POLYTECHNIQUE
MONTRÉAL

INF1500
Logique des systèmes numériques

Laboratoire 5

Soumis par :
Boukaftane, Hamza - 2183376
Lidder, Arman - 2174916

12 avril 2022

1- Description du système

a) Description fonctionnelle

Le circuit à réaliser est un circuit séquentiel simulant un système de mot de passe de 4 caractères. Ce système affiche des informations sur les 4 afficheurs 7 segments en fonction des différents états définissant le système. Les entrées sont A (1 bit), B (1 bit), C (1 bit), RST (1 bit) et CLK (1 bit). En ce qui concerne A, B et C, elles correspondent aux caractères qui sont affichés sur les afficheurs quand l'utilisateur veut saisir le mot de passe du système. Pour RST, cette entrée est une fonctionnalité qui permet de réinitialiser le système en forçant le retour du circuit à son état initiale. Toutes ces entrées sont implémentées sur différent boutons de la carte FPGA, à l'exception de CLK qui est le signal d'horloge sur lequel se synchronise le circuit qui constitue une entrée pour tous les modules du circuit. Les sorties du circuit sont SEVEN_SEG (8 bits) qui correspond à l'encodage permettant d'afficher soit le caractère entré soit l'état d'un système, et AN (8 bits) dont la valeur hexadécimale varie entre 0xFE (position 0), 0xFD (position 1), 0xFB (position 2) et 0xF7 (position 3) ce qui permet de choisir la position de l'afficheur 7 segments sur laquelle est affichée la sortie SEVEN_SEG. Pour bien fonctionner, le circuit complet est composé de 4 modules distincts : Debounce, PulseGenerator, Digilock, Disp7seg. En effet, le module Debounce permet de filtrer le signal du bouton appuyé afin d'éviter les rebondissements de bouton envoyant une multitude de signaux. Ce module nous est fourni par les chargés de cour. Ensuite, le module PulseGenerator qui permet de synchroniser le signal généré par l'appui d'un bouton au signal d'horloge CLK et de générer des pulsions unique indifférente du temps de pressage du bouton. Puis, le module Digilock qui décrit le comportement par le biais de 4 états globales d'affichage : INIT (---L), Entrée (AC--), OUVERT (oooo) et Alarme (AL--). Le système commence à l'état INIT et le pressage de n'importe quel bouton permet le passage à l'état Entrée. Dans l'état Entrée, l'afficheur affiche les boutons que l'utilisateur presse. Si le code correspond à CACB, le circuit passe à l'état OUVERT. Sinon, le circuit retourne à l'état INIT à l'intérieur d'un cycle de 3 échecs de codes. Après 3 échecs, il y a transition à l'état Alarme. Pour sortir de cet état, il faut que l'utilisateur entre la séquence C suivie de B peu importe les boutons pressés avant C. Si la séquence est bien entrée, l'état du système passe à l'état INIT. Quand l'état du système est OUVERT, il

suffit d'appuyer sur n'importe quel bouton pour passer à l'état INIT. Ce module génère 4 sorties qui correspondent chacun à l'encodage binaire sur 4 bits du caractère qui sera affiché sur un des 4 afficheurs (SEG_3 = 0xF7, SEG_2 = 0xFB, SEG_1=0xFD, 0xFE). Enfin, les sorties du module Digilock sont traitées dans le module disp7seg qui permet une traduction des signaux encodé en binaire vers un encodage en hexadécimale qui permettra d'afficher le caractère voulu. Ce module nous est gracieusement fourni par les chargés de projet. Finalement, en regroupant les modules décrits plus tôt comme sur le schéma ci-dessous, on obtient un circuit permettant de simuler un système de mot de passe de 4 caractères.

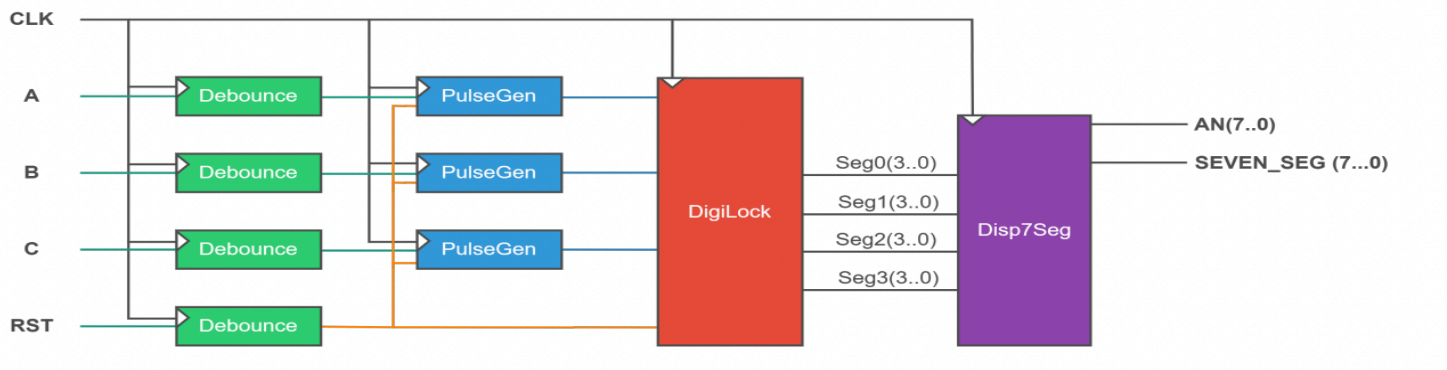


Figure 1. Circuit complet

b) Description des diagrammes

1- Module PulseGenerator :

