# Basic Logic Gates

## Theory: -

A logic gate is a basic building block of a digital circuit that has two inputs and one output. Logic gates are used in microprocessors, microcontrollers, embedded system applications, and in electronic and electrical project circuits. The basic logic gates are categorized into seven: AND, OR, XOR, NAND, NOR, XNOR, and NOT. These logic gates with their logic gate symbols and truth tables are explained below.

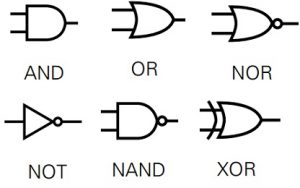


Basic Logic Gates Operation

The basic logic gates are classified into seven types:

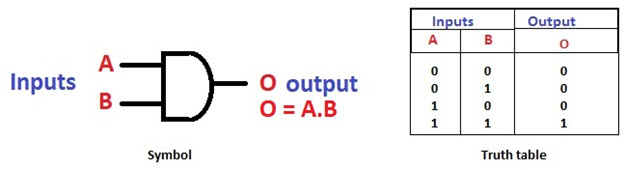
1. AND gate
2. OR gate
3. XOR gate
4. NAND gate
5. NOR gate
6. XNOR gate
7. NOT gate

The different types of logic gates and symbols with truth tables are discussed below.



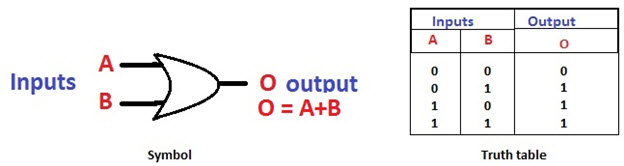
### Basic Logic Gates

#### AND Gate: -

The AND gate is a digital logic gate with ‘n’ i/ps one o/p, which performs logical conjunction based on the combinations of its inputs. The output of this gate is true only when all the inputs are true. When one or more inputs of the AND gate’s i/ps are false, then only the output of the AND gate is false. The symbol and truth table of an AND gate with two inputs is shown below.

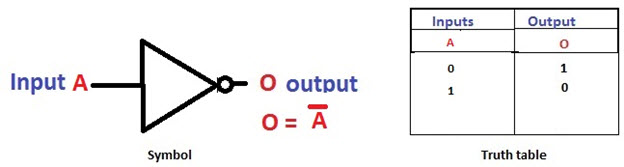
AND Gate and its Truth Table

#### OR Gate

The OR gate is a digital logic gate with ‘n’ i/ps and one o/p, that performs logical conjunction based on the combinations of its inputs. The output of the OR gate is true only when one or more inputs are true. If all the i/ps of the gate are false, then only the output of the OR gate is false. The symbol and truth table of an OR gate with two inputs is shown below.

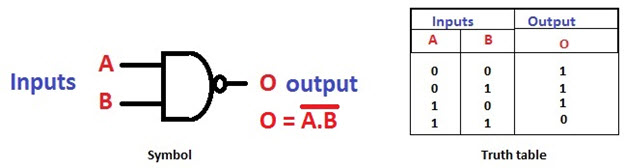
OR Gate and its Truth Table

#### NOT Gate

The NOT gate is a digital logic gate with one input and one output that operates an inverter operation of the input. The output of the NOT gate is the reverse of the input. When the input of the NOT gate is true then the output will be false and vice versa. The symbol and truth table of a NOT gate with one input is shown below. By using this gate, we can implement NOR and NAND gates

NOT Gate and Its Truth Table

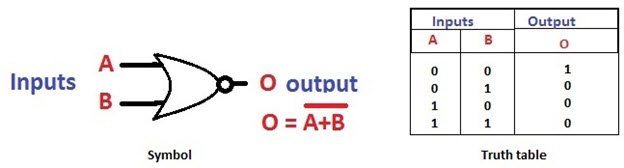
#### NAND Gate

The NAND gate is a digital logic gate with ‘n’ i/ps and one o/p, that performs the operation of the AND gate followed by the operation of the NOT gate. NAND gate is designed by combining the AND and NOT gates. If the input of the NAND gate high, then the output of the gate will be low. The symbol and truth table of the NAND gate with two inputs is shown below.

NAND Gate and its Truth Table

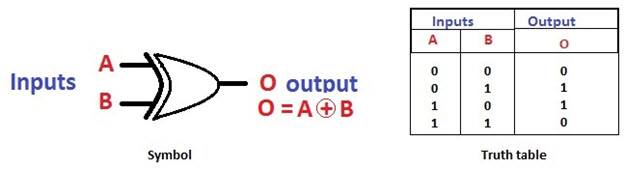
#### NOR Gate

The NOR gate is a digital logic gate with n inputs and one output, that performs the operation of the OR gate followed by the NOT gate. NOR gate is designed by combining the OR and NOT gate. When any one of the i/ps of the NOR gate is true, then the output of the NOR gate will be false. The symbol and truth table of the NOR gate with the truth table is shown below.



