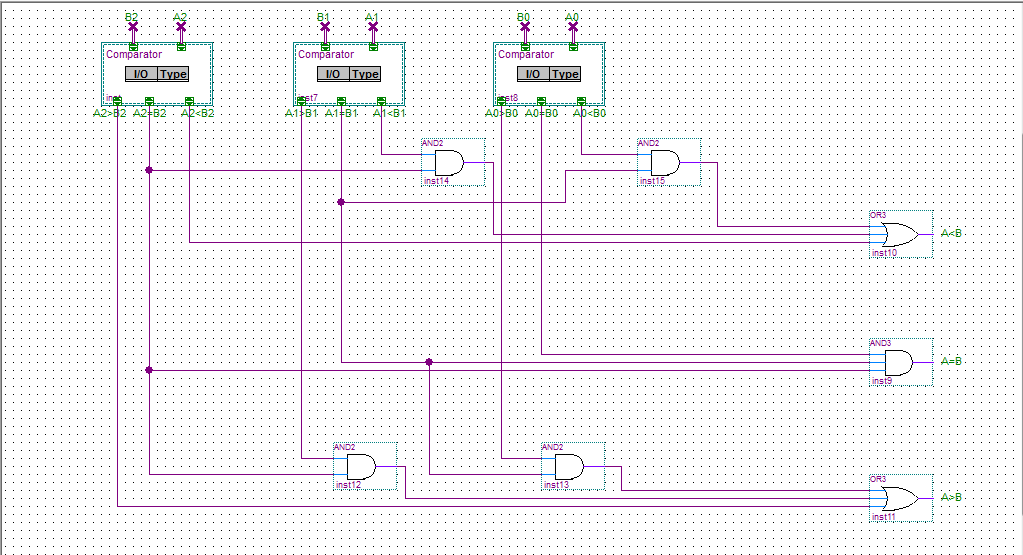


A1,A2,A3,A4,B1,B2,B3,B4 as inputs.

The result was:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| A4  (sw7) | A3  (sw6) | A2  (sw5) | A1  (sw4) | B4  (sw3) | B3  (sw2) | B2  (sw1) | B1  (sw0) | A>B | A<B | A=B |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 |
| 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 |

\* The design of three-bit comparator using the basic comparator:

****

**\* Half – and Full adder circuits:**

To know more about half and full adder read the following:

For half adder:

Truth table shown below

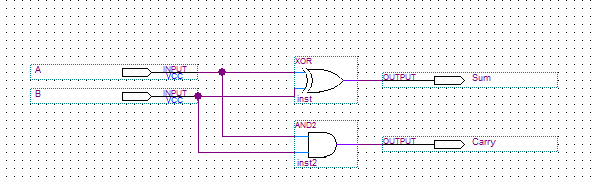
|  |  |  |  |
| --- | --- | --- | --- |
| A | B | Sum | Carry |
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |

Boolean function:

Sum = AB`+A`B = AB

Carry = AB

Logic diagram:

****

For full adder:

Truth table:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| A | B | C | Sum | Carry |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

Boolean function:

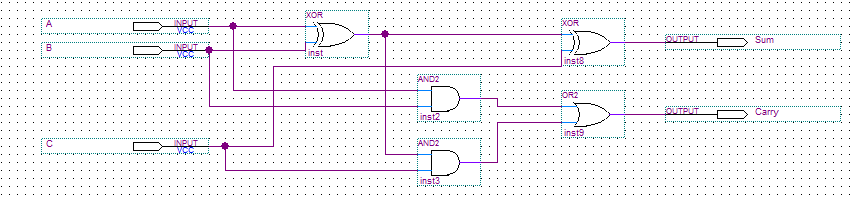
Sum = A`BC + A`BC`+ AB`C`+ABC = ABC

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

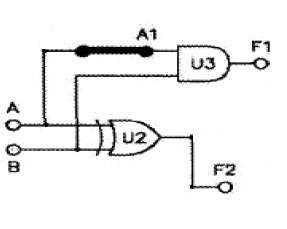
Carry= AB + BC + AC

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

Logic diagram:

****

Using basic logic gate we construct half adder as following:

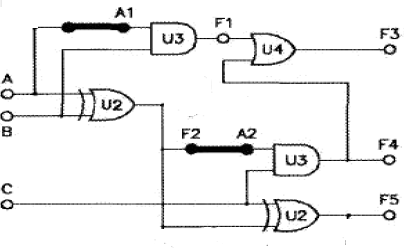
 

A connected to sw0, B connected to sw1 as inputs, F1,F2 the outputs.

