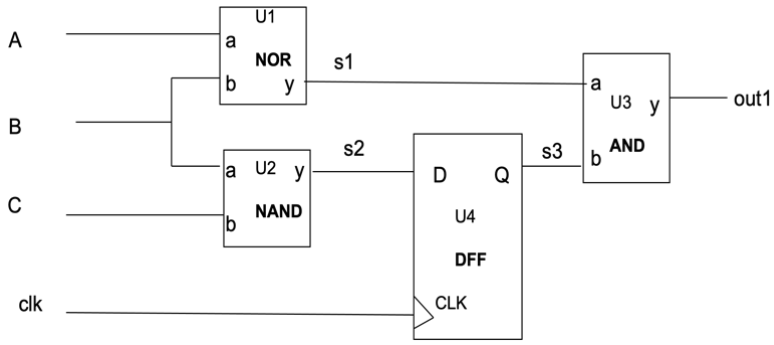
1. [20 pts] Complete the entity and architecture pair of VHDL structural description for a structural representation of a "top_level" circuit shown below. You need to declare the components (i.e., N**AND** for 2-input NAND gate, **AND** for 2-input AND gate, **NOR** for 2-input NOR gate, and **DFF** for D flip-flop) used in the schematic diagram. Assume these basic components have been compiled. Ensure all signals and ports of **STD_LOGIC** type and labeled in the schematic diagram match your VHDL code. Use **BY-NAME** method of port mapping.




```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.all;
ENTITY top_level IS
    PORT (A, B, C, CLK : in std_logic;
          Out1: out std_logic);

END ENTITY top_level;

    architecture structure of top_level is
        -- Component and signal declarations

        component NAND
            port (a, b: in std_logic;
                  y: out std_logic);
        end component;

        component AND
            port (a,b: in std_logic;
                  y: out std_logic);
        end component;

        component NOR
            port (a,b: in std_logic;
                  y: out std_logic);
        end component;

        component DFF
            port (d, clk: in std_logic;
                  q: out std_logic);
        end component;

        signal s1, s2, s3: std_logic

    begin
-- Component instantiations statements

    u1: NOR
```

```
            port map (a=>A, b=>B, y=>s1);
    u2: NAND
            port map (a=>B, b=>C, y=>s2);
    u3: AND
            port map (a=>s1, b=>s3, y=>out1);
    u4: DFF
            port map (d=>s2, clk=>CLK, q=>s3);

    end structure;
```

2. [20 pts] The VHDL compiler has found multiple errors with the following code. Identify 10 errors. Identifying non-errors will result in a loss of 2 points.

Type your corrected line # and VHDL code in the provided text box.

```
1.  library ieee
2.  use ieee.std_logic_1164.all;
3.
4.  entity priority is
5.  port (x: in std_logic_vector (3 downto 0),
6.          a: out std_logic_vector (3 downto 0));
7.  end priority;
8.
9.  architecture looparch of priority
10. begin
11.         p0: process (x)
12.
13.             a <= 00;
14.             for k in 3 downto 1
15.                if x(k) = 0 then
16.                   a(k) <= 1;
17.                elsif
18.                   a(k) <= 0;
19.                end if;
20.             end loop;
21.
22. end looparch;
```

```
1.  library ieee
2.  use ieee.std_logic_1164.all;
3.
4.  entity priority is
5.  port (x: in std_logic_vector (3 downto 0);
6.          a: out std_logic_vector (3 downto 0));
7.  end priority;
8.
9.  architecture looparch of priority is
10. begin
11.         p0: process (x)
12.         begin
13.             a <= "00";
14.             for k in 3 downto 1 loop
15.                if x(k) = '0' then
16.                   a(k) <= '1';
17.                else
18.                   a(k) <= '0';
19.                end if;
20.             end loop;
21.         end process;
22. end looparch;
```

