École Polytechnique de Montréal

Département de génie informatique et génie logiciel

INF1500

Logique des systèmes numériques

Laboratoire 5 : Rapport sur ‘‘Bloc heure/minuterie de cuisinière’’

Soumis par

Tamer Arar (1828776)

Éric Chao (1827022)

Section de labo #2

11 avril 2016

Critères	**Points**
Rapport : spécifications Moore/Mealy, schémas et diagrammes d’états des blocs *pulsegen* et *oventimetimerfsm*	/2
Rapport : simulations	/1
Reste du rapport	/1
Sources VHDL des circuits *pulsegen*, *timekeeper*, *timer* et *oventimetimerfsm*	/2
Fonctionnement de l’implémentation (*top*)	/1
Total	/7

Introduction

Ce laboratoire nous permet de nous familiariser avec la programmation en code VHDL en nous permettant de programmer des composants d'un bloc/heure minuterie nous permettant d'afficher sur l'écran LCD du FPGA le temps et une minuterie qui sont ajustables et de simuler un four par l'implémentation de divers composants incluant le bloc/heure minuterie.

Un ensemble de modules en logique séquentiel ont été conçus dans ce laboratoire dont deux où nous avons dû concevoir le diagramme d'états et le tableau de transition en plus du codage en VHDL :

- Le module générateur d'impulsion qui génère une impulsion pendant un cycle lors que le bouton poussoir est appuyé.
- Le module de gardien du temps qui affiche le temps et qui permet de l'ajuster.
- Le module minuterie qui affiche un temps en minutes et secondes qui décrémentent lorsqu'il est initialisé et qui est ajustable.
- La machine à états finis du bloc heure/minuterie relie les modules générateurs d'impulsions, gardien du temps et minuterie en les combinant en une machine à états finis et qui sert de vecteurs entre ces composants et l'afficheur LCD.

Générateur d'impulsion (*pulsegen*)

Ce bloc sert à détecter le signal quand un bouton est enfoncé 100 millions de fois par secondes (100MHz). Dès qu'un bouton enfoncé est détecté, il avertit en envoyant un signal au bloc *oventimetimerfsm* que l'on expliquera après. Toutefois, il existe une condition qui stipule que l'impulsion doit seulement durée 1 cycle. En d'autres mots, quand l'entrée *btn* est à 1, le signal de sortie *pulse* est à 1 également, mais seulement pour une durée de 1/1000000 seconde.

Nous avons tout d'abord conçu son diagramme d'états de Mealy :

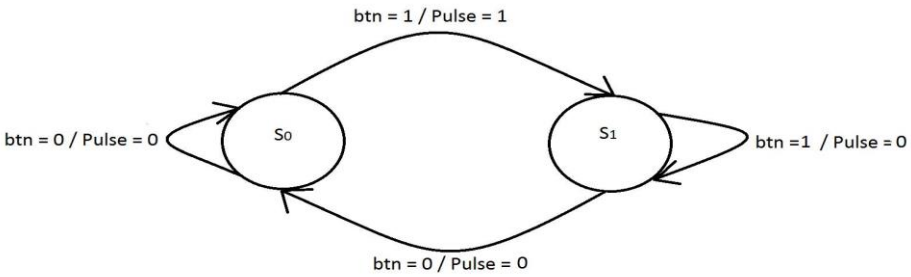

Figure 1 : Machine à états finis de Mealy du bloc *pulsegen*.

Par la suite, en se servant de ce diagramme, nous avons réalisé son tableau d'états :

Tableau I : Table de transitions pour le machine à états finis de Mealy du bloc pulsegen avec *btn* en entrée et *pulse* en sortie (encodage one-hot).

Entrée	btn	
États	**0**	**1**
S_0 => 01	01, 0	10, 1
S_1 => 10	01, 0	10, 1

Ainsi, grâce au diagramme des états et au tableau d'états nous avons pu écrire le code du module :

Start Page **pulsegen.vhd (Text Document)**

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

entity pulsegen is
    port(
        clk : in std_logic;
        btn : in std_logic;
        pulse : out std_logic := '0'
    );
end entity;

architecture a of pulsegen is
    type state_t is (           -- liste des noms d'états ici
        s0,
        s1
    );

begin
    process(clk)
        variable state : state_t := s0;     -- variable contenant l'état en cours et l'état initial

    begin
        if rising_edge(clk) then

            case state is                   -- vérification de l'état à chaque cycle

                when s0=>                   -- état spécifique
                    if btn = '1' then
                        pulse <= '1';
                        state := s1;        -- change d'état au prochain cycle
                    else
                        state := s0;
                    end if;

                when s1=>
                    if btn = '1' then
                        pulse <= '0';
                        state := s1;
                    else
                        state := s0;
                    end if;
            end case;
        end if;
    end process;

