LAB-1

1) Write a C program to convert decimal number into binary, octal and hexadecimal number system.

Source Code:

```
/* Program to convert a positive decimal number to Binary, Octal
and Hexadecimal */
#include <stdio.h>
void convert (int, int);
int main()
{
    int num;
    printf("Enter a positive decimal number: ");
    scanf("%d", &num);
    printf("\nBinary number:: ");
    convert(num, 2);
    printf("\n Octal number:: ");
    convert(num, 8);
    printf("\n Hexadecimal number:: ");
    convert(num, 16);
    return 0;
}
void convert(int num, int base)
{
    int rem = num % base;
    if (num == 0)
        return;
    convert(num/base, base);
    if (rem < 10)
        printf("%d", rem);
    else
        printf("%c", rem-10+'A');
}
```

Output:

LAB 1.

2) Write a C program to convert binary number into decimal, octal and hexadecimal number system.

Source Code:

```
/* Program to convert binary to decimal, octal and hexadecimal*/
#include<stdio.h>
#include<math.h>
int binaryToDecimal(long long binary);
void convert(int, int);
int main()
{
    long long binary;
    printf("Enter a binary number: ");
    scanf("%lld", &binary);
    int decimal = binaryToDecimal(binary);
    printf("Decimal number: %d", decimal);
    printf("\n Octal number: ");
    convert(decimal, 8);
    printf("\n Hexadecimal number: ");
    convert(decimal, 16);
    return 0;
}
int binaryToDecimal(long long binary){
    int decimal = 0, i=0, rem;
    while (binary!=0){
        rem=binary%10;
        binary /=10;
        decimal += rem*pow(2,i);
        ++i;
    }return decimal;
}
void convert(int num, int base){
    int rem = num % base;
    if (num==0)
        return;
    convert(num/base, base);
    if(rem<10)
        printf("%d", rem);
    else
        printf("%c", rem-10+'A');
}
```

Output:

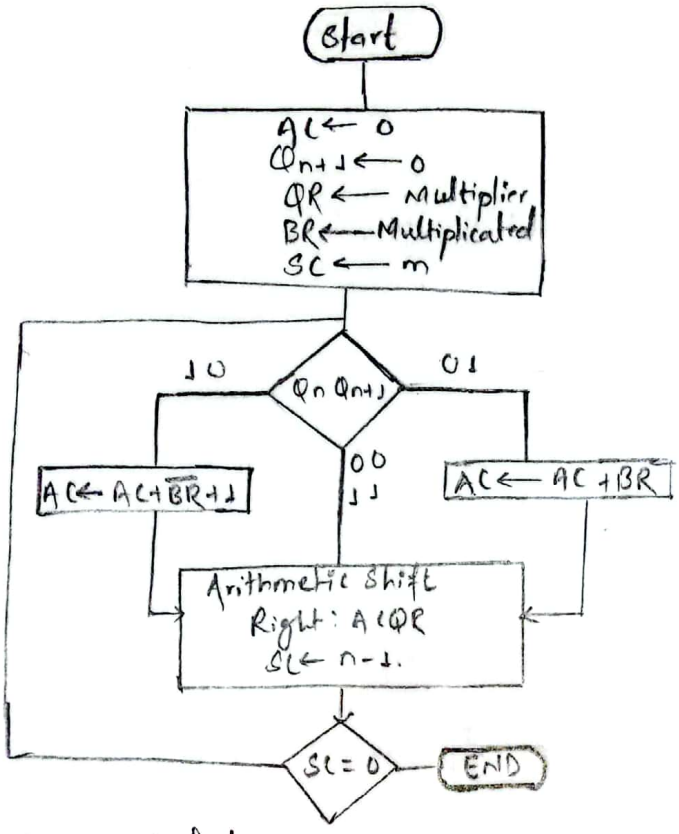
LAB 2

Write a C program to implement Booth algorithm.

Theory:

Booth's Algorithm is a technique used for multiplying two signed ~~numbe~~ binary numbers efficiently. It reduces the number of additions needed by performing shift and add operations instead.

