

Mini-Projet LU3EE100

Console SU-EE100 sur FPGA

Réalisé par :

- **KADI Koussaila**
- **CHABANE Sabrina**

Groupe B2

Enseignant de TP :

Mr. Orlando Chuquimia Camacho

Ajout d'un module de division d'horloge :

Déscription et simulation de diviseur d'horloge sous Modelsim :

Afin d'obtenir une horloge de fréquence 25MHz on a utilisé un diviseur d'horloge qui permet de diviser sur quatre la fréquence de l'horloge de la carte **Nexys 4** qui est cadencée à 100 MHz.

le code de diviseur d'Horloge : 100 MHz --> 25 MHz « clkdiv »

```
-- Diviseur d'Horloge : 100 MHz --> 25 MHz

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity ClkDiv is
port(clk100 , reset : in std_logic;  -- Horloge 100 Mhz et Reset Asynchrone
      clk25 : out std_logic );   -- Horloge 25 MHz
end ClkDiv;

architecture archi of ClkDiv is

-- Compteur pour Horloge 25 MHz
signal cpt : std_logic_vector(1 downto 0);

-- Signal Tampon pour l'horloge 25 MHz
signal clk25M : std_logic := '0';

begin
-- GESTION DES COMPTEURS DE DIVISION
--          ET GENERATION DE L'HORLOGE 25 MHz
process(clk100,reset)
begin
  if reset='0' then clk25M<='0'; cpt<="01";
  elsif rising_edge(clk100) then
    cpt<=cpt+1 ;
    if cpt="10" then clk25M<= not clk25M; cpt<="01";
    end if;
  end if;
end process;

-- Affectation Horloge 25 MHz
clk25<=clk25M;
end archi;
```

Le code de testbench tb_clkdiv.vhd : tester le bon fonctionnement de diviseur d'horloge 25MHz.

```
-- Company: UPMC
-- Engineer: Julien Denoulet
-- Testbench Diviseur d'Horloge
--

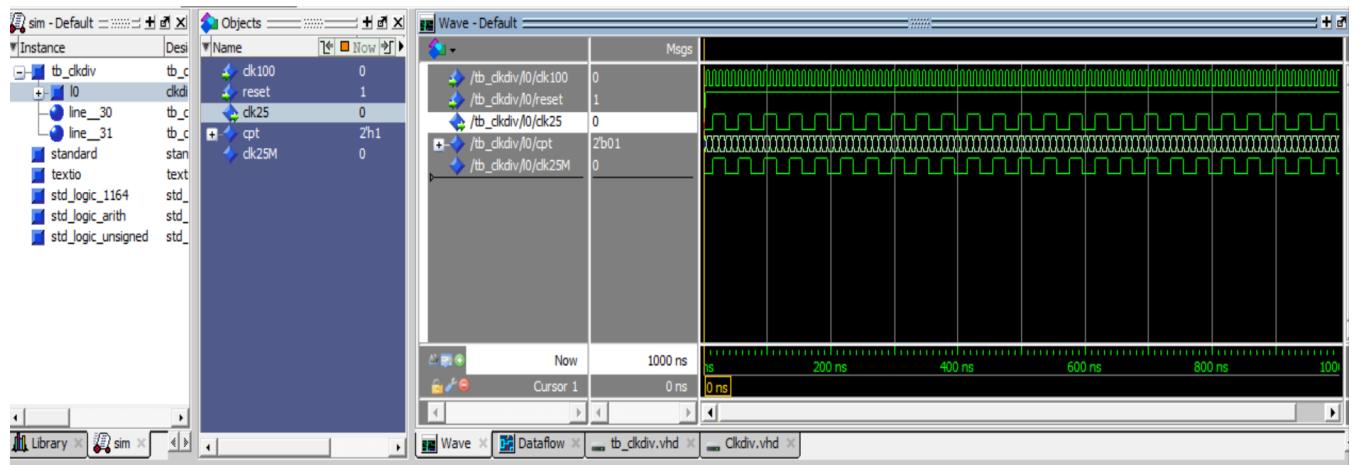
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity TB_ClkDiv is
end TB_ClkDiv;

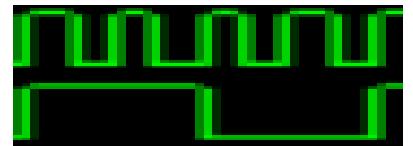
architecture Behavioral of TB_ClkDiv is
signal H_100, RAZ: std_logic:='0';
signal H_25: std_logic;
begin
begin
-- Instanciation Diviseur d'horloge
10: entity work.ClkDiv
    port map(
        clk100  => H_100,
        reset   => RAZ,
        clk25   => H_25);

-- Génération des Signaux d'Entrée
H_100<=not H_100 after 5 ns;
RAZ <='1' after 3 ns;
end Behavioral;
```

La simulation : Vérifier le bon fonctionnement du module de division d'horloge.



On faisant le zoom sur le chronogramme obtenue lors de la simulation on constate que la période de l'horloge de la carte **Nexys 4** qui est de 10 ns est multiplié par quatre afin d'obtenir une période de 40 ns qui correspond à une fréquence de 25MHz.



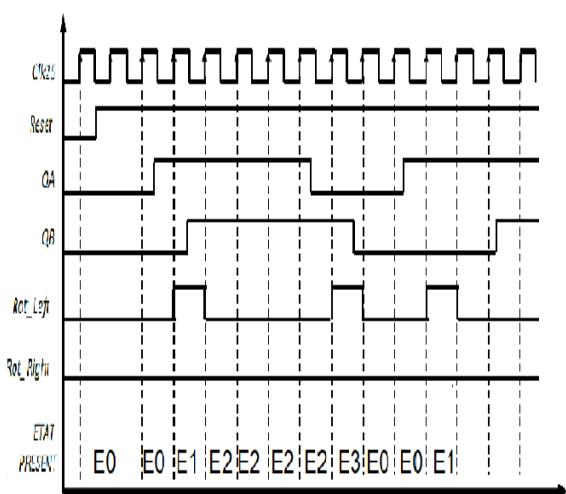
DEPLACEMENT DE LA RAQUETTE AVEC L'ENCODEUR ROTATIF

cette partie consiste à réaliser une IP (Intellectual Property) permettant de gérer l'encodeur rotatif connecté à la carte Nexys 4 et ainsi effectuer les déplacements de la raquette du jeu Casse-Briques.

