Université Grenoble-Alpes
MISTRE - Microélectronique Intégration des Systèmes Temps Réel Embarqué
2020 – 2021

UE - TP Modélisation des Systèmes Numériques

Travaux pratiques:

VHDL

Liége MALDANER

Prof. Dr. Laurance PIERRE

Université Grenoble-Alpes, Grenoble, France

Table des matières

Chapitre 1

Introduction

Dans ce laboratoire, un système de contrôle simplifié a été conçu pour un four à micro-ondes. L'automate a été complété et la description VHDL de celui-ci a été effectuée et testé. Ensuite, la synthèse logique était fait, ainsi que le *technology mapping* sur la technologie CORELIB 0.35 μm pour un circuit intégrés ASIC. Finalement, le test post-synthétique du circuit a été effectué. Les résultats seront présentés et commentés dans les sections suivantes.

Chapitre 2

Le système complet

Le système complet est formé d'un contrôleur et d'un compteur, interconnectés comme montré dans la Figure 2.1.

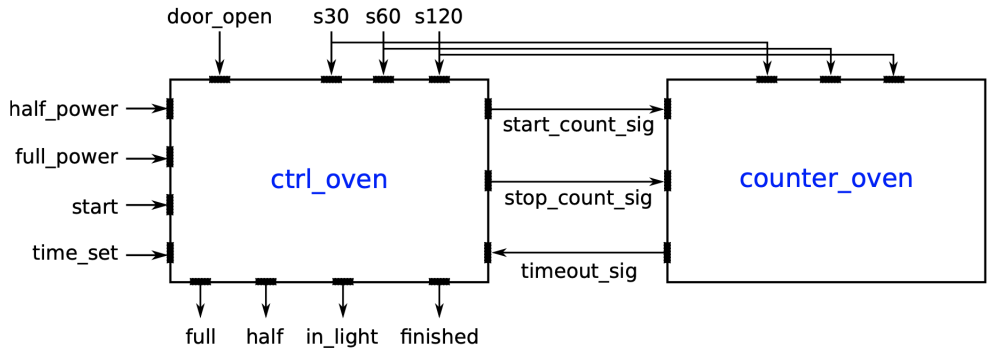

FIGURE 2.1 – Système complet (four à micro-ondes)

2.1 L'entité VHDL ctrl_oven

1. Le dessin complet de l'automate est fourni dans la Figure 2.2

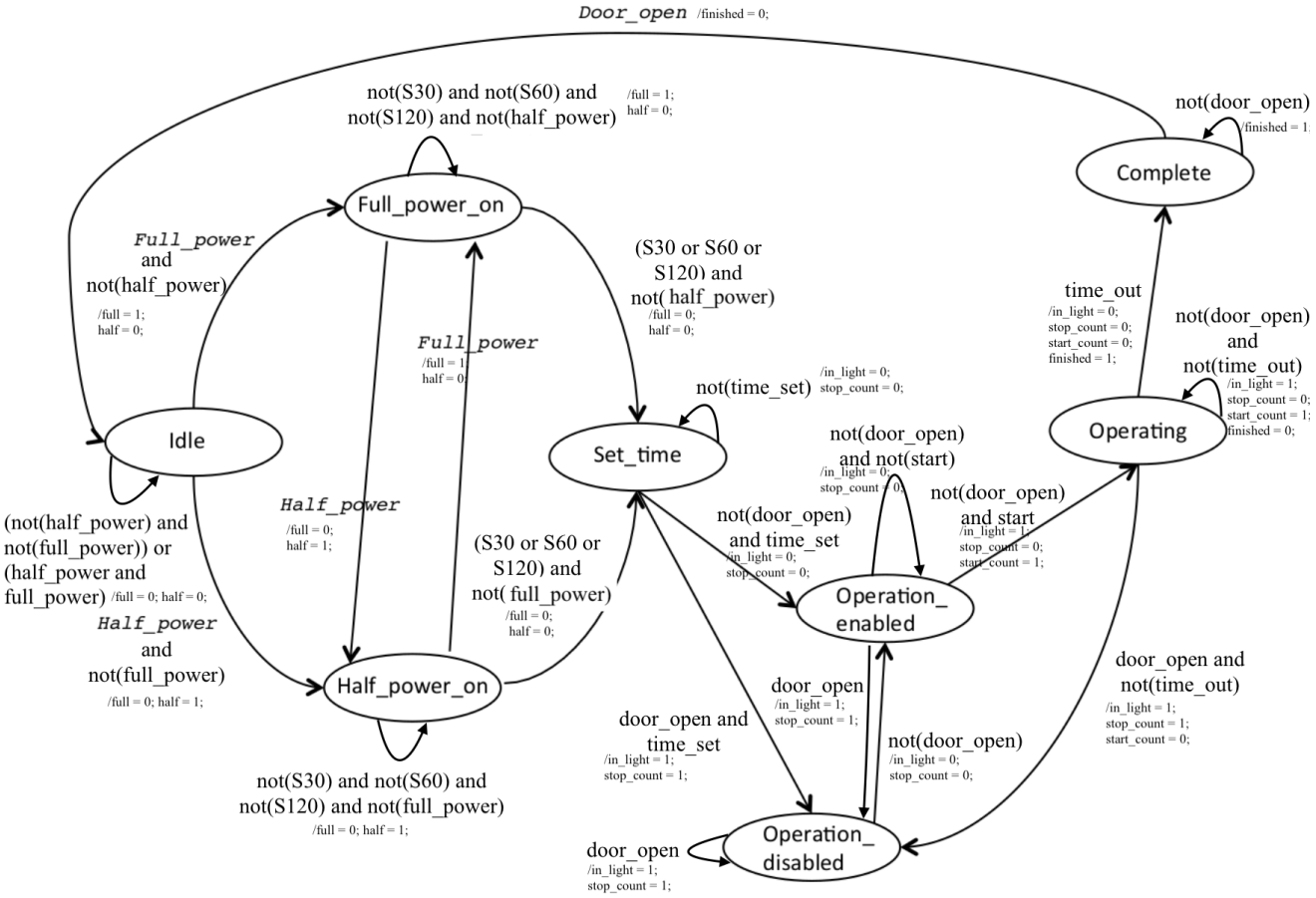

FIGURE 2.2 – Automate de Mealy

2. La description VHDL comportementale de ce système est développée en Annexe A. Il est divisé en deux partie :

   - la partie combinatoire : Cette partie fait l'évaluation des sorties et l'évaluation de l'état future. Vue qu'on décrit une machine *Mealy*, les sorties dépendent de la valeur courant de l'état et des valeurs courantes des entrées.
   - la partie séquentiel : Cette partie est synchronisée avec l'horloge du circuit et fait la mémorisation de l'état courant a chaque front montante d'horloge. Cette partie comprend également le *reset* asynchrone active haut.

   Le nom des états, entrés et sorties sont les mêmes qui ceux définit dans la Figure 2.2.