The result is:

|  |  |  |  |
| --- | --- | --- | --- |
| Sw1(B) | Sw0(A) | Carry(F1) | Sum(F2) |
| 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 |

Using basic logic gates we constructed full adder as following:

A,B,C as inputs , F3, F5 the outputs.

The result is:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Sw3(C) | Sw2(B) | Sw1(A) | Carry(F3) | Sum(F5) |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 |

Using IC we construct FULL ADDER as following:



x0,x1,x2,x3,y0,y1,y2,y3 as inputs , ,,, ,F1 the outputs.

X= x3x2x1x0, Y = y3y2y1y0, =

The result is:

|  |  |  |  |
| --- | --- | --- | --- |
| Y | X |  | F1 |
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 0 | 6 | 6 | 0 |
| 0 | 9 | 9 | 0 |
| 0 | F | F | 0 |
| 1 | 3 | 4 | 0 |
| 1 | 6 | 7 | 0 |
| 1 | 8 | 9 | 0 |
| 3 | 6 | 9 | 0 |
| 4 | 8 | D | 0 |
| 4 | F | 3 | 1 |
| 8 | 7 | F | 0 |
| 9 | 9 | 2 | 1 |
| A | B | 5 | 1 |
| C | E | A | 1 |
| F | F | E | 1 |

**\*BCD Adder:**

Using IC we construct BCD adder as following:

X3 connected to sw3, x2 to sw2, x1 to sw1, x0 to sw0, Y3 to sw7, Y2 to sw6, Y1 to sw5, Y0 to sw4 as inputs, F1,F2,F3,F4,F5,F6,F7,F8,F9,F10,F11 the outputs.

The result is:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **X3** | **X2** | **X1** | **X0** | **Y3** | **Y2** | **Y1** | **Y0** | **F1** | **F11** | **F10** | **F9** | **F8** | **F2** | **F3** | **F7** | **F6** | **F5** | **F4** |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 |
| 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 1 |
| 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 |
| 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 |

****

**BCD ADDER**

**\*Half – and Full subtractor circuits:**

Read the following to know more about half and full subtractor:

Half – subtractor:

Truth table:

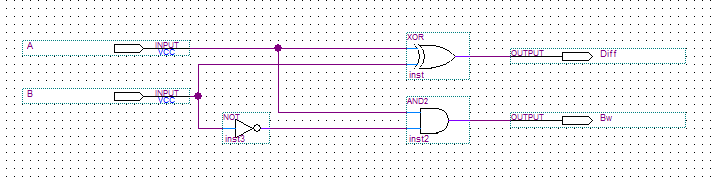
|  |  |  |  |
| --- | --- | --- | --- |
| A | B | Diff | Bw |
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 |

Boolean function:

Diff = A`B+AB` = AB

Bw = A`B

Logic diagram:

****

Full subtractor:

Truth table:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| A | B | C | Diff | Bw |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 |

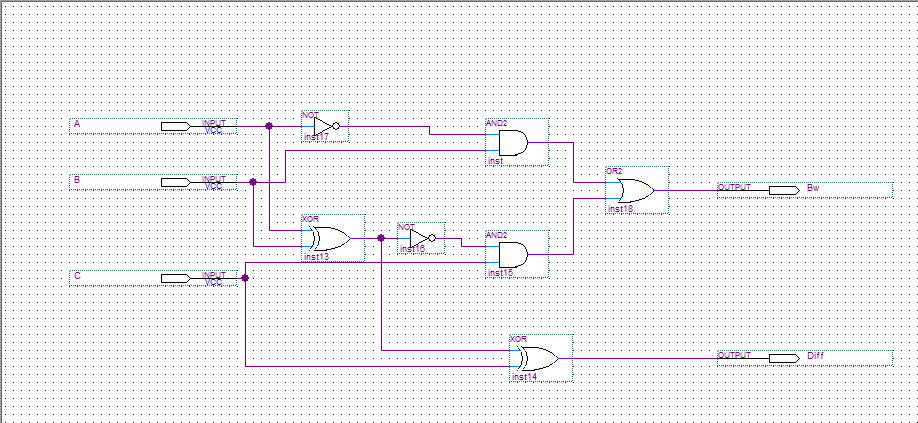
Boolean function:

Diff = A`B`C+A`BC`+AB`C`+ABC = (AB) C

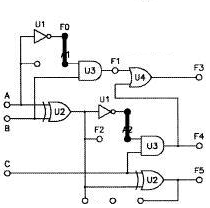
Bw= A`B+BC+A`C

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

Logic diagram:

****

Using basic logic gate we construct subtractor as following:



A connected to sw0, B connected to sw1, C connected to sw2 as inputs F1,F2,F3,F5 the outputs.

