33) Write down VHDL code for following:

⇒ a) AND gate:

```
-- Header file declaration
library Ieee;
use Ieee.std_logic_1164.all;

--Entity declaration

entity andGate is
        port(A: in std_logic; --A input
             B: in std_logic; -- B input
             Y: out std_logic); -- Output
end andGate;

architecture andLogic of andGate is

begin
        Y<=A AND B;

end andLogic;
```

⇒ "--" is used to comment.

b) 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;
```

c) 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;
```

d) NOR Gate:

→
```
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 no_gate is
begin
     Y <= not(A OR B)
end norlogic;
```

e) 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 nandlogic of nand_gate is
begin
     Y <= not (A and B);
end nandlogic;
```

f) XOR Gate:

→
```
library IEEE;
use IEEE.std_logic_1164.all;
entity XOR_gate is
port(A: in std_logic;
     B: in std_logic;
     Y: out std_logic);
end XOR_gate;
architecture XORlogic of XOR_gate is
begin
    Y<=A XOR B;
end XORlogic;
```

g) XNOR Gate:

→
```
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;
```

h) Half adder:

→
```
library IEEE;
use IEEE.std_logic_1164.all;
entity H_adder is
port (A: in std_logic;
      B: in std_logic;
      Sum: out std_logic;
      Carryout: out std_logic);
end H_adder;
architecture flow of H_adder is
begin
    sum <= A xor B;
    carryout <= A and B;
end flow;
```

i) full adder:

→
```
library IEEE;
use IEE.std_logic_1164.all;
entity full_adder is
port (A, B, carry_in: in std_logic;
      Sum, carr_out: out std_logic);
end full_adder;
architecture flow of full_adder is
component half_adder is
port (A, B: in std_logic;
      Sum, carryout: out std_logic);
end component;
component or_gate is
port (A, B: in std_logic;
Y: out std_logic);
```

```
end component;
signal T1, T2, T3: std_logic;
begin
half_adder1: half_adder port map
   (A => A, B => B, sum => T1, carryout => T2);
half_adder2: half_adder port map
   (A => T1, B => carry_in, sum => sum,
    carryout => T3);
or_gate1: or_gate port map (A => T3, B => T2,
          Y => carry_out);
end flow;
```

j) Half subtractor:

```
library IEEE;
use IEEE.std_logic_1164.all;
entity half_sub is
  port (A,B: in std_logic;
        diff, borrow: out std_logic);
end half_sub;
architecture flow of half_sub is
begin
   diff <= A xor B;
   borrow <= not(A) and B;
   end flow;
```

K) Full Subtractor:

→
```
library IEEE;
use IEEE.std_logic_1164.all;

entity full_sub is
  port(A, B, borrow_in: in std_logic;
  diff, borrow_out: out std_logic);

end full_sub;
architecture flow of full_sub is
component half_sub is
port(A, B: in std_logic;
     diff, borrow: out std_logic);
end component;

component or_gate is
port(A, B: in std_logic;
     Y: out std_logic);
end component;
signal T1, T2, T3: std_logic;
begin
half_sub1: half_sub port map
          (A => A, B => B, diff => T1, borrow => T2);
half_sub2: half_sub port map
          (A => T1, B => borrow_in, diff => diff,
          borrow => T3);
or_gate1: or_gate port map(A => T3, B => T2,
          Y => borrow_out);
end flow;
```

1) 4x1 multiplexer;

→
```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_ARITH.all;
entity mux_tb
use ieee.std_logic_unsigned.all;

entity MUX_SOURCE is
    Port (S: in std_logic_vector (1 downto 0);
          I: in std_logic_vector (3 downto 0);
          Y: out std_logic);
end MUX_SOURCE

architecture Behavioral of MUX_SOURCE is
begin
    process (S, I)
begin
  if (S <= "00") then
      Y <= I(0);
  elsif (S <= "01") then
  Y <= I(1);
  elsif (S <= "10") then
  Y <= I(2);
  else
   Y <= I(3);

end if;
end process;
end Behavioral;
```

m) 1x4 demultiplexer:

→
```
library IEEE;
use IEEE.std_logic_1164.all;

