Digital Logic Design - Chapter 8-VHDL

(In addition to the code include design documentation either as comments in the code or separate document.)

1S. Write a VHDL behavioral program to implement a pulse-triggered D latch.

Solution:

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

entity Vdlatch is
    port (D, CLK: in std_logic;
          Q, QN: buffer std_logic);   -- note buffer is a new mode type.
end Vdlatch;

architecture Vdlatch_b of Vdlatch is
begin
process (CLK, D)
    begin
          if (CLK = '1') then
                Q<=D;
          end if;
          QN <= not Q;
    end process;
end Vdlatch_b;
```

1U. Write entity code for a (2-input) XNOR.

Solution:

2S. Complete a VHDL behavioral design for an edge-triggered D latch .

Note: The expression (**signal'event)** is only true when the signal is changing. This is referred to as the event attribute of the signal. This attribute in conjunction with process() should be considered in completing this design. <Why is "**signal'event**" even mentioned; the way that this is written suggests that **signal'event** was mentioned before.>

Solution:

"CLK'event is no required since the process will be entered only when CLK changes."

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

entity Vdff is
    port (D, CLK: in std_logic;
          Q : out std_logic);
end Vdff;

architecture Vdff_b of Vdff is
begin
process(CLK)
    begin
          if (CLK'event and CLK='1') then
                Q <= D;
          end if;
    end process;
end Vdff_b;
```

2U. Write a VHDL behavioral program to implement a rising-edge-triggered JK flip flop.

Solution:

3S. Design a VHDL model for a 16-bit register with Clock Enable, active-low Output Enable, and active-low Clear.

Solution:

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

entity Vreg16 is
    port (CLK, CLKEN, OE_L, CLR_L: in STD_LOGIC;
          D: in STD_LOGIC_VECTOR (1 to 16);          -- Input bus
          Q: out STD_ULOGIC_VECTOR (1 to 16) );      -- output bus (three-
                                                     --   state, unresolved)
end Vreg16;

architecture Vreg16 of Vreg16 is
signal CLR, OE: STD_LOGIC;                           -- active-high version of
                                                     --   signals
signal IQ: STD_LOGIC_VECTOR(1 to 16);                -- internal Q signal
begin
process (CLK, CLR_L, CLR, OE_L, OE, IQ)
    begin
          CLR <= not CLR_L;
          OE <= not OE_L;
          if (CLR = '1') then
               IQ <= (others =>'0');                 <Explain this line.>
          elsif (CLK'event and CLK='1') then
               if (CLKEN = '1') then
                    IQ <= D;
               end if;
          end if;
          if OE = '1' then
               Q <= To_StdULogicVector(IQ);          <Explain this line.>
          else Q <= (others => 'Z');                 <Explain this line.>
          end if;
    end process;
end Vreg16;
```

3U. Design a VHDL model for a 12-bit register with Clock Enable, active-high Output Enable, and active-high Clear.

Solution:

4S. Using VHDL, design a 4-to-1 MUX. The two bit input "sel", selects the input d_0-d_3 that is connected to output "f". For example when sel="01", f=d_1.

Solution:

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

entity MUX4 is
```

```
    port ( sel : in std_logic_vector(1 downto 0);
           d  : in std_logic_vector(3 downto 0);
           f : out std_logic);
end MUX4;

architecture dataflow of MUX4 is

begin

f <= d(0) when sel = "00" else
     d(1) when sel = "01" else
        d(2) when sel = "10" else
        d(3);

end dataflow;
```

4U. Using VHDL, design a 8-to-1 MUX. The three bit input "sel", selects the input d_0-d_8 that is connected to output "f". For example when sel="010", f=d_2.

Solution:

5S. Design a 2–Bit Ripple Carry Adder with carry in and a carry out using VHDL.