3. [20 pts] Complete VHDL that implements two functions: F(A,B,C) = AB + B'C' and G(A,B,C) = A'B' + BC

a) [6 pts] Use ***concurrent VHDL code*** (*use the exact Boolean expression* F(A,B,C) = AB + B'C' and G(A,B,C) = A'B' + BC

```
entity function is
        port ( A,B,C : in std_logic;
                F,G : out std_logic);
end function;

architecture concurrent of function is
begin
            F <= (A AND B) OR (NOT B AND NOT C);

            G <= (NOT A AND NOT B) OR (B AND C);
end concurrent;
```

b) [7 pts] Use ***if*** statement (*use the exact Boolean expression* F(A,B,C) = AB + B'C' and G(A,B,C) = A'B' + BC)

```
entity function is
        port ( A,B,C : in std_logic;
                F,G : out std_logic);
end function;

architecture behavior_1 of function is
begin
    proc1: process(A,B,C)

    begin

        if (A = '1' and B = '1') then
                F <= '1';
        elsif (B = '0' and C = '0') then
                F <= '1';
        else
                F <= '0';
        end if;

        if (A = '0' and B = '0') then
                G <= '1';
        elsif (B = '1' and C = '1') then
                G <= '1';
        else
                G <= '0';
        end if;
  end process proc1;
end behavior_1;
```

c) [7 pts] Complete VHDL that implements two functions: F(A,B,C) = AB + B'C' and G(A,B,C) = A'B' + BC

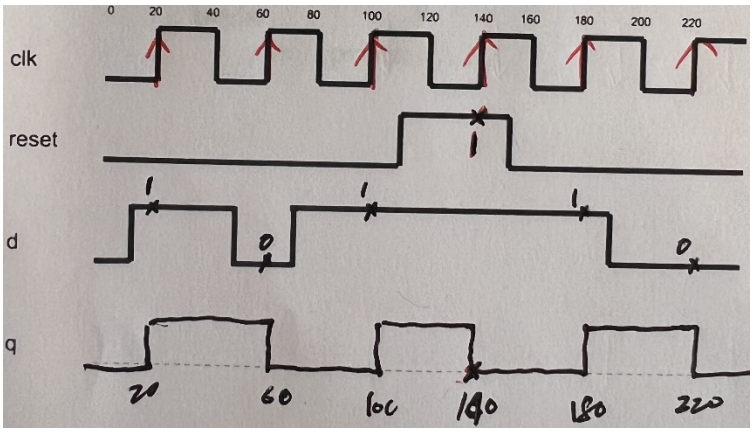
Use ***case*** statement. Note: *Use two case statements; one for F and the other for G.*

```
entity function is
        port ( A,B,C : in std_logic;
                F,G : out std_logic);
end function;

architecture behavior_2 of function is

signal ABC: std_logic_vector(2 downto 0);
begin
        ABC <= (A,B,C);
        my_proc: process (ABC)

        begin

                case (ABC) is
                when "000" => F <= '1';
                when "100" => F <= '1';
                when "110" => F <= '1';
                when "111" => F <= '1';
                when others => F <= '0';
                end case;

                case (ABC) is
                when "000" => G <= '1';
                when "001" => G <= '1';
                when "011" => G <= '1';
                when "111" => G <= '1';
                when others => G <= '0';
                end case;
        end process my_proc;
end behavior_2;
```

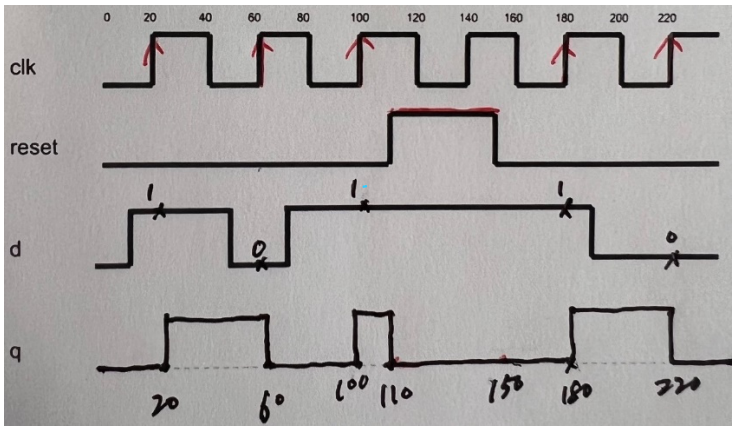
4. [20 pts] Complete the waveform for the output **q** for the following different architectures of D flip-flop.

```
ENTITY dff_reset IS
   PORT (reset, clk: IN BIT;
         d: IN STD_LOGIC;
         q: OUT STD_LOGIC);
END ENTITY dff_reset;
```

a) [10 pts]
```
ARCHITECTURE a_behavior OF dff_reset IS
 BEGIN
     PROCESS (clk)
     BEGIN
          IF (clk'EVENT AND clk = '1') THEN
           IF (reset = '1') THEN
                   q <=  '0';
           ELSE
                   q <= d;
           END IF;
          END IF;
    END PROCESS;
END ARCHITECTURE a_behavior;
```



b) [10 pts]
```
ARCHITECTURE b_behavior OF dff_reset IS
 BEGIN
   PROCESS (clk, reset)
   BEGIN
        IF (reset = '1') THEN
           q <=  '0';
        ELSIF (clk'EVENT AND clk = '1') THEN
           q <= d;
        END IF;
   END PROCESS;
END ARCHITECTURE b_behavior;
```



