VHDL CODE

What is VHDL code?

VHDL stands for **VHSIC Hardware Description Language**.

- It is a programming language used to **describe and model digital electronic systems**, like circuits and chips.
- VHDL lets engineers **design, simulate, and verify hardware** before physically building it.
- It is commonly used for designing **FPGA** (Field Programmable Gate Arrays) and **ASICs** (Application Specific Integrated Circuits).
- Unlike software programming languages, VHDL describes **hardware behavior and structure**.

How to write VHDL code?

VHDL code usually consists of two main parts:

1. **Entity** — describes the interface (inputs and outputs).
2. **Architecture** — describes the internal behavior or structure of the design.

Basic VHDL code structure example:

```
-- importing required packages
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;


-- Entity:declaring all input and output ports
entity AND_Gate is
    port (
        A : in STD_LOGIC;
        B : in STD_LOGIC;
        Y : out STD_LOGIC
    );
end AND_Gate;


- Architecture:main body section
architecture Behavioral of AND_Gate is
begin
    Y <= A and B; -- logic for AND gate
end Behavioral;
```

Explanation:

- **library IEEE;** and **use IEEE.STD_LOGIC_1164.ALL;**
  Import standard logic package.

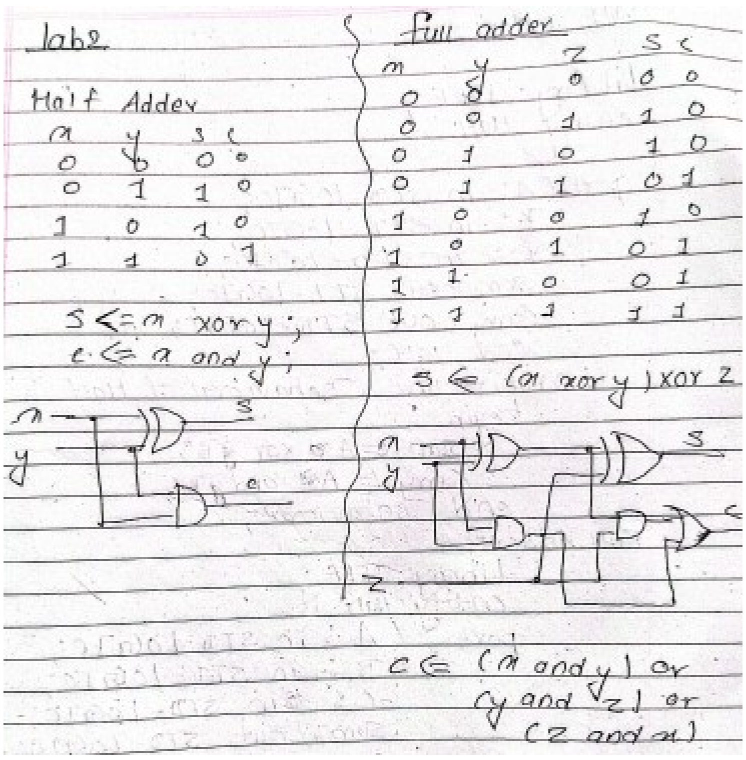
- **entity AND_Gate is ... end AND_Gate;**
  Defines the interface of the AND gate with inputs A and B, output Y.
- **architecture Behavioral of AND_Gate is ... end Behavioral;**
  Defines how the output Y is assigned as the logical AND of inputs A and B.

Labs:

<u>Lab1:Basic Logic Gates</u>

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity LogicGates is
    port (
        A : in STD_LOGIC;
        B : in STD_LOGIC;
        C : out STD_LOGIC;
        D : out STD_LOGIC;
        E : out STD_LOGIC;
        F : out STD_LOGIC
    );
end LogicGates;

architecture Behavioral of LogicGates is
begin
    C <= A AND B;
    D <= A OR B;
    E <= NOT A;
    F <= A XOR B;
end Behavioral;
```

<u>Lab2:Half Adder and Full Adder</u>



lab2

Half Adder

x	y	s	c
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

```
s <= x xor y;
c <= x and y;
```

Full adder

x	y	z	s	c
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

