Gaspard BONNET-SAINT-GEORGES

Augustin LAPRAIS

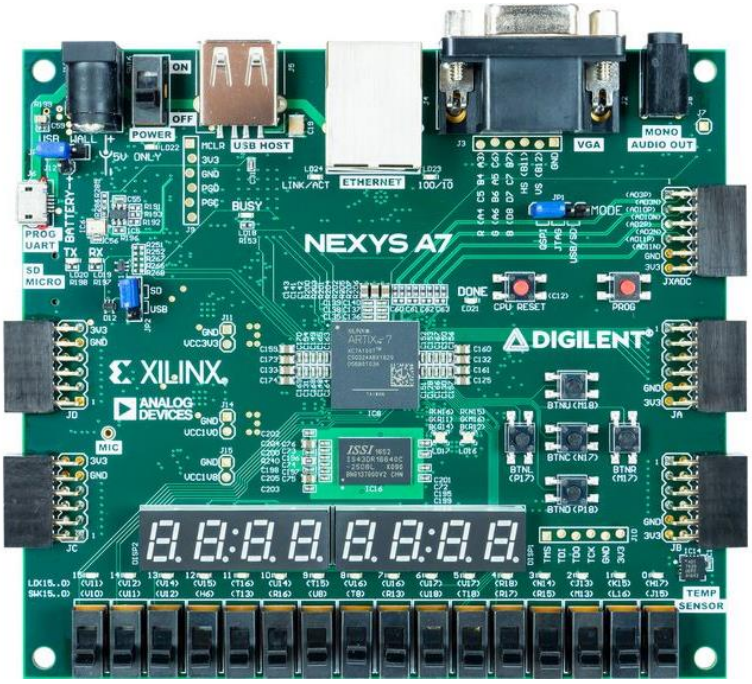
Audrey NICOLLE

Thibault WEBER

CPE LYON

Projet Scoring 2.0

Affichage de la durée et du score d'un match de football à l'aide d'un FPGA



Professeurs :

Abdelbassat MASSOURI

Bastien DEVEAUTOUR

Table des matières

Introduction

Peu importe le sport, un panneau d'affichage du score et du temps est crucial pour que le public et les athlètes puissent suivre le cours de la partie. Au cours de notre projet, 3 séances de TP de quatre heures, nous avons donc essayé de programmer ce tableau d'affichage. Lors de la première séance, nous avions pour objectif de mettre en œuvre un système d'affichage sur les afficheurs 7-segements de la carte. Puis lors de la séance suivante, nous devions mettre en œuvre un système de comptage pour le score et le temps. Enfin, la dernière séance, le but était de regrouper touts les modules, valider leur bon fonctionnement et tester le programme sur la carte. La carte sur laquelle nous avons travaillé est la carte NEXYS A7 alimenté sous 4,5V-5,5V, et nous avons programmée en VHDL, sous le logiciel ISE XILLINX. Pour programmer la carte, nous avons chargé le fichier *Xillinx Bitstream*, obtenu à partir du logiciel de programmation. Ce fichier a été chargé via un câble USB-A.

Lors de ce projet, nous avons dû mettre en place des boutons pour activer les différentes fonctions du Chronoscore comme vous pouvez voir sous la figure 1.1 ci-dessous.

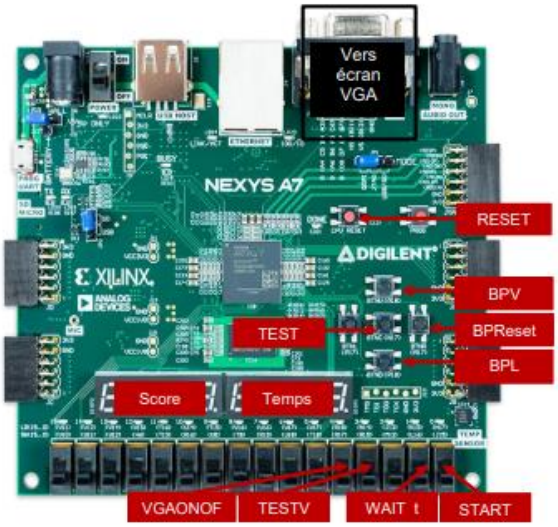


Figure 0.1 : Carte NEXYS A7.

Les fonctions utilisées dans le projet sont les suivantes :

- *RESET* qui permet de réinitialiser le chronomètre.

- *START qui* permet de lancer le chronomètre.et

- *WAIT_t* qui de mettre le chronomètre en pause.

- *BPL, BPV* qui permettent d'incrémenter respectivement le score de l'équipe local et l'équipe visiteur.

- *BPreset* qui permet de remettre le score des deux équipes à 0.

I) Architecture du module ***chronoscore***

Le module ***chronoscore*** est le fichier top de notre système. Il est composé comporte 11 signaux d'entrée. On retrouve :

- *GCLK* : oscillateur qui permet de générer une fréquence de 100MHz
- *START* : démarrage du chronomètre
- *WAIT_t* : mise en pause du chronomètre
- *RESET* : remise à zéro du chronomètre
- *BPL* : incrémentation du score pour l'équipe locale
- *BPV* : incrémentation du score pour l'équipe visiteur
- *BPreset* : remise à zéro du score
- *VGA ONOFF* : activation de l'écran VGA
- *TEST VGA* : activation des images de test de l'écran VGA
- *TEST_HSLS* : non étudié ici
- *TEST :* non étudié ici

Ensuite, ***chronoscore*** dispose de 7 signaux de sortie. On retrouve :

- *AN [7 :0]* : signaux de commande des anodes des afficheurs 7-segments
- *LEDS [6 :0]* : signaux de commande des cathodes des afficheurs 7-segments (segments a à g)
- *LED [7]* : signal de commande du point entre les afficheurs 7-segments
- *HSYNCH :* synchronisation horizontale des afficheurs 7-segments
- *VSYNCH :* synchronisation verticale des afficheurs 7-segments
- *RED :* contrôle des pixels de couleur rouge
- *BLUE :* contrôle des pixels de couleur bleu
- *GREEN* : contrôle des pixels de couleur vert

NB : *LEDS [6 :0]* et *LED [7]* représentent une seule sortie

Après avoir présenté les signaux d'entrée et de sortie de ***chronoscore***, nous allons maintenant nous intéresser à sa décomposition en sous-modules. Le système ***chronoscore*** se découpe alors en cinq sous-modules représentés ci-dessous (figure 1.1) :

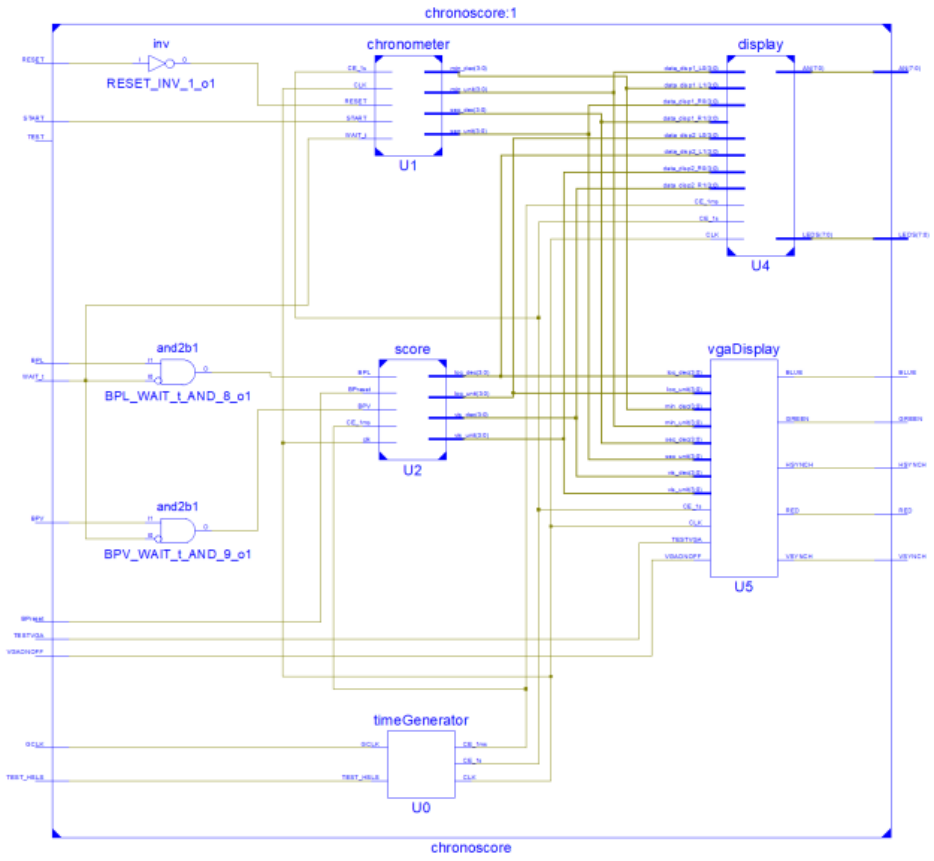


*Figure 1.1 : Architecture du module **chronoscore**.*

On retrouve alors les blocs suivants :

- *timeGenerator*
- *display*
- *vgaDisplay*
- *chronometer*
- *score*

Tout d'abord, le module ***timeGenerator*** permet de garder tout le système ***chronoscore*** synchrone. Ci-dessous son architecture interne :

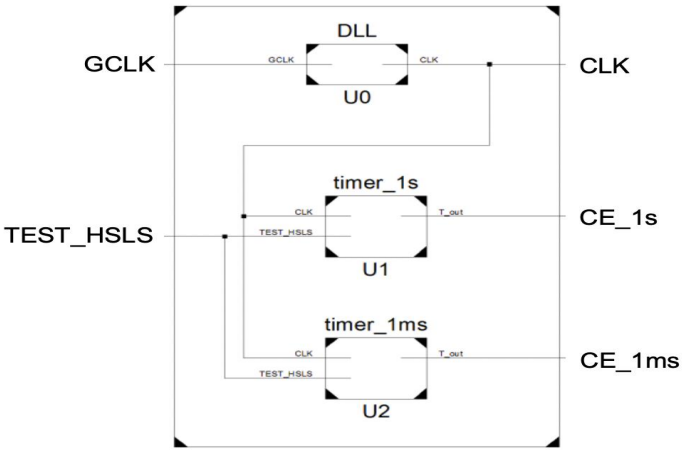


*Figure 1.2 : Architecture du module **timeGenerator**.*

En effet, l'une de ses sorties est une horloge CLK, qui va commander tous les autres sous-systèmes. Le bloc *timer_1ms* génère un signal périodique *CE_1ms* de période 1ms et le bloc *timer_1s* génère un signal périodique *CE_1s* de période 1s. Ce module ne sera pas expliqué plus précisément par la suite car ce n'est pas nous qui l'avons programmé.

Ensuite, le module ***display*** permet de gérer les données à afficher sur les 8 afficheurs 7-segments de la carte. Les quatre afficheurs de droite (disp1 : afficheurs 0 à 3) doivent indiquer le temps écoulé en minutes et secondes. Les quatre afficheurs de gauche (disp2 : afficheurs 4 à 7) doivent indiquer le score de l'équipe locale et celui de l'équipe des visiteurs.

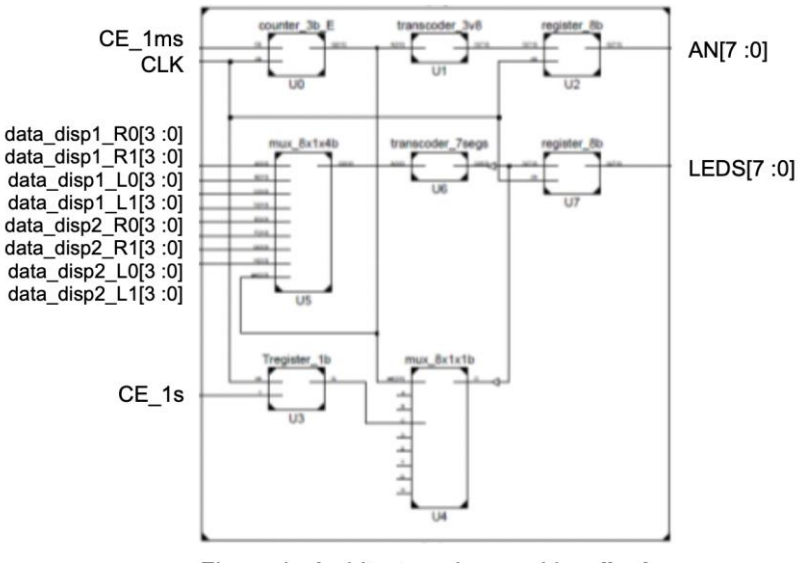


*Figure 1.3 : Architecture du module **display**.*

Le sous-bloc ***vgaDisplay*** récupère les données temporelles et le score grâce aux signaux *min_dec, min_unit, loc_dec, loc_unit, sec_dec, sec_unit, vis_dec, vis_unit*. Puis grâce à cela génère l'affichage d'images auto-générées de ces données à l'aide des deux signaux de synchronisation *HSYNCH* et *VSYNCH* et gère la couleur grâce aux signaux *RED, BLUE, GREEN*. La description précise de ce bloc ne sera pas présentée par la suite car ce n'est pas nous qui avons réalisé ce module.

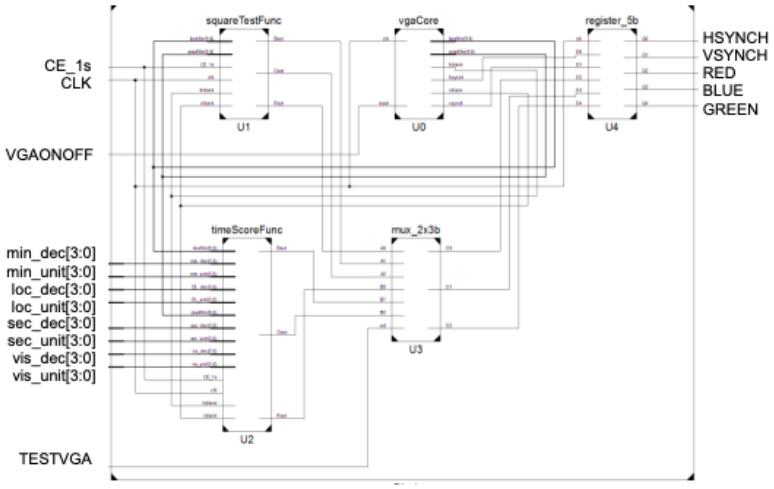


*Figure 1.4 : Architecture du module **vgaDisplay**.*

Pour la gestion du temps, nous avons créée le bloc ***chronometer***. Ce bloc permet de fournir aux modules ***display*** et ***vgaDisplay***, les modules de gestion de l'affichage, les données temporelles, soit minutes et secondes à afficher. Ces données temporelles sont transmises via les signaux *sec_unit*, *sec_dec, min_unit* et *min_dec*. L'architecture du module est la suivante :

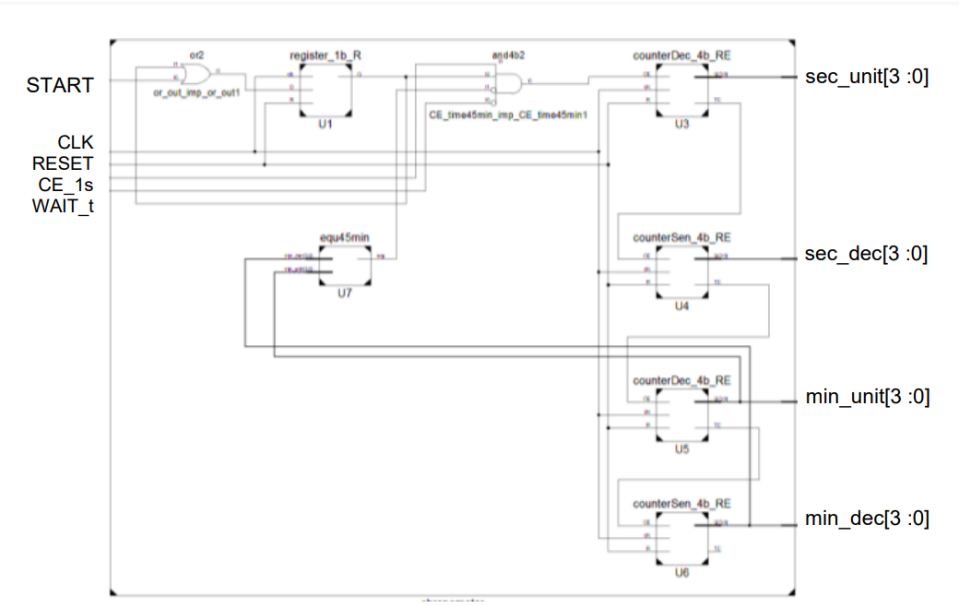


*Figure 1.5 : Architecture du module **chronometer**.*

Enfin, le module ***score*** permet de fournir au module ***display*** et ***vgaDisplay***, les données du score de l'équipe visiteur et l'équipe local à afficher (unités et dizaines). Ces données sont transmises via les signaux *loc_unit*, *loc_dec* (local) et *vis_unit*, *vis_dec* (visiteur).
L'architecture du module est la suivante :

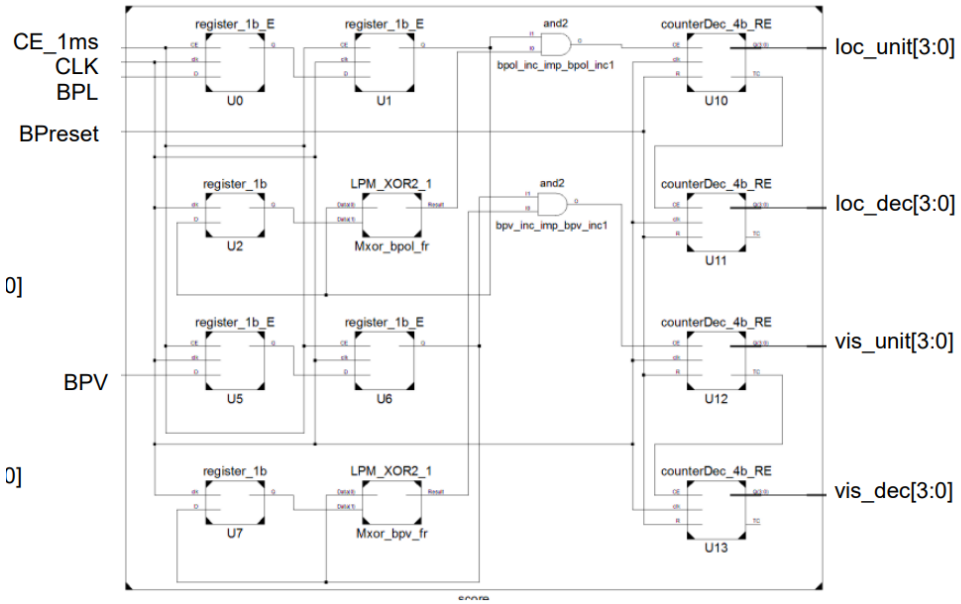


*Figure 1.6 : Architecture du module **score**.*

Les architectures des différents modules se trouvent en version pleine page en annexe (p.19-21)

II) Sous-module *display*

II.1) Présentation du sous-module

Ce sous-module permet de gérer les données à afficher sur les 8 afficheurs 7-segements de la carte *NEXYX A7*. Le temps écoulé en minutes et secondes, ce qui correspond aux signaux en entrée du module de type *data_disp1* (voir figure 3.1), est affiché sur les quatre afficheurs de droite. L'affichage du score des deux équipes, ce qui correspond aux signaux en entrée du module du type *data_disp2* (voir figure 3.1), locale et visiteurs, se fait, quant à lui, sur les quatre afficheurs de gauche. L'architecture du module est la suivante :

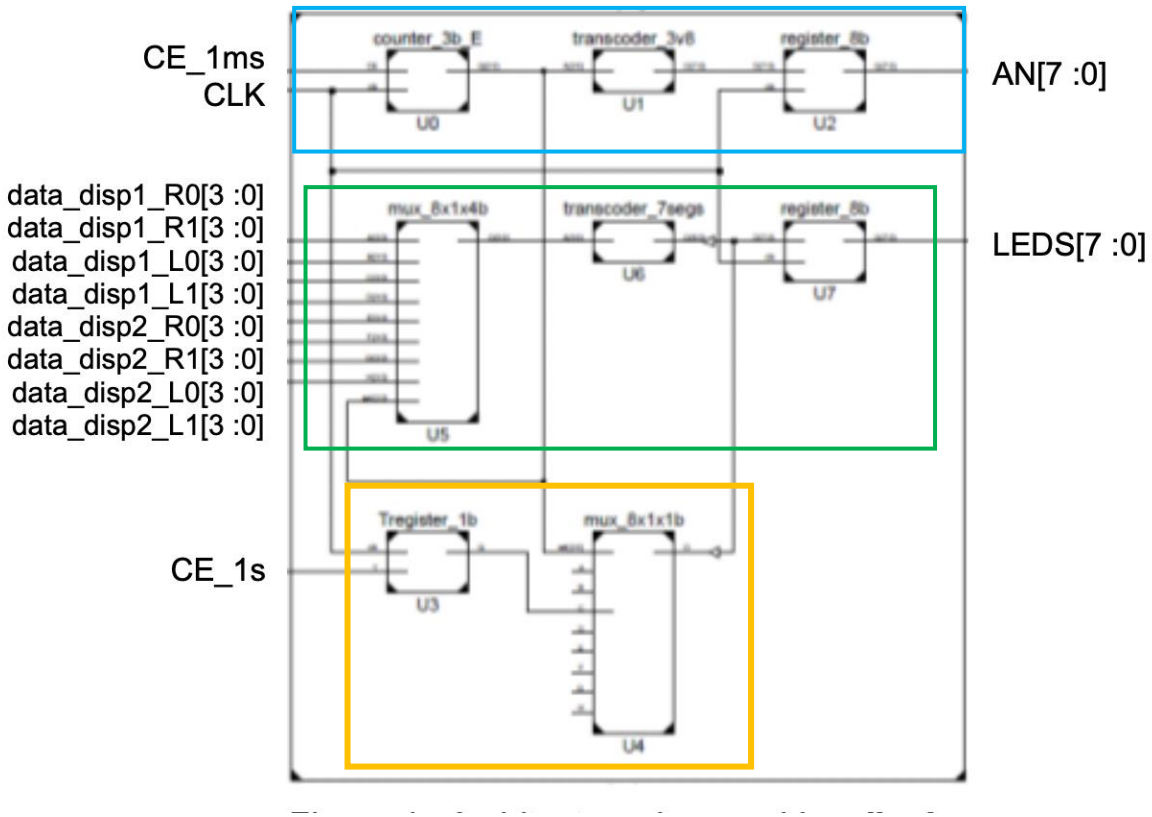


Figure 2.1 : Architecture du sous-module ***display***

Ainsi, ce module permet d'afficher le temps et le score envoyé à notre module via les signaux de type *data* depuis les autres sous-modules ***chronometer*** et ***score***. La sortie *AN* permet de sélectionner l'anode de l'afficheur et la sortie *LEDS* permet de sélectionner les segments de l'anode sélectionnée que l'on souhaite allumer. La sélection de chaque anode se fait toutes les millisecondes grâce au signal *CE_1ms* ce qui permet de donner cette impression d'affichage constant des résultats pour l'œil humain. Seul l'affichage du point pour différencier l'espace sur l'afficheur de droite dédié aux minutes et aux secondes se fait moins souvent à l'aide du signal *CE_1s,* ce qui permet à l'utilisateur de le voir clignoter. On peut séparer ce module en deux parties, une partie (encadré en bleu) qui permet de sélectionner l'anode sur l'afficheur et l'autre partie (encadré en vert) qui permet de

sélectionner les segments de l'anode à allumer. L'affichage du point est géré à l'aide de ces deux parties et aux blocs U3 et U4 (encadré en orange).

II.2) Comment sélectionne-t-on l'anode ?

Cette sélection est faite à l'aide des blocs U0, U1 et U2 (voir figure 2.1). En effet, le compteur *counter_3b_E* (U0) permet de sélectionner quelle anode va être allumé en fournissant un chiffre de 0 à 7 en binaire ce qui correspond bien au fait que l'on a 8 anodes disponibles avec les deux afficheurs. Ce compte est fait toutes les millisecondes, à l'aide du signal *CE_1ms* qui est relié à l'entrée *CE* (count enable) de notre compteur. Cette entrée est synchrone à l'horloge de notre compteur qui est relié au signal horloge *CLK* du module. Ainsi, lorsque le signal CE_1ms est à l'état haut et qu'il y a front montant pour l'horloge de notre compteur, celui-ci incrémente sa sortie de 1 jusqu'à 7 avant de recommencer.

Ensuite, la sortie de ce compteur est reliée à l'entrée de la fonction combinatoire *transcoder_3v8* (U1) qui est un décodeur, contrairement à ce que suppose son nom. Ce décodeur met à l'état bas le numéro du signal de sortie qui correspond à l'équivalent décimal du code binaire sur trois bits appliqués sur ses entrées. Donc son signal de sortie est dans notre cas codé sur 8 bits. Par exemple, si l'on a en entrée le code binaire « 010 », le code de sortie sera « 11111011 ».

Enfin, il y a la fonction *register_8b* (U2). C'est un registre synchrone constitué de 8 bascules D actives sur front montant du signal d'horloge CLK. Ce registre permet la synchronisation des données qui permettent de sélectionner l'anode avec les données qui permettent de sélectionner les segments à afficher.

Le programme de ces fonctions et leur simulation se trouvent en annexe.

II.3) Comment sélectionne-t-on les segments de l'anode à allumer ?

Cette sélection est faite à l'aide des blocs U5, U6, U7 (voir figure). En effet, le multiplexeur *mux_8x1x4b* reçoit en continue en entrée les données à afficher. À l'aide du signal issu du compteur U0, il sait sur quelle anode va se faire l'affichage, c'est-à-dire que ce signal lui permet de électionner le signal de type *data* à fournir en sortie. Ensuite ce signal est donné en entrée à la fonction *transcoder_7segs.* Se transcoder permet de traduire le code binaire sur 4 bits reçu en entrée en un code sur 7 bits où les valeurs à l'état bas représente le segment allumé. En effet, les afficheurs fonctionnent selon le schéma suivant :

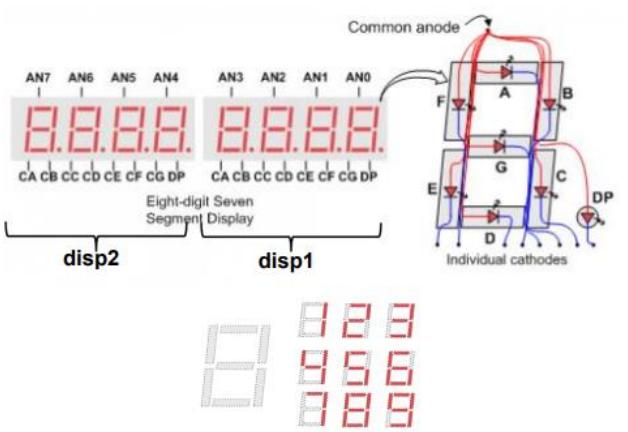


Figure 2.2 : Fonctionnement des afficheurs

Par exemple, si le transcoder reçoit le code binaire suivant « 1000 », alors en sortie du transcoder on aura le code suivant « 0000000 » ce qui affichera le chiffre 8 sur l'afficheur. En effet, on aura allumé tous les segments de l'anode. La table de vérité du transcoder se trouve en annexe pour plus de précision.

Ensuite, de même que pour le bloc qui sélectionne l'anode, le signal de sortie est envoyé en entrée d'un registre *register_8b* qui permet la synchronisation des données sélectionnent les segments avec les données qui sélectionne l'anode.

Le programme de ces fonctions et leur simulation se trouvent en annexe.

II.4) Comment fonctionne l'affichage du point ?

L'affichage du point se fait toutes les secondes de façon que l'utilisateur puisse le voir clignoter. Pour cela, on utilise deux blocs, la bascule T *Tregister_1b* (U3) et le multiplexeur *mux_8x1_1b* (U4). En effet, la bascule T a pour entrée le signal *CE_1s* ce qui fait que la bascule T en sortie fournit un signal qui est à l'état bas ou haut pendant une seconde. Ce signal est en entrée du multiplexeur. Ce multiplexeur à comme signal de sélection le signal issu du compteur qui permet de sélectionner l'anode. Ainsi, tous les signaux des anodes où l'on ne veut jamais afficher de point sont toujours à l'état haut, car l'affichage des segments de l'anode se fait pour les valeurs à l'état bas. Ainsi, il n'y a que la valeur en entrée du multiplexeur qui représente l'anode AN2 (voir figure 2.2), soit celle qui est relié à la bascule T, qui a une valeur qui varie. Ce signal est ensuite concaténé au signal en sortie du transcodeur (U6).

II.5) Chronogramme du module

Le chronogramme ci-dessous est la simulation behavorial du module. Les signaux *CE_1ms* et *CE_1s* pour la simulation ne dure pas réellement une milliseconde et une seconde pour permettre d'avoir un chronogramme lisible.

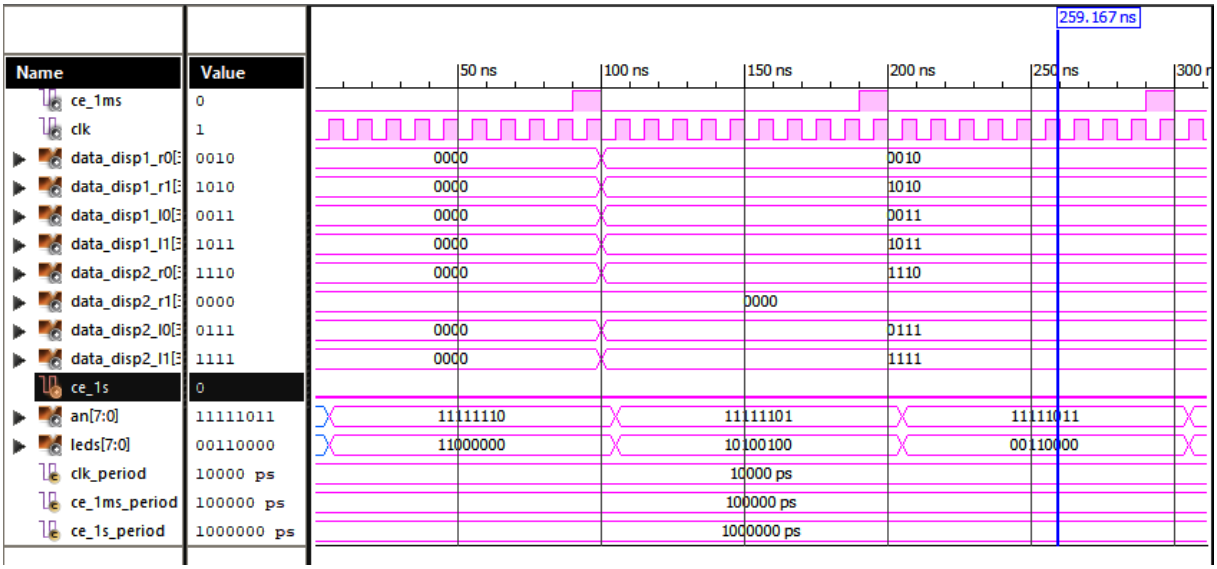


Figure 2.3 : Simulation du module

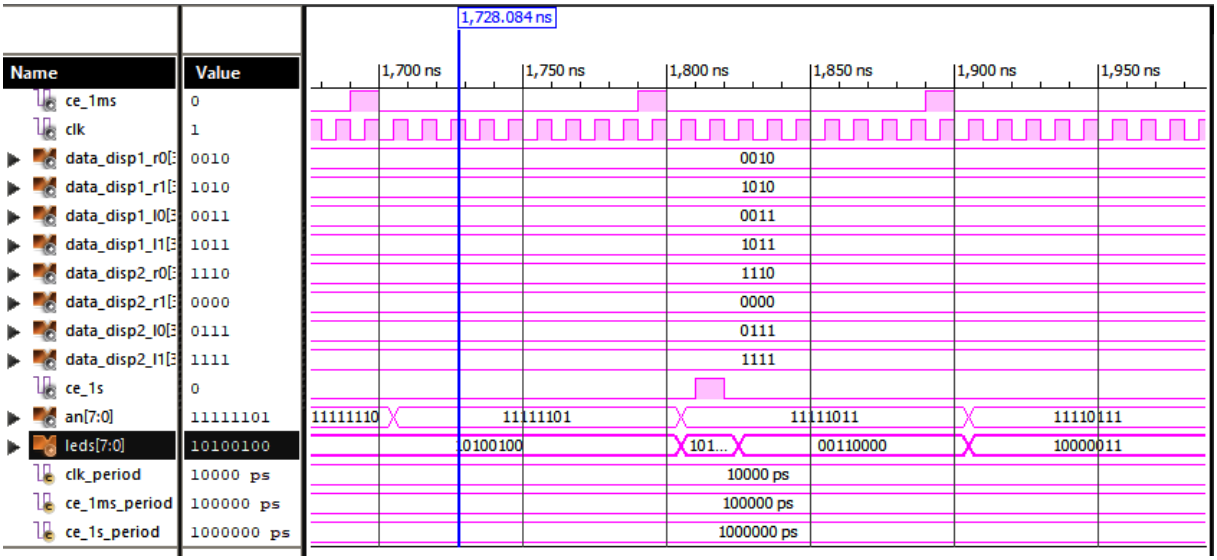


Figure 2.4 : Affichage du point

On peut voir sur la figure 2.3 que la sélection des anodes et l'affichage des segment de celle-ci se fait bien toutes les millisecondes, au front montant du signal *CE_1ms*. En effet, la valeur en sorti du signal *AN*, change à fois, le bit 0 se décalant à chaque fois. Enfin, on peut vérifier avec la table de vérité en annexe, que les valeurs sur 8 bits du signal *LED* correspond bien aux valeurs des différents signaux *data*. Enfin sur la figure 2.4, on peut voir qu'au front montant du signal *CE_1s*, le signal *AN* à le code correspondant à l'anode 2 (voir figure 2.2) et le signal *LED* à un de ces trains de données avec un 0 au début de son code, 0 qui représente

le fait que le point s'allume. Ce chronogramme confirme le bon fonctionnement du sous-module.

III) Sous-module *chronometer*

III.1) Présentation du sous-module

Ce sous-module permet de fournir au module ***display*** et ***vgaDisplay***, les modules de gestion de l'affichage, les données temporelles, soient minutes et secondes à afficher. Ces données temporelles sont transmises via les signaux *sec_unit*, *sec_dec, min_unit* et *min_dec*. L'architecture du module est la suivante (voir figure 3.1) :

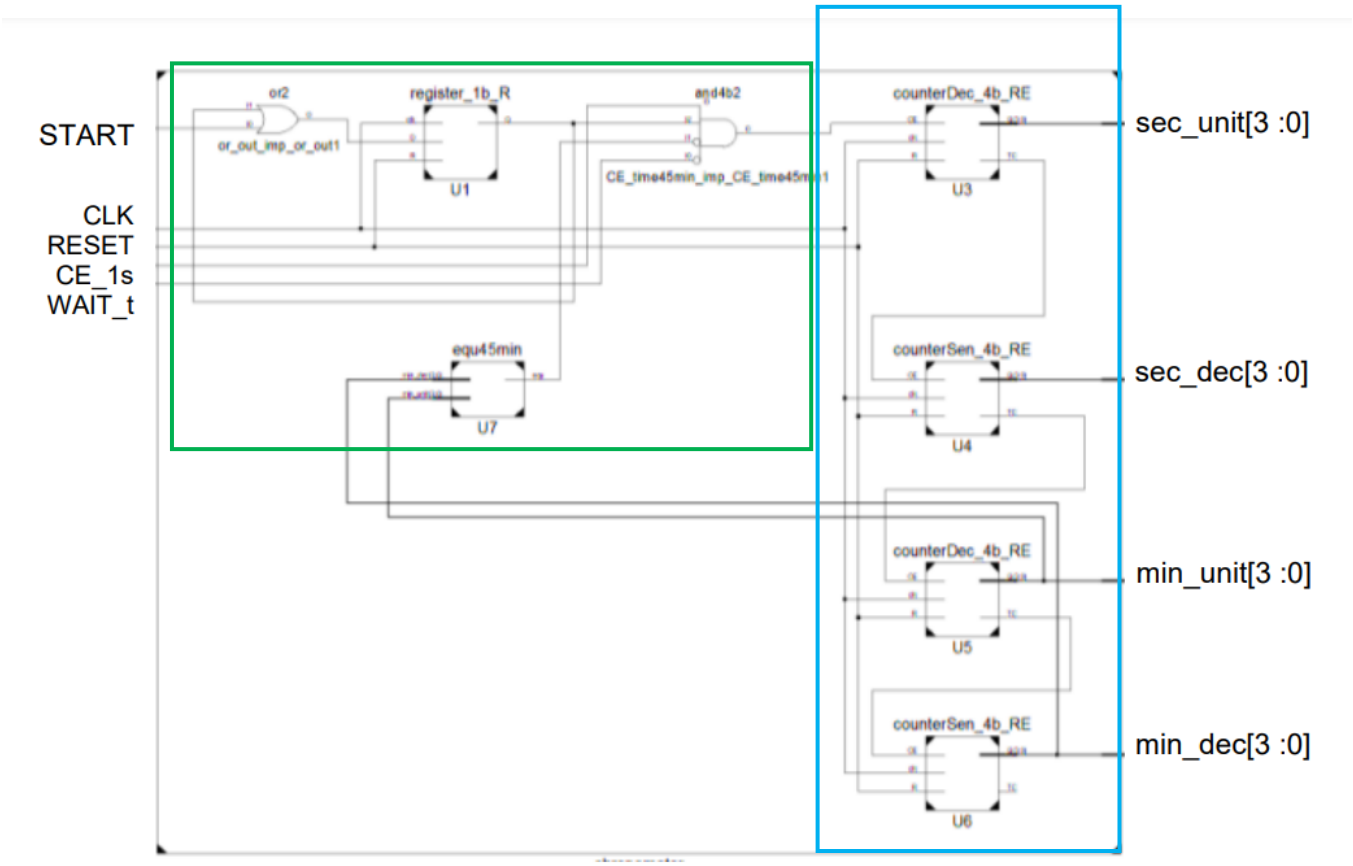


Figure 3.1 : Architecture de l'entité ***chronometer***

Ce module permet :

- de démarrer le chronomètre à l'aide du signal *START*.

- de remettre à zéro le chronomètre à l'aide du signal *RESET*.

- de mettre en pause le chronomètre à l'aide du signal *WAIT_t*.

On peut séparer ce module en deux partis, la première partie (encadrée en bleu) qui ne fait que compter et la deuxième partie (encadré en vert) qui pilote la première pour permettre le comptage désiré et les fonctions précisées précédemment.

III.2) Comment est effectué le comptage ?

Tout d'abord, comme on a besoin d'afficher les dizaines de minutes et les dizaines de secondes, cela revient à dire que l'on va compter deux fois de 0 jusqu'à 9, pour les valeurs des unités, et deux fois de 0 jusqu'à 5, pour les valeurs des décimales. En effet, notre système a besoin de compter une mi-temps entière soit 45 :00. Ainsi, comme on peut le voir sur la figure 4.1, notre module est composé de 4 compteurs. Le programme VHDL de ces compteurs se trouvent en annexe.

Ces compteurs sont des compteurs synchrones 4 bits qui comptent soit de 0 à 5 (*counterSen_4b_RE*), soit de 0 à 9 (*counterDec_4b_RE*) et qui sont actifs sur front montant du d'horloge *CLK*. Ils disposent d'une entrée de remise à zéro *R* (reset), relié au signal *RESET* en entrée du sous-module. Cette entrée est asynchrone active à l'état haut. Ensuite, ils possèdent une entrée de validation synchrone *CE* (clock enable) active à l'état haut. Enfin, en sortie ils délivrent le signal *Q,* qui sera relié aux différents signaux de sortie du module qui représente les données temporelles, et le signal *TC*. Le signal *TC* permet de savoir si le compteur a fini de compter, donc soit à 9 ou à 5. Pour cela il reproduit un motif du signal *clk*, lorsque le compte est fini. C'est ce signal qui va permettre de relier les compteurs en donnant l'état du compteur qui précède. (Voir figure en annexe)

Cette description des compteurs nous amène à expliquer leur fonctionnement dans l'ensemble pour compter une mi-temps entière. Tout d'abord, le premier compteur (U3) a son entrée *CE* qui reçoit le signal *CE_1s*. Ainsi, ce compteur est autorisé à compter toutes les secondes. Il représente ainsi le compte des unités des secondes. Lorsque ce compteur a fini de compter, il transmet cette information au compteur suivant (U4) via le signal *TC* qui est reçu par l'entrée *CE* du compteur suivant. Ainsi ce compteur U4 est lui autorisé qu'à compter toutes les dizaines de secondes. Ils représentent ainsi les décimales des secondes. Quand celui-ci a fini de compter jusqu'à 5 cette fois-ci, il transmet cette information au compteur suivant U5 grâce à *TC* reçu sur *CE*. Et cette fois-ci, le compteur U5 lui est autorisé à compter toutes les minutes et ainsi de suite avec le compteur U6.

III.3) Comment sont pilotés les compteurs ?

Tout d'abord, chaque entrée *R* (reset) des compteurs est reliée à l'entrée *RESET* du chronomètre pour permettre la remise à zéro immédiate du chronomètre. De plus, pour que le système soit synchrone chaque horloge des compteurs est reliée au signal de l'horloge *CLK* du chronomètre.

Pour gérer le compte du premier compteur U3, on a relié sont entrée *CE* (count enable) à une porte AND, and4b2, qui permet de :

- déclencher le compte à partir du moment où le signal d'entrée du chronomètre a été à l'état haut.

- ne déclencher le compte que toutes les secondes avec l'entrée *CE_1s* reliés à cette porte AND.

- mettre le comptage en pause lorsque que le signal d'entrée du chronomètre *WAIT_t* est à l'état Haut (entrée sur la porte AND inverser). En effet, tant que le signal *WAIT_t* est à 1, la sortie de la porte AND relié à CE est égal à 0 quelque soit la valeur des autres entrées de cette porte.

- arrêter le comptage lorsque 45 min sont passées à l'aide du signal interne *equ*, sortie de la fonction *equ45min* (U7). En effet, tant que le signal *equ* est à 1, la sortie de la porte AND relié à CE est égal à 0 quelque soit la valeur des autres entrées de cette porte (*equ* est une entrée inversée).

Comme ce compteur est celui qui commande les entrées *CE* des autres en cascades, chacune des actions sur *CE* de U3 se retrouve au niveau des aitres compteurs. De plus, on note sur le schéma une structure particulière pour le signal *START.* En effet, grâce à cette structure lorsque le signal est passé à l'état haut au moins une fois, celui-ci est mémorisé et ainsi même ci ce signal repasse à 0, le comptage continuera de se dérouler jusqu'à ce qu'il y soit une remise à 0. Dans ce cas là, il est nécessaire de remettre le signal *START* à l'état haut au moins le temps d'une période. Pour avoir ce fonctionnement on a utilisé un registre, *register_1b_R* (U1) et une porte OR, *or2*. Ce registre est relié à l'horloge *CLK* du chronomètre pour assurer sa synchronisation et avec le signal *RESET* du chronomètre pour permettre sa remise à 0 asynchrone. Il a en entrée le signal issu de la porte OR qui a pour entrée le signal *START* et la sortie du registre. Ainsi, lorsque le signal *START* passe à l'état haut, l'entrée du registre passe à un et donc sa sortie aussi. Cette sortie est réinjectée en entrée de la porte OR qui quelque soit la valeur du signal *START* fourni une entrée à l'état Haut au registre. Tant que le registre n'aura pas eu de de remise à zéro, il gardera en mémoire la valeur à l'état haut du signal START, même ci celui-ci a été modifié entre temps. (Voir figure en annexe)

Enfin, pour savoir quand est ce que le compteur a atteint les 45 minutes, on récupère constamment les données en sortie des compteurs U5 et U6, soit *min_dec* et *min_unit* et on les compare à l'aide de la fonction *equ45min* au nombres 4 en binaire pour *min_dec* et 5 en binaire pour *min_unit*. Si ces deux signaux sont égaux à ces valeurs, on a le signal *equ* en sortie de cette fonction, qui passe à l'état haut et met le comptage en pause comme vu précédemment. Pour reprendre le comptage, il est obligatoire de faire une remise à zéro avec le signal *RESET*.

III.4) Chronogramme du module

On peut ainsi voir sur le chronogramme ci-dessous le comportement du sous-module ***chronometer*** :

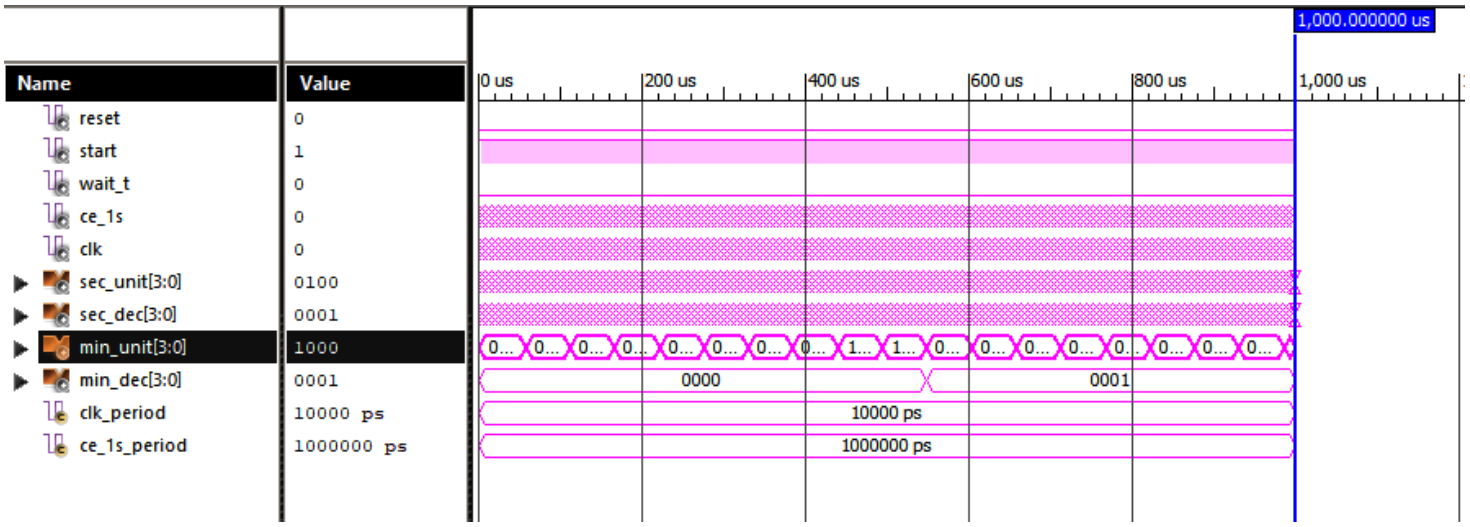


Figure 3.2 : Simulation behavorial du module ***chronometer***

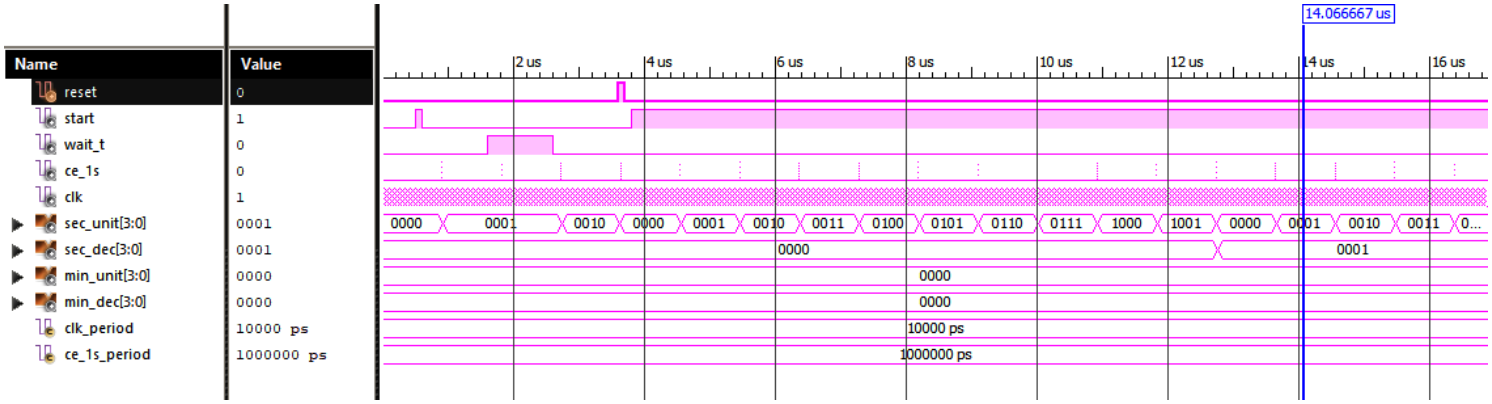


Figure 3.3 : Zoom de la simulation précédente

Grâce à ces simulations, on peut observer le bon fonctionnement de ce module. Tout d'abord sur le premier chronogramme, figure 4.2, on observe que le compte est réalisé correctement avec l'affichage des dizaines des minutes, signal *min_dec*, et des unités de minutes, signal *min_unit*.

Pour vérifier correctement le fonctionnement des secondes et des signaux *RESET, WAIT_t* et *START*, nous avons zoomé, figure 4.3. On observe, premièrement la mise à l'état haut du signal start pour déclencher le compte. On note que même si le signal *START* repasse à l'état bas, le compte continue donc sa mise à l'état haut a bien été enregistré par le système. Ensuite, il y a une mise à l'état haut du signal *WAIT_t*. On peut noter que tant que ce signal est à l'état haut, le compte est en pause, voir signal *sec_unit*. LE signal *RESET* est ensuite ms à l'état haut et on observe bien une remise à zéro du compte sur le signal *sec_unit.* Le compte recommence bien avec une mise à 1 du signal *START*.

IV) Sous-module *score*

IV.1) Présentation du sous-module

Ce sous module permet de fournir au module *display* et *vgaDisplay,* les données du score de l'équipe visiteur et l'équipe local à afficher (unités et dizaines). Ces données sont transmises via les signaux *loc_unit*, *loc_dec* (local) et *vis_unit*, *vis_dec* (visiteur).

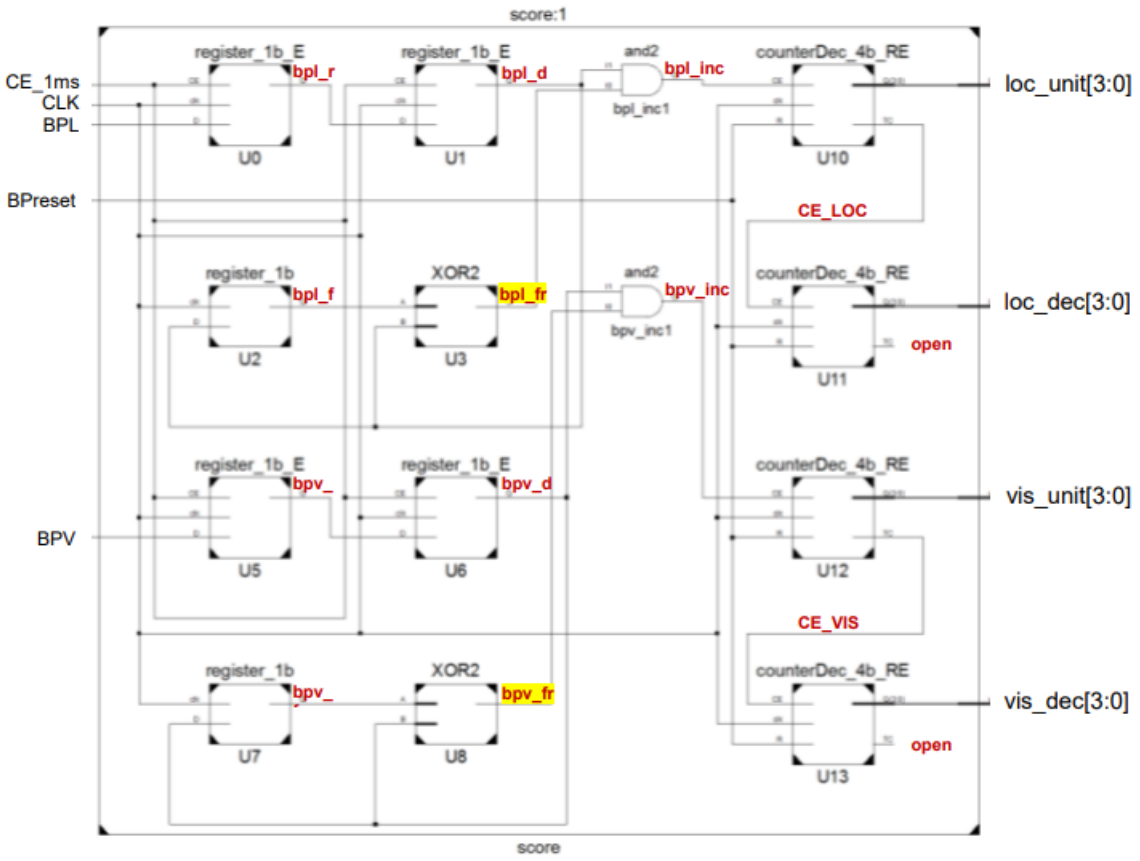


Figure 4.1 : Architecture du module score

Pour augmenter le score on appuie sur le bouton *BPV* et, idem, pour le score local avec le bouton *BPL* qui sont deux entrées du bloc, on a aussi une entrée *BPreset* qui remet le score à zéro. Le sous-bloc ***score*** est donc composé d'une partie qui gère le score local et une autre qui gère le score visiteur.

IV.2) Fonctionnement des compteurs

On a donc réalisé notre sous-bloc en VHDL où l'on a mis pour les parties visiteur et local deux compteurs à chaque fois, un pour les unités et l'autre pour les décimales. On veut donc compter de 0 à 9 pour les unités et les secondes grâce à un compteur synchrone 4 bits counter_*Dec_4b_RE*, qui compte de 0 à 9 avec en entrés le signal *CLK* (l'horloge) et le signal

du bouton reset pour remettre l'unité à l'état zéro, ainsi que le signal *bpv_inc* (visiteur) ou *bpl_inc* (local) qui indique que le bouton du score a été enclenché. Le premier compteur affiche en sortie un signal sur 4 bits pour pouvoir compter jusqu'à 9. Chaque fois que le compteur des unités arrive à 9, le signal *TC* va autoriser la mise en marche du second compteur qui permettra de compter pour les décimales et d'envoyer un signal de sortie sur 4 bits aussi. Donc le deuxième compteur pour les visiteurs et le local (U13 et U11) n'est autorisé à compter que lorsque le premier compteur (U12 et U10) à fini de compter jusqu'à 9 à chaque fois qu'on appuie sur un bouton. On peut maintenant se demander comment sont pilotés les quatre compteurs et comment éviter un anti-rebond avec les boutons *BPV* et *BPL*.

IV.3) Pilotage des compteurs

Les compteurs reçoivent les signaux bpv*_inc* et *bpl_inc,* qui correspond aux enclenchements des boutons, ces signaux sont réalisés par une succession de registres sur front montant de l'horloge. Comme nous avons deux boutons en entrées où l'on ne veut pas d'anti-rebond, nous avons deux fois le même bloc composé de deux fonction *register_1b_E* (U0, U1 et U5, U6), une fonction *register_1b* (U2, U7) et une porte XOR (U3, U8).

Ainsi, les entrées du sous-bloc score sont reliés à deux *register_1b_E*, l'entrée *CE_1ms* est une entrée enable qui permet d'autoriser la mise en marche des registres, l'entrée *CLOCK* est l'horloge et les autres sont les boutons. Le bouton de reset agit directement sur les compteurs, tandis que lorsque l'on appuie sur un des deux autres boutons, on envoie un signal (bpl*_d* ou *bpv_d*) à un deux registres (*register_1b_E*). Ces registres permettent d'ajouter un certain délai, pour être sûr de ne pas considérer un rebond comme un nouvel appuie sur le bouton.
Ensuite, la sortie de ce deuxième registre envoie un signal sur un bloc *register_1b* (U2, U5) *et* une porte XOR (U3, U8), et sur une porte AND. Cet ensemble *register_1b/*XOR nous permet de vérifier s'il n'y a pas de rebonds entre deux coups d'horloge due aux boutons d'entrée. En effet, le registre reçoit et le retarde d'un coup d'horloge puis le fourni en entrée à la porte XOR. Ainsi la porte XOR reçoit en entrée, le signal issu du registre et le signal bpl_d, issu du bloc de registre précédent (U1 ou U6). Ainsi, elle peut comparer si les signaux sont les mêmes ou pas. Si les signaux sont différents, elle envoie un signal à l'état haut en entrée de la porte AND. Ainsi si le registre U1 ou U6 avait enregistré l'appuie sur un bouton alors on envoie bien un signal (*bpv_inc* ou *bpl_inc*) pour dire au compteur d'incrémenter le nouveau score. Cela permet de n'envoyer qu'une seule impulsion et non plusieurs à cause des rebonds qui sont « filtré » par les délais et la comparaison dans la XOR.

IV.4) Chronogramme du module

On observe bien sur le chronogramme le bon fonctionnement des compteurs :

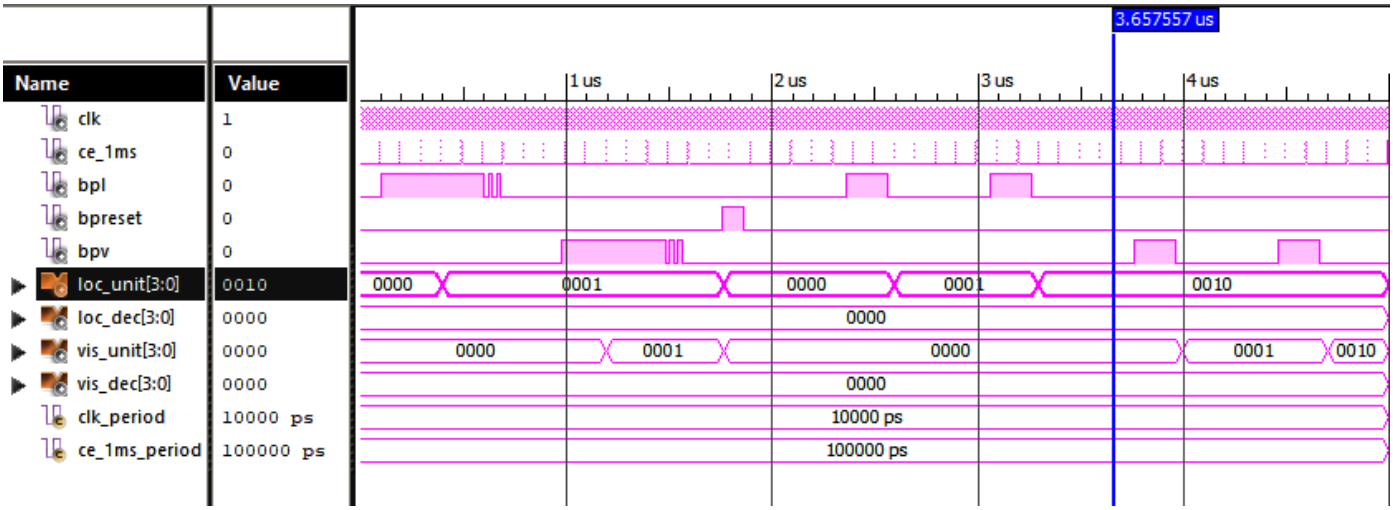


*Figure 4.2 : Simulation behavorial de **score***

En effet, on observe tout d'abord que le score de chaque équipe s'incrémente, signal *loc_unit* et *vis_unit*, à chaque appuie de bouton, *BPL* et *BPV.* Ensuite, on observe que l'anti-rebond fonctionne correctement. En effet, pour les premiers appuies sur *BPV et BPL*, nous avons simulé un rebond et on observe que celui n'est bien pas pris en compte par le système. De même, lorsqu'on reste appuyer longtemps, le score ne s'incrémente pas plusieurs fois. Ensuite on peut noter qu'à la mise à l'état haut du signal *BPreset*, le compte des deux signaux est remis à zéro.

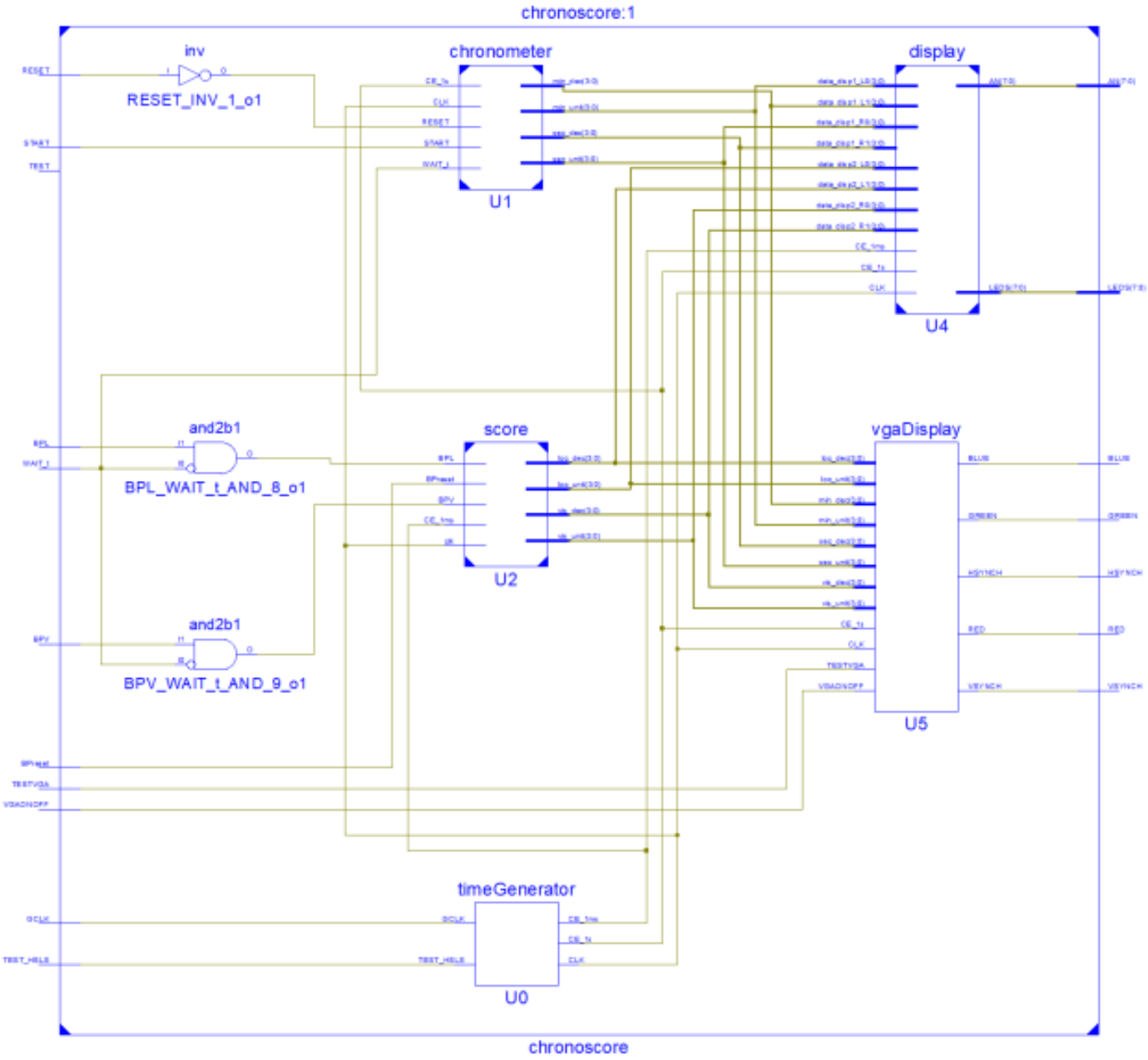
Conclusion

Dans ce projet, nous avons donc développé un module, ***chronoscore***, permettant d'afficher le temps et le score d'un match. Nous avons apprécié travailler au sein du groupe dans le but de concevoir un projet concret. En effet, cela nous a permis d'utiliser nos connaissances et de les appliquer dans une situation réelle. Nous avons fait face à quelques difficultés lors de ce travail, notamment la compréhension globale de tout le système et l'utilité de tous les blocs pour son bon fonctionnement, mais une fois ce problème surmonté il nous suffisait juste de lier tous les blocs entre eux correctement. Un de nos problèmes dont nous nous sommes rendus-compte lors de l'implémentation de notre programme sur la carte, que nous avions mal relié les modules internes du sous-module ***chronometer***. Cette erreur impliquait que le temps ne s'arrêtait pas au bout de 45 minutes. Nous avons trouvé le problème en observant la vue RTL de ce module.

