16 July 2013

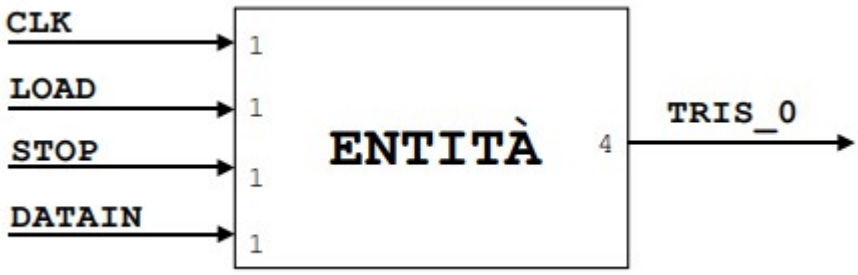
Create a clock-synchronous VHDL entity which implements a sequence controller for the Tic-Tac-Toe game.

Starting from a **LOAD** pulse, the entity must read from the serial input **DATAIN** and generate a 4-bit BCD-coded parallel output TRIS_0 containing the number of adjacent groups of 3 zeros ("tris") contained in the last 9 bits from **DATAIN**. This output must be maintained until a **STOP** pulse is given. If there no "tris" in the input sequence, the output must be set to high impedance.

Assume that data fills the 3x3 table from the top-left position:

1	0	1
1	0	1
1	0	0

In this <u>example</u> there is only 1 "tris" in the central column so the output must be "0001".



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


entity TicTacToe is
   port(CLK, LOAD, STOP, DATAIN: in std_logic;
        tris_0: out std_logic_vector (3 downto 0) );
end TicTacToe;


architecture Behavioral of TicTacToe is
   --reg: for uploding tic tac toe table;
   --zero_V1, zero_V2, zero_V3: for VERTICAL tris combinations;
   --zero_Or1, zero_Or2, zero_Or3: for HORIZONTAL ones;
   --zero_Ob1, zero_Ob2: for OBLIQUE ones;
   --cnt_1: to count during the DATAIN uploading;
   --check: for splitting the possible tris combinations storage
   --and the count of tris combinations.
   signal reg: std_logic_vector (8 downto 0);
   signal zero_V1, zero_V2, zero_V3: std_logic_vector (2 downto 0);
   signal zero_Or1, zero_Or2, zero_Or3: std_logic_vector (2 downto 0);
   signal zero_Ob1, zero_Ob2: std_logic_vector (2 downto 0);
   signal cnt_1: integer range 0 to 20:=0;
   signal check: std_logic:='0';
   begin

      process (CLK, STOP)
      --to count tris combinations.
      --NOTE: it's a variable because an immediate count is required.
      variable cnt_2: integer range 0 to 10:=0;

      begin
         --If a STOP pulse occurs, a general reset is executed:
         if (STOP = '1') then
            reg <= (others => '0');
            tris_0 <= (others => '0');
            cnt_1 <= 0;
            cnt_2 := 0;
            check <= '0';

         elsif (rising_edge(CLK)) then
            if (LOAD = '1') then
               if (cnt_1 < 9)then   --Just 9 values must be uploaded.
                  reg <= reg(7 downto 0) & DATAIN;
                  cnt_1 <= cnt_1 + 1;
               end if;

               if(cnt_1 >= 9) then
                  --if DATAIN uploading is complete, all the possible tris combination
                  --are stored.
                  --Since the signal assignation is done on the next clock pulse
                  --"check" is used so that tris combinations are counted once all
                  --the "zeroes" vectors have been assigned:
                  --at the beginning check is equal to '0'; after the storage of
                  --zeroes vectors, check is assigned to '1', but it will be equal
                  --to '1' on the next clock pulse, so just the first case is executed;
                  -- on the next clock pulse, just the second case is executed;
                  --if check is equal to another value, nothing is executed.
                  case check is
                     when '0' => --Horizontal combinations:
                                 zero_Or1 <= reg (8 downto 6);
                                 zero_Or2 <= reg (5 downto 3);
                                 zero_Or3 <= reg (2 downto 0);

                                 --vertical combinations
                                 zero_V1 <= reg(8) & reg(5) & reg (2);
                                 zero_V2 <= reg(7) & reg(4) & reg (1);
                                 zero_V3 <= reg(6) & reg(3) & reg (0);

                                 --oblique combinations
                                 zero_Ob1 <= reg(8) & reg(4) & reg (0);
                                 zero_Ob2 <= reg(6) & reg(4) & reg (2);
                                 check <= '1';

```