The result is:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| C | A | B | F1(BW1) | F2(DF1) | F3(BW2) | F5(DF2) |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 1 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 | 1 | 0 | 0 |
| 1 | 1 | 1 | 0 | 0 | 1 | 1 |

Note: when c =0, this circuit acts as half subtractor, with difference (F2) and borrow (F1), but when c =1, the circuit acts as full subtractor with difference (F5) and borrow (F3).

Using IC we construct full subtractor as following:



x3,x2,x1,x0,Y3,Y2,Y1,Y0 as inputs , F1,F11,F10,F9,F8 the outputs.

The result is:

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| X3  Sw7 | X2  Sw6 | X1  Sw5 | X0  Sw4 | Y3  Sw3 | Y2  Sw2 | Y1  Sw1 | Y0  Sw0 | Borrow  F1 | F11 | F10 | F9 | F8 |
| 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 |
| 1 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |

**\* Conclusion:**

From this experiment we conclude that if we want to construct any combinational circuit we need truth table, Boolean function and K-map , also we understand the importance of universal gate (NAND ,NOR)gates, because we can construct any circuit using NAND gate only , or NOR gate only , in addition to that, we realize how the comparator works , and construct it using gates and IC , we get used to how half –full adder and subtractor works , for example half adder and subtractor doesn’t take the previous output unlike full adder and subtractor ( borrow for subtractor, carry for adder ) , beside that we construct BCD adder , when the result is great than 9 we add 6 by using and gates. Finally we hope our result is correct.

**\*logic problems:**

**1- Truth table:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| A | B | C | D | F(output) |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 |

**2- (2,3,5,6,7,11,13,15)**

**3-** F = A’B’CD’ + A’B’CD + A’BC’D +A’BCD’+A’BCD +AB’CD+ABC’D+ABCD

F = A’B’C (D’+D) +A’BC (D+D’) +BC’D (A+A’) +ACD (B+B’)

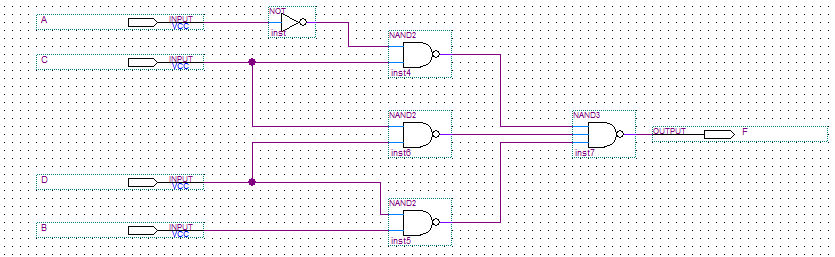
F = A’B’C (1) +A’BC (1) +BC’D (1) +ACD (1)

F = A’C + BC’D + ACD

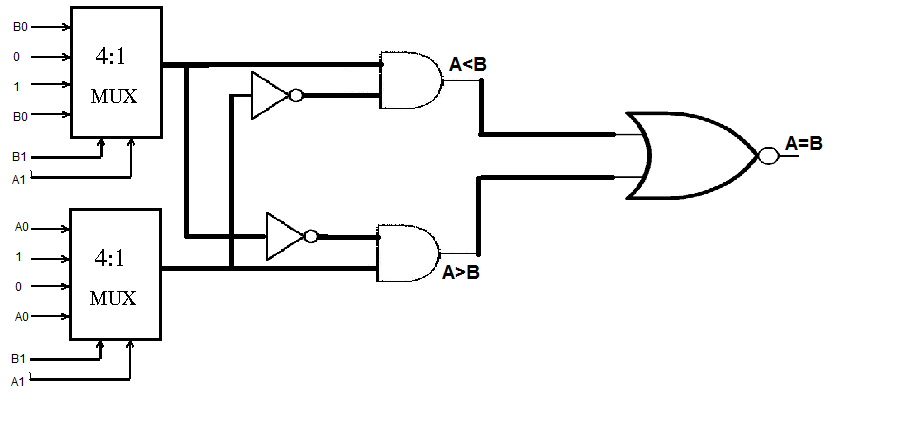
4-

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| AB/CD | 00 | 01 | 11 | 10 |
| 00 | 0 | 0 | 1 | 1 |
| 01 | 0 | 0 | 1 | 1 |
| 11 | 0 | 1 | 1 | 0 |
| 10 | 0 | 1 | 1 | 0 |

