Projet de Système de Loto Numérique

EN103 - Projet numérique VHDL et FPGA

21 avril 2025

Auteurs :

Mattéo BINET

Alexandre BROMET

Sommaire

Introduction **1**

1 Cahier des Charges et Matériel Utilisé **1**
1.1 Comportement Fonctionnel du Système 1
1.2 Contraintes de Cadencement 1
1.3 Interface d'Entrées/Sorties 1
1.4 Matériel Utilisé 2

2 Modules de l'Architecture **2**
2.1 Gestion des Signaux d'Activation — Gestion Horloge 2
2.2 Fonction LOTO - Top Level Tirage 3
2.3 Transcodage pour Afficheur Sept Segments - Transcodeur 4
2.4 Sélection du Nombre à Afficher - Mod6 5
2.5 Sélection de l'Anode — Mod4 6
2.6 Module d'Affichage 7 Segments - Mux6 6
2.7 Module d'Affichage 7 Segments - Mux4 7

3 Module Tirage **9**
3.1 Compteur 1 à 49 - cpt_tirage 9
3.2 Compteur 1 à 6 - cpt_valeurs 10
3.3 Sauvegarde des 6 Valeurs - mem 10
3.4 Comparaison des 6 Valeurs - comp 11
3.5 Contrôle du Processus du LOTO - FSM 12

4 Top Level **13**

Conclusion **13**

Annexe **15**

Introduction

Le projet LOTO, réalisé dans le cadre du cours EN103 : Projet numérique, consiste en la conception et le prototypage d'un système de tirage aléatoire de 6 nombres distincts compris entre 1 et 49. Ce projet expérimental a pour objectif de nous familiariser avec la conception d'architectures numériques en utilisant le langage VHDL sur Vivado et nous initier au prototypage sur circuit FPGA.

Le système est implémenté sur une carte NEXYS4 de Digilent, équipée d'un FPGA Xilinx Artix-7. Il comprend plusieurs modules fonctionnels, notamment la gestion des signaux d'horloge, l'affichage sur des afficheurs 7 segments, et le mécanisme de tirage aléatoire. Chaque module a été conçu, simulé et validé dans l'environnement Vivado avant d'être intégré et testé sur la carte FPGA.

1 Cahier des Charges et Matériel Utilisé

1.1 Comportement Fonctionnel du Système

Le système démarre après une initialisation par le signal `RAZ (Remise à Zéros)`, qui place tous les registres à zéro et les afficheurs dans un état neutre.

À chaque appui sur le bouton de tirage, un compteur synchrone défile de 1 à 49 à haute fréquence. Lorsque le bouton est relâché, la valeur courante du compteur est figée.

- Si cette valeur n'a pas encore été tirée, elle est mémorisée.
- Si elle est déjà présente parmi les valeurs précédemment tirées, le tirage est à recommencer.

Ce processus est répété jusqu'à l'obtention de six valeurs distinctes.

1.2 Contraintes de Cadencement

La fréquence d'horloge de base est de **100 MHz**, fournie par le quartz de la carte. Trois signaux de type `Clock Enable` (CE) permettent d'adapter les cadences aux différentes parties du système :

- $CE_{traitement}$: commande le compteur de tirage — fréquence $\approx$ 25 MHz pour garantir un bon aléatoire.
- $CE_{affichage}$: cadence le rafraîchissement des afficheurs 7 segments via multiplexage — fréquence $\approx$ 3 kHz.
- $CE_{incrément}$: permet de faire défiler les positions de 1 à 6 pour la visualisation des valeurs — fréquence $\approx$ 1 Hz.

1.3 Interface d'Entrées/Sorties

- **Entrées :**
  - `horloge` : signal d'horloge principal (100 MHz),
  - `RAZ` : remise à zéro asynchrone,
  - `Bouton` : commande manuelle du tirage.
- **Sorties :**

— `Sept_Segments` : signal pour affichage sur 7 segments (7 bits),
— `AN` : sélection d'afficheur actif (4 bits).

1.4 Matériel Utilisé

- **Carte de développement :** Digilent NEXYS4 A7.
- **FPGA :** Xilinx Artix-7 (référence XC7A100T-1CSG324 ou XC7A50T-1CSG324).
- **Périphériques intégrés :**
  — 4 afficheurs 7 segments à anode commune,
  — 16 LEDs, boutons poussoirs, interrupteurs,
  — quartz d'horloge à 100 MHz.
- **Référence :** `https://digilent.com/reference/programmable-logic/nexys-a7/reference-manual`

2 Modules de l'Architecture

2.1 Gestion des Signaux d'Activation — Gestion Horloge

Ce module a pour rôle de dériver, à partir de l'horloge principale du FPGA (100 MHz), trois signaux d'activation (`Clock Enable`) destinés à cadencer les différents sous-systèmes du projet. Il est fondamental pour adapter la vitesse d'exécution des blocs au besoin réel de traitement ou d'affichage.

Chacun des signaux est généré à l'aide d'un compteur associé à un seuil spécifique. Lorsqu'un compteur atteint son seuil, un front actif du signal CE correspondant est produit pendant une demi-période d'horloge, puis le compteur est remis à zéro.

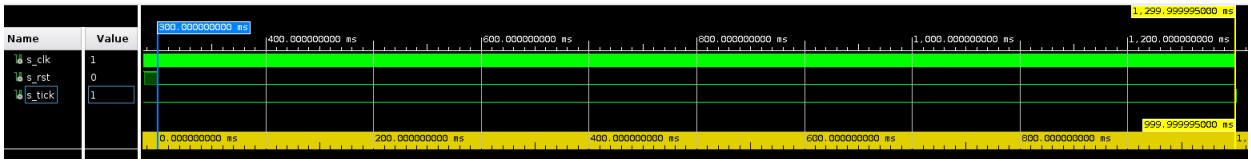

FIGURE 1 – Chronogrammes : CE 1Hz

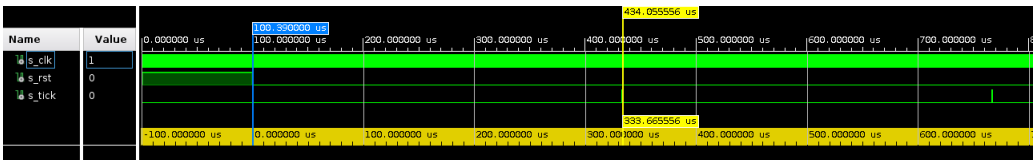

FIGURE 2 – Chronogrammes : CE 3kHz

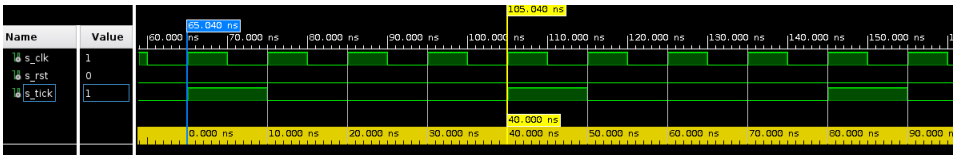

FIGURE 3 – Chronogrammes : CE 25MHz

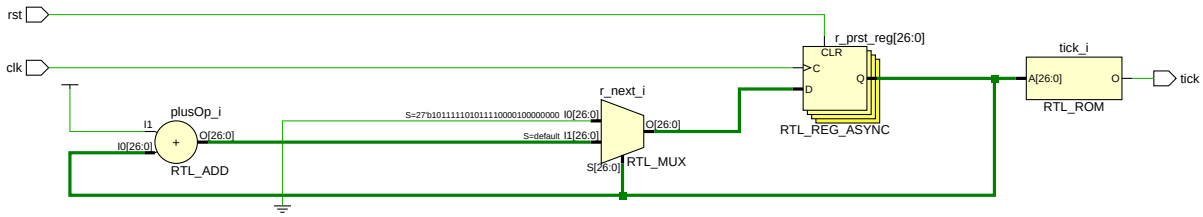

FIGURE 4 – Schématique : CE 1Hz

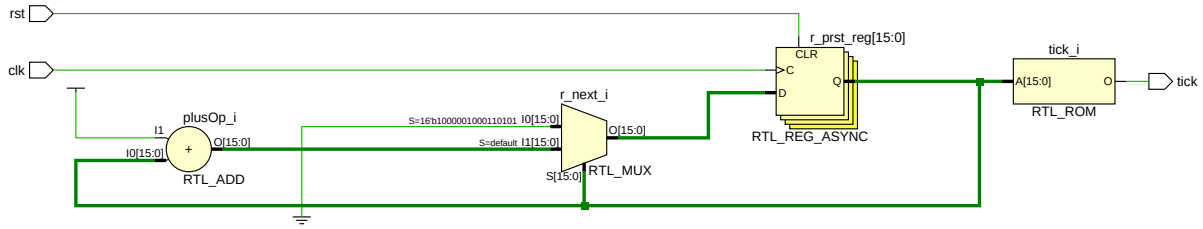

FIGURE 5 – Schématique : CE 3kHz

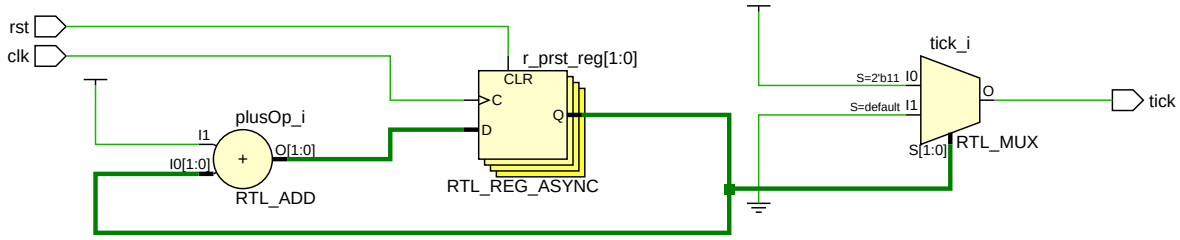

FIGURE 6 – Schématique : CE 25MHz

2.2 Fonction LOTO - Top Level Tirage

Le module `top_lvl_tirage` constitue le cœur du système de tirage du LOTO. C'est l'entité principale qui relie entre eux les différents modules fonctionnels responsables du comptage, de la validation, de la mémorisation et du contrôle du tirage des six valeurs distinctes.