NOR Gate and Its Truth Table

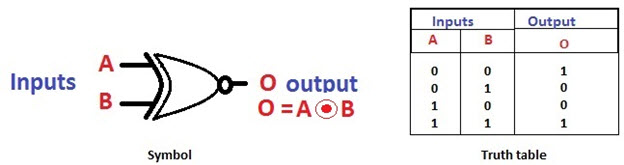
#### Exclusive-OR Gate

The Exclusive-OR gate is a digital logic gate with two inputs and one output. The short form of this gate is Ex-OR. It performs based on the operation of the OR gate. . If any one of the inputs of this gate is high, then the output of the EX-OR gate will be high. The symbol and truth table of the EX-OR are shown below.

EX-OR gate and Its Truth Table

#### Exclusive-NOR Gate

The Exclusive-NOR gate is a digital logic gate with two inputs and one output. The short form of this gate is Ex-NOR. It performs based on the operation of the NOR gate. When both the inputs of this gate are high, then the output of the EX-NOR gate will be high. But, if any one of the inputs is high (but not both), then the output will be low. The symbol and truth table of the EX-NOR are shown below.



EX-NOR Gate and Its Truth Table

The easiest way to learn the function of basic logic gates is explained below.

* For AND Gate – If both the inputs are high then the output is also high
* For OR Gate – If a minimum of one input is high then the output is High
* For XOR Gate – If the minimum one input is high then only the output is high
* NAND Gate – If the minimum one input is low then the output is high
* NOR Gate – If both the inputs are low then the output is high.

## Source Code

### And Gate Code

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

-- Uncomment the following library declaration if using

-- arithmetic functions with Signed or Unsigned values

--use IEEE.NUMERIC\_STD.ALL;

-- Uncomment the following library declaration if instantiating

-- any Xilinx leaf cells in this code.

--library UNISIM;

--use UNISIM.VComponents.all;

entity and\_gate is

Port ( a : in STD\_LOGIC;

b : in STD\_LOGIC;

c : out STD\_LOGIC);

end and\_gate;

architecture Behavioral of and\_gate is

begin

c<= a and b;

end Behavioral;

### NAND Gate Code

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

-- Uncomment the following library declaration if using

-- arithmetic functions with Signed or Unsigned values

--use IEEE.NUMERIC\_STD.ALL;

-- Uncomment the following library declaration if instantiating

-- any Xilinx leaf cells in this code.

--library UNISIM;

--use UNISIM.VComponents.all;

entity nand\_gate is

Port ( a : in STD\_LOGIC;

b : in STD\_LOGIC;

c : out STD\_LOGIC);

end nand\_gate;

architecture Behavioral of nand\_gate is

begin

c <= a nand b;

end Behavioral;

### XNOR Gate

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

-- Uncomment the following library declaration if using

-- arithmetic functions with Signed or Unsigned values

--use IEEE.NUMERIC\_STD.ALL;

-- Uncomment the following library declaration if instantiating

-- any Xilinx leaf cells in this code.

--library UNISIM;

--use UNISIM.VComponents.all;

entity xnor\_gate is

Port ( a : in STD\_LOGIC;

b : in STD\_LOGIC;

c : out STD\_LOGIC);

end xnor\_gate;

architecture Behavioral of xnor\_gate is

begin

c <= a xnor b;

end Behavioral;

## Testbench Code

### And Gate Code

library IEEE;

use IEEE.Std\_logic\_1164.all;

use IEEE.Numeric\_Std.all;

entity and\_gate\_tb is

end;

architecture bench of and\_gate\_tb is

component and\_gate

Port ( a : in STD\_LOGIC;

b : in STD\_LOGIC;

c : out STD\_LOGIC);

end component;

signal a: STD\_LOGIC;

signal b: STD\_LOGIC;

signal c: STD\_LOGIC;

begin

uut: and\_gate port map ( a => a,

b => b,

c => c );

stimulus: process

begin

-- Put initialisation code here

a<='0';

b<='0';

wait for 5ns;

a<='0';

b<='1';

wait for 5ns;

a<='1';

b<='0';

wait for 5ns;

a<='1';

b<='1';

wait for 5ns;

-- Put test bench stimulus code here

wait;

end process;

end;

### NAND Gate

use IEEE.Std\_logic\_1164.all;

use IEEE.Numeric\_Std.all;

entity nand\_gate\_tb is

end;

architecture bench of nand\_gate\_tb is

component nand\_gate

Port ( a : in STD\_LOGIC;

b : in STD\_LOGIC;

c : out STD\_LOGIC);

end component;

signal a: STD\_LOGIC;

signal b: STD\_LOGIC;

signal c: STD\_LOGIC;

begin

uut: nand\_gate port map ( a => a,

b => b,

c => c );

stimulus: process

begin

-- Put initialisation code here

a <= '0';

b <= '0';

wait for 10ns;

a <= '0';

b <= '1';

wait for 10ns;

a <= '1';

b <= '0';

wait for 10ns;

a <= '1';

b <= '1';

wait for 10ns;

-- Put test bench stimulus code here

wait;

end process;

end;

