

PROJECT NO. – 1

Developed by: Aashi Srivastava
National Institute of Technology, Warangal

AIM: To study basic logic gate functions and their truth tables.

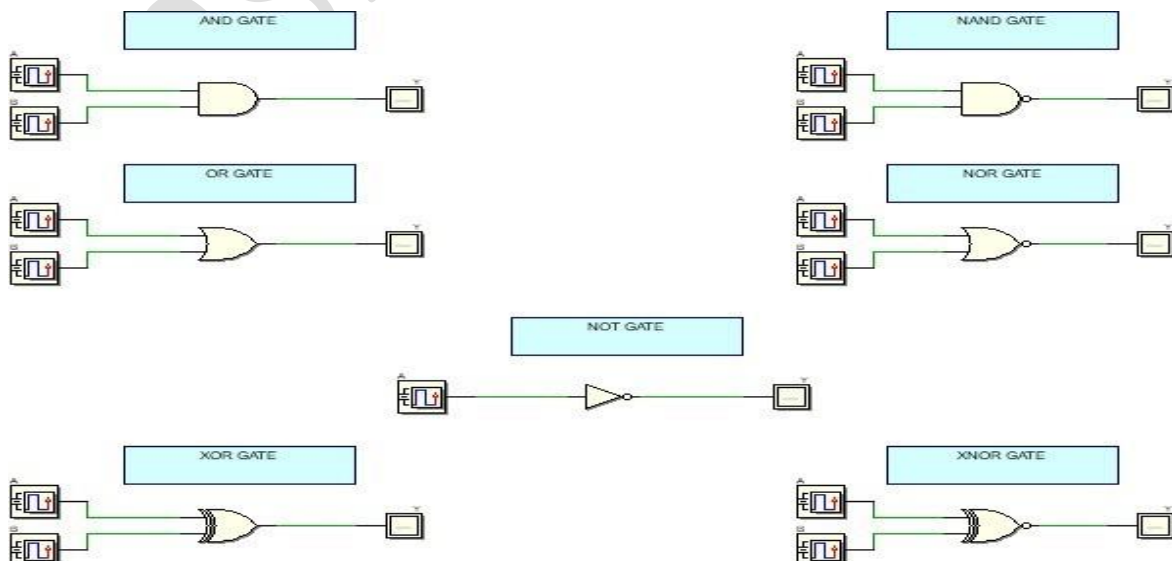
→ **DIGITAL LOGIC GATES:** The Digital Logic Gate is the basic building block from which all digital electronic circuits and microprocessor-based systems are constructed from. Basic digital logic gates perform logical operations of AND, OR and NOT on binary numbers.

In digital logic design only two voltage levels or states are allowed, and these states are generally referred to as Logic “1” and Logic “0”, or HIGH and LOW, or TRUE and FALSE. These two states are represented in Boolean Algebra and standard truth tables by the binary digits of “1” and “0” respectively.

A good example of a digital state is a simple light switch. The switch can be either “ON” or “OFF”, one state or the other, but not both at the same time. Then we can summarize the relationship between these various digital states as being:

Boolean algebra	Boolean Logic	Voltage State
Logic “1”	True(T)	High(H)
Logic “0”	False(F)	Low(L)

LOGIC GATES: 1. AND GATE 2. OR GATE 3. NOT GATE 4. NAND GATE
5. NOR GATE 6. X-OR GATE 7. X-NOR GATE



1. The NOT Function

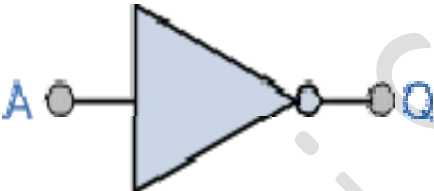
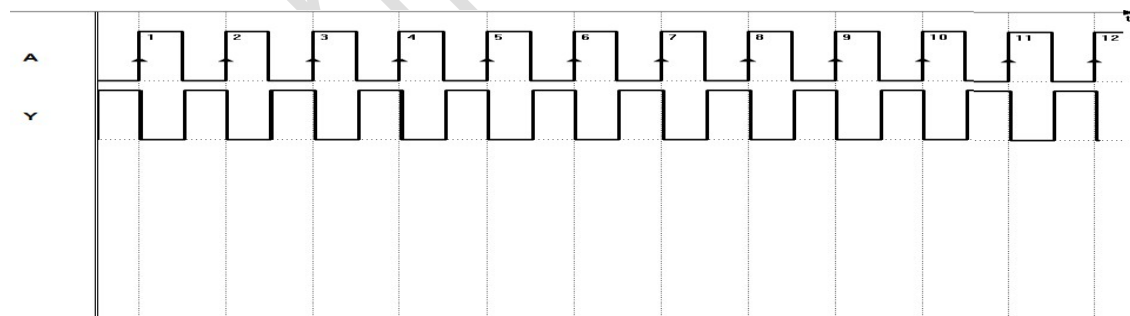
The NOT gate, which is also known as an “inverter” is given a symbol whose shape is that of a triangle pointing to the right with a circle at its end. This circle is known as an “inversion bubble”.

