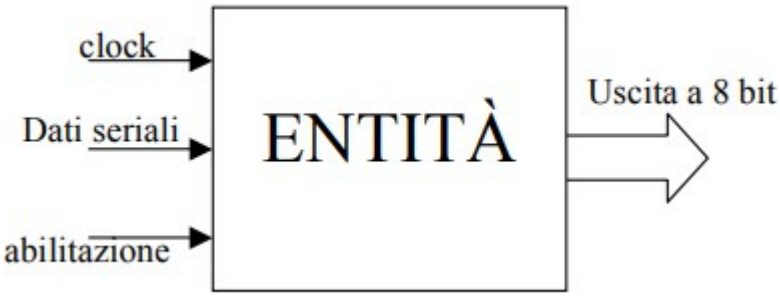
24 June 2005

Implement a clock-synchronous VHDL entity with a serial data input and an enable input signal to activate the entity.

The entity receives 16 bits as serial input, from the least significant bit (b0) to the most significant bit (b15), and transfer these 16 bits to a parallel, 8-bits output in 2 consecutive bytes with the format:

`[b15, … , b8]` followed by `[b7, …, b0]`.



Dati seriali = Serial data
abilitatione = enable
Uscita = Output

```
1   library IEEE;
2   use IEEE.STD_LOGIC_1164.ALL;
3   use IEEE.STD_LOGIC_ARITH.ALL;
4   use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6
7   entity SPConv is
8      port(CLK, Sin, EN: in std_logic;
9           Pout: out std_logic_vector (7 downto 0));
10  end SPConv;
11
12
13  architecture Behavioral of SPConv is
14
15  signal cnt: integer range 0 to 20:=0;
16  signal B: std_logic_vector (15 downto 0):= (others => 'Z');
17
18     begin
19
20        process(CLK, EN)
21        begin
22           if (EN = '0') then
23              cnt <= 0;
24              B <= (others => 'Z');
25           elsif rising_edge(CLK) then
26              if (cnt < 18) then
27                 case cnt is
28                    when 0 to 15 => B <= Sin & B(15 downto 1);
29                                    cnt <= cnt + 1;
30                    when 16 => Pout <= B(15 downto 8);
31                               cnt <= cnt + 1;
32                    when 17 => Pout <= B(7 downto 0);
33                               cnt <= cnt + 1;
34                    when others => null;
35                 end case;
36              end if;
37
38           end if;
39
40        end process;
41
42  end Behavioral;
```

```
1   library IEEE;
2   use IEEE.STD_LOGIC_1164.ALL;
3   use IEEE.STD_LOGIC_ARITH.ALL;
4   use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6   --Serial input: 16 bits from LSB to MSB;
7   --Parallel output: 2 bytes tranferred in two clock periods of time
8   --(one period for each byte): first period: MSBs; second period: LSBs.
9   --if the entity is not enabled, POUT must be set in high impedance state.
10
11  --NB: 16 bit are stored from LSB to MSB, but must be tranferred to PUOT
12  --in reverse order; the first byte cannot be tranferred to POUT until
13  --all of the 16 bit have been uploaded from SIN
14
15
16  entity S_to_P is
17     port (CLK, SIN, EN: in std_logic;
18           POUT: out std_logic_vector (7 downto 0));
19  end S_to_P;
20
21  architecture Behavioral of S_to_P is
22     signal cnt: integer range 0 to 18:=0;
23     signal regO1, regO2, regOUT: std_logic_vector (0 to 7):=(others => 'Z');
24
25  begin
26
27  process (CLK)
28  begin
29  if (rising_edge(CLK)) then
30     --1) upload LSBs in regO1 (0<=cnt<8) and regOUT='Z';
31     if (cnt < 8) then
32        regO1 <= SIN & regO1(0 to 6);
33        regOUT <= (others => 'Z');
34        cnt <= cnt + 1;
35     --2) upload MSBs in regO2 (8<=cnt<16) and regOUT='Z';
36     elsif (cnt > 7 and cnt < 16) then
37        regO2 <= SIN & regO2 (0 to 6);
38        regOUT <= (others => 'Z');
39        cnt <= cnt + 1;
40     --3) MSBs transferring to regOUT (cnt=16);
41     elsif (cnt = 16) then
42        regOUT <= regO2;
43        cnt <= cnt + 1;
44     --4) LSBs tranferring to regOUT (cnt=17);
45     elsif (cnt = 17) then
46        regOUT <= regO1;
47        cnt <= cnt + 1;
48     else
49        regOUT <= (others => 'Z');
50     end if;
51  end if;
52  end process;
53
54  --5) regOUT is tranferred to POUT if EN='1', otherwise POUT='Z':
55  POUT <= regOUT when (EN = '1') else "ZZZZZZZZ";
56
57  end Behavioral;
58
```

This is a solution that was provided by the teacher.
It uses 2 processes instead of 1.
In my opinion it was unnecessary, but it can be useful to show how to use multiple processes.

```
-- This solution splits the problem into 2 processes (pro1, pro2)
-- The first defines the functioning states.
-- The second determines the output and/or the state of the internal signals

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

entity converter1to2x8 is
      port (ck, sdin, en: in STD_LOGIC;
            OutByte: out STD_LOGIC_VECTOR (7 downto 0) );
end converter1to2x8;

architecture Behavioral of converter1to2x8 is
      -- 4 possible states
      type stato is (S1, S2, S3, S4);
      signal CS: stato;
      -- instantiation of 2 8-bits registers to memorize input data
      signal reg8MSByte, reg8LSByte: STD_LOGIC_VECTOR(7 downto 0);
      signal conta: INTEGER range 0 to 31;

begin
-- 2 distinct processes:
-- The first determines how to pass from a state to the others
-- The second decides what to do in each state

pro1: process(ck,en)
begin
      if (ck'event and ck='1') then
            if en ='1' then
                  conta <= conta+1;
                  if conta <= 7 then
                        CS <= S1;
                  elsif conta > 7 and conta < 15 then
                        CS <=S2;
                  elsif conta = 16 then
                        CS <=S3;
                  elsif conta > 16 then
                        CS <=S4;
                        conta <= 0;
                  end if;
            end if;
      end if;
end process;

pro2: process (CS,conta)
begin

case CS is
      when S1 =>
            -- let the LSBs flow into reg8LSByte
            reg8LSbyte <= sdin & reg8LSbyte (7 downto 1);
            -- output is in high impedance in the meantime
            OutByte <= "ZZZZZZZZ";
```

```
      when S2 =>
            -- let the MSBs flow into reg8LSByte
            reg8MSbyte <= sdin & reg8MSbyte (7 downto 1);
            OutByte <= "ZZZZZZZZ";
      when S3 =>
            -- in S3 the MSBs are transferred to the output
            OutByte(7 downto 0)<= reg8MSbyte(7 downto 0);
      when S4 =>
            -- in S4 the LSBs are transferred to the output
            OutByte(7 downto 0)<= reg8LSbyte(7 downto 0);
      end case;

end process;
end Behavioral;
```