### XNOR Gate

library IEEE;

use IEEE.Std\_logic\_1164.all;

use IEEE.Numeric\_Std.all;

entity xnor\_gate\_tb is

end;

architecture bench of xnor\_gate\_tb is

component xnor\_gate

Port ( a : in STD\_LOGIC;

b : in STD\_LOGIC;

c : out STD\_LOGIC);

end component;

signal a: STD\_LOGIC;

signal b: STD\_LOGIC;

signal c: STD\_LOGIC;

begin

uut: xnor\_gate port map ( a => a,

b => b,

c => c );

stimulus: process

begin

-- Put initialisation code here

a <= '0' ;

b <= '0';

wait for 10ns;

a <= '0';

b <= '1';

wait for 10ns;

a <= '1';

b <= '0';

wait for 10ns;

a <= '1';

b <= '1';

wait for 10ns;

-- Put test bench stimulus code here

wait;

end process;

## Observation

### And Gate Waveform

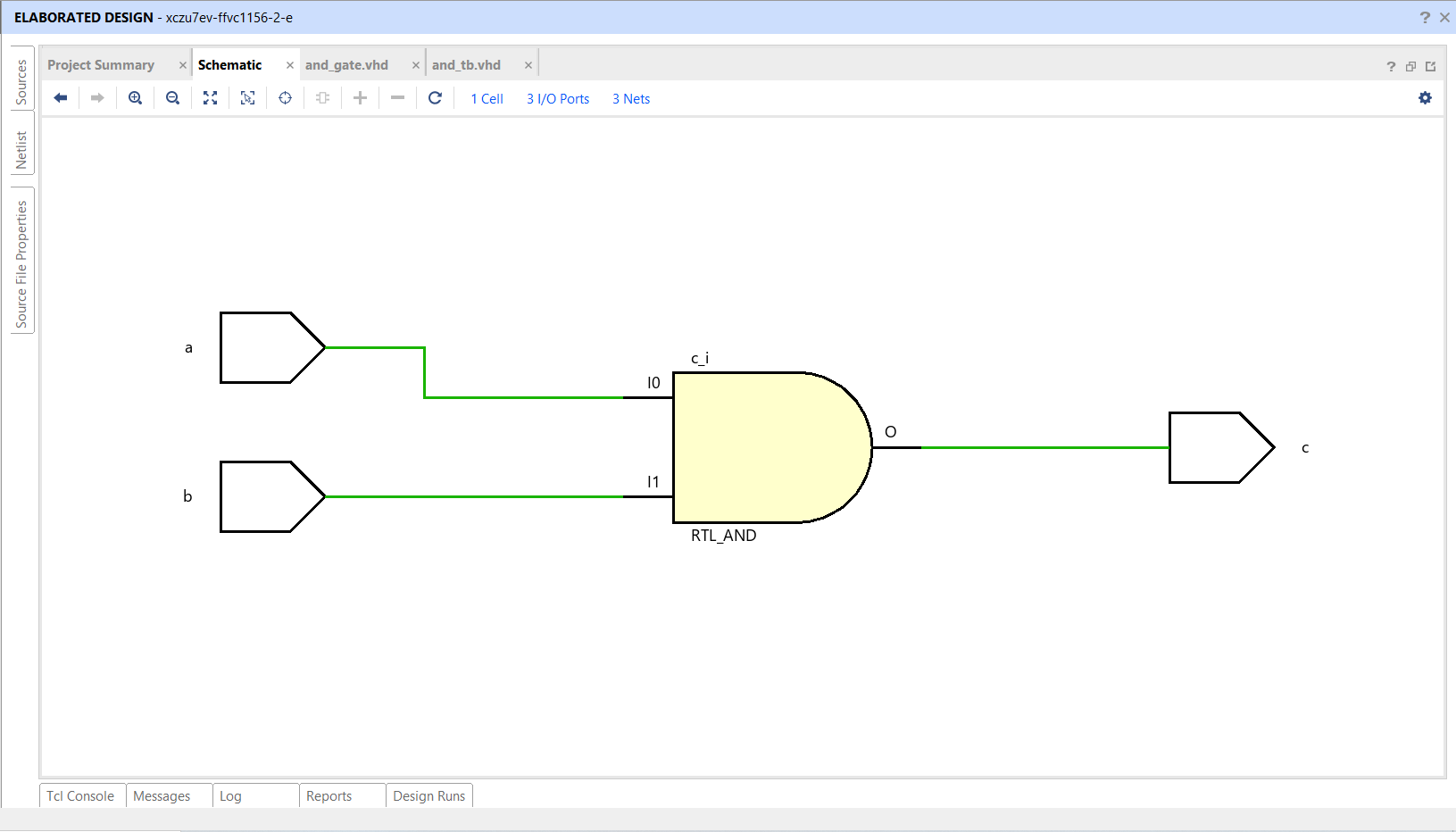
### NAND Gate Waveform

### XNOR Gate

### 

## Output

### And Gate



### NAND Gate

### 

### XNOR Gate

### 