Assignment 1

IT451 : COA LAB

SOUMABRATA
BHATTACHARYA
IT, 4th Semester
ID: 510817021 (Hx-20)

04-02-2019

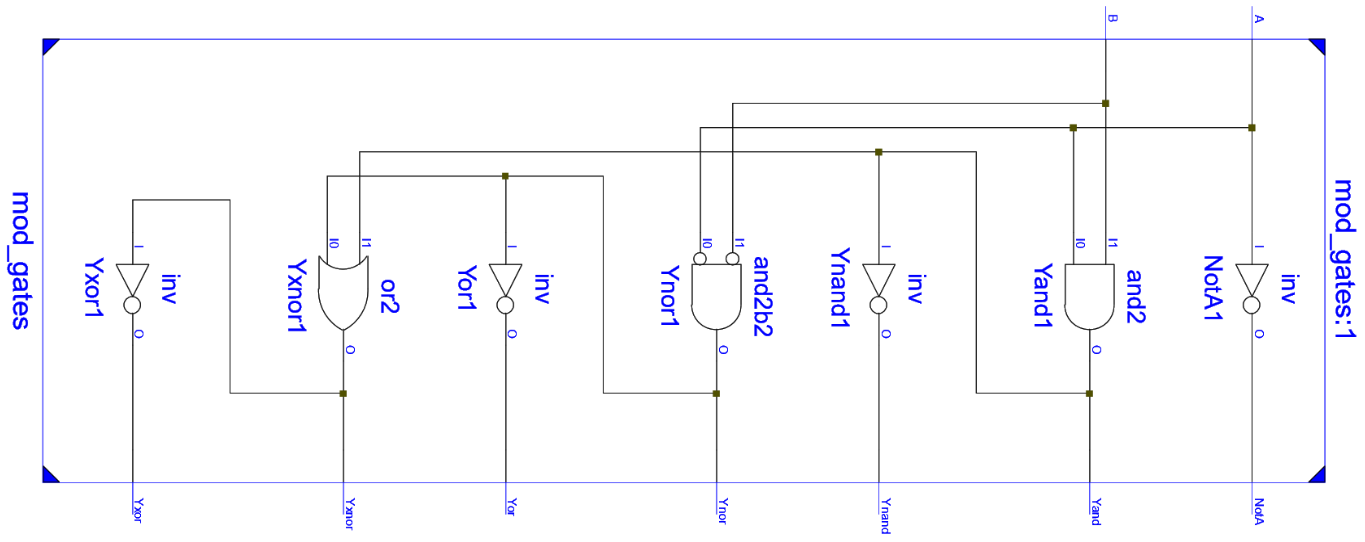
Question 1

Design and Simulate the Behavioral Model of basic gate (AND, OR, NOT), universal gates, XOR, XNOR gates

VHDL Module: *mod_gates.vhd*

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

entity mod_gates is
    Port ( A, B               : in  STD_LOGIC;
           Yand, Yor, NotA    : out  STD_LOGIC;
           Ynand, Ynor  : out  STD_LOGIC;
           Yxor, Yxnor  : out  STD_LOGIC);
end mod_gates;

architecture Behavioral of mod_gates is

begin
      Yand  <= '1' when (A = '1' and B = '1') else '0';
      Yor   <= '0' when (A = '0' and B = '0') else '1';
      NotA  <= '1' when (A = '0')             else '0';

      Ynand <= '0' when (A = '1' and B = '1') else '1';
      Ynor  <= '1' when (A = '0' and B = '0') else '0';

      Yxor  <= '0' when ((A = '0' and B = '0') or (A = '1' and B = '1')) else '1';
      Yxnor <= '1' when ((A = '0' and B = '0') or (A = '1' and B = '1')) else '0';
end Behavioral;
```

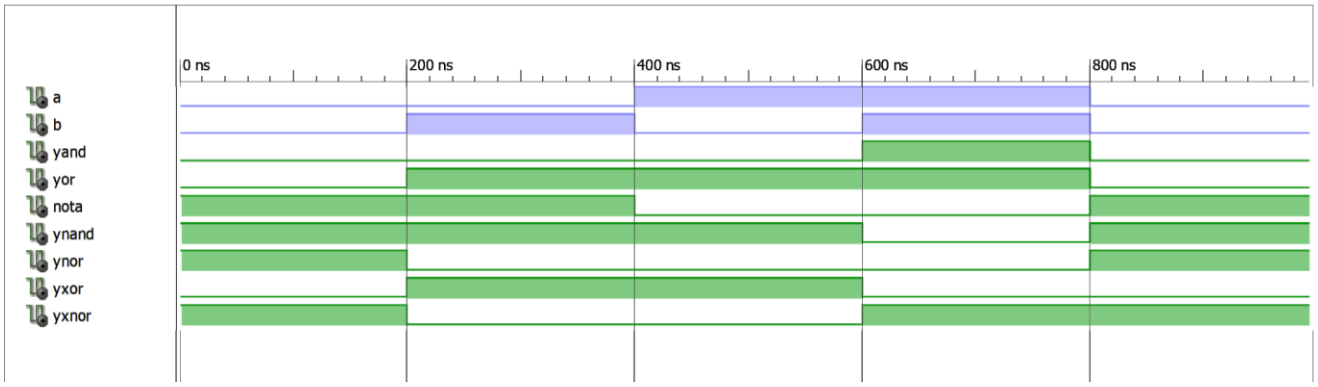
RTL Schematic:



VHDL Test Bench: *tb_gates.vhd*

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY tb_gates IS
END tb_gates;

ARCHITECTURE behavior OF tb_gates IS
    COMPONENT mod_gates
    PORT(
         A        : IN  std_logic;
         B        : IN  std_logic;
         Yand     : OUT  std_logic;
         Yor      : OUT  std_logic;
         NotA     : OUT  std_logic;
         Ynand    : OUT  std_logic;
         Ynor     : OUT  std_logic;
         Yxor     : OUT  std_logic;
         Yxnor    : OUT  std_logic
        );
    END COMPONENT;

   signal A : std_logic := '0';
   signal B : std_logic := '0';

   signal Yand    : std_logic;
   signal Yor     : std_logic;
   signal NotA    : std_logic;
   signal Ynand   : std_logic;
   signal Ynor    : std_logic;
   signal Yxor    : std_logic;
   signal Yxnor   : std_logic;

BEGIN
   uut: mod_gates PORT MAP (
          A       => A,
          B       => B,
          Yand    => Yand,
          Yor     => Yor,
          NotA    => NotA,
          Ynand   => Ynand,
          Ynor    => Ynor,
          Yxor    => Yxor,
            Yxnor   => Yxnor
          );

      process
      begin
         A <= '0', '1' after 400ns, '0' after 800ns;
         B <= '0', '1' after 200ns, '0' after 400ns, '1' after 600ns, '0' after 800ns;
         wait;
      end process;
   END;
```

Simulation:



Discussions:

Components that can be designed based on another existing component is connected accordingly, e.g. an inverter on an AND gate produces NAND gate.

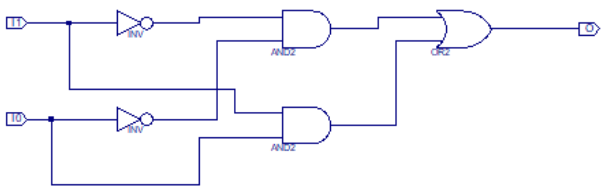
Question 2

Design and Simulate the Behavioral Model of a 1-bit Magnitude Comparator.
It should accept two input bits and give three output lines (greater, less, equal).

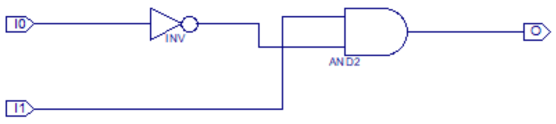
VHDL Module: *mod_magComp1Bit.vhd*

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

entity mod_magComp1Bit is
      Port ( A, B        : in  STD_LOGIC;
             eq, ls, gr : out  STD_LOGIC);
      end mod_magComp1Bit;

architecture Behavioral of mod_magComp1Bit is

begin
      process(A, B)
      begin
            gr <= '0'; eq <= '0'; ls <= '0';
            if    (A > B) then gr <= '1';
            elsif (A = B) then eq <= '1';
            else               ls <= '1';
            end if;
      end process;
end Behavioral;
```

Schematic:



LUT for Equal Case

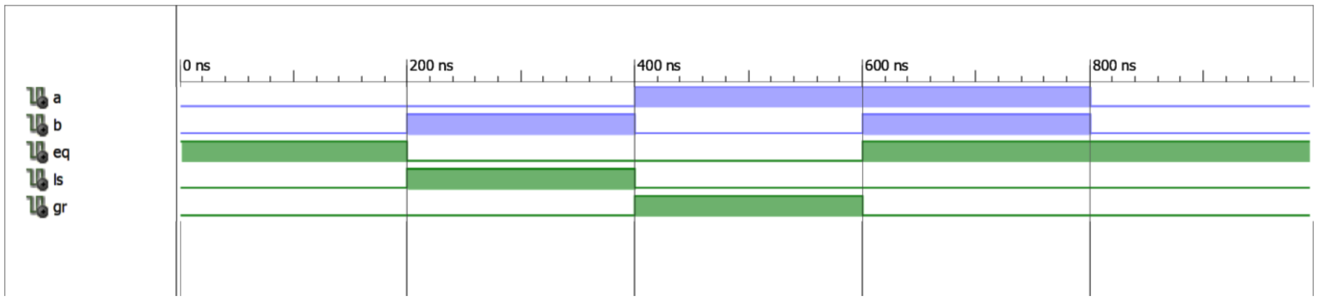


LUT for Greater/Less Case

VHDL Test Bench: *tb_magComp1Bit.vhd*

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY tb_magComp1Bit IS
END tb_magComp1Bit;

ARCHITECTURE behavior OF tb_magComp1Bit IS
    COMPONENT mod_magComp1Bit
    PORT(
         A : IN  std_logic;
         B : IN  std_logic;
         eq : OUT  std_logic;
         ls : OUT  std_logic;
         gr : OUT  std_logic
        );
    END COMPONENT;

   signal a : std_logic := '0';
   signal b : std_logic := '0';

   signal eq : std_logic;
   signal ls : std_logic;
   signal gr : std_logic;

BEGIN
   uut: mod_magComp1Bit PORT MAP (
          A => a,
          B => b,
          eq => eq,
          ls => ls,
          gr => gr
        );

   process
   begin
      a <= '0', '1' after 400ns, '0' after 800ns;
      b <= '0', '1' after 200ns, '0' after 400ns , '1' after 600ns, '0' after
      800ns;
      wait;
   end process;
END;
```

Simulation:



Discussions:

A XNOR gate operation is used to device the Equal Case while an AND gate operation with one of its inputs inverted is used to device the Greater than and Less than Cases. In either case, the alternating inputs are inverted.

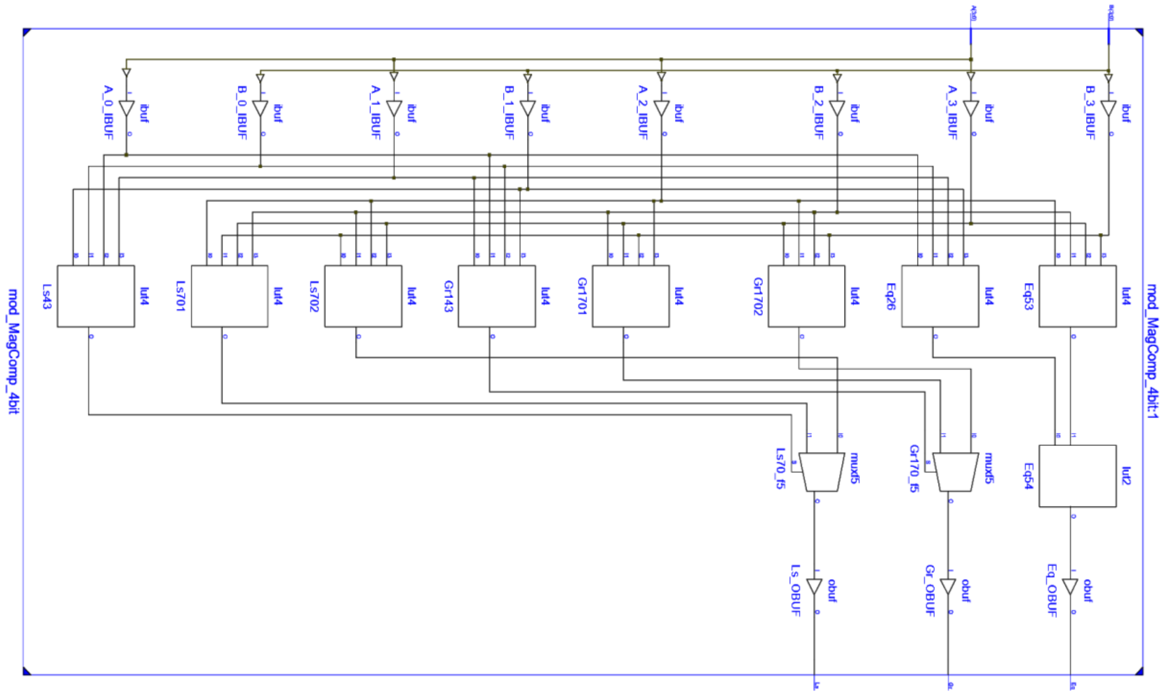
Question 3

Design the Behavioral Model (with process statement) of a 4-bit Magnitude Comparator and Simulate.