The NOT function is not a decision making logic gate like the AND, or OR gates, but instead is used to invert or complement a digital signal. In other words, its output state will always be the opposite of its input state.

The NOT gate symbol has a single input and a single output as shown.

The Logic NOT Gate

SYMBOL	TRUTH TABLE	
	A	Q
	0	1
	1	0

The symbol for a NOT gate is a light blue triangle pointing to the right, with a small circle (inversion bubble) at its tip. A blue input line labeled 'A' enters the left side of the triangle, and a blue output line labeled 'Q' exits from the bubble.

Verilog Code:

1. Using Continuous Assignment

```
// Developed by Aashi Srivastava
```

```
//TITLE: NOT GATE WITH CONTINUOUS ASSIGNMENT
// Date: 06.10.23, 11:10 IST

module notGate_CA (in , out); //module name assignment
    input in; // one-bit input
    output out;

    assign out=~in; // Continuous Assignment using assign. "Out" is net type
    while "in" is reg type.

endmodule
```

2. Using module instantiation

```
// Developed by Aashi Srivastava
//TITLE: NOT GATE WITH CONTINUOUS ASSIGNMENT
// Date: 06.10.23, 11:10 IST

module notGate_CA (in , out); //module name assignment
    input in; // one-bit input
    output out;

    assign out=~in; // Continuous Assignment using assign. "Out" is net type
    while "in" is reg type.

endmodule
```

3. 3.using primitives

```
/ Developed by Aashi Srivastava
//TITLE: NOT GATE INSTANTIATION
// Date: 06.10.23, 11:14 IST

primitive notGate_primitive (out,in); //primitive name assignment
input in; //one-bit input
output out;
table // truth table of not gate
// in:out
    0:1;
    1:0;
endtable
endprimitiv
```

Test-Bench:

```
// Developed by Aashi Srivastava
//TITLE: TEST BENCH FOR NOT GATE
// Date: 06.10.23, 11:21 IST
module not_tb (
);
    reg in; //input is declared as reg type
    wire out; //output is declared as net type

    notGate_CA N(in,out); // notgate module instantiation

    initial begin
        in=0;
    end

    initial begin //input signal generation
        repeat(20)
            #5 in=~in;

        end

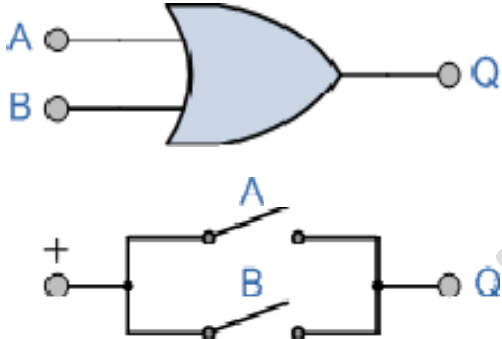
    initial begin
        $dumpfile("notGate_CA.vcd"); //for gtkwave opening
        $dumpvars(0,not_tb); // specifies to dump all the variables
        $monitor($time,"out=%b in=%b", out, in); //to monitor real-time change
        in variables
        #30 $finish;
    end

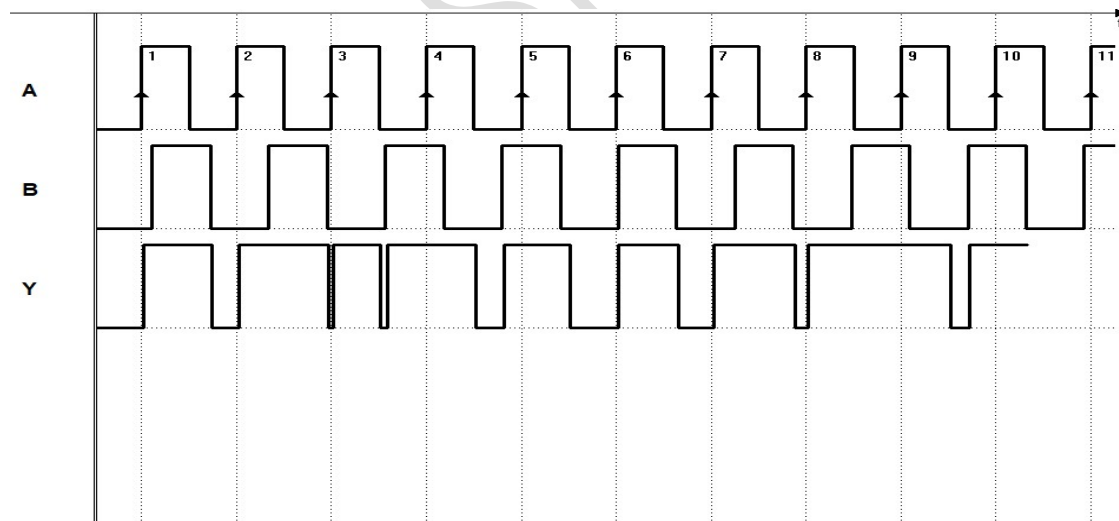
endmodule
```

1. The OR Function

In mathematics, the number or quantity obtained by adding two (or more) numbers together is called the sum. In Boolean Algebra the OR function is the equivalent of addition so its output state represents the addition of its inputs. In Boolean Algebra the OR function is represented by a “plus” sign (+) so for a two input OR gate the Boolean equation is given as: $Q = A+B$, that is Q equals either A OR B.