5. [20 pts] Construct the **truth table**. Use K-maps to derive the sum-of-products for **b(3), b(2), b(1), and b(0).**

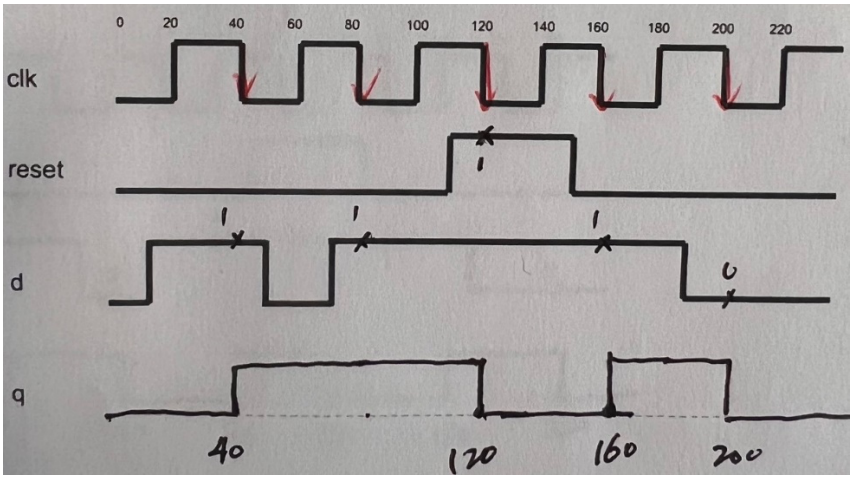
```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
entity decoder1
     port(a : in BIT_VECTOR(1 downto 0);
          EN: in BIT;
          b : out BIT_VECTOR(3 downto 0));
end decoder1;

architecture bhv1 of decoder1 is
begin
  process(a, EN)
  begin
    if (EN = '0') then
          if (a="01") then
               b <= "0010";
          elsif (a="10") then
               b <= "0100";
          else
               b <= "0000";
          end if;
     else
          if (a="00") then
               b <= "0001";
          elsif (a="11") then
               b <= "1000";
          else
               b <= "1111";
     end if;
    end if;
  end process;
end bhv1;
```

EN	a(1)	a(0)	b(3)	b(2)	b(1)	b(0)
0	0	0	0	0	0	0
0	0	1	0	0	1	0
0	1	0	0	1	0	0
0	1	1	0	0	0	0
1	0	0	0	0	0	1
1	0	1	1	1	1	1
1	1	0	1	1	1	1
1	1	1	1	0	0	0

b(3) = EN a(1) + EN a(0)
b(2) = a(1) a(0)' + EN a(1)' a(0)
b(1) = a(1)' a(0) + EN a(1) a(0)'
b(0) = EN a(1)' + EN a(0)'

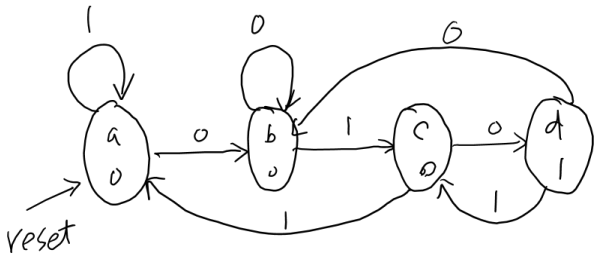
[Bonus]
6. [10 pts] Complete the waveform for the output q for the following different architectures of D flip-flop.

```
ENTITY dff_reset IS
   PORT (reset, clk: IN BIT;
              d: IN BIT;
              q: OUT BIT);
END ENTITY dff_reset;


ARCHITECTURE c_behavior OF dff_reset IS
BEGIN
    PROCESS (clk)
    BEGIN
          IF (clk'EVENT AND clk = '0') THEN  --  Note: Negative-edge trigger.
            IF (reset = '1') THEN
                    q <= '0';
            ELSE
                    q <= d;
           END IF;
         END IF;
   END PROCESS;
   END ARCHITECTURE c_behavior;
```