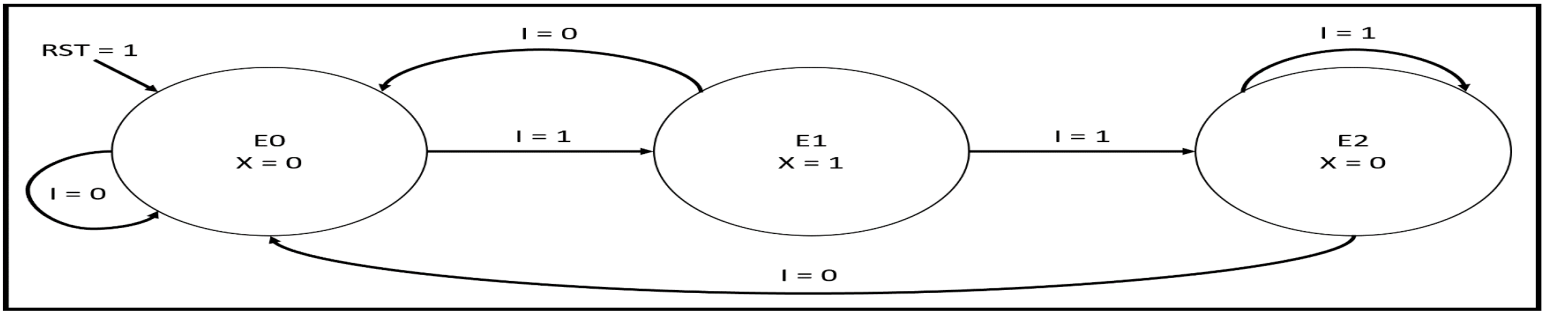


Figure 2. Diagramme de Moore PulseGenerator.

Le diagramme suivant permet de décrire les 3 étapes constituant notre générateur de pulsion et les transitions entre ces différentes étapes. L'entrée est I et la sortie est X. Ainsi, initialement, nous sommes à l'état 0 et quand il y a détecté de pression de bouton, on passe à l'état 1 qui demeure effective sur une période d'un seul front montant d'horloge. L'étape 2 force la sortie à 1 même si I est toujours à 1 après étape 1.