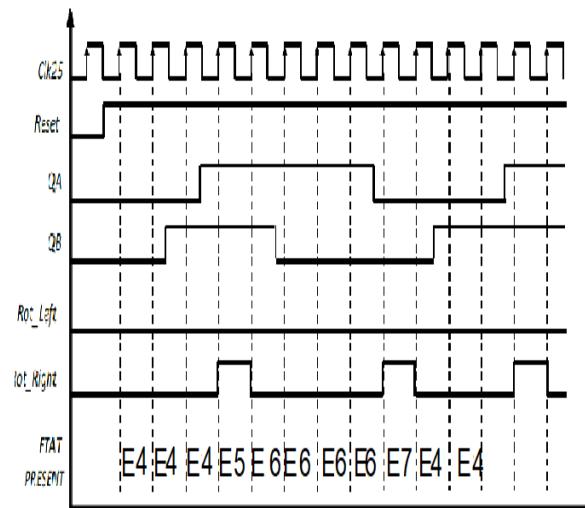
Développement de la machine à états de l'IP Rotary :

les chronogrammes :

Les états successives sur le Rot_left

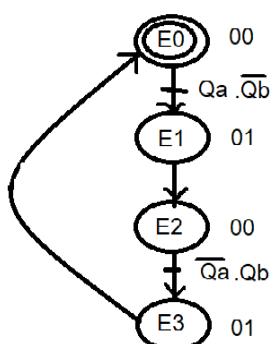


Les états successives sur le Rot_right



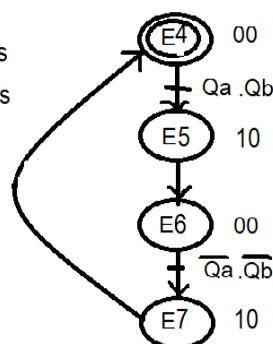
✓ **les graphes d'états :**

sortie=[Rot_right Rot_left]



E0=E4 les mêmes sorties
E2=E6 les mêmes sorties

sortie=[Rot_right Rot_left]



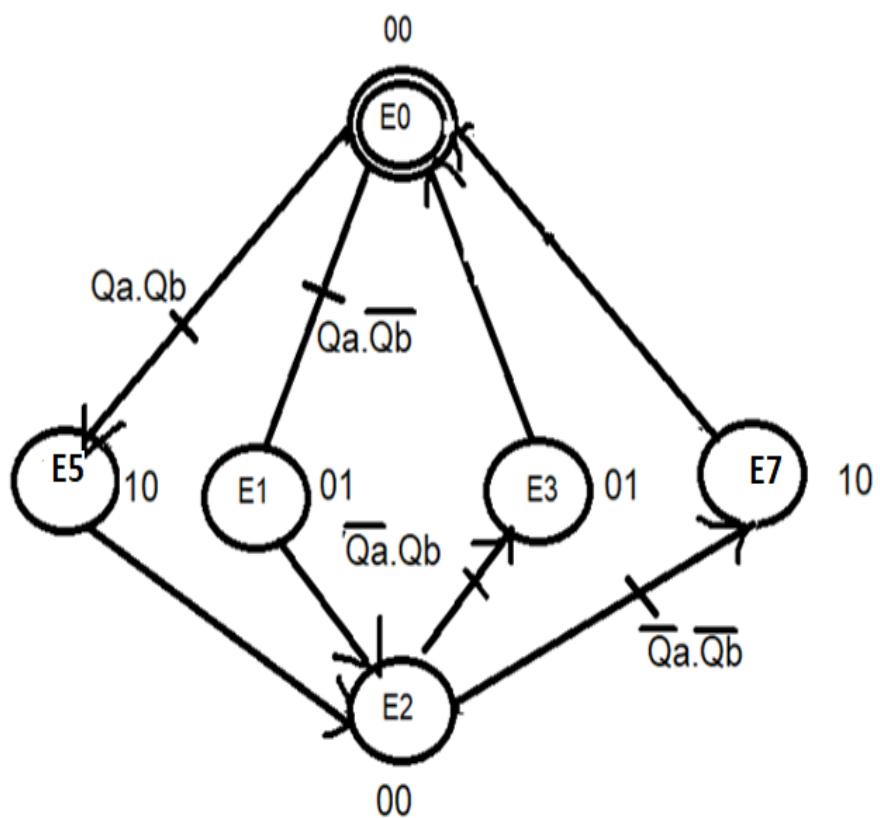
Machine à états de Rot-Left

Machine à états de Rot_Right

✓ Le graphe d'états globale « graohe d'états du module Move »

graphe d'état du
module Move

sortie= [Rot_right Rot_left]



□

Déscription et simulation du graphe d'états « module Move » sous Modelsim :

Le code de MAE du module Move « move.vhd » :

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity move is
port (clk25,reset: in std_logic;
      qa,qb: in std_logic;
      rot_left,rot_right:out std_logic
      );
end move;