7. Let's construct the sequence detector for the sequence ***010*** using **Moore** state machine. The output of the State machine depends only on the present state. The output of state machine is only updated at the clock edge. ***The FSM uses a "positive" reset (i.e., rst = 1); the clock has a higher priority than the reset. All flip-flops are rising edge-triggered.***

The Moore state machine require **four states *a, b, c,* and *d*** to detect the ***010*** sequence. The initial state is ***state a (sta)***.
a) Draw the state transition graph (STG).



Complete the entity and architecture pair of VHDL description for the Moore machine.

```
library ieee;
use ieee.std_logic_1164.all;
entity seq_detect is
        port (clk, reset, x : in std_logic;
        y : out std_logic);
end seq_detect;

architecture enum of seq_detect is
type state is (sta, stb, stc, std); -- use "sta" to represent 'state a", etc.
signal present_state, next_state : state;

begin

nx_state: process (present_state, x) -- next state process
begin
-- use CASE and then IF statement
```

b)

```
CASE present_state IS
   WHEN sta =>
      IF (x = '0') THEN
         next_state <= stb;
      ELSE
         next_state <= sta;
      END IF;
   WHEN stb =>
      IF (x = '1') THEN
         next_state <= stc;
      ELSE
         next_state <= stb;
      END IF;
   WHEN stc =>
      IF (x = '0') THEN
         next_state <= std;
      ELSE
         next_state <= sta;
      END IF;
   WHEN std =>
```

```
            IF (x= '1') THEN
               next_state <= stc;
            ELSE
               next_state <= stb;
            END IF;
         WHEN OTHERS =>
               next_state <= sta;
      END CASE;
   end process nx_state;

   state_reg: process (clk, reset) -- state register process
      begin
```

c)

```
      IF (clk'EVENT AND clk = '1') THEN
        IF (reset = '1') THEN
              present_state <= state_a;
        ELSE
              present_state <= next_state;
        END IF;
   end process state_reg;
```

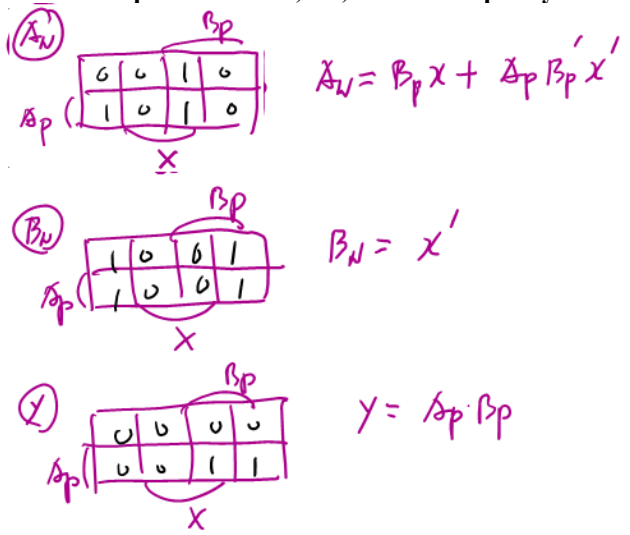
-- Concurrent VHDL for output assignment

d)

```
   y <= '1' WHEN (present_state = std) ELSE '0';
end enum;
```

(e) **Construct state table for the FSM machine**. There are 4 states, sta, stb, stc, and std. You need **2 flip-flops, A and B** for implementation. Use state assignment (A B) = (0 0) for state **sta**, (0 1) for state **stb**, (1 0) for state **stc**, and (1 1) for state **std**. Note: In your state table, use Ap for the present state A, Bp for the present state B, An for the next state A, Bn for the next state B, "x" for the input, and "y" for the output. **Derive Boolean equation for An, Bn, and the output "y".**

Ap	Bp	X	An	Bn	Y
0	0	0	0	1	0
0	0	1	0	0	0
0	1	0	0	1	0
0	1	1	1	0	0
1	0	0	1	1	0
1	0	1	0	0	0
1	1	0	0	1	1
1	1	1	1	0	1



(f) Sketch the FSM logic design.