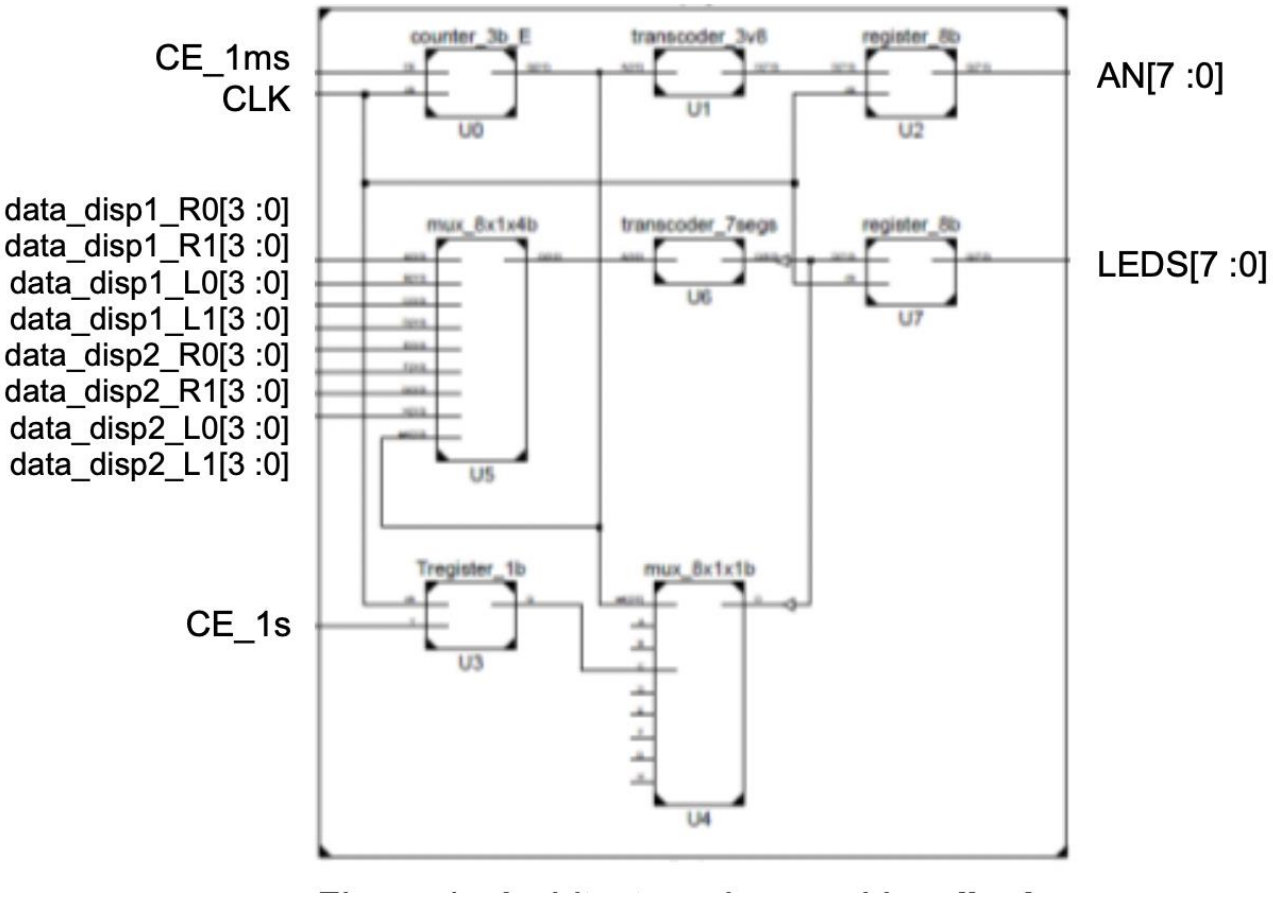
En guise d'amélioration du projet, on pourrait proposer d'afficher les numéros des joueurs entrants et sortants lors d'un remplacement, afficher le temps additionnel, ou encore comptabiliser les différentes fautes en ne les affichant que sur demande et non constamment.

Annexes

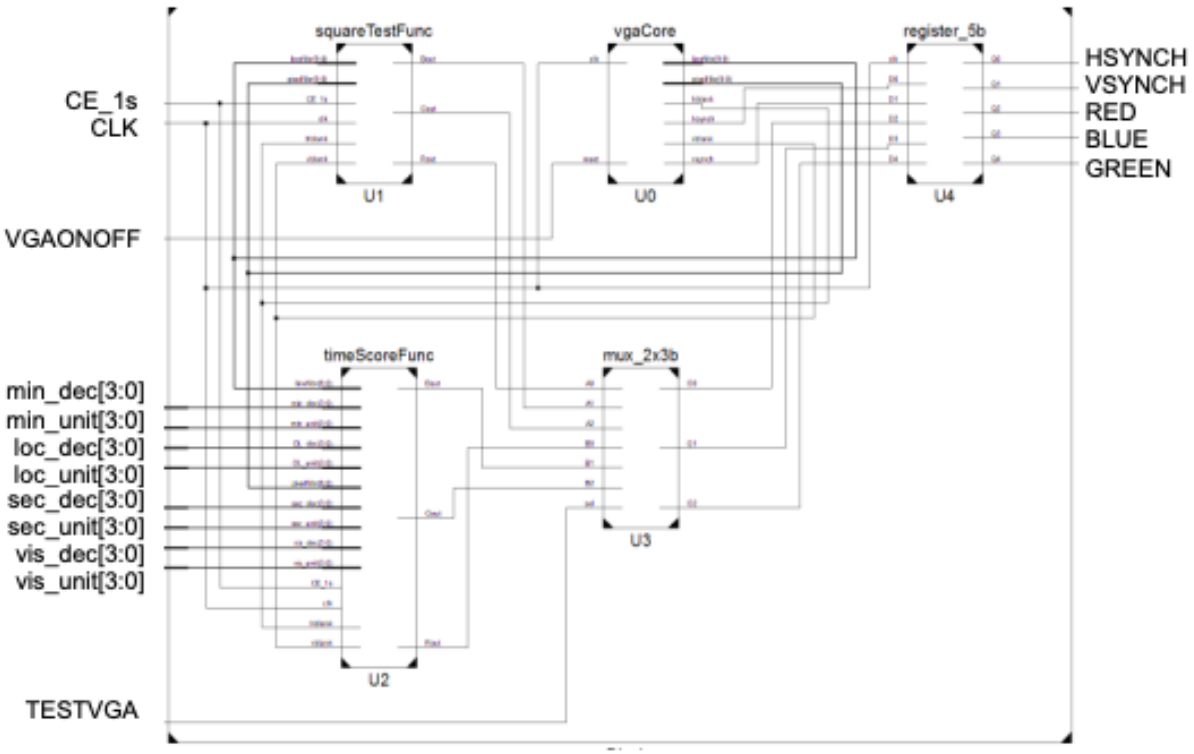
Architecture des modules



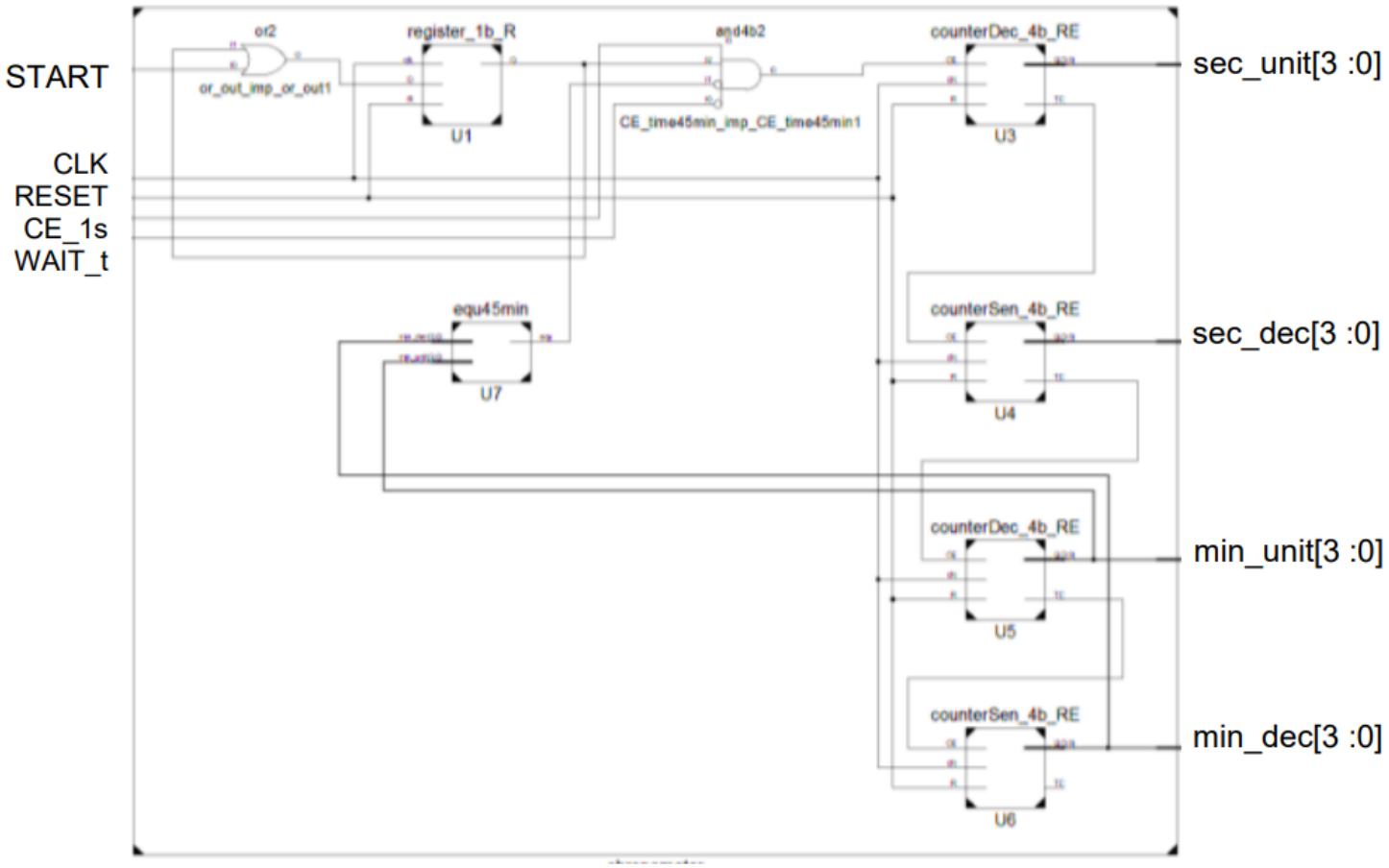
Architecture du module ***chronoscore***.



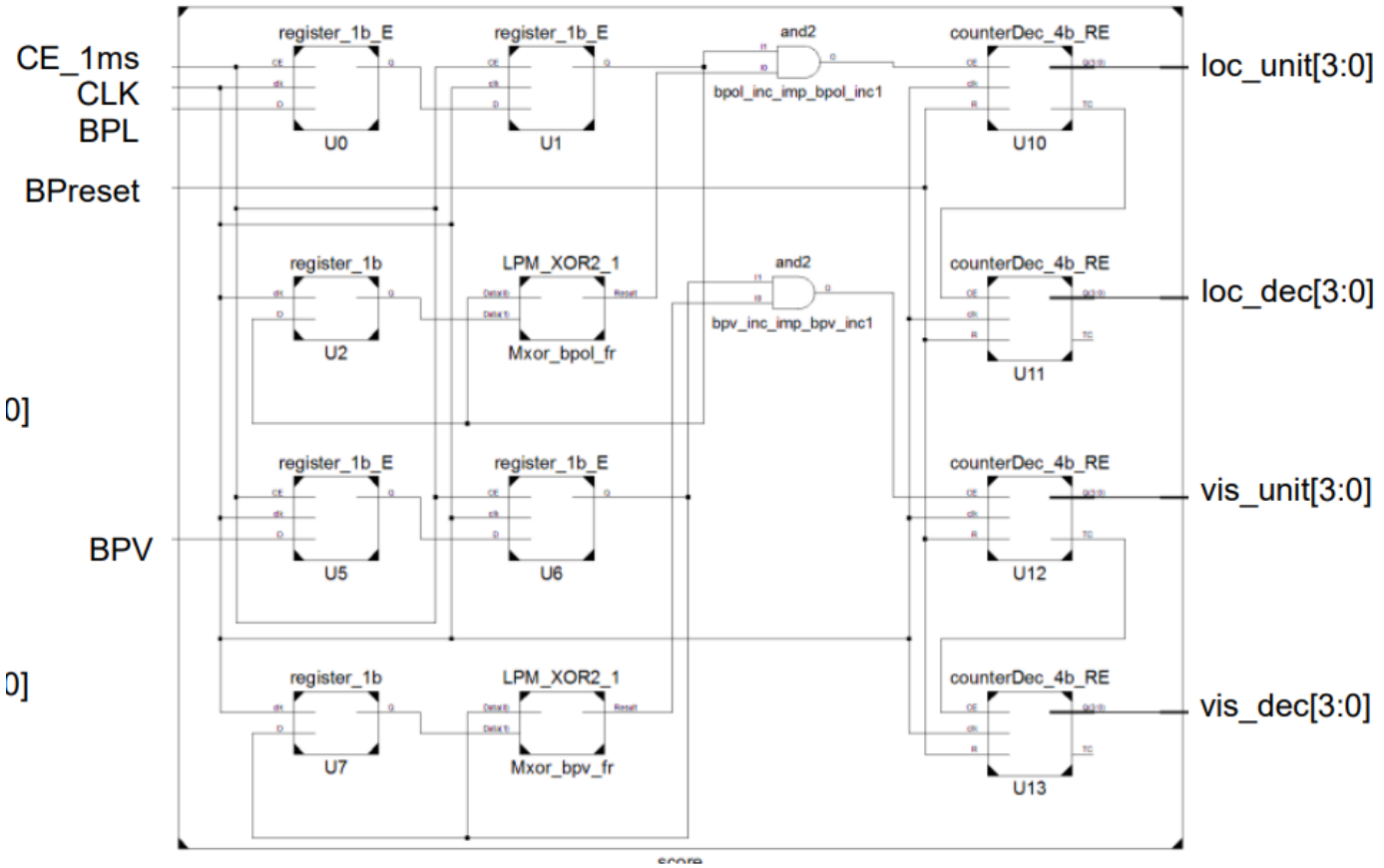
Architecture du module ***display***.



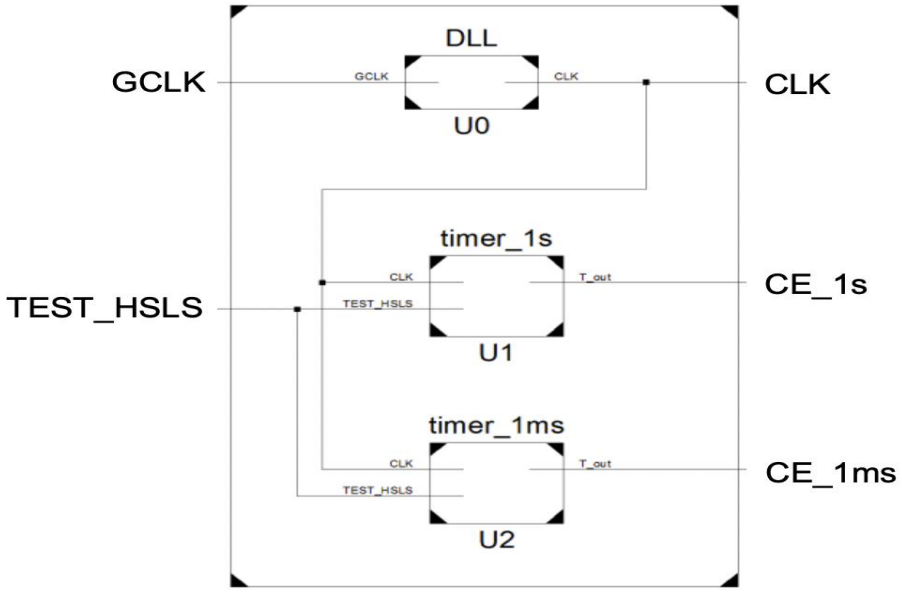
Architecture du module ***vgaDisplay***.



Architecture du module ***chronometer***.



Architecture du module **score**.

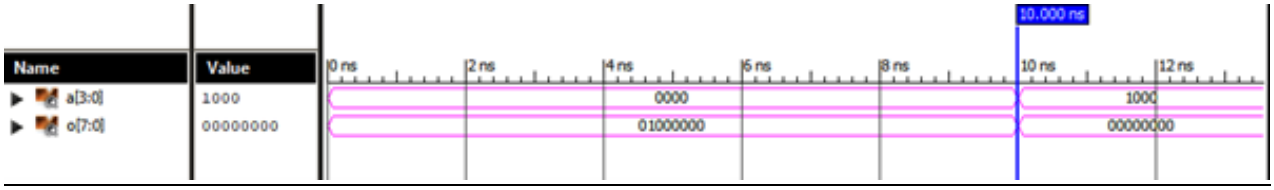


Architecture du module ***timeGenerator***.

Annexes module *display*

Fonction transcoder 7segs :

- Simulation behavioral :



On a simulé pour vérifier si on arrivait à afficher la lettre A et le chiffre 0. Le code des segments correspond bien aux deux éléments cités précédemment.

Signal d'entrée A [3 :0]	Signal de sortie 0[6:0]	Valeur affiché
0000	1000000	0
0001	1111001	1
0010	0100100	2
0011	0110000	3
0100	0011001	4
0101	0010010	5
0110	0000010	6
0111	1111000	7
1000	000000	8
1001	0010000	9
1010	0001000	A
1011	0000011	B
1100	1000110	C
1101	0100001	D

1110	0000110	E
1111	0001110	F

Fonction transcoder 3v8:

Voici les équations logiques:

00 = A1 + A2 + A0

01 = A1 + A2 + A0\

02 = A1\ + A2 + A0

03 = A1\ + A2 + A0\

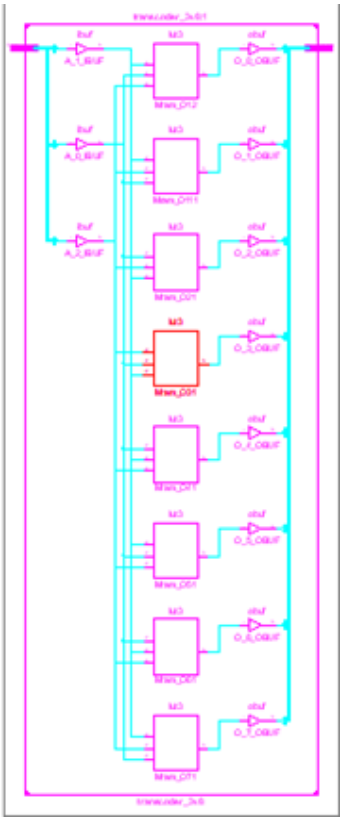
04 = A1 + A2\ + A0

05 = A1 + A2\ + A0\

06 = A1\ + A2\ + A0

07 = A1\ + A2\ + A0\

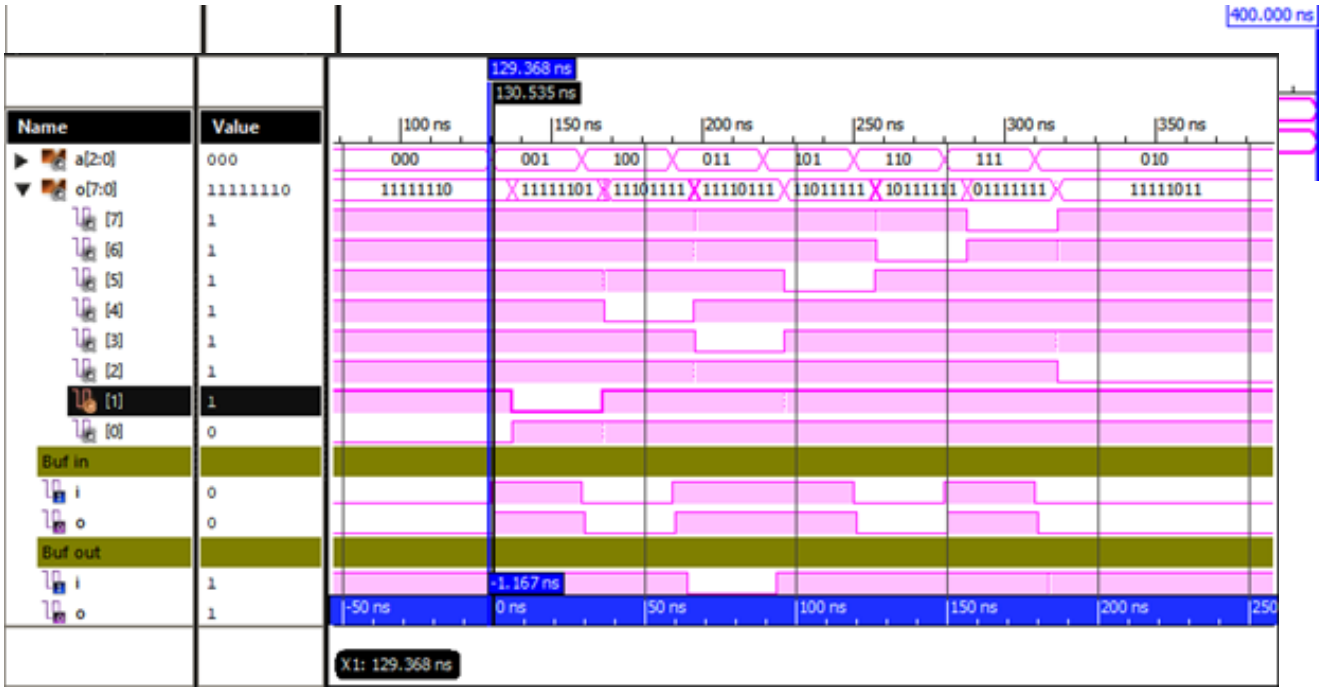
- Vue Technologique :



On retrouve donc dans chaque bloc des portes OU et des inverseurs qui réalisent les équations ci-dessus.

On a des inverseurs à la sortie de chaque bloc pour mettre à l'état BAS le signal.

- Simulation Behavioral :



- Simulation Post-Route:

On remarque un temps de retard sur les buffers. On peut expliquer cela par le retard induit par la propagation du signal dans chaque porte (cf. vue technologique).

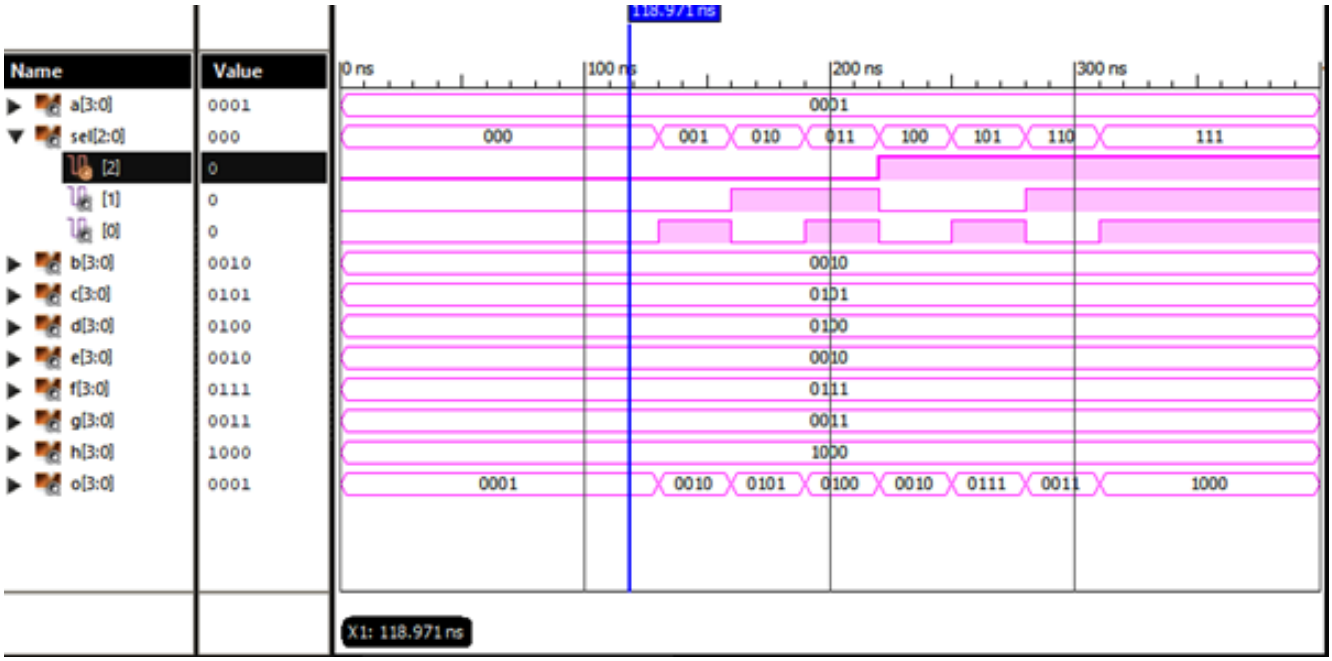
<u>Fonction mux_8x1x4b:</u>

- Programme :

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

entity mux_8x1x1b is

   Port ( A : in  STD_LOGIC;

          B : in  STD_LOGIC;

          C : in  STD_LOGIC;

          D : in  STD_LOGIC;

          E : in  STD_LOGIC;

          F : in  STD_LOGIC;

          G : in  STD_LOGIC;

          H : in  STD_LOGIC;

          sel : in  STD_LOGIC_VECTOR (2 downto 0);

          O : out  STD_LOGIC);
```

```
end mux_8x1x1b;

architecture Behavioral of mux_8x1x1b is

begin


        O <=    A when sel = "000" else
                        B when sel = "001" else
                        C when sel = "010" else
                        D when sel = "011" else
                        E when sel = "100" else
                        F when sel = "101" else
                        G when sel = "110" else
                        H ;