```
s <= (x xor y) xor z
```

```
c <= (x and y) or
     (y and z) or
     (z and x)
```

Code:

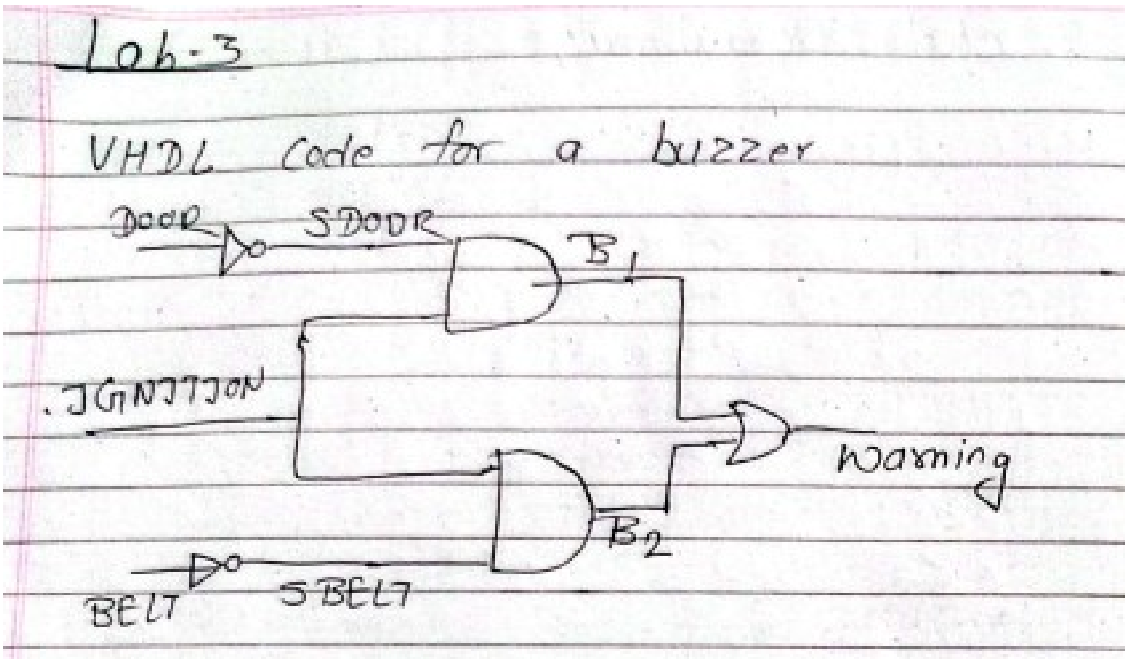
a)HalfAdder:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity HalfAdder is
    port (
        x : in STD_LOGIC;
        y : in STD_LOGIC;
        Sum : out STD_LOGIC;
        Carry : out STD_LOGIC
    );
end HalfAdder;

architecture Behavioral of HalfAdder is
begin
    Sum <= x XOR y;        -- Sum = x ⊕ y
    Carry <= x AND y;      -- Carry = x • y
end Behavioral;
```

b)Full Adder:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity FullAdder is
    port (
        x : in STD_LOGIC;
        y : in STD_LOGIC;
        Cin : in STD_LOGIC;
        Sum : out STD_LOGIC;
        Carry : out STD_LOGIC
    );
end FullAdder;

architecture Behavioral of FullAdder is
begin
    Sum <= (x XOR y) XOR Cin;                          -- Sum = x ⊕ y ⊕ Cin
    Carry <= (x AND y) OR (Cin AND y) OR (Cin AND x);    -- Carry logic
end Behavioral;
```

<u>Lab3:</u>* *<u>car Warning system</u>



code:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity CarWarningSystem is
   port (
      Door : in STD_LOGIC;      -- Door sensor (1 = open)
      Belt: in STD_LOGIC;      -- Seat belt sensor (1 = unbuckled)
      Ignition : in STD_LOGIC;  -- Ignition status (1 = on)
      Warning : out STD_LOGIC   -- Warning light (1 = activate)
   );
end CarWarningSystem;

architecture Behavioral of CarWarningSystem is
   signal SDoor, SBELT, B1, B2 : STD_LOGIC;
begin
   -- NOT gates for input inversion
   U1 : entity work.NOT_gate port map(Door, SDoor);
   U2 : entity work.NOT_gate port map(Belt, SBELT);

   -- AND gates for condition checking (with Ignition)
   U3 : entity work.AND_gate port map(SDoor, Ignition, B1);
   U4 : entity work.AND_gate port map(SBELT, Ignition, B2);

   -- OR gate for final warning output
   U5 : entity work.OR_gate port map(B1, B2, Warning);

end Behavioral;
```