The 2-input Logic OR Gate

SYMBOL		TRUTH TABLE		
		A	B	Q
		0	0	0
		0	1	1
		1	0	1
		0	1	1



Verilog Code:

1. Using Continuous Assignment

```
// Developed by Aashi Srivastava
//TITLE: OR GATE WITH CONTINUOUS ASSIGNMENT
// Date: 06.10.23, 11:33 IST
module orGate_CA (in1 ,in2 ,out); //module name assignment
    input in1,in2; // one-bit inputs (input-1, input-2)
    output out;

    assign out=in1|in2; // Continuous Assignment using assign. "Out" is net
type while "in" is reg type.
endmodule
```

2. Using module instantiation

```
// Developed by Aashi Srivastava
//TITLE: OR GATE INSTANTIATION
// Date: 06.10.23, 11:14 IST

module orGate_Ins (in1, in2, out); //module name assignment
    input in1, in2; //one-bit inputs (input-1, input-2)
    output out;

    or N(out,in1, in2); //here or module (in-built) has been instantiated

endmodule
```

3. Using primitives

```
// Developed by Aashi Srivastava
//TITLE: or GATE primitive
// Date: 06.10.23, 11:14 IST
primitive orGate_primitive (out,a,b); //primitive name assignment
input a,b;
output out;
table // truth table of or gate
    //in1 in2:out
        0 0:0;
        0 1:1;
        1 0:1;
        1 1:1;
endtable
endprimitive
```

Test bench:

```
// Developed by Aashi Srivastava
//TITLE: TEST BENCH FOR OR GATE
// Date: 06.10.23, 11:21 IST

module orGate_tb();
    reg in1,in2; //input is declared as reg type
    wire out; //output is declared as net type

    orGate_primitive s(out,in1,in2); //calling primitive
    /* primitives are directly instantiated in the test-bench*/

    initial begin
        in1=0;
        in2=0;
        #1 in2=1;
        end

    initial begin //input signal generation
        repeat(20)
            #5 in1=~in1;
            #5 in2=~in2;
        end

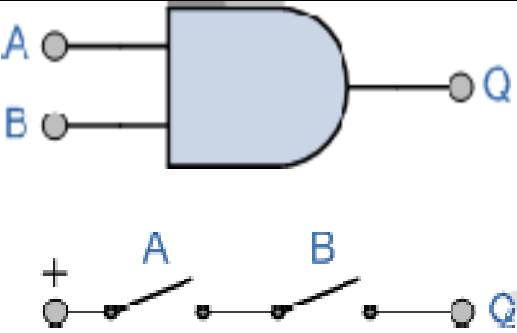
    initial begin
        $dumpfile("orGate_primitive.vcd"); //for gtkwave opening
        $dumpvars(0,orGate_tb); // specifies to dump all the variables
        $monitor($time,"out=%b in1=%b in2=%b",out,in1,in2); //to monitor real-
time change in variables
        #30 $finish;
    end

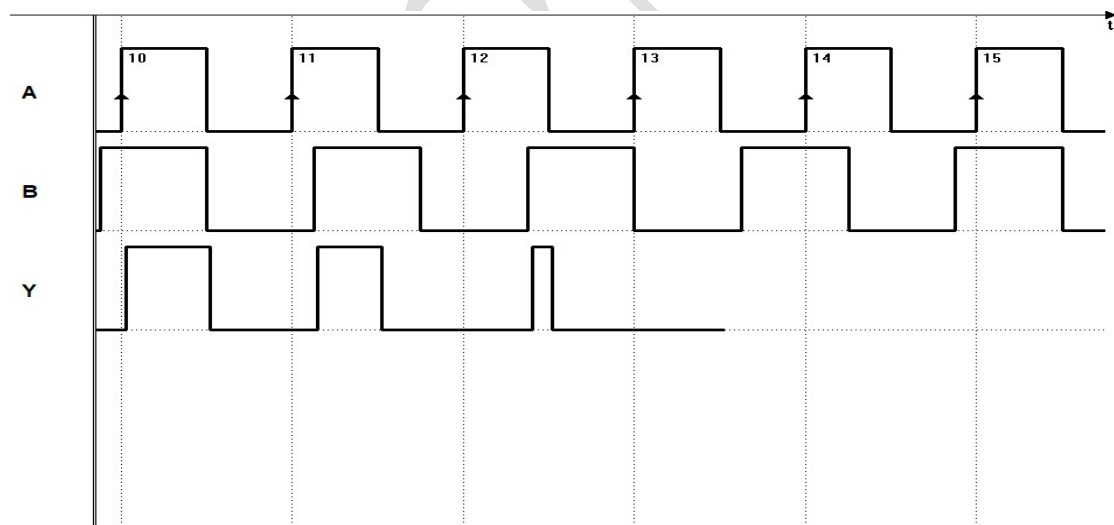
endmodule
```