architecture archi of move is
--définition des états
type etat is (E0,E1,E2,E3,E5,E7);
signal EF,EP:etat;
signal r,l : std_logic;
begin
--process du registre d'état
process(clk25,reset)
begin
if reset='0' then EP<=E0;
elsif rising_edge(clk25) then
EP<=EF;
end if;
end process;
--combinatoire des états et des sorties
process(qa,qb,EP)
begin
case (EP) is
when E0 => r<='0';l<='0';
if qa='1' then
      if qb='1' then EF<=E5;
      else EF<=E1;
      end if;
      else EF<=E0;
      end if;
when E1 => r<='0';l<='1';
EF<=E2;
when E2 => r<='0';l<='0';
if qa='0' then
      if qb='1' then EF<=E3;
      else EF<=E7;
      end if;
      else EF<=E2;
      end if;
when E3 => r<='0';l<='1';
EF<=E0;
when E5 => r<='1';l<='0';
EF<=E2;
when E7 => r<='1';l<='0';
EF<=E0;
end case;
end process;
rot_left<=l;
rot_right<=r;
end archi ;
```

Testbench de La MAE de module Move « test_move.vhd » :

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity test_move is

end test_move;

architecture archi of test_move is
signal clk25,reset: std_logic :='0';
signal qa,qb: std_logic;
signal rot_left,rot_right :std_logic;
signal xa,xb: std_logic;
begin
begin
instance1: entity work.move
    port map(clk25,reset,qa,qb,rot_left,rot_right);

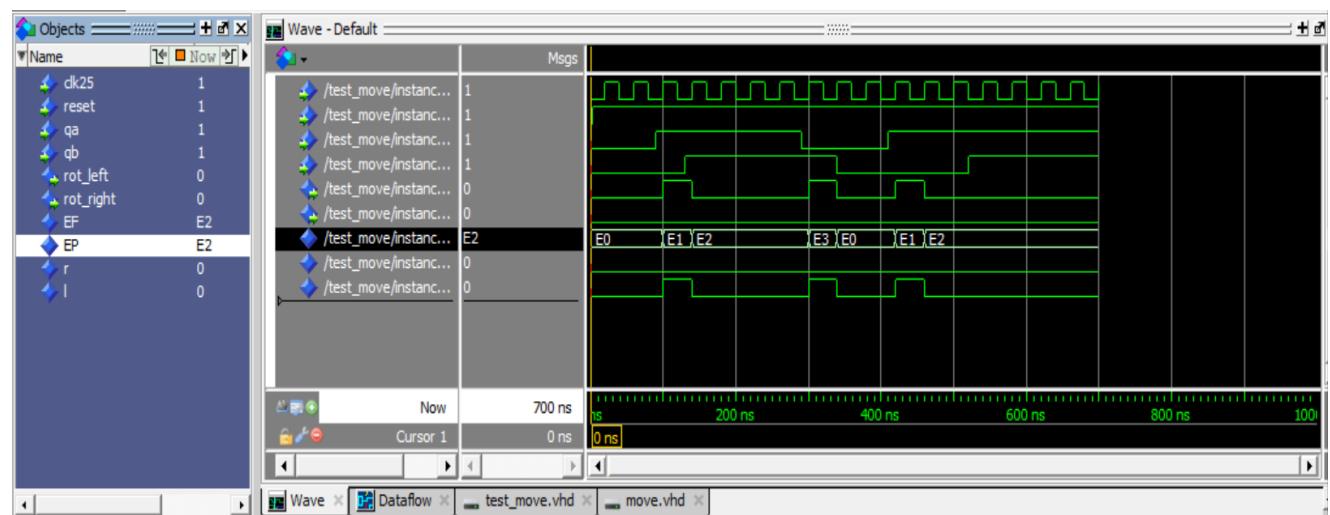
--instance2: entity work.move
--    port map(clk25,reset,xa,xb,rot_left,rot_right);

clk25<=not clk25 after 20 ns ;
reset<='1' after 3 ns;
|
--test rot_left comme sur la fiche de TP
qa<='0','1' after 90 ns , '0' after 290 ns , '1' after 410 ns;
qb<='0','1' after 130 ns, '0' after 340 ns , '1' after 520 ns;

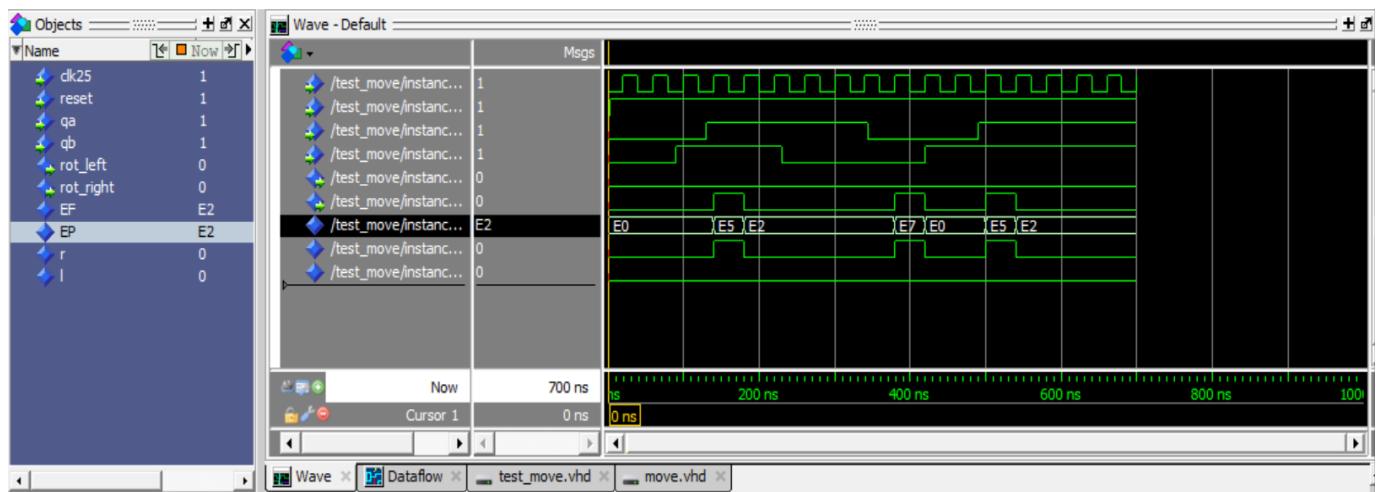
--test rot_right comme sur la fiche de TP
--xa<='0','1' after 130 ns , '0' after 345 ns , '1' after 490 ns;
--xb<='0','1' after 90 ns, '0' after 230 ns , '1' after 420 ns;
end archi;
```

La simulation : Vérifier le bon fonctionnement du module de Move.

Chronogramme correspond à une rotation à gauche du codeur « Rot-Left » :



Chronogramme correspond à une rotation à droite du codeur « Rot-Reight » :



Instanciation de la MAE de move.vhd dans le fichier ip_rotary.vhd :

```
-- Generation des Commandes de Rotation
-----
-- REMPLACER CES 2 INSTRUCTIONS PAR L'INSTANCE DU MODULE MOVE --
instance: entity work.move(archi)
port map(
    clk25 => clk25,
    reset => reset,
    qa => qa,
    qb => qb,
    rot_left => rot_left,
    rot_right=> rot_right);
```

Testbench de module Ip_Rotary : création d'un testbench VHDL afin de tester le bonne fonctionnement de module Ip_Rotary.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity test_move is
end test_move;

architecture archi of test_move is
signal clk25,reset: std_logic :='0';
signal qa,qb: std_logic;
signal rot_left,rot_right :std_logic;
signal xa,xb: std_logic;
begin
--instance1: entity work.move
    -- port map(clk25,reset,qa,qb,rot_left,rot_right);

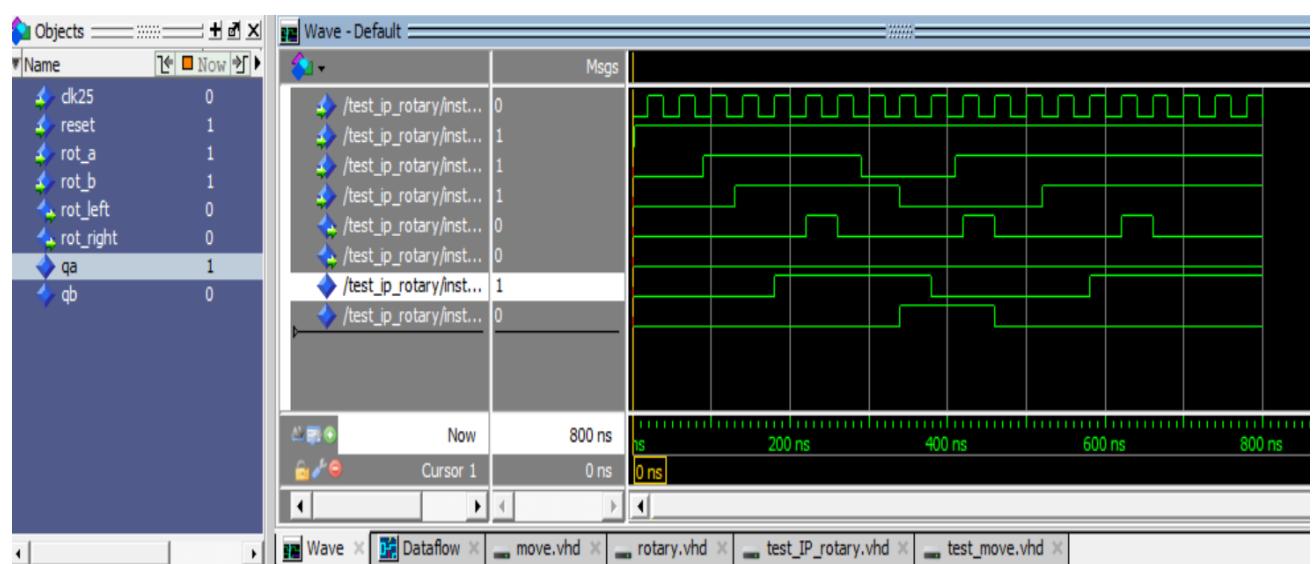
instance2: entity work.move
    port map(clk25,reset,xa,xb,rot_left,rot_right);

clk25<=not clk25 after 20 ns ;
reset<='1' after 3 ns;

