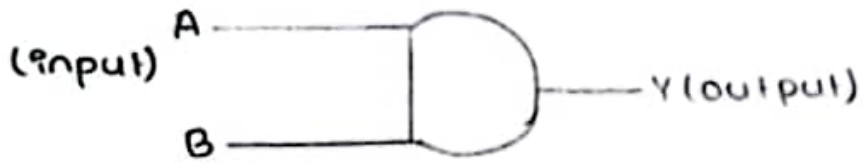
# Lab 1

## VHDL SIMULATION OF AND GATE

**Theory:**

AND gates are fundamental digital logic gate that outputs high(1) signal only when both of their output signals are high. Otherwise the output is zero(0)

**circuit diagram:**




**Logic expression:**

Output = A AND B

**Truth table:**

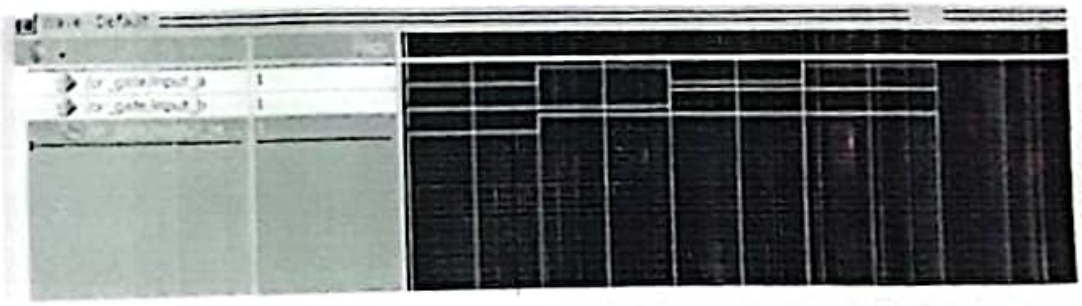
| A | B | output |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

## VHDL code

```
--AND gate implementation
   library IEE;
use IEEE.STD_LOGIC_1164.ALL;

   entity AND-GATE is
      Port (
         A: in STD_LOGRC;
         B: in STD_LOGIC;
         output:out STD_LOGIC;
         );
   end AND-GATE;
   architecture Behavioral of AND-GATE is
   begin
      output <=A AND B; --AND gate logic
   end Behavioral;
```

## Result:-

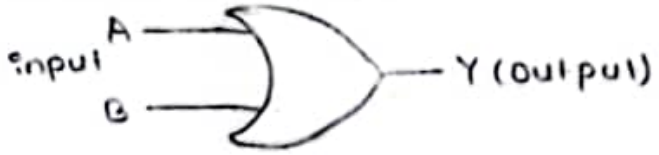


# VHDL SIMULATION OF OR GATE

## ~~To~~ THEORY:

OR GATES are fundamental logic gates that output a high (1) signal if at least one of their input signals is high; else the output remain's low(0).

## Circuit diagram:




## Logical expression:

output = A OR B

## Truth table:

| A | B | output |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

VHDL Code:-

```
-- OR gate implementation
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

  entity OR-GATE is
      port (
            A:in STD_LOGIC;
            B: in STD_LOGIC;
            output:out STD_LOGIC;
            );
          end OR_GATE;
 architecture Behavioral of OR_GATE is
 begin
    output <= A OR B;
  end Behavioral;
```

Result :-

# VHDL SIMULATION OF NOT GATE

## THEORY:

A NOT gate, also known as an inverter is a basic logic gate that produces a high (1) output when its input is low(0) and vice versa.