1. The AND Function

In mathematics, the number or quantity obtained by multiplying two (or more) numbers together is called the product. In Boolean Algebra the AND function is the equivalent of multiplication and so its output state represents the product of its inputs. The AND function is represented in Boolean Algebra by a single "dot" (.) so for a two input AND gate the Boolean equation is given as: $Q = A.B$, that is Q equals both A AND B.

The 2-input Logic AND Gate

SYMBOL		TRUTH TABLE		
		A	B	Q
		0	0	0
		0	1	0
		1	0	0
		1	1	1



Verilog Code:

1. Using Continuous Assignment

```
// Developed by Aashi Srivastava
//TITLE: AND GATE WITH CONTINUOUS ASSIGNMENT
// Date: 06.10.23, 11:33 IST
module andGate_CA (in1 ,in2 ,out); //module name assignment
    input in1,in2; // one-bit inputs (input-1, input-2)
    output out;

    assign out=in1 & in2; // Continuous Assignment using assign. "Out" is
net type while "in" is reg type."&" operator is used.
endmodule
```

2. Using module instantiation

```
// Developed by Aashi Srivastava
//TITLE: OR GATE INSTANTIATION
// Date: 06.10.23, 11:14 IST

module andGate_Ins (in1, in2, out); //module name assignment
    input in1, in2; //one-bit inputs (input-1, input-2)
    output out;

    and N(out,in1, in2); //here or module (in-built) has been instantiated

endmodule
```

3. Using primitives

```
// Developed by Aashi Srivastava
//TITLE: AND GATE primitive
// Date: 06.10.23, 11:14 IST
primitive andGate_primitive (out,a,b); //primitive name assignment
input a,b;
output out;
table // truth table of and gate
    //in1 in2:out
        0 0:0;
        0 1:0;
        1 0:0;
        1 1:1;
endtable
endprimitive
```

Test-bench:

```
// Developed by Aashi Srivastava
//TITLE: TEST BENCH FOR AND GATE
// Date: 06.10.23, 11:21 IST

module andGate_tb();
    reg in1,in2; //input is declared as reg type
    wire out; //output is declared as net type

    andGate_primitive s(out,in1,in2); //calling primitive
    /* primitives are directly instantiated in the test-bench*/

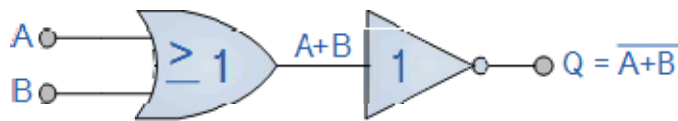
    initial begin
        in1=0;
        in2=0;
        #1 in2=1;
        end

    initial begin //input signal generation
        repeat(20)
            #5 in1=~in1;
            #5 in2=~in2;
        end

    initial begin
        $dumpfile("andGate_primitive.vcd"); //for gtkwave opening
        $dumpvars(0,andGate_tb); // specifies to dump all the variables
        $monitor($time,"out=%b in1=%b in2=%b",out,in1,in2); //to monitor real-
time change in variables
        #30 $finish;
    end

endmodule
```

1. THE NOR FUNCTION

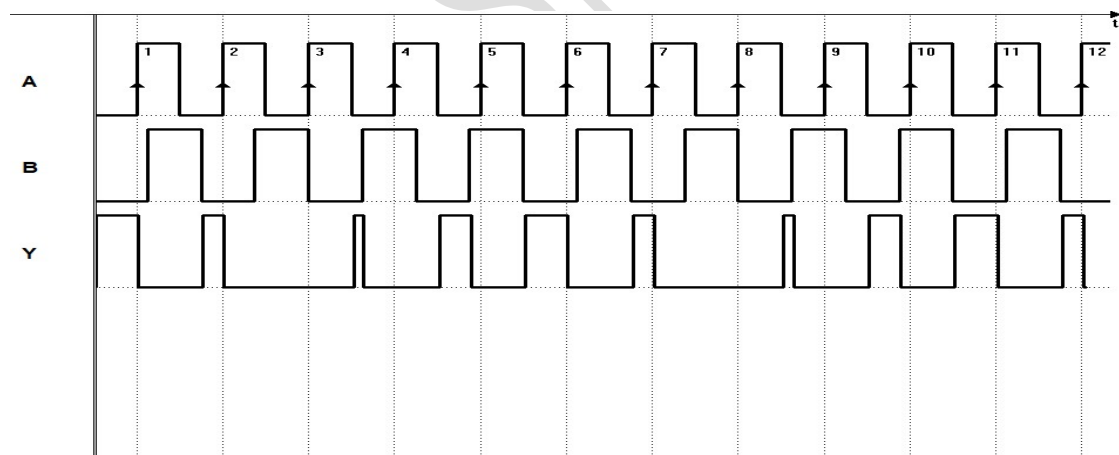


2-input "OR" gate plus a "NOT" gate

The logic or Boolean expression given for a logic NOR gate is that for Logical Multiplication which it performs on the complements of the inputs. The Boolean expression for a logic NOR gate is denoted by a plus sign, (+) with a line or Overline, ($\overline{\quad}$) over the expression signify the NOT or logical negation of the NOR gate giving us the Boolean expression of: $(A+B)' = Q$.

