



UNIVERSITÉ
TOULOUSE III
PAUL SABATIER



Université
de Toulouse

Rapport sur la fonction simple du BE_VHDL

Gestion Compas

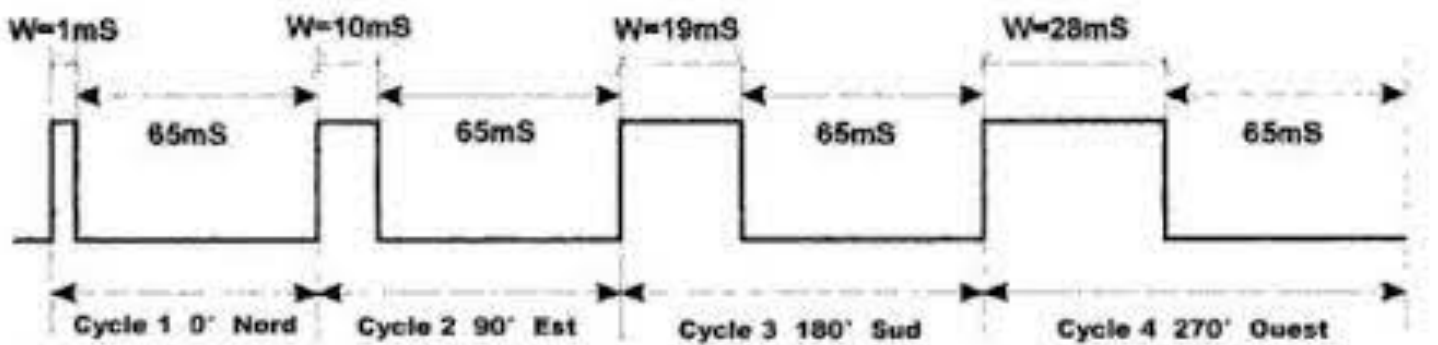
Rédigé par :

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 $359,9^\circ$, 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 0°) et de 102 ms maximum (pour un angle de $359,9^\circ$).

I - Analyse fonctionnelle

Le module compas présente les entrées et sorties suivantes :

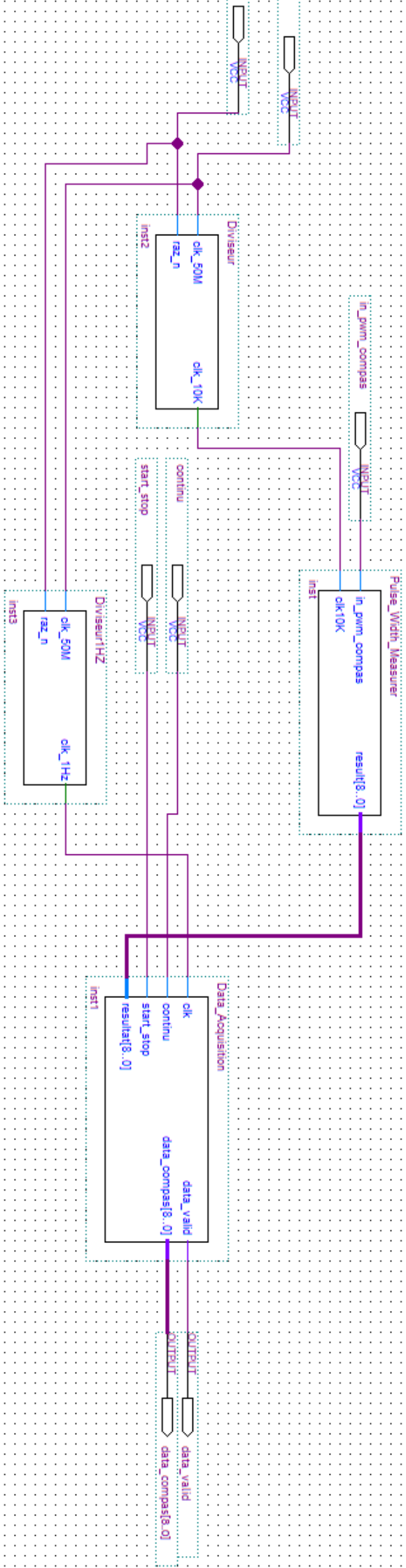
Entrées

- 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 :



