UNIVERSITÉ TOULOUSE III PAUL SABATIER

Université de Toulouse

Rapport sur la fonction simple du BE_VHDL

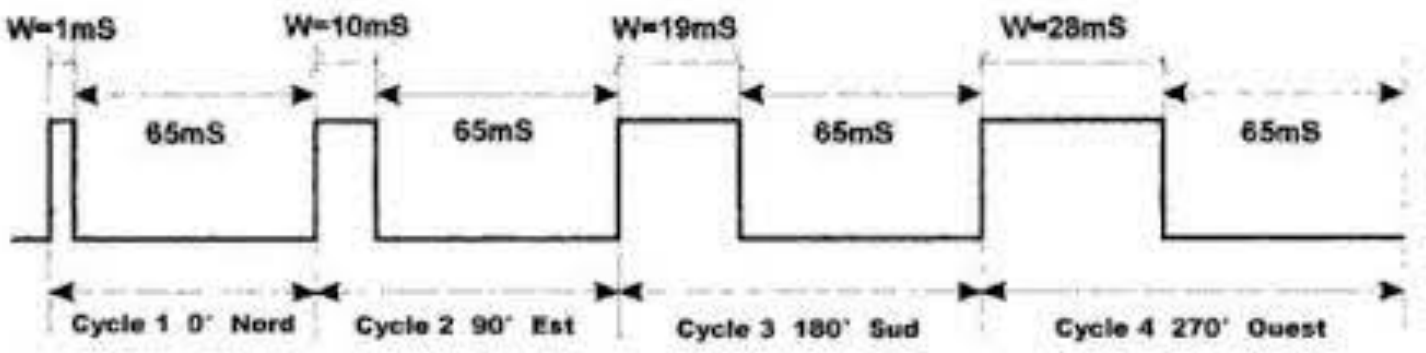
Gestion Compas

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 us/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

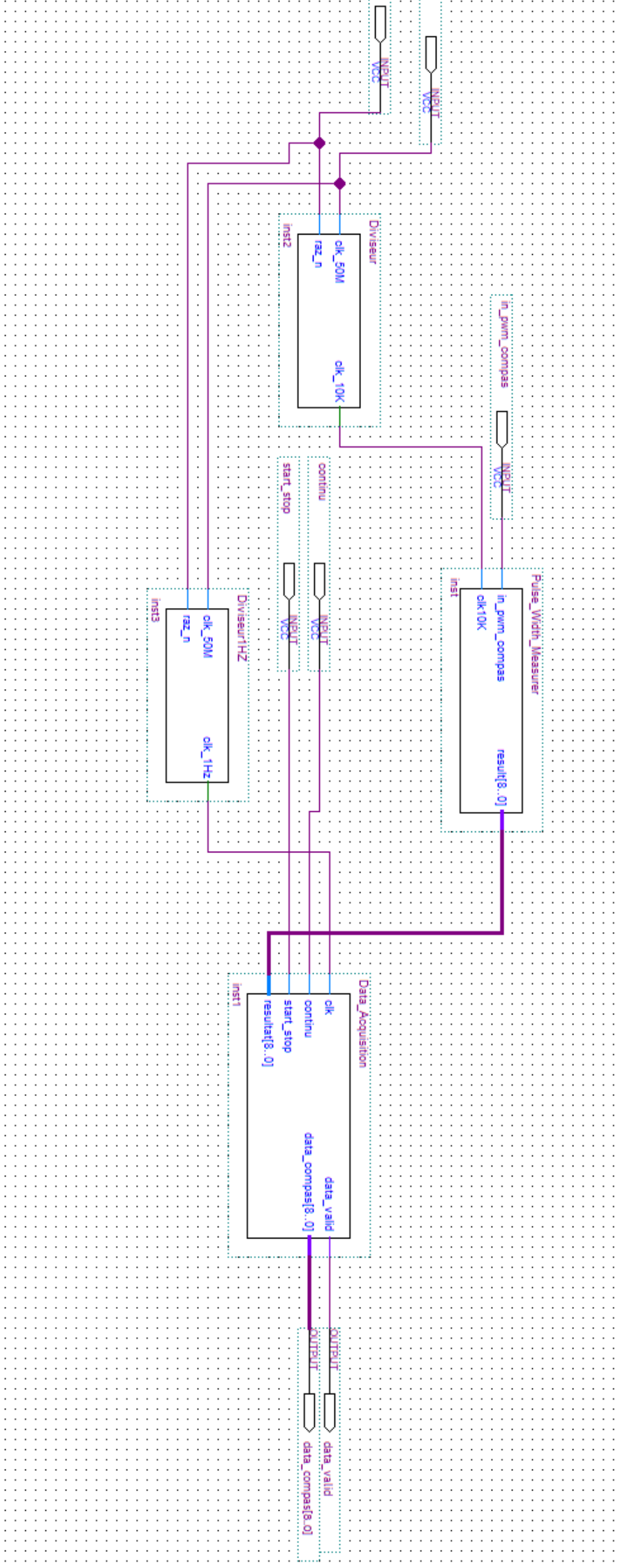
- in_pwm_compas : signal PWM d'entrée de la boussole qui va donc varier entre 1 ms et 36,9 ms
- start_stop : elle permet de démarrer une acquisition lorsque le mode fixé est en monocoup et que c'est = 1.
- Continu : permet de fixer le mode de fonctionnement du compas, soit en mode en continu lorsque 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 :

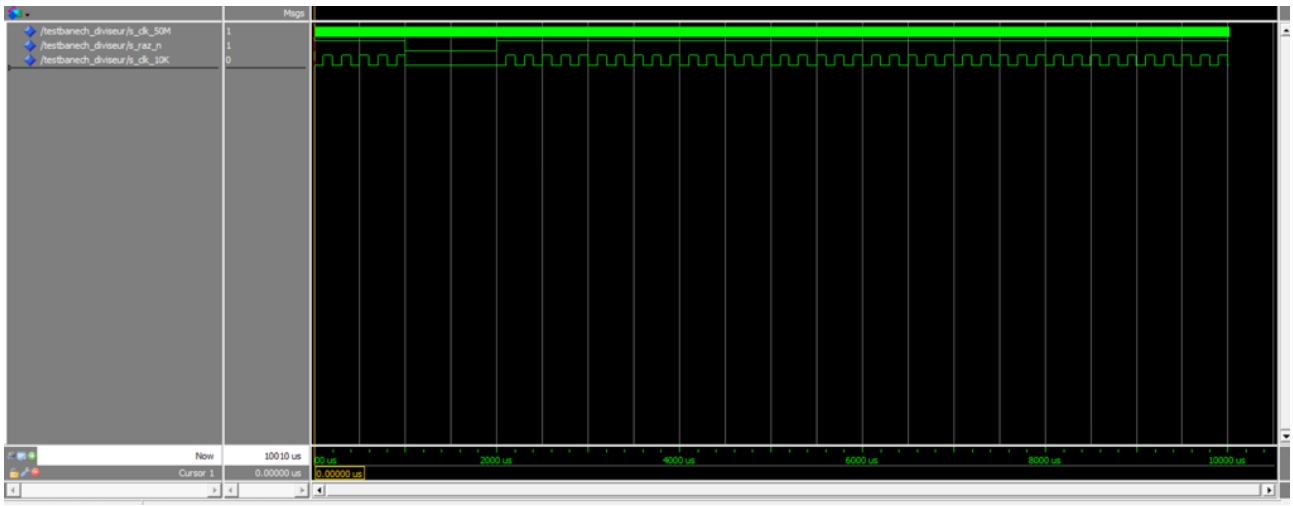

VCC
INPUT
VCC
INPUT
Diviseur
inst2
clk_50M
raz_n
clk_10K
in_pwm_compas
VCC
INPUT
start_stop
VCC
INPUT
continu
VCC
INPUT
Diviseur1HZ
inst3
clk_50M
raz_n
clk_1Hz
Pulse_Width_Measurer
inst
clk10K
in_pwm_compas
result[8..0]
Data_Acquisition
inst1
clk
continu
start_stop
resultat[8..0]
data_valid
data_compas[8..0]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

II - Code et simulation

- 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 frequence
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

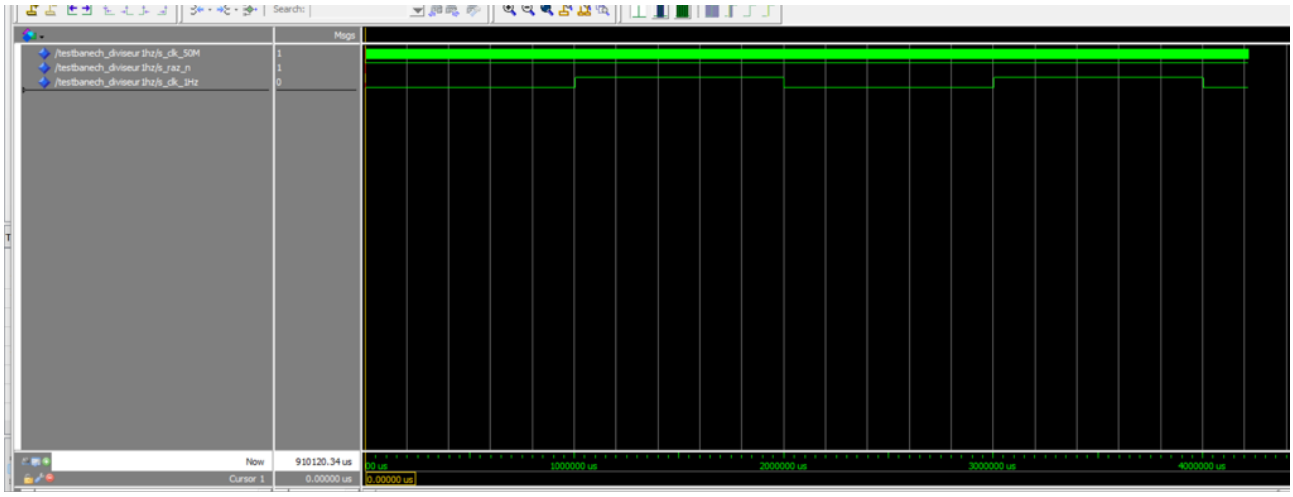


Simulation du diviseur 10 khz

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

```vhdl
--Diviseur de frequence
LIBRARY ieee;
USE ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;


entity Diviseur1HZ is
    Port ( clk_50M : in STD_LOGIC;
           raz_n : in STD_LOGIC;
           clk_1Hz : inout STD_LOGIC
           );
end Diviseur1HZ;

architecture Behavioral of Diviseur1HZ is
begin
process (clk_50M, raz_n)
         variable count : integer range 0 to 49999999 ; -- Compteur pour diviser le signal

    begin
        if raz_n = '0' then
            count := 0; -- Réinitialisation du compteur lorsque raz_n est actif
                clk_1Hz <= '0';
        else if (clk_50M'event and clk_50M='1') then
            count := count + 1;
            if count = 49999999 then
                count := 0;
                clk_1Hz <= not clk_1Hz; -- Inversion de la sortie pour générer une fréquence de 1Hz
            else
                count := count ;

            end if;
        end if;
         end if;

    end process;
end Behavioral;

```

Code du Diviseur 1HZ

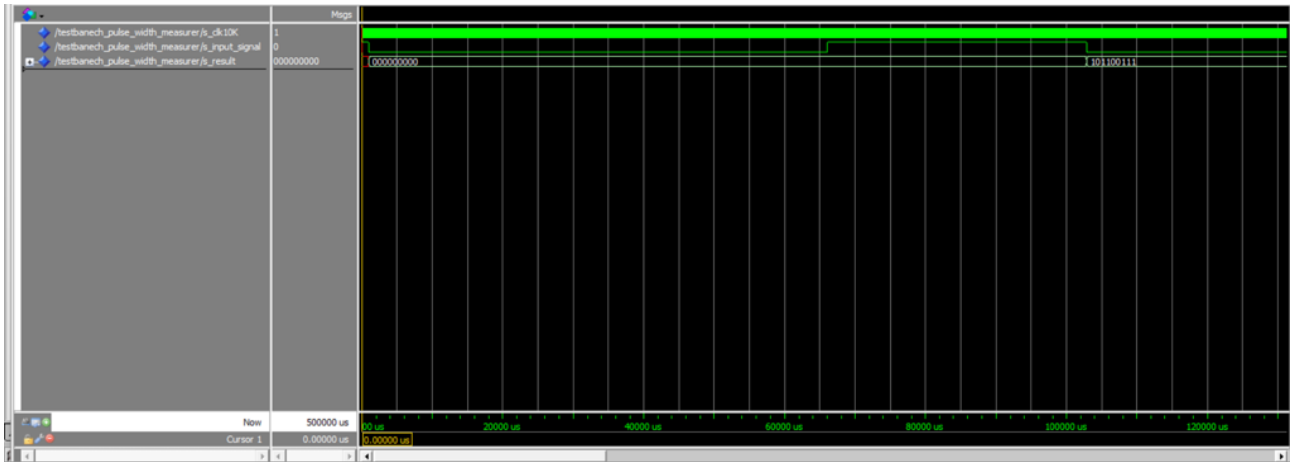


Simulation du diviseur 1hz

- 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 ( in_pwm_compas : 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 = in_pwm_compas
 --   signal prev_clk : STD_LOGIC := '0';

begin
    process (clk10K)
    begin
        if (clk10K'event and clk10K='1') then
            if in_pwm_compas = '1'  then
                pulse_width <= pulse_width + 1;--compteur
            else
                 if     pulse_width/=0 then
                --    pulse_width<=pulse_width;
                     result <= std_logic_vector(to_unsigned(pulse_width, 9));--mettre la valeur de compteur en resultat
                     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;
```

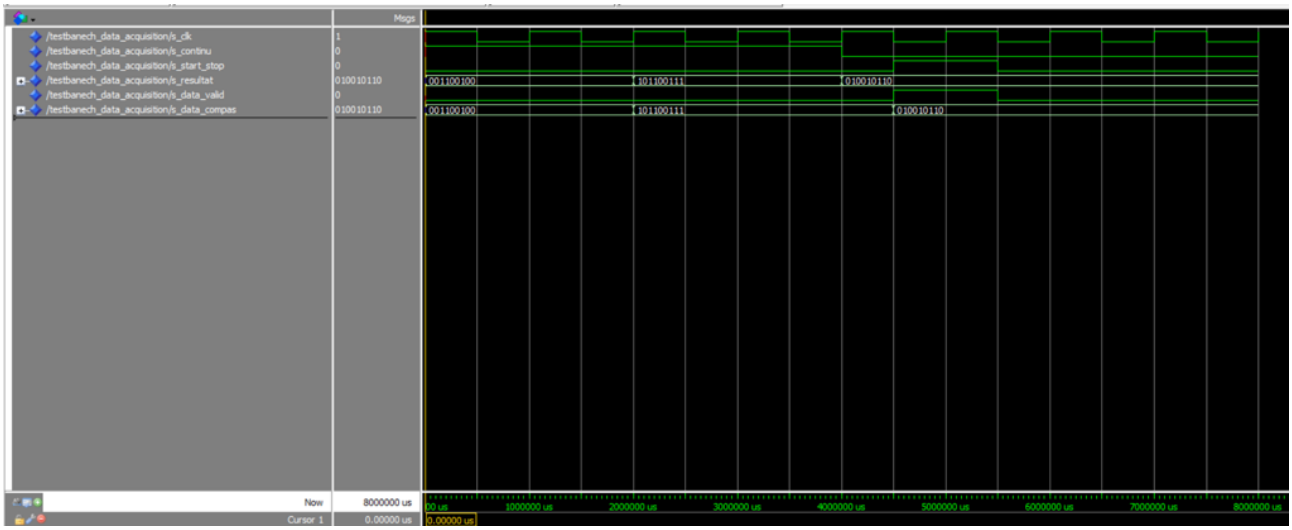
Code du Compteur



Simulation du compteur

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

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


entity Data_Acquisition is
    Port ( clk : in STD_LOGIC; -- Horloge 1 Hz
           continu : in STD_LOGIC; -- Mode Continu (1) ou Monocoup (0)
           start_stop : in STD_LOGIC; -- D√©marre une acquisition (1) ou remet √† 0 data_valid (0)
           resultat : in STD_LOGIC_VECTOR(8 downto 0); -- Valeur de degr√© en binaire sur 9 bits
           data_valid : out STD_LOGIC; -- Indicateur de mesure valide
           data_compas : out STD_LOGIC_VECTOR(8 downto 0); -- R√©sultat de l'acquisition
              raz_n : in STD_LOGIC
         );
end Data_Acquisition;

architecture Behavioral of Data_Acquisition is
    signal data_compas_internal : STD_LOGIC_VECTOR(8 downto 0);
    signal data_valid_internal : STD_LOGIC := '0';

begin
    process (clk,start_stop,raz_n)
    begin

          if(raz_n='0') then --mettre ‡ le circuit ‡ zero
          data_valid_internal <= '0';
          data_compas_internal <= "000000000";
        else
            if (clk'event and clk='1')  then
           if continu = '1' then-- Mode continu
               data_compas_internal <= resultat;

               else
                  data_compas_internal <= "000000000";
            end if;

             end if;
        if (start_stop = '1' and raz_n='1') then-- en mode monocoup
               data_valid_internal <= '1';
                    data_compas_internal <= resultat;
           else
               data_valid_internal <= '0';
        end if;
        end if;
    end process;

     data_valid <= data_valid_internal;
     data_compas <= data_compas_internal;
end Behavioral;
```



Simulation du bloc d’acquisition

Conclusion

Pour conclure, notre module compas est fonctionnelle en simulation. On a par ailleurs aussi pu visualiser la durée du rapport cyclique du signal PWM via les leds de la carte DE2 qui pourront être convertis par la suite en degrés dans le code C.