## Circuit diagram:


## Logical Expression:

output = NOT A

## Truth table:

| Input | output |
|---|---|
| 0 | 1 |
| 1 | 0 |

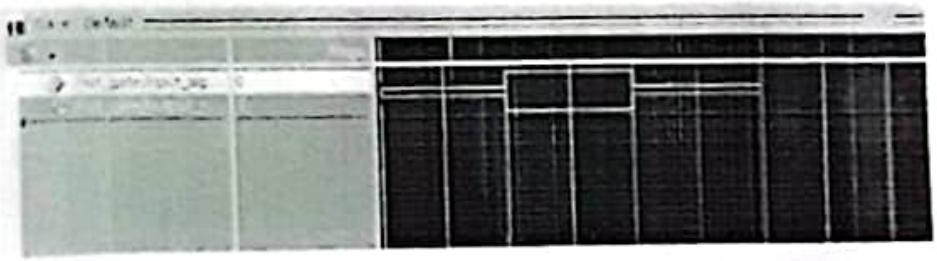
VHDL Code;

```
--NOT gate implementation
 library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

  entity NOT_GATE is
      Port (
            Input : in STD_LOGIC;
            output : out STD_LOGIC;
            );
      end NOT_GATE;

  architecture Behavioral of NOT_GATE is
  begin
      output <= not Input;
   end Behavioral;
```

Result:-



# VHDL SIMULATION OF NAND GATE

## THEORY:

A NAND gate is a basic logic gate that gives low(0) output only when both its input are high; otherwise it produces a high(1) output.

## Circuit diagram:


## Logical expression:

output = A NAND B

## Truth table:

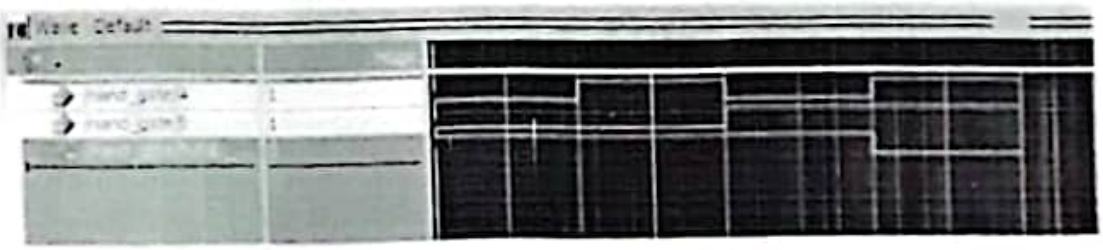
| Input(A) | B | output |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

VHDL Code:

```
--NAND gate implementation
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity NAND_GATE is
    port (
        A: in STD_LOGIC;
        B: in STD_LOGIC;
    output: out STD_LOGIC;
        );
    end NAND_GATE;
architecture Behavioral of NAND_GATE is
begin
    output <= not (A and B);
  end Behavioral;
```

Result:-



# VHDL SIMULATION OF XOR GATE

## THEORY:

An XOR gate, also known as an exclusive OR gate produces a high (1) when its two input signals are different; otherwise it output low (0).

## circuit diagram:


## Logical expression:

output = A XOR B

## Truth table:

| Input (A) | Input (B) | Output |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

VHDL code :

```
-- XOR gate implementation
 library IEEE;
 use IEEE.STD.LOGIC_1164.ALL;

 entity XOR_GATE is
      port (
            A: in STD_LOGIC;
            B: in STD_LOGIC;
        output: out STD_LOGIC;
            );
     end XOR_GATE;
 architectural Behavioral of XOR_GATE is
 begin
     output <= A XOR B;
   end Behavioral;
```

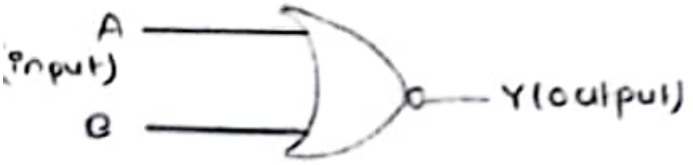
Result:

# VHDL SIMULATION OF NOR GATE

## THEORY:

A NOR gate is a fundamental digital logic gate that generates a high (1) output only when both input signals are low; otherwise it produces a low (0) output