2-input NOR Gate

SYMBOL	TRUTH TABLE		
	A	B	Q
	0	0	1
	0	1	0
	1	0	0
	1	1	0



Verilog Code:

1. Using Continuous Assignment

```
// Developed by Aashi Srivastava
//TITLE: NOR GATE WITH CONTINUOUS ASSIGNMENT
// Date: 06.10.23, 11:33 IST
module norGate_CA (in1 ,in2 ,out); //module name assignment
    input in1,in2; // one-bit inputs (input-1, input-2)
    output out;

    assign out=~(in1 | in2); // Continuous Assignment using assign. "Out"
    is net type while "in" is reg type."~|" operator is used.
endmodule
```

2. Using module instantiation

```
// Developed by Aashi Srivastava
//TITLE: NOR GATE INSTANTIATION
// Date: 06.10.23, 11:14 IST

module norGate_Ins (in1, in2, out); //module name assignment
    input in1, in2; //one-bit inputs (input-1, input-2)
    output out;

    nor N(out,in1, in2); //here or module (in-built) has been instantiated
endmodule
```

3. Using primitives

```
// Developed by Aashi Srivastava
//TITLE: NOR GATE primitive
// Date: 06.10.23, 11:14 IST
primitive norGate_primitive (out,a,b); //primitive name assignment
input a,b;
output out;
table // truth table of nor gate
    //in1 in2:out
        0 0:1;
        0 1:0;
        1 0:0;
        1 1:0;
endtable
endprimitive
```

Test-bench:

```
// Developed by Aashi Srivastava
//TITLE: TEST BENCH FOR NOR GATE
// Date: 06.10.23, 11:21 IST

module norGate_tb();
    reg in1,in2; //input is declared as reg type
    wire out; //output is declared as net type

    norGate_primitive s(out,in1,in2); //calling primitive
    /* primitives are directly instantiated in the test-bench*/

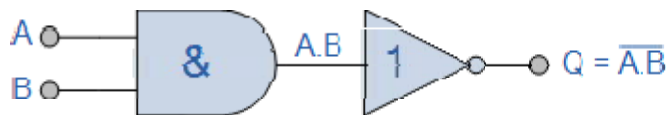
    initial begin
        in1=0;
        in2=0;
        #1 in2=1;
    end

    initial begin //input signal generation
        repeat(20)
            #5 in1=~in1;
            #5 in2=~in2;
    end

    initial begin
        $dumpfile("norGate_primitive.vcd"); //for gtkwave opening
        $dumpvars(0,norGate_tb); // specifies to dump all the variables
        $monitor($time,"out=%b in1=%b in2=%b",out,in1,in2); //to monitor real-
time change in variables
        #30 $finish;
    end

endmodule
```

1. THE NAND FUNCTION

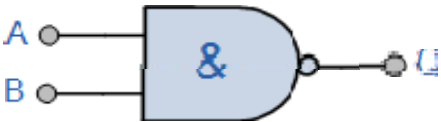


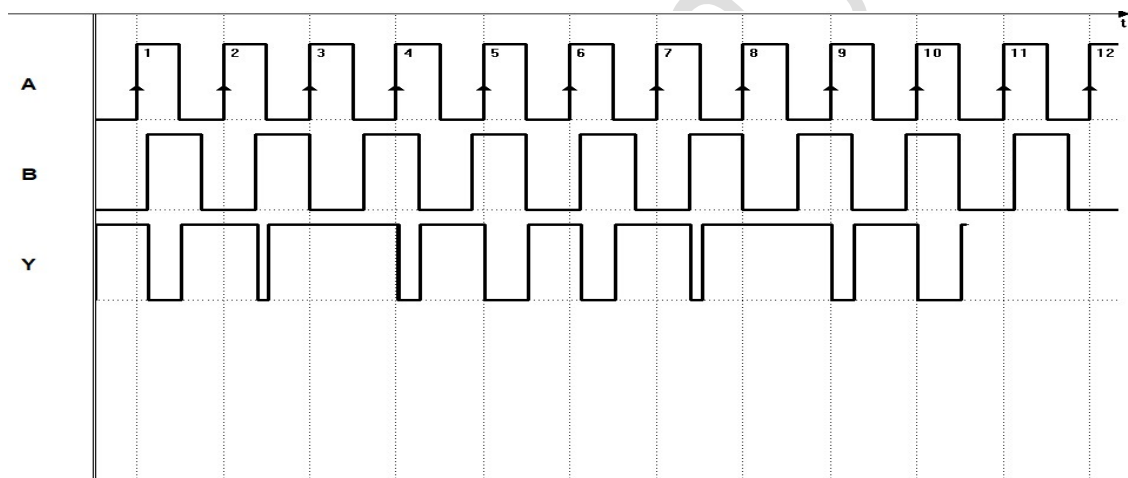
2-input AND gate plus a NOT gate

The logic or Boolean expression given for a logic NAND gate is that for Logical Addition,

