UNIVERSITÉ TOULOUSE III PAUL SABATIER

Université de Toulouse

Rapport sur la fonction simple du BE_VHDL

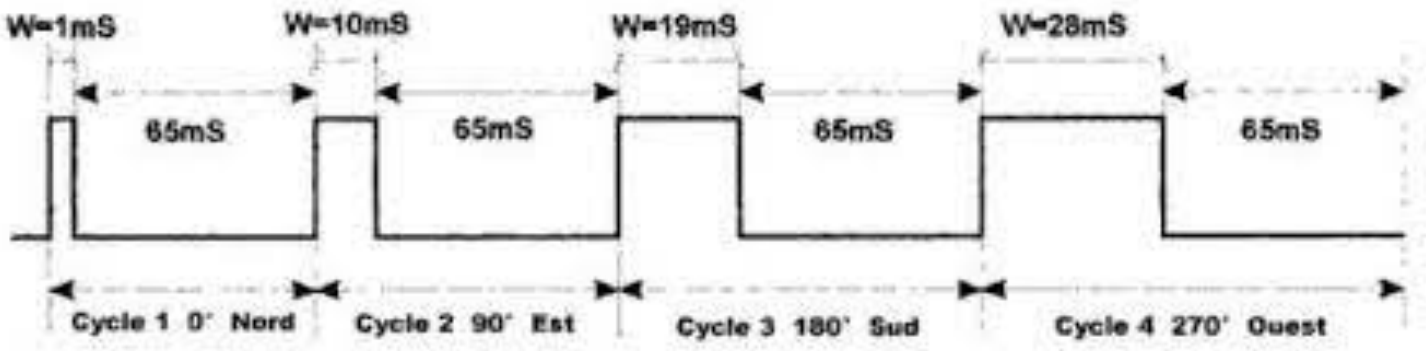
Gestion Compas

Rédigé par :

MBOUNGOU Frangely
BA WAZIR Ahmed

Introduction

L'objectif de ce rapport est de présenter la fonction simple que nous avons choisi dans notre BE, dans notre cas il s'agit du module gestion_compas pour boussole. Cette fonction a pour but de récupérer des mesures d'angles afin de fixer le cap. Le module compas permet de faire l'acquisition de données, pour en délivrer les directions suivante Nord, Sud, Est et Ouest.
Celui-ci utilise notamment un signal d'entrée PWM qui fait en sorte que lorsque la boussole se met en rotation, une impulsion est générée. La largeur d'impulsion varie de 1 ms pour 0° à 36,99 ms pour 359,9°, cela équivaut à 100 ms/degré avec un décalage de +1 ms pour 0°. On peut donc déduire que 1 ms est équivalent à 10°.



La période a donc une durée minimum de 66ms (pour un angle de 0º) et de 102 ms maximum (pour un angle de 359,9º).

I - Analyse fonctionnelle

Le module compas présente les entrées et sorties suivantes :