## Circuit diagram:




## Logic expression:

output = A NOR B

## Truth table:

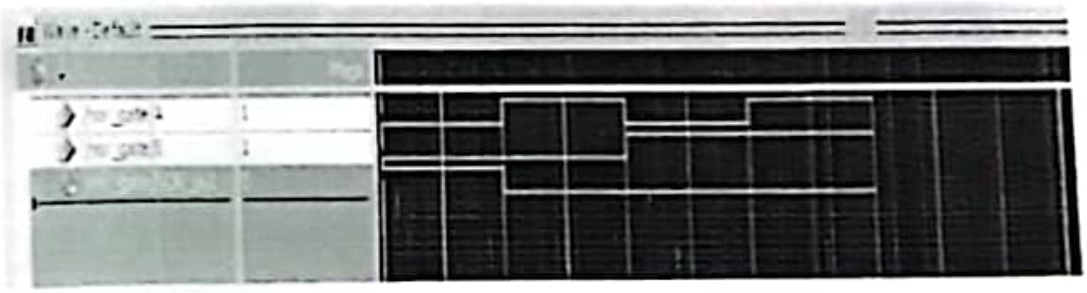
| A | B | output |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 0 |

VHDL code :

```
--NOR gate implementation
 libary IEEE;
use IEEE.STD_LOGIC_1164.All;

   entity NOR_GATE is
       Port (
            A: in STD_LOGIC;
            B: in STD-LOGIC;
     output: out STD-LOGIC;
          );
     end NOR-GATE;
architecture Behavioral of NOR_GATE is
 begin
     output <= not (A or B);
  end Behavioral;
```

Result :-

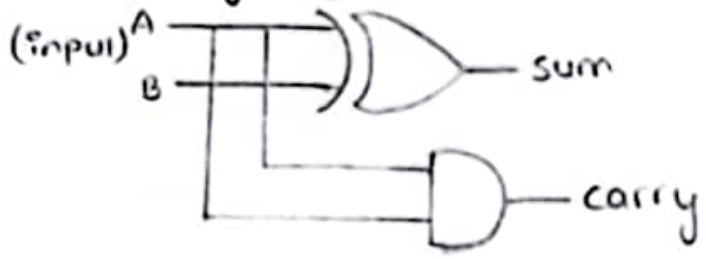


# VHDL SIMULATION OF NOR GATE

## THEORY:

A NOR gate is a fundamental digital logic gate that generates a high (1) output only when both input signals are low; otherwise it produces a low (0) output.

## Circuit diagram:




## Logical expression:

Sum = A XOR B,

Carry = A AND B

## Truth table:

| A | B | sum | carry |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |

## VHDL Code:

```
--Half adder implementation
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity HALF_ADDER is
    Port (
        A: in STD_LOGIC;
        B: in STD_LOGIC;
    Sum: out STD_LOGIC;
    carry: out STD_LOGIC;
    );
end HALF_ADDER

architecture Behavioral of HALF_ADDER is
begin
    Sum <= A xor B;
    carry <= A and B;
end Behavioral;
```

## Result:

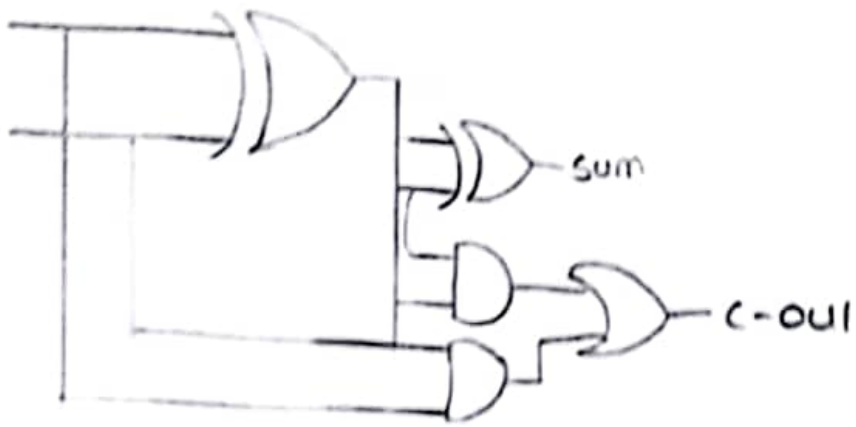
# VHDL SIMULATION OF FULL ADDER



## THEORY:

A full adder is a digital circuit that adds three binary digits: the two inputs (A and B) and a carry input (C-in), producing sum and carry output.