which is the opposite to the AND gate, and which it performs on the complements of the inputs. The Boolean expression for a logic NAND gate is denoted by a single dot or full stop symbol, (.) with a line or Overline, ($\overline{\quad}$) over the expression to signify the NOT or logical negation of the NAND gate giving us the Boolean expression of: $(A.B)' = Q$.

2-input Logic NAND Gate

SYMBOL	TRUTH TABLE		
	A	B	Q
	0	0	1
	0	1	1
	1	0	1
	1	1	0



Verilog Code:

1. Using Continuous Assignment

```
// Developed by Aashi Srivastava
//TITLE: NAND GATE WITH CONTINUOUS ASSIGNMENT
// Date: 06.10.23, 11:33 IST
module nandGate_CA (in1 ,in2 ,out); //module name assignment
    input in1,in2; // one-bit inputs (input-1, input-2)
    output out;

    assign out=~(in1 & in2); // Continuous Assiggment using assign. "Out" is
net type while "in" is reg type."~&" operator is used.
endmodule
```

2. Using module instantiation

```
// Developed by Aashi Srivastava
//TITLE: NAND GATE INSTANTIATION
// Date: 06.10.23, 11:14 IST

module nandGate_Ins (in1, in2, out); //module name assignment
    input in1, in2; //one-bit inputs (input-1, input-2)
    output out;

    nand N(out,in1, in2); //here or module (in-built) has been instantiated
endmodule
```

3. Using primitives

```
// Developed by Aashi Srivastava
//TITLE: NAND GATE primitive
// Date: 06.10.23, 11:14 IST
primitive nandGate_primitive (out,a,b); //primitive name assignment
input a,b;
output out;
table // truth table of nand gate
    //in1 in2:out
        0 0:1;
        0 1:1;
        1 0:1;
        1 1:0;
endtable
endprimitive
```

Test-bench:

```
// Developed by Aashi Srivastava
//TITLE: TEST BENCH FOR NAND GATE
// Date: 06.10.23, 11:21 IST

module nandGate_tb();
    reg in1,in2; //input is declared as reg type
    wire out; //output is declared as net type

    nandGate_primitive s(out,in1,in2); //calling primitive
    /* primitives are directly instantiated in the test-bench*/

    initial begin
```

```
in1=0;
in2=0;
#1 in2=1;
end

initial begin //input signal generation
    repeat(20)
        #5 in1=~in1;
        #5 in2=~in2;
    end

    initial begin
        $dumpfile("nandGate_primitive.vcd"); //for gtkwave opening
        $dumpvars(0,nandGate_tb); // specifies to dump all the variables
        $monitor($time,"out=%b in1=%b in2=%b",out,in1,in2); //to monitor real-
time change in variables
        #30 $finish;
    end
end

endmodule
```

:

2. THE X-OR FUNCTION

The Exclusive-OR Gate function, or Ex-OR for short, is achieved by combining standard logic gates together to form more complex gate functions that are used extensively in building arithmetic logic circuits, computational logic comparators and error detection circuits.

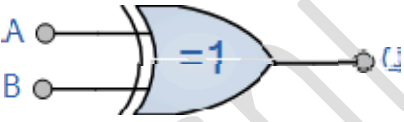
An logic output “1” is obtained when ONLY A = “1” or when ONLY B = “1” but NOT both together at the same time, giving the binary inputs of “01” or “10”, then the output will be “1”. This type of gate is known as an Exclusive-OR function or more commonly an Ex-Or function for short. This is because its boolean expression excludes the “OR BOTH” case of $Q = “1”$ when both A and B = “1”.