Entrées

- clk_50M : une horloge à 50 MHZ

- raz_n : reset actif à 0 afin d'initialiser le circuit

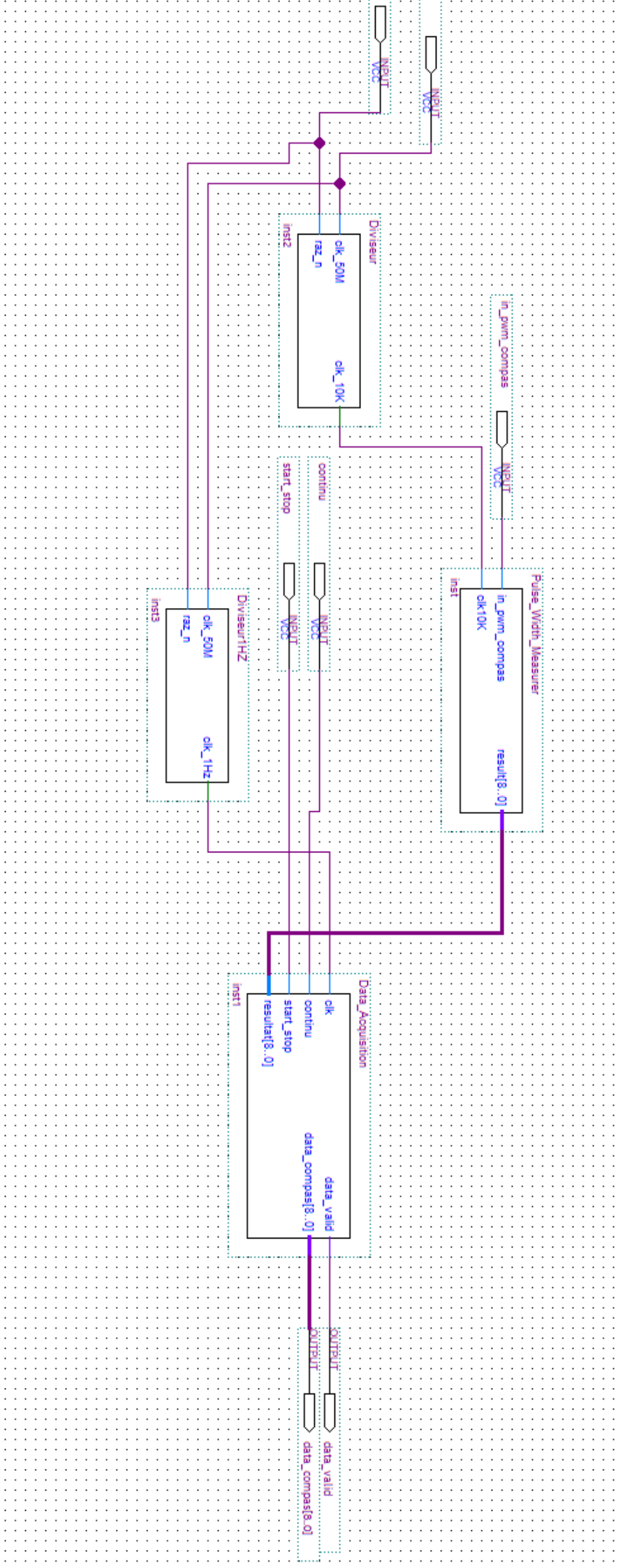
- in_pwm_compas : signal PWM d'entrée de la boussole qui va donc varier entre 1 ms et 36,9 ms
- start_stop : elle permet de démarrer une acquisition lorsque le mode fixé est en monocoup et que c'est = 1.
- Continu : permet de fixer le mode de fonctionnement du compas, soit en mode en continu lorsque continu = 1 en rafraichissement les données toutes les secondes. Sinon si continu = 0, on est donc en mode monocoup qui est donc activé si l'entrée start_stop = 1 pour valider la prise en compte des données.

Sorties

- data_compas : elle permet d'exprimer en degré la valeur du cap codé sur 9 bits.

- data_valid : elle permet de vérifier la validité d'une mesure. Elle est notamment nécessaire quand le mode de fonctionnement est en mode monocoup afin d'être sûr de notre valeur de sortie data_compas.

On peut donc décomposer la gestion du module compas tel que :


VCC
INPUT
VCC
INPUT
Diviseur
inst2
clk_50M
raz_n
clk_10K
in_pwm_compas
VCC
INPUT
start_stop
VCC
INPUT
continu
VCC
INPUT
Diviseur1HZ
inst3
clk_50M
raz_n
clk_1Hz
Pulse_Width_Measurer
inst
clk10K
in_pwm_compas
result[8..0]
Data_Acquisition
inst1
clk
continu
start_stop
resultat[8..0]
data_valid
data_compas[8..0]
output
output
data_compas[8..0]
data_valid