entity Demultiplexer1x4 is
port(
    A: in std_logic;
    S: in std_logic_vector (1 downto 0);
    F, G, H, I: out std_logic);
end entity Demultiplexer1x4;

architecture Behavioral of Demultiplexer1x4 is
begin
    with S select
        F <= A when "00",
             '0' when others;
    with S select
        G <= A when "01",
             '0' when others;
    with S select
        H <= A when "10",
             '0' when others;
    with S select
        H <= A when "10",
             '0' when others;
    with S select
        I <= A when "11",
             '0' when others;

end architecture Behavioral;
```

n) 8x3 encoder:

→
```
library ieee;
use ieee.std_logic_1164.all;

entity Encoder8x3 is
  port(
      A,B,C,D,E,F,G,H: in std_logic;
      Y: out std_logic_vector(2 downto 0));
end entity Encoder8x3;

architecture Behavioral of Encoder8x3 is
begin
   Y<="000" when A = '1' else
      "001" when B = '1' else
      "010" when C = '1' else
      "011" when D = '1' else
      "100" when E = '1' else
      "101" when F = '1' else
      "110" when G = '1' else
      "111" when H = '1'
      "000";
end architecture Behavioral;
```

0) 3x8 decoder:

→
```
library ieee;
use ieee.std_logic_1164.all;

entity Decoder3x8 is
    port(
        A, B, C: in std_logic;
        Y: out std_logic_vector(7 downto 0));
end entity Decoder3x8;

architecture Behavioral of Decoder3x8 is
begin
```

| | |
|---|---|
| ~~Y <= "00000001"~~<br>~~"00000010"~~<br>~~"00000011"~~<br>~~"00000100"~~<br>~~"00000101"~~<br>~~"00000110"~~<br>~~"00000~~<br>" | Y <= "00000001" when (A='0' and B='0' and C='0') else<br>"00000010" when (A='0' and B='0' and C='1') else<br>"00000100" when (A='0' and B='1' and C='0') else |

```
"00001000" when (A='0' and B='1' and C='1') else
"00010000" when (A='1' and B='0' and C='0') else
"00100000" when (A='1' and B='0' and C='1') else
"01000000" when (A='1' and B='1' and C='0') else
"10000000" when (A='1' and B='1' and C='1') else
"00000000";
end architecture Behavioral;
```

p) SR flipflop:

→
```
library ieee;
use ieee.std_logic_1164.all;

entity SR_Flipflop is
  port (
      S, R: in std_logic;
      Q, Qbar: out std_logic);
end entity SR_Flipflop;

architecture Behavioral of SR_Flipflop is
signal Q_int, Qbar_int: std_logic;
begin
  process (S, R)
  begin
      if (R = '1') then
        Q_int <= '0';
        Qbar_int <= '1';
    elsif (S = '1') then
        Q_int <= '1';
        Qbar_int <= '0';
    endif;
  end process;

  Q <= Q_int;
  Qbar <= Qbar_int;

end architecture Behavioral;
```

q) JK Flipflop:

→
```
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, CLOCK: in std_logic;
Q, QB: out std_logic);
end JK_FF;

architecture behavioral of JK_FF is
begin
PROCESS (CLOCK)
variable TMP: std_logic;
begin
  if (CLOCK='1' and CLOCK'EVENT) then
   if (J='0' and K='0') then
    TMP:=TMP;
  elsif (J='1' and K='1') then
    TMP:= not TMP;
  elsif (J='0' and K='1') then
    TMP:='0';
  else
    TMP:='1';
  endif;
  end if;
   Q<=TMP;
   Q<= not TMP;
  end process;
   end behavioral;
```

x) T flipflop:

→
```
library ieee;
use ieee.std_logic_1164.all;

entity T_FF is
port(T: in std_logic;
 clock: in std_logic;
  Q: out std_logic);
  end T_FF;

architecture behavioral of T_FF is
signal tmp: std_logic;
begin
process(clock)
 begin
 if (clock'event and clock = '1' then
 if T = '0' then
 tmp <= tmp
 elsif T = '1' then
 tmp <= not(tmp);
 end if;
 end if;
 end process;
  Q <tmp;
 end Behavioral;
```

5) D Flipflop:

→
```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity D_FF is
PORT (D, CLOCK: in std_logic;
Q: out std_logic);
end D_FF;

