|  |  |  |
| --- | --- | --- |
|  | effinergie - IUT GEII  **BUT2 ALT**  **SAE ROUE**  **2023 - 2024** | **Partie 1 (Quartus et uc)** |
| **Objectifs :**   * **Comprendre la commande trapèze d’un moteur brushless** * **Réaliser les 4 blocs VHDL** * **Tester la communication entre le MAX et le LPC1768** | | |

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

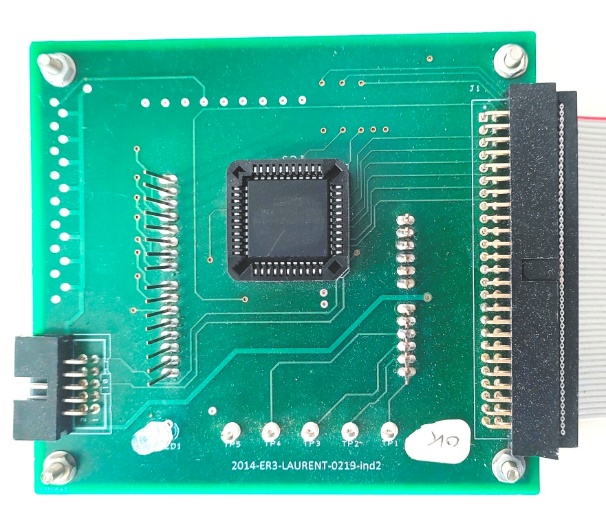
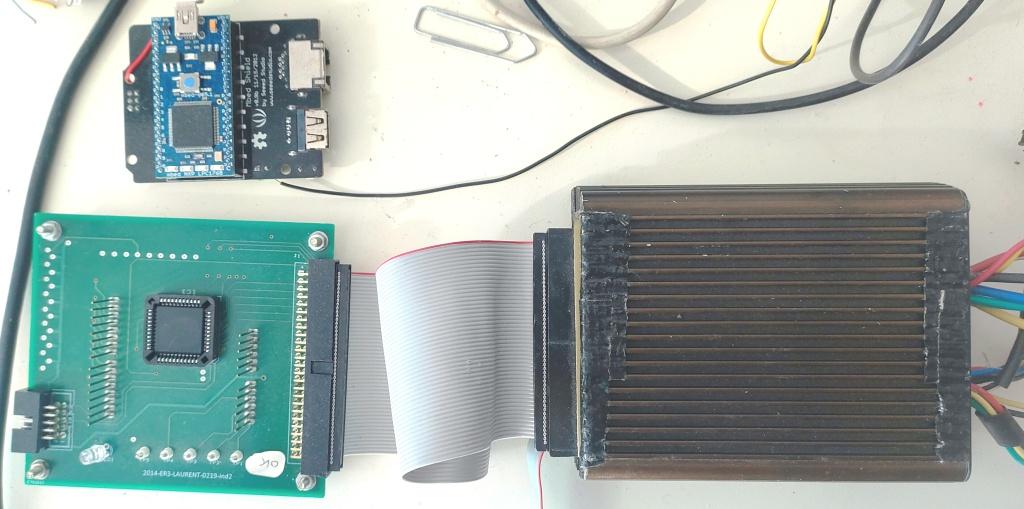
 

Figure 1: carte Max7000S utilisée en SAE roue( gauche) et ensemble des cartes de la SAE Roue VAE (droite)

## Création du projet

Créer votre projet sur 

Une image contenant texte, capture d’écran, Police, Marque

Description générée automatiquementUne image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement

Une image contenant texte, Appareils électroniques, capture d’écran, logiciel

Description générée automatiquement

Puis new File Une image contenant texte, capture d’écran, Police, ligne

Description générée automatiquement

Une image contenant texte, capture d’écran, Police, logiciel

Description générée automatiquement Une image contenant texte, ligne, Police, Tracé

Description générée automatiquement

Puis insérer input et output

Une image contenant texte, capture d’écran, affichage, logiciel

Description générée automatiquement

Voici le résultat

Une image contenant texte, capture d’écran, diagramme, ligne

Description générée automatiquement

Mettre en top level

Une image contenant texte, capture d’écran, logiciel, Icône d’ordinateur

Description générée automatiquement

puis compiler

Une image contenant texte, capture d’écran, Police, symbole

Description générée automatiquement

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

Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement

Une image contenant texte, capture d’écran, nombre, logiciel

Description générée automatiquement

Votre projet est prêt à l’emploi !

# Architecture générale du projet SAE Roue de VAE

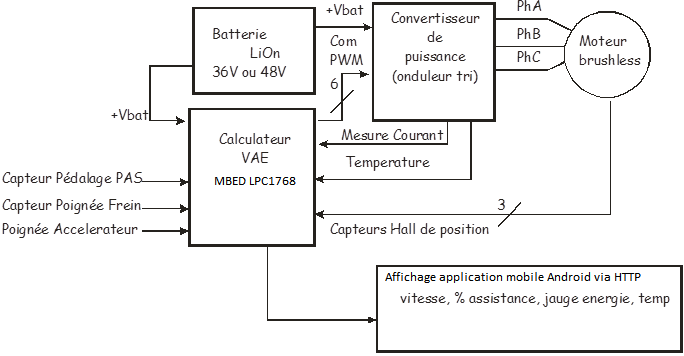


Figure 2: architecture générale du projet SAE roue VAE

*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).

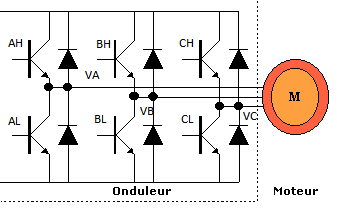
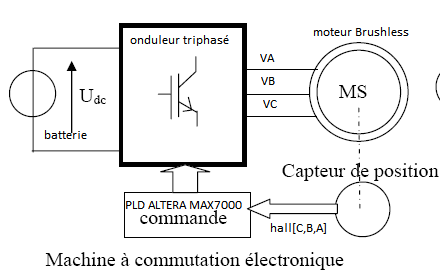


Figure 3: architecture de l'autopilotage (gauche) et détail de l'onduleur (droite)

La Partie puissance est assurée par un onduleur triphasé préfabriqué (boite 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.

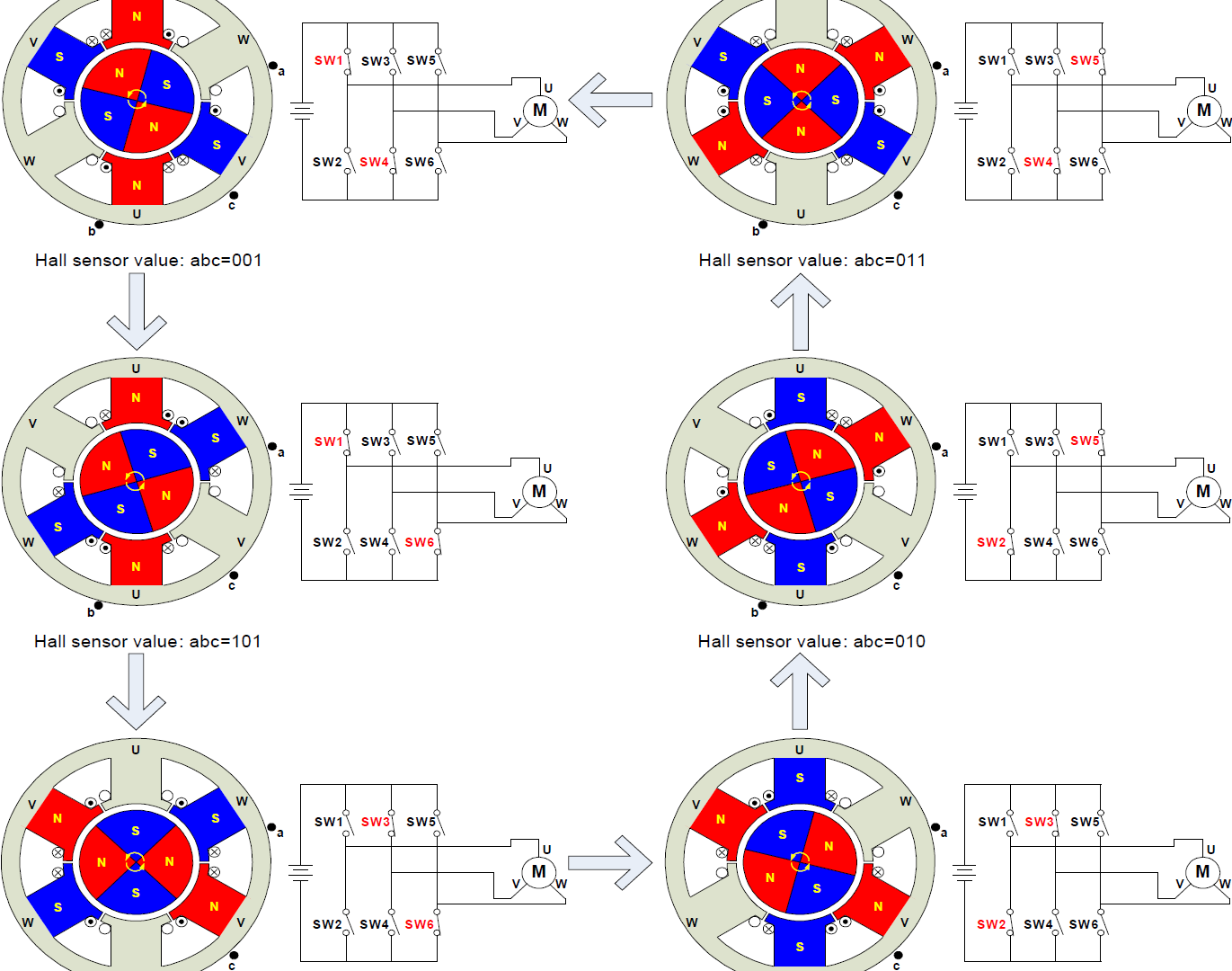
A partir de là il élabore les 3 consignes de tension VA,VB,VC en fonction de la position actuelle du rotor (capteurs Hall[C,B,A] ) et de la consigne de vitesse que désire l’utilisateur (Référence PWM au format 8 bits // fournis par le uC MBED au PLD ALTERA).

*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).



# Création des blocs et tests unitaires (simulation)

Vous allez créer 4 blocs VHDL :

* Collecteur numérique
* Gene PWM
* Filtre
* SPI\_Slave

## Une image contenant texte, capture d’écran, Police, ligne Description générée automatiquementCollecteur 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.

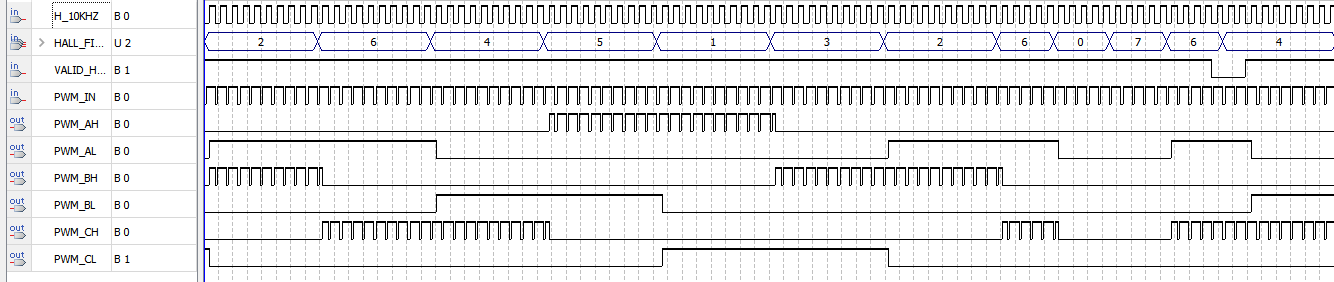
**VALID :** indique à quel moment les capteurs de halls présentent une valeur stable. (Actif à 1)

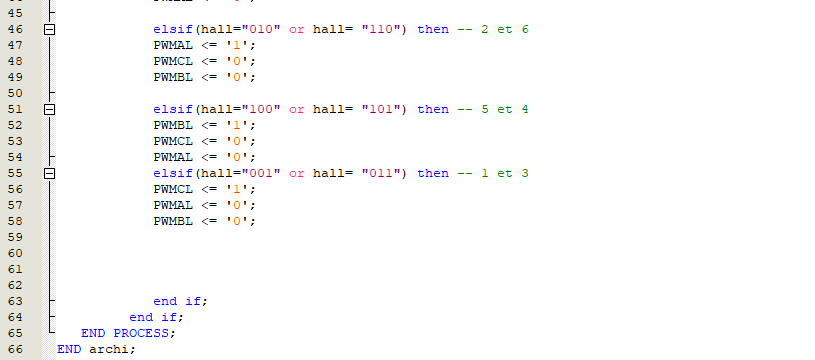
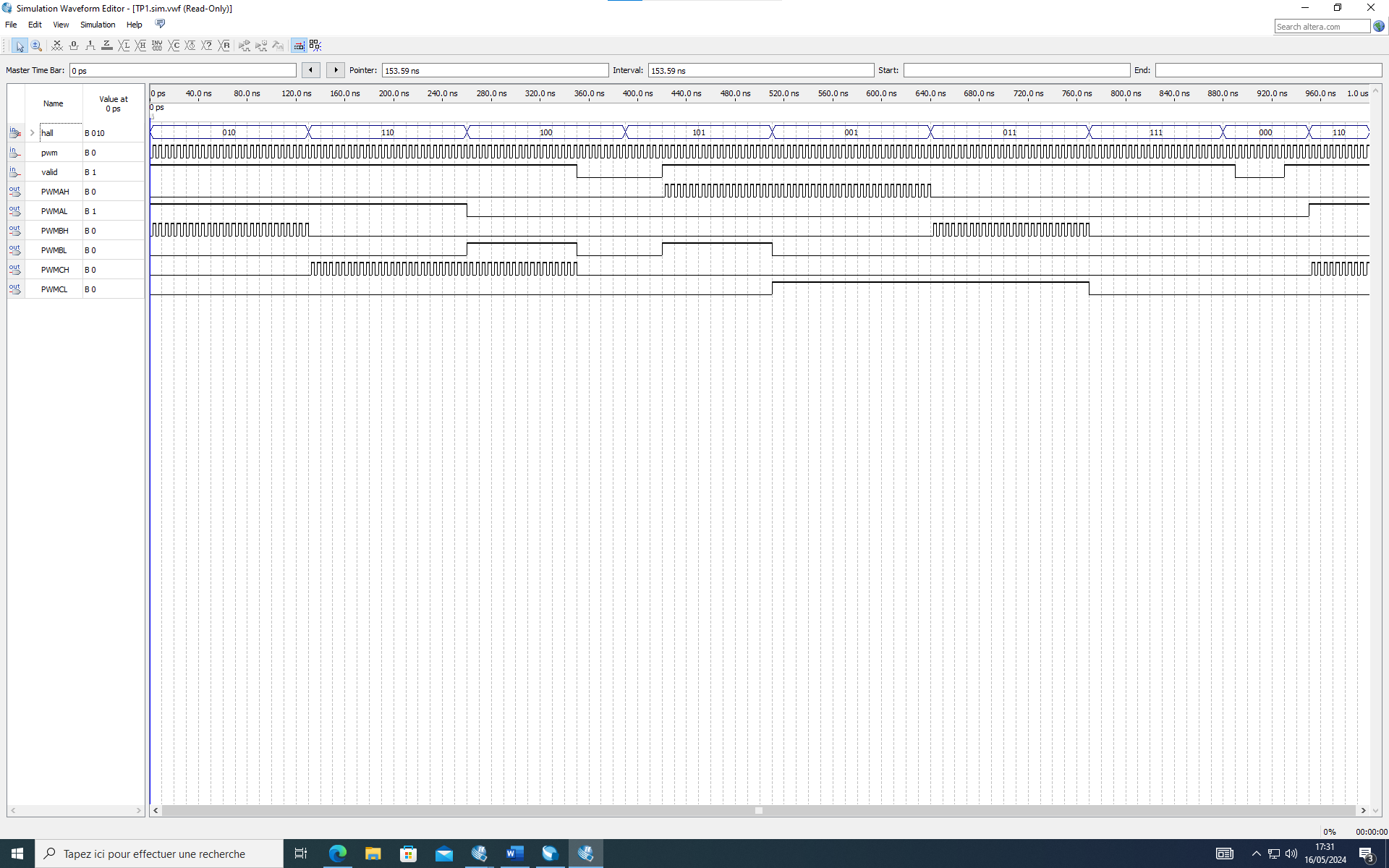
**PWM** : entrée de PWM modulée en largeur d’impulsion suivant la consigne de pédalage ou de gaz fréquence 10kHz.

***Ressources de sortie :***

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

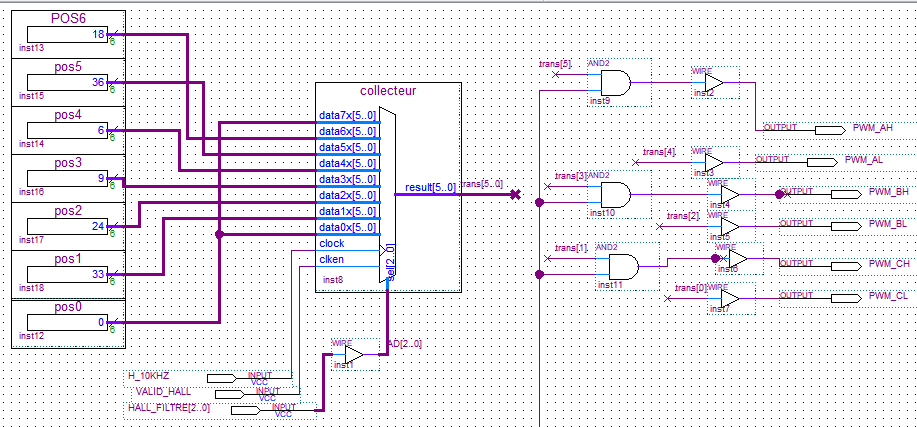
Simulation de fonctionnement attendu pour la marche avant





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;

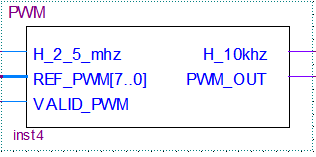
PWMBH <=PWMBHi and pwm;

PWMCH <=PWMCHi and pwm;

PWMAH <=PWMAHi and pwm;

END archi;

## 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’entré****e* :

**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

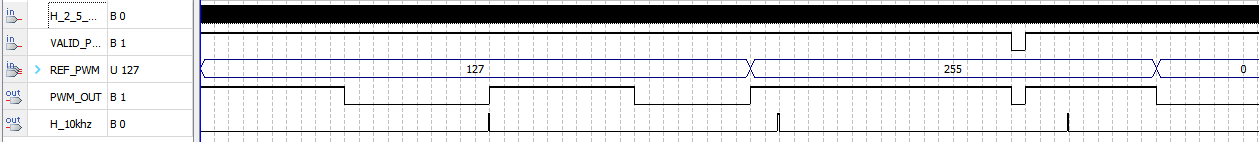
**PWM\_REF[7..0]** : consigne PWM d’entrée fournie par le uC MBED.

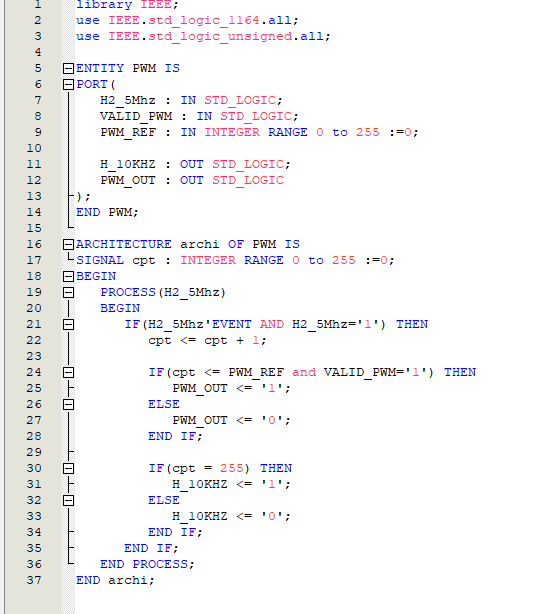
***Ressources de sortie****:*

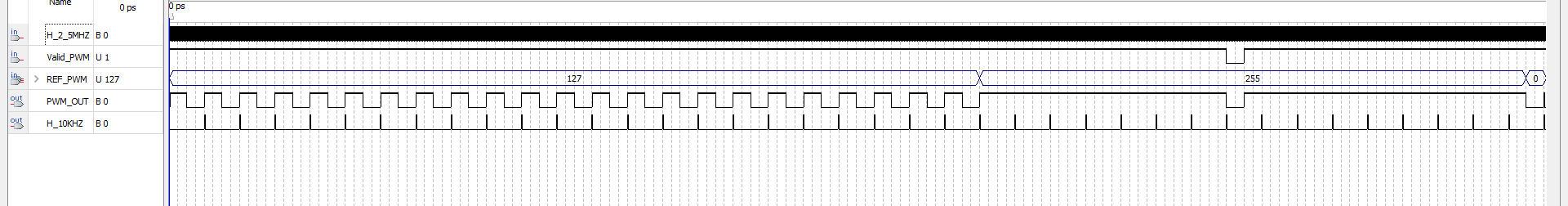
**H\_10Khz**: sortie horloge 10Khz pour usage interne dans le PLD.

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

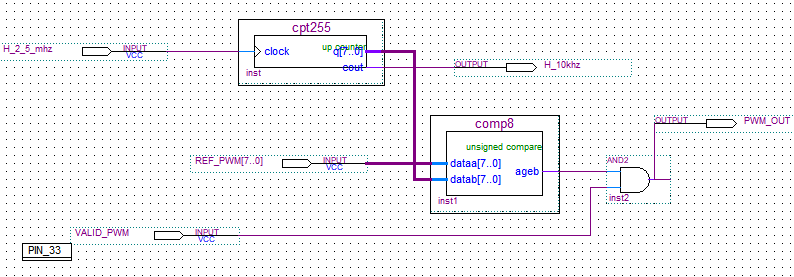
Exemple de Fonctionnement attendu du bloc PWM



Insérer ici le code VHDL et le résultat de simulation. 



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;

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

Dans le cas d’un moteur brushless de véhicule electrique, pour les petites puissances (<3KW) il est courant d avoir les trois signaux logiques issus des capteur de position à effet hall qui transistent dans la même gaine electrique 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 electromagnétiques au moment des découpages de tension sur les fils de phase d’alimentaiton. Ce couplage capacitif parasite a d’autant plus d’effets indesirables 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’algorythme de filtrage est très simple car il s’agit du principe du registre à coincidence. L’idée est de mémoriser les deux états précédents (sur deux periodes d’horloge H\_10khz) des valeurs de Hall[C,B,A] et de les compararer 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 derniere 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

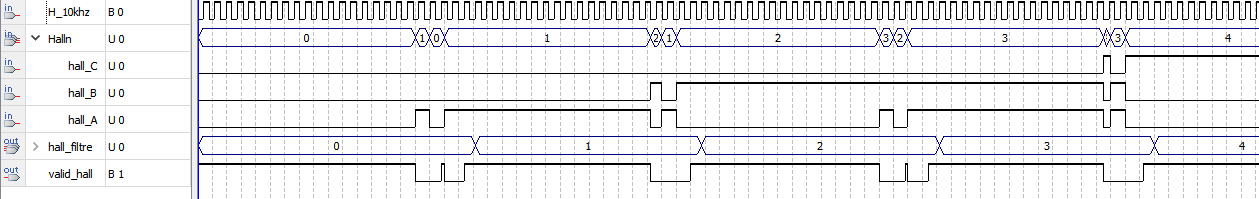
valid\_hall<='0';

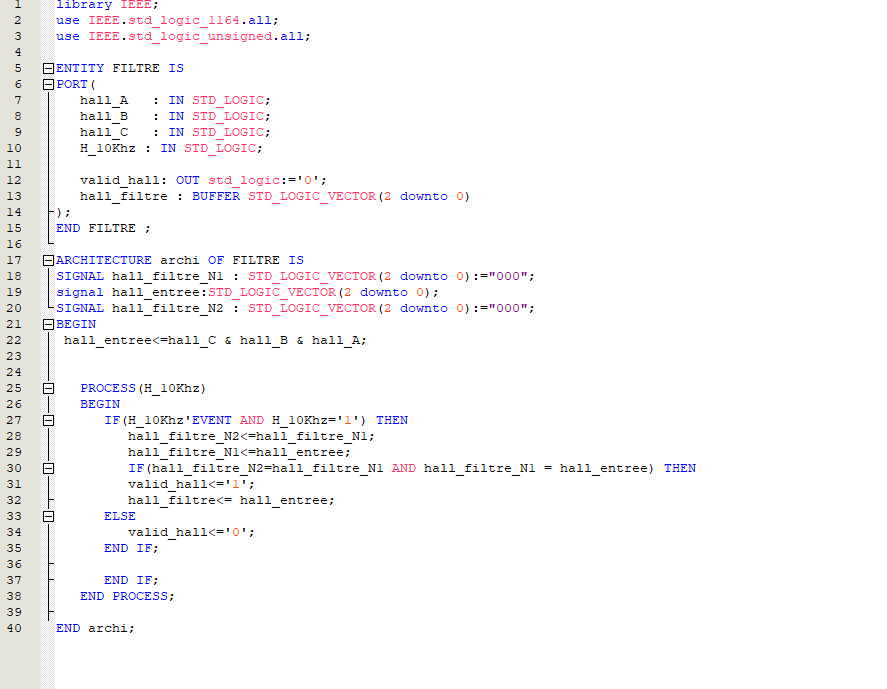
END IF;

END IF;

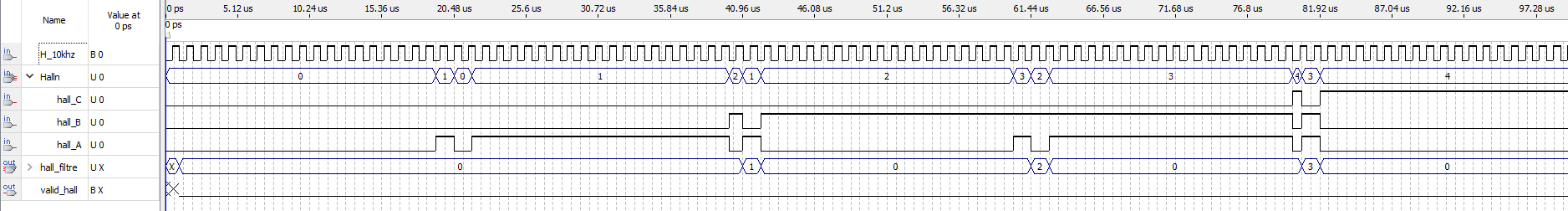
END PROCESS;

END archi;





Insérer ici le code VHDL et le résultat de simulation.



## SPI Slave

Rappels

Une image contenant texte, capture d’écran, Police, ligne

Description générée automatiquement

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 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

# Intégration et test

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

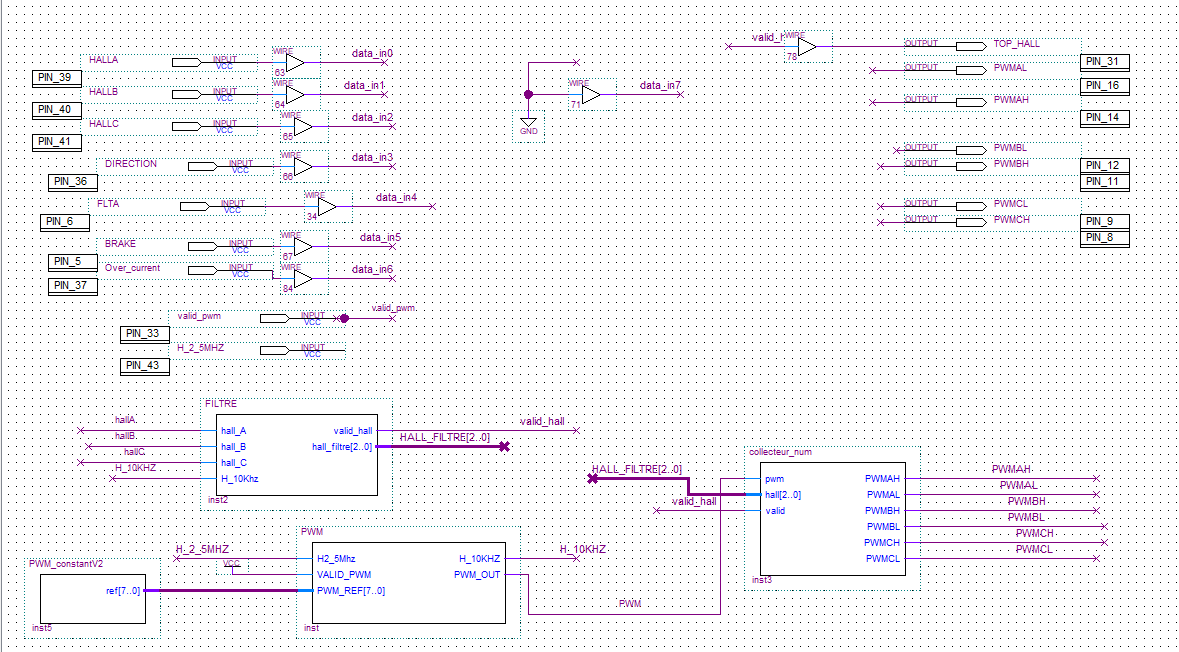
* Placer une constante 8 bits = 127 sur l’entrée REF\_PWM du bloc collecteur\_num, et on validera à VCC en permanence l’entrée Valid\_Pwm du bloc générateur PWM pour ce test.
* Terminer les interconnexions sur le fichier général d’interconnexions **top.bdf**, le placer en top level entity et lancer une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

Une image contenant texte, capture d’écran, logiciel, Police

Description générée automatiquement

* 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)



## 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)

# Résumé VHDL

## 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

## 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 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';

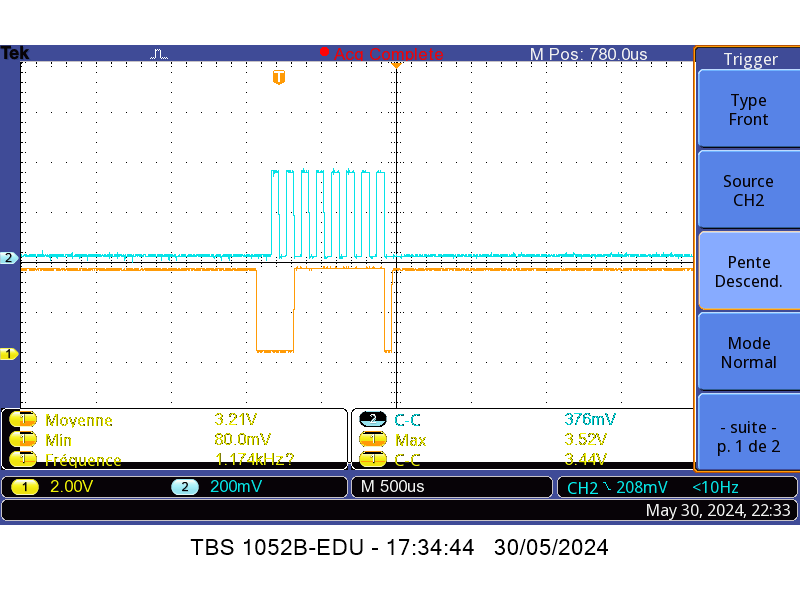
--------------------------------------------------------------------

--Sampling MOSI Data on rising\_edge

--------------------------------------------------------------------

end rtl;

Oscilo



Mbed