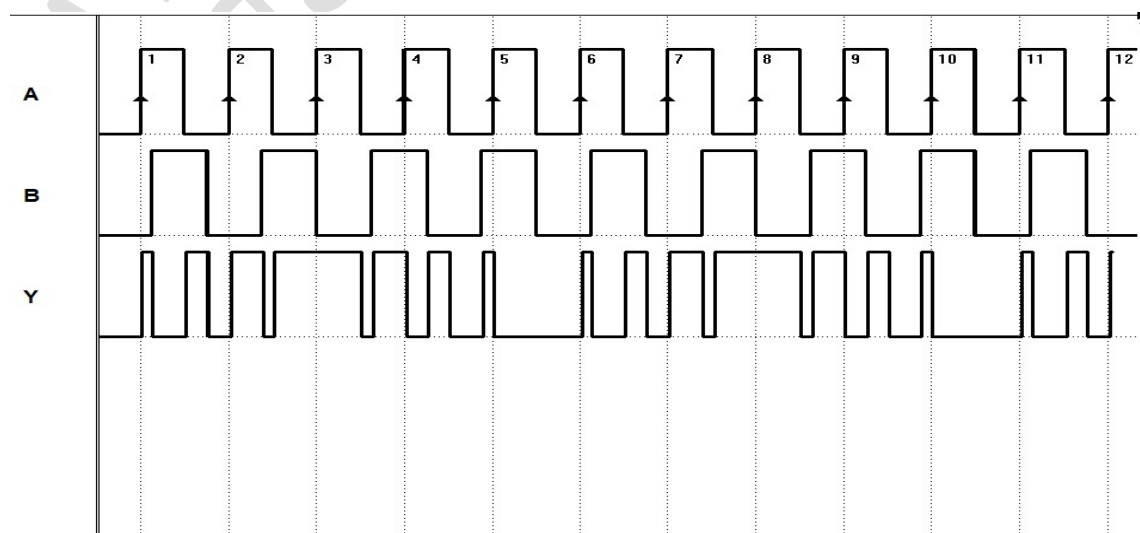
In other words the output of an Exclusive-OR gate ONLY goes “HIGH” when its two input terminals are at “DIFFERENT” logic levels with respect to each other.

An odd number of logic “1’s” on its inputs gives a logic “1” at the output. These two inputs can be at logic level “1” or at logic level “0” giving us the Boolean expression of:

$$Q = (A \oplus B) = A'.B + A.B'$$

2-input Ex-OR Gate

	TRUTH TABLE		
	A	B	Q
	0	0	0
	0	1	1
	1	0	1
	1	1	0



Verilog Code:

1. Using Continuous Assignment

```
// Developed by Aashi Srivastava
//TITLE: XOR GATE WITH CONTINUOUS ASSIGNMENT
// Date: 06.10.23, 11:33 IST
module xorGate_CA (in1 ,in2 ,out); //module name assignment
    input in1,in2; // one-bit inputs (input-1, input-2)
    output out;

    assign out=in1^in2; // Continuous Assignmnet using assign. "Out" is net
type while "in" is reg type.
endmodule
```

2. Using module instantiation

```
// Developed by Aashi Srivastava
//TITLE: XOR GATE INSTANTIATION
// Date: 06.10.23, 11:14 IST

module xorGate_Ins (in1, in2, out); //module name assignment
    input in1, in2; //one-bit inputs (input-1, input-2)
    output out;

    xor N(out,in1, in2); //here or module (in-built) has been instantiated

endmodule
```

3. Using primitives

```
// Developed by Aashi Srivastava
//TITLE: xor GATE primitive
// Date: 06.10.23, 11:14 IST
primitive xorGate_primitive (out,a,b); //primitive name assignment
input a,b;
output out;
table // truth table of xor gate
    //in1 in2:out
        0 0:0;
        0 1:1;
        1 0:1;
        1 1:0;
endtable
endprimitive
```

Test-bench:

```
// Developed by Aashi Srivastava
//TITLE: TEST BENCH FOR XOR GATE
// Date: 06.10.23, 11:21 IST

module xorGate_tb();
    reg in1,in2; //input is declared as reg type
    wire out; //output is declared as net type

    xorGate_primitive s(out,in1,in2); //calling primitive
    /* primitives are directly instantiated in the test-bench*/

    initial begin
        in1=0;
        in2=0;
        #1 in2=1;
    end

    initial begin //input signal generation
        repeat(20)
            #5 in1=~in1;
            #5 in2=~in2;
    end

    initial begin
        $dumpfile("xorGate_primitive.vcd"); //for gtkwave opening
        $dumpvars(0,xorGate_tb); // specifies to dump all the variables
        $monitor($time,"out=%b in1=%b in2=%b",out,in1,in2); //to monitor real-
time change in variables
        #30 $finish;
    end

endmodule
```

1. X-NOR FUNCTION



2-input "Ex-OR" gate plus a "NOT" gate

Basically the “Exclusive-NOR” gate is a combination of the Exclusive-OR gate and the NOT gate but has a truth table similar to the standard NOR gate in that it has an output that is normally at logic level “1” and goes “LOW” to logic level “0” when ANY of its inputs are at logic level “1”.

However, an output “1” is only obtained if BOTH of its inputs are at the same logic level, either binary “1” or “0”. For example, “00” or “11”. This input combination would then give us the Boolean expression of: $Q = (A \oplus B)' = (A.B)' + A.B$

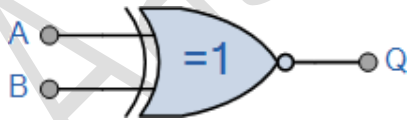
Then the output of a digital logic Exclusive-NOR gate ONLY goes “HIGH” when its two input terminals, A and B are at the “SAME” logic level which can be either at a logic level “1” or at a logic level “0”. In other words, an even number of logic “1’s” on its inputs gives a logic “1” at the output, otherwise is at logic level “0”.