3. On a écrit le *test bench* avec les spécifications des changements d'états et le changement d'entrées correspondent, comme décrit ci-dessous. Pour cela, on utilise une fréquence d'horloge de 50 MH. Le résultat de simulation avec ModelSim est montré dans la Figure 2.3. Le code VHDL est développée en Annexe B

   (a) $f_{clk} = 50\text{MHz}$, $T_{clk} = 20$ ns
   $T/2_{clk} = 10$ ns

   (b) Reset

(c) Idle > idle
full_power = 0
half_power = 0

(d) Idle > full_power_on
full_power = 1
half_power = 0

(e) full_power_on > full_power_on
half_power = 0

(f) full_power_on > half_power_on
half_power = 1

(g) half_power_on > full_power_on
full_power = 1

(h) full_power_on > set_time
half_power = 0
s30 = 1

(i) set_time > set_time
time_set = 0

(j) set_time > operation_disabled
time_set = 1
door_open = 1

(k) operation_disabled > operation_disabled
door_open = 1

(l) operation_disabled > operation_enabled
door_open = 0

(m) operation_enabled > operation_enabled
door_open = 0
start = 0

(n) operation_enabled > operation_disabled
door_open = 1

(o) operation_disabled > operation_enabled
door_open = 0

(p) operation_enabled > operating
start = 1
door_open = 0

(q) operating > operating
oor_open = 0
timeout = 0

(r) operating > operation_disabled
door_open = 1
timeout = 0

(s) operation_disabled > operation_enabled
door_open = 0

(t) operation_enabled > operating
start = 1
door_open = 0

(u) operating > complete
timeout = 1

(v) complete > complete
door_open = 0

(w) complete > idle
door_open = 1

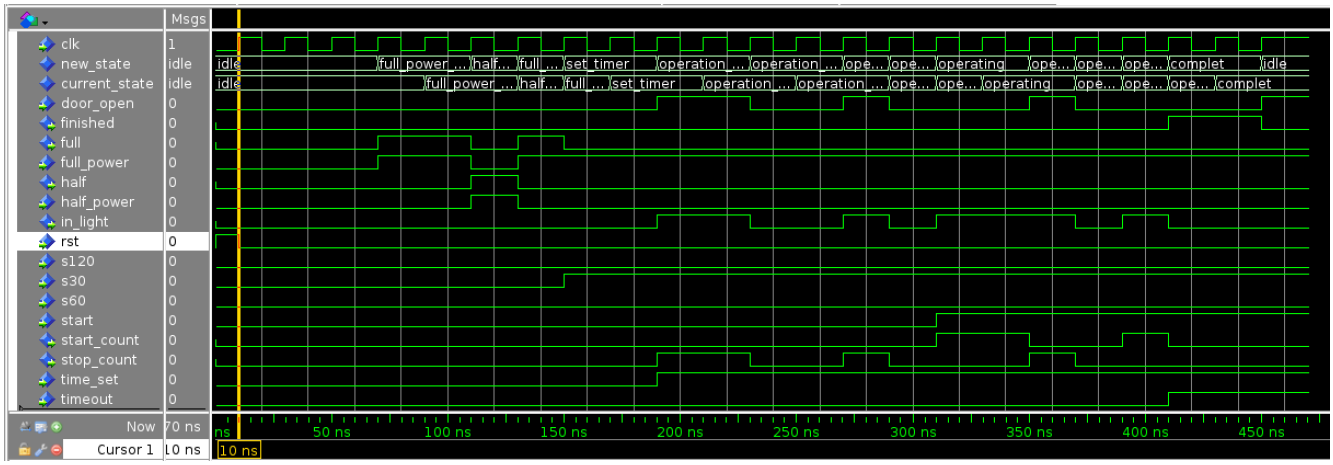

FIGURE 2.3 – Résultats de simulation du ctrl_oven

2.2 L'entité VHDL counter_oven

4. Ci dessous on montré le résultat de la description VHDL complété.

```
library IEEE;
use IEEE.std_logic_1164.all;

architecture sequentiel of counter_oven is

type States is (idle, counting, stopped);
signal state, nextstate: States;
signal c, nextc: natural;

begin
process (state, c, start, stop, s30, s60, s120)
    begin
    case state is
    when idle =>
        if start='1' then
            nextstate <= counting;
            nextc <= c+1;
        else
            nextstate <= idle;
            nextc <= 0;
        end if;
```

```
    when counting =>
        if stop = '1' then
            nextstate <= stopped;
            nextc <= c;
        elsif ((s30='1' and c >= 30) or (s60='1' and c >= 60)
              or (s120='1' and c >= 120)) then
            nextstate <= idle;
            nextc <= 0;
        else
            nextstate <= counting;
            nextc <= c+1;
        end if;
    when stopped =>
        if start='1' then
            nextstate <= counting;
            nextc <= c+1;
        else
            nextstate <= stopped;
            nextc <= c;
        end if;
    end case;
end process;

process (rst, clk)
begin
    -- if asynchronous reset
    if (rst = '1') then
        state <= idle;
        c <= 0;
    -- if rising edge
    elsif (clk'event and clk='1') then
        state <= nextstate;
        c <= nextc;
    end if;
end process;

aboveth <= '1' when ((s30='1' and c >= 30) or (s60='1' and c >= 60) or
    (s120='1' and c >= 120)) else '0';

end architecture;
```

Le résultat du *test bench* est montre dans la Figure 2.4. On a teste les états idle à idle, idle à counting, counting à counting, counting à stopped, stopped à counting et counting (jusqu'à 30 ns) à idle . Le code VHDL est développée en Annexe C.

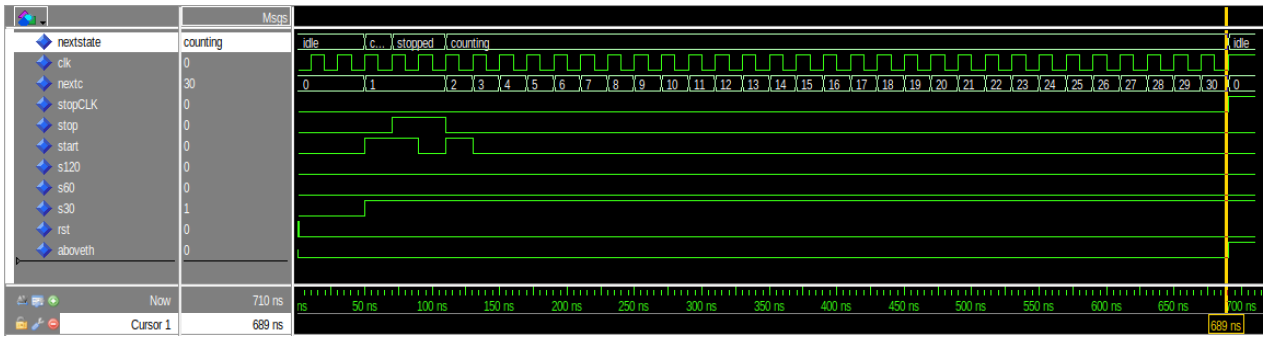

FIGURE 2.4 – Résultats de simulation du counter_oven

2.3 L'entité VHDL du système complet

Dans la Figure 2.5 on montre le test de la synthèse VHDL du système formé par l'interconnexion du contrôleur et du compteur.

