25 June 2013

Create a clock-synchronous VHDL entity which implements a vertical and horizontal parity generator.

Starting from a **LOAD** pulse, the entity must read four 4-bits words from the input **DATAIN** and a control signal **PARITY** which determines whether the parity is even or odd.

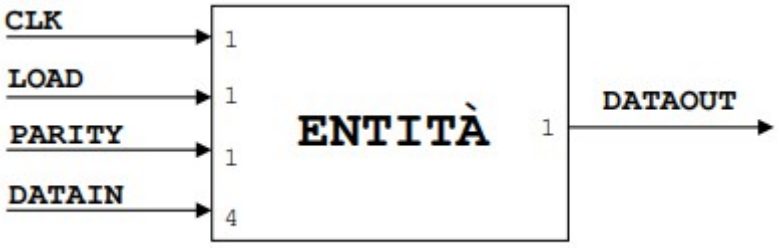
The entity must produce a serial output **DATAOUT** containing the input values and the computed parity values, properly collocated in time.

Example:

#	DATI IN INGRESSO				PL
1	0	0	1	1	1
2	1	1	1	1	1
3	0	0	0	1	0
4	0	1	0	1	1
PT	0	1	1	1	0

PL = Horizontal parity
PT = Vertical parity
DATI IN INGRESSO = Input Data

The entity must check the flow of input to avoid data accumulation inside the entity and emit output data in the shortest time possible.



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


entity Parity_Gen is
   port(CLK, LOAD, parity: in std_logic;
        DATAIN: in std_logic_vector (3 downto 0);
        DATAOUT: out std_logic);
end Parity_Gen;


architecture Behavioral of Parity_Gen is

   --regOut must contain 4 words of 4 bits, 4 trasverse parity bits,
   --4 longitudinal parity bits and one trasverse and longitudinal bit.
   signal regOut: std_logic_vector (24 downto 0):=(others => 'Z');
   signal cnt: integer range 0 to 35:=0;

   --to store parity input
   signal p: std_logic;

   begin

   process (CLK, LOAD)
   begin
      if (LOAD = '0') then
         DATAOUT <= 'Z';
         cnt <= 0;
      elsif rising_edge(CLK) then
         if (cnt < 31) then
            --Parity bit is computed as follows: the exclusive or (XOR) of the 4 bits of
            --DATAIN tells if the number of ones is even (0) or odd (1);
            --the XOR of this result and parity input gives the correct longitudinal
            --parity input for each DATAIN.

            --At the beginning, parity input is stored so that the same value (the first one
            --after LOAD becomes '1') is used during the output processing.
            --NOTE: Parity bit cannot be processed right after the assignment because the
            --present value of every signal is used during the process execution and the
            --new values are assigned once the process is ended. So the parity bit, for each
            --case, must be processed in the next case:
            case cnt is
               when 0 => p <= parity;
                         regOut(24 downto 21) <= DATAIN;
                         DATAOUT <= 'Z';
                         cnt <= cnt + 1;

               when 1 => regOut(19 downto 16) <= DATAIN;

                         regOut(20) <= regOut(24) xor regOut(23) xor
                                       regOut(22) xor regOut(21) xor p;

                         DATAOUT <= 'Z';
                         cnt <= cnt + 1;

               when 2 => regOut(14 downto 11) <= DATAIN;

                         regOut (15) <= regOut(19) xor regOut(18) xor
                                        regOut(17) xor regOut(16) xor p;

                         DATAOUT <= 'Z';
                         cnt <= cnt + 1;

               when 3 => regOut(9 downto 6) <= DATAIN;

                         regOut (10) <= regOut(14) xor regOut(13) xor
                                        regOut(12) xor regOut(11) xor p;

                         DATAOUT <= 'Z';
                         cnt <= cnt + 1;
```

```
77
78                --Trasverse parity bits must be processed after the whole input data is stored
79                --for sure; in this context also the last longitudinal parity bit is processed.
80                when 4 => regOut(5) <= regOut(9) xor regOut(8) xor
81                                       regOut(7) xor regOut(6) xor p;
82
83                          regOut(4) <= regOut(24) xor regOut(19) xor
84                                       regOut(14) xor regOut(9) xor p;
85
86                          regOut(3) <= regOut(23) xor regOut(18) xor
87                                       regOut(13) xor regOut(8) xor p;
88
89                          regOut(2) <= regOut(22) xor regOut(17) xor
90                                       regOut(12) xor regOut(7) xor p;
91
92                          regOut(1) <= regOut(21) xor regOut(16) xor
93                                       regOut(11) xor regOut(6) xor p;
94
95                          cnt <= cnt + 1;
96                          DATAOUT <= 'Z';
97
98                --When every longitudinal parity bit is processed, the last trasverse parity bit
99                --can be processed as well
100               when 5 => regOut(0) <= regOut(4) xor regOut(3) xor regOut(2) xor regOut(1) xor p;
101                         cnt <= cnt + 1;
102                         DATAOUT <= 'Z';
103
104               --Output processing: the MSB of regOut is sent to serial output and then all of
105               --the values of regOut are shifted to the left and the LSB is replaced by 'Z':
106               when 6 to 31 => DATAOUT <= regOut(24);
107                               regOut <= regOut(23 downto 0) & 'Z';
108                               cnt <= cnt + 1;
109               when others => null;
110
111            end case;
112
113         --After the output processing is finished, the output bitstream is set to high impedance
114         elsif (cnt >= 31) then
115            DATAOUT <= 'Z';
116         end if;
117
118      end if;
119
120   end process;
121
122   end Behavioral;
123
```

```
1   --      Possibile Soluzione della Prova del 25 Giugno 2013      -    Giaconia         --
2   -- La codifica si basa sulla presenza di un  contatore ed un registro a 25 bit che    --
3   -- ospita i dati in ingresso. L'entità nei primi 5 clock a partire dall'impulso Load  --
4   -- carica il registro con i dati e calcola tramite xor i bit di parità; poi fa uno    --
5   -- shift a sinistra del registro e mette in uscita il registro stesso. La macchina    --
6   -- riparte quando vi è un nuovo impulso di load azzerando il contatore e campionando  --
7   -- nuovamente il segnale di parità da usare                                           --
8
9   library IEEE; use IEEE.STD_LOGIC_1164.ALL;
10  use IEEE.STD_LOGIC_ARITH.ALL; use IEEE.STD_LOGIC_UNSIGNED.ALL;
11
12  entity parity_check is
13     port(clk, load, parity: in std_logic; din: in std_logic_vector(3 downto 0);
14        dout: out std_logic);
15  end parity_check;
16
17  architecture Behavioral of parity_check is
18  signal regpar:std_logic:='0'; signal cnt: integer :=0; -- segnali per parità ed il conteggio
19  signal vett: std_logic_vector(24 downto 0):=(others =>'Z');
20  begin
21  process(clk)
22  begin
23  if load ='1' then
24     cnt<=0; regpar<=parity;          -- qui azzero il conteggio e registro la parità
25  elsif clk'event and clk ='1' then   -- da usare
26     if cnt<=31 then cnt<=cnt+1; end if;  -- il contatore cadenza 32 stati (da 0 a 31)
27
28     case cnt is
29        when 0 => vett(24 downto 21) <= din;  -- carico il primo dato
30        when 1 => vett(19 downto 16) <= din;  -- carico il secondo dato
31                  -- calcolo la parità del primo con i dati al passo precedente
32                  vett(20) <= vett(24) xor vett(23)xor vett(22)xor vett(21)xor regpar;
33        when 2 => vett(14 downto 11) <= din; -- carico il terzo dato
34                  vett(15) <= vett(19) xor vett(18)xor vett(17)xor vett(16)xor regpar;
35        when 3 => vett(9 downto 6) <= din;  -- carico il quarto dato
36                  vett(10) <= vett(14) xor vett(13)xor vett(12)xor vett(11)xor regpar;
37                  -- calcolo la parità del quarto dato
38        when 4 => vett(5) <= vett(9) xor vett(8)xor vett(7)xor vett(6)xor regpar;
39                  -- calcolo i 4 bit per la parità trasversale
40                  vett(4) <= vett(24) xor vett(19)xor vett(14)xor vett(9)xor regpar;
41                  vett(3) <= vett(23) xor vett(18)xor vett(13)xor vett(8)xor regpar;
42                  vett(2) <= vett(22) xor vett(17)xor vett(12)xor vett(7)xor regpar;
43                  vett(1) <= vett(21) xor vett(16)xor vett(11)xor vett(6)xor regpar;
44                  -- calcolo la parità longitudinale del dato di par. trasversale
45        when 5 => vett(0) <= vett(20) xor vett(15)xor vett(10)xor vett(5)xor regpar;
46                        -- faccio lo shift a sinistra dei dati ed emetto in uscita
47        when 6 to 31 => vett <= vett(23 downto 0) & 'Z';
48                        dout <=vett(24);
49        when others => null;
50     end case;
51  end if;
52  end process;
53  end Behavioral;
54
```