VHDL Module: *mod_MagComp_4bit.vhd*

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

entity mod_MagComp_4bit is
      Port ( A, B       : in  STD_LOGIC_VECTOR (3 downto 0);
             Gr, Eq, Ls : out  STD_LOGIC);
end mod_MagComp_4bit;

architecture Behavioral of mod_MagComp_4bit is

begin
      process(A, B)
      begin
            Gr <= '0'; Eq <= '0'; Ls <= '0';
            if    (A > B) then Gr <= '1';
            elsif (A = B) then Eq <= '1';
            elsif (A < B) then Ls <= '1';
            end if;
      end process;
end Behavioral;
```

Schematic:

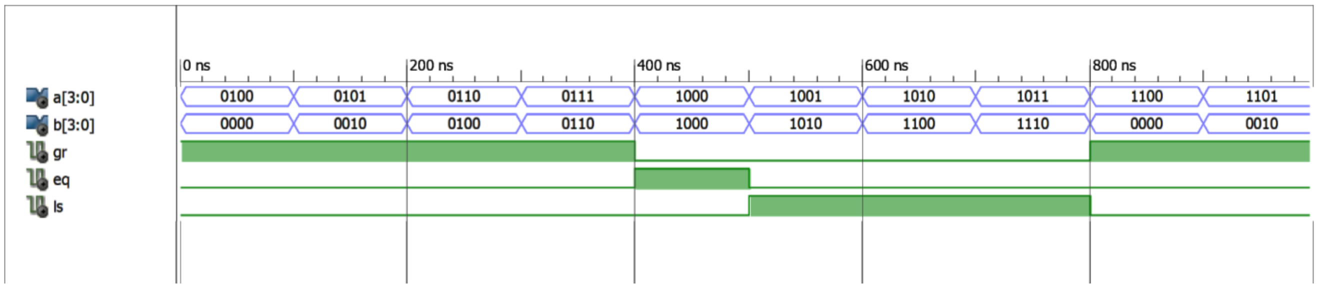


VHDL Test Bench:

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

ENTITY tb_MagComp_4bit IS
END tb_MagComp_4bit;

ARCHITECTURE behavior OF tb_MagComp_4bit IS
      COMPONENT mod_MagComp_4bit
          PORT(
               A : IN  std_logic_vector(3 downto 0);
               B : IN  std_logic_vector(3 downto 0);
               Gr : OUT  std_logic;
               Eq : OUT  std_logic;
               Ls : OUT  std_logic
              );
      END COMPONENT;

signal a : std_logic_vector(3 downto 0) := (others => '0');
signal b : std_logic_vector(3 downto 0) := (others => '0');

signal gr : std_logic;
signal eq : std_logic;
signal ls : std_logic;

BEGIN
      uut: mod_MagComp_4bit PORT MAP (
          A => a,
          B => b,
          Gr => gr,
          Eq => eq,
          Ls => ls
        );

      proc_A: process
      begin
            a <= "0100";
            while a >= "0000" loop
                  wait for 100ns;
                  a <= std_logic_vector(unsigned(a) + 1);
            end loop;
            wait;
      end process;
```

```
        proc_B: process
        begin
              b <= "0000";
              while b >= "0000" loop
                    wait for 100ns;
                    b <= std_logic_vector(unsigned(b) + 2);
              end loop;
              wait;
        end process;
  END;
```

<u>Simulation:</u>



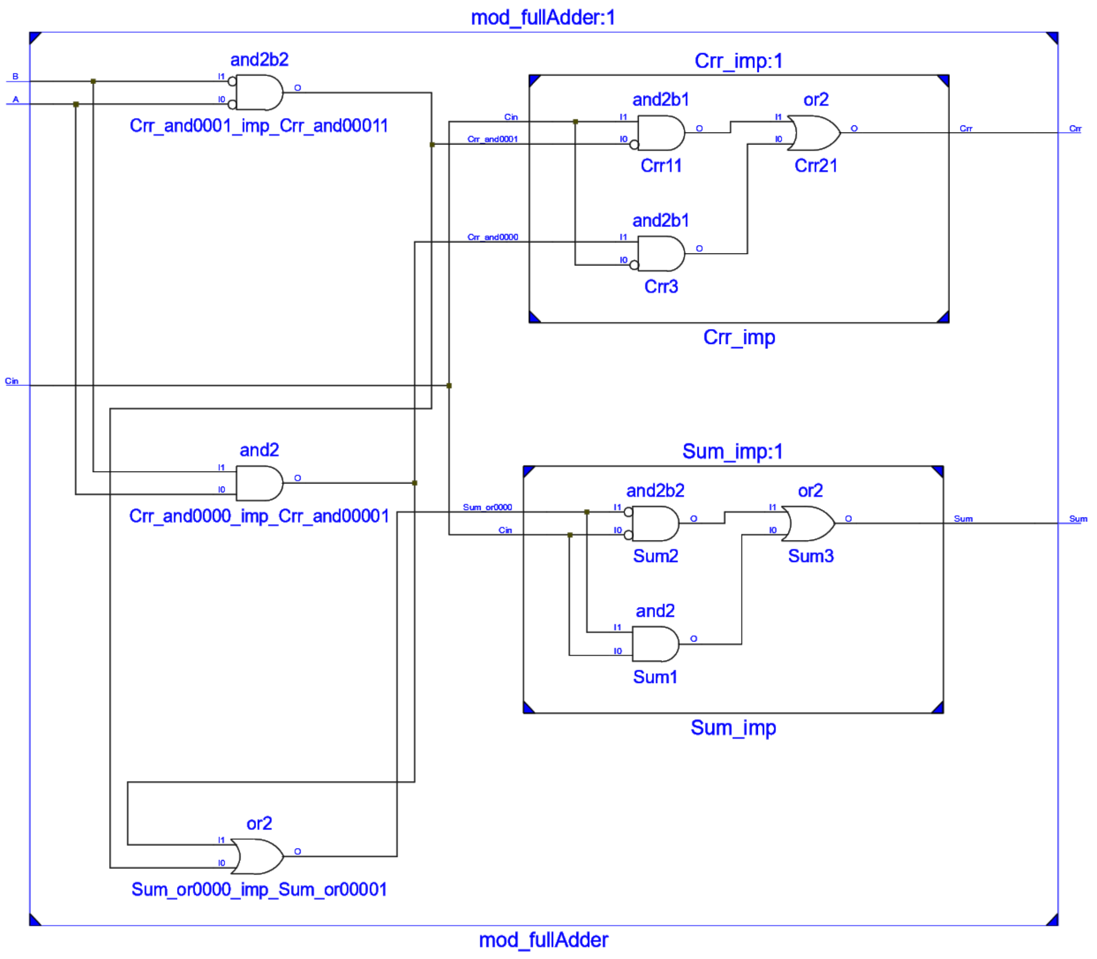
Question 4

Design and Simulate the Behavioral Model of a 1-bit Full Adder.

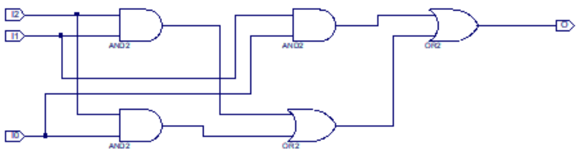
VHDL Module: *mod_fullAdder.vhd*

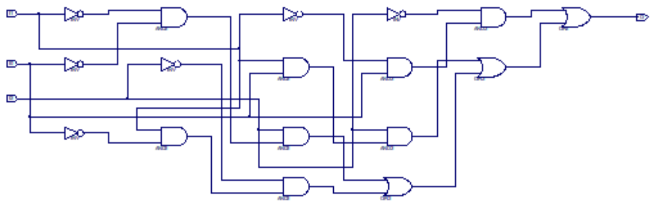
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity mod_fullAdder is
    Port ( Cin, A, B    : in  STD_LOGIC;
           Sum, Crr     : out  STD_LOGIC);
end mod_fullAdder;

architecture Behavioral of mod_fullAdder is

begin
      process(Cin, A, B)
      begin
            if(Cin = '0') then
                  if((A = '0' and B = '0') or (A = '1' and B = '1')) then Sum <= '0';
                  else Sum <= '1';
                  end if;
                  if(A = '1' and B = '1') then Crr <= '1';
                  else Crr <= '0';
                  end if;
            else
                  if((A = '0' and B = '0') or (A = '1' and B = '1')) then Sum <= '1';
                  else Sum <= '0';
                  end if;
                  if(A = '0' and B = '0') then Crr <= '0';
                  else Crr <= '1';
                  end if;
            end if;
      end process;
end Behavioral;
```

RTL Schematic:



Technology Schematic:





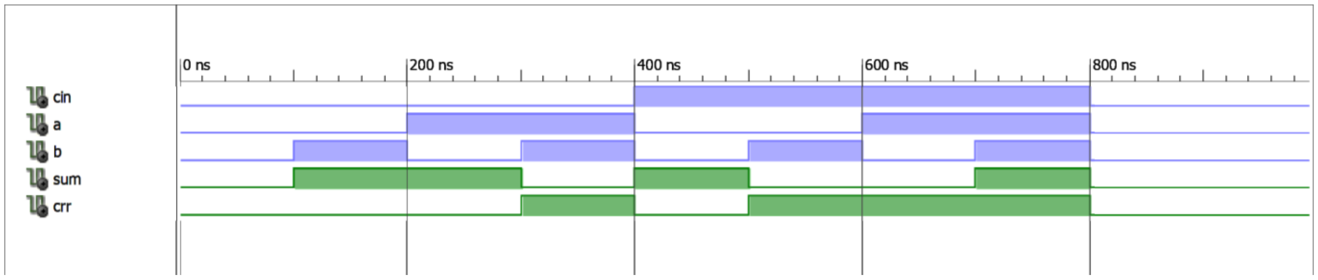
LUT for Carry

LUT for Sum

VHDL Test Bench: *tb_fullAdder.vhd*

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY tb_fullAdder IS
END tb_fullAdder;

ARCHITECTURE behavior OF tb_fullAdder IS
    COMPONENT mod_fullAdder
    PORT(
         Cin      : IN  std_logic;
         A        : IN  std_logic;
         B        : IN  std_logic;
         Sum      : OUT  std_logic;
         Crr      : OUT  std_logic
        );
    END COMPONENT;

   signal cin     : std_logic := '0';
   signal a       : std_logic := '0';
   signal b       : std_logic := '0';

   signal sum     : std_logic;
   signal crr     : std_logic;

BEGIN
   uut: mod_fullAdder PORT MAP (
          Cin     => cin,
          A       => a,
          B       => b,
          Sum     => sum,
          Crr     => crr
        );

   stim_proc: process
   begin
      cin   <= '0', '1' after 400ns, '0' after 800ns;
      a <= '0', '1' after 200ns, '0' after 400ns, '1' after 600ns, '0' after 800ns;
      b <= '0', '1' after 100ns, '0' after 200ns, '1' after 300ns, '0' after 400ns, '1'
      after 500ns, '0' after 600ns, '1' after 700ns, '0' after 800ns;
      wait;
   end process;
END;
```

Simulation:



Discussions:

Full adder isn't composed of smaller unit components, i.e. half adders in this case. Two components are generated: one for sum and the other for carry. The sum component produces sum of any three input bits, not necessarily inputs of a full adder. The same goes for the carry component.