## Circuit diagram:




## Logical expression:

sum(s) = A XOR B XOR C_in

Carry out(C_out) = (A AND B) OR (C_in AND (A XOR B))

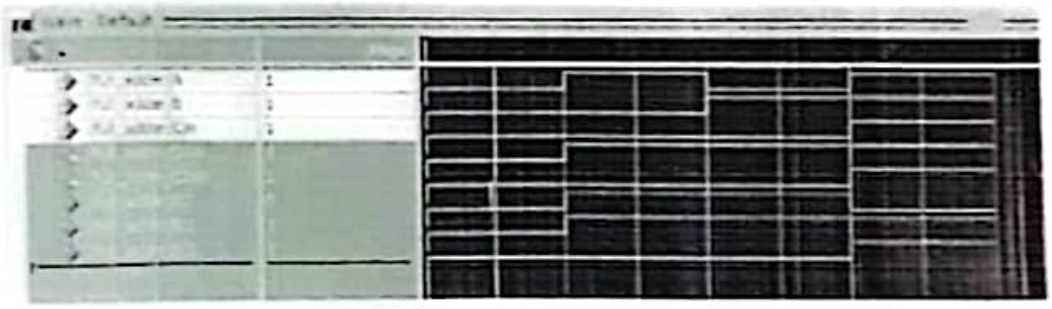
## Truth Table:

| A | B | C_in | sum(s) | C_out |
|---|---|---|---|---|
| 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 |

VHDL code:

```
-- full Adder implementation
 library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
 entity FULL_ADDER is
        Port (
              A: in STD_LOGIC;
              B: in STD_LOGIC;
          C_in: in STD_LOGIC;
           Sum: out STD_LOGIC;
        C_out: out STD_LOGIC
             );
    end FULL_ADDER;
 architecture Behavioral of FULL ADDER is
 begin
    Sum <= (A XOR B) XOR C_in;
   C_out <= (A and B) or (C_in and (A XOR B);
    end Behavioral;
```

Result:-

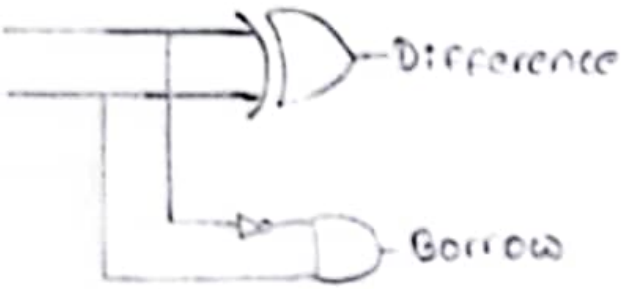


# VHDL SIMULATION OF HALF SUBTRACTOR

## THEORY:

A half subtractor is a basic digital circuit that subtracts two binary digits, producing a difference bit and a borrow bit.

## Circuit diagram:




## Logic expression:

Difference (D) = A XOR B

Borrow (B-out) = (NOT A) AND B

## Truth Table:

| A | B | Difference (D) | Borrow (B-out) |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 |

VHDL Code:

```
-- Half Subtractor Implementation
 library IEEE;
 use IEEE.STD_LOGIC_1164.ALL;

  entity HALF_SUBTRACTOR is
      Port (
            A: in STD_LOGIC;
            B: in STD-LOGIC;
     Difference: out STD-LOGIC;
        Borrow: out STD-LOGIC
          );
   end HALF_SUBTRACTOR
 architecture Behavioral of HALF_SUBTRACTOR is
 begin
     Difference <= A xor B;
     Borrow <= (not A) and B;
  end Behavioral;
```

