TDD: Test driven development.

Contador decimal con precarga, reset sincrono y clock enable

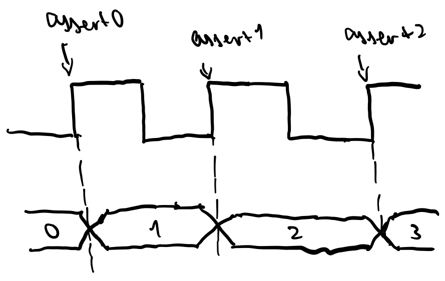
1- Escribir un test para comprobar una caracteristica
2- Ejecutar el test
3- No falla -> Volver a 1.
4- Diseñar para incorporar una caracteristica
5- Ejecutar el test
6- Falla -> Volver a 4.
7- Repetir para la siguiente caracteristica

```
-- Design
Library IEEE;
Use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity bcd_cntr is
 port(
   Clk : in std_logic; --Leading-edge active clock
   rst_n : in std_logic; -- Active-low asynchronous reset
    ce : in std_logic; -- Clock enable.
   load : in std_logic; -- Synchronous count load.
   din : in unsigned (3 downto 0); -- Count input.
   dout : out unsigned (3 downto 0); -- Count output
   );
 end entity bcd_cntr;

 architecture behavioral of bcd_cntr is
  signal : dout_i : unsigned (dout'range);
  begin
    dout <= dout_i;
    Process (Clk, rst_n)
     begin
     if rst_n = '0' then
      dout_i <= (others => '0');
      elsif rising_edge (CLK) then

       if ce = '1' then
          if load = '1' then
            dout_i <= din
          else
            dout_i <= (dout_i+1) mod 10;    σ
                                            | if dout_i < 9 then
                                            |   dout_i <= dout_i + 1;
                                            | else
                                            |   dout_i <= (others => '0');
                                            | end if;
          end if;
       end if;
    end if;
    end process;
    end architecture behavioral;
```

Doxygen = Generar info a partir de comentarios

```
-- testbench:

Library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity bcd_cntr_tb is

end entity;

architecture testbench of bcd_cntr_tb is
Component bcd_cntr is
 port(
   Clk : in std_logic:='0'; --Leading-edge active clock
   rst_n : in std_logic; -- Active-low asynchronous reset
    ce : in std_logic; -- clock enable.
   load : in std_logic; -- synchronous count load.
   din : in unsigned (3 downto 0); -- count input.
   dout : out unsigned(3 downto 0); -- Count output
   );

   end component;
      Signal Clk : std_logic,
      Signal rst_n : std_logic;
      signal ce : std_logic;
      signal load : std_logic;
      Signal din : unsigned (3 downto 0);
      Signal dout : unsigned(3 downto 0);
      Constant clk_freq: positive := 100_000_000;
      Constant clk_period : time := 1sec/clk_freq;
      begin
       UUT: bcd_cntr is
       port map (
              Clk => Clk,
              rst_n => rst_n,
              ce => ce,
              load => load,
              din => din,
              dout => dout
              );
```

```
genclk : clk <= not clk after 0,5 * 10 ns;
Stimulous_gen: process
begin
 -- Implementación de la función reset.
rst-n <= '0' after 0,25 * clk_period,
         '1' after 0,75 * clk_period;

wait until rst_n = '1';
assert dout = 0
 report "rst_n malfunction"
  severity failure;
 -- Implementación de la función contar.
ce <= '1';
load <= '0';
for i in 0 to 12 loop
Wait until clk = '1';
assert dout = i mod 10
    report "[FAILURE]: count malfunction"
    Severity failure;
 end loop;
 -- Implementación de la función carga
din <= X"6"; o din <= to_unsigned(6, din'length);
 Wait until clk = '0';
  Load <= '1';
  Wait until clk = '1';
  Wait for 0,1 * CLK_PERIOD;
  assert dout = 6
   report "[FAILURE]: Load malfunction"
    severity failure;

    for i in 1 to 2 loop
        Wait until Clk = '1';
     end loop;
   assert false.
   report "[PASSED]: simulation finished"
    Severity failure;
 end process;
```



```
Ce_monitor : process (Clk)
begin
 Wait until Clk ='1';
 Wait for 0,1 * CLK_PERIOD;
  if Ce = '1' and rst_n /= '0' then
   assert dout /= dout'delayed(0,11 * CLK_PERIOD)
    report "[FAILURE]: Clock enable malfunction"
     Severity failure;
   else
    assert dout /= dout'delayed(0,11 * CLK_PERIOD)
     report "[FAILURE]: Clock enable malfunction"
      Severity failure;
   end if;
   end process;

   end architecture testbench;
```