-- **component definations**

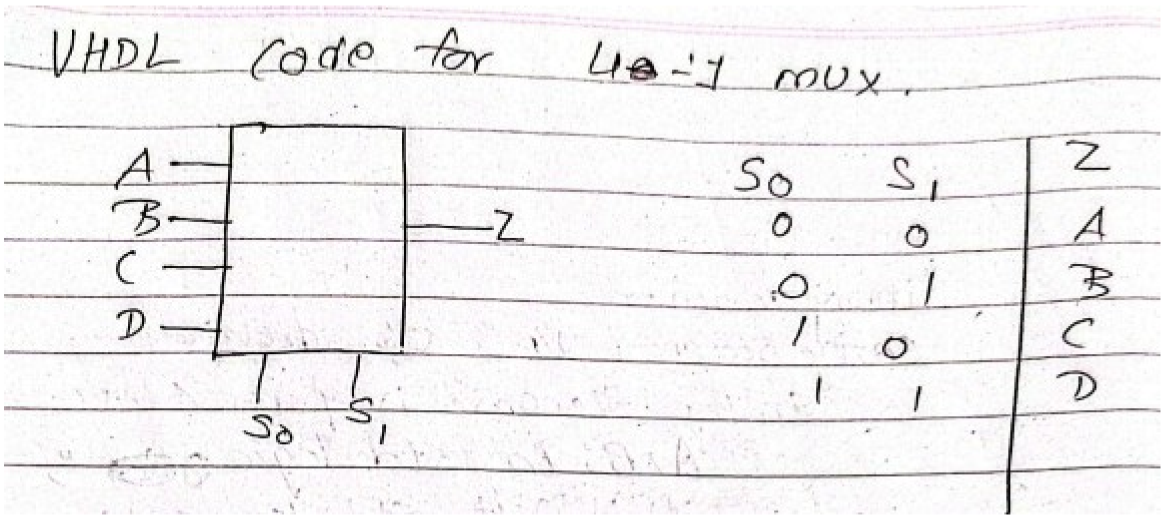
-- NOT gate

```
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 Behavioral of NOT_gate is
begin
   Y <= not A;
end Behavioral;

-- AND gate
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity AND_gate is
   port (
      A, B : in STD_LOGIC;
      Y : out STD_LOGIC
   );
end AND_gate;

architecture Behavioral of AND_gate is
begin
   Y <= A and B;
end Behavioral;
-- OR gate
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity OR_gate is
   port (
      A, B : in STD_LOGIC;
      Y : out STD_LOGIC
   );
end OR_gate;

architecture Behavioral of OR_gate is
begin
   Y <= A or B;
end Behavioral;
```

****** Functionality:********

- `Warning` turns on if:
    - Door is **not closed** and ignition is on
      OR
    - Seat belt is **not fastened** and ignition is on