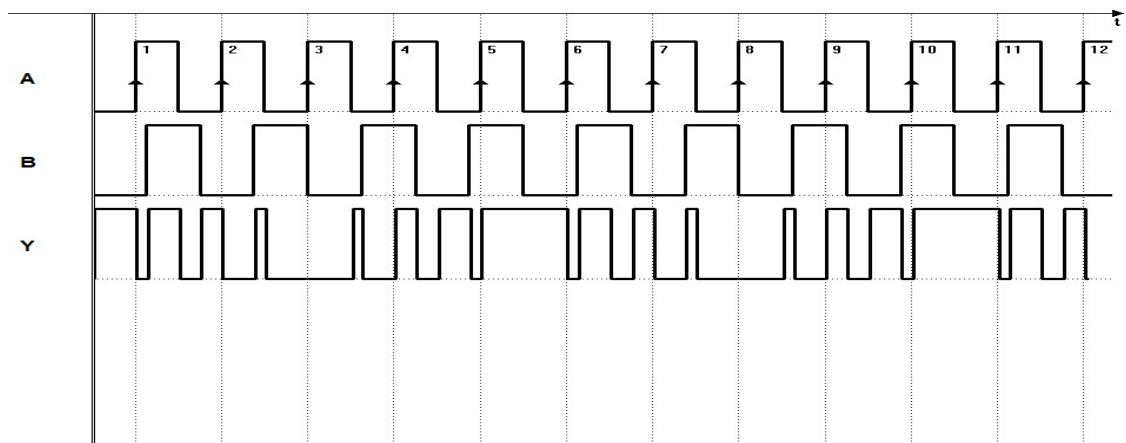
Then this type of gate gives an output “1” when its inputs are “logically equal” or “equivalent” to each other, which is why an Exclusive-NOR gate is sometimes called an Equivalence Gate.

The logic symbol for an Exclusive-NOR gate is simply an Exclusive-OR gate with a circle or “inversion bubble”, (o) at its output to represent the NOT function. Then the Logic Exclusive-NOR Gate is the reverse or “Complementary” form of the Exclusive-OR gate, $(A \oplus B)$ we have seen previously.

The Exclusive-NOR Gate, also written as: “Ex-NOR” or “XNOR”, function is achieved by combining standard gates together to form more complex gate functions and an example of a 2-input Exclusive-NOR gate is given below.

2-input Ex-NOR Gate

SYMBOL	TRUTH TABLE		
	A	B	Q
	0	0	1
	0	1	0
	1	0	0
	1	1	1



Aashi Srivastava

Verilog Code:

1. Using Continuous Assignment

```
// Developed by Aashi Srivastava
//TITLE: XNOR GATE WITH CONTINUOUS ASSIGNMENT
// Date: 06.10.23, 11:33 IST
module xnorGate_CA (in1 ,in2 ,out); //module name assignment
    input in1,in2; // one-bit inputs (input-1, input-2)
    output out;

    assign out=~(in1^in2); // Continuous Assignment using assign. "Out" is
net type while "in" is reg type.
endmodule
```

2. Using module instantiation

```
// Developed by Aashi Srivastava
//TITLE: XNOR GATE INSTANTIATION
// Date: 06.10.23, 11:14 IST

module xnorGate_Ins (in1, in2, out); //module name assignment
    input in1, in2; //one-bit inputs (input-1, input-2)
    output out;

    xnor N(out,in1, in2); //here or module (in-built) has been instantiated
endmodule
```

3. Using primitives

```
// Developed by Aashi Srivastava
//TITLE: xnor GATE primitive
// Date: 06.10.23, 11:14 IST
primitive xnorGate_primitive (out,a,b); //primitive name assignment
input a,b;
output out;
table // truth table of xnor gate
    //in1 in2:out
        0 0:1;
        0 1:0;
        1 0:0;
        1 1:1;
endtable
endprimitive
```

Test-bench:

```
// Developed by Aashi Srivastava
//TITLE: TEST BENCH FOR XNOR GATE
// Date: 06.10.23, 11:21 IST

module xnorGate_tb();
    reg in1,in2; //input is declared as reg type
    wire out; //output is declared as net type

    xnorGate_primitive s(out,in1,in2); //calling primitive
    /* primitives are directly instantiated in the test-bench*/

    initial begin
        in1=0;
        in2=0;
        #1 in2=1;
        end

    initial begin //input signal generation
        repeat(20)
            #5 in1=~in1;
            #5 in2=~in2;
        end

    initial begin
        $dumpfile("xnorGate_primitive.vcd"); //for gtkwave opening
        $dumpvars(0,xnorGate_tb); // specifies to dump all the variables
        $monitor($time,"out=%b in1=%b in2=%b",out,in1,in2); //to monitor real-
time change in variables
        #30 $finish;
    end

endmodule
```

Aashi Srivastava