architecture behavioral of D_FF is
begin
process(CLOCK)
begin
if(CLOCK='1' and CLOCK'EVENT) then
Q<=D;
end if;
end process;
end behavioral;
```

34) Write down the VHDL code for MODULO 6 counter using a low level of abstraction.

⇒
```
library ieee;
use ieee.std_logic_1164.all;

entity mod6 is
  port (
        v, clk: in std_logic;
        q2, q1, q0: buffer std_logic;
        v2, v1, v0, c: out std_logic);
end mod 6;

architecture amod6 of mod6 is
begin
    cct_mod6: process (q2, q1, q0, v, clk)
    begin
        if rising_edge(clk) then
q2 <= (q2 and (not q1) and (not q0)) or
      (q2 and (not q1) and (not v)) or
      ((not q2) and q1 and q0 and v);

q1 <= ((not q2) and q1 and (not q0)) or
      ((not q2) and q1 and (not v)) or
      ((not q2) and (not q1) and q0 and v);

q0 <= ((not q2) and (not q0) and v) or
      ((not q1) and (not q0) and v) or
      ((not q2) and q0 and (not v)) or
      ((not q1) and q0 and (not v));
end if;
v2 <= q2;
```

```
    v1 <= q1;
    v0 <= (q2 and q1) or q0;
     c <= not (q2 or q1 or q0);
end process ct_mod6;
end amod6;
```

35) Write a VHDL code for the following combinational circuits.

a) $F = AB + A'B'$

→
```
library ieee;
use ieee.std_logic_1164.all;

entity CombinationalCircuit1 is
    port (
        A,B: in std_logic;
        F : out std_logic);
end entity CombinationalCircuit1;

architecture Behavioral of CombinationalCircuit1
is
begin
    F <= A and B or (not A and not B);
end architecture Behavioral;
```

(b) $F(x,y,z) = \Sigma(1,3,6,7)$ → sum of product
(c) $F(x,y,z) = \Sigma(1,3,4,6)$ → sum of product
(d) $F(x,y,z) = \Pi(0,2,5,7)$ → product of sum

Solution:

(b)

→

| x \ yz | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | | 1 | 1 | |
| 1 | | - | 1 | 1 |

$F = x'z + xy$

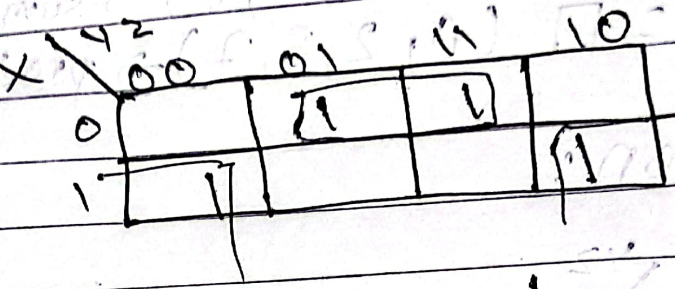
VHDL code:

→
```
library ieee;
use ieee.std_logic_1164.all;

entity CombinationalCircuit2 is
port (
    A, B : in std_logic;
    F : out std_logic);
end entity CombinationalCircuit2;

architecture Behavioral of CombinationalCircuit2
is
begin
    F <= (not A and B) or (A and B);
end architecture Behavioral;
```

(c) Soln




$F = x'z + xz'$

VHDL code:

```
library ieee;
use ieee.std_logic_1164.all;

entity circuit3 is
    port
        (A,B : in std_logic;
         F : out std_logic);
end entity circuit3;

architecture Behavioral of circuit3 is
begin
    F<= (not A and B) or (A and not B
end architecture Behavioral;
```

(d) Soln

| x \ y2 | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 | 1 |

$(x+z)(x'+z')$

VHDL code:

```
library ieee;
use ieee.std_logic_1164.all;

entity circuit4 is
    port
        (A, B: in std_logic;
         F: out std_logic);
    end entity circuit4;

architecture Behavioral of circuit4 is
begin
    F <= (A or B) and (not A or not B);
end architecture Behavioral;
```