Question 5

Design the Structural Model of a Full Adder using Half Adder as a component.

VHDL Module:

mod_halfAdder.vhd

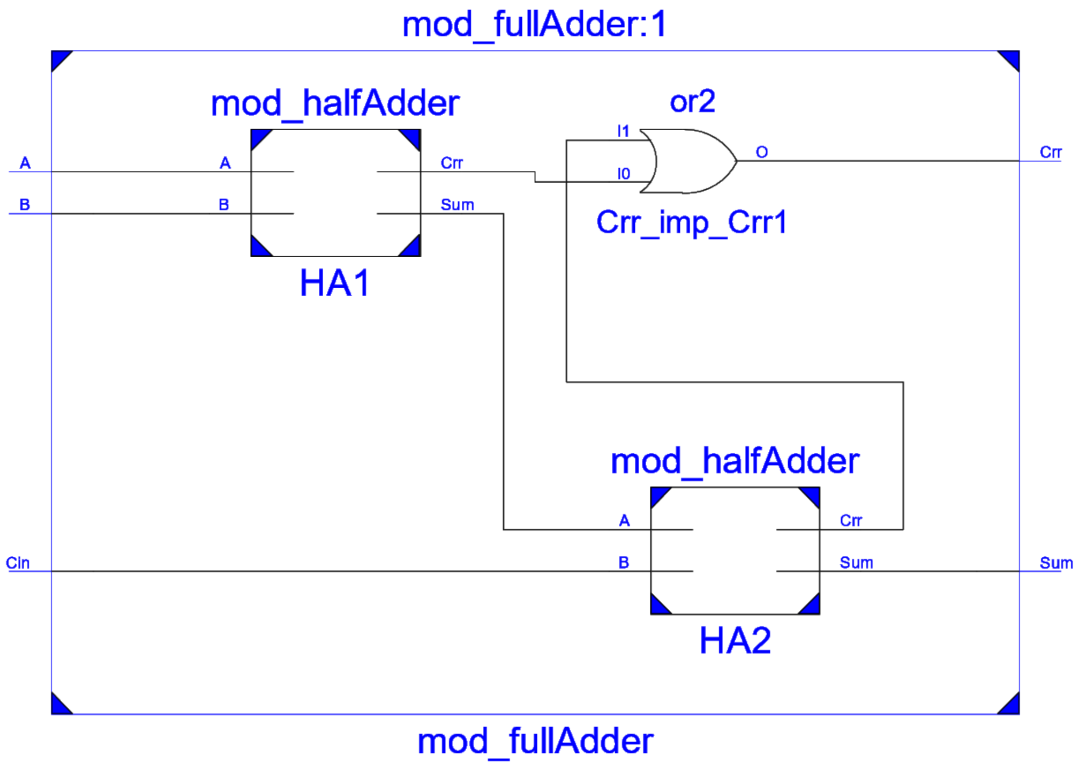
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity mod_halfAdder is
      Port ( A, B       : in  STD_LOGIC;
            Sum, Crr    : out  STD_LOGIC);
      end mod_halfAdder;

architecture Behavioral of mod_halfAdder is
begin
      process(A, B)
      begin
            if((A = '0' and B = '0') or (A = '1' and B = '1')) then Sum <= '0';
            else Sum <= '1';
            end if;
            if(A = '1' and B = '1') then Crr <= '1';
            else Crr <= '0';
            end if;
      end process;
end Behavioral;
```

mod_fullAdder.vhd

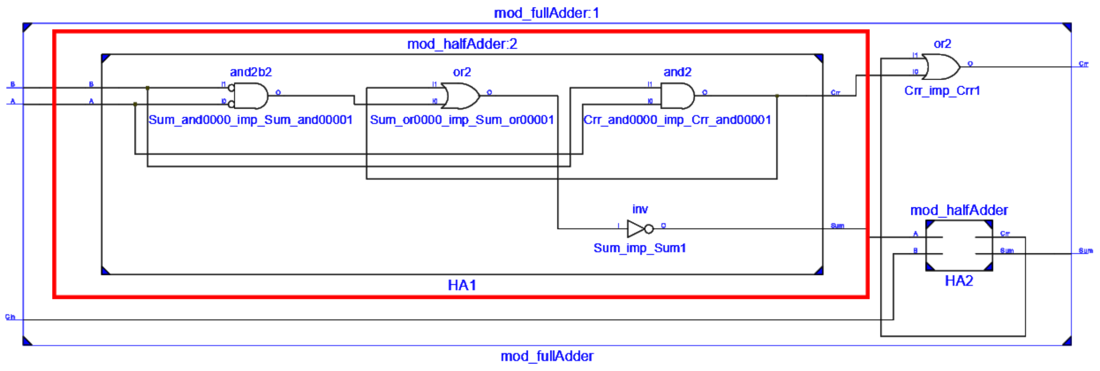
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity mod_fullAdder is
      Port ( Cin, A, B  : in  STD_LOGIC;
            Sum, Crr    : out  STD_LOGIC);
end mod_fullAdder;

architecture Structural of mod_fullAdder is
      component mod_halfAdder is
            Port ( A, B       : in  STD_LOGIC;
                   Sum, Crr   : out  STD_LOGIC);
      end component;

signal S1, C1, C2 : STD_LOGIC ;
```

```
begin
      HA1 : mod_halfAdder port map(A, B, S1, C1);
      HA2 : mod_halfAdder port map(S1, Cin, Sum, C2);
      Crr <= C1 or C2;
end Structural;
```

RTL Schematic:



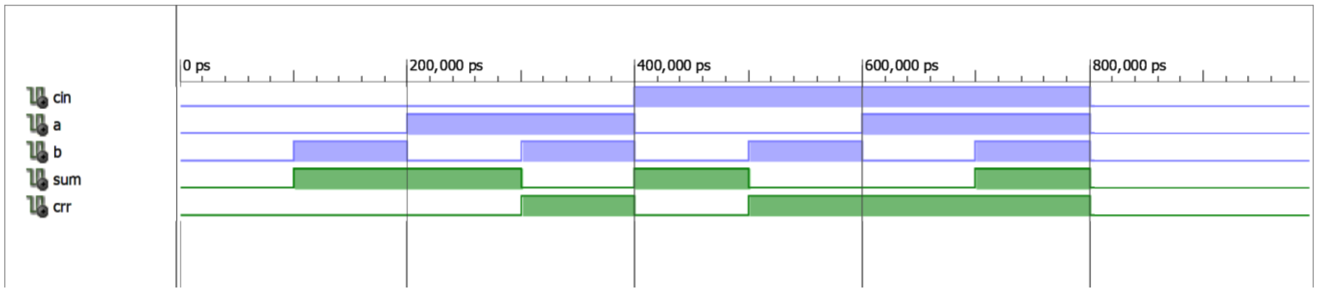
Half Adder Component



VHDL Test Bench: *tb_fullAdder.vhd*

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY tb_fullAdder IS
END tb_fullAdder;

ARCHITECTURE behavior OF tb_fullAdder IS
    COMPONENT mod_fullAdder
          PORT(
               Cin      : IN  std_logic;
               A        : IN  std_logic;
               B        : IN  std_logic;
               Sum      : OUT  std_logic;
               Crr      : OUT  std_logic
              );
    END COMPONENT;

   signal cin     : std_logic := '0';
   signal a       : std_logic := '0';
   signal b       : std_logic := '0';

   signal sum     : std_logic;
   signal crr     : std_logic;

BEGIN
   uut: mod_fullAdder PORT MAP (
          Cin     => cin,
          A       => a,
          B       => b,
          Sum     => sum,
          Crr     => crr
        );

   stim_proc: process
   begin
      cin   <= '0', '1' after 400ns, '0' after 800ns;
      a <= '0', '1' after 200ns, '0' after 400ns, '1' after 600ns, '0' after 800ns;
      b <= '0', '1' after 100ns, '0' after 200ns, '1' after 300ns, '0' after 400ns,
      '1' after 500ns, '0' after 600ns, '1' after 700ns, '0' after 800ns;
      wait;
   end process;
END;
```

Simulation:



Discussions:

Here, two half adders are used as components to generate the full adder which is reflected in the design as well. The Sum of the first adder serves as an input of the second adder along with the carry around. The Carry is a disjunction of the carry outs of the two adders.

In comparison with the previous question, this is modular and implements reusability of pre-generated components (half adder module in this case).

Question 6

Design a 4-bit Binary Parallel Adder using Structural Design Flow hierarchically.

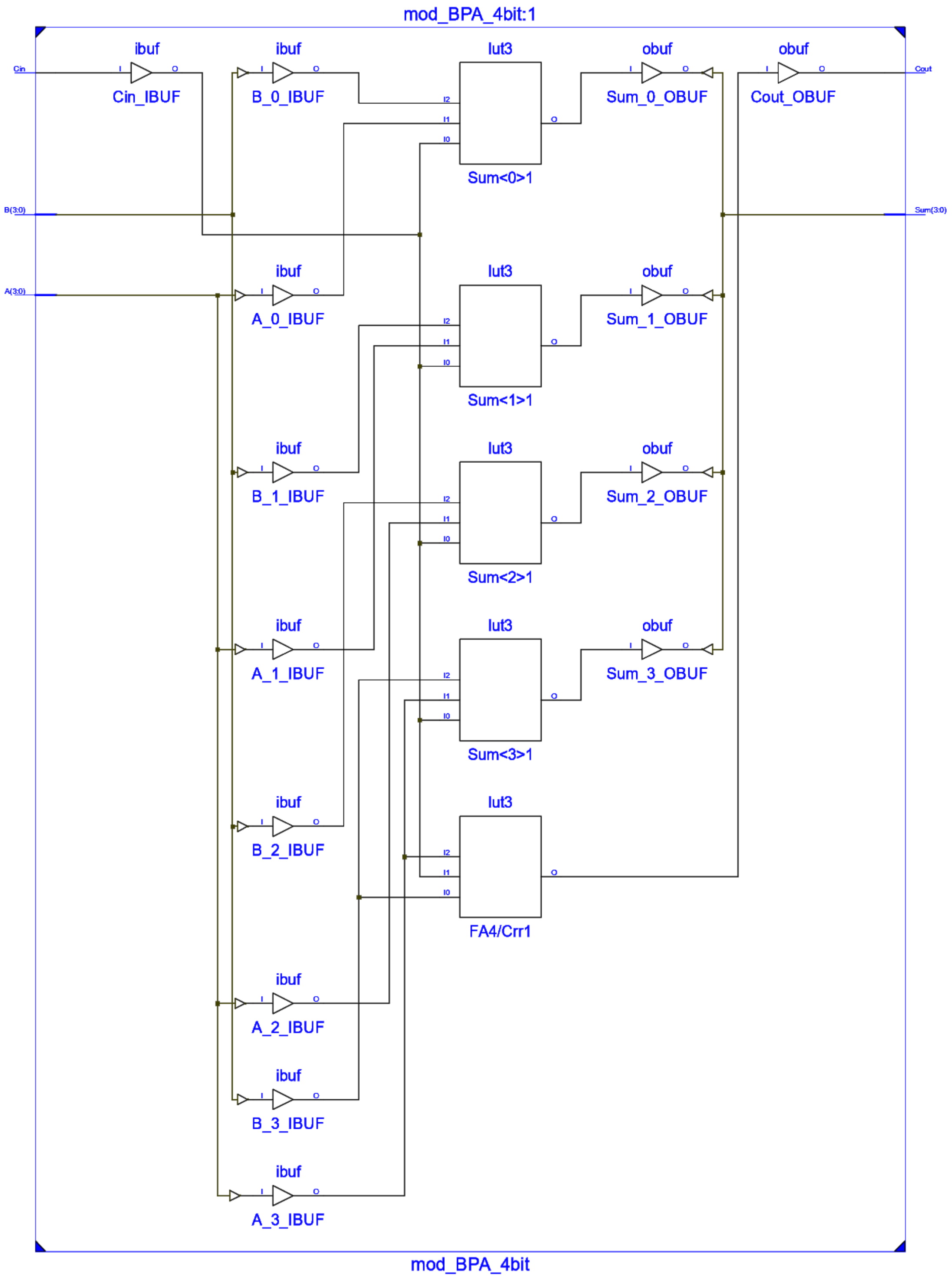
VHDL Module:

Copy of Source *mod_fullAdder.vhd* added from Question 5
Copy of Source *mod_halfAdder.vhd* added from Question 5

mod_BPA_4bit.vhd

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

entity mod_BPA_4bit is
    Port ( Cin    : in  STD_LOGIC;
           A, B   : in  STD_LOGIC_VECTOR (3 downto 0);
           Sum    : out  STD_LOGIC_VECTOR (3 downto 0);
           Cout   : out  STD_LOGIC);
end mod_BPA_4bit;

architecture Structural of mod_BPA_4bit is
      component mod_fullAdder is
            Port ( Cin, A, B  : in  STD_LOGIC;
                   Sum, Crr   : out  STD_LOGIC);
      end component;

signal C0, C1, C2 : STD_LOGIC;

begin
      FA1 : mod_fullAdder port map(Cin, A(0), B(0), Sum(0), C0);
      FA2 : mod_fullAdder port map(Cin, A(1), B(1), Sum(1), C1);
      FA3 : mod_fullAdder port map(Cin, A(2), B(2), Sum(2), C2);
      FA4 : mod_fullAdder port map(Cin, A(3), B(3), Sum(3), Cout);
end Structural;
```