--test rot_left comme sur la fiche de TP
--qa<='0','1' after 90 ns , '0' after 290 ns , '1' after 410 ns;
--qb<='0','1' after 130 ns, '0' after 340 ns , '1' after 520 ns;

--test rot_right comme sur la fiche de TP
xa<='0','1' after 130 ns , '0' after 345 ns , '1' after 490 ns;
xb<='0','1' after 90 ns, '0' after 230 ns , '1' after 420 ns;
end archi;
```

La simulation : Vérifier le bon fonctionnement du module de Ip_Rotary .

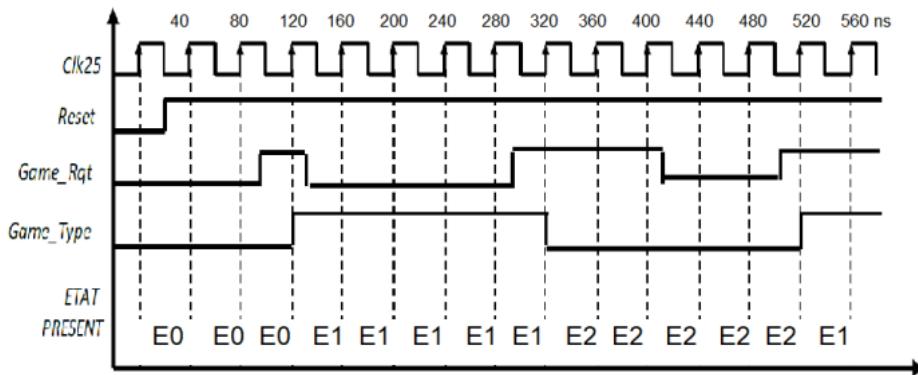


SELECTION DU JEU

Nous allons à présent implémenter la fonction permettant de choisir le jeu de la console SU-EE100 (Pong ou CasseBriques).

Développement de la machine à états de Module Game Manager:

Le chronogramme :

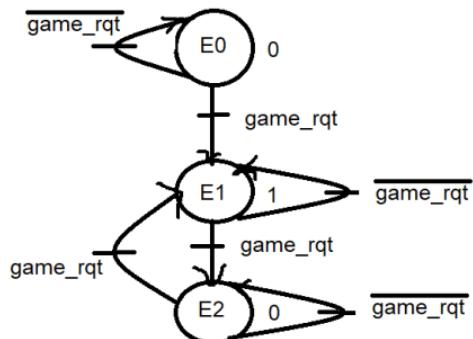


Le graphe d'états « graohe d'états du module Game Manager » :

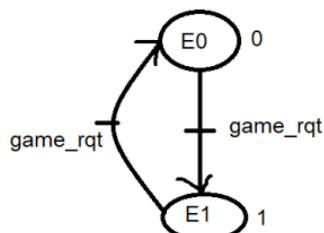
ci-dessous les deux possibilités de la machine à états qu'on peut associer au module **Game Manager**.

Dans notre programme VHDL et pour la suite de notre développement, on a choisie la deuxième possibilité qui correspond à deux états E0 et E1 sachant qu'on obtient les même résultats si on utilise la première proposition .

✓ Première possibilité :



✓ Deuxième possibilité



Déscription et simulation du graphe d'états « module Game Manager » sous Modelsim :

Le code de MAE du module Game Manager « game_mgr.vhd » :

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity game_mgr is
port(clk25,reset: in std_logic;
      game_rqt: in std_logic;
      game_type: out std_logic );
end game_mgr ;

architecture archi of game_mgr is
-- définition des état:
type etat is (E0,E1);
signal EP,EF: etat ;
signal sortie: std_logic;
begin

--- signal de sortie
game_type<=sortie;

--registre d'état
process(clk25,reset)
begin
if reset='0' then EP<=E0;
elsif rising_edge(clk25) then
    EP<=EF;
end if;
end process;
--combénatoires des états et sorties
process(game_rqt,EP)
begin
case (EP) is

when E0 => sortie<='0';
            if rising_edge(game_rqt) then EF<=E1;

            if game_rqt'event and game_rqt='0' then EF<=E0;
            end if;
            end if;

when E1 => sortie<='1';
            if rising_edge(game_rqt) then EF<=E0;
            if game_rqt'event and game_rqt='0' then EF<=E0;
            end if;
            end if;

end case;
end process;

end archi;
```

Testbench de La MAE de module Game Manager « test_game_manager.vhd » :

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity test_game_manager is
end test_game_manager ;

architecture archi of test_game_manager is
signal clk25,reset : std_logic :='0';
signal game_rqt : std_logic;
signal game_type :std_logic;
begin
instance: entity work.game_mgr
    port map(clk25,reset,game_rqt,game_type);

--les signaux
game_rqt <='0','1' after 90 ns, '0' after 130 ns , '1' after 290 ns , '0' after 410 ns, '1' after 500 ns;
clk25 <= not clk25 after 20 ns;
reset<='1' after 3 ns;
end archi;
```

La simulation : Vérifier le bon fonctionnement du module **Game Manager**.



Instanciation de la MAE de game_mgr.vhd dans le fichier game.vhd:

GESTION DES JEUX

Nous allons à présent générer des signaux indiquant quand les parties de Pong ou Casse-Briques sont gagnées ou perdue. Nous allons également faire en sorte que le bouton de l'encodeur rotatif puisse mettre le jeu en pause.

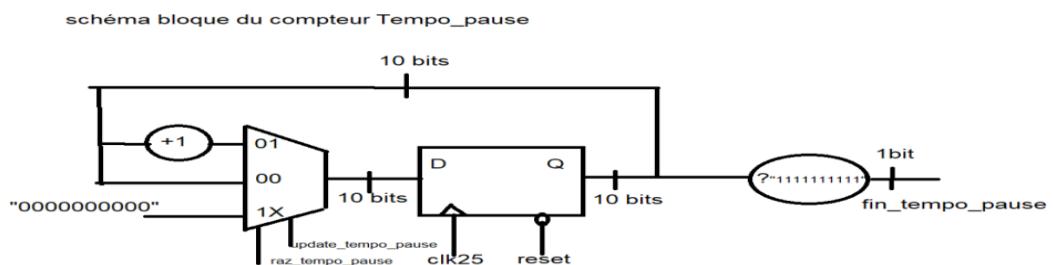
Affectation du bouton central pour la gestion de la pause :

On remplace dans l'instanciation du module Game qui est dans le fichier top.vhd par celle mise en commentaire juste en dessous :

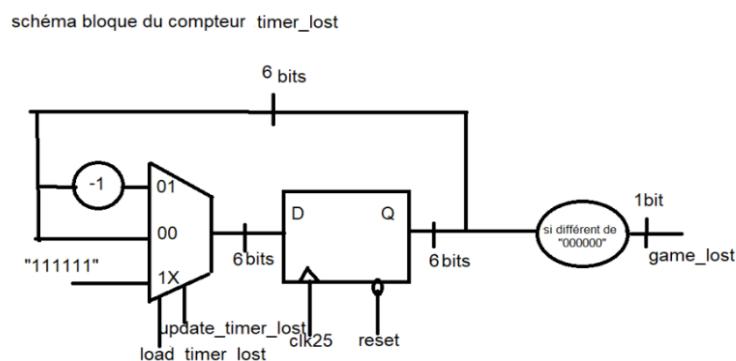
```
-- ****REPLACER L'INSTRUCTION CI-DESSOUS PAR CELLE EN COMMENTAIRE ****
pause_rqt          => pause_rqt,           -- Demande de Pause - Appui sur Bouton Encodeur
--pause_rqt         => 81, ,               -- Demande de Pause - Appui sur Bouton Encodeur
-- ****
```

Les schémas blocs des deux compteurs Tempo_Pause et Timer_Lost :

⊕ Compteur Tempo_Pause :



⊕ Compteur Timer_Lost :



Déscription et simulation des compteurs :

✓ Tempo_Pause :

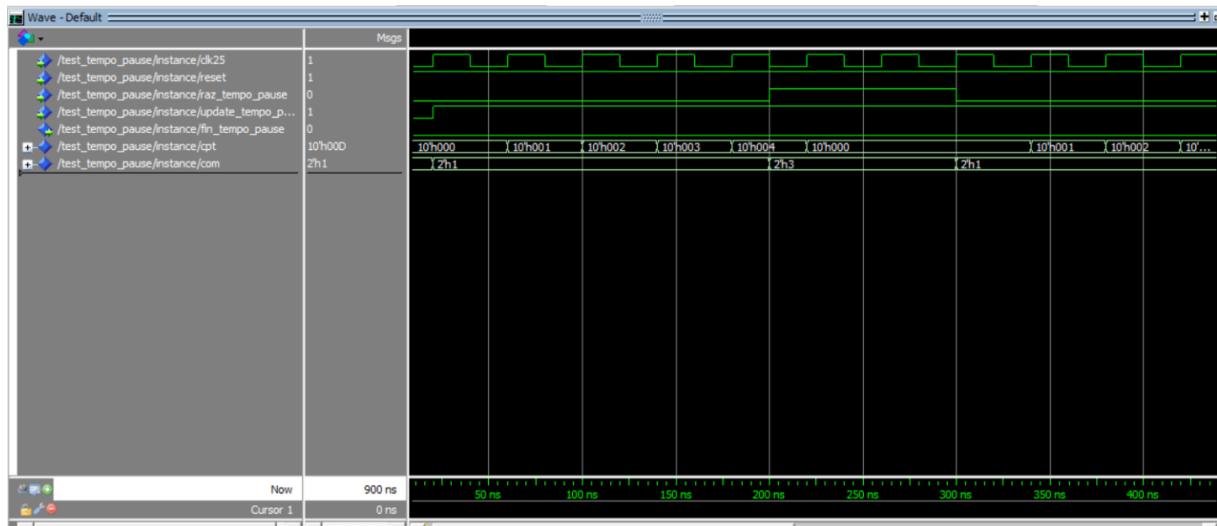
Le Code du Tompo_Pause« tempo_pause.vhd » :

```
Ln# 1 library ieee;
2 use ieee.std_logic_1164.all;
3 use ieee.std_logic_unsigned.all;
4
5 entity tempo_pause is
6 port(clk25,reset: in std_logic;
7      raz_tempo_pause:in std_logic;
8      update_tempo_pause: in std_logic;
9      fin_tempo_pause : out std_logic );
10 end tempo_pause ;
11
12 architecture archi of tempo_pause is
13 signal cpt: std_logic_vector(9 downto 0) ;
14 signal com :std_logic_vector(1 downto 0);
15
16 begin
17 -- definition de com
18 com<=raz_tempo_pause&update_tempo_pause;
19 --le registre du compteur
20 process(clk25,reset,com)
21 begin
22 if reset='0' then cpt<=(others=>'0');
23 elsif rising_edge(clk25) then
24
25 case (com) is
26 when "00" => cpt<=cpt;
27 when "01" => cpt<=cpt+1;
28 when "10" => cpt<=(others=>'0');
29 when "11" => cpt<=(others=>'0');
30 when others => null;
31 end case;
32 end if;
33 fin_tempo_pause<=cpt(9) and cpt(8) and cpt(7) and cpt(6) and cpt(5) and cpt(4) and cpt(3) and cpt(4) and cpt(2) and cpt(1) and cpt(0);
34 end process;
35
36 end archi;
```

Testbench du Tompo_Pause « test_tempo_pause.vhd »

```
Ln# 1 library ieee;
2 use ieee.std_logic_1164.all;
3 use ieee.std_logic_unsigned.all;
4
5 entity test_tempo_pause is
6 end test_tempo_pause ;
7
8 architecture archi of test_tempo_pause is
9 signal clk25,reset: std_logic := '0';
10 signal raz_tempo_pause : std_logic;
11 signal update_tempo_pause : std_logic;
12 signal fin_tempo_pause :std_logic;
13 signal count : std_logic_vector(9 downto 0) ;
14 begin
15 instance: entity work.tempo_pause
16     port map(clk25,reset,raz_tempo_pause,update_tempo_pause,fin_tempo_pause);
17
18 clk25<=not clk25 after 20 ns;
19 reset<='1' after 5 ns;
20 raz_tempo_pause<='0','1' after 200 ns, '0' after 300 ns;
21 update_tempo_pause<='0', '1' after 20 ns , '0' after 500 ns, '1' after 600 ns;
22 end archi;
```

La simulation : Vérifier le bon fonctionnement du **Tompo_Pause** :



✓ **Timer_Lost :**

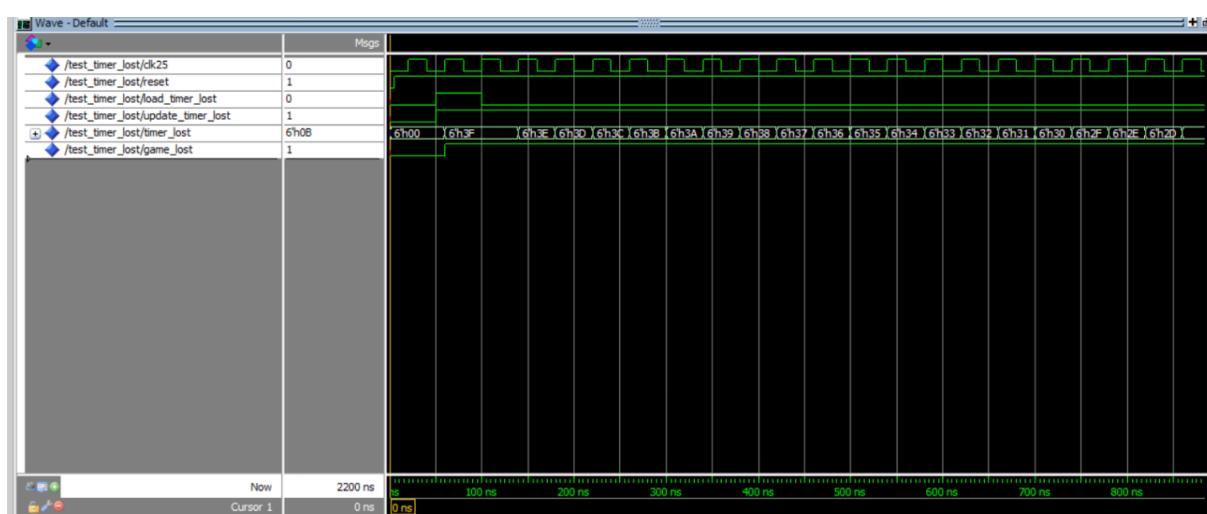
Le Code du Timer_Lost »timer_lost.vhd » :

```
Ln#  
1  library ieee;  
2  use ieee.std_logic_1164.all;  
3  use ieee.std_logic_unsigned.all;  
4  
5  entity timerlost is  
6  port (clk25,reset: in std_logic;  
7        load_timer_lost : in std_logic;  
8        update_timer_lost : in std_logic;  
9        timer_lost : out std_logic_vector(5 downto 0);  
10       game_lost : out std_logic );  
11  end timerlost ;  
12  architecture archi of timerlost is  
13  signal cpt : std_logic_vector(5 downto 0);  
14  signal com : std_logic_vector(1 downto 0);  
15  begin  
16  
17  --identification de la condition com  
18  com<=load_timer_lost&update_timer_lost;  
19  --registre du compteur  
20  process(clk25,reset)  
21  begin  
22  if reset='0' then cpt<=(others=>'0');  
23  elsif rising_edge(clk25) then  
24  case(com) is  
25    when "00" => cpt<=cpt;  
26    when "01" => cpt<=cpt-1;  
27    when "10" => cpt<="111111";  
28    when "11" => cpt<="111111";  
29    when others => null;  
30  end case;  
31  end if;  
32  end process;  
33  timer_lost<=cpt;  
34  game_lost<= not ((not cpt(5)) and (not cpt(4)) and (not cpt(3)) and (not cpt(2)) and (not cpt(1)) and (not cpt(0)));  
35  end archi;
```

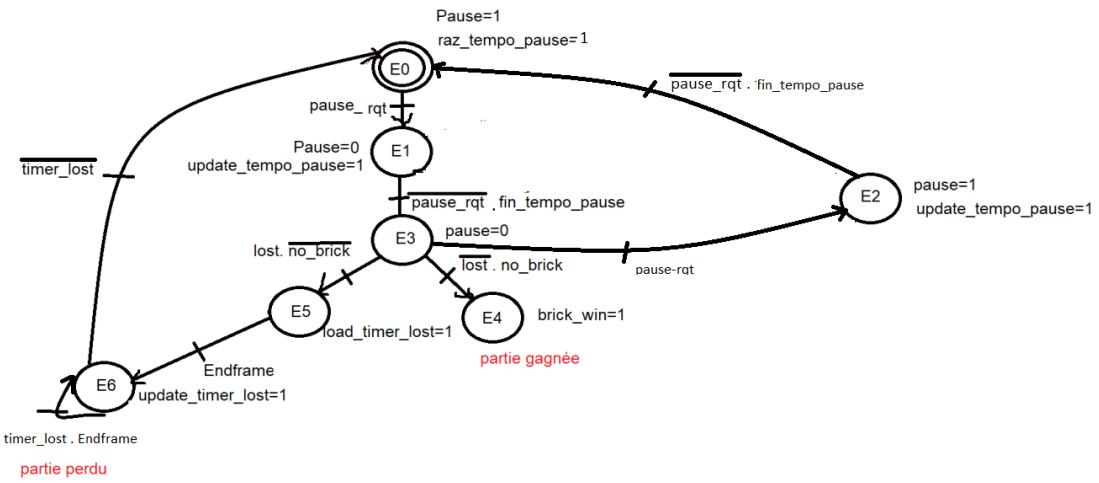
Testbench du Timer_lost « test_timer_lost.vhd »

```
Ln# |  
1  library ieee;  
2  use ieee.std_logic_1164.all;  
3  use ieee.std_logic_unsigned.all;  
4  
5  entity test_timer_lost is  
6  end test_timer_lost ;  
7  
8  architecture archi of test_timer_lost is  
9  signal clk25,reset: std_logic :='0' ;  
10 signal load_timer_lost : std_logic;  
11 signal update_timer_lost: std_logic;  
12 signal timer_lost: std_logic_vector(5 downto 0);  
13 signal game_lost: std_logic;  
14 begin  
15 instance: entity work.timerlost  
16     port map (clk25,reset,load_timer_lost,update_timer_lost,timer_lost,game_lost);  
17  
18 --les signaux d'entrées  
19 load_timer_lost <= '0' , '1' after 50 ns, '0' after 100 ns ;  
20 update_timer_lost<='0' , '1' after 50 ns;  
21 clk25<= not clk25 after 20 ns;  
22 reset<= '1' after 5 ns;  
23 end archi;
```

La simulation : Vérifier le bon fonctionnement du Timer_Lost :



Le graphe d'états de la MAE :



Déscription et simulation du graphe d'états « MAE » sous Modelsim :

Le code de MAE « mae.vhd » :

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity MAE is
port (clk25,reset: in std_logic;
      pause_rqt : in std_logic;
      endframe: in std_logic;
      lost: in std_logic;
      no_brick : in std_logic;
      fin_tempo_pause: in std_logic;
      timer_lost: in std_logic_vector(5 downto 0);

      brick_win: out std_logic; ---a
      pause: out std_logic; ---b
      update_tempo_pause: out std_logic; ---c
      raz_tempo_pause: out std_logic; ---d
      load_timer_lost: out std_logic; ---e
      update_timer_lost: out std_logic ---f
      );
end MAE;

architecture archi of MAE is
--declaration des etats
type etat is (E0,E1,E2,E3,E4,E5,E6);
signal EF,EP :etat;
signal a,b,c,d,e,f : std_logic;
begin
  --registre d'etat
  process(clk25,reset)
  begin
    if reset='0' then EP<=E0;
    elsif rising_edge(clk25) then
      EP<=EF;
    end if;
  end process;
  
```

```

-- combinatoires des entrées sorties:
process(EP,pause_rqt,endframe,lost,no_brick,fin_tempo_pause,timer_lost)
begin
    case(EP) is
        when E0 => a<='0'; b<='1'; c<='0' ; d<='1' ;e<='0';f<='0';
            if pause_rqt='1' then EF<=E1; else EF<=E0; end if;
        when E1 => a<='0'; b<='0'; c<='1' ; d<='0' ;e<='0';f<='0';
            if pause_rqt='0' then
                if fin_tempo_pause='1' then EF<=E3;
                else EF<=E1;
                end if;
            end if;
        when E2 => a<='0'; b<='1'; c<='1' ; d<='0' ;e<='0';f<='0';
            if pause_rqt='0' then
                if fin_tempo_pause='1' then EF<=E0;
                else EF<=E2;
                end if;
            end if;
        when E3 => a<='0'; b<='0'; c<='0' ; d<='0' ;e<='0';f<='0';
            if pause_rqt='1' then EF<=E2; end if;
            if lost='0' then
                if no_brick='1' then EF<=E4 ;
                else EF<=E3;
                end if;
            end if;
            if lost='1' then
                if no_brick='0' then EF<=E5 ;
                else EF<=E3;
                end if;
            end if;
        when E4 => a<='1'; b<='0'; c<='0' ; d<='0' ;e<='0';f<='0';
            EF<=E4;
        when E5 => a<='0'; b<='0'; c<='0' ; d<='0' ;e<='1';f<='0';
            if endframe='1' then EF<=E6;

                else EF<=E5;
            end if;
        when E6 => a<='0'; b<='0'; c<='0' ; d<='0' ;e<='0';f<='1';
            if timer_lost="000000" then EF<=E0 ;
            elsif endframe='1' then EF<=E6;
            end if;
    end case;
end process;
brick_win<=a;
pause<=b;
update_tempo_pause<=c;
raz_tempo_pause<=d;
load_timer_lost<=e;
update_timer_lost<=f;
end archi;

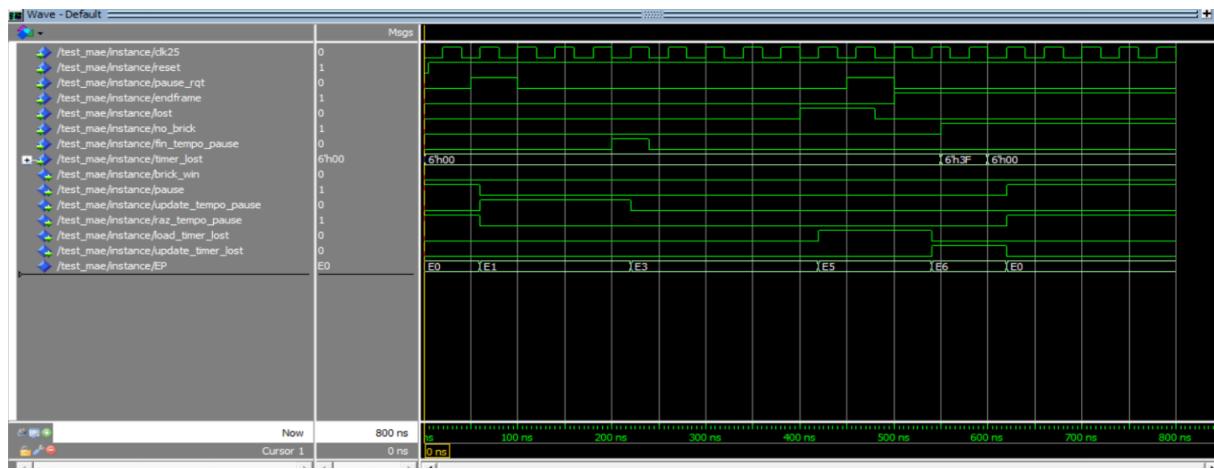
```

Testbench de La MAE « test_mae.vhd » :

```
D:/L3 EEA/L3 s6/3E100/simulation projet/test_MAE.vhd (/test_mae) - Default
Ln# |
```