Flowchart:



Source Code:

```
#include <stdio.h>
#include<math.h>
int a=0, b=0, c=0, a1=0, b1=0, com[5]={1,0,0,0,0};
int anum[5]={0}, anumcp[5]={0}, bnum[5]={0};
int acomp[5]={0}, bcomp[5]={0}, pro[5]={0}, res[5]={0};
void binary(){
      a1=fabs(a);
      b1=fabs(b);
      int r, r2, i, temp;
      for(i=0; i<5; i++){
            r= a1%2;
            a1 = a1/2;
            r2= b1%2;
            b1 = b1/2;
            anum[i]= r;
            anumcp[i] = r;
            bnum[i]= r2;
```

```
    if (r2==0){
      bcomp[i]=1;
    }
    if(r==0){
    acomp[i]=1;
    }
}
c=0;
for(i=0;i<5;i++){
      res[i]=com[i]+bcomp[i]+c;
       if(res[i]>=2){
          c=1;
        }else
          c=0;
       res[i]=res[i]%2;
    }
  for(i=4;i>=0;i--){
     bcomp[i]=res[i];
  }
  if(a<0){
     c=0;
    for(i=4;i>=0;i--){
        res[i]=0
    }
    for(i=0;i<5;i++){
        res[i]=com[i]+acomp[i]+c;
         if(res[i]>=2){
           c=1;
        }else
           c=0;
         res[i]=res[i]%2;
    }
    for(i=4;i>=0;i--){
        anum[i]=res[i];
        anump[i]=res[i];
    }
  }
  if(b<0){
       for(i=0;i<5;i++){
          temp=bnum[i];
          bnum[i]=bcomp[i];
          bcomp[i]=temp;
       }
    }
}
void add(int num[]){
    int i;
    c=0;
    for(i=0;i<5;i++){
        res[i]=pro[i]+num[i]+c;
        if(res[i]>=2){
          c=1;
        }else{
          c=0;
        }
```

```
    res[i] = res[i] % 2;
}
for(i=4; i>=0; i--) {
    pro[i] = res[i];
}
}
void arshift() {
    int temp = pro[4], temp2 = pro[0], i;
    for(i=1; i<5; i++) {
        pro[i-1] = pro[i];
    }
    pro[4] = temp;
    for(i=1; i<5; i++) {
        anumcp[i-1] = anumcp[i];
    }
    anumcp[4] = temp2;
}
void main() {
    int i, q=0;
    printf("\t\t Booth's Multiplication Algorithm");
    printf("\n Enter two numbers to multiply: ");
    printf("\n Both numbers must be less than 16");
    do{
        printf("\n Enter A: ");
        scanf("%d", &a);
        printf("\n Enter B: ");
        scanf("%d", &b);
    }while(a>=16 || b>=16);
    printf("\n Expected product = %d", a*b);
    binary();
    for(i=0; i<5; i++) {
        if(anum[i]==q) {
            arshift();
            q = anum[i];
        }else if(anum[i]==1 && q==0) {
            add(bcomp);
            arshift();
            q = annum[i];
        }else {
            add(bnum);
            arshift();
            q = anum[i];
        }
    }
```

```
printf ("\n Product is = ");
    for (i=4 ; i>=0 ; i--)
        { printf("%d", pro[i]);
        }
       for (i=4 ; i>=0; i--) {
           printf("%d", anum[p[i]]);
    } }
```

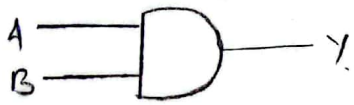
Output :-

LAB 8. 1) Write VHDL program for basic gates : AND Gate.

Theory:-

AND gates are basic logic gates that yields outputs high (1) signal only when both of the input signals are high (1) otherwise output is zero (0).

Circuit diagram



Truth Table.

A	B	Output (Y)
0	0	0
0	1	0
1	0	0
1	1	1

Logical Expression:

Y = A AND B

= A . B .

VHDL CODE:

```
-- VHDL code for AND gate.
library IEEE;
use IEEE.std_logic_1164.all;
    entity andGate is
        port (A: in std_logic;
              B: in std_logic;
              Y: out std_logic);
    end andGate;
    architecture andLogic of andGate is
     begin
        Y<=A AND B;
    end andLogic.
```

Output:

LAB-3

LAB 3

2) Write a VHDL Program for OR Gate.

Theory:

OR Gates are basic logic gates that yields high output (1) if at least one of two inputs is high(1) otherwise zero.

Circuit diagram

Truth Table,

A	B	Output(Y)
0	0	0
0	1	1
1	0	1
1	1	1

Logical Expression:

Y = A OR B = A + B.

