



# Ecole d'ingénieurs Sup Galilée

**Spécialité Instrumentation**

*TP projet FPGA 3*

***Réalisation des  
dispositifs (RPPE)***

***Codes VHDL***

Carte électronique moteur pas à pas

9/2/2022

## Sommaire

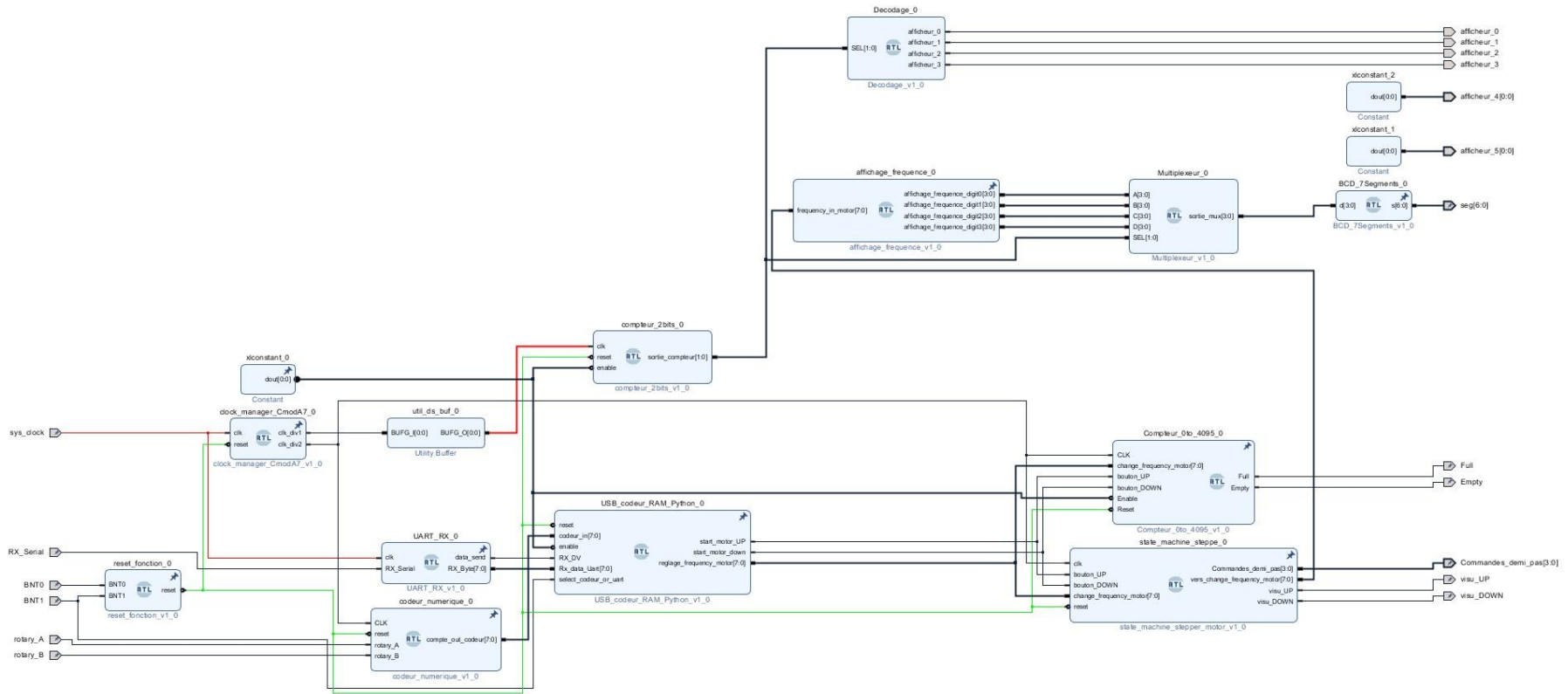
### Projet HDL sous Vivado 2020.1

- Vue d'ensemble schéma TP moteur pas à pas sur Vivado
- Module VHDL gestion des horloges + code VHDL
- Module VHDL machine d'état moteur pas à pas + code VHDL
- Module VHDL UART liaison série (RX) + code VHDL
- Modules VHDL pour multiplexer et afficher la fréquence du moteur + codes VHDL
- Module VHDL encodeur numérique + machine d'état + code VHDL
- Module VHDL gestion des données d'entrées + code VHDL
- Fichier de contrainte \*.xdc

### Script Python pour programmer la vitesse du moteur et le sens de rotation

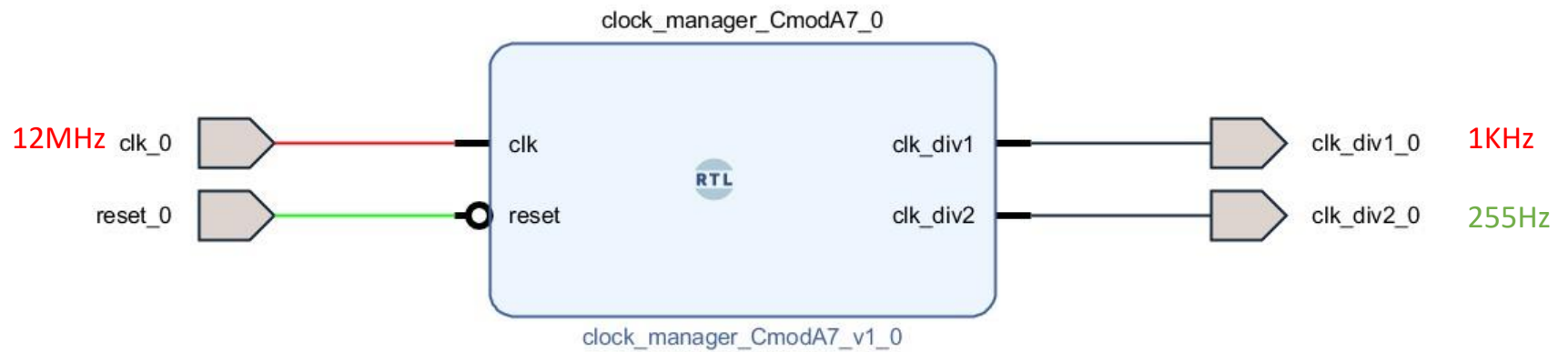
1. <https://www.framboise314.fr/installer-python-et-pyserial-sur-windows/>
2. script rampe de fréquence linéaire et en cos

## Vue d'ensemble schéma TP moteur pas à pas sur Vivado 2020.1



Fabrice Wiotte LPL

## Module VHDL gestion des horloges



Fabrice Wiotte LPL

## Code VHDL gestion des horloges

```
-- Déclaration des bibliothèques utilisées
library IEEE;
use IEEE.STD_LOGIC_1164.ALL; -- inclus signaux améliorés du type std_ulogic--
use IEEE.STD_LOGIC_UNSIGNED.all; -- calcul numérique non signé sur le type std_logic_vector-
```

```
entity clock_manager_CmodA7 is
```

```
Port ( clk : in STD_LOGIC; --12MHz
      clk_div1 : out STD_LOGIC; --1000Hz
      clk_div2 : out STD_LOGIC; --255Hz
      ce : in STD_LOGIC;
      reset : in STD_LOGIC);
```

Déclaration des  
Entrées-sorties

```
end clock_manager_CmodA7;
```

```
architecture Behavioral of clock_manager_CmodA7 is
```

```
--pour compter jusqu'a (100000 -1) il faut 17 bits ( $2^{17}=131072$ )
```

```
signal count1: INTEGER range 0 to 100000 := 0;
SIGNAL clock_int1: STD_LOGIC := '0';
signal count2: INTEGER range 0 to 392157:= 0;
SIGNAL clock_int2: STD_LOGIC := '0';
CONSTANT M1: INTEGER := 100000; -- resultat de la division pour 1000Hz
CONSTANT M2: INTEGER := 392157; -- resultat de la division pour 255Hz
```

Déclaration des  
signaux

```
Begin
```

```
--Divise par 10000 FOUT = 1000Hz synchro affichage et synchro state machine--
```

```
PROCESS(clk,ce,reset)
BEGIN
if reset='1' then
    count1 <= 0;
    count2 <= 0;
ELSIF rising_edge(clk) then
    if ce ='1' then
        IF count1 <= M1-1 THEN --Divise par 10000 FOUT = 1000Hz synchro affichage
            count1 <= count1 + 1;
        ELSE
            count1 <= 0;
        END IF;
        IF count2 <= M2-1 THEN --Divise par 255 FOUT = 255Hz synchro state machine
            Count2 <= count2 + 1;
        ELSE
            Count2 <= 0;
        END IF;
    end if;
END IF;
END PROCESS;
--à la moitié du comptage on change la valeur de clock_1Hz_int (rapport cyclique = 1/2)
clock_int1 <= '1' WHEN count1 <= M1/2 ELSE '0';
clk_div1 <= clock_int1;
clock_int2 <= '1' WHEN count2 <= M2/2 ELSE '0';
clk_div2 <= clock_int2;
```

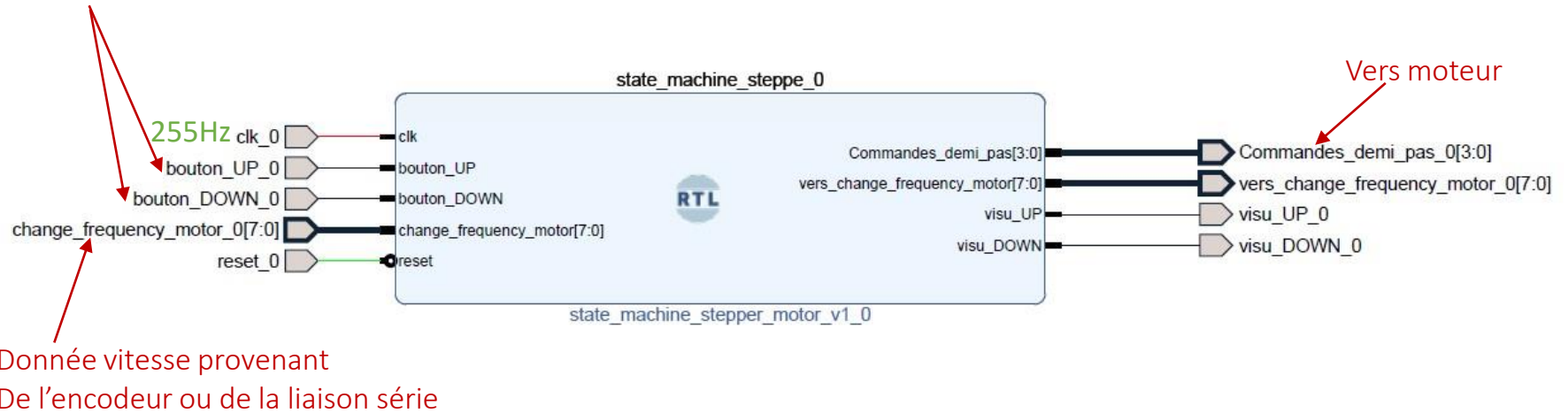
Le  
process  
Division  
d'horlog  
e

Affectation des  
Signaux déclarés sur les sorties

```
end Behavioral;
```

## Module VHDL machine d'état moteur pas à pas

Commandes sens de rotation



Fabrice Wiotte LPL

## Code VHDL machine d'état moteur pas à pas

```
-- Déclaration des bibliothèques utilisées
library IEEE;
use IEEE.STD_LOGIC_1164.all; -- inclus signaux améliorés du type std_ulogic--
use IEEE.STD_LOGIC_ARITH.all;
use IEEE.STD_LOGIC_UNSIGNED.all;
use ieee.numeric_std.all; -- fournit le calcul numérique tableaux signés non signés
de type std_logic pour les signaux--
```

```
entity state_machine_stepper_motor is
```

```
Port (
    clk : in STD_LOGIC; --255Hz--
    bouton_UP : in STD_LOGIC;
    bouton_DOWN : in STD_LOGIC;
    change_frequency_motor : in STD_LOGIC_VECTOR (7 downto 0);
    Commandes_demi_pas : out STD_LOGIC_VECTOR (3 downto 0);
    vers_change_frequency_motor : out STD_LOGIC_VECTOR (7 downto 0);
    reset : in STD_LOGIC;
    visu_UP : out STD_LOGIC;
    visu_DOWN : out STD_LOGIC);
```

```
end state_machine_stepper_motor;
```

Déclaration des  
Entrées-sorties

```
architecture Behavioral of state_machine_stepper_motor is
    TYPE etat IS (attente, position1, position2, position3, position4, position5, position6,
    position7, position8);
    SIGNAL state_machine: etat;
    SIGNAL count4 : INTEGER range 0 to 255 := 0; --8 bits compteur--
    SIGNAL clock_int4: STD_LOGIC := '0';
    signal M : INTEGER range 0 to 255;
```

```
begin
```

```
--Divise par M en fonction de change_frequency_motor--
```

```
PROCESS(clk,M,reset,change_frequency_motor)
```

```
BEGIN
```

```
if reset='1' then
```

```
    clock_int4 <= '0';
```

```
    count4 <= 0;
```

```
elsif rising_edge(clk) then
```

```
    M <= 255/conv_integer(change_frequency_motor);
```

```
        IF count4 <= M-1 THEN
```

```
            count4 <= count4 + 1;
```

```
        ELSE
```

```
            count4 <= 0;
```

```
        END IF;
```

```
--à la moitié du comptage on change la valeur de clock_1Hz_int (rapport cyclique = 1/2)-
```

```
        IF count4 <= M/2 THEN
```

```
            clock_int4 <= '0';
```

```
        ELSE
```

```
            clock_int4 <= '1';
```

```
        END IF;
```

```
end if;
```

```
end process;
```

Process  
Diviseur  
d'horloge

La fréquence du moteur = f (change\_frequency\_motor)

Fabrice Wiotte LPL

## Code VHDL machine d'état moteur pas à pas

--state machine moteur pas a pas--

process(clock\_int4,Reset,bouton\_UP,bouton\_DOWN)

begin

if reset='1' then

state\_machine <= attente;

Commandes\_demi\_pas <="0000";

elsif rising\_edge(clock\_int4) then -- sur front montant de clock\_int4--

case state\_machine is

when attente => Commandes\_demi\_pas <="0000"; --0--

if bouton\_UP = '1' and bouton\_DOWN='0' then

state\_machine <= position1;

elsif bouton\_UP = '0' and bouton\_DOWN='1' then

state\_machine <= position8;

else

state\_machine <= attente;

end if;

--UP & DOWN--

when position1 => Commandes\_demi\_pas <="0001"; --1--

if bouton\_UP = '1' and bouton\_DOWN='0' then

state\_machine <= position2;

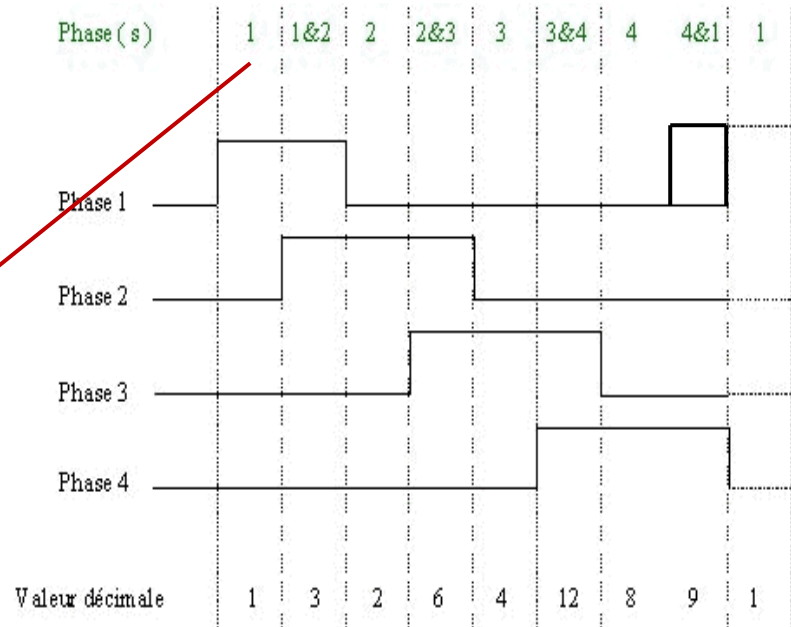
elsif bouton\_UP = '0' and bouton\_DOWN='1' then

state\_machine <= position8; --9--

else

state\_machine <= attente;

end if;



Fabrice Wiotte LPL



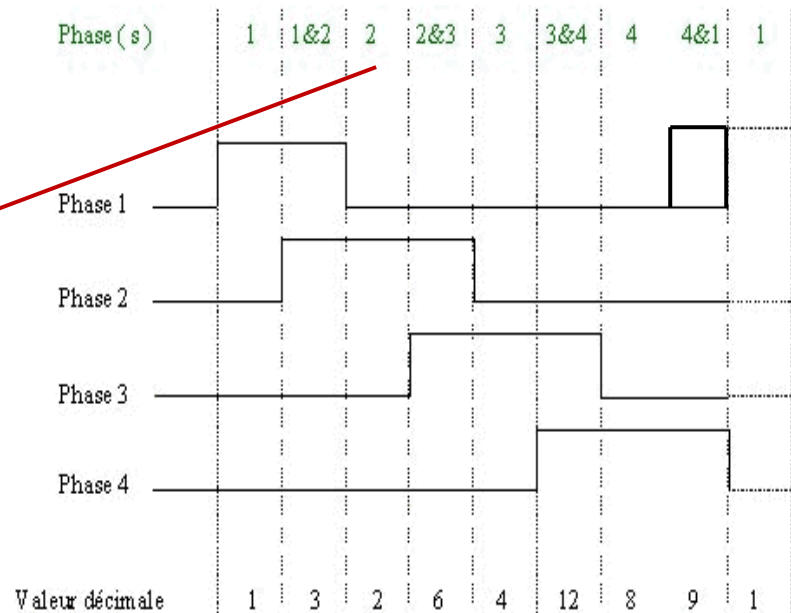
## Code VHDL machine d'état moteur pas à pas

--state machine moteur pas a pas--

```

--UP & DOWN--
when position2 => Commandes_demi_pas <="0011"; --3--
  if bouton_UP = '1' and bouton_DOWN='0' then
    state_machine <= position3;
  elsif bouton_UP = '0' and bouton_DOWN='1' then
    state_machine <= position8; --1--
  else
    state_machine <= attente;
  end if;
when position3 => Commandes_demi_pas <="0010"; --2--
  if bouton_UP = '1' and bouton_DOWN='0' then
    state_machine <= position4;
  elsif bouton_UP = '0' and bouton_DOWN='1' then
    state_machine <= position2; --3--
  else
    state_machine <= attente;
  end if;
when position4 => Commandes_demi_pas <="0110"; --6--
  if bouton_UP = '1' and bouton_DOWN='0' then
    state_machine <= position5;
  elsif bouton_UP = '0' and bouton_DOWN='1' then
    state_machine <= position2; --2--
  else
    state_machine <= attente;
  end if;

```



Fabrice Wiotte LPL

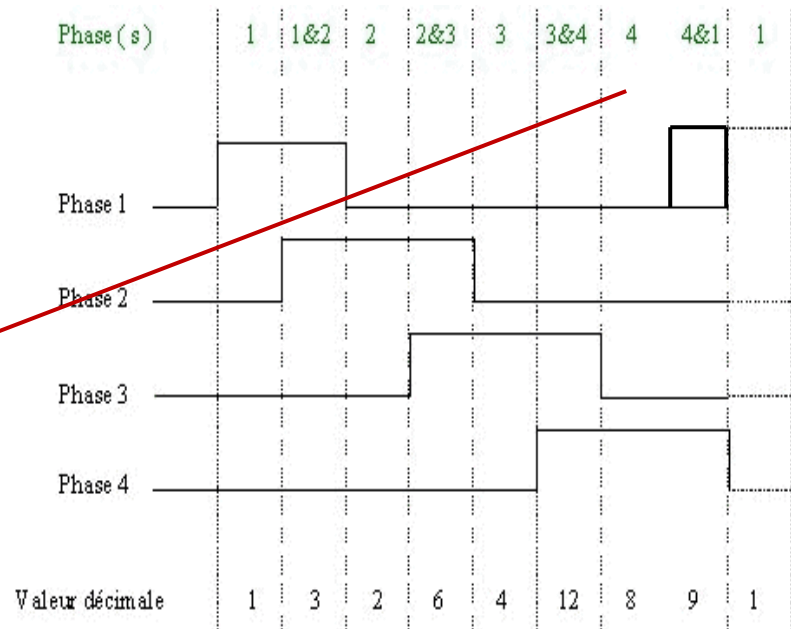
## Code VHDL machine d'état moteur pas à pas

--state machine moteur pas a pas--

```

--UP & DOWN--
when position5 => Commandes_demi_pas <="0011"; --4--
    if bouton_UP = '1' and bouton_DOWN='0' then
        state_machine <= position6;
    elsif bouton_UP = '0' and bouton_DOWN='1' then
        state_machine <= position4; --6--
    else
        state_machine <= attente;
    end if;
when position6 => Commandes_demi_pas <=" 1100"; --12--
    if bouton_UP = '1' and bouton_DOWN='0' then
        state_machine <= position7;
    elsif bouton_UP = '0' and bouton_DOWN='1' then
        state_machine <= position5; --4--
    else
        state_machine <= attente;
    end if;
when position7 => Commandes_demi_pas <=" 1000"; --8--
    if bouton_UP = '1' and bouton_DOWN='0' then
        state_machine <= position8;
    elsif bouton_UP = '0' and bouton_DOWN='1' then
        state_machine <= position6; --12--
    else
        state_machine <= attente;
    end if;

```



Fabrice Wiotte LPL

## Code VHDL machine d'état moteur pas à pas

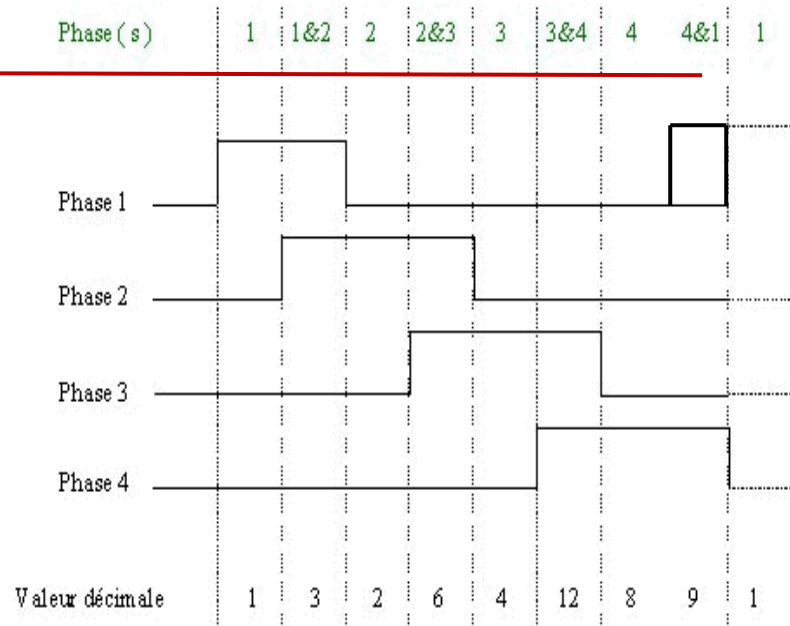
--state machine moteur pas a pas--

```

--UP & DOWN--
when position8 => Commandes_demi_pas <= « 1001"; --9--
    if bouton_UP = '1' and bouton_DOWN='0' then
        state_machine <= position1;
    elsif bouton_UP = '0' and bouton_DOWN='1' then
        state_machine <= position7; --8--
    else
        state_machine <= attente;
    end if;
End case;
end if;
end process;
vers_change_frequency_motor <= std_logic_vector(change_frequency_motor);
visu_UP <= bouton_UP;
visu_DOWN <= bouton_DOWN;

end Behavioral;

```



## Module VHDL UART liaison série

Un UART est une interface qui envoie généralement un octet à la fois sur un seul fil. Il ne transmet pas le long d'une horloge avec les données, c'est pourquoi il est appelé *asynchrone* par opposition à synchrone. Les UART peuvent fonctionner en semi-duplex (deux émetteurs partageant une ligne) ou en duplex intégral (deux émetteurs chacun avec leur propre ligne). Les UART ont plusieurs paramètres qui peuvent être définis par l'utilisateur. Ceux-ci sont :

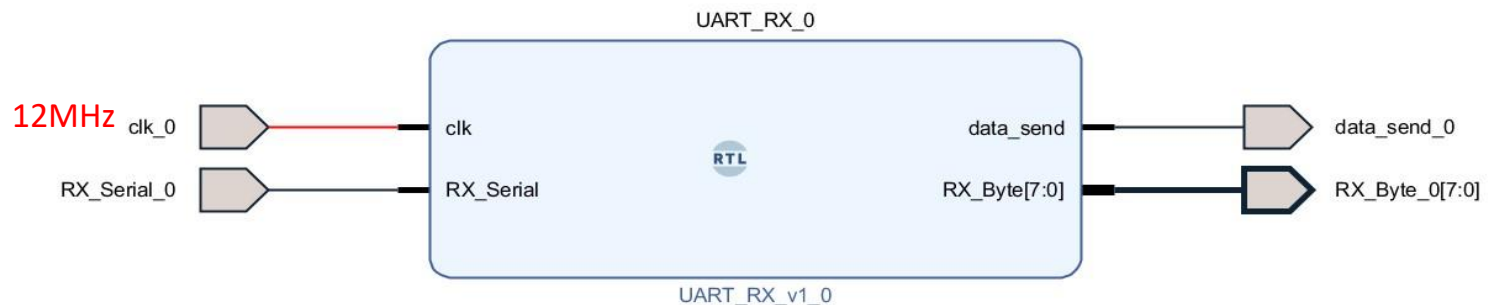
Débit en bauds (*9600*, 19200, 115200, autres)

Nombre de bits de données (7, *8*)

Bit de parité (activé, *désactivé*)

Bits d'arrêt (0, 1, 2)

Contrôle de flux (*aucun*, activé, matériel)



Machine d'état pour récupérer les données séries d'un PC/d'une Raspberry Pi/d'un uC en fonction du Baud Rate

Fabrice Wiotte LPL

--Code UART liaison Série asynchrone code complet--  
 -- This file contains the UART Receiver and UART Transmitter.  
 --This receiver is able to receive 8 bits of serial data, one start bit, one stop bit, and no parity bit. --  
 When receive is complete **data\_send** will be driven high for one clock cycle.  
 -- CLKS\_PER\_BIT = (Frequency of Clk) / (Frequency of UART) 100 MHz Clock, 9600 baud UART--  

$$(12000000) / (9600) = 1250$$

```
Library ieee;
use ieee.std_logic_1164.ALL;
use ieee.numeric_std.all;
```

```
entity UART_RX is
  generic (
    CLKS_PER_BIT : integer := 1250 -- pour 12MHz d'horloge et 9600 bps
  );
  port (
    clk      : in std_logic;
    RX_Serial : in std_logic;
    data_send : out std_logic;
    RX_Byte   : out std_logic_vector(7 downto 0)
  );
end UART_RX;
```

```
architecture Behavioral of UART_RX is
```

```
type etat is (attente, Start_Bit, RX_Data_Bits, Stop_Bit, s_Cleanup);
```

```
signal state_machine : etat;
```

```
signal r_RX_Data_R : std_logic := '0';
signal r_RX_Data   : std_logic := '0';
signal Clk_Count   : integer range 0 to CLKS_PER_BIT-1 := 0;
signal r_Bit_Index : integer range 0 to 7 := 0; -- 8 Bits Total
signal r_RX_Byte   : std_logic_vector(7 downto 0) := (others => '0');
signal r_data_send : std_logic := '0';
```

```
Begin
```

```
-- Purpose: Double-register the incoming data.
-- This allows it to be used in the UART RX Clock Domain.
-- (It removes problems caused by metastability)
process (clk)
begin
  if rising_edge(clk) then
    r_RX_Data_R <= RX_Serial;
    r_RX_Data   <= r_RX_Data_R;
  end if;
end process;
```

Déclaration  
des  
Signaux sous  
architecture

Déclaration des  
Entrées-sorties

Fabrice Wiotte LPL

--Code UART liaison Série asynchrone code complet--  
 -- This file contains the UART Receiver and UART Transmitter.  
 --This receiver is able to receive 8 bits of serial data, one start bit, one stop bit, and no parity bit. --  
 When receive is complete **data\_send** will be driven high for one clock cycle.  
 -- CLK\_PER\_BIT = (Frequency of Clk) / (Frequency of UART) 100 MHz Clock, 9600 baud UART--  

$$(12000000) / (9600) = 1250$$

-- Purpose: Control RX state machine

```
process(clk)
begin
  if rising_edge(clk) then

    case state_machine is

      when attente =>
        r_data_send <= '0';
        Clk_Count <= 0;
        r_Bit_Index <= 0;

        if r_RX_Data = '0' then -- Start bit detected
          state_machine <= Start_Bit;
        else
          state_machine <= attente;
        end if;

    end case;

  end if;
end process;
```

-- Check middle of start bit to make sure it's still low

-- Check middle of start bit to make sure it's still low

```
when Start_Bit =>
  if Clk_Count = (CLKS_PER_BIT-1)/2 then
    if r_RX_Data = '0' then
      Clk_Count <= 0; -- reset counter since we found the middle
      state_machine <= RX_Data_Bits;
    else
      state_machine <= attente;
    end if;
  else
    Clk_Count <= Clk_Count + 1;
    state_machine <= Start_Bit;
  end if;
end when;
```

-- Wait CLK\_PER\_BIT-1 clock cycles to sample serial data

```
when RX_Data_Bits =>
  if Clk_Count < CLK_PER_BIT-1 then
    Clk_Count <= Clk_Count + 1;
    state_machine <= RX_Data_Bits;
  else
    Clk_Count <= 0;
    r_RX_Byte(r_Bit_Index) <= r_RX_Data;
    state_machine <= attente;
  end if;
end when;
```

Fabrice Wiotte LPL

--Code UART liaison Série asynchrone code complet--

-- This file contains the UART Receiver and UART Transmitter.

--This receiver is able to receive 8 bits of serial data, one start bit, one stop bit, and no parity bit. --

When receive is complete **data\_send** will be driven high for one clock cycle.

-- CLKS\_PER\_BIT = (Frequency of Clk) / (Frequency of UART) 100 MHz Clock, 9600 baud UART--  
(12000000) / (9600) = **1250**

-- Check if we have sent out all bits

```
if r_Bit_Index < 7 then
  r_Bit_Index <= r_Bit_Index + 1;
  state_machine <= RX_Data_Bits;
else
  r_Bit_Index <= 0;
  state_machine <= Stop_Bit;
end if;
end if;
```

-- Receive Stop bit. Stop bit = 1

```
when Stop_Bit =>
  -- Wait CLKS_PER_BIT-1 clock cycles for Stop bit to finish
  if Clk_Count < CLKS_PER_BIT-1 then
    Clk_Count <= Clk_Count + 1;
    state_machine <= Stop_Bit;
  else
    r_data_send <= '1';
    Clk_Count <= 0;
    state_machine <= s_Cleanup;
  end if;
```

-- Stay here 1 clock

```
when s_Cleanup =>
  state_machine <= attente;
  r_data_send <= '0';
```

```
when others =>
  state_machine <= attente;
```

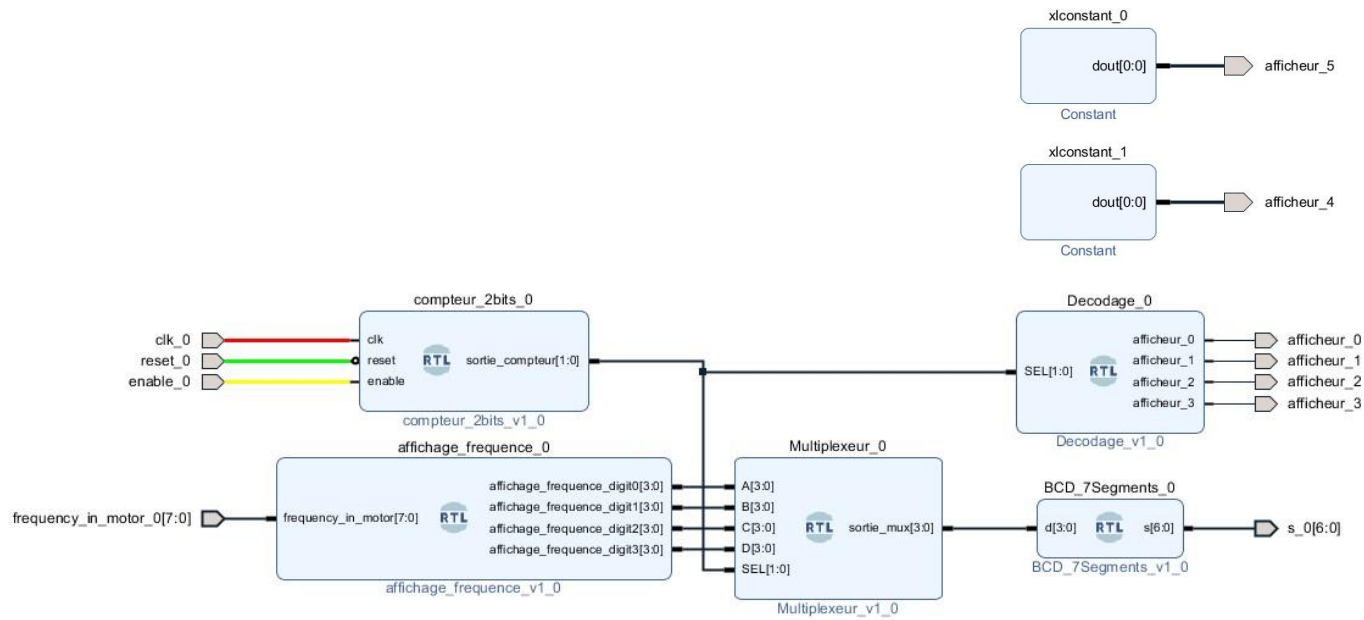
```
end case;
end if;
end process;
```

```
data_send <= r_data_send;
RX_Byte <= r_RX_Byte;
```

end Behavioral;

Fabrice Wiotte LPL

## Modules VHDL pour multiplexer et afficher la fréquence du moteur sur LED 7 segments



X 6

On n'utilisera que 4 afficheurs pour ce TP

Fabrice Wiotte LPL



## Codes VHDL pour multiplexer et afficher la fréquence du moteur sur LED 7 segments

### --Code Mux 4 to 1--

```
-- Déclaration des bibliothèques utilisées
library IEEE;
use IEEE.STD_LOGIC_1164.ALL; -- inclus signaux améliorés du type std_ulogic--

entity MUX is
  Port (
    A : in STD_LOGIC_VECTOR (3 downto 0);
    B : in STD_LOGIC_VECTOR (3 downto 0);
    C : in STD_LOGIC_VECTOR (3 downto 0);
    D : in STD_LOGIC_VECTOR (3 downto 0);
    SEL : in STD_LOGIC_VECTOR (1 downto 0);
    sortie_mux : out STD_LOGIC_VECTOR (3 downto 0));
end MUX;
```

Déclaration des  
Entrées-sorties

### architecture Behavioral of MUX is

```
begin
  process(sel,A,B,C,D)
  begin
    case sel is
      when "00" => Sortie_Mux <= A;
      when "01" => Sortie_Mux <= B;
      when "10" => Sortie_Mux <= C;
      -- when "11" => Sortie_Mux <= D;
      when others => Sortie_Mux <= D;
    end case;
  end process;
end Behavioral;
```

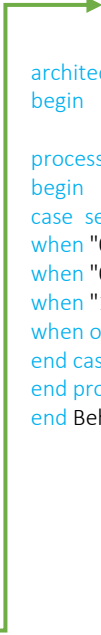
Process  
combinatoire

## Codes VHDL pour multiplexer et afficher la fréquence du moteur sur LED 7 segments

### *--Code decode 2 to 4 --*

```
-- Déclaration des bibliothèques utilisées
library IEEE;
use IEEE.STD_LOGIC_1164.ALL; -- inclus signaux améliorés du type std_ulogic--
```

```
entity DECODE_2_to_4 is
  Port (
    SEL : in STD_LOGIC_VECTOR (2 downto 0);
    DP1 : out STD_LOGIC;
    afficheur_0 : out STD_LOGIC;
    afficheur_1 : out STD_LOGIC;
    afficheur_2 : out STD_LOGIC;
    afficheur_3 : out STD_LOGIC;
    afficheur_4 : out STD_LOGIC;
    afficheur_5 : out STD_LOGIC);
end DECODE_2_to_8;
```



```
architecture Behavioral of DECODE_2_to_4 is
begin

  process(Sel)
  begin
    case sel is
      when "00" => afficheur_0 <='0'; afficheur_1 <='1'; afficheur_2 <='1'; afficheur_3 <='1';
      when "01" => afficheur_0 <='1'; afficheur_1 <='0'; afficheur_2 <='1'; afficheur_3 <='1';
      when "10" => afficheur_0 <='1'; afficheur_1 <='1'; afficheur_2 <='0'; afficheur_3 <='1';
      when others => afficheur_0 <='1'; afficheur_1 <='1'; afficheur_2 <='1'; afficheur_3 <='0';
    end case;
  end process;
end Behavioral;
```

Fabrice Wiotte LPL

## Codes VHDL pour multiplexer et afficher la fréquence du moteur sur LED 7 segments

### --Code BCD 7 segments --

Déclaration  
des  
Entrées-sorties

```
-- Déclaration des bibliothèques utilisées
library IEEE;
use IEEE.STD_LOGIC_1164.ALL; -- inclus signaux améliorés du type std_ulogic--
entity BCD_7Segments is
  Port (
    d : in STD_LOGIC_VECTOR (3 downto 0);
    s : out STD_LOGIC_VECTOR (6 downto 0));
end BCD_7Segments;

architecture Behavioral of BCD_7Segments is

  -- segment encoding
  -- 0
  -- ---
  -- 5 | | 1
  -- --- <- 6
  -- 4 | | 2
  -- ---
  -- 3
```

4 entrées correspondent à la représentation binaire d'un chiffre entre 0 et 15.

```
begin
  with d select
```

```
s<="1111001" when "0001" --1
    "0100100" when "0010", --2
    "0110000" when "0011", --3
    "0011001" when "0100", --4
    "0010010" when "0101", --5
    "0000010" when "0110", --6
    "1111000" when "0111", --7
    "0000000" when "1000", --8
    "0010000" when "1001", --9
    "0001000" when "1010", --A
    "0000011" when "1011", --B
    "1000110" when "1100", --C
    "0100001" when "1101", --d
    "0000110" when "1110", --E
    "0001110" when "1111", --F
    "1000000" when others; --0
```

```
end Behavioral;
```

Table de vérité

Fabrice Wiotte LPL

## Codes VHDL pour multiplexer et afficher la fréquence du moteur sur LED 7 segments

### --Code compteur 2 bits--

```
-- Déclaration des bibliothèques utilisées
library IEEE;
use ieee.std_logic_1164.all; -- inclus signaux améliorés du type std_ulogic--
use ieee.std_logic_arith.all; -- fournit le calcul numérique--use
ieee.std_logic_unsigned.all; -- calcul numérique non signé sur le type
std_logic_vector--

entity compteur_2bits is
  Port ( clk : in STD_LOGIC;
        reset : in STD_LOGIC;
        enable : in STD_LOGIC;
        sortie_compteur : out STD_LOGIC_VECTOR (1 downto 0)); --2 bits si on
        affiche sur 4 digits--
end compteur_3bits;
```

Déclaration  
des  
Entrées-sorties

```
architecture Behavioral of compteur_3bits is
  signal compte : std_logic_vector(1 downto 0); -- on declare un signal de comptage--
begin

  --compteur 2 bits on compte au max jusqu'à 2^2 max--
  process(CLK,reset)
  BEGIN
    if reset ='1' then
      compte <= "00";
    elsif rising_edge(CLK) then
      if enable ='1' then
        compte <= compte + 1;
      end if;
    end if;
  END PROCESS;

  Sortie_compteur <= compte;
end Behavioral;
```

Compteur simple

On affecte le signal compte dans sortie\_compteur

## Codes VHDL pour multiplexer et afficher la fréquence du moteur sur LED 7 segments

--code affichage fréquence --

**E/S**

```

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 affichage_frequence is
    Port ( frequency_in_motor : in STD_LOGIC_VECTOR (7 downto 0);
          affichage_frequence_digit0: out STD_LOGIC_VECTOR (3 downto 0);
          affichage_frequence_digit1: out STD_LOGIC_VECTOR (3 downto 0);
          affichage_frequence_digit2: out STD_LOGIC_VECTOR (3 downto 0);
          affichage_frequence_digit3: out STD_LOGIC_VECTOR (3 downto 0));
end affichage_frequence;

architecture Behavioral of affichage_frequence is
    signal Q : STD_LOGIC_VECTOR (7 downto 0);
begin

    Q <= frequency_in_motor;

    process(Q)
        -- variable temporaire
        variable temp : STD_LOGIC_VECTOR ( 7 downto 0);
        variable bcd : STD_LOGIC_VECTOR ( 15 downto 0);
    begin
    
```

```

-- 2 4 3
-- 0010 0100 0011

```

```

--
--<-----ORIGINAL
--000 0000 0000 11110011
--Le double dabble est un algorithme utilisé pour convertir des nombres d'un système--
--binaire vers un système décimal. Pour des raisons pratiques, le résultat est--
--généralement stocké sous la forme de décimal codé en binaire (BCD)--
--En partant du registre initial, l'algorithme effectue n itérations (soit 8 dans l'exemple)
--a chaque itération, le registre est décalé d'un bit vers la gauche. Avant d'effectuer cette
opération,
--la partie au format BCD est analysée, décimale par décimale. Si une décimale en BCD (4 bits)
--est plus grande que 4 alors on lui ajoute 3. Cet incrément permet de s'assurer qu'une valeur
de 5,
--après incrémentation et décalage, devient 16 et se propage correctement à la décimale
suivante.
--- 0000 0000 0000 11110011 Initialisation
--- 0000 0000 0001 11100110 Décalage
--- 0000 0000 0011 11001100 Décalage
--- 0000 0000 0111 10011000 Décalage
--- 0000 0000 1010 10011000 Ajouter 3 à la première décimale BCD, puisque sa
valeur était 7
--- 0000 0001 0101 00110000 Décalage
--- 0000 0001 1000 00110000 Ajouter 3 à la première décimale BCD, puisque sa
valeur était 5
--- 0000 0011 0000 01100000 Décalage
--- 0000 0110 0000 11000000 Décalage
--- 0000 1001 0000 11000000 Ajouter 3 à la seconde décimale BCD, puisque sa
valeur était 6
--- 0001 0010 0001 10000000 Décalage
--- 0010 0100 0011 00000000 Décalage
--2---4---3--

```

Fabrice Wiotte LPL

## Codes VHDL pour multiplexer et afficher la fréquence du moteur sur LED 7 segments

### --code affichage fréquence --

--mettre à zéro la variable bcd

```
bcd := (others => '0');
```

```
temp(7 downto 0) := Q;
```

```
for i in 0 to 7 loop
```

```
    if bcd(3 downto 0) > 4 then
```

```
        bcd(3 downto 0) := bcd(3 downto 0) + 3;
```

```
    end if;
```

```
    if bcd(7 downto 4) > 4 then
```

```
        bcd(7 downto 4) := bcd(7 downto 4) + 3;
```

```
    end if;
```

```
    if bcd(11 downto 8) > 4 then
```

```
        bcd(11 downto 8) := bcd(11 downto 8) + 3;
```

```
    end if;
```

```
    if bcd(11 downto 8) > 4 then
```

```
        bcd(11 downto 8) := bcd(11 downto 8) + 3;
```

```
    end if;
```

-- thousands can't be >4 for a 12-bit input number  
-- so don't need to do anything to upper 4 bits of bcd

-- shift bcd left by 1 bit, copy MSB of temp into LSB of bcd  
bcd := bcd(14 downto 0) & temp(7);

-- shift temp left by 1 bit  
temp := temp(6 downto 0) & '0';  
end loop;

-- set outputs

```
affichage_frequence_digit0 <= STD_LOGIC_VECTOR (bcd(3 downto 0));  
affichage_frequence_digit1 <= STD_LOGIC_VECTOR (bcd(7 downto 4));  
affichage_frequence_digit2 <= STD_LOGIC_VECTOR (bcd(11 downto 8));  
affichage_frequence_digit3 <= STD_LOGIC_VECTOR (bcd(15 downto 12));
```

```
end process;  
end Behavioral;
```

Fabrice Wiotte LPL

## Codes VHDL compteur BCD

--le compteur BCD permet d'afficher la valeur du pas 0 to 4096 sur 4 digits--

--Code compteur BCD complet--

-- Déclaration des bibliothèques utilisées  
library IEEE;  
use ieee.std\_logic\_1164.all; -- inclus signaux améliorés du type std\_ulogic--  
use ieee.std\_logic\_arith.all; -- fournit le calcul numérique--use  
ieee.std\_logic\_unsigned.all;  
--le compteur BCD permet d'afficher la valeur du pas sur 4 digits--

entity CompteurBCD is

Port (

CLK : in STD\_LOGIC; --255Hz  
change\_frequency\_motor : in STD\_LOGIC\_VECTOR (7 downto 0);  
bouton\_UP : in STD\_LOGIC;  
bouton\_DOWN : in STD\_LOGIC;  
Enable : in STD\_LOGIC;  
Reset : in STD\_LOGIC;  
Full : out STD\_LOGIC;  
Empty : out STD\_LOGIC;  
BCD\_U : out STD\_LOGIC\_VECTOR (3 downto 0);  
BCD\_D : out STD\_LOGIC\_VECTOR (3 downto 0);  
BCD\_H : out STD\_LOGIC\_VECTOR (3 downto 0);  
BCD\_T : out STD\_LOGIC\_VECTOR (3 downto 0);  
);

end CompteurBCD;

architecture Behavioral of CompteurBCD is

SIGNAL count3 : INTEGER range 0 to 255 := 0; --8 bits compteur  
SIGNAL clock\_int3: STD\_LOGIC;  
signal M : INTEGER range 0 to 255;

signal COUNTER\_U: INTEGER range 0 to 9;  
signal COUNTER\_D: INTEGER range 0 to 9;  
signal COUNTER\_H: INTEGER range 0 to 9;  
signal COUNTER\_T: INTEGER range 0 to 9;  
signal IS\_4096: STD\_LOGIC;  
signal IS\_0000: STD\_LOGIC;

Begin

Fabrice Wiotte LPL

## Codes VHDL compteur BCD

--le compteur BCD permet d'afficher la valeur du pas 0 to 4096 sur 4 digits--

### --Code compteur BCD complet--

--Divise par M en fonction de change\_frequency\_motor on modifie la Vitesse du compteur--

```
PROCESS(clk,reset,Enable,M,change_frequency_motor)
```

```
BEGIN
```

```
if reset='1' then
```

```
    count3 <= 0;
```

```
elsif rising_edge(clk) then
```

```
if enable='1' then
```

```
    M <= 255/conv_integer(change_frequency_motor);
```

```
        IF count3 <= M-1 THEN
```

```
count3 <= count3 + 1;
```

```
ELSE
```

```
    count3 <= 0;
```

```
END IF;
```

--à la moitié du comptage on change la valeur de clock\_1Hz\_int (rapport cyclique = 1/2)--

```
IF count3 <= M/2 THEN
```

```
    clock_int3 <= '0';
```

```
ELSE
```

```
    clock_int3 <= '1';
```

```
END IF;
```

```
end if;
```

```
end if;
```

```
END PROCESS;
```

Process division d'horloge

--le compteur BCD permet d'afficher la valeur du pas 0 to 4096 sur 4 digits--

```
process(clock_int3, Enable,reset,bouton_UP,bouton_DOWN)
```

```
begin
```

```
if Reset='1' then --on initialize le compteur au démarrage--
```

```
    COUNTER_U <= 0;
```

```
    COUNTER_D <= 0;
```

```
    COUNTER_H <= 0;
```

```
    COUNTER_T <= 0;
```

```
elsif rising_edge(clock_int3) then
```

```
if Enable = '1' then
```

```
    if bouton_UP='1' then
```

```
        if IS_4096 = '1' then
```

```
            COUNTER_U <= 0;
```

```
            COUNTER_D <= 0;
```

```
            COUNTER_H <= 0;
```

```
            COUNTER_T <= 0;
```

```
        elsif IS_4096 = '0' then
```

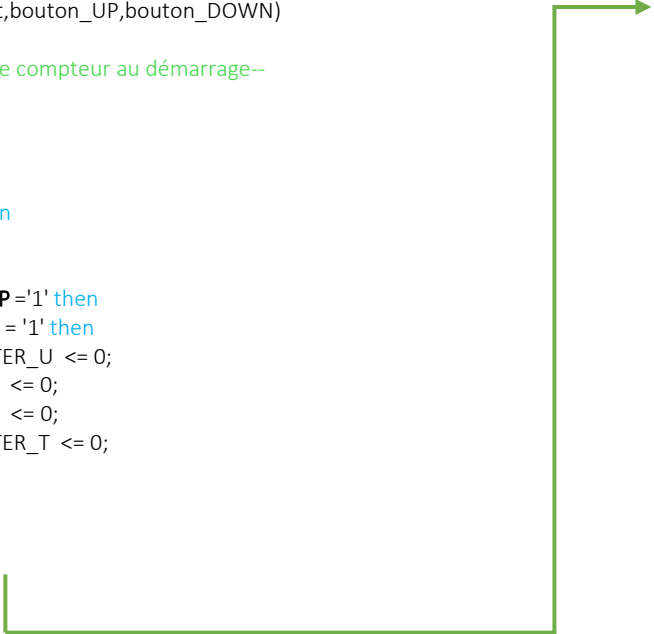


## Codes VHDL compteur BCD

--le compteur BCD permet d'afficher la valeur du pas 0 to 4096 sur 4 digits--

```
process(clock_int3, Enable,reset,bouton_UP,bouton_DOWN)
begin
if Reset='1' then --on initialize le compteur au démarrage--
    COUNTER_U <= 0;
    COUNTER_D <= 0;
    COUNTER_H <= 0;
    COUNTER_T <= 0;
elsif rising_edge(clock_int3) then

    if Enable = '1' then
        if bouton_UP='1' then
            if IS_4096 = '1' then
                COUNTER_U <= 0;
                COUNTER_D <= 0;
                COUNTER_H <= 0;
                COUNTER_T <= 0;
            elsif IS_4096 = '0' then
```



```
        if COUNTER_U = 9 then
            COUNTER_U <= 0;
            if COUNTER_D = 9 then
                COUNTER_D <= 0;
                if COUNTER_H = 9 then
                    COUNTER_H <= 0;
                    if COUNTER_T = 9 then
                        COUNTER_T <= 0;
                    else
                        COUNTER_T <= COUNTER_T + 1;
                    end if;
                else
                    COUNTER_H <= COUNTER_H + 1;
                end if;
            else
                COUNTER_D <= COUNTER_D + 1;
            end if;
        else
            COUNTER_U <= COUNTER_U + 1;
        end if;
    end if;
end if;
```

## Codes VHDL compteur BCD

--le compteur BCD permet d'afficher la valeur du pas 0 to 4096 sur 4 digits--

```
if bouton_DOWN='1' then
```

```
  if IS_0000='0' then
```

```
    if COUNTER_U = 0 then
```

```
      COUNTER_U <= 9;
```

```
    if COUNTER_D = 0 then
```

```
      COUNTER_D <= 9;
```

```
    if COUNTER_H = 0 then
```

```
      COUNTER_H <= 9;
```

```
    if COUNTER_T = 0 then
```

```
      COUNTER_T <= 9;
```

```
    else
```

```
      COUNTER_T <= COUNTER_T - 1;
```

```
    end if;
```

```
    else
```

```
      COUNTER_H <= COUNTER_H - 1;
```

```
    end if;
```

```
    else
```

```
      COUNTER_D <= COUNTER_D - 1;
```

```
    end if;
```

```
    else
```

```
      COUNTER_U <= COUNTER_U - 1;
```

```
    end if;
```

```
  end if;
```

```
end if;
```

```
end if;
```

```
end if;
```

```
end process;
```

```
BCD_U <= CONV_STD_LOGIC_VECTOR(COUNTER_U,4);
```

```
BCD_D <= CONV_STD_LOGIC_VECTOR(COUNTER_D,4);
```

```
BCD_H <= CONV_STD_LOGIC_VECTOR(COUNTER_H,4);
```

```
BCD_T <= CONV_STD_LOGIC_VECTOR(COUNTER_T,4);
```

```
-- on defini les limites du compteur ici 4096 pas--
```

```
IS_4096 <= '1' when (COUNTER_U = 6 and COUNTER_D = 9 and COUNTER_H = 0 and  
COUNTER_T = 4) else '0';
```

```
IS_0000 <= '1' when (COUNTER_U = 0 and COUNTER_D = 0 and COUNTER_H = 0 and  
COUNTER_T = 0) else '0';
```

```
Full <= IS_4096;
```

```
Empty <= IS_0000;
```

```
end Behavioral;
```

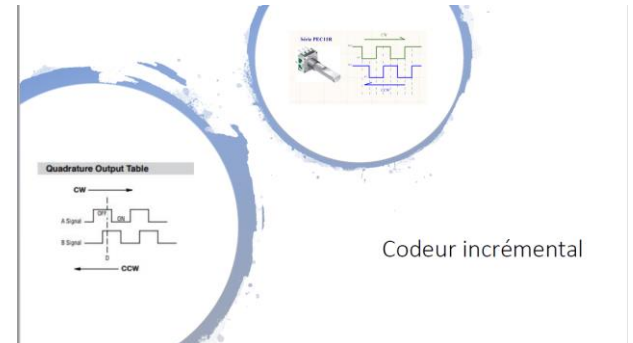
Codage décimal	Codage binaire naturel	Codage Gray ou binaire réfléchi
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100

## Module VHDL encodeur numérique

Basé sur Le code de Gray, également appelé binaire réfléchi, permet de ne faire changer qu'un seul bit à la fois quand un nombre est incrémenté ou décrémenté d'une unité.

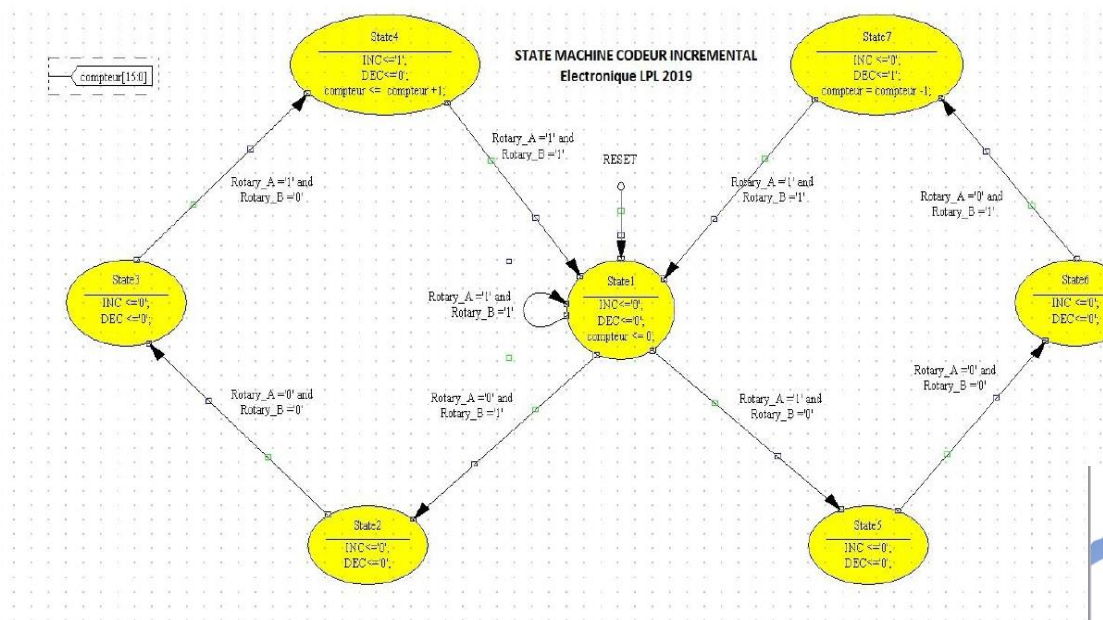


On incrémente localement la fréquence du moteur

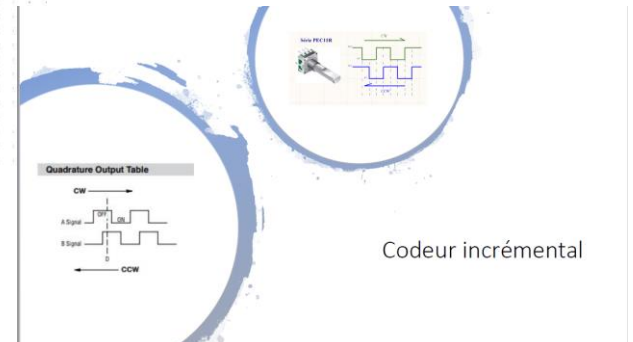


Fabrice Wiotte LPL

## Module VHDL encodeur numérique



Machine d'état répondant au fonctionnement de l'encodeur



Fabrice Wiotte LPL

## Code VHDL encodeur numérique

### --Code codeur numérique--

-- Déclaration des bibliothèques utilisées

library IEEE;

use IEEE.STD\_LOGIC\_1164.all; -- inclus signaux améliorés du type std\_ulogic--

use IEEE.STD\_LOGIC\_ARITH.all; -- fournit le calcul numérique--

use IEEE.STD\_LOGIC\_UNSIGNED.all; -- calcul numérique non signé sur le type std\_logic\_vector--

entity codeur\_numerique is

Port (  
 CLK : in STD\_LOGIC; --255Hz--  
 Reset : in STD\_LOGIC;  
 rotary\_A : in STD\_LOGIC;  
 rotary\_B : in STD\_LOGIC;  
 compte\_out\_codeur : out STD\_LOGIC\_VECTOR(7 downto 0));

end codeur\_numerique;

architecture Behavioral of codeur\_numerique is

type etat\_codeur is(S1,S2,S3,S4,S5,S6,S7); -- déclaration de la machine d'état et du nombre d'état--

signal etat : etat\_codeur;

signal compteur: INTEGER range 0 to 255;

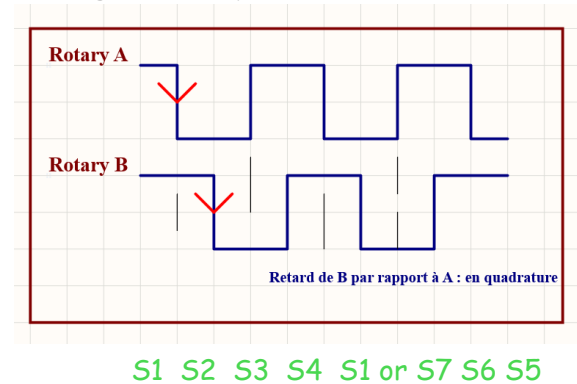
-- déclaration d'un compteur 8 bits qui servira à modifier la fréquence

--de fonctionnement du moteur--

-- process machine d'état encodeur numérique--

begin

### Chronogramme pour CW → or CCW ←



Machine d'état répondant au fonctionnement due l'encodeur

Fabrice Wiotte LPL

## Code VHDL encodeur numérique

### --Code codeur numérique--

```

process(CLK,reset,rotary_A,rotary_B) -- liste de sensibilité toujours des entrées ou des
signaux déclarés--
begin
if reset ='1' then

    compte_out_codeur <="00000000"; -- on s'assure que le compteur
de sortie soit à zéro à l'initialisation --
    compteur <= 0; -- compteur interne à zéro à l'initialisation--
    etat <= S1; -- on va à l'état S1 --

elsif CLK'event and CLK ='1' then -- sur front montant d'horloge--

case etat is -- machine d'état on décrit tous les cas possibles--

when S1=> if rotary_A ='1' and rotary_B = '1' then -- si pas d'action on reste en S1--
            etat <= S1;
        elsif rotary_A ='0' and rotary_B = '1' then -- si rotation CW --
            etat <= S2;
        elsif rotary_A ='1' and rotary_B = '0' then -- si rotation CCW --
            etat <= S5;
        end if;

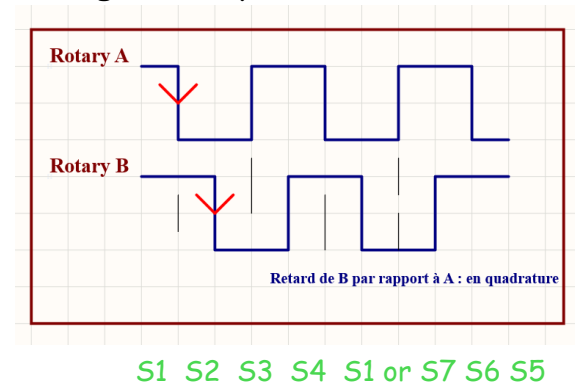
when S2=> if rotary_A ='0' and rotary_B = '0' then
            etat <= S3;
        end if;

when S3=> if rotary_A ='1' and rotary_B = '0' then
            etat <= S4;
        end if;

```

Machine d'état répondant au fonctionnement de l'encodeur

### Chronogramme pour CW → or CCW ←



## Code VHDL encodeur numérique

### --Code codeur numérique--

```
when S4=> if rotary_A='1' and rotary_B='1' then
    compteur <= compteur + 1;
    etat <= S1;
end if;

when S5=> if rotary_A='0' and rotary_B='0' then
    etat <= S6;
end if;

when S6=> if rotary_A='0' and rotary_B='1' then
    etat <= S7;
end if;

when S7=> if rotary_A='1' and rotary_B='1' then
    compteur <= compteur - 1;
    etat <= S1;
end if;

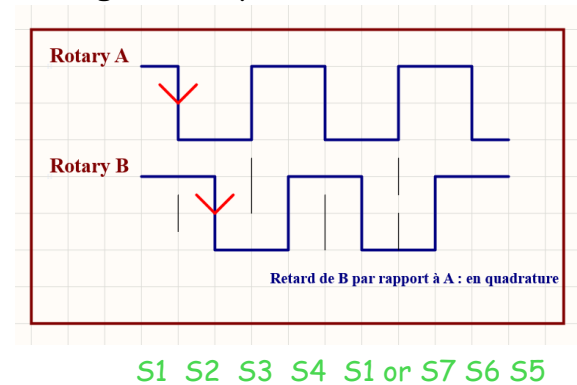
end case;
compte_out_codeur <= CONV_STD_LOGIC_VECTOR(compteur,8);
-- on convertit des entiers en std_logic_vectors, bus logic.
end if;
end process;

end Behavioral;
```

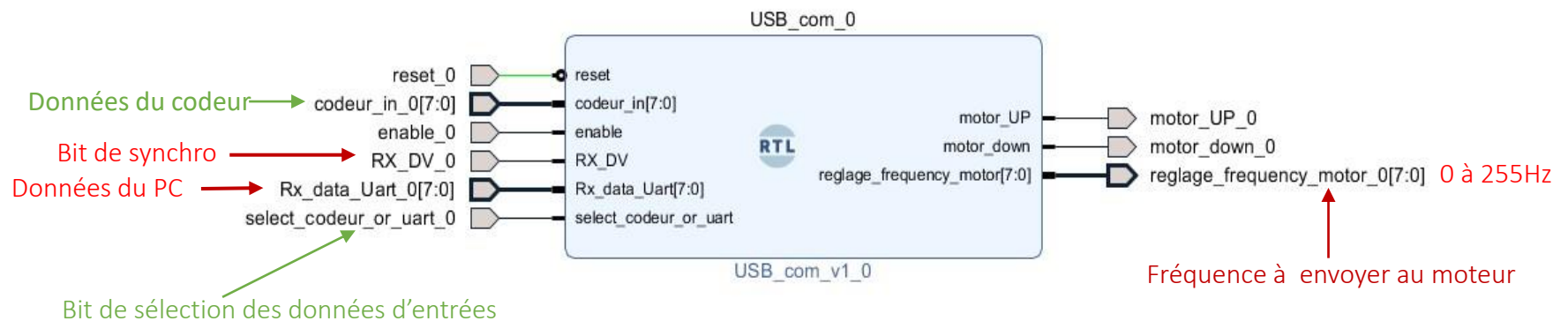
Machine d'état répondant au fonctionnement de l'encodeur

Fabrice Wiotte LPL

### Chronogramme pour CW → or CCW ←



## Module VHDL gestion des données d'entrées : UART-sérial ou Codeur numérique



Fabrice Wiotte LPL



## Code VHDL gestion des données d'entrées : UART-sérial ou Codeur numérique

### --Code Communication USB or codeur and RAM UART--

```
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 USB_codeur_and_RAM is
  Port (
    reset : in std_logic;
    codeur_in : in STD_LOGIC_VECTOR (7 downto 0);
    enable: in std_logic;
    RX_DV : in STD_LOGIC;
    Rx_data_Uart : in STD_LOGIC_VECTOR (7 downto 0);
    select_codeur_or_uart : in STD_LOGIC;
    start_motor_UP : out STD_LOGIC;
    start_motor_down : out STD_LOGIC;
    reglage_frequency_motor : out STD_LOGIC_VECTOR (7 downto 0)
  );
end USB_codeur_and_RAM;

architecture Behavioral of USB_codeur_and_RAM is
  -- define the new type for the 2x8 octets RAM
  type RAM_ARRAY is array (0 to 1) of std_logic_vector (7 downto 0);
  signal RAM_ADDR: STD_LOGIC_VECTOR (1 downto 0); -- Address to write/read RAM
```

```
-- initial values in the RAM
signal RAM: RAM_ARRAY :=( x"00",x"00 " );
```

```
Begin
process(RX_DV,enable,reset)
begin
if reset ='1' then
    RAM_ADDR <= "00";
    RAM(0) <=X"00";
    RAM(1) <=X"00";
elsif rising_edge(RX_DV) then
```

```
if enable ='1' then
    IF RAM_ADDR <=2 THEN
        RAM_ADDR <= RAM_ADDR + 1;
    else
        RAM_ADDR <=0;
    END IF;
    RAM((RAM_ADDR)) <= Rx_data_Uart;
end if;
end if;
end process;
```

```
reglage_frequency_motor <= RAM(0) when select_codeur_or_uart ='1' else codeur_in;
start_motor_UP <= '1' when RAM(1)= X"01" else '0';
start_motor_DOWN <= '1' when RAM(1)= X"02" else '0';
```

```
end Behavioral;
```

Déclaration d'une zone de mémoire Pour les deux premiers octets envoyés

Process D'incrémentation en mémoire

## Fichier de contrainte pour la Cmod A7-35T

```
set_property PACKAGE_PIN L17 [get_ports sys_clock]
set_property PACKAGE_PIN M3 [get_ports {seg[0]}]
set_property PACKAGE_PIN L3 [get_ports {seg[1]}]
set_property PACKAGE_PIN A16 [get_ports {seg[2]}]
set_property PACKAGE_PIN K3 [get_ports {seg[3]}]
set_property PACKAGE_PIN C15 [get_ports {seg[4]}]
set_property PACKAGE_PIN H1 [get_ports {seg[5]}]
set_property PACKAGE_PIN A15 [get_ports {seg[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg[6]}]
set_property PACKAGE_PIN W6 [get_ports afficheur_0]
set_property PACKAGE_PIN U3 [get_ports afficheur_1]
set_property PACKAGE_PIN U7 [get_ports afficheur_2]
set_property PACKAGE_PIN W7 [get_ports afficheur_3]
set_property PACKAGE_PIN U8 [get_ports afficheur_4]
set_property PACKAGE_PIN V8 [get_ports afficheur_5]
set_property IOSTANDARD LVCMOS33 [get_ports afficheur_0]
set_property IOSTANDARD LVCMOS33 [get_ports afficheur_1]
set_property IOSTANDARD LVCMOS33 [get_ports afficheur_2]
set_property IOSTANDARD LVCMOS33 [get_ports afficheur_3]
set_property IOSTANDARD LVCMOS33 [get_ports afficheur_4]
set_property IOSTANDARD LVCMOS33 [get_ports afficheur_5]
set_property IOSTANDARD LVCMOS33 [get_ports BNT0]
set_property IOSTANDARD LVCMOS33 [get_ports BNT1]
```

```
set_property IOSTANDARD LVCMOS33 [get_ports Empty]
set_property IOSTANDARD LVCMOS33 [get_ports Full]
set_property IOSTANDARD LVCMOS33 [get_ports rotary_A]
set_property IOSTANDARD LVCMOS33 [get_ports rotary_B]
set_property IOSTANDARD LVCMOS33 [get_ports RX_Serial]
set_property IOSTANDARD LVCMOS33 [get_ports visu_DOWN]
set_property IOSTANDARD LVCMOS33 [get_ports visu_UP]
set_property PACKAGE_PIN A18 [get_ports BNT0]
set_property PACKAGE_PIN B18 [get_ports BNT1]
set_property PACKAGE_PIN J17 [get_ports RX_Serial]
set_property PACKAGE_PIN A17 [get_ports visu_DOWN]
set_property PACKAGE_PIN C16 [get_ports visu_UP]
set_property PACKAGE_PIN B17 [get_ports Empty]
set_property PACKAGE_PIN B16 [get_ports Full]
set_property PACKAGE_PIN L1 [get_ports rotary_A]
set_property PACKAGE_PIN L2 [get_ports rotary_B]
set_property PACKAGE_PIN U4 [get_ports {Commandes_demi_pas[1]}]
set_property PACKAGE_PIN V3 [get_ports {Commandes_demi_pas[2]}]
set_property PACKAGE_PIN V4 [get_ports {Commandes_demi_pas[0]}]
set_property PACKAGE_PIN W5 [get_ports {Commandes_demi_pas[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports sys_clock]
set_property IOSTANDARD LVCMOS33 [get_ports {Commandes_demi_pas[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Commandes_demi_pas[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Commandes_demi_pas[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Commandes_demi_pas[3]}]
```

## Script Python côté PC pour contrôler le moteur pas à pas vitesse et sens de rotation rampe de fréquence

# script python pour contrôler le moteur pas à pas via la liaison USB série

# Fabrice Wiotte 07/07/2022

# -\*- coding: utf-8 -\*-

# On importe Tkinter

import serial

import time

```
ser = serial.Serial(  
    port = 'COM11',  
    baudrate = 9600,  
    parity = serial.PARITY_NONE,  
    stopbits = serial.STOPBITS_ONE,  
    bytesize = serial.EIGHTBITS,  
    #timeout = 100  
)
```

def compteur1(): # rampe up

i = 1

while i <= 255:

time.sleep(0.02)

values = bytearray([i, 2]) # write two byte value frequency and sense of rotation

ser.write(values)

print(i)

i = i + 1

if i == 255:

compteur2()

def compteur2(): # rampe down

i = 255

while i >= 0:

time.sleep(0.05)

values = bytearray([i, 1]) # write two byte value frequency and sense of rotation

ser.write(values)

print(i)

i = i - 1

if i == 0:

compteur1()

try:

while True:

compteur1()

except KeyboardInterrupt: # ctrl+c

ser.close()

print("Fin du programme")

Fabrice Wiotte LPL

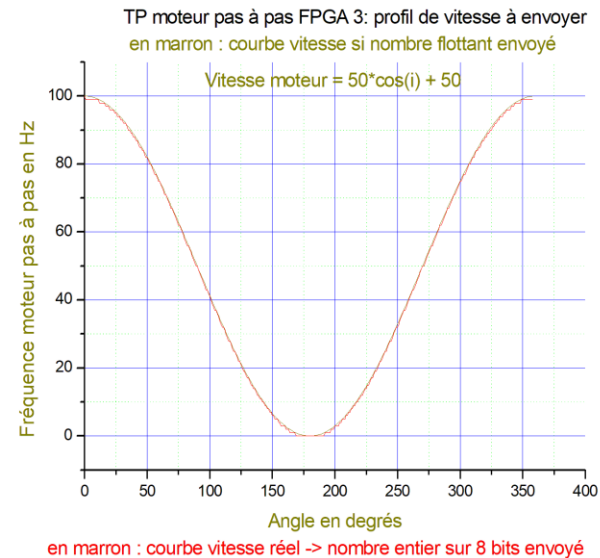
## Script Python côté PC pour contrôler le moteur pas à pas vitesse et sens de rotation rampe de fréquence en cos

```
# -*- coding: utf-8 -*-
# On importe Tkinter
import numpy as np
import matplotlib.pyplot as plt
from pylab import *
import serial
import struct
import time
import math # importation du module
import sys
```

```
ser = serial.Serial(
    port = 'COM11',
    baudrate = 9600,
    parity = serial.PARITY_NONE,
    stopbits = serial.STOPBITS_ONE,
    bytesize = serial.EIGHTBITS,
    timeout = 100
)
def courbe_cos():
```

```
    for z in range(0,360):
        i = math.radians(z)
        w = 50*cos(i) + 50
        time.sleep(0.05)
        #print(w)
        f= int(w)
        print(f)
        values = bytearray([f, 1]) # write two byte value frequency and sense of rotation
        ser.write(values)
```

```
    try:
        while True:
            courbe_cos()
    except KeyboardInterrupt: # ctrl+c
        print("Fin du programme")
        print("ARRET MOTEUR")
        print("fermeture UART")
        ser.close()
```



Fabrice Wiotte LPL