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.

```
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 Diviseur is
8  Port ( clk_50M : in STD_LOGIC;
9        raz_n : in STD_LOGIC;
10       clk_10K : inout STD_LOGIC
11       );
12  end Diviseur;
13
14  architecture Behavioral of Diviseur is
15
16  begin
17  process (clk_50M, raz_n)
18      variable count : integer range 0 to 4999 ; -- Compteur pour diviser le signal
19
20  begin
21      if raz_n = '0' then
22          count := 0; -- Réinitialisation du compteur lorsque raz_n est actif
23          clk_10K <= '0';
24      else if (clk_50M'event and clk_50M='1') then
25          count := count + 1;
26          if count = 4999 then
27              count := 0;
28              clk_10K <= not clk_10K; -- Inversion de la sortie pour générer une fréquence de 10 KHz
29          else
30              count := count ;
31              -- clk_10K <= '1';
32          end if;
33      end if;
34  end if;
35
36  end process;
37  end Behavioral;
```

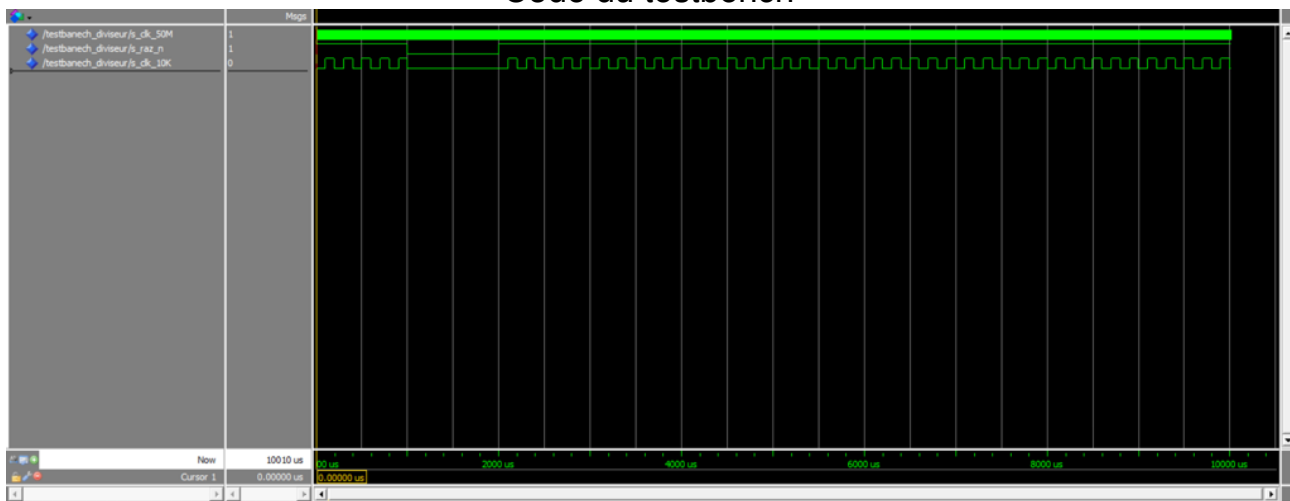
Code du diviseur à 10 khz

```

13 architecture arch_testbanch_Diviseur of testbanch_Diviseur is
14
15     component Diviseur is
16     Port (
17         clk_50M : in STD_LOGIC := '0'; --on définit l'entrée clk_50M
18         raz_n : in STD_LOGIC := '0'; --on définit l'entrée raz_n
19         clk_10K : inout STD_LOGIC := '0'; --on définit l'entrée clk_10K
20     );
21     end component;
22     signal s_clk_50M : STD_LOGIC := '0'; --on crée le signal qui est relié avec l'entrée clk_50M
23     signal s_raz_n : STD_LOGIC := '0'; --on crée le signal qui est relié avec l'entrée s_raz_n
24     signal s_clk_10K : STD_LOGIC := '1'; --on crée le signal qui est relié avec la sortie clk_10k
25     begin
26         Diviseur1 : Diviseur
27         port map (
28             clk_50M => s_clk_50M ,
29             raz_n => s_raz_n,
30             clk_10K => s_clk_10K
31         );
32         tb_clk_50M_process : process
33         begin
34             s_clk_50M <= not(s_clk_50M); -- mettre les valeur de s_clk_50M pour la Simulation
35             wait for 10ns;
36         end process;
37         tb_reset_process : process
38         begin
39             s_raz_n <= '1' ; -- mettre les valeur de s_raz_n pour la Simulation
40             wait for 1ms;
41             s_raz_n <= '0' ;
42             wait for 1ms;
43             s_raz_n <= '1' ;
44             wait;
45         end process;
46     end arch_testbanch_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              end if;
31          end if;
32      end if;
33  end process;
34  end Behavioral;
35
36
37