end architecture;
```

Figure 2 : Code en langage vhdl du bloc *pulsegen.*

Ensuite, une simulation est effectuée après le codage en VHDL du bloc *pulsegen* (figure 2) pour tester s'il marche :

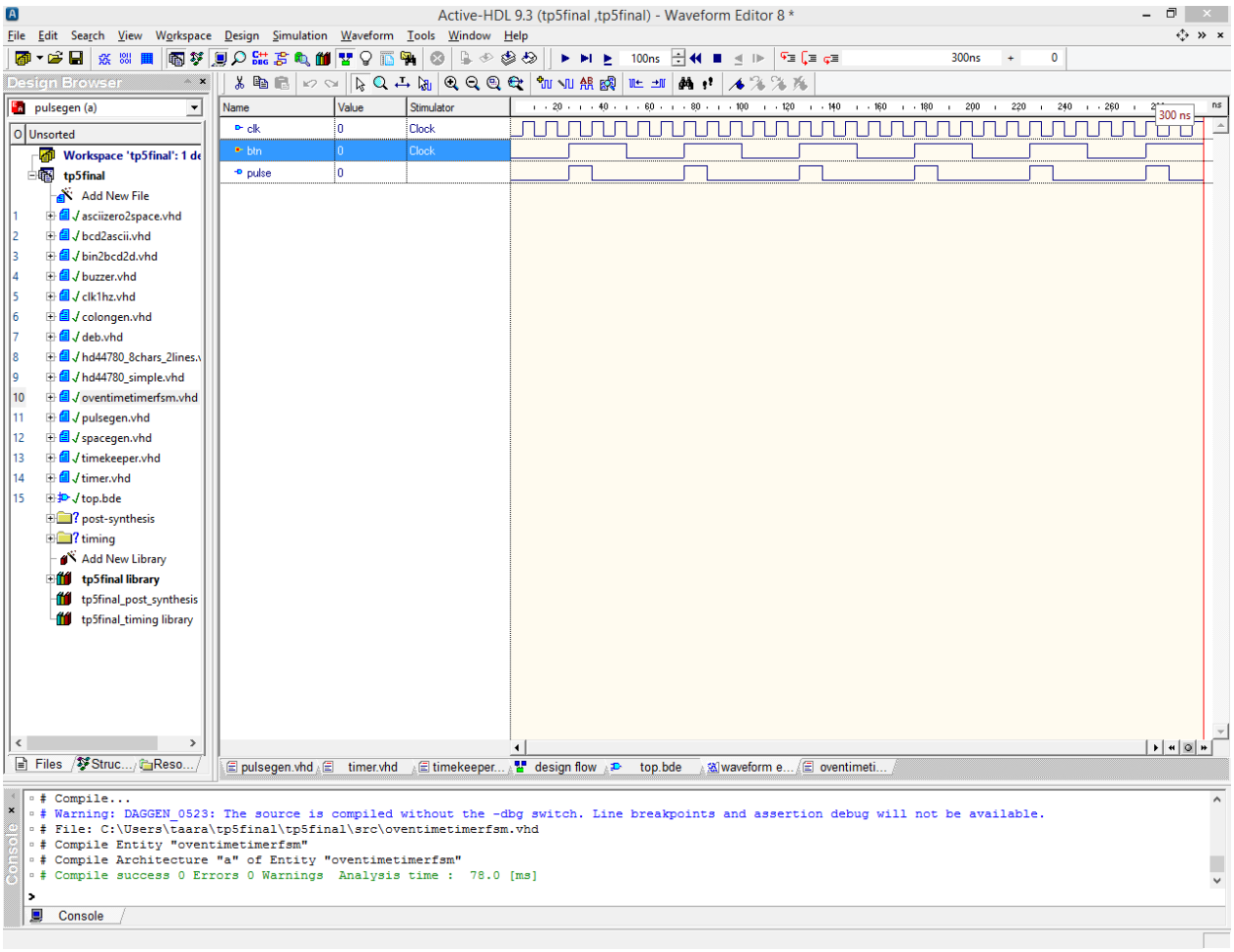

Figure 3 : Simulation du bloc *pulsegen.*

Comme on peut le constater, la simulation est vraie pour la fonctionnalité du code voulu. En effet, le signal *pulse* devient à 1 dès que le signal *btn* est à 1 et pour une durée d'un cycle seulement.

Gardien du temps (*timekeeper*)

Ce bloc sert à montrer et à modifier l'heure actuelle de la minuterie de cuisinière. Il possède une horloge rapide (100 MHz) pour décrémenter l'heure. Il a aussi une horloge plus lente (1 Hz) pour incrémenter le temps grâce aux entrées *set* (permet le changement de l'heure et arrête l'heure actuelle), *pulse_h* (ajoute une valeur en heure), *pulse_m* (ajoute une valeur en minute) et *rst* (réinitialise les heures et les minutes à 0). Ainsi, on obtient trois sorties sur 7 bits : *th* (affiche les heures), *tm* (affiche les minutes) et *ts* (affiche les secondes). Un code en langage VHDL qui permet ces fonctionnalités est alors écrit :

Start Page **timekeeper.vhd (Text Document)**

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

entity timekeeper is
    port(
        clk_100mhz : in std_logic;
        clk_1hz : in std_logic;
        rst : in std_logic;
        pulse_h : in std_logic;
        pulse_m : in std_logic;
        set : in std_logic;
        th : out std_logic_vector(6 downto 0);
        tm : out std_logic_vector(6 downto 0);
        ts : out std_logic_vector(6 downto 0)
    );
end entity;

architecture a of timekeeper is
    signal ch : unsigned(6 downto 0) := (others => '0');
    signal cm : unsigned(6 downto 0) := (others => '0');
    signal cs : unsigned(6 downto 0) := (others => '0');
begin
    th <= std_logic_vector(ch);
    tm <= std_logic_vector(cm);
    ts <= std_logic_vector(cs);

    process(clk_100mhz)
    begin
        if rst = '1' then
            -- reinitialisation asynchrone
            ch <= (others => '0');
            cm <= (others => '0');
            cs <= (others => '0');
        elsif rising_edge(clk_100mhz) then
            -- voir "timer.vhd" pour avoir une idee de la structure ici
            if set = '1' then
                if pulse_h = '1' then
                    ch <= ch + 1 ;
                    if ch = "0010111" then
                        ch <= "0000000";
                        end if;
                end if;
                if pulse_m = '1' then
                    cm <= cm + 1;
                    if cm = "0111011" then
                        cm <= "0000000";
                        ch <= ch + 1;
                        end if;
                end if;
            elsif clk_1hz = '1' and set = '0' then
                cs <= cs + 1;
                if cs = "0111011" then
                    cs <= "0000000";
                    cm <= cm + 1;
                    if cm = "0111011" then
                        cm <= "0000000";
                        ch <= ch + 1;
                        if ch = "0010111" then
                            ch <= "0000000";
                        end if;
                    end if;
                end if;
                end if;
        end if;
    end process;
end architecture;
```

Figure 4 : Code en langage VHDL du bloc *timekeeper*.

Après avoir créé le code, on regarde si le bloc marche à l'aide de plusieurs simulations avec les 2 horloges (clk_100mhz et clk_1hz) à 100 MHz pour des simulations plus rapides :

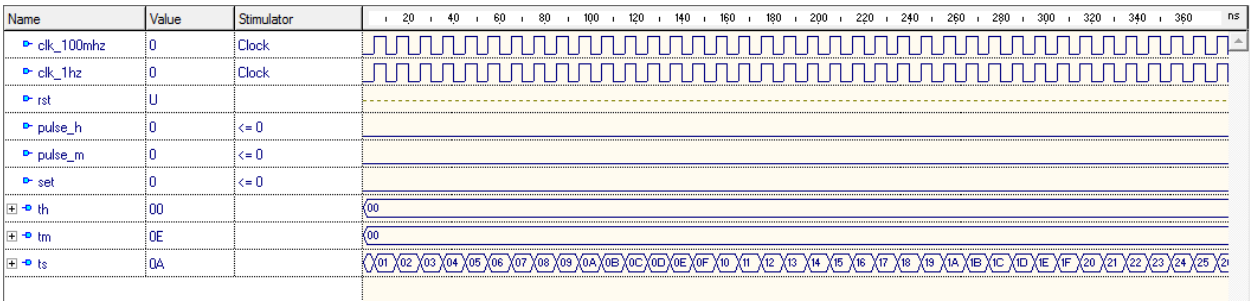

Figure 5 : Première simulation du bloc *timekeeper.*

Dans ce premier cas, les secondes montent graduellement de 1 à chaque cycle. Ainsi, les secondes s'additionnent et le temps progresse.

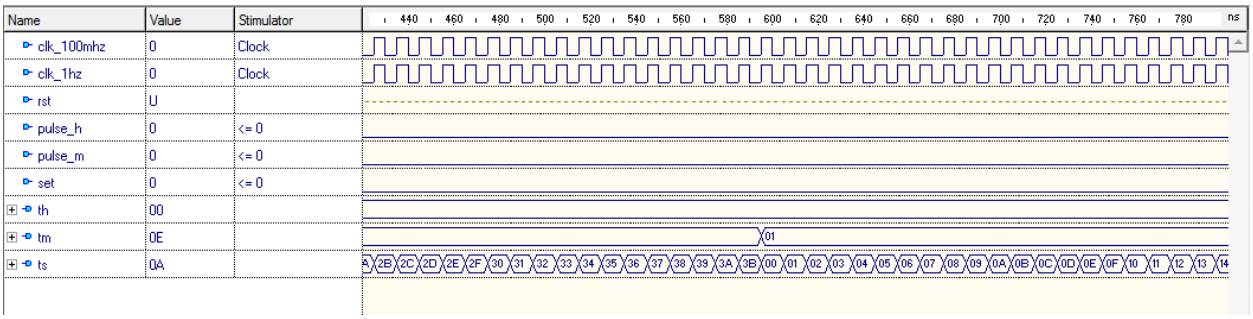

Figure 6 : Deuxième simulation du bloc *timekeeper.*

Dans ce deuxième cas, après 59 secondes, 1 minute est incrémentée et les secondes retombent à 0.

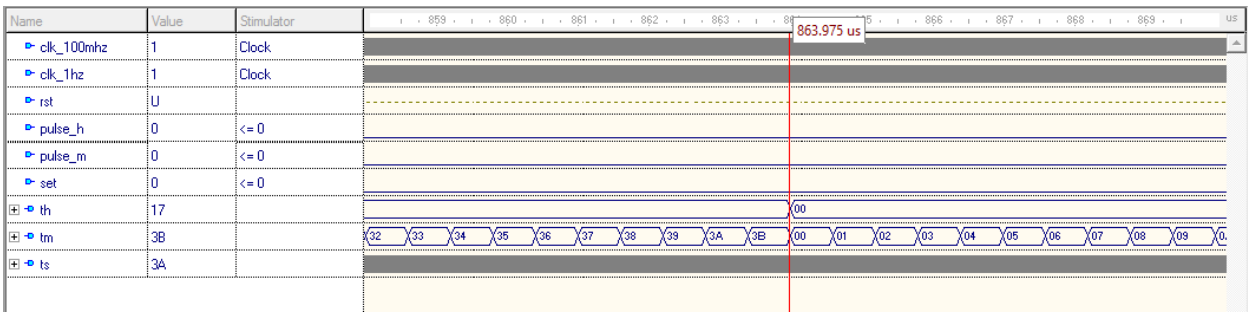

Figure 7 : Troisième simulation du bloc *timekeeper.*

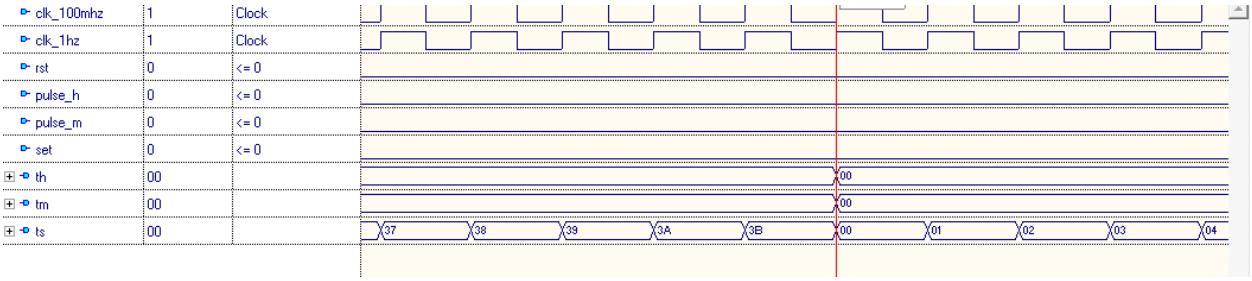

Figure 8 : Troisième simulation du bloc *timekeeper.*

Dans ce troisième cas, après 23 heures, 59 minutes et 59 secondes, l'heure, les minutes et les secondes retombent tous à la valeur 0.

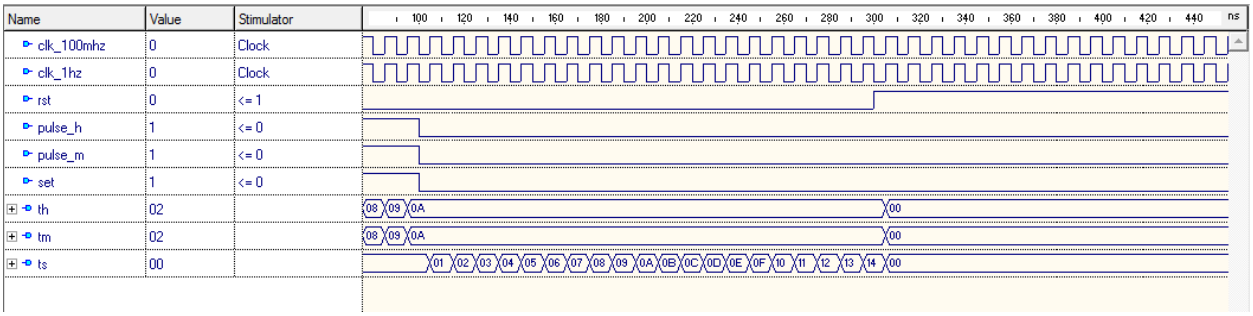

Figure 9 : Quatrième simulation du bloc *timekeeper.*

Dans ce quatrième cas, on aperçoit que lorsque l'entrée *rst* (reset) est à 1, l'heure est réinitialisée (remise à 0 pour les heures, les minutes et les secondes).

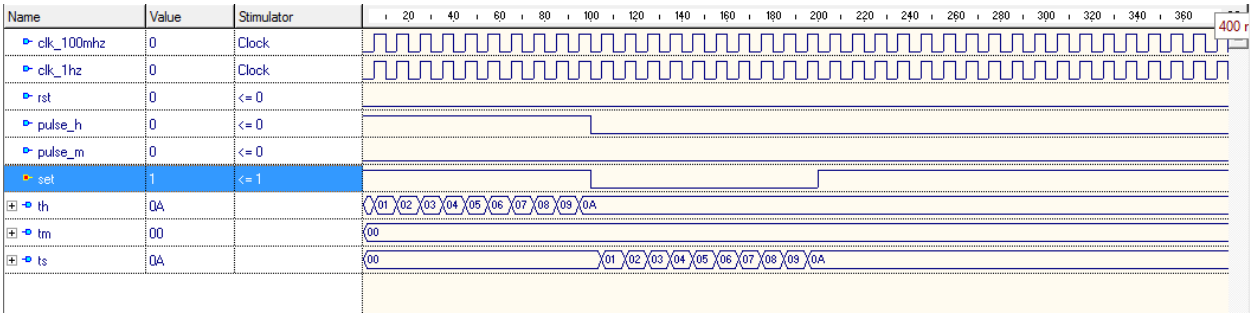

Figure 10 : Cinquième simulation du bloc *timekeeper.*

Aussi, lorsque *set* est activé, l'heure cesse d'avancer…

Minuterie (*timer*)

La minuterie est un module très important, car c'est pendant l'intervalle de temps indiqué par ce dernier que les aliments du four cuisent. Contrairement au gardien du temps, la minuterie a pour fonction de décrémenter le temps plutôt que de l'incrémenter. Ainsi, il affiche sur l'afficheur LCD le temps en minutes avec une limite de 90 minutes et les secondes provenant des sorties tm et ts en binaire qui seront converties afin de pouvoir les afficher. Les minutes sont ajustables en appuyant sur le bouton set, représenté par l'entrée set, qui arrête la mise à jour du temps et en appuyant par la suite sur un bouton poussoir représenté par pulse_m, qui augmente les minutes à chaque poussée. Ainsi, la minuterie décrémente le temps à chaque seconde de l'horloge à un hertz (clk 1 hz) tant que l'activation de la minuterie est active et de plus, nous pouvons l'arrêter et le remettre à zéro en appuyant sur le bouton reset, représenté par l'entrée rst, qui initialise le temps à zéro minute et zéro seconde.

De ce fonctionnement, nous avons programmé ce module en langage VHDL :

Start Page **timer.vhd (Text Document)**

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

entity timer is
    port(
        clk_100mhz : in std_logic;
        clk_1hz : in std_logic;
        rst : in std_logic;
        pulse_m : in std_logic;
        set : in std_logic;
        tm : out std_logic_vector(6 downto 0);
        ts : out std_logic_vector(6 downto 0)
    );
end entity;

architecture a of timer is
    signal cm : unsigned(6 downto 0) := (others => '0');
    signal cs : unsigned(6 downto 0) := (others => '0');
begin
    tm <= std_logic_vector(cm);
    ts <= std_logic_vector(cs);

    process(clk_100mhz)
    begin
        if rst = '1' then                    -- réinitialisation asynchrone
            cm <= (others => '0');
            cs <= (others => '0');

        elsif rising_edge(clk_100mhz) then
            if set = '1' then                -- ajustement de la minuterie
                if pulse_m = '1' then
                    cm <= cm + 1;
                    if cm = "1011010" then
                        cm <= "0000000";
                        end if;
                end if;

            elsif clk_1hz = '1' and set = '0' then
                cs <= cs - 1;                -- décrémenter les secondes à chaque seconde
                if cs = "0000000" then
                    cs <= "0111011";
                    cm <= cm-1;
                    if cm = "0000000" then
                        cs <= "0000000";
                        cm <= "0000000";

                    end if;

                end if;
            end if;
        end if;
    end process;
end architecture;
```

Figure 11 : Code en langage VHDL du bloc *timer*.

Après la programmation du module, nous l'avons simulé afin de tester son fonctionnement en utilisant les 2 horloges (clk_100mhz et clk_1hz) à 100 MHz pour des simulations plus rapides :

Comme nous pouvons le voir dans cette première simulation, lorsque l'entrée set est à un, chaque impulsion de pulse_m augmente les minutes de un.

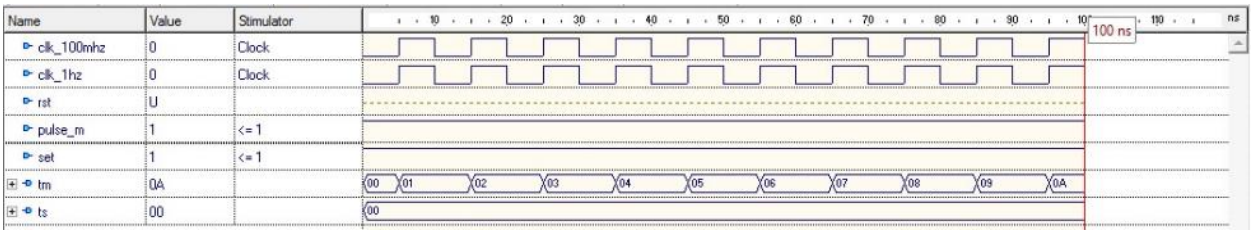

Figure 12 : Première simulation du bloc *timer.*

Dans cette deuxième simulation, nous voyons que lorsque nous initialisons l'entrée set à zéro afin de permettre au temps de se décrémenter, ce dernier se décrémente en effet en laissant les secondes diminuer à chaque cycles en parant de 59 secondes.

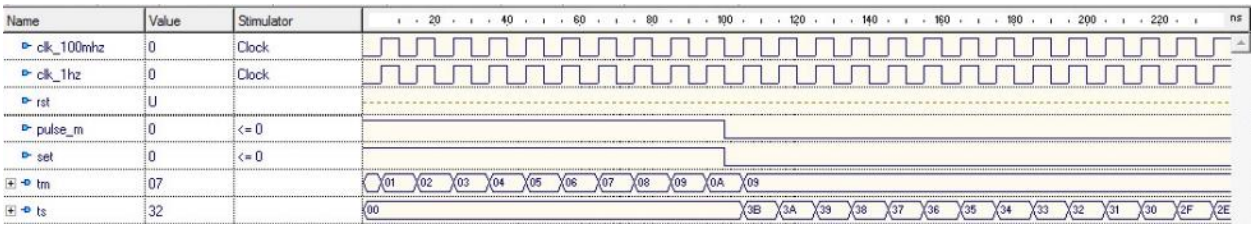

Figure 13 : Deuxième simulation du bloc *timer.*

Dans cette troisième simulation, nous remarquons que lorsque le temps passe d'une minute à une autre lors de son décrémentation, nous pouvons remarquer que les secondes sont à zéro avant de passer à 59 secondes au changement de minute.

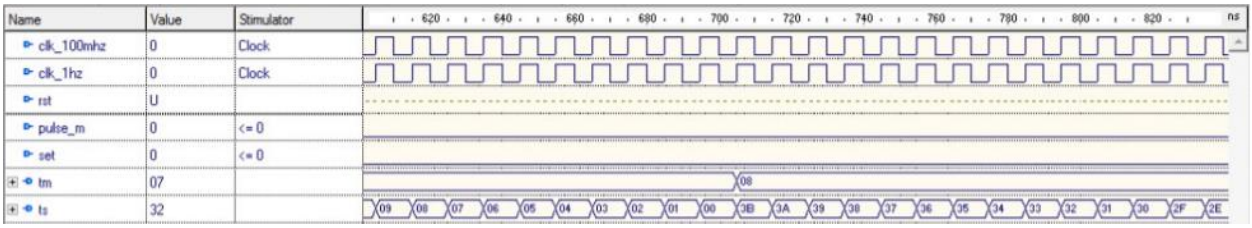

Figure 14 : Troisième simulation du bloc *timer.*

Dans cette quatrième simulation, nous pouvons remarquer que lorsque nous voulons ajuster les minutes par les impulsions de pulse_m et que nous atteignons 90 minutes, ici représenté par 5A en hexadécimal, et que nous appuyons une fois de plus, les minutes passe à zéro.

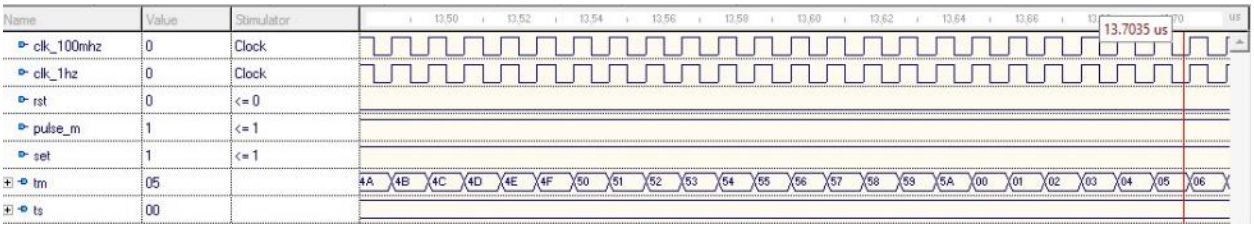

Figure 15 : Quatrième simulation du bloc *timer.*

Dans cette cinquième simulation, nous remarquons que l'entrée set est active (set = 1), le temps arrête de décrémenter.

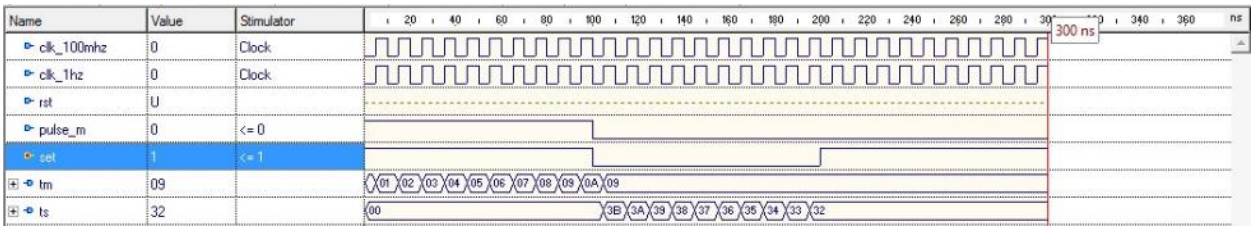

Figure 16 : Cinquième simulation du bloc *timer.*

Dans cette sixième simulation, nous pouvons voir que lorsque l'entrée reset (rst) est active (rst = 1), le temps est initialisé à zéro.

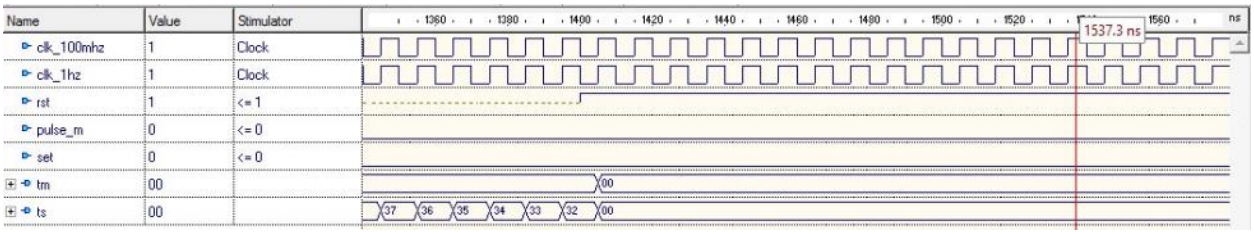

Figure 17 : Sixième simulation du bloc *timer.*

Dans cette septième simulation, nous pouvons voir que lorsque le temps atteint zéro (les minutes et les secondes sont à zéro), le temps reste à zéro. Cette figure représente la première partie de la simulation sept qui montre les minutes qui passent à zéro.

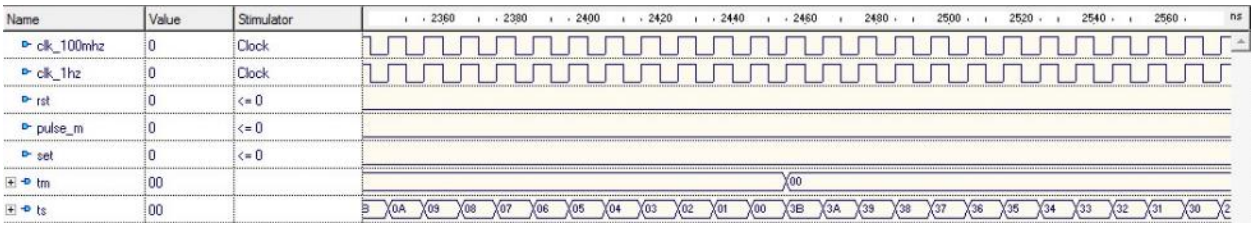

Figure 18 : Septième simulation du bloc *timer.*

Suite de la figure précédente, cette figure montre lorsque les secondes atteignent zéros, ainsi puisque c'est la suite de la figure précédente, les minutes et les secondes sont ainsi à zéro et continue de l'être tant que la minuterie n'est pas réajustée.

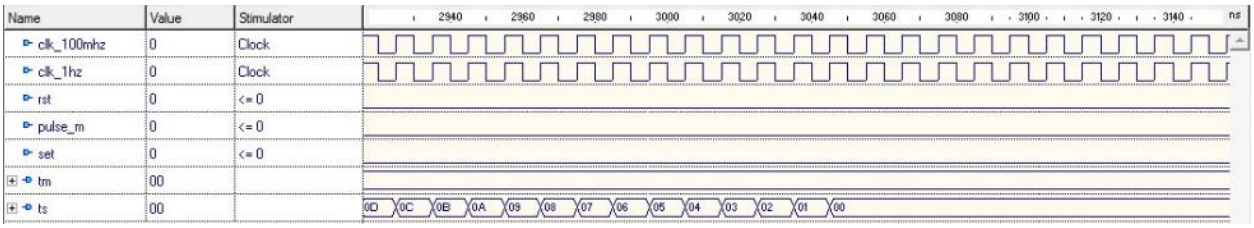

Figure 19 : Septième simulation du bloc *timer.*

Bloc heure/minuterie (*oventimetimerfsm*)

Ce bloc est une machine à états qui servira de contrôle sur le temps et la minuterie de la cuisinière. Il sera codé à l'aide de la machine à états finis de Moore suivante :

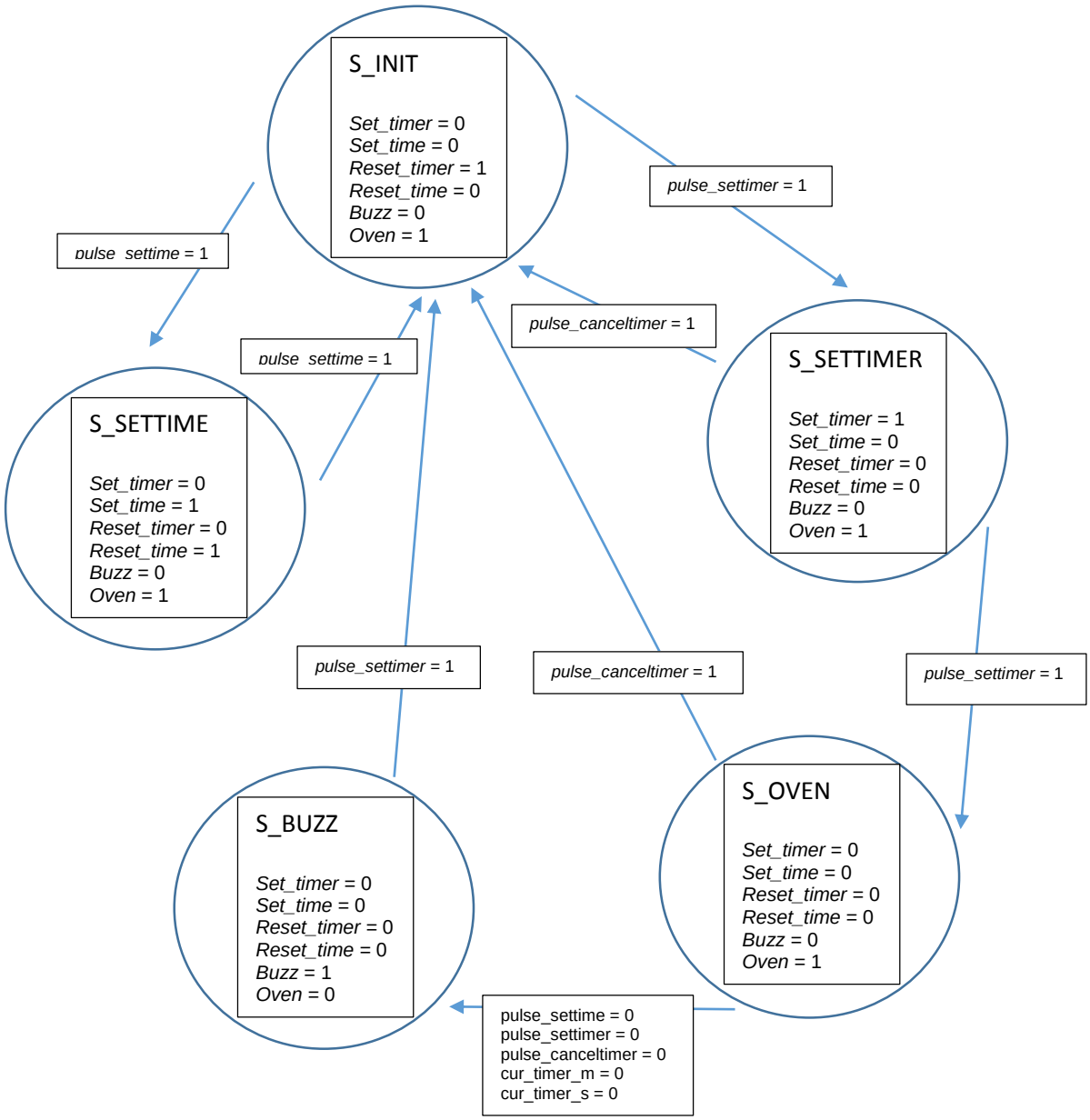

Figure 20 : Machine à états finis de Moore du bloc *oventimetimerfsm*.

Tableau II : Table de transitions pour le machine à états finis de Moore du bloc *oventimetimerfsm*.

	rst pulse_settime * pulse_settimer * pulse_canceltimer * cur_timer_m * cur_timer_s*				*set_timer * set_time * reset_timer* reset_time * buzz * oven*
États	**entrées**				**sorties**
	10000	01000	00100	00000	
S_INIT	S_SETTIME	S_SETTIMER	S_INIT	S_INIT	001001
S_SETTIME	S_INIT	S_SETTIME	S_SETTIME	S_SETTIME	010101
S_SETTIMER	S_SETTIMER	S_OVEN	S_INIT	S_SETTIMER	100001
S_OVEN	S_OVEN	S_OVEN	S_INIT	S_BUZZ	000001
S_BUZZ	S_BUZZ	S_INIT	S_BUZZ	S_BUZZ	000010

On peut alors écrire un code en langage VHDL qui représente notre diagramme à états :

Start Page | timekeeper.vhd (Text Document) | **oventimetimerfsm.vhd (Text Document)**

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

entity oventimetimerfsm is
    port(
        rst : in std_logic;
        clk : in std_logic;
        pulse_settime : in std_logic;
        pulse_settimer : in std_logic;
        pulse_canceltimer : in std_logic;
        cur_timer_m : in std_logic_vector(6 downto 0);
        cur_timer_s : in std_logic_vector(6 downto 0);
        set_timer : out std_logic;
        set_time : out std_logic;
        reset_timer : out std_logic;
        reset_time : out std_logic;
        buzz : out std_logic;
        oven : out std_logic
    );
end entity;

architecture a of oventimetimerfsm is
    type state_t is (
        -- liste des noms d'etats ici
        S_INIT,
        S_SETTIME,
        S_SETTIMER,
        S_OVEN,
        S_BUZZ
    );
begin
    process(clk)
        -- variable contenant l'etat en cours et l'etat initial
        variable cur_state : state_t := S_INIT;
    begin
        if rst = '1' then
            -- reinitialisation : retour a l'etat initial
            cur_state := S_INIT;

            -- sorties initiales
            --set_timer <= '0';
            --set_time <= '0';
            reset_timer <= '1';
            reset_time <= '1';
            --buzz <= '0';
            --oven <= '0';

        elsif rising_edge(clk) then
            case cur_state is
                when S_INIT =>
                    set_timer <= '0';
                    set_time <= '0';
                    reset_timer <= '1';
                    reset_time <= '0';
                    buzz <= '0';
                    oven <= '1';
                    if pulse_settime = '1' then
                        cur_state := S_SETTIME;
                    elsif pulse_settimer = '1' then
                        cur_state := S_SETTIMER;
                    end if;
```

```
                when S_SETTIME =>
                    set_timer <= '0';
                    set_time <= '1';
                    reset_timer <= '0';
                    reset_time <= '0';
                    buzz <= '0';
                    oven <= '1';
                    if pulse_settime = '1' then
                        cur_state := S_INIT;
                    elsif pulse_settime = '0' then
                        cur_state := S_SETTIME;
                    end if;

                when S_SETTIMER =>
                    set_timer <= '1';
                    set_time <= '0';
                    reset_timer <= '0';
                    reset_time <= '0';
                    buzz <= '0';
                    oven <= '1';
                    if pulse_settimer = '1' then
                        cur_state := S_OVEN;
                    elsif pulse_canceltimer = '1' then
                        cur_state := S_INIT;
                    end if;

                when S_OVEN =>
                    set_timer <= '0';
                    set_time <= '0';
                    reset_timer <= '0';
                    reset_time <= '0';
                    buzz <= '0';
                    oven <= '1';
                    if cur_timer_m = "0000000" and cur_timer_s = "0000000" then
                        cur_state := S_BUZZ;
                    elsif pulse_canceltimer = '1' then
                        cur_state := S_INIT;
                    end if;

                when S_BUZZ =>
                    set_timer <= '0';
                    set_time <= '0';
                    reset_timer <= '0';
                    reset_time <= '0';
                    buzz <= '1';
                    oven <= '0';
                    if pulse_canceltimer = '1' then
                        cur_state := S_INIT;
                    end if;
            end case;
        end if;
    end process;
end architecture;
```

Figure 21 : Code en langage VHDL du bloc *oventimetimerfsm*.

Ainsi, quand l'entrée *pulse_settime* est activée, un signal de sortie en *set_time* est émis vers le bloc du gardien du temps qui permettra l'ajustement de l'heure.

Quand l'entrée *pulse_settimer* est activée, un signal de sortie en *set_timer* est émis vers le bloc de la minuterie qui favorisera l'ajustement de celle-ci.

L'entrée *pulse_canceltimer*, elle, peut être activée à tout moment pour arrêter la minuterie ou l'alarme. Cette dernière est la sortie *buzz* qui sera activée au moment où les entrées *cur_timer_m* et *cur_timer_s* sont toutes les deux à 0, donc quand le temps de minuterie est nul. De même, bien que le four doit en tout temps être allumé par la sortie *oven*=1, son courant sera coupé dans l'état S_BUZZ quand les 2 conditions précédentes sont atteintes dans l'état S_OVEN. Toutefois, lorsque *pulse_canceltimer*=1 dans l'état S_BUZZ, le courant est rétabli dans la cuisinière et la sonnerie arrête ce qui nous faire revenir à l'état initial S_INIT.

Par ailleurs, l'entrée *reset* existe et permet aux sorties *reset_time* et *reset_timer* de s'activer et deux signaux s'envoient, respectivement, au bloc du gardien du temps et au bloc de la minuterie pour réinitialiser leurs temps en cours à 0.

Conclusion

Pour conclure, l'objectif de ce laboratoire est de réaliser les modules : générateur d'impulsion, gardien du temps, minuterie et machines à états finis du bloc heure/minuterie en programmation VHDL en commençant par concevoir les diagrammes d'états du générateur d'impulsion et de la machine à états finis du bloc heure/minuterie, par la suite leurs tableaux d'états et finalement leurs codes VHDL. Les simulations du générateur d'impulsions, du gardien du temps et de la minuterie démontrent que les modules fonctionnent comme tel qu'indiqué et de plus, nous avons implémenté le circuit complet dans la carte FPGA et nous en avons vérifié le fonctionnement sur l'afficheur LCD. D'après les tests, tout fonctionne comme prévu. Par ailleurs, nous pouvons améliorer ce laboratoire en permettant à la minuterie de nous signaler un bruit sonore à chaque intervalle de temps que l'usager aurai choisis. Disons, à chaque 10 minutes, la minuterie sonne pour nous prévenir que 10 minutes se sont écoulées.