2- Module Digilock

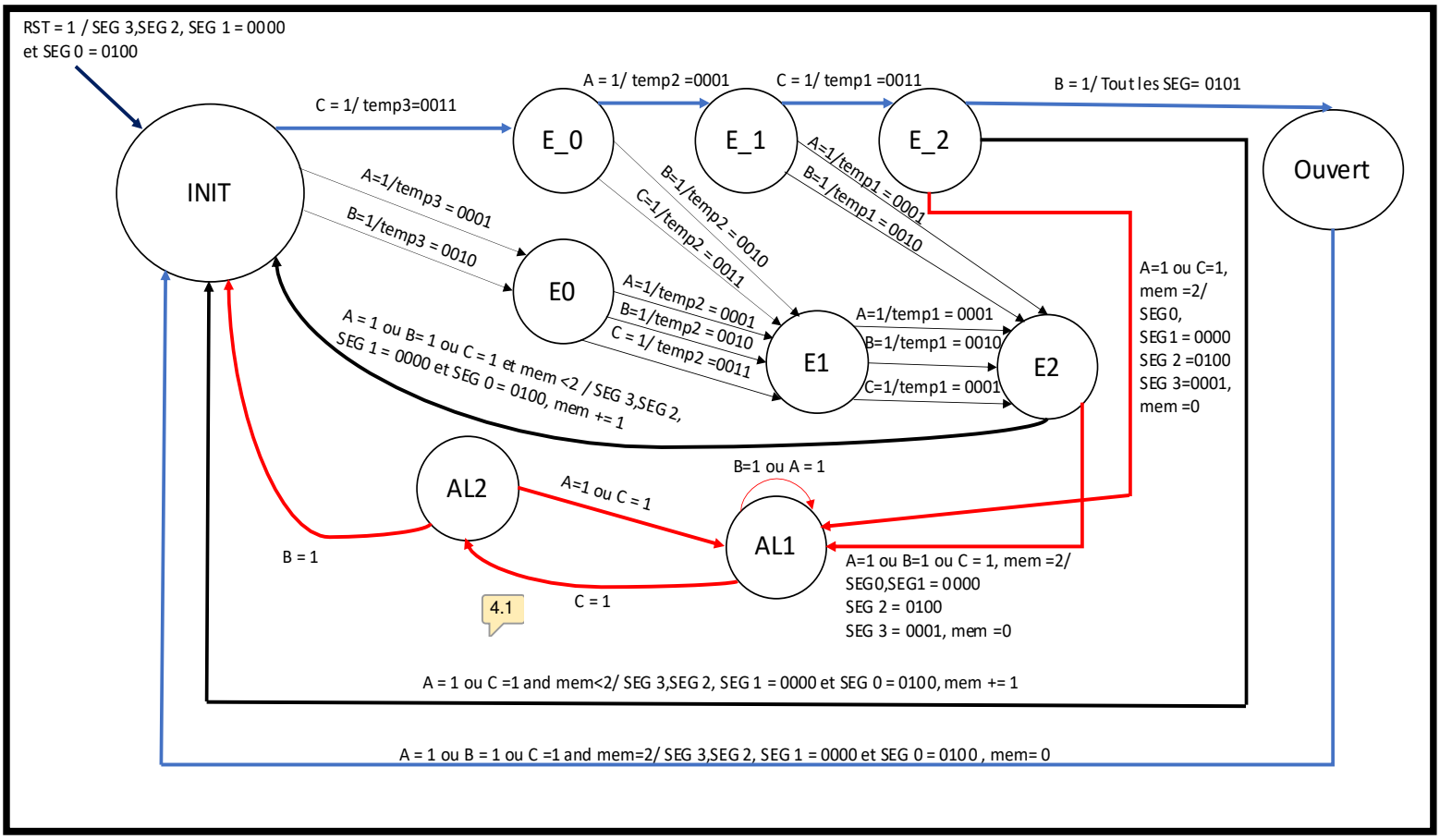


Figure 3. Diagramme de Mealy Digilock.

* Afin d'alléger le diagramme, nous avons omis de signaler les informations suivantes :

E0 et E_0 : seg0, seg1, seg2= 0000 et seg3=temps3

E1 et E_1 : seg0, seg1 = 0000 et seg3=temps3, seg2= temps2

E2 et E_2 : seg0 = 0000 et seg3=temps3, seg2= temps2 et seg1 = temps1

* De plus, comme il n'est pas clairement exigé d'afficher le dernier caractère dans l'étape Entrée, nous avons décidé de ne pas l'afficher mais de l'utilisé directement comme activateur de la transition vers autre étape.

Dans ce diagramme, l'état Entrée est décomposé en 6 sous états. Quand E sans « _ », le code entré est mauvais et quand E avec « _ », le code entré correspond à CACB. Les chiffres de cette étape correspondent à la position de l'entrée sur la séquence de quatre. On décompose aussi l'état Alarme en deux sous états AL1 correspondant à la première entrée saisie et AL2 correspondant à la deuxième entrée saisie.

c) Description des modules en VHDL :

1- Module PulseGenerator

```
21
22  library IEEE;
23  use IEEE.STD_LOGIC_1164.ALL;
24
25  entity PulseGenerator is
26      Port(   CLK, RST: in std_logic;
27              I: in std_logic;
28              X: out std_logic);
29  end PulseGenerator;
30  architecture comportementale of PulseGenerator is
31  type E is (E0, E1, E2);
32  signal Ep , Ef: E := E0;
33  begin
34  process (CLK, RST)
35  begin
36          if (RST = '1') then
37              Ep <= E0;
38          elsif rising_edge(CLK) then
39              Ep <= Ef;
40          end if;
41  end process;
42  process (Ep, I)
43  begin
44      case Ep is                                  5.1
45          when E0 =>
46              X <= '0';
47              if (I = '1') then
48                  Ef <= E1;
49              else
50                  Ef <= E0;
51              end if;
52          when E1 =>
53              X <= '1';
54              if (I = '1') then
55                  Ef <= E2;
56              else
57                  Ef <= E1;
58              end if;
59           when E2 =>
60              X <= '0';
61              if (I = '1') then
62                  Ef <= E2;
63              else
64                  Ef <= E0;
65              end if;
66      end case;
67  end process;
68  end comportementale;
69
```

Générée à partir du diagramme 1.b).1.