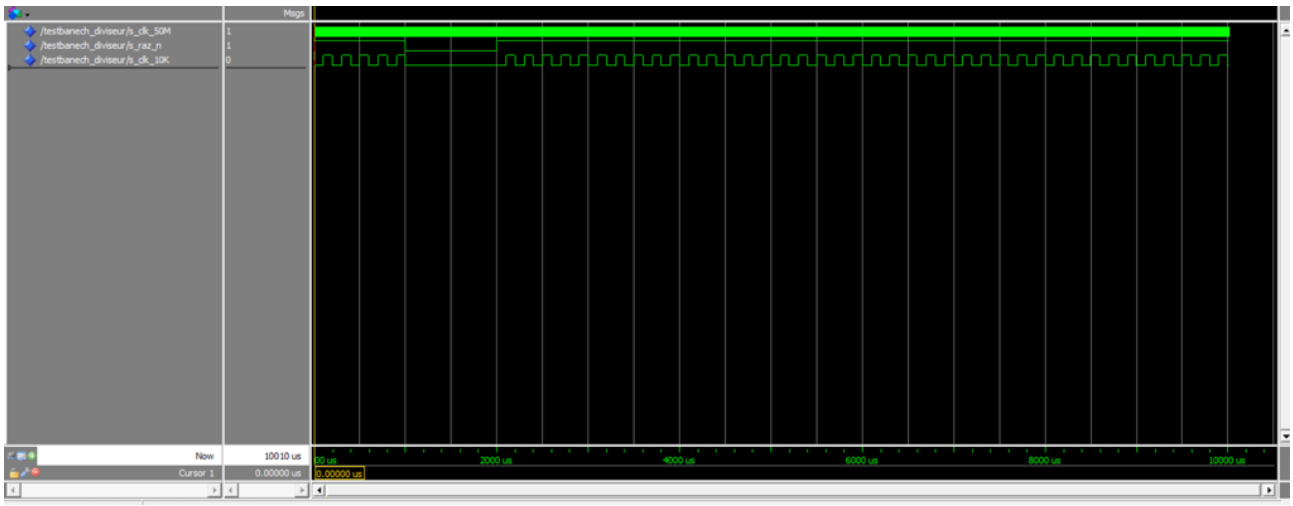
- Diviseur : ce bloc représente un diviseur permettant de générer un signal de fréquence à 10 kHz équivalent à un degré. Il va notamment permettre de déterminer le nombre de degrés dans chaque période du signal PWM.

```
--Diviseur de fr?quence
LIBRARY ieee;
USE ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;


entity Diviseur is
    Port ( clk_50M : in STD_LOGIC;
           raz_n : in STD_LOGIC;
           clk_10K : inout STD_LOGIC
           );
end Diviseur;

architecture Behavioral of Diviseur is

begin
    process (clk_50M, raz_n)
       variable count : integer range 0 to 4999 ; -- Compteur pour diviser le signal

    begin
        if raz_n = '0' then
            count := 0; -- Réinitialisation du compteur lorsque raz_n est actif
            clk_10K <= '0';
       else if (clk_50M'event and clk_50M='1') then
         count := count + 1;
           if count = 4999 then
               count := 0;
               clk_10K <= not clk_10K; -- Inversion de la sortie pour générer une fréquence de 10 KHz
           else
               count := count ;
              -- clk_10K <= '1';
           end if;
       end if;
      end if;

    end process;
end Behavioral;
```

Code du diviseur à 10 khz

```
architecture arch_testbanech_Diviseur of testbanech_Diviseur is

   component Diviseur is
   Port (
         clk_50M : in STD_LOGIC :='0';--on définit l'entrée  clk_50M
         raz_n : in STD_LOGIC:='0';--on définit l'entrée  raz_n
         clk_10K : inout STD_LOGIC:='0'--on définit l'entrée  clk_10K
           );
   end component ;
    signal   s_clk_50M :  STD_LOGIC :='0';--on crée le signal qui est relié avec l'entrée clk_50M
    signal   s_raz_n   :  STD_LOGIC:='0';--on crée le signal qui est relié avec l'entrée s_raz_n
    signal   s_clk_10K :  STD_LOGIC:='1'; --on crée le signal qui est relié avec la sortie clk_10k
   begin
   Diviseur1 :Diviseur
   port map (
   clk_50M => s_clk_50M ,
   raz_n => s_raz_n,
   clk_10K => s_clk_10K
   );
   tb_clk_50M_process :process
   begin
     s_clk_50M <= not(s_clk_50M);-- mettre les valeur de s_clk_50M pour la Simulation
     wait for 10ns;
   end process ;
   tb_reset_process:process
   begin
   s_raz_n<= '1' ;-- mettre les valeur de s_raz_n pour la Simulation
   wait for 1ms;
   s_raz_n<= '0' ;
   wait for 1ms;
   s_raz_n<= '1' ;
   wait;
   end process;

end arch_testbanech_Diviseur;
```

