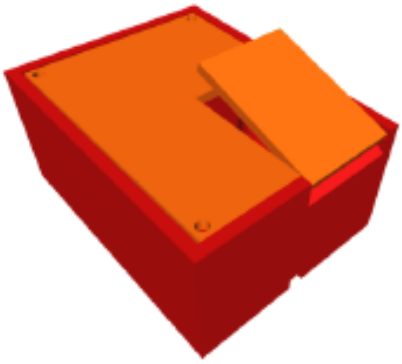
31/05/2021

# Useless Box

## Compte Rendu - ER2



Lucas GAUVAIN & Romuald CONTAMINE
IUT GEII ANGOULEME
ENSEIGNANT : M. GARCIER

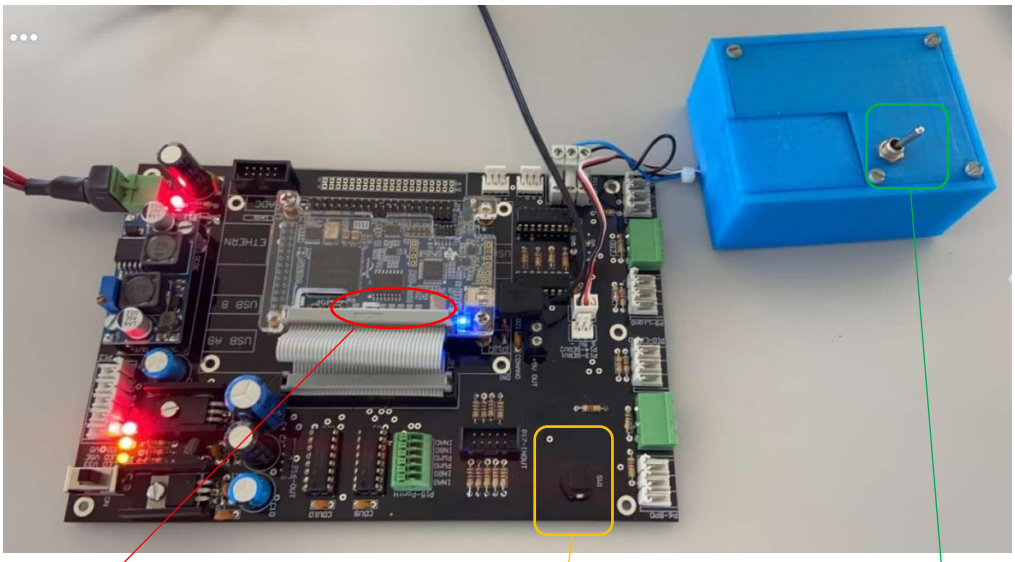
# Table des matières

## Présentation du projet

L'objectif de ce projet est de reproduire par la programmation VHDL de la DEO Nano, le fonctionnement par défaut de la Useless Box mise à notre disposition.

La Useless Box est connectée à une carte FPGA de type Cyclone IV de référence EP4CE22F17C6. Sur cette carte se trouve 8 LED ainsi que des boutons poussoirs.

Le fonctionnement de cette Useless Box est le suivant : Nous devons enclencher un interrupteur. A cet instant, les 8 LED présente sur la carte sont allumée, et, chaque seconde une LED s'éteint. Au bout des 8 secondes, c'est-à-dire lorsque toute les LED sont éteintes, un doigt mécanique sort de la Useless Box et vient repousser l'interrupteur. Les 8 LED se rallument alors une par une. Le temps nécessaire à l'allumage de deux LED successives est donc de 1 seconde.



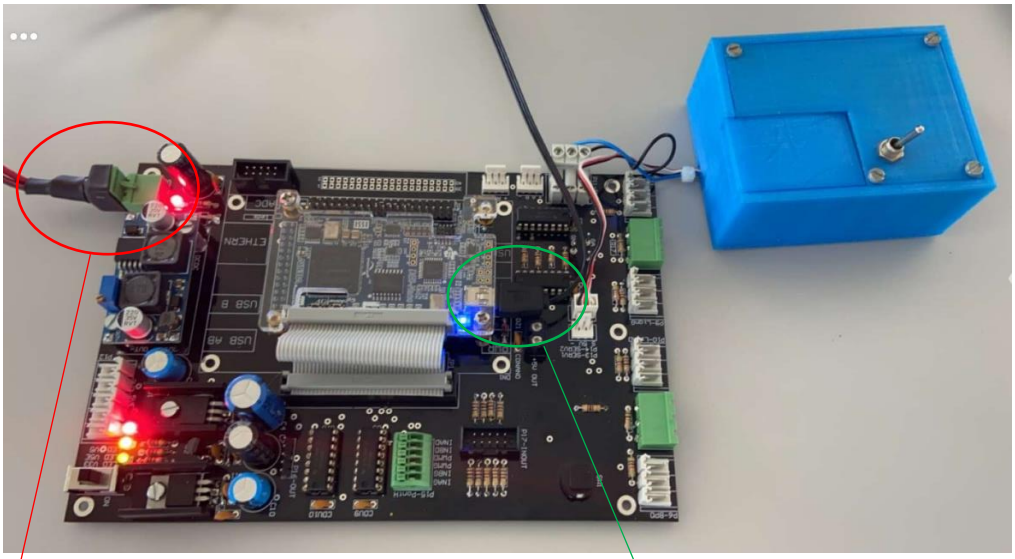
Endroit où se trouvent les 8 LED, éteintes ici.

Bouton poussoir relier à l'interrupteur

Interrupteur à enclencher

Au début de chaque séance, nous réaliserons les branchements suivants pour alimenter correctement la carte :

- Brancher le connecteur vert sur la carte DEO Nano.
- Allumer l'alimentation stabilisée (METRIX AX 502) et régler une tension de sortie de 12V ainsi qu'un courant maximal.
- Brancher la masse.
- Brancher l'alimentation à la sortie + 12V.
- Connecter la carte FPGA à l'ordinateur avec le câble USB.



Connecteur vert d'alimentation

Cable USB reliant le FPGA à l'ordinateur

Nous nous servirons principalement du logiciel Quartus pour réaliser ce projet.

Nous commencerons par paramétrer une base de temps. Ensuite, nous créerons le projet TopUselessBox où se trouvera l'ensemble de nos composant, nous réaliserons un bargraph pour allumer et éteindre les LED. Pour finir, nous programmerons un servomoteur et un multiplexeur pour actionner le doigt mécanique et reproduire le fonctionnement de base ce cette Useless Box.

## Paramétrage de la base de temps

Pour réaliser ce premier élément, nous avons créé sur le disque I de l'ordinateur un dossier ER2.

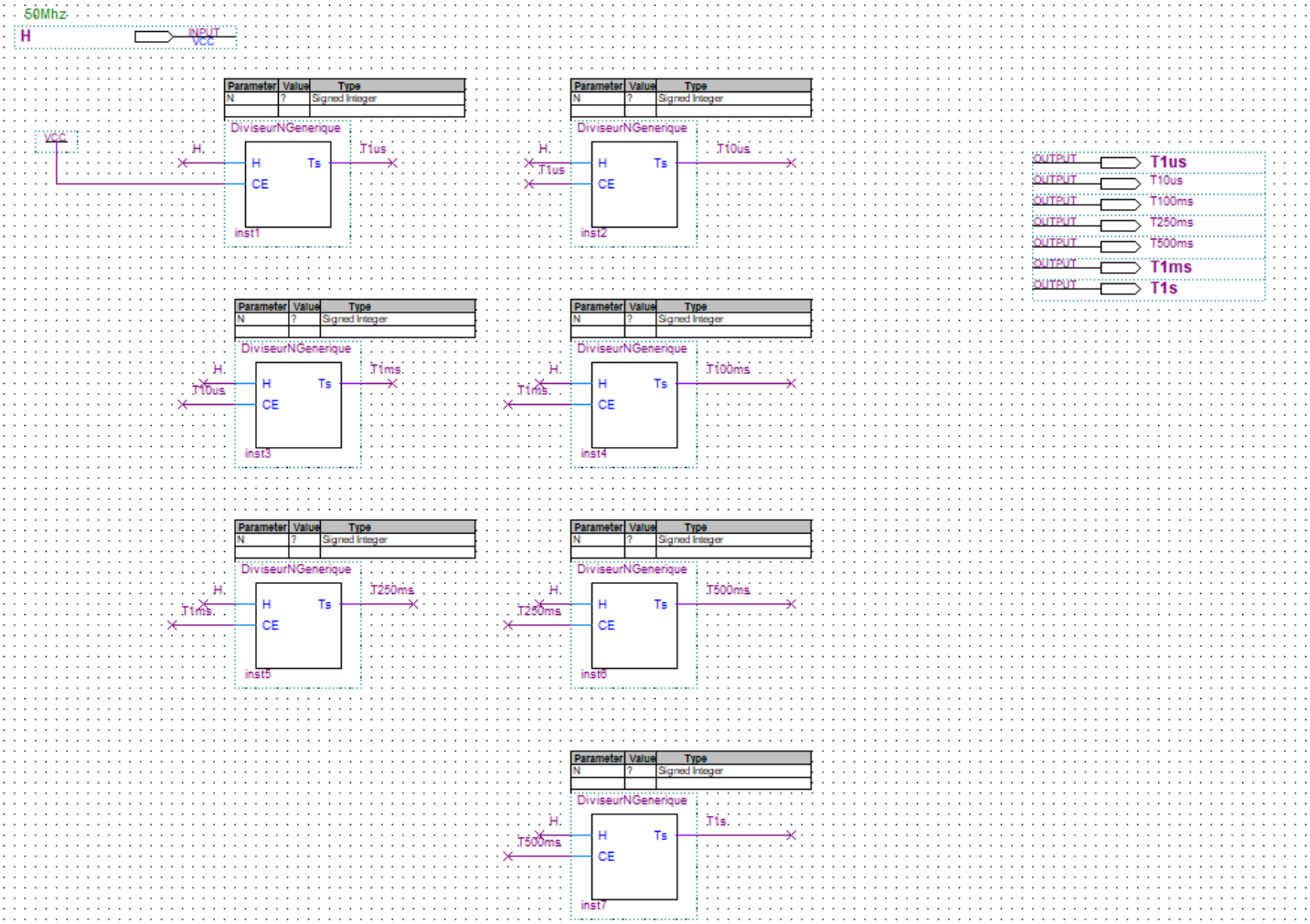
A l'intérieur de ce dossier nous avons créé deux autres dossiers qui sont DiviseurNGenerique et BaseDeTemps. Nous avons ensuite téléchargé le fichier DiviseurNGenerique.vhd dans notre dossier DiviseurNGenerique et le fichier BaseDeTemps.bdf dans notre dossier BaseDeTemps. Ces deux fichiers étaient téléchargeables via l'ENT/Garcier/ER2/Fichiers.

Nous pouvons maintenant commencer la création du projet BaseDeTemps. Pour cela nous avons créé sur Quartus un projet BaseDeTemps dans notre dossier BaseDeTemps avec comme référence le Cyclone IV EP4CE22F17C6. Puis nous avons indiquez à notre projet le chemin du fichier DiviseurNGenerique.vhd à partir du menu Project/Add Remove File in Project.

Le fichier DiviseurNGenerique.vhd est le suivant :

```
1  -- Ce composant génére un signal présentant un NLH pendant une période de H toutes les N périodes de H
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4  use ieee.std_logic_unsigned.all;
5
6  ENTITY DiviseurNGenerique IS
7  generic(N:integer:=50);
8  PORT
9     (
10    H,CE:IN std_logic;
11    Ts:out std_logic:='0'
12    );
13 END ;
14 ARCHITECTURE a_DiviseurNGenerique OF DiviseurNGenerique IS
15    signal etat:integer range 0 to N;
16 BEGIN
17    PROCESS (H)
18    BEGIN
19       if (H'event and H='1') then
20          case etat is
21             when  0=>
22                Ts<='0';
23                if CE='1' then etat<=1; end if;
24             when N-1=>
25                if CE='1' then
26                   etat<=0;
27                   Ts<='1';
28                end if;
29             when others =>
30                if CE='1' then
31                   etat<=etat+1;
32                end if;
33          end case;
34       end IF;
35    END PROCESS;
36 END;
37
```

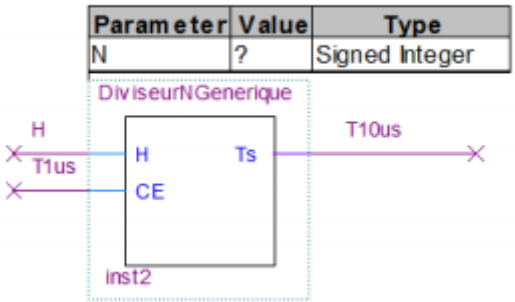
Le fichier BaseDeTemps.bdf est le suivant :




Commençons le paramétrage de la base de temps :

Nous devons d'abord déterminer la période de cette horloge. L'horloge est cadencée à 50MHz alors : $T = \frac{1}{F} = \frac{1}{50*10^9} = 2 * 10^{-8}\ seconde\ soit\ 20\ ns$.

Ceci est un des 7 diviseurs de fréquence de la base de temps :



Chacun des diviseurs est paramétrable selon la valeur de N. A chaque front montant de H ce diviseur regarde si CE est à l'état haut, si c'est le cas un compteur est incrémenté. Lorsque N incréments la sortie Ts passe à l'état haut pendant une période de H.

On peut constater qu'ici, la période de sortie vaut dix fois la période de l'entrée ce qui nous donne :

$$1.10^{-6} * N = 10.10^{-6}$$

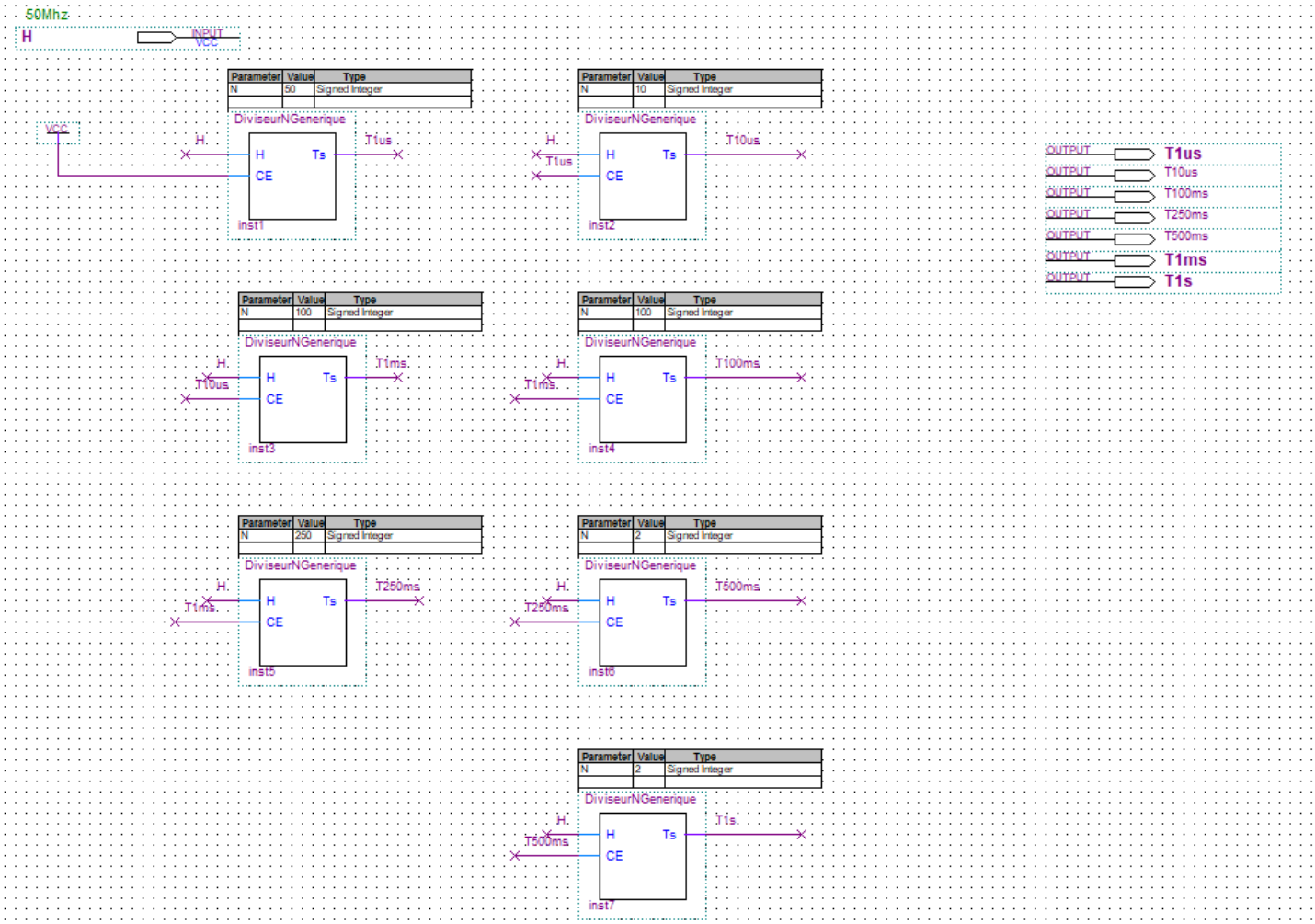
$$<=> N = \frac{10.10^{-6}}{1.10^{-6}} = 10$$

De façon littérale la valeur de N en fonction de Ts et CE est $N = \frac{Ts}{CE}$.

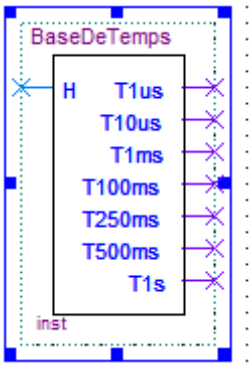
Avec cette formule, nous sommes alors capables de déterminer la valeur de N pour chaque diviseur.

- Pour l'inst1 : $N = \frac{1.10^{-6}}{20.10^{-9}} = 50$
- Pour l'inst2 : $N = \frac{10.10^{-6}}{1.10^{-6}} = 10$
- Pour l'inst3 : $N = \frac{1.10^{-3}}{10.10^{-6}} = 100$
- Pour l'inst4 : $N = \frac{100.10^{-3}}{1.10^{-3}} = 100$
- Pour l'inst5 : $N = \frac{250.10^{-3}}{1.10^{-3}} = 250$
- Pour l'inst6 : $N = \frac{500.10^{-3}}{250.10^{-3}} = 2$
- Pour l'inst7 : $N = \frac{1}{500.10^{-3}} = 2$

Avec ces valeurs, nous obtenons alors la schématique suivante :

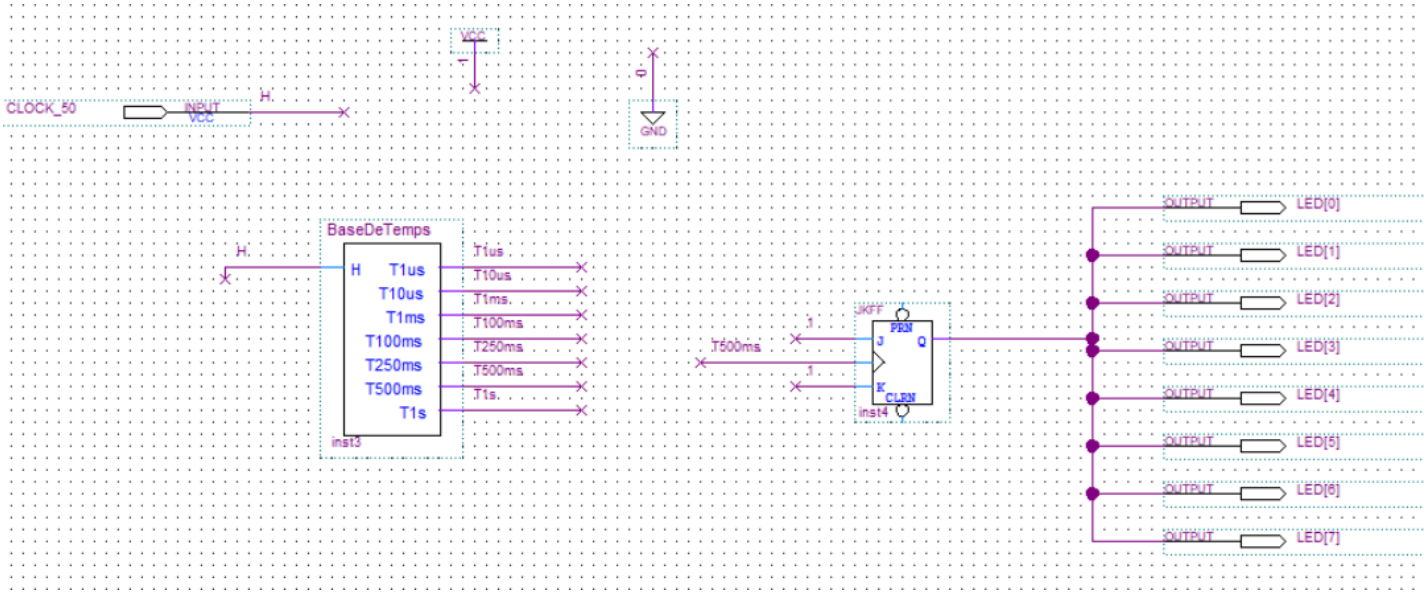



Ensuite, nous avons compiler notre projet afin de créer le symbole, puis nous avons fait des ajustements c'est-à-dire que nous l'avons modifié afin que les sorties soient triées selon un temps croissant, ce qui donne :



## Création du projet TopUselessBox

Pour créer ce projet, nous avons créé le dossier TopUselessBox puis nous l'avons créé dans Quartus avec la référence Cyclone IV EP4CE22F17C6, nous avons fait un lien vers DiviseurNGenerique.vhd et BaseDeTemps.bdf à partir du menu : Project/Add file to Project, pour pouvoir créer la schématique suivante :



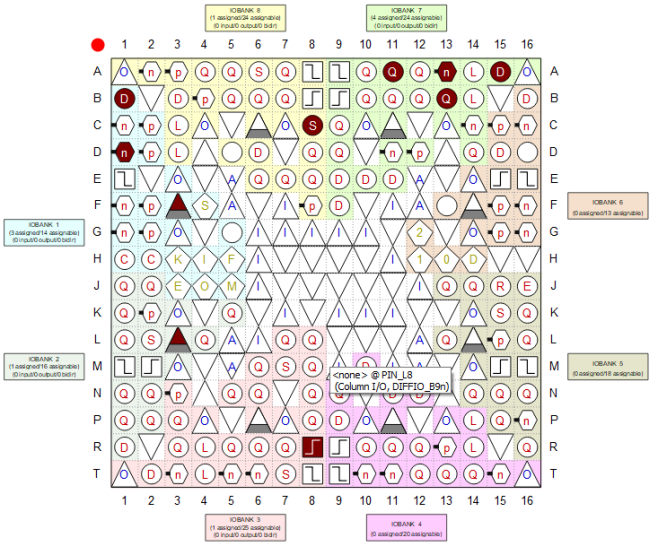

L'objectif de cette schématique est de faire clignoter les 8 LED avec une période de 1 seconde. Nous allons utiliser la sortie T500ms de la base de temps reliée à l'entrée de l'horloge de la bascule JK. En effet, la bascule JK sert à faire un diviseur de fréquence lorsqu'elle est mise en mode Toggle (J = K = 1) et donc de multiplicateur de période. Nous avons donc en sortie une période d'une seconde. On aurait directement une période de 500 ms si les LED étaient directement branchées sur la sortie 500ms.

Une table de vérité de la bascule JK est la suivante :

| C | J | K | $Qn$ | $\overline{Qn}$ | Signification |
|---|---|---|---|---|---|
| ↑ | **0** | **0** | $Qn-1$ | $\overline{Qn-1}$ | MEMOIRE |
| ↑ | **1** | **0** | 1 | 0 | SET |
| ↑ | **0** | **1** | 0 | 1 | RESET |
| ↑ | **1** | **1** | $\overline{Qn-1}$ | $Qn-1$ | TOGGLE |
| ↓ | **X** | **X** | $Qn-1$ | $\overline{Qn-1}$ | MEMOIRE |

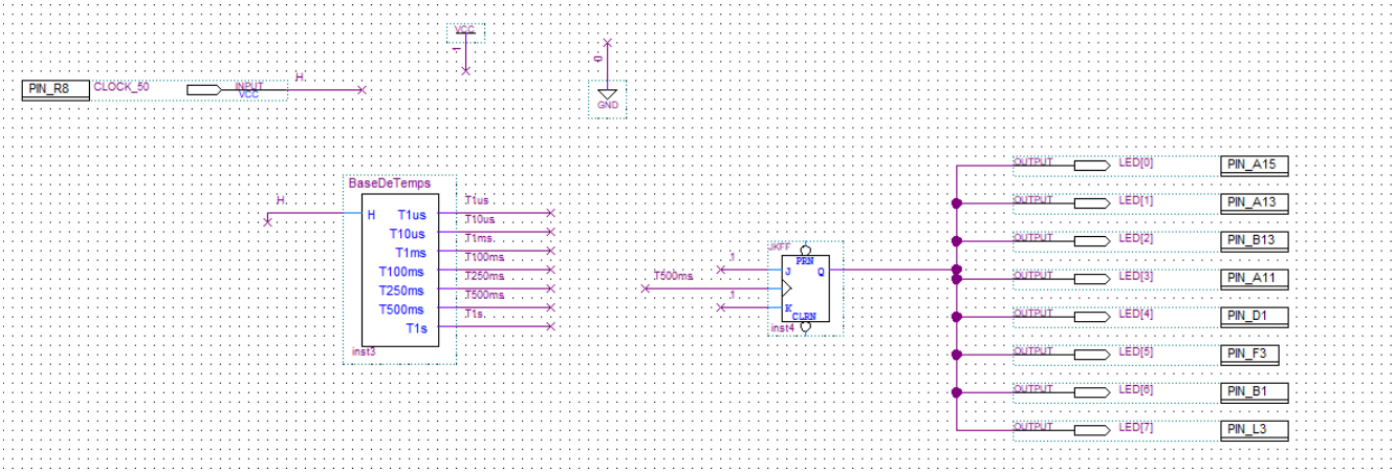
Pour l'assignation nous avons utilisé le Pin Planner via le menu Assignement/Pin Planner, afin de réaliser l'assignation des broches, ce qui nous donne :

Top View - Wire Bond
Cyclone IV E - EP4CE22F17C6




| Node Name | Direction | Location | I/O Bank | VREF Group | I/O Standard | Reserved | Current Strength | Slew Rate | Differential Pair |
|---|---|---|---|---|---|---|---|---|---|
| CLOCK_50 | Unknown | PIN_R8 | 3 | B3_N0 | 2.5 V (default) | | 8mA (default) | | |
| LED[7] | Unknown | PIN_L3 | 2 | B2_N0 | 2.5 V (default) | | 8mA (default) | | |
| LED[6] | Unknown | PIN_B1 | 1 | B1_N0 | 2.5 V (default) | | 8mA (default) | | |
| LED[5] | Unknown | PIN_F3 | 1 | B1_N0 | 2.5 V (default) | | 8mA (default) | | |
| LED[4] | Unknown | PIN_D1 | 1 | B1_N0 | 2.5 V (default) | | 8mA (default) | | |
| LED[3] | Unknown | PIN_A11 | 7 | B7_N0 | 2.5 V (default) | | 8mA (default) | | |
| LED[2] | Unknown | PIN_B13 | 7 | B7_N0 | 2.5 V (default) | | 8mA (default) | | |
| LED[1] | Unknown | PIN_A13 | 7 | B7_N0 | 2.5 V (default) | | 8mA (default) | | |
| LED[0] | Unknown | PIN_A15 | 7 | B7_N0 | 2.5 V (default) | | 8mA (default) | | |
| S_IN | Unknown | PIN_C8 | 8 | B8_N0 | 2.5 V (default) | | 8mA (default) | | |

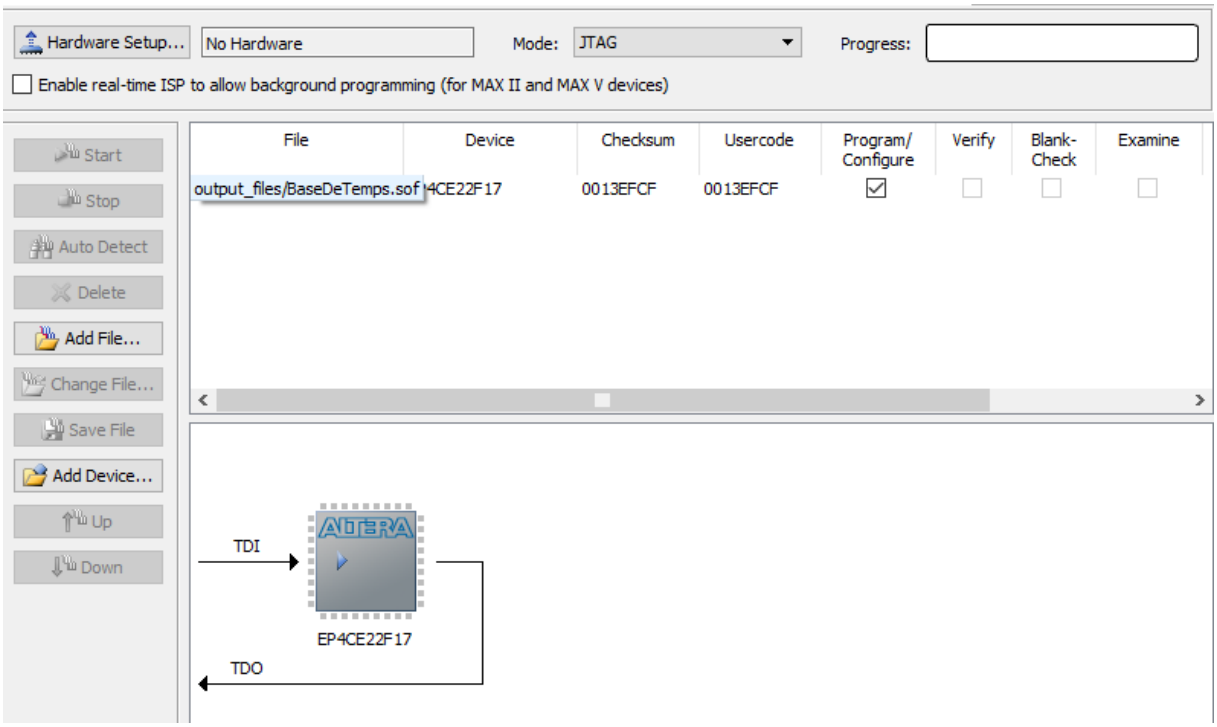
Ainsi, après l'assignation des Pin Planners, nous obtenons une schématique avec les pins suivants :



Nous pouvons à présent compiler notre projet pour programmer la DEO Nano.

Tout au long de ce projet, nous programmerons toujours en mémoire SRAM (Static Random Access Memory). La mémoire SRAM est une mémoire ne pouvant se passer d'alimentation. Lorsque la carte DEO Nano n'est plus alimenté, la mémoire SRAM est effacé et remplacé par le contenu de la mémoire Flash. Ici, la mémoire Flash contient le programme complet du projet que nous avons à reproduire, il ne faut donc pas l'effacer.

Pour programmer, allons dans le menu programmer, sélectionnons le mode JTAG, dans Hardware Setup sélectionnons USB-BLASTER puis sélectionnons le fichier avec l'extension «.sof », comme ci-dessous :




PS : Cette capture d'écran n'a pas été faite à l'IUT, nous ne pouvons donc pas sélectionner USB-BLASTER dans la zone Hardware Setup.

Ainsi, nous obtenons bien un clignotement de période d'une seconde sur les 8 LED de la DEO Nano. Nous pouvons dire que notre base de temps et notre projet TopUselessBox fonctionnent.

# Projet Bargraph

L'objectif du Bargraph est de créer un composant qui reçoivent sur une entrée de x bits le nombre de LED à allumer. Il y aura un bus de sortie de 8 bits sur lesquels seront branchées les 8 LED.

Déterminons le nombre x de bits nécessaire sur le bus d'entrée :

Nous savons que le bus doit permettre d'indiquer le nombre de LED à allumer. Sachant qu'il y a au maximum 8 LED à allumer, nous pouvons choisir un bus d'entrée de 4 bits car le chiffre 8 tient sur 4 bits (1000).

Nous pouvons faire le bilan des entrées – sorties :

| Entrée | Sortie |
|---|---|
| 0000 | 0000 0000 |
| 0001 | 0000 0001 |
| 0010 | 0000 0011 |
| 0011 | 0000 0111 |
| 0100 | 0000 1111 |
| 0101 | 0001 1111 |
| 0110 | 0011 1111 |
| 0111 | 0111 1111 |
| 1000 | 1111 1111 |

Pour les sorties, 0 correspond aux LED éteintes et 1 aux LED allumées.

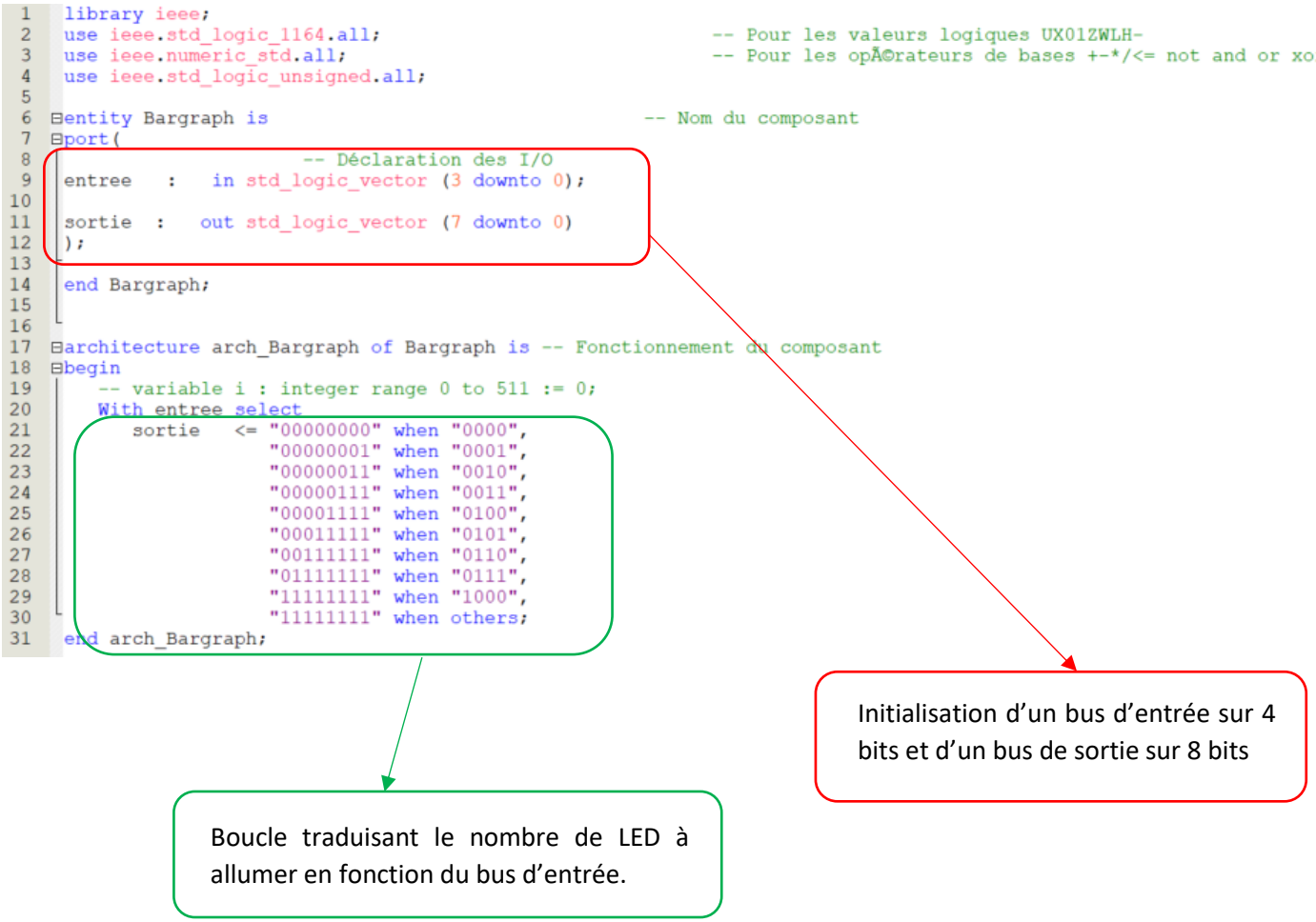
Cependant, avec 4 bits, nous pouvons faire $2^4 = 16$ combinaisons, de 0 à 15. Ainsi, pour les valeurs comprises entre 9 et 15, nous choisirons d'allumer les 8 LED, c'est-à-dire d'avoir un bus de sortie ayant pour valeur 1111 1111.

Pour réaliser la description VHDL du Bargraph, nous utiliserons une solution concurrente car toutes les instructions doivent s'exécuter en même temps. En effet, l'ordre des instructions n'a pas de sens ici, le programme donnera le même résultat quel que soit l'ordre dans lequel les instructions sont écrites.

Dans une description concurrente, nous ne pouvons pas utiliser les structures de contrôles if, case et les boucles loop.

Cependant, nous pouvons utiliser les instructions When… Else… et With… Select…

Nous allons donc écrire la solution concurrente suivante :



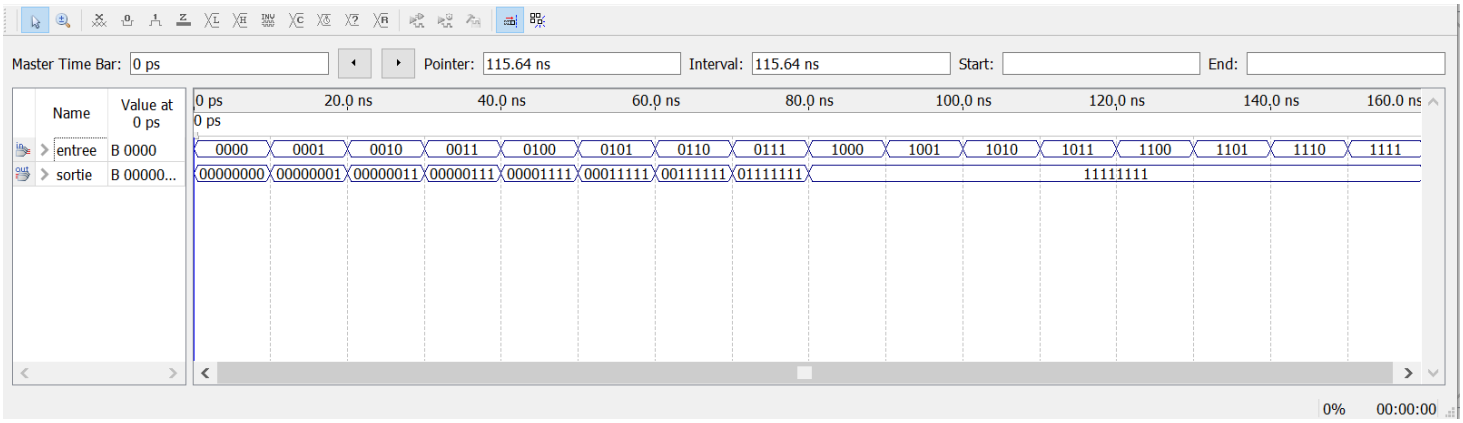
```
1  library ieee;
2  use ieee.std_logic_1164.all;                    -- Pour les valeurs logiques UX01ZWLH-
3  use ieee.numeric_std.all;                       -- Pour les opÃ©rateurs de bases +-*/<= not and or xor
4  use ieee.std_logic_unsigned.all;
5
6  entity Bargraph is                              -- Nom du composant
7  port(
8                      -- Déclaration des I/O
9  entree   :   in std_logic_vector (3 downto 0);
10
11 sortie   :   out std_logic_vector (7 downto 0)
12 );
13
14 end Bargraph;
15
16
17 architecture arch_Bargraph of Bargraph is -- Fonctionnement du composant
18 begin
19    -- variable i : integer range 0 to 511 := 0;
20    With entree select
21       sortie   <= "00000000" when "0000",
22                   "00000001" when "0001",
23                   "00000011" when "0010",
24                   "00000111" when "0011",
25                   "00001111" when "0100",
26                   "00011111" when "0101",
27                   "00111111" when "0110",
28                   "01111111" when "0111",
29                   "11111111" when "1000",
30                   "11111111" when others;
31 end arch_Bargraph;
```

Initialisation d'un bus d'entrée sur 4 bits et d'un bus de sortie sur 8 bits

Boucle traduisant le nombre de LED à allumer en fonction du bus d'entrée.

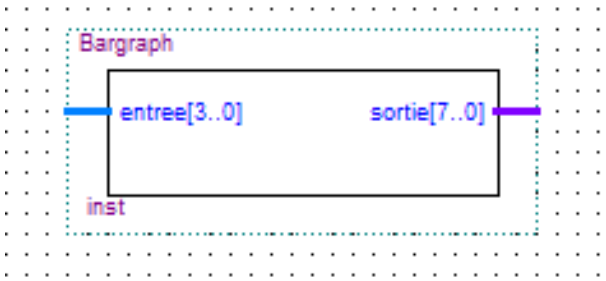
Nous allons ensuite compiler notre code et réaliser une simulation afin de vérifier le bon fonctionnement de notre composant.

Nous obtenons la simulation suivante :




Nous constations que le nombre du bit d'entrée correspond au nombre de LED à allumer sur le bit de sortie. De plus toutes les LED sont allumées dans tous les autres cas.

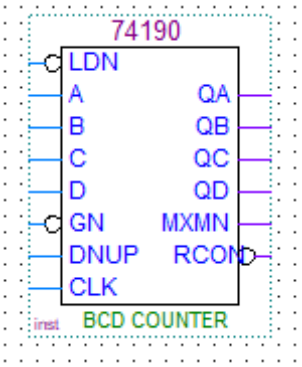
Nous pouvons conclure que notre composant Bargraph fonctionne correctement et nous pouvons alors créer son composant.



## Mise à jour du Projet TopUselessBox

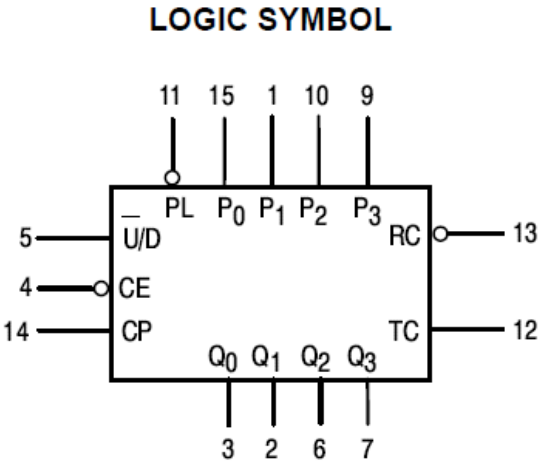
Ajoutons à notre projet TopUselessBox le composant Bargraph ainsi que le composant 74LS190 qui est un compteur/décompteur entre 0 et 9.

Le symbole de ce composant est le suivant :




Faisons une étude de ce composant en étudiant la datasheet :

Le symbole de ce composant sur la datasheet est :




Nous pouvons constater que les noms des entrées et des sorties sont différents sur les deux symboles. Nous devons donc faire le lien entre ces différents noms.

Pour les entrées, nous pouvons faire le lien suivant :

| Entrées | |
|---|---|
| **Datasheet** | **Quartus** |
| PL | LDN |
| P0 | A |
| P1 | B |
| P2 | C |
| P3 | D |
| $\overline{U}/D$ | DNUP |
| CE | GN |
| CP | CLK |

De même, nous pouvons faire le lien suivant pour les sorties :

| Sorties | |
|---|---|
| **Datasheet** | **Quartus** |
| Q0 | QA |
| Q1 | QB |
| Q2 | QC |
| Q3 | QD |
| RC | RCOND |
| TC | MXMN |

En regardant le fonctionnement de ce composant nous pouvons déterminer que :

Lorsque l'entrée inverseuse $\overline{CE}$ active à l'état bas est à 0, le compteur/décompteur fonctionne en fonction de l'horloge (le choix du comptage ou du décomptage se fera en fonction de $\overline{U}/D$). Lorsque $\overline{CE}$ est à 1, le composant ne compte/décompte pas.

Lorsque l'entrée inverseuse PL est active à 0, les entrées P0 jusqu'à P3 sont recopiées sur les sorties Q0 jusqu'à Q3. Si PL est à 1, les bits de sorties Q0 jusqu'à Q3 dépendent du comptage/décomptage.

$\overline{\boldsymbol{U/D}}$ est l'entrée qui va permettre d'indiquer au composant si l'on va compter ou décompter. Cependant, pour compter ou décompter, le compteur dépend aussi de PL, CE et CP. Ainsi, pour effectuer un comptage nous avons besoin que :

- PL soit à 1
- $\overline{CE}$ soit à 0
- $\overline{\boldsymbol{U/D}}$ soit à 0
- Front montant sur l'horloge (CLK)

De même, pour effectuer un décomptage il nous faut que :

- PL soit à 1
- $\overline{CE}$ soit à 0
- $\overline{\boldsymbol{U/D}}$ soit à 1
- Front montant sur l'horloge (CLK)

L'entrée CP correspond à l'entrée de l'horloge active sur front montant.

Les entrées P0, P1, P2, P3 sont des entrées représentant un mot binaire sur 4 bits. La valeur de ces bits peut être préchargées sur les bits de sorties si PL est à 0.

Nous pouvons établir la table de vérité suivante :

| **Entrée** | | | | **Mode** |
|---|---|---|---|---|
| PL | $\overline{CE}$ | $\overline{U/D}$ | CP | |
| 1 | 0 | 0 | ↑ | Comptage |
| 1 | 0 | 1 | ↑ | Décomptage |
| 0 | X | X | X | Preset asynchrone |
| 1 | 1 | X | X | Maintien de la valeur |

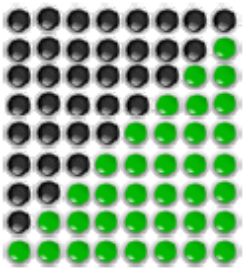
La sortie TC (Terminal Count) est à 0 lorsque le composant compte ou décompte et passe à 1 lorsque le comptage/décomptage est terminé. La sortie reste donc à 1 jusqu'à la relance du compteur.

Les sorties Q0, Q1, Q2, Q3 correspondent au mot binaire de sortie, en fonction du comptage si PL est à 1 ou en fonction des entrées P0, P1, P2, P3 si PL est à 0.

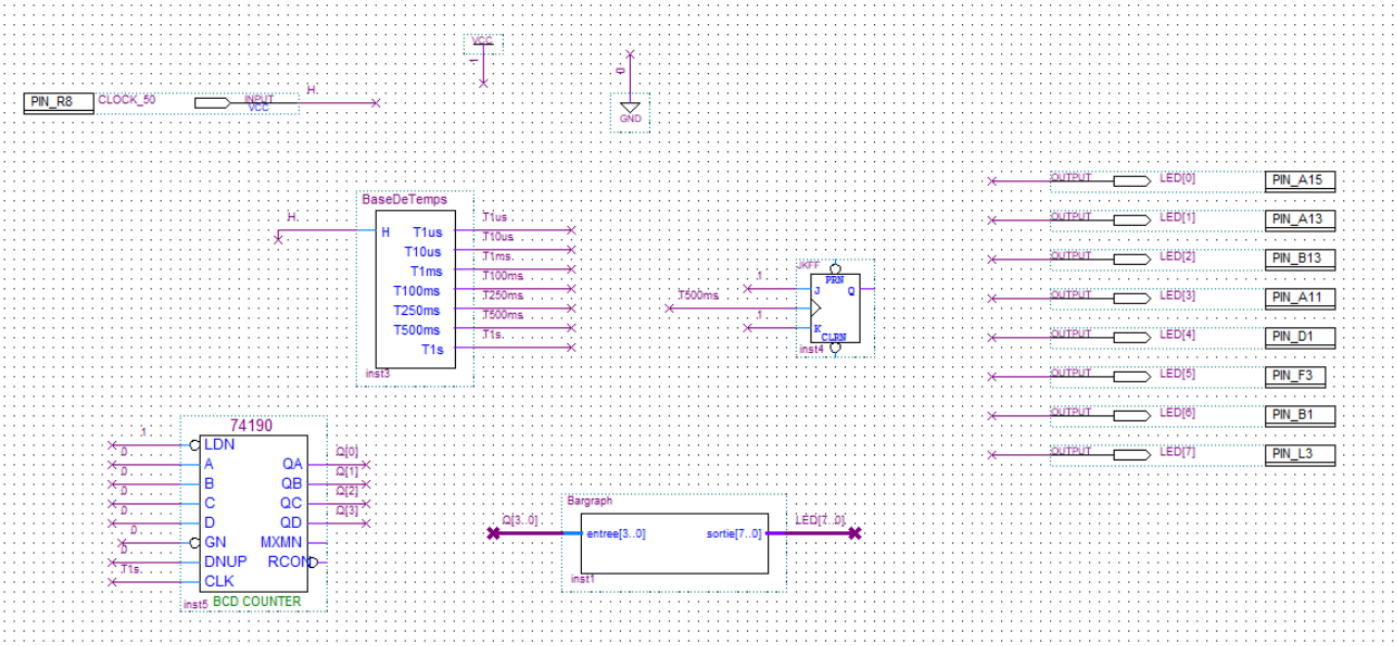
Nous pouvons à présent câbler correctement notre composant ainsi que le bargraph précédemment créer.

## Balayage simple

L'objectif est de créer un chenillard allumant une par une les 8 LED comme le modèle ci-dessous :



Grâce à l'étude précédente, nous câblerons le compteur/décompteur comme sur la schématique suivante.

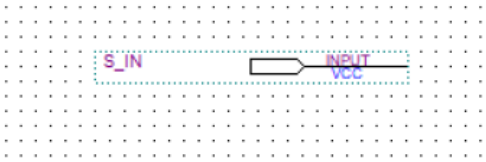



Nous branchons DNUP à 0 pour être en mode comptage et donc pour allumer une LED à chaque front montant de l'horloge. L'horloge est connectée à la sortie T1s de la base de temps pour pouvoir avoir un front montant toutes les secondes et donc pour allumer une LED par seconde. On place également un 0 sur chaque bit d'entrée pour initialiser le compteur à 0. De plus le fait de placer un 0 sur l'entrée GN permet d'autoriser le comptage.

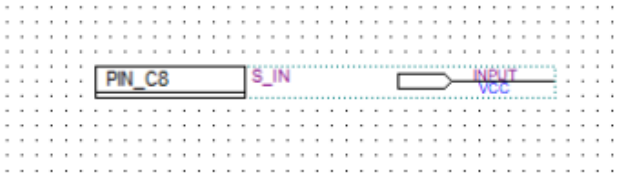
Après avoir compiler notre projet, et programmer la DEO Nano, nous obtenons bien un chenillard avec les 8 LED.

## Aller-Retour

Cette fois nous allons déterminer le sens du chenillard en utilisant l'interrupteur présent sur la Useless Box. Pour ce faire, nous allons devoir saisir l'interrupteur dans notre schématique, comme ci-dessous :



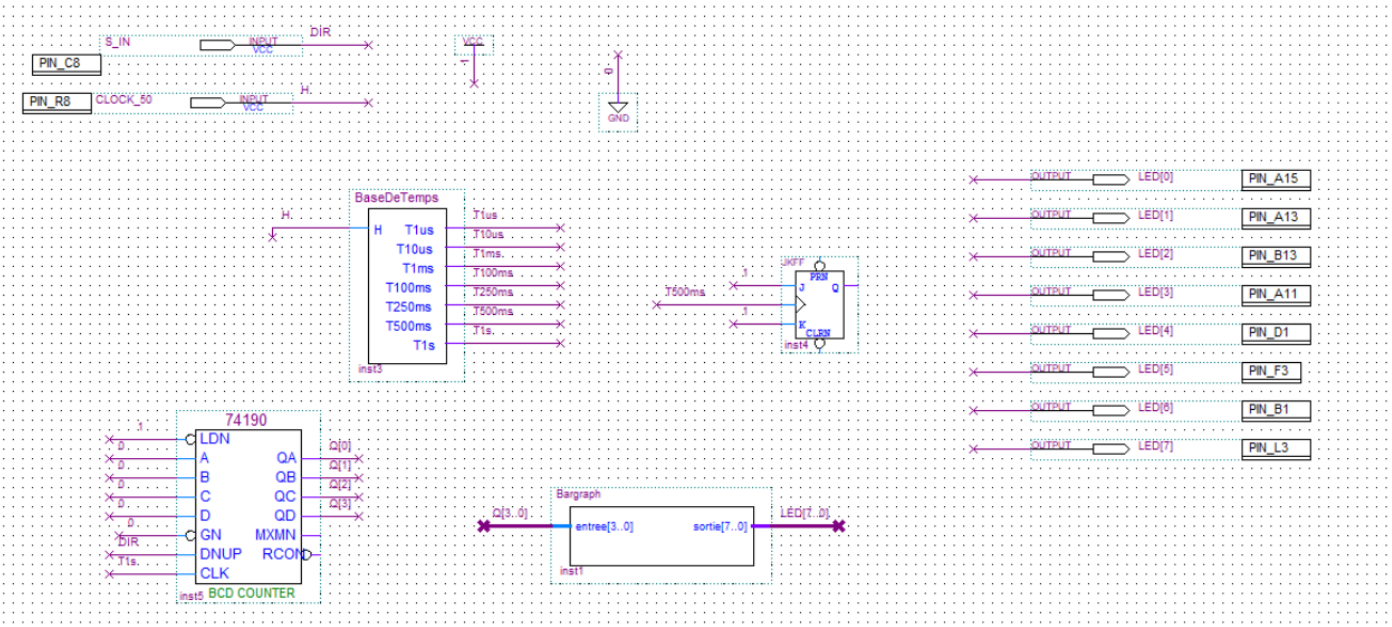

Compilons maintenant notre projet pour pouvoir attribuer l'interrupteur à la broche C8 dans le menu des PIN PLANNERS. Après compilations nous obtenons :




Ainsi, le signal S_IN vaut 0 ou 1 en fonction de la position de l'interrupteur : Si l'interrupteur est à gauche, $S\_IN = 1$ et les LED s'allument vers la gauche. Si l'interrupteur est à droite, $S\_IN = 0$ et les LED s'éteignent vers la droite.

Pour réaliser cette partie, nous connecterons l'entrée DNUP à la sortie de l'interrupteur (DIR sur la schématique). Grâce à cela, la valeur en sortie de l'interrupteur déterminera le comptage ou le décomptage.

Nous obtenons alors la schématique suivante :



## Aller-Retour avec blocage

Dans cette partie on cherche à figer l'état des LED tel que :

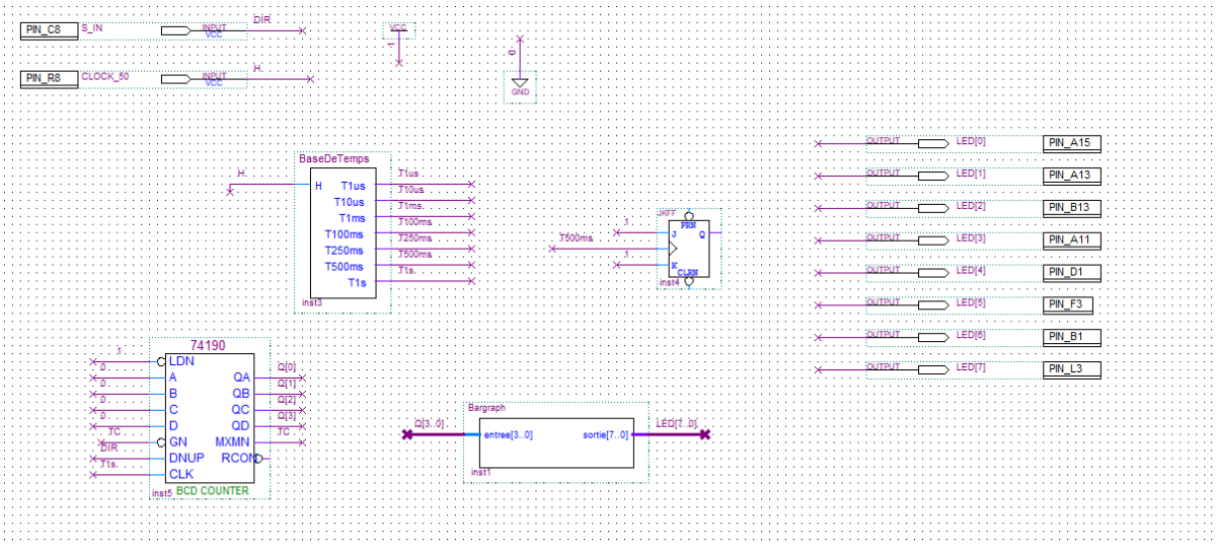
- Lorsque l'interrupteur est à gauche, les LED s'allument toutes au fur et à mesure et lorsqu'elles sont toutes allumées la situation se fige jusqu'à ce que l'on agisse sur l'interrupteur.

- Lorsque l'interrupteur est à droite, les LED s'éteignent toutes au fur et à mesure et lorsqu'elles sont toutes éteintes la situation se fige jusqu'à ce que l'on agisse sur l'interrupteur.

Pour réaliser ce procédé, nous devons envoyer la bonne valeur sur l'entrée inverseuse GN permettant d'autoriser ou non le comptage/décomptage. On rappelle que CE doit recevoir un 0 pour autoriser le processus de comptage/décomptage. On souhaite que CE reçoive un 1 à la fin du comptage pour maintenir la situation des LED.

On sait que la sortie MXMN est à 0 pendant le comptage/décomptage et passe à 1 en fin de processus. Nous devons alors relier cette sortie à l'entrée inverseuse GN pour figer la situation jusqu'à un nouvel enclanchement de l'interrupteur.

En réalisant cela, nous obtenons la schématique suivante :




Nous obtenons bien le résultat attendu. La situation des LED se fige à la fin du comptage ou du décomptage jusqu'à un nouvel enclanchement de l'interrupteur.

Nous obtenons bien le résultat attendu et nous pouvons passer à la suite de notre projet.

# Servomoteur

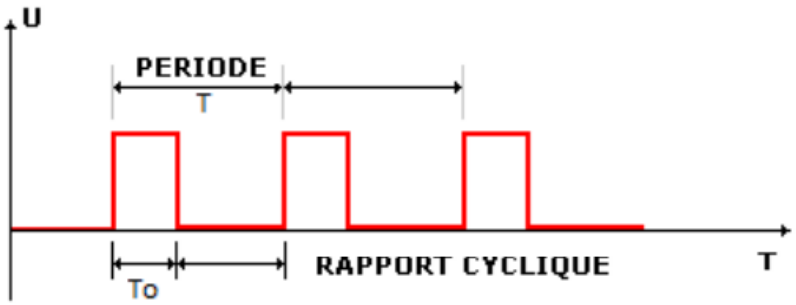
Avant de décrire cette partie, il est nécessaire de connaitre les principes d'un signal de type PWM (Pulse With Modulation ou Modulation de Largeur d'Impulsion).

## Le signal PWM

L'intérêt d'un signal PWM est de pouvoir générer un signal continu à partir d'un système fonctionnant en tout ou rien. Le principe est d'utiliser le filtre passe-bas des systèmes, comme l'inertie d'un moteur. Ainsi, à partir d'une succession d'états discrets pendant un temps T, on peut obtenir une valeur continue moyenne pendant ce temps T.
La valeur moyenne du PWM dépend du rapport cyclique $\frac{T0}{T}$. La valeur moyenne est exprimée par la relation suivante :

$$U_{moy} = \frac{1}{T}\int_0^T u(t)dt = \frac{1}{T}[U_{max} * t]_0^{T0} = U_{max} * \frac{T0}{T}.$$

Le rapport cyclique s'exprime en pourcentage (%).

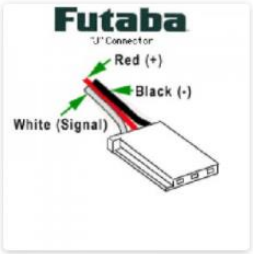



Ainsi, pour l'utilisation du PWM, deux données sont importantes : la période T et le rapport cyclique.

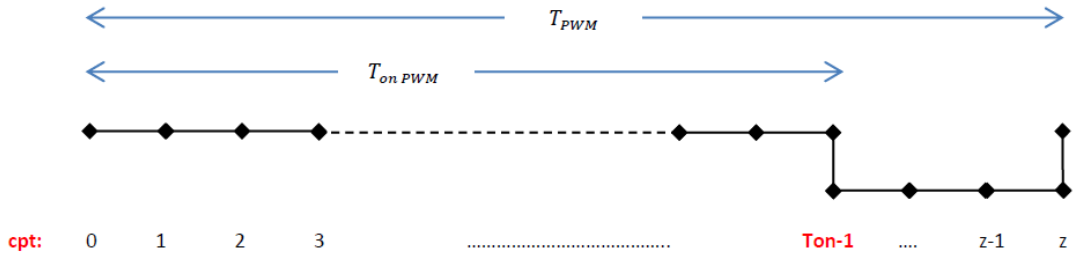
## Fonctionnement d'un servomoteur

Un servomoteur est un appareil qui a pour but de reproduire un mouvement, c'est à dire d'atteindre des positions déterminées et de les maintenir. La position est, pour un moteur rotatif, une position d'angle. Dans notre cas, la sortie peut varier entre 0° et 120°.Les servomoteurs contiennent un petit moteur connecté via des engrenages à un axe de sortie.

Les servomoteurs sont composés de 3 fils (rouge, noir et blanc). Ces trois fils permettent d'alimenter le moteur et de lui transmettre des ordres de positions sous forme d'un signal codé en largeur d'impulsion (PWM). Cela signifie que c'est la durée des impulsions qui détermine l'angle de l'axe de sortie et donc la position du bras de commande du servomoteur. Le signal est répété périodiquement ($T_{PWM}$), toutes les 20 millisecondes, ce qui permet à l'électronique de contrôler et de corriger continuellement la position angulaire de l'axe de sortie, cette dernière étant mesurée par le potentiomètre.




Le fil rouge correspond à l'alimentation positive (4.8V – 6V), le fil noir correspond à la masse et le fil blanc correspond à l'entrée du signal de commande (PWM dans notre cas).

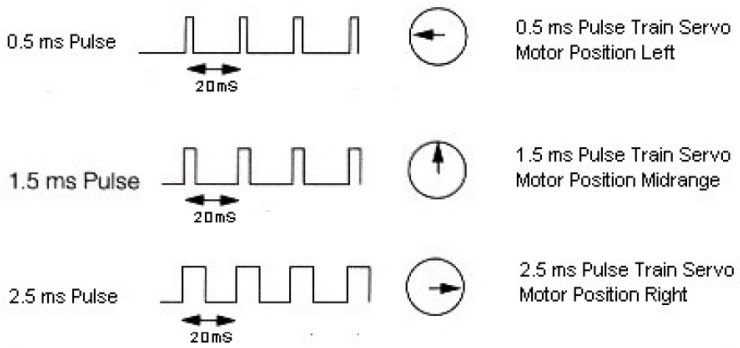



Représentation du signal PWM dans notre cas

Ainsi, pour positionner le bras de commande à la position voulue, il faut jouer sur la durée de l'impulsion $T_{onPWM}$. D'après la datasheet de notre servomoteur, nous pouvons constater que :

- Pour un angle de 0°, la durée de l'impulsion doit être de 0.5ms.
- Pour un angle de 90°, la durée de l'impulsion doit être de 1.5ms.
- Pour un angle de 180°, la durée de l'impulsion doit être de 2.5ms.

Attention ici, pour notre projet la valeur maximale de l'angle de sortie est de 120°.




| Pulse Width Duty / ms | Angle / degrees |
|---|---|
| 0.5 | 0 |
| 1.0 | 45 |
| 1.5 | 90 (center) |
| 2.0 | 135 |
| 2.5 | 180 |

Extrait de la datasheet du Servomoteur

Ainsi, un angle servomoteur de 0° correspond au bras complétement rentré, et un angle de 120° permet d'actionner l'interrupteur de la Useless Box.

Sur le schéma du signal PWM ci-dessus, cpt est une variable du code VHDL qui est incrémentée à chaque front montant de l'horloge et dont la valeur nous servira à établir si la sortie PWM est à l'état haut ou bas. On sait que la période de l'horloge est de 10µs et que la période du signal PWM est de 20ms. On peut alors en déduire que la valeur maximale que pourra prendre le compteur est $\frac{20.10^{-3}}{10.10^{-6}} = 2000$.

Nous devons donc déterminer la durée de la largeur d'impulsion lorsque notre servomoteur a le bras sorti, c'est-à-dire pour un angle de 120°. Dans la datasheet, nous avons une formule permettant de calculée cette durée (en microseconde µs) :

$$\mu s = \frac{Angle*111}{10} + 500$$

Donc dans notre cas :

$$\frac{120 * 111}{10} + 500 = 1832\ \mu s = 1{,}832\ ms$$

Nous pouvons alors en déduire le tableau suivant :

| Angle | Temps à l'état haut du signal de commande | Valeur de $T_{On}$ |
|---|---|---|
| **0°** | 0,5 ms | $(50)_{10} = (0011\ 0010)_2$ |
| **90°** | 1,5 ms | $(150)_{10} = (1001\ 0110)_2$ |
| **120°** | 1,8 ms | $(180)_{10} = (1011\ 0100)_2$ |
| **180°** | 2,5 ms | $(250)_{10} = (1111\ 1010)_2$ |

On sait que la valeur de $T_{On}$ dépend du temps à l'état haut du signal en fonction de la période de l'horloge H. Pour déterminer la valeur de $T_{On}$ (en base 10), nous allons donc utiliser la formule : $(T_{On})_{10} = \frac{Temps\ à\ l'état\ haut}{H}$. Nous devrons ensuite convertir cette valeur en binaire.

Ainsi :

- $\frac{0{,}5.10^{-3}}{10.10^{-6}} = (50)_{10} = (0011\ 0010)_2$
- $\frac{1{,}5.10^{-3}}{10.10^{-6}} = (150)_{10} = (1001\ 0110)_2$
- $\frac{1{,}8.10^{-3}}{10.10^{-6}} = (180)_{10} = (1011\ 0100)_2$
- $\frac{2{,}5.10^{-3}}{10.10^{-6}} = (250)_{10} = (1211\ 1010)_2$

## Programmation du servomoteur

Nous devons à présent réaliser la description VHDL pour le fonctionnement du Servomoteur :

```
1  library ieee;
2  use ieee.std_logic_1164.all;                          -- Pour les valeurs logiques UX01ZWLH-
3  use ieee.numeric_std.all;                             -- Pour les opérateurs de bases +-*/<= not and or xor
4  use ieee.std_logic_unsigned.all;
5
6  entity Servo is
7  port(
8  H        :   in std_logic;                      -- Déclaration des I/O
9  Ton      :   in std_logic_vector (10 downto 0) ;
10
11 PWM      :  out bit     -- PWM = 0 ou 1
12 );
13
14 end Servo;
15
16
17 architecture arch_Servo of Servo is -- Fonctionnement du composant
18 begin
19     process(H)
20     variable cpt : integer range 0 to 2000 := 0;
21
22     begin
23         if rising_edge(H) then
24
25             if cpt >= 1999 then
26                cpt := 0;
27             Else
28                cpt:= cpt +1;
29             end if;
30
31             if cpt < Conv_integer(Ton) then
32                PWM <= '1';
33             Else
34                PWM <= '0';
35             end if;
36         end if;
37     end process;
38 end arch_Servo;
```

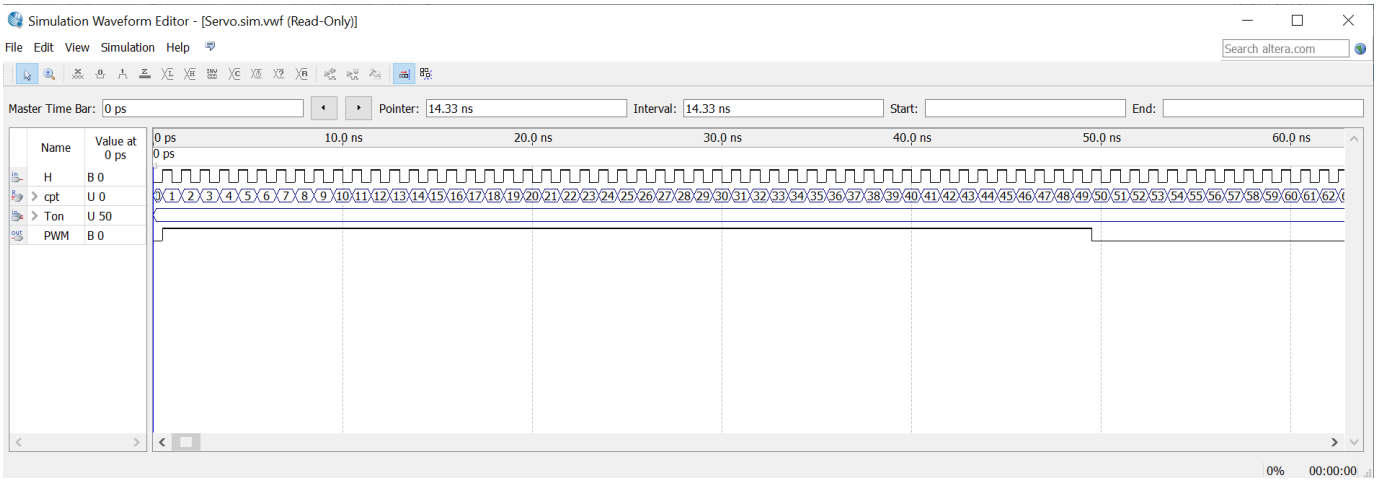
Nous avons ainsi créé une variable compteur « cpt » allant de 0 à 2000. Cette variable correspond au nombre d'impulsion du signal d'horloge. Elle sera donc incrémentée à chaque front montant sur l'horloge et sera remise à 0 lorsqu'elle atteindra 1999, c'est-à-dire la valeur maximale du nombre de front montant sur une période. On choisit 1999 car on commence à compter à partir de 0.

Ensuite le signal du PWM sera à l'état haut lorsque la valeur de cpt sera inférieur à la valeur de $(T_{On})_{10}$. Le signal PWM vaudra 0 dans le cas contraire.

Nous devons maintenant effectuer une simulation afin de vérifier si notre code est bon. Il faut donc vérifier que non avons bien 2000 impulsions d'horloge et que la durée du signal à l'état haut correspond bien à la valeur entrée sur $T_{On}$.

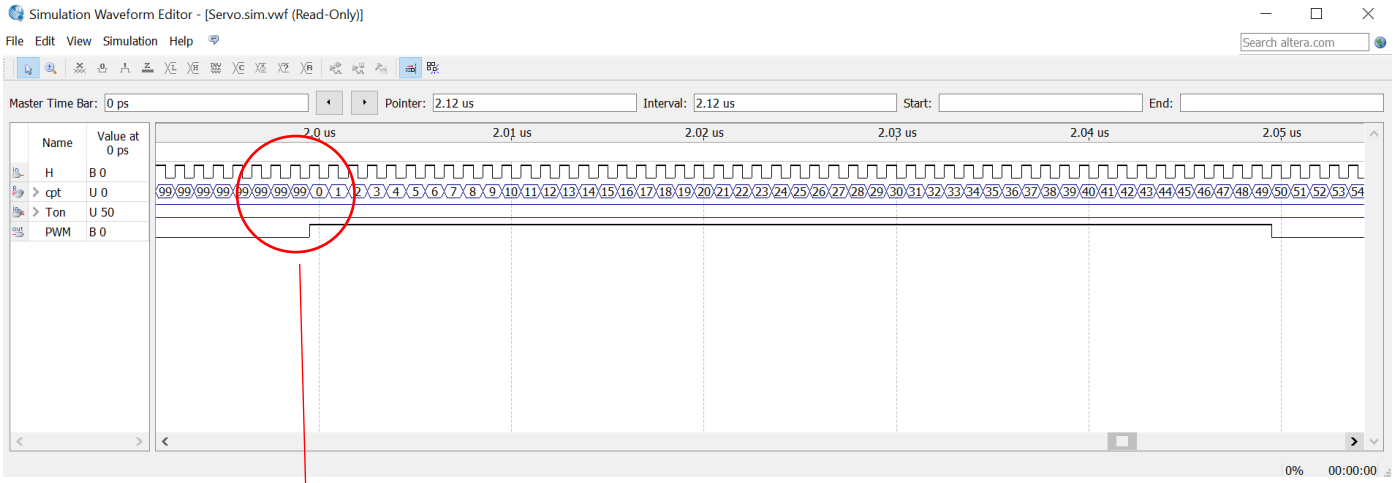
Cependant pour les simulations, Quartus ne peut simuler sur plus de 6µs. Or, notre période du PWM est de 20ms et la période de l'horloge est réglé à 10µs. Etant donné que nous voulons compter un nombre d'impulsion sur le signal d'horloge, nous allons régler l'horloge de la simulation à 1ns car $1.10^{-9} * 2000 = 2.10^{-6} = 2\mu s < 6\mu s$.

Ainsi, pour une valeur de 50 sur $T_{On}$ (correspondant à un angle de 0°) on obtient :

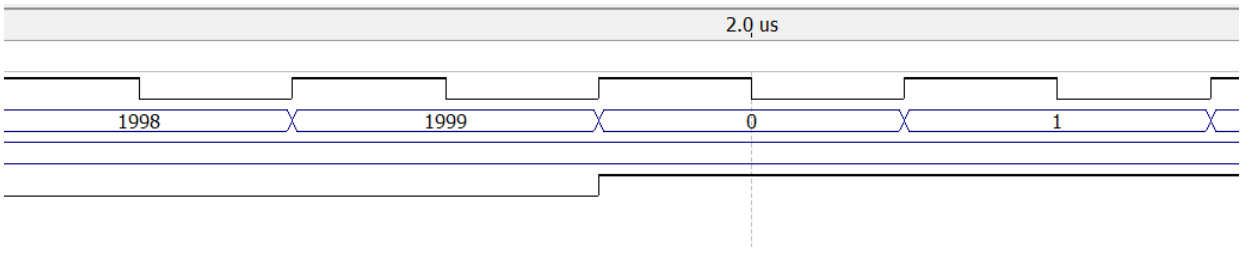



Début de la simulation pour un angle de 0°

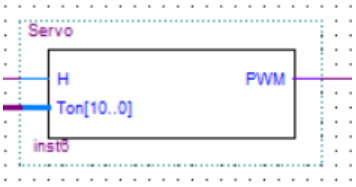
On constate que nous avons un signal à l'état haut pendant 50 impulsions d'horloge



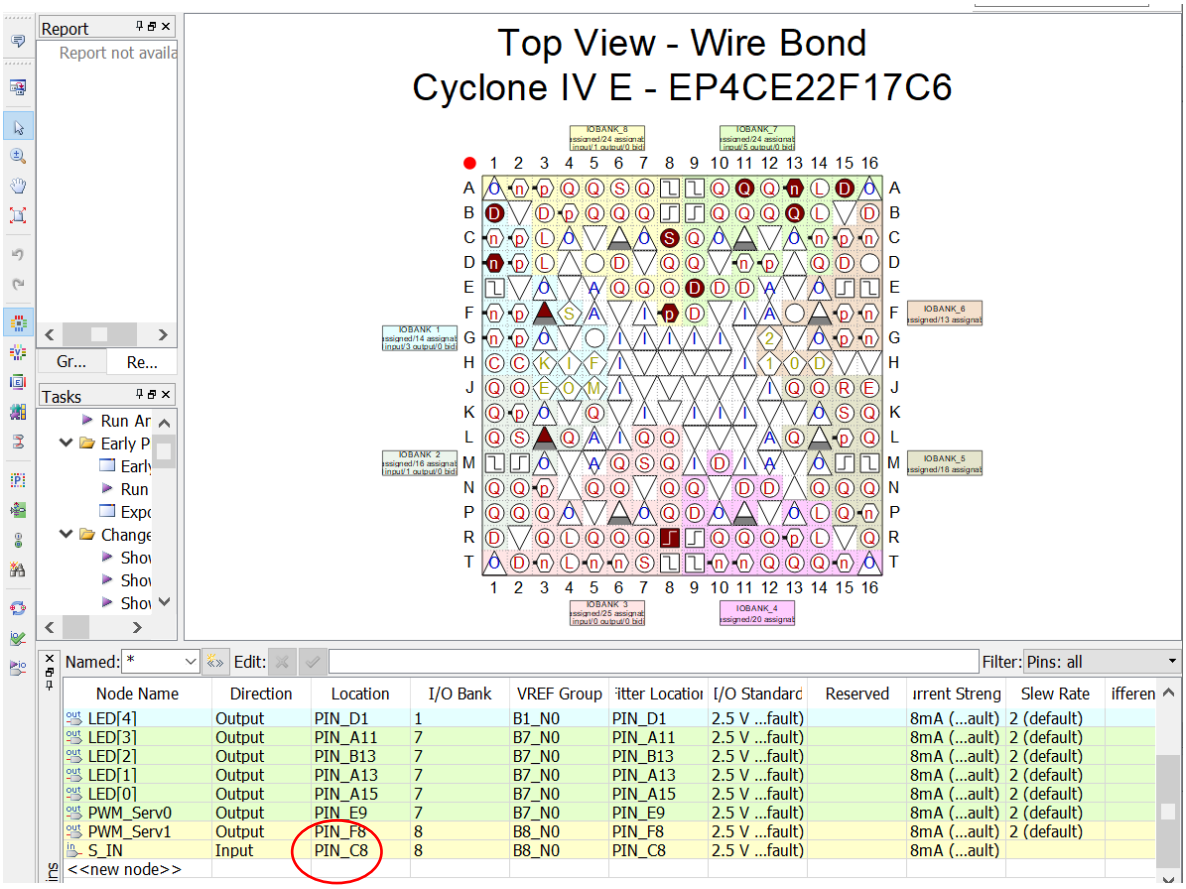

Milieu de la simulation pour un angle de 0°



Créons ensuite le symbole correspondant :




Nous allons alors mettre sur l'entrée H la sortie $T_{1us}$, sur $T_{on}$ la valeur de 50 en binaire sur 11 bits, soit $(000\ 0011\ 0010)_2$ et nous connecterons la sortie PWM au PIN_E9.