Il instancie et connecte cinq composants principaux :

· Un compteur rapide pour défiler les valeurs de 1 à 49,

· Un compteur de rang pour suivre la progression des tirages (1 à 6),
· Un comparateur pour éviter les doublons,
· Une mémoire pour stocker les 6 valeurs validées,
· Une FSM (machine à états) pour gérer la logique du tirage.

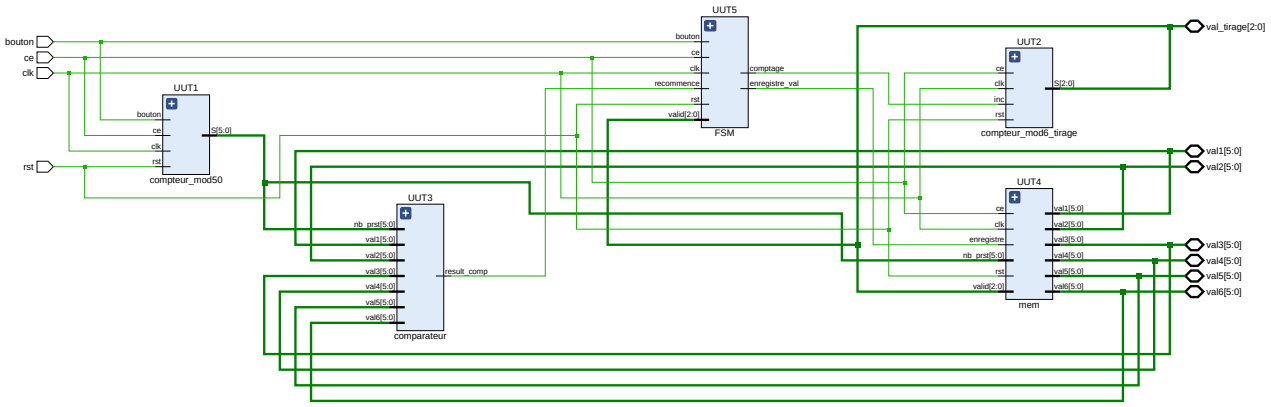

FIGURE 7 – Schématique : Loto

2.3 Transcodage pour Afficheur Sept Segments - Transcodeur

Le module `chiffre_7segments` a pour fonction de convertir une valeur décimale codée sur 4 bits (donc de 0 à 9) en une sortie compatible avec un afficheur 7 segments. Ce transcodage permet d'afficher chaque chiffre des nombres tirés (unités ou dizaines) sur les afficheurs de la carte NEXYS4.

Le codage des segments est réalisé en logique négative, comme l'exige le matériel utilisé : une valeur basse ('0') active un segment, tandis qu'une valeur haute ('1') l'éteint. Par exemple, pour afficher le chiffre '2', la sortie est '"0100100"' ce qui allume les segments 'a', 'b', 'd', 'e' et 'g'.

Le transcodeur est utilisé deux fois dans le système : une fois pour le chiffre des unités, et une autre pour le chiffre des dizaines. Cela permet d'afficher correctement tous les nombres entre 1 et 49 sur deux digits.

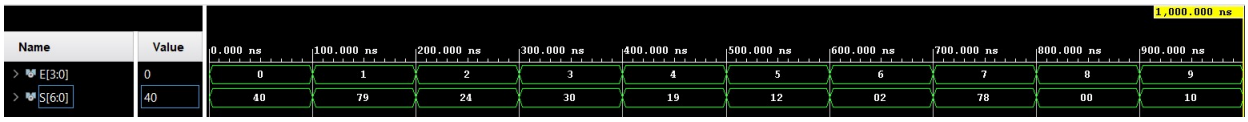

FIGURE 8 – Chronogrammes : Transcodeur

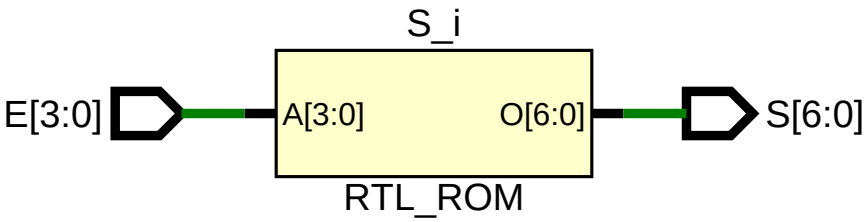

FIGURE 9 – Schématique : Transcodeur

2.4 Sélection du Nombre à Afficher - Mod6

Le module `compteur_mod6` est un compteur cyclique de 0 à 5. Il permet de sélectionner, de manière périodique, l'une des six valeurs tirées au sort à afficher sur les afficheurs 7 segments.

Chaque valeur du LOTO (entre 1 et 49) est stockée dans un registre. Ce module génère un index cyclique (modulo 6), qui permet à un multiplexeur (comme `mux6`) de choisir quelle valeur doit être aiguillée vers le transcodeur et ensuite affichée.

Le comptage est cadencé par un signal d'activation `CE`, généré à basse fréquence (1 Hz) pour permettre une rotation lisible à l'œil humain des six valeurs enregistrées.

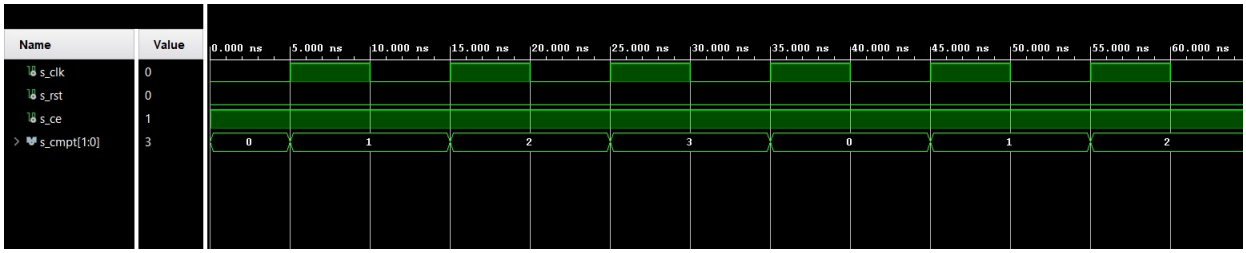

FIGURE 10 – Chronogrammes : Mod6

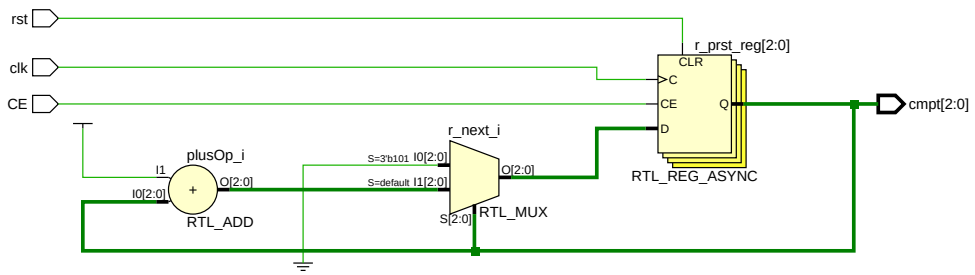

FIGURE 11 – Schématique : Mod6

2.5 Sélection de l'Anode — Mod4

Le module `mod4` est un compteur modulo 4 qui permet de sélectionner cycliquement l'une des quatre anodes des afficheurs 7 segments de la carte. Étant donné que ces afficheurs partagent les mêmes segments, ils doivent être activés l'un après l'autre à haute fréquence afin de tirer parti de la persistance rétinienne.

Ce compteur s'incrémente à chaque front actif du signal $CE_{affichage}$, généré par le module de gestion d'horloge. À tout instant, une seule des quatre sorties est active (logique négative), ce qui permet d'alimenter un seul afficheur à la fois.

Le signal de sortie `AN` est codé sur 4 bits, chaque bit correspondant à une anode. Le comptage se fait sur un cycle de 4 états (de 0 à 3), ce qui permet de piloter les afficheurs de manière fluide et répétitive.

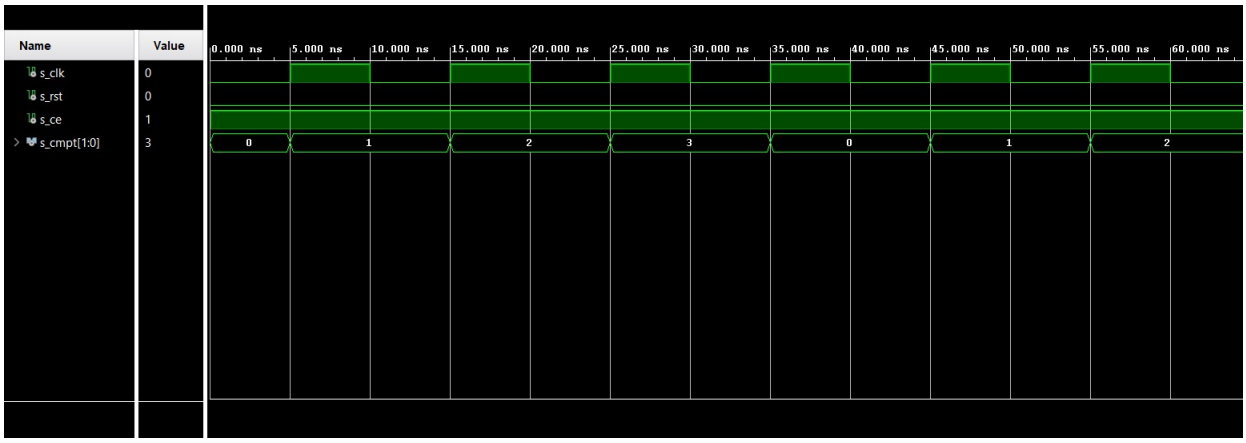

FIGURE 12 – Chronogrammes : Mod4

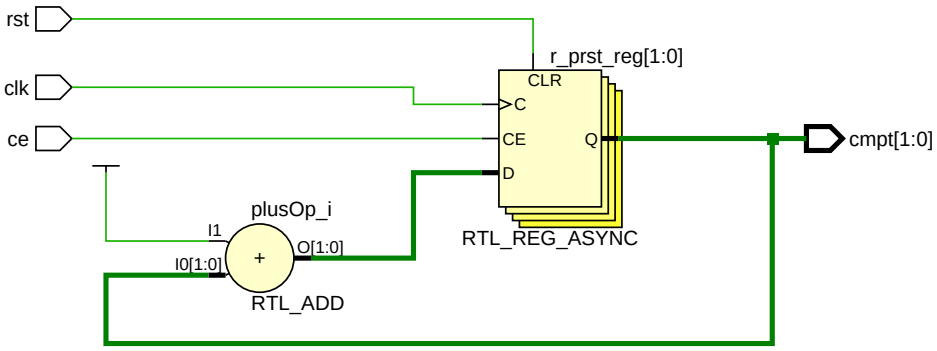

FIGURE 13 – Schématique : Mod4

2.6 Module d'Affichage 7 Segments - Mux6

Le module `mux6` est un multiplexeur à 6 entrées permettant de sélectionner l'une des six valeurs tirées par le système de loto. Chacune de ces valeurs est codée sur 6 bits (de 1 à 49), et représente un nombre unique enregistré après validation.