Code du testbench



Simulation du diviseur 10 khz

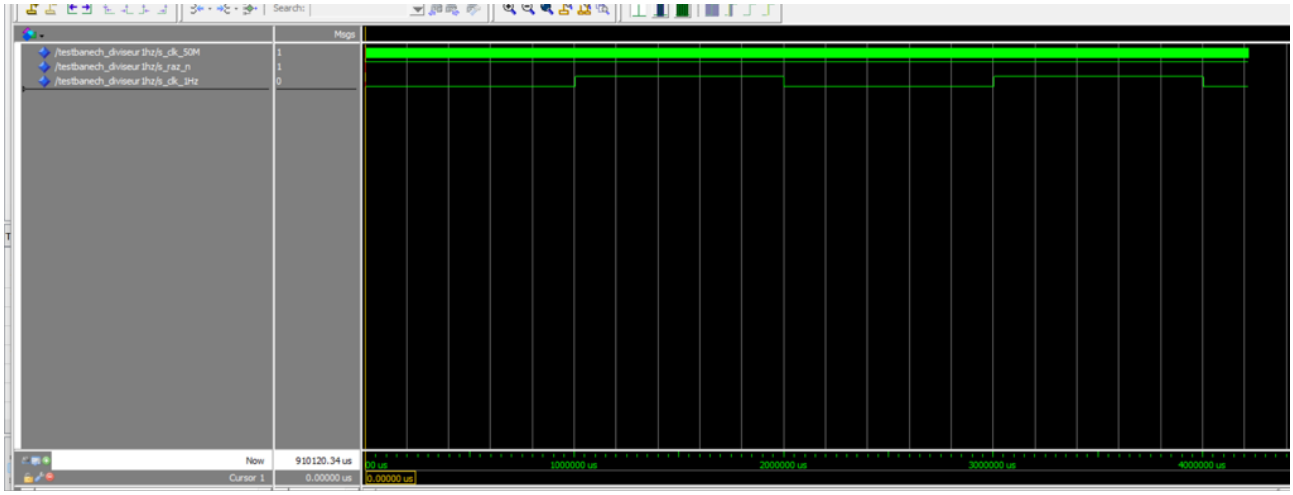
- Diviseur1HZ : Ce bloc va permettre de générer la seconde pour le rafraîchissement des données

```
1   --Diviseur de fr?quence
2   LIBRARY ieee;
3   USE ieee.std_logic_1164.all;
4   use ieee.std_logic_unsigned.all;
5
6
7   entity Diviseur1HZ is
8       Port ( clk_50M : in STD_LOGIC;
9              raz_n : in STD_LOGIC;
10             clk_1Hz : inout STD_LOGIC
11             );
12  end Diviseur1HZ;
13
14  architecture Behavioral of Diviseur1HZ is
15  begin
16  process (clk_50M, raz_n)
17          variable count : integer range 0 to 49999999 ; -- Compteur pour diviser le signal
18
19      begin
20          if raz_n = '0' then
21              count := 0; -- Réinitialisation du compteur lorsque raz_n est actif
22              clk_1Hz <= '0';
23          else if (clk_50M'event and clk_50M='1') then
24            count := count + 1;
25              if count = 49999999 then
26                  count := 0;
27                  clk_1Hz <= not clk_1Hz; -- Inversion de la sortie pour générer une fréquence de 1Hz
28              else
29                  count := count ;
30
31              end if;
32          end if;
33        end if;
34
35      end process;
36  end Behavioral;
```