Maintenant, pour s'assurer que nous allons bien envoyer le bon signal dans le servomoteur (au risque de l'endommager), nous allons visualiser le signal correspondant sur l'oscilloscope à l'aide d'une sonde oscilloscope. Pour ce faire, nous allons débrancher le servomoteur au niveau de la carte DEO Nano. Nous allons de plus connecter la sortie PWM au PIN_F8 de la carte DEO Nano pour y placer la sonde oscilloscope.

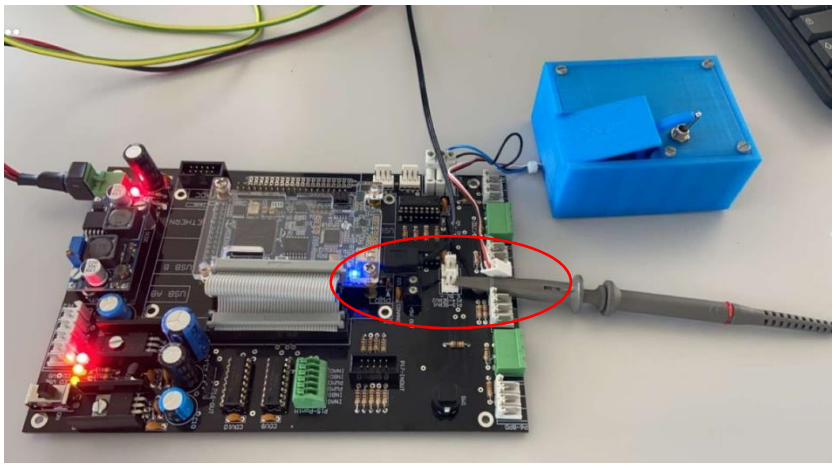
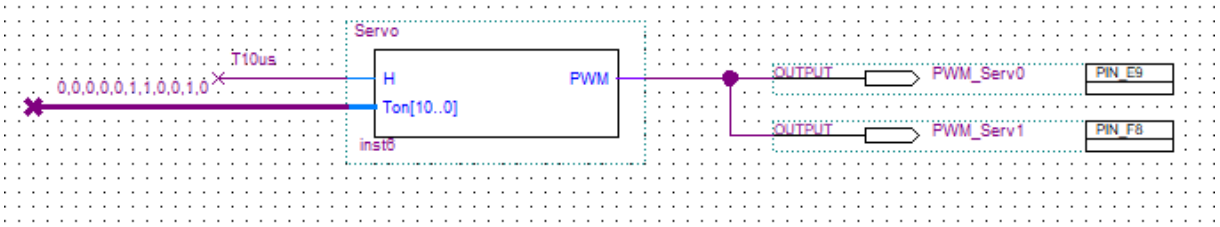
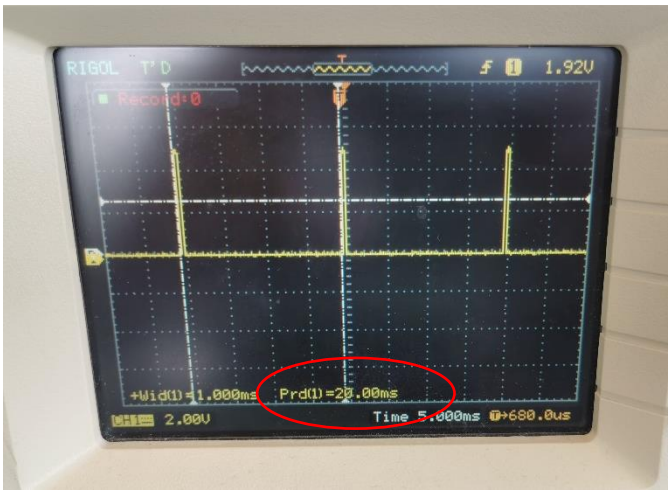


Photo sur seromoteur débranché et de la sonde oscilloscope

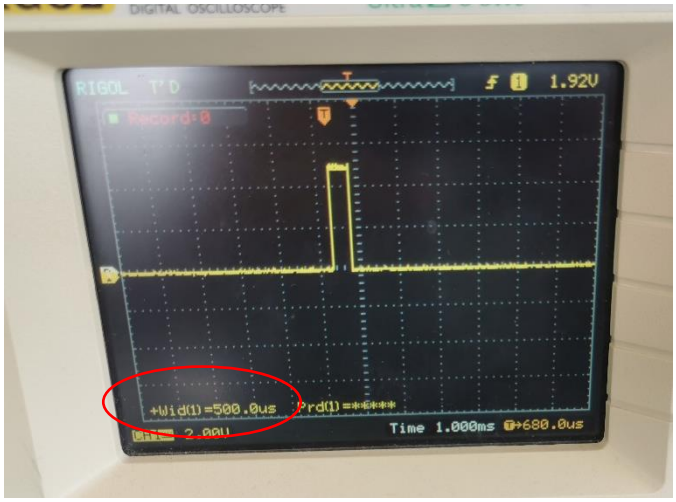
Sur le bus de $T_{on}$, plaçons la valeur binaire $(000\ 0011\ 0010)_2 = (50)_{10}$ correspondant à un angle de 0°.




Nous obtenons le signal suivant :




Signal sur 3 périodes

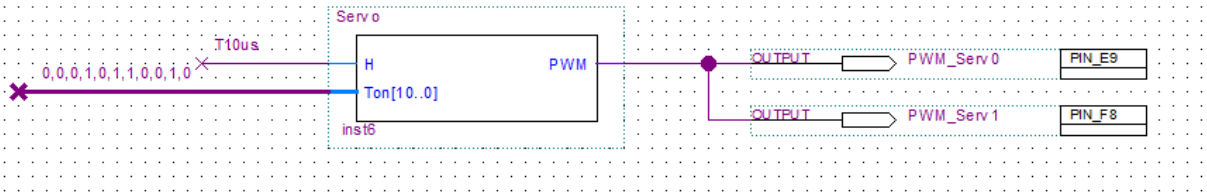



Signal sur 1 période

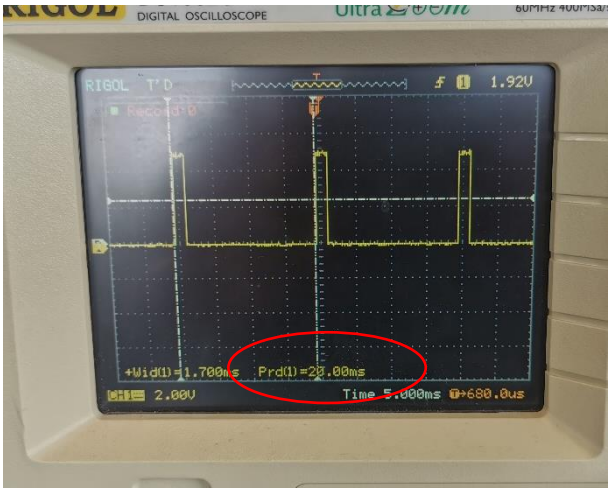
On constate que nous avons bien les caractéristiques du signal attendu, soit une période de 20ms et un temps à l'état haut de 0,5ms. Nous pouvons donc envoyer le programme dans la carte en rebranchant le servomoteur sur la DEO Nano.

Le bras ne sort donc pas car l'angle de 0° correspond à la position où le bras est rentré dans la UselessBox.

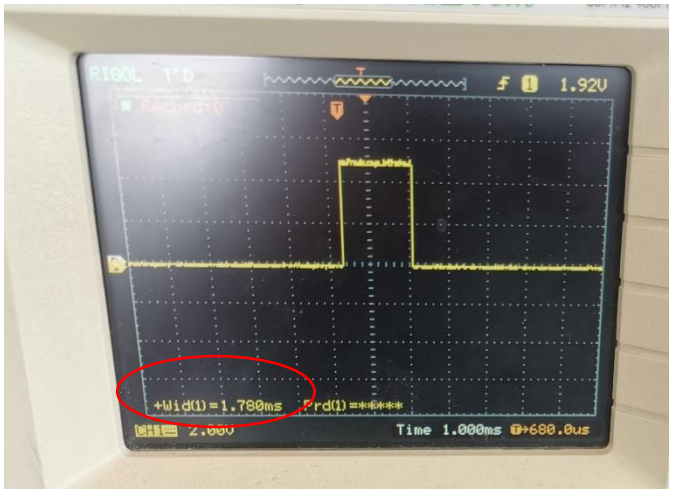
De même, nous allons vérifier le signal pour une valeur de de $T_{on} = (000\ 1011\ 0010)_2 = (178)_{10}$ correspondant à un angle de $116°^*$.




Sur l'oscilloscope, nous obtenons :




Signal sur 3 périodes




Signal sur 1 période

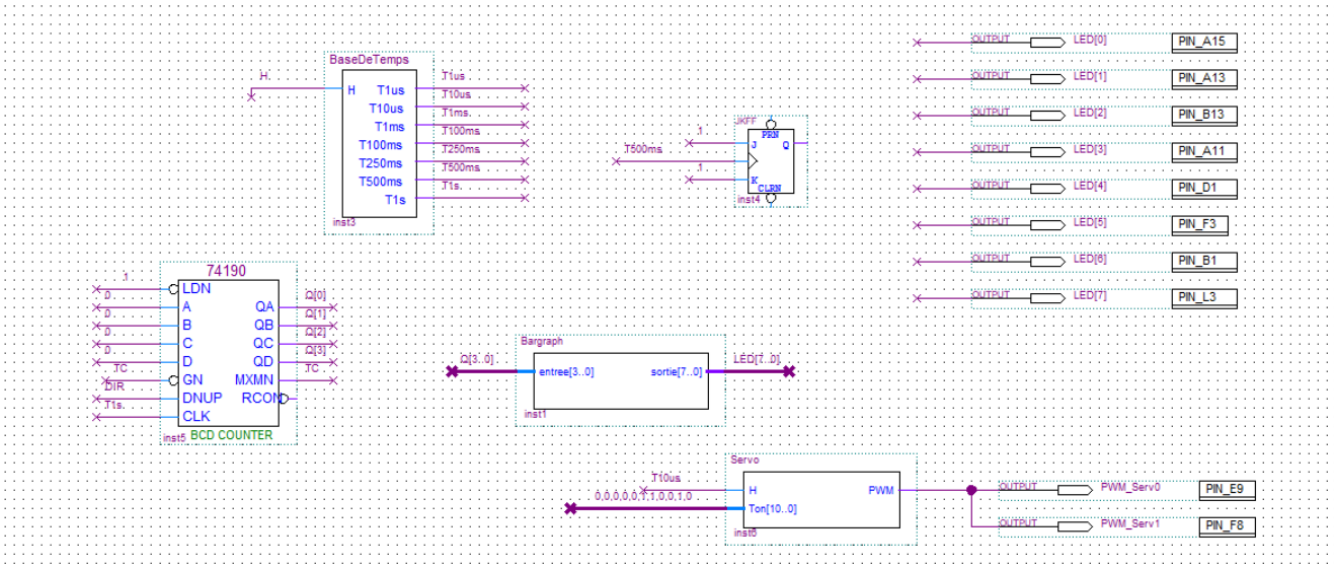
<u>Remarque</u> : Les photos sur l'oscilloscope ont été prises après quelques tests sur la UselessBox. Le signal envoyé dans le servomoteur pour sortir le bras ne correspond pas à un angle de 120° mais à un angle de 116°. En effet, pour un angle de 120°, le bras appuyé un peu trop fort sur l'interrupteur. On peut donc déterminer le temps à l'état haut pour une valeur d'angle de 116° : $\frac{116*111}{10} + 500 = 1{,}78ms$.

Nous constatons que les signaux sur l'oscilloscope correspondent au signal attendues.

Ainsi, selon la valeur que nous entrons sur $T_{on}$ :

- $(000\ 0011\ 0010)_2 = (50)_{10}$ : Le bras mécanique n'est pas sorti.
- $(000\ 1011\ 0010)_2 = (178)_{10}$ : Le bras sort et pousse l'interrupteur.