2- Module Digilock

```
22  library IEEE;
23  use IEEE.STD_LOGIC_1164.ALL;
24  entity Digilock is
25      Port ( CLK, RST : in STD_LOGIC;
26             A,B,C : in STD_LOGIC;
27             Seg0, Seg1, Seg2, Seg3 : out std_logic_vector (3 downto 0));
28  end Digilock;
29  architecture Behavioral of Digilock is
30  type E is (INIT,E0,E1,E2,E_0,E_1,E_2,OUVERT, AL1, AL2);
31  signal Ep, Ef: E := INIT;
32  signal printM : integer;
33  begin
34  process (CLK, RST)
35  begin
36          if (RST = '1') then
37              Ep <= INIT;
38          elsif rising_edge(CLK) then
39              Ep <= Ef;
40          end if;
41  end process;
42  process (Ep, A, B,C)
43  variable temp1, temp2, temp3: std_logic_vector (3 downto 0);
44  variable mem: integer := 0;
45  begin
46      case Ep is
47          -- INITIATION du coffre fort ---L
48          when INIT =>
49              Seg3 <= "0000";
50              Seg2 <= "0000";
51              Seg1 <= "0000";
52              Seg0 <= "0100";
53              if A = '1' then
54                  Ef <= E0;
55                  temp3 := "0001";
56              elsif B = '1' then
57                  Ef <= E0;
58                  temp3 := "0010";
59
60              elsif C = '1' then
61                  Ef <= E_0;
62                  temp3 := "0011";
63              else
64                  Ef <= INIT;
65              end if;
66          -- Entree utilisateur 3 steps with good pass CACB
67          when E_0 =>
68              Seg3 <= temp3;
69              Seg2 <= "0000";
70              Seg1 <= "0000";
71              Seg0 <= "0000";
72              if A = '1' then
73                  Ef <= E_1;
74                  temp2 := "0001";
75              elsif B = '1' then
76                  Ef <= E1;
```

```
77                  temp2 := "0010";
78              elsif C = '1' then
79                  Ef <= E1;
80                  temp2 := "0011";
81              else
82                  Ef <= E_0;
83              end if;
84          when E_1 =>
85              Seg3 <= temp3;
86              Seg2 <= temp2;
87              Seg1 <= "0000";
88              Seg0 <= "0000";
89              if A = '1' then
90                  Ef <= E2;
91                  temp1 := "0001";
92              elsif B = '1' then
93                  Ef <= E2;
94                  temp1 := "0010";
95              elsif C = '1' then
96                  Ef <= E_2;
97                  temp1 := "0011";
98              else
99                  Ef <= E_1;
100             end if;
101         when E_2 =>
102             Seg3 <= temp3;
103             Seg2 <= temp2;
104             Seg1 <= temp1;
105             Seg0 <= "0000";
106             if A = '1' and mem < 2 then
107                 Ef <= INIT;
108                 mem := mem + 1;
109             elsif B = '1' and mem < 2 then
110                 Ef <= OUVERT;
111                 mem := 0;
112             elsif C = '1' and mem < 2 then
113                 Ef <= INIT;
114                 mem := mem + 1;
115             elsif A = '1' and mem = 2  then
116                 Ef <= AL1;
117                 mem := 0;
118             elsif B = '1' and mem = 2 then
119                 Ef <= OUVERT;
120                 mem := 0;
121             elsif C = '1' and mem = 2 then
122                 Ef <= A11;
123                 mem := 0;
124             else
125                 Ef <= E_2;
126             end if;
127         -- OUVERT state oooo
128         when OUVERT =>
129             Seg3 <= "0101";
130             Seg2 <= "0101";
```

```
131             Seg1 <= "0101";
132             Seg0 <= "0101";
133             if A = '1' then
134                 Ef <= Init;
135             elsif B = '1' then
136                 Ef <= Init;
137             elsif C = '1' then
138                 Ef <= Init;
139             else
140                 Ef <= OUVERT;
141             end if;
142         -- Entree utilisateur 3 steps without good password
143         when E0 =>
144             Seg3 <= temp3;
145             Seg2 <= "0000";
146             Seg1 <= "0000";
147             Seg0 <= "0000";
148             if A = '1' then
149                 Ef <= E1;
150                 temp2 := "0001";
151             elsif B = '1' then
152                 Ef <= E1;
153                 temp2 := "0010";
154             elsif C = '1' then
155                 Ef <= E1;
156                  temp2 := "0011";
157             else
158                 Ef <= E0;
159             end if;
160         when E1 =>
161             Seg3 <= temp3;
162             Seg2 <= temp2;
163             Seg1 <= "0000";
164             Seg0 <= "0000";
165             if A = '1' then
166                 Ef <= E2;
167                 temp1 := "0001";
168             elsif B = '1' then
169                 Ef <= E2;
170                 temp1 := "0010";
171             elsif C = '1' then
172                 Ef <= E2;
173                  temp1 := "0011";
174             else
175                 Ef <= E1;
176             end if;
177          when E2 =>
178             Seg3 <= temp3;
179             Seg2 <= temp2;
180             Seg1 <= temp1;
181             Seg0 <= "0000";
182             if A = '1'  and mem < 2 then
```

```
183                 Ef <= INIT;
184                 mem := mem + 1;
185               elsif B = '1' and mem < 2 then
186                   Ef <= INIT;
187                   mem := mem + 1;
188               elsif C = '1' and mem < 2 then
189                   Ef <= INIT;
190                   mem := mem + 1;
191               elsif  mem = 2 then
192                   Ef <= AL1;
193                   mem := 0;
194               else
195                   Ef <= E2;
196                   mem := mem;
197             end if;
198           --- 2 step Alarme State when mem = 3
199           when AL1 =>
200             Seg3 <= "0001";
201             Seg2 <= "0100";
202             Seg1 <= "0000";
203             Seg0 <= "0000";
204             if C = '1' then
205                 Ef <= AL2;
206             else
207                 Ef <= AL1;
208             end if;
209           when AL2 =>
210             Seg3 <= "0001";
211             Seg2 <= "0100";
212             Seg1 <= "0000";
213             Seg0 <= "0000";
214
215             if B = '1' then
216                 Ef <= INIT;
217             elsif A = '1' then
218                 Ef <= AL1;
219             elsif B = '1' then
220                 Ef <= AL1;
221             else
222                 Ef <= AL2;
223             end if;
224     end case;
225     printM <= mem;
226 end process;
227 end  Behavioral;
```

Générée par le diagramme 1.b).2.