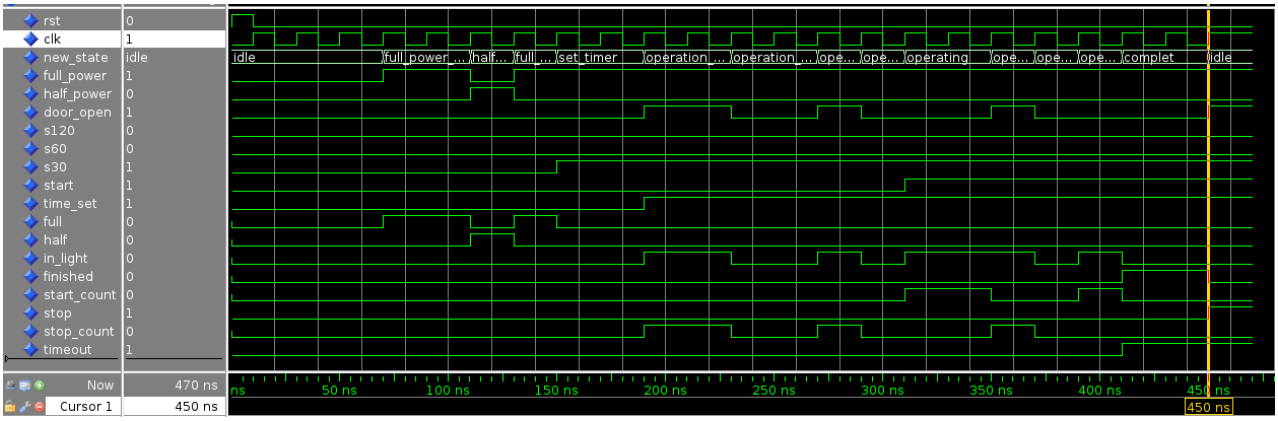

FIGURE 2.5 – Résultats de simulation du système complet

Les scénarios testés avec le *test bench* sont réalisés avec les spécifications et les changements d'états décrit ci-dessous, ainsi que les variations d'entrées qui provoquent le changement d'état de la FSM. On cherche a couvrir un maximum de cas en variant les entrées et en passant pour tout les états de la FSM.

(a) $f_{clk} = 50\text{MHz}$, $T_{clk} = 20$ ns
$T/2_{clk} = 10$ ns

(b) Reset

(c) Idle > idle
full_power = 0
half_power = 0

(d) Idle > full_power_on
full_power = 1
half_power = 0

(e) full_power_on > full_power_on
half_power = 0

(f) full_power_on > half_power_on
half_power = 1

(g) half_power_on > full_power_on
half_power = 0

(h) full_power_on > set_time
s30 = 1

(i) set_time > set_time
time_set = 0

(j) set_time > operation_disabled
time_set = 1
door_open = 1

(k) operation_disabled > operation_disabled
door_open = 1

(l) operation_disabled > operation_enabled
door_open = 0

(m) operation_enabled > operation_enabled
door_open = 0
start = 0

(n) operation_enabled > operation_disabled
door_open = 1

(o) operation_disabled > operation_enabled
door_open = 0

(p) operation_enabled > operating
start = 1
door_open = 0

(q) operating > operating
door_open = 0
timeout = 0

(r) operating > operation_disabled
door_open = 1
timeout = 0

(s) operation_disabled > operation_enabled
door_open = 0

(t) operation_enabled > operating (ctrl_oven) et counting (counter_oven)
start = 1
door_open = 0

(u) operating (ctrl_oven) et counting (counter_oven)> complete
timeout = 1

(v) complete > complete
door_open = 0

(w) complete > idle
door_open = 1

Chapitre 3

L'analyse de couverture

Avec l'option de couverture du logiciel Modelsim, on a simulé le modèle et généré le rapport de couverture correspondant :

```
Coverage Report Summary Data by file

File: /tp/xm2mist/xm2mist023/tp_vhdl_msn/TP3/Sources/counter_oven_behavioral.vhd
    Enabled Coverage              Active       Hits     Misses % Covered
    ----------------              ------       ----     ------ ---------
    Stmts                             20         20          0     100.0
    Branches                          14         14          0     100.0
    FSMs                                                             93.7
        States                         3          3          0     100.0
        Transitions                    8          7          1      87.5

File: /tp/xm2mist/xm2mist023/tp_vhdl_msn/TP3/Sources/ctrl_oven_behavioral.vhd
    Enabled Coverage              Active       Hits     Misses % Covered
    ----------------              ------       ----     ------ ---------
    Stmts                            117        117          0     100.0
    Branches                          32         32          0     100.0
    FSMs                                                             89.2
        States                         8          8          0     100.0
        Transitions                   28         22          6      78.5

Total Coverage By File (code coverage only, filtered view): 96.7%
```

Afin d'obtenir le taux de couverture proche de 100%, on a inclus quelques modification dans le *test bench* afin de couvrir toutes les possibilités existantes de variation d'états de la FSM. Les inclusions et modifications effectuées sont décrites ci-dessous.

Modifications :

(u) operating (ctrl_oven) et counting (counter_oven)> complete
    full_power = 0
    timeout = 1

(v) complete > complete
    door_open = 0

(w) complete > idle
    door_open = 1

Inclusions :

(x) Idle > idle
door_open = 1

(y) Idle > half_power_on
half_power = 1

(z) half_power_on > half_power_on
s30 <= '0'
s60 <= '0'
s120 <= '0'

(a.1) half_power_on > set_time
s30 = 1

(b.2) set_time > operation_enabled
time_set = 1
door_open = 0

(c.2) operation_enabled > operating
start = 1

(d.2) operating > complete

(e.2) complete > idle
door_open = 1

Les informations sur le parcours d'états sont montrés graphiquement par simulation, comme montrée dans la Figure 3.1.

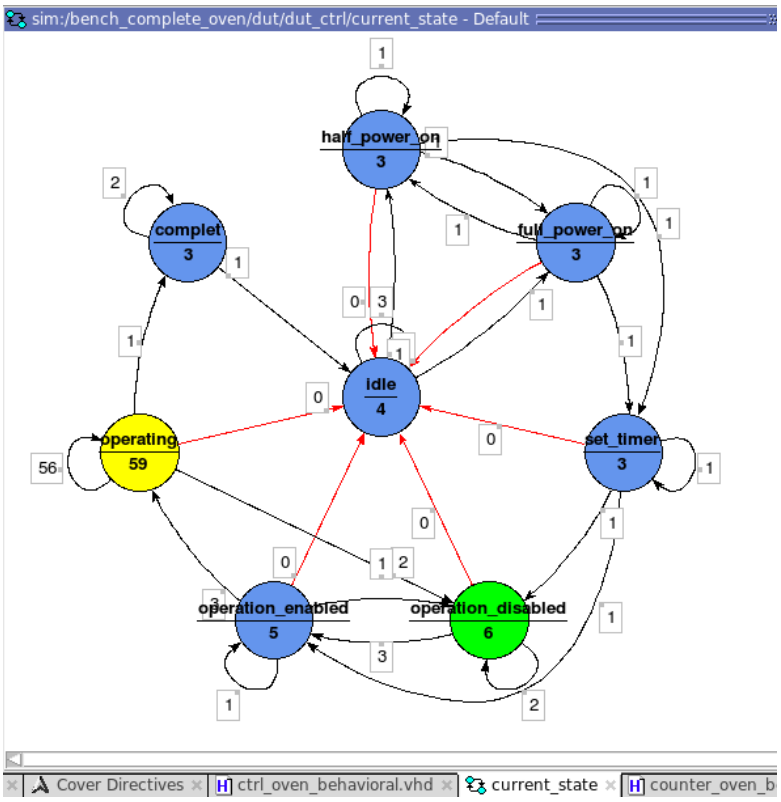

FIGURE 3.1 – Résultats de couverture

Dans les Figures 3.2, 3.3, 3.4 et 3.5 on montre le test de la synthèse VHDL du système complet post-analyse de couverture. On peut conclure que la FSM attend les spécifications demandées.