```
                        --After all of the possible tris combination are stored,
                        --occurring tris combinations are counted by an increment
                        --of cnt_2 for each occurrence.
                        when '1' => if(zero_V1 = "000") then cnt_2 := cnt_2 + 1; end if;
                                    if(zero_V2 = "000") then cnt_2 := cnt_2 + 1; end if;
                                    if(zero_V3 = "000") then cnt_2 := cnt_2 + 1; end if;
                                    if(zero_Or1 = "000") then cnt_2 := cnt_2 + 1; end if;
                                    if(zero_Or2 = "000") then cnt_2 := cnt_2 + 1; end if;
                                    if(zero_Or3 = "000") then cnt_2 := cnt_2 + 1; end if;
                                    if(zero_Ob1 = "000") then cnt_2 := cnt_2 + 1; end if;
                                    if(zero_Ob2 = "000") then cnt_2 := cnt_2 + 1; end if;

                                    --Outuput is computed!
                                    if ( cnt_2 /= 0) then
                                       tris_0 <= conv_std_logic_vector (cnt_2, 4);
                                    else
                                       tris_0 <= "ZZZZ";
                                    end if;

                                    cnt_2 := 0;
                                    --the last instruction is necessary, because this case
                                    --will be executed until a STOP pulse occurs.
                                    --So, everytime this case is executed, cnt_2 starts
                                    --from 0 and the same value is used to compute tris_0.
                        when others => null;
                     end case;
                  end if;

               else --if (LOAD /= '1')
                  tris_0 <= "ZZZZ";
               end if;
            end if;

         end process;

   end Behavioral;
```

```vhdl
--      Possibile Soluzione della Prova del 16 Luglio 2013      -    Giaconia         --
-- La codifica si basa sulla presenza di un  contatore ed un registro a 9 bit che     --
-- ospita i dati in ingresso. L'entità carica i dati per 9 clk quando il load è attivo--
-- In modo concorrente vengono assegnati 8 registri a 3 bit che identificano le       --
-- sequenze tris, mentre una variabile aggiorna il loro conteggio. L'impulso di stop  --
-- infine azzerra contatore ed uscita.

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

entity TTT is
port ( clk,load,stop,datain: in std_logic;
       tris_0: out std_logic_vector (3 downto 0));
end TTT;

architecture Behavioral of TTT is
-- contatore e registro di appoggio per i dati d'ingresso
signal cnt: integer :=0; signal rttt: std_logic_vector(8 downto 0);
-- registri a tre bit per le sequenze tris a zero
signal r876,r543,r210,r852,r741,r630,r840,r642: std_logic_vector(2 downto 0);
begin
process (clk)
variable c_tris: integer:=0;    -- questa è la variabile di conteggio che deve
begin                           -- contare le sequenze tris più velocemente
if stop = '1' then              -- rispetto all'evoluzione del processo
    cnt <=0; tris_0 <="ZZZZ";   -- l'impulso di stop riazzera la macchina
elsif clk'event and clk='1' then
   if load ='1' then            -- il load abilita il caricamento dati
      rttt <= datain & rttt(8 downto 1); -- eccolo!

      if cnt <= 9 then          -- Per i primi 9 impulsi devo solo riempire il registro
         cnt <= cnt +1;
         tris_0 <="ZZZZ";
      else                      -- dopo posso contare le sequenze tris sfruttando i
       if r876 = "000" then c_tris := c_tris +1; end if; -- registri sequenza rxxx
       if r543 = "000" then c_tris := c_tris +1; end if; -- che aggiorno in modo
       if r210 = "000" then c_tris := c_tris +1; end if; -- concorrente (vedi alla fine)
       if r852 = "000" then c_tris := c_tris +1; end if; -- del process)
       if r741 = "000" then c_tris := c_tris +1; end if;
       if r630 = "000" then c_tris := c_tris +1; end if;
       if r840 = "000" then c_tris := c_tris +1; end if;
       if r642 = "000" then c_tris := c_tris +1; end if;
       tris_0 <= CONV_STD_LOGIC_VECTOR(c_tris,4); -- sfrutto la funzione di conversione
       c_tris :=0;                                -- per assegnare l'uscita tris_0
      end if;
    end if;
end if;
end process;
-- ecco l'assegnazione concorrente dei registri sequenza
r876 <= rttt(8 downto 6);              -- sequenza orizzontale
r543 <= rttt(5 downto 3);              -- sequenza orizzontale
r210 <= rttt(2 downto 0);              -- sequenza orizzontale
r852 <= rttt(8) & rttt(5) & rttt(2);   -- sequenza verticale
r741 <= rttt(7) & rttt(4) & rttt(1);   -- sequenza verticale
r630 <= rttt(6) & rttt(3) & rttt(0);   -- sequenza verticale
r840 <= rttt(8) & rttt(4) & rttt(0);   -- sequenza diagonale
r642 <= rttt(6) & rttt(4) & rttt(2);   -- sequenza diagonale
end Behavioral;
```