Ce multiplexeur est contrôlé par un signal de sélection `cmd` sur 3 bits, qui correspond à l'index du registre à lire (de 0 à 5). Le module fournit en sortie la valeur sélectionnée sur le port `S`, qui sera ensuite transcodée pour affichage.

En résumé, `mux6` permet d'aiguiller cycliquement l'une des six valeurs vers le module de transcodage en fonction du compteur de sélection issu de `mod6`.

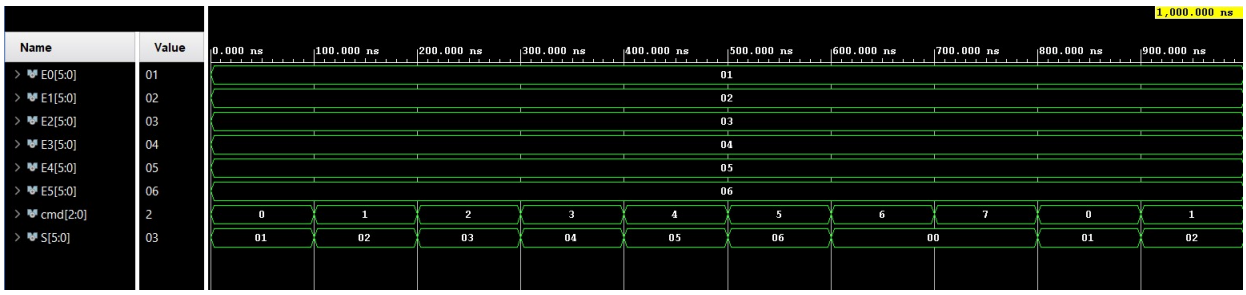

FIGURE 14 – Chronogrammes : Mux6

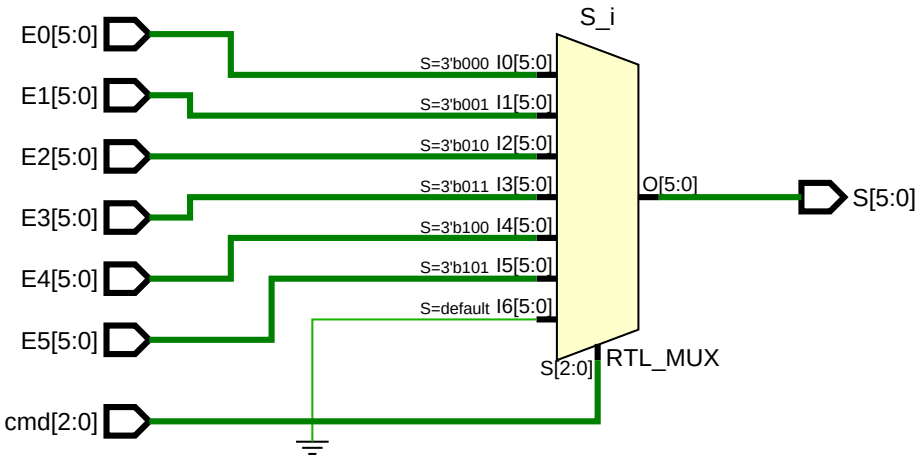

FIGURE 15 – Schématique : Mux6

2.7 Module d'Affichage 7 Segments - Mux4

Mux4 - 4bits :

Le module `mux4_4` est un multiplexeur à 4 entrées de 4 bits. Il permet de sélectionner dynamiquement l'un des quatre chiffres à afficher (par exemple : les chiffres des dizaines ou des unités issus du transcodeur).

Il est utilisé dans l'affichage sur 7 segments pour choisir le chiffre (sur 4 bits) à envoyer à l'afficheur actif, en fonction d'un index (cmd sur 2 bits). Cette sélection est cyclique grâce au module `mod4`, qui indique quelle anode est activée à chaque instant.

Mux4 - 7bits :

Le module `mux4_7` fonctionne exactement comme `mux4_4`, mais les entrées sont sur 7 bits, ce qui correspond à un codage 7 segments (segments a–g). Il est utilisé pour acheminer les codes 7 segments des différents chiffres vers l'afficheur actif.

Concrètement, ce module permet de sélectionner, à chaque instant, le code 7 segments associé au chiffre qui doit être affiché sur l'une des 4 anodes. Cela fait partie de la logique de multiplexage de l'affichage cyclique sur la carte FPGA.

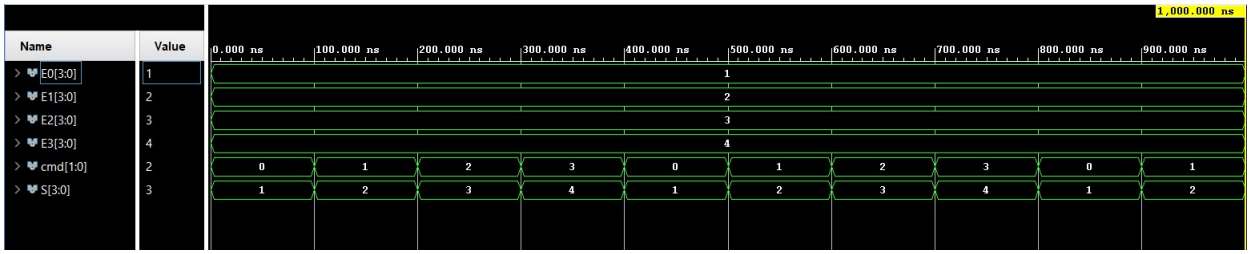

FIGURE 16 – Chronogrammes : Mux4 4 bits

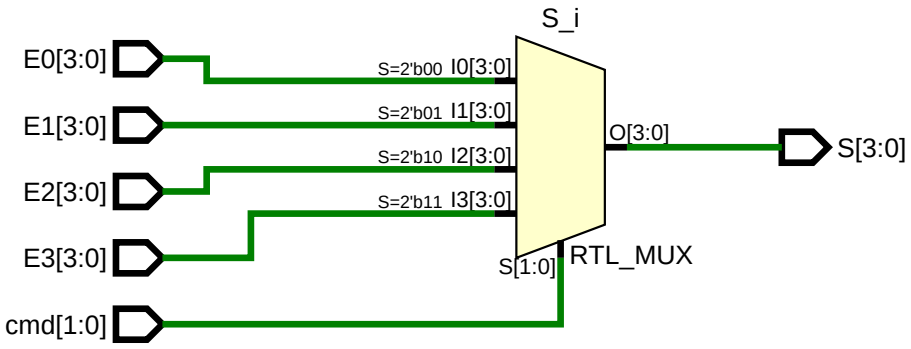

FIGURE 17 – Schématique : Mux4 4 bits

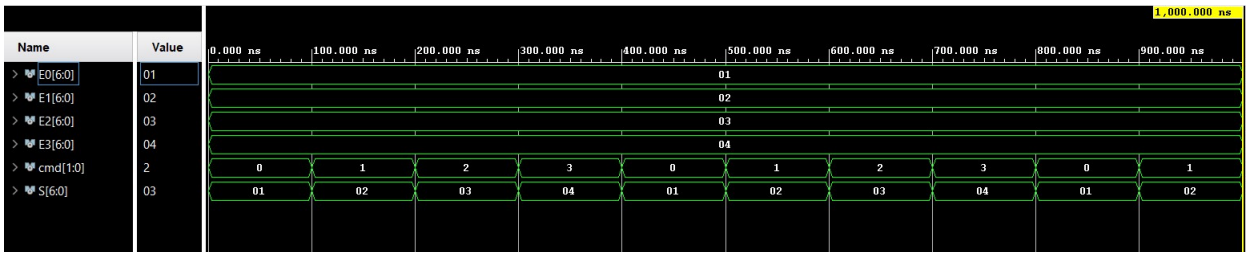

FIGURE 18 – Chronogrammes : Mux4 7 bits

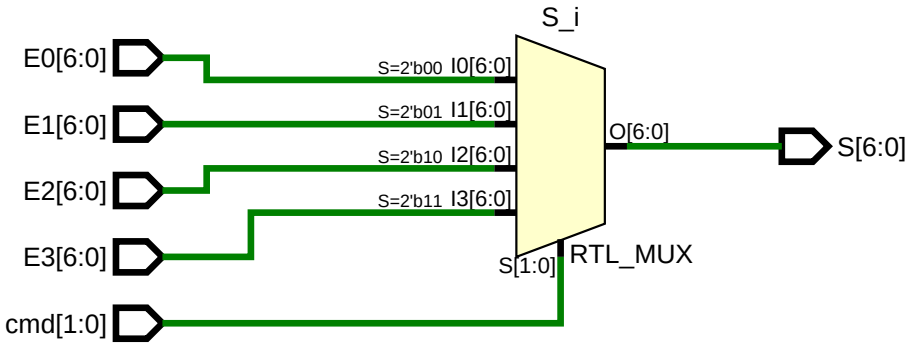

FIGURE 19 – Schématique : Mux4 7 bits

3 Module Tirage

3.1 Compteur 1 à 49 - cpt_tirage

Le module `compteur_mod50` correspond au compteur de tirage principal du système. Lorsqu'un tirage est lancé (lorsque l'utilisateur appuie sur le bouton), ce compteur défile rapidement de 1 à 49 de manière circulaire.

Ce fonctionnement donne à l'utilisateur l'illusion d'un tirage aléatoire : le compteur s'incrémente à haute fréquence, et s'arrête à la valeur atteinte lors du relâchement du bouton. C'est cette valeur qui est ensuite comparée aux précédentes et éventuellement enregistrée.

La sortie S contient en permanence la valeur courante du compteur, codée sur 6 bits pour couvrir les 49 valeurs possibles.

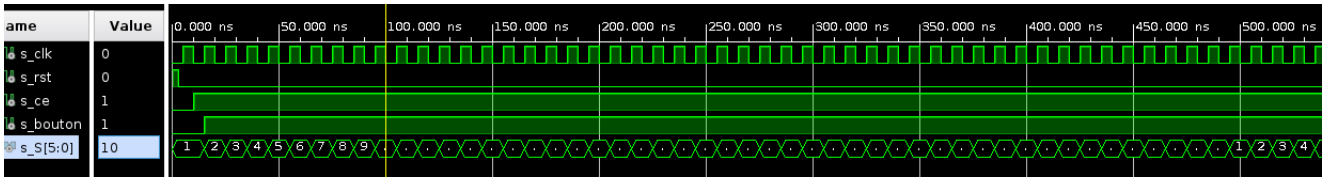

FIGURE 20 – Chronogrammes : cpt_tirage

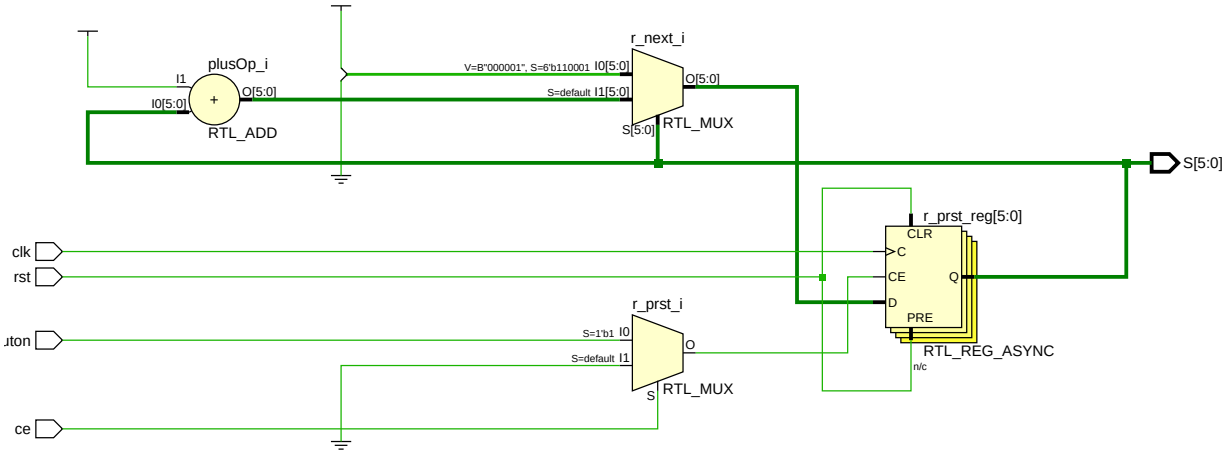

FIGURE 21 – Schématique : cpt_tirage

3.2 Compteur 1 à 6 - cpt_valeurs

Le module `compteur_mod6` correspond au compteur du numéro du nombre en cours de tirage.

Il compte de 1 jusqu'à 6 car une fois les 6 nombres tirés, le tirage s'arrête (à moins de reset le système). En plus de la clock, le clock enable et le reset, il y a l'incrément en entrée qui indique si le compteur peux compter. La variable incrément est issue de la FSM et est telle qu'à chaque fois qu'elle passe à 1, le compteur augmente seulement de 1. Ce n'est pas le cas sur le chronogramme car la variable incrément reste fixée à 1, ce qui n'est pas le cas dans le système complet.

La schématique de ce compteur est la même que celui précédent, seulement les sorties sont codées sur 3 bits pour ne compter que jusqu'à 6.

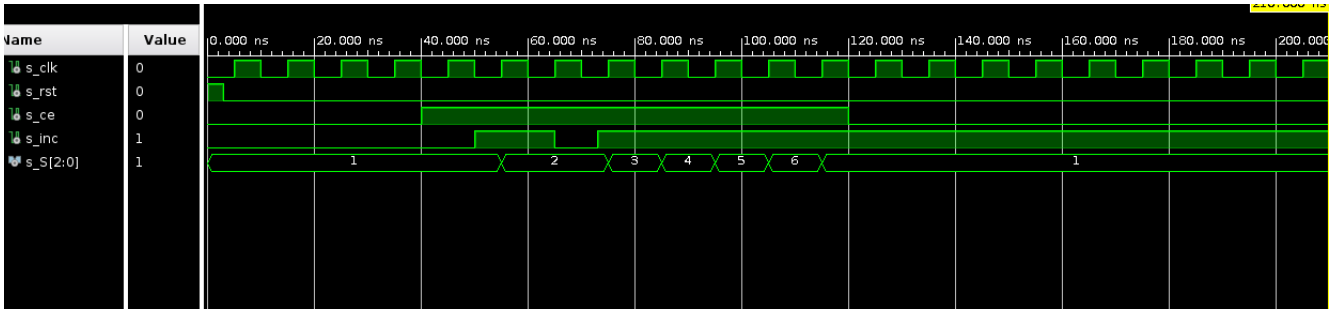

FIGURE 22 – Chronogrammes : cpt_valeurs

3.3 Sauvegarde des 6 Valeurs - mem

Le module mémoire permet d'affecter la valeur du nombre tiré à un des 6 emplacements possible en fonction du numéro de ce tirage, donné par le `compteur_mod6`.

C'est la variable enregistre issue de la FSM qui indique si le module peut enregistrer ou non.

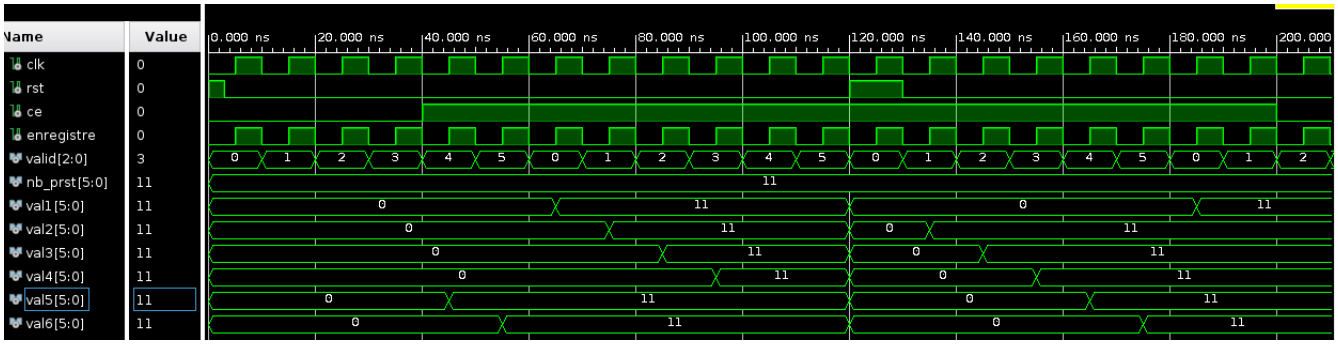

FIGURE 23 – Chronogrammes : mem

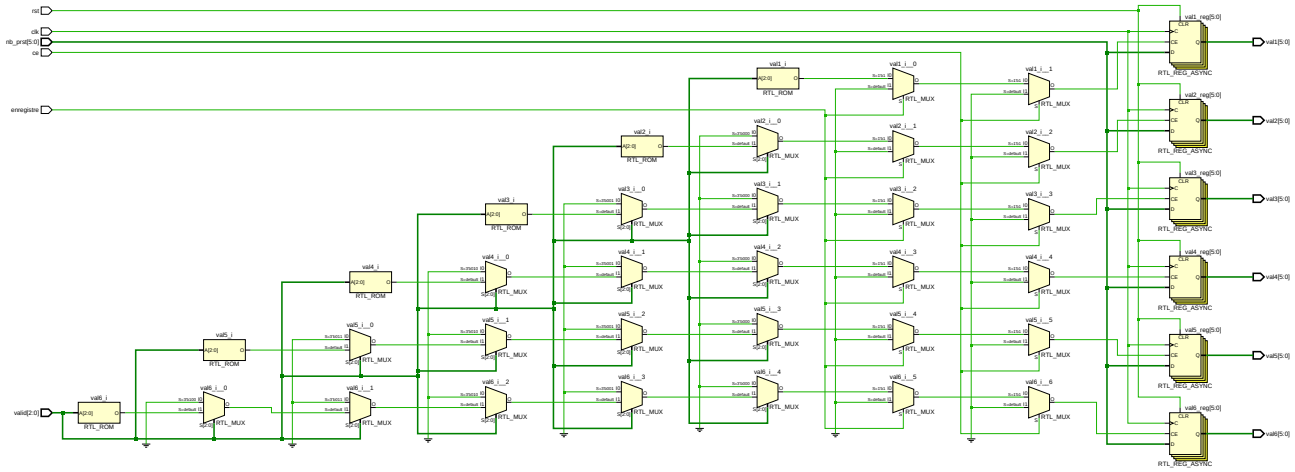

FIGURE 24 – Schématique : mem

3.4 Comparaison des 6 Valeurs - comp

D'après les règles du LOTO, un nombre tiré ne peut être tiré à nouveau. C'est le module comparateur qui permet de vérifier cette condition.

Ce module compare le nombre tiré à tous les autres nombres déjà enregistrés pour ne pas avoir de doublon. Si le nombre à déjà été tiré, la sortie du comparateur passe à 1, sinon elle reste à 0.

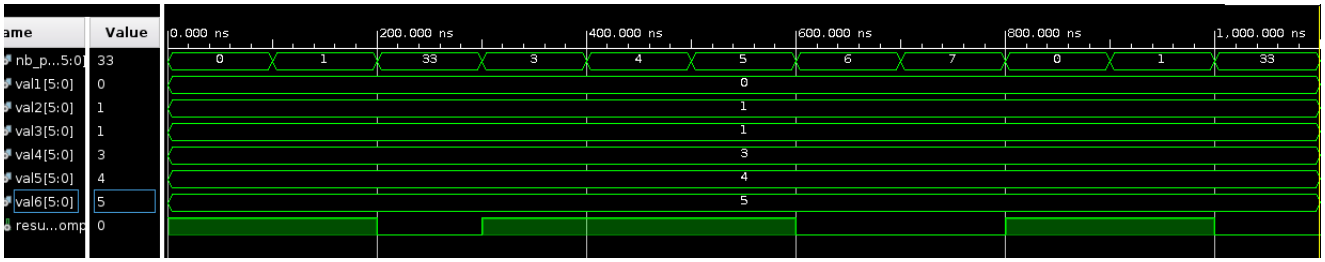

FIGURE 25 – Chronogrammes : comp

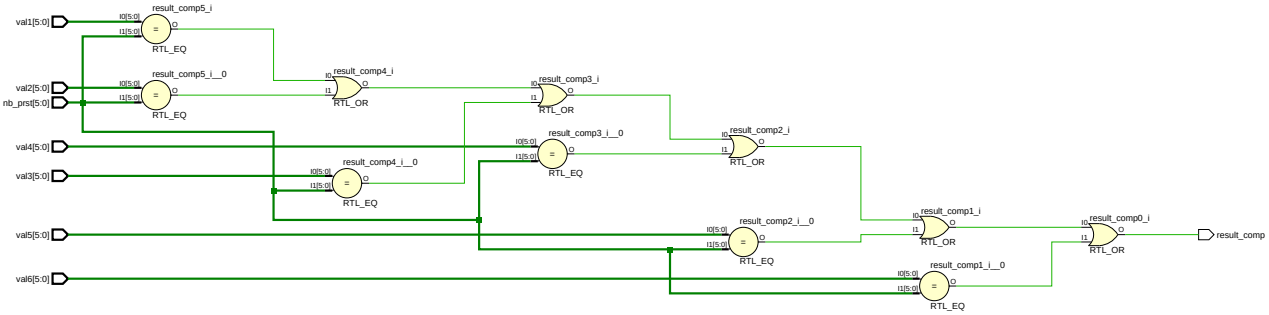

FIGURE 26 – Schématique : comp

3.5 Contrôle du Processus du LOTO - FSM

La FSM (First State Machine) est une machine d'état gérant les modules `compteur_mod6` et mémoire.

En effet, lorsque le bouton est appuyé, la valeur du nombre tiré doit être enregistrée et le numéro de tirage doit passer au suivant. Cependant si le nombre a déjà été tiré (indiqué par la sortie du comparateur), le tirage doit alors être effectué une deuxième fois sans passer au numéro suivant ni l'enregistrer.

Il existe donc plusieurs états provoquant des actions différentes, et nécessitent des conditions différentes pour passer d'un état à l'autre. D'où le nom de machine d'états (FSM).

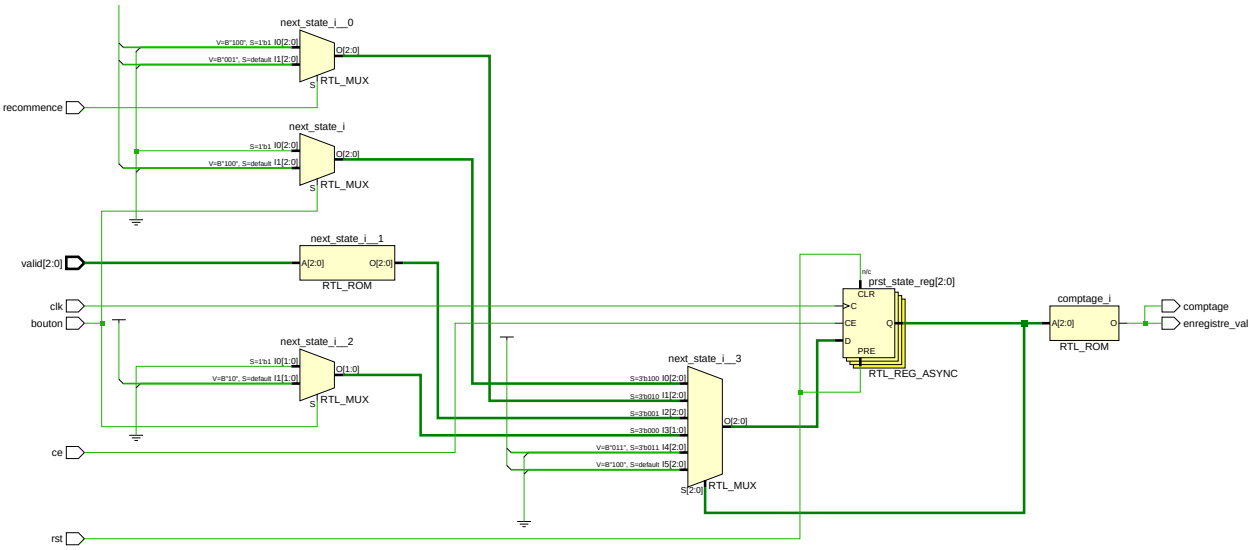

FIGURE 27 – Schématique : FSM

4 Top Level

Le module `top_lvl_aff` assure la coordination complète du système de loto numérique, en reliant le tirage des six valeurs aléatoires à leur affichage dynamique sur les afficheurs 7 segments. Il intègre à la fois la logique de tirage (`top_lvl_tirage`), les modules d'affichage, les multiplexeurs, ainsi que les diviseurs d'horloge nécessaires à la synchronisation.

Chaque seconde, une valeur tirée est sélectionnée via un compteur modulo 6, convertie en décimal puis en code 7 segments. Le rang de cette valeur est aussi affiché, formant un triplet "x - y". Les trois chiffres sont ensuite affichés cycliquement grâce à un multiplexage rapide. Ce top-level offre ainsi une interface fluide, lisible et entièrement automatisée du tirage de loto sur FPGA.

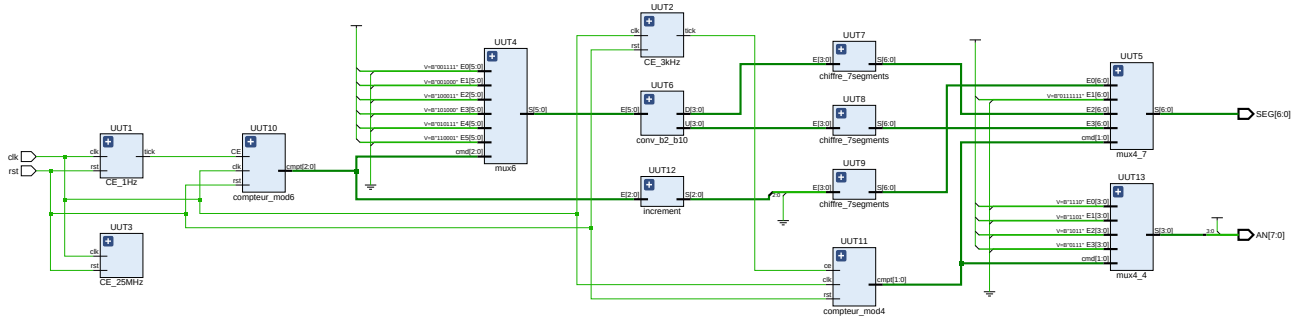

FIGURE 28 – Schématique : Top level sans loto

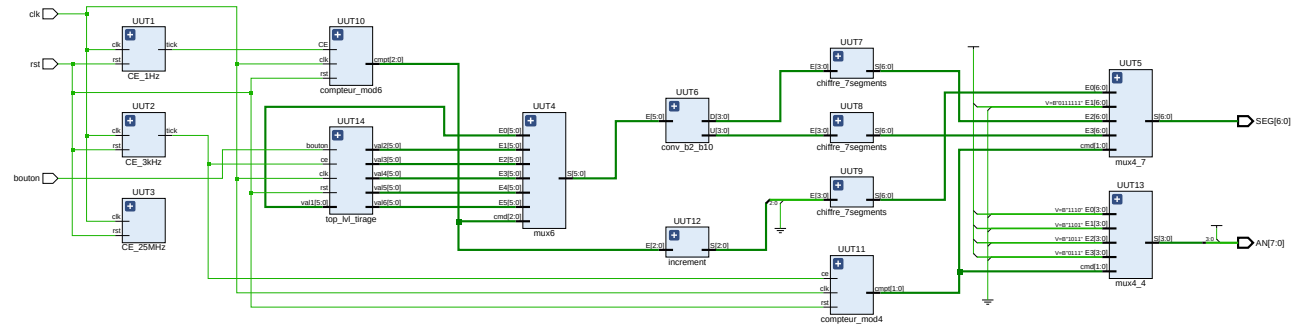

FIGURE 29 – Schématique : Top level avec loto

Conclusion

Ce projet de système de LOTO numérique nous a permis de mettre en pratique les connaissances acquises en logique séquentielle et en conception de circuits numériques. À travers la réalisation d'une architecture complète sur FPGA, nous avons manipulé des

modules variés : compteurs synchrones, machine à états, multiplexeurs, convertisseurs, et modules d'affichage.

La démarche modulaire adoptée nous a permis de concevoir, simuler, tester puis intégrer progressivement les différents blocs du système. L'utilisation de VHDL, associée à l'outil Vivado, nous a permis de valider le fonctionnement du tirage, d'assurer l'unicité des valeurs, et de garantir un affichage fluide et lisible.

Au final, ce projet nous a offert une expérience concrète et complète de la conception numérique, du cahier des charges jusqu'à l'implémentation matérielle sur carte FPGA. Il constitue une étape significative dans notre formation en architecture numérique.

Annexe

Modules de l'Architecture

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity CE_1Hz is
    Port ( clk : in STD_LOGIC;
           rst : in STD_LOGIC;
           tick : out STD_LOGIC);
end CE_1Hz;

architecture Behavioral of CE_1Hz is

signal r_prst , r_next : unsigned (26 downto 0);

begin
    process(clk, rst)
    begin
        if (rst = '1') then
            r_prst <= to_unsigned(0,27);
        elsif (clk'event and clk='1') then
            r_prst <= r_next;
        end if;
     end process;

     cal_sortie : process(r_prst)
     begin
     if (r_prst = to_unsigned(100000000,27)) then
        tick <= '1';
     else
        tick <= '0';
     end if;
     end process cal_sortie;

    cal_next : process(r_prst)
    begin
        if (r_prst = to_unsigned(100000000,27)) then
            r_next <= to_unsigned(0, 27);
        else
            r_next <= r_prst + 1;
        end if;
    end process;

end Behavioral;
```

Listing 1 – CE_1Hz.vhd

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity CE_3kHz is
    Port ( clk : in STD_LOGIC;
           rst : in STD_LOGIC;
           tick : out STD_LOGIC);
end CE_3kHz;

architecture Behavioral of CE_3kHz is

signal r_prst : unsigned (15 downto 0) := to_unsigned(0, 16);
signal r_next : unsigned (15 downto 0);

begin
    process(clk, rst)
    begin
        if (rst = '1') then
            r_prst <= to_unsigned(0,16);
        elsif (clk'event and clk='1') then
            r_prst <= r_next;
        end if;
     end process;

     cal_sortie : process(r_prst)
     begin
        if (r_prst = to_unsigned(33333,16)) then
            tick <= '1';
        else
            tick <= '0';
        end if;
     end process cal_sortie;

    cal_next : process(r_prst)
    begin
        if (r_prst = to_unsigned(33333, 16)) then
            r_next <= to_unsigned(0, 16);
        else
            r_next <= r_prst + 1;
        end if;
    end process;

end Behavioral;
```

Listing 2 – CE_3kHz.vhd

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity CE_25MHz is
    Port ( clk : in STD_LOGIC;
           rst : in STD_LOGIC;
           tick : out STD_LOGIC);
end CE_25MHz;

architecture Behavioral of CE_25MHz is

signal r_prst , r_next : unsigned (1 downto 0) := to_unsigned(0,
   2);

begin
    process(clk, rst)
    begin
        if (rst = '1') then
            r_prst <= to_unsigned(0,2);
        elsif (clk'event and clk='1') then
            r_prst <= r_next;
        end if;
     end process;

     cal_sortie : process(r_prst)
     begin
        if (r_prst = to_unsigned(3,2)) then
            tick <= '1';
        else
            tick <= '0';
        end if;
     end process cal_sortie;

    cal_next : process(r_prst)
    begin
        r_next <= r_prst + 1;
    end process;

end Behavioral;
```

Listing 3 – CE_25MHz.vhd

Fonction LOTO - Top Level Tirage

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity top_lvl_tirage is
    Port ( bouton : in STD_LOGIC;
           ce : in STD_LOGIC;
           clk : in STD_LOGIC;
           rst : in STD_LOGIC;
           val_tirage : inout STD_LOGIC_VECTOR (2 downto 0);
           val1 : inout STD_LOGIC_VECTOR (5 downto 0);
           val2 : inout STD_LOGIC_VECTOR (5 downto 0);
           val3 : inout STD_LOGIC_VECTOR (5 downto 0);
           val4 : inout STD_LOGIC_VECTOR (5 downto 0);
           val5 : inout STD_LOGIC_VECTOR (5 downto 0);
           val6 : inout STD_LOGIC_VECTOR (5 downto 0));
end top_lvl_tirage;

architecture Behavioral of top_lvl_tirage is

component compteur_mod50 is
    port ( ce : in STD_LOGIC;
           clk : in STD_LOGIC;
           rst : in STD_LOGIC;
           bouton : in STD_LOGIC;
           S : out STD_LOGIC_VECTOR (5 downto 0));
end component;

component compteur_mod6_tirage is
    port ( ce : in STD_LOGIC;
           clk : in STD_LOGIC;
           rst : in STD_LOGIC;
           inc : in STD_LOGIC;
           S : out STD_LOGIC_VECTOR (2 downto 0));
end component;

component comparateur is
    port (val1 : in STD_LOGIC_VECTOR (5 downto 0);
        val2 : in STD_LOGIC_VECTOR (5 downto 0);
        val3 : in STD_LOGIC_VECTOR (5 downto 0);
        val4 : in STD_LOGIC_VECTOR (5 downto 0);
        val5 : in STD_LOGIC_VECTOR (5 downto 0);
        val6 : in STD_LOGIC_VECTOR (5 downto 0);
        nb_prst : in STD_LOGIC_VECTOR (5 downto 0);
        result_comp : out STD_LOGIC);
end component;

component FSM is
    port ( ce : in STD_LOGIC;
           clk : in STD_LOGIC;
```

```
            rst : in STD_LOGIC;
            bouton : in STD_LOGIC;
            recommence : in STD_LOGIC;
            valid : in STD_LOGIC_VECTOR (2 downto 0);
            comptage : out STD_LOGIC;
            enregistre_val : out STD_LOGIC);
end component;

component mem is
    port ( clk : in STD_LOGIC;
            rst : in STD_LOGIC;
            ce : in STD_LOGIC;
            enregistre : in STD_LOGIC;
            valid : in STD_LOGIC_VECTOR (2 downto 0);
            nb_prst : in STD_LOGIC_VECTOR (5 downto 0);
            val1 : out STD_LOGIC_VECTOR (5 downto 0);
            val2 : out STD_LOGIC_VECTOR (5 downto 0);
            val3 : out STD_LOGIC_VECTOR (5 downto 0);
            val4 : out STD_LOGIC_VECTOR (5 downto 0);
            val5 : out STD_LOGIC_VECTOR (5 downto 0);
            val6 : out STD_LOGIC_VECTOR (5 downto 0));
end component;


signal nb_prst : STD_LOGIC_VECTOR (5 downto 0);
signal inc, recommence, enregistre : STD_LOGIC;

begin
    UUT1 : compteur_mod50
        port map ( ce => ce,
                clk => clk,
                rst => rst,
                bouton => bouton,
                S => nb_prst);

    UUT2 : compteur_mod6_tirage
        port map ( ce => ce,
                clk => clk,
                rst => rst,
                inc => inc,
                S => val_tirage);

    UUT3 : comparateur
        port map (val1 => val1,
                val2 => val2,
                val3 => val3,
                val4 => val4,
                val5 => val5,
                val6 => val6,
                nb_prst => nb_prst,
                result_comp => recommence);
```

```
101
102     UUT4 : mem
103         port map (val1 => val1,
104                   val2 => val2,
105                   val3 => val3,
106                   val4 => val4,
107                   val5 => val5,
108                   val6 => val6,
109                   nb_prst => nb_prst,
110                   clk => clk,
111                   rst => rst,
112                   ce => ce,
113                   enregistre => enregistre,
114                   valid => val_tirage);
115
116     UUT5 : FSM
117         port map (clk => clk,
118                   rst => rst,
119                   ce => ce,
120                   bouton => bouton,
121                   enregistre_val => enregistre,
122                   recommence => recommence,
123                   valid => val_tirage,
124                   comptage => inc);
125
126 end Behavioral;
```

Listing 4 – top_lvl_tirage.vhd

Transcodage pour Afficheur Sept Segments - Transcodeur

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity chiffre_7segments is
    Port ( E : in STD_LOGIC_VECTOR (3 downto 0);
           S : out STD_LOGIC_VECTOR (6 downto 0));
end chiffre_7segments;

architecture Behavioral of chiffre_7segments is

begin
        cal_cmd : process(E)
    begin
        case E is
                        when "0000" =>
                                S <= "1000000";--0
                        when "0001" =>
                                S <= "1111001";--1
                        when "0010" =>
                                S <= "0100100";--2
                        when "0011" =>
                                S <= "0110000";--3
                        when "0100" =>
                S <= "0011001";--4
            when "0101" =>
                S <= "0010010";--5
            when "0110" =>
                S <= "0000010";--6
            when "0111" =>
                S <= "1111000";--7
            when "1000" =>
                S <= "0000000";--8
            when "1001" =>
                S <= "0010000";--9
                        when others =>
                S <= "0000000";
                end case;
        end process;
end Behavioral;
```

Listing 5 – chiffre_7segments.vhd

Sélection du Nombre à Afficher - Mod6

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity compteur_mod6 is
    Port ( clk : in STD_LOGIC;
           CE : in STD_LOGIC;
           rst : in STD_LOGIC;
           cmpt : out STD_LOGIC_VECTOR (2 downto 0));
end compteur_mod6;

architecture Behavioral of compteur_mod6 is

signal r_prst : unsigned (2 downto 0) := to_unsigned(0, 3);
signal r_next : unsigned (2 downto 0);

begin
    process(clk, rst)
    begin
        if (rst = '1') then
            r_prst <= to_unsigned(0,3);
        elsif (clk'event and clk='1') then
            if (CE='1') then
                r_prst <= r_next;
            end if;
        end if;
     end process;

    cal_sortie : process(r_prst)
     begin
        cmpt <= std_logic_vector(r_prst);
     end process cal_sortie;

    cal_next : process(r_prst)
    begin
        if (r_prst = to_unsigned(5,3)) then
            r_next <= to_unsigned(0,3);
        else
            r_next <= r_prst + 1;
        end if;
    end process;

end Behavioral;
```

Listing 6 – compteur_mod6.vhd

Sélection de l'Anode — Mod4

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity compteur_mod4 is
    Port ( clk : in STD_LOGIC;
           rst : in STD_LOGIC;
           ce : in STD_LOGIC;
           cmpt : out STD_LOGIC_VECTOR (1 downto 0));
end compteur_mod4;

architecture Behavioral of compteur_mod4 is

signal r_prst : unsigned (1 downto 0) := to_unsigned(0, 2);
signal r_next : unsigned (1 downto 0);

begin
    process(clk, rst)
    begin
        if (rst = '1') then
            r_prst <= to_unsigned(0,2);
        elsif (clk'event and clk='1') then
            if (CE='1') then
                r_prst <= r_next;
            end if;
        end if;
     end process;

    cal_sortie : process(r_prst)
     begin
        cmpt <= std_logic_vector(r_prst);
     end process cal_sortie;

    cal_next : process(r_prst)
    begin
        --if (r_prst = to_unsigned(5,3)) then
            --r_next <= to_unsigned(0,3);
        --else
            r_next <= r_prst + 1;
        --end if;
    end process;

end Behavioral;
```

Listing 7 – compteur_mod4.vhd

Module d'Affichage 7 Segments - Mux6

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity mux6 is
    Port ( E0 : in STD_LOGIC_VECTOR (5 downto 0);
           E1 : in STD_LOGIC_VECTOR (5 downto 0);
           E2 : in STD_LOGIC_VECTOR (5 downto 0);
           E3 : in STD_LOGIC_VECTOR (5 downto 0);
           E4 : in STD_LOGIC_VECTOR (5 downto 0);
           E5 : in STD_LOGIC_VECTOR (5 downto 0);
           cmd : in STD_LOGIC_VECTOR (2 downto 0);
           S : out STD_LOGIC_VECTOR (5 downto 0));
end mux6;

architecture Behavioral of mux6 is

begin
        cal_cmd : process(cmd)
    begin
        case cmd is
                        when "000" =>
                                S <= E0;
                        when "001" =>
                                S <= E1;
                        when "010" =>
                                S <= E2;
                        when "011" =>
                                S <= E3;
                        when "100" =>
                S <= E4;
            when "101" =>
                S <= E5;
                        when others =>
                S <= "000000";
                end case;
        end process;
end Behavioral;
```

Listing 8 – mux6.vhd

Module d'Affichage 7 Segments - Mux4

Mux4 - 4bits :

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity mux4_4 is
    Port ( E0 : in STD_LOGIC_VECTOR (3 downto 0);
           E1 : in STD_LOGIC_VECTOR (3 downto 0);
           E2 : in STD_LOGIC_VECTOR (3 downto 0);
           E3 : in STD_LOGIC_VECTOR (3 downto 0);
           cmd : in STD_LOGIC_VECTOR (1 downto 0);
           S : out STD_LOGIC_VECTOR (3 downto 0));
end mux4_4;

architecture Behavioral of mux4_4 is


begin
        cal_cmd : process(cmd)
    begin
        case cmd is
                            when "00" =>
                                    S <= E0;
                            when "01" =>
                                    S <= E1;
                            when "10" =>
                                    S <= E2;
                            when "11" =>
                                    S <= E3;
                            when others =>
                S <= "0000";
                end case;
        end process;
end Behavioral;
```

Listing 9 – mux4_4.vhd

Mux4 - 7bits :

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity mux4_7 is
    Port ( E0 : in STD_LOGIC_VECTOR (6 downto 0);
           E1 : in STD_LOGIC_VECTOR (6 downto 0);
           E2 : in STD_LOGIC_VECTOR (6 downto 0);
           E3 : in STD_LOGIC_VECTOR (6 downto 0);
           cmd : in STD_LOGIC_VECTOR (1 downto 0);
           S : out STD_LOGIC_VECTOR (6 downto 0));
end mux4_7;

architecture Behavioral of mux4_7 is


begin
        cal_cmd : process(cmd)
    begin
        case cmd is
                        when "00" =>
                                S <= E0;
                        when "01" =>
                                S <= E1;
                        when "10" =>
                                S <= E2;
                        when "11" =>
                                S <= E3;
                        when others =>
                        S <= "0000000";
                end case;
        end process;
end Behavioral;
```

Listing 10 – mux4_7.vhd

Modules Tirage

Compteur 1 à 49 - cpt_tirage

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity compteur_mod50 is
    Port ( clk : in STD_LOGIC;
           ce : in STD_LOGIC;
           bouton : in STD_LOGIC;
           rst : in STD_LOGIC;
           S : out STD_LOGIC_VECTOR (5 downto 0));
end compteur_mod50;

architecture Behavioral of compteur_mod50 is

signal r_prst : unsigned (5 downto 0) := to_unsigned(0, 6);
signal r_next : unsigned (5 downto 0);

begin
    process(clk, rst)
    begin
        if (rst = '1') then
            r_prst <= to_unsigned(1,6);
        elsif (clk'event and clk='1') then
            if (ce='1') then
                if (bouton = '1') then
                    r_prst <= r_next;
                end if;
            end if;
        end if;
     end process;

    cal_sortie : process(r_prst)
     begin
        S <= std_logic_vector(r_prst);
     end process cal_sortie;

    cal_next : process(r_prst)
    begin
        if (r_prst = to_unsigned(49,6)) then
            r_next <= to_unsigned(1,6);
        else
            r_next <= r_prst + 1;
        end if;
    end process;

end Behavioral;
```

Listing 11 – compteur_mod50.vhd

Compteur 1 à 6 - cpt_valeurs

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity compteur_mod6_tirage is
    Port ( ce : in STD_LOGIC;
           clk : in STD_LOGIC;
           rst : in STD_LOGIC;
           inc : in STD_LOGIC;
           S : out STD_LOGIC_VECTOR (2 downto 0));
end compteur_mod6_tirage;

architecture Behavioral of compteur_mod6_tirage is

signal r_prst : unsigned (2 downto 0) := to_unsigned(1, 3);
signal r_next : unsigned (2 downto 0);

begin
    process(clk, rst)
    begin
        if (rst = '1') then
            r_prst <= to_unsigned(1,3);
        elsif (clk'event and clk='1') then
            if (ce='1') then
                if (inc = '1') then
                    r_prst <= r_next;
                end if;
            end if;
        end if;
     end process;

    cal_sortie : process(r_prst)
     begin
        S <= std_logic_vector(r_prst);
     end process cal_sortie;

    cal_next : process(r_prst)
    begin
        if (r_prst = to_unsigned(6,3)) then
            r_next <= to_unsigned(1,3);
        else
            r_next <= r_prst + 1;
        end if;
    end process;

end Behavioral;
```

Listing 12 – compteur_mod6_tirage.vhd

Sauvegarde des 6 valeurs - mem

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity mem is
    Port ( clk : in STD_LOGIC;
           rst : in STD_LOGIC;
           ce : in STD_LOGIC;
           enregistre : in STD_LOGIC;
           valid : in STD_LOGIC_VECTOR (2 downto 0);
           nb_prst : in STD_LOGIC_VECTOR (5 downto 0);
           val1 : out STD_LOGIC_VECTOR (5 downto 0);
           val2 : out STD_LOGIC_VECTOR (5 downto 0);
           val3 : out STD_LOGIC_VECTOR (5 downto 0);
           val4 : out STD_LOGIC_VECTOR (5 downto 0);
           val5 : out STD_LOGIC_VECTOR (5 downto 0);
           val6 : out STD_LOGIC_VECTOR (5 downto 0));
end mem;

architecture Behavioral of mem is

begin

    process(clk, rst)
    begin
        if (rst = '1') then
           val1 <= "000000";
           val2 <= "000000";
           val3 <= "000000";
           val4 <= "000000";
           val5 <= "000000";
           val6 <= "000000";


        elsif (clk'event and clk='1') then
            if (ce='1') then
                if (enregistre = '1') then
                    if (valid = "001") then
                        val1 <= nb_prst;
                    elsif (valid = "010") then
                        val2 <= nb_prst;
                    elsif (valid = "011") then
                        val3 <= nb_prst;
                    elsif (valid = "100") then
                        val4 <= nb_prst;
                    elsif (valid = "101") then
                        val5 <= nb_prst;
                    elsif (valid = "110") then
                        val6 <= nb_prst;
                    end if;
```

```
50                  end if;
51              end if;
52          end if;
53      end process;
54
55  end Behavioral;
```

Listing 13 – mem.vhd

Comparaison des 6 valeurs - comp

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity comparateur is
    Port ( nb_prst : in STD_LOGIC_VECTOR (5 downto 0);
           val1 : in STD_LOGIC_VECTOR (5 downto 0);
           val2 : in STD_LOGIC_VECTOR (5 downto 0);
           val3 : in STD_LOGIC_VECTOR (5 downto 0);
           val4 : in STD_LOGIC_VECTOR (5 downto 0);
           val5 : in STD_LOGIC_VECTOR (5 downto 0);
           val6 : in STD_LOGIC_VECTOR (5 downto 0);
           result_comp : out STD_LOGIC);
end comparateur;

architecture Behavioral of comparateur is
begin

    process (nb_prst, val1, val2, val3, val4, val5, val6)
    begin
        if (unsigned(val1) = unsigned(nb_prst)) or
           (unsigned(val2) = unsigned(nb_prst)) or
           (unsigned(val3) = unsigned(nb_prst)) or
           (unsigned(val4) = unsigned(nb_prst)) or
           (unsigned(val5) = unsigned(nb_prst)) or
           (unsigned(val6) = unsigned(nb_prst)) then
            result_comp <= '1';
        else
            result_comp <= '0';
        end if;
    end process;

end Behavioral;
```

Listing 14 – comparateur.vhd

Contrôle du Precessus du Loto - FSM

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity FSM is
    Port ( clk : in STD_LOGIC;
           rst : in STD_LOGIC;
           ce : in STD_LOGIC;
           bouton : in STD_LOGIC;
           recommence : in STD_LOGIC;
           valid : in STD_LOGIC_VECTOR (2 downto 0);
           comptage : out STD_LOGIC;
           enregistre_val : out STD_LOGIC);
end FSM;

architecture Behavioral of FSM is

type state_type is (compter, enregistrer, recommencer, fin,
   neutre);
signal prst_state, next_state : state_type;

begin
    process(clk, rst)
    begin
        if (rst = '1') then
            prst_state <= neutre;
        elsif (clk'event and clk='1') then
            if (ce='1') then
                prst_state <= next_state;
            end if;
        end if;
     end process;


    cal_sortie : process(prst_state)
    begin
        case (prst_state) is
            when compter =>
                comptage <= '0';
                enregistre_val <= '0';
            when enregistrer =>
                comptage <= '1';
                enregistre_val <= '1';
            when recommencer =>
                comptage <= '0';
                enregistre_val <= '0';
            when fin =>
                comptage <= '0';
                enregistre_val <= '0';
```

```
49              when neutre =>
50                  comptage <= '0';
51                  enregistre_val <= '0';
52          end case;
53      end process;
54
55      cal_next_state : process(prst_state, bouton, recommence)
56      begin
57          case (prst_state) is
58              when neutre =>
59                  if (bouton = '1') then
60                      next_state <= compter;
61                  else
62                      next_state <= neutre;
63                  end if;
64
65              when recommencer =>
66                  if (recommence = '1') then
67                      next_state <= neutre;
68                  else
69                      next_state <= enregistrer;
70                  end if;
71
72              when enregistrer =>
73                  if (valid = "110") then
74                      next_state <= fin;
75                  else
76                      next_state <= neutre;
77                  end if;
78
79              when compter =>
80                  if (bouton = '1') then
81                      next_state <= compter;
82                  else
83                      next_state <= recommencer;
84                  end if;
85
86              when fin =>
87                  next_state <= fin;
88
89              when others =>
90                  next_state <= neutre;
91
92          end case;
93      end process;
94
95  end Behavioral;
```

Listing 15 – FSM.vhd

Top Level

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity top_lvl_aff is
    Port ( clk : in STD_LOGIC;
           rst : in STD_LOGIC;
           bouton : in STD_LOGIC;
           SEG : out STD_LOGIC_VECTOR (6 downto 0);
           AN : out STD_LOGIC_VECTOR (7 downto 0));
end top_lvl_aff;

architecture Behavioral of top_lvl_aff is

component CE_1Hz is
    port(clk, rst : in std_logic;
        tick : out std_logic);
end component;

component CE_3kHz is
    port(clk, rst : in std_logic;
        tick : out std_logic);
end component;

component CE_25MHz is
    port(clk, rst : in std_logic;
        tick : out std_logic);
end component;

component mux6 is
    port ( E0 : in STD_LOGIC_VECTOR (5 downto 0);
           E1 : in STD_LOGIC_VECTOR (5 downto 0);
           E2 : in STD_LOGIC_VECTOR (5 downto 0);
           E3 : in STD_LOGIC_VECTOR (5 downto 0);
           E4 : in STD_LOGIC_VECTOR (5 downto 0);
           E5 : in STD_LOGIC_VECTOR (5 downto 0);
           cmd : in STD_LOGIC_VECTOR (2 downto 0);
           S : out STD_LOGIC_VECTOR (5 downto 0));
end component;

component mux4_7 is
    port ( E0 : in STD_LOGIC_VECTOR (6 downto 0);
           E1 : in STD_LOGIC_VECTOR (6 downto 0);
           E2 : in STD_LOGIC_VECTOR (6 downto 0);
           E3 : in STD_LOGIC_VECTOR (6 downto 0);
           cmd : in STD_LOGIC_VECTOR (1 downto 0);
           S : out STD_LOGIC_VECTOR (6 downto 0));
end component;

```

```
50  component conv_b2_b10 is
51      port ( E : in STD_LOGIC_VECTOR (5 downto 0);
52             D : out STD_LOGIC_VECTOR (3 downto 0);
53             U : out STD_LOGIC_VECTOR (3 downto 0));
54  end component;
55
56  component chiffre_7segments is
57      port ( E : in STD_LOGIC_VECTOR (3 downto 0);
58             S : out STD_LOGIC_VECTOR (6 downto 0));
59  end component;
60
61  component increment is
62      port ( E : in STD_LOGIC_VECTOR (2 downto 0);
63             S : out STD_LOGIC_VECTOR (2 downto 0));
64  end component;
65
66  component compteur_mod6 is
67      port ( CE : in STD_LOGIC;
68             clk : in STD_LOGIC;
69             rst : in STD_LOGIC;
70             cmpt : out STD_LOGIC_VECTOR (2 downto 0));
71  end component;
72
73  component compteur_mod4 is
74      port ( ce : in STD_LOGIC;
75             clk : in STD_LOGIC;
76             rst : in STD_LOGIC;
77             cmpt : out STD_LOGIC_VECTOR (1 downto 0));
78  end component;
79
80  component mux4_4 is
81      port ( E0 : in STD_LOGIC_VECTOR (3 downto 0);
82             E1 : in STD_LOGIC_VECTOR (3 downto 0);
83             E2 : in STD_LOGIC_VECTOR (3 downto 0);
84             E3 : in STD_LOGIC_VECTOR (3 downto 0);
85             cmd : in STD_LOGIC_VECTOR (1 downto 0);
86             S : out STD_LOGIC_VECTOR (3 downto 0));
87  end component;
88
89
90  component top_lvl_tirage is
91      Port ( bouton : in STD_LOGIC;
92             ce : in STD_LOGIC;
93             clk : in STD_LOGIC;
94             rst : in STD_LOGIC;
95             val_tirage : inout STD_LOGIC_VECTOR (2 downto 0);
96             val1 : inout STD_LOGIC_VECTOR (5 downto 0);
97             val2 : inout STD_LOGIC_VECTOR (5 downto 0);
98             val3 : inout STD_LOGIC_VECTOR (5 downto 0);
99             val4 : inout STD_LOGIC_VECTOR (5 downto 0);
100            val5 : inout STD_LOGIC_VECTOR (5 downto 0);
```

```
            val6 : inout STD_LOGIC_VECTOR (5 downto 0));
end component;

--anode
signal AN_4 : STD_LOGIC_VECTOR (3 downto 0);
-- clk enable
signal clk_1Hz, clk_3kHz, clk_25MHz  : std_logic;
signal bouton_sync : std_logic;
--mux6
signal E0_6, E1_6, E2_6, E3_6, E4_6, E5_6, S_6 : STD_LOGIC_VECTOR
    (5 downto 0);
--mux4_7
signal cmd4 : STD_LOGIC_VECTOR (1 downto 0);
--conv_b2_b10
signal D, U : STD_LOGIC_VECTOR (3 downto 0); -- la sortie est S_6
--chiffre_7segments
signal E0_47, E1_47, E2_47, E3_47: STD_LOGIC_VECTOR (6 downto 0);
signal INC_4 : STD_LOGIC_VECTOR (3 downto 0);
--increment
signal INC_3 : STD_LOGIC_VECTOR (2 downto 0);
--compteur_mod6
signal MOD_6,val_tirage : STD_LOGIC_VECTOR (2 downto 0);
--mux4_4
signal E0_44, E1_44, E2_44, E3_44: STD_LOGIC_VECTOR (3 downto 0);

begin
    UUT1 : CE_1Hz
        port map(clk => clk,
                 rst => rst,
                 tick => clk_1Hz);
    UUT2 : CE_3kHz
        port map(clk => clk,
                 rst => rst,
                 tick => clk_3kHz);
    UUT3 : CE_25MHz
        port map(clk => clk,
                 rst => rst,
                 tick => clk_25MHz);

    UUT4 : mux6
        port map(E0 => E0_6,
            E1 => E1_6,
            E2 => E2_6,
            E3 => E3_6,
            E4 => E4_6,
            E5 => E5_6,
            cmd => MOD_6,
            S => S_6);

    UUT5 : mux4_7
        port map(E0 => E0_47,
```

```
151             E1 => E1_47,
152             E2 => E2_47,
153             E3 => E3_47,
154             cmd => cmd4,
155             S => SEG);
156 
157     UUT6 : conv_b2_b10
158         port map(E => S_6,
159             D => D,
160             U => U);
161 
162     UUT7 : chiffre_7segments --dixaines
163         port map(E => D,
164             S => E2_47);
165 
166     UUT8 : chiffre_7segments --unitees
167         port map(E => U,
168             S => E3_47);
169 
170     UUT9 : chiffre_7segments
171         port map(E => INC_4,
172             S => E0_47);
173 
174     UUT10 : compteur_mod6
175         port map(CE => clk_1Hz,
176             clk => clk,
177             rst => rst,
178             cmpt => MOD_6);
179 
180     UUT11 : compteur_mod4
181         port map(CE => clk_3kHz,
182             clk => clk,
183             rst => rst,
184             cmpt => cmd4);
185 
186     UUT12 : increment
187         port map(E => MOD_6,
188             S => INC_3);
189 
190     UUT13 : mux4_4
191         port map(E0 => E0_44,
192             E1 => E1_44,
193             E2 => E2_44,
194             E3 => E3_44,
195             cmd => cmd4,
196             S => AN_4);
197 
198     UUT14 : top_lvl_tirage
199         Port map( bouton => bouton,
200            ce => clk_3kHz,
201            clk => clk,
```

```
202             rst => rst,
203             val_tirage => val_tirage,
204             val1 => E0_6,
205             val2 => E1_6,
206             val3 => E2_6,
207             val4 => E3_6,
208             val5 => E4_6,
209             val6 => E5_6);
210 
211 
212 
213     INC_4 <= '0' & INC_3;
214 
215     AN <= "1111" & AN_4;
216 
217     E1_47 <= "0111111";
218 
219 
220     E0_44 <= "1110";
221     E1_44 <= "1101";
222     E2_44 <= "1011";
223     E3_44 <= "0111";
224 
225 end Behavioral;
```

Listing 16 – top_lvl_aff.vhd