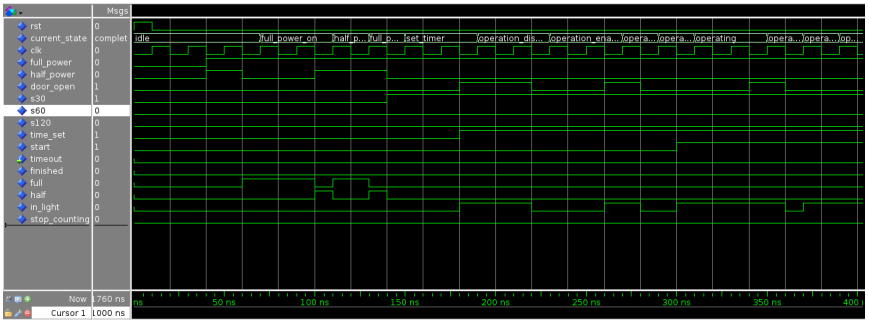

FIGURE 3.2 – Résultats de simulation

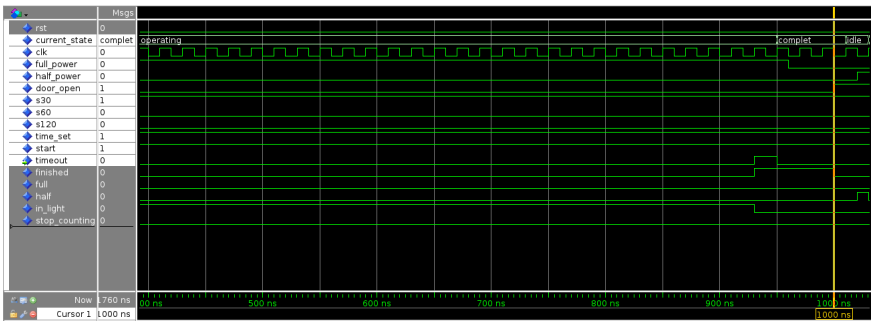

FIGURE 3.3 – Continuation des résultats de simulation

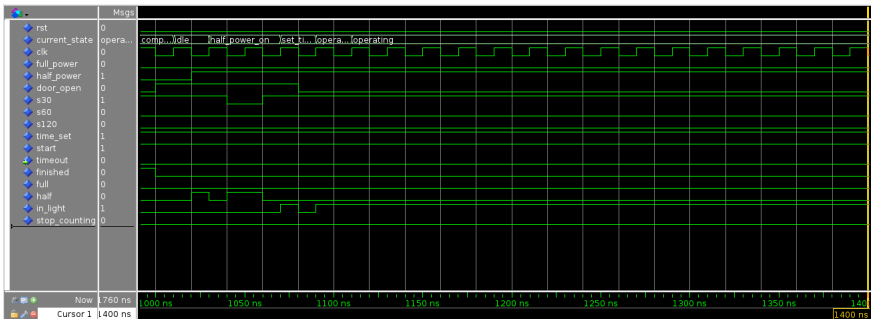

FIGURE 3.4 – Continuation des résultats de simulation

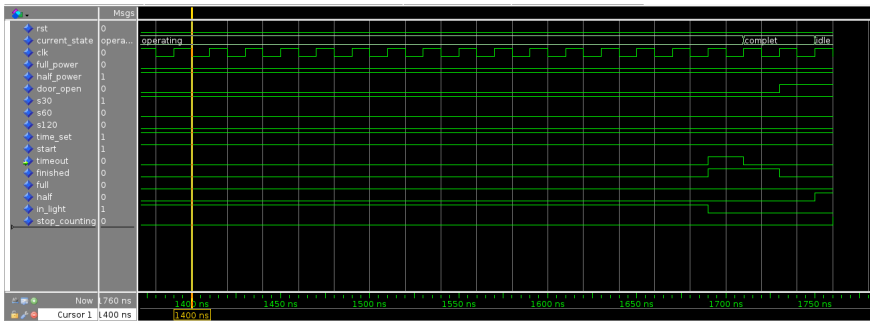

FIGURE 3.5 – Continuation des résultats de simulation

Chapitre 4

Synthèse avec Leonardo Spectrum

Après avoir écrit le code VHDL et testé sa fonctionnalité à l'aide du logiciel Modelsim, nous pouvons procéder aux étapes de synthèse logique et de *technology mapping* grâce au logiciel Leonardo. La technologie utilisée est celle de la bibliothèque Corelib 0.35 μm. Leonardo nous permet de choisir le type d'encodage de l'état de la FSM. De cette façon, nous générons différentes *netlists* en utilisant 3 types de codage : Binaire, Gray et One Hot.

Comme notre FSM ctrl_oven compte 8 états, on obtient 3 registres d'états en codage Binaire et Gray, et 8 registres en codage One Hot, comme le montrent les tableaux 4.2 et 4.3. Les codages configurées par Leonardo sont montré dans la Table 4.1.

value type_state[2-0]	**Binary**	**Gray**	**One Hot**
idle	000	000	00000001
full_power_on	001	001	00000010
half_power_on	010	011	00000100
set_timer	011	010	00001000
operation_disabled	100	110	00010000
operation_enabled	101	111	00100000
operating	110	101	01000000
complet	111	100	10000000

TABLE 4.1 – Codage des états

Il est également possible de faire l'optimisation du circuit, par exemple en termes de surface utilisée, ou en termes de délai pour augmenter la fréquence du circuit. Les tableaux 4.2 et 4.3 montrent les résultats obtenus en termes de nombre de bascules, de surface utilisée (en μm^2), et de fréquence d'horloge, pour les 3 types de codages, et suivant les 2 types d'optimisation.

On peut conclure que l'optimisation en surface génère des circuits plus petits (comparé à l'autre implémentation) ce qui peut être intéressant dans la fabrication de circuits à grande échelle, en diminuant le coût de production. On remarque que l'optimisation en délai génère une fréquence d'horloge plus faible que pour l'autre implémentation pour le codage de Gray. Cela peut s'expliquer par le fait que notre circuit est très petit, et les méthodes d'optimisation de délai comme la réalisation de calcul en parallèle n'est peut-être pas très efficace ici, alors que l'optimisation en surface a pu trouver une disposition favorable en surface et même en délai.

Avec les rapports produits par Leonardo on peut conclure que les *netlists* générée n'utilise pas de *Latches*, mais uniquement des *Flip-Flops* DF1 (D-Type Flip Flop (1x)). De plus, les autres composants utilisés appartiennent bien à la même bibliothèque c35_CORELIB.

Codage	Binaire	Gray	One Hot
Nombre de Flip-Flops (cellules)	3 (DF1)	3 (DF1)	8 (DF1)
Surface [um2] (Bibliothèque)	4514 (c35_CORELIB)	4131 (c35_CORELIB)	4404 (c35_CORELIB)
Fréquence [MHz]	233.9	384.6	729.1

TABLE 4.2 – Optimisation en surface

Codage	Binaire	Gray	One Hot
Nombre de Flip-Flops (cellules)	3 (DF1)	3 (DF1)	8 (DF1)
Surface [um2] (Bibliothèque)	5296 (c35_CORELIB)	4859 (c35_CORELIB)	4404 (c35_CORELIB)
Fréquence [MHz]	250.4	320.0	729.1

TABLE 4.3 – Optimisation en délai

Après avoir réalisé les étapes de synthèse logique et de *technology mapping*, nous devons refaire la simulation (simulation post-synthèse) pour nous assurer que le circuit fonctionne toujours selon les spécifications. La Figure 4.1 montre les résultats.

On conclue qu'il y a des *glitches* sur des sorties à 90, 210, 290 et 330 ns. En analysant ces *glitches* (en Annexe D), on voit qu'ils ne se produisent pas au même temps que le front montant d'horloge se produit. Si on voudrais passer ces sorties sans *glitches* à un autre composant, une solution est la mémorisation de sorties. De cette façon, les *glitches* n'auraient pas influencer la évaluation des entrées synchrone du composant. Par contre, ces *glitches* peuvent augmenté la consommation d'énergie du circuit.

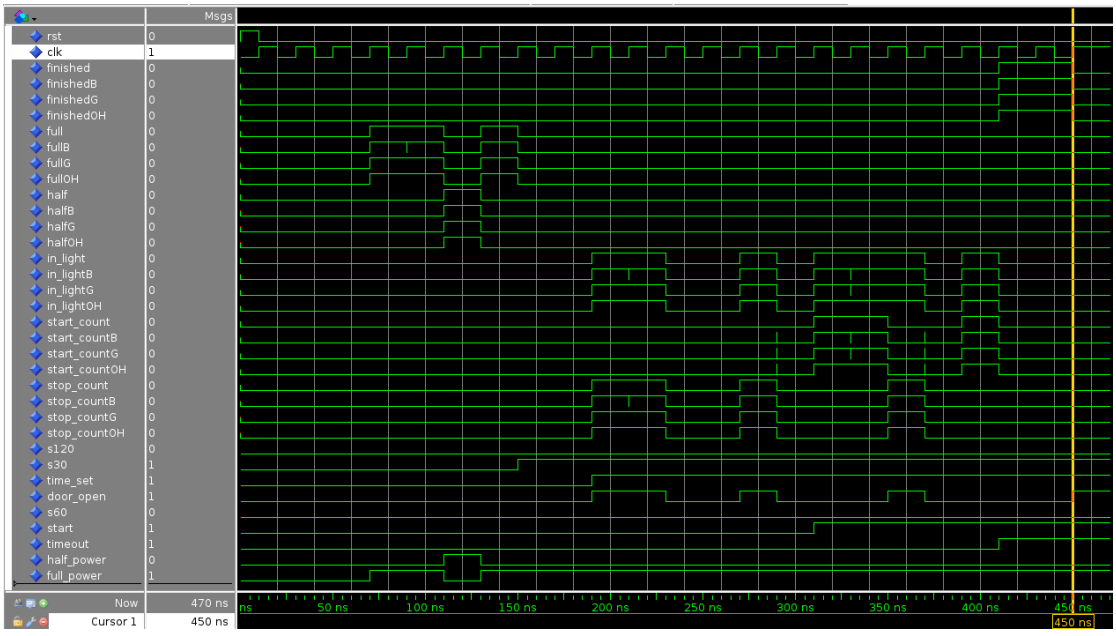

FIGURE 4.1 – Résultats de simulation (résultats de optimisation en surface) [1]

1. Les signaux sont différenciés par une ou deux lettres à la fin : B = Binaire, G = Gray, OH = One Hot

Chapitre 5

Conclusion

Avec ce projet VHDL, il est possible de comprendre les concepts liés à la conception de circuits intégrés pour une FSM. Cela était fait en suivant différentes étapes de conception. La premier étape était l'écriture de la description VHDL du circuit et le test en utilisant le logiciel Modelsim, ainsi que l'analyse de couverture pour valide la taux des états de la FSM qui ont été testés.

La deuxième étape est la synthèse logique du circuit en utilisant le logiciel LeonardoSpectrum. Ensuite, il était possible de produire le *technology mapping* de la technologie 0,35 μm pour la conception du circuit ASIC. Pour cela, on a utilisé trois différents types de codage d'état (Gray, Binaire et One Hot), ainsi l'optimisation du circuit en termes de minimisation du délai et de surface du circuit. Enfin, l'analyse post-synthèse a été faite pour s'assurer du bon fonctionnement du circuit après le *technology mapping*.

On peut ainsi conclure que ce laboratoire a été important pour consolider les connaissances en microélectronique pour la conception des circuits intégrés.

Annexe A

Architecture VHDL ctrl_oven

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

architecture sequentiel of ctrl_oven is

--definition des signals et des etat
type type_state is (idle,full_power_on,half_power_on,set_timer,operation_disabled,
operation_enabled,operating,complet);
signal new_state : type_state;
signal current_state : type_state;

begin
-- partie combinationel: calcule de l'etat new_state
process(RST, half_power, full_power, start, s30, s60, s120,
  time_set, door_open, timeout, current_state)
  begin
  case(current_state) is
    when idle =>
      in_light <= '0';
      finished <= '0';
      start_count <= '0';
      stop_count <= '0';
      if (((not(full_power) and not(half_power)) or (half_power and full_power))='1') then
        new_state <= idle;
        full <= '0';
        half <= '0';
      elsif ((full_power and not(half_power)) = '1') then
        new_state <= full_power_on;
        full <= '1';
        half <= '0';
      else --if ((half_power and not(full_power)) = '1') then
        new_state <= half_power_on;
        half <= '1';
        full <= '0';
      end if;
    when full_power_on =>
      in_light <= '0';
      finished <= '0';
```

```
      start_count <= '0';
      stop_count <= '0';
      if ((not(S30) and not(S60) and not(S120) and not(half_power)) = '1') then
        new_state <= full_power_on;
        full <= '1';
        half <= '0';
      elsif (half_power = '1') then
        new_state <= half_power_on;
        half <= '1';
        full <= '0';
      else--if ((S30 or S60 or S120 and not(half_power)) = '1') then
        new_state <= set_timer;
        half <= '0';
        full <= '0';
      end if;
    when half_power_on =>
      in_light <= '0';
      finished <= '0';
      start_count <= '0';
      stop_count <= '0';
      if ((not(S30) and not(S60) and not(S120) and not(full_power)) = '1') then
        new_state <= half_power_on;
        half <= '1';
        full <= '0';
      elsif (full_power = '1') then
        new_state <= full_power_on;
        half <= '0';
        full <= '1';
      else--if ((S30 or S60 or S120 and not(full_power)) = '1') then
        new_state <= set_timer;
        half <= '0';
        full <= '0';
      end if;
    when set_timer =>
      full <= '0';
      half <= '0';
      finished <= '0';
      start_count <= '0';
      if (not(time_set) = '1') then
        new_state <= set_timer;
        in_light <= '0';
        stop_count <= '0';
      elsif ((not(door_open) and time_set) = '1') then
        new_state <= operation_enabled;
        in_light <= '0';
        stop_count <= '0';
      else --if (door_open and time_set) then
        new_state <= operation_disabled;
        in_light <= '1';
        stop_count <= '1';
      end if;
    when operation_disabled =>
      full <= '0';
      half <= '0';
      finished <= '0';
      start_count <= '0';
```

```
      if ((door_open) = '1') then
        new_state <= operation_disabled;
        in_light <= '1';
        stop_count <= '1';
      else --if (not(door_open) = '1') then
        new_state <= operation_enabled;
        in_light <= '0';
        stop_count <= '0';
      end if;
    when operation_enabled =>
      full <= '0';
      half <= '0';
      in_light <= '0';
      finished <= '0';
      stop_count <= '0';
      if ((not(door_open) and not(start)) = '1') then
        new_state <= operation_enabled;
        in_light <= '0';
        stop_count <= '0';
        start_count <= '0';
      elsif (door_open = '1') then
        new_state <= operation_disabled;
        in_light <= '1';
        stop_count <= '1';
        start_count <= '0';
      else --if (not(door_open) and start = '1')
        new_state <= operating;
        in_light <= '1';
        stop_count <= '0';
        start_count <= '1';
      end if;
    when operating =>
      full <= '0';
      half <= '0';
      in_light <= '0';
      finished <= '0';
      start_count <= '0';
      if ((not(door_open) and not(timeout)) = '1') then
        new_state <= operating;
        in_light <= '1';
        start_count <= '1';
        finished <= '0';
        stop_count <= '0';
      elsif (timeout = '1') then
        new_state <= complet;
        in_light <= '0';
        start_count <= '0';
        finished <= '1';
        stop_count <= '0';
      else --if(door_open and not(timeout) <= '1') then
        new_state <= operation_disabled;
        in_light <= '1';
        start_count <= '0';
        finished <= '0';
        stop_count <= '1';
      end if;
```

```
    when complet =>
      full <= '0';
      half <= '0';
      in_light <= '0';
      start_count <= '0';
      stop_count <= '0';
      if ( not(door_open) = '1') then
        new_state <= complet;
        finished <= '1';
      else --if (door_open <= '1') then
        new_state <= idle;
        finished <= '0';
      end if;
  end case;
end process;

-- partie sequenciel: mise a jout de l'etat current_state
process (rst, clk)
  begin
      -- if asynchronous reset
      if (rst = '1') then
          current_state <= idle;
      -- if rising edge
      elsif (clk'event and clk='1') then
          current_state <= new_state;
      end if;
  end process;

end sequentiel;
```

Annexe B

Test Bench de la FSM ctrl_oven

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

library LIB_TP3;
use     LIB_TP3.ctrl_oven;

entity bench_ctrl_oven is
end bench_ctrl_oven;

architecture test of bench_ctrl_oven is

--define component
component ctrl_oven
port(
  clk, rst, half_power, full_power, start, s30, s60, s120,
  time_set, door_open, timeout : in std_logic;
  full, half, in_light, finished, start_count, stop_count : out std_logic
  );
end component;

-- define signals
signal rst: std_logic := '0';
signal clk, half_power, full_power, start, s30, s60, s120,
  time_set, door_open, timeout, stop : std_logic := '0';
signal full, half, in_light, finished, start_count, stop_count : std_logic;

begin
  -- design uder test
    dut : ctrl_oven
  port map(
    clk,
    rst,
    half_power,
    full_power,
    start,
    s30,
    s60,
    s120,
```

```
    time_set,
    door_open,
    timeout,
    full,
    half,
    in_light,
    finished,
    start_count,
    stop_count
  );

  -- Generation des stimulis
  -- clk stimuli
  stop <=  '1' after 450 ns;

  process(stop,clk)
  begin
  if stop='0' then
    clk <= not clk after 10 ns;
  end if;
  end process;

  -- inputs stimulis
  process
  begin
    if stop='0' then
    rst <= '1';
    wait for 10 ns; -- idle
    rst <= '0';
    wait for 29 ns; -- idle
    full_power <= '1';
    half_power <= '1';
    wait for 20 ns; -- full_power
    half_power <= '0';
    wait for 40 ns; -- half_power
    half_power <= '1';
    wait for 40 ns; -- full_power > set_time
    half_power <= '0';
    s30 <= '1';
    wait for 40 ns; -- operation_disabled
    time_set <= '1';
    door_open <= '1';
    wait for 40 ns; -- operation_enabled
    door_open <= '0';
    wait for 40 ns; -- operation_disabled
    door_open <= '1';
    wait for 20 ns; -- operation_enabled
    door_open <= '0';
    wait for 20 ns; -- operating
    start <= '1';
    wait for 40 ns; -- operation_disabled
    door_open <= '1';
    wait for 20 ns; -- operation_enabled
    door_open <= '0';
    wait for 40 ns; -- operating
    timeout <= '1';
```

```
    wait for 40 ns; -- complete
    door_open <= '1';
    wait for 20 ns; --idle
    else wait;
    end if;
  end process;

end architecture;
```

Annexe C

Test Bench de la FSM counter_oven

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

library LIB_TP3;
use     LIB_TP3.counter_oven;

entity bench_counter_oven is
end bench_counter_oven;

architecture test of bench_counter_oven is

--define component
component counter_oven
port(
  clk, rst, start, stop, s30, s60, s120 : in std_logic;
  aboveth : out std_logic
);
end component;

-- define signals
signal rst: std_logic := '0';
signal clk, start, stop, stop_counting, s30, s60, s120  : std_logic := '0';
signal aboveth : std_logic;

begin
-- design uder test
dut : counter_oven
port map(
  clk, rst, start, stop, s30, s60, s120,
  aboveth
);

-- Generation des stimulis
-- clk stimuli
stop_counting <=  '1' after 700 ns;

process(stop_counting,clk)
begin
```

```
if stop_counting='0' then
  clk <= not clk after 10 ns;
end if;
end process;

-- inputs stimulis
process
begin
  if stop_counting='0' then
  rst <= '1';
  wait for 1 ns; -- idle
  rst <= '0';
  wait for 9 ns; -- idle
  start <= '0';
  s30 <= '1';
  wait for 30 ns; -- counting
  start <= '1';
  wait for 40 ns; -- stopped
  stop <= '1';
  wait for 20 ns; -- stopped
  start <= '0';
  wait for 40 ns; -- counting
  stop <= '0';
  start <= '1';
  wait for 600 ns; -- counting until idle
  else wait;
  end if;
end process;

end architecture;
```

Annexe D

Analyse des *glitches* post-synthèse

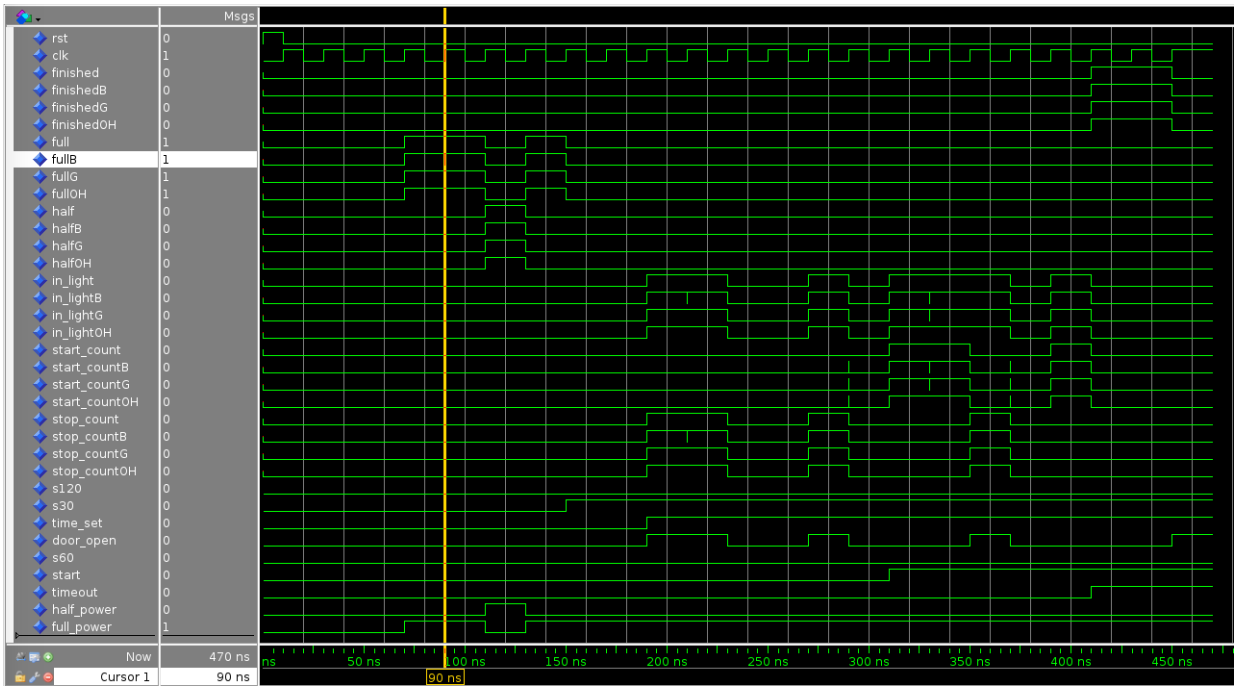

FIGURE D.1 – Glitches en 90 ns

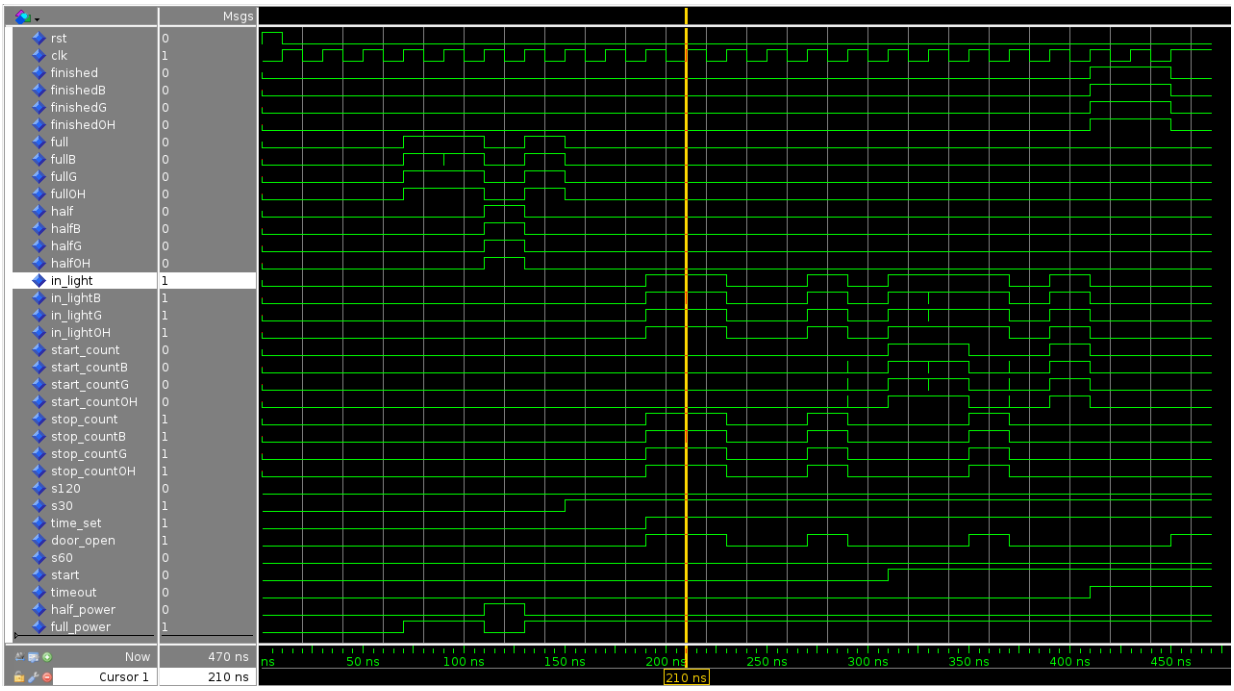

FIGURE D.2 – Glitches en 210 ns

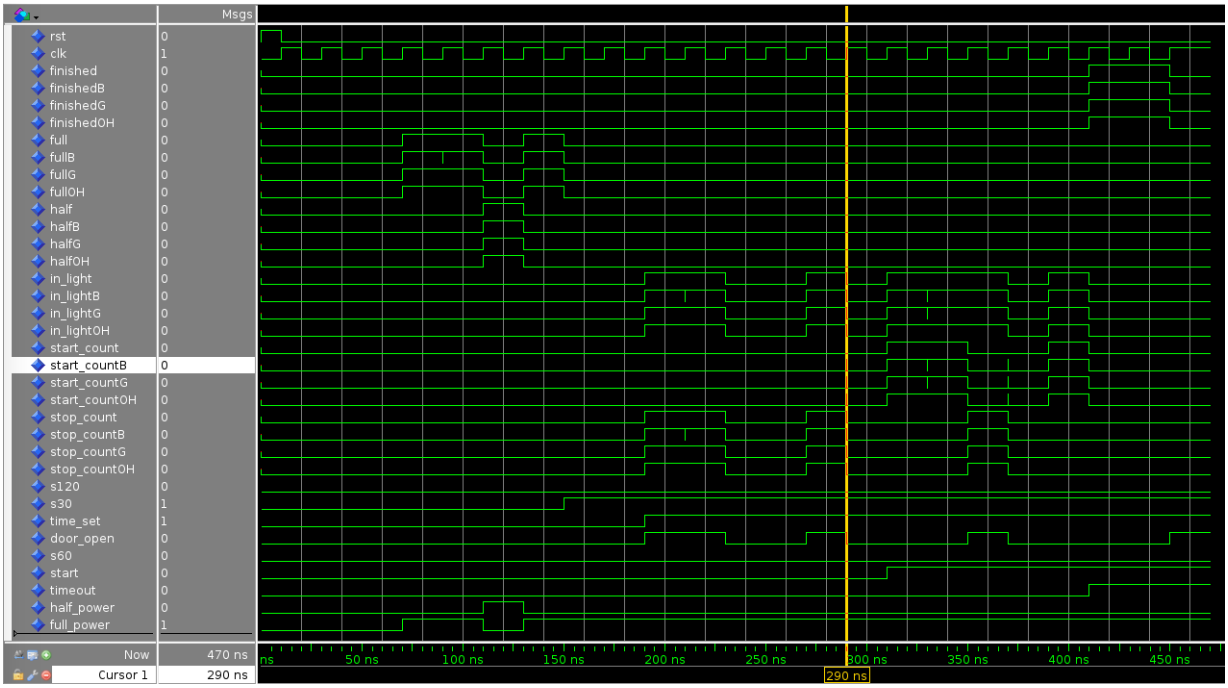

FIGURE D.3 – Glitches en 290 ns

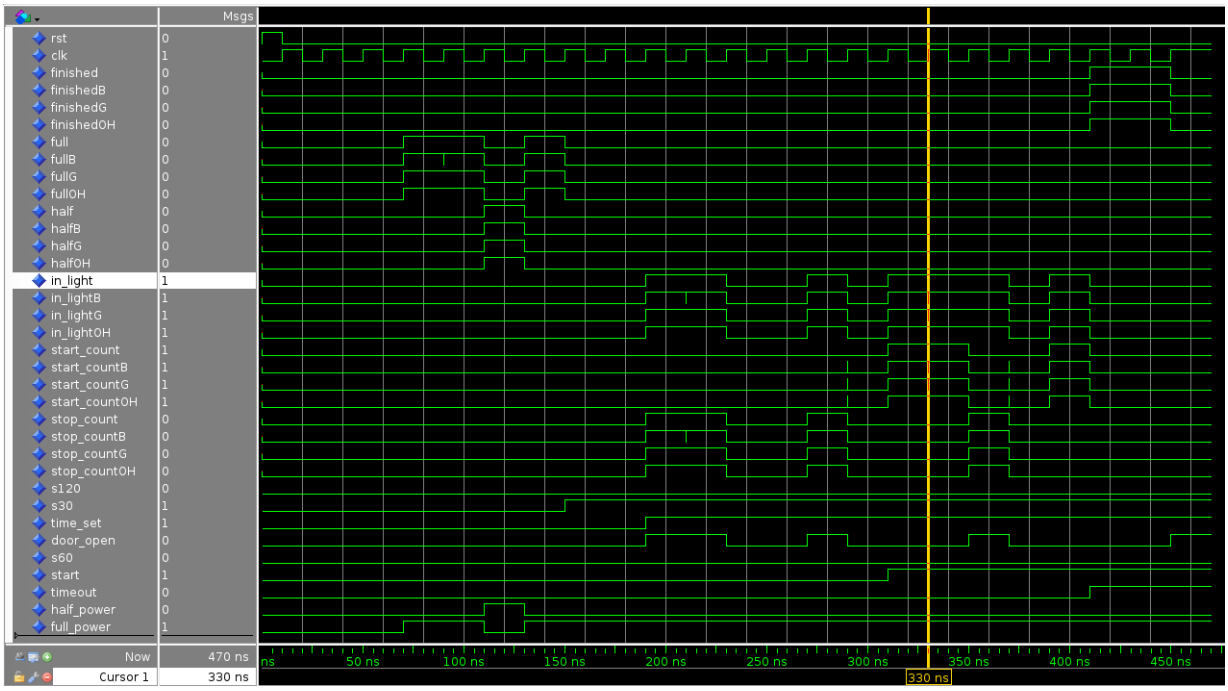

FIGURE D.4 – Glitches en 330 ns

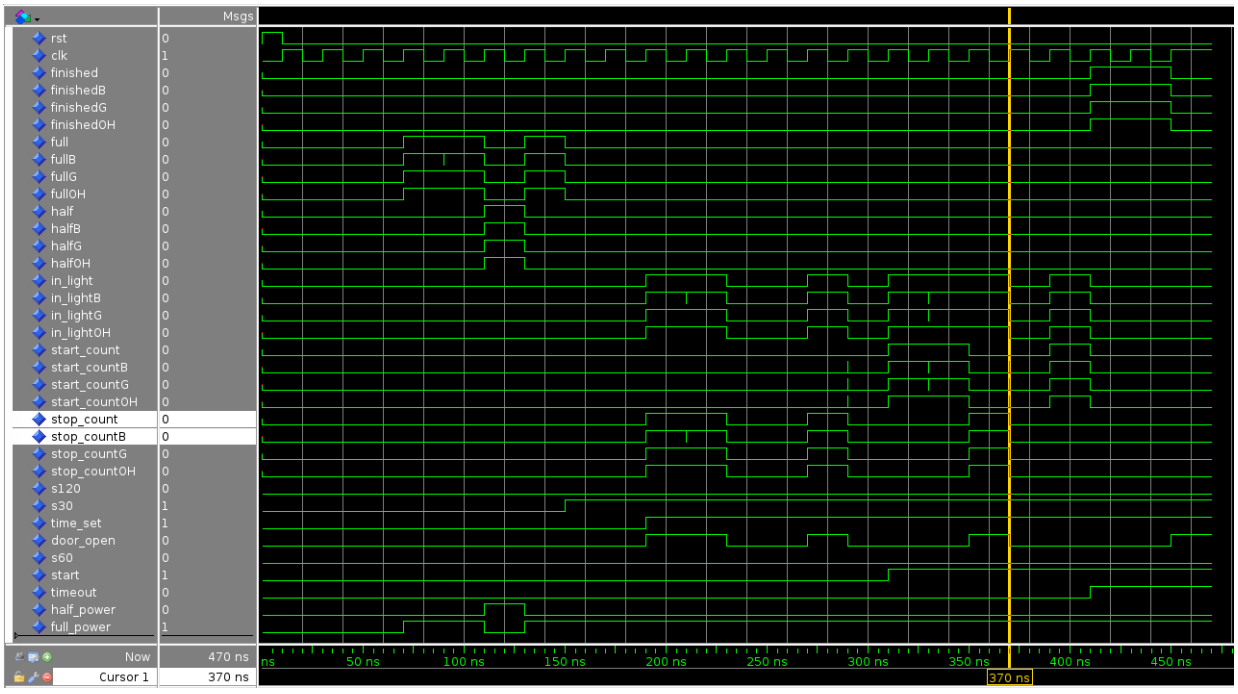

FIGURE D.5 – Glitches en 370 ns