Solution:

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

entity adder2_b is
     port ( ci : in std_logic;                          --carry in signal
          a, b : in std_logic_vector (1 downto 0);      --2 2-bit numbers, a and b
          s : out std_logic_vector (1 downto 0);        --2-bit sum
          co: out std_logic);                           --carry out term or bit (overflow)

end adder2_b;

architecture dataflow of adder2_b is
signal sum: std_logic_vector (2 downto 0);          --Sets the signal 'sum' to a 3-bit number,
                                                    --   to account for an overflow bit (carry out)
begin
     sum <= ('0' & a) + b + ci;                     --('0' & a) forces a to become a 3-bit number
     s <= sum (1 downto 0) ;                        --   in order to account for a carry-in bit
     co <= sum (2);                                 --sum(0) and sum (1)  bits are the sum
                                                    --   of a and b
end dataflow;                                       --  sum(2) is the carry-out bit (overflow)
```

5U. Design a 8–Bit Ripple Carry Adder with carry in and a carry out using VHDL.

Solution:

6S. Design the architectures for a rising edge D flip-flop for each of the following cases:

a) with asynchronous reset using an **if** statement.
b) with synchronous reset using an **if** statement.
c) with synchronous reset using a **wait until** statement.

Solution

All of the architectures have the following entity declaration in common:

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

entity dff is

        port (CLK,D,RESET: in std_logic;
              Q: out std_logic);

end dff;
```

a) The asynchronous reset architecture using an **if** statement:

```
architecture asyn_reset of dff is

begin
        process (CLK, RESET, D)
        begin
                if RESET = '1' then
                        Q <= '0';
                elsif rising_edge (CLK) then
                        Q <= D;
                end if;
        end process;

end asyn_reset;
```

b) The synchronous reset architecture using an **if** statement:

```
architecture syn_reset of dff is
begin
        process (CLK, RESET, D)
        begin
            if rising_edge (CLK) then
                if RESET = '1' then
                    Q <= '0';
                else
                    Q <= D;
                end if;
            end if;
        end process;
end syn_reset;
```

c) The synchronous reset architecture using a **wait until** statement:

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

entity dff is
        port (CLK,D,RESET: in std_logic;
            Q: out std_logic);
end dff;

architecture syn_reset of dff is
begin
        process                                 -- no sensitivity list.
```

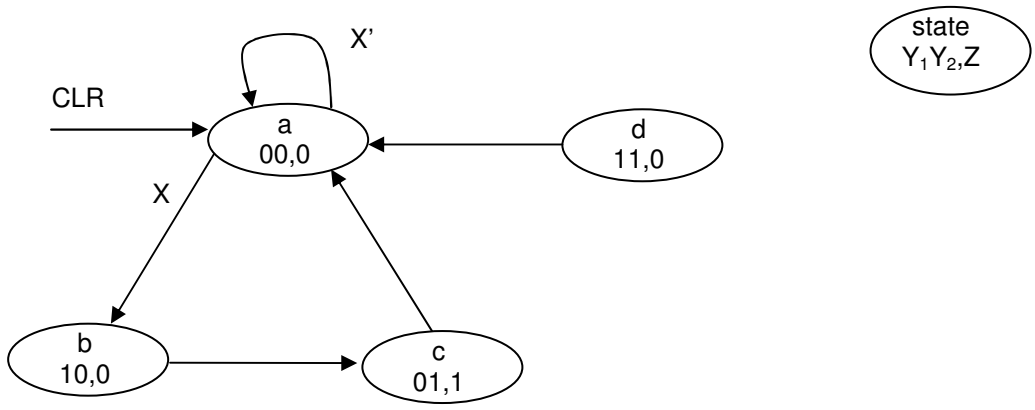
```
        begin
                wait until rising_edge (CLK);
                        if RESET = '1' then
                                Q <= '0';
                        else
                                Q <= D;
                        end if;
        end process;
end syn_reset;
```

6U. Design the architectures for a rising edge JK flip-flop for each of the following cases:

a) with asynchronous reset using an **if** statement.
b) with synchronous reset using an **if** statement.
c) with synchronous reset using a **wait until** statement.

Solution

7S. Create a complete VHDL design for the system described by the following state diagram:.



Solution:

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

entity fsm_1_d is
        port (CLK, CLR, X:  in std_logic;
              Y: out std_logic_vector (1 to 2);
              Z: out std_logic);

end fsm_1_d;

architecture design of fsm_1_d is

        type state_type is (a, b, c, d);
        signal PS, NS: state_type;

begin
sync_proc: process (CLK, CLR, NS)
    begin
        if (CLR = '1') then
                PS <= a;
                Z  <= '0';
        elsif rising_edge (CLK) then
```

