| CE312 | Report<br>TP1 - Modulo 16, 4-bits Counter |
|---|---|
| **Date:** | 03/11/2020 |
| **Students:** | CELLARD Rémi - Damien Pierson-Maury |
| **Subject:** | Modulo 16 Counter on 4-bits |

## 1. Introduction

In this report, we'll see how we can create a modulo 16 4-bits counter with VHDL code and simulate it in Vivado.

## 2. Code of the modulo 16 4-Bits counter component

All the code for the component and the testbench are included at the end of the report, in annexes 1 and 2.
The chronogram of the testbench made by hand can be found in annex 3 and the chronogram resulting of the simulation in Vivado can be found in annex 4.

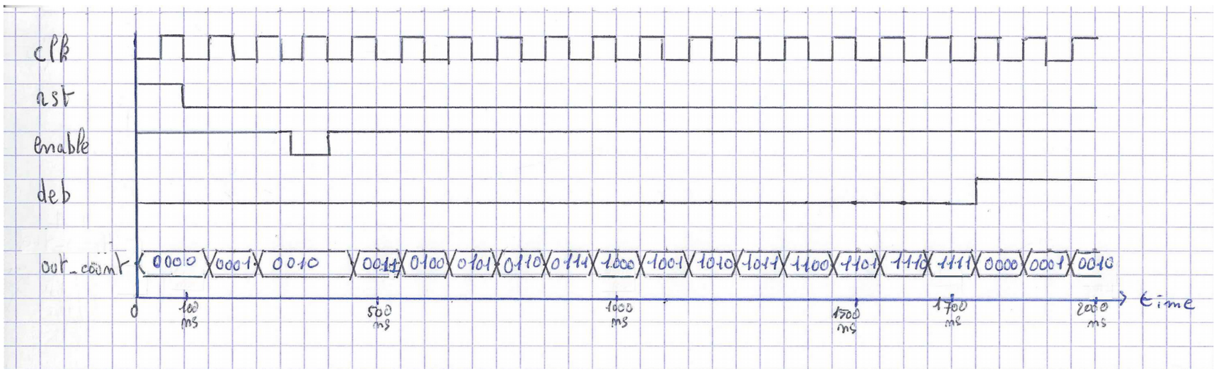
*Annex 1 - Modulo 16 counter component code*

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity compteur_16 is
port (
        clk : in STD_LOGIC; -- clock signal
        rst : in STD_LOGIC; -- reset signal
        enable : in STD_LOGIC; -- enable signal
        deb  : out STD_LOGIC; -- bit overflow
        out_count : out STD_LOGIC_VECTOR (3 downto 0) -- counter signal
);
end entity;

architecture archcompteur of compteur_16 is
  signal debtmp : STD_LOGIC := '0';
  signal outtmp : STD_LOGIC_VECTOR (3 downto 0) := (others=>'0');
  -- two signal for made operations on it, because we can’t on out an in signal
  begin
        process(clk)   -- active when event happen on signal clk
        begin
        if (clk'event and clk='1') then
          if rst='1' then
            debtmp <= '0';
            outtmp <= (others=>'0');
          elsif (enable='1') then     -- no reset, counter enable
            if (outtmp = "1111") then -- add bit overflow + reset out_count
              outtmp <= (others=>'0');
              debtmp <= not debtmp;
            else
              outtmp <= std_logic_vector(unsigned(outtmp)+ 1);
              -- add 1 to unsigned
            end if;
          end if;
        end if;
        end process;

        -- output assignment
        deb <= debtmp;
        out_count <= outtmp;
end architecture;
```

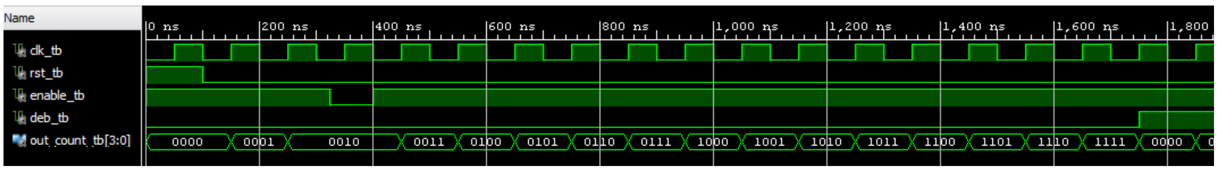
*Annex 2 - Testbench code for the modulo 16 counter component*

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
entity tb_compteur_16 is
end tb_compteur_16;
architecture tb of tb_compteur_16 is
  component compteur_16
            port (
               clk : in STD_LOGIC;
               rst : in STD_LOGIC;
               enable : in STD_LOGIC;
               deb  : out STD_LOGIC; -- bit overflow
               out_count : out STD_LOGIC_VECTOR (3 downto 0)
            );
   end component;
   signal clk_tb, rst_tb, enable_tb, deb_tb : STD_LOGIC;
   signal out_count_tb : STD_LOGIC_VECTOR (3 downto 0);
   begin
            compteur: compteur_16 port map (clk=>clk_tb, rst=>rst_tb, enable=>enable_tb, deb=>deb_tb,
out_count=>out_count_tb);
            clock_10mhz_generation : process -- simulation of the clock signal
            begin
               clk_tb <= '0';
               wait for 50 ns;
               clk_tb <= '1';
               wait for 50 ns;
            end process;
            procreset: process -- simulation of the  reset signal
            begin
               rst_tb <= '1';
               wait for 100 ns; -- we maintain the rst_tb signal on 100 ns
               rst_tb <= '0';
               wait for 1000 ns; -- we maintain the rst_tb signal on 100 ns
               rst_tb <= '1';
               wait for 100 ns; -- we maintain the rst_tb signal on 100 ns
               rst_tb <= '0';
               wait; -- we remove it permanently
            end process;
            procenable: process -- simulation of the enable signal
            begin
               enable_tb <= '1';
               wait for 325 ns;
               enable_tb <= '0';
               wait for 75 ns;
               enable_tb <= '1';
               wait;
            end process;
end architecture;
```

*Annex 3 - Hand-processed chronogram of the testbench*



*Annex 4 - Vivado-processed chronogram of the testbench, after simulation*


Name
clk_tb
rst_tb
enable_tb
deb_tb
out_count_tb[3:0]
0 ns
200 ns
400 ns
600 ns
800 ns
1,000 ns
1,200 ns
1,400 ns
1,600 ns
1,800
0000
0001
0010
0011
0100
0101
0110
0111
1000
1001
1010
1011
1100
1101
1110
1111
0000