3- Circuit complet

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity Circuit_complet is
    Port (  CLK,RST: in std_logic;
            A,B,C: in std_logic;
            S_AN: out std_logic_vector(7 downto 0);
            S: out std_logic_vector (7 downto 0));
end Circuit_complet;
architecture Behavioral of Circuit_complet is
component Debounce is
PORT(
        CLK    : IN  STD_LOGIC;
       button  : IN  STD_LOGIC;
       result  : OUT STD_LOGIC);
end component;
component PulseGenerator is
    Port(   CLK, RST: in std_logic;
            I: in std_logic;
            X: out std_logic);
end component;
component Digilock is
    Port (  CLK, RST : in STD_LOGIC;
            A : in STD_LOGIC;
            B : in STD_LOGIC;
            C : in STD_LOGIC;
            Seg0, Seg1, Seg2, Seg3 : out std_logic_vector (3 downto 0));
end component;
component DISP_7_SEG_LAB4 is
PORT (  CLK :       IN STD_LOGIC;
        SEG_0 :     IN STD_LOGIC_VECTOR(3 downto 0);
        SEG_1 :     IN STD_LOGIC_VECTOR(3 downto 0);
        SEG_2 :     IN STD_LOGIC_VECTOR(3 downto 0);
        SEG_3 :     IN STD_LOGIC_VECTOR(3 downto 0);
        AN :        OUT STD_LOGIC_VECTOR(7 downto 0);
        SEVEN_SEG : OUT STD_LOGIC_VECTOR(7 downto 0));
end component;
    signal DA, DB, DC, DRST : std_logic;
    signal PA, PB, PC : std_logic;
    signal tmp0, tmp1, tmp2, tmp3: std_logic_vector (3 downto 0);
begin
    U2: DISP_7_SEG_LAB4 port map ( SEG_0 => tmp0, SEG_1 => tmp1, SEG_2 => tmp2, SEG_3 => tmp3, CLK => CLK, SEVEN_SEG => S, AN => S_AN);
    U1: Digilock port map (A => PA, B => PB, C => PC, CLK => CLK, RST => DRST, Seg0 => tmp0, Seg1 => tmp1, Seg2 => tmp2, Seg3 => tmp3);
    P_C: PulseGenerator port map (I => DC, X => PC, CLK => CLK, RST => DRST);
    P_B: PulseGenerator port map (I => DB, X => PB, CLK => CLK, RST => DRST);
    P_A: PulseGenerator port map (I => DA, X => PA, CLK => CLK, RST => DRST);
    D_RST: Debounce port map (button => RST, result => DRST, clk => CLK);
    D_C: Debounce port map (button => C, result => DC, CLK => CLK);
    D_B: Debounce port map (button => B, result => DB, CLK => CLK);
    D_A: Debounce port map (button => A, result => DA, CLK => CLK);
end Behavioral;
```

Généré en connectant les différentes composantes du circuit comme sur la Figure 1.

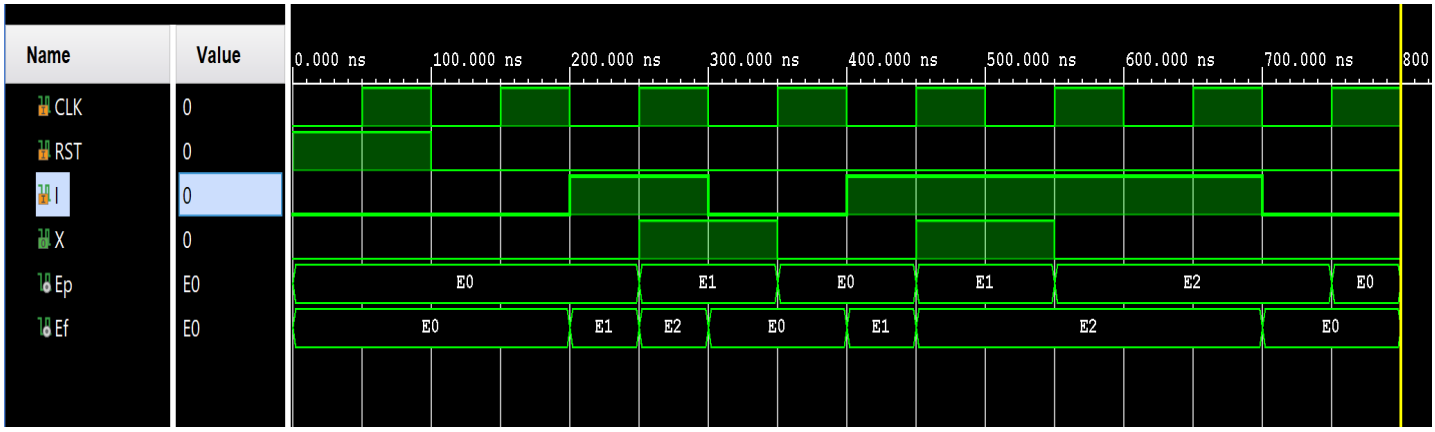
2- Vérification du système

a) Choix de validation

Nous avons utilisé un test non exhaustif afin de valider individuellement tous les modules de notre circuit. Une fois tous les modules validés, nous avons procédé au test exhaustif du circuit complet. Le test nous exhaustif permet de tester les transitions entre les différents états définies en paramétrant l'intervalle de temps dans lequel le signal d'horloge change de 0 à 1 et en forçant des valeurs aux entrées voulues successivement par période d'horloge. Ainsi, nous pouvons observer le comportement du circuit et déterminer sa validité en fonction du comportement attendu.

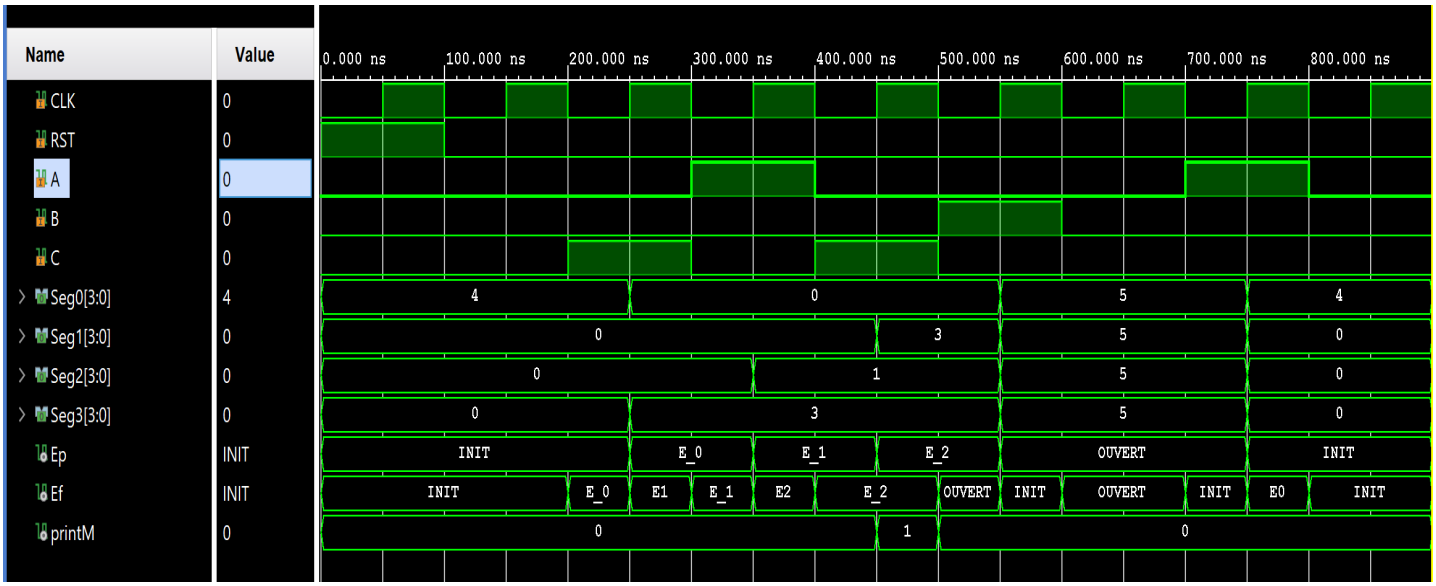
b) Simulation

1- Module PulseGenerator:

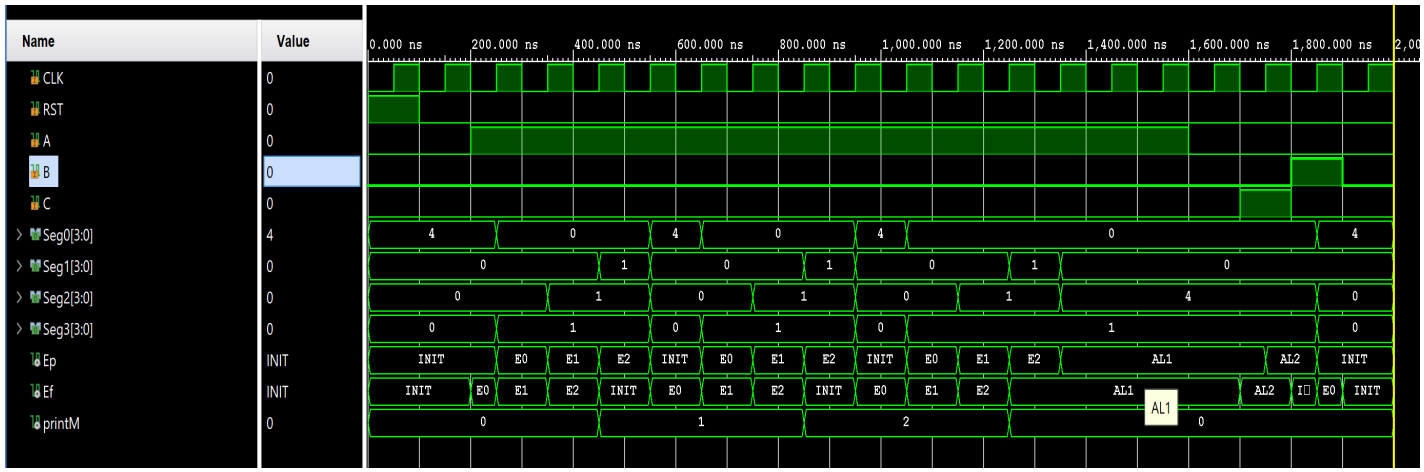


2- Module Digilock

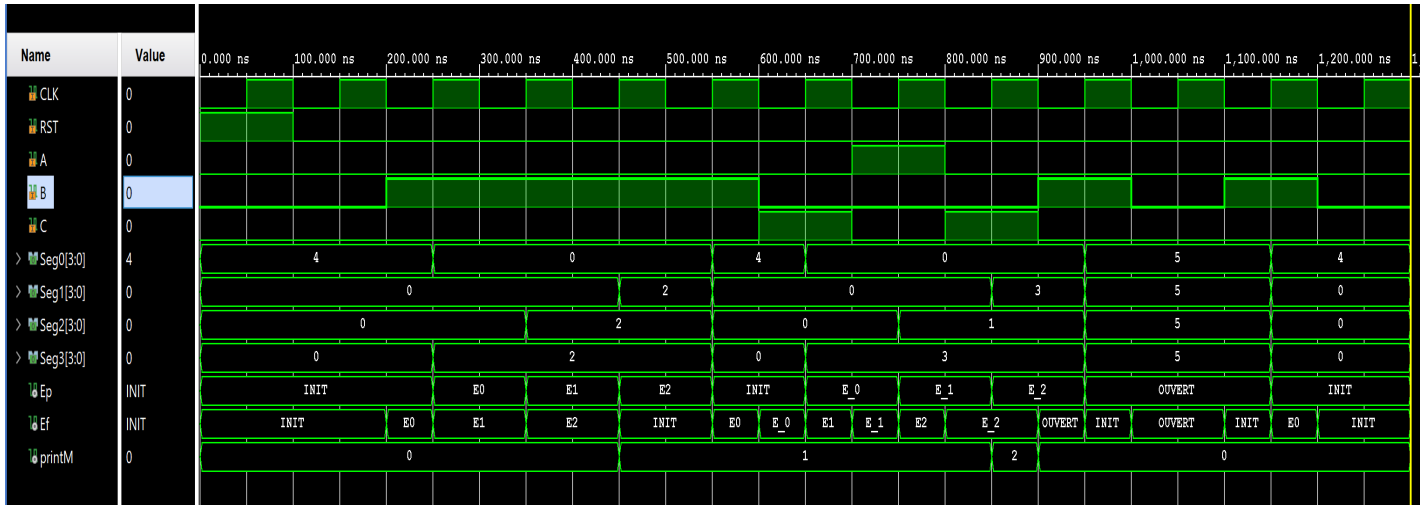
Séquence = INIT = > Entrée (Bon Code 1 fois) => OUVERT = > INIT



Séquence = INIT => Entré (3x mauvais code) => Alarme (C=>B) => INIT

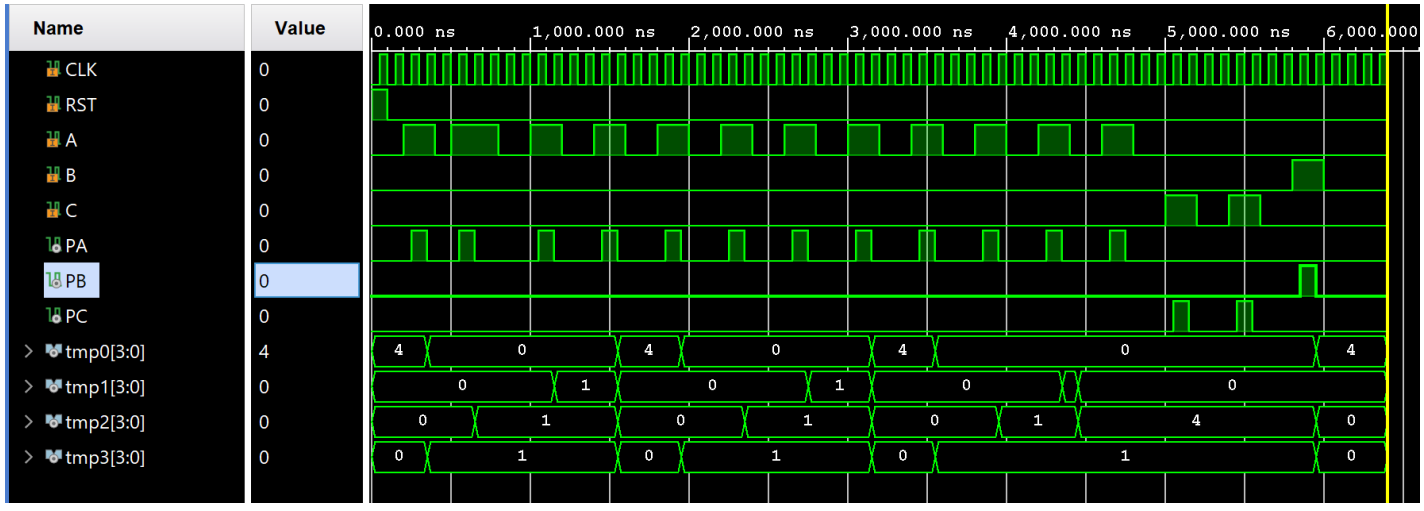


Séquence = INIT => Entré (mauvais code => bon code) => OUVERT => INIT

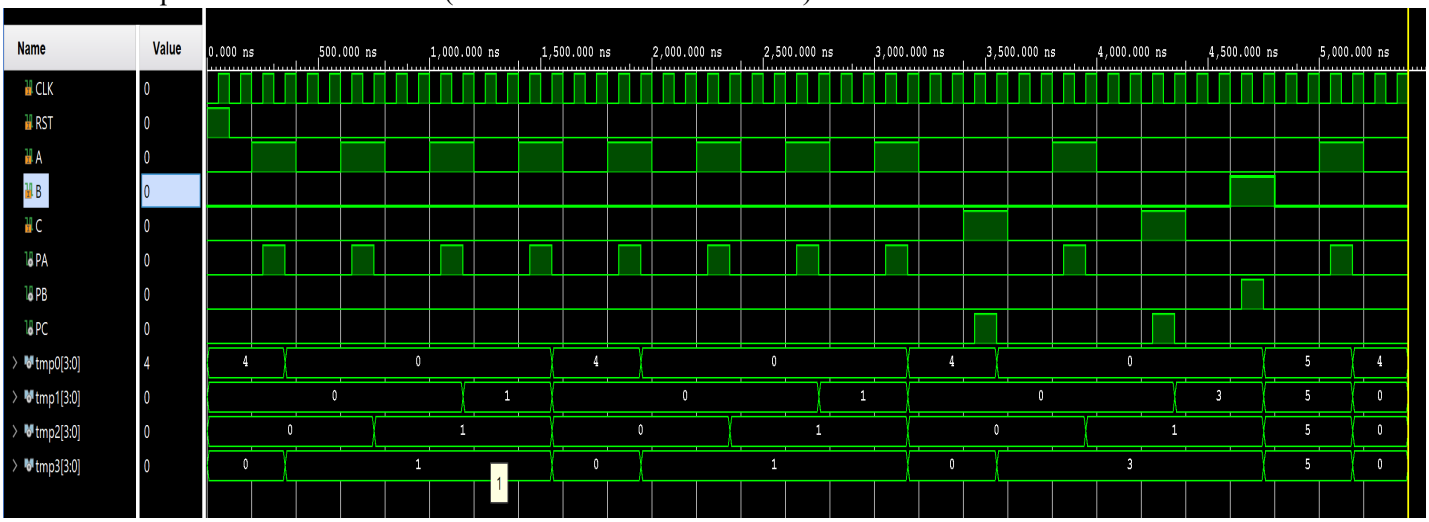


3- Circuit complet (PulseGenerator et Digilock reliée seulement pas les autres modules)

Séquence = INIT => Entré (3xmauvais code) => Alarme => INIT



Séquence = INIT => Entré (2xmauvais code => Bon code) => OUVERT => INIT



c) Observations et validation

Après avoir effectué les simulations ci-dessus, nous pouvons constater que pour le test non-exhaustif (concept expliqué dans la section 2.a) du rapport) de chaque module, les différentes transitions d'état correspondent parfaitement aux résultats théoriques recherchés. Ainsi, puisque tous les modules sont valides, nous pouvons procéder au test non-exhaustif du circuit complet. Encore une fois, dans la simulation, le comportement du circuit complet en fonction de certains scénarios de transitions d'état correspond parfaitement aux résultats théoriques recherchés. Dans cette optique, nous pouvons affirmer que notre circuit est valide, car la simulation de ce dernier respecte les diagrammes d'état décrit définissant notre circuit et concorde parfaitement avec les résultats théoriques recherchés. Par conséquent, nos descriptions VHDL sont valides.

* Hors Contexte

Grand Merci à M. Alexy et M. Jean-Baptiste pour votre aide lors des séances de lab.

On vous souhaite un excellent été!

Sincèrement,

Hamza Boukaftane

Arman Lidder

Index des commentaires

4.1 Ici la séquence CCB ne permet pas de sortir de l'alarme, pourtant elle contient "CB"

5.1 Ici vous restez dans E1 si I = 0, donc ca ne corresponde pas a votre diagrame.