VHDL code:

```
-- VHDL code for OR gate.
Library IEEE;
use IEEE.std_logic_1164.all;
  entity orGate is
      port (A: in std_logic;
            B: in std_logic;
            Y: out std_logic);
    end orGate;
  architecture orLogic of orGate is
    begin Y <= A OR B;
      end orLogic;
```

Output:

LAB-3

8. Write a VHDL program for NOT Gate.

Theory:

NOT gate is a basic logic gate which inverts the input i.e. if input is low (0) it yields high output (1) and vice-versa.

Circuit diagram

Truth Table.

A	Output (Y)
0	1
1	0

Logic Expression:

$Y = \overline{A}$

VHDL code:

```
-- VHDL Code for NOT Gate.
library IEEE;
use IEEE.std_logic_1164.all;
 entity not_gate is
        port(A : in std_logic;
             Y: out std_logic;
 end not_gate;
 architecture notLogic of not_gate is
  begin
        Y <= not(A);
        end notLogic;
```

Output :-

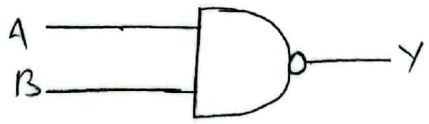
LAB 3

4. Write a VHDL program to implement NAND Gate.

Theory:

(1) NAND gate is a basic gate which gives high output if either of two inputs is zero otherwise gives low output (0).

Circuit diagram



Truth table

A	B	output (Y)
0	0	1
0	1	1
1	0	1
1	1	0

Logic Expression:

$Y = A \text{ NAND } B$

$= \overline{A \cdot B}$

VHDL Code:

```
-- Implementation of NAND Gate.
library IEEE;
use IEEE.std_logic_1164.all;
entity nand_gate is
    port (A: in std_logic;
          B: in std_logic;
          Y: out std_logic;
end nand_gate;
architecture ~~NANDLO~~ nandLogic of nand_gate is
begin
    Y <= ~~NOT~~ not (A and B);
end nandLogic;
```

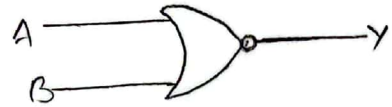
Output:

LAB 3

5) Write a VHDL Program to implement NOR Gate.

Theory:- NOR gate is a basic logic gate in digital circuit which gives high output(1) if both the inputs are low otherwise gives low output.

Circuit diagram:



Truth Table

A	B	Output Y
0	0	1
0	1	0
1	0	0
1	1	0

Logic Expression:-

$$Y = A \text{ (NOR) } B$$
$$= \overline{A + B}$$

VHDL Code:-

```
-- NOR Gate implementation.
library IEEE;
use IEEE.std_logic_1164.all;
entity nor_gate is
    port (A: in std_logic;
          B: in std_logic;
          Y: out std_logic);
end nor_gate;
architecture norlogic of nor_gate is
begin
    Y <= not (A or B);
end norLogic;
```

Output:-

LAB-3

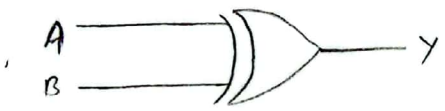
LAB 3

6. Write a VHDL program to implement XOR Gate.

Theory:

X-OR Gate is also called exclusive OR gate which produces high output (1) when its inputs are different; otherwise produces low output (0)

Circuit diagram



Truth table.

A	B	Output(Y)
0	0	0
0	1	1
1	0	1
1	1	0

Logic expression:

$Y = A \text{ XOR } B$

$= A \oplus B$

VHDL Code:

```
-- XOR Gate implementation.
library IEEE;
use IEEE.std_logic_1164.all;
    port (A : in std_logic;
          B : in std_logic;
          Y : out std_logic;
    end xor_gate;
      architecture xorLogic of xorgate is
        begin
          Y <= A xor B;
        end xorLogic;
```

Output:

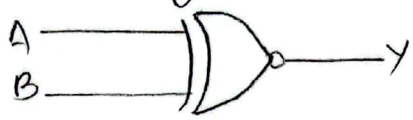
LAB-3

7) Write a program (VHDL) to implement X-NOR gate.

Theory:

X-NOR gate is also called exclusive NOR gate which produces high output (1) if both the inputs are same otherwise zero.