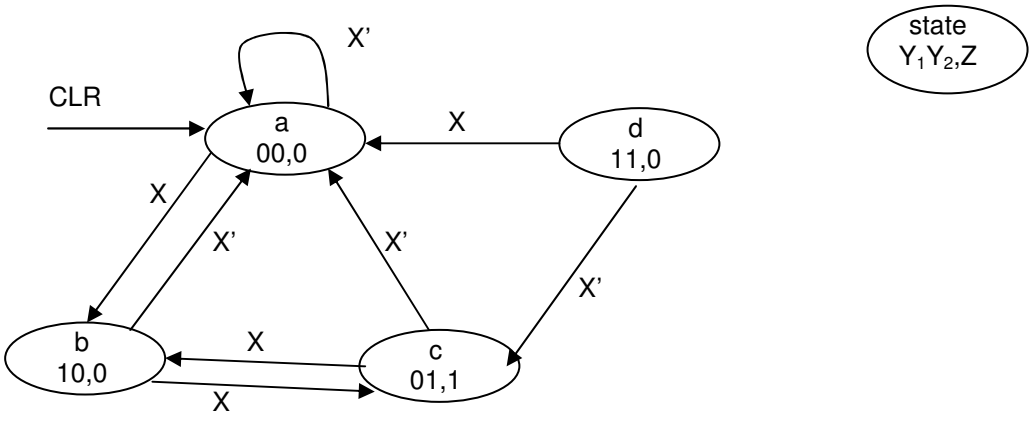
```
                PS <= NS;
            end if;
        end process sync_proc;

    comb_proc: process (PS, X)
        begin
            case PS is
                when a => Z <= '0';
                    if X = '1' then
                        NS <= b;
                    else
                        NS <= a;
                    end if;
                when b => Z <= '0';
                    NS <= c;
                when c => Z <= '1';
                    NS <= a;
                when others => Z <= '0';
                    NS <= a;
              end case;
        end process comb_proc;

    with PS select
        Y <= "00" when a,
            "10" when b,
            "01" when c,
            "11" when d,
            "00" when others;

    end design;
```

7U. Create a complete VHDL design for the system described by the following state diagram:.



Solution:

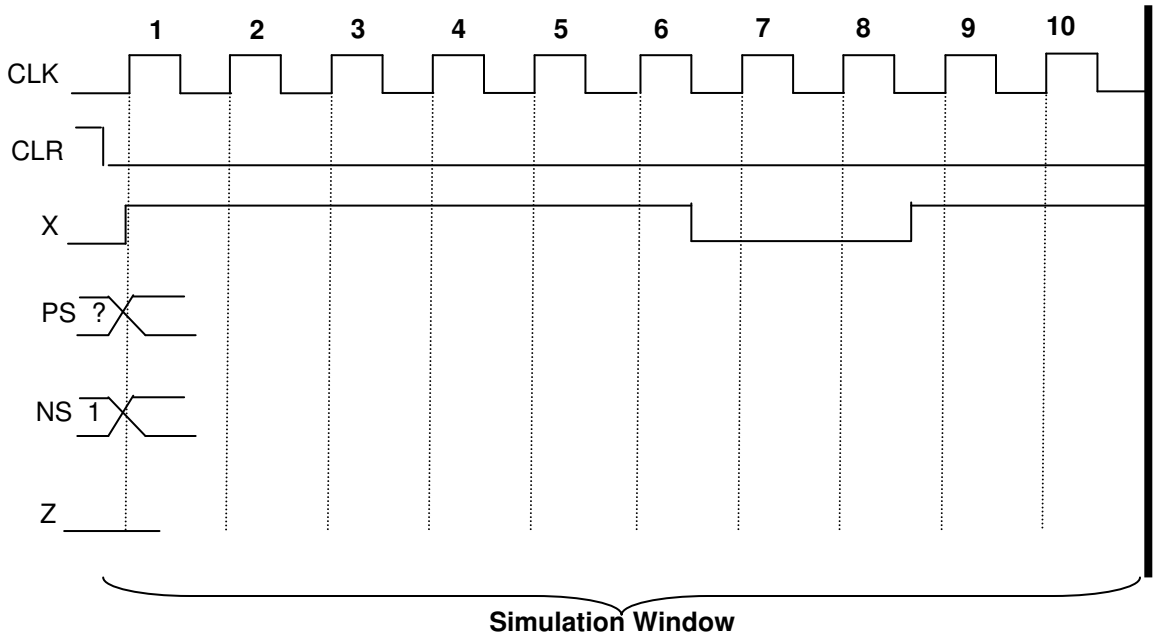
8S. Complete the timing diagram in simulation window drawn at the bottom of the page based on the following the VHDL Code segment:

```
    library IEEE;
    use IEEE.STD_LOGIC_1164.all;
    entity quiz_ent is
```

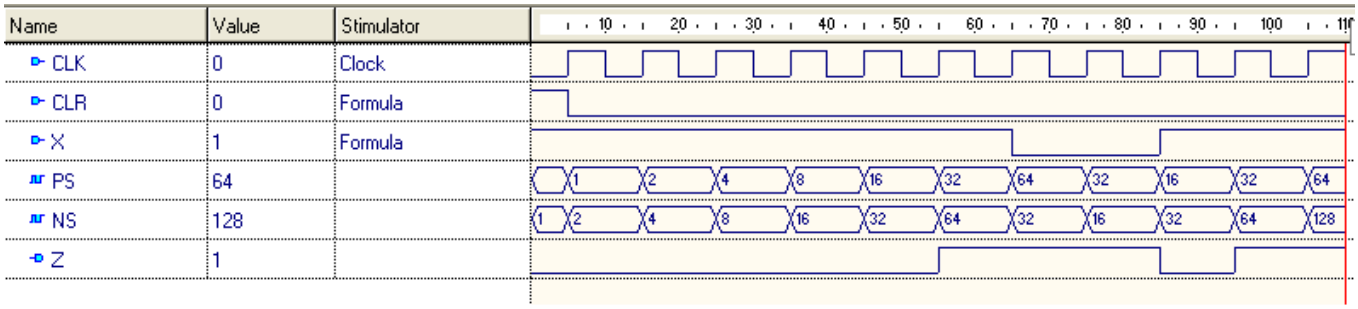
```
        port (CLK, CLR, X: in std_logic;
                        Z: out std_logic
                );
end quiz_ent;
architecture quiz_arch of quiz_ent is
        Signal PS, NS: integer;
begin
        proc_a: process (CLK, CLR)
        begin
                if (CLR = '1') then          NS <= 1;
                elsif rising_edge (CLK) then
                        PS <= NS;
                        If ( NS < 2**8 and x='1') then NS <= NS*2;
                        elsif ( NS > 0 and x='0') then NS <= NS/2;
                        end if;
                end if;
        end process proc_a;

        proc_b: process (PS, X)
        begin
                if (PS > 31) then Z <= '1';
                else Z <= '0';
                end if;
        end process proc_b;
end quiz_arch;
```

TIMING DIAGRAM



Solution:



8U. Write VHDL code for a vending machine that accepts 5, 10 and 25 cents coins and deliver the product when $1 has been deposited.

Solution: