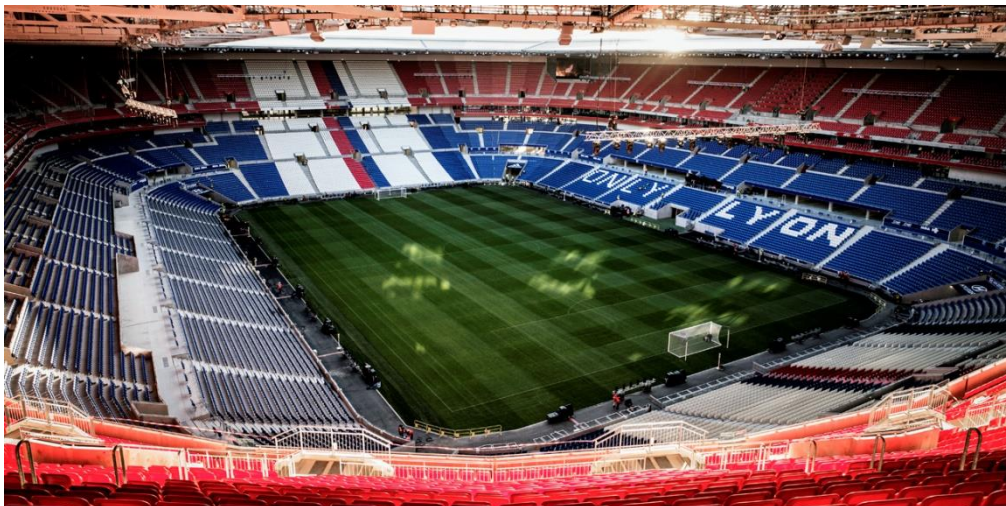


PROJET SCORING 2.0 : COMPTE-RENDU

*Projet Scoring 2.0 : Afficheur Durée de Match et Score pour le
Stade des Lumières*



Sommaire

1) Introduction

- a. Cahier des charges
- b. Méthode d'analyse

2) Entité « display »

- a. transcoder_2v4
- b. mux_4x1x1b
- c. mux_4x1x4b
- d. transcoder_7segs
- e. register_4b
- f. register_8b
- g. Tregister_1b
- h. counter_2b_E
- i. display

3) Entité « multiplexdata »

- a. mux_2x1x4b
- b. freqgen_4b_E
- c. multiplexdata

4) Entité « chronometer »

- a. OR
- b. register_1b_R
- c. counterSen_4b_RE
- d. counterDec_4b_RE
- e. equ45min
- f. AND4n2
- g. chronometer

5) Entité « score »

- a. register_1b_E
- b. register_1b
- c. XOR
- d. NAND
- e. score

6) Entité « chronoscore »

- a. chronoscore

7) Bilan

- a. Technique
- b. Personnel

I. Introduction

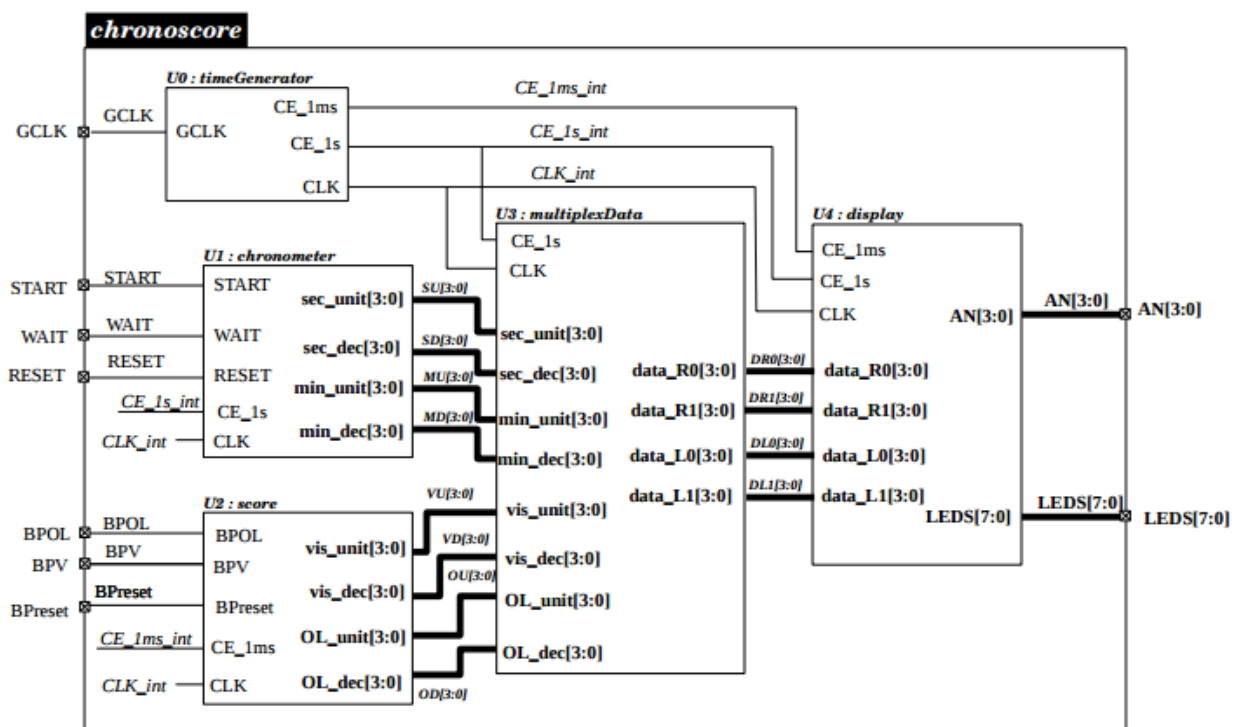
a) Cahier des charges

L'objectif de ce projet est de réaliser un afficheur de durée et de score d'un match de foot pour la société « Olympique Lyonnais ». Ce système sera intégré dans le Stade des Lumières.

Cet afficheur permet d'afficher sur un écran avec une alternance de 10 secondes:

- Le temps écoulé au format minutes-secondes
- Le score sur des Leds (un bouton-poussoir permettra de simuler un but marqué)

L'affichage des données sur un écran VGA sera l'objet d'une 2^{ème} partie de projet et ne sera pas évoquée dans ce rapport. Le système global sera représenté par une entité **chronoscore** :



Cette entité se décompose en 5 sous-entités :

- Une entité **timeGenerator** : celle-ci génère un signal d'horloge stable et des signaux périodiques non carrés au cœur du FPGA.
- Une entité **chronometer** : celle-ci génère des valeurs binaires en base 10 qui s'incrémentent chaque seconde/minute jusqu'à une valeur limite de 45min (correspond à la mi-temps du match).
- Une entité **score** : celle-ci génère des valeurs binaires en base 10 qui s'incrémentent à chaque nouveau but d'un des 2 camps.

- Une entité **multiplexData** : c'est un multiplexeur qui permet de sélectionner des données relatives au temps et au score (générées par les entités chronometer et score) et de les transmettre à une entité d'affichage.
- Une entité **display** : celle-ci réalise l'affichage des données de multiplexdata sur 4 afficheurs 7-segments.

b) Méthode d'analyse

Dans le but d'acquérir peu à peu une démarche d'ingénieur au cours de notre cursus, nous réaliserons pour chaque sous-bloc de chaque entité une étude fine. Nous réaliserons le plus possible d'interprétations des résultats obtenus et mettrons en évidence les éléments pertinents.

Dans un premier temps, nous coderons chaque sous-bloc en langage VHDL avec le logiciel ISE Design Tools de Xilinx, les codes seront disponible en annexe. Nous réaliserons également les bancs de tests associés à chaque sous-bloc.

Pour la plupart des sous-blocs, nous réaliserons une simulation comportementale pour vérifier la cohérence avec la table de vérité. Ensuite, nous réaliserons une simulation temporelle et relèverons les contraintes temporelles, à savoir les temps de propagation sans prendre en compte les temps des buffers. Nous en déduirons la fréquence d'utilisation maximum associée. Dans le cas des bascules, nous nous intéresserons au temps de setup. Lors de l'utilisation future, le respect de cette fréquence permettra d'éviter des erreurs de hold (signal trop rapide, risque de négliger un état logique) et de setup (signal trop lent, risque de rester trop longtemps sur un état stable).

Nous relèverons également le nombre de LUT utilisés pour créer l'entité et le comparerons au nombre de LUT total du système global (3840). Les LUT (« Look Up Table ») sont les composants de base des FPGA. Ils permettent de réaliser des circuits logiques génériques. Ce sont des tables de correspondances stockées en mémoires qui servent à implémenter des équations logiques, on peut d'ailleurs retrouver ces équations en étudiant la table de vérité du composant. Les LUT sont reliées entre elles par un routage automatique grâce au logiciel.

Nous nous intéresserons aussi à d'autres données : le nombre de buffers entrée/sortie et leur comparaison avec le nombre total de buffers du système (173).

Pour les composants manipulant des incréments de bits (registres, compteurs, chronomètres...), il sera intéressant de relever le nombre de bascules.

II. Entité « display »

Il s'agit du sous bloc permettant de gérer les données issues du multiplexdata afin de les afficher correctement sur 4 afficheurs 7 segments.

a) Transcoder 2v4

Le rôle de cet élément est de convertir un signal sur 2bits en un signal sur 4bits qui permettra la sélection des 4afficheurs en sortie. L'entité à concevoir est la suivante :

transcoder_2v4	
A[1:0]	O[3:0]
00	1110
01	1101
10	1011
11	0111

Code :

```
entity transcoder_2v4 is
    Port ( A : in  STD_LOGIC_VECTOR (1 downto 0);
          O : out  STD_LOGIC_VECTOR (3 downto 0));
end transcoder_2v4;

architecture Behavioral of transcoder_2v4 is

begin

    WITH A SELECT

        O <=  "1110" WHEN "00",
              "1101" WHEN "01",
              "1011" WHEN "10",
              "0111" WHEN others;

end Behavioral;
```

Simulation comportementale :

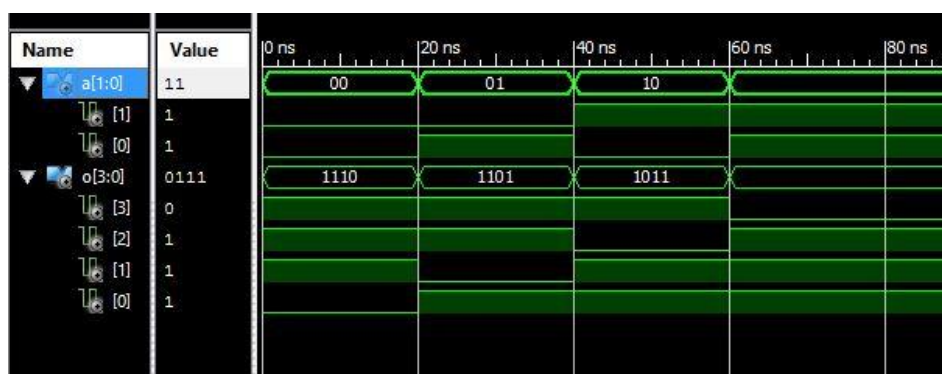


Figure 1 : Simulation comportementale de l'entité « transcoder_2v4 »

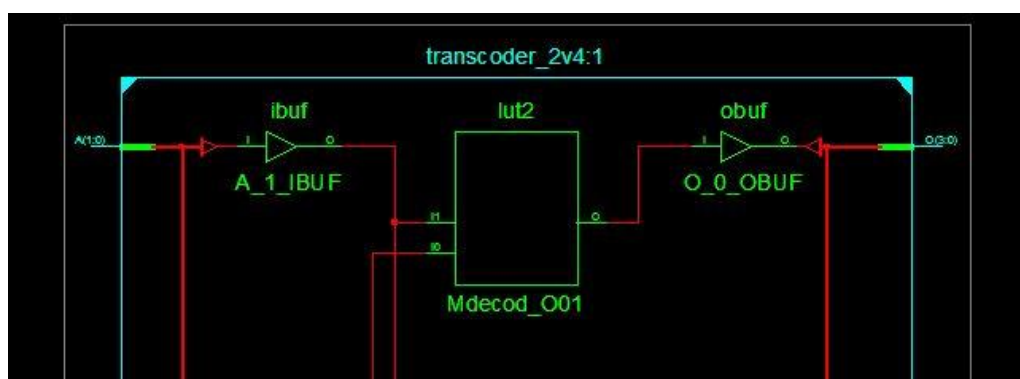
On remarque que la table de vérité est vérifiée. On associe à une valeur de A sur 2 bits une valeur de O sur 4 bits. C'est un circuit convertisseur qui permet le passage d'un code d'entrée à un code de sortie.

La console nous renvoie des informations sur les composants utilisés :

```
=====
Advanced HDL Synthesis Report

Macro Statistics
# Decoders                               : 1
  1-of-4 decoder                           : 1
=====
```

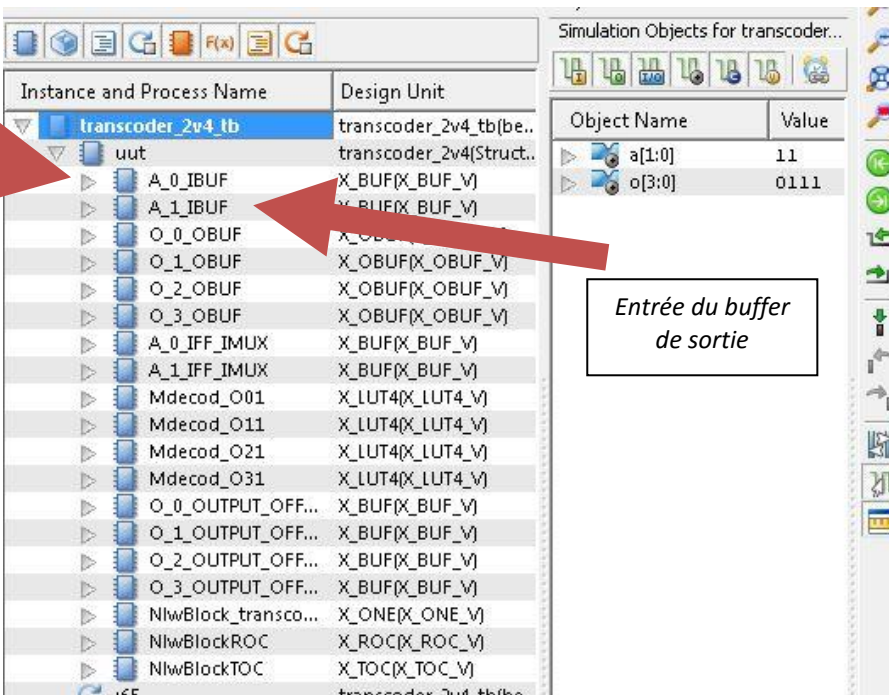
Ici, 4 décodeurs sont utilisés. Ceux-ci sont représentés par des LUT (« Look Up Table »), visible sur un schéma technologique. Nous pouvons représenter le composant par un schéma technologique :



On remarque la présence des buffers d'entrée et de sortie. Ces buffers apparaissent car nous sommes en mode « Top module », il faudra donc ne pas les prendre en compte lors d'une étude temporelle car ils seront seulement en entrée et sortie de l'entité globale. Nous allons maintenant relever les contraintes temporelles, à savoir les temps de propagation. Ces temps de propagation représentent le temps que met l'information pour traverser le composant. Ces composants se représentent donc par des LUT (en parallèle dans notre cas), nous prendrons également en compte les fils autour de ces LUT. L'information met également un temps conséquent à traverser ces fils car en réalité ceux-ci représentent un grand nombre de transistors. De plus, un fil est un filtre passe-bas admettant un temps τ (à 66%).

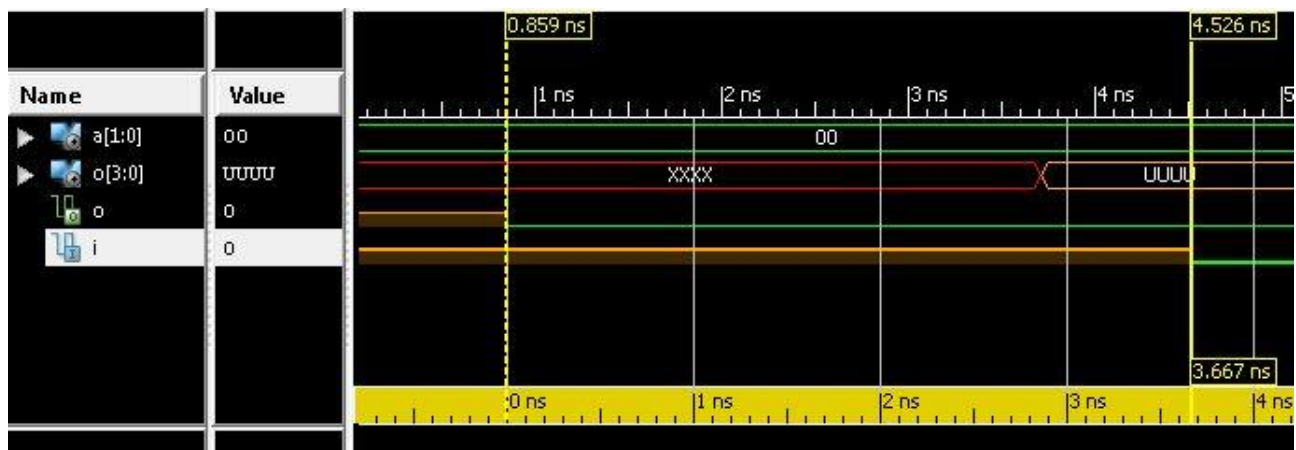
Pour relever des temps, on réalisera donc une simulation temporelle Post-Route. On analysera les signaux suivants :

Sortie du buffer d'entrée



Entrée du buffer de sortie

On relève ensuite les temps en utilisant la méthode suivante :



Simulation temporelle :

$$\tau_{propagation} = 3,667 \text{ ns}$$

$$f_{utilisationmax} = 272,7 \text{ MHz}$$

Nous pouvons également nous intéresser au nombre de composants utilisés.

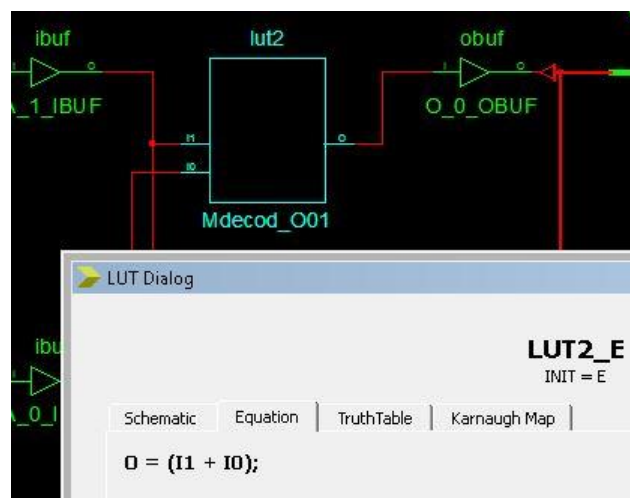
Composants :

4 LUT soit **0,1%** du nombre de LUT total disponible utilisé

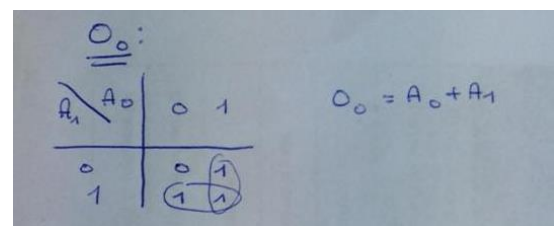
6 buffers entrée/sortie soit **3%** du nombre de buffer total disponible utilisé

2 Slices utilisées

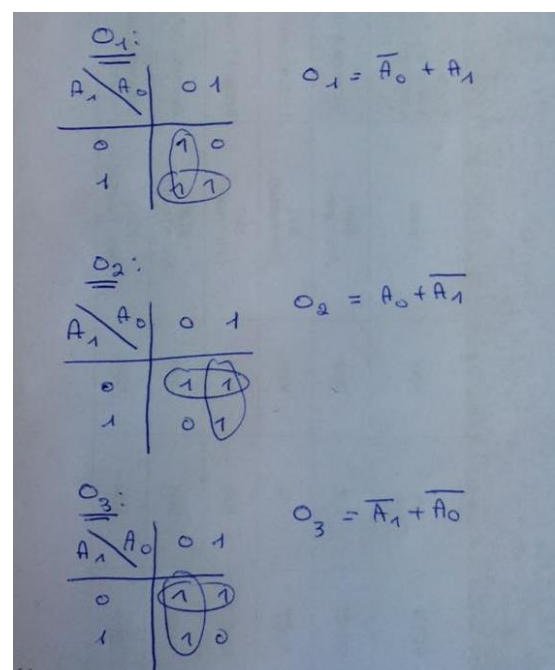
On remarque que le nombre de LUT utilisé est très faible comparé au nombre total de LUT disponible. Par ailleurs, le nombre de buffers est quant à lui plus important. On peut s'intéresser aux équations réalisées, celle pour la LUT n°1 est :



- Un point qui peut donc être intéressant est de faire l'équation à la main et voir si on tombe sur la même équation. On a donc :

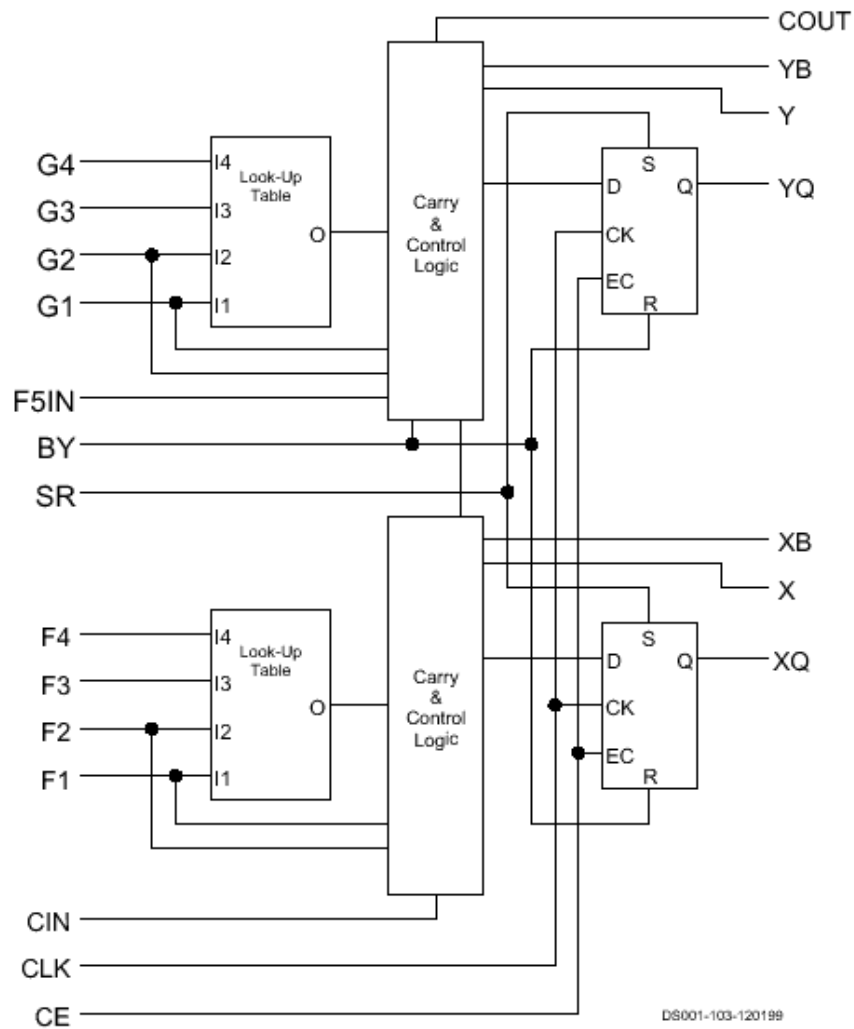


- Pour les autres LUT, on retrouve les équations suivantes qui correspondent bien aux équations données sur Xilinx :



- Architecture de base d'une slice

Lors du projet, on peut remarquer que le logiciel utilise des slices, il s'agit d'un élément composé de deux LUT ainsi que deux bascules D :



b) Mux 4x1x1b

Il s'agit ici d'un multiplexeur permettant l'orientation de 4 différents signaux vers une seule sortie en fonction de la valeur des 2bits de sélection (sel). L'entité à ce concevoir est la suivante :

mux_4x1x1b						
A	B	C	D	sel[1:0]		O
A _n	-	-	-	00		A _n
-	B _n	-	-	01		B _n
-	-	C _n	-	10		C _n
-	-	-	D _n	11		D _n

Code :

```
entity mux_4x1x1b is
    Port ( A : in  STD_LOGIC;
          B : in  STD_LOGIC;
          C : in  STD_LOGIC;
          D : in  STD_LOGIC;
          sel : in  STD_LOGIC_VECTOR (1 downto 0);
          O : out  STD_LOGIC);
end mux_4x1x1b;

architecture Behavioral of mux_4x1x1b is
begin
    WITH sel SELECT
        O <= A WHEN "00",
             B WHEN "01",
             C WHEN "10",
             D WHEN others;
end Behavioral;
```

Simulation comportementale :

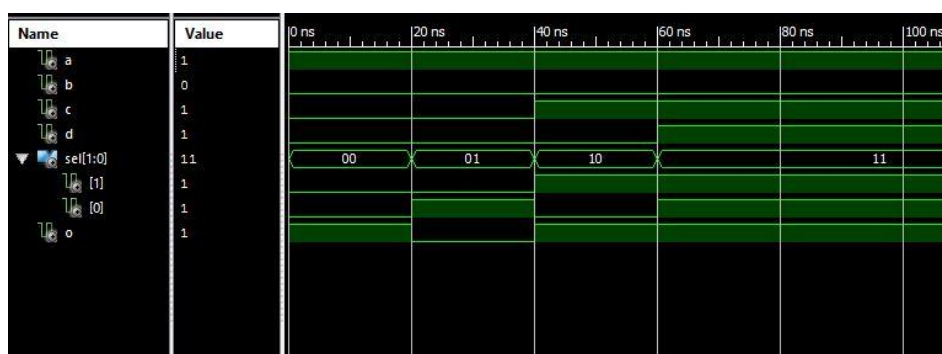


Figure 2 : Simulation comportementale de l'entité « mux_4x1x1b »

Ainsi, on remarque bien que la table de vérité est vérifiée.

Simulation temporelle :

$$\tau_{propagation} = 3,392 \text{ ns}$$

$$f_{utilisationmax} = 294,8 \text{ MHz}$$

Composants :

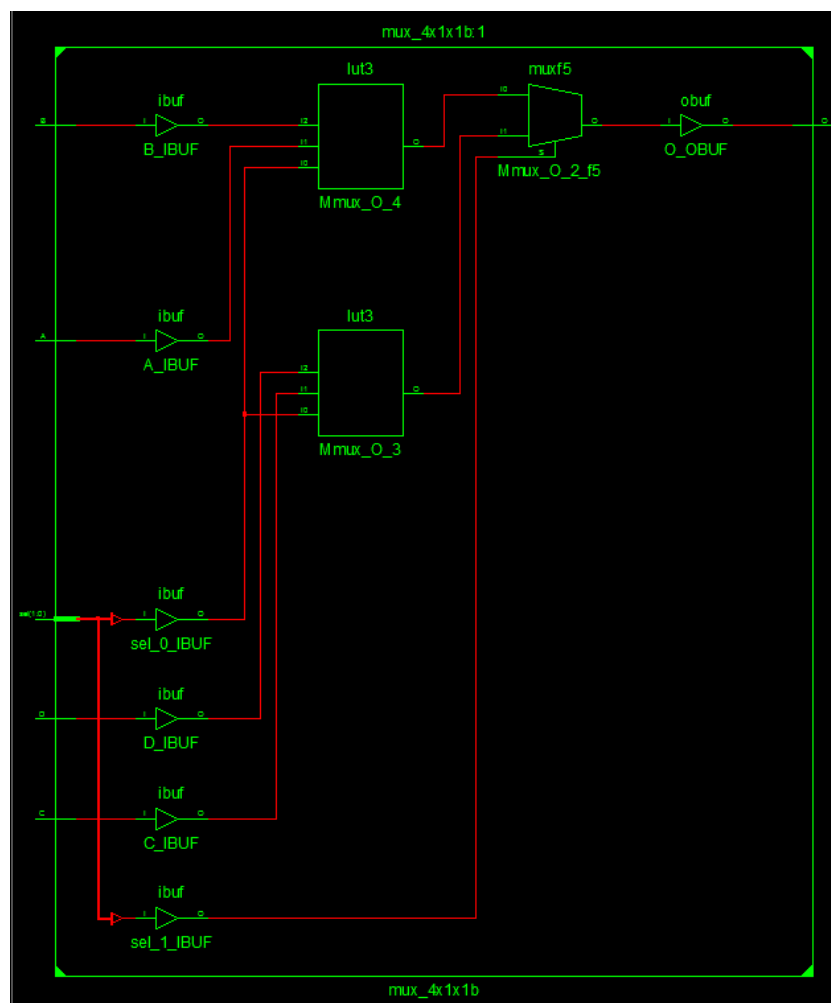
2 LUT soit **0,2%** du nombre de LUT total disponible utilisé

7 buffers entrée/sortie soit **4%** du nombre de buffer total disponible utilisé

1 Slice utilisée

On remarque que le nombre de LUT utilisé est très faible comparé au nombre total de LUT disponible. Le nombre de buffers est quant à lui plus important. Nous n'utilisons pas dans ce cas de bascules.

Vue technologique d'un multiplexeur :



c) Mux 4x1x4b

L'entité à concevoir est la suivante :

mux_4x1x4b					
A[3:0]	B[3:0]	C[3:0]	D[3:0]	sel[1:0]	O[3:0]
A _n	-	-	-	00	A _n
-	B _n	-	-	01	B _n
-	-	C _n	-	10	C _n
-	-	-	D _n	11	D _n

Code :

```
entity mux_4x1x4b is
    Port ( A : in  STD_LOGIC_VECTOR (3 downto 0);
          B : in  STD_LOGIC_VECTOR (3 downto 0);
          C : in  STD_LOGIC_VECTOR (3 downto 0);
          D : in  STD_LOGIC_VECTOR (3 downto 0);
          sel : in  STD_LOGIC_VECTOR (1 downto 0);
          O : out  STD_LOGIC_VECTOR (3 downto 0));
end mux_4x1x4b;

architecture Behavioral of mux_4x1x4b is

begin

WITH sel SELECT
    O <= A WHEN "00",
         B WHEN "01",
         C WHEN "10",
         D WHEN others;

end Behavioral;
```

Simulation comportementale :

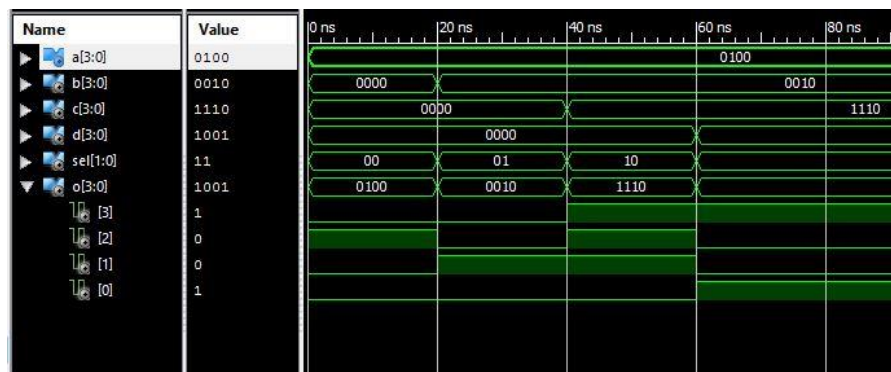


Figure 3 : Simulation comportementale de l'entité « mux_4x1x4b »

Cette simulation correspond bien au comportement attendu, et respecte bien la table de vérité.

Simulation temporelle :

$$\tau_{propagation} = 4,489 \text{ ns}$$

$$f_{utilisationmax} = 222,8 \text{ MHz}$$

Composants :

8 LUT soit **0,2%** du nombre de LUT total disponible utilisé

22 buffers entrée/sortie soit **12%** du nombre de buffer total disponible utilisé

4 Slices utilisées

On remarque que le nombre de LUT et de buffers est le même que pour le composant « mux_4x1x1b », ce qui est normal.

d) Transcoder 7segs

On associe à une valeur de A sur 4 bits une valeur de O sur 8 bits. C'est un circuit convertisseur qui permet le passage d'un code d'entrée à un code de sortie. Il assure ainsi la bonne translate du signal sur 4bits en un signal sur 8bits afin d'être compréhensible par nos afficheurs 7 segments en aval. L'entité à concevoir est la suivante :

transcoder_7segs		
A[3:0]	O[6:0]	Caractère associé
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	0000000	8
1001	0010000	9
1010	0001000	A
1011	0000011	B
1100	1000110	C
1101	0100001	D
1110	0000110	E
1111	0001110	F

Code :

```

entity transcoder_7segs is
    Port ( A : in  STD_LOGIC_VECTOR (3 downto 0);
          O : out  STD_LOGIC_VECTOR (6 downto 0));
end transcoder_7segs;

architecture Behavioral of transcoder_7segs is

begin

WITH A SELECT

    O <= "1111001" when "0001",
        "0100100" when "0010",
        "0110000" when "0011",
        "0011001" when "0100",
        "0010010" when "0101",
        "0000010" when "0110",
        "1111000" when "0111",
        "0000000" when "1000",
        "0010000" when "1001",
        "0001000" when "1010",
        "0000011" when "1011",
        "1000110" when "1100",
        "0100001" when "1101",
        "0000110" when "1110",
        "0001110" when "1111",
        "1000000" when others;

end Behavioral;

```

Simulation comportementale :

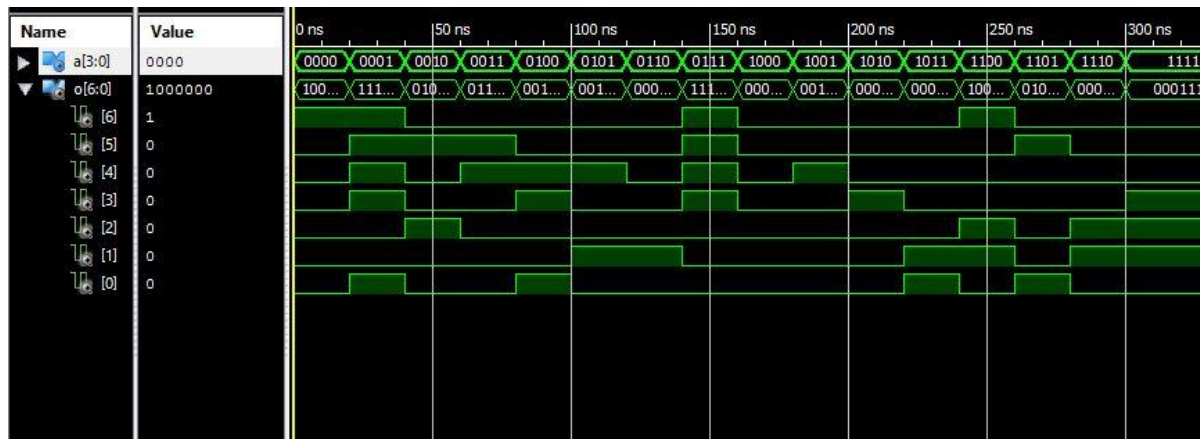


Figure 4 : Simulation comportementale de l'entité « transcoder_7segs »

Simulation temporelle :

$$\tau_{propagation} = 3,641 \text{ ns}$$

$$f_{utilisationmax} = 274,6 \text{ MHz}$$

Composants :

7 LUT soit **0,2%** du nombre de LUT total disponible utilisé

11 buffers entrée/sortie soit **6%** du nombre de buffer total disponible utilisé

4 Slices utilisées

On remarque que le nombre de LUT reste très bas et que le nombre de buffer moins élevé que dans le cas d'un multiplexeur.

e) register_4b

Cet élément est un registre 4bits actif sur front montant, il permet de faire une sauvegarde de l'information propagée. L'entité à concevoir est la suivante :

register_4b		
D[3:0]	clk	Q _{n+1} [3:0]
D _n	\nearrow	D _n
-	-	Q _n

Code :

```
entity register_4b is
  Port ( D : in  STD_LOGIC_VECTOR (3 downto 0);
        clk : in  STD_LOGIC;
        Q : out  STD_LOGIC_VECTOR (3 downto 0));
end register_4b;

architecture Behavioral of register_4b is

  signal Q_decalage : std_logic_vector (3 downto 0) := "0000"; --Initialisation du signal temporaire utilisé pour le décalage

begin

  PROCESS (clk)
  BEGIN
    IF (rising_edge(clk)) THEN
      Q_decalage <= D;
    END IF;
  END PROCESS;

  Q <= Q_decalage;

end Behavioral;
```

Simulation comportementale :

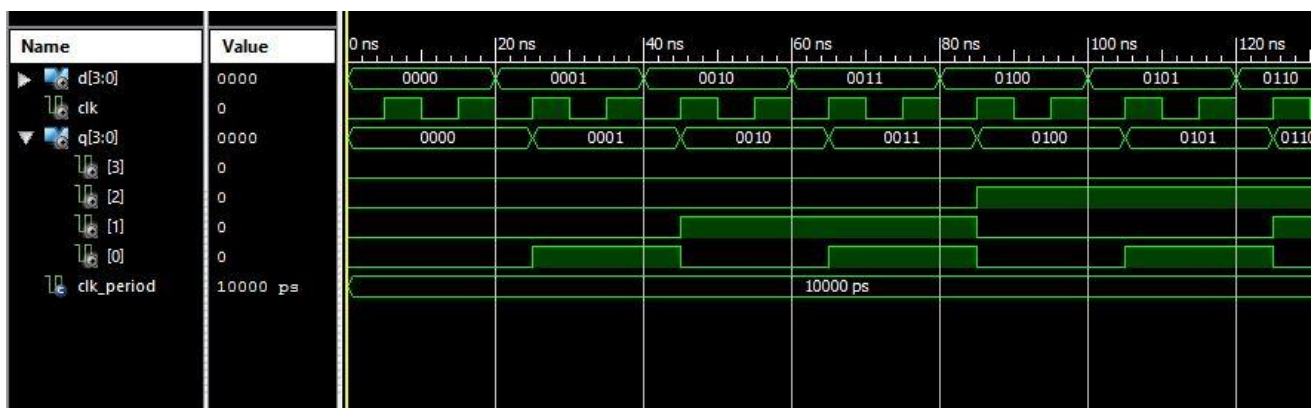


Figure 5 : Simulation comportementale de l'entité « register_4b »

Simulation temporelle :

$$\tau_{propagation} =$$

$$\tau_{propagation} = 9,141ns$$

$$f_{utilisationmax} = 109.4 MHz$$

Composants :

4 bascules D utilisées

9 buffers entrée/sortie soit **5%** du nombre de buffer total disponible utilisé

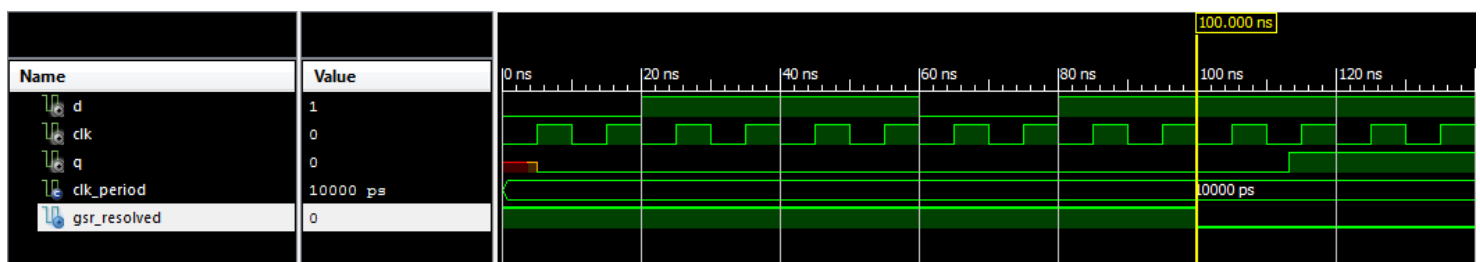
Pas d'utilisation de LUT

On remarque que 4 bascules D sont utilisées, ce qui est logique car c'est un registre à décalage 4 bits. Un point important à souligner avec l'utilisation de bascules est le signal GSR :

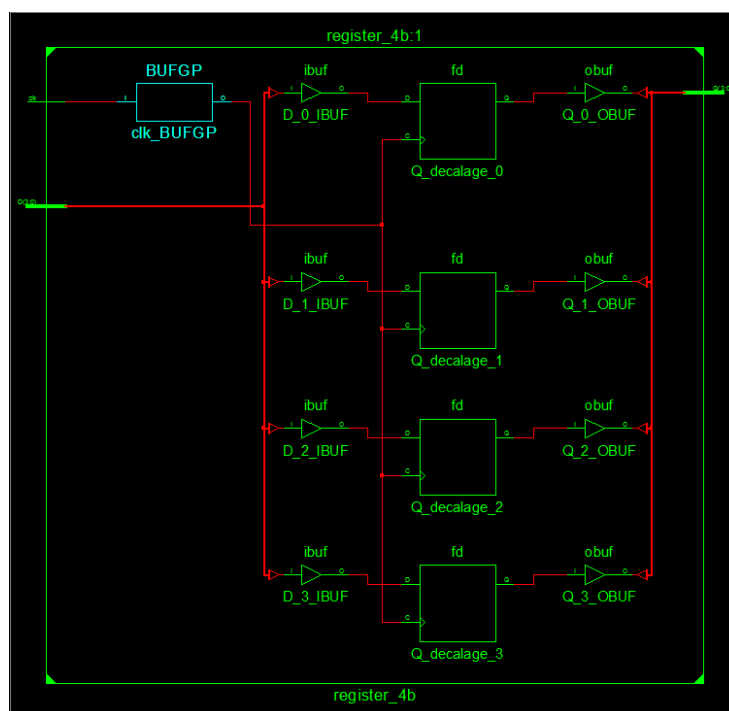
- Mise en évidence du GSR (Global Set & Reset)

Il s'agit du temps nécessaire pour l'initialisation des différentes bascules utilisées (configurer le FPGA), le constructeur le fixe à 100ns ainsi pendant les 100 premières ns, les bascules ne changent pas d'état.

Mise en évidence :




- Vue technologique d'un registre 4 bits :



f) Register 8b

L'entité à concevoir est la suivante :

register_8b		
D[7:0]	clk	Q _{n+1} [7:0]
D _n		D _n
-	-	Q _n

Code :

```
entity register_8b is
    Port ( D : in  STD_LOGIC_VECTOR (7 downto 0);
          clk : in  STD_LOGIC;
          Q : out  STD_LOGIC_VECTOR (7 downto 0));
end register_8b;

architecture Behavioral of register_8b is

    signal Q_decalage : std_logic_vector (7 downto 0) := "00000000";

begin

    PROCESS (clk)
    BEGIN
        IF (rising_edge(clk)) THEN
            Q_decalage <= D;
        END IF;
    END PROCESS;

    Q <= Q_decalage;

end Behavioral;
```

Simulation comportementale :

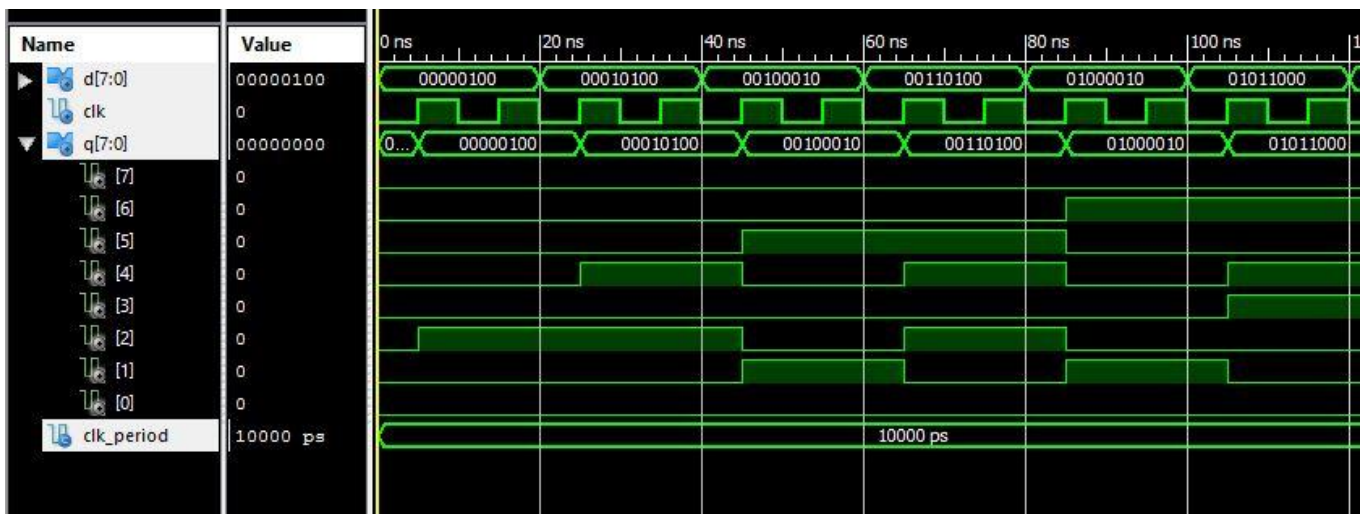


Figure 6 : Simulation comportementale de l'entité « register_8b »

Simulation temporelle :

$$\tau_{propagation} = 8,359ns$$

$$f_{utilisationmax} = 119,6MHz$$

Composants :

8 bascules D utilisées

17 buffers entrée/sortie soit **9%** du nombre de buffer total disponible utilisé

Pas d'utilisation de LUT

On remarque que 8 bascules D sont utilisées, ce qui est logique car c'est un registre à décalage 8 bits.

g) Tregister 1b

Cet élément est une bascule T active sur front montant. Il permet de générer une horloge d'une seconde dans notre cas. L'entité à concevoir est la suivante :

Tregister_1b		
T	clk	Q_{n+1}
0	↗	Q_n
1	↗	toggle
-	-	Q_n

Code :

```
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_interne : STD_LOGIC := '0';

begin

    PROCESS (clk)
    BEGIN
        IF (rising_edge(clk)) THEN
            IF (T='1') THEN
                Q_interne <= NOT(Q_interne);
            END IF;
        END IF;

    END PROCESS;

    Q<=Q_interne;

end Behavioral;
```

Simulation comportementale :

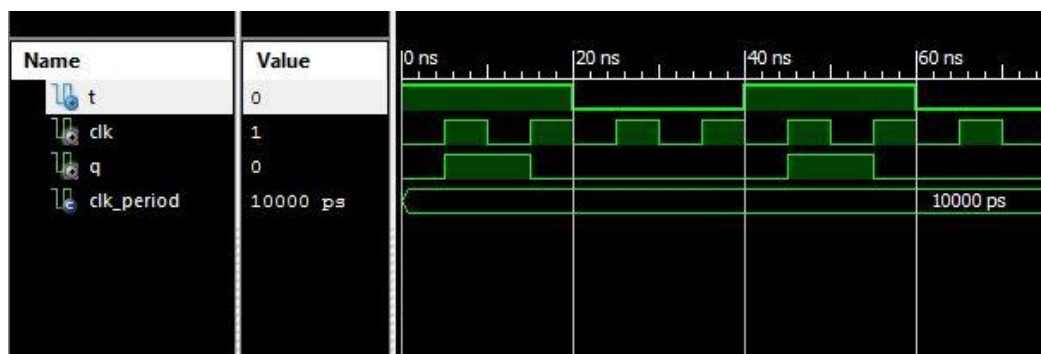


Figure 7 : Simulation comportementale de l'entité « Tregister_1b »

On remarque que la table de vérité est vérifiée. Quand T est à 1, q change d'état au prochain front montant. On s'intéresse au cas suivant :

Simulation temporelle :

$$\tau_{propagation} = 8,4 \text{ ns}$$

$$f_{utilisationmax} = 119 \text{ MHz}$$

Composants :

1 LUT soit **0,03%** du nombre de LUT total disponible utilisé

3 buffers entrée/sortie soit **0,02%** du nombre de buffer total disponible utilisé

1 bascule D utilisée

1 Slice utilisée

On remarque que peu de composants sont utilisés. On utilise une bascule D et un inverseur pour réaliser une bascule T.

h) Counter 2b E

Cet élément représente un compteur sur 2bits, il va générer l'entrée de sélection qui ira ensuite dans le mux_4x1x4b par exemple. L'entité à concevoir est telle que :

counter_2b_E			
CE	Q _n	clk	Q _{n+1}
1	00	↗	01
1	01	↗	10
1	10	↗	11
1	11	↗	00
0	-	-	Q _n

Code :

```
entity counter_2b_E is
    Port ( CE : in  STD_LOGIC;
          clk : in  STD_LOGIC;
          Q : out  STD_LOGIC_VECTOR (1 downto 0));
end counter_2b_E;

architecture Behavioral of counter_2b_E is

    signal Q_interne : STD_LOGIC_VECTOR (1 downto 0) := "00";

begin

    PROCESS (clk)
    BEGIN
        IF (rising_edge(clk)) THEN
            IF (CE='1') THEN
                Q_interne <= Q_interne+1;
            END IF;
        END IF;

    END PROCESS;

    Q<=Q_interne;

end Behavioral;
```

Simulation comportementale :

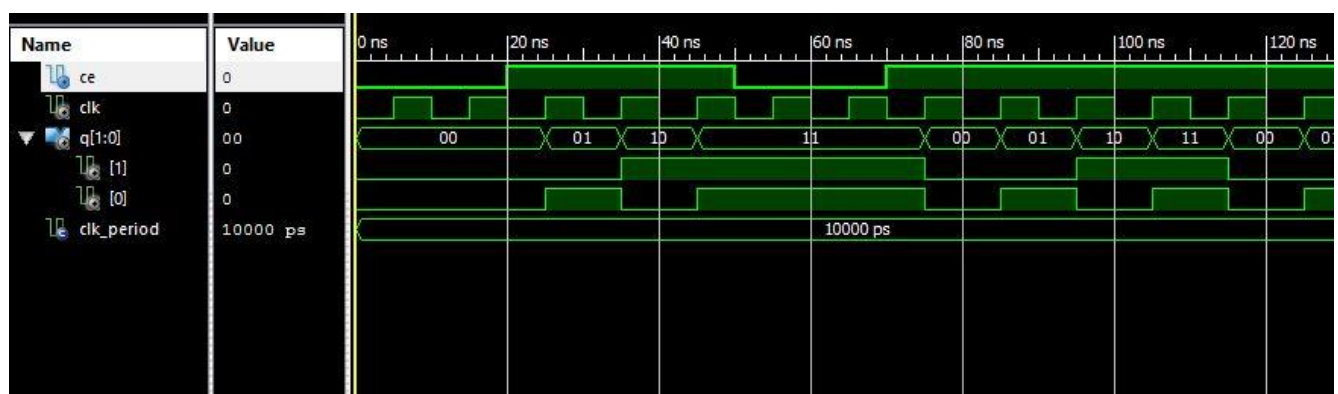


Figure 8 : Simulation comportementale de l'entité « counter_2b_E »

On remarque que la table de vérité est bien vérifiée. Quand CE est à 1 (qui indique soit le débordement du compteur précédent ou un signal d'horloge) le compteur s'incrémente correctement et quand CE est à 0, l'incrémement du compteur est suspendue. On peut observer qu'on compte bien jusqu'à 3 puis revenons à 0 au prochain coup d'horloge.

Simulation temporelle :

$$\tau_{propagation} = 8,75ns$$

Composants :

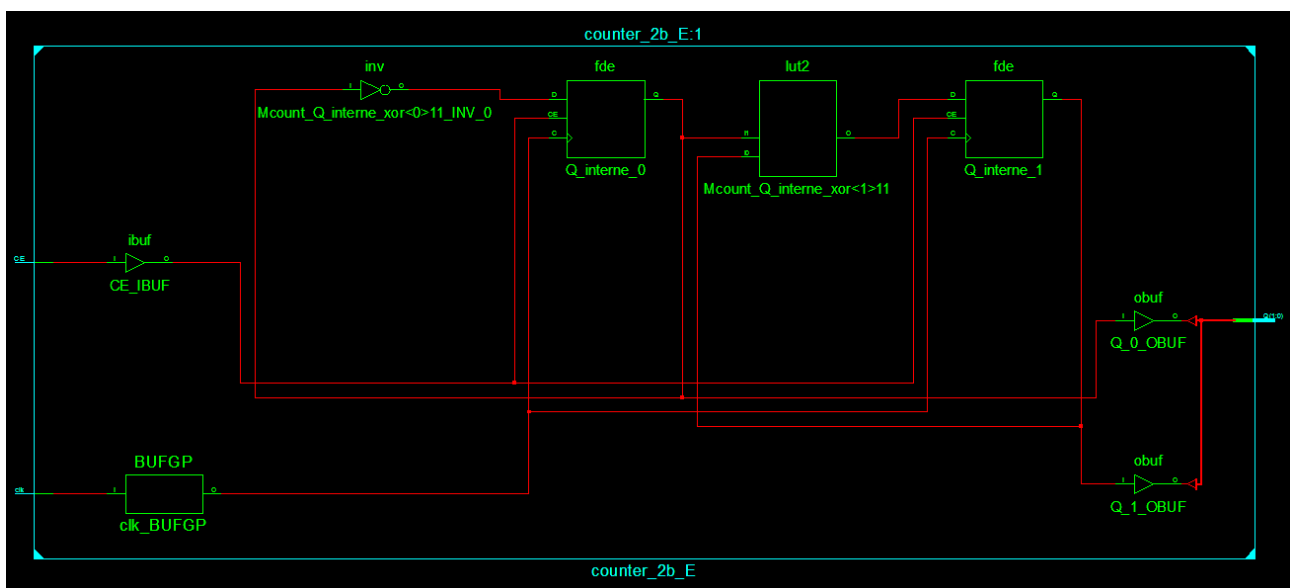
1 LUT soit **0,025%** du nombre de LUT total disponible utilisé

4 buffers entrée/sortie soit **0,02%** du nombre de buffer total disponible utilisé

2 bascules D utilisées

1 Slices utilisée

On a, pour le counter_2b, cette vue technologique :



Si on analyse l'équation de la LUT, on a :

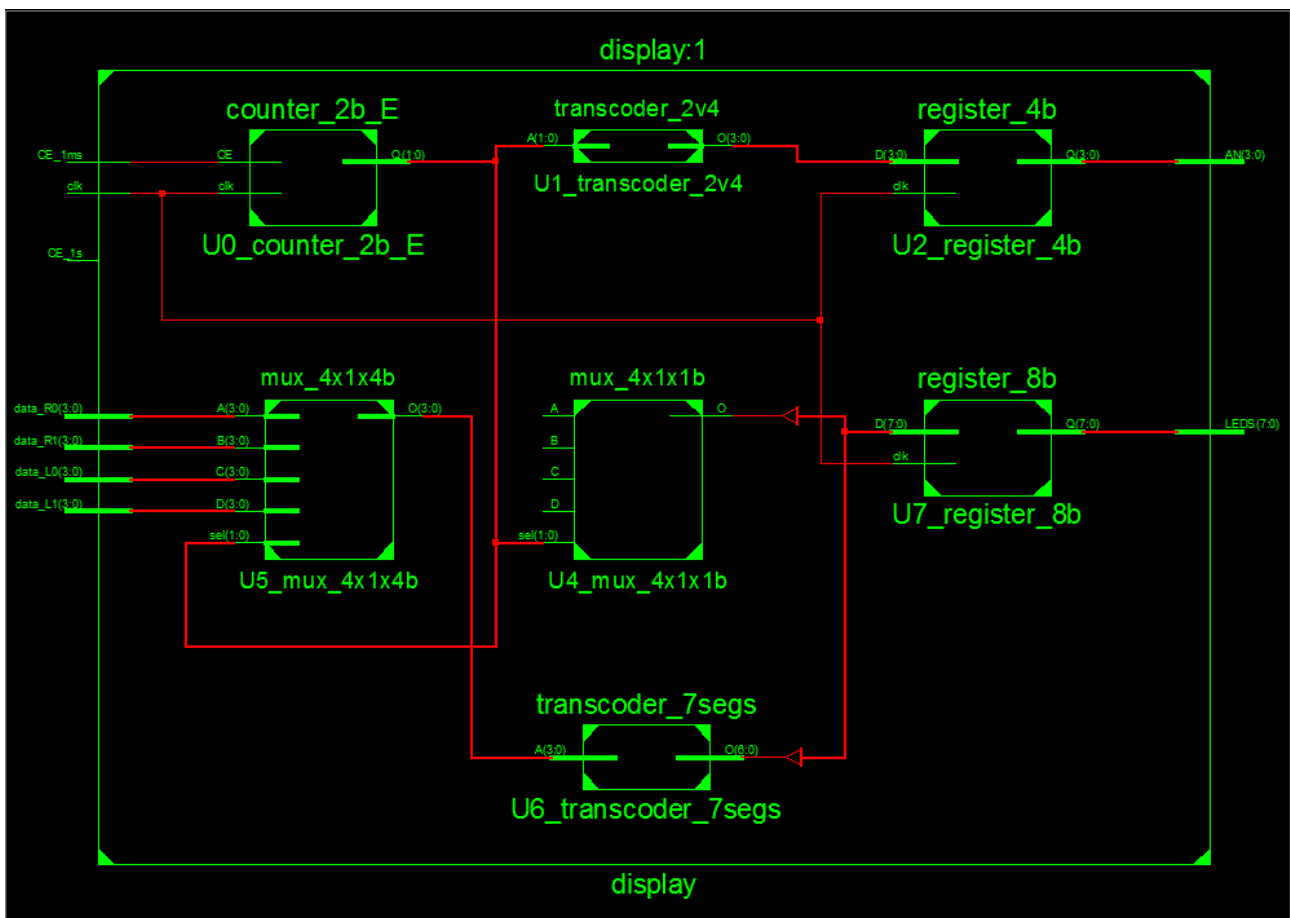
$$O = ((!IO * I1) + (IO * !I1));$$

Or cette équation correspond bien à l'équation d'un compteur qu'on a pu voir en cours par exemple :

$$D_i = Q_i \pi (Q_j \text{ XOR } Q_{j-1}).$$

i) Display

Il s'agit du sous bloc qui a pour rôle de gérer les données issues de multiplexdata afin de les afficher sur



Voici un tableau récapitulatif des différents éléments constituant notre sous bloc display :

	LUT	Slice	Buffers	% Buffers	Bascules	Temps de propagation (ns)	Fréquence maximum d'utilisation (MHz)
transcoder_2v4	4	2	6	3,5	0	3,667	272,7
mux_4x1x1b	2	1	7	4	0	3,392	294,8
mux_4x1x4b	8	4	22	13	0	4,489	222,8
transcoder_7segs	7	4	11	4	0	3,641	274,6
register_4b	0	0	9	5,2	4	9,141	109,4
register_8b	0	0	17	9,8	8	8,359	119,6
Tregister_1b	0	1	3	1,7	1	8,4	119
counter_2b_E	1	1	4	2,3	2	9,75	102,6
TOTAL display	22	13	79	54,3	15	50,839	

L'information met alors 50,839 ns pour transiter depuis les entrées data_R0, data_R1, data_L0, data_L1 vers les sorties LEDS. On remarque par ailleurs qu'il faut bascules 14 D, 17 LUT et 13 Slices pour concevoir cet élément, on en déduit donc que Xilinx procède à une simplification lors de la compilation de tous les éléments entre eux ce que l'on avait déjà pu observer avec le projet météo d'eln1. Par ailleurs, on a pu observer qu'on pouvait avoir l'utilisation de bascules D sans forcément de Slice, en regardant de plus près, on a remarqué qu'on pouvait, lorsqu'on n'utilise pas de LUT, utiliser les bascules D des buffers d'entrée.

III. Entité « multiplexdata »

a) Mux 2x1x4b

Cet élément est nécessaire afin d'afficher, selon l'entrée de sélection sel, d'envoyer à l'afficheur soit la durée du match soit le tableau des scores des équipes. L'entité à concevoir est la suivante :

mux_2x1x4b				
A[3:0]	B[3:0]	sel		O[3:0]
A _n	-	0		A _n
-	B _n	1		B _n

Code :

```
entity mux_2x1x4b is
    Port ( A : in  STD_LOGIC_VECTOR (3 downto 0);
          B : in  STD_LOGIC_VECTOR (3 downto 0);
          sel : in  STD_LOGIC;
          O : out STD_LOGIC_VECTOR (3 downto 0));
end mux_2x1x4b;

architecture Behavioral of mux_2x1x4b is

begin

    O <= A WHEN sel = '0' ELSE
        B WHEN sel = '1';

end Behavioral;
```

Simulation comportementale :

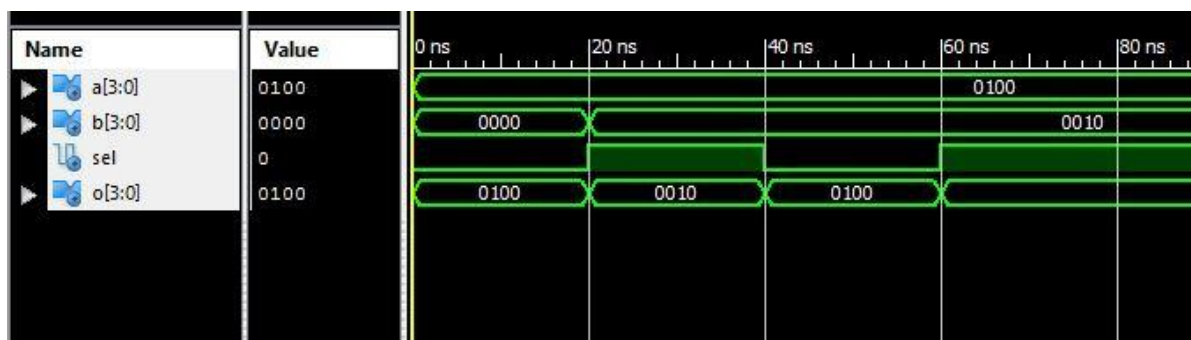


Figure 9 : Simulation comportementale de l'entité « mux_2x1x4b »

Simulation temporelle :

$$\tau_{propagation} = 8,396 \text{ ns}$$

$$f_{utilisationmax} = 119\text{MHz}$$

Composants :

4 LUT soit **0,1%** du nombre de LUT total disponible utilisé

13 buffers entrée/sortie soit **7,5%** du nombre de buffer total disponible utilisé

Pas d'utilisation de bascules

b) Freggen 4b E

Lorsque l'entrée CE synchrone est à l'état haut, cet élément, actif sur front montant du signal d'horloge, génère un signal carré dont la période est de 10secondes. Il permet de faire l'alternance entre temps du match et score des équipes. L'entité à concevoir est telle que :

freggen_4b_E					
CE	Qint _n	clk	Qint _{n+1}	clk_5s _{n+1}	
1	0000	↗	0001	0	
1	0001	↗	0010	0	
1	0010	↗	0011	0	
1	0011	↗	0100	0	
1	0100	↗	0101	1	
1	0101	↗	0110	1	
1	0110	↗	0111	1	
1	0111	↗	1000	1	
1	1000	↗	1001	1	
1	1001	↗	0000	0	
0	-	-	Qint _n	clk_5s _n	

Code :

```
entity freqgen_4b_E is
  Port ( CE : in  STD_LOGIC;
        clk : in  STD_LOGIC;
        clk_5s : out STD_LOGIC);
end freqgen_4b_E;

architecture Behavioral of freqgen_4b_E is
  signal Qint : STD_LOGIC_VECTOR (3 downto 0) := "0000";
  signal clk_5sint : STD_LOGIC := '0';

begin

  PROCESS(clk)
  BEGIN
    IF (rising_edge (clk)) THEN
      IF(CE='1') THEN
        Qint <= Qint+1;
        IF(Qint = "0100") THEN
          clk_5sint <= '1';
        END IF;

        IF(Qint = "1001") THEN
          clk_5sint <= '0';
          Qint <= "0000";
        END IF;

      END IF;
    END IF;

  END PROCESS;

  clk_5s <= clk_5sint;

end Behavioral;
```

Simulation comportementale :

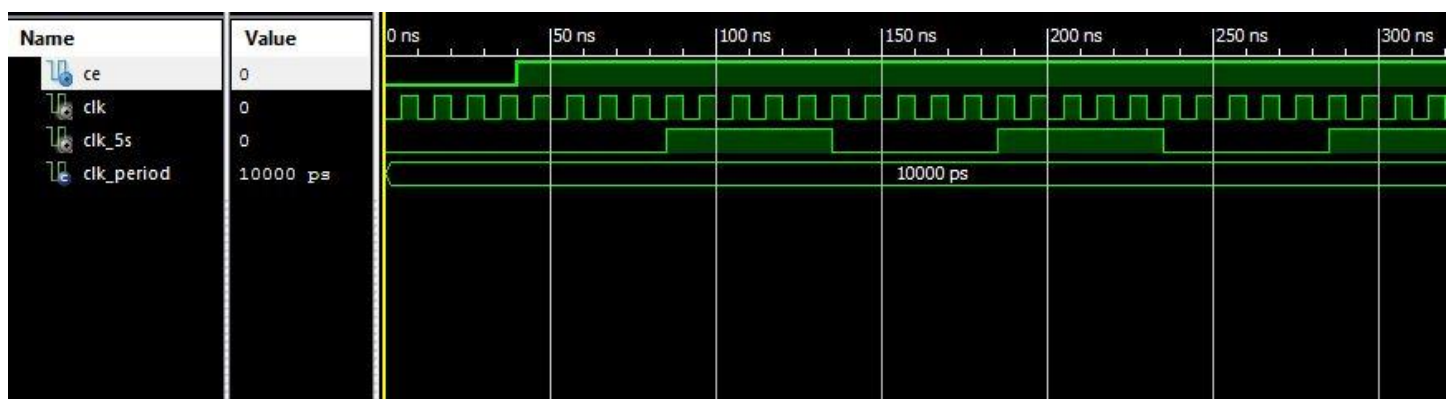


Figure 10 : Simulation comportementale de l'entité « freqgen_4b_E »

On remarque bien que notre signal clk_5s change d'état tous les 6 fronts montant de notre horloge, ce qui est le comportement attendu puisque dans le cas réel nous prendrons une horloge d'une seconde.