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.std_logic_unsigned.all;
4
5  entity test_MAE is
6  end test_MAE ;
7
8  architecture archi of test_MAE is
9  --signaux entrées
10 signal clk25,reset: std_logic :='0' ;
11 signal pause_rqt : std_logic;
12 signal endframe : std_logic;
13 signal lost : std_logic;
14 signal no_brick : std_logic;
15 signal fin_tempo_pause : std_logic;
16 signal timer_lost : std_logic_vector(5 downto 0);
17
18 --signaux sorties
19 signal brick_win : std_logic;
20 signal pause : std_logic;
21 signal update_tempo_pause : std_logic;
22 signal raz_tempo_pause : std_logic;
23 signal load_timer_lost : std_logic;
24 signal update_timer_lost : std_logic;
25 begin
26
27 begin
28 instance: entity work.MAE
29     port map (clk25,reset,
30               pause_rqt,
31               endframe,
32               lost,
33               no_brick,
34               fin_tempo_pause,
35               timer_lost,
36
37               brick_win,
38               pause,
39               update_tempo_pause,
40               raz_tempo_pause,
41               load_timer_lost,
42               update_timer_lost );
43
44
45 --les signaux d'entrées
46 pause_rqt<='0', '1' after 50 ns, '0' after 100 ns , '1' after 450 ns, '0' after 500 ns;
47 fin_tempo_pause<='0', '1' after 200 ns, '0' after 240 ns;
48 no_brick<='0', '1' after 550 ns;
49 lost<='0', '1' after 400 ns, '0' after 480 ns;
50 timer_lost<="000000","111111" after 550 ns , "000000" after 600 ns;
51 endframe<='0','1' after 500 ns;
52
53 clk25<= not clk25 after 20 ns;
54 reset<= '1' after 5 ns;
55 end archi;
```

La simulation : Vérifier le bon fonctionnement du **MAE**.



Déscription du graphe d'états et des deux compteurs sous Modelsim :

Le code « mode.vhd » :

Testbench de module Mode « test_mode.vhdl »

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

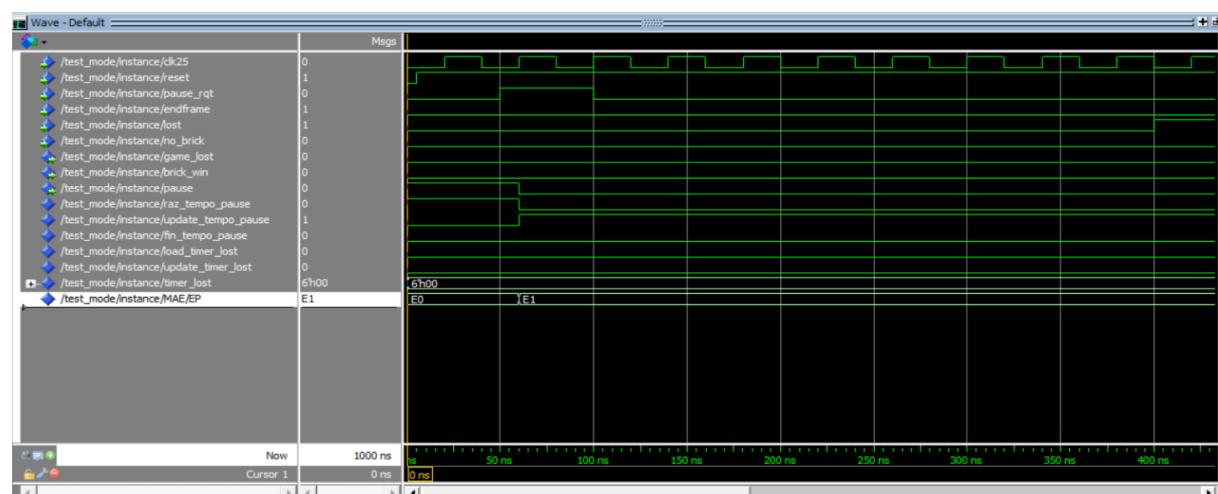
entity test_mode is
end test_mode;

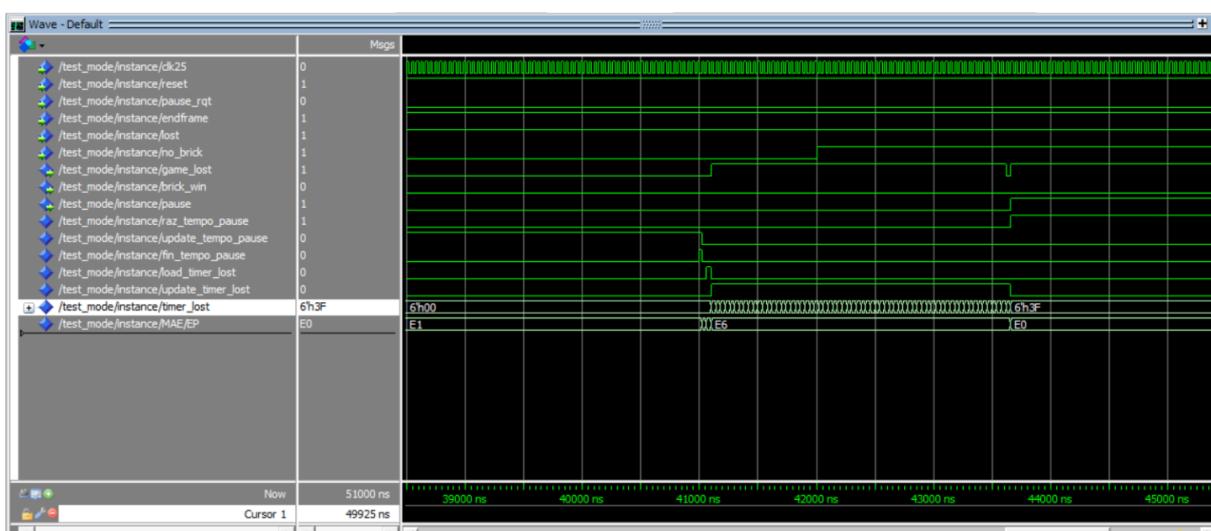
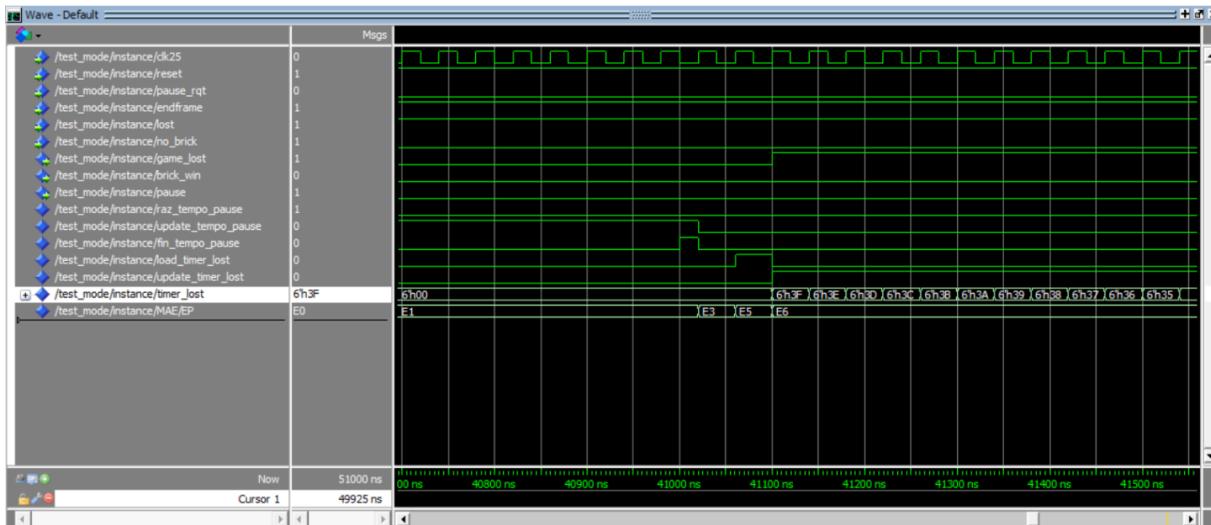
architecture archi of test_mode is
signal clk25,reset: std_logic :='0';
signal pause_rqt: std_logic;
signal endframe: std_logic;
signal lost : std_logic;
signal no_brick : std_logic;
signal game_lost: std_logic;
signal brick_win : std_logic;
signal pause : std_logic ;
begin
--instanciation
instance : entity work.mode
port map(clk25,reset,
         pause_rqt,
         endframe,
         lost,
         no_brick,
         ----- sorties
         game_lost,
         brick_win,
         pause);

--signaux d'entées:
pause_rqt<='0', '1' after 50 ns, '0' after 100 ns , '1' after 450 ns, '0' after 500 ns;
no_brick<='0', '1' after 42000 ns;
lost<='0', '1' after 400 ns, '0' after 50000 ns;
endframe<='0','1' after 500 ns;
clk25<= not clk25 after 20 ns;
reset<= '1' after 5 ns;
end archi;

```

La simulation : Vérifier le bon fonctionnement du module Mode.





On remarque que le signal fin_tempo_pause devient 1 après 41000 ns ce qui va permettre de passer de l'état E1 à l'état E3 comme illustré dans la simulation 2 , dans cette simulation on a choisi de passer de l'état E3 à l'état E5 (on a choisi le chemin perdu pour tester le retour à l'état E0)

On a bien vu que à l'état E6 on va rester dans ce cas pendant 64 images qui passent cela veut dire que le compteur Timer_lost va se décrémenter de 63 vers 0, une fois arriver à 0 en passe à l'état E0 et là on est bien sur que toute les images sont passées avec aussi le signal endframe.