

### Objectifs :

- Comprendre la commande trapèze d'un moteur brushless
- Réaliser les 4 blocs VHDL
- Tester la communication entre le MAX et le LPC1768

## 1 Mise à disposition de la carte ALTERA MAX7064S SAE Roue de Vae

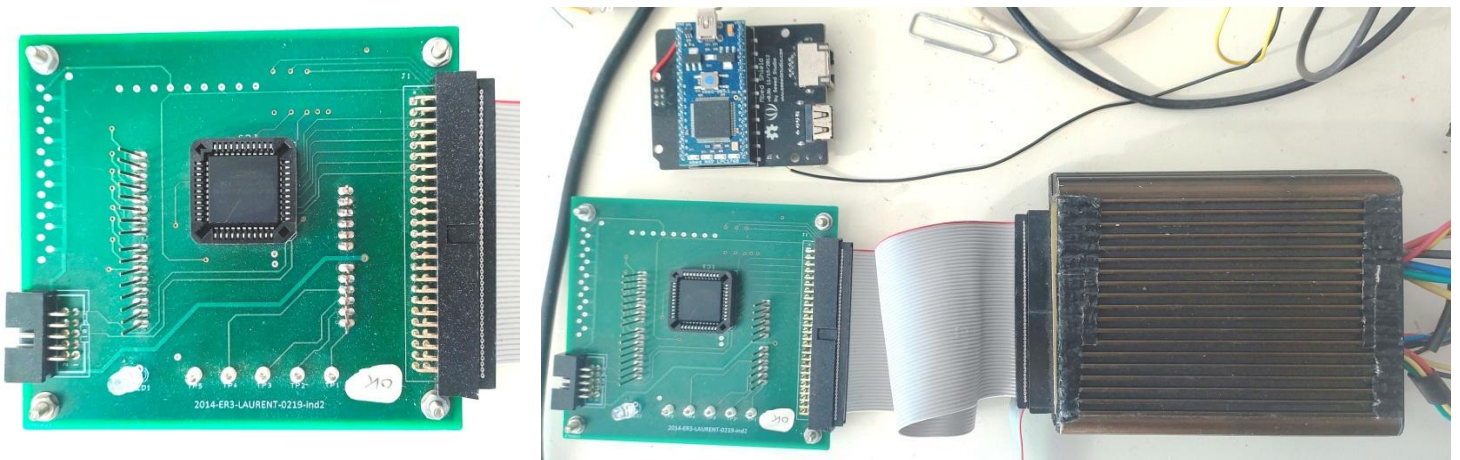

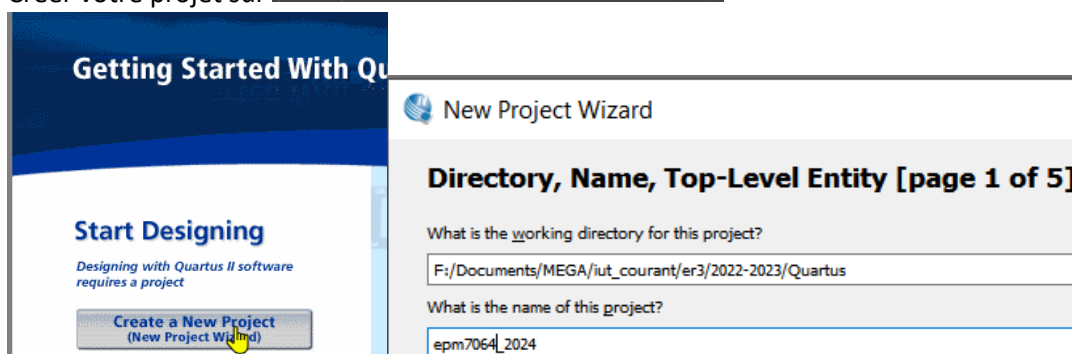
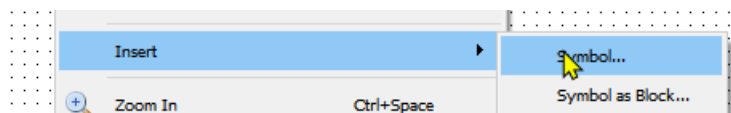
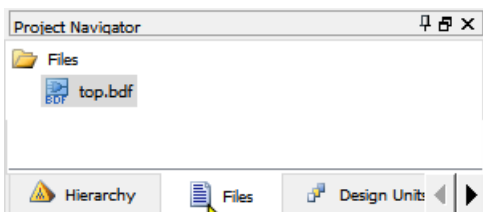
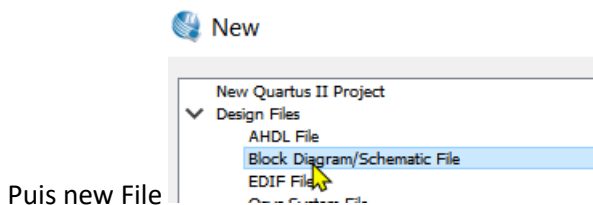
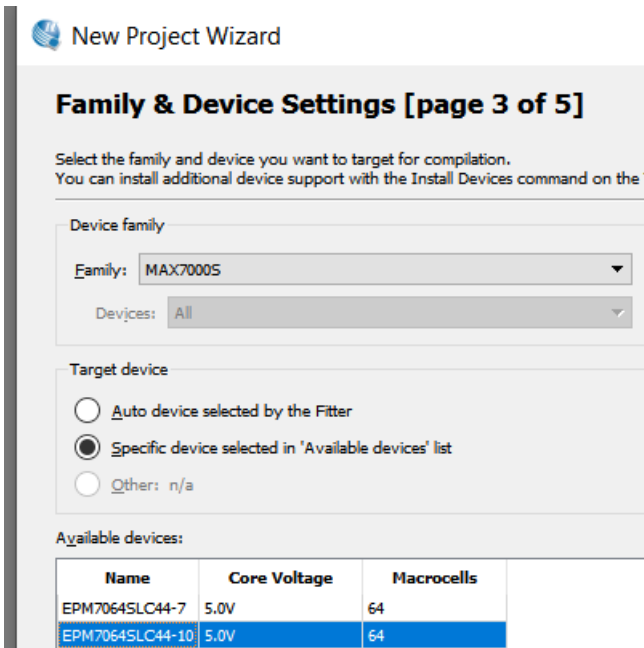


Figure 1: carte Max7000S utilisée en SAE roue( gauche) et ensemble des cartes de la SAE Roue VAE (droite)

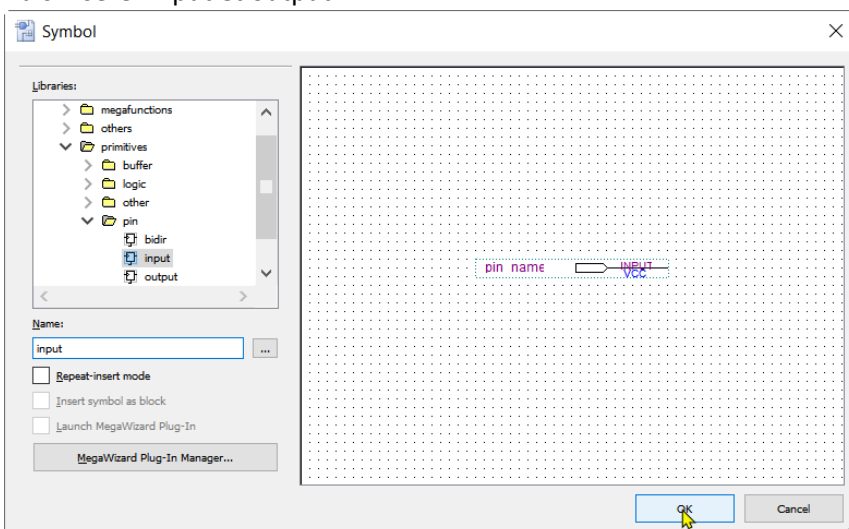
### 1.1 Création du projet

Créer votre projet sur  Quartus II 13.0sp1 (64-bit)

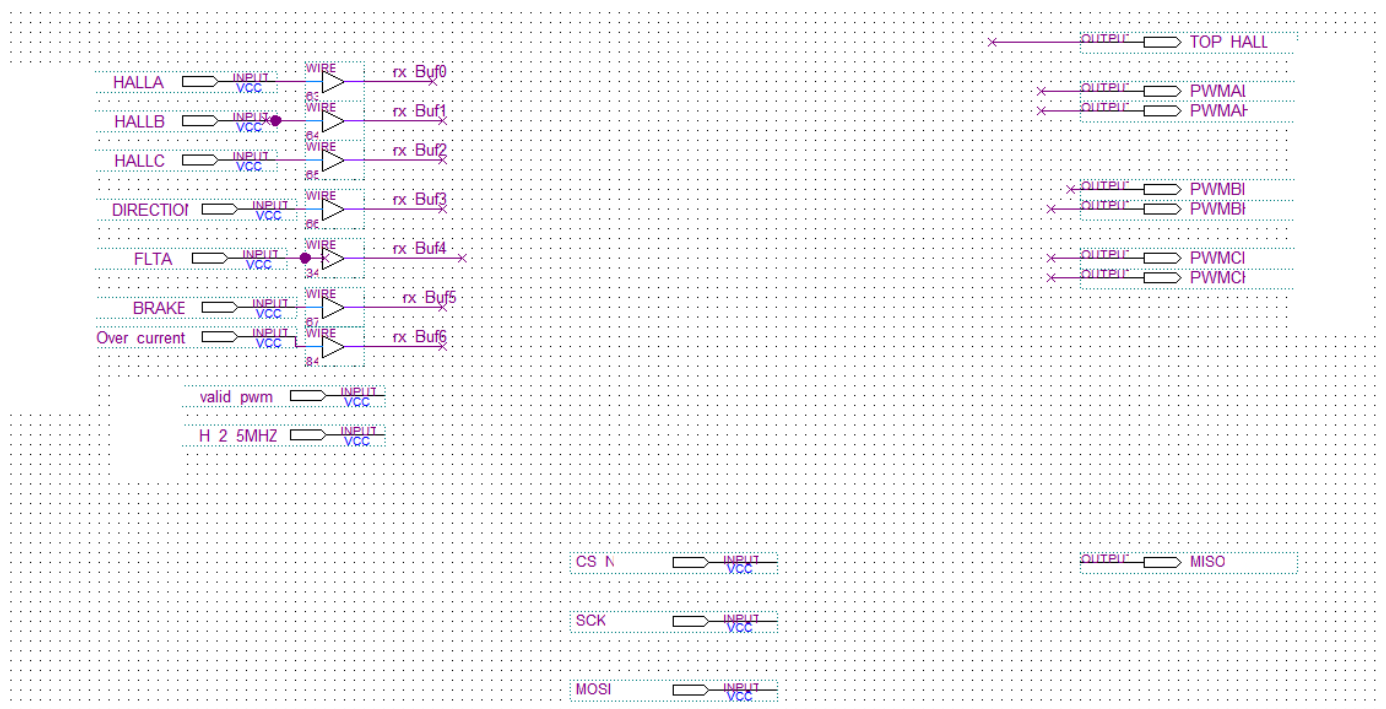




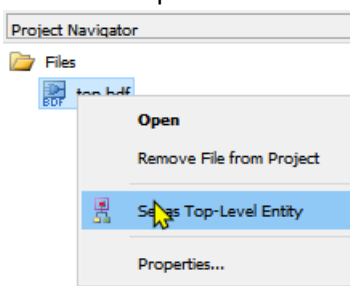
Puis insérer input et output



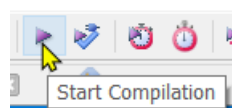
Voici le résultat



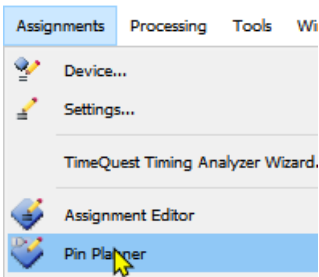
Mettre en top level



puis compiler



après compilation lancer le pin planner pour mettre les numéros de broches



File Edit View Processing Tools Window Help

Report not available

Groups Report

Tasks

- Run Analy
- Early Pin F
- Early
- Run I

Top View

MAX7000S

EPM7064SLC44-10

Node Name	Direction	Location	Fitter Location	Reserved
HALLC	Input	PIN_41	PIN_41	
HALLA	Input	PIN_39	PIN_39	
DIRECTION	Input	PIN_36	PIN_36	
Over_current	Input	PIN_37	PIN_37	
valid_pwm	Input	PIN_33	PIN_33	
H_2_5MHZ	Input	PIN_43	PIN_43	
HALLB	Input	PIN_40	PIN_40	
PWMCL	Output	PIN_9	PIN_9	
PWMCH	Output	PIN_8	PIN_8	
PWMBL	Output	PIN_12	PIN_12	
PWMBH	Output	PIN_11	PIN_11	
PWMAH	Output	PIN_14	PIN_14	
BRAKE	Input	PIN_5	PIN_5	
PWMAL	Output	PIN_16	PIN_16	
FLTA	Input	PIN_6	PIN_6	
TOP_HALL	Output	PIN_31	PIN_31	
CS_N	Input	PIN_21	PIN_17	
MISO	Output	PIN_25	PIN_4	
MOSI	Input	PIN_26	PIN_28	
SCK	Input	PIN_24	PIN_29	

Votre projet est prêt à l'emploi !

## 2 Architecture générale du projet SAE Roue de VAE

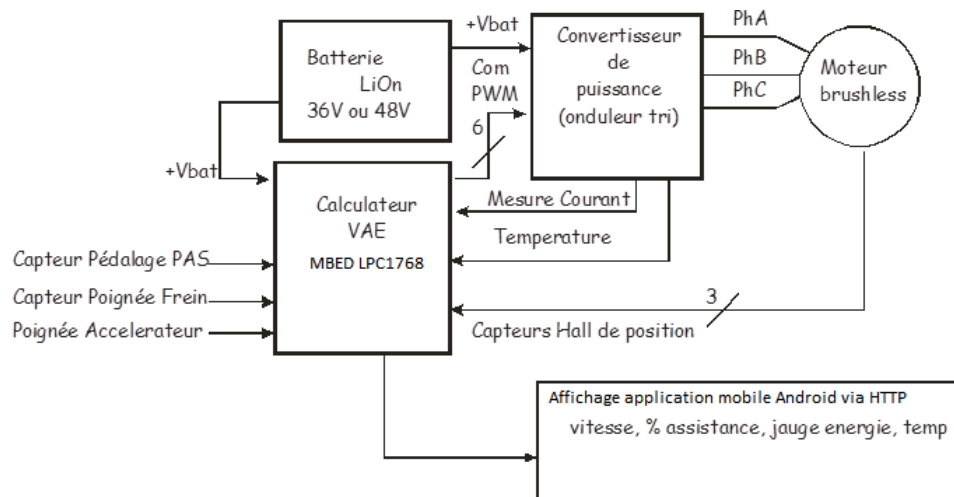


Figure 2: architecture générale du projet SAE roue VAE

**Objectif général du projet :** Il s'agit de pouvoir piloter un moteur roue de technologie Brushless (moteur courant continu sans balais) en commande trapèze et en vitesse variable à partir d'un microcontrôleur MBED et d'un circuit logique programmable MAX7064SLC44-10.

Le microcontrôleur sera chargé de la supervision non-temps réel de la commande du moteur (variation de vitesse, communication avec l'utilisateur, mesure des paramètres physiques...)

Le PLD altera sera chargé de l'autopilotage temps réel en vitesse variable du moteur (la boucle d'autopilotage visible sur la figure2 est en effet critique et présente des contraintes temps réel dures).

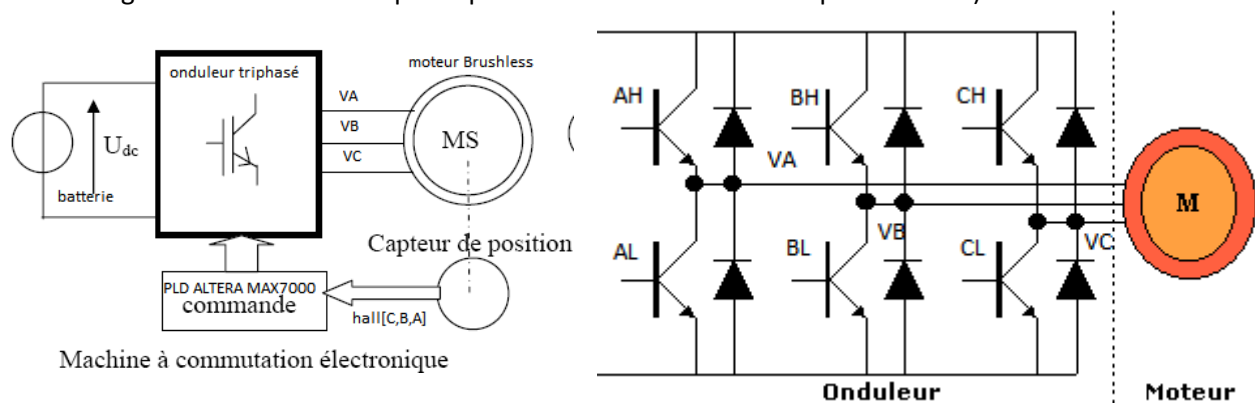


Figure 3: architecture de l'autopilotage (gauche) et détail de l'onduleur (droite)

La Partie puissance est assurée par un onduleur triphasé préfabriqué (boîte noire voir sur la figure 1 ) sur lequel nous avons extrait et dirigé les 6 commandes de transistor des bras d'onduleur vers le PLD ALTERA à savoir AH/AL , BH,BL, CH/CL (c'est le PLD qui assurera donc le commande des 3 bras d'onduleur) à partir de la connaissance de la position instantanée du rotor (inducteur) du moteur.

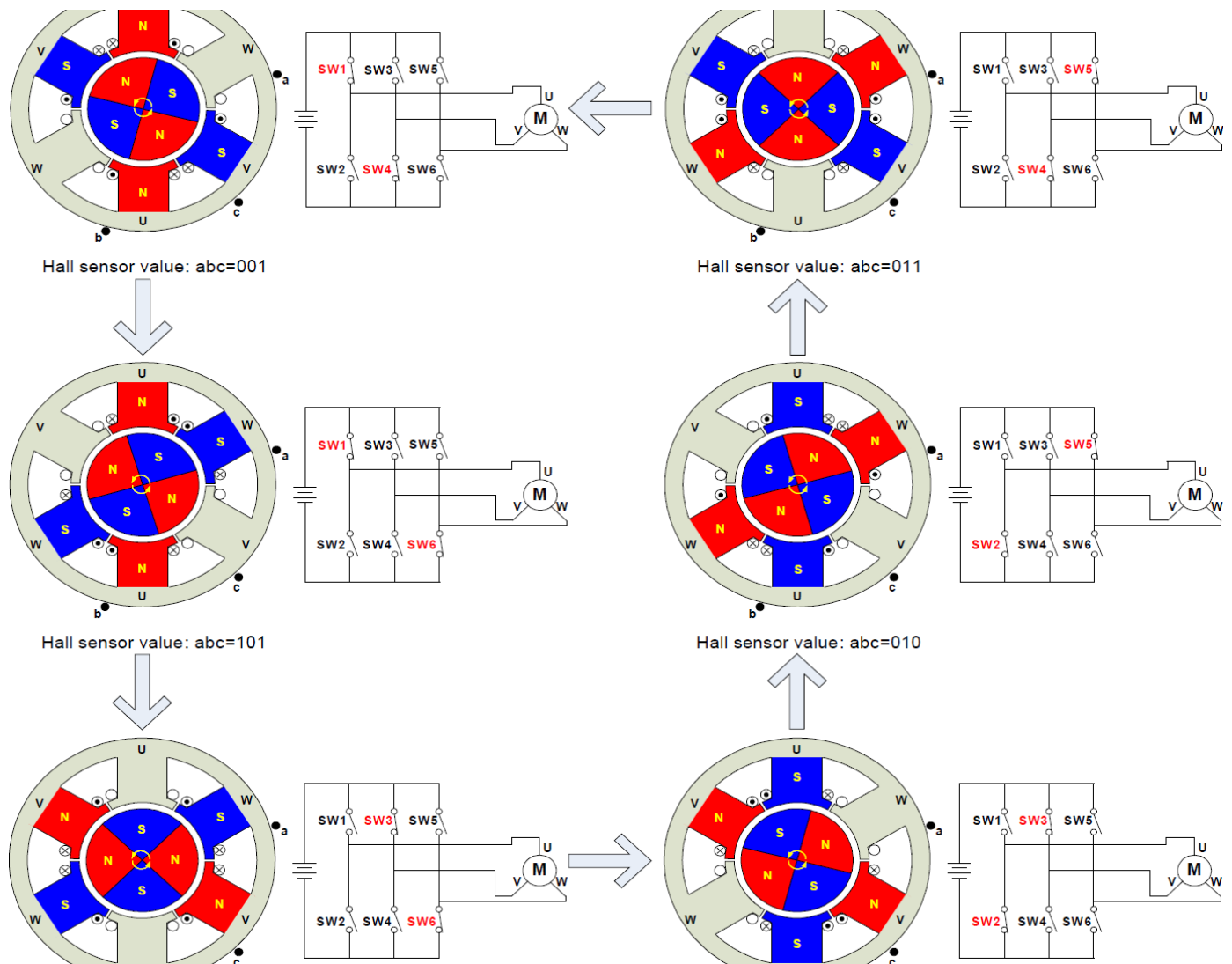
A partir de là il élabore les 3 consignes de tension VA,VB,VC en fonction de la position actuelle du rotor (capteurs Hall[C,B,A] ) et de la consigne de vitesse que désire l'utilisateur (Référence PWM au format 8 bits // fournis par le uC MBED au PLD ALTERA).

### Principe de commande du moteur synchrone brushless en mode trapèze.

Il y a en fait 6 états possible d'alimentation du stator du moteur brushless en fonction des 6 positions différentes mesurables par les 3 capteurs de position du Rotor à effets hall (les positions « 000 » et « 111 » étant impossibles ou équivalentes à une défaillance du moteur).

Figure 4: position angulaire "101" <=> 5 (gauche) et « 100 » <=> 4 à droite

Comme le montrent les figures 4,5 et 6, la commande des 6 transistors de l'onduleur doit être SYNCHRONE » de la position instantanée du rotor (c'est ce que l'on appelle l'autopilotage du brushless). Cette fonction est en fait l'équivalent d'un collecteur numérique (sa version numérique existe dans le moteur à courant continu classique).

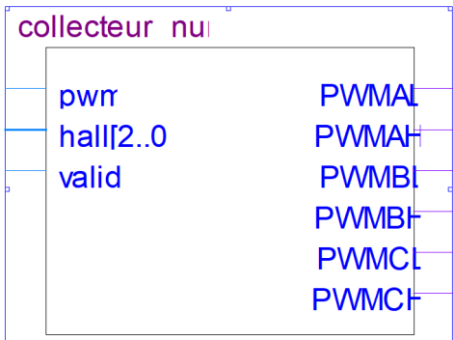


### 3 Création des blocs et tests unitaires (simulation)

Vous allez créer 4 blocs VHDL :

- ✓ Collecteur numérique
- ✓ Générateur PWM
- ✓ Filtre
- ✓ SPI\_Slave

### 3.1 Collecteur numérique du moteur brushless



**Rôle :** Ce bloc permet de commander les 6 transistors de l’onduleur triphasé. Les transistor du Bas PWMxL sont commandés en mode pleine onde et réalisent la fonction de collecteur numérique. Les sorties PWMxH distribue la consigne de PWM reçue à l’entrée en direction des transistors situés en haut de l’onduleur.

**Ressources d’entrées :**

**Hall[2..0]:** Ce bloc reçoit l’information sur la valeur des 3 capteurs hall. Correspondant à HallC, HallB, HallA. En fonction de cette information, ce bloc pilote les 6 transistors du bas PWMxL et distribue la MLI sur l’un des 3 transistors du Haut PWMxH.

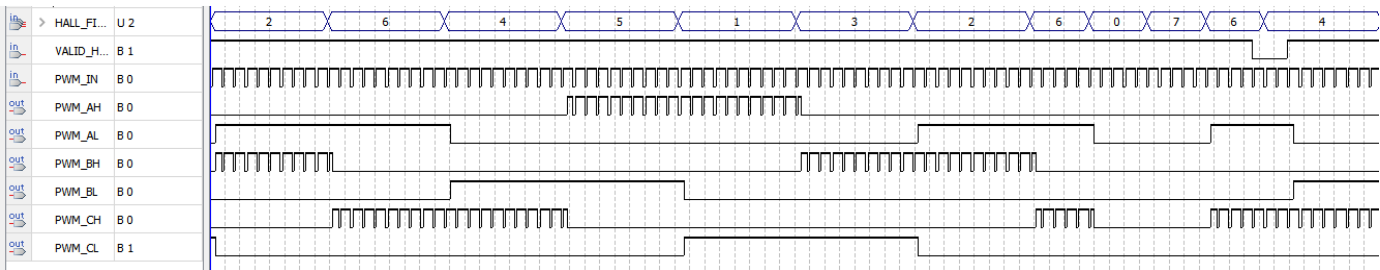
**VALID :** indique à quel moment les capteurs de halls présentent une valeur stable. (Actif à 1)

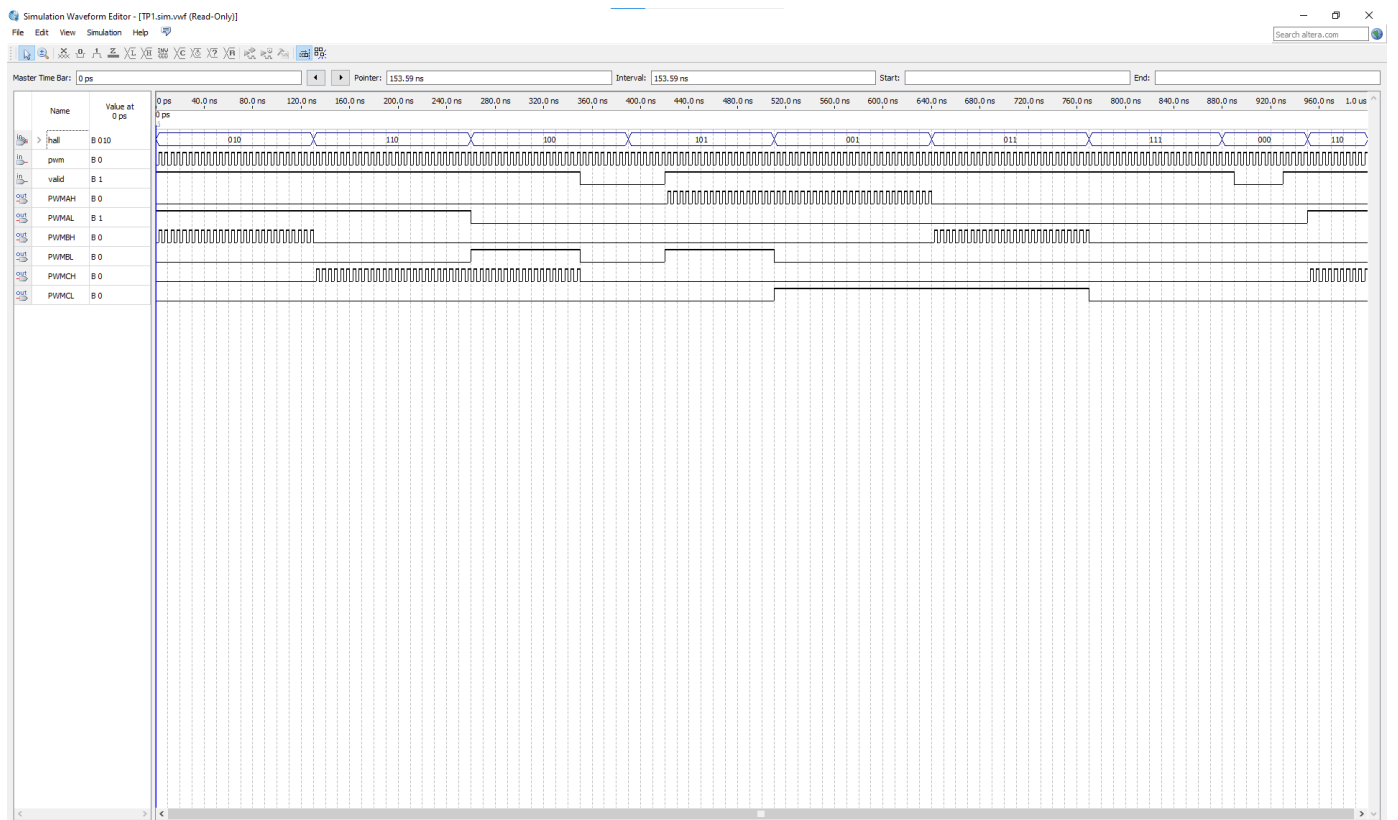
**PWM :** entrée de PWM modulée en largeur d’impulsion suivant la consigne de pédalage ou de gaz fréquence 10kHz.

**Ressources de sortie :**

**PWMxL et PWMxH :** commandes logique des transistors de l’onduleur.

Simulation de fonctionnement attendu pour la marche avant





```

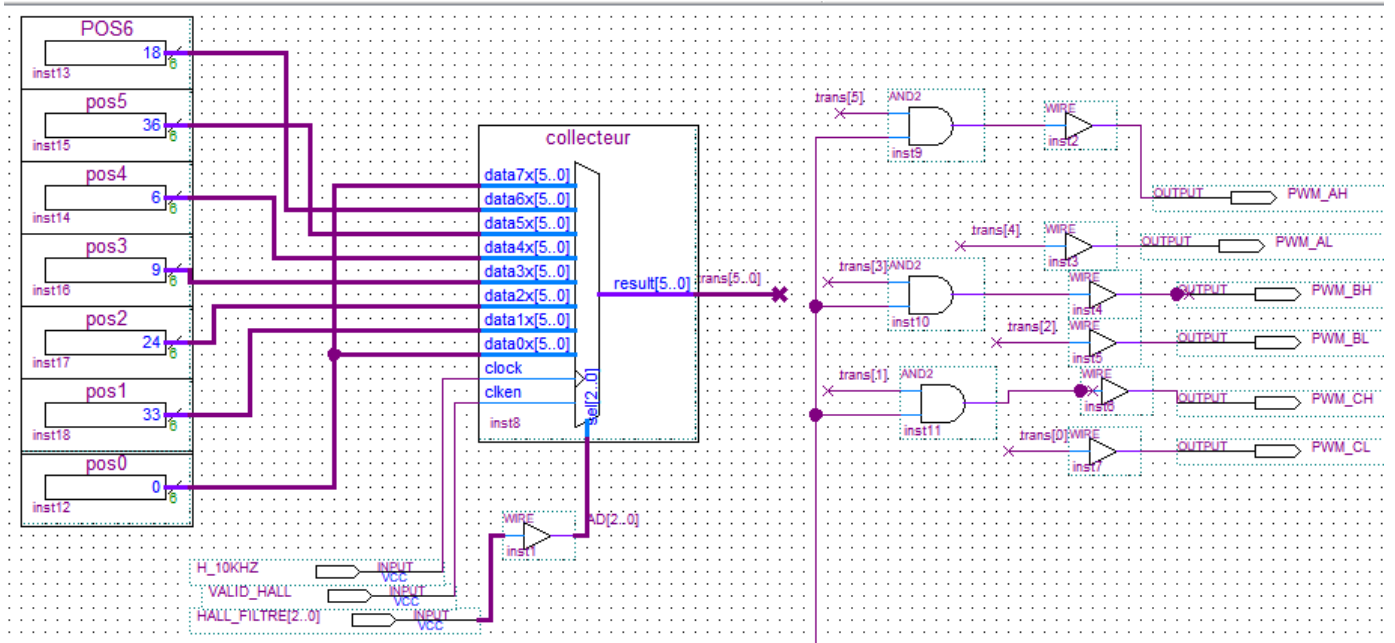
45
46     elsif(hall="010" or hall= "110") then -- 2 et 6
47         PWMAL <= '1';
48         PWMCL <= '0';
49         PWMBL <= '0';
50
51     elsif(hall="100" or hall= "101") then -- 5 et 4
52         PWMBL <= '1';
53         PWMCL <= '0';
54         PWMAL <= '0';
55     elsif(hall="001" or hall= "011") then -- 1 et 3
56         PWMCL <= '1';
57         PWMAL <= '0';
58         PWMBL <= '0';
59
60
61
62
63         end if;
64     end if;
65     END PROCESS;
66 END archi;

```

Insérer ici le code VHDL et le résultat de simulation.

Exemple en BDF





Code :

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.std\_logic\_unsigned.all;

entity collecteur\_num is

port(

pwm : in std\_logic; --spi clk from master

hall : in std\_logic\_vector(2 downto 0); --active low slave select

valid : in std\_logic;

PWMAH : out std\_logic;

PWMAL : out std\_logic;

PWMBH : out std\_logic;

PWMBL : out std\_logic;

PWMCH : out std\_logic;

PWMCL : out std\_logic);

end collecteur\_num;

architecture archi of collecteur\_num is

signal PWMAHi, PWMBHi,PWMCHi: std\_logic;

BEGIN

PROCESS(hall,valid)

BEGIN

--IF (hall'EVENT AND hall='1') THEN

if(valid='1') then

if(hall="010" or hall="011") then

--PWMBH <= pwm;

PWMBHi <= '1';

PWMCHi <= '0';

PWMAHi <= '0';

elsif(hall="100" or hall= "110") then

```

--PWMCH <= pwm;
PWMCHi <='1';

PWMAHi <= '0';
PWMBHi <= '0';

elsif(hall="001" or hall= "101") then
--PWMAH <= pwm;
PWMAHi <='1';

PWMCHi <= '0';
PWMBHi <= '0';

end if;
if(hall = "000" or hall = "111") then
PWMCL <= '0';
PWMBL <= '0';
PWMAL <= '0';

elsif(hall="010" or hall= "110") then -- 2 et 6
PWMAL <= '1';
PWMCL <= '0';
PWMBL <= '0';

elsif(hall="100" or hall= "101") then -- 5 et 4
PWMBL <= '1';
PWMCL <= '0';
PWMAL <= '0';
elsif(hall="001" or hall= "011") then -- 1 et 3
PWMCL <= '1';
PWMAL <= '0';
PWMBL <= '0';

end if;
end if;
END PROCESS;
PWMBH <=PWMBHi and pwm;
PWMCH <=PWMCHi and pwm;
PWMAH <=PWMAHi and pwm;

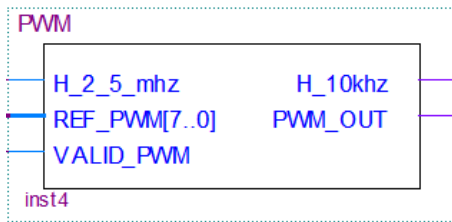
```

```

END archi;

```

### 3.2 générateur de signaux PWM



Dans notre projet, le uC mbed doit pouvoir envoyer au pld un mot de 8 bits // représentant une consigne de PWM (Ref\_PWM[7..0]) variant entre 0 et 255 qui permettra de moduler en largeur d'impulsion la commande des trois transistors du haut de l'onduleur à savoir PWM\_AH, PWM\_BH et PWM\_CH.

La fréquence de cette PWM sera d'environ de 10Khz et nous partirons pour ce faire une horloge rapide présente sur la maquette H\_2\_5Mhz.

**Rôle :** Ce bloc reçoit la consigne de PWM sur 8 bits en // en provenance du microcontrôleur et fournit un signal de PWM de rapport cyclique variable et de fréquence de découpage de 10Khz sur un bit. Ce signal sera plus tard distribué par le bloc collecteur numérique précédent. Il génère également une horloge interne de 10Khz environ à destination des autres blocs fonctionnels

#### Ressources d'entrée :

**H2.5Mhz :** horloge rapide environ 2.5Mhz (présente dans la maquette)

**Valid\_PWM :** signal de validation global de la PWM (actif à 1) permettant de prévoir un arrêt d'urgence ou une limitation en courant par détection de sur intensité. Si Valid\_PWM=0, la sortie PWM\_OUT=0

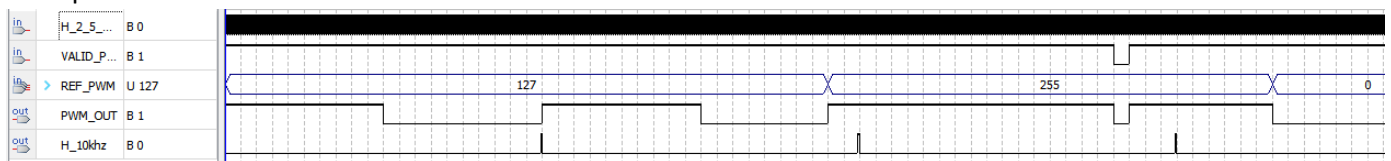
**PWM\_REF[7..0] :** consigne PWM d'entrée fournie par le uC MBED.

#### Ressources de sortie :

**H\_10Khz :** sortie horloge 10Khz pour usage interne dans le PLD.

**PWM\_OUT :** sortie PWM modulée en fonction de PWM\_REF[7..0]

#### Exemple de Fonctionnement attendu du bloc PWM

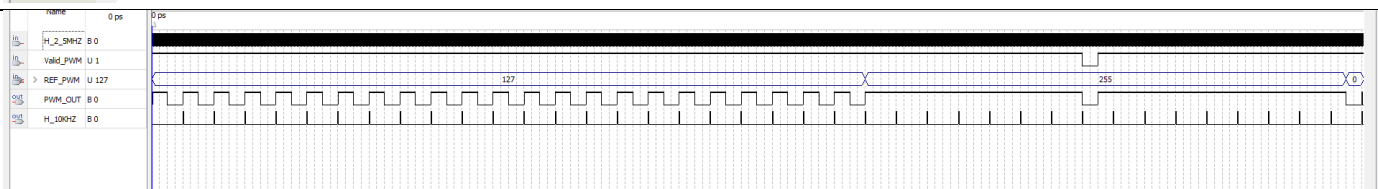


Insérer ici le code VHDL et le résultat de simulation.

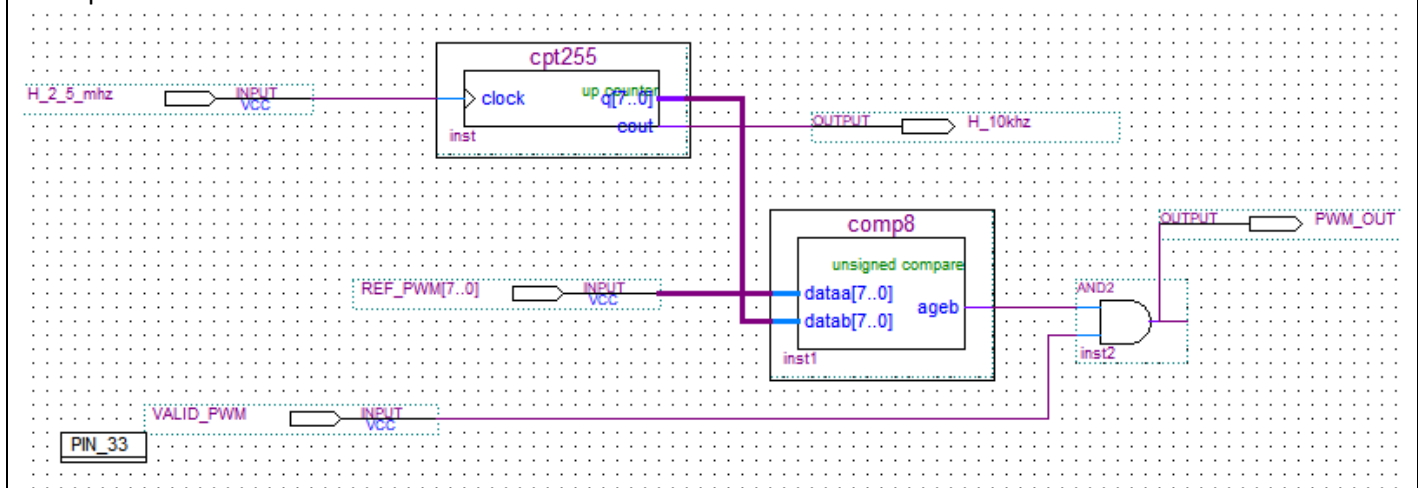
```

1  library IEEE;
2  use IEEE.std_logic_1164.all;
3  use IEEE.std_logic_unsigned.all;
4
5  ENTITY PWM IS
6  PORT(
7      H2_5Mhz : IN STD_LOGIC;
8      VALID_PWM : IN STD_LOGIC;
9      PWM_REF : IN INTEGER RANGE 0 to 255 :=0;
10
11      H_10KHZ : OUT STD_LOGIC;
12      PWM_OUT : OUT STD_LOGIC
13  );
14  END PWM;
15
16  ARCHITECTURE archi OF PWM IS
17  SIGNAL cpt : INTEGER RANGE 0 to 255 :=0;
18  BEGIN
19      PROCESS (H2_5Mhz)
20      BEGIN
21          IF (H2_5Mhz'EVENT AND H2_5Mhz='1') THEN
22              cpt <= cpt + 1;
23
24              IF (cpt <= PWM_REF and VALID_PWM='1') THEN
25                  PWM_OUT <= '1';
26              ELSE
27                  PWM_OUT <= '0';
28              END IF;
29
30              IF (cpt = 255) THEN
31                  H_10KHZ <= '1';
32              ELSE
33                  H_10KHZ <= '0';
34              END IF;
35          END IF;
36      END PROCESS;
37  END archi;

```



Exemple en BDF



Code :

```

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

```

```

ENTITY PWM IS
PORT(
    H2_5Mhz : IN STD_LOGIC;
    VALID_PWM : IN STD_LOGIC;
    PWM_REF : IN INTEGER RANGE 0 to 255 :=0;

    H_10KHZ : OUT STD_LOGIC;
    PWM_OUT : OUT STD_LOGIC
);
END PWM;

ARCHITECTURE archi OF PWM IS
SIGNAL cpt : INTEGER RANGE 0 to 255 :=0;
BEGIN
    PROCESS(H2_5Mhz)
    BEGIN
        IF(H2_5Mhz'EVENT AND H2_5Mhz='1') THEN
            cpt <= cpt + 1;

            IF(cpt <= PWM_REF and VALID_PWM='1') THEN
                PWM_OUT <= '1';
            ELSE
                PWM_OUT <= '0';
            END IF;

            IF(cpt = 255) THEN
                H_10KHZ <= '1';
            ELSE
                H_10KHZ <= '0';
            END IF;
        END IF;
    END PROCESS;
END archi;

```

### 3.3 filtrage de la position mesurée par les capteurs à effets hall

Dans le cas d'un moteur brushless de véhicule électrique, pour les petites puissances (<3KW) il est courant d'avoir les trois signaux logiques issus des capteurs de position à effet hall qui transistent dans la même gaine électrique que les phases d'alimentation de puissance.

De ce fait de cette proximité, par couplage capacitif entre ces fils, il est courant d'observer, sur les signaux logiques hall[CBA], des parasites transmis par couplage électromagnétique au moment des découpages de tension sur les fils de phase d'alimentation. Ce couplage capacitif parasite a d'autant plus d'effets indésirables que les  $dV/dT$  observés au niveau des alimentations de phases sont élevés (cas d'un onduleur à commutation dure et rapide) ce qui est notre cas.

Il faut donc filtrer le signal en provenance des capteurs halls pour diminuer l'effet de ces parasites.

L'algorithme de filtrage est très simple car il s'agit du principe du registre à coïncidence. L'idée est de mémoriser les deux états précédents (sur deux périodes d'horloge H\_10kHz) des valeurs de Hall[C,B,A] et de les comparer à sa valeur actuelle.

Si la condition  $hall[CBA]_{n-2} = hall[C,B,A]_{n-1} = hall[C,B,A]_n$  alors cela signifie que la valeur actuelle  $hall[C,B,A]_n$  est stable depuis deux périodes d'horloge et donc quelle est fiable. Dans ce cas on présente cette nouvelle valeur sur la sortie du bloc ( $hall\_filtre[C,B,A]$ ) et on indique que cette valeur est exploitable ( $Valid\_Hall='1'$ ). Sinon on maintient en sortie la dernière valeur stable connue de  $Hall[C,B,A]$  et on désactive  $Valid\_Hall='0'$ .

```
Code : library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
```

ENTITY FILTRE IS

```
PORT(
    hall_A : IN STD_LOGIC;
    hall_B : IN STD_LOGIC;
    hall_C : IN STD_LOGIC;
    H_10Khz : IN STD_LOGIC;

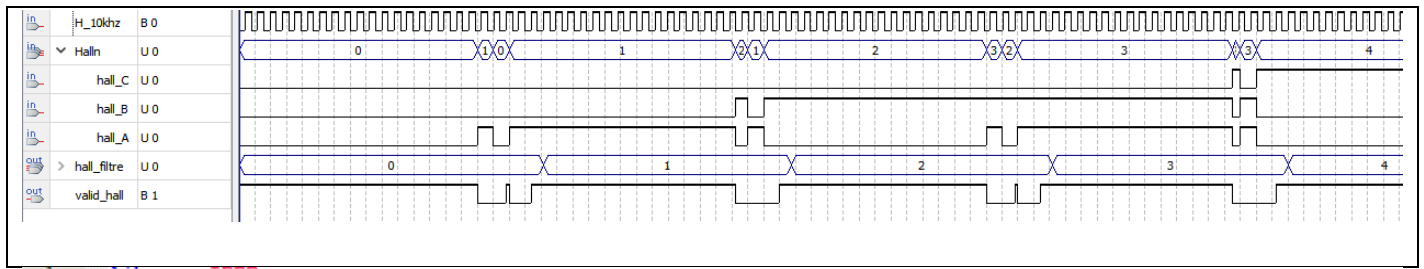
    valid_hall: OUT std_logic:= '0';
    hall_filtre : BUFFER STD_LOGIC_VECTOR(2 downto 0)
);
END FILTRE ;
```

ARCHITECTURE archi OF FILTRE IS

```
SIGNAL hall_filtre_N1 : STD_LOGIC_VECTOR(2 downto 0):="000";
signal hall_entree:STD_LOGIC_VECTOR(2 downto 0);
SIGNAL hall_filtre_N2 : STD_LOGIC_VECTOR(2 downto 0):="000";
BEGIN
    hall_entree<=hall_C & hall_B & hall_A;
```

```
    PROCESS(H_10Khz)
    BEGIN
        IF(H_10Khz'EVENT AND H_10Khz='1') THEN
            hall_filtre_N2<=hall_filtre_N1;
            hall_filtre_N1<=hall_entree;
            IF(hall_filtre_N2=hall_filtre_N1 AND hall_filtre_N1 = hall_entree) THEN
                valid_hall<='1';
                hall_filtre<= hall_entree;
            ELSE
                valid_hall<='0';
            END IF;
        END IF;
    END PROCESS;
```

```
END archi;
```

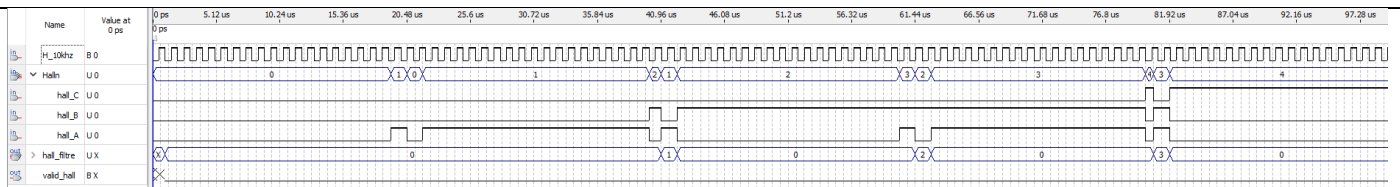


```

1  library IEEE;
2  use IEEE.std_logic_1164.all;
3  use IEEE.std_logic_unsigned.all;
4
5  ENTITY FILTRE IS
6  PORT (
7      hall_A : IN STD_LOGIC;
8      hall_B : IN STD_LOGIC;
9      hall_C : IN STD_LOGIC;
10     H_10Khz : IN STD_LOGIC;
11
12     valid_hall: OUT std_logic:= '0';
13     hall_filtre : BUFFER STD_LOGIC_VECTOR(2 downto 0)
14 );
15 END FILTRE ;
16
17 ARCHITECTURE archi OF FILTRE IS
18     SIGNAL hall_filtre_N1 : STD_LOGIC_VECTOR(2 downto 0) := "000";
19     signal hall_entree:STD_LOGIC_VECTOR(2 downto 0);
20     SIGNAL hall_filtre_N2 : STD_LOGIC_VECTOR(2 downto 0) := "000";
21 BEGIN
22     hall_entree<=hall_C & hall_B & hall_A;
23
24
25     PROCESS(H_10Khz)
26     BEGIN
27         IF (H_10Khz'EVENT AND H_10Khz='1') THEN
28             hall_filtre_N2<=hall_filtre_N1;
29             hall_filtre_N1<=hall_entree;
30             IF (hall_filtre_N2=hall_filtre_N1 AND hall_filtre_N1 = hall_entree) THEN
31                 valid_hall<='1';
32                 hall_filtre<= hall_entree;
33             ELSE
34                 valid_hall<='0';
35             END IF;
36         END IF;
37     END IF;
38 END PROCESS;
39
40 END archi;

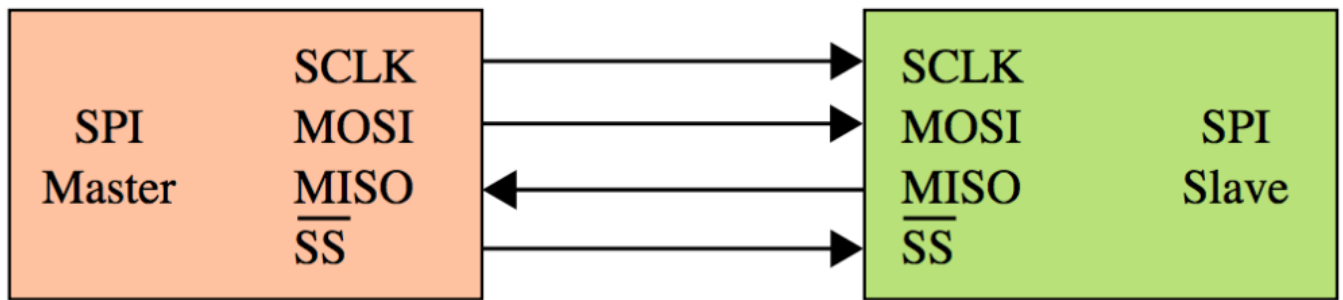
```

Insérer ici le code VHDL et le résultat de simulation.



### 3.4 SPI Slave

#### Rappels



Analyser le programme, le tester et le simuler.

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;

entity spi_slave is
port(
sck      : in std_logic;    --spi clk from master
cs_n     : in std_logic;    --active low slave select
mosi     : in std_logic;    --master out, slave in
miso     : out std_logic); --master in, slave out
rx_buf   : std_logic_vector(7 downto 0) := (others => '0'); --receiver buff
tx_buf   : std_logic_vector(7 downto 0) := (others => '0'); --transmit buff
end spi_slave;

architecture rtl of spi_slave is

begin

-----
--Sampling MISO on Falling rising_edge
-----

process (sck)
signal bitCount: integer range 0 to 7;
begin

if falling_edge(sck) then
  if(cs_n='0') then
    miso<=tx_buf(bitCount); --TX data;
    bitCount:=bitCount+1;
    if(bitCount=7) then
      bitCount:=0;
    end if;
  end if;
end if;
end process;

-----
--Sampling MOSI Data on rising_edge
-----

process (sck)
signal bitCount: integer range 0 to 7;
begin
  
```



```

if rising_edge(sck) then
  if (CS_N='0') then
    rx_buf(bitCount)<=mosi; -- LSB bit is received first
    bitCount:=bitCount+1;
    if (bitCount=8) then
      bitCount:=0;
    end if;
  end if;
end if;
end process.

```

#### Autre solution de registre à décalage pour le mosi

```

signal rx_buf_int: std_logic_vector(rx_buf'range);
process (sck)
begin
  if rising_edge(sck) then
    if (CS_N='0') then
      for i in 1 to 7 loop
        rx_buf_int (8-i) <= rx_buf_int (7-i);
      end loop;
      rx_buf_int (0) <= Mosi;
    end if;
  end if;
end process.
rx_buf <= rx_buf_int ;

```

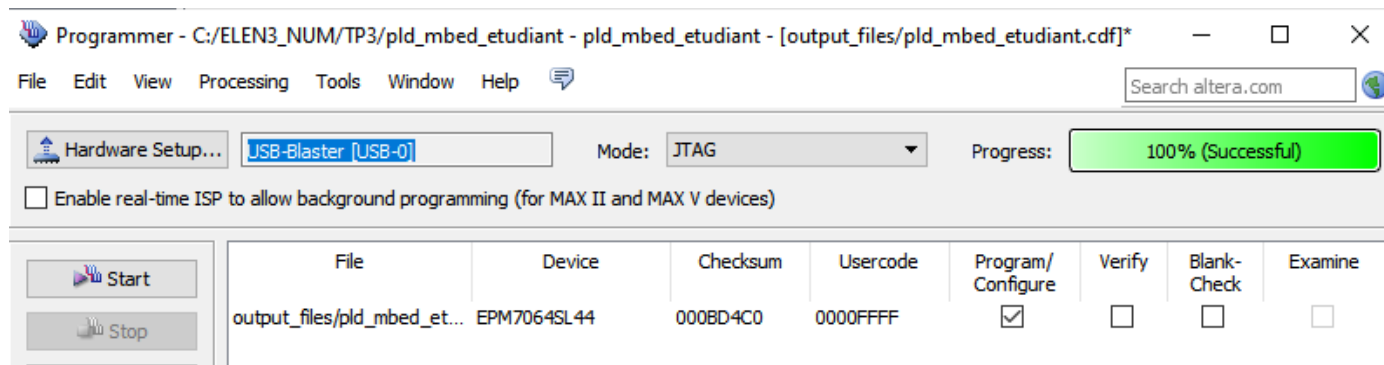
Dans ce programme, il y a un problème au moment du transfert des données vers tx\_buf. Corriger le problème.

Insérer ici le fichier vhdl corrigé et le résultat de simulation

## 4 Intégration et test

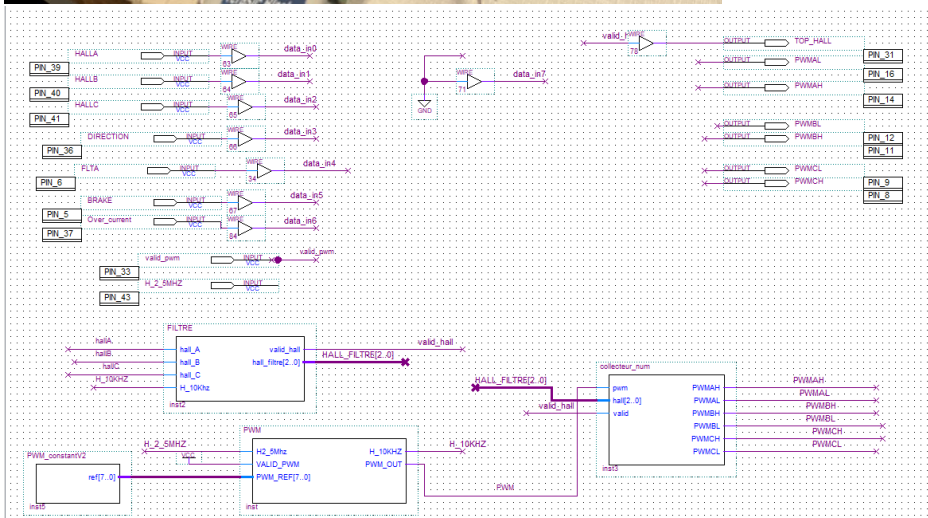
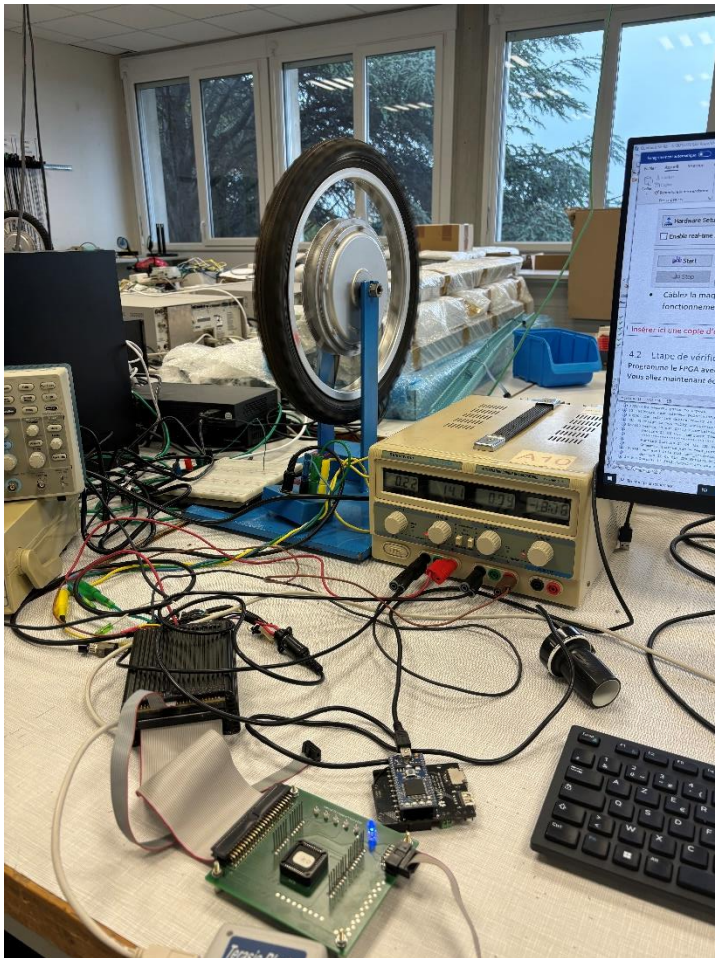
### 4.1 Etape de vérification n°1 avec une réf pwm constante:

- Placer une constante 8 bits = 127 sur l'entrée REF\_PWM du bloc collecteur\_num, et on validera à VCC en permanence l'entrée Valid\_Pwm du bloc générateur PWM pour ce test.
- Terminer les interconnexions sur le fichier général d'interconnexions **top.bdf**, le placer en top level entity et lancer une compilation générale
- Programmez le PLD Altera 7064SLC44-10 avec le fichier **epm7064\_2024.pof** issue de la compilation générale.



- Câblez la maquette SAE roue sur le moteur brushless de la roue de vélo ou de scooter et tester le fonctionnement pratique de l'autopilotage . La roue doit se mettre à tourner.

Insérer ici une copie d'écran de top.bdf et une preuve de fonctionnement (photo)



## 4.2 Etape de vérification n°2 : test du spi

Programme le FPGA avec la solution finale (communication spi)

Vous allez maintenant écrire un programme de test en SPI sur Keil Studio et tester la communication avec le FPGA

Insérer ici une copie d'écran de top.bdf et une preuve de fonctionnement (oscilloscope des SCK MOSI)

## 5 Résumé VHDL

## 5.1 Exemple d'un multiplexeur : 4 solutions

```
library IEEE;

use IEEE.std_logic_1164.all;

entity MUX is

port(Sel : in std_logic_vector(1 downto 0) ;

A, B, C, D : in std_logic;

Y1, Y2, Y3, Y4 : out std_logic);

end MUX;
```

```
architecture RTL of MUX is

signal tmp : std_logic ;

begin

Y1 <= A when Sel="00" else

    B when Sel="01" else

    C when Sel="10" else

    D when Sel="11" ;
```

```
with Sel select Y2 <=

    A when "00",

    B when "01",

    C when "10",

    D when "11";
```

```
p0 : process (A, B, Sel)

begin

    if (Sel = "00") then          Y3 <= A;

    elsif (Sel = "01") then      Y3 <= B;
```

```

        elsif (Sel = "10") then            Y3 <= C;

        else                                Y3 <= D;

end process;

```

```

process (A, B, C, D, Sel)

begin

    case Sel is

        when "00" => Y4 <= A;

        when "01" => Y4 <= B;

        when "10" => Y4 <= C;

        when "11" => Y4 <= D;

        when others => Y4 <= A;

    end case;

end proces

```

## 5.2 Exemple d'un multiplexeur : 4 solutions

Essaie SPI : M laurent VHDL

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;

entity SPI_laurent is
port(
sck      : in std_logic;    --spi clk from master
cs_n     : in std_logic;    --active low slave select
mosi     : in std_logic;    --master out, slave in
miso     : out std_logic;   --master in, slave out
    rx_buf : out std_logic_vector(7 downto 0) := (others => '0');
--receiver buff
    tx_buf : in std_logic_vector(7 downto 0) := (others => '0') -
--transmit buff
    );
end spi_laurent;

architecture rtl of spi_laurent is
signal bitCount1: integer range 0 to 7;
signal bitCount2: integer range 0 to 7;

```

```

        signal rx_temp : std_logic_vector(7 downto 0) := (others => '0'); --
receiver buff
        signal tx_temp : std_logic_vector(7 downto 0) := (others => '0'); --
transmit buff
        signal MISO_TEMP : std_logic;

begin

-----
--Sampling MISO on Falling rising_edge
-----

--process (sck)
process (cs_n, sck,tx_temp,bitCount1 )
begin
    if(cs_n'event and cs_n='0') then

        tx_temp <= tx_buf;
        --MISO_TEMP<=tx_temp(7);
        end if;

        if (cs_n='0') then
        if (sck='0') then

            MISO_TEMP<=tx_temp(7-bitCount1); --TX data;
            end if;

            if (sck'event and sck='1') then
                -- if (sck='1') then
                bitCount1<=bitCount1+1;

                rx_temp(7-
bitCount1)<=mosi; -- LSB bit is received first

                end if;

            else
                bitCount1<=0;

            end if;

        end if;
    end process;

process (cs_n)
begin

    if (cs_n'event and cs_n='1') then

        rx_buf <=rx_temp;
        end if;
    end process ;

```

```

MISO<=MISO_TEMP; --when cs_n='0' else 'Z';
-----
--Sampling MOSI Data on rising_edge
-----

end rtl;


M Salvat VHDL

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;

entity spi is
port(
sck      : in std_logic;    --spi clk from master
cs_n     : in std_logic;    --active low slave select
mosi     : in std_logic;    --master out, slave in
miso     : out std_logic;   --master in, slave out
          rx_buf : out std_logic_vector(7 downto 0) := (others => '0');
--receiver buff
          tx_buf : in std_logic_vector(7 downto 0) := (others => '0') -
--transmit buff
          );
end spi;

architecture rtl of spi is
signal bitCount1: integer range 0 to 7 :=0;

      signal rx_temp : std_logic_vector(7 downto 0) ;
      signal tx_temp : std_logic_vector(7 downto 0) ;
      signal MISO_TEMP : std_logic;

begin

-----
--Sampling MISO on Falling rising_edge
-----

--process (sck)
process (cs_n, sck)
begin
      if(cs_n='0') then
            if falling_edge(sck) then
                  bitCount1<=bitCount1+1;
                  MISO_TEMP<=tx_temp(6-bitCount1); --TX data;
            end if;
      else
            bitCount1<=0;
            tx_temp <= tx_buf;

```

```

                                MISO_TEMP<=tx_temp(7);
        end if;
    end process;
    process (sck, cs_n)
    begin
        if (cs_n='1') then
            rx_buf <=rx_temp;

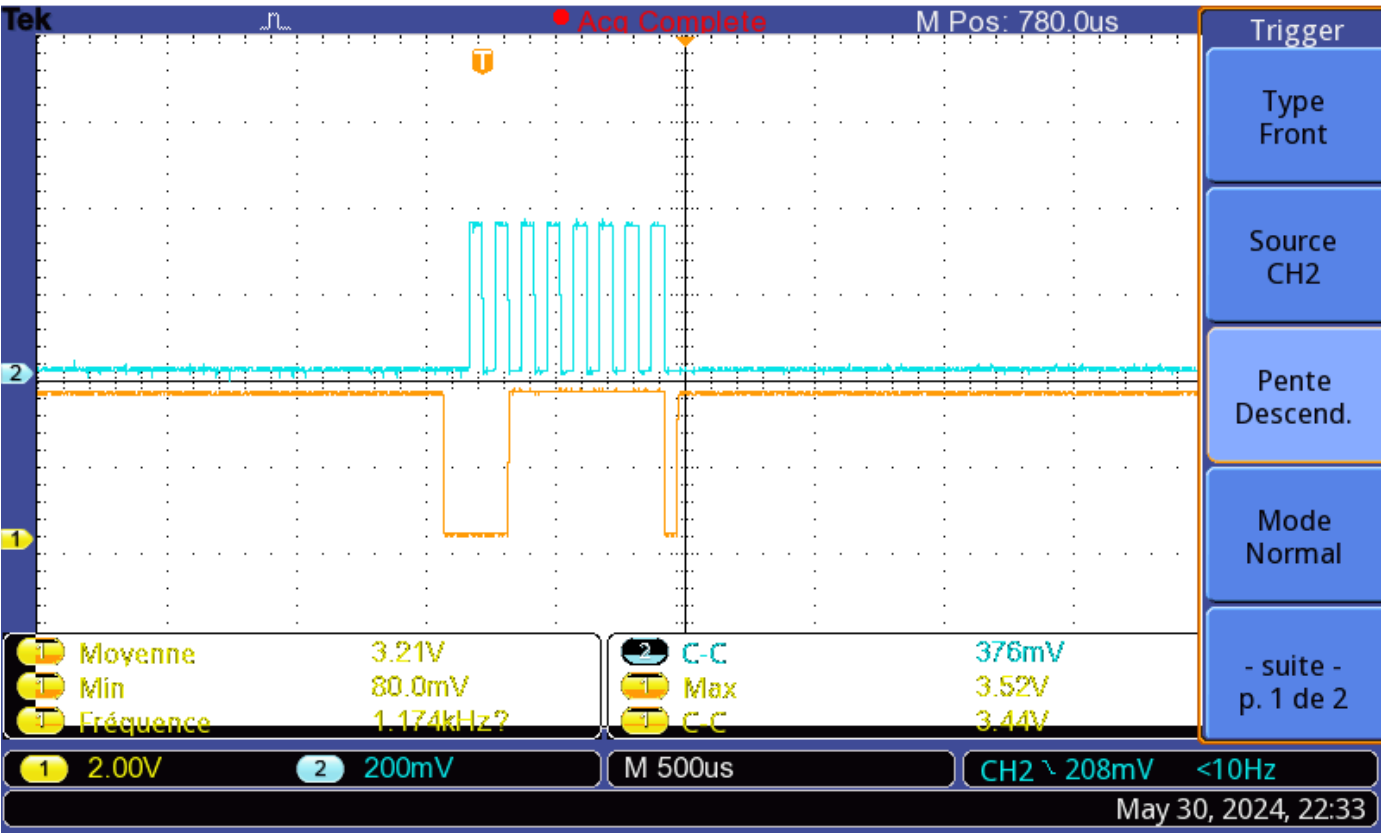
            else
                if rising_edge(sck) then
                    rx_temp(7-bitCount2)<=mosi; -- LSB bit is received
first
                    bitCount2<=bitCount2+1;
                    end if;
                end if;

    end process;
    MISO<=MISO_TEMP when cs_n='0' else 'Z';
    -----
    --Sampling MOSI Data on rising_edge
    -----

end rtl;

```





TBS 1052B-EDU - 17:34:44 30/05/2024

## Mbed

```
1  #include "mbed.h"
2
3  SPI spi(p5, p6, p7); // mosi, miso, sclk
4  DigitalOut cs(p8);
5  BusOut myleds(LED4, LED3, LED2, LED1);
6
7  Serial pc(USBTX, USBRX); // tx, rx
8
9  DigitalOut VALID_PWM(p21); // Générateur PWM
10 InterruptIn VALID_HALL(p22); // Capteurs HALL
11 AnalogIn Vgaz(p17); // Poignée de gaz
12 AnalogIn Vbat(p18); // Tension de la batterie
13 AnalogIn VTemp(p19); // Temperature du convertisseur
14 AnalogIn Vi(p20); // Courant mesuré de la batterie
15
16 int main() {
17     cs.write(1);
18     cs.write(0);
19     cs.write(1);
20
21     // spi.frequency(10000);
22     // spi.format(8,3);
23     spi.frequency(100000);
24     while (1) {
25         // float Vgazlu = Vgaz.read();
26         // printf("valeur lue = %f\n\r", Vgazlu);
27         unsigned int valeur = 0;
28         scanf("%d",&valeur);
29
30         cs.write(0);
31         unsigned char val_lue = spi.write(valeur);
32         // wait(0.001);
33
34         cs.write(1);
35         // valeur = (valeur >> 1) & 0xFFFF;
36         printf("valeur lue = %d\n\r", val_lue);
37         wait(0.02);
38     }
39 }
40
```