Simulation temporelle :

$$\tau_{propagation} = 15,156 \text{ ns}$$

$$f_{utilisationmax} = 66 \text{ MHz}$$

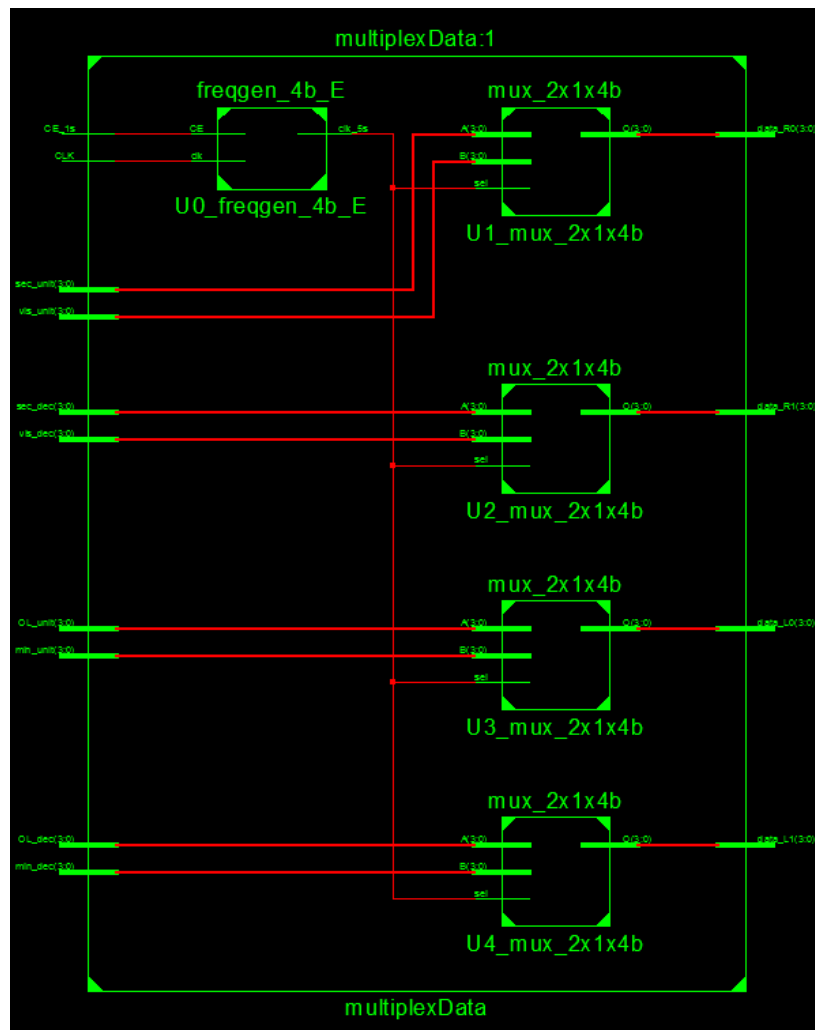
Composants :

8 LUT soit **0,2%** du nombre de LUT total disponible utilisé

3 buffers entrée/sortie soit **1,7%** du nombre de buffer total disponible utilisé

5 bascules D utilisées

c) Multiplexdata



	LUT	Slice	Buffers	% Buffers	Bascules	Temps de propagation (ns)	Fréquence maximum d'utilisation (MHz)
mux_2x1x4b	4	2	13	7,5	0	8,396	119,1
freqgen_4b_E	7	4	3	1,7	5	15,156	66
TOTAL multiplexdata	11	6	16	9,2	5		

En réalité, nous observons que l'entité multiplexdata a besoin de 23 LUT, 5 bascules et 12 Slices. Par ailleurs, une fois testé, nous nous sommes aperçus que nous avions sur les afficheurs une fois les minutes et le score d'une équipe puis les secondes et le score de la deuxième équipe, ce qui ne correspondait pas au cahier des charges. Pour résoudre ce problème nous avons juste inversé les entrées A et B des multiplexeurs 3 et 4 pour avoir le score des équipes puis le temps.

IV. Entité « chronometer »

Cette entité a pour rôle de générer des valeurs binaires en base 10 qui s'incrémentent toutes les minutes ou toutes les secondes pour atteindre la valeur limite de 45 minutes 00 seconde.

a) OR

Il ne sera pas nécessaire de créer un composant pour cette fonctionnalité, on pourra l'implémenter directement dans l'entité « chronometer ».

On écrit alors :

```
--MODELISATION DU COMPOSANT OR
--U0
U0_OR : or_out <= start_reg OR START;
```

b) Register 1b R

L'entité à concevoir est telle que :

register_1b_R			
R	D _n	clk	Q _{n+1}
1	.	↗	0
0	D _n	↗	D _n
-	-	-	Q _n

Code :

```
entity register_1b_R is
    Port ( D : in  STD_LOGIC;
          R : in  STD_LOGIC;
          clk : in  STD_LOGIC;
          Q : out STD_LOGIC);
end register_1b_R;

architecture Behavioral of register_1b_R is

    --Signaux internes
    signal Q_int : STD_LOGIC:='0';

begin
    PROCESS(clk)
    BEGIN
        IF(rising_edge(clk)) THEN
            IF(R='1') THEN
                Q_int <= '0';

                ELSIF(R='0') THEN
                    Q_int <= D;
                END IF;
            END IF;

        END IF;

    END PROCESS;
    Q <= Q_int;

end Behavioral;
```

Simulation comportementale :



Figure 5 : Simulation comportementale de l'entité « register_1b_R »

Simulation temporelle :

$\tau_{propagation} =$

$$\tau_{propagation} = 8,260ns$$

$$f_{utilisationmax} = 109,4 \text{ MHz}$$

Composants :

1 bascule D utilisée

4 buffers entrée/sortie soit **2%** du nombre de buffer total disponible utilisé

c) counterSen 4b RE

Il s'agit d'un compteur sénaire qui s'incrémente à chaque débordement de compteur décimal (CE), représentant les dizaines de secondes et dizaines de minutes avec un possible reset. L'entité à concevoir est telle que :

counterSen_4b_RE								
R	CE	clk	Qint _n [3:0]		Qint _{n+1} [3:0]		Q _{n+1} [3:0]	TC
1	-	-	-		0000		0000	0
0	1	↗	0000		0001		0001	0
0	1	↗	0001		0010		0010	0
0	1	↗	0010		0011		0011	0
0	1	↗	0011		0100		0100	0
0	1	↗	0100		0101		0101	0
0	1	↗	0101		0000		0000	1
0	0	↗	0000		0000		0000	0
0	0	-	Qint _n		Qint _n		Qint _n	0

Code :

```
--SIGNAUX INTERNES
signal Q_int: STD_LOGIC_VECTOR (3 downto 0) := "0000";
signal TC_int : STD_LOGIC := '0';

begin

process(clk, R, CE)
begin
    IF (R='1') THEN          --SI RESET ALORS ON REMET A 0
        Q_int <= "0000";
        TC_int <= '0';

    ELSIF (R='0' AND rising_edge(clk) AND CE='1') THEN      --SI PAS DE RESET + FRONT + COMPTEUR PRECEDENT A 9
        IF (Q_int = "0101") THEN      --ON REMET A 0 LE COMPTEUR ET ON MET TC A 1
            Q_int <= "0000";

        ELSE      --SINON
            Q_int <= Q_int + 1;      --ON INCREMENTE
        END IF;
    END IF;

    IF ( Q_int ="0101" AND CE='1') THEN      --SI Q = 5 ET QUE CE=1 DONC LE COMPTEUR PRECEDENT =9 ALORS TC=1
        TC_int <= '1';
    ELSE
        TC_int <= '0';
    END IF;

end process;

Q <= Q_int;
TC <= TC_int;

end Behavioral;
```

Simulation comportementale :

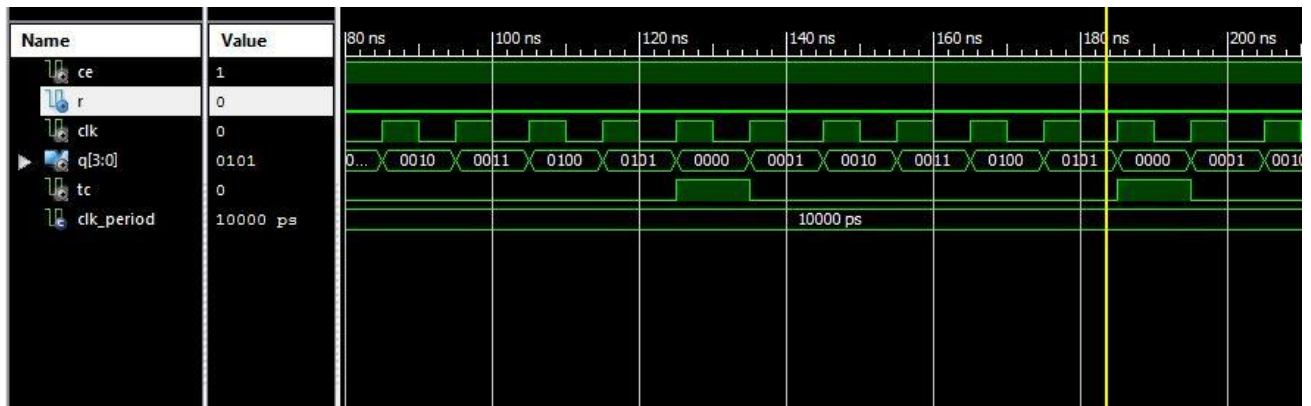


Figure 6 : Simulation comportementale de l'entité « counterSen4b_RE »

Simulation temporelle :

$\tau_{propagation} =$

$$\tau_{propagation} = 6,342 \text{ ns}$$

$$f_{utilisationmax} = 157,7 \text{ MHz}$$

Composants :

4 bascules D utilisées

5 LUT soit **1%** du nombre de LUT totales disponibles utilisées

8 buffers

d) counterDec 4b RE

L'entité à concevoir est telle que :

counterDec_4b_RE						
R	CE	clk	Qint _a [3:0]	Qint _{a+1} [3:0]	Q _{n+1} [3:0]	TC
1	-	-	-	0000	0000	0
0	1	↗	0000	0001	0001	0
0	1	↗	0001	0010	0010	0
0	1	↗	0010	0011	0011	0
0	1	↗	0011	0100	0100	0
0	1	↗	0100	0101	0101	0
0	1	↗	0101	0110	0110	0
0	1	↗	0110	0111	0111	0
0	1	↗	0111	1000	1000	0
0	1	↗	1000	1001	1001	0
0	1	↗	1001	0000	0000	1
0	0	↗	0000	0000	0000	0
0	0	-	Qint _a	Qint _a	Qint _a	0

Code :

```
--SIGNAUX INTERNES
signal Q_int: STD_LOGIC_VECTOR (3 downto 0) := "0000";
signal TC_int : STD_LOGIC := '0';

begin

process(clk, R, CE)
begin
    IF (R='1') THEN
        Q_int <= "0000";
        TC_int <= '0';

    ELSIF (R='0' AND rising_edge(clk) AND CE='1') THEN

        IF (Q_int = "1001") THEN --ON REMET A 0 LE COMPTEUR ET ON MET TC A 1 QUAND Q_INT = 1001
            Q_int <= "0000";

        ELSE
            Q_int <= Q_int + 1; --SINON ON INCREMENTE
        END IF;

    END IF;

    IF( Q_int = "1001" AND CE='1') THEN
        TC_int <= '1';
    ELSE
        TC_int <= '0';
    END IF;

end process;

Q <= Q_int;
TC <= TC_int;

end Behavioral;
```

Simulation comportementale :



Figure 7 : Simulation comportementale de l'entité « counterDec4b_RE »

Simulation temporelle :

$\tau_{propagation} =$

$$\tau_{propagation} = 1.238 \text{ ns}$$

Composants :

4 bascules D utilisées

5 LUT soit **1%** du nombre de LUT totales utilisées.

8 buffers entrée/sortie soit **4%** du nombre de buffer total disponible utilisé

e) equ45min

L'entité à concevoir est telle que :

equ45min			
min_dec	min_unit		equ
0000	-		0
0001	-		0
0010	-		0
0011	-		0
0100	0000		0
0100	0001		0
0100	0010		0
0100	0011		0
0100	0100		0
0100	0101		1

Il ne sera pas nécessaire de créer un composant pour cette fonctionnalité, on pourra l'implémenter directement dans l'entité « chronometer ». On écrit alors :

```
U7_equ45min : equ45 <= '1' when (min_unit_int ="0101" and min_dec_int ="0100") else '0';
```

Avec « min_unit_int » et « min_dec_int » des signaux internes.

f) AND4n2

L'entité à concevoir est telle que :

AND4n2				
A	B	C	D	O
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

On écrit alors :

```
--MODELISATION DU COMPOSANT AND4N2
--U2
U2_AND4N2 : CE_time45min <= ((not WAIT_t) and start_reg and CE_1s and (not equ45));
```

g) Chronometer

Une fois les sous-entités codées et simulées, nous pouvons réaliser une simulation globale de l'entité « chronometer ».