Dans le projet TopUselessBox, nous obtenons la schématique suivante :

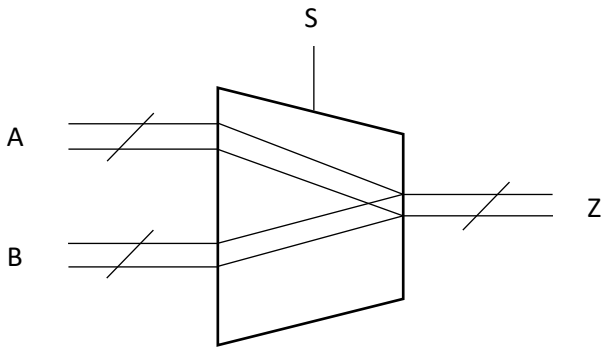


## Multiplexage

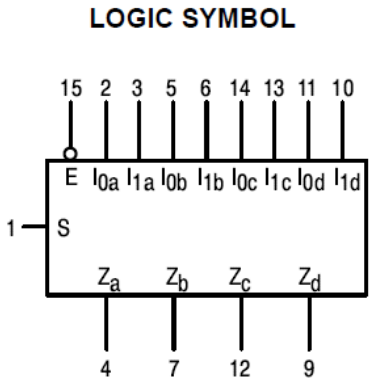
Le principe ici du multiplexage est de pouvoir envoyer sur l'entrée $T_{on}$ la valeur correspondant au bras ouvert ou fermé en fonction de la position de l'interrupteur mais aussi du décompte des LED.

Un multiplexeur 2 bus vers 1 est composé de 2 bus d'entrée (A et B), un « common select » et un bus de sortie (Z). En fonction de la valeur du « common select » (0 ou 1), le bus de sortie Z va prendre la valeur du bus A ou du bus B.

Le symbole d'un multiplexeur est le suivant :




Dans notre projet, nous utiliserons le modèle 74LS157, dont chaque bus est composé de 4 bits. Voici le symbole de ce multiplexeur extrait de la datasheet :




Sur ce symbole on peut identifier :

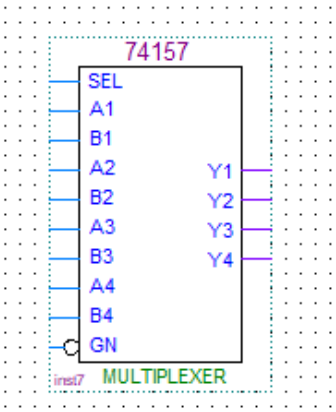
- $I_{0A} - I_{0D}$ correspond au bus d'entrée de 4 bits de la source 0.
- $I_{1A} - I_{1D}$ correspond au bus d'entrée de 4 bits de la source 1.
- $Z_A - Z_D$ correspond au bus de sortie sur 4 bits.
- $S$ est l'entrée correspondant au « Common select ».
- $E$ est une entrée inverseuse pour forcer la valeur de sortie à 0.

Ainsi le fonctionnement de ce multiplexeur est le suivant :

- Si $\bar{E}$ vaut 0 :
    - Si S vaut 0 : Le bus de sortie prend la valeur du bus d'entrée de la source 0.
    - Si S vaut 1 : Le bus de sortie prend la valeur du bus d'entrée de la source 1.

- Si $\bar{E}$ vaut 1 : Les 4 bits du bus de sortie sont forcés à 0.

Nous pouvons retrouver ce composant dans Quartus, en recherchant « 74157 », afin d'obtenir le symbole suivant :




<u>Pour les entrées</u> :

- SEL correspond au « Common Select », noté S sur la datasheet.
- A1, A2, A3, A4 correspondent aux bits d'entrée du bus de la source 0, noté $I_{0A} - I_{0D}$ sur la datasheet.
- B1, B2, B3, B4 correspondent aux bits d'entrée du bus de la source 1, noté $I_{1A} - I_{1D}$ sur la datasheet.
- L'entrée inverseuse GN correspond à l'entrée $\bar{E}$ de la datasheet.

<u>Pour les sorties</u> :

- Y1, Y2, Y3, Y4 correspondent aux bits du bus de sortie, noté $Z_A - Z_D$ sur la datasheet.

Cependant, nous pouvons constater que les 3 bus de ce composant disposent de 4 bits. Or, le but du multiplexeur est de pouvoir envoyer sur l'entrée $T_{on}$ du servomoteur la valeur de 50 ou 178 en binaire sur 11 bits. Nous devons donc associer plusieurs multiplexeurs afin d'obtenir au minimum 11 bits. Pour associer les multiplexeurs, il suffit de relier toutes les entrées SEL entre elle.

Au premier regard, nous pourrons choisir d'utiliser 3 multiplexeurs, pour avoir au maximum 12 bits. Dans cette situation, nous pourrions écrire la valeur de $(000\ 0011\ 0010)_2 = (50)_{10}$ à l'aide de chaque entrée de la source 0 et laisser la 1ère sortie du 1er multiplexeur, car seulement 3 bits peuvent être utilisés.

De même nous pourrions écrire la valeur de $(000\ 1011\ 0010)_2 = (178)_{10}$ à l'aide de chaque entrée de la source 1 et laisser la 1ère sortie du 1er multiplexeur, car, ici aussi seulement 3 bits peuvent être utilisés.

Cependant, on constate que le premier multiplexeur devra avoir en sortie, sur les bits Y2, Y3 et Y4 la valeur de 0. Par simplification, nous pouvons supprimer ce multiplexeur et écrire sur l'entrée $T_{on}$ que les 3 premiers bits seront 0.

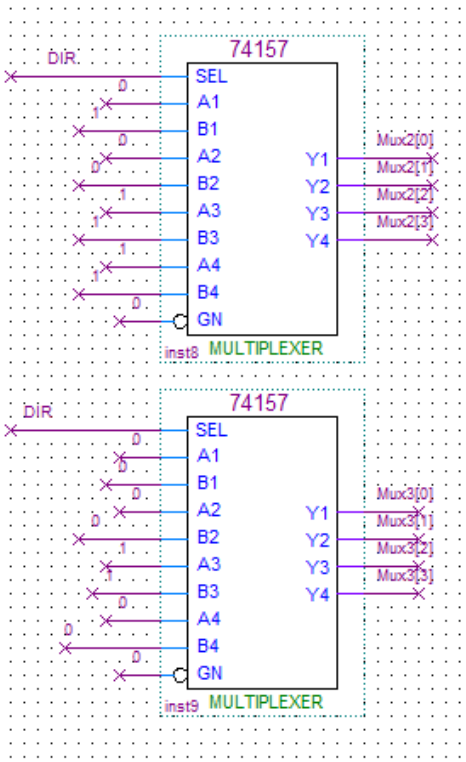
Nous nommerons également les sorties de chaque multiplexeur d'une telle façon à pouvoir les reconnaitre facilement sur le bus d'entrée de $T_{on}$.

Ainsi, une table de vérité complète de ce montage serait :

| | | Multiplexeur 1 | | | | | | | | Sortie | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $\overline{GN}$ | SEL | A1 | A2 | A3 | A4 | B1 | B2 | B3 | B4 | Y1 | Y2 | Y3 | Y4 |
| 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 |
| 1 | X | X | X | X | X | X | X | X | X | 0 | 0 | 0 | 0 |

| | | Multiplexeur 2 | | | | | | | | Sortie | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $\overline{GN}$ | SEL | A1 | A2 | A3 | A4 | B1 | B2 | B3 | B4 | Y1 | Y2 | Y3 | Y4 |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| 1 | X | X | X | X | X | X | X | X | X | 0 | 0 | 0 | 0 |

On obtient alors les deux multiplexeurs comme ci-dessous :




En lisant de haut en bas les valeurs sur les entrées A :

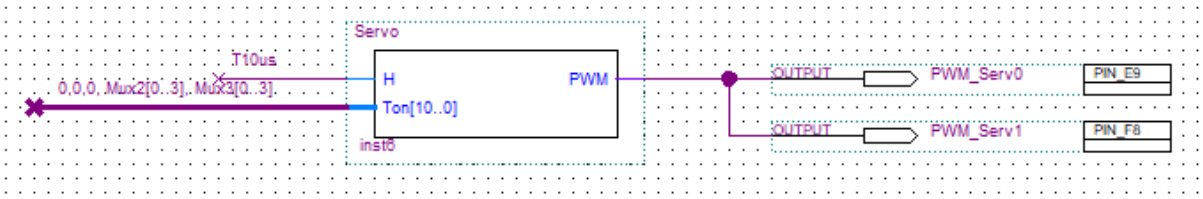
- $(0011\ 0010)_2 = (50)_{10}$

De même sur les entrées B :

- $(1011\ 0010)_2 = (178)_{10}$

Les entrées SEL sont connectées à la sortie DIR de l'interrupteur.

Ainsi, on peut écrire ces valeurs sur le bus $T_{on}$ de la façon suivante :




Avant d'envoyer ce nouveau programme dans la carte, nous vérifions à l'oscilloscope si les signaux n'ont pas changé et s'ils changent lorsque l'on bouge la position de l'interrupteur. Nous obtenons bien les mêmes signaux que précédemment.

Cependant, à cette étape nous allons constater que nous avons inverser les sources d'entrées qui sont recopier en sortie en fonction de SEL. Pour ne pas tout modifier, nous placerons un inverseur à la sortie de l'interrupteur, de la manière suivante :


Nous pouvons alors envoyer le code dans la Useless Box. On constate que l'interrupteur sort et rentre correctement.

Nous devons à présent améliorer ce programme pour que l'interrupteur sorte après le décompte des 8 LED. En effet, lorsque l'on enclenche l'interrupteur, les 8 LED doivent s'allumer et s'éteindre une par une chaque seconde.

Pour réaliser ce processus, nous allons devoir nous servir de la sortie TC du compteur, qui passe à l'état haut à la fin du comptage ou du décomptage.

L'entrée SEL du multiplexeur permet de déterminer quelle valeur binaire doit se trouver sur le bus de sortie.

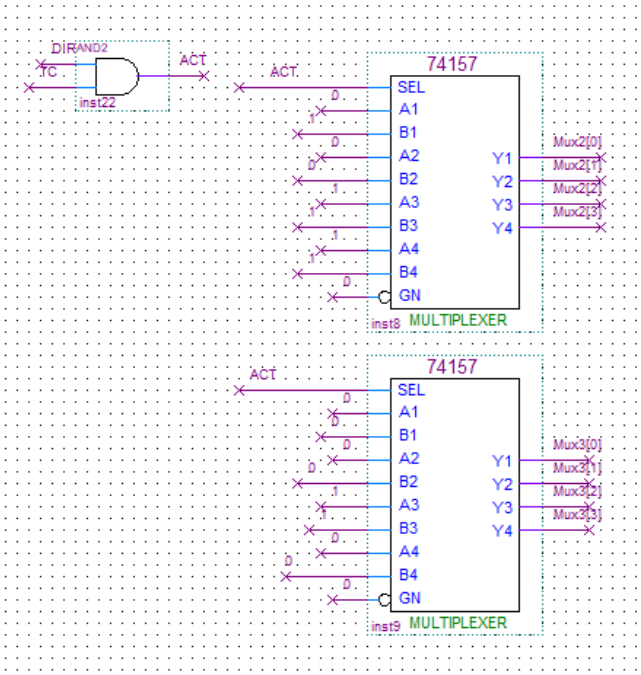
Nous allons donc réaliser une table de vérité pour mieux comprendre quelles sont les conditions nécessaires pour faire basculer l'entrée SEL à 0 ou à 1.

| DIR | TC | SEL |
|---|---|---|
| 1 | 1 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 0 | 0 | 0 |

Nous pouvons constater que l'entrée SEL doit passer à 1 seulement si $DIR = 1$ et $TC = 1$. En effet, $SEL = 1$ signifie que les bus recopiés sur les sorties sont ceux permettant de faire sortir le bras. Dans les autres cas, il nous faut $SEL = 0$.

On peut donc conclure que $SEL = DIR\ ET\ TC$.

Ainsi, nous allons envoyer cette formule logique sur les entrées SEL des multiplexeurs de la manière suivantes :




Lorsque nous envoyons ce nouveau programme dans la carte, nous obtenons bien le résultat attendu. Lors de l'enclanchement de l'interrupteur, le compteur décompte en éteignant une par une les 8 LED. A la fin du décompte, le doigt mécanique repousse l'interrupteur à sa position de départ et les LED se rallument une par une.

# Conclusion du projet Useless Box

Pour conclure, ce projet nous a permis de pratiquer la programmation VHDL d'une DEO Nano pour reproduire le fonctionnement de base de la Useless Box à notre disposition. Nous avons également pu nous servir du logiciel Quartus tout au long de ce projet.

Le projet nous a permis de refaire une base de temps comme nous avons vu au 1er semestre, de mieux comprendre comment fonctionne un bargraph (ou chenillard) et nous avons découvert ce qu'est un servomoteur. Nous avons alors mis en application le multiplexeur.

Le résultat de base de cette Useless Box a été refait. Cependant, nous aurions pu effectuer quelques améliorations, comme réduire la latence de l'extinction de la première LED ou alors pouvoir choisir à quelle vitesse les LED doivent s'éteindre ou s'allumer. Nous n'avons pas pu réaliser ces améliorations par manque de temps.

Le projet Useless Box fut un projet intéressant par le fait qu'il nous a fallu développer une démarche scientifique pour aller au bout des étapes et donc pour avoir une solution fonctionnelle.