end Behavioral;
```

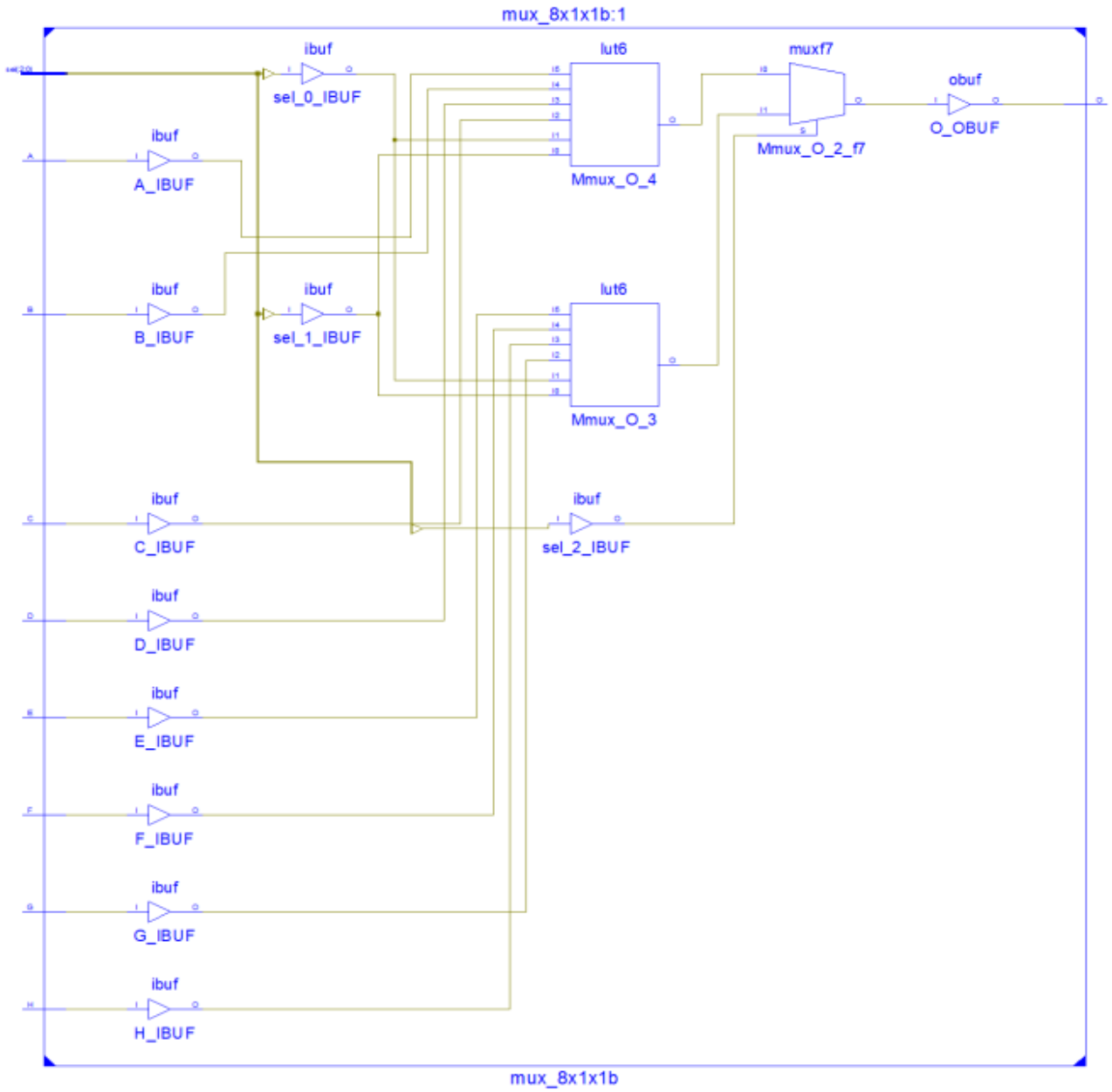
- Simulation Behavioral :



On remarque que la sortie o est égale à la valeur de l'entrée sélectionnée par sel.

Fonction mux_8x1x1b:

- Vue Technologique :

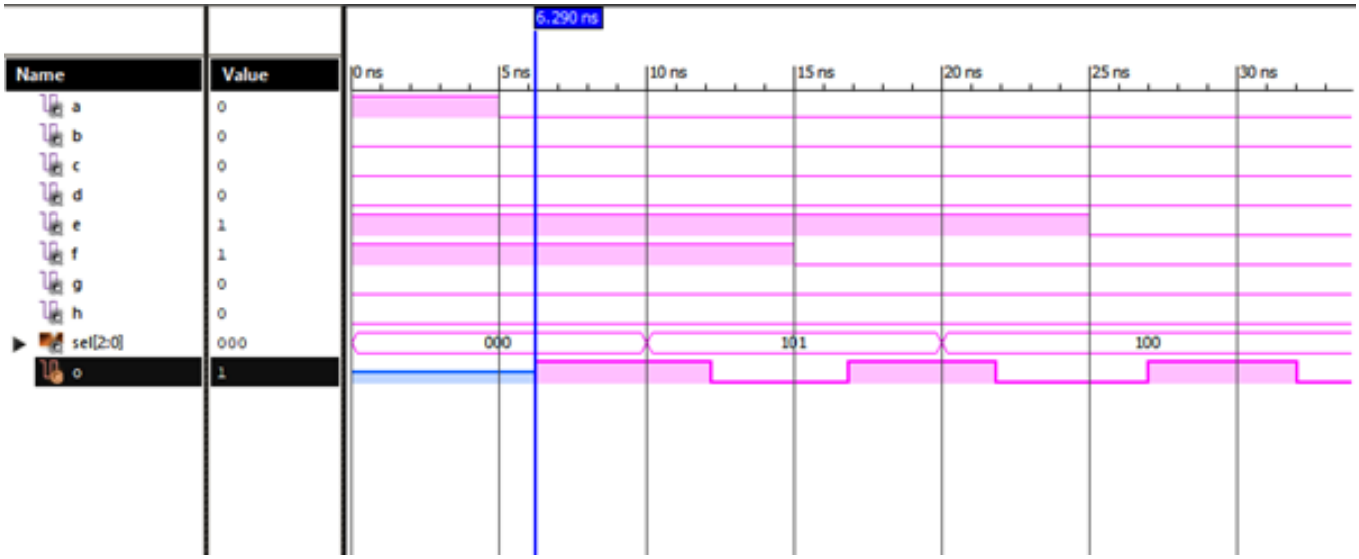


Sur la vue technologique on observe les buffers permettant tout d'abord de synchroniser les signaux ainsi que de rehausser si nécessaire leur état bas pour une bonne lecture de ces derniers. Ensuite sont ajoutés deux lut dans le but de gérer les contraintes physiques dues à l'organisation de la place des composants sur le FPG. Chaque lut n'ayant que 5 entrées, il y en a deux pour gérer les 10 signaux envoyés. Enfin les sorties de ces deux lut sont multiplexés dans le mux avant de passer dans la sortie O permettant d'avoir un signal optimisé en sortie.

- Simulation Behavioral :

On obtient une sortie cohérente à nos entrées.

- Simulation Post-Route :



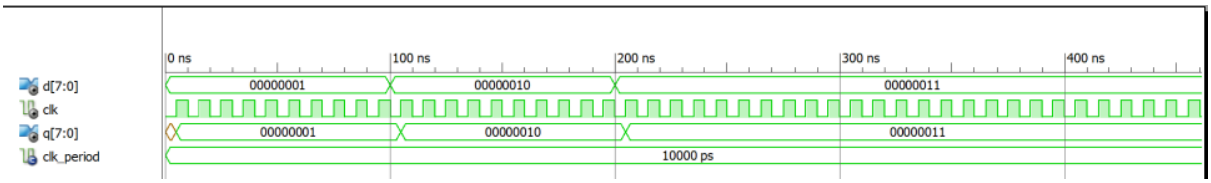
On obtient un temps de propagation de 6.29 ns.

<u>Fonction register_8b:</u>

On attend 100 ns pour la simulation post-route pour permettre au système de s'initialiser correctement afin d'obtenir le bon résultat en sortie.

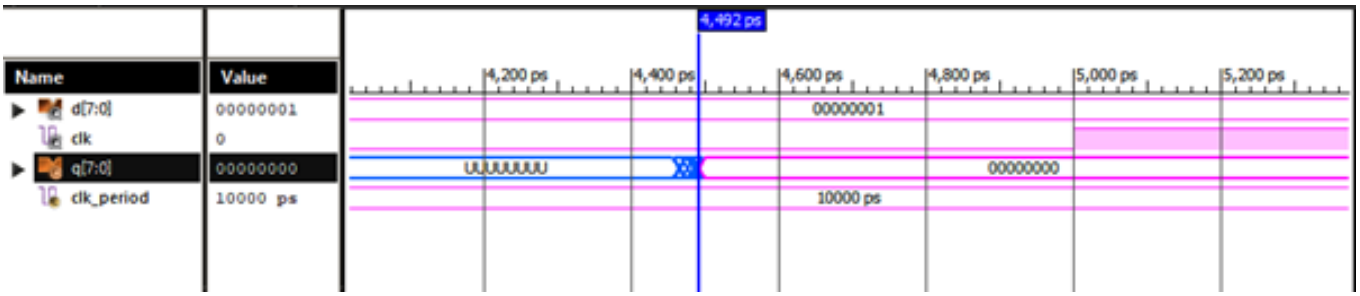
- Simulation Behavioral:

On observe bien les changements opérés au front montant de la courbe. Nous n'avons pas initialisé la sortie ce qui explique que le signal soit indéterminé (partie orange du



chronogramme) les 5 premières nanosecondes.

- Simulation Post-Route:



On remarque un temps de propagation de 4.492 ns. On explique ce décalage par rapport à la simulation Behavioral par le fait que l'on prenne les caractéristiques des composants pour mesurer le temps.

<u>Fonction Tregister_1b:</u>

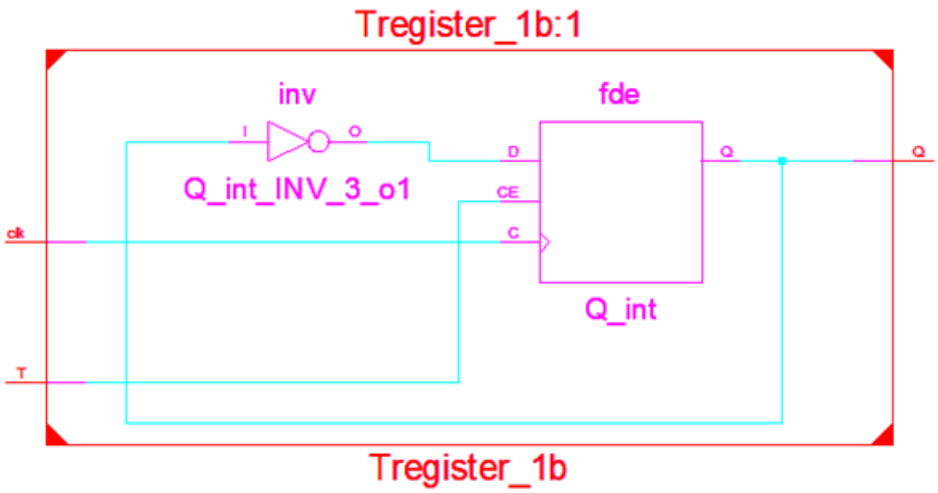
- Programme :

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity Tregister_1b is
   Port ( T : in  STD_LOGIC;
          clk : in  STD_LOGIC;
          Q : out  STD_LOGIC);
end Tregister_1b;
architecture Behavioral of Tregister_1b is
signal Q_int : STD_LOGIC :='0';
begin
          regTregister_1b:process(clk)
                    begin
                              if (rising_edge (clk)and T = '1') then
                                        Q_int <= NOT Q_int;
                                        else
                                        Q_int <= Q_int;
                              end if;
                    end process;
```

Q <= Q_int;

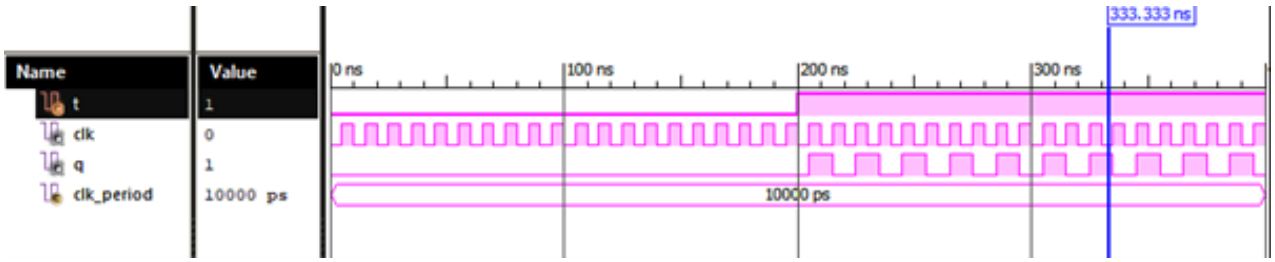
end Behavioral;

- Vue Technologique :



Pour la bascule T, si son entrée est active, elle bascule à chaque impulsion d'horloge. Si son entrée T est inactive, elle conserve son état. On remarque que sur la vue technologique, l'inverseur inverse Q_int en fonction de T.

- Simulation Behavioral :



On remarque que la bascule T permet de diviser la fréquence du signal d'horloge par 2 d'après le chronogramme ci-dessus.

<u>Fonction counter_3b_E:</u>

- Programme :

library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

use IEEE.NUMERIC_STD.ALL;

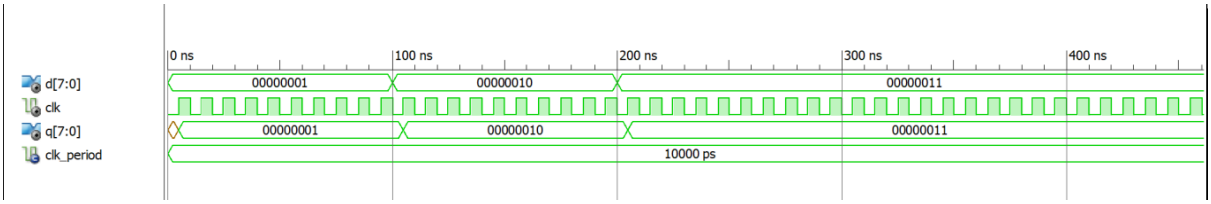
entity counter_3b_E is

Port (CE : in STD_LOGIC;

clk : in STD_LOGIC;

```
        Q : out  STD_LOGIC_VECTOR (2 downto 0));
end counter_3b_E;
architecture Behavioral of counter_3b_E is
signal cnt : unsigned(2 downto 0) := "000";
begin
        process(CE,clk)
        begin
                if (CE = '1' and rising_edge(clk)) then
                        if (cnt = "111") then
                                cnt <= "000";
                        else
                        cnt <= cnt + 1;
                        end if ;
                end if;
        end process;
Q <= STD_LOGIC_VECTOR(cnt);
end Behavioral;
```

- Simulation Behavioral :

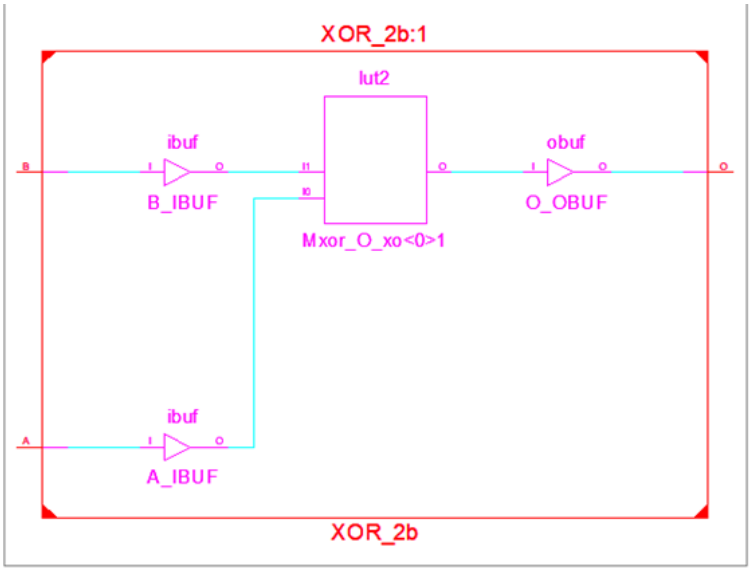


La sortie n'ayant pas été initialisée, il est normal que le signal soit indéterminé sur les premières nanosecondes (partie orange). De plus, le signal est légèrement décalé en raison du front montant. Enfin, on observe néanmoins une sortie cohérente à nos entrées.

Annexes module *score* et *chronometer*

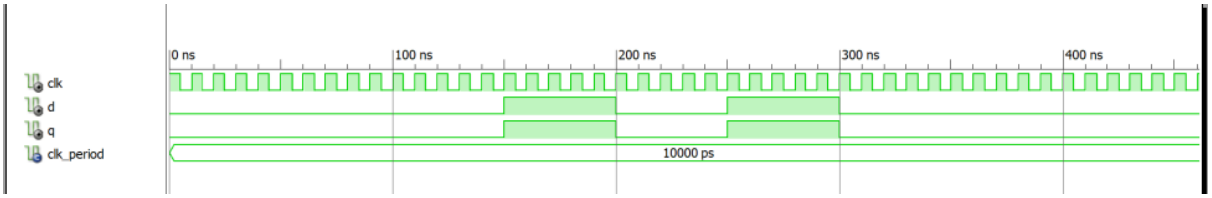
Fonction XOR_2b:

- Vue technologique :

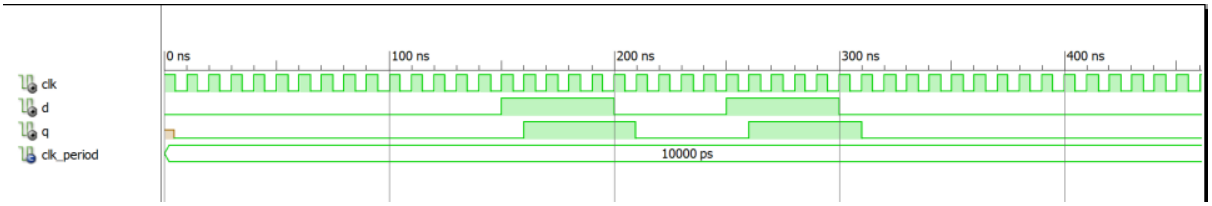


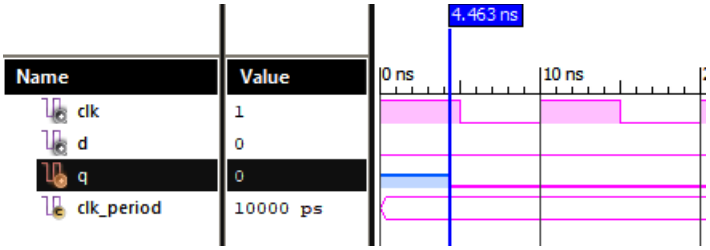
Fonction Register_1_b

- Simulation Behavioral



- Simulation Post-Route

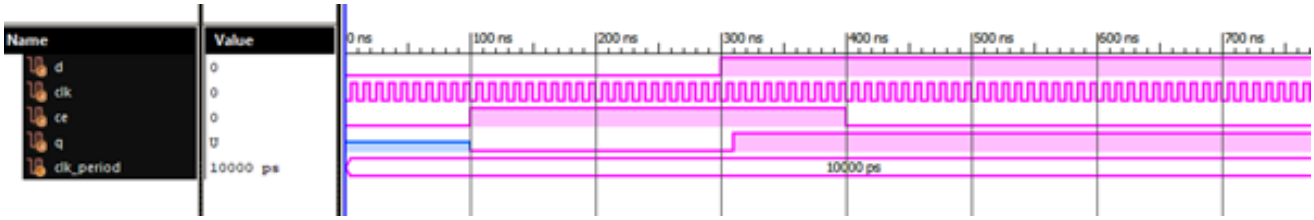




On observe bien un décalage de l'horloge ce qui prouve la mise en mémoire la donnée D d'entrée

Fonction register_1b_E:

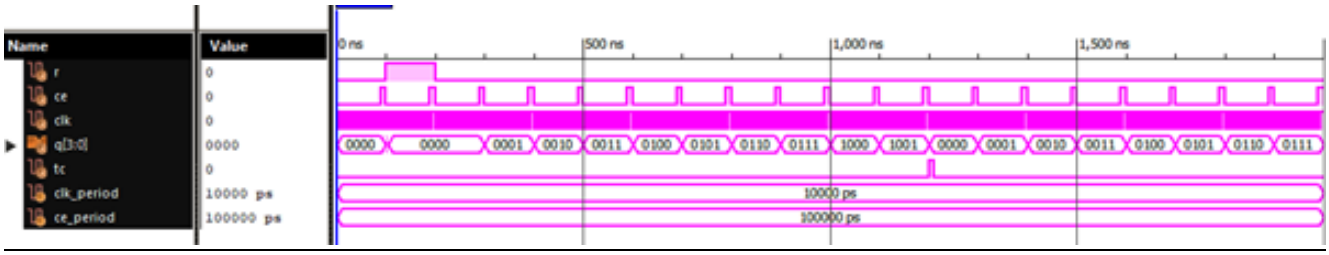
- Simulation Behavioral



Lorsque la sortie CE est à 1 on a bien mémorisation de l'entrée

Fonction counterDec_4b_RE:

- Simulation Behavioral



L'allure du signal TC est conforme aux attentes du document de préparation. En effet, lorsque Q passe de 9 à 0.

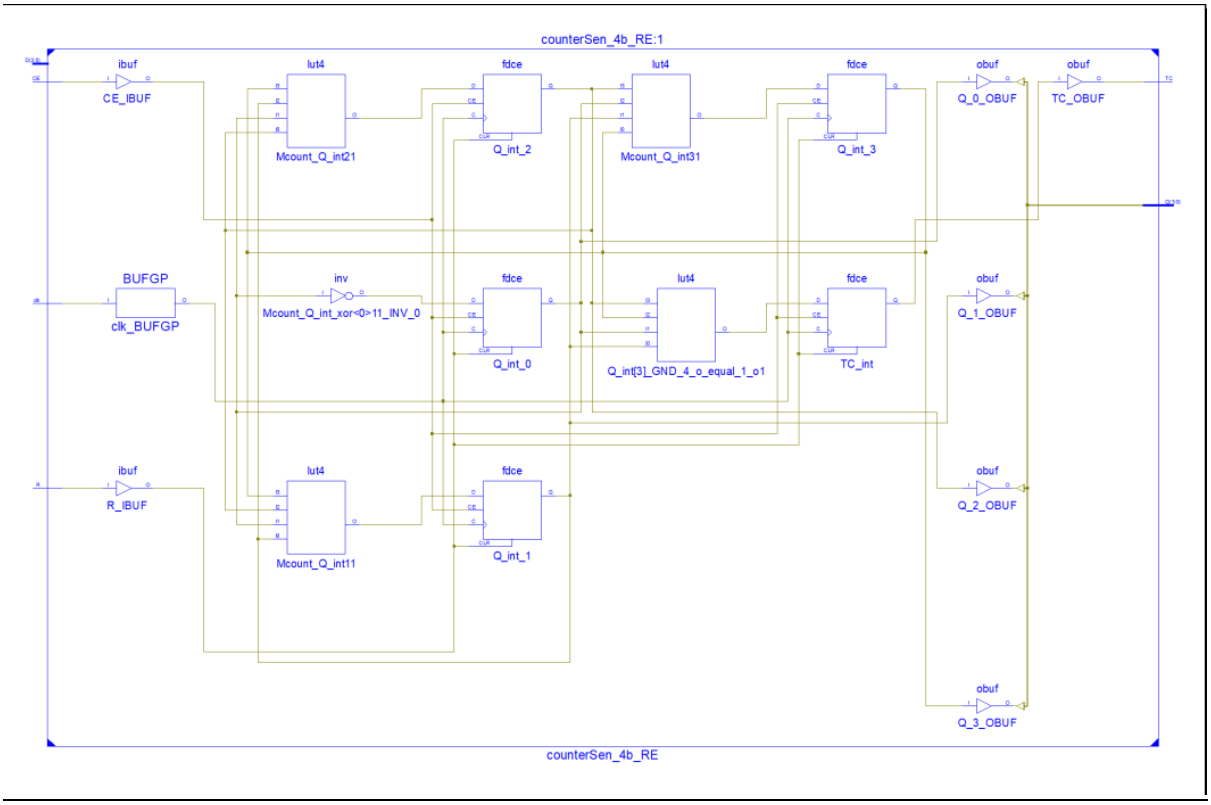
- Programme

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
entity counterDec_4b_RE is
   Port ( R : in  STD_LOGIC;
          CE : in  STD_LOGIC;
          CLK : in  STD_LOGIC;
          Q : out  STD_LOGIC_VECTOR (3 downto 0);
          TC : out  STD_LOGIC);
end counterDec_4b_RE;
architecture Behavioral of counterDec_4b_RE is
signal TC_int : STD_LOGIC :='0';
signal Q_int : UNSIGNED(3 downto 0) :="0000";
```

```
begin
process(CLK,CE,R)
begin
	if (R = '1') then
		TC_int <= '0';
		Q_int <= "0000";
	elsif (rising_edge(CLK)) then
		TC_int <= '0';
		if (CE = '1') then
			if (Q_int = "1001") then
				TC_int <= '1';
				Q_int <= "0000";
			else
				Q_int <= Q_int+1;
			end if;
		end if;
	end if;
	Q <= STD_LOGIC_VECTOR(Q_int);
	TC <= TC_int;
end process;
end Behavioral;
```

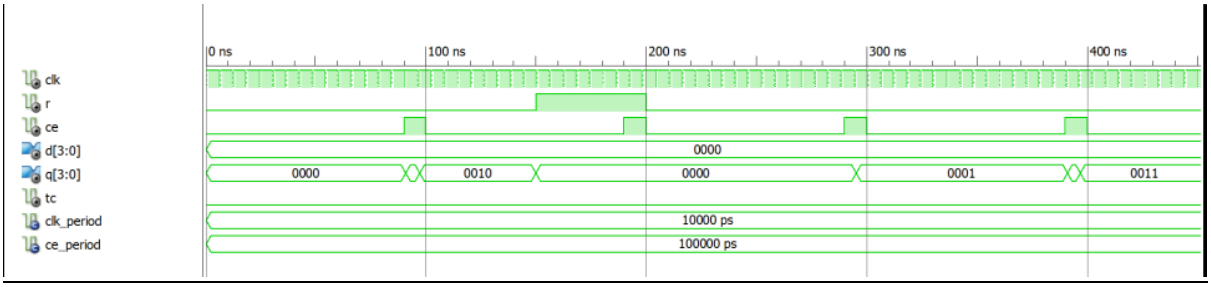
Fonction CounterSen_4b_RE:

- Vue Technologique

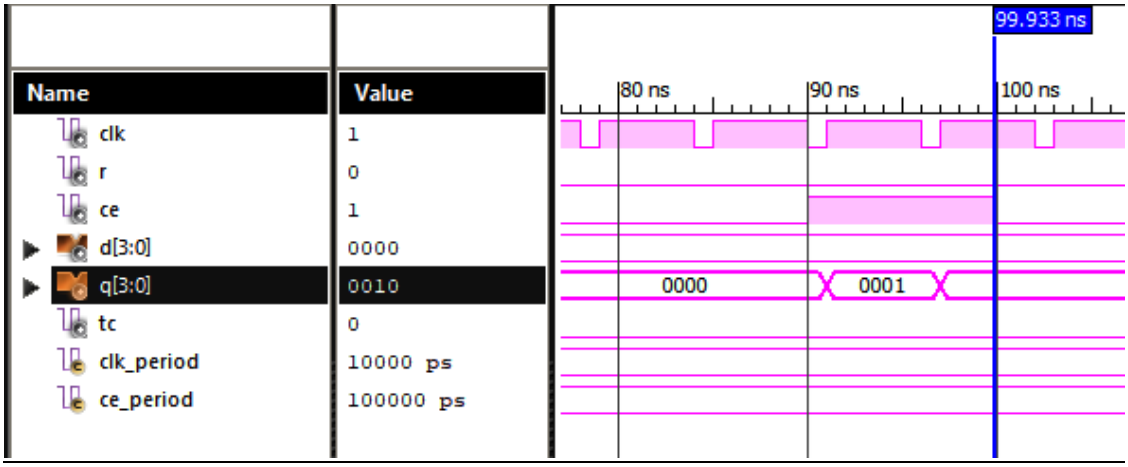


Sur la vue technologique on observe les buffers permettant tout d'abord de synchroniser les signaux ainsi que de rehausser si nécessaire leur état bas pour une bonne lecture de ces derniers. On observe aussi 4 blocs additionneurs lut.

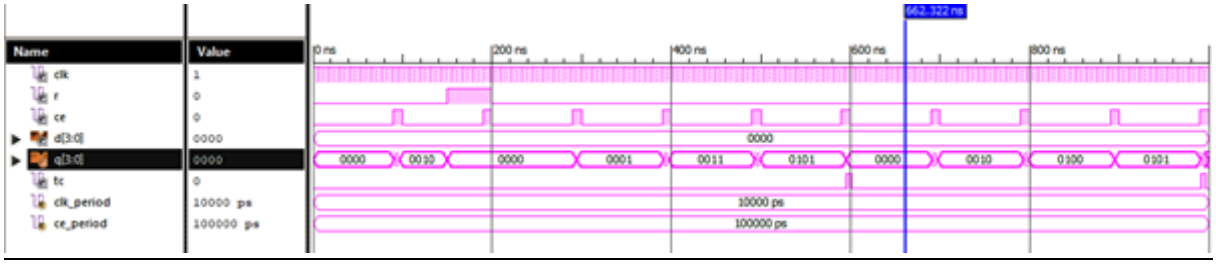
- Chronogramme



- Zoom du chronogramme précédent



- Chronogramme



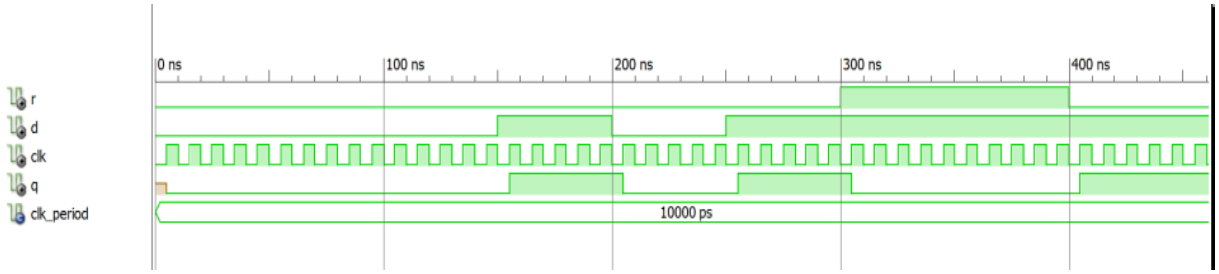
Sur ce chronogramme on visualise bien que lors du passage à 5 on à TC qui copie la clock sur une période.

- Post-Route

On observe ainsi un retard de 4.372 ns

<u>Fonction Register_1b_R:</u>

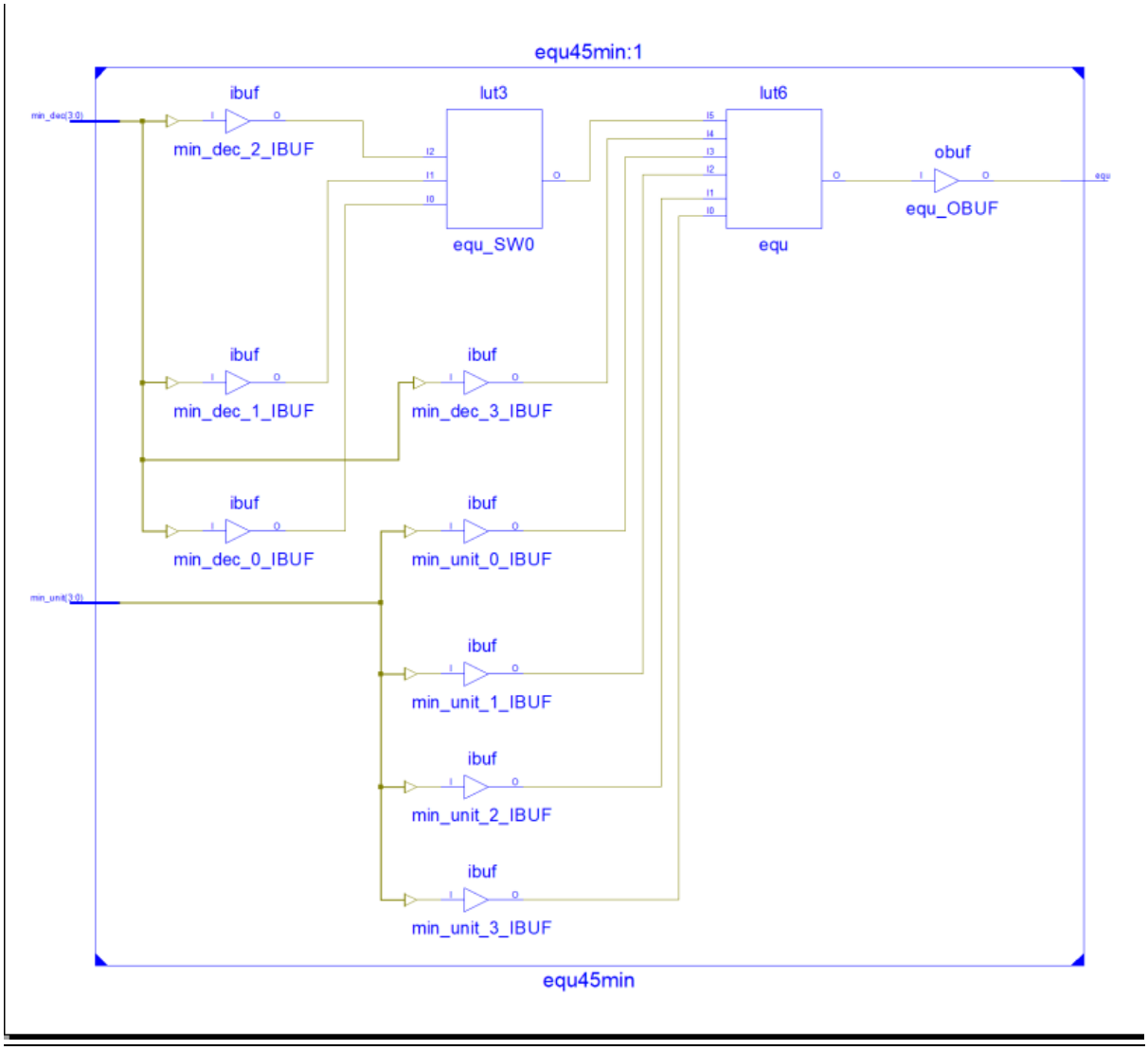
- Simulation Post-Route



On observe le temps d'initialisation de l'entrée avec le décalage orange. De plus, lorsque Reset passe à un, la sortie se comporte inversement et passe à zéro.

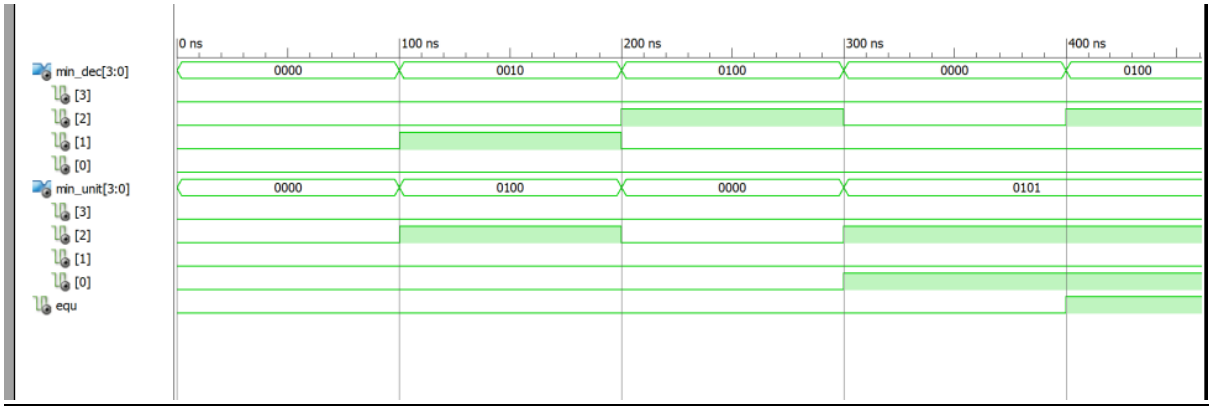
Fonction equ45min:

- Vue Technologique



Sur la vue technologique on observe les buffers permettant tout d'abord de synchroniser les signaux ainsi que de rehausser si nécessaire leur état bas pour une bonne lecture de ces derniers. Ensuite sont ajoutés deux lut dans le but de gérer les contraintes physiques dues à l'organisation de la place des composants sur le FPG. Chaque lut n'ayant que 5 entrées, il y en a deux pour gérer les 10 signaux envoyés.

- Simulation Behavioral



Tous les programmes se trouvent sur le lien suivant :
https://github.com/AudreyNicolle/Projet_ELN2_Scoring_2.0.git