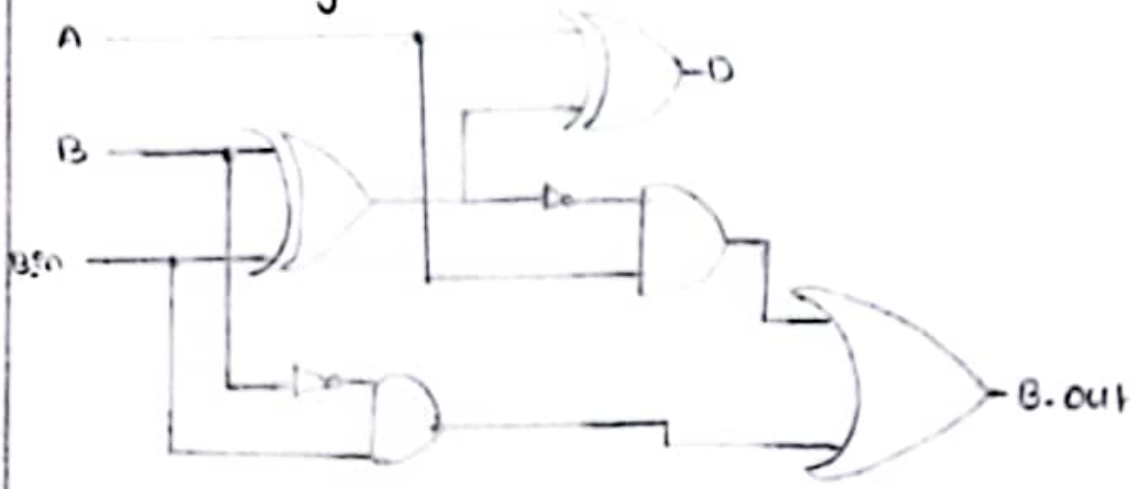
Result:

# VHDL SIMULATION OF FULL SUBTRACTOR

## THEORY:

A full subtractor is a digital circuit that performs subtraction of three binary digits: the two inputs (A and B) and a borrow input (B-in), resulting in difference and borrow output.

## circuit diagram:




## Logic expression:

Difference (D) = (A XOR B) XOR B-in

Borrow-out (B-out) = (B-in AND (NOT A) OR (B AND (NOT (A XOR B))))

## Truth table:

| A | B | B-in | Difference (D) | Borrow (B-out) |
|---|---|---|---|---|
| 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 |

VHDL code:

```
-- Full subtractor implementation
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity FULL_SUBTRACTOR is
    Port (
        A: in STD_LOGIC;
        B: in STD_LOGIC;
        B_in: in STD_LOGIC;
        D: in out STD_LOGIC;
        B_out: out STD_LOGIC;
        );
    end FULL_SUBTRACTOR
architecture Behavioral of FULL_SUBTRACTOR is
begin
    D <= (A XOR B) xor B_in;
    B_out <= (B_in and (not A) or (B and (not (A OR B)))
end Behavioral;
```

Result:

Program to convert decimal to hexa-decimal

Program:

```
#include <stdio.h>
    void decimal to Hexadecimal (int decimal){
        char hexa[20];
        int i=0;
while (decimal>0){
    int remainder = decimal % 16;
    if (remainder<10){
        hexa [i] = remainder + '0';
    }else{
        hexa [i] = remainder - 10 + 'A';
    }
     decimal /= 16;
      i++;
    }
    print (" Hexadecimal: 0x");
    for (int i=i-1; i>0=0; i--){
        print ("%c", hexa [i]);
    }
     Print f("\n");
    }
     int main (){
       int decimal;
       Print f("Enter a decimal number:");
       Scan ("%d", decimal);
      decimal To Hexadecimal (decimal);
       return 0;
     }
```

Output:

```
Enter a decimal number:20
Hexadecimal: 0x14
```

Program to convert decimal to binary

Program:

```
#include <stdio.h>
void decimal to binary (int decimal){
    if (decimal = 0){
      printf("Binary:0\n");
    return;
  }
  int binary[32];
  int i=0;
while (decimal Number >0){
    binary[i] = decimal %. 1;
    decimal /=2;
     i++;
  }
 printf ("Binary:");
 for (int ;2i-1; i>=0; i--){
   print ("%a", binary[i]);
  }
   print ("\n");
 }
int main(){
  int decimal;
  printf("Enter adecimal number:");
  scanf("%a", decimal);
  decimal To Binary(decimal);
  return 0;
}
```

Output:

Enter a decimal number:20

10100