F = A’C + BD + CD

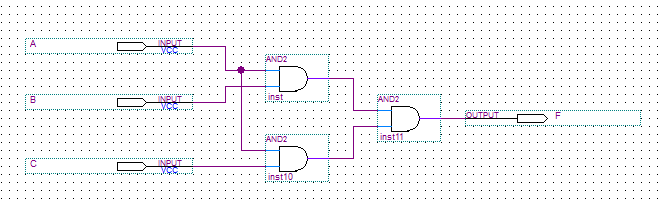
**5- **

**Problem from experment2:**

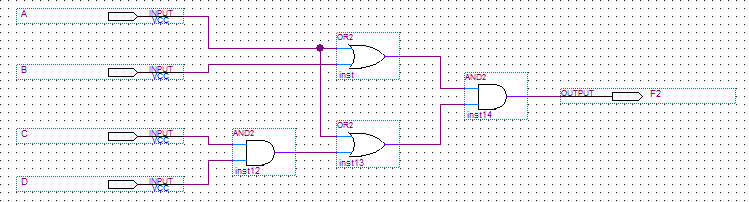
****

**\*post lab (experiment 1)**

**1- F= AB(CA)**

****

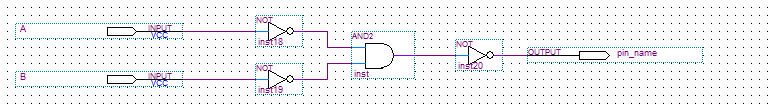
**2- F2 = (A+B).(CD +A)**

****

**F3 = (ABC +D) . C**

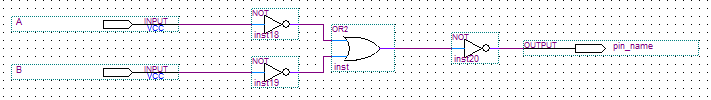
****

**3- Constructing OR gate using AND and NOT gates :**

****

**A+B = (A’ . B’)’**

**4- Constructing AND gate using OR and NOT gates:**

****

**AB = (A’+B’)’**

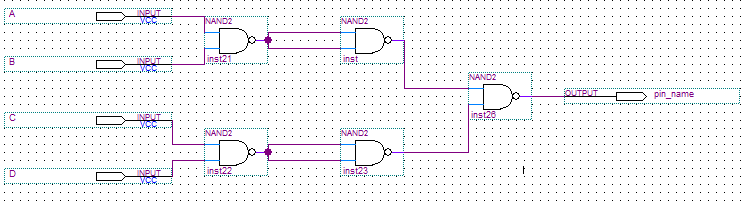
**5- F1 = AB + A’B’**

**= (AB + A’B’)’**

**= (A’+B’)(A+B)**

**= AA’+AB’+A’B+B’B**

**= AB’+A’B = F2**

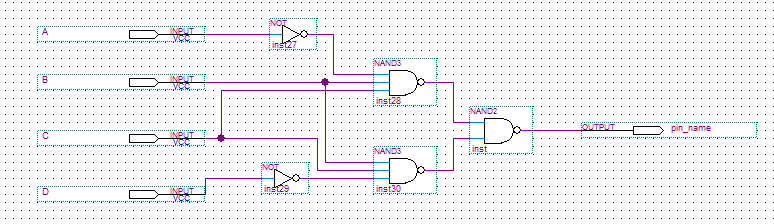
**6- **

**7- F = A’BCD + ABCD’+A’BCD’+ABCD’**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| AB/CD | 00 | 01 | 11 | 10 |
| 00 |  |  |  |  |
| 01 |  |  | 1 | 1 |
| 11 |  |  |  | 1 |
| 10 |  |  |  |  |

**F=** A’BC + BCD’

**Using NAND gate:**

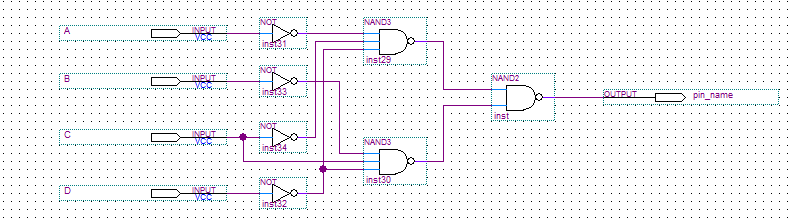
****

**F2 = A'B'C'D' + AB'CD' + A'B'CD' + A'BC'D'**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| AB/CD | 00 | 01 | 11 | 10 |
| 00 | 1 |  |  | 1 |
| 01 | 1 |  |  |  |
| 11 |  |  |  |  |
| 10 |  |  |  | 1 |

F2= A’C’D’ + B’CD’

Using NAND gate

****