<u>Lab4:(MUX 4-to-1)</u>



S0	S1	Z
0	0	A
0	1	B
1	0	C
1	1	D

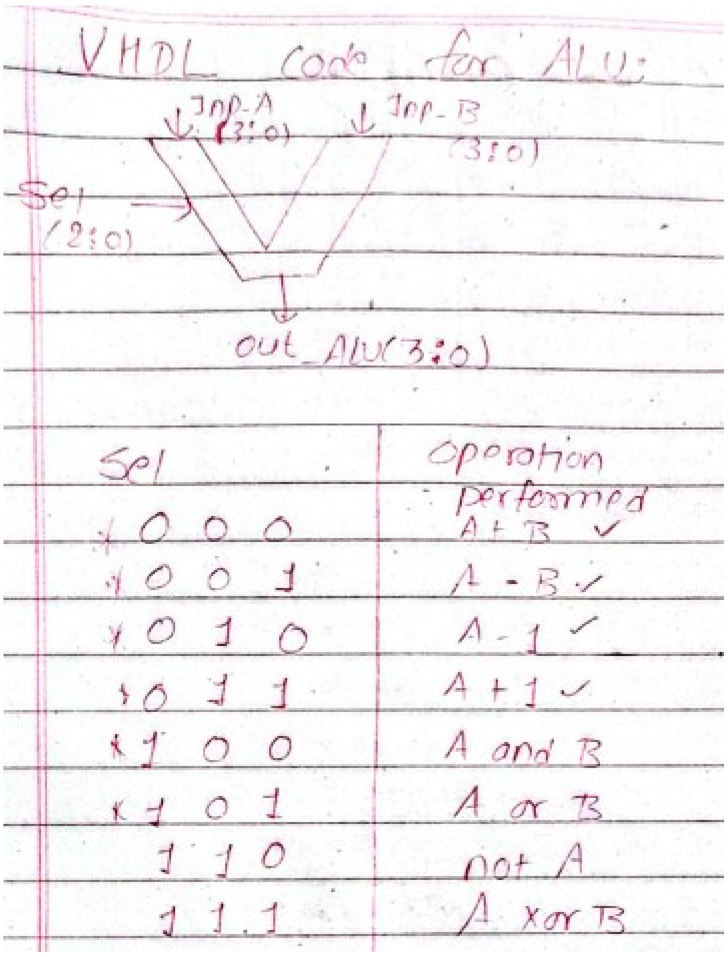
Code:
a)using if -elsif

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Mux4to1 is
    port (
        A, B, C, D : in STD_LOGIC;
        X, Y       : in STD_LOGIC;
        Z          : out STD_LOGIC
    );
end Mux4to1;

architecture Behavioral of Mux4to1 is
begin
    process(A, B, C, D, X, Y)  -- only inputs
    begin
        if (X = '0' and Y = '0') then
            Z <= A;
        elsif (X = '0' and Y = '1') then
            Z <= B;
        elsif (X = '1' and Y = '0') then
            Z <= C;
        else  -- X='1' and Y='1'
            Z <= D;
        end if;
    end process;
end Behavioral;
```

b)using array

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Mux4to1_Array is
    port (
        Inputs : in STD_LOGIC_VECTOR(3 downto 0);  -- D, C, B, A
        Sel    : in STD_LOGIC_VECTOR(1 downto 0);  -- {X, Y}
        Z      : out STD_LOGIC
    );
end Mux4to1_Array;

architecture Behavioral of Mux4to1_Array is
begin
    with Sel select
        Z <= Inputs(0) when "00",  -- A
             Inputs(1) when "01",  -- B
             Inputs(2) when "10",  -- C
             Inputs(3) when others; -- D
end Behavioral;
```

lab5:ALU operations


VHDL code for ALU:

Sel	operation performed
0 0 0	A + B ✓
0 0 1	A - B ✓
0 1 0	A - 1 ✓
0 1 1	A + 1 ✓
1 0 0	A and B
1 0 1	A or B
1 1 0	not A
1 1 1	A xor B

code:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity ALU is
   Port (
      inp_a  : in  STD_LOGIC_VECTOR (3 downto 0);
      inp_b  : in  STD_LOGIC_VECTOR (3 downto 0);
      sel    : in  STD_LOGIC_VECTOR (2 downto 0);
      out_alu : out STD_LOGIC_VECTOR (3 downto 0)
   );
end ALU;

architecture Behavioral of ALU is
begin
   process(inp_a, inp_b, sel)
      variable a, b : unsigned(3 downto 0);
      variable result : unsigned(3 downto 0);  -- out_alu=result
   begin
      a := unsigned(inp_a);  -- a=inp-a
      b := unsigned(inp_b);  --  b=inp_b
      case sel is
         when "000" =>  -- A + B
            result := a + b;         -- note:instead you can use inp_a+inp_b directly

         when "001" =>  -- A - B
            result := a - b;

         when "010" =>  -- A + 1
            result := a + 1;

         when "011" =>  -- A - 1
            result := a - 1;

         when "100" =>  -- A AND B
            result := a and b;

         when "101" =>  -- A OR B
            result := a or b;

         when "110" =>  -- NOT A
            result := not a;

         when "111" =>  -- A XOR B
            result := a xor b;
            when others =>
                result := (others => '0');
        end case;

        out_alu <= std_logic_vector(result); -- out_alu=result
    end process;
end Behavioral;
```