Circuit diagram:



Truth Table

A	B	Output (Y)
0	0	1
0	1	0
1	0	0
1	1	1

Logic expression:

$Y = A \text{ (X-NOR) } B$

$= \overline{A \oplus B}$

VHDL code:

```
-- XNOR Gate implementation
library IEEE;
use IEEE.std_logic_1164.all;
entity xnor_gate is
    port ( A: in std_logic;
           B: in std_logic;
           Y: out std_logic);
end xnor_gate;
architecture xnorlogic of xnor_gate is
begin
    Y <= not (A xor B);
end xnorlogic;
```

Output:

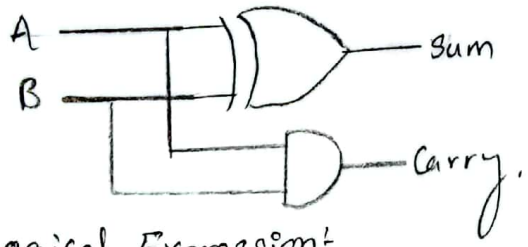
LAB-4

1. Write a VHDL program to implement half adder.

Theory:

A half adder is a digital logic circuit that performs binary addition of two single-bit binary numbers. It has two inputs, A and B and two outputs, Sum and Carry.

Circuit Diagram



Truth table.

A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Logical Expression:

Sum = A XOR B = $A \oplus B$

Carry = A AND B = $A \cdot B$

VHDL Code:

```
-- Implementation of Half Adder
library ieee;
entity half_adder is
    port (a, b : in std_logic;
          sum, carry_out : out std_logic);
end half_adder;
architecture dataflow of half_adder is
  begin
    sum <= a xor b;
    carry_out <= a and b;
  end dataflow;
```

Output:

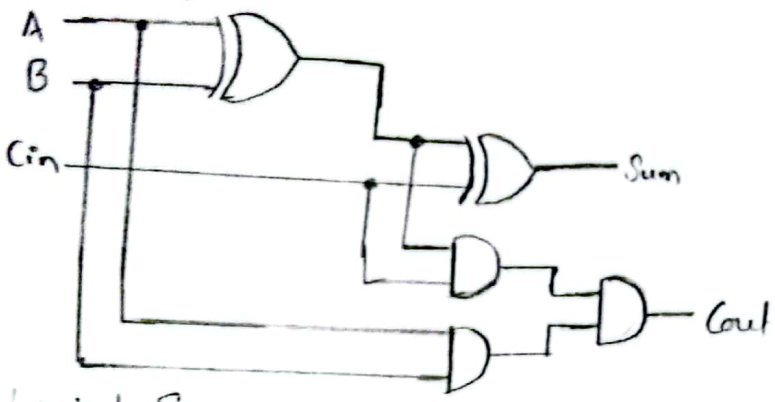
LAB-4.

2. Write a VHDL program to implement Full Adder

Theory:

A full adder is a digital circuit that adds three binary bits and gives two outputs i.e. Sum and carry (Cout)

Circuit diagram:



Logical Expression

Sum = A ⊕ B ⊕ C

Carry (Cout) = A·B + (A ⊕ B)·Cin

Truth Table.

A	B	Cin	Sum	Cout
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:

```
-- Implementation of Full Adder
library ieee;
use ieee.std_logic_1164.all;
entity adder is
    port( A, B, Ci: in std_logic;
          S, Co: out std_logic);
end adder;
architecture behavior of adder is
begin
    S <= A xor B xor Ci;
    Co <= (A and B) or ((a xor b) and Ci);
end behavior.
```

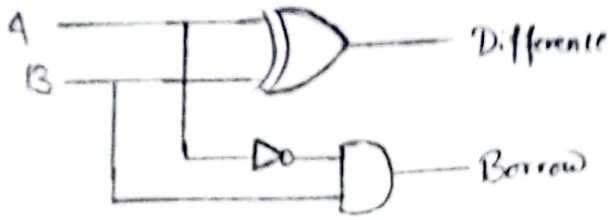
LAB-4

8. Write a program (VHDL) to implement a 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



Truth Table

A	B	Difference	Borrow
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

Logic Expression:

Difference (D) = $A \oplus B$

Borrow (B_{out}) = $\bar{A} \cdot B$

VHDL Code:

```
-- Implementation of half subtracter.
Library ieee;
use ieee.std_logic_1164.all;
  entity half_sub is
      port (a,b : in std_logic;
            d,f, bo : out std_logic);
  end half_sub;
  architecture sub_arch of half_sub is
    begin
      d,f <= a xor b;
        bo <= (not a) and b;
    end sub_arch;
```

Output:

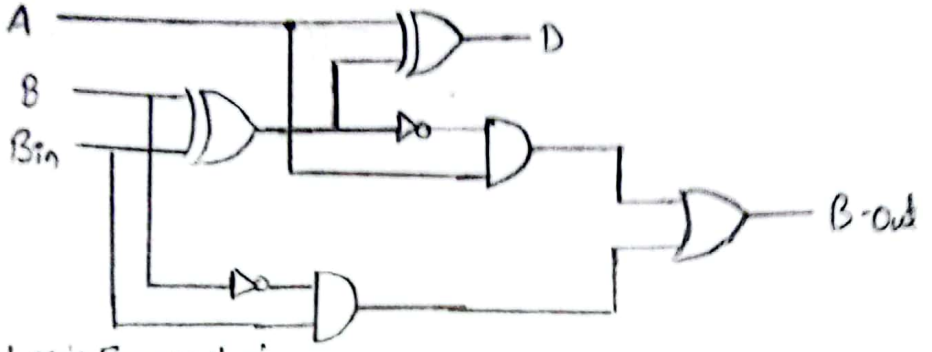
LAB-4

4. Write a VHDL program to implement 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 and produces difference and borrow as an outputs.

Circuit diagram.



Logic Expression:

Difference (D) = $A \oplus B \oplus Bin$

Borrow out (Bout) = $\bar{A}B + \bar{A}Bin + B\,Bin$

Truth Table:

A	B	Bin	Difference	Borrow
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:

```
-- Implementation of full subtractor.
Library IEEE;
Use IEEE.STD_LOGIC_1164.ALL;
entity full_sub is
    port(a,b,c: in std_logic;
         sub,borrow: out std_logic);
end full_sub;
architecture data of full_sub is
begin
    sub <= a xor b xor c;
    borrow <= ((b xor c) and (not a)) or (b and c);
end data;
```

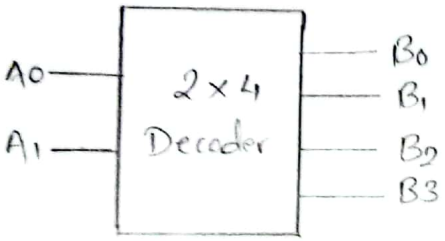
LAB-5

1. Write a VHDL Program to implement 2 x 4 decoder.

Theory:

2 x 4 decoder has 2-bit input lines and 4 output lines. The decoder analyzes the input combination and activates the corresponding output line.

Diagram



Truth Table:

Input		Output			
A_1	A_0	B_3	B_2	B_1	B_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

VHDL Program:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity Decoder_Source is
    Port ( I : in STD_LOGIC_VECTOR (1 downto 0);
           Y : out STD_LOGIC_VECTOR (3 downto 0));
end Decoder_Source;
architecture Behavioral of Decoder_Source is
begin
    process (I)
    begin
        case I is
            when "00" => Y <= "0001";
            when "01" => Y <= "0010";
            when "10" => Y <= "0100";
            when others => Y <= "1000";
        end case;
    end process;
end Behavioral;
```

Output:

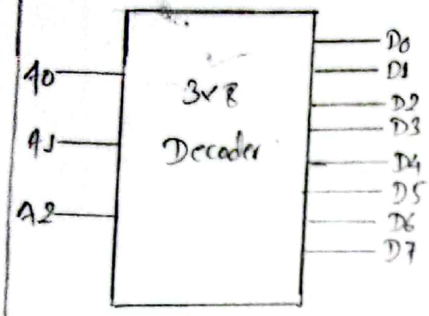
LAB-5

2. Write a VHDL program to implement 3x8 decoder.

Theory:

3x8 decoder is a combinational circuit that takes three bit input lines and 8 output lines.

Diagram:



Truth table.

A_0	A_1	A_2	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

VHDL code:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity decoder is
    Port ( i2,i1,i0: in STD_LOGIC;
           d: out STD_LOGIC_vector (7 downto 0));
end decoder;
architecture Behavioral of decoder is
begin
    process (i0,i1,i0)
    variable input: std_logic_vector (2 downto 0);
    begin
        input := i2 & i1 & i0;
        case input is
            when "000" => d <= "00000001";
            when "001" => d <= "00000010";
            when "010" => d <= "00000100";
            when "011" => d <= "00001000";
            when "100" => d <= "00010000";
            when "101" => d <= "00100000";
            when "110" => d <= "01000000";
            when others => d <= "10000000";
        end case;
    end process;
end behavioral;
```

Output:

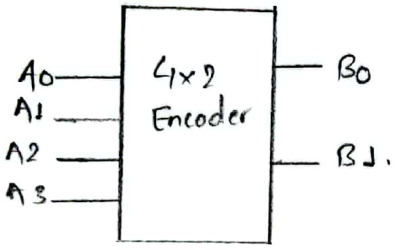
LAB-5

3. Write a VHDL Program to implement 4x2 encoder.

Theory:

4x2 encoder is a combination logic circuit which produces two bit outputs by analysing 4-bit input lines.