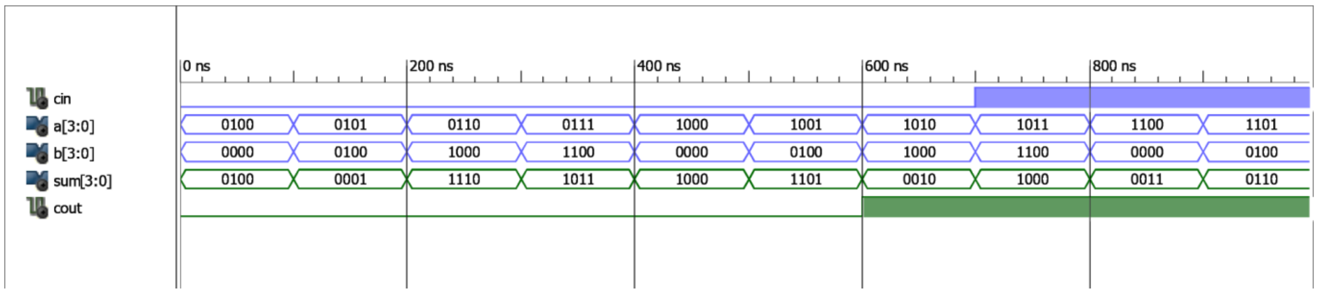
Technology Schematic:



<u>VHDL Test Bench:</u>   *tb_BPA_4bit.vhd*

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

ENTITY tb_BPA_4bit IS
END tb_BPA_4bit;

ARCHITECTURE behavior OF tb_BPA_4bit IS
    COMPONENT mod_BPA_4bit
          PORT(
               Cin      : IN  std_logic;
               A        : IN  std_logic_vector(3 downto 0);
               B        : IN  std_logic_vector(3 downto 0);
               Sum      : OUT  std_logic_vector(3 downto 0);
               Cout     : OUT  std_logic
              );
    END COMPONENT;

signal cin : std_logic := '0';
signal a : std_logic_vector(3 downto 0) := (others => '0');
signal b : std_logic_vector(3 downto 0) := (others => '0');
signal sum  : std_logic_vector(3 downto 0);
signal cout : std_logic;

BEGIN
   uut: mod_BPA_4bit PORT MAP (
          Cin     => cin,
          A       => a,
          B       => b,
          Sum     => sum,
          Cout    => cout
        );

   cin <= '0', '1' after 700ns;
   proc_A: process
   begin
      a <= "0100";
      while a >= "0000" loop
            wait for 100ns;
            a <= std_logic_vector(unsigned(a) + 1);
      end loop;
      wait;
   end process;
   proc_B: process
   begin
      b <= "0000";
      while b >= "0000" loop
            wait for 100ns;
            b <= std_logic_vector(unsigned(b) + 4);
      end loop;
      wait;
   end process;
END;
```

Simulation:



Discussions:

Four full adders output the same bit of four respective bits of the two inputs. In each case, the carry is carried up and added as an input to the next significant bit's adder. The last carry is the final carry out of the ripple adder.

Question 7 – a

Design the structural model of a BCD adder using previously designed BPA model as a component.

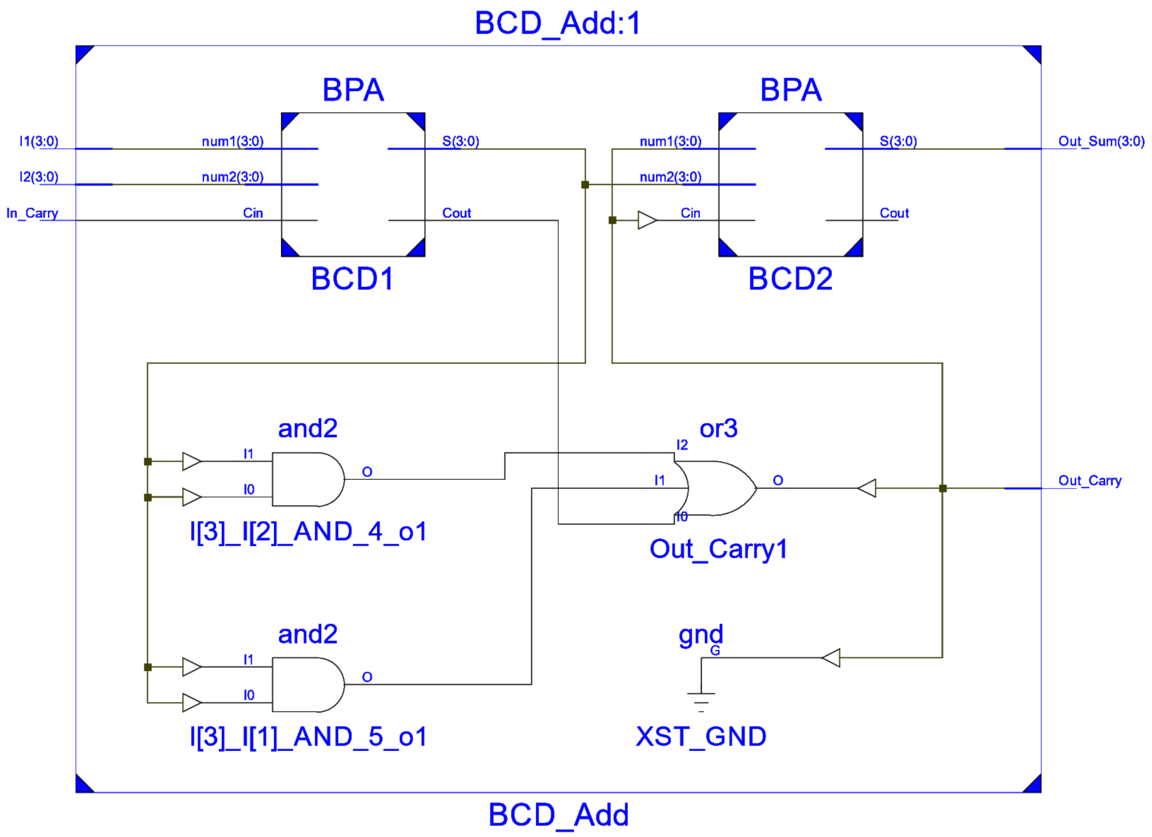
VHDL Module:

Copy of Source *mod_BPA_4bit.vhd* added from Question 6
Copy of Source *mod_fullAdder.vhd* added from Question 5
Copy of Source *mod_halfAdder.vhd* added from Question 5

mod_BCDAdder.vhd

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

entity mod_BCDAdder is
      Port ( Cin  : IN STD_LOGIC;
             A    : IN STD_LOGIC_VECTOR (3 downto 0);
             B    : IN STD_LOGIC_VECTOR (3 downto 0);
             Sum  : OUT STD_LOGIC_VECTOR (3 downto 0);
             Cout : INOUT STD_LOGIC);
      end mod_BCDAdder;

architecture Structural of mod_BCDAdder is
      Component mod_BPA_4bit
            Port ( Cin  : IN STD_LOGIC;
                   A, B : IN STD_LOGIC_VECTOR (3 downto 0);
                   Sum  : OUT STD_LOGIC_VECTOR(3 downto 0);
                   Cout : OUT STD_LOGIC);
      end Component;

      signal S1, H : STD_LOGIC_VECTOR(3 downto 0);
      signal C1, P : STD_LOGIC;

begin
      BPA1  : mod_BPA_4bit port map(Cin, A, B, S1, C1);
      Cout  <= C1 or (S1(3) and S1(2)) or (S1(3) and S1(1));
      H     <= "0110" when (Cout = '1') else "0000";
      BPA2  : mod_BPA_4bit port map('0', H, S1, Sum, P);
end Structural;
```

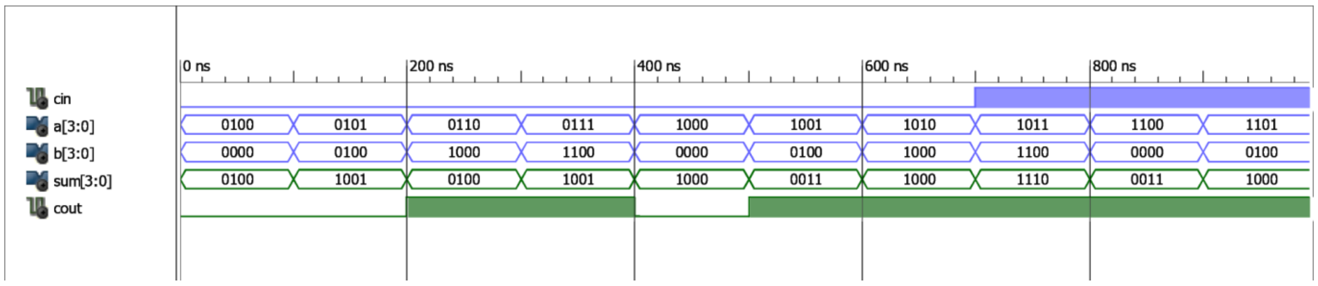
RTL Schematic:



VHDL Test Bench: *tb_BCDAdder.vhd*

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

ENTITY tb_BCDAdder IS
END tb_BCDAdder;

ARCHITECTURE behavior OF tb_BCDAdder IS
    COMPONENT mod_BCDAdder
          PORT(
               Cin      : IN  std_logic;
               A        : IN  std_logic_vector(3 downto 0);
               B        : IN  std_logic_vector(3 downto 0);
               Sum      : OUT  std_logic_vector(3 downto 0);
               Cout     : INOUT  std_logic
              );
    END COMPONENT;

   signal cin     : std_logic := '0';
   signal a       : std_logic_vector(3 downto 0) := (others => '0');
   signal b       : std_logic_vector(3 downto 0) := (others => '0');

   signal sum     : std_logic_vector(3 downto 0);
   signal cout    : std_logic;

BEGIN
   uut: mod_BCDAdder PORT MAP (
          Cin     => cin,
          A       => a,
          B       => b,
          Sum     => sum,
          Cout    => cout
        );

   cin <= '0', '1' after 700ns;
   proc_A: process
   begin
      a <= "0100";
      while a >= "0000" loop
            wait for 100ns;
            a <= std_logic_vector(unsigned(a) + 1);
      end loop;
      wait;
   end process;

   proc_B: process
   begin
      b <= "0000";
      while b >= "0000" loop
            wait for 100ns;
            b <= std_logic_vector(unsigned(b) + 4);
      end loop;
      wait;
   end process;
END;
```

Simulation:



```
a[3:0]    0100 0101 0110 0111 1000 1001 1010 1011 1100 1101
b[3:0]    0000 0100 1000 1100 0000 0100 1000 1100 0000 0100
sum[3:0]  0100 1001 0100 1001 1000 0011 1000 1110 0011 1000
```

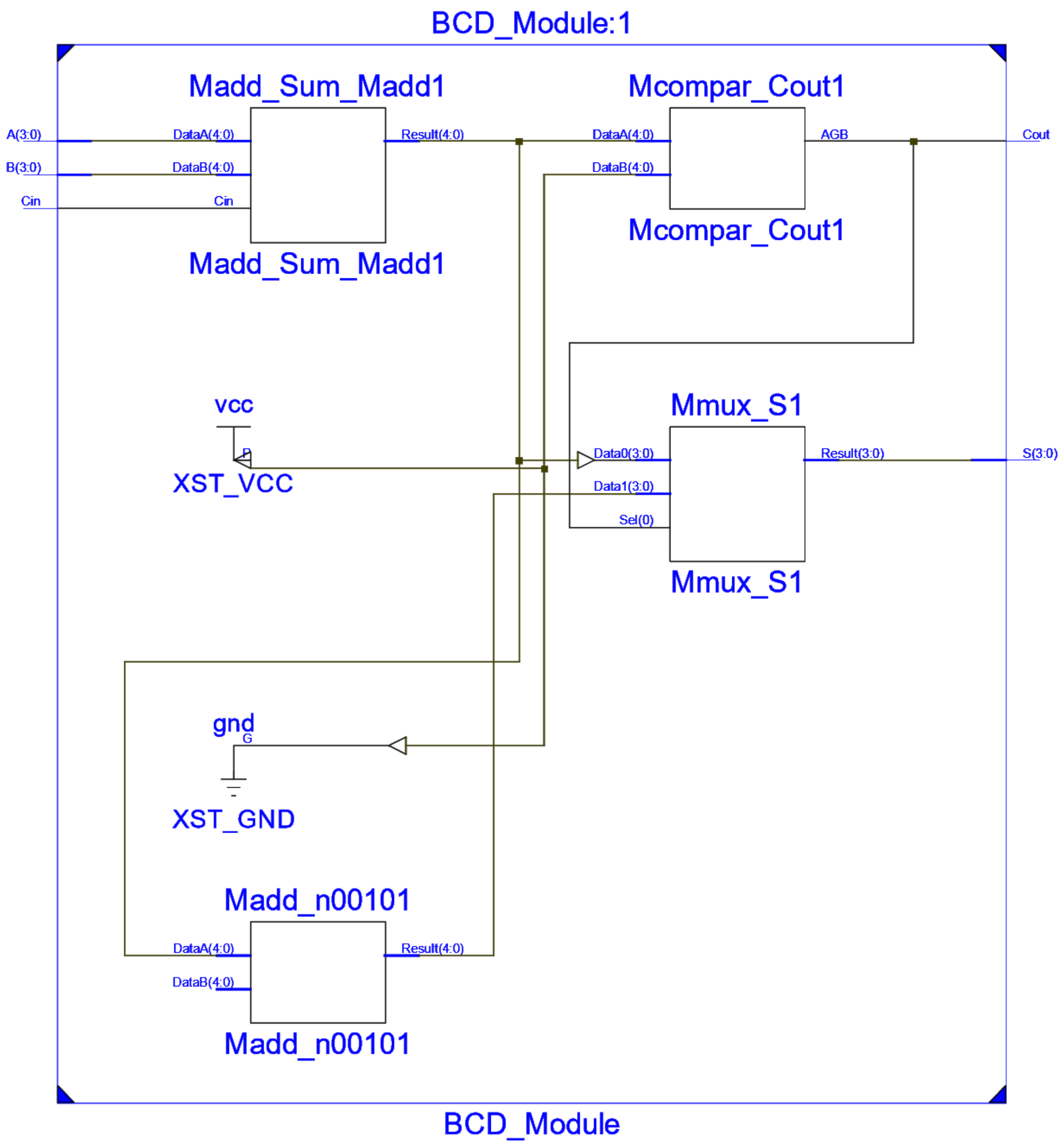
Question 7 – b

Design the BCD adder without using BPA (behavioral model to be used).

VHDL Module: *mod_BCDAdder.vhd*

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

entity mod_BCDAdder is
    Port ( Cin    : in  STD_LOGIC;
           A      : in  unsigned (3 downto 0);
           B      : in  unsigned (3 downto 0);
           Sum    : out  unsigned (3 downto 0);
           Cout   : out  STD_LOGIC);
end mod_BCDAdder;

architecture Behavioral of mod_BCDAdder is

begin
      process(Cin, A, B)
      variable S : unsigned (4 downto 0);
      begin
            S := ('0' & A) + ('0' & B) + ("0000" & Cin);
            if(S > 9) then
                  Cout <= '1';
                  Sum <= resize((S + "00110"), 4);
            else
                  Cout <= '0';
                  Sum <= S(3 downto 0);
            end if;
      end process;
end Behavioral;
```

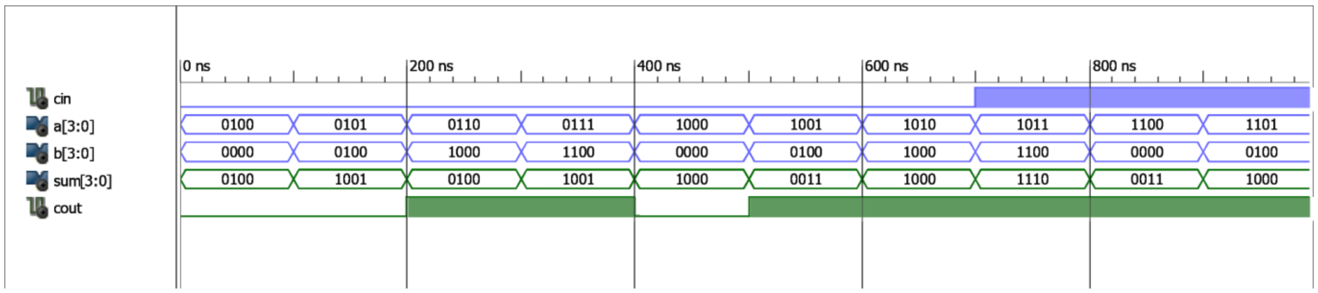
RTL Schematic:



VHDL Test Bench: *tb_BCDAdder.vhd*

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

ENTITY tb_BCDAdder IS
END tb_BCDAdder;

ARCHITECTURE behavior OF tb_BCDAdder IS
    COMPONENT mod_BCDAdder
          PORT(
               Cin      : IN  std_logic;
               A        : IN  unsigned(3 downto 0);
               B        : IN  unsigned(3 downto 0);
               Sum      : OUT  unsigned(3 downto 0);
               Cout     : OUT  std_logic
              );
    END COMPONENT;

   signal cin     : std_logic := '0';
   signal a       : unsigned(3 downto 0) := (others => '0');
   signal b       : unsigned(3 downto 0) := (others => '0');

   signal sum     : unsigned(3 downto 0);
   signal cout    : std_logic;

BEGIN
   uut: mod_BCDAdder PORT MAP (
          Cin     => cin,
          A       => a,
          B       => b,
          Sum     => sum,
          Cout    => cout
        );

   cin <= '0', '1' after 700ns;
   proc_A: process
   begin
      a <= "0100";
      while a >= "0000" loop
            wait for 100ns;
            a <= a + 1;
      end loop;
      wait;
    end process;

    proc_B: process
    begin
       b <= "0000";
       while b >= "0000" loop
             wait for 100ns;
             b <= b + 4;
       end loop;
       wait;
    end process;
 END;
```

Simulation:



Discussions:

In part a of the question, the BCD adder is constructed using two Binary Parallel Adders, which are in turn constituted by Full Adders in Parallel. However, in this part of the question, yet again two adders are implemented but the operations are split by bits. Each component for the purpose contains two full adders, a comparator and a multiplexer.

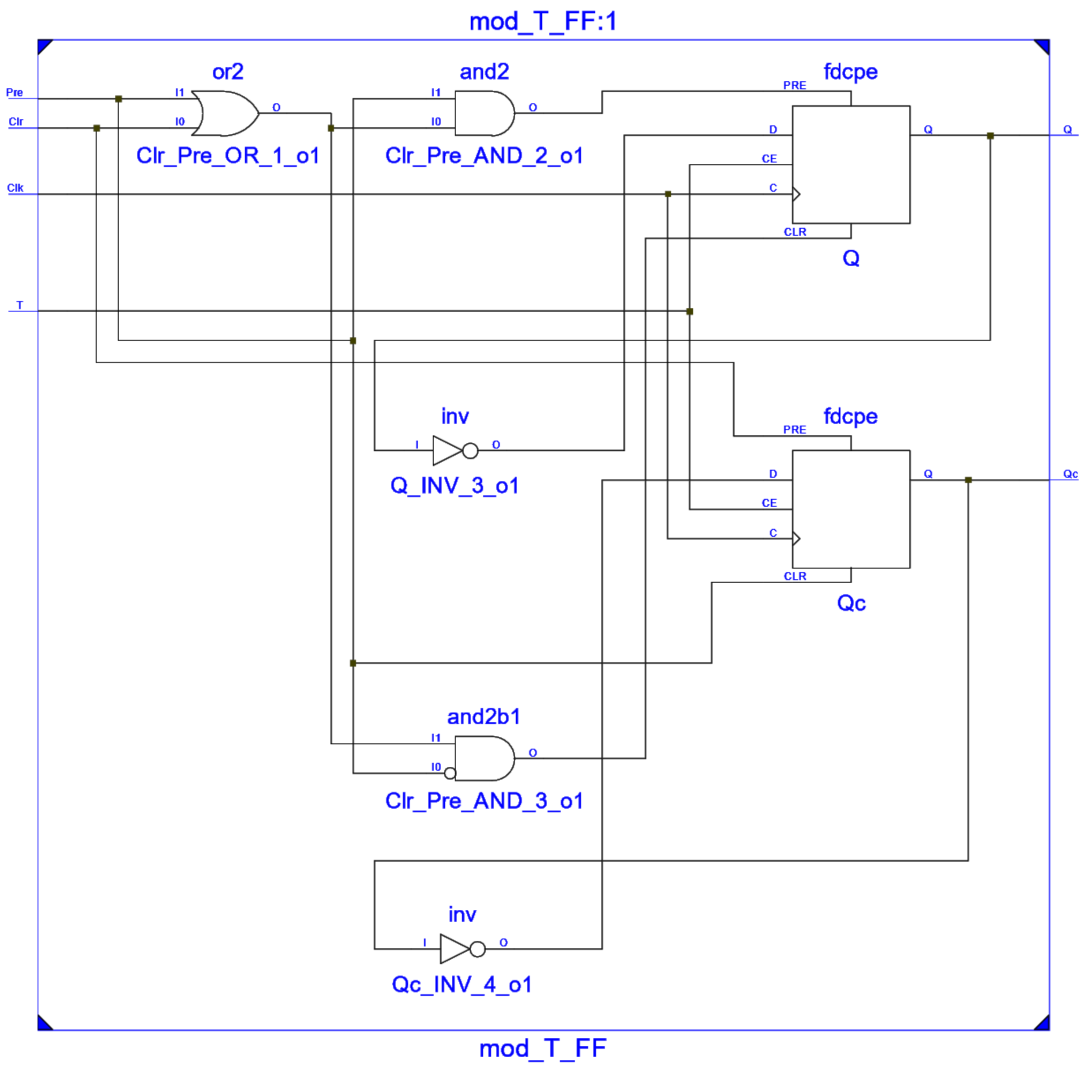
Question 8

Design and simulate the behavioral model of a T flip-flop.
Simulate for all possible input combinations. Note down the synthesis report.

VHDL Module: *mod_T_FF.vhd*

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

entity mod_T_FF is
      Port ( Pre, Clr   : in STD_LOGIC;
             Clk, T     : in STD_LOGIC;
             Q, Qc      : inout STD_LOGIC);
end mod_T_FF;

architecture Behavioral of mod_T_FF is

begin
      process(Pre, Clr, Clk, T)
      begin
            if(Pre = '1') then
                  Q <= '1'; Qc <= '0';
            elsif(Clr = '1') then
                  Q <= '0'; Qc <= '1';
            elsif(rising_edge(Clk)) then
                  if(T = '1') then
                        Q     <= not Q;
                        Qc    <= not Qc;
                  end if;
            end if;
      end process;
end Behavioral;
```

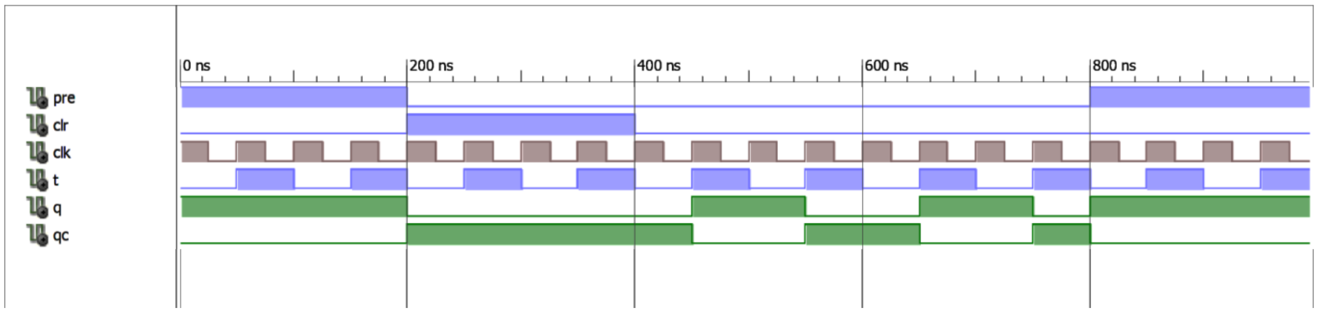
RTL Schematic:



VHDL Test Bench: *tb_T_FF.vhd*

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY tb_T_FF IS
END tb_T_FF;

ARCHITECTURE behavior OF tb_T_FF IS
    COMPONENT mod_T_FF
          PORT(
               Pre : IN  std_logic;
               Clr : IN  std_logic;
               Clk : IN  std_logic;
               T : IN  std_logic;
               Q : INOUT  std_logic;
               Qc : INOUT  std_logic
              );
    END COMPONENT;

signal pre  : std_logic := '0';
signal clr  : std_logic := '0';
signal clk  : std_logic := '0';
signal t          : std_logic := '0';

signal q          : std_logic;
signal qc   : std_logic;

constant clk_period : time := 50 ns;

BEGIN
   uut: mod_T_FF PORT MAP (
          Pre     => pre,
          Clr     => clr,
          Clk     => clk,
          T       => t,
          Q       => q,
          Qc      => qc
        );

   clk_process: process
   begin
      clk <= '1';
      wait for clk_period/2;
      clk <= '0';
      wait for clk_period/2;
   end process;

   set_process: process
   begin
      pre <= '1'; clr <= '0';
      wait for 200ns;
      pre <= '0'; clr <= '1';
      wait for 200ns;
      pre <= '0'; clr <= '0';
      wait for 400ns;
      pre <= '1'; clr <= '1';
   end process;

   stm_process: process
   begin
      while(t >= '0') loop
            t <= '0'; wait for 50ns;
            t <= '1'; wait for 50ns;
      end loop;
      wait;
   end process;
END;
```

Simulation:



Discussions:

The two outputs: Q and Q' come from two different repeating components, unlike the simplified circuit of a flip flop.

Synthesis Report:

```
=========================================================================
*                          HDL Parsing                                  *
=========================================================================
Parsing VHDL file "D:\IT Labs\IT451-COA\Assignment-1\Q08-T-Flip-FLop\mod_T_FF.vhd"
into library work
Parsing entity <mod_T_FF>.
Parsing architecture <Behavioral> of entity <mod_t_ff>.

=========================================================================
*                            HDL Elaboration                            *
=========================================================================

Elaborating entity <mod_T_FF> (architecture <Behavioral>) from library <work>.

=========================================================================
*                           HDL Synthesis                               *
=========================================================================

Synthesizing Unit <mod_T_FF>.
    Related source file is "D:\IT Labs\IT451-COA\Assignment-1\Q08-T-Flip-
FLop\mod_T_FF.vhd".
    Found 1-bit register for signal <Qc>.
    Found 1-bit register for signal <Q>.
    Summary:
      inferred   2 D-type flip-flop(s).
Unit <mod_T_FF> synthesized.

=========================================================================
HDL Synthesis Report

Macro Statistics
# Registers                                            : 2
 1-bit register                                        : 2

=========================================================================

=========================================================================
*                       Advanced HDL Synthesis                          *
=========================================================================



=========================================================================
Advanced HDL Synthesis Report
```

```
Macro Statistics
# Registers                                            : 2
 Flip-Flops                                            : 2

=========================================================================

=========================================================================
*                        Low Level Synthesis                            *
=========================================================================
List of register instances with asynchronous set and reset:
    Q in unit <mod_T_FF>
    Qc in unit <mod_T_FF>


Optimizing unit <mod_T_FF> ...

Mapping all equations...
Building and optimizing final netlist ...
Found area constraint ratio of 100 (+ 5) on block mod_T_FF, actual ratio is 0.

Final Macro Processing ...

=========================================================================
Final Register Report

Macro Statistics
# Registers                                            : 4
 Flip-Flops                                            : 4

=========================================================================

=========================================================================
*                          Partition Report                             *
=========================================================================

Partition Implementation Status
-------------------------------

  No Partitions were found in this design.

-------------------------------

=========================================================================
*                           Design Summary                              *
```

```
=========================================================================

Clock Information:
------------------
-----------------------------------+------------------------+-------+
Clock Signal                       | Clock buffer(FF name)  | Load  |
-----------------------------------+------------------------+-------+
Clr                                | IBUF+BUFG              | 1     |
Clk                                | BUFGP                  | 4     |
Pre                                | IBUF+BUFG              | 1     |
-----------------------------------+------------------------+-------+

Asynchronous Control Signals Information:
----------------------------------------
No asynchronous control signals found in this design

Timing Summary:
---------------
Speed Grade: -2

   Minimum period: 1.683ns (Maximum Frequency: 594.177MHz)
   Minimum input arrival time before clock: 3.596ns
   Maximum output required time after clock: 5.363ns
   Maximum combinational path delay: No path found

=========================================================================

Process "Synthesize - XST" completed successfully
```

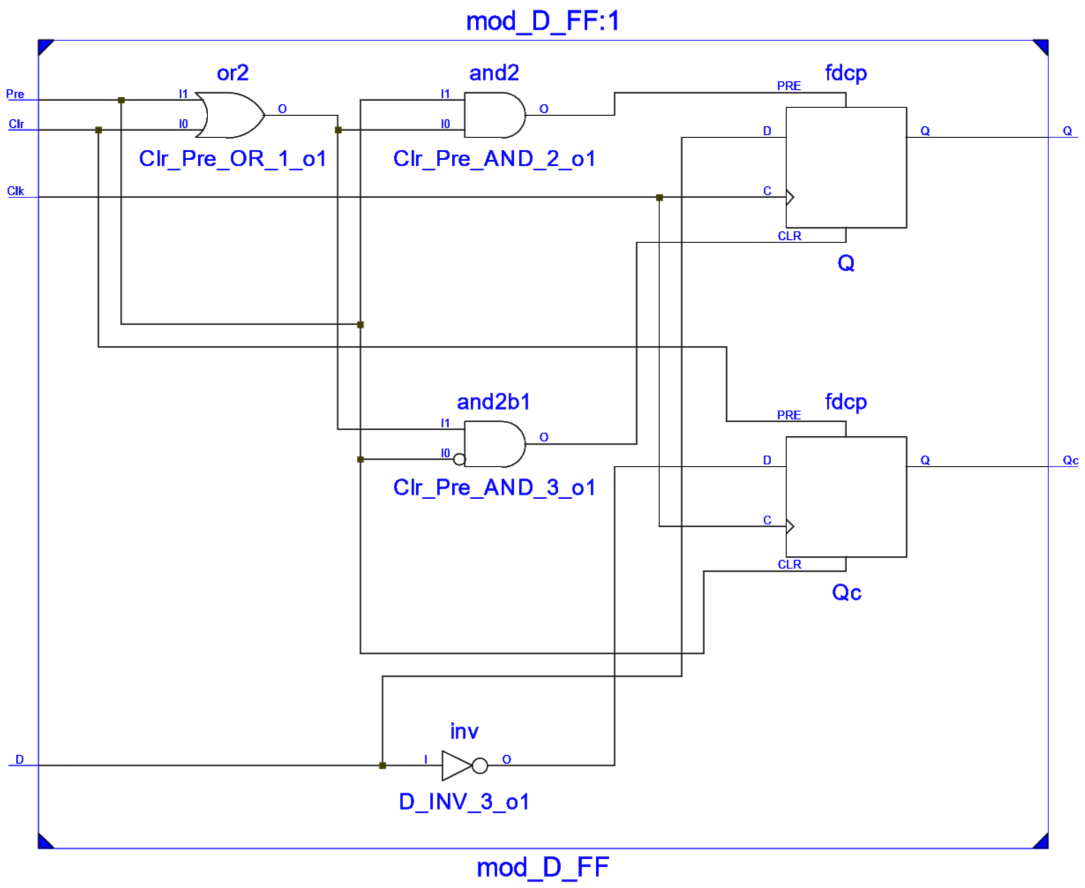
Question 9

Design and simulate the behavioral model of a D flip-flop.
Simulate for all possible input combinations. Note down the synthesis report.

VHDL Module: *mod_D_FF.vhd*

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

entity mod_D_FF is
      Port ( Pre, Clr   : in STD_LOGIC;
             Clk, D     : in STD_LOGIC;
             Q, Qc      : inout STD_LOGIC);
end mod_D_FF;

architecture Behavioral of mod_D_FF is

begin
      process(Pre, Clr, Clk, D)
      begin
            if(Pre = '1') then
                  Q <= '1'; Qc <= '0';
            elsif(Clr = '1') then
                  Q <= '0'; Qc <= '1';
            elsif(rising_edge(Clk)) then
                  if(D = '1') then
                        Q     <= '1';
                        Qc    <= '0';
                  else
                        Q     <= '0';
                        Qc    <= '1';
                  end if;
            end if;
      end process;
end Behavioral;
```

RTL Schematic:

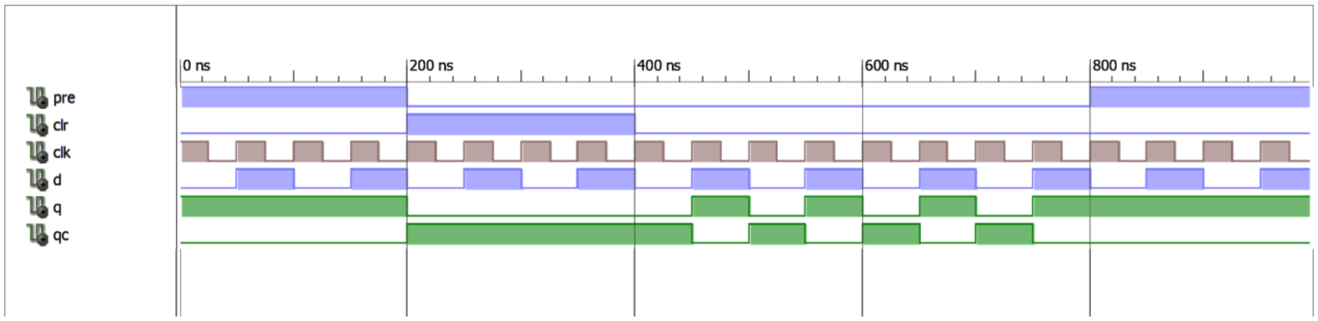


VHDL Test Bench: *tb_D_FF.vhd*

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY tb_D_FF IS
END tb_D_FF;

ARCHITECTURE behavior OF tb_D_FF IS
    COMPONENT mod_D_FF
          PORT(
               Pre : IN  std_logic;
               Clr : IN  std_logic;
               Clk : IN  std_logic;
               D : IN  std_logic;
               Q : INOUT  std_logic;
               Qc : INOUT  std_logic
              );
    END COMPONENT;

signal pre  : std_logic := '0';
signal clr  : std_logic := '0';
signal clk  : std_logic := '0';
signal d    : std_logic := '0';

signal q    : std_logic;
signal qc   : std_logic;

constant clk_period : time := 50 ns;

BEGIN
   uut: mod_D_FF PORT MAP (
          Pre     => pre,
          Clr     => clr,
          Clk     => clk,
          D       => d,
          Q       => q,
          Qc      => qc
        );

   clk_process: process
   begin
      clk <= '1';
      wait for clk_period/2;
      clk <= '0';
```

```
         wait for clk_period/2;
      end process;

      set_process: process
      begin
         pre <= '1'; clr <= '0';
         wait for 200ns;
         pre <= '0'; clr <= '1';
         wait for 200ns;
         pre <= '0'; clr <= '0';
         wait for 400ns;
         pre <= '1'; clr <= '1';
      end process;

      stm_process: process
      begin
         while(d >= '0') loop
               d <= '0'; wait for 50ns;
               d <= '1'; wait for 50ns;
         end loop;
         wait;
      end process;
   END;
```

Simulation:



Discussions:

The two outputs: Q and Q’ come from two different repeating components, unlike the simplified circuit of a flip flop.

Synthesis Report:

```
=========================================================================
*                          HDL Parsing                                  *
=========================================================================
Parsing VHDL file "D:\IT Labs\IT451-COA\Assignment-1\Q09-D-Flip-Flop\mod_D_FF.vhd"
into library work
Parsing entity <mod_D_FF>.
Parsing architecture <Behavioral> of entity <mod_d_ff>.

=========================================================================
*                            HDL Elaboration                            *
=========================================================================

Elaborating entity <mod_D_FF> (architecture <Behavioral>) from library <work>.

=========================================================================
*                           HDL Synthesis                               *
=========================================================================

Synthesizing Unit <mod_D_FF>.
    Related source file is "D:\IT Labs\IT451-COA\Assignment-1\Q09-D-Flip-
Flop\mod_D_FF.vhd".
    Found 1-bit register for signal <Qc>.
    Found 1-bit register for signal <Q>.
    Summary:
      inferred   2 D-type flip-flop(s).
Unit <mod_D_FF> synthesized.

=========================================================================
HDL Synthesis Report

Macro Statistics
# Registers                                            : 2
 1-bit register                                        : 2

=========================================================================

=========================================================================
*                       Advanced HDL Synthesis                          *
=========================================================================


=========================================================================
Advanced HDL Synthesis Report
```

```
Macro Statistics
# Registers                                            : 2
 Flip-Flops                                            : 2

=========================================================================

=========================================================================
*                        Low Level Synthesis                            *
=========================================================================
List of register instances with asynchronous set and reset:
    Q in unit <mod_D_FF>
    Qc in unit <mod_D_FF>


Optimizing unit <mod_D_FF> ...

Mapping all equations...
Building and optimizing final netlist ...
Found area constraint ratio of 100 (+ 5) on block mod_D_FF, actual ratio is 0.

Final Macro Processing ...

=========================================================================
Final Register Report

Macro Statistics
# Registers                                            : 4
 Flip-Flops                                            : 4

=========================================================================

=========================================================================
*                          Partition Report                             *
=========================================================================

Partition Implementation Status
-------------------------------

  No Partitions were found in this design.

-------------------------------

=========================================================================
*                           Design Summary                              *
```

```
=========================================================================

Clock Information:
------------------
-----------------------------------+------------------------+-------+
Clock Signal                       | Clock buffer(FF name)  | Load  |
-----------------------------------+------------------------+-------+
Clr                                | IBUF+BUFG              | 1     |
Clk                                | BUFGP                  | 4     |
Pre                                | IBUF+BUFG              | 1     |
-----------------------------------+------------------------+-------+

Asynchronous Control Signals Information:
----------------------------------------
No asynchronous control signals found in this design

Timing Summary:
---------------
Speed Grade: -2

   Minimum period: No path found
   Minimum input arrival time before clock: 3.596ns
   Maximum output required time after clock: 5.319ns
   Maximum combinational path delay: No path found

=========================================================================

Process "Synthesize - XST" completed successfully
```

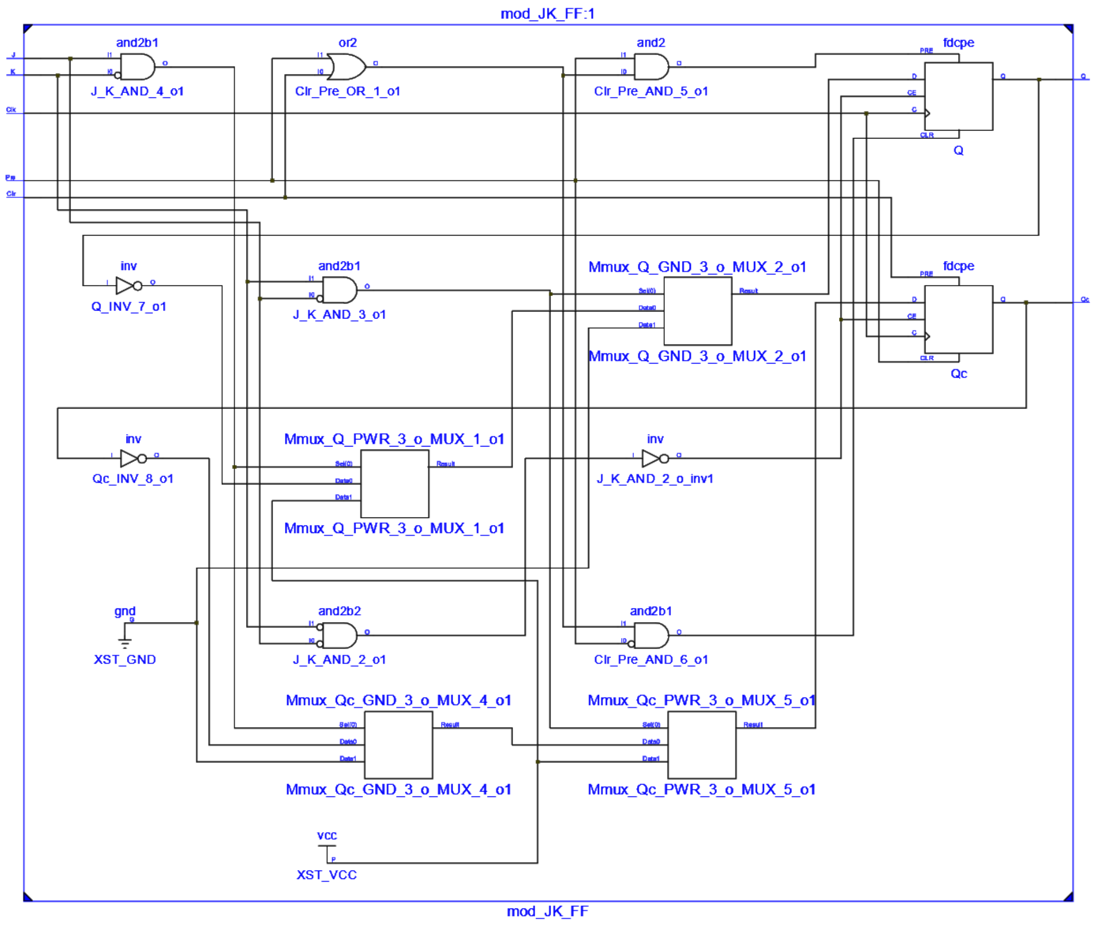
Question 10

Design and simulate the behavioral model of a JK flip-flop.
Simulate for all possible input combinations. Note down the synthesis report.

VHDL Module: *mod_JK_FF.vhd*

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

entity mod_JK_FF is
      Port ( Pre, Clr   : in STD_LOGIC;
             Clk, J, K  : in STD_LOGIC;
             Q, Qc      : inout STD_LOGIC);
end mod_JK_FF;

architecture Behavioral of mod_JK_FF is

begin
      process(Pre, Clr, Clk, J, K)
      begin
            if(Pre = '1') then
                  Q <= '1'; Qc <= '0';
            elsif(Clr = '1') then
                  Q <= '0'; Qc <= '1';
            elsif(rising_edge(Clk)) then
                  if(J = '0' and K = '0') then
                        Q     <= Q;
                        Qc    <= Qc;
                  elsif(J = '0' and K = '1') then
                        Q     <= '0';
                        Qc    <= '1';
                  elsif(J = '1' and K = '0') then
                        Q     <= '1';
                        Qc    <= '0';
                  else
                        Q     <= not Q;
                        Qc    <= not Qc;
                  end if;
            end if;
      end process;
end Behavioral;
```

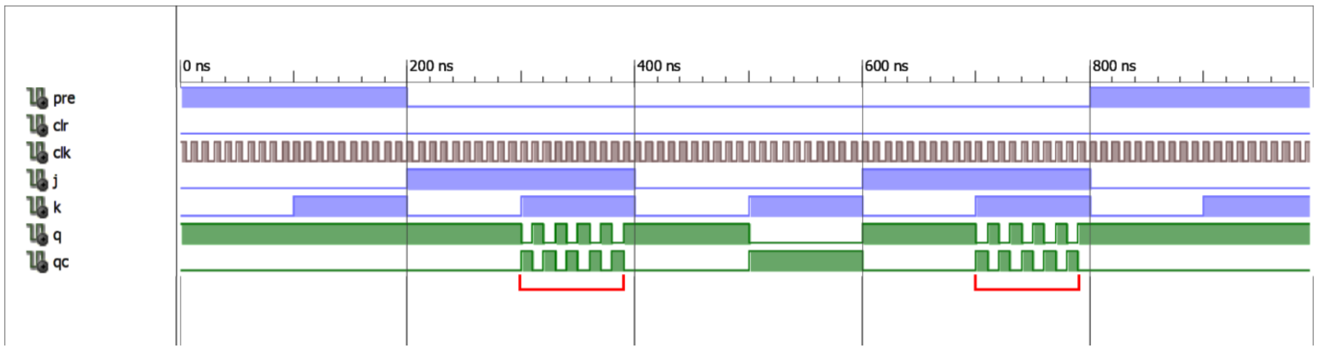
RTL Schematic:



VHDL Test Bench: *tb_JK_FF.vhd*

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY tb_JK_FF IS
END tb_JK_FF;

ARCHITECTURE behavior OF tb_JK_FF IS
    COMPONENT mod_JK_FF
          PORT(
               Pre : IN  std_logic;
               Clr : IN  std_logic;
               Clk : IN  std_logic;
               J : IN  std_logic;
               K : IN  std_logic;
               Q  : INOUT  std_logic;
               Qc : INOUT  std_logic
              );
    END COMPONENT;

   signal pre : std_logic := '0';
   signal clr : std_logic := '0';
   signal clk : std_logic := '0';
   signal j : std_logic := '0';
   signal k : std_logic := '0';

   signal q  : std_logic;
   signal qc : std_logic;

   constant clk_period : time := 10 ns;

BEGIN
   uut: mod_JK_FF PORT MAP (
          Pre     => pre,
          Clr     => clr,
          Clk     => clk,
          J       => j,
          K       => k,
          Q       => q,
          Qc      => qc
        );
   clk_process: process
   begin
      Clk <= '1';
      wait for Clk_period/2;
      Clk <= '0';
      wait for Clk_period/2;
   end process;

   set_process: process
   begin
      pre <= '1'; clr <= '0';
      wait for 200ns;
      pre <= '0'; clr <= '0';
      wait for 600ns;
      pre <= '0'; clr <= '1';
   end process;

   stm_process: process
   begin
      while(clk >= '0') loop
            j <= '0'; k <= '0'; wait for 100ns;
            j <= '0'; k <= '1'; wait for 100ns;
            j <= '1'; k <= '0'; wait for 100ns;
            j <= '1'; k <= '1'; wait for 100ns;
      end loop;
      wait;
   end process;
END;
```

Simulation:



Discussions:

Racing Condition observed when both J & K are ‘1’. The output continuously toggles at every clock edge triggering.

Synthesis Report:

```
=========================================================================
*                          HDL Parsing                                  *
=========================================================================
Parsing VHDL file "D:\IT Labs\IT451-COA\Assignment-1\Q10-JK-Flip-
Flop\mod_JK_FF.vhd" into library work
Parsing entity <mod_JK_FF>.
Parsing architecture <Behavioral> of entity <mod_jk_ff>.

=========================================================================
*                            HDL Elaboration                            *
=========================================================================

Elaborating entity <mod_JK_FF> (architecture <Behavioral>) from library <work>.

=========================================================================
*                           HDL Synthesis                               *
=========================================================================

Synthesizing Unit <mod_JK_FF>.
    Related source file is "D:\IT Labs\IT451-COA\Assignment-1\Q10-JK-Flip-
Flop\mod_JK_FF.vhd".
    Found 1-bit register for signal <Qc>.
    Found 1-bit register for signal <Q>.
    Summary:
      inferred   2 D-type flip-flop(s).
      inferred   4 Multiplexer(s).
Unit <mod_JK_FF> synthesized.

=========================================================================
HDL Synthesis Report

Macro Statistics
# Registers                                            : 2
 1-bit register                                        : 2
# Multiplexers                                         : 4
 1-bit 2-to-1 multiplexer                              : 4

=========================================================================

=========================================================================
*                       Advanced HDL Synthesis                          *
=========================================================================
=========================================================================
Advanced HDL Synthesis Report

Macro Statistics
# Registers                                            : 2
 Flip-Flops                                            : 2
# Multiplexers                                         : 4
 1-bit 2-to-1 multiplexer                              : 4

=========================================================================

=========================================================================
*                         Low Level Synthesis                           *
=========================================================================
List of register instances with asynchronous set and reset:
    Q in unit <mod_JK_FF>
    Qc in unit <mod_JK_FF>


Optimizing unit <mod_JK_FF> ...

Mapping all equations...
Building and optimizing final netlist ...
Found area constraint ratio of 100 (+ 5) on block mod_JK_FF, actual ratio is 0.

Final Macro Processing ...

=========================================================================
Final Register Report

Macro Statistics
# Registers                                            : 4
 Flip-Flops                                            : 4

=========================================================================

=========================================================================
*                           Partition Report                            *
=========================================================================

Partition Implementation Status
-------------------------------

  No Partitions were found in this design.
```

```
-------------------------------

=========================================================================
*                            Design Summary                             *
=========================================================================

Clock Information:
------------------
-----------------------------------+------------------------+-------+
Clock Signal                       | Clock buffer(FF name)  | Load  |
-----------------------------------+------------------------+-------+
Clr                                | IBUF+BUFG              | 1     |
Clk                                | BUFGP                  | 4     |
Pre                                | IBUF+BUFG              | 1     |
-----------------------------------+------------------------+-------+

Asynchronous Control Signals Information:
----------------------------------------
No asynchronous control signals found in this design

Timing Summary:
---------------
Speed Grade: -2

   Minimum period: 1.683ns (Maximum Frequency: 594.177MHz)
   Minimum input arrival time before clock: 3.596ns
   Maximum output required time after clock: 5.363ns
   Maximum combinational path delay: No path found

=========================================================================

Process "Synthesize - XST" completed successfully
```

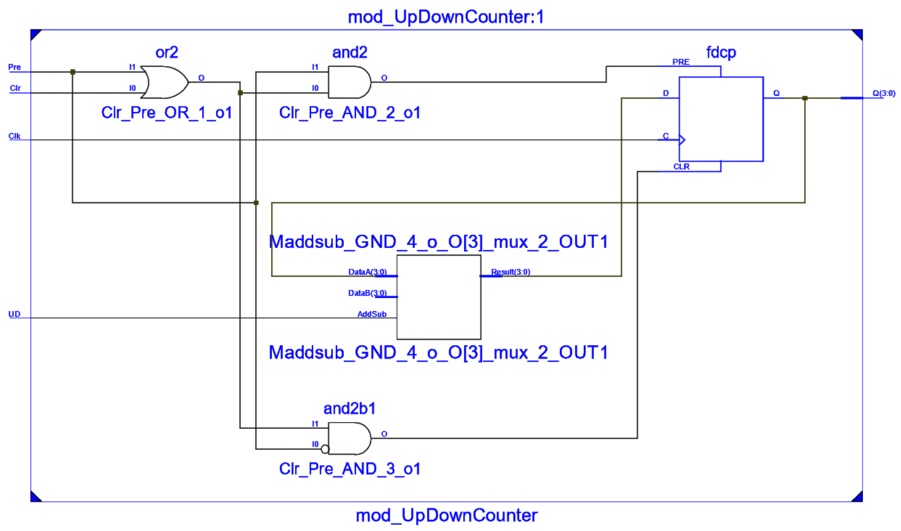
Question 11 – a

Design and simulate the behavioral model of a 4-bit UP/DOWN counter.

VHDL Module: *mod_UpDownCounter.vhd*

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

entity mod_UpDownCounter is
    Port ( Pre, Clr, Clk, UD  : in  STD_LOGIC;
           Q : out  STD_LOGIC_VECTOR (3 downto 0));
end mod_UpDownCounter;

architecture Behavioral of mod_UpDownCounter is
      signal O : STD_LOGIC_VECTOR (3 downto 0);

begin
      process(Pre, Clr, Clk, UD)
      begin
            if          (Pre = '1') then O <= "1111";
            elsif (Clr = '1') then O <= "0000";
            elsif (rising_edge(Clk)) then
                  if(UD = '1') then O <= std_logic_vector(unsigned(O) + 1);
                  else              O <= std_logic_vector(unsigned(O) - 1);
                  end if;
            end if;
      end process;
      Q <= O;
end Behavioral;
```

RTL Schematic:

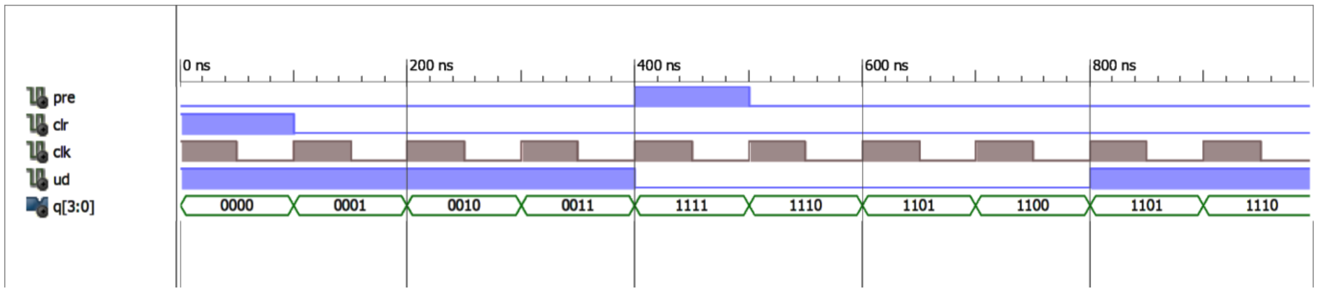


VHDL Test Bench: *tb_UpDownCounter.vhd*

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY tb_UpDownCounter IS
END tb_UpDownCounter;

ARCHITECTURE behavior OF tb_UpDownCounter IS
    COMPONENT mod_UpDownCounter
          PORT(
                Pre : IN  std_logic;
                Clr : IN  std_logic;
                Clk : IN  std_logic;
                UD : IN  std_logic;
                Q  : OUT  std_logic_vector(3 downto 0)
               );
    END COMPONENT;


   signal pre : std_logic := '0';
   signal clr : std_logic := '0';
   signal clk : std_logic := '0';
   signal ud  : std_logic := '0';

   signal q : std_logic_vector(3 downto 0);

   constant clk_period : time := 100 ns;

BEGIN
   uut: mod_UpDownCounter PORT MAP (
          Pre     => pre,
          Clr     => clr,
          Clk     => clk,
          UD      => ud,
          Q       => q
        );

   clk_process: process
   begin
      clk <= '1';
      wait for clk_period/2;
      clk <= '0';
      wait for clk_period/2;
   end process;
```

```
    set_process: process
    begin
       pre <= '0'; clr <= '1';
       wait for 100ns;
       pre <= '0'; clr <= '0';
       wait for 300ns;
       pre <= '1'; clr <= '0';
       wait for 100ns;
       pre <= '0'; clr <= '0';
       wait;
    end process;

    stm_process: process
    begin
       ud <= '1', '0' after 400ns, '1' after 800ns;
       wait;
    end process;
 END;
```

Simulation:



Discussions:

D Flip-Flops are used as registers for the counter. A Multiplexer operation is used to choose the direction of the counter. Counter counts up for UD = ‘1’ and down for UD = ‘0’.

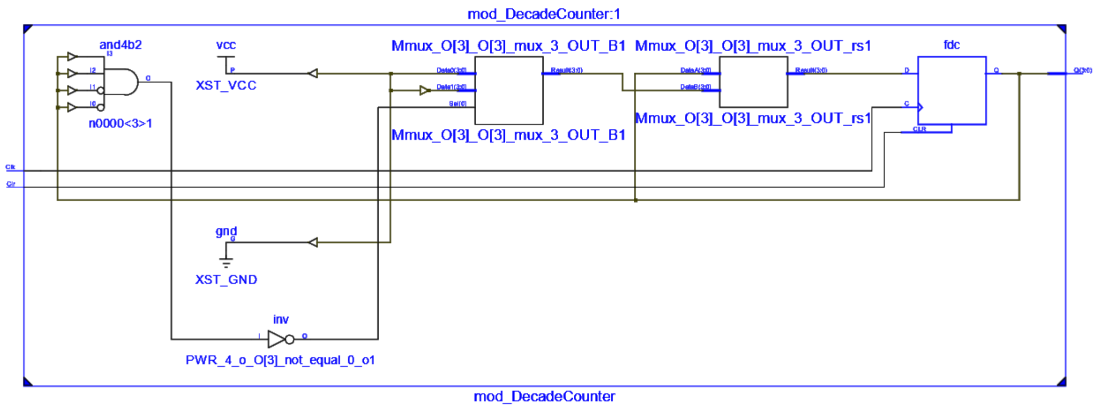
Question 11 – b

Modify the above counter program to make it a DECADE counter.

VHDL Module: *mod_DecadeCounter.vhd*

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

entity mod_DecadeCounter is
    Port ( Clr, Clk : in  STD_LOGIC;
           Q : out  STD_LOGIC_VECTOR (3 downto 0));
end mod_DecadeCounter;

architecture Behavioral of mod_DecadeCounter is
      signal O : STD_LOGIC_VECTOR (3 downto 0);

begin
      process(Clr, Clk)
      begin
            if    (Clr = '1')        then O <= "0000";
            elsif (rising_edge(Clk)) then
                  if    (O /= "1001") then O <= std_logic_vector(unsigned(O) + 1);
                  else                     O <= std_logic_vector(unsigned(O) + 7);
                  end if;
            end if;
      end process;
      Q <= O;
end Behavioral;
```

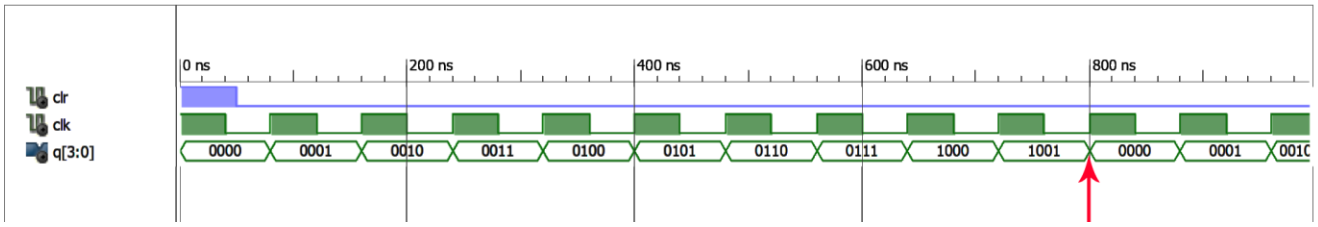
RTL Schematic:


mod_DecadeCounter:1
and4b2
n0000<3>1
vcc
XST_VCC
gnd
XST_GND
inv
PWR_4_o_O[3]_not_equal_0_o1
Mmux_O[3]_O[3]_mux_3_OUT_B1
Mmux_O[3]_O[3]_mux_3_OUT_B1
Mmux_O[3]_O[3]_mux_3_OUT_rs1
Mmux_O[3]_O[3]_mux_3_OUT_rs1
fdc
Clk
Clr
Q(3:0)
mod_DecadeCounter

VHDL Test Bench: *tb_DecadeCounter.vhd*

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY tb_DecadeCounter IS
END tb_DecadeCounter;

ARCHITECTURE behavior OF tb_DecadeCounter IS
    COMPONENT mod_DecadeCounter
          PORT(
               Clr : IN  std_logic;
               Clk : IN  std_logic;
               Q : OUT  std_logic_vector(3 downto 0)
              );
    END COMPONENT;

   signal clr : std_logic := '0';
   signal clk : std_logic := '0';

   signal q : std_logic_vector(3 downto 0);

   constant clk_period : time := 80 ns;

BEGIN
   uut: mod_DecadeCounter PORT MAP (
          Clr     => clr,
          Clk     => clk,
          Q       => q
        );

   clk_process: process
   begin
      clk <= '1';
      wait for clk_period/2;
      clk <= '0';
      wait for clk_period/2;
   end process;

   set_process: process
   begin
      clr <= '1'; wait for 50ns;
      clr <= '0'; wait;
   end process;
END;
```

Simulation:



Discussions:

Counter starts from 0 (“0000”) till 9 (“1001”) after which 6 (“0110”) is added to the value which becomes 16 (“10000”). Since, the Counter is of 4-bits, the leading ‘1’ is dropped and the value becomes 0 (“0000”), i.e. the initial value.