Code du diviseur à 1 hz

```
architecture arch_testbanech_Diviseur1HZ of testbanech_Diviseur1HZ is

    component Diviseur1HZ is
    Port (
          clk_50M : in STD_LOGIC :='0';--on définit l'entrée  clk_50M
          raz_n : in STD_LOGIC:='0';--on définit l'entrée  raz_n
          clk_1Hz : inout STD_LOGIC:='0'--on définit l'entrée  clk_1Hz
            );
    end component ;
     signal   s_clk_50M :  STD_LOGIC :='0';--on crée le signal qui est relié avec l'entrée clk_50M
     signal   s_raz_n   :  STD_LOGIC:='0';--on crée le signal qui est relié avec l'entrée s_raz_n
     signal   s_clk_1Hz :  STD_LOGIC:='1'; --on crée le signal qui est relié avec la sortie clk_1Hz
    begin
    Diviseur2 :Diviseur1HZ
    port map (
    clk_50M => s_clk_50M ,
    raz_n => s_raz_n,
    clk_1Hz => s_clk_1Hz
    );
    tb_clk_50M_process :process
    begin
      s_clk_50M <= not(s_clk_50M);-- mettre les valeur de s_clk_50M pour la Simulation
      wait for 10ns;
    end process ;
    tb_reset_process:process
    begin
    s_raz_n<= '1' ;-- mettre les valeur de s_raz_n pour la Simulation
    wait for 1ms;
    s_raz_n<= '0' ;
    wait for 1ms;
    s_raz_n<= '1' ;
    wait;
    end process;

end arch_testbanech_Diviseur1HZ;
```

Code du testbench



Simulation du diviseur 1hz

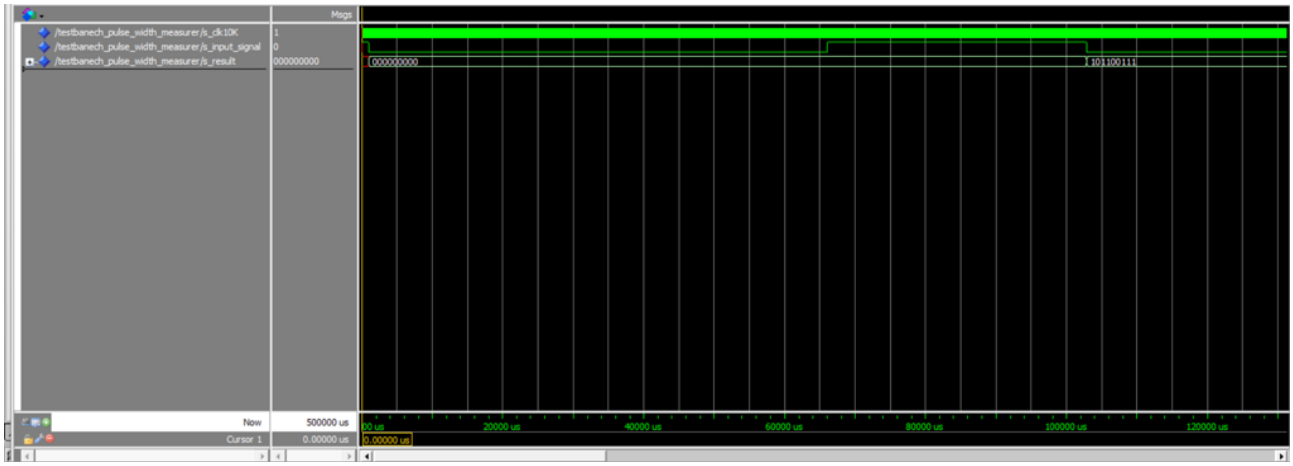
- Pulse_Width_Measurer : ce bloc va permettre la recopie du signal du diviseur de 10 kHz suivant les fronts montants du signal PWM

```vhdl
library IEEE;
use IEEE.STD_LOGIC_ARITH.ALL;
use ieee.numeric_std.all;
use ieee.std_logic_unsigned.all;
use ieee.numeric_std.all;
use ieee.std_logic_1164.all;

entity Pulse_Width_Measurer is
    Port ( input_signal : in STD_LOGIC;
           clk10K : in STD_LOGIC;
           result : out STD_LOGIC_VECTOR(8 downto 0));
end Pulse_Width_Measurer;

architecture Behavioral of Pulse_Width_Measurer is
    signal pulse_width : natural := 0; --compteur de temps haut de signal PWM_in = input_signal
  --   signal prev_clk : STD_LOGIC := '0';

begin
    process (clk10K)
    begin
        if (clk10K'event and clk10K='1') then
            if input_signal = '1'  then
                pulse_width <= pulse_width + 1;--compteur
            else
              if  pulse_width/=0 then
                result <= std_logic_vector(to_unsigned(pulse_width-10, 9));--mettre la valeur de compteur en resu
                pulse_width <= pulse_width-pulse_width;-- on remetle compteur à 0
             else
                pulse_width <= pulse_width;--pas de changement
            end if;
         --   prev_clk <= clk10K;
        end if;
        end if;
    end process;

end Behavioral;
```

Code du Compteur

```vhdl
        );
   end component ;
    signal   s_clk10K :  STD_LOGIC :='0';--on crée le signal qui est relié avec l'entrée clk10K
    signal   s_input_signal    :  STD_LOGIC:='0';--on crée le signal qui est relié avec l'entrée s_input_signal
    signal   s_result :  STD_LOGIC_VECTOR(8 downto 0) :=(others=>'0'); --on crée le signal qui est relié avec la
   begin
   Pulse_Width_Measurer1 :Pulse_Width_Measurer
   port map (
   clk10K => s_clk10K ,
   input_signal => s_input_signal,
   result => s_result
   );
   tb_clk10K_process :process
   begin
     s_clk10K <= not(s_clk10K);-- mettre les valeur de s_clk_10K pour la Simulation
     wait for 50us;
   end process ;
   tb_input_signal_process:process
   begin
    s_input_signal<= '1' ;-- mettre le temps pour une degré=0 pour la Simulation
    wait for 1ms;
    s_input_signal<= '0' ;
    wait for 65ms;
---------
    s_input_signal<= '1' ;-- mettre le temps pour une degré=359 pour la Simulation
    wait for 36900us;
    s_input_signal<= '0' ;
    wait for 65ms;
----------
    s_input_signal<= '1' ;-- mettre le temps pour une degré pour la Simulation
    wait for 10000us;
    s_input_signal<= '0' ;
    wait for 65ms;
    ----------
   end process;

end arch_testbanech_Pulse_Width_Measurer;
```

Code du testbench



Simulation du compteur

Data_Acquisition : Ce bloc représente le traitement des données pour aboutir aux signaux de sorties attendus suivant le mode fonctionnement.

```
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  entity Data_Acquisition is
7      Port ( clk : in STD_LOGIC; -- Horloge 1 Hz
8             continu : in STD_LOGIC; -- Mode Continu (1) ou Monocoup (0)
9             start_stop : in STD_LOGIC; -- Démarre une acquisition (1) ou remet à 0 data_valid (0)
10            resultat : in STD_LOGIC_VECTOR(8 downto 0); -- Valeur en binaire sur 9 bits
11            data_valid : out STD_LOGIC; -- Indicateur de mesure valide
12            data_compas : out STD_LOGIC_VECTOR(8 downto 0) -- Résultat de l'acquisition
13           );
14 end Data_Acquisition;
15
16 architecture Behavioral of Data_Acquisition is
17     signal data_compas_internal : STD_LOGIC_VECTOR(8 downto 0);
18     signal data_valid_internal : STD_LOGIC := '0';
19
20 begin
21     process (clk)
22     begin
23         if rising_edge(clk) then
24             if continu = '1' then
25                 data_compas_internal <= resultat;
26             else
27                 if start_stop = '1' then
28                     data_compas_internal <= resultat;
29                 else
30                     data_compas_internal <= "000000000";
31                 end if;
32             end if;
33
34             if start_stop = '1' then
35                 data_valid_internal <= '1';
36             else
37                 data_valid_internal <= '0';
38             end if;
39         end if;
40     end process;
41
42     data_valid <= data_valid_internal;
43     data_compas <= data_compas_internal-10;
44 end Behavioral;
45
```

Code du bloc d'acquisition

Conclusion :

Pour conclure, l'ensemble de nos blocs fonctionnent distinctement et se simulent. Néanmoins nos signaux de sorties ne se simulent pas ensemble afin de représenter le fonctionnement du compas.