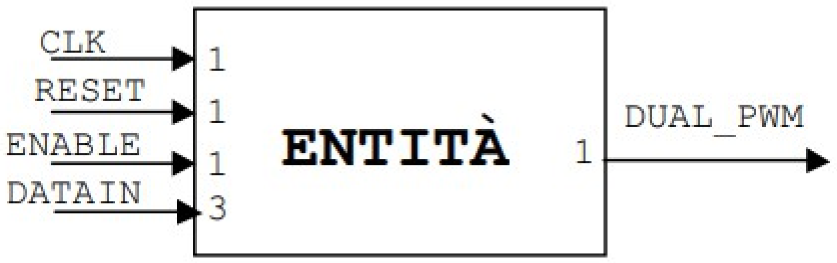
16 July 2008

Create a clock-synchronous VHDL entity which implements a variable PWM generator.
Starting from a reset, the entity must wait to receive 2 values at the 3-bits input **DATAIN**. These values are interpreted as duty cycles for the output **DUAL_PWM**.
The output is normally at high impedance but when the second **DATAIN** value is received, it generates a signal whose period is 1/10th of the clock and the PWM value alternate between the 2 input values.

Example:
If the input values are 110 (6) and 010 (2), the output is 1 for 6 clock cycles and 0 for 4, then it is 1 again for 2 clock cycles and 0 for 8, and then it repeats.

Finally, when **ENABLE** is set to 0, the entity is deactivated and the output is at high impedance.



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

entity PWM is
   port (CLK, RST, EN: in std_logic;
         DATAIN: in std_logic_vector (2 downto 0);
         DUALPWM: out std_logic);
end PWM;

architecture Behavioral of PWM is
   --reg1, reg2: to store the input;
   --P1, P2, i, j: Forfcomputing the output;
   --cnt: to scan all the operations while the entity is enabled.
   signal reg1: std_logic_vector (2 downto 0):="000";
   signal reg2: std_logic_vector (2 downto 0):="000";
   signal P1, P2, i, j: integer range 0 to 10:=0;
   signal cnt: integer range 0 to 25:=0;
   begin

      process (CLK, RST, EN)
      begin
         if (RST = '1') then
            --General reset:
            DUALPWM <= 'Z';
            reg1 <= "000";
            reg2 <= "000";
            P1 <= 0;
            P2 <= 0;
            cnt <= 0;
            i <= 0;
            j <= 0;

         elsif rising_edge(CLK) then
            --Entity enabled:
            if (EN = '1') then
               case cnt is
                  --In the first two cases, the input is stored:
                  when 0 => reg1 <= DATAIN;    --(1)
                            cnt <= cnt + 1;
                            DUALPWM <= 'Z';

                  when 1 => reg2 <= DATAIN;    --(2)
                            cnt <= cnt + 1;
                            DUALPWM <= 'Z';

                  --then they are converted to integer:
                  when 2 => P1 <= conv_integer(reg1);
                            P2 <= conv_integer(reg2);
                            DUALPWM <= 'Z';
                            cnt <= cnt + 1;

                  --For ten pulses the output is computed:
                  --during the first pulses (in numbers of P1)
                  --output is high and for the remaining ones is low;
                  --at the end i is set to 0 so the computation can be
                  --repeated.
                  when 3 to 12 => if (i < P1) then
                                     DUALPWM <= '1';
                                     i <= i + 1;
                                     cnt <= cnt + 1;
                                  else
                                     DUALPWM <= '0';
                                     cnt <= cnt + 1;
                                  end if;

                                  if (cnt = 12) then
                                     i <= 0;
                                  end if;
                  --similar computation for the second input value;
                  --at the end j is set to zero and cnt is set to 3
                  --so the computation can start from the beginning until
                  --a reset occurs.
                  when 13 to 22 => if (j < P2) then
                                      DUALPWM <= '1';
                                      j <= j + 1;
                                      cnt <= cnt + 1;
                                   else
                                      DUALPWM <= '0';
                                      cnt <= cnt + 1;
                                   end if;

                                   if (cnt = 22) then
                                      j <= 0;
                                      cnt <= 3;
                                  end if;

                  when others => null;
               end case;

            --Entity Disabled.
            else
               DUALPWM <= 'Z';
            end if;
         end if;
      end process;

end Behavioral;
```