```

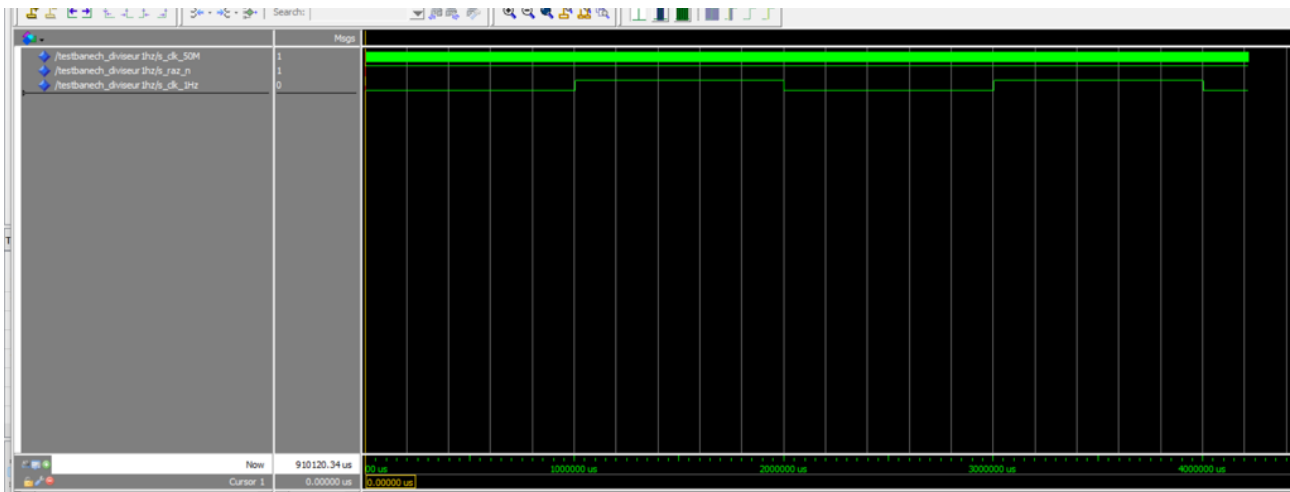
Code du diviseur à 1 hz

```

13  architecture arch_testbanech_Diviseur1HZ of testbanech_Diviseur1HZ is
14
15  component Diviseur1HZ is
16  Port (
17      clk_50M : in STD_LOGIC := '0'; --on définit l'entrée clk_50M
18      raz_n : in STD_LOGIC := '0'; --on définit l'entrée raz_n
19      clk_1Hz : inout STD_LOGIC := '0'; --on définit l'entrée clk_1Hz
20  );
21  end component;
22  signal s_clk_50M : STD_LOGIC := '0'; --on crée le signal qui est relié avec l'entrée clk_50M
23  signal s_raz_n : STD_LOGIC := '0'; --on crée le signal qui est relié avec l'entrée s_raz_n
24  signal s_clk_1Hz : STD_LOGIC := '1'; --on crée le signal qui est relié avec la sortie clk_1Hz
25  begin
26  Diviseur2 : Diviseur1HZ
27  port map (
28      clk_50M => s_clk_50M ,
29      raz_n => s_raz_n,
30      clk_1Hz => s_clk_1Hz
31  );
32  tb_clk_50M_process : process
33  begin
34      s_clk_50M <= not(s_clk_50M); -- mettre les valeur de s_clk_50M pour la simulation
35      wait for 10ns;
36  end process;
37  tb_reset_process : process
38  begin
39      s_raz_n <= '1' ; -- mettre les valeur de s_raz_n pour la simulation
40      wait for 1ms;
41      s_raz_n <= '0' ;
42      wait for 1ms;
43      s_raz_n <= '1' ;
44      wait;
45  end process;
46
47  end arch_testbanech_Diviseur1HZ;
48

```

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

```

1  library IEEE;
2  use IEEE.STD_LOGIC_ARITH.ALL;
3  use IEEE.numeric_std.all;
4  use IEEE.std_logic_unsigned.all;
5  use IEEE.numeric_std.all;
6  use IEEE.std_logic_1164.all;
7
8  entity Pulse_Width_Measurer is
9  port ( input_signal : in STD_LOGIC;
10       clk10K : in STD_LOGIC;
11       result : out STD_LOGIC_VECTOR(8 downto 0));
12  end Pulse_Width_Measurer;
13
14  architecture Behavioral of Pulse_Width_Measurer is
15  signal pulse_width : natural := 0; --compteur de temps haut de signal PWM_in = input_signal
16  -- signal prev_clk : STD_LOGIC := '0';
17
18  begin
19  process (clk10K)
20  begin
21  if (clk10K'event and clk10K='1') then
22  if input_signal = '1' then
23  pulse_width <= pulse_width + 1; --compteur
24  else
25  if pulse_width/=0 then
26  result <= std_logic_vector(to_unsigned(pulse_width-10, 9)); --mettre la valeur de compteur en resu
27  pulse_width <= pulse_width-pulse_width; -- on remetle compteur à 0
28  else
29  pulse_width <= pulse_width; --pas de changement
30  end if;
31  -- prev_clk <= clk10K;
32  end if;
33  end if;
34  end process;
35
36  end Behavioral;
37

```

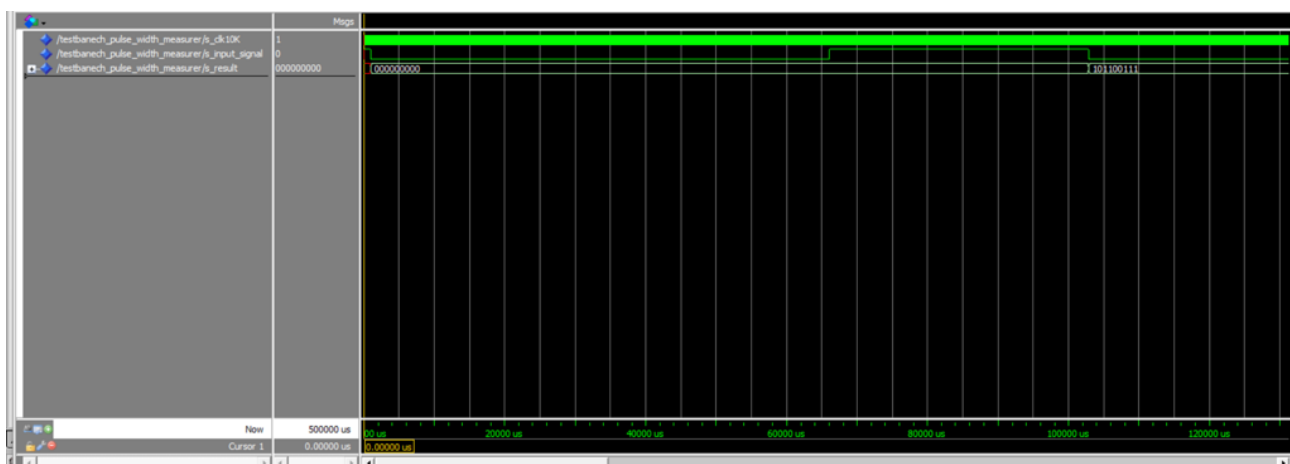
Code du Compteur


```

20
21 end component ;
22 signal s_clk10K : STD_LOGIC := '0'; -- on crée le signal qui est relié avec l'entrée clk10K
23 signal s_input_signal : STD_LOGIC := '0'; -- on crée le signal qui est relié avec l'entrée s_input_signal
24 signal s_result : STD_LOGIC_VECTOR(8 downto 0) := (others => '0'); -- on crée le signal qui est relié avec la
25 begin
26 Pulse_Width_Measurer1 : Pulse_Width_Measurer
27 port map (
28 clk10K => s_clk10K ,
29 input_signal => s_input_signal,
30 result => s_result
31 );
32 tb_clk10K_process : process
33 begin
34 s_clk10K <= not(s_clk10K); -- mettre les valeur de s_clk10K pour la Simulation
35 wait for 50us;
36 end process ;
37 tb_input_signal_process : process
38 begin
39 s_input_signal <= '1' ; -- mettre le temps pour une degré=0 pour la Simulation
40 wait for 1ms;
41 s_input_signal <= '0' ;
42 wait for 65ms;
43
44 s_input_signal <= '1' ; -- mettre le temps pour une degré=359 pour la Simulation
45 wait for 36900us;
46 s_input_signal <= '0' ;
47 wait for 65ms;
48
49 s_input_signal <= '1' ; -- mettre le temps pour une degré pour la Simulation
50 wait for 10000us;
51 s_input_signal <= '0' ;
52 wait for 65ms;
53
54 end process;
55
56 end arch_testbench_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.

```

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 de degré 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; -- on récupère 1 valeur de degré
26
27       end if;
28     end if;
29
30     if start_stop = '1' then
31       data_valid_internal <= '1'; -- une acquisition est validé
32       data_compas_internal <= resultat; -- on récupère 1 valeur de degré
33     else
34       data_valid_internal <= '0';
35     end if;
36   end process;
37
38   data_valid <= data_valid_internal;
39   data_compas <= data_compas_internal;
40 end Behavioral;
41

```

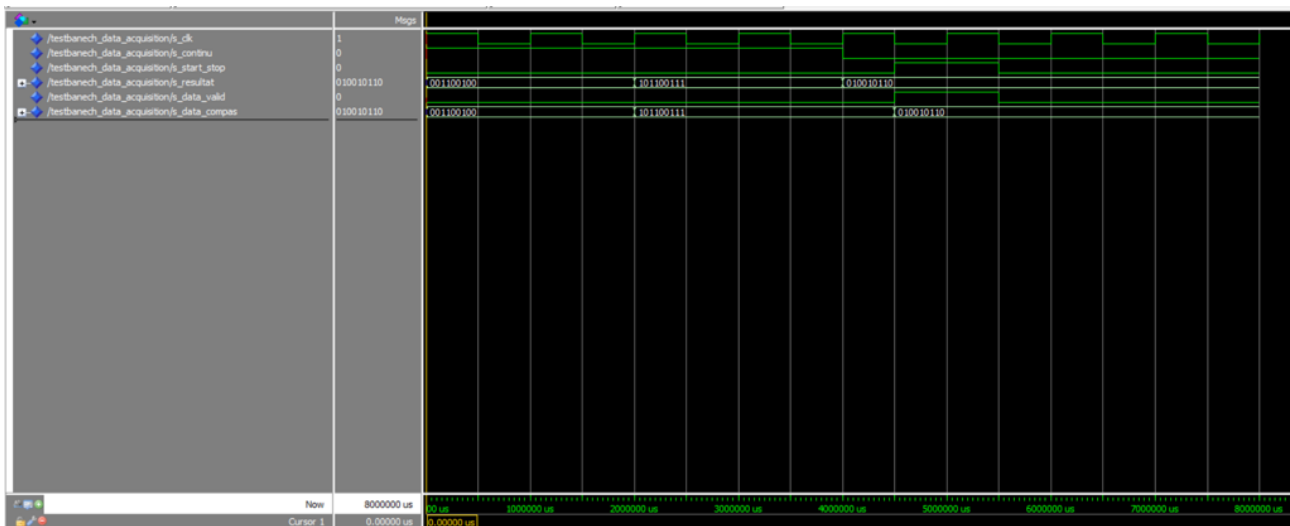
Code du bloc d'acquisition

```

44
45 tb_clk_process : process
46 begin
47   s_clk <= not(s_clk); -- mettre les valeur de s_clk pour la Simulation
48   wait for 500ms;
49 end process ;
50 tb_s_continu_process : process
51 begin
52   s_continu <= '1'; -- mettre le temps pour pour la Simulation
53   wait for 4000ms;
54   s_continu <= '0';
55   wait;
56 end process;
57 -----
58 tb_s_start_stop_process : process
59 begin
60   s_start_stop <= '0'; -- mettre le temps pour la Simulation
61   wait for 4500ms;
62   s_start_stop <= '1';
63   wait for 1000ms;
64   s_start_stop <= '0';
65   wait;
66 end process;
67 -----
68 tb_s_resultat_process : process
69 begin
70   s_resultat <= "001100100"; -- mettre les valeur de pour une degré=100 pour la Simulation
71   wait for 2000ms;
72   s_resultat <= "101100111"; -- mettre les valeur de pour une degré=359 pour la Simulation
73   wait for 2000ms;
74   s_resultat <= "010010110"; -- mettre les valeur de pour une degré=150 pour la Simulation
75   wait;
76 end process ;
77
78
79 end arch_testbanech_Data_Acquisition;

```

Code du testbench



Simulation du bloc d'acquisition

Conclusion

Pour conclure, notre module compas est fonctionnelle en simulation. On doit donc par la suite le valider via la carte afin de l'intégrer à notre SOPC.