On obtient alors le comportement suivant :

Figure 8 :
Comptage des unités secondes,
incréméntation des dizaines secondes

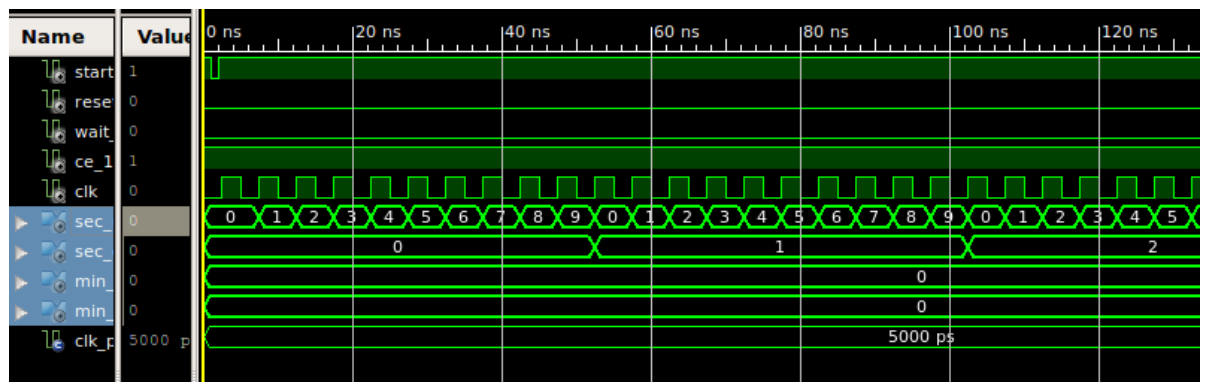


Figure 9 :
Comptage des dizaines secondes,
incréméntation des unités minute

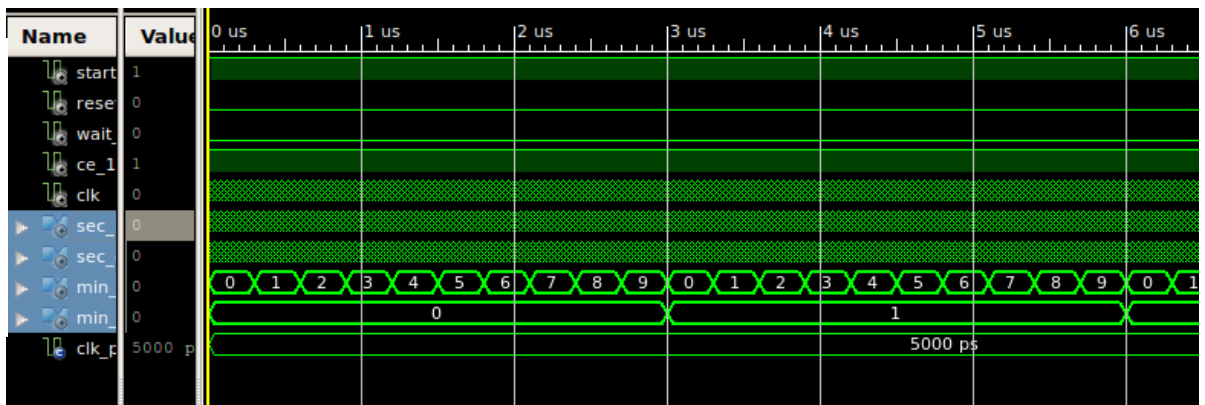


Figure 10 :
Comptage des unités minute,
incréméntation des dizaines minute.
Blocage à 45 min
0 sec

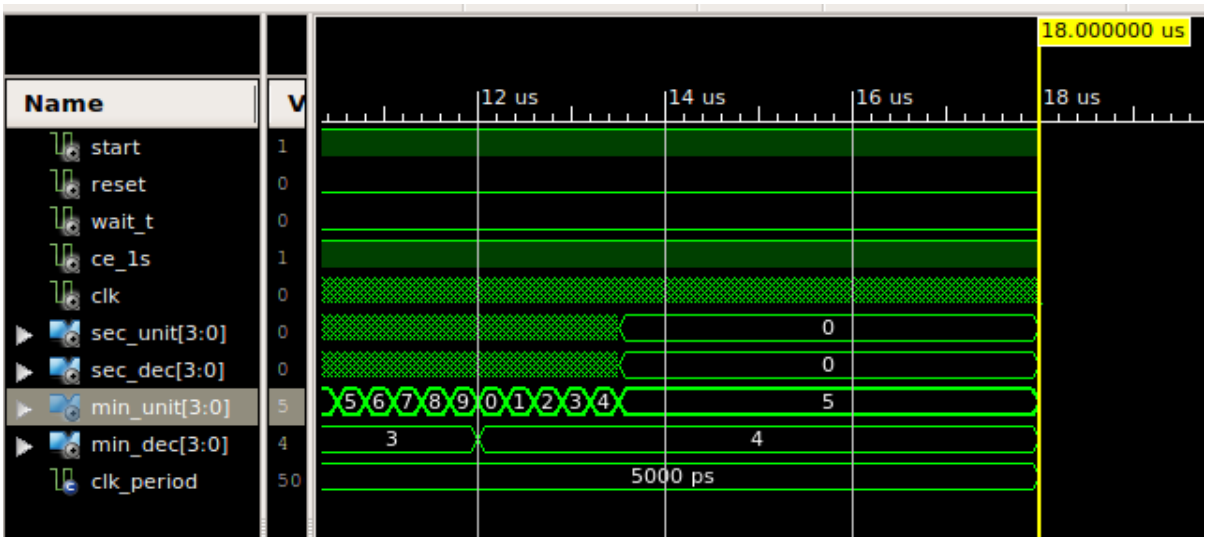
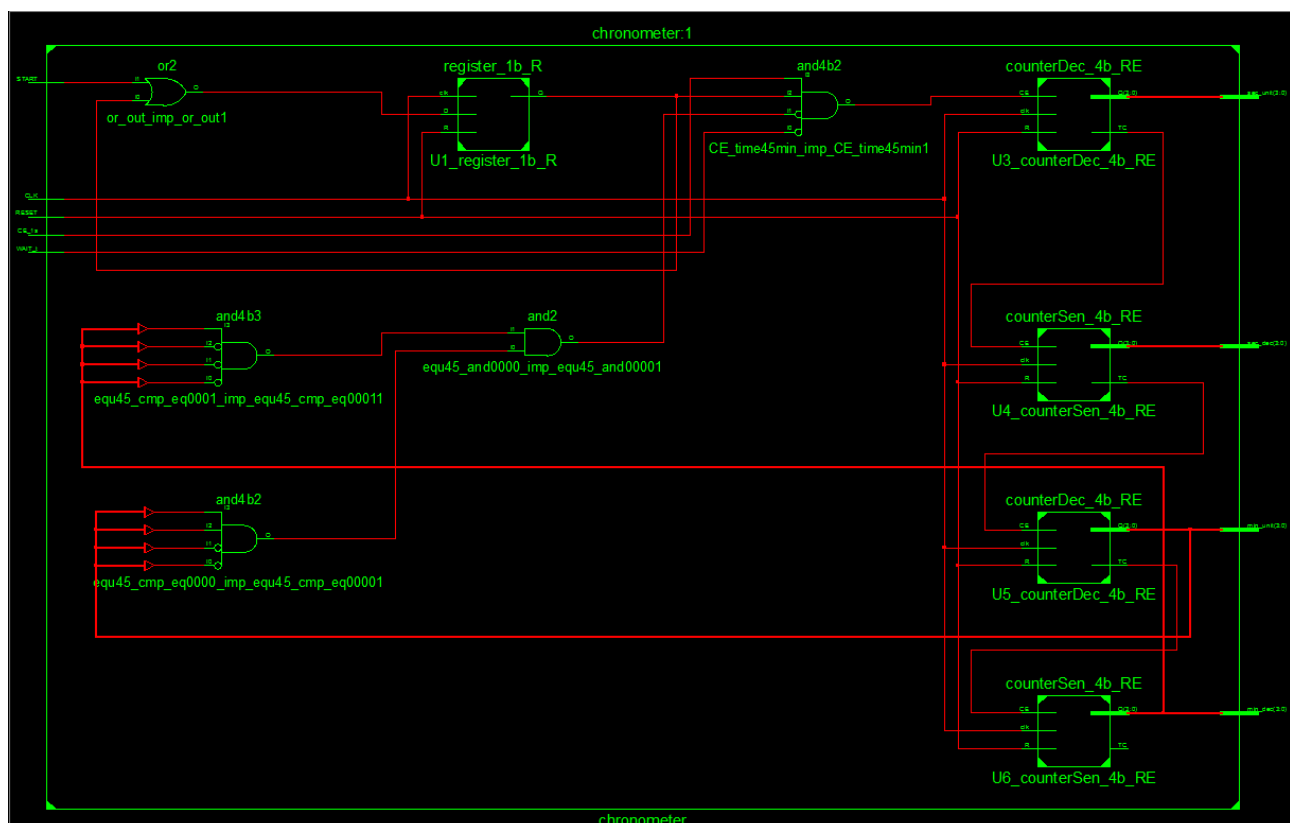


Figure 11 : Simulation comportementale de l'entité « chronometer »

Voici un tableau récapitulatif sur l'entité « chronometer »

	LUT	Slice	Buffers	% Buffers	Bascules	Temps de propagation (ns)	Fréquence maximum d'utilisation
OR + register_1b_R	0	0	4	2	1		
counterSen_4b_RE	5	3	8	5	4	6.342	
counterDec_4b_RE	5	3	8	5	4	1.238	
TOTAL chronometer	10	6	25	6,9	12		

Par ailleurs, pour concevoir cet élément, il utilise 24 LUT, 17 bascules et 13 Slices. Voici la vue RTL de notre entité :



V. Entité « score »

a) Register 1b E

Cet élément est indispensable pour réaliser l'anti-rebond lié à l'appui sur les 2 différents boutons. L'entité à concevoir est telle que :

register_1b_E				
CE	D _n	clk		Q _{n+1}
1	D _n	↗		D _n
1	D _n	-		Q _n
0	-	-		Q _n

Code :

```
entity register_1b_E is
    Port ( CE : in  STD_LOGIC;
          D : in  STD_LOGIC;
          clk : in  STD_LOGIC;
          Q : out STD_LOGIC);
end register_1b_E;

architecture Behavioral of register_1b_E is
    --Signaux internes
    signal Q_int : STD_LOGIC:='0';
begin
    PROCESS(clk)
    BEGIN
        IF(rising_edge(clk)) THEN
            IF(CE='1') THEN
                Q_int <= D;
            END IF;
        END IF;
    END PROCESS;
    Q <= Q_int;
end Behavioral;
```

Simulation comportementale :

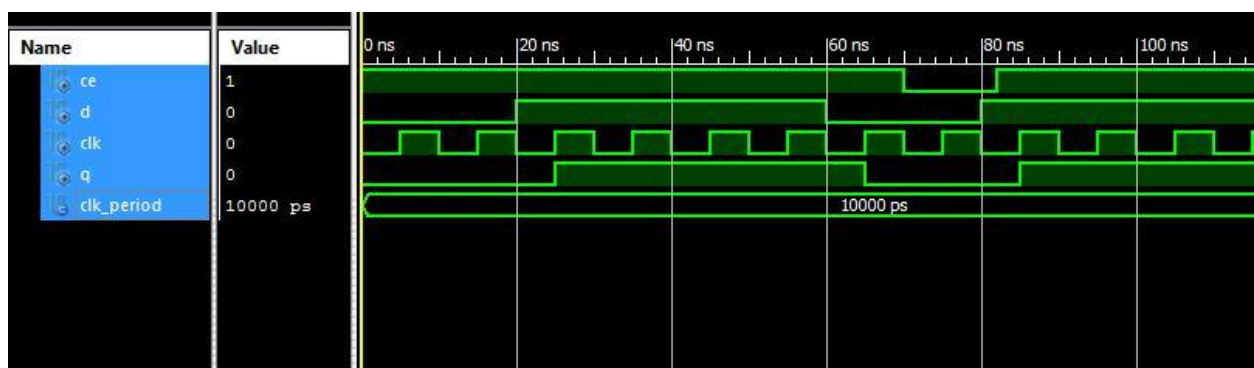


Figure 12 : Simulation comportementale de l'entité « register_1b_E »

Simulation temporelle :

$\tau_{propagation} =$

$$\tau_{propagation} = 7.165 \text{ ns}$$

$$f_{utilisationmax} = 140 \text{ MHz}$$

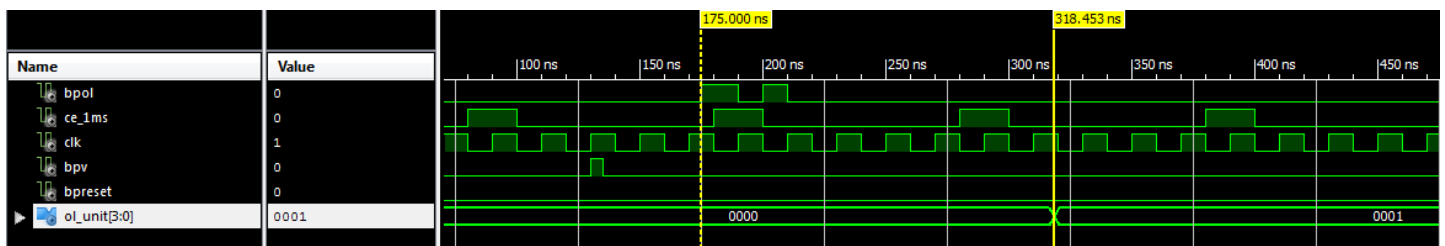
Composants :

1 bascule D utilisée

4 buffers entrée/sortie soit **2%** du nombre de buffer total disponible utilisé

0 LUT

Nous allons maintenant voir si cet élément joue bien le rôle d'anti-rebond qu'on recherche :



Ainsi, on observe que lorsque bpol par exemple passe deux fois à un très rapidement comme c'est le cas dans l'exemple au-dessus (pour illustrer le cas d'un rebond en relâchant le bouton), notre sortie ol_unit passe seulement à un et non pas à deux. Ici est tout le monde d'un système anti-rebond.

b) Register 1b

L'entité à concevoir est telle que :

register_1b_R		
D_n	clk	Q_{n+1}
D_n	\nearrow	D_n
-	-	Q_n

Code :

```
entity register_1b is
    Port ( D : in  STD_LOGIC;
          clk : in  STD_LOGIC;
          Q : out  STD_LOGIC);
end register_1b;

architecture Behavioral of register_1b is

    --Signaux internes
    signal Q_int : STD_LOGIC:='0';

begin
    PROCESS(clk)
    BEGIN
        IF(rising_edge(clk)) THEN
            Q_int <= D;
        END IF;

    END PROCESS;
    Q <= Q_int;
end Behavioral;
```

Simulation comportementale :

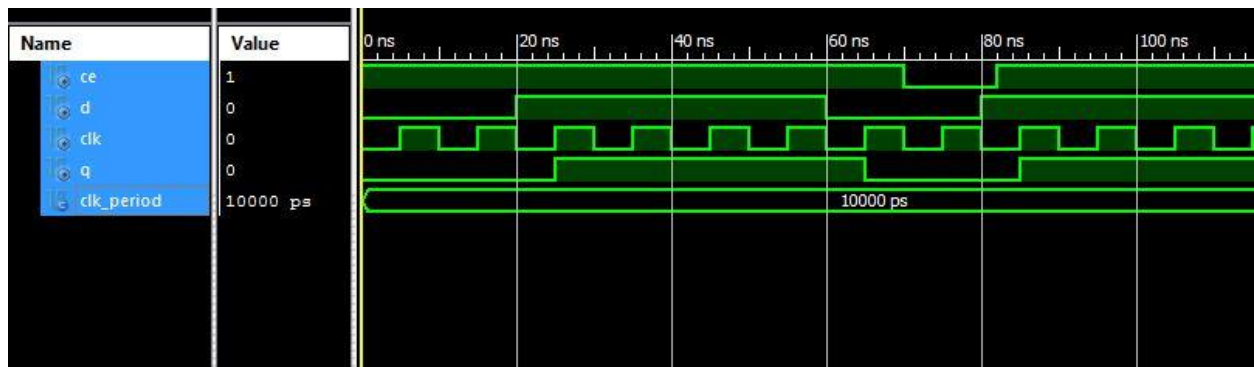


Figure 13 : Simulation comportementale de l'entité « register_1b »

Simulation temporelle :

$\tau_{propagation} =$

$$\tau_{propagation} = 1.521 \text{ ns}$$

Composants :

1 bascule D utilisée

3 buffers entrée/sortie soit **2%** du nombre de buffer total disponible utilisé

Pas d'utilisation de LUT

Lorsqu'on regarde les temps de propagation dans le résumé, on remarque que le temps pour une bascule est égal à 0.720ns mais on a aussi le temps pour la connectique qui est de 0.801, qui est donc non négligeable. On fait le choix de sommer les deux pour avoir notre temps de propagation.

c) XOR

Il ne sera pas nécessaire de créer un composant pour cette fonctionnalité, on pourra l'implémenter directement dans l'entité « score ». On écrit alors :

```
--MODELISATION DES COMPOSANTS XOR
--U3
bpol_fr <= bpol_f XOR bpol_d;

--U8
bpv_fr <= bpv_f XOR bpv_d;
```

d) NAND

Il ne sera pas nécessaire de créer un composant pour cette fonctionnalité, on pourra l'implémenter directement dans l'entité « score ».

On écrit alors :

```
--MODELISATION DES COMPOSANTS NAND
--U4
bpol_inc <= bpol_fr AND bpol_d;

--U9
bpv_inc <= bpv_fr AND bpv_d;
```


e) Score

Cette entité a pour rôle de générer des valeurs binaires en base 10 qui s'incrémente à chaque nouveau but d'une équipe. Une fois les sous-entités codées et simulées, nous pouvons réaliser une simulation globale de l'entité « score ». On obtient alors :

Dans cet exemple, l'OL marque un but, puis les visiteurs égalisent. Finalement l'OL marque un second but et gagne la partie.

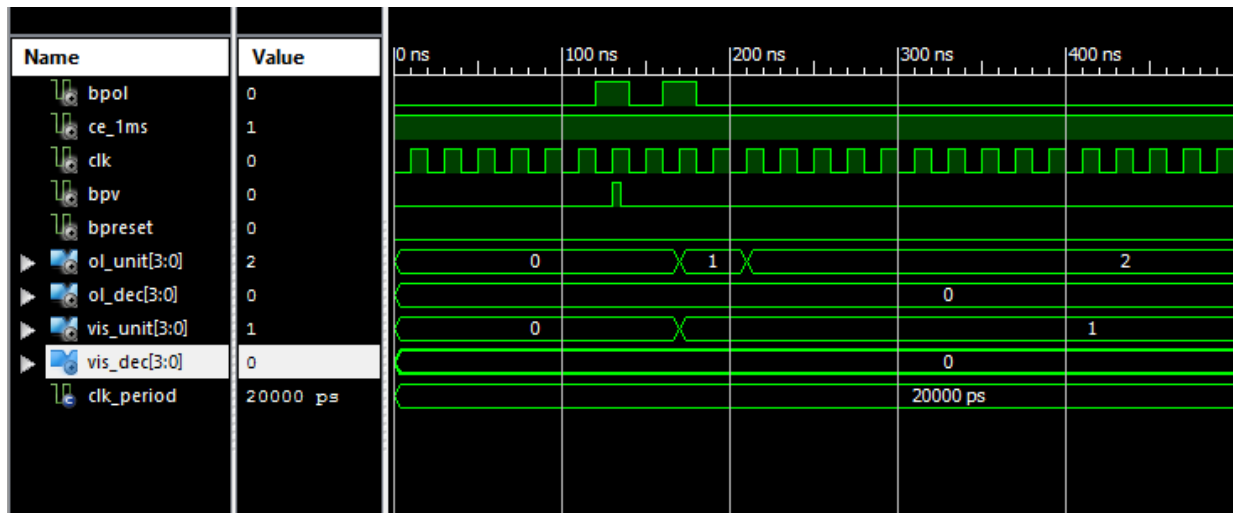


Figure 14 : Simulation comportementale de l'entité « score »

	LUT	Slice	Bascules	Temps de propagation
TOTAL score	20	16	20	7.367 ns

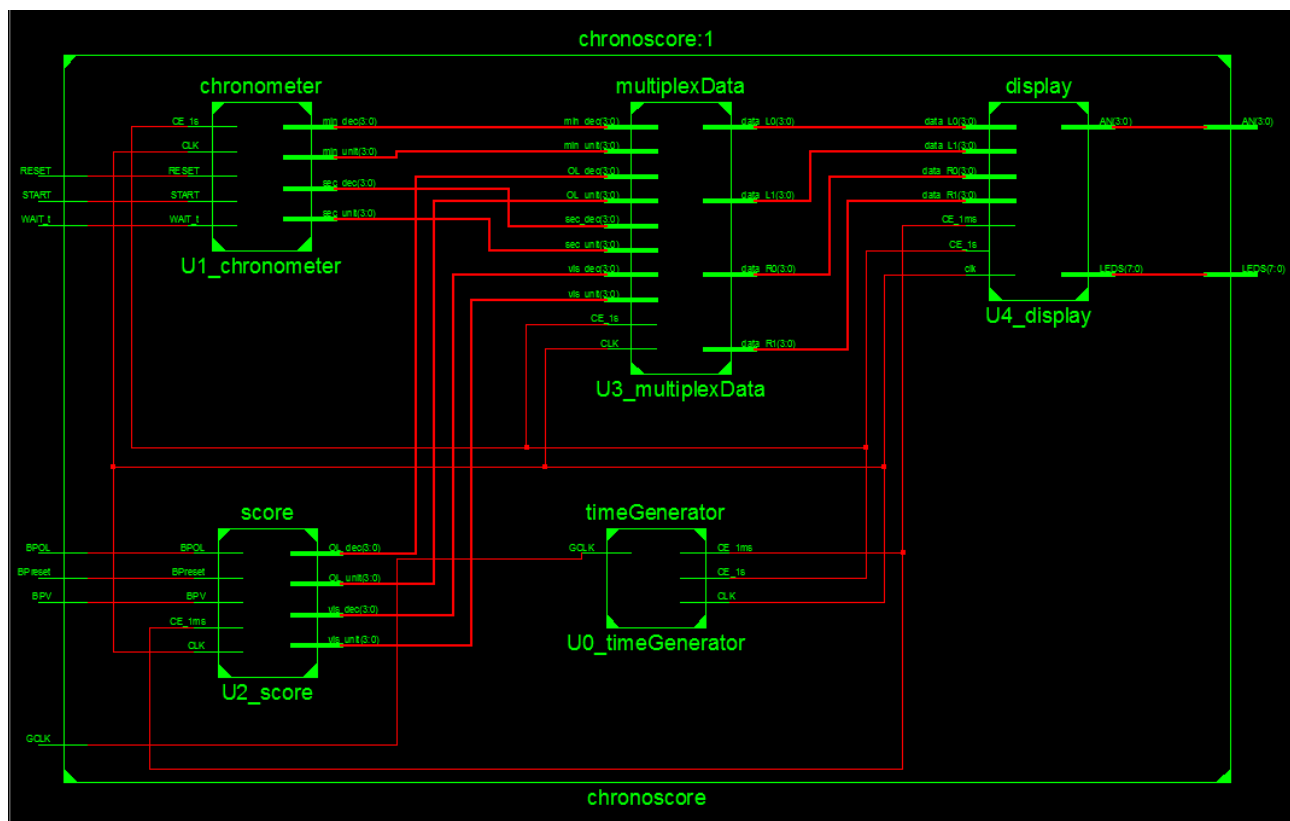
VI. Entité « chronoscore »

a) Chronoscore

Nous pouvons faire un bilan du nombre de composants totaux utilisés pour réaliser l'entité chronoscore :

	LUT	Slice	Buffers	% Buffers	Bascules	Temps de propagation
TOTAL chronoscore	138 (3%)	84 (4%)	19	10	99 (2%)	8,8 ns

L'important est de remarquer qu'on utilise vraiment très peu des ressources de notre FPGA avec tout notre projet. Voici la vue RTL de l'entité chronoscore :



VII. Bilan

a) Technique

En conclusion, l'afficheur est fonctionnel après injection du programme sur notre carte. En effet, le temps écoulé au format minutes-secondes et le score s'affichent désormais avec une alternance de 10 secondes. Le temps écoulé s'incrémente correctement jusqu'à une durée limite de 45 minutes. Nous pouvons également simuler les buts marqués pour l'Olympique Lyonnais ou pour les visiteurs à l'aide d'un bouton poussoir et pouvons observer que le score change correctement. Le cahier des charges est donc respecté et le projet peut être livré avec succès à la société « Olympique Lyonnais ».

b) Personnel

Au cours de ces 3 séances de projet et notre investissement hors-séance, nous avons dû nous organiser le mieux possible, compte-tenu du grand nombre d'entités qui composent ce projet. Cette organisation s'est axée sur plusieurs phases : nous avons réalisé un codage de tous les composants dans un premier temps puis relevé tous les temps de propagations. Nous avons ensuite fait toutes les captures d'écran nécessaires au compte-rendu, puis avons fait un effort d'interprétation de nos résultats. Dans une moindre échelle, nos analyses nous ont forcés à adopter une démarche d'ingénieur dans un des domaines de notre filière qui nous intéresse le plus : l'électronique numérique. Le bilan est donc très positif, c'est pourquoi nous avons aimé ce projet.