Diagram:



Truth Table:

Input				Output	
A3	A2	A1	A0	B1	B0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

VHDL code:

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
entity encoder is
    port ( a: in STD_LOGIC_VECTOR (3 downto 0);
           b: out STD_LOGIC_VECTOR (1 downto 0));
end encoder;
architecture bhv of encoder is
begin
    process(a)
    begin
    case a is
    when "1000" => b <= "00";
    when "0100" => b <= "01";
    when "0010" => b <= "10";
    when "0001" => b <= "11";
    end case;
    end process;
end bhv;
```

Output:

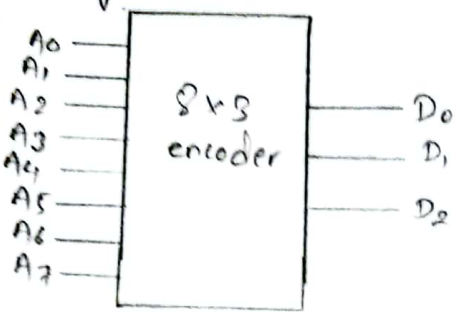
LAB-5

4. Write a VHDL program to implement 8x3 encoder.

Theory:

8x3 encoder is a combinational circuit that has 8 input lines and produces 3 output lines.

Diagram:



Truth table

A7	A6	A5	A4	A3	A2	A1	A0	D2	D1	D0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1

VHDL code:

```
library ieee;
use ieee.std_logic_1164.all;
entity enco8x3_seq is
    port ( i : in std_logic_vector (7 downto 0);
           o : out std_logic_vector (2 downto 0));
end enco8x3_seq;
architecture beh of enco8x3_seq is
begin
    enco: process (i)
    variable temp: std_logic_vector (2 downto 0);
    begin
        case i is
            when "00000001" => temp := "000";
            when "00000010" => temp := "001";
            when "00000100" => temp := "010";
            when "00001000" => temp := "011";
            when "00010000" => temp := "100";
            when "00100000" => temp := "101";
            when "01000000" => temp := "110";
            when "10000000" => temp := "111";
        end case;
        o <= temp;
    end process enco;
end beh;
```

Output:

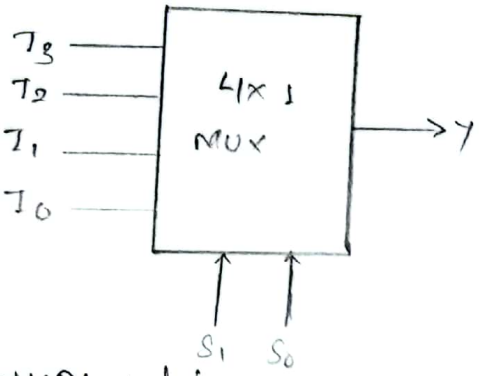
LAB-5

5. Write a VHDL Program to implement 4x1 Multiplexer

Theory:

4x1 MUX. is a combinational circuit that has four data inputs I_3, I_2, I_1 and I_0 and two selection lines S_1 & S_0 and one output Y.

Diagram:



Truth table.

S_1	S_0	Output (Y)
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

VHDL code:

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
entity mux_4to1 is
    port( A,B,C,D: in std_Logic;
          S0,S1: in std_logic;
          z: out std_logic);
end mux_4to1;
architecture bhv of mux_4to1 is
begin
    process (A,B,C,D,S0,S1) is
        begin
            if (S0='0' and S1='0') then
                z<=A;
            elsif (S0='1' and S1='0')then
                z<=B
            else if (S0='0' and S1='1')then
                z<=C
            else z<=D;
            end if;
    end process;
    end bhv;
```

Output:

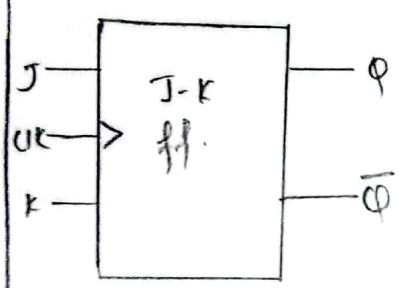
LAB 6

1. Write a VHDL program to implement JK flip flop.

Theory:

JK Flip flop is most widely used of all flip-flop designs where there are two inputs i.e. J & K - It is considered as a universal flip-flop.

Diagram:



Truth Tables.

Clk	J	K	Q	$\overline{Q}$	State.
↓	0	0	Q	$\overline{Q}$	No change
↓	0	1	0	1	Resets Q to 0
↓	1	0	1	0	Sets Q to 1
↓	1	1	#	-	Toggles

VHDL code:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity JK_FF is
    port (J,k,clk, rst: in std_logic;
          Q, Qbar : out std_logic);
end JKFF
 architecture behavioral of JK_FF is
   begin
      process (clk, rst)
       variable qn : std_logic;
       begin
          if (rst='1' then qn:='0';
          elseif ((clk'event and clk='1' then
           if (J='0' and k='0') then
              qn:=qn;
           elseif( J='0' and k='1') then qn:='0';
           elseif (J='1' and k='0') then qn:='1';
           elseif (J='1' and k='1') then qn:= not qn;
       else
          null;
       endif;
        else null;
        end if;
          Q<=qn;
          Qbar <= not qn;
       end process;
       end behavioral;
```

LAB-6

2. Write a VHDL program to implement SR flip flop.

Theory:

SR flipflop is a basic flipflop with two inputs one is S and other is R. S here stands for set and R here stands for Reset.

Diagram

Truth table

S	R	Q_{n+1}
0	0	Q_n
0	1	0
1	0	1
1	1	Indeterminate

VHDL code:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity SR_FLIPFLOP_SOURCE is
    port (S,R, RST, CLK): in STD_LOGIC;
          Q,Qb : out STD_LOGIC);
end SR_FLIPFLOP_SOURCE;
architecture Behavioral of SR_FLIPFLOP_SOURCE is
begin
    process (S,R,RST,CLK)
    begin
        if (RST='1') then  Q<='0';
        elseif (RISING_EDGE (CLK)) then
        if (S/=R) then Q <= S;
                       Qb<= R;
        elseif (S='1' AND R='1') then
              Q <='z';
              Qb <='z';
        end if;
        end if;
        end process;
        end behavioral;
```

Output:

LAB-7.

1. Write a VHDL program to implement 4-bit ALU design.

Theory:

It consists of 4 gates and a multiplexer each of the four logic operations is generated through a gate that performs the required logic.

VHDL code:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
entity alu is
    Port (inp_a : in signed (3 downto 0);
          inp_b : in signed (3 downto 0);
          sel : in STD_LOGIC_VECTOR (2 downto 0);
          out_alu : out signed (3 downto 0));
end alu;
architecture Behavioral of alu is
begin
    process (inp_a, inp_b, sel)
    begin
        case sel is
            when "000" => out_alu <= inp_a + inp_b;
            when "001" => out_alu <= inp_a - inp_b;
            when "010" => out_alu <= inp_a - 1;
            when "011" => out_alu <= inp_a + 1;
            when "100" => out_alu <= inp_a and inp_b;
            when "101" => out_alu <= inp_a or inp_b;
            when "110" => out_alu <= not inp_a;
            when others => out_alu <= inp_a xor inp_b;
        end case;
    end process;
end Behavioral;
```

Output:

LAB-7

2) Write a VHDL program to implement 8-bit ALU.

VHDL code:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
entity eightALU is
    Port ( a : in signed (7 downto 0);
           b : in signed (7 downto 0);
         sel : in STD_LOGIC_VECTOR (2 downto 0);
     out_alu : out (7 downto 0));
end eightALU;
architecture Behavioral of eightALU is.
begin
  process (a, b, sel)
    begin
      case sel is
          when "000" => out_alu <= a+b;
          when "001" => out_alu <= a-b;
          when "010" => out_alu <= a-1;
          when "011" => out_alu <= a+1;
          when "100" => out_alu <= a and b;
          when "101" => out_alu <= a or b;
          when "110" => out_alu <= not a;
          when "111" => out_alu <= a xor b;
      end case;
    end process;
end behavioral;
```

Output: