Manipulation 5: Servo-moteur commandé par PWM

Département: **TIC**

Unité d'enseignement: **CSN**

Auteur(s):

- **PILLONEL Bastien**
- **BOUGNON-PEIGNE Kévin**

Professeur:

- **MESSERLI Etienne**

Assistant:

- **JACCARD Anthony**

Date:

- 2023

Introduction

Le but de ce laboratoire est de réaliser un système qui pilote un servo-moteur (abrégé *servo* pour le reste du rapport), sur commande par PWM (Pulse Width Modulation).

Objectif

Les objectifs sont de concevoir, développer, simuler et tester un contrôleur de servo-moteur, sous la forme d'un système séquentiel simple.

Le système utilise le principe d'un PWM ("Pulse Width Modulation" ou "modulation à largeur d'impulsion") qui permet de transmettre une information analogique via un signal binaire. Ce signal PWM est responsable de la transmission de la consigne de position au servo.

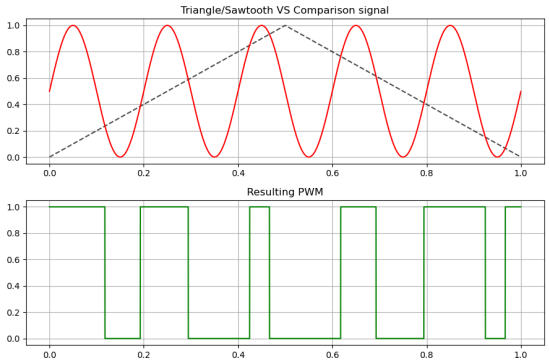
Le laboratoire est décomposé en deux parties. Dans la première partie il s'agit de générer un signal PWM à l'aide d'un compteur en "dent de scie" et d'une comparaison (plus d'explications dans la section "Première partie: Création du PWM"). Dans la deuxième partie il s'agit de gérer la position courante du servo, selon le mode de fonctionnement et l'état des signaux de commande (voir la section "Deuxième partie: Gestion de la position").

Spécification

PWM: Pulse Width Modulation

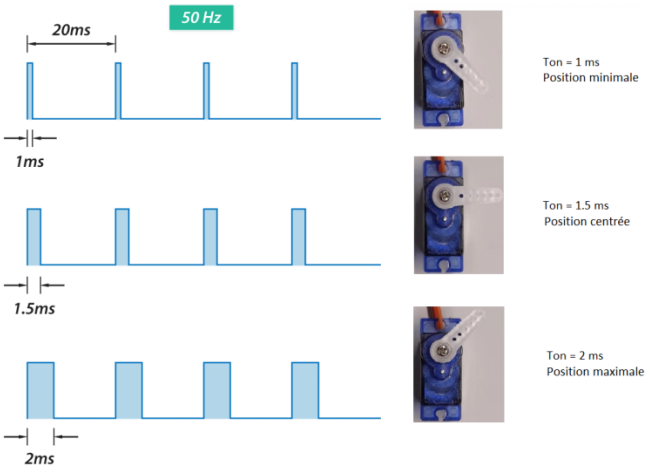
Un PWM est un signal carré de période fixe, à rapport cyclique changeant. Pour réaliser ce genre de signal, l'on se base sur un signal triangulaire (ou en dent de scie, dans le cas présent) et on le compare avec un signal de contrôle.

Voici une démonstration:



Comportement du servo

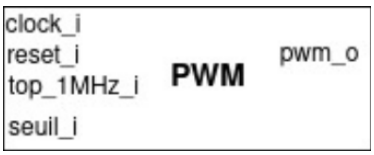
Dans le cadre de ce laboratoire, le PWM du servo fonctionne selon ces informations:



Analyse

Première partie: Gestion de la création du PWM

Soit le bloc de cette partie représentée par:



Pour générer un PWM, voici les fonctions nécessaires:

- asynchrone: Reset
- synchrone : Un compteur de la période du PWM
- synchrone : Un élément mémoire pour le compteur précédent
- synchrone : Un rebouclement de ce compteur (chargement à 0)
- synchrone : Un comparateur entre le compteur de la période du PWM et le seuil d'entrée, pour fixer la sortie pwm_o, soit à '1', soit à '0'.

Selon cette liste, on voit que la table de fonctions synchrones ne peut faire intervenir que les éléments liés au comtpeur. Car l'entrée **seuil_i** et la sortie **pwm.o** sont régies par la règle:

```
pwm_o = '1' if cpt_period <= seul_i else '0'
```

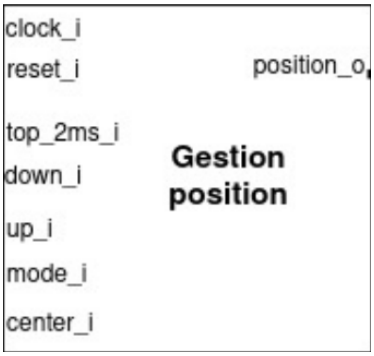
qui permet de générer le PWM.

On obtient alors le décodeur d'états futurs du compteur:

top_1MHz_i	cpt_period	cpt_fut_period	Commentaires
0	-	=cpt_period	Maintien des valeurs
1	=19999	=0	Rebouclement de la période
1	/	=cpt_period+1	Incrémentation du compteur

Deuxième partie: Gestion de la position

Voici tout d'abord la liste des entrées et sortie de notre bloc gestion de position:



Les fonctions nécessaires sont :

- asynchrone: reset
- synchrone : un element mémoire pour le compteur précédent
- synchrone : un compteur/décompteur pour le temps de l'impulsion haute (T_{on})*
- synchrone : un rebouclement du compteur une fois arrivé au maximum de T_{on} (pour le mode automatique)
- synchrone : un détecteur des valeurs min et max de T_{on} avec maintien de la valeur une fois arrivé (pour le mode manuel) ou chargement d'un T_{on} correspondant à la position centrale si T_{on} est hors min ou max

Note: T_{on} correspond à la valeur de notre compteur. Avec un minimum 1ms pour la valeur 999 et un maximum de 2ms pour 1999

Si l'on reprend la donnée du labo, on remarque qu'il est important de donner des priorités pour chaque fonction:

1) Chargement pos. centrale si hors limite
2) Chargement pos. centrale si center_i actif
3) Boucle d'incrémentation si mode_i est actif (mode auto)
4) Incrément jusqu'à T_{on} max puis maintien si up_i actif
5) Décrément jusqu'à T_{on} min puis maintien si down_i

Voici la table de fonctions synchrones:

center_i	mode_i	up_i	down_i	reg_pres	reg_fut	Commentaires
/	/	/	/	<999 ou >1999	=1499	Chargement pos. centrale si hors limite
1	/	/	/	/	=1499	Chargement pos. centrale
/	1	/	/	=1999	=999	Rebouclement
/	1	/	/	autres	=reg_pres+1	Incrément (mode auto.)
/	/	1	/	=1999	=reg_pres	Maintien
/	/	1	/	autres	=reg_pres+1	Incrément (mode man.)
/	/	/	1	=999	=reg_pres	Maintien
/	/	/	1	autres	=reg_pres-1	Soustraction

Un premier regroupement peut être effectuer, on obtient alors la table suivante:

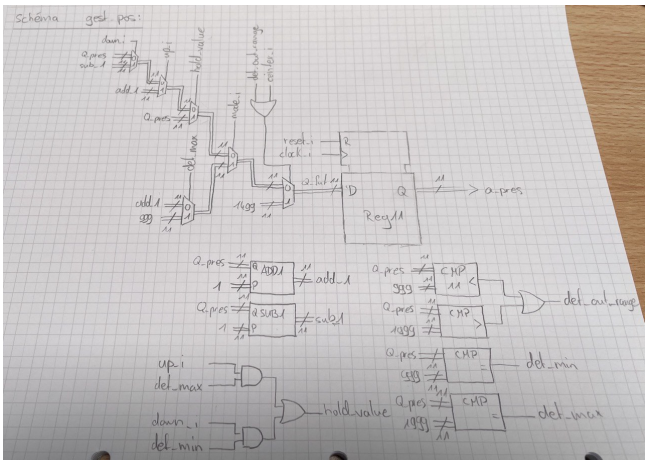
center_i	mode_i	up_i	down_i	reg_pres	reg_fut	Commentaires
/	/	/	/	<999 ou >1999	=1499	Chargement pos. centrale si hors limite
1	/	/	/	/	=1499	Chargement pos. centrale
/	1	/	/	=1999	=999	Rebouclement
/	/	1	/	=1999	=reg_pres	Maintien
/	/	/	1	=999	=reg_pres	Maintien
/	/	/	1	autres	=reg_pres-1	Soustraction
/	/	/	/	autres	=reg_pres+1	Incrément

L'addition et la soustraction peuvent être effectuer par le même bloc additionneur (pour une soustraction, on met le report d'entrée à '1' et on inverse le second nombre d'entrée).

On peut donc coupler ces états et terminer avec la table suivante:

center_i	mode_i	up_i	down_i	reg_pres	reg_fut	Commentaires
/	/	/	/	<999 ou >1999	=1499	Chargement pos. centrale si hors limite
1	/	/	/	/	=1499	Chargement pos. centrale
/	1	/	/	=1999	=999	Rebouclement
/	/	1	/	=1999	=reg_pres	Maintien
/	/	/	1	=999	=reg_pres	Maintien
/	/	/	/	autres	=reg_pres “op.” 1	Incrément/Soustraction

Schéma bloc de la gestion de position:

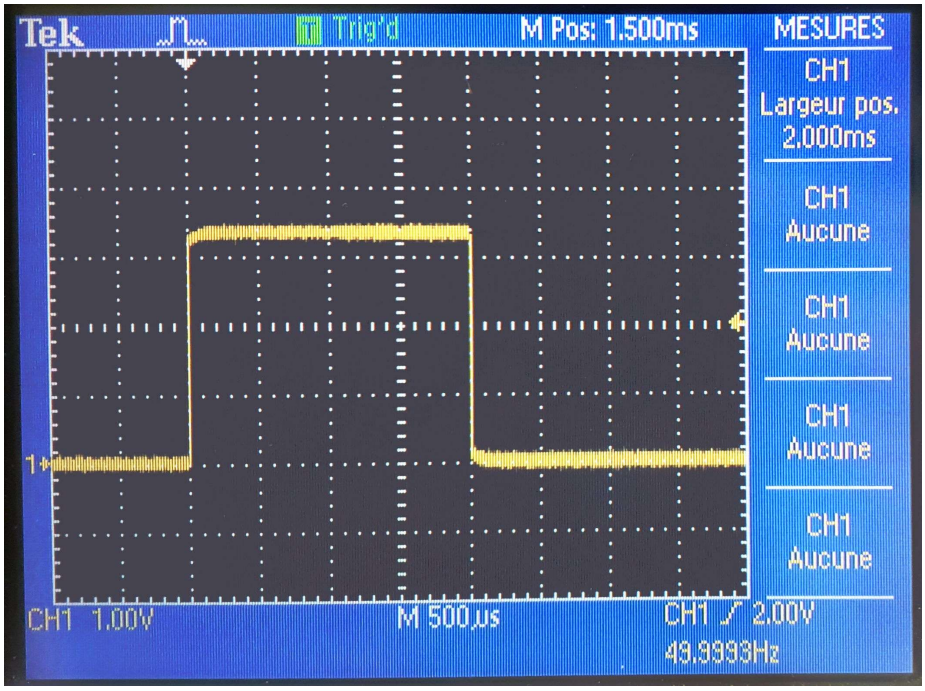


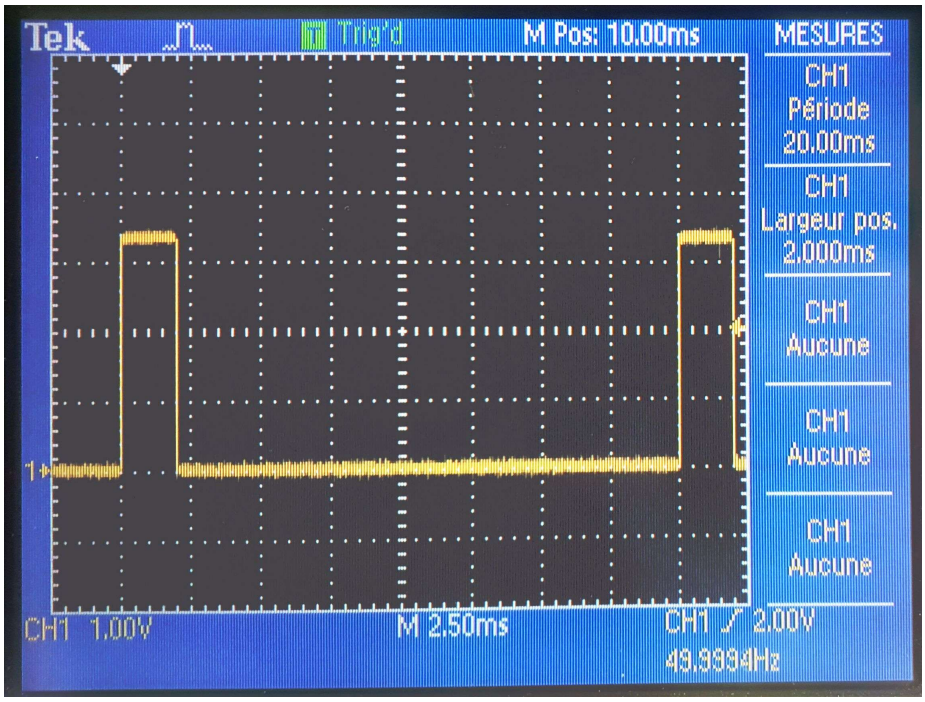
Réalisation et implantation

Simulation

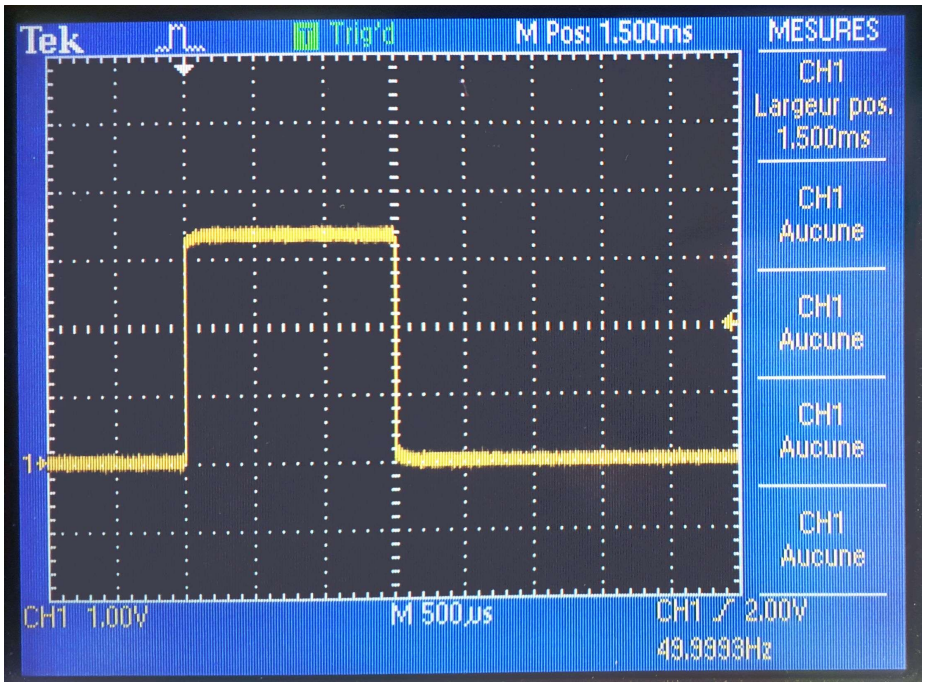
Intégration/Mesure

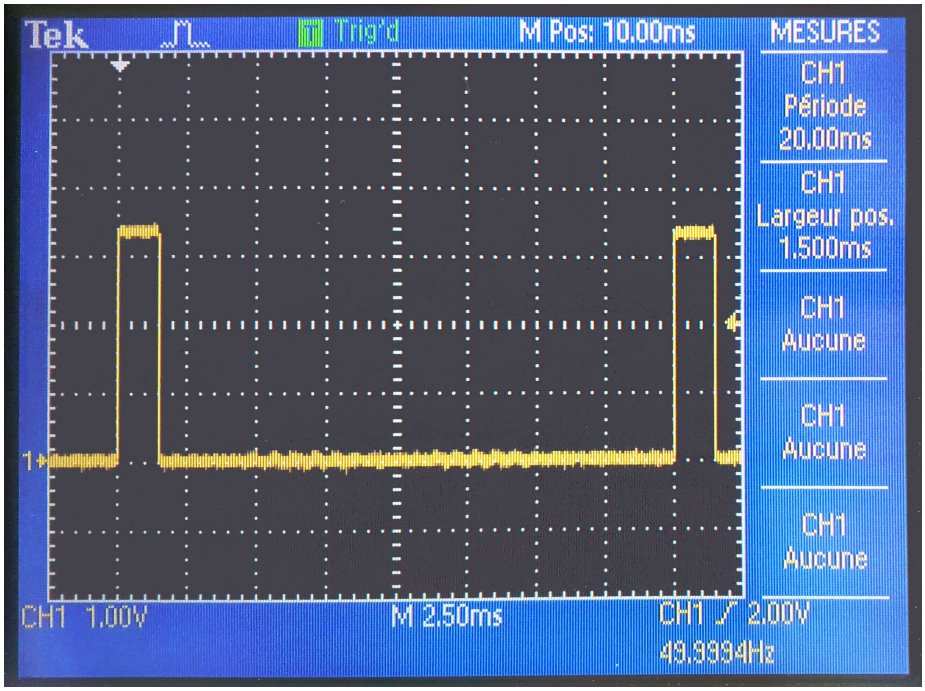
Angle maximale



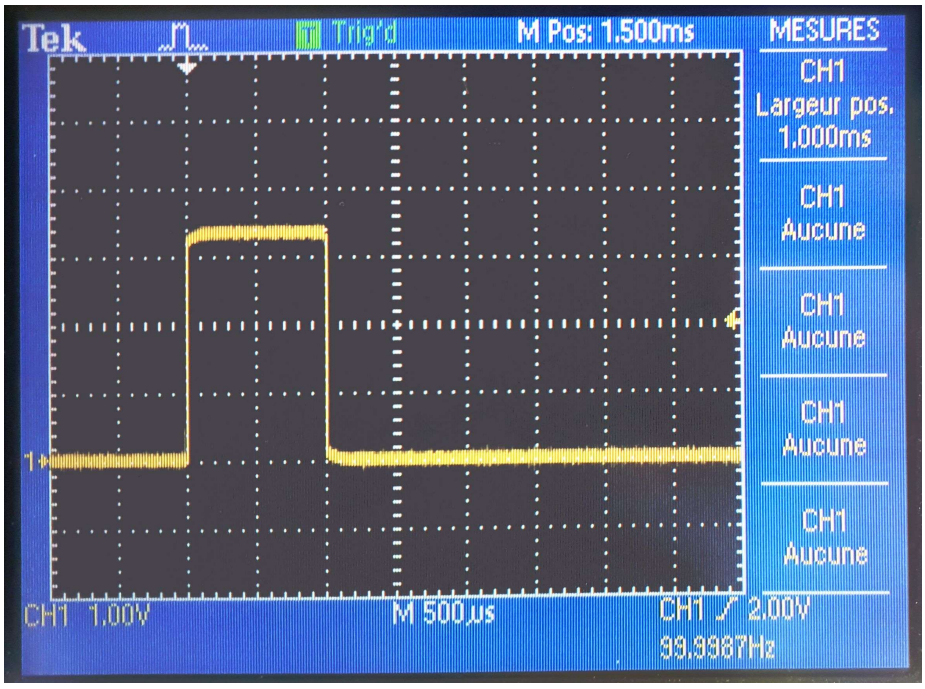

Tek
Trig'd
M Pos: 10.00ms
MESURES
CH1
Période
20.00ms
CH1
Largeur pos.
2.000ms
CH1
Aucune
CH1
Aucune
CH1
Aucune
CH1 1.00V
M 2.50ms
CH1 ⁄ 2.00V
49.9994Hz

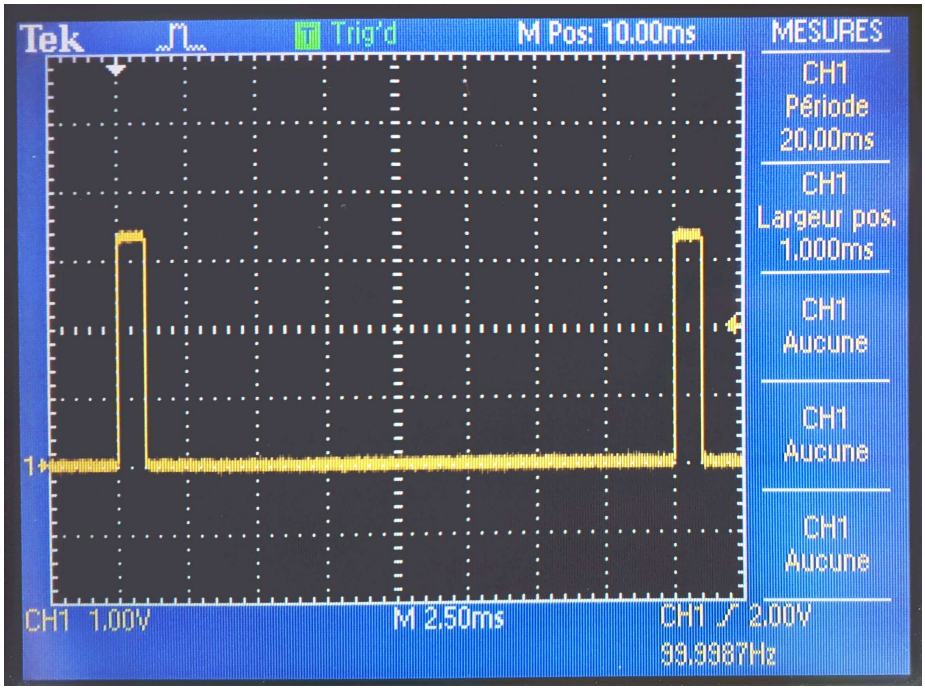
Angle milieu





Angle minimale





Conclusion

Date: Date de rendu

- **PILLONEL Bastien**
- **BOUGNON-PEIGNE Kévin**

Annexe(s)

- pwm.vhd
- gestion_position.vhd
- Gestion position optimisée
  - Schéma attendu
  - Vue RTL
  - gestion_position.vhd (optimisé)

pwm.vhd

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity pwm is
    port (-- Sync
          clock_i     : in std_logic;
          reset_i     : in std_logic;
          -- Inputs
          top_1MHz_i  : in std_logic;
          seuil_i     : in std_logic_vector(14 downto 0); -- range: [0, 2000]
          -- Outputs
          pwm_o       : out std_logic
    );
end entity pwm;

architecture comp of pwm is
    -- TO COMPLETE: Signals declaration
    --| Signals |-----------------------------------------------------------
    -- Period counter with range [0, 20000] <=> 15bits needed
    signal cpt_period_reg_pres_s : unsigned(seuil_i'range);
    signal cpt_period_reg_fut_s  : unsigned(seuil_i'range);
    signal pwm_s                 : std_logic;

    begin
    -- TO COMPLETE: Sawtooth counter generation
    -- Hold current value when top 1MHz ("enable" like) is down
    -- Add1 or Loop period's counter otherwise
    cpt_period_reg_fut_s <= cpt_period_reg_pres_s     when top_1MHz_i = '0'              else
                            cpt_period_reg_pres_s + 1 when cpt_period_reg_pres_s < 20000 else
                            to_unsigned(0, cpt_period_reg_fut_s'length);

    -- D Flip-Flop / Register
    process(reset_i, clock_i)
      --
    begin
      if reset_i = '1' then
        cpt_period_reg_pres_s <= (others => '0');
      elsif rising_edge(clock_i) then
        cpt_period_reg_pres_s <= cpt_period_reg_fut_s;
      end if;
    end process;

    -- TO COMPLETE: PWM signal generation and output
    -- Comparator
    -- Variant1: Compare directly period counter with threshold
    pwm_s <= '1' when cpt_period_reg_pres_s <= unsigned(seuil_i) else
             '0';
    -- Variant2: First comparison with cpt_period > 20'000
    --           and then compare specific range with threshold
    --pwm_s <= '0' when cpt_period_reg_pres_s              >= 2000                           else
    --         '0' when cpt_period_reg_pres_s(10 downto 0) >  unsigned(seuil_i(10 downto 0)) else
    --         '1';

    pwm_o <= pwm_s;

end architecture;
```

gestion_position.vhd

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity gestion_position is
    port (-- Sync
          clock_i     : in std_logic;
          reset_i     : in std_logic;
          -- Inputs
          down_i      : in std_logic;
          up_i        : in std_logic;
          mode_i      : in std_logic;
          top_2ms_i   : in std_logic;
          center_i    : in std_logic;
          -- Outputs
          position    : out std_logic_vector(10 downto 0)
    );
end entity gestion_position;

architecture logic of gestion_position is
    --| Signals |-----------------------------------------------------------------
    signal Q_pres_s, Q_fut_s, add_1_s, sub_1_s, mode_select_s, going_up_s, manual_mode_s : std_logic_vector(10 downto 0);

    signal det_min_s, det_max_s, det_out_range_s, center_s, enable_count_s, hold_value_s : std_logic;

    signal count_value_s    : unsigned(10 downto 0);

    --| Constants |---------------------------------------------------------------
    constant COUNT_MAX : unsigned(10 downto 0) := "11111001111"; -- unsigned 1999
    constant COUNT_MIN : unsigned(10 downto 0) := "01111100111"; -- unsigned 999
    constant COUNT_MID : unsigned(10 downto 0) := "10111011011"; -- unsigned 1499

    begin
    -- TO COMPLETE: Calculate position
    -- Intern signal to enable the flipflop during max Ton of the pwm
    enable_count_s  <=  top_2ms_i;

    det_max_s       <= '1' when count_value_s = COUNT_MAX else
                       '0';
    det_min_s       <= '1' when count_value_s = COUNT_MIN else
                       '0';
    det_out_range_s <= '1' when (count_value_s < COUNT_MIN OR count_value_s > COUNT_MAX) else
                       '0';

    count_value_s   <= unsigned(Q_pres_s);
    add_1_s         <= std_logic_vector(count_value_s + 1);
    sub_1_s         <= std_logic_vector(count_value_s - 1);

    center_s        <= det_out_range_s or center_i;
    hold_value_s    <= (up_i and det_max_s) or (down_i and det_min_s);
```

```
    -- Decoder of futur state
    going_up_s      <=  std_logic_vector(COUNT_MIN) when det_max_s = '1' else
                        add_1_s;

    manual_mode_s   <=  Q_pres_s when hold_value_s = '1' else
                        add_1_s  when up_i = '1'         else
                        sub_1_s  when down_i = '1'       else
                        Q_pres_s;

    mode_select_s   <=  going_up_s when mode_i = '1' else
                        manual_mode_s;

    Q_fut_s         <=  std_logic_vector(COUNT_MID) when center_s = '1' else
                        mode_select_s;

    -- Process of a enabled flipflop
    process(reset_i, clock_i)
    begin
        -- reset the flip flop
        if reset_i = '1' then
            Q_pres_s  <= (others => '0');
        elsif rising_edge(clock_i) then
            if enable_count_s = '1' then
                Q_pres_s  <= Q_fut_s;
            end if;
        end if;

    end process;

    -- TO COMPLETE: Position output
    position <= Q_pres_s;

end logic;
```

Gestion position optimisée

Schéma attendu

Vue RTL

gestion_position.vhd (optimisé)

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity gestion_position is
    port (-- Sync
          clock_i     : in std_logic;
          reset_i     : in std_logic;
          -- Inputs
          down_i      : in std_logic;
          up_i        : in std_logic;
          mode_i      : in std_logic;
          top_2ms_i   : in std_logic;
          center_i    : in std_logic;
          -- Outputs
          position    : out std_logic_vector(10 downto 0)
    );
end entity gestion_position;

architecture logic of gestion_position is
    --| Signals |-----------------------------------------------------------------
    signal reg_pres_s, reg_fut_s : unsigned(position'range);
    signal le_999_s , eq_999_s   : std_logic;
    signal gt_1999_s, eq_1999_s  : std_logic;

    signal sub_carry_s           : std_logic;
    signal cst_one_s             : unsigned(position'range);
    signal reg_plus_minus_one_s  : unsigned(position'range);

    signal center_out_limits_s      : std_logic;
    signal up_limit_s, down_limit_s : unsigned(position'range);
    signal loop_auto_mode_s         : unsigned(position'range);

    --| Constants |---------------------------------------------------------------
    constant LIMIT_UPPER_BOUND : unsigned(position'range) :=
                                          to_unsigned(1999, position'length);
    constant CENTER_VAL        : unsigned(position'range) :=
                                          to_unsigned(1499, position'length);
    constant LIMIT_LOWER_BOUND : unsigned(position'range) :=
                                          to_unsigned( 999, position'length);
```

```
    begin
    -- TO COMPLETE: Calculate position
    -- Add 1 / Sub 1 part
    sub_carry_s <= down_i and (not up_i) and (not mode_i);
    cst_one_s   <= not to_unsigned(0, cst_one_s'length) when sub_carry_s = '1' else
                   to_unsigned( 1, cst_one_s'length);
    reg_plus_minus_one_s <= reg_pres_s + cst_one_s;

    -- In between signals
    center_out_limits_s  <= center_i or le_999_s or gt_1999_s;

    loop_auto_mode_s <= LIMIT_LOWER_BOUND when eq_1999_s = '1' else
                        reg_plus_minus_one_s;

    up_limit_s <= reg_pres_s            when eq_1999_s = '1' else
                  reg_plus_minus_one_s;

    down_limit_s <= reg_pres_s          when eq_999_s = '1' else
                    reg_plus_minus_one_s;

    -- Main separation of Futur States Decoder
    reg_fut_s <= CENTER_VAL       when center_out_limits_s = '1'  else
                 loop_auto_mode_s when mode_i = '1'               else
                 up_limit_s       when up_i = '1'                 else
                 down_limit_s     when down_i = '1'               else
                 reg_pres_s;

    -- D Flip-Flop / Register
    process(reset_i, clock_i)
      --
    begin
      if reset_i = '1' then
        reg_pres_s <= (others => '0');
      elsif rising_edge(clock_i) then
        if top_2ms_i = '1' then
          reg_pres_s <= reg_fut_s;
        end if;
      end if;
    end process;

    -- TO COMPLETE: Position output
    position <= std_logic_vector(reg_pres_s);

    le_999_s <= '1' when reg_pres_s < LIMIT_LOWER_BOUND else
                '0';
    eq_999_s <= '1' when reg_pres_s = LIMIT_LOWER_BOUND else
                '0';

    gt_1999_s <= '1' when reg_pres_s > LIMIT_UPPER_BOUND else
                 '0';
    eq_1999_s <= '1' when reg_pres_s = LIMIT